Modeling, Design, and Control of Multiple Quadrotors

by

Abdullah E. Altawaitan

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved July 2019 by the
Graduate Supervisory Committee:

Armando A. Rodriguez, Chair
Konstantinos Tsakalis
Spring Berman

ARIZONA STATE UNIVERSITY

August 2019

ABSTRACT

In the last few decades, with the revolution of availability of low-cost microelectronics, which allow fast and complex computations to be performed on board, there has been increasing attention to aerial vehicles, especially rotary-wing vehicles. This is because of their ability to vertically takeoff and land (VTOL), which make them appropriate for urban environments where no runways are needed. Quadrotors took considerable attention in research and development due to their symmetric body, which makes them simpler to model and control compared to other configurations. One contribution of this work is the design of a new open-source based Quadrotor platform for research. This platform is compatible with both HTC Vive Tracking System (HVTS) and OptiTrack Motion Capture System, Robot Operating System (ROS), and MAVLINK communication protocol. The thesis examined both nonlinear and linear modeling of a 6-DOF rigid-body quadrotor's dynamics along with actuator dynamics. Nonlinear/linear models are used to develop control laws for both low-level and high-level hierarchical control structures. Both HVTS and OptiTrack were used to demonstrate path following for single and multiple quadrotors. Hardware and simulation data are compared. In short, this work establishes a foundation for future work on formation flight of multi-quadrotor.

To my parents.

ACKNOWLEDGMENTS

I would like to thank all people who have supported me to complete this thesis. I would like to express my gratitude to my advisor Dr. Armando Rodriquez for his continuous support of my studies and research; for his invaluable knowledge that I learned from throughout my journey at ASU. Also for giving me the opportunity and the room to work on exciting control topics; and for his help at all times for which I am grateful for.

I would also like to thank my thesis committee members, Dr. Konstantinos Tsakalis and Dr. Spring Berman for their support and help. Further, I would like to thank Dr. Panagiotis Artemiadis for giving me the opportunity to work in the ASU Drone Studio.

I also thank my colleagues at ASU - Shi Lu, Sai Shravan, Nirangkush Das, and Kaustav Mondal for their help and support throughout this journey.

TABLE OF CONTENTS

Page

CHAPTER Page

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

Figure Page

Figure Page

Figure Page

Figure Page

Figure Page

Figure Page

Figure Page

Figure Page

Figure Page

Chapter 1

INTRODUCTION AND MOTIVATION

Rotary-wing vehicles became widely used in the last decade. The is due to the availability of low-cost and small-size microelectronics which allow fast and complex computations to be performed on board. It has been used in many applications: surveillance, transportation, mapping, exploration, building structures within [1, 2], etc. Much has been done in the field of aerial robotics, especially quadrotors. The reason is that because of its symmetry, it simpler and more convenient to work with instead of hexarotor or other configurations. What makes quadrotors an interesting problem is the fact that they are an unstable and under-actuated system, i.e., there are four control inputs and 6-DOF which require some control system to be stable the quadrotor.

1.1 Literature Servery: State of the art

- **Quadrotor Modeling.** Modeling of quadrotors is addressed within [3–5]. A quadrotor consists of four motors - each can be controlled independently by a PWM input signal. The output is the rotor's rotation rate ω. For simplicity, we assume that the motors are identical. The sum of these PWM input signals produces a single collective thrust T which controls the vehicle's altitude z and $\dot{z}$ while the different combination of them create three differential thrusts (torques around 3-axes) which are used to control the vehicles' attitude Θ and $\dot{\Theta}$.

  - *Kinematic Model.* Quadrotor's kinematic model is presented within [6, 7]. Two coordinate systems will be used in this work, the global frame

and body frame. For the vehicle's position, in this work, we consider the Cartesian coordinate system, i.e., Earth-centered Earth-fixed (ECEF) frame, and Euler angles for the vehicle's orientation. The reason we use the global frame is that to control the vehicle's position, we need to know its position relative to the global frame. On the other hand, the body frame is useful for the control inputs where we keep them relative to the body frame for simplicity. A transformation from the body frame to the global frame can be done using transformation matrices, and vice versa. In this work, we consider the Z-Y-X sequence of rotations [1] , i.e., the first rotation around the z-axis, second rotation around the y-axis, and last rotation around the x-axis. Also, rotations are always counterclockwise around their axis which is a convention that is used in this work.

– *Dynamical Model.* Within [6], the author addressed the nonlinear equations of motion of a quadrotor and the assumptions that were taken into consideration to linearize these equations. Within [6, 8, 9], the authors assume to have direct control of forces and moments as inputs (controls) for the system. The reason is that because of the coupling of rotor's rotation rates ω, it is much simpler to control (u_1, u_2, u_3, u_4) independently and then convert to $(\omega_1, \omega_2, \omega_3, \omega_4)$ using a mapping [2] These input forces and moments change the state of the vehicle. Within [6], a single-input-single-output (SISO) model is presented for the inner-loop control.

- **Brushless DC Motor.** A model for the BLDC is addressed within [10]. How-

[1] It is a matter of convention; some papers use Z-X-Y as within [8]. In spite of the convention used, there is a problem of singularity that we need to keep in mind. That is where the quaternion system comes in when we do an estimation of angles of orientation

[2] A linear combination of rotor's rotation rates ω as input force thrust (u_1) and moments around three axes (u_2, u_3, u_4) to the system.

ever, from a controls-perspective, a first-order model for the rotor's rotation rate ω to the commanded rate ω_c is presented within [9]. This model will be used as the actuator dynamics of the plant for the inner-loop control.

- **Classical Controls.** SISO control design ideas are addressed within [11], including pole-placement control design, lead-lag control design, internal model principle, etc.

- **Vehicle Inner-loop Control.** From [12], inner-outer loop control hierarchical structure is used such that inner-loop is associated with faster dynamics while outer-loop is associated with slower dynamics. Within this work, the inner-loop controls the vehicle's angular velocities, namely (p, q, r) which are approximately equal to $(\dot{\phi}, \dot{\theta}, \dot{\psi})$ near hover. This is addressed within [13–15].

- **Vehicle Outer-loop Control.** In this work, the outer-loop controls the vehicle's angle of orientation (ϕ, θ, ψ). Three approaches were addressed within this work: PID Tuning control, Pole-Placement control, and Design for Bandwidth and Robustness. The PID control is addressed within [6], where we obtain the desired performance by tuning the proportional-integral-derivative parameters. On the other hand, with the Pole-Placement, we place the dominant poles at the desired pole locations to meet the design specifications. In control design for bandwidth and robustness, the unity gain crossover frequency and the phase margin are taken into account. Both are addressed within [11].

- **Vehicle Position/Trajectory Following Control.** Within [16], a differential flatness based control for trajectory following is introduced where a full-state feedback LQR controller is designed for the control system. In [17], limitations of differential flatness based control were addressed where the control induces

trajectory bias error. In this thesis, an LQR with Dynamical Augmentation is used to obtain a zero steady-state error in trajectories.

- **Vehicle Formation Control.** Formation control has been used in many applications in aerial vehicles such as cooperative application [18], assembling structures [2], and performing dance [19]. Within [20], quadrotor flight formation control using leader-follower approach is addressed. In this [21], formation flight control of multiple quadrotors with a virtual leader is addressed.

1.2 Contributions

In this work, the following questions will be adderssed.

- **How to build a low-cost quadrotor for a research platform?** With the evolution of technology and fast microcontrollers, it has been possible to build such quadrotors with low-cost off-the-shelf components. Many theses and papers addressed the question such as [13, 22]. There are many commercially available quadrotors that have been used as a research platform. Table 1.1, shows a list of the available quadrotors in market.

Product Name	**Hardware Components**	**Cost**
Parrot AR Drone	On board computer, Wi-Fi, IMU, ultra-sonic sensor, camera	$299.99
DJI Matrice 100 (Guidance)	On board computer, Wi-Fi, IMU, GPS, ultra-sonic sensor, camera	$4548.00
Intel Aero	On board computer, Wi-Fi, IMU, GPS, camera	$1,099.00
AscTec Hummingbird	On board computer, Wi-Fi, IMU, GPS	$4000.00

Table 1.1: Commercially Available Quadrotors

However, in this work, we will present a low-cost quadrotor designed as a research platform for FAME. The following are the components that were used to build the quadrotor which will put together in a PCB designed in [23]:

1. ATG-250 carbon-fiber frame 250mm airframe, has a 250mm wheelbase, 3mm of board thickness, 145 of weight.

Figure 1.1: 250mm Airframe with X Configuration

2. Teensy 3.2 USB-based microcontroller development system (32 Bit, 72 MHz Cortex-M4, 3.3V signals, 5V tolerant, compatible with Arduino libraries and IDE) for inner-loop attitude control.

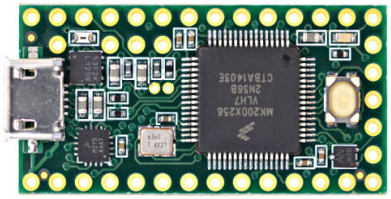


Figure 1.2: Teensy 3.2 Usb-based Microcontroller Development System

3. DJI Snail 2305 Brushless DC motor (2400rpm/V, 23×5mm stator, 27.8g weight) for the propulsion system for the four arms.

Figure 1.3: Dji Snail 2305 Brushless DC Motor

4. DJI Snail 430-R ESC (32-bit 100MHz MCU) as an electronic speed controller (ESC) for each Brushless DC motor.

Figure 1.4: DJI Snail 430-R ESC

5. MPU6050 IMU Digital-output of 6-axis MotionFusion data (2.375-3.46 V power supply, ±2/±4/±8/±12 gauss magnetic, ±2/±4/±6/±8/±16 g linear acceleration, 250/500/2000 dps, 300 to 1100 hPa pressure,) as an inertial measurement unit (IMU).

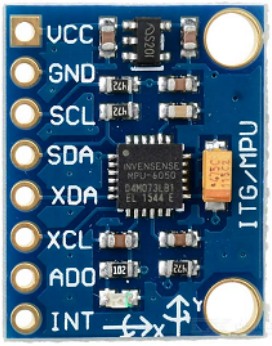


Figure 1.5: MPU6050 IMU

6. Digi XBee3 Zigbee 3.0 (Zigbee, 802.15.4, DigiMesh® and BLE protocols) as a wireless communication module between the flight-controller and the computer station.

Figure 1.6: Digi XBee3 Zigbee 3.0

Product	Quantity	Price ($)
ATG-250 airframe	1	$14.00
Teensy 3.2	1	$27.00
Digi XBee3	2	$36.00
DJI Snail 2305	1	$100.00
IMU-6050	1	$11.00
DJI Snail 5048 Tri-blade	4	$28.00
RadioLink T8FB	1	$45.08
Lipo Battery	1	$30.0
Total Price		**$291.00**

Table 1.2: Hardware Components for Our Low-cost Quadrotor

- **How to design an open-source platform for quadrotors?** In Chapter 2, a description of the quadrotor platform is discussed. The platform was designed using open-source tools, softwares, and operating systems. Mainly, there are three essential components in the platform:

    1. **Ubuntu 16.04 (OS).** The open-source Linux-distribution operating system.
    2. **Robot Operating System (ROS).** Most recent state-of-the-art libraries and tools that were built for robot applications. The system is widely used in many modern research applications such as [8, 9]. In this thesis, ROS Kinetic (10th distribution) is used, which runs on Ubuntu 16.04 (OS). ROS is compatible with HTC Vive Tracking System and OptiTrack Motion Capture System.
    3. **MAVLINK.** An open-source communication protocol that is developed

for micro air vehicle applications which proves to be reliable since 2009 [24]. It is very efficient where 8-bits per packet for overhead. Also, very reliable where it can detect packets drop, corruption, etc. In this thesis, MAVLINK version 1 is used.

4. **HTC Vive Tracking System (HVTS).** A virtual reality (VR) tracking system which runs on SteamVR. In this thesis, two SteamVR base stations version 2.0 were used in $3.0m \times 3.0m$ space to perform the hardware demonstrations. HVTS has a refresh rate of 90Hz with millimeter-accuracy with 7ms latency.

Figure 1.7: Htc Vive Base Station

5. **OptiTrack Motion Capture System.** A real-time tracking system with sub-$20m$ accuracy in optimal conditions. In this thesis, 18 Prime 17W cameras were used in an $11.28m \times 11.28m$ space to perform the hardware demonstrations. OptiTrack Mocap has a refresh rate of 360Hz with sub-millimeter accuracy with 2.8ms latency.

Figure 1.8: OptiTrack Prime 17W Camera

- **What is a suitable inner-loop model and control structure?** In this thesis, the quadrotor is assumed to be performing near the hover position, i.e., attitude angles are small. Therefore, the inner-loop model becomes a Single-Input Single-Output (SISO) system. From [25], a classical decentralized PID controller is used for the inner loop. (See work within Chapter 5)

- **What is a suitable outer-loop model and control structure?** From [12], as a result of a well designed inner-loop controller, the outer-loop model is just the closed loop system T_{ry} of the inner-loop. Therefore, a proportional controller is sufficient to stabilize the outer loop system. (See work within Chapter 5)

- **What is a suitable high-level model and control structure?** After designing a low-level control system, where we have $(\phi^{ref}, \theta^{ref}, r^{ref})$ as a reference commands and (ϕ, θ, r) as outputs, we can treat the outputs of the low-level control system as inputs to the high-level control system. Three different controller designs were presented: (1) LQ Servo, (2) Weighted $\mathcal{H}^{\infty}$ Sensitivity Optimization, (3) LQG/LTRO. (See work within Chapter 6)

1.3 Organization of Thesis

The remainder of the thesis is organized as follows

- Chapter 2 (page 13) describes the quadrotor platform architecture that is used to perform the hardware demonstration in this thesis.
- Chapter 3 (page 18) describes the mathematical preliminaries in the thesis of Earth Centered Earth Fixed Coordinate Frame (ECEF) and Body Coordinate Frame, and Attitude Representation.
- Chapter 4 (page 22) describes the model of Motor Dynamics, Airframe Design, Moment of Inertia, and Vehicle Dynamics.
- Chapter 5 (page 40) describes the low-level control of the inner-outer loop control hierarchical structure for body rate and attitude control.
- Chapter 6 (page 97) describes the high-level control of position/path following of quadrotor.
- Chapter 7 (page 179) describes multiple quadrotor formation control using leader-follwer approach along with hardware demonstration.
- Chapter 8 (page 190) is a summary and directions for future research.

Chapter 2

OVERVIEW OF THE QUADROTOR PLATFORM

2.1 Introduction and Overview

In this chapter, we want to show an overview of the Quadrotor platform that was used in this thesis to command, control, communications, computing (C^4), and sensing (S) as in [12]. The C^4S requirements in this thesis are based on open-source where Ubuntu (OS), Robot Operating System (ROS), MAVLINK communication protocol, and Arduino IDE are used in this work.

2.2 Platform amd C^4S Requirements

In this section, a detailed overview of the platform and the C^4S Requirements will be addressed. The Quadrotor platform architecture is shown in Figure 2.1. The architecture consists of two levels of control: (a) low-level control, (b) high-level control. The low-level control also consists of two levels: (1) inner-loop control, (2) outer-loop control. As in [12], inner-outer loop control hierarchical structure is widely used in robotics applications such that the slower dynamics (outer-loop) to are be followed by faster dynamics (inner-loop) for robust performance.

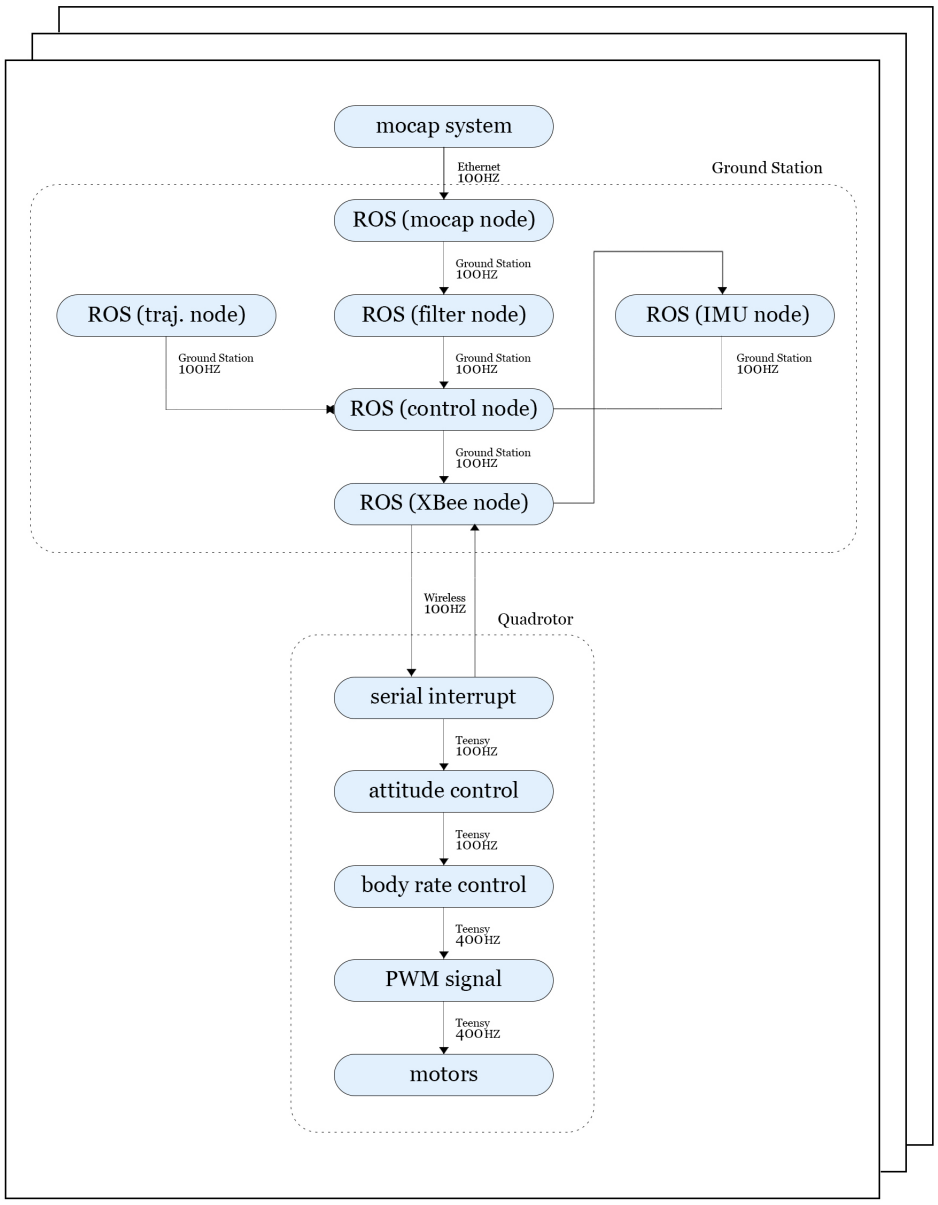


Figure 2.1: Quadrotor Architecture

- **Command, Control, and Computing.** There are two stages of a control command for a quadrotor: (1) Manual mode and (2) Central mode. The Manual mode is the dominant one that can take over when both modes are running at the same time. The reason is that for safety purposes when something goes wrong, the human-person can take control over the quadrotor manually using the Remote Control. The Central mode can be activated by the Remote Control using Channel 5 for the ground station to take control over the quadrotor. Once the Central mode is enabled, commands of (roll, pitch, thrust, and yaw rate) from the ground station will be received by the flight controller. The ground station acts as a central control command which performs all the heavy computations of global sensing, filtering, estimation, trajectory generation, position control, and logging of data while the flight-controller performs only the attitude control. In this work, the attitude control is performed locally on the quadrotor instead of being performed globally on the ground station. The reason is that, as stated before, for safety purposes so that the human-person can switch to Manual mode and be able to control the quadrotor to a safe landing.

  - *Low-Level Control.* As stated above, low-level control consists of an inner and outer loop control hierarchical structure. The outer-loop is associated with the attitude angles (ϕ, θ, ψ) obtained at a rate of 100Hz. When the quadrotor moves forward, the outer-loop outputs the desired angular rates commands of $(p_{des}, q_{des}, r_{des})$ for the inner-loop to follow. The inner-loop is associated with body-rate control where the body-rate measurements (p, q, r) obtained at a rate of 400Hz and the control commands (u_1, u_2, u_3, u_4) computed at the same rate as well to deliver the PWM signals required for each motor. For instance, if we want the quadrotor to tilt

at a certain attitude angle θ. The outer-loop outputs the desired body-rate q for each sample time (0.0025) seconds until the attitude angle reaches θ. Hence, the desired body-rate q is zero at the desired attitude angle θ.

- *High-Level Control.* The high-level control generates the desired attitude angles of the roll (ϕ) and pitch (θ), thrust (T), and yaw rate (r) commands for the low-level control to follow. The high-level computes the desired commands at a rate of 100Hz on the ground station. For instance, if we want the quadrotor to move forward to the x position, the high-level control computes the desired pitch angle θ for the quadrotor for each sample time (0.01) seconds until the quadrotor reaches the desired position.

- **Central Communication.** In this work, communication between quadrotors and ground station is done through XBee wireless modules with one end on ground station and the other on a quadrotor. The communication is a two-way communication where each quadrotor sends (angular rates, angles, and battery voltage) to the ground station and on the other hand, the ground station sends (roll, pitch, thrust, and yaw rate commands) to each quadrotor at a rate of 100Hz. One of the most reliable communication protocols that are widely employed in MAV community is the MAVLINK Micro Air Vehicle Communication Protocol. The reason for using such a protocol is because of its reliability in transmitting data for MAV applications and open-source library. For the 3D motion data, HTC Vive and OptiTrack have different ways of communicating with the ground station. In HTC Vive, data sent wirelessly from HTC Vive Base stations to the ground station with a maximum update rate of 120Hz. However, in OptiTrack Motion Capture System data were sent over direct Ethernet cable with a maximum update rate of 360Hz.

- **Local and Global Sensing.** Each quadrotor is equipped with an IMU, MPU-6050, that can measure the angular body rates along with attitude angles to perform attitude control independently from the ground station. In case of failure in the ground station, a human person can be able to control the quadrotor using a Remote Control in Manual mode. The local sensing for angular rates runs at 400Hz while attitude angles at 100Hz. For global sensing, in this thesis, two global sensing systems were used: (1) HTC Vive Base Stations, (2) OptiTrack Motion Capture System. Both systems provide 3D motion data of positions, velocities, angles, and angular rates in a millimeter accuracy. Also, both systems are compatible with ROS.

Figure 2.2: Quadrotor with Optitrack Motion Markers in ASU Drone Studio

Chapter 3

MATHEMATICAL PRELIMINARIES

3.1 Overview

In this chapter, we present the mathematical preliminaries that are used throughout the thesis. There are many conventions used in aerial vehicles, however, this thesis focuses on one type of convention that is used throughout the thesis.

3.2 Earth Centered Earth Fixed Coordinate Frame and Body Coordinate Frame

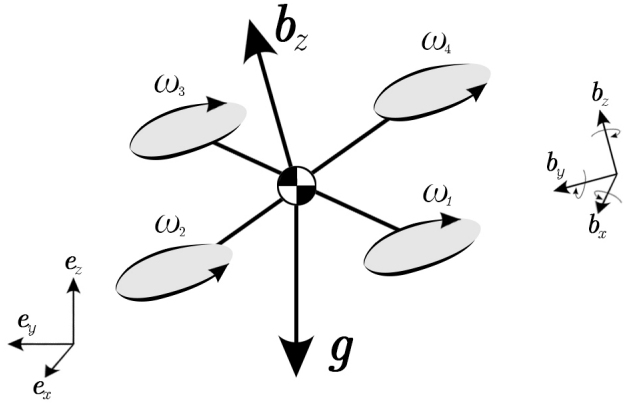


Figure 3.1: Earth Coordinate Frame **e** and Body Coordinate Frame **b**

Earth Centered Earth Fixed Coordinate Frame (ECEF). The most famous representation of position which is a linearized cartesian coordinate where every point in space is represented by (x, y, z). Both x and y represent the lateral distance from the origin which forms a plane tangent to the earth surface while z is perpendicular

to the surface. The right-hand rule is commonly used in aeronautics in order to determine the positive axis in the coordinate frame, i.e. North-East-Down (NED). However, more intuitively, in this thesis, the left-hand rule is considered, i.e. North-East-Up (NEU). Let $\mathbf{e}_i \in \mathcal{R}^3$ for every $i \in [x, y, z]$ such that

$$\mathbf{e}_x = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \quad \mathbf{e}_y = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \quad \mathbf{e}_z = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \tag{3.1}$$

represent a unit vector from the origin along (x, y, z) axes.

Body Coordinate Frame. The origin of the body-frame can be any point on the vehicle. For instance, a sensor can be the origin of the body-frame. Yet, it is often the case in aerial vehicles to have the center of mass of the vehicle as the origin point of the body-frame. Why should a body-frame be used? There are many reasons for using the body-frame: (1) It is more convenient to work with control inputs (u_1, u_2, u_3, u_4) in the body-frame by changing the magnitude of thrust and moments instead of keeping track of the orientation of these vectors (2) It is more intuitive to consider the orientation or attitude of the vehicle as a sequence of finite rotations around a fixed-point. In this work, the rotations are considered to be counter-clockwise around the positive axis.

3.3 Attitude Representation

Euler Angles. The most used attitude or orientation representation is Euler angles. The reason is that it is more intuitive to think of roll (ϕ), pitch (θ), and yaw (ψ) instead of quaternions. The following Euler angles are defined as

- ϕ : Pitch angle is the counter clockwise rotation around the x-axis which is the angle between the $\mathbf{b}_x$ and (x,y) plane;

- θ : Roll angle is the counter clockwise rotation around the y-axis which is the angle between the $\mathbf{b}_y$ and (x,y) plane;
- ψ : Yaw angle is the counter clockwise rotation around the z-axis which is the angle between the projection of $\mathbf{b}_x$ in (x,y) plane and $\mathbf{e}_x$ vector;

The relationship between Euler angle rates $(\dot{\phi}, \dot{\theta}, \dot{\psi})$ and the body rotation rates or body angular velocity (p, q, r) as follows

$$\begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} 1 & 0 & sin(\theta) \\ 0 & cos(\phi) & -sin(\phi)cos(\theta) \\ 0 & sin(\phi) & cos(\theta)cos(\phi) \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \tag{3.2}$$

One of the downsides of using Euler angles is the problem of gimbal lock. This problem occurs when one of the angles goes to 90 deg and as a result we lose one degree of freedom out of 3 degrees of freedom and the matrix becomes singular.

Rotation Matrices. We can represent the orientation of a vehicle from the body frame to the earth frame by a series of rotations. In this work, we consider rotation around z-axis first, then y-axis, and finally x-axis. The rotation matrices are as follows

$$R_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\phi) & -sin(\phi) \\ 0 & sin(\phi) & cos(\phi) \end{bmatrix} \tag{3.3}$$

$$R_y(\theta) = \begin{bmatrix} cos(\theta) & 0 & sin(\theta) \\ 0 & 1 & 0 \\ -sin(\theta) & 0 & cos(\theta) \end{bmatrix} \tag{3.4}$$

$$R_z(\psi) = \begin{bmatrix} cos(\psi) & sin(\psi) & 0 \\ -sin(\psi) & cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.5}$$

such that the new orientation can be computed using

$$\Omega = R_z(\psi)R_y(\theta)R_x(\phi)\dot{\Theta} \tag{3.6}$$

where $\Omega = [p, q, r]^T$ and $\Theta = [\phi, \theta, \psi]$. The rotation matrix R is defined as

$$\begin{aligned} R_B^E &= R_z(\psi)R_y(\theta)R_x(\phi) \\ &= \begin{bmatrix} c(\psi)c(\theta) & c(\phi)s(\psi) + c(\psi)s(\phi)s(\theta) & c(\phi)c(\psi)s(\theta) - s(\phi)s(\psi) \\ -c(\theta)s(\psi) & c(\phi)c(\psi) - s(\phi)s(\psi)s(\theta) & -c(\psi)s(\phi) - c(\phi)s(\psi)s(\theta) \\ -s(\theta) & c(\theta)s(\phi) & c(\phi)c(\theta) \end{bmatrix} \end{aligned} \tag{3.7}$$

where c and s represent cosine and sine, respectively.

Chapter 4

MODELING OF QUADROTOR

4.1 Overview

A quadrotor can vertically take-off and land (VTOL) which differentiated from a fixed-wing aerial vehicle. Also, it has four rotors that generate upward forces when spinning. The four rotors create a symmetry that makes the modeling simpler. Also, because of the symmetry of the four rotors, the anti-torque moments of each rotor will be canceled out by the others. Within [26], different configurations of a quadrotor are presented, and we will consider the X configuration in this work. We will take a look at the parts that assemble the quadrotor separately. The following is a block diagram overview of the quadrotor system

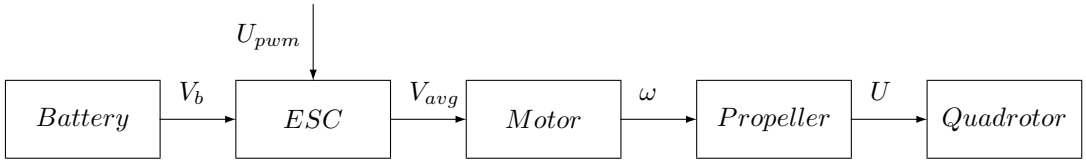


Figure 4.1: Overview of System Block Diagram

when connected directly to DC voltage source as they require commutation and feedback control for speed for commutation and PWM signal that comes from the flight controller is too weak to drive the BLDC motor directly [26, 27]. In addition to the power protection and brake, modeling and designing an ESC is beyond the scope of our research.

4.2 Airframe Design

The air-frame affects the maneuverability of a quadrotor, which means the maximum acceleration [26]. The relationship between both the propeller's thrust T_p and moment M_p with propeller radius r_p as follows

$$\begin{aligned} T_p &= \frac{1}{(2\pi)^2} C_T \rho \omega^2 (2r_p)^4 \\ M_p &= \frac{1}{(2\pi)^2} C_M \rho \omega^2 (2r_p)^5 \end{aligned} \tag{4.1}$$

where C_T and C_M are the thrust and moment coefficients, respectively, ρ is the air density, ω is the propeller's angular velocity, r_p is the propeller's radius. As shown in the equations, the propeller's thrust T_p and moment M_p are proportional to $(2r_p)^4$ and $(2r_p)^5$, respectively. According to [8], if the propeller's radius r_p scales linearly with characteristic length d_p, $(r_p \sim d)$, the following relation approximately holds

$$\begin{aligned} & m \sim d_p^3 \sim r_p^3, \qquad I \sim d_p^5 \sim r_p^5 \\ & a = \dot{v}, \qquad\quad \alpha = \dot{\omega} \\ & a = \frac{\omega^2 r_p^4}{r_p^3} = \omega^2 r_p, \quad \alpha \sim \frac{\omega^2 r_p^5}{r_p^5} = \omega^2 \end{aligned} \tag{4.2}$$

where m is the mass of the quadrotor and I is the moment of inertia, a is the linear acceleration, α is the angular acceleration. From [8, 28], two approaches commonly used to study the scaling of rotor speed with length in aerial vehicles, which are Mach scaling and Froude scaling. From that, Mach scaling and Froude scaling conclude that $\omega \sim \frac{1}{r_p}$ and $\omega \sim \frac{1}{\sqrt{r_p}}$, respectively. The following is the relation with a and α:

$$\begin{aligned} & a \sim \frac{1}{r_p}, \quad \alpha \sim \frac{1}{r_p^2}, \quad Mach \\ & a \sim 1, \quad \alpha \sim \frac{1}{r_p}, \quad Froude \end{aligned} \tag{4.3}$$

and from [26] the relationship between the maximum propeller's radius r_{max} and the airframe radius R_a is

$$r_{max} = R_a sin(\frac{\theta_p}{2}) \tag{4.4}$$

where θ_p is the angle between propellers, in our case 90 degrees, from that, we can see that the linear acceleration has a small effect on the size of the airframe, while the angular acceleration is affected the most by ($\frac{1}{\sqrt{2}R_a}$ to $\frac{1}{2R_a^2}$).

4.3 Moment of Inertia

To determine the moment of inertia of the quadrotor, within [5] addressed the bifilar pendulum theory.

$$I = \frac{mgT_I^2 d_I^2}{4\pi^2 L_I} \tag{4.5}$$

where m is the mass of the quadrotor, T_I is the swing period, d_I is the distance between the two robes, and L_T is the length of the robe. The parameter values found in Table.

Parameter	**Definition**	**Values**
I_{xx}	moment of inertia in x-axis	0.0019005 $Kg.m^2$
I_{yy}	moment of inertia in y-axis	0.0019536 $Kg.m^2$
I_{zz}	moment of inertia in z-axis	0.0036894 $Kg.m^2$

Table 4.1: Nominal Values for Moment of Inertia

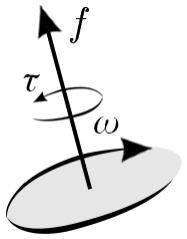


Figure 4.2: Motor Force, Torque, and Rotation Rate

Brushless DC Motor. Within [10], the small size of the quadrotor limited the power sources on batteries which provide a direct current. Consequently, BLDC became popular and widely used in quadrotors. Also, they provide feedback on the rotational speed of the rotor. In Figure 4.3, a simplification for the BLDC model, which is a DC motor, is shown as follows

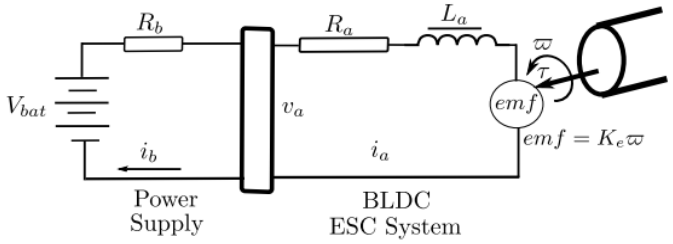


Figure 4.3: Simplified Brush-less DC Motor Model with Battery Source

DC motors convert electrical energy into mechanical energy where we get the motor's rotation/spinning rate ω. The model of the motor consists of two parts: a mechanical part and an electrical part such that

$$MechanicalPart \begin{cases} T - b\dot{\theta} = J\ddot{\theta} \\ T = \frac{1}{k_v} i \end{cases}, \quad ElectricalPart \begin{cases} V - R_a i - L_a \frac{di}{dt} - V_{emf} = 0 \\ V_{emf} = \frac{1}{k_v}\dot{\theta} \end{cases} \tag{4.6}$$

The K_v rating is an essential part of the BLDC. It is defined by $\omega = K_v v$, where ω is the propeller's rotational speed, V is voltage. For small quadrotors with small propellers, we need a high K_v rating while we need low K_v rating with large quadrotors.

Propellers/Rotors. They came in different shapes and number of blades. Within [29], the weakest performance resulted in the one bladed propeller, and the best performance is the two-bladed propeller. However, the three-bladed propeller is not so much different from the two-bladed propeller in terms of performance. Therefore, we considered the three-bladed propeller in our design of the quadrotor. To produce thrust, a propeller is placed on top of the motor. From [6], the relationship the propeller's rotation rate and the force generated is as follows:

$$\begin{aligned} f_i &= k_f \omega_i^2 \\ \tau_i &= k_\tau \omega_i^2 \end{aligned} \tag{4.7}$$

where k_F and k_τ are the thrust and moment coefficients, respectively, obtained from

$$\begin{aligned} k_f &= (\frac{1}{2\pi})^2 C_T \rho \omega^2 (2r_p)^4 \\ k_\tau &= (\frac{1}{2\pi})^2 C_M \rho \omega^2 (2r_p)^5 \end{aligned} \tag{4.8}$$

In [23], the values for k_f and k_τ were already determined as follows

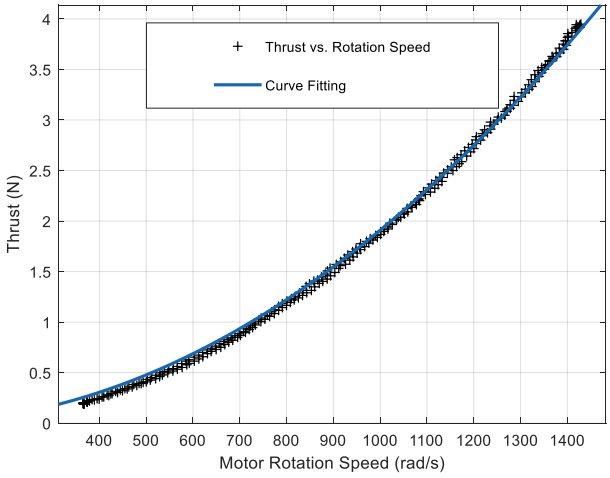


Figure 4.4: Relationship Between Propeller's Angular Velocity and Thrust

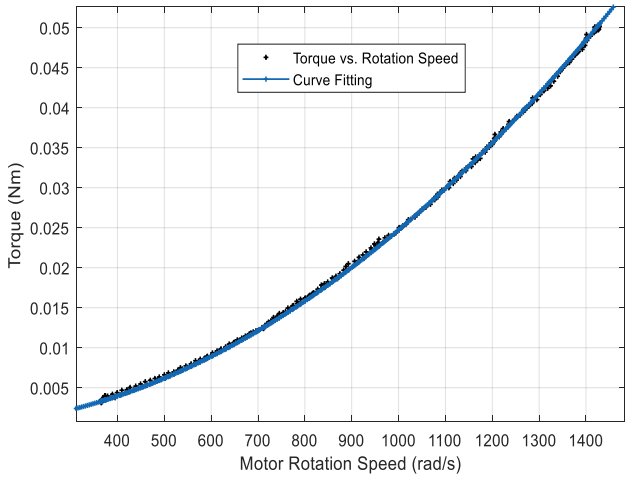


Figure 4.5: Relationship Between Propeller's Angular Velocity and Torque

where the thrust constant and torque constant coefficients are $1.91 \times 10^{-6} N/rad^2$ and $2.47 \times 10^{-8} Nm/rad^2$, respectively.

Practical Issues. This is a question that might come to someone's mind:

Why are we not modeling the electronic speed controllers (ESC) with the dynamics of the motor to obtain a model for the propulsion system?

Because of the complexity of the ESC, we will model the dynamics of a single rotor thrust (ESC, Motor, Propeller) as a first-order system as presented in [13]. In that way, we simplify the propulsion system (ESC, Motor, Propeller) into a first-order system as follows

$$\dot{f} = \frac{1}{\tau}(f^{des} - f) \tag{4.9}$$

The input to the propulsion system is a PWM signal ranges between (300 to 1700), and the output is the propeller's angular velocity w. We only have control over the PWM signal; we cannot control the output voltage from the ESC, as shown in the following block diagram in Figure 4.6.

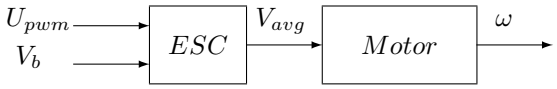


Figure 4.6: Propulsion System Block Diagram

where V_{avg} is the average voltage output from the ESC which is a resultant by battery voltage V_b and PWM signal shown by the relation

$$V_{avg} = U_{pwm} V_b \tag{4.10}$$

then the motor-propeller block gets V_{avg} as an input and outputs $\omega^d es$

$$\omega^{des} = C_b U_{pwm} V_b + \omega_c \tag{4.11}$$

where C_b is a proportional constant, by using a thrust stand and dynamo-meter, we were able to measure the PWM signal U_{pwm} versus the propeller's angular velocity ω at a fixed battery voltage V_b as shown in Figure 4.7.

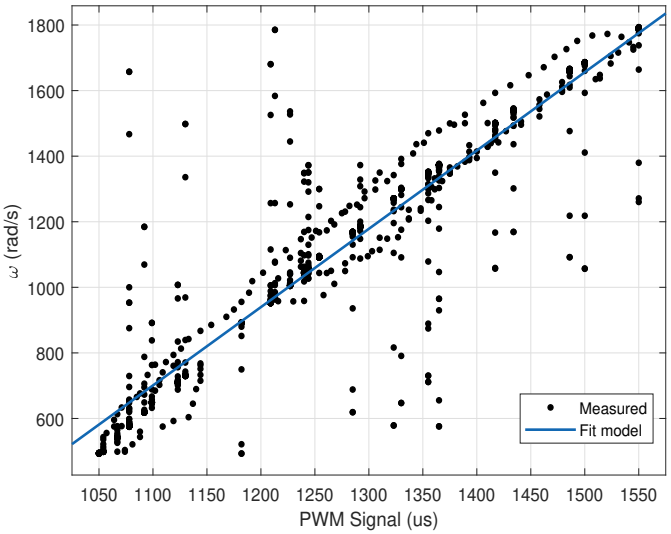


Figure 4.7: Relationship Between PWM Signal and Propeller's Angular Velocity

$$\omega^{des} = 2.388U_{pwm} - 1926 \tag{4.12}$$

the fit model was approximated around the hover position where $\omega = \sqrt{\frac{mg}{4k_f}} \approx 924.75$ (rad/s) for each propeller. We have assumed the battery voltage is fixed and does not change over time; however, in reality, the battery voltage decreases over time, which will result in low motor efficiency. Therefore, a model fit that captures the angular velocity ω with respect to PWM signal U_{pwm} and battery voltage V_b in Figure 4.8 and presented as follows in the equation

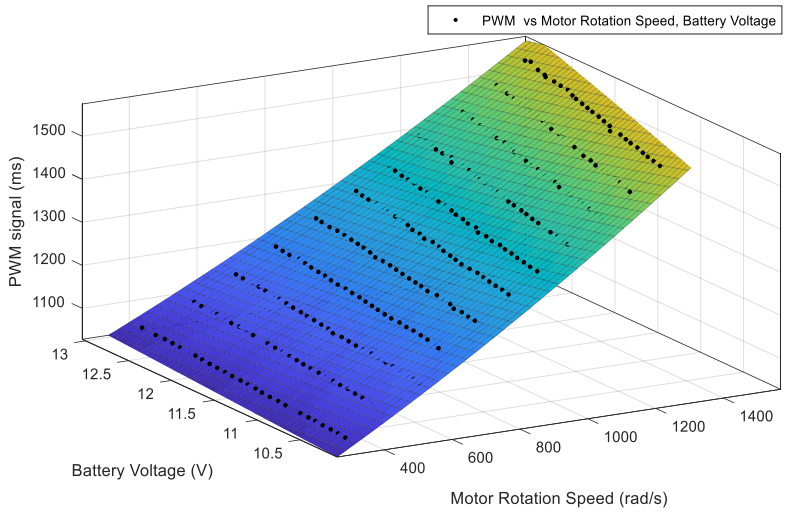


Figure 4.8: Mapping Between Angular Velocity, PWM Signal, and Battery Voltage

$$U_{pwm} = \frac{\omega^2 + 5393\omega + 29960}{1166V_b + 1544} + 895 \tag{4.13}$$

Then, because of the motor dynamics, the output ω cannot be achieved instantaneously without a delay which can be approximated by a first order transfer function as follows

$$\omega = \frac{1}{\tau s + 1}\omega^{des} \tag{4.14}$$

where the time constant τ_m was obtained by performing a step response experiment to find the 90% settling time of the motor, the RCBenchmark Series 1580 Thrust Stand and Dynamometer was used as shown in Figure 4.9.

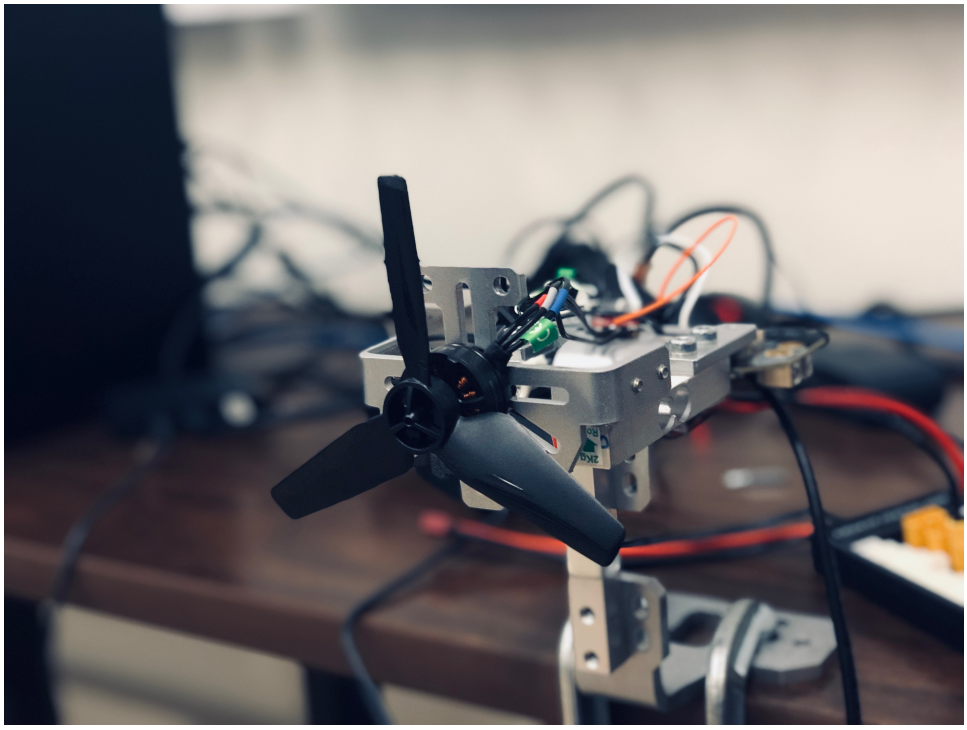


Figure 4.9: RCbenchmark Series 1580 Thrust Stand and Dynamometer

The 90% settling time found to be around $t_s \approx 0.138$ seconds. Therefore, $\tau \approx \frac{t_s}{2.3} = 0.06$ seconds. As a result, the transfer function from the desired angular velocity ω^{des} to the actual angular velocity ω is as follows:

$$G(s) = \frac{16.67}{s + 16.67} \tag{4.15}$$

The System Identification Toolbox in MATLAB was used to validate our model with the measured data. The system identified model obtained from the toolbox is as follows

$$G_{id}(s) = \frac{31.06}{s + 31.64} \tag{4.16}$$

where we can see that the rise time of G_{id} model is closer to the measured data but has steady state error. Therefore, we will be using the G model when designing the

inner-loop control while taking G_{id} model into consideration in terms of the maximum bandwidth of actuator dynamics.

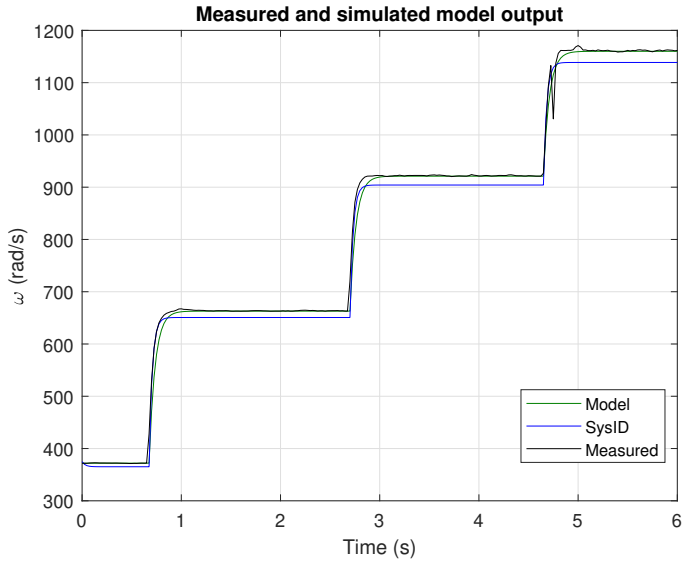


Figure 4.10: Measured and Simulated Model Output for Propulsion System

Parameter	**Definition**	**Values**
k_f	thrust coefficient	$1.91 \times 10^{-6} N/rad^2$
k_τ	moment coefficient	$2.47 \times 10^{-8} Nm/rad^2$
t_s	90% settling time	0.138 s
τ_m	motor's time constant	0.06 s
ω_{min}	minimum propeller's rotation rate	4774 rpm
ω_{max}	maximum propeller's rotation rate	17188 rpm

Table 4.2: Propulsion System Parameter Values

4.5 Vehicle Dynamics

Dynamics are concerned with the motion of the vehicle under the influence of forces. We will study how spinning propellers create forces which influence the vehicle's state of motion.

4.5.1 Nonlinear Dynamical Vehicle Model

The equations of motion are divided into two coordinate frames, global and body frames. The translation dynamics are expressed in the global frame while the rotational dynamics are represented in the body frame as shown in (?) and (?), respectively

$$m\ddot{\zeta} = T(R_B^E)^T e_3 - mg \tag{4.17}$$

where m is the total mass of the vehicle, $\zeta = [x, y, z]^T$ is the position vector in the earth-frame, T is the total upward thrust in the body-frame, $g = [0, 0, 9.81]^T$ is the acceleration vector due to gravity vector

$$I\dot{\Omega} = M - \Omega \times (I\Omega) \tag{4.18}$$

where I is the inertia matrix, $\Omega = [p, q, r]$ is the angular velocity vector, $M = [M_x, M_y, M_z]^T$ is the moment developed around each axis. By putting everything together, we obtain the following nonlinear model for the vehicle

$$\dot{x} = f(x, u) \tag{4.19}$$

$$\begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \\ \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} \frac{T}{m}(-cos(\psi)cos(\phi)sin(\theta) - sin(\psi)sin(\phi))) \\ \frac{T}{m}(cos(\psi)sin(\phi) - cos(\phi)sin(\psi)sin(\theta)) \\ \frac{T}{m}(cos(\phi)cos(\theta)) - g \\ \frac{1}{I_{xx}}(M_x + I_{yy}qr - I_{zz}qr) \\ \frac{1}{I_{yy}}(M_y - I_{xx}pr + I_{zz}pr) \\ \frac{1}{I_{zz}}(M_z + I_{xx}pq - I_{yy}pq) \end{bmatrix} \tag{4.20}$$

4.5.2 Linearization of Nonlinear Dynamical Vehicle Model

Many assumptions were made to obtain the linearized model. The quadrotor is assumed to be near hover position where the thrust magnitude is equal to the force of gravity and the rotation of the vehicle near zero [1] . Therefore, the orientation can be approximated by

$$\begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \tag{4.21}$$

Also, because we are near hover, the body rotation rates (p, q, r) are low. That means, if we have two small numbers multiplied together, we will get a tiny number close to zero, which we can omit such as qr, pr, and pq. As a result, we will end up with the following expression for the rotational dynamics

$$\begin{bmatrix} \ddot{\phi} \\ \ddot{\theta} \\ \ddot{\psi} \end{bmatrix} = \begin{bmatrix} \frac{1}{I_{xx}}M_x \\ \frac{1}{I_{yy}}M_y \\ \frac{1}{I_{zz}}M_z \end{bmatrix} \tag{4.22}$$

We combine the rotational dynamics with actuator dynamics to obtain a MIMO LTI model from applied torque $(\tau_\phi^{des}, \tau_\theta^{des}), \tau_\psi^{des})$ to vehicle's rotational rates $(\dot{\phi}, \dot{\theta}, \dot{\psi})$. The

[1] Roll (ϕ) and pitch (θ) angles are restricted to be near 0 ± 15 degrees.

sixth order MIMO LTI state-space representation is given by

$$\dot{x} = Ax + Bu \qquad\qquad y = Cx + Du \tag{4.23}$$

where $x = [\tau_\phi, \tau_\theta, \tau_\psi, \dot{\phi}, \dot{\theta}, \dot{\psi}]^T$, $y = [\dot{\phi}, \dot{\theta}, \dot{\psi}]^T$, $u = [\tau_\phi^{des}, \tau_\theta^{des}, \tau_\psi^{des}]^T$,

$$A = \begin{bmatrix} -\frac{1}{\tau} & 0 & 0 & 0 & 0 & 0 \\ 0 & -\frac{1}{\tau} & 0 & 0 & 0 & 0 \\ 0 & 0 & -\frac{1}{\tau} & 0 & 0 & 0 \\ \frac{1}{I_{xx}} & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{I_{yy}} & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{I_{zz}} & 0 & 0 & 0 \end{bmatrix} \qquad B = \begin{bmatrix} \frac{1}{\tau} & 0 & 0 \\ 0 & \frac{1}{\tau} & 0 \\ 0 & 0 & \frac{1}{\tau} \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{4.24}$$

$$C = \begin{bmatrix} 0_{3\times3} & I_{3\times3} \end{bmatrix} \qquad\qquad D = \begin{bmatrix} 0_{3\times3} \end{bmatrix} \tag{4.25}$$

where τ is the motor's time constant and (I_{xx}, I_{yy}, I_{zz} are the vehicle's moment of inertia around the x-axis, y-axis, and z-axis, respectively. From that, we can obtain the associated transfer function matrix

$$P(s) = C(sI - A)^{-1}B + D = \begin{bmatrix} P_{11} & P_{12} & P_{13} \\ P_{21} & P_{22} & P_{23} \\ P_{31} & P_{32} & P_{33} \end{bmatrix} \tag{4.26}$$

Nominal Transfer Functions. By taking the nominal parameter values from Table, we compute the numerical values to obtain the following numerical MIMO transfer function matrix:

$$P_{(\dot{\phi},\dot{\theta},\dot{\psi})} = C(sI - A)^{-1}B + D \approx \begin{bmatrix} 5152.6 & 0 & 0 \\ 0 & 5020.5 & 0 \\ 0 & 0 & 2653.1 \end{bmatrix} \frac{1}{s(s + 16.67)} \tag{4.27}$$

where the off-diagonal elements are zero, which means the system is decoupled. Also, assuming that motors are identical. Therefore, we can treat the system as a SISO system for every axis of rotation. Also, numerically the system has no transmission zeros which means we can apply classical SISO controllers with high bandwidth [2] .

Frequency Response Trade Studies: Moment of Inertia and Mass Variations. In Figure 4.11, we see that as a moment of inertia increases, the diagonal magnitudes at all frequencies decrease, and vice versa. As stated before, there is no relationship in the off-diagonal because of the decoupling. As we know, the moment of inertia increases as mass increases, we can have the same relationship for mass variation as well.

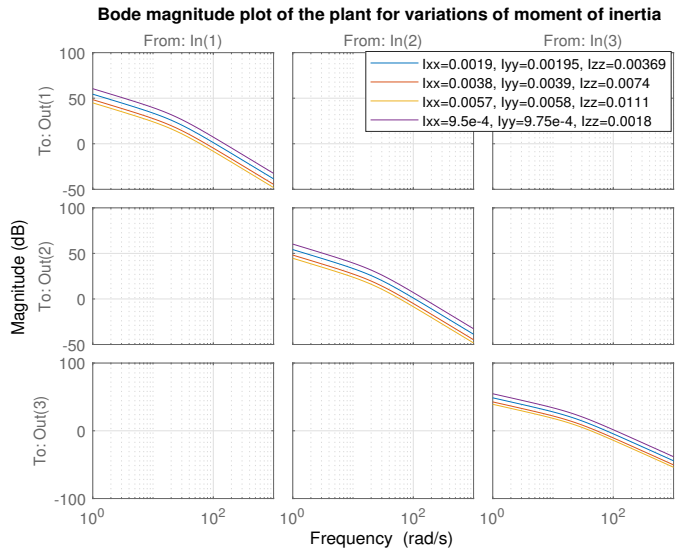


Figure 4.11: Bode Magnitude for Vehicle (Torque to Rotation Rates) - I Variations

Frequency Response Trade Studies: Motor's Time Constant τ. In Figure 4.12, we see that as motor's time constant increases at low frequencies (below say 10 rad/s) the magnitudes do not change. However, above say 10 rad/s the diagonal

[2]Ideally we can have infinity upward gain margin, but we know this is not true in real-world applications because everything has limitations, i.e., actuators have limitations

magnitudes decrease as motor's time constant increase and vice versa.

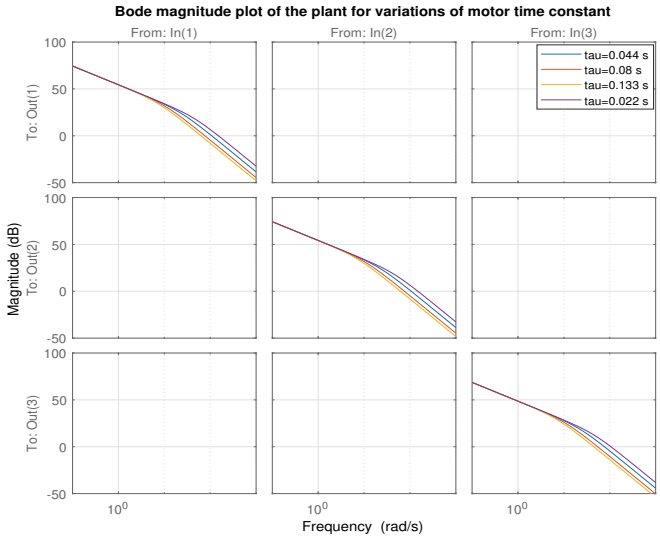


Figure 4.12: Bode Magnitude for Vehicle (Torque to Rotation Rates) - τ Variations

The same thing can be done with the translation dynamics where we linearize around the hover position

$$\begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \\ \ddot{\phi} \\ \ddot{\theta} \\ \ddot{\psi} \end{bmatrix} = \begin{bmatrix} \frac{T}{m}(-cos(\psi)cos(\phi)sin(\theta) - sin(\psi)sin(\phi))) \\ \frac{T}{m}(cos(\psi)sin(\phi) - cos(\phi)sin(\psi)sin(\theta)) \\ \frac{T}{m}(cos(\phi)cos(\theta)) - g \\ \frac{1}{I_{xx}}M_x \\ \frac{1}{I_{yy}}M_y \\ \frac{1}{I_{zz}}M_z \end{bmatrix} \tag{4.28}$$

where $x = [x, y, z, \phi, \theta, \psi, \dot{x}, \dot{y}, \dot{z}, \dot{\phi}, \dot{\theta}, \dot{\psi}]^T$ and $u = [T, M_x, M_y, M_z]^T$.

$$\dot{x} = f(x, u) \tag{4.29}$$

$$x = x_e + \delta x \quad , \quad u = u_e + \delta u \tag{4.30}$$

where $x_e = [x_o, y_o, z_o, 0, 0, 0, 0, 0, 0, 0, 0, 0]^T$ and $u_e = [mg, 0, 0, 0]^T$.

$$\delta\dot{x} \approx f(x_e + u_e) + \left[\frac{\partial f(x,u)}{\partial x}\right]\Big|_{(x_e,u_e)} \delta x + \left[\frac{\partial f(x,u)}{\partial u}\right]\Big|_{(x_e,u_e)} \delta u \tag{4.31}$$

where we can ignore the higher order terms for simplicity. Also, for simplicity, we can treat $\delta x = x$ and $\delta u = u$.

$$\dot{x} = Ax + Bu \qquad\qquad y = Cx + Du \tag{4.32}$$

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & -g & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & g & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \; B = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \frac{1}{m} & 0 & 0 & 0 \\ 0 & \frac{1}{I_{xx}} & 0 & 0 \\ 0 & 0 & \frac{1}{I_{yy}} & 0 \\ 0 & 0 & 0 & \frac{1}{I_{zz}} \end{bmatrix} \tag{4.33}$$

where $A = \left[\frac{\partial f(x,u)}{\partial x}\right]\Big|_{(x_e,u_e)}$ and $B = \left[\frac{\partial f(x,u)}{\partial u}\right]\Big|_{(x_e,u_e)}$.

Controllability of the Linearized Model. From [30], here we ask a fundamental question:

Does there exist a state transferring control $u(t)$ to take us from initial state $x(0)$ to final state $x(t_f)$ in finite time $t_f < \infty$?

This is where we check if the rank of the controllability matrix. The controllability matrix $C(A, B)$ is defined as follows

$$C(A, B) \stackrel{\text{def}}{=} [B \; AB \; \ldots \; A^{n-1}B] \tag{4.34}$$

where n is the number of states, in our case $n = 12$. Computing the rank of $C(A, B)$ yields that the controllability matrix has a full row rank, which means the system (A, B, C, D) is controllable.

Chapter 5

LOW-LEVEL CONTROL: INNER-OUTER LOOP CONTROL SYSTEM DESIGN

5.1 Overview

Our system consists of two parts: a digital part and a continuous part. The reference commands, controller, and sensors are in the digital domain where the plant in the continuous domain. There are two ways to compute digital controllers. The first way is the approximate method where the controller is designed in continuous-time $C(s)$ and then discretized using one of the discretization methods. The other method is the exact method where the controller is designed in the discrete-time $C(z)$ for the discrete-time plant $P(z)$.

There are three standard discretization methods: Forward Euler (FE), Backward Euler (BE), and Tustin.

- FE method: s is approximated by $s = \frac{z-1}{T_s}$. Although, it is simple [1] and preserves the relative degree of transfer functions, however, FE could yield an unstable approximation for stable transfer functions for bad choices of T_s.

- BE method: s is approximated by $s = \frac{1-z^{-1}}{T_s}$. BE yields biproper transfer function with a good approximation up to $\frac{1}{10}$ of Nyquist. However, BE yields a stable digital approximation for stable transfer functions for all T_s. But for unstable continuous-time transfer functions we may end up with a stable digital transfer functions.

[1] In terms of calculation and implementation.

- Tustin method: s is approximated by $s = \frac{2}{T_s}\frac{z-1}{z+1}$. This approximation yields a stable transfer function with a good approximation up to $\frac{1}{3}$ of Nyquist.

Choosing Sample Time T_s. To choose a proper sampling time T_s for our digital implementation, here are some guidelines to follow from [31]:

- Rule 1: Sampling frequency at a fraction of $\frac{1}{10}$ of Nyquist,

$$T_s < \frac{\pi}{\omega_g} \times \frac{1}{10} \tag{5.1}$$

- Rule 2: Six samples per rise time t_r,

$$T_s = \frac{t_r}{6} \tag{5.2}$$

- Rule 3: Phase Margin deterioration of 12 degrees,

$$T_s < \frac{0.2}{\omega_g} \tag{5.3}$$

wherein our design we have a unity gain cross over frequency $w_g \approx 24 rad/s$ and step response rise time $tr \approx 0.05s$. Therefore, in rule 1 we get $T_s < 0.0131s$, rule 2 we get $T_s = 0.0083s$, and rule 3 we get $T_s < 0.008s$. As a result, we have chosen a sampling time $T_s = 0.0025s$ to meet all the three rules with some margin. Because we are dealing with a digital controller, therefore, a delay is presented due to D/A and A/D conversions. This delay needs to be accounted for before designing the controller.

The following block diagram presents the control stages of the system. Starting from controls, propeller's rotation rates, forces and moments generated, and then transnational and angular accelerations.

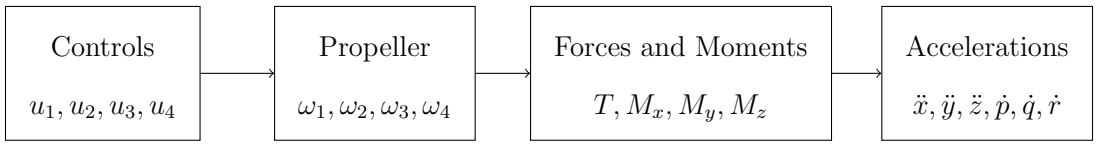


Figure 5.1: Block Diagram for Control Stages

where we have direct control over the propeller's angular rates; however, we will assume we have direct control over translational and angular accelerations from control inputs as illustrated in Figure 5.2. The reason is that we want each controller of

Figure 5.2: Block Diagram from Controls to Accelerations

the four to control four acceleration separately. In other words, $u_1 \rightarrow \ddot{z}$, $u_2 \rightarrow \dot{p}$, $u_3 \rightarrow \dot{q}$, and $u_4 \rightarrow \dot{r}$. Then, by mapping the control inputs to the propeller's angular rates we can directly control propeller's angular rates as in

$$\omega_1 = \sqrt{\frac{\frac{u_1}{k_f} + \frac{u_2}{k_f l} + \frac{u_3}{k_f l} + \frac{u_4}{k_m}}{4}} \tag{5.4}$$

$$\omega_2 = \sqrt{\frac{\frac{u_1}{k_f} - \frac{u_2}{k_f l} + \frac{u_3}{k_f l} - \frac{u_4}{k_m}}{4}} \tag{5.5}$$

$$\omega_3 = \sqrt{\frac{\frac{u_1}{k_f} - \frac{u_2}{k_f l} - \frac{u_3}{k_f l} + \frac{u_4}{k_m}}{4}} \tag{5.6}$$

$$\omega_4 = \sqrt{\frac{\frac{u_1}{k_f} + \frac{u_2}{k_f l} - \frac{u_3}{k_f l} - \frac{u_4}{k_m}}{4}} \tag{5.7}$$

where we can define the control inputs as in

$$\begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} = \begin{bmatrix} T \\ M_x \\ M_y \\ M_z \end{bmatrix} = \begin{bmatrix} f_1 + f_2 + f_3 + f_4 \\ l(f_1 - f_2 - f_3 + f_4) \\ l(f_1 + f_2 - f_3 - f_4) \\ M_1 + M_2 + M_3 + M_4 \end{bmatrix} = \begin{bmatrix} k_F(\omega_1^2 + \omega_2^2 + \omega_3^2 + \omega_4^2) \\ lk_F(\omega_1^2 - \omega_2^2 - \omega_3^2 + \omega_4^2) \\ lk_F(\omega_1^2 + \omega_2^2 - \omega_3^2 - \omega_4^2) \\ k_\tau(\omega_1^2 - \omega_2^2 + \omega_3^2 - \omega_4^2) \end{bmatrix} \tag{5.8}$$

where l is the perpendicular distance from motor to axes which can be found from $l = \frac{\sqrt{2}L}{2}$, where L is the motor to body center distance.

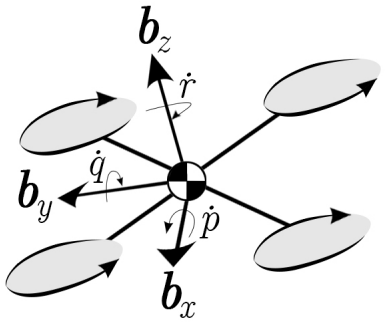


Figure 5.3: Quadrotor Body Rotation Rates (p, q, r) and Accelerations $(\dot{p}, \dot{q}, \dot{r})$

Consider the linearized system for the rotation dynamics

$$\begin{bmatrix} \ddot{\phi} \\ \ddot{\theta} \\ \ddot{\psi} \end{bmatrix} = \mathbf{0}_{3\times 3} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} + \begin{bmatrix} \frac{1}{I_{xx}} & 0 & 0 \\ 0 & \frac{1}{I_{yy}} & 0 \\ 0 & 0 & \frac{1}{I_{zz}} \end{bmatrix} \begin{bmatrix} M_x \\ M_y \\ M_z \end{bmatrix} \tag{5.9}$$

(5.10)

where $C = \boldsymbol{I}_{3\times 3}$ and $D = \mathbf{0}_{3\times 3}$. Therefore, the above system is both controllable and observable. In order to obtain the transfer function matrix of the system we compute

$$P(s) = C(sI - A)^{-1}B + D = \begin{bmatrix} \frac{526.3}{s} & 0 & 0 \\ 0 & \frac{512.8}{s} & 0 \\ 0 & 0 & \frac{271}{s} \end{bmatrix} \tag{5.11}$$

P matrix is diagonal, which means each control input affects one output independently. Therefore, we can treat the system as a Single-Input Single-Output (SISO) system. We will consider one axis $\boldsymbol{b}_x$, and the other can follow the same thing analogously. The transfer function from the input M_x to the output $\dot{\phi}$ is as follows

$$P_{M_x\dot{\phi}}(s) \;=\; \frac{526.3}{s} \tag{5.12}$$

Main Goal. Want y to follow or track r. Typically, reference commands and disturbances are low-frequency signals, while sensor noise is high-frequency signals. Therefore, we want to design K on the basis of nominal model P_o for actual system P [2] such that the tracking error $e_T = r - y$ is small for anticipated r, d_i, d_o, n. Therefore, we want the output to be $y \approx r$. Because our plant is in the analog domain (s-domain) and our controller is a digital controller (z-domain) there exist a DAC to convert digital signals to analog signals. Whenever we see a DAC, a half sample of delay will occur due to Zero-Order-Hold (ZOH), i.e., we lose some phase. Therefore, we want to take the ZOH into account by discretizing the plant $P(s)$ using ZOH with a sample time of $T_s = 0.0025$ $(400\ Hz)$ to obtain $P(z)$. Then, by using the Tustin method to go back to continuous domain $P(s)$ where we end up with

$$P(s) = \frac{-9.51 \times 10^{-5}(s - 800)(s + 1.15 \times 10^5)}{s(s + 16.66)} \tag{5.13}$$

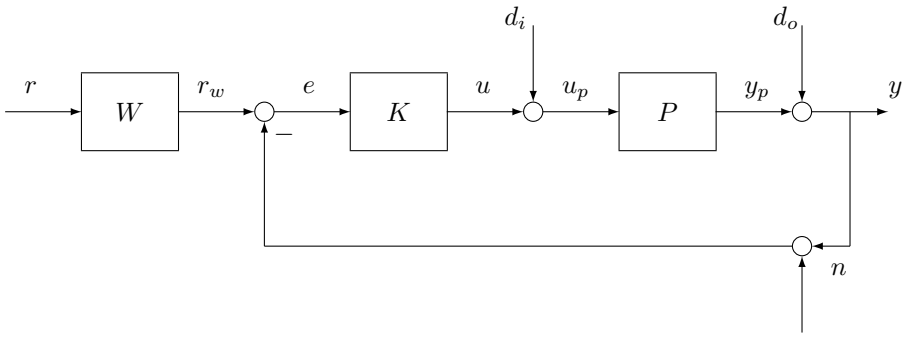


Figure 5.4: Inner-loop Feedback Block Diagram

Control Design and Internal Model Principles. In this method, we follow the ideas of Root Locus and Pole-Placement, as presented within [11]. We define the following specifications: (1) the closed loop system to be stable, (2) exhibits a

[2] Here P represents the actual system which is impossible to know exactly, but we can get an idea of how far away is P_o from the actual P.

zero steady-state error to step reference input r, output disturbances d_o, and input disturbances d_i. Therefore, we need an integrator in the controller to achieve the control design specifications. Also, we know that propeller harmonics start around 500 rad/s to 2000 rad/s, therefore, we want the open loop transfer function L to be low at those frequencies. A good practice is to stay away by a decade below the first propeller harmonic, i.e., the bandwidth need not be higher than 50 rad/s.

5.2.1 Control System Design - PID Tuning

In this design, we tune the three parameters of the Proportional-Integral-Derivative (PID) controller until we achieve a reasonable response. The tuning is done in the discrete-time, and then the controller is converted back to continuous-time to be analyzed. Then, we do a feedback system analysis to examine the open-loop and closed-loop properties of the system. The controller is a Proportional-Integral-Derivative (PID) structure with a high-frequency roll-off term as follows

$$K(s) = \frac{g(s+z_1)(s+z_2)}{s}\left[\frac{a}{s+a}\right] \tag{5.14}$$

where g achieves the unity gain crossover frequency ω_g, and zero at $-z$ meets the desired phase-margin specification. Also, for simplicity, a roll-off was neglected in the controller design because motors, in reality, have roll-off to attenuate high-frequencies where $K(\infty) = 0$. Therefore, $g = 0.0027$, $z_1 = 6.079$, $z_2 = 26.31$ such that

$$K(s) = \frac{0.0027(s+6.079)(s+26.31)}{s}\left[\frac{800}{s+800}\right] \tag{5.15}$$

wherein digital implementation, the integrator windup problem comes into the picture. This is when the integral action of the PID controller keeps accumulating while either the error sign changes or the controller output saturates. Therefore, an integrator anti-windup needed to resolve that issue. One of the anti-windup methods in

[32], suggests that we clamp or not to clamp an integral part of the PID depending on two conditions as follows:

- The sign of the error does not match the sign of the controller output
- The controller output saturates

if both conditions were met, a clamping for integral part will occur. The saturation for u_1 was set to $[0, 15]$ while u_2, u_3, u_4 for $[-1, 1]$.

Analysis of the Feedback System. The analysis of the system will involve using Bode plots, Root Locus, and stability margins. The open loop transfer function is defined as follows

$$L = PK = \frac{-0.00020741(s + 1.152e05)(s - 800)(s + 26.31)(s + 6.079)}{s^2(s + 800)(s + 16.67)} \tag{5.16}$$

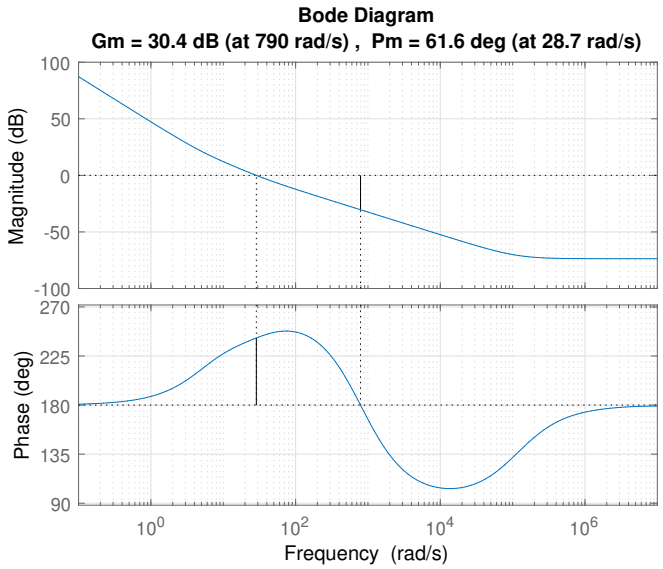


Figure 5.5: Bode Magnitude and Phase Plots for Open Loop L Transfer Function for Inner-loop - PID Tuning Design

Open Loop Frequency Response. By breaking the loop at the error signal, we get the open loop transfer function L. The plot shows large values at low frequencies,

which shows the integrator action. Also, a slope of -20dB/dec at unity crossover frequency for good phase margin. The downward gain margin $\downarrow GM \approx 0$, upward gain margin $\uparrow GM \approx 33$, phase margin $PM \approx 61.5°$, unity gain crossover frequency $\omega_g \approx 28.66\ rad/s$, delay margin $DM \approx 0.0375\ s$. Usually, we want to have good values for $\uparrow GM, \downarrow GM, PM$, but we have no guarantees for stability robustness. Because we might have, but we still have poor peak sensitivity. Therefore, we need to look at the Nyquist plot

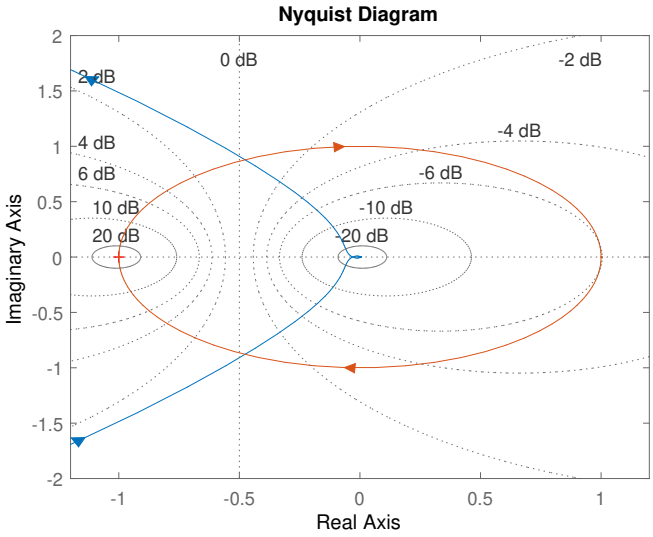


Figure 5.6: Nyquist Plot for Open Loop L Transfer Function for Inner-loop - PID Tuning Design

we can see that Nyquist plot is away from the -1 point where we get; as a result, low peak in the sensitivity transfer function plot. As the plot goes near the -1 point, we might end up with a significant bump in the sensitivity. Also, we have zero closed-loop unstable poles according to

$$P_{u,cl} = P_{i,ol} + N_{cw} = 0 \tag{5.17}$$

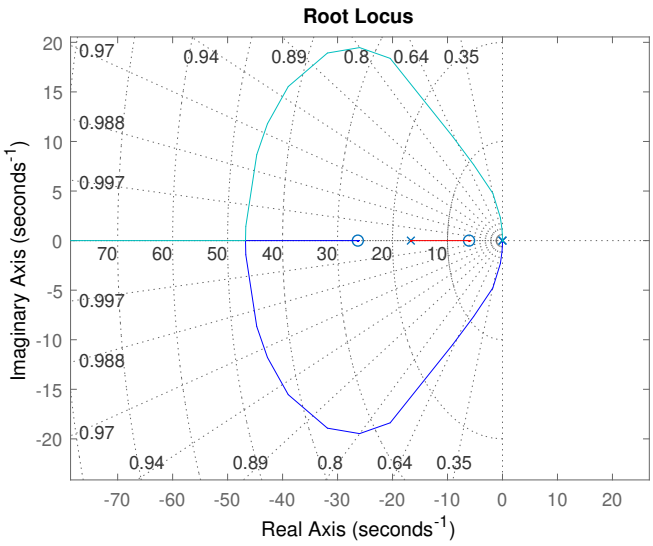


Figure 5.7: Root Locus for Open Loop L Transfer Function for Inner-loop at Low Frequencies - PID Tuning Design

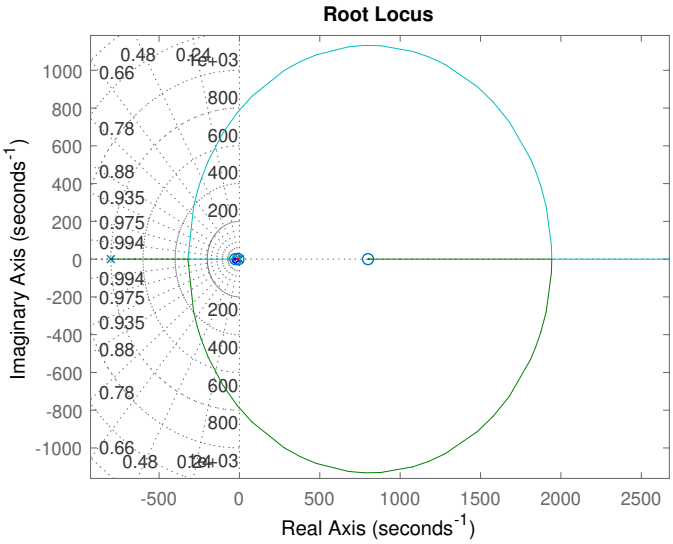


Figure 5.8: Root Locus for Open Loop L Transfer Function for Inner-loop at High Frequencies - PID Tuning Design

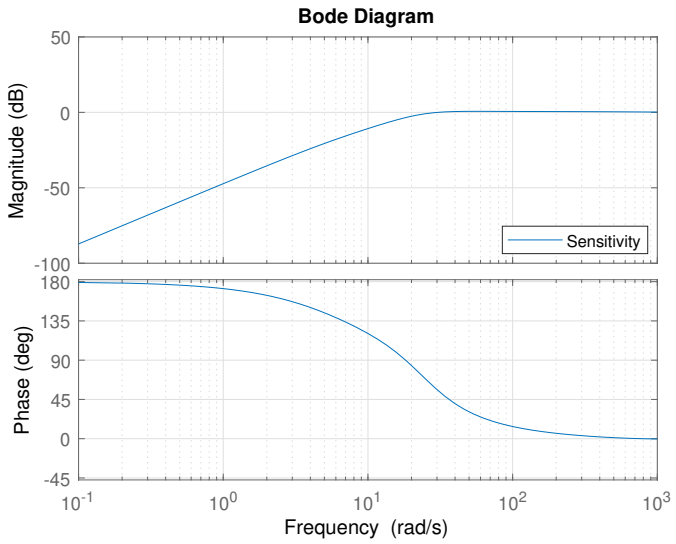


Figure 5.9: Sensitivity $|S|$ Bode Plot for Inner-loop - PID Tuning Design

Sensitivity Frequency Response. From the sensitivity transfer function plot, in general, we want Bode plot magnitude to look small at low frequencies for good low-frequency reference command following and good low-frequency output disturbance attenuation.The frequency at which the Bode magnitude of the sensitivity equals -20dB (0.1) is $w_l \approx 5.23$ rad/s. That is a good definition for bandwidth, which means reference commands will be followed within a 10% error.

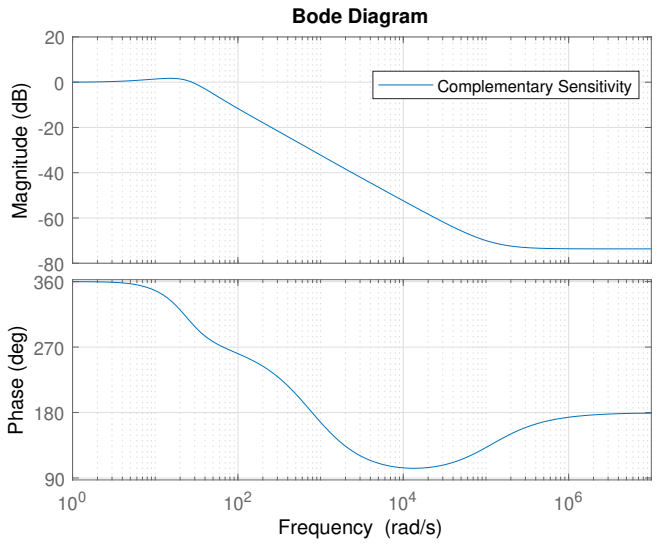


Figure 5.10: Complementary Sensitivity $|T|$ Bode Plot for Inner-loop - PID Tuning Design

Complementary Sensitivity Frequency Response. From the complementary sensitivity plot, in general, we want Bode plot magnitude to look small at high frequencies for good high-frequency noise attenuation. The frequency at which the Bode magnitude of the complementary sensitivity equals -20dB (0.1) is $w_h \approx 254$ rad/s which means noise will be attenuated above w_h by a factor of 10 and the closed-loop system is robust with respect to high-frequency uncertain dynamics. Also, we have a peak of 1.7dB at $\omega = 15$ rad/s, which suggests that step reference commands will result in an overshoot. We can obtain the lower bounds for the sensitivity $||S||_{\mathcal{H}^\infty}$ and complementary sensitivity $||T||_{\mathcal{H}^\infty}$ using the following

$$\begin{aligned} ||S||_{\mathcal{H}^\infty} &\geq max\left\{ \frac{\uparrow GM}{\uparrow GM - 1}, \ \frac{\downarrow GM}{1 - \downarrow GM}, \ \frac{1}{2\ sin(\frac{|PM|}{2})} \right\} = \left\{ 1.0313, \ 0, \ 0.9771 \right\} \\ ||T||_{\mathcal{H}^\infty} &\geq max\left\{ \frac{1}{\uparrow GM - 1}, \ \frac{1}{1 - \downarrow GM}, \ \frac{1}{2\ sin(\frac{|PM|}{2})} \right\} = \left\{ 0.0312, \ 1, \ 0.9771 \right\} \end{aligned} \tag{5.18}$$

Time Domain Analysis. The closed loop system from reference commands r to

Poles	Damping	Frequency (rad/sec)
-7.12e+00	1.00e+00	7.12e+00
-1.70e+01 + 1.68e+01i	7.10e-01	2.39e+01
-1.70e+01 - 1.68e+01i	7.10e-01	2.39e+01
-7.52e+02	1.00e+00	7.52e+02

Table 5.1: Closed Loop Poles for the Inner-loop - PID Tuning Design

the output y is the transfer function $T_{ry} = \frac{WPK}{1+PK}$ as follows:

$$T_{ry} = \frac{25.42(s+6.079)(s+26.31)}{(s+7.115)(s^2+33.93s+571.4)} \tag{5.19}$$

Step reference commands were applied to obtain the closed loop system step response.

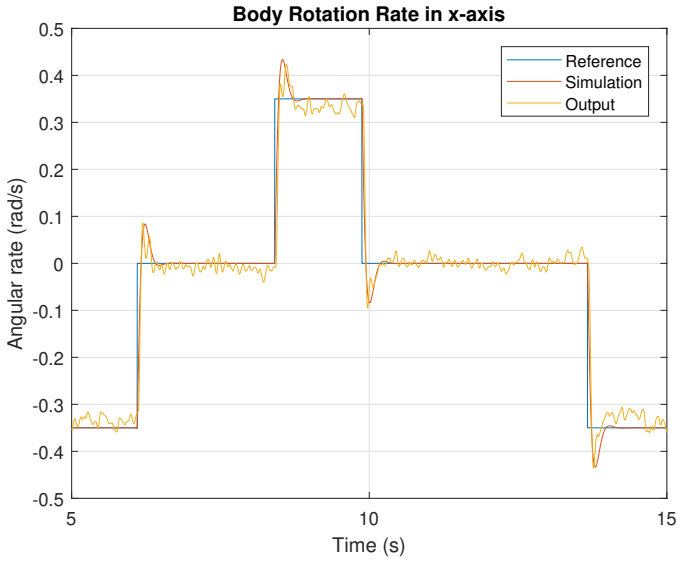


Figure 5.11: Inner-loop Step Response for p Reference Command - PID Tuning Design

Parameters	**Without Prefilter**
Rise Time	0.0445 s
Settling Time	0.2902 s
Overshoot	19.49%
Undershoot	0.02%
Peak	1.1949

Table 5.2: Closed Loop Step Response for the Inner-loop - PID Tuning Design

5.2.2 Control System Design - Pole Placement

From [11], in this design, the controller K will be designed such that the closed loop system step response properties of settling time, overshoot, etc are met by placing the poles at certain locations. We define the following specifications for the closed-loop system: (1) the closed loop system to be stable, (2) exhibits a zero steady-state error to step reference input r, output disturbances d_o, and input disturbances d_i, (3) exhibits a settling time to step reference commands r of $t_s \approx 0.5s$ (4) overshoot below 10%, i.e. $M_p = 0.1$. The controller K has the following structure:

$$K(s) = \frac{g(s+z)(s+16.67)}{s}\left[\frac{200}{s+200}\right]^2 \tag{5.20}$$

where g and z are the two parameters that need to be designed to achieve the design specifications. Furthermore, we cancel the actuator dynamic's pole at 16.67 by introducing a zero in K at the same location. Additionally, a pole at the origin, an integrator, to exhibit a zero steady-state error to step di because the plant has an additional integrator to obtain a zero steady-state error to step reference commands, from internal model principle ideas. Furthermore, a roll-off of almost a decade above

the unity gain crossover frequency to attenuate high-frequencies where $K(\infty) = 0$.

$$\tau = \frac{t_s}{5}, \qquad Re\{Pole\} = \frac{-1}{\tau} \tag{5.21}$$

$$\zeta = \frac{-ln(M_p)}{\sqrt{ln(M_p)^2 + \pi^2}}, \qquad \theta = asin(\zeta)\frac{180}{\pi} \tag{5.22}$$

$$Im\{Pole\} = \frac{|Re\{Pole\}|}{|tan(\theta \frac{\pi}{180})|} \tag{5.23}$$

where the characteristic equation is as follows

$$char = s^2 + 2\sigma s + \sigma^2 + \omega_n^2 \tag{5.24}$$

where $\sigma = -Re\{Pole\}$ and $\omega_n = Im\{Pole\}$.

$$g = \frac{2\sigma}{8.77 \times 10^3}, \qquad z = \frac{\sigma^2 + \omega_n^2}{g \times 8.77 \times 10^3} \tag{5.25}$$

where $g = 0.0023$ and $z = 14.3076$ and we end up with the following K:

$$K(s) = \frac{0.0023(s + 14.3076)(s + 16.67)}{s}\left[\frac{200}{s + 200}\right]^2 \tag{5.26}$$

Open Loop Frequency Response. By breaking the loop at the error e or the plant output y, we obtain the open loop transfer function $L = PK$ as follows:

$$L(s) = \frac{-0.0086806(s + 1.152e05)(s - 800)(s + 16.67)(s + 14.31)}{s^2(s + 200)^2(s + 16.66)} \approx \frac{20(s + 14.31)}{s^2} \tag{5.27}$$

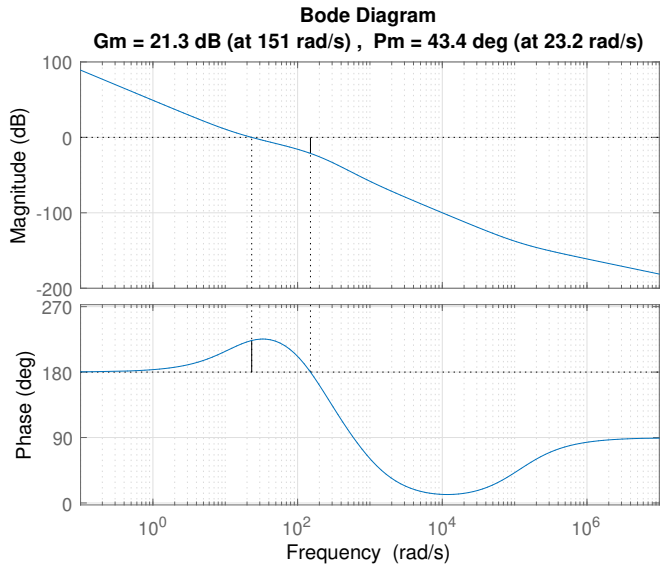


Figure 5.12: Bode Magnitude and Phase Plots for Open Loop L Transfer Function for Inner-loop - Pole Placement Design

where we have a high gain at low frequencies due to integral action along with an increase in the phase around crossover ω_g for closed-loop stability. The unity gain crossover $\omega_g = 23.2$ rad/s and phase margin $PM = 43.3°$. The downward gain margin $\downarrow GM \approx 0$, upward gain margin $\uparrow GM \approx 11.55$, delay margin $DM \approx 0.032s$. Usually, we want to have good values for $\uparrow GM, \downarrow GM, PM$, but we have no guarantees for stability robustness. Because we might have, but we still have poor peak sensitivities. Therefore, we need to look at the Nyquist plot as follows

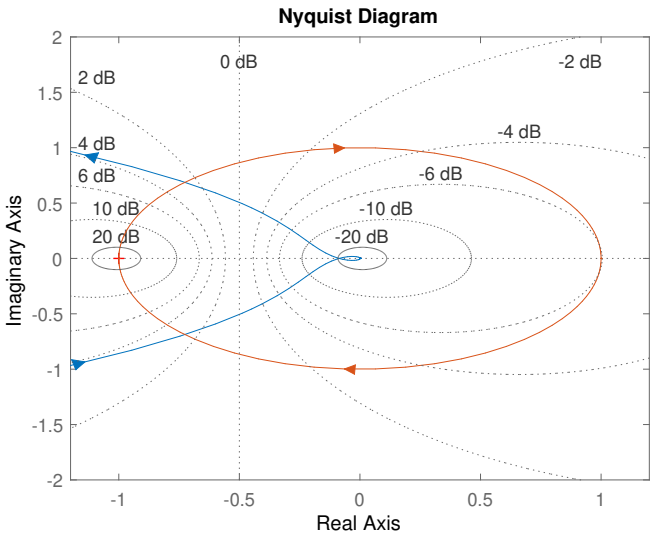


Figure 5.13: Nyquist Plot for Open Loop L Transfer Function for Inner-loop - Pole Placement Design

the plot suggests that we are quite away from the -1 point, i.e., the distance from instability point. The closer the plot to -1 point implies that we will have a large sensitivity peak $|S|$ which is not desirable. Also, we have zero closed-loop unstable poles according to

$$P_{u,cl} = P_{i,ol} + N_{cw} = 0 \tag{5.28}$$

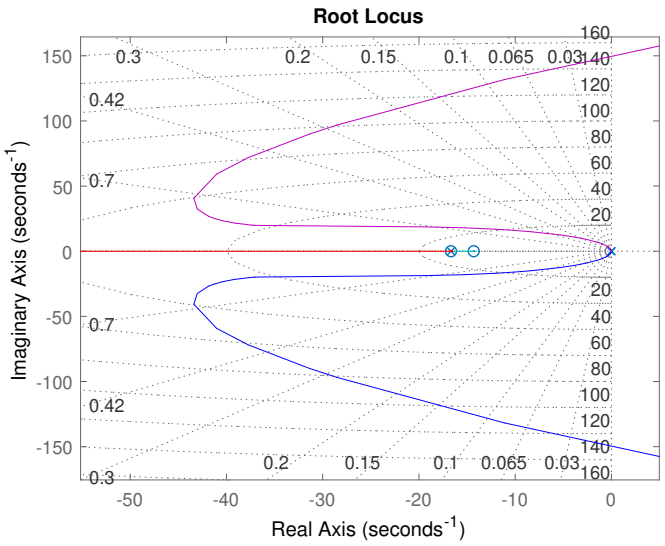


Figure 5.14: Root Locus for Open Loop L Transfer Function for Inner-loop at Low Frequencies - Pole Placement Design

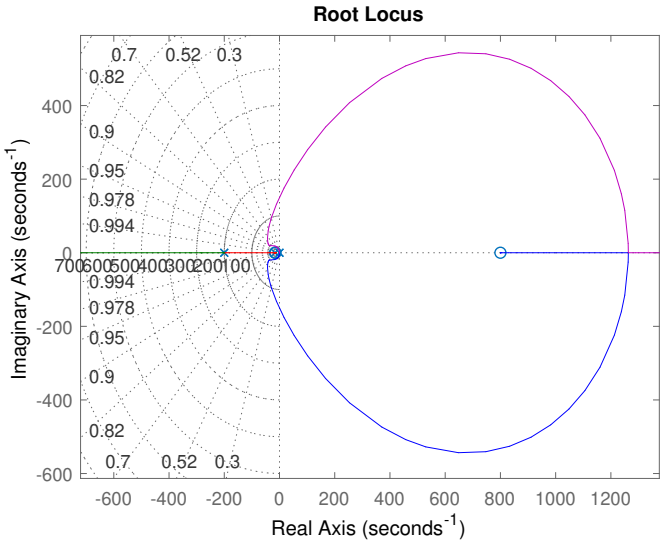


Figure 5.15: Root Locus for Open Loop l Transfer Function for Inner-loop at High Frequencies - Pole Placement Design

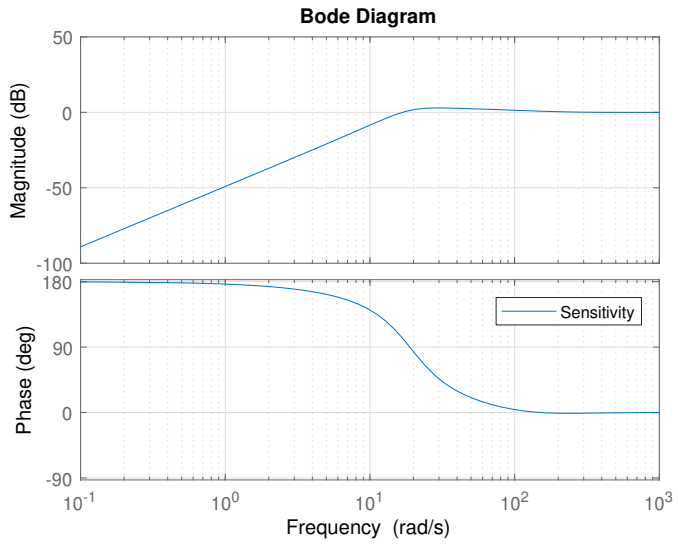


Figure 5.16: Sensitivity $|S|$ Bode Plot for Inner-loop - Pole Placement Design

Sensitivity Frequency Response. The sensitivity transfer function $T_{re} = S$ is from r to e which we want to look like a zero at low frequencies. In other words, the impact of reference commands r on the error e should be zero; otherwise we will end up with large error values in e. The plot magnitude looks small at low frequencies for good low-frequency reference command following and good low-frequency output disturbance attenuation. The frequency at which the Bode magnitude of the sensitivity equals -20dB (0.1) is $w_l \approx 5.28$ rad/s. That is a good definition for bandwidth, which means reference commands with frequency content below 5.28 rad/s will be followed within 20dB, i.e., with a 10% steady-state error. Similarly, with the same amount output disturbances d_o with frequency content below 5.28 rad/s will be attenuated as well. The peak in the sensitivity is small, around 2dB, that is almost negligible.

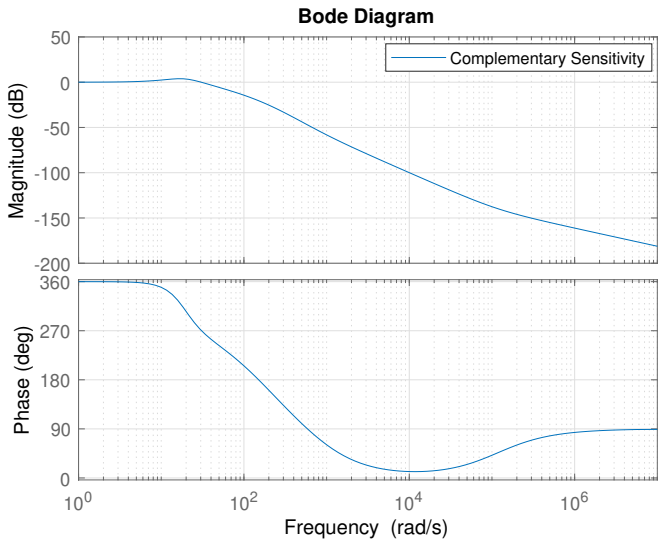


Figure 5.17: Complementary Sensitivity $|T|$ Bode Plot for Inner-loop - Pole Placement Design

Complementary Sensitivity Frequency Response. The complementary sensitivity transfer function T is from r to y or d_i to u, which we want to look like unity at low frequencies and zero at high frequencies. In other words, we want the reference commands r to show up in the plant output y. The plot suggests that low-frequency content reference commands will be followed to some point. Also, the presence of a bump of 3.83dB in T suggests that an overshoot will be present to step reference commands r. Therefore, a prefilter W might be needed to reduce excessive overshoot in step reference commands r. Therefore, a first-order prefilter W with pole close to the controller zero 14.3 is as follows:

$$W(s) = \frac{14.3076}{s + 14.3076} \tag{5.29}$$

where the complementary sensitivity transfer function $T = \frac{WPK}{1+PK}$ looks like

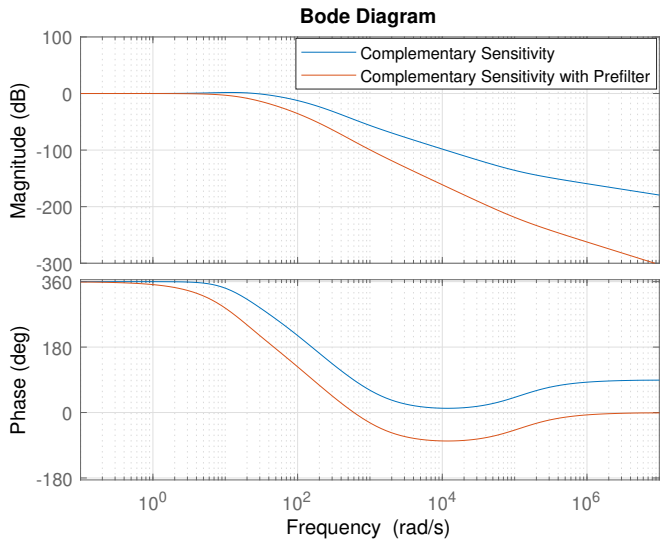


Figure 5.18: Complementary Sensitivity $|T|$ with Prefilter W Bode Plot for Inner-loop - Pole Placement Design

where the prefilter reduces the overshoot significantly., also, the plot shows that high-frequency sensor noise n with content above 146 rad/s will be attenuated within 20dB. However, with the prefilter, high-frequency sensor noise n with content above 58 rad/s will be attenuated within 20dB. We can obtain the lower bounds for the sensitivity $||S||_{\mathcal{H}^\infty}$ and complementary sensitivity $||T||_{\mathcal{H}^\infty}$ using the following

$$\begin{aligned} ||S||_{\mathcal{H}^\infty} &\geq max\left\{\frac{\uparrow GM}{\uparrow GM-1}, \frac{\downarrow GM}{1-\downarrow GM}, \frac{1}{2\ sin(\frac{|PM|}{2})}\right\} = \left\{1.0948,\ 0,\ 1.3552\right\} \\ ||T||_{\mathcal{H}^\infty} &\geq max\left\{\frac{1}{\uparrow GM-1}, \frac{1}{1-\downarrow GM}, \frac{1}{2\ sin(\frac{|PM|}{2})}\right\} = \left\{0.0948,\ 1,\ 1.3552\right\} \end{aligned} \tag{5.30}$$

Time Domain Analysis. The closed loop system from reference commands r to the output y is the transfer function $T_{ry} = \frac{WPK}{1+PK}$ as follows:

$$T_{ry} = \frac{-0.1242(s+1.152e05)(s-800)(s+16.67)}{(s+261.8)(s+117.1)(s+16.66)(s^2+21.01s+373.3)} \tag{5.31}$$

Step reference commands were applied to obtain the closed loop system step response as follows

Pole	Damping	Frequency (rad/s)	Time Constant (s)
-1.43e+01	1.00e+00	1.43e+01	6.99e-02
-1.67e+01	1.00e+00	1.67e+01	6.00e-02
-1.05e+01 + 1.62e+01i	5.44e-01	1.93e+01	9.52e-02
-1.05e+01 - 1.62e+01i	5.44e-01	1.93e+01	9.52e-02
-1.17e+02	1.00e+00	1.17e+02	8.54e-03
-2.62e+02	1.00e+00	2.62e+02	3.82e-03

Table 5.3: Closed Loop Poles for the Inner-loop - Pole Placement Design

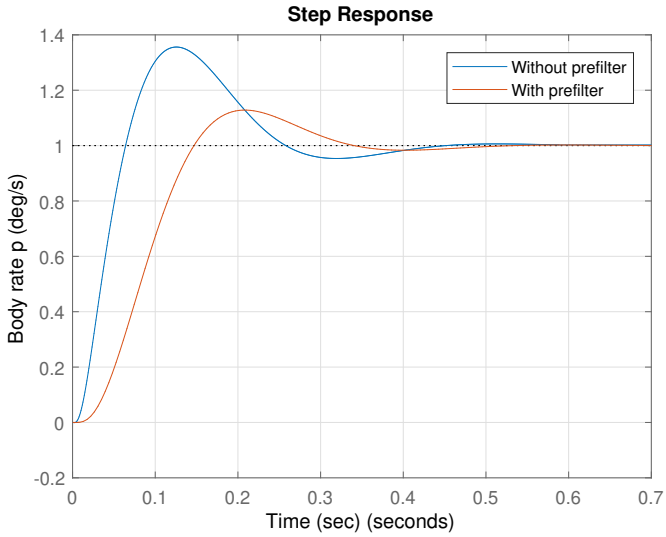


Figure 5.19: Inner-loop Step Response for p with and Without a Prefilter W - Pole Placement Design

where the prefilter almost retrieved the closed-loop system design specifications and reduces the excessive overshoot in the closed-loop step response.

Parameters	**Without Prefilter**	**With Prefilter**
Rise Time	0.0427 s	0.0914 s
Settling Time	0.3947 s	0.3151 s
Overshoot	35.59%	12.81%
Undershoot	0.06%	0.0%
Peak	1.3560	1.1282

Table 5.4: Closed Loop Step Response for the Inner-loop - Pole Placement Design

Time Domain Trade Studies: Varying the Settling Time t_s Design Parameter. In Figure 5.19, we see that as t_s increases, the response speed decreases because we are shooting for a lower settling time as we increase t_s. However, in practical implementation, as t_s goes below 0.3 s the system will experience large oscillations due to pushing the system too much. Therefore, we have chosen the design parameter t_s to be 0.5 s to avoid the oscillations and end up with a more stable system without pushing the system.

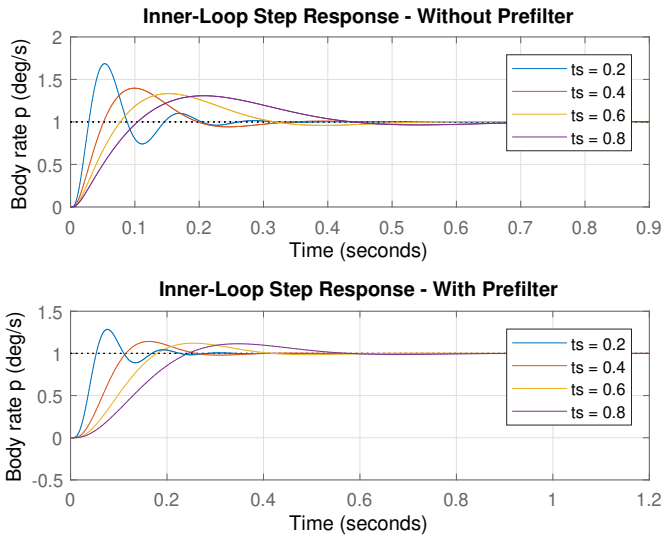


Figure 5.20: Inner-loop Step Response for Different t_s Design Parameters - Pole Placement Design

Time Domain Trade Studies: Varying the Overshoot M_p Design Parameter. In Figure 5.20, we see that as M_p increases, the response speed increases (faster response) with more significant overshoot as a trade-off. Therefore, there is a trade-off between how fast we want the system to behave versus the percentage overshoot that we get. Consequently, we have chosen the design parameter M_p to be 0.1 (10%) overshoot as a reasonable response.

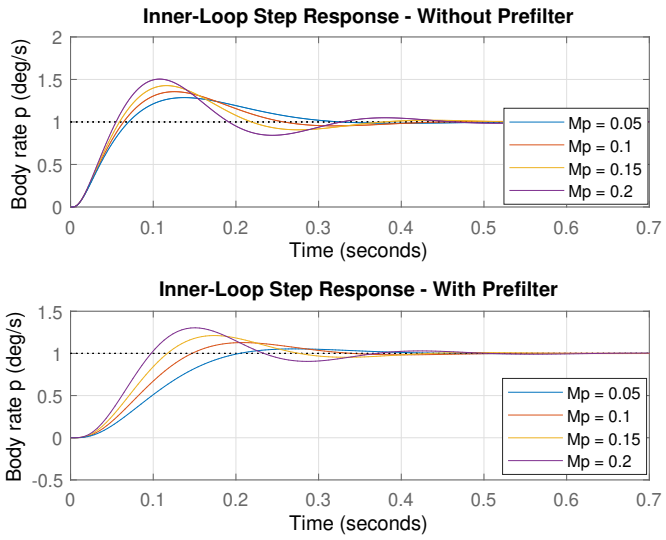


Figure 5.21: Inner-loop Step Response for Different M_p Design Parameters - Pole Placement Design

5.2.3 Control System Design - Design for Bandwidth and Robustness

Design for Bandwidth and Robustness. In this design, we do the opposite where we design a controller to meet the desired specifications in the closed-loop system. From [30], phase margin PM, and the unity gain crossover frequency ω_g will be taken beforehand into consideration when we design the controller. The reason is that, in reality, we do not know how to place poles at certain locations to achieve the desired performance. However, what we can do is that we design a controller to achieve a certain bandwidth and phase margin by loop shaping. In loop shaping L, we want high gain at low frequencies for good low-frequencies command following, good low-frequencies output disturbance attenuation, and large phase around the crossover frequency. We define the following specifications for the closed-loop system: (1) the closed loop system to be stable, (2) exhibits a zero steady-state error to step reference input r, output disturbances d_o, and input disturbances d_i, (3) unity gain crossover

$\omega_g \approx 25$ rad/s (4) phase margin $PM \approx 60$. Therefore, Proportional-Derivative (PD) controller is not sufficient to achieve the defined specifications because to reject step input disturbances d_i we need at least an integrator in the controller K. While it is sufficient to have an integrator in L to reject step output disturbances d_o. Therefore, an integrator in K is necessary to reject both d_i and d_o. Also, because the first propeller harmonic is around 500 rad/s, we need the bandwidth to be at least a decade below that, i.e., maximum reasonable bandwidth of 50 rad/s. Also, from optimal control, a 60 deg phase margin is sufficient to have nice closed-loop properties and robustness. Therefore, K has the following structure:

$$K(s) = \frac{g(s+z)(s+16.67)}{s}\left[\frac{200}{s+200}\right]^2 \tag{5.32}$$

where g achieves the unity gain crossover frequency ω_g, and a zero at z meets the desired phase-margin specification. Furthermore, we cancel the actuator dynamic's pole at 16.67 by introducing a zero in K at the same location. Also, a pole at the origin, an integrator, to exhibit a zero steady-state error to step di because the plant has an additional integrator to obtain a zero steady-state error to step reference commands, from internal model principle ideas. Additionally, a roll-off of almost a decade above the unity gain crossover frequency to attenuate high-frequencies where $K(\infty) = 0$.

$$PM = 180^\circ + \angle P(j\omega_g)^\circ + \angle K(j\omega_g)^\circ \tag{5.33}$$

$$|L(j\omega_g)| = 1 \tag{5.34}$$

$$z = \frac{\omega_g}{tan(PM^\circ - 180^\circ - \angle P(j\omega_g)^\circ + 90^\circ + 2atan(\frac{\omega_g}{200})^\circ - atan(\frac{\omega_g}{16.67})^\circ)} \tag{5.35}$$

$$g = \frac{\omega_g(\omega_g^2 + 200^2)}{|P(j\omega_g)|200^2\sqrt{\omega_g^2 + z^2}\sqrt{\omega_g^2 + 16.67^2}} \tag{5.36}$$

where we end up with $g = 0.0028$ and $z = 7.0482$. The resulted controller is as follows

$$K(s) = \frac{0.0025(s + 7.048)(s + 16.67)}{s} \left[\frac{200}{s + 200}\right]^2 \tag{5.37}$$

Open Loop Frequency Response. By breaking the loop at the error e or the plant output y, we obtain the open loop transfer function $L = PK$ as follows:

$$L(s) = \frac{-0.01(s + 1.15 \times 10^5)(s - 800)(s + 16.67)(s + 7.048)}{s^2(s + 200)^2(s + 16.66)} \approx \frac{24.4316(s + 7.048)}{s^2} \tag{5.38}$$

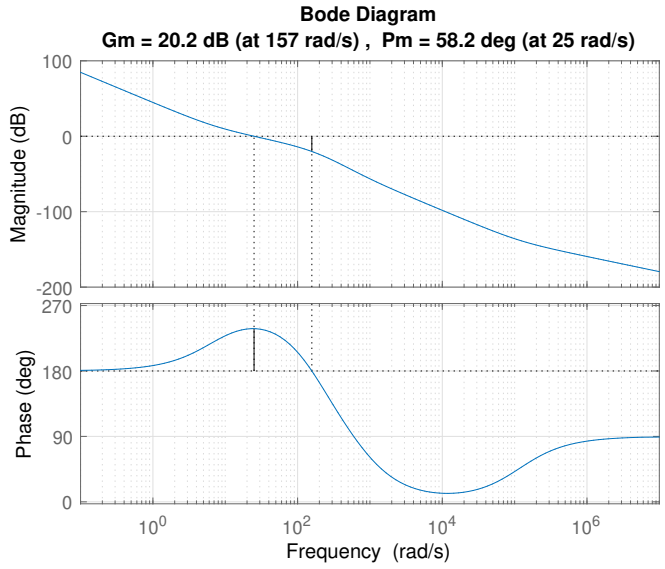


Figure 5.22: Bode Magnitude and Phase Plots for Open Loop L Transfer Function for Inner-loop - Bandwidth and Robustness Design

where we have a high gain at low frequencies due to integral action along with an increase in the phase around crossover ω_g for closed-loop stability. The unity gain crossover $\omega_g = 25$ rad/s and phase margin $PM = 58°$ were closely met as expected. The downward gain margin $\downarrow GM \approx 0$, upward gain margin $\uparrow GM \approx 10.2$, delay margin $DM \approx 0.04s$. Usually, we want to have good values for $\uparrow GM, \downarrow GM, PM$, but we have no guarantees for stability robustness. Because we might have, but we

still have poor peak sensitivities. Therefore, we need to look at the Nyquist plot as follows

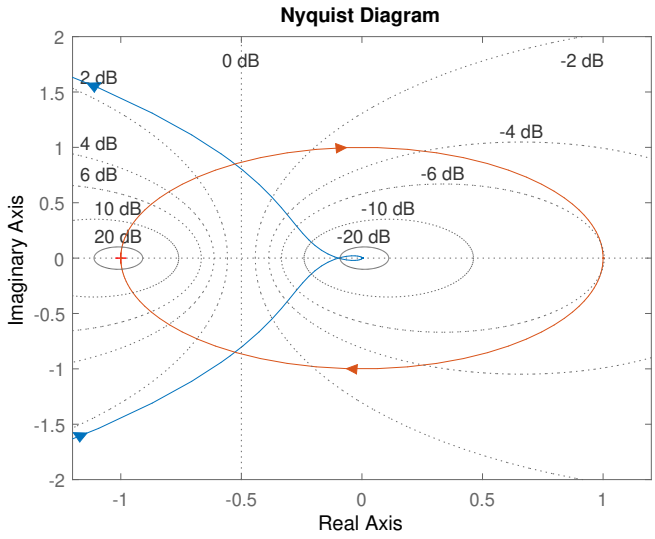


Figure 5.23: Nyquist Plot for Open Loop L Transfer Function for Inner-loop - Bandwidth and Robustness Design

the plot suggests that we are away from the -1 point, i.e., the distance from instability point. The closer the plot to -1 point implies that we will have a large sensitivity peak $|S|$ which is not desirable. Also, we have zero closed-loop unstable poles according to

$$P_{u,cl} = P_{i,ol} + N_{cw} = 0 \tag{5.39}$$

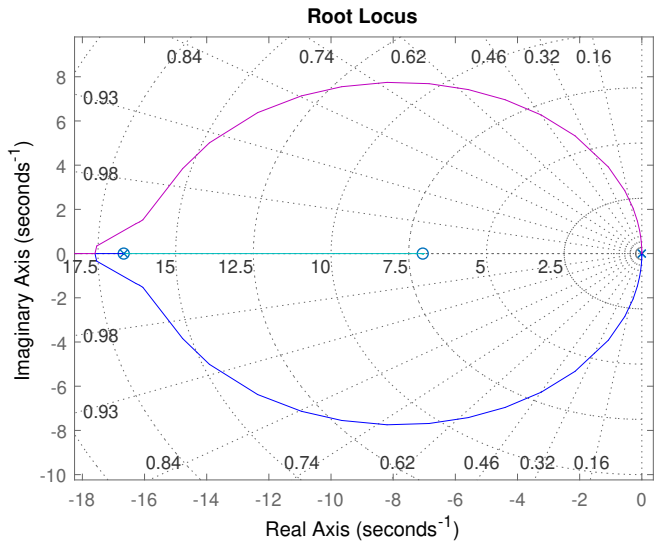


Figure 5.24: Root Locus for Open Loop L Transfer Function for Inner-loop at Low Frequencies - Bandwidth and Robustness Design

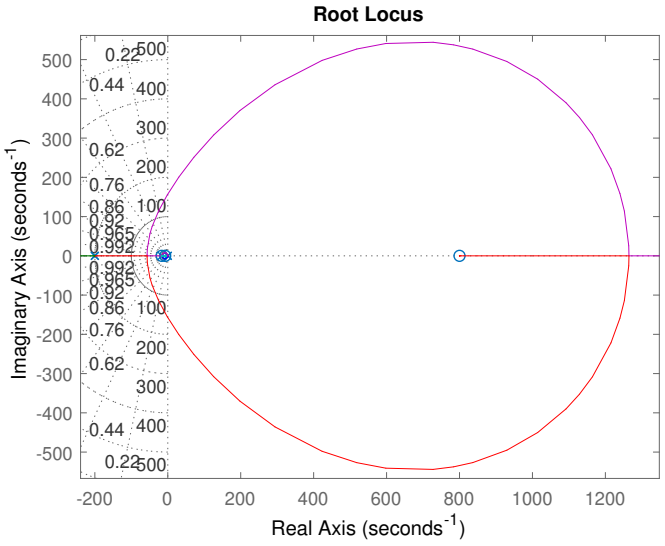


Figure 5.25: Root Locus for Open Loop L Transfer Function for Inner-loop at High Frequencies - Bandwidth and Robustness Design

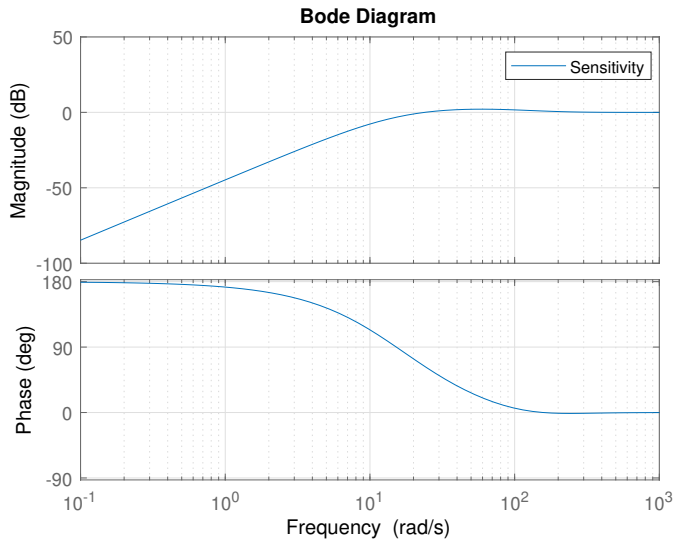


Figure 5.26: Sensitivity $|S|$ Bode Plot for Inner-loop - Bandwidth and Robustness Design

Sensitivity Frequency Response. The sensitivity transfer function $T_{re} = S$ is from r to e which we want to look like a zero at low frequencies. In other words, the impact of reference commands r on the error e should be zero; otherwise we will end up with large error values in e. The plot magnitude looks small at low frequencies for good low-frequency reference command following and good low-frequency output disturbance attenuation. The frequency at which the Bode magnitude of the sensitivity equals -20dB (0.1) is $w_l \approx 4.32$ rad/s. That is a good definition for bandwidth, which means reference commands with frequency content below 4.32 rad/s will be followed within 20dB, i.e., with a 10% steady-state error. Similarly, with the same amount output disturbances d_o with frequency content below 4.32 rad/s will be attenuated as well. The peak in the sensitivity is small, around 2dB, that is almost negligible.

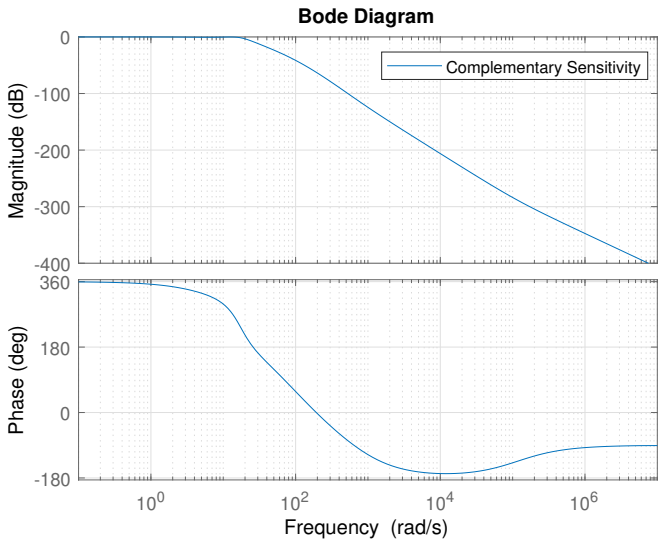


Figure 5.27: Complementary Sensitivity $|T|$ Bode Plot for Inner-loop - Bandwidth and Robustness Design

Complementary Sensitivity Frequency Response. The complementary sensitivity transfer function T is from r to y or d_i to u, which we want to look like unity at low frequencies and zero at high frequencies. In other words, we want the reference commands r to show up in the plant output y. The plot suggests that low-frequency content reference commands will be followed to some point. Also, the presence of a bump of 1.83dB in T suggests that an overshoot will be present to step reference commands r. Therefore, a prefilter W might be needed to reduce excessive overshoot in step reference commands r. Therefore, a first-order prefilter W with pole close to the controller zero 7.048 is as follows:

$$W(s) = \frac{7.048}{s + 7.048} \tag{5.40}$$

where the complementary sensitivity transfer function $T = \frac{WPK}{1+PK}$ looks like

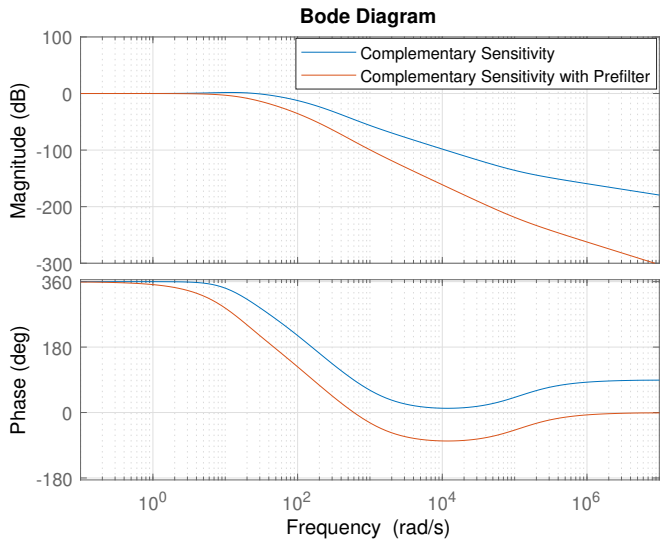


Figure 5.28: Complementary Sensitivity $|T|$ with Prefilter W Bode Plot for Inner-loop - Bandwidth and Robustness Design

where the prefilter reduces the overshoot significantly, also, the plot shows that high-frequency sensor noise n with content above 164 rad/s will be attenuated within 20dB. However, with the prefilter, high-frequency sensor noise n with content above 45 rad/s will be attenuated within 20dB. We can obtain the lower bounds for the sensitivity $||S||_{\mathcal{H}^\infty}$ and complementary sensitivity $||T||_{\mathcal{H}^\infty}$ using the following

$$\begin{aligned} ||S||_{\mathcal{H}^\infty} &\geq max\left\{\frac{\uparrow GM}{\uparrow GM-1},\ \frac{\downarrow GM}{1-\downarrow GM},\ \frac{1}{2\ sin(\frac{|PM|}{2})}\right\} = \left\{1.1087,\ 0,\ 1.0313\right\} \\ ||T||_{\mathcal{H}^\infty} &\geq max\left\{\frac{1}{\uparrow GM-1},\ \frac{1}{1-\downarrow GM},\ \frac{1}{2\ sin(\frac{|PM|}{2})}\right\} = \left\{0.1087,\ 1,\ 1.0313\right\} \end{aligned} \tag{5.41}$$

Time Domain Analysis. The closed loop system from reference commands r to the output y is the transfer function $T_{ry} = \frac{WPK}{1+PK}$ as follows:

$$T_{ry} = \frac{-0.07(s+1.15\times10^5)(s-800)(s+16.67)}{(s+268.7)(s+98.21)(s+19.92)(s+16.89)(s+12.94)} \tag{5.42}$$

Step reference commands were applied to obtain the closed loop system step response as follows

Pole	Damping	Frequency (rad/s)	Time Constant (s)
-1.29e+01	1.00e+00	1.29e+01	7.73e-02
-1.69e+01	1.00e+00	1.69e+01	5.92e-02
-1.99e+01	1.00e+00	1.99e+01	5.02e-02
-5.00e+01	1.00e+00	5.00e+01	2.00e-02
-9.82e+01	1.00e+00	9.82e+01	1.02e-02
-2.69e+02	1.00e+00	2.69e+02	3.72e-03

Table 5.5: Closed Loop Poles for the Inner-loop For $\omega_g = 25$ rad/s, $PM = 60$, and $W = \frac{50}{s+50}$ - Bandwidth and Robustness Design

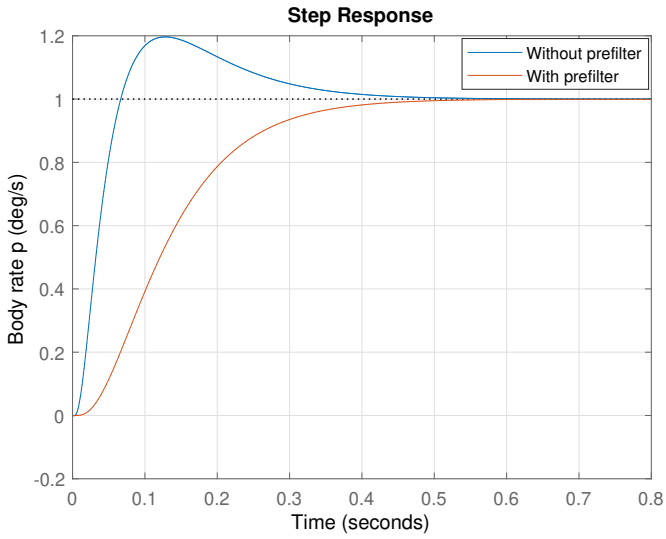


Figure 5.29: Inner-loop Step Response for p with and Without a Prefilter W - Bandwidth and Robustness Design

where the prefilter got rid off the excessive overshoot in the closed-loop step response.

Parameters	**Without Prefilter**	**With Prefilter**
Rise Time	0.0434 s	0.2170 s
Settling Time	0.3747 s	0.3937 s
Overshoot	19.65%	0.0%
Undershoot	0.08%	0.0011%
Peak	1.1965	0.9991

Table 5.6: Closed Loop Step Response for the Inner-loop - Bandwidth and Robustness Design

Time Domain Trade Studies: Varying the Unity Gain Crossover Frequency ω_g Design Parameter. In Figure 5.29, we see that as ω_g increases, the response speed increases as well. However, in practical implementation, as ω_g goes above 30 rad/s the system will experience large oscillations due to pushing the system too much. Therefore, we have chosen the design parameter ω_g to be 25 rad/s to avoid the oscillations and end up with a more stable system without pushing the system.

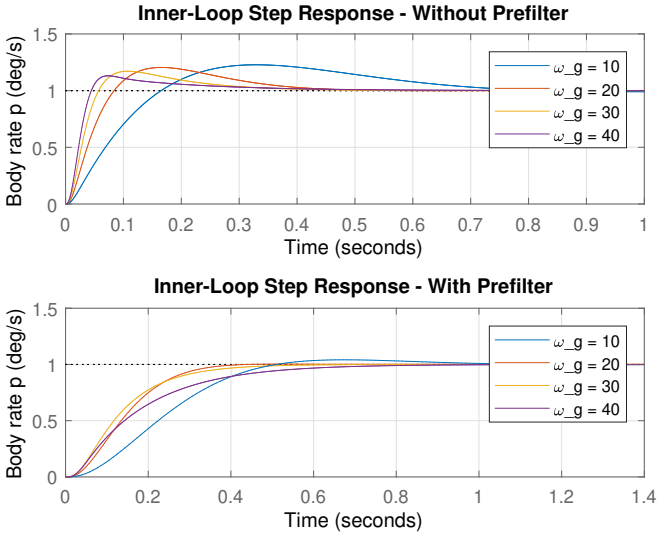


Figure 5.30: Inner-loop Step Response for Different ω_g Design Parameters - Bandwidth and Robustness Design

Time Domain Trade Studies: Varying the Phase Margin PM Design Parameter. In Figure 5.30, we can see that as the phase margin PM goes from 20° to 60°, the step response of the closed-loop system goes more stable with fewer oscillations. However, with $\omega_g = 25$ rad/s, the maximum phase margin that we can achieve with this design is 75°. Therefore, to have some room, we have chosen the minimum phase margin from optimal LQR control, which is $PM = 60°$.

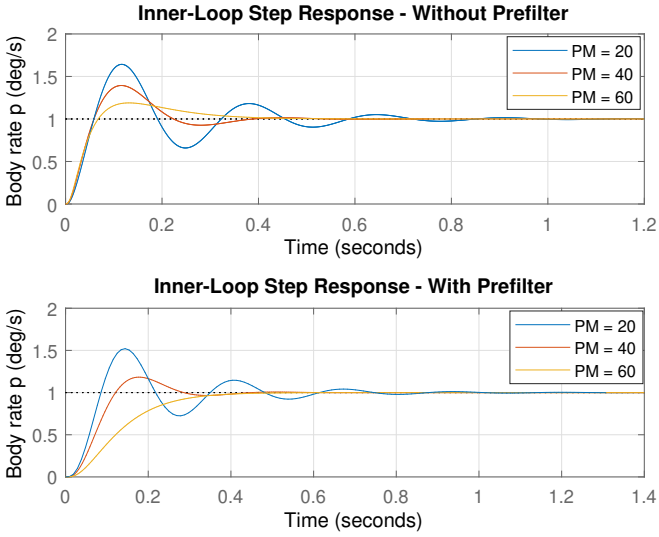


Figure 5.31: Inner-loop Step Response for Different PM Design Parameters - Bandwidth and Robustness Design

Time Domain Trade Studies: Varying the Prefilter W Pole Location. In Figure 5.31, we can see that as the pole of the prefilter W increases, the step response increase as well. However, there is a tradeoff with the faster response, which is an excessive overshoot in the step response. Because in the inner-loop we care more about the speed of the response, a prefilter of the form

$$W(s) = \frac{50}{s + 50} \tag{5.43}$$

is a reasonable selection for our design and has an overshoot of almost 20%.

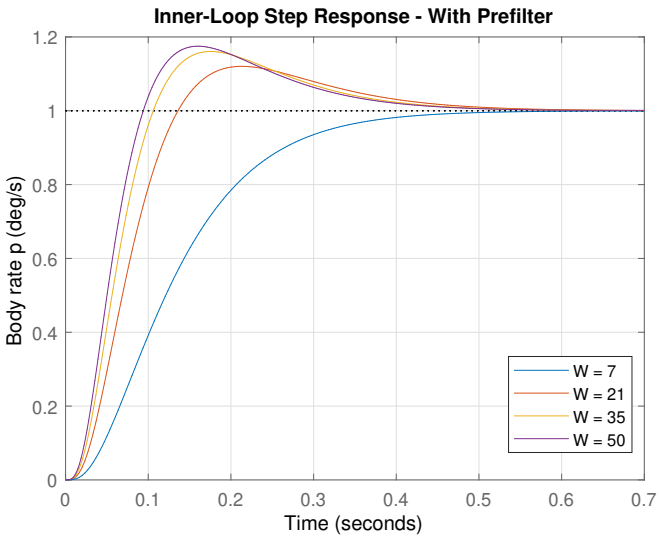


Figure 5.32: Inner-loop Step Response for Different Prefilter W Designs - Bandwidth and Robustness Design

the final closed-loop transfer function for the inner-loop from the reference command r to the output y is as follows:

$$T_{ry} = \frac{WPK}{1+PK} \approx \frac{37(s+7.048)(s+16.67)}{(s+12.94)(s+16.89)(s+19.92)} \tag{5.44}$$

5.3 Outer-Loop: (ϕ, θ, ψ) Attitude Control

From [12], the inner-outer loop control hierarchical structure is used where slower high-level commands to be followed by a faster inner control loop. For a good inner-outer loop command following, the inner loop should be faster than the outer loop by order of magnitude. However, in some applications, the inner-loop is 2-4 times the outer loop where we push the system to its limits. In this design, a factor of close to 2.5 was chosen so that the outer loop bandwidth is around 10 rad/s while the inner loop is 25 rad/s.

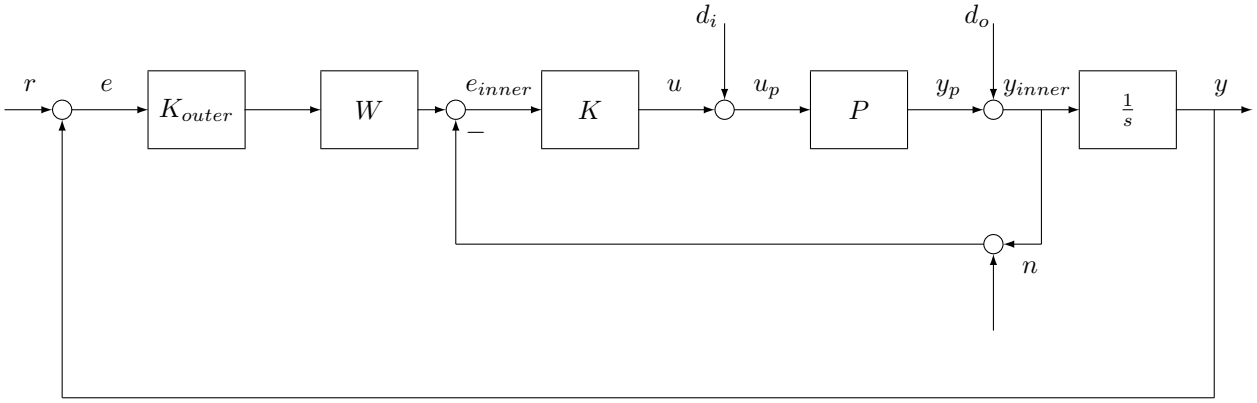


Figure 5.33: Outer-loop Feedback Block Diagram

5.3.1 Control System Design - PID Tuning Design

After designing the inner-loop control system by tuning each parameter (Proportional-Integral-Derivative), the outer-loop plant model is the closed loop transfer function of the inner loop T_{ry} from reference command r to output y multiplied by an integrator as follows:

$$P_{(p^{ref},\phi)} = T_{ry}\left[\frac{1}{s}\right] \approx \frac{25.42(s+6.079)(s+26.31)}{s(s+7.115)(s^2+33.93s+571.4)} \tag{5.45}$$

The controller of the outer loop is just a proportional controller

$$K_{outer} = 9 \tag{5.46}$$

Open Loop Frequency Response. By breaking the loop at the error $e = \phi_{ref} - \phi$ or the plant output y (ϕ) we obtain the open loop transfer function $L = PK$ as follows:

$$L_{outer} \approx \frac{228.78(s + 6.079)(s + 26.31)}{s(s + 7.115)(s^2 + 33.93s + 571.4)} \tag{5.47}$$

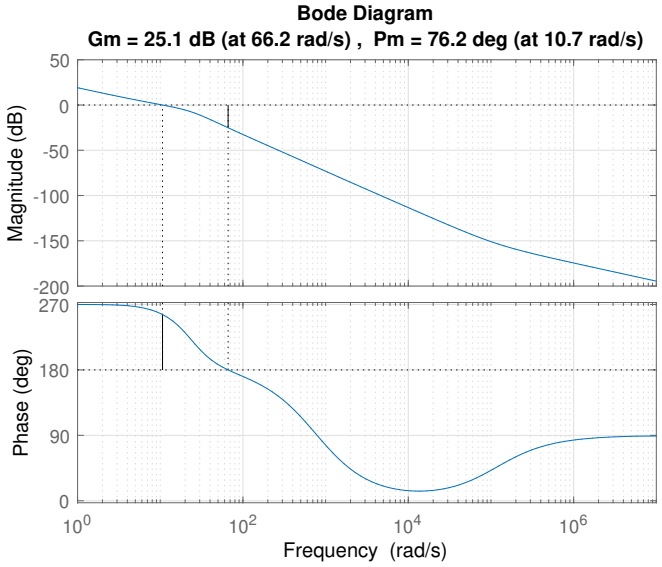


Figure 5.34: Bode Magnitude and Phase Plots for Open Loop L_{Outer} Transfer Function - PID Tuning Design

where we have a slope of -20dB/dec at low frequencies due to integral action. The unity gain crossover for the outer loop is $\omega_g = 10.65$ rad/s, which is lower than the inner loop by a factor of 2.69, which is expected for inner-outer loop control hierarchical structure. The phase margin is $PM = 76.2°$. The downward gain margin $\downarrow GM \approx 0$, upward gain margin $\uparrow GM \approx 18$, delay margin $DM \approx 0.1248s$. Usually,

we want to have good values for $\uparrow GM, \downarrow GM, PM$, but we have no guarantees for stability robustness. Because we might have, but we still have poor peak sensitivities. Therefore, we need to look at the Nyquist plot as follows

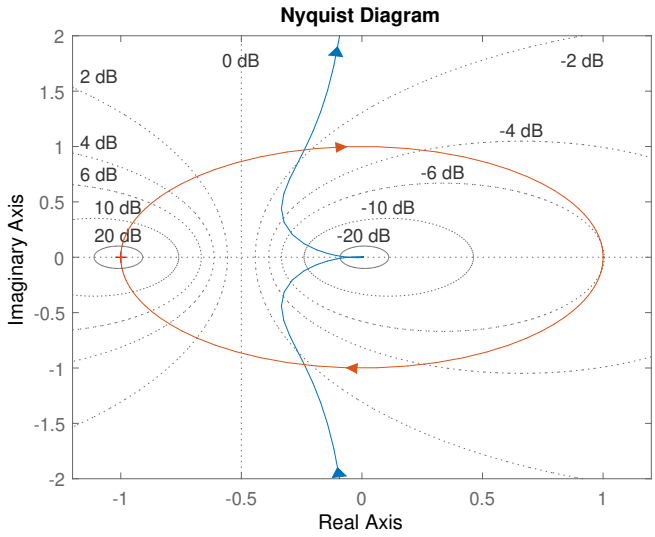


Figure 5.35: Nyquist Plot for Open Loop L_{Outer} Transfer Function - PID Tuning Design

the plot suggests that we are away from the -1 point, i.e., the distance from instability point. The closer the plot to -1 point implies that we will have a large sensitivity peak $|S|$ which is not desirable. Also, we have zero closed-loop unstable poles according to

$$P_{u,cl} = P_{i,ol} + N_{cw} = 0 \tag{5.48}$$

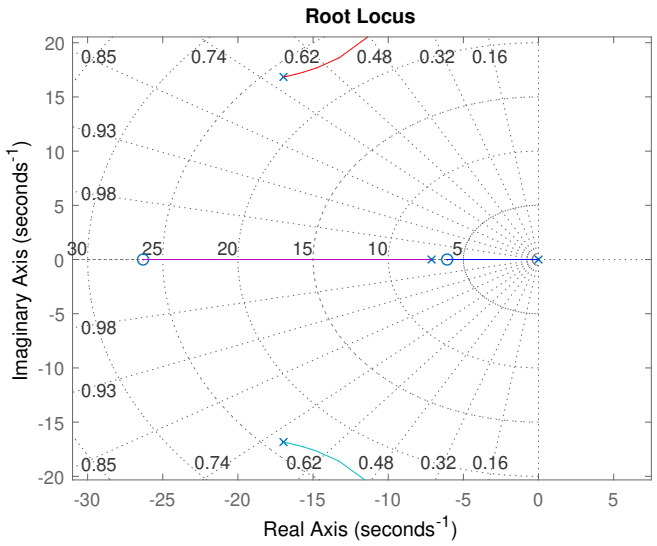


Figure 5.36: Root Locus for Open Loop L_{Outer} Transfer Function at Low Frequencies - PID Tuning Design

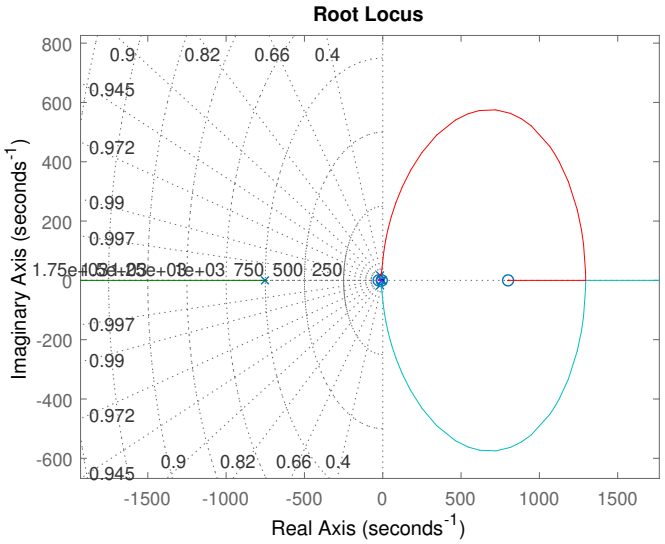


Figure 5.37: Root Locus for Open Loop L_{Outer} Transfer Function at High Frequencies - PID Tuning Design

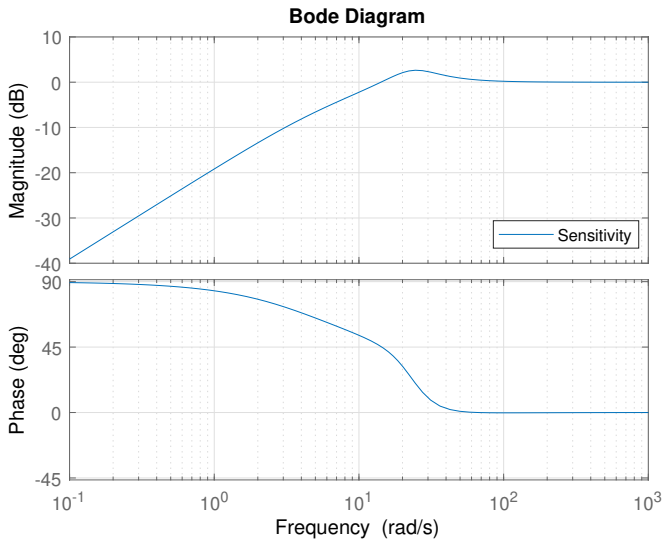


Figure 5.38: Sensitivity $|S|$ Bode Plot for Outer-loop - PID Tuning Design

Sensitivity Frequency Response. The sensitivity transfer function $T_{re} = S$ is from r to e which we want to look like a zero at low frequencies. In other words, the impact of reference commands r on the error e should be zero; otherwise we will end up with large error values in e. The plot magnitude looks small at low frequencies for good low-frequency reference command following and good low-frequency output disturbance attenuation. The frequency at which the Bode magnitude of the sensitivity equals -20dB (0.1) is $w_l \approx 0.9$ rad/s. That is a good definition for bandwidth which means reference commands with frequency content below 0.9 rad/s will be followed within 20dB, i.e. with a 10% steady-state error. Similarly, with the same amount output disturbances d_o with content below 0.9 rad/s will be attenuated as well. The peak in the sensitivity is around 6.4dB.

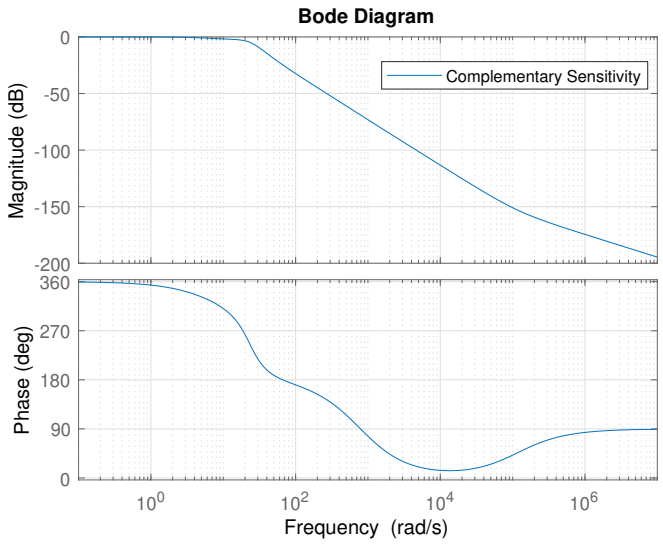


Figure 5.39: Complementary Sensitivity $|T|$ Bode Plot for Outer-loop - PID Tuning Design

Complementary Sensitivity Frequency Response. The complementary sensitivity transfer function T is from r to y or d_i to u, which we want to look like unity at low frequencies and zero at high frequencies. In other words, we want the reference commands r to show up in the plant output y. The plot suggests that low-frequency content reference commands will be followed to some point. Additionally, there is no presence of a bump in the complementary sensitivity plot. Also, the plot shows that high-frequency sensor noise n with content above 52.3 rad/s will be attenuated within 20dB. We can obtain the lower bounds for the sensitivity $||S||_{\mathcal{H}^\infty}$ and complementary sensitivity $||T||_{\mathcal{H}^\infty}$ using the following

$$\begin{aligned} ||S||_{\mathcal{H}^\infty} &\geq max\left\{\frac{\uparrow GM}{\uparrow GM-1}, \frac{\downarrow GM}{1-\downarrow GM}, \frac{1}{2\ sin(\frac{|PM|}{2})}\right\} = \left\{1.0588,\ 0,\ 0.8103\right\} \\ ||T||_{\mathcal{H}^\infty} &\geq max\left\{\frac{1}{\uparrow GM-1}, \frac{1}{1-\downarrow GM}, \frac{1}{2\ sin(\frac{|PM|}{2})}\right\} = \left\{0.0588,\ 1,\ 0.8103\right\} \end{aligned} \tag{5.49}$$

Pole	Damping	Frequency (rad/s)	Time Constant (s)
-5.23e+00	1.00e+00	5.23e+00	1.91e-01
-1.27e+01	1.00e+00	1.27e+01	7.89e-02
-1.13e+01 + 2.06e+01i	4.80e-01	2.35e+01	8.87e-02
-1.13e+01 - 2.06e+01i	4.80e-01	2.35e+01	8.87e-02
-7.53e+02	1.00e+00	7.53e+02	1.33e-03

Table 5.7: Closed Loop Poles for the Outer-loop - PID Tuning Design

Time Domain Analysis.

$$T_{ry} \approx \frac{228.63(s+26.31)(s+6.079)}{(s+12.67)(s+5.229)(s^2+22.56s+552)} \tag{5.50}$$

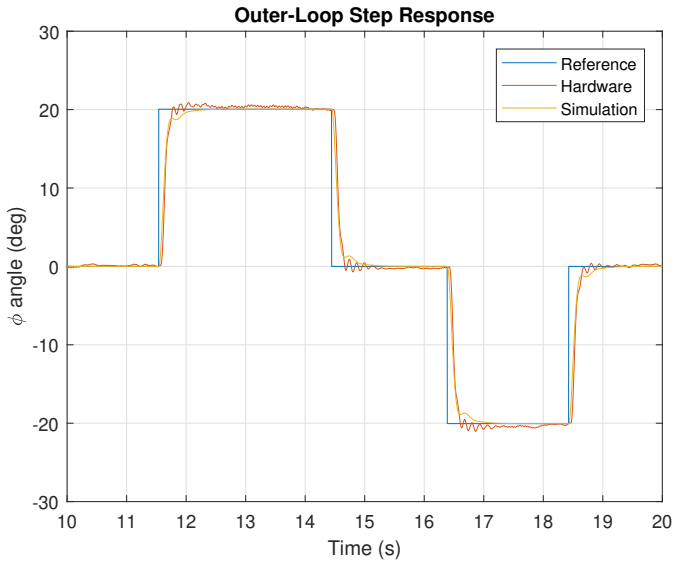


Figure 5.40: Step Response for ϕ Reference Command - PID Tuning Design

System Identification for the Outer-Loop. By using MATLAB's System Identification Toolbox we have obtained the following closed-loop transfer function T_{ry}:

$$T_{id} = \frac{211(s^2-99.53s+5226)}{(s^2+33.4s+368.7)(s^2+15.61s+2956)} \tag{5.51}$$

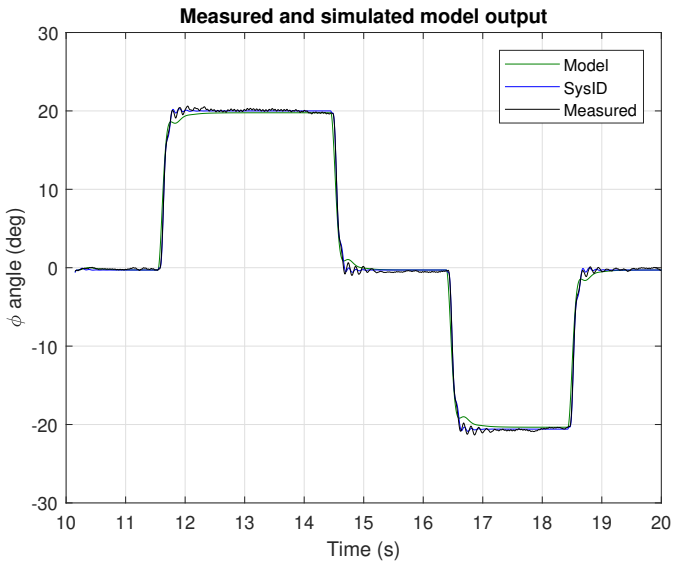


Figure 5.41: System Identification for the Closed-loop Transfer Function T_{ry} for Outer-loop - PID Tuning Design

5.3.2 Control System Design - Pole Placement Design

Because of a well designed inner-loop control system for the body rotation rates (p, q, r), the outer-loop is just a proportional controller where the reference command is (ϕ, θ, ψ). The closed loop system of the inner loop is T_{ry} and the new outer loop plant is as follows

$$P_{(p^{ref},\phi)} = T_{ry}\left[\frac{1}{s}\right] \approx \frac{373(s+16.67)(s+14.31)}{s(s+16.66)(s+14.31)(s^2+21.01s+373.3)} \tag{5.52}$$

$$K_{outer} = 8 \tag{5.53}$$

Open Loop Frequency Response. By breaking the loop at the error $e = \phi_{ref} - \phi$ or the plant output y (ϕ) we obtain the open loop transfer function $L = PK$ as follows:

$$L_{outer} \approx \frac{2.98 \times 10^3(s+16.67)}{s(s+16.66)(s^2+21.01s+373.3)} \tag{5.54}$$

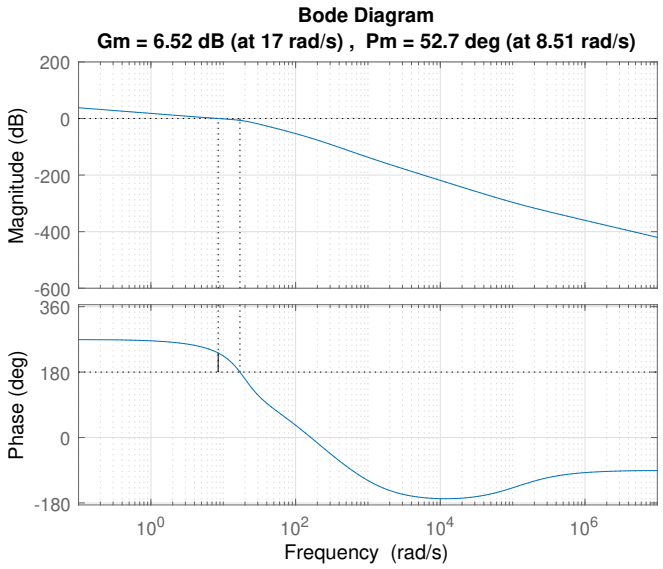


Figure 5.42: Bode Magnitude and Phase Plots for Open Loop L_{Outer} Transfer Function - Pole Placement Design

where we have a slope of -20dB/dec at low frequencies due to integral action. The unity gain crossover for the outer loop is $\omega_g = 8.5$ rad/s, which is lower than the inner loop by a factor of 2.7, which is expected for inner-outer loop control hierarchical structure. The phase margin is $PM = 52.6°$. The downward gain margin $\downarrow GM \approx 0$, upward gain margin $\uparrow GM \approx 2.11$, delay margin $DM \approx 0.1081s$. Usually, we want to have good values for $\uparrow GM, \downarrow GM, PM$, but we have no guarantees for stability robustness. Because we might have, but we still have poor peak sensitivities. Therefore, we need to look at the Nyquist plot as follows

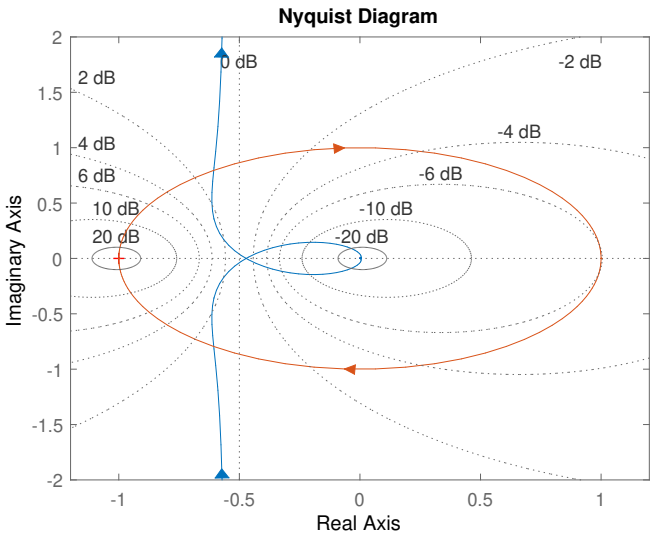


Figure 5.43: Nyquist Plot for Open Loop L_{Outer} Transfer Function - Pole Placement Design

the plot suggests that we are away from the -1 point, i.e., the distance from instability point. The closer the plot to -1 point implies that we will have a large sensitivity peak $|S|$ which is not desirable. Also, we have zero closed-loop unstable poles according to

$$P_{u,cl} = P_{i,ol} + N_{cw} = 0 \tag{5.55}$$

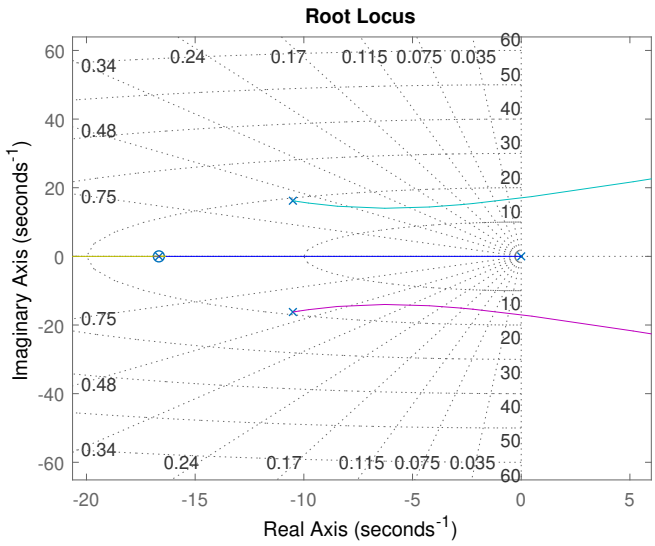


Figure 5.44: Root Locus for Open Loop L_{Outer} Transfer Function at Low Frequencies - Pole Placement Design

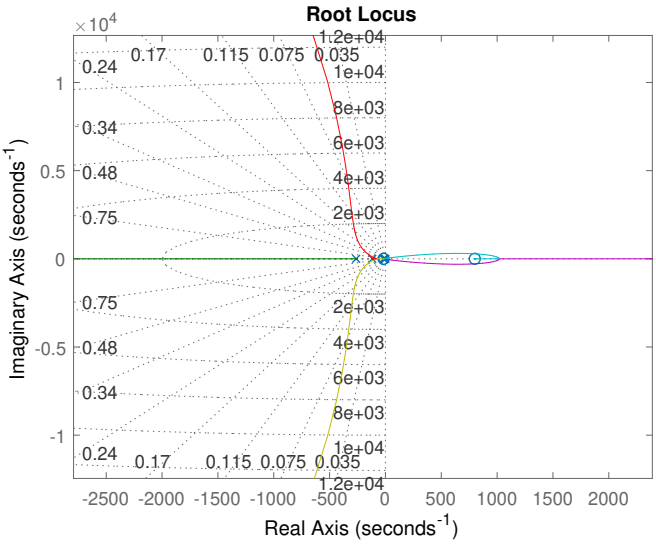


Figure 5.45: Root Locus for Open Loop L_{Outer} Transfer Function at High Frequencies - Pole Placement Design

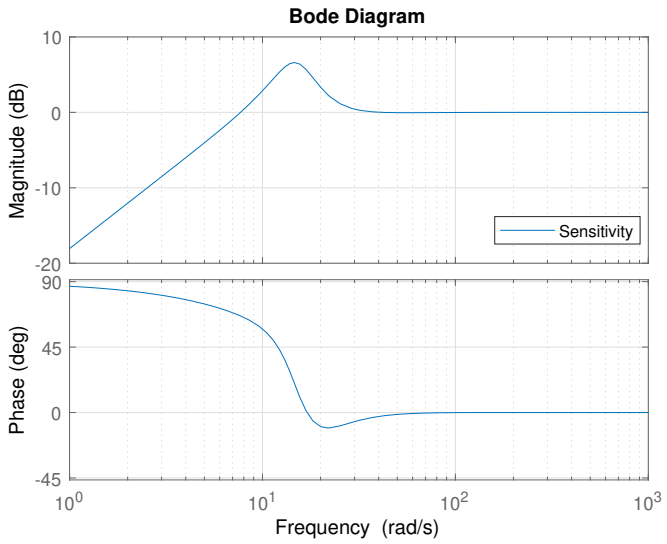


Figure 5.46: Sensitivity $|S|$ Bode Plot for Outer-loop - Pole Placement Design

Sensitivity Frequency Response. The sensitivity transfer function $T_{re} = S$ is from r to e which we want to look like a zero at low frequencies. In other words, the impact of reference commands r on the error e should be zero; otherwise we will end up with large error values in e. The plot magnitude looks small at low frequencies for good low-frequency reference command following and good low-frequency output disturbance attenuation. The frequency at which the Bode magnitude of the sensitivity equals -20dB (0.1) is $w_l \approx 0.8$ rad/s. That is a good definition for bandwidth which means reference commands with frequency content below 0.8 rad/s will be followed within 20dB, i.e. with a 10% steady-state error. Similarly, with the same amount output disturbances d_o with content below 0.8 rad/s will be attenuated as well. The peak in the sensitivity is around 6.6dB.

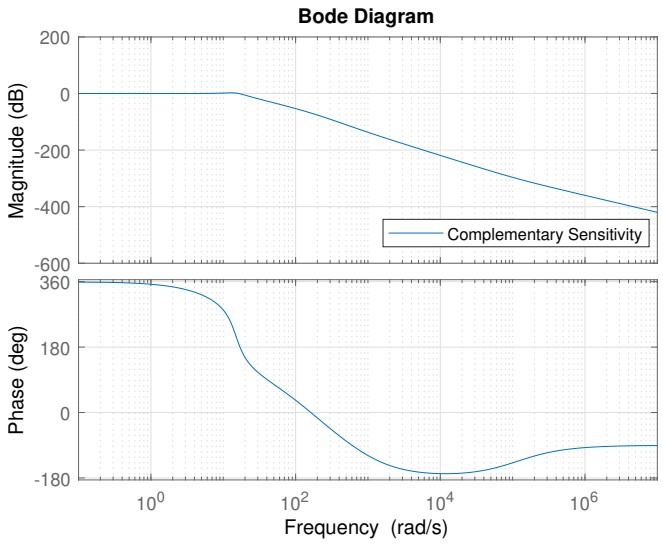


Figure 5.47: Complementary Sensitivity $|T|$ Bode Plot for Outer-loop - Pole Placement Design

Complementary Sensitivity Frequency Response. The complementary sensitivity transfer function T is from r to y or d_i to u, which we want to look like unity at low frequencies and zero at high frequencies. In other words, we want the reference commands r to show up in the plant output y. The plot suggests that low-frequency content reference commands will be followed to some point. Additionally, there is no presence of a bump in the complementary sensitivity plot. Also, the plot shows that high-frequency sensor noise n with content above 32 rad/s will be attenuated within 20dB. We can obtain the lower bounds for the sensitivity $||S||_{\mathcal{H}^\infty}$ and complementary sensitivity $||T||_{\mathcal{H}^\infty}$ using the following

$$\begin{aligned} ||S||_{\mathcal{H}^\infty} &\geq max\left\{\frac{\uparrow GM}{\uparrow GM-1},\ \frac{\downarrow GM}{1-\downarrow GM},\ \frac{1}{2\ sin(\frac{|PM|}{2})}\right\} = \left\{1.9,\ 0,\ 1.1285\right\} \\ ||T||_{\mathcal{H}^\infty} &\geq max\left\{\frac{1}{\uparrow GM-1},\ \frac{1}{1-\downarrow GM},\ \frac{1}{2\ sin(\frac{|PM|}{2})}\right\} = \left\{0.9,\ 1,\ 1.1285\right\} \end{aligned} \tag{5.56}$$

Pole	Damping	Frequency (rad/s)	Time Constant (s)
-1.33e+01	1.00e+00	1.33e+01	7.49e-02
-4.09e+00 + 1.44e+01i	2.72e-01	1.50e+01	2.45e-01
-4.09e+00 - 1.44e+01i	2.72e-01	1.50e+01	2.45e-01
-1.66e+01	1.00e+00	1.66e+01	6.01e-02
-1.17e+02	1.00e+00	1.17e+02	8.58e-03
-2.62e+02	1.00e+00	2.62e+02	3.82e-03

Table 5.8: Closed Loop Poles for the Outer-loop - Pole Placement Design

Time Domain Analysis.

$$T_{ry} \approx \frac{2.9 \times 10^3 (s + 16.67)}{(s^2 + 8.175s + 225.1)(s + 13.34)(s + 16.64)} \tag{5.57}$$

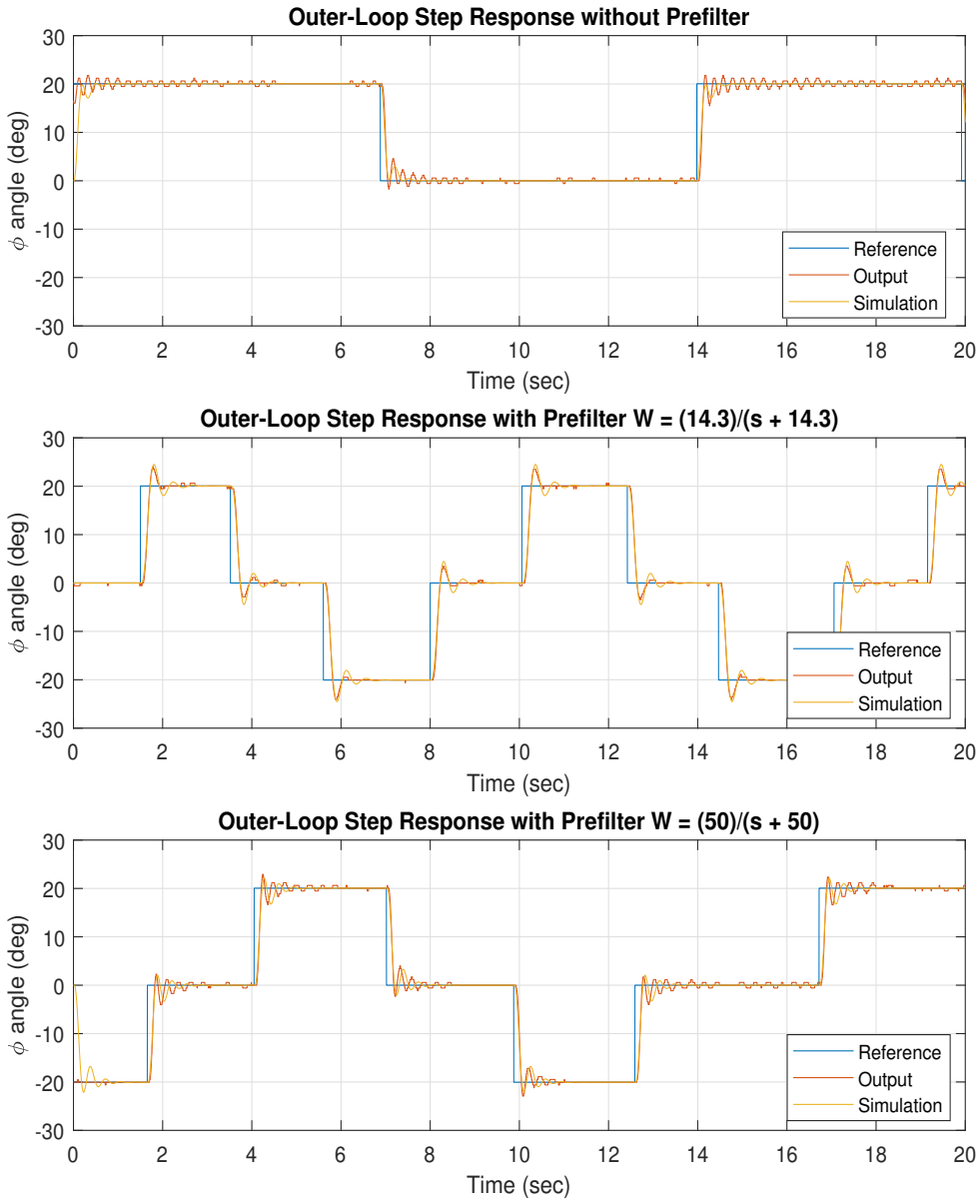


Figure 5.48: Step Response for ϕ Reference Command - Pole Placement Design

System Identification for the Outer-Loop. By using MATLAB's System Iden-

tification Toolbox we have obtained the following closed-loop transfer function T_{ry}:

$$T_{id} = \frac{-3233(s - 58.51)}{(s^2 + 18.6s + 293)(s^2 + 31.07s + 643.7)} \tag{5.58}$$

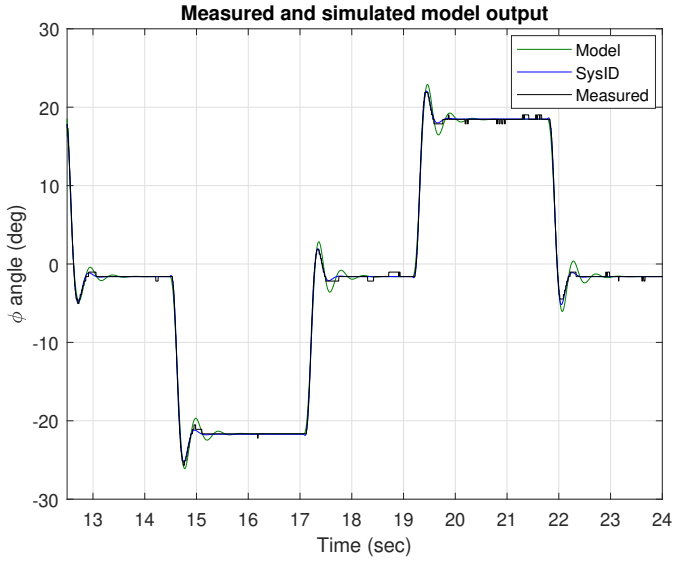


Figure 5.49: System Identification for the Closed-loop Transfer Function T_{ry} for Outer-loop - Pole Placement Design

Frequency Domain Trade Studies: Varying the Gain of the Controller. By increasing the proportional gain of the controller K_{outer} from 2 to 12, the peak sensitivity S increases as well, which is not desirable. However, as K_{outer} gets larger, reference commands with higher frequency contents will be followed with a lower steady-state error.

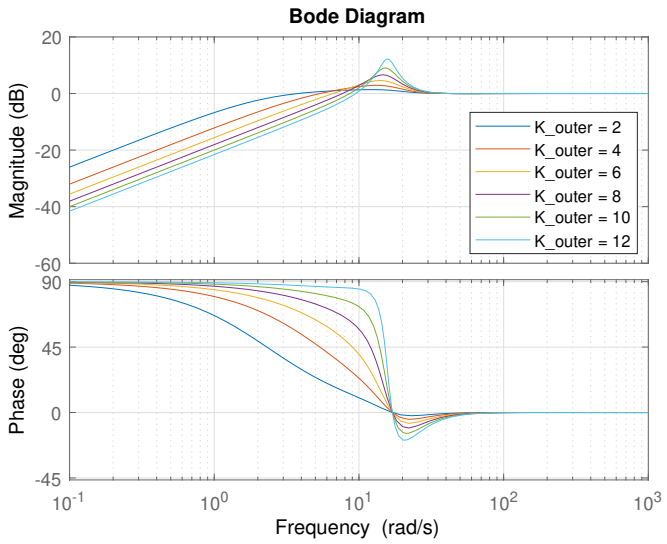


Figure 5.50: Sensitivity $|S|$ Bode Plot for Outer-loop for Different Gain Values of K_{outer} - Pole Placement Design

Furthermore, the bump in the complementary sensitivity gets larger as K_{outer} increases. As a result, there will be an excessive overshoot in the output response y due to step reference commands r.

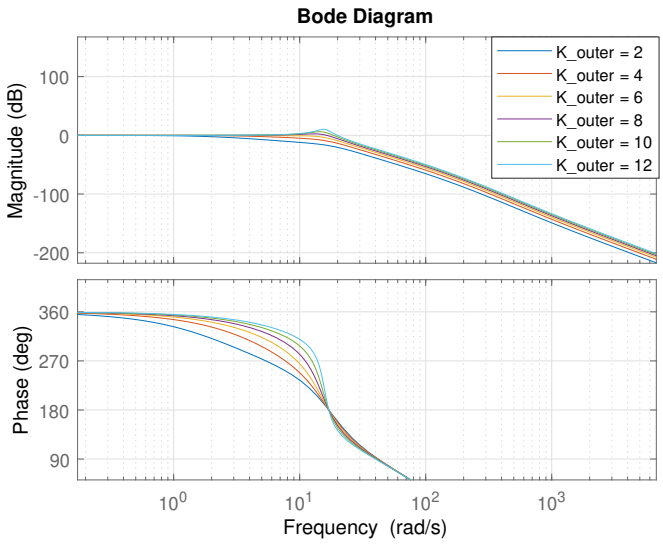


Figure 5.51: Complementary Sensitivity $|T|$ Bode Plot for Outer-loop for Different Gain Values of K_{outer} - Pole Placement Design

Time Domain Trade Studies: Varying the Gain of the Controller. By increasing the proportional gain of the controller K_{outer} from 2 to 12, i.e., pushing the system more and more, the rise time t_r of the output response y becomes smaller, and as a trade of the output, y exhibits excessive overshoot as well. Therefore, there is a trade-off between the speed of the response and the overshoot.

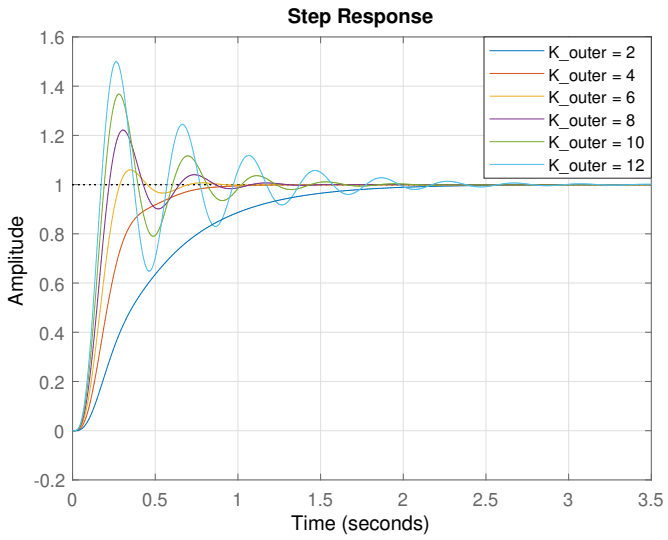


Figure 5.52: Step Response for ϕ Reference Command for Different Gain Values of K_{outer} - Pole Placement Design

5.3.3 Control System Design - Design for Bandwidth and Robustness

Because of a well designed inner-loop control system for the body rotation rates (p, q, r), the outer-loop is just a proportional controller where the reference command is (ϕ, θ, ψ). The closed loop system of the inner loop is T_{ry} and the new outer loop plant is as follows

$$P_{(p^{ref},\phi)} = T_{ry}\left[\frac{1}{s}\right] \approx \frac{37(s+16.67)(s+7.048)}{s(s+19.92)(s+16.89)(s+12.94)}, \qquad K_{outer} = 8 \tag{5.59}$$

Open Loop Frequency Response. By breaking the loop at the error $e = \phi_{ref} - \phi$ or the plant output y (ϕ) we obtain the open loop transfer function $L = PK$ as follows:

$$L_{outer} \approx \frac{296.27(s+16.67)(s+7.048)}{s(s+19.92)(s+16.89)(s+12.94)} \tag{5.60}$$

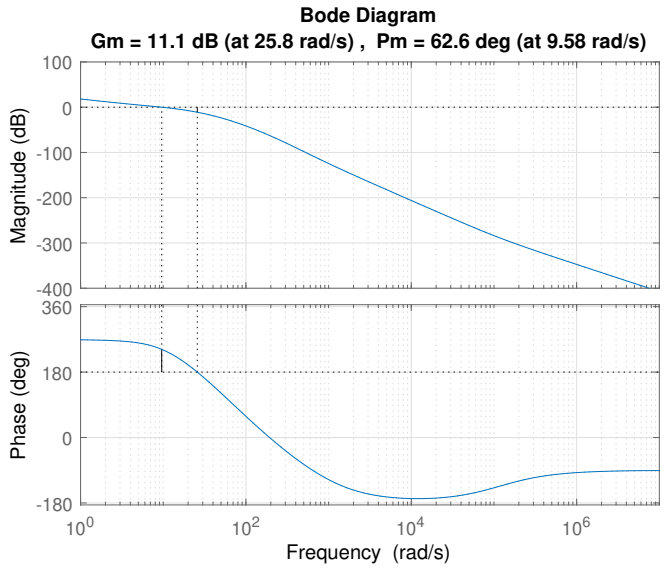


Figure 5.53: Bode Magnitude and Phase Plots for Open Loop L_{outer} Transfer Function - Bandwidth and Robustness Design

where we have a slope of -20dB/dec at low frequencies due to integral action. The unity gain crossover for the outer loop is $\omega_g = 9.58$ rad/s, which is lower than the inner loop by a factor of 2.6, which is expected for inner-outer loop control hierarchical structure. The phase margin is $PM = 62.6°$. The downward gain margin $\downarrow GM \approx 0$, upward gain margin $\uparrow GM \approx 3.59$, delay margin $DM \approx 0.1141s$. Usually, we want to have good values for $\uparrow GM, \downarrow GM, PM$, but we have no guarantees for stability robustness. Because we might have, but we still have poor peak sensitivities. Therefore, we need to look at the Nyquist plot as follows

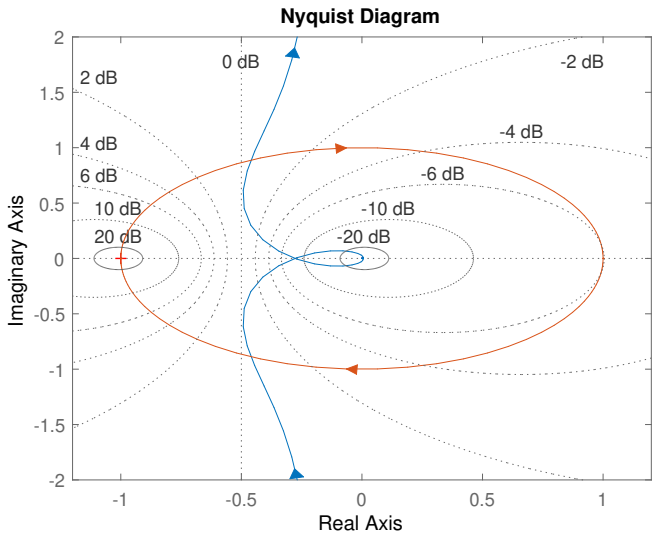


Figure 5.54: Nyquist Plot for Open Loop L_{outer} Transfer Function - Bandwidth and Robustness Design

the plot suggests that we are away from the -1 point, i.e., the distance from instability point. The closer the plot to -1 point implies that we will have a large sensitivity peak $|S|$ which is not desirable. Also, we have zero closed-loop unstable poles according to

$$P_{u,cl} = P_{i,ol} + N_{cw} = 0 \tag{5.61}$$

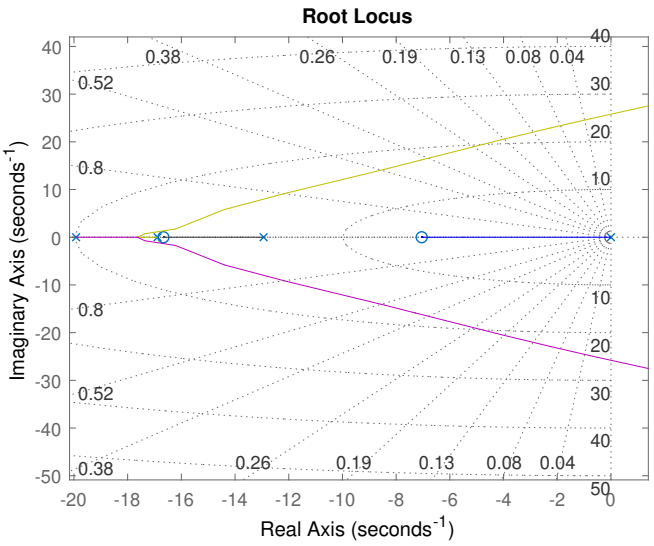


Figure 5.55: Root Locus for Open Loop L_{outer} Transfer Function at Low Frequencies - Bandwidth and Robustness Design

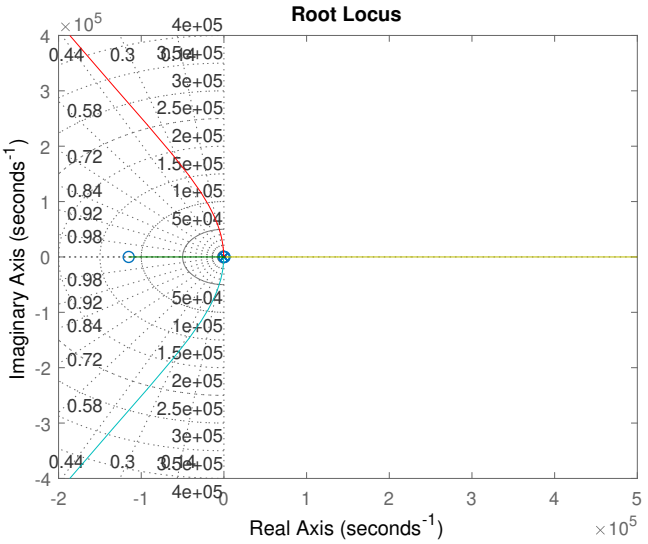


Figure 5.56: Root Locus for Open Loop L_{outer} Transfer Function at High Frequencies - Bandwidth and Robustness Design

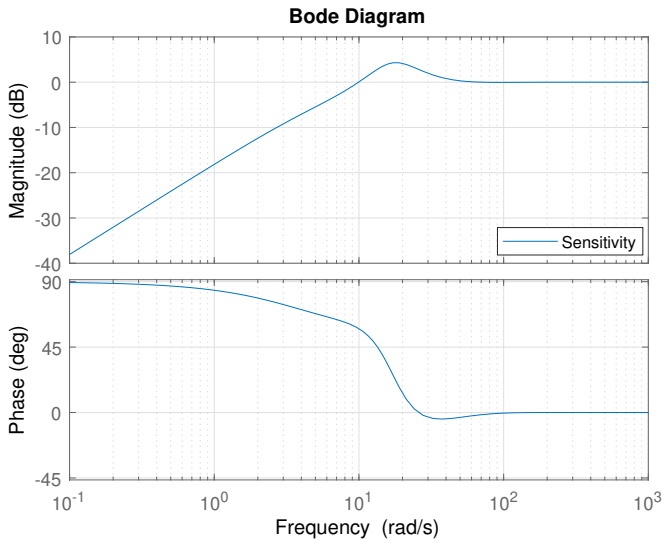


Figure 5.57: Sensitivity $|S|$ Bode Plot for Outer-loop - Bandwidth and Robustness Design

Sensitivity Frequency Response. The sensitivity transfer function $T_{re} = S$ is from r to e which we want to look like a zero at low frequencies. In other words, the impact of reference commands r on the error e should be zero; otherwise we will end up with large error values in e. The plot magnitude looks small at low frequencies for good low-frequency reference command following and good low-frequency output disturbance attenuation. The frequency at which the Bode magnitude of the sensitivity equals -20dB (0.1) is $w_l \approx 0.8$ rad/s. That is a good definition for bandwidth, which means reference commands with frequency content below 0.8 rad/s will be followed within 20dB, i.e., with a 10% steady-state error. Similarly, with the same amount output disturbances d_o with frequency content below 0.8 rad/s will be attenuated as well. The peak in the sensitivity is around 4.29dB.

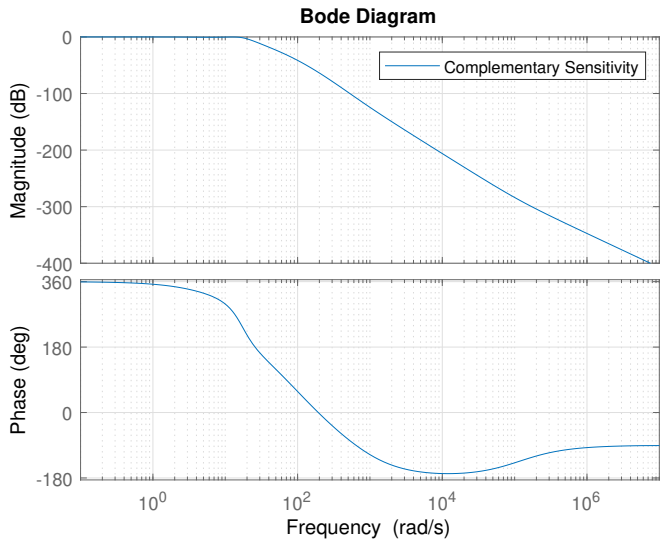


Figure 5.58: Complementary Sensitivity $|T|$ Bode Plot for Outer-loop - Bandwidth and Robustness Design

Complementary Sensitivity Frequency Response. The complementary sensitivity transfer function T is from r to y or d_i to u, which we want to look like unity at low frequencies and zero at high frequencies. In other words, we want the reference commands r to show up in the plant output y. The plot suggests that low-frequency content reference commands will be followed to some point. Additionally, there is no presence of a bump in the complementary sensitivity plot. Also, the plot shows that high-frequency sensor noise n with content above 43 rad/s will be attenuated within 20dB. We can obtain the lower bounds for the sensitivity $||S||_{\mathcal{H}^\infty}$ and complementary sensitivity $||T||_{\mathcal{H}^\infty}$ using the following

$$
\begin{aligned}
||S||_{\mathcal{H}^\infty} &\geq max\left\{\frac{\uparrow GM}{\uparrow GM-1},\ \frac{\downarrow GM}{1-\downarrow GM},\ \frac{1}{2\ sin(\frac{|PM|}{2})}\right\} = \left\{1.38,\ 0,\ 0.9622\right\} \\
||T||_{\mathcal{H}^\infty} &\geq max\left\{\frac{1}{\uparrow GM-1},\ \frac{1}{1-\downarrow GM},\ \frac{1}{2\ sin(\frac{|PM|}{2})}\right\} = \left\{0.3855,\ 1,\ 0.9622\right\}
\end{aligned}
\tag{5.62}
$$

Pole	Damping	Frequency (rad/s)	Time Constant (s)
-5.38e+00	1.00e+00	5.38e+00	1.86e-01
-1.67e+01	1.00e+00	1.67e+01	6.00e-02
-7.56e+00 + 1.55e+01i	4.39e-01	1.72e+01	1.32e-01
-7.56e+00 - 1.55e+01i	4.39e-01	1.72e+01	1.32e-01
-7.55e+01	1.00e+00	7.55e+01	1.32e-02
-8.51e+02	1.00e+00	8.51e+02	1.18e-02
-2.69e+02	1.00e+00	2.69e+02	3.72e-03

Table 5.9: Closed Loop Poles for the Outer-loop - Bandwidth and Robustness Design

Time Domain Analysis.

$$T_{ry} \approx \frac{226.31(s+16.67)(s+7.048)}{(s+16.66)(s+5.379)(s^2+15.12s+296.7)} \tag{5.63}$$

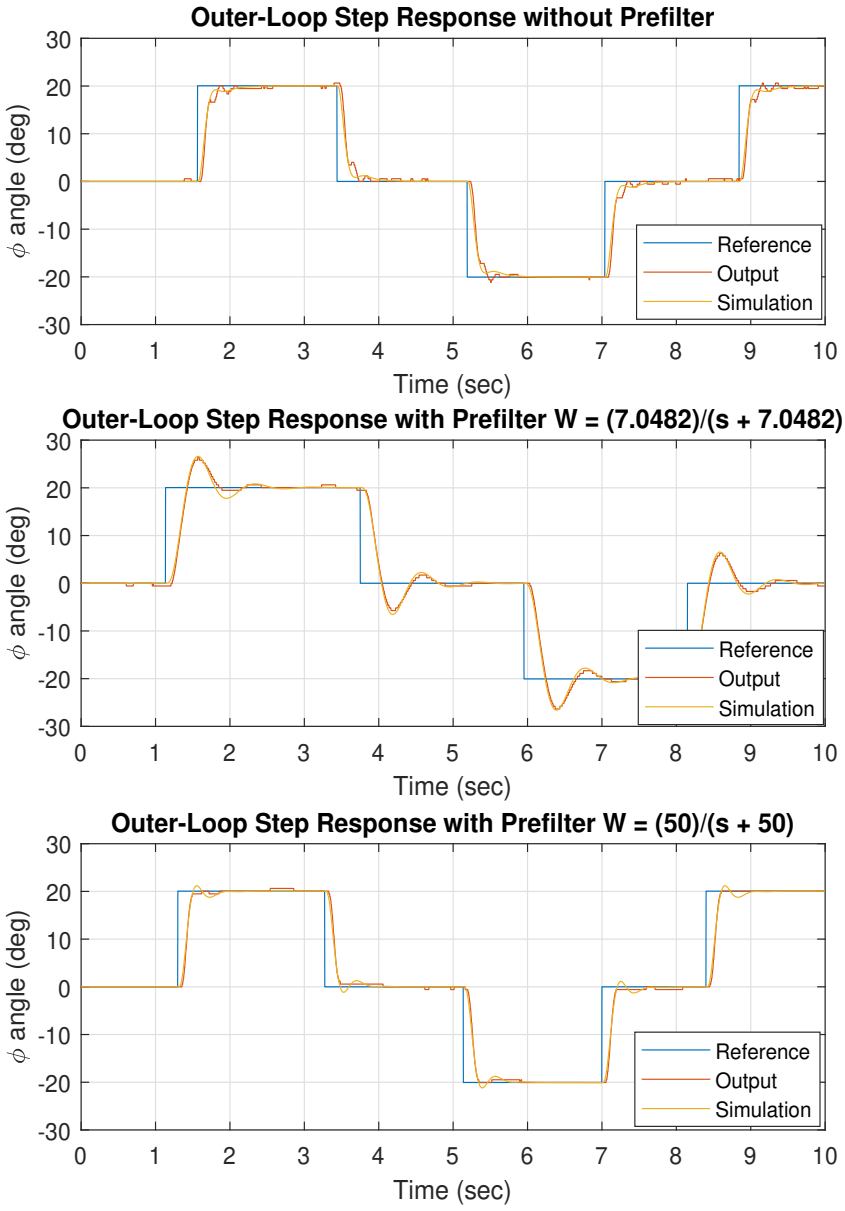


Figure 5.59: Step Response for ϕ Reference Command - Bandwidth and Robustness Design

System Identification for the Outer-Loop. By using MATLAB's System Iden-

tification Toolbox we have obtained the following closed-loop transfer function T_{ry}:

$$T_{id} = \frac{258.67(s^2 - 89.99s + 3648)}{(s + 41.83)(s + 19.15)(s^2 + 30.39s + 1178)} \tag{5.64}$$

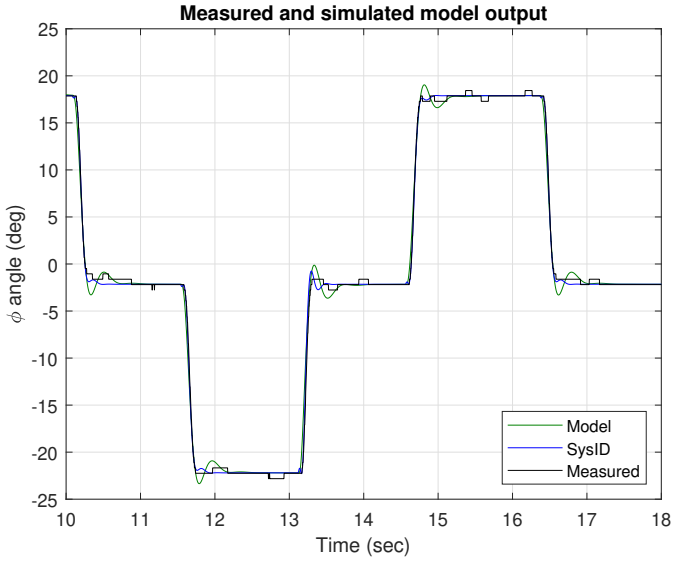


Figure 5.60: System Identification for the Closed-loop Transfer Function T_{ry} for Outer-loop - Bandwidth and Robustness Design

Frequency Domain Trade Studies: Varying the Gain of the Controller. By increasing the proportional gain of the controller K_{outer} from 2 to 12, the peak sensitivity S increases as well, which is not desirable. However, as K_{outer} gets larger, reference commands with higher frequency contents will be followed with a lower steady-state error.

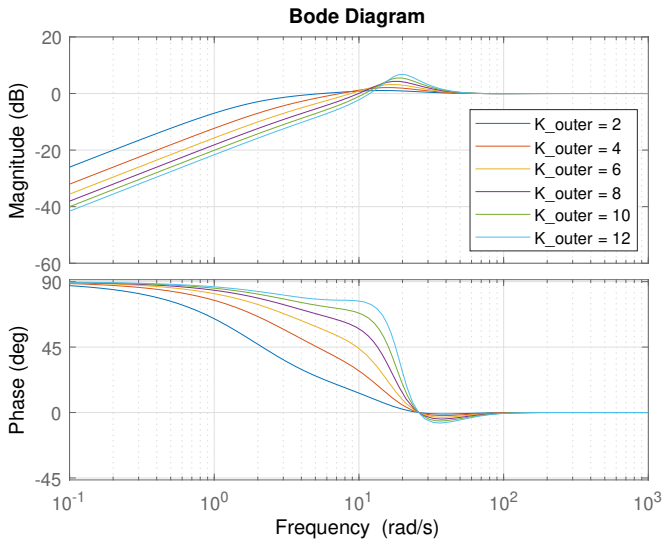


Figure 5.61: Sensitivity $|S|$ Bode Plot for Outer-loop for Different Gain Values of K_{outer} - Bandwidth and Robustness Design

Additionally, the bump in the complementary sensitivity gets larger as K_{outer} increases. As a result, there will be an excessive overshoot in the output response y due to step reference commands r.

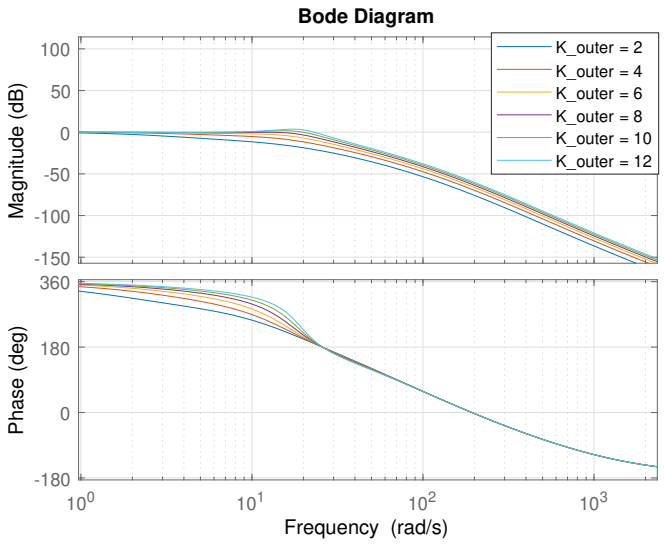


Figure 5.62: Complementary Sensitivity $|T|$ Bode Plot for Outer-loop for Different Gain Values of K_{outer} - Bandwidth and Robustness Design

Time Domain Trade Studies: Varying the Gain of the Controller. By increasing the proportional gain of the controller K_{outer} from 2 to 12, i.e., pushing the system more and more, the rise time t_r of the output response y becomes smaller, and as a trade of the output, y exhibits excessive overshoot as well. Therefore, there is a trade-off between the speed of the response and the overshoot.

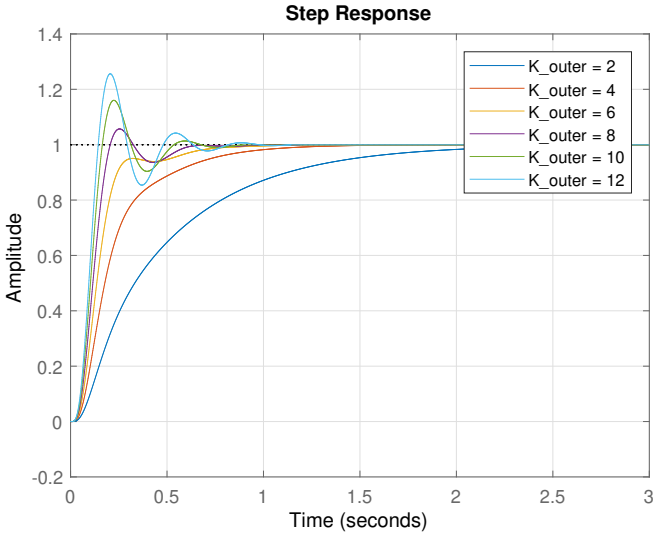


Figure 5.63: Step Response for ϕ Reference Command for Different Gain Values of K_{outer} - Bandwidth and Robustness Design

Chapter 6

HIGH-LEVEL CONTROL: POSITION AND PATH FOLLOWING CONTROL

6.1 Overview

In this chapter, a nonlinear quadrotor translational dynamical were linearized around two different operating points. The first linearization is around hovering position while the other around forwarding flight. Model analysis of both linear models was addressed to show the effect of coupling in translational dynamics. Three different controllers were designed for the decoupled linear model as follows: (1) Linear Quadratic (LQ) Servo (2) Weighted $\mathcal{H}^\infty$ Sensitivity Optimization (3) Linear Quadratic Gaussian with Loop Transfer Recover at the Output (LQG/LTRO). For each control design, a frequency domain analysis and time domain analysis were presented as well.

6.2 Quadrotor Nonlinear Translational Dynamical Model

Since we have designed an attitude controller (inner-loop), now we will design the outer-loop control. The quadrotor system has 12 states $x = [x, y, z, \dot{x}, \dot{y}, \dot{z}, \phi, \theta, \psi, p, q, r]^T$ but since we have an attitude controller which takes care of (ϕ, θ, p, q, r), we will end up with 7 states for the outer loop control $x = [x, y, z, \dot{x}, \dot{y}, \dot{z}, \psi]^T$ and $\nu = [T, \phi, \theta, r]^T$. The nonlinear translational dynamics of the quadrotor as follows

$$\dot{x} = f(x, \nu) \tag{6.1}$$

$$\begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix} = \begin{bmatrix} -c(\phi)s(\theta)c(\psi) - s(\phi)s(\psi) \\ -c(\phi)s(\theta)s(\psi) + s(\phi)c(\psi) \\ c(\phi)c(\theta) \end{bmatrix} \frac{T}{m} + \begin{bmatrix} 0 \\ 0 \\ -g \end{bmatrix} \tag{6.2}$$

$$\dot{\psi} = \frac{s(\phi)}{c(\theta)}p + \frac{c(\phi)}{c(\theta)}r \tag{6.3}$$

Nonlinearity Issue. This is a question that might come to someone's mind:

How do we obtain a linear model for the system without restricting ψ to be a constant value?

From [16], we replace the nonlinearity term by $u_p \in \mathcal{R}^3$ and solving for it. After solving for u_p, we can solve for (ϕ, θ, ψ) because we have three equations and three unknowns. We have the following

$$u_p \stackrel{\text{def}}{=} f_p(x,\nu) = R_z^T(\psi)R_y^T(\theta)R_x^T(\phi)\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}\frac{T}{m} = \begin{bmatrix} -c(\phi)s(\theta)c(\psi) - s(\phi)s(\psi) \\ -c(\phi)s(\theta)s(\psi) + s(\phi)c(\psi) \\ c(\phi)c(\theta) \end{bmatrix}\frac{T}{m} \tag{6.4}$$

(6.5)

$$u_\psi \stackrel{\text{def}}{=} f_\psi(x,\nu) = \frac{s(\phi)}{c(\theta)}p + \frac{c(\phi)}{c(\theta)}r \tag{6.6}$$

where we end up with

$$\begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix} = u_p + \begin{bmatrix} 0 \\ 0 \\ -g \end{bmatrix} \tag{6.7}$$

$$\dot{\psi} = u_\psi \tag{6.8}$$

where $u_p = [u_{p1}, u_{p2}, u_{p3}]^T$ is the linear control input for $(\ddot{x}, \ddot{y}, \ddot{z})$ and u_ψ is the linear control input for $\dot{\psi}$. Now we can obtain the linear system

$$\dot{x} = Ax + Bu + hg \tag{6.9}$$

where $x = [x, y, z, \dot{x}, \dot{y}, \dot{z}, \psi]^T$, $u = [u_p, u_\psi]^T$.

$$A = \begin{bmatrix} 0_{3\times3} & I_{3\times3} & 0_{3\times1} \\ 0_{3\times3} & 0_{3\times3} & 0_{3\times1} \\ 0_{1\times3} & 0_{1\times3} & 0_{1\times1} \end{bmatrix},\ B = \begin{bmatrix} 0_{3\times3} & 0_{3\times1} \\ I_{3\times3} & 0_{3\times1} \\ 0_{1\times3} & I_{1\times1} \end{bmatrix},\ h = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ -1 \\ 0 \end{bmatrix} \tag{6.10}$$

Once we obtain u, we can convert back using the inverse functions

$$\nu = \begin{bmatrix} T \\ \phi \\ \theta \\ r \end{bmatrix} = \begin{bmatrix} f_p^{-1}(x, \nu) \\ f_\psi^{-1}(x, \nu) \end{bmatrix} \tag{6.11}$$

to obtain T, we take the 2-norm for both sides of

$$u_p = R_z^T(\psi) R_y^T(\theta) R_x^T(\phi) \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \frac{T}{m} \tag{6.12}$$

because $R^T R$ is the identity for rotation matrices. Therefore, we obtain the following

$$T = m\sqrt{u_{p1}^2 + u_{p2}^2 + u_{p3}^2} \tag{6.13}$$

basically, for $f_p(x, \nu)$ we have three equations and four unknowns (T, ϕ, θ, ψ). Because ψ does not affect the path tracking, we multiple both sides by $R_z(\psi)$ and define $z = [z_1, z_2, z_3]^T$

$$\begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix} = R_z(\psi) u_p \frac{m}{T} = R_y^T(\theta) R_x^T(\phi) \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \tag{6.14}$$

after computing z values, we can solve for ϕ and θ

$$\begin{aligned} \phi &= sin^{-1}(z_2) \\ \theta &= tan^{-1}(\frac{-z_1}{z_3}) \end{aligned} \tag{6.15}$$

finally we can solve for r using $f_\psi^{-1}(x, \nu)$

$$r = u_\psi \frac{c(\theta)}{c(\phi)} + q\frac{s(\phi)}{c(\phi)} \tag{6.16}$$

Differential Flatness. From [16, 17, 33], a flat differential system is one in which the states and inputs can be expressed as functions of the output and its time derivatives without integration.

$$\begin{aligned} y &= y(x, u, \dot{u}, \ddot{u}, ...) \\ x &= x(y, \dot{y}, \ddot{y}, ...) \\ u &= u(y, \dot{y}, \ddot{y}, ...) \end{aligned} \tag{6.17}$$

where $y^{ref} = [x^{ref}, y^{ref}, z^{ref}, \psi^{ref}]^T$, $x^{ref} = [y^{ref}, \dot{y}^{ref}]^T$, and $u^{ref} = [y^{ref}, \dot{y}^{ref}, \ddot{y}^{ref}]^T$. From above, we know that $u^{ref} = [u_p^{ref}, u_\psi^{ref}]^T$ where we end up with the following system

$$\dot{x}^{ref} = Ax^{ref} + Bu^{ref} + hg \tag{6.18}$$

where error state-space model becomes as follows

$$\dot{e} = Ae + Bu_e \tag{6.19}$$

where $e = x^{ref} - x$ and $u_e = u^{ref} - u$. The full-state feedback controller $u_e = -Ge$ as shown in the block diagram

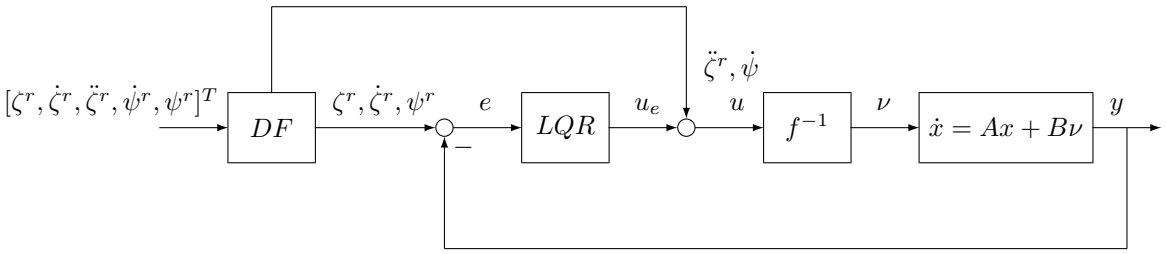


Figure 6.1: High-level Feedback Block Diagram

Algorithm 1 Compute High-Level Control Commands

1. Compute all the seven error states:

   $e_x = x^{ref} - x, \qquad e_y = y^{ref} - y, \qquad e_z = z^{ref} - z$

   $e_{vx} = \dot{x}^{ref} - \dot{x}, \qquad e_{vy} = \dot{y}^{ref} - \dot{y}, \qquad e_{vz} = \dot{z}^{ref} - \dot{x}, \qquad e_\psi = \psi^{ref} - \psi$

2. Compute the four error integral term for (x, y, z):

   $e_{xi} += e_x \; dt, \qquad e_{yi} += e_y \; dt, \qquad e_{zi} += e_z \; dt, \qquad e_{\psi i} += e_\psi \; dt$

3. Compute the LQR output control commands (u_1, u_2, u_3, u_4) as follows:

   $u_1 = g_1 e_x + g_2 e_{vx} + g_3 e_{xi}, \qquad u_2 = g_1 e_y + g_2 e_{vy} + g_3 e_{yi}$

   $u_3 = g_4 e_z + g_5 e_{vz} + g_6 e_{zi}, \qquad u_4 = g_7 e_\psi + g_8 e_{\psi i}$

4. Compute the feed-forward terms:

   $u_{p1} = u_1 + \ddot{x}^{ref}, \qquad u_{p2} = u_2 + \ddot{y}^{ref}, \qquad u_{p3} = u_3 + \ddot{z}^{ref} + 9.81$

   $u_{p\psi} = u_4 + \dot{\psi}^{ref}$

5. Compute the desired commands (θ, ϕ, T, r) from the inverse mapping:

   $T = m\sqrt{u_{p1}^2 + u_{p2}^2 + u_{p3}^2}$

   $z_1 = \frac{m}{T}(u_{p1} cos(\psi) + u_{p2} sin(\psi))$

   $z_2 = \frac{m}{T}(-u_{p1} sin(\psi) + u_{p2} cos(\psi))$

   $z_3 = \frac{m}{T} u_{p3}$

   $\phi = asin(z_2)$

   $\theta = atan(\frac{-z_1}{z_3})$

   $r = u_{p\psi} \frac{cos(\theta)}{cos(\phi)} + q \frac{sin(\phi)}{cos(\phi)}$

6.2.1 Linearization of Nonlinear Translational Dynamics Near Hover

From [34], linearizing the nonlinear translational quadrotor dynamics around the hover position $\phi = \theta = \psi = 0°$, $T = mg$, yields the following linear model:

$$\begin{aligned} \dot{x}_p &= A_p x_p + B_p u_p \\ y_p &= C_p x_p \\ u_p &= \begin{bmatrix} \theta & \phi & T & r \end{bmatrix}^T \\ x_p &= \begin{bmatrix} x & y & z & \dot{x} & \dot{y} & \dot{z} & \psi \end{bmatrix}^T \\ y_p &= \begin{bmatrix} x & y & z & \psi \end{bmatrix}^T \end{aligned} \tag{6.20}$$

$$A_{p1} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \; B_{p1} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ -9.81 & 0 & 0 & 0 \\ 0 & 9.81 & 0 & 0 \\ 0 & 0 & 1.5015 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.21}$$

$$C_{p1} = \begin{bmatrix} I_{3\times 3} & 0_{3\times 3} & 0 \\ 0_{1\times 3} & 0_{1\times 3} & 1 \end{bmatrix}$$

By performing coordinate transformation as in [30]

$$\begin{aligned} u &= S_u u_{p1} \\ x &= S_x x_{p1} \\ y &= S_y y_{p1} \end{aligned} \tag{6.22}$$

where

$$S_u = \begin{bmatrix} \frac{180}{\pi} & 0 & 0 & 0 \\ 0 & \frac{180}{\pi} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & \frac{180}{\pi} \end{bmatrix}, S_x = \begin{bmatrix} I_{6\times 6} & 0_{6\times 1} \\ 0_{1\times 6} & \frac{180}{\pi} \end{bmatrix}, S_y = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & \frac{180}{\pi} \end{bmatrix} \quad (6.23)$$

we end up with the following system

$$\begin{aligned} A_p &= S_x A_{p1} S_x^{-1} \\ B_p &= S_x B_{p1} S_u^{=1} \\ C_p &= S_y C_{p1} S_x^{-1} \\ D_p &= S_y D_{p1} S_u^{-1} \end{aligned} \quad (6.24)$$

where the eigenvalues/eigenvectors of the A_p matrix is given using MATLAB 'eig(A)' command:

$$V = \begin{bmatrix} I_{3\times 3} & -I_{3\times 3} & 0_{3\times 1} \\ 0_{4\times 3} & 0_{4\times 3} & 0_{4\times 1} \end{bmatrix}, \; U = 0_{7\times 7} \quad (6.25)$$

where V and U is the eigenvector and eigenvalues matrix, respectively. The system has no finite transmission zeros. The system transfer function matrix is as follows

$$P(s) = C_p(sI - A_p)^{-1}B_p = \begin{bmatrix} \frac{-0.17122}{s^2} & 0 & 0 & 0 \\ 0 & \frac{0.17122}{s^2} & 0 & 0 \\ 0 & 0 & \frac{1.502}{s^2} & 0 \\ 0 & 0 & 0 & \frac{1}{s} \end{bmatrix} \quad (6.26)$$

The off-diagonal elements are zero, which implies that there is no coupling between inputs and outputs.

System Analysis at Low Frequencies. Performing singular value decomposition

(SVD) at $w_o = 0.01 rad/s$ yields $P = USV^H$

$$U = \begin{bmatrix} u_1 & u_2 & u_3 & u_4 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$S = diag(\sigma_1, \sigma_2, \sigma_3, \sigma_4) = \begin{bmatrix} 15015 & 0 & 0 & 0 \\ 0 & 1712 & 0 & 0 \\ 0 & 0 & 1712 & 0 \\ 0 & 0 & 0 & 100 \end{bmatrix} \tag{6.27}$$

$$V = \begin{bmatrix} v_1 & v_2 & v_3 & v_4 \end{bmatrix} = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

From SVD, we have the following:

- maximum singular value of 15015 is associated with the vehicle's z position ($u_1 = [\ 0\ \ 0\ \ 1\ \ 0\ ]^T$), and the vehicle's input T ($v_1 = [\ 0\ \ 0\ \ 1\ \ 0\ ]$).
- minimum singular value of 100 is associated with vehicle's yaw angle ψ ($u_4 = [\ 0\ \ 0\ \ 0\ \ 1\ ]^T$), and the vehicle's input r ($v_4 = [\ 0\ \ 0\ \ 0\ \ 1\ ]$).

From the above, we can see that it is easier to thrust in the z-direction than to yaw at an angle.

Plant Singular Values. In Figure 6.2 below, singular values have large magnitudes at low frequencies; this is expected since we have integrators in each channel in the plant. The blue singular values are associated with the plant model without drag term and the red ones associated with plant model with linear drag term. Also, in

Figure 6.3, we see a comparison between singular values for the plant without control input dynamics (blue) and with control input dynamics (black).

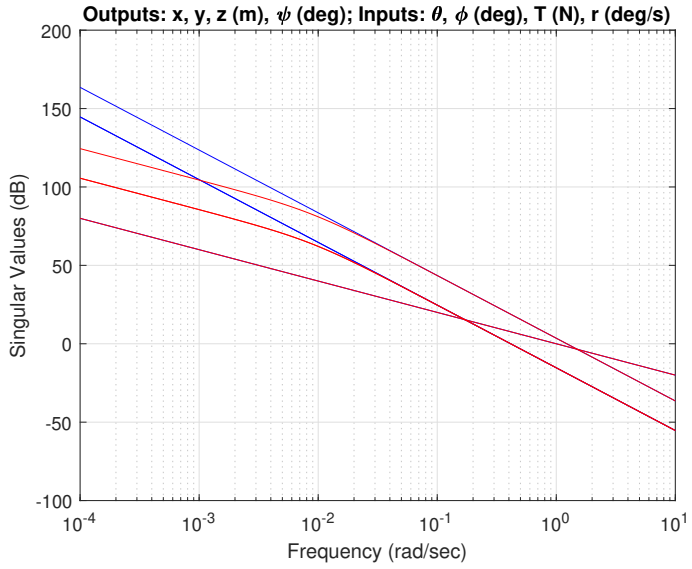


Figure 6.2: Quadrotor Singular Values: (Blue) Model Without Drag, (Red) Model with Drag

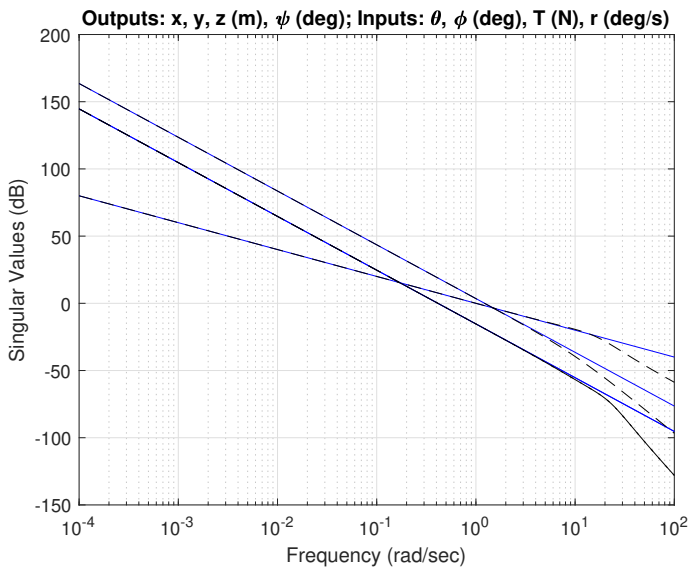


Figure 6.3: Quadrotor Singular Values: (Blue) Model Without Drag, (Black Dots) Model with Inner-loop Dynamics

6.2.2 Linearization of Nonlinear Translational Dynamics for Forward Flight

In [35], the aerial drag force is expressed as follows

$$F_d = \frac{1}{2}\rho C_d v^2 A = \frac{1}{2}\rho C_d v^2 (S_t sin(\alpha) + S_s cos(\alpha)) \tag{6.28}$$

where ρ is the air density, v is the quadrotor's speed, C_d is the drag force coefficient, A is an efficient drag area, S_t is the planar area from top view, S_s is the planar area from side, α is the angle of attack. For small α, we can neglect the S_t term. Also, we can treat α as the pitch angle ϕ. The acceleration along the x-axis with drag term can be expressed as follows:

$$\ddot{x} = \frac{T}{m}\big(- c(\phi)s(\theta)c(\psi) - s(\phi)s(\psi)\big) - \frac{1}{2m}\rho C_d v^2 S_s c(\theta) \tag{6.29}$$

Linearizing the nonlinear translational quadrotor dynamics in forward flight where

$\theta_o = -30°$, $\phi_o = \psi_o = 0°$, $T_o = \frac{mg}{cos(\theta_o)}$, yields the following linear model:

$$
\begin{aligned}
\dot{x}_{p1} &= A_{p1}x_{p1} + B_{p1}u_{p1} \\
y_{p1} &= C_{p1}x_{p1} \\
u_{p1} &= \begin{bmatrix} \theta & \phi & T & r \end{bmatrix}^T \\
x_{p1} &= \begin{bmatrix} x & y & z & \dot{x} & \dot{y} & \dot{z} & \psi \end{bmatrix}^T \\
y_{p1} &= \begin{bmatrix} x & y & z & \psi \end{bmatrix}^T
\end{aligned}
\tag{6.30}
$$

$$
A_{p1} = \begin{bmatrix}
0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & -0.0078 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 5.6638 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix},
$$

$$
B_{p1} = \begin{bmatrix}
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
-9.8233 & 0 & 0.7508 & 0 \\
0 & 11.3276 & 0 & 0 \\
5.6638 & 0 & 1.3003 & 0 \\
0 & 0 & 0 & 1.1547
\end{bmatrix}
\tag{6.31}
$$

$$
C_{p1} = \begin{bmatrix}
I_{3\times3} & 0_{3\times3} & 0 \\
0_{1\times3} & 0_{1\times3} & 1
\end{bmatrix}
$$

by performing coordinate transformation

$$\begin{aligned} u &= S_u u_{p1} \\ x &= S_x x_{p1} \\ y &= S_y y_{p1} \end{aligned} \tag{6.32}$$

where

$$S_u = \begin{bmatrix} \frac{180}{\pi} & 0 & 0 & 0 \\ 0 & \frac{180}{\pi} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & \frac{180}{\pi} \end{bmatrix}, S_x = \begin{bmatrix} I_{6\times 6} & 0_{6\times 1} \\ 0_{1\times 6} & \frac{180}{\pi} \end{bmatrix}, S_y = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & \frac{180}{\pi} \end{bmatrix} \tag{6.33}$$

where we end up with the following system

$$A_p = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -0.0078 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.0989 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \; B_p = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ -0.1714 & 0 & 0.7508 & 0 \\ 0 & 0.1977 & 0 & 0 \\ 0.0989 & 0 & 1.3003 & 0 \\ 0 & 0 & 0 & 1.1547 \end{bmatrix} \tag{6.34}$$

$$C_p = \begin{bmatrix} I_{3\times 3} & 0_{3\times 3} & 0 \\ 0_{1\times 3} & 0_{1\times 3} & 1 \end{bmatrix}$$

where the eigenvalues/eigenvectors of the A_p matrix is given using MATLAB 'eig(A)'

command:

$$V = \begin{bmatrix} 1 & 0 & 0 & -1.0000 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & -1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0.0078 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$U = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -0.0078 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{6.35}$$

where V and U is the eigenvector and eigenvalues matrix, respectively. The system is both fully controllable and fully observable. Also, the system has no finite

transmission zeros where the system transfer function matrix is as follows

$$
\begin{aligned}
P(s) &= C_p(sI - A_p)^{-1}B_p \\
&= \begin{bmatrix}
\frac{-\left[\frac{C_d\rho(-S_s sin(\theta))v_x}{2} + \frac{Tcos(\theta)}{m}\right]}{s\left(s+C_d\rho v_x(S_s cos(\theta))\right)} & 0 & \frac{-sin(\theta)}{ms\left(s+C_d\rho v_x(S_s cos(\theta))\right)} & 0 \\
0 & \frac{T}{ms^2} & 0 & \frac{-Tsin(\theta)}{ms^3 cos(\theta)} \\
\frac{-Tsin(\theta)}{ms^2} & 0 & \frac{cos(\theta)}{ms^2} & 0 \\
0 & 0 & 0 & \frac{1}{scos(\theta)}
\end{bmatrix} \\
&= \begin{bmatrix}
\frac{-0.1715}{s(s+0.0077)} & 0 & \frac{0.7508}{s(s+0.0077)} & 0 \\
0 & \frac{0.1977}{s^2} & 0 & \frac{0.11414}{s^3} \\
\frac{0.09885}{s^2} & 0 & \frac{1.3003}{s^2} & 0 \\
0 & 0 & 0 & \frac{1.1547}{s}
\end{bmatrix}
\end{aligned}
\tag{6.36}
$$

where off-diagonal elements are nonzero, which implies that inputs are cross-coupled to the outputs, furthermore, we can observe that as the quadrotor's speed increases the poles of $P_{1,1}$ and $P_{1,2}$ move more to the left half plane and as pitch angle θ goes from $0^\circ \rightarrow -90^\circ$ the off-diagonal elements' magnitude goes up. In other words, the coupling between the inputs and outputs increases. To get a decoupled system (zero off-diagonal), we need θ to be close to zero, i.e., the attitude angles need to be small.

System Analysis at Low Frequencies. Performing singular value decomposition

(SVD) at $w_o = 0.01 rad/s$ yields $P = USV^H$

$$U = \begin{bmatrix} u_1 & u_2 & u_3 & u_4 \end{bmatrix} = \begin{bmatrix} 0 & 0.3047 & 0.9525 & 0 \\ 1 & 0 & 0 & -0.0010 \\ 0 & 0.9525 & -0.3047 & 0 \\ 0.0010 & 0 & 0 & 1 \end{bmatrix}$$

$$S = diag(\sigma_1, \sigma_2, \sigma_3, \sigma_4) = \begin{bmatrix} 114160 & 0 & 0 & 0 \\ 0 & 13690 & 0 & 0 \\ 0 & 0 & 1220 & 0 \\ 0 & 0 & 0 & 2 \end{bmatrix} \tag{6.37}$$

$$V = \begin{bmatrix} v_1 & v_2 & v_3 & v_4 \end{bmatrix} = \begin{bmatrix} 0 & 0.0474 & -0.9989 & 0 \\ 0.0173 & 0 & 0 & -0.9999 \\ 0 & 0.9989 & 0.0474 & 0 \\ 0.9999 & 0 & 0 & 0.0173 \end{bmatrix}$$

From this svd, we have the following:

- maximum singular value of 114160 is associated with the vehicle's y position ($u_1 = [\ 0\ \ 1\ \ 0\ \ 0.001\ ]^T$), and the vehicle's input r ($v_1 = [\ 0\ \ 0.0173\ \ 0\ \ 0.9999\ ]$).
- minimum singular value of 2 is associated with vehicle's yaw angle ψ ($u_4 = [\ 0\ \ -0.001\ \ 0\ \ 1\ ]^T$), and the vehicle's input ϕ ($v_4 = [\ 0\ \ -0.9999\ \ 0\ \ 0.0173\ ]$).

From the above, we can see that $P(j0.01)v = USV^H v = \sum_{i=1}^{4} \sigma_i u_i v_i^H$. Since the second component of u_1 is much larger than the other and the fourth component of v_1 is much larger than the other, it follows that the singular value $\sigma_1 = 114160$ is primarily associated with r and y. Also, the third component of u_2 is much larger than the other and the third component of v_2 is much larger than the other, it follows that the singular value $\sigma_2 = 13690$ is primarily associated with T and z. The first

component of u_3 is much larger than the other and the first component of v_3 is much larger than the other, it follows that the singular value $\sigma_3 = 1220$ is primarily associated with θ and x. The fourth component of u_4 is much larger than the other and the second component of v_3 is much larger than the other, it follows that the singular value $\sigma_4 = 2$ is primarily associated with ϕ and ψ. In other words, we can say that it is easier to turn in the y-axis position using r control input than to yaw at an angle with using ϕ control input.

System Analysis at Different Frequencies. We perform the singular value decomposition (SVD) at a particular frequencies of interest. Starting with frequency $\omega = 0.078$ rad/s where we see two singular values approximately match at that frequency. From SVD, at $\omega = 0.078$ rad/s both singular values σ_1 and σ_2 are almost equal to each other. Also, at $\omega = 0.8$ rad/s both singular values σ_3 and σ_4 are almost equal to each other. Similarly, both singular values σ_1 and σ_2 match each other at $\omega = 1.32$ rad/s. **Singular Values.**

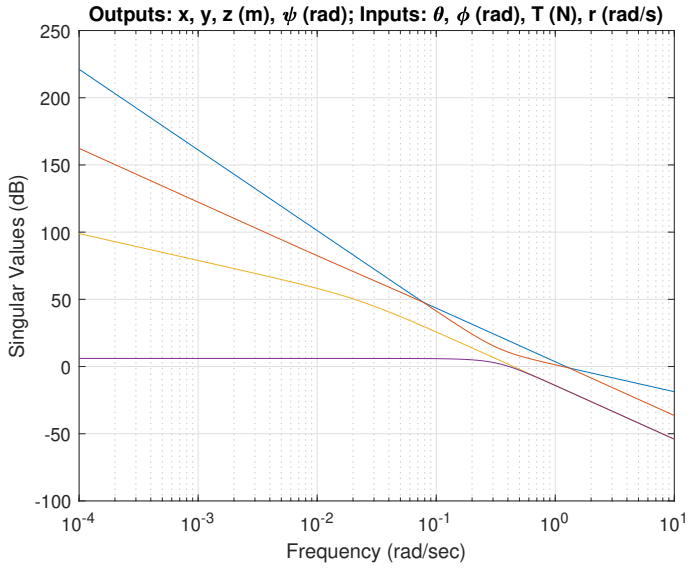


Figure 6.4: Quadrotor Singular Values for Forward Flight

6.3 LQ Servo Design

From now on, we will consider the decoupled model where the angles near zero (near hover) without the coordinate transformation. The reason is that our attitude system was designed in units of radians; therefore, we need to design the position control in the same units as well. From [30], the objective is to design a full state feedback control law where $u = -Gx$. The goal is to obtain stabilizing $G's$, i.e. $Re\lambda_i(A - BG) < 0$. The approach is to optimize a quadratic "cost functional"

$$\begin{aligned} &\min_u J(u) \stackrel{def}{=} \frac{1}{2}\int_0^\infty (x^T Q x + u^T R u) d\tau \\ &subject\ to \qquad \dot{x} = Ax + Bu \\ &\qquad\qquad x(0) = x_0 \end{aligned} \tag{6.38}$$

where $Q = Q^T \geq 0 \in \mathcal{R}^{nxn}$ is the state weighting matrix, $R = R^T > 0 \in \mathcal{R}^{mxm}$ is the control weighting matrix.

LQ Servo Design 1. The plant P was augmented with an integrator in each channel to guarantee a zero steady-state error to step reference commands as follows

$$A = \begin{bmatrix} A_p & 0_{7\times 4} \\ C_p & 0_{4\times 4} \end{bmatrix},\ B = \begin{bmatrix} B_p \\ 0_{4\times 4} \end{bmatrix},\ C = \begin{bmatrix} C_p & 0_{4\times 4} \end{bmatrix} \tag{6.39}$$

where the state of the system $x = [x_p \quad x_i]^T$. The solution of the optimization problem is as follows

$$u = -Gx \tag{6.40}$$

$$G = R^{-1}B^T K \tag{6.41}$$

where K is the unique symmetric ($K = K^T$) at least positive semi-definite ($K \geq 0$) of the Control Algebriac Riccati Equation (CARE)

$$0 = KA + A^T K + M^T M - KBR^{-1}B^T K \tag{6.42}$$

where

$$Q = diag(10, 10, 10, 10, 10, 10, 1, 100, 100, 100, 1), \quad R = \rho I_{4\times 4}, \quad \rho = 0.1 \tag{6.43}$$

$$\begin{aligned}
G_y &= \begin{bmatrix} -40.0977 & 0.0000 & 0.0000 & -0.0000 \\ 0.0000 & 40.0977 & 0.0000 & 0.0000 \\ -0.0000 & 0.0000 & 29.0671 & 0.0000 \\ 0.0000 & -0.0000 & 0.0000 & 4.0404 \end{bmatrix} \\
G_i &= \begin{bmatrix} -31.6228 & -0.0000 & 0.0000 & -0.0000 \\ -0.0000 & 31.6228 & -0.0000 & 0.0000 \\ -0.0000 & 0.0000 & 31.6228 & -0.0000 \\ 0.0000 & 0.0000 & 0.0000 & 3.1623 \end{bmatrix} \\
G_r &= \begin{bmatrix} -23.8408 & 0.0000 & 0.0000 \\ -0.0000 & 23.8408 & -0.0000 \\ -0.0000 & -0.0000 & 11.7778 \\ 0.0000 & 0.0000 & 0.0000 \end{bmatrix}
\end{aligned} \tag{6.44}$$

From [12, 30], the following closed loop LQ servo block diagram is shown below

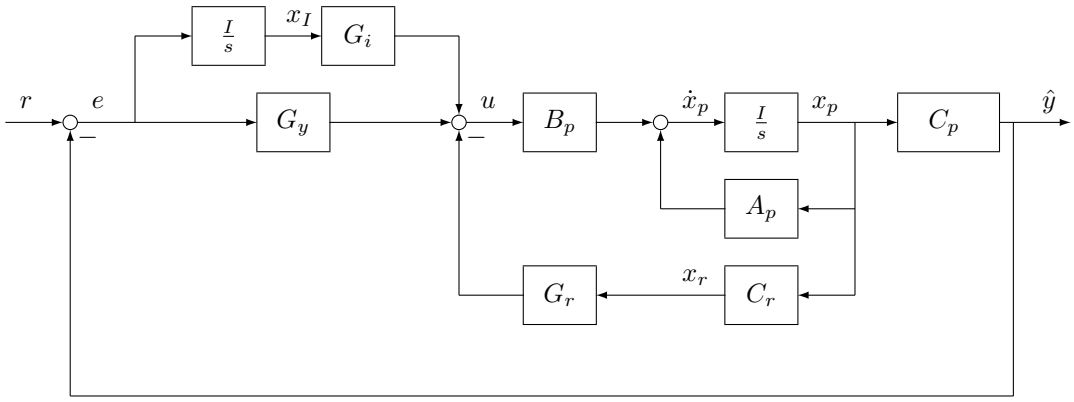


Figure 6.5: LQ Servo with Dynamic Augmentation Block Diagram - LQ Servo Design 1

From the block diagram, we can obtain the state-space representation for the open-loop at the error signal e or the plant output y as follows

$$A_{ol} = \begin{bmatrix} A_p - B_p G_r C_r & B_p G_i \\ 0_{4\times 7} & 0_{4\times 4} \end{bmatrix},\ B_{ol} = \begin{bmatrix} B_p G_y \\ I_{4\times 4} \end{bmatrix},\ C_{ol} = \begin{bmatrix} C_p & 0_{4\times 4} \end{bmatrix},\ D_{ol} = 0_{4\times 4} \tag{6.45}$$

where the state of the system $x = [x_p \quad x_i]$ and the input to the system is the error signal $e = r - y$. From that, we can obtain the closed loop system as follows

$$A_{cl} = A_{ol} - B_{ol} C_{ol} \qquad B_{cl} = B_{ol} \qquad C_{cl} = C_{ol} \qquad D_{cl} = D_{ol} \tag{6.46}$$

The closed loop poles are shown in Table 6.1.

Poles	Damping	Frequency (rad/sec)
-1.06e+00	1.00e+00	1.06e+00
-1.04e+00 + 1.27e+00i	6.32e-01	1.64e+00
-1.04e+00 - 1.27e+00i	6.32e-01	1.64e+00
-1.04e+00 + 1.27e+00i	6.32e-01	1.64e+00
-1.04e+00 - 1.27e+00i	6.32e-01	1.64e+00
-1.35e+00 + 1.16e+00i	7.59e-01	1.78e+00
-1.35e+00 - 1.16e+00i	7.59e-01	1.78e+00
-2.00e+00	1.00e+00	2.00e+00
-2.00e+00	1.00e+00	2.00e+00
-2.98e+00	1.00e+00	2.98e+00
-1.50e+01	1.00e+00	1.50e+01

Table 6.1: Closed Loop Poles - LQ Servo Design 1

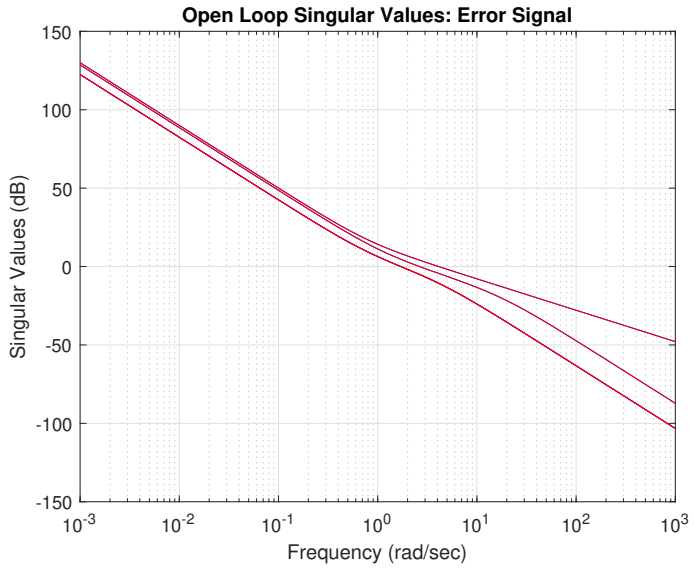


Figure 6.6: Quadrotor Open Loop Singular Values at Error: (Blue) No Drag (Red) with Drag - LQ Servo Design 1

Open Loop Frequency Response at Error. By breaking the loop at the error signal, we get the open loop transfer function matrix L_e. As shown in Figure 6.6, at low frequencies, we have a slope of at least -20dB/dec for all singular values due to an integral action in each control channel. The plot shows that reference commands r below 0.38 rad/s will be followed within 10% steady-state error. Similarly, output disturbances d_o below 0.38 rad/s will be attenuated by 20dB (0.1). In addition to sensor noise n with frequencies above 40 rad/s will be attenuated by 20dB (0.1).

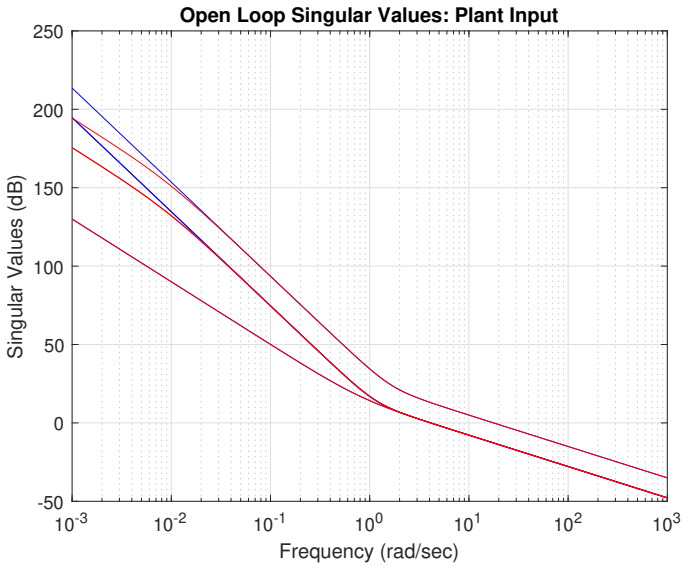


Figure 6.7: Quadrotor Open Loop Singular Values at Input: (Blue) No Drag (Red) with Drag - LQ Servo Design 1

Open Loop Frequency Response at Plant Input. By breaking the loop at the plant input, we get the open loop transfer function matrix L_u. As shown in Figure 6.7, the input disturbances d_i below 0.6 rad/s will be attenuated by 20dB. In other words, the input signal u will have a disturbance of 10% of the magnitude of d_i.

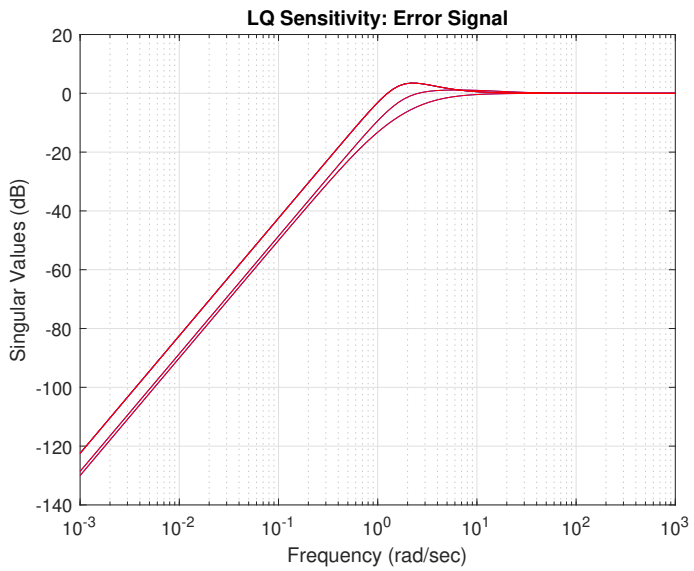


Figure 6.8: Quadrotor Sensitivity Frequency Response at Error: (Blue) No Drag (Red) with Drag - LQ Servo Design 1

Sensitivity Frequency Response at Error. The sensitivity frequency response plot suggests that reference commands below 0.3 rad/s will be followed within 10% steady-state error. Also, output disturbances d_o below 0.3 rad/s will be attenuated by $20dB$. A bump at 2 rad/s with a magnitude of $3.3dB$ is shown in the plot, which will result in an overshoot in step reference commands.

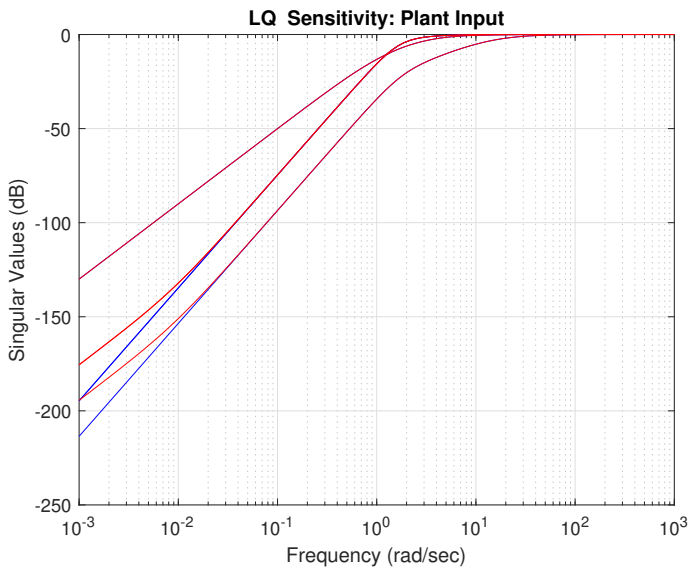


Figure 6.9: Quadrotor Sensitivity Frequency Response at Input: (Blue) No Drag (Red) with Drag - LQ Servo Design 1

Sensitivity Frequency Response at Input. The sensitivity transfer function S_u, which shows the impact of the input disturbance d_i on the input signal u. From the plot, we see that d_i will be attenuated by $20dB$ for frequencies below 0.6 rad/s.

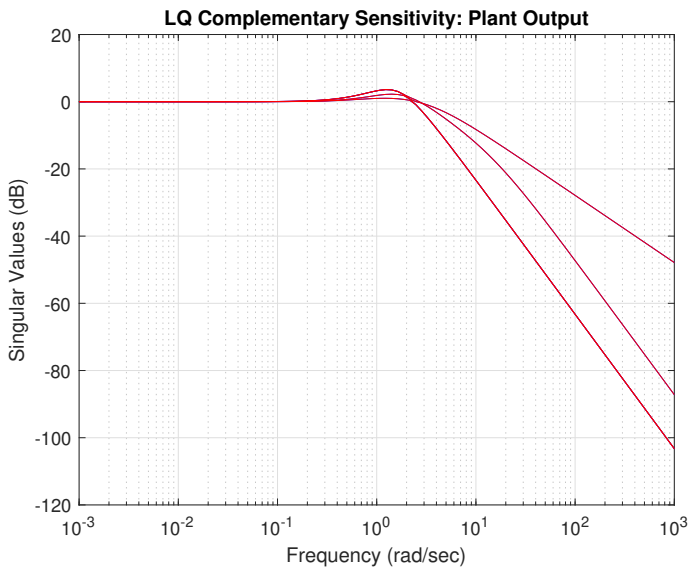


Figure 6.10: Quadrotor Complementary Sensitivity Frequency Response at Output: (Blue) No Drag (Red) with Drag - LQ Servo Design 1

Complementary Sensitivity Frequency Response at Output. The singular values of the complementary sensitivity at the output T_e suggest that noise n for frequencies above 40 rad/s will be attenuated by $20dB$. Also, a peak of $3.6dB$ is present at $\omega \approx 1.2$ rad/s, which suggests an overshoot to step reference commands.

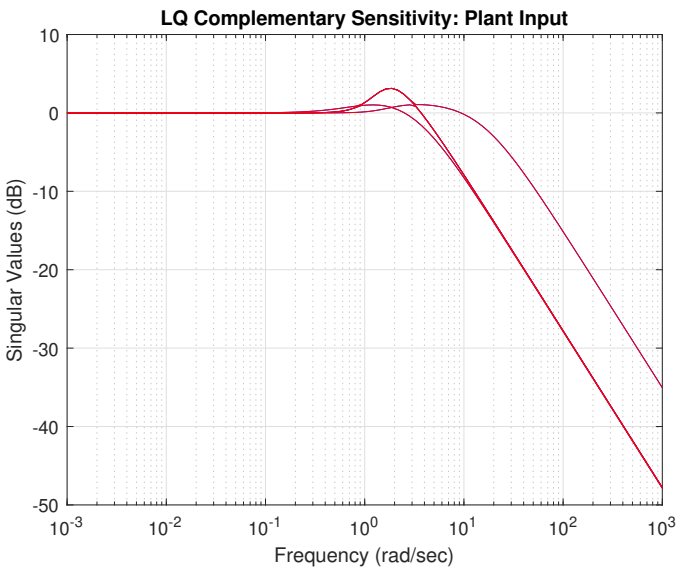


Figure 6.11: Quadrotor Complementary Sensitivity Frequency Response at Input: (Blue) No Drag (Red) with Drag - LQ Servo Design 1

LQ Servo Design: Time Domain Analysis. Step reference commands were applied for each output y to obtain the closed loop system step response.

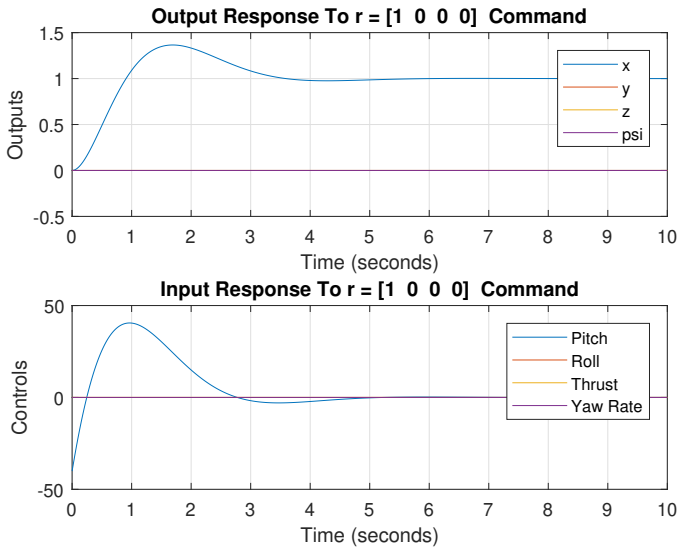


Figure 6.12: Step Response for Reference Command $r = [1\ 0\ 0\ 0]^t$ - LQ Servo Design 1

For Figure,

- The output response has an overshoot of 41%, rise time of $0.89s$, settling time of $6.9s$.
- The control response is not aggressive where the angle goes from approximately $-45°$ to $45°$.

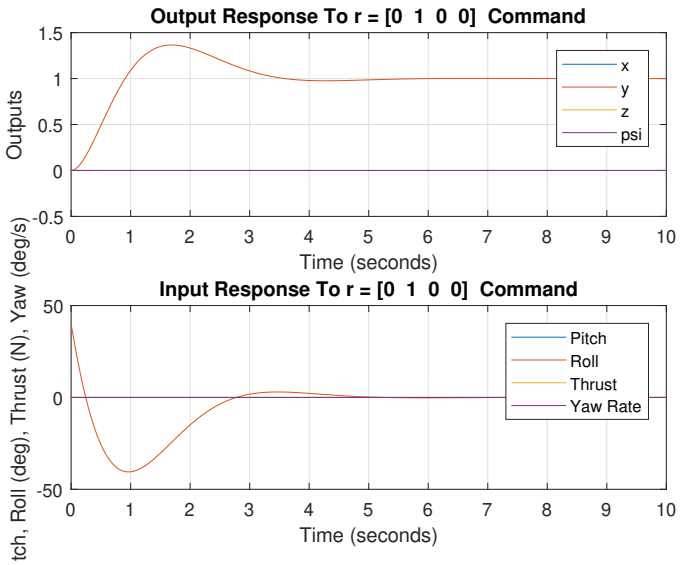


Figure 6.13: Step Response for Reference Command $r = [0\ 1\ 0\ 0]^t$ - LQ Servo Design 1

For Figure,

- The output response has an overshoot of 41%, rise time of $0.89s$, settling time of $6.9s$.
- The control response is not aggressive where the angle goes from approximately $-45°$ to $45°$.

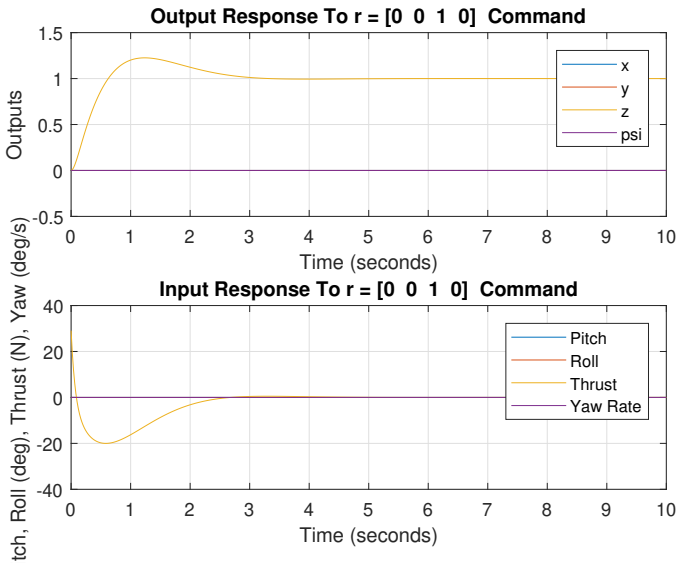


Figure 6.14: Step Response for Reference Command $r = [0\ 0\ 1\ 0]^t$ - LQ Servo Design 1

For Figure,

- The output response has an overshoot of 29%, rise time of $0.49s$, settling time of $2.98s$.
- The control response is too aggressive where thrust reaches a magnitude of $20N$ from the equilibrium.

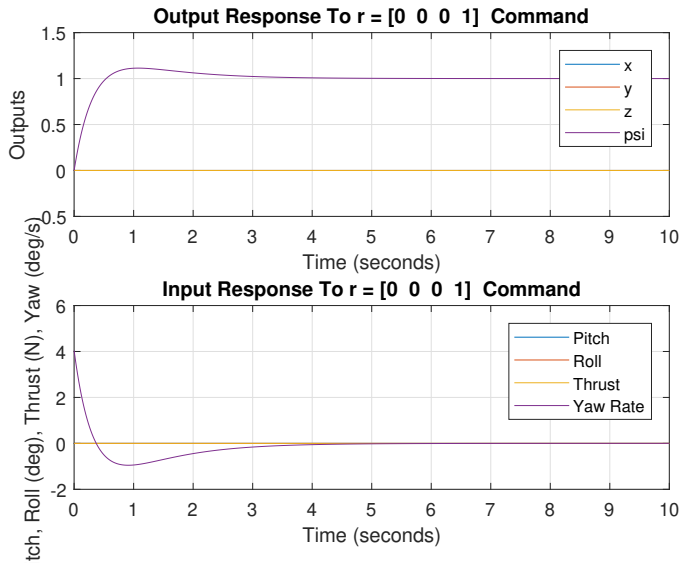


Figure 6.15: Step Response for Reference Command $r = [0\ 0\ 0\ 1]^t$ - LQ Servo Design 1

For Figure,

- The output response has an overshoot of 16.3%, rise time of $0.78s$, settling time of $5.1s$.
- The control response is not aggressive where the yaw angular rate goes from approximately $4^\circ/s$ to $-1^\circ/s$.

LQ Servo Design 2. The plant P was augmented with an integrator in each channel to guarantee a zero steady-state error to step reference commands as follows

$$A = \begin{bmatrix} A_p & 0_{7\times4} \\ C_p & 0_{4\times4} \end{bmatrix},\ B = \begin{bmatrix} B_p \\ 0_{4\times4} \end{bmatrix},\ C = \begin{bmatrix} I_{7\times7} \end{bmatrix} \tag{6.47}$$

where the state of the system $x = [x_p \quad x_i]^T$. The solution of the optimization problem is as follows

$$u = -Gx \tag{6.48}$$

$$G = R^{-1}B^{T}K \tag{6.49}$$

where K is the unique symmetric ($K = K^{T}$) at least positive semi-definite ($K \geq 0$) of the Control Algebriac Riccati Equation (CARE)

$$0 = KA + A^{T}K + M^{T}M - KBR^{-1}B^{T}K \tag{6.50}$$

where

$$Q = diag(10, 10, 10, 10, 10, 10, 100, 100, 100, 100, 1), \quad R = \rho I_{4\times 4}, \quad \rho = 0.2 \tag{6.51}$$

$$G_y = \begin{bmatrix} -30.9362 & 0.0000 & 0.0000 & -20.2822 & 0.0000 & 0.0000 & 0.0000 \\ 0.0000 & 30.9362 & 0.0000 & 0.0000 & 20.2822 & 0.0000 & 0.0000 \\ -0.0000 & 0.0000 & 1.0713 & 0.0000 & 0.0000 & 8.8378 & 0.0000 \\ 0.0000 & -0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 22.4605 \end{bmatrix}$$

$$G_i = \begin{bmatrix} -22.3607 & 0.0000 & 0.0000 & 0.0000 \\ 0.0000 & 22.3607 & 0.0000 & 0.0000 \\ 0.0000 & 0.0000 & 22.3607 & 0.0000 \\ 0.0000 & 0.0000 & 0.0000 & 2.2361 \end{bmatrix} \tag{6.52}$$

From [12, 30], the following closed loop LQ servo block diagram is shown below

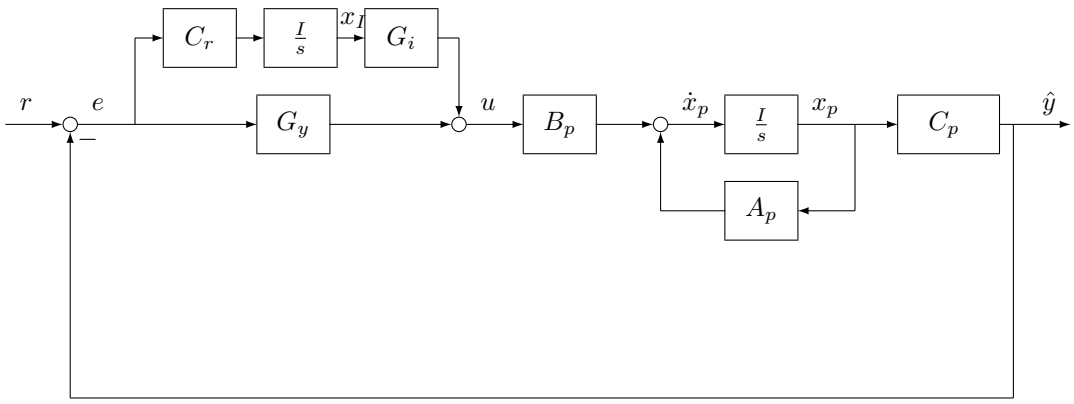


Figure 6.16: LQ Servo with Dynamic Augmentation Block Diagram - LQ Servo Design 2

From the block diagram, we can obtain the state-space representation for the open-loop at the error signal e or the plant output y as follows

$$A_{ol} = \begin{bmatrix} A_p & B_pG_i \\ 0_{4\times7} & 0_{4\times4} \end{bmatrix},\ B_{ol} = \begin{bmatrix} B_pG_y \\ C_r \end{bmatrix},\ C_{ol} = \begin{bmatrix} C_p & 0_{4\times4} \end{bmatrix},\ D_{ol} = 0_{7\times7} \quad (6.53)$$

where the state of the system $x = [x_p \quad x_i]$ and the input to the system is the error signal $e = r - y$. From that, we can obtain the closed loop system as follows

$$A_{cl} = \begin{bmatrix} A_p - B_pG_yC_p & B_pG_i \\ -C_rC_p & 0_{4\times4} \end{bmatrix},\ B_{cl} = \begin{bmatrix} B_pG_y \\ C_r \end{bmatrix},\ C_{cl} = \begin{bmatrix} C_p & 0_{4\times4} \end{bmatrix},\ D_{cl} = 0_{7\times7} \quad (6.54)$$

where the closed loop poles are shown in Table 6.2.

Poles	Damping	Frequency (rad/sec)
-1.00e-01	1.00e+00	1.00e-01
-8.99e-01 + 1.22e+00i	5.95e-01	1.51e+00
-8.99e-01 - 1.22e+00i	5.95e-01	1.51e+00
-8.99e-01 + 1.22e+00i	5.95e-01	1.51e+00
-8.99e-01 - 1.22e+00i	5.95e-01	1.51e+00
-1.67e+00	1.00e+00	1.67e+00
-1.67e+00	1.00e+00	1.67e+00
-1.35e+00 + 1.17e+00i	7.56e-01	1.78e+00
-1.35e+00 - 1.17e+00i	7.56e-01	1.78e+00
-1.06e+01	1.00e+00	1.06e+01
-2.24e+01	1.00e+00	2.24e+01

Table 6.2: Closed Loop Poles - LQ Servo Design 2

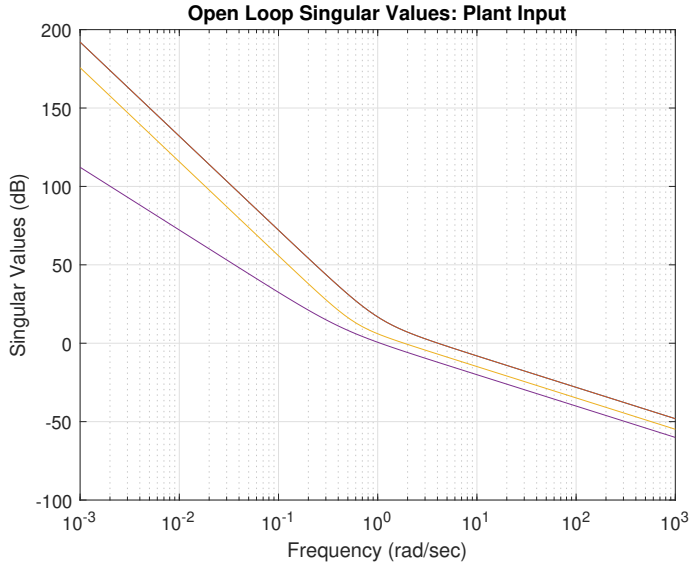


Figure 6.17: Quadrotor Open Loop Singular Values at Error - LQ Servo Design 2

Open Loop Frequency Response at Error. By breaking the loop at the error signal we get the open loop transfer function matrix L_e. As shown in Figure 6.17, at low frequencies we have a slope of at least -20dB/dec for all singular values due to an integral action in each control channel. The plot shows that reference commands r below 1.0 rad/s will be followed within 10% steady-state error. Also, output disturbances d_o below 1.0 rad/s will be attenuated by 20dB (0.1). In addition to sensor noise n with frequencies above 345 rad/s will be attenuated by 20dB (0.1).

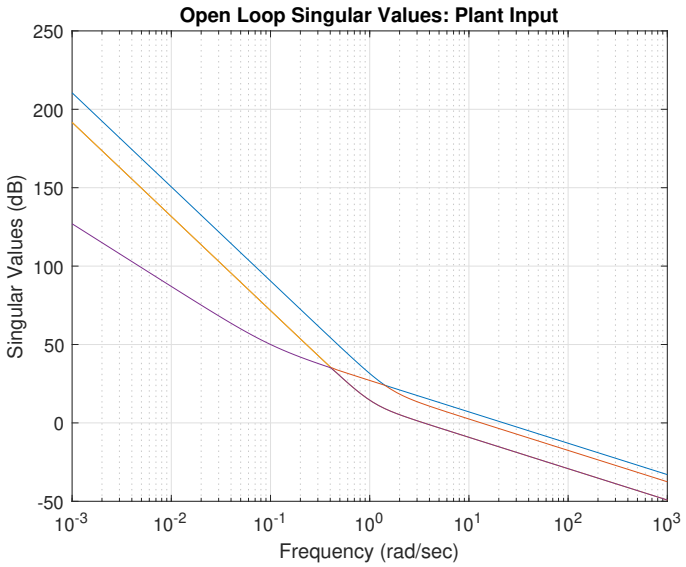


Figure 6.18: Quadrotor Open Loop Singular Values at Input - LQ Servo Design 2

Open Loop Frequency Response at Plant Input. By breaking the loop at the plant input, we get the open loop transfer function matrix L_u. As shown in Figure 6.18, the input disturbances d_i below 0.7 rad/s will be attenuated by 20dB. In other words, the input signal u will have a disturbance of 10% of the magnitude of d_i.

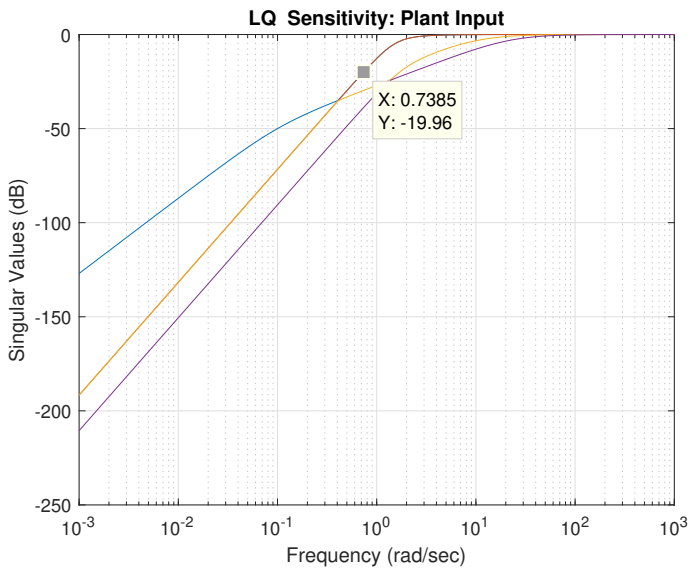


Figure 6.19: Quadrotor Sensitivity Frequency Response at Input - LQ Servo Design 2

Sensitivity Frequency Response at Input. The sensitivity transfer function S_u, which shows the impact of the input disturbance d_i on the input signal u. From the plot, we see that d_i will be attenuated by $20dB$ for frequencies below 0.7 rad/s.

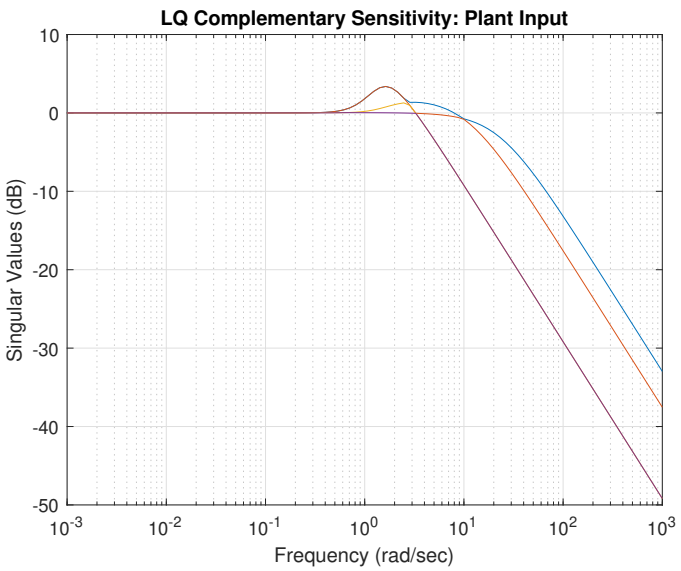


Figure 6.20: Quadrotor Complementary Sensitivity Frequency Response at Input - LQ Servo Design 2

LQ Servo Design: Time Domain Analysis. Step reference commands were applied for each output y to obtain the closed loop system step response.

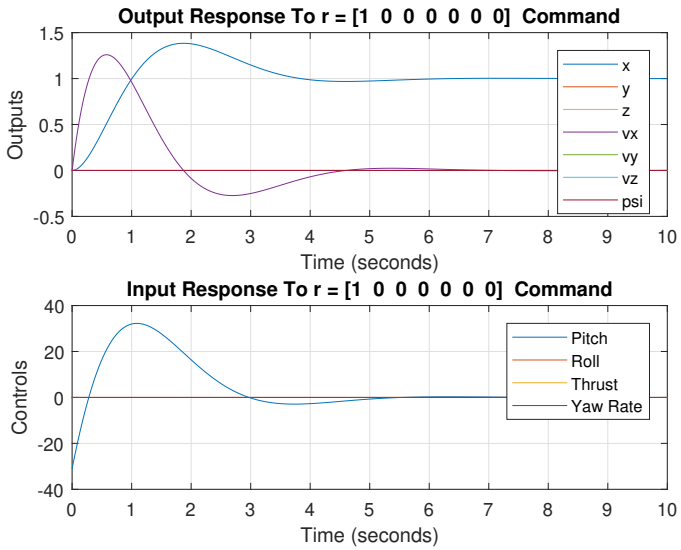


Figure 6.21: Step Response for Reference Command $r = [1\ 0\ 0\ 0\ 0\ 0\ 0]^t$ - LQ Servo Design 2

For Figure,

- The output response has an overshoot of 38%, rise time of $0.69s$, settling time of $5.3s$.
- The control response is not aggressive where the angle goes from approximately $-30°$ to $30°$.

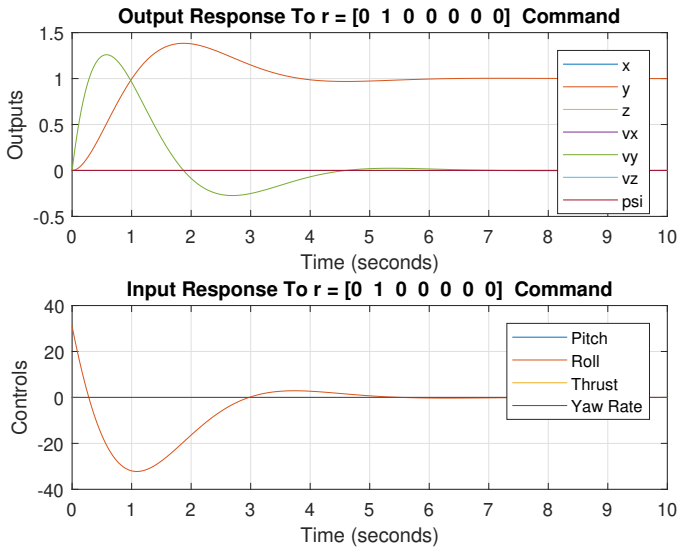


Figure 6.22: Step Response for Reference Command $r = [0\ 1\ 0\ 0\ 0\ 0\ 0]^t$ - LQ Servo Design 2

For Figure,

- The output response has an overshoot of 38%, rise time of $0.69s$, settling time of $5.3s$.
- The control response is not aggressive where the angle goes from approximately $-30°$ to $30°$.

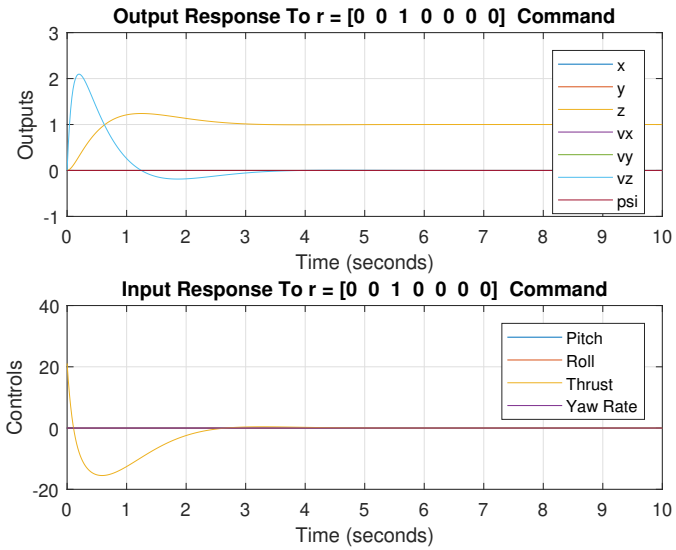


Figure 6.23: Step Response for Reference Command $r = [0\ 0\ 1\ 0\ 0\ 0\ 0]^t$ - LQ Servo Design 2

For Figure,

- The output response has an overshoot of 23%, rise time of $0.44s$, settling time of $2.88s$.
- The control response is too aggressive where thrust reaches a magnitude of $20N$ from the equilibrium.

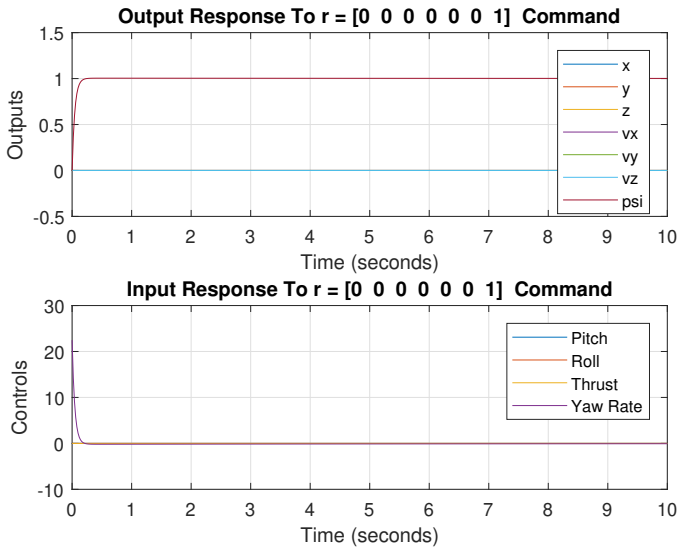


Figure 6.24: Step Response for Reference Command $r = [0\ 0\ 0\ 0\ 0\ 0\ 1]^t$ - LQ Servo Design 2

For Figure,

- The output response has an overshoot of 0.42%, rise time of $0.09s$, settling time of $0.16s$.
- The control response no too aggressive where the yaw angular rate goes from approximately $20°/s$ to $-20°/s$.

6.4 Weighted $\mathcal{H}^\infty$ Sensitivity Optimization

From [30, 36], we want to design a K controller for the plant $P = [A_p, B_p, C_p, D_p]$ such that the closed-loop systems exhibit the following: (1) stable closed loop system, (2) good low frequency reference command following, (3) good low frequency disturbance attenuation, (4) good high frequency noise attenuation/rejection, (5) robustness with high frequency unmodeled dynamics. Therefore, the Weighted $\mathcal{H}^\infty$ control how large the real "peaks" are by using stable weighting functions to shape the closed loop functions that we care about. The following are the closed loop functions that we care about:

$$T_{re} = S = \frac{1}{1+PK}, \quad T_{ru} = KS = \frac{K}{1+PK}, \quad T_{ry} = T = \frac{PK}{1+PK} \tag{6.55}$$

where the problem is to find a stabilizing K such that

$$K = \arg\left\{ \min_{K \ stabilizing} \gamma \ \left| \left\| \begin{bmatrix} W_1 S \\ W_2 KS \\ W_3 T \end{bmatrix} \right\|_{\mathcal{H}^\infty} < \gamma \right. \right\} \tag{6.56}$$

where W_1, W_2, and W_3 are the weighting matrices and γ is a parameter to be minimized. In general, W_1 needs to be large at low-frequencies and small at high-frequencies such that when we take W_1^{-1}, it shapes the desired S that we want to achieve. On the other hand, W_2 often selected to be a large constant so that we are lowering the bandwidth of the system, i.e., low control action. Finally, W_3 generally chosen to be small at low-frequencies and large at high-frequency so that T_{ry} looks large at low-frequencies and small at high-frequencies which what we desire.

Weighted Transfer Functions: Design 1. The weights were selected by following the guides in [30].

- For the sensitivity weighting, $\omega_{b_1}, \omega_{b_2}, \omega_{b_3}$ was selected to be a decade below

input dynamics (15 rad/s) which equals to 0.15. ω_{b_4} was chosen to be 15 rad/s in order to have a faster response in yaw.

- For the control sensitivity weighting, initially, constant values were selected M_u. By decreasing M_u, more control action will be applied.

- For the complementary sensitivity, $\omega_{bc_1}, \omega_{bc_2}, \omega_{bc_3}$ were selected initially to to be 15 rad/s, a decade above the desired closed loop bandwidth of 1.5 rad/s. Then increased to have more closed loop bandwidth. Similarly, ω_{bc_4} was selected to be 150 rad/s, one decade above the desired closed loop bandwidth of 153 rad/s.

$$
\begin{aligned}
W_1 &= \begin{bmatrix} \frac{s/M_{s_1}+\omega_{b_1}}{s+\epsilon\omega_{b_1}} & 0 & 0 & 0 \\ 0 & \frac{s/M_{s_2}+\omega_{b_2}}{s+\epsilon\omega_{b_2}} & 0 & 0 \\ 0 & 0 & \frac{s/M_{s_3}+\omega_{b_3}}{s+\epsilon\omega_{b_3}} & 0 \\ 0 & 0 & 0 & \frac{s/M_{s_4}+\omega_{b_4}}{s+\epsilon\omega_{b_4}} \end{bmatrix} \\
W_2 &= \begin{bmatrix} M_{u_1} & 0 & 0 & 0 \\ 0 & M_{u_2} & 0 & 0 \\ 0 & 0 & M_{u_3} & 0 \\ 0 & 0 & 0 & M_{u_4} \end{bmatrix} \\
W_3 &= \begin{bmatrix} \frac{s+\omega_{bc_1}/M_{y_1}}{\epsilon s+\omega_{bc_1}} & 0 & 0 & 0 \\ 0 & \frac{s+\omega_{bc_2}/M_{y_2}}{\epsilon s+\omega_{bc_2}} & 0 & 0 \\ 0 & 0 & \frac{s+\omega_{bc_3}/M_{y_3}}{\epsilon s+\omega_{bc_3}} & 0 \\ 0 & 0 & 0 & \frac{s+\omega_{bc_4}/M_{y_4}}{\epsilon s+\omega_{bc_4}} \end{bmatrix}
\end{aligned}
\tag{6.57}
$$

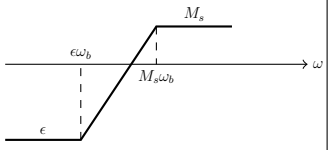
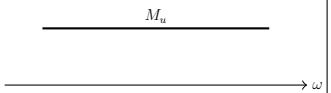
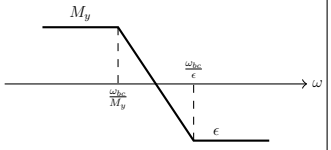
Weight (W)	Figure (Approx. W^{-1})	Parameters of M and ω
$W_1(s) = \frac{s/M_s+\omega_b}{s+\epsilon\omega_b}$		$M_{s_1} = 10,\ M_{s_2} = 10,$ $M_{s_3} = 10,\ M_{s_4} = 10$ $\omega_{b_1} = 0.15,\ \omega_{b_2} = 0.15,$ $\omega_{b_3} = 0.15,\ \omega_{b_4} = 15$
$W_2(s) = M_u$		$M_{u_1} = 10,\ M_{u_2} = 10,$ $M_{u_3} = 0.01,\ M_{u_4} = 10$
$W_3(s) = \frac{s+\omega_{bc}/M_y}{\epsilon s+\omega_{bc}}$		$M_{y_1} = 10,\ M_{y_2} = 10,$ $M_{y_3} = 10,\ M_{y_4} = 10$ $\omega_{bc_1} = 30,\ \omega_{bc_2} = 30,$ $\omega_{bc_3} = 30,\ \omega_{bc_4} = 150$

Table 6.3: Weighting Transfer Functions and Parameters - $\mathcal{H}^\infty$ Design 1

where $\epsilon = 0.001$ and the closed loop poles are as follows:

Poles	Damping	Frequency (rad/s)
-1.00e+00 + 1.00e-06i	1.00e+00	1.00e+00
-1.00e+00 - 1.00e-06i	1.00e+00	1.00e+00
-1.41e+00	1.00e+00	1.41e+00
-1.41e+00	1.00e+00	1.41e+00
-1.98e+00	1.00e+00	1.98e+00
-2.00e+00 + 2.31e-03i	1.00e+00	2.00e+00
-2.00e+00 - 2.31e-03i	1.00e+00	2.00e+00
-2.00e+00	1.00e+00	2.00e+00
-2.00e+00 + 1.04e-03i	1.00e+00	2.00e+00
-2.00e+00 - 1.04e-03i	1.00e+00	2.00e+00
-2.02e+00	1.00e+00	2.02e+00
-2.10e+00	1.00e+00	2.10e+00
-2.00e+00 + 7.05e-01i	9.43e-01	2.12e+00
-2.00e+00 - 7.05e-01i	9.43e-01	2.12e+00
-5.09e+00 + 1.95e+00i	9.34e-01	5.45e+00
-5.09e+00 - 1.95e+00i	9.34e-01	5.45e+00
-3.00e+04	1.00e+00	3.00e+04
-3.00e+04	1.00e+00	3.00e+04
-3.00e+04	1.00e+00	3.00e+04
-1.50e+05	1.00e+00	1.50e+05
-1.08e+07	1.00e+00	1.08e+07
-1.08e+07	1.00e+00	1.08e+07

Table 6.4: Closed Loop Poles for T_o - $\mathcal{H}^\infty$ Design 1

Weighted Transfer Functions: Design 2. For the second design, the same pro-

cedures applied in the first design were used here except for the control sensitivity weighting W_2. Because the first design does not 'request' a roll-off for the controller, a first-order transfer function matrix was designed.

- $\omega_{bu_1}, \omega_{bu_2}, \omega_{bu_3}, \omega_{bu_4}$ were selected to be approximately one decades above the control input dynamics.

- $M_{u_1}, M_{u_2}, M_{u_4}$ were selected to be 0.001 to have a decent control action while M_{u_3} was selected to be larger to have more control action in thrust control T.

$$
\begin{aligned}
W_1 &= \begin{bmatrix} \frac{s/M_{s_1}+\omega_{b_1}}{s+\epsilon\omega_{b_1}} & 0 & 0 & 0 \\ 0 & \frac{s/M_{s_2}+\omega_{b_2}}{s+\epsilon\omega_{b_2}} & 0 & 0 \\ 0 & 0 & \frac{s/M_{s_3}+\omega_{b_3}}{s+\epsilon\omega_{b_3}} & 0 \\ 0 & 0 & 0 & \frac{s/M_{s_4}+\omega_{b_4}}{s+\epsilon\omega_{b_4}} \end{bmatrix} \\
W_2 &= \begin{bmatrix} \frac{s+\omega_{bu_1}M_{u_1}}{\epsilon s+\omega_{bu_1}} & 0 & 0 & 0 \\ 0 & \frac{s+\omega_{bu_2}M_{u_2}}{\epsilon s+\omega_{bu_2}} & 0 & 0 \\ 0 & 0 & \frac{s+\omega_{bu_3}M_{u_3}}{\epsilon s+\omega_{bu_3}} & 0 \\ 0 & 0 & 0 & \frac{s+\omega_{bu_4}M_{u_4}}{\epsilon s+\omega_{bu_4}} \end{bmatrix} \\
W_3 &= \begin{bmatrix} \frac{s+\omega_{bc_1}/M_{y_1}}{\epsilon s+\omega_{bc_1}} & 0 & 0 & 0 \\ 0 & \frac{s+\omega_{bc_2}/M_{y_2}}{\epsilon s+\omega_{bc_2}} & 0 & 0 \\ 0 & 0 & \frac{s+\omega_{bc_3}/M_{y_3}}{\epsilon s+\omega_{bc_3}} & 0 \\ 0 & 0 & 0 & \frac{s+\omega_{bc_4}/M_{y_4}}{\epsilon s+\omega_{bc_4}} \end{bmatrix}
\end{aligned}
\tag{6.58}
$$

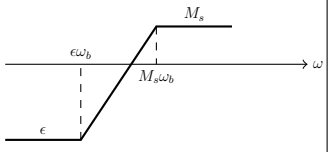
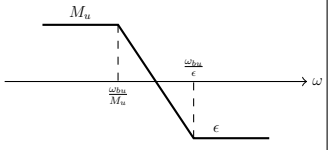
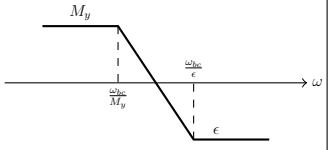
Weight (W)	Figure (Approx. W^{-1})	Parameters of M and ω
$W_1(s) = \frac{s/M_s+\omega_b}{s+\epsilon\omega_b}$		$M_{s_1} = 10,\ M_{s_2} = 10,$ $M_{s_3} = 10,\ M_{s_4} = 10$ $\omega_{b_1} = 0.15,\ \omega_{b_2} = 0.15,$ $\omega_{b_3} = 0.15,\ \omega_{b_4} = 15$
$W_2(s) = \frac{s+\omega_{bu}M_u}{\epsilon s+\omega_{bu}}$		$M_{u_1} = 0.001,\ M_{u_2} = 0.001,$ $M_{u_3} = 0.01,\ M_{u_4} = 0.001$ $\omega_{bu_1} = 150,\ \omega_{bu_2} = 150,$ $\omega_{bu_3} = 150,\ \omega_{bu_4} = 150$
$W_3(s) = \frac{s+\omega_{bc}/M_y}{\epsilon s+\omega_{bc}}$		$M_{y_1} = 10,\ M_{y_2} = 10,$ $M_{y_3} = 10,\ M_{y_4} = 10$ $\omega_{bc_1} = 30,\ \omega_{bc_2} = 30,$ $\omega_{bc_3} = 30,\ \omega_{bc_4} = 150$

Table 6.5: Weighting Transfer Functions and Parameters - $\mathcal{H}^\infty$ Design 2

where $\epsilon = 0.001$ and the closed loop poles are as follows:

Poles	Damping	Frequency (rad/s)
-1.66e+00 + 3.20e-01i	9.82e-01	1.69e+00
-1.66e+00 - 3.20e-01i	9.82e-01	1.69e+00
-1.66e+00 + 3.20e-01i	9.82e-01	1.69e+00
-1.66e+00 - 3.20e-01i	9.82e-01	1.69e+00
-1.74e+00 + 2.87e-01i	9.87e-01	1.77e+00
-1.74e+00 - 2.87e-01i	9.87e-01	1.77e+00
-2.00e+00	1.00e+00	2.00e+00
-1.89e+00 + 1.14e+00i	8.56e-01	2.21e+00
-1.89e+00 - 1.14e+00i	8.56e-01	2.21e+00
-1.89e+00 + 1.14e+00i	8.56e-01	2.21e+00
-1.89e+00 - 1.14e+00i	8.56e-01	2.21e+00
-2.55e+00	1.00e+00	2.55e+00
-2.55e+00	1.00e+00	2.55e+00
-2.79e+00	1.00e+00	2.79e+00
-2.99e+00 + 2.63e+00i	7.51e-01	3.98e+00
-2.99e+00 - 2.63e+00i	7.51e-01	3.98e+00
-4.24e+00	1.00e+00	4.24e+00
-1.24e+01 + 1.02e+01i	7.72e-01	1.60e+01
-1.24e+01 - 1.02e+01i	7.72e-01	1.60e+01
-6.17e+01	1.00e+00	6.17e+01
-5.12e+02	1.00e+00	5.12e+02
-5.12e+02	1.00e+00	5.12e+02
-3.00e+04	1.00e+00	3.00e+04
-3.00e+04	1.00e+00	3.00e+04
-3.00e+04	1.00e+00	3.00e+04
-1.50e+05	1.00e+00	1.50e+05

Table 6.6: Closed Loop Poles for T_o - $\mathcal{H}^\infty$ Design 2

Bilinear Transformation. A bilinear transformation was performed in order to move the poles at the origin to the right half plane (RHP) by $p_1 = -1$ and $p_2 = -10^{20}$. p_1 was chosen to be -1 which is approximately a decade below the attitude inner-loop bandwidth (15 rad/s). By increasing the magnitude of p_1, i.e., more negative, the bandwidth of the system increases and as a result, more control action is needed as well.

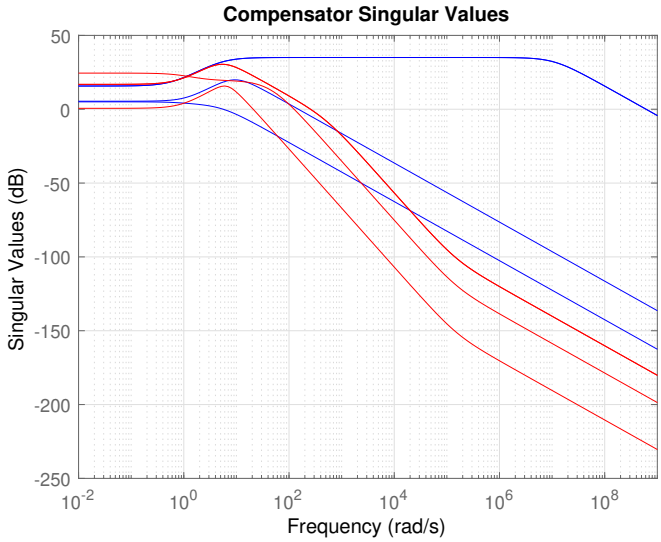


Figure 6.25: Compensator Singular Values: (Blue) for Design 1 (Red) for Design 2

Compensator Singular Values. The compensator singular values are constant at low frequencies (say 0.3 rad/s), and this is due to the bilinear transformation shifting the integral behavior 'requested' from W_1. The minimum singular value is $0.6dB$ for Design 2, which is very low compared to 5dB for Design 1. By performing an SVD for K for Design 1 at 0.01 rad/s, we see that singular values σ_3 and σ_4 are close to

each other and associated with $T \rightarrow z$ and $r \rightarrow \psi$, respectively.

$$
\begin{aligned}
U &= \begin{bmatrix} -0.9999 & 0.0120 & 0.0000 & 0.0000 \\ -0.0120 & -0.9999 & 0.0000 & 0.0000 \\ 0.0000 & 0.0000 & 1.0000 & 0.0000 \\ 0.0000 & 0.0000 & 0.0000 & -1.0000 \end{bmatrix} \\
S &= \begin{bmatrix} 6.2056 & 0 & 0 & 0 \\ 0 & 6.2056 & 0 & 0 \\ 0 & 0 & 1.8791 & 0 \\ 0 & 0 & 0 & 1.7741 \end{bmatrix} \\
V &= \begin{bmatrix} 0.9999 & -0.0120 & 0.0000 & 0.0000 \\ -0.0120 & -0.9999 & 0.0000 & 0.0000 \\ 0.0000 & 0.0000 & 1.0000 & 0.0000 \\ 0.0000 & 0.0000 & 0.0000 & -1.0000 \end{bmatrix}
\end{aligned}
\tag{6.59}
$$

Similarly, for Design 2, the minimum singular value is associated with $T \rightarrow z$. That will result in a steady state error to reference commands for z and ψ for Design 1, and z in Design 2. In Design 1, the maximum singular value does not roll off at high frequencies, which in result does not attenuate high-frequency noise n in e. This is due to the absence of roll off 'request' in W_2 weighting transfer function matrix. This is not the case in Design 2, where the weighting transfer function matrix W_2 requested a roll off in the compensator singular values.

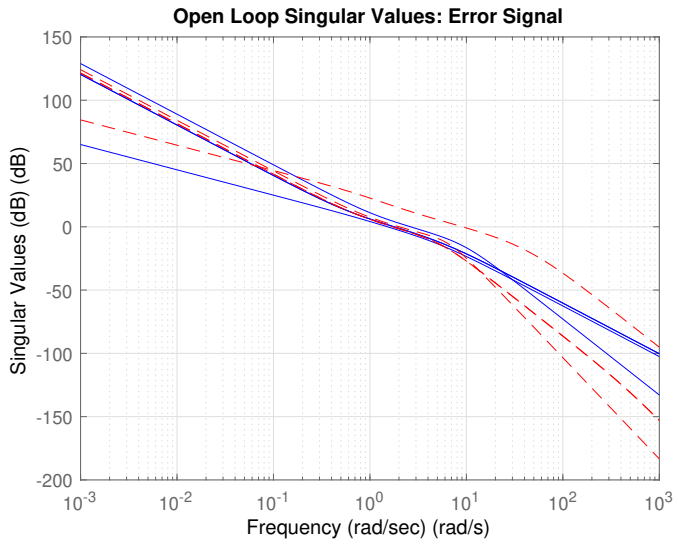


Figure 6.26: Open Loop Singular Values at Plant Output - $\mathcal{H}^\infty$ Design

Open Loop Singular Values at Plant Output. By breaking the loop at the plant output y or the error signal e, we obtain the transfer function matrix L_o. As shown in the plot, at low frequencies, reference commands r will be followed within a 10% steady state error below 0.17 rad/s. Similarly, output disturbances d_o will be attenuated by 20dB (0.1) below 0.17 rad/s. Also, sensor noise n with frequencies above 11.9 rad/s will be attenuated by 20dB as well. In Design 2, reference commands with frequency content below 0.3 rad/s will be followed within a 10% steady state error. Similarly, output disturbances d_o below 0.3 rad/s will be attenuated by 20dB. However, sensor noise n with frequencies above 40 rad/s will be attenuated by 20dB. Therefore, Design 2 provides better reference command following and output disturbance attenuation while Design 1 provides better sensor noise attenuation.

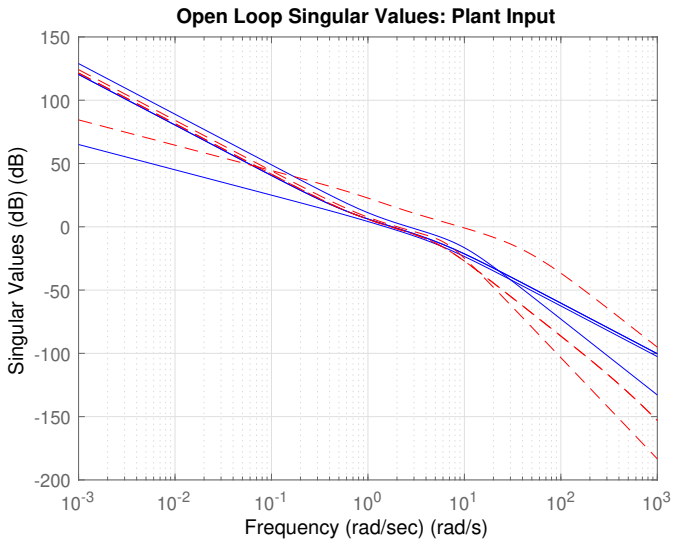


Figure 6.27: Open Loop Singular Values at Plant Input - $\mathcal{H}^\infty$ Design

Open Loop Singular Values at Plant Input. By breaking the loop at the plant input u we obtain the transfer function matrix L_i. The plot shows that input disturbances d_i will be attenuated by 20dB for frequencies below 0.17 rad/s. In Design 2, the input disturbances d_i below 0.3 rad/s will be attenuated by $20dB$.

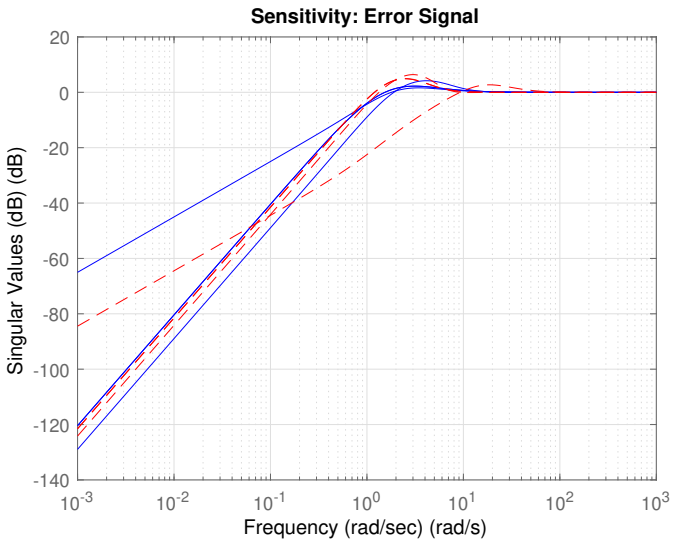


Figure 6.28: Sensitivity Singular Values at Plant Output - $\mathcal{H}^\infty$ Design

Sensitivity Singular Values at Plant Output. The plot shows that reference commands r will be followed within a 10% steady state error for frequencies below 0.17 rad/s. Similarly, output disturbances d_o will be attenuated by 20dB for frequencies below 0.17 rad/s. Also, the plot shows a peak of 4dB at 4 rad/s, which is acceptable. In Design 2, reference commands with frequency content below 0.3 rad/s will be followed within a 10% steady state error. Similarly, output disturbances d_o below 0.3 rad/s will be attenuated by 20dB. However, Design 2 has a trade-off between better reference commands following and output disturbance attenuation, and a higher bump in the sensitivity of $6dB$.

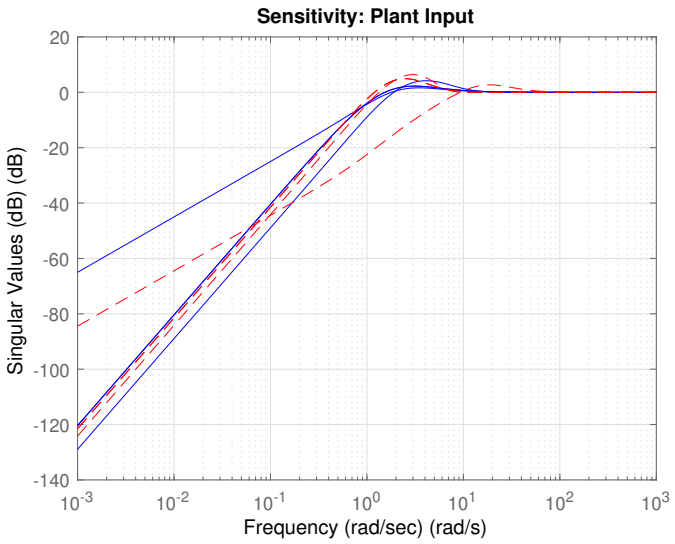


Figure 6.29: Sensitivity Singular Values at Plant Input - $\mathcal{H}^{\infty}$ Design

Sensitivity Singular Values at Plant Input. The plot shows that input disturbances d_i with frequencies below 0.17 rad/s will be attenuated by 20dB. In Design 2, input disturbances d_i with frequencies below 0.3 rad/s will be attenuated by 20dB. However, a peak of $6dB$ around 3 rad/s is present in Design 2, which is a tradeoff between bandwidth and peak in the sensitivity.

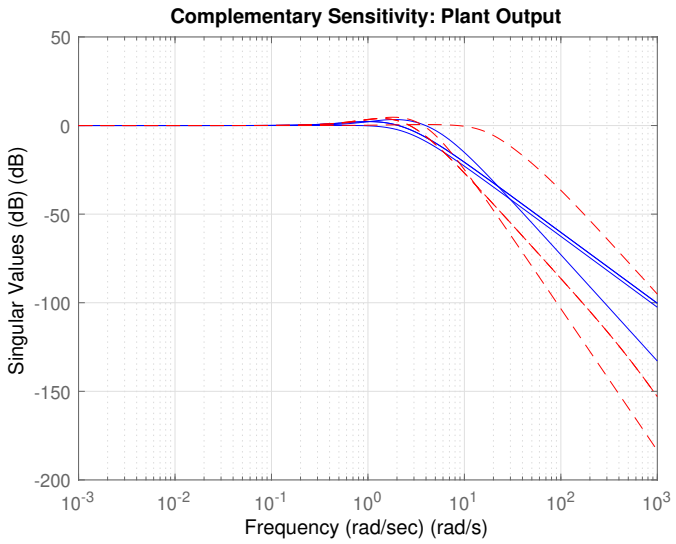


Figure 6.30: Complementary Sensitivity Singular Values at Plant Output - $\mathcal{H}^{\infty}$ Design

Complementary Sensitivity Singular Values at Plant Output. The plot shows that low-frequency reference commands will be followed. The S_o present a better interpretation for reference commands following. Design 1 crosses $-3dB$ point at 2.2 rad/s while Design 2 at 3.3 rad/s. In other words, Design 2 has more closed loop bandwidth. However, Design 2 has a larger bump of $4.6dB$ where Design 1 has $3.3dB$ at the same frequency of 1.8 rad/s/. Both bumps start to grow around 0.8 rad/s due to compensator zeros. A prefilter can be used to reduce excessive overshoot in y.

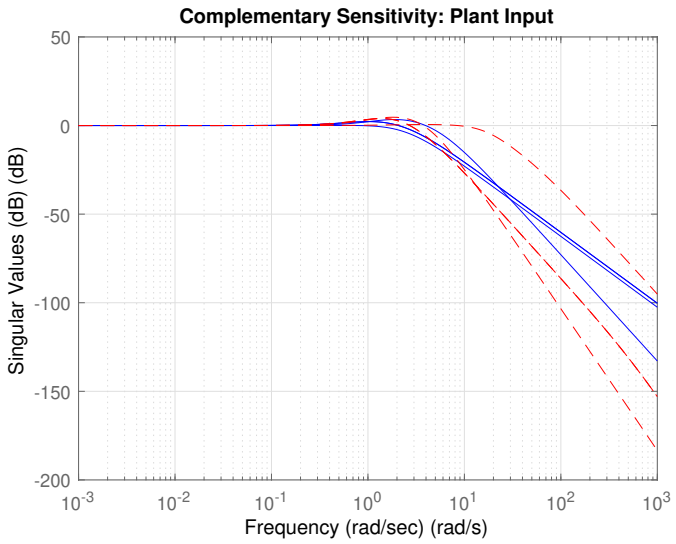


Figure 6.31: Complementary Sensitivity Singular Values at Plant Input - $\mathcal{H}^\infty$ Design

Complementary Sensitivity Singular Values at Plant Input. The plot shows that in both Design 1 and Design 2 input disturbances d_i for frequencies below 0.3 rad/s will not be attenuated in u, which is not desirable.

Step Response.

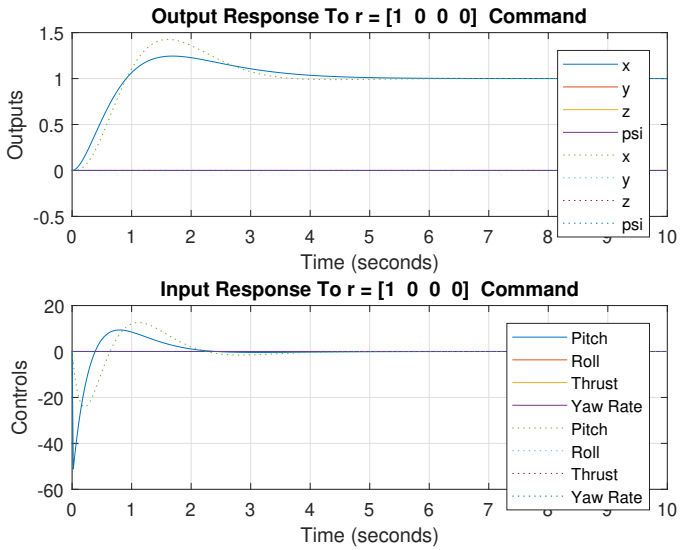


Figure 6.32: Step Response for Reference Command $r = [1\ 0\ 0\ 0]^t$: Design 1 (Solid) and Design 2 (Dotted) - $\mathcal{H}^\infty$ Design

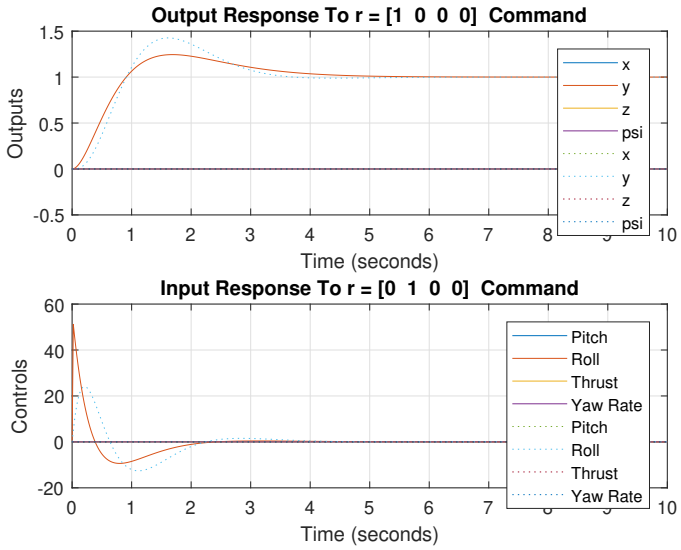


Figure 6.33: Step Response for Reference Command $r = [0\ 1\ 0\ 0]^t$: Design 1 (Solid) and Design 2 (Dotted) - $\mathcal{H}^\infty$ Design

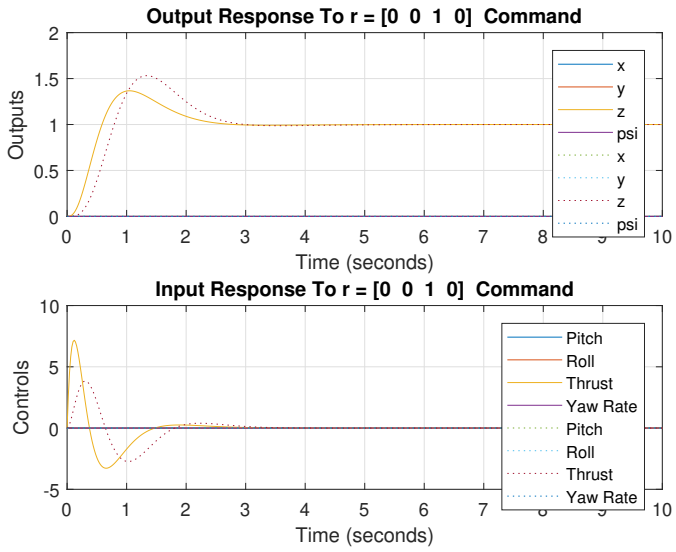


Figure 6.34: Step Response for Reference Command $r = [0\ 0\ 1\ 0]^t$: Design 1 (Solid) and Design 2 (Dotted) - $\mathcal{H}^\infty$ Design

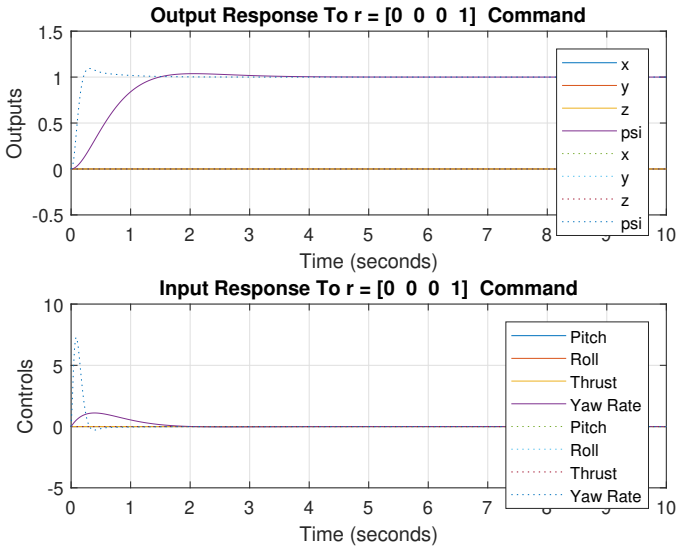


Figure 6.35: Step Response for Reference Command $r = [0\ 0\ 0\ 1]^t$: Design 1 (Solid) and Design 2 (Dotted) - $\mathcal{H}^\infty$ Design

6.5 LQG/LTRO Design for Quadrotor Translational Dynamics

In the beginning, we augment the plant P with integrators in each channel to form $P_d = [A, B, C]$ to get zero steady-state error to step reference commands. The singular values for the plant P and design plant P_d are shown below

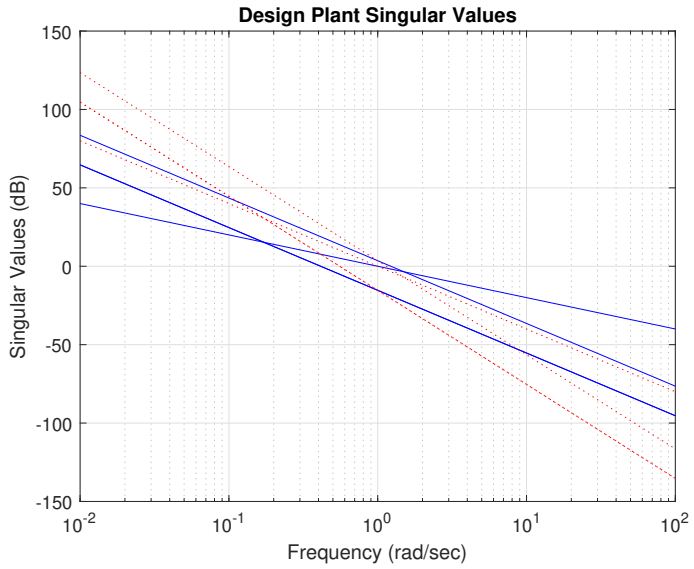


Figure 6.36: Plant P (Blue) and Design Plant P_d (Red) Singular Values - LQG/LTRO Design

where

$$A = \begin{bmatrix} 0_{4\times4} & 0_{4\times7} \\ B_p & A_p \end{bmatrix}, \; B = \begin{bmatrix} I_{4\times4} \\ 0_{7\times4} \end{bmatrix}, \; C = \begin{bmatrix} 0_{4\times4} & C_p \end{bmatrix} \tag{6.60}$$

and the state of the system P_d is $x = [x_i \; x_p]^T$ where x_i is the integrator state and x_p is the plant P state. From [30] we can design the target open loop transfer function matrix $L_o = G_{KF} = C(sI - A)^{-1}H$ by shaping G_{FOL} (assuming G_{FOL} is large at low frequencies). Given that, we have the following

$$L_o(jw) = G_{KF}(jw) \approx \frac{1}{\sqrt{\mu}} G_{FOL}(jw) \tag{6.61}$$

where L matrix is used to shape L_o, and μ is used for increase/decreasing loop bandwidth. The L matrix for Design 1 is as follows

$$\begin{aligned} L &= \begin{bmatrix} L_L \\ L_H \end{bmatrix} \\ L_L &= \left[C_p(j\omega_o I - A_p)^{-1}B_p\right]^{-1} \\ L_H &= C_p^T\left[C_pC_p^T\right]^{-1} \end{aligned} \tag{6.62}$$

where $\omega_o = 0.01 rad/s$. The matrix L_L match G_{FOL} at low frequencies while L_H matches G_{FOL} at high frequencies. That is, L matrix matches G_{FOL} at all frequencies. This will result in a G_{FOL} with a unity gain crossover frequency where we can increase/decrease the bandwidth of the closed-loop system by changing the unity gain crossover by a factor of $\frac{1}{\sqrt{\mu}}$. In this case, μ was chosen to be 0.05 to get a unity gain crossover frequency of approximately 1.5 rad/s (a decade below inner loop dynamics). In Design 2, L was chosen to be

$$L = B \tag{6.63}$$

Solving $\Theta = \mu I 4 \times 4$ using the Filter Algebraic Riccati Equation (FARE) yields the following

$$0 = A\Sigma + \Sigma A^T + LL^T - \Sigma C^T \Theta^{-1} C\Sigma \tag{6.64}$$

where H is given below

$$H = \Sigma C^T \Theta^{-1} \tag{6.65}$$

where target closed loop poles for $\lambda_i(A - HC)$ for Design 1 are given in Table:

Poles	Damping	Frequency (rad/s)
-1.00e-04	1.00e+00	1.00e-04
-1.00e-04	1.00e+00	1.00e-04
-1.58e+00	1.00e+00	1.58e+00
-1.31e-08	1.00e+00	1.31e-08
-3.92e-05	1.00e+00	3.92e-05
-1.36e-04	1.00e+00	1.36e-04
-1.00e-04 + 2.37e-09i	1.00e+00	1.00e-04
-1.00e-04 - 2.37e-09i	1.00e+00	1.00e-04
-1.58e+00	1.00e+00	1.58e+00
-1.58e+00	1.00e+00	1.58e+00
-1.58e+00	1.00e+00	1.58e+00

Table 6.7: Target Closed Loop Poles for $\lambda_i(A - HC)$ - LQG/LTRO Design

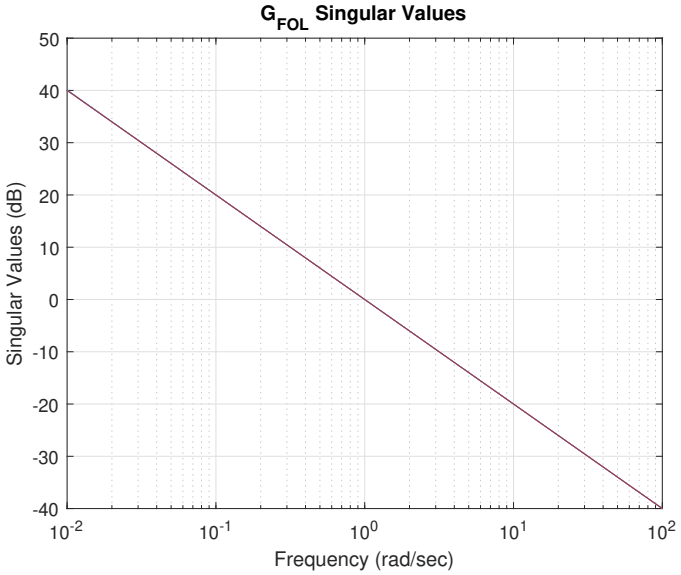


Figure 6.37: Quadrotor G_{FOL} Singular Values - LQG/LTRO Design

In the target open loop singular values plot, the gain crossover frequency is 1 rad/s as expected. Also, the loop shape looks like an integrator because L matrix was designed in such a way.

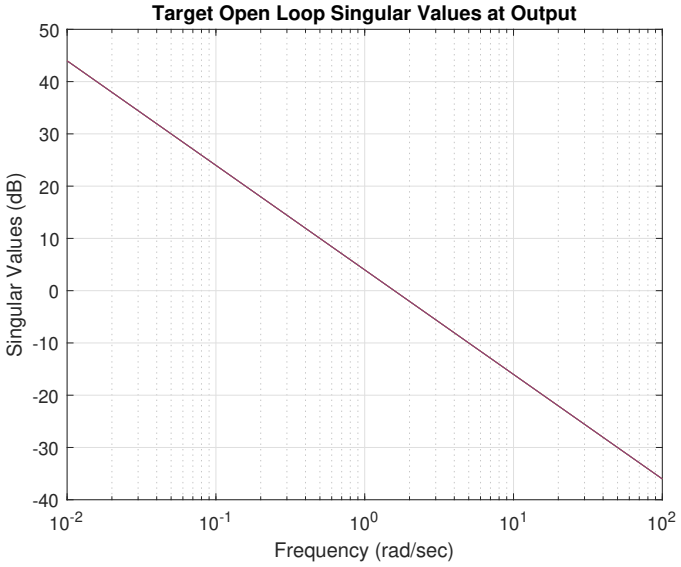


Figure 6.38: Quadrotor Target Loop G_{KF} Singular Values - LQG/LTRO Design

The target loop G_{KF} singular values plot shows a unity gain crossover at 1.5 rad/s as expected. Since μ was selected to be 0.4 rad/s and $\frac{1}{\sqrt{(0.4)}} \approx 1.5$.

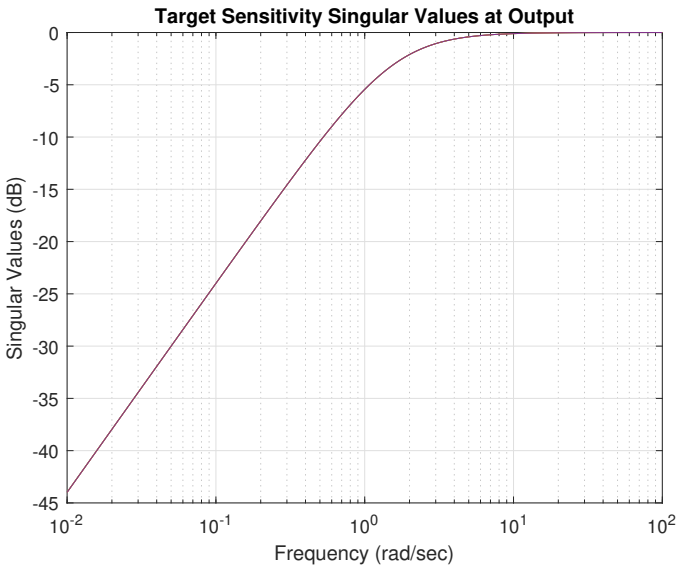


Figure 6.39: Quadrotor Target Sensitivity S_{KF} Singular Values - LQG/LTRO Design

The target sensitivity follows reference commands r below 0.14 rad/s within 10% steady state error. Similarly, output disturbances d_o below 0.14 rad/s are going to be attenuated by $20dB$. Also, we can see that the sensitivity plot has no bump, which desirable.

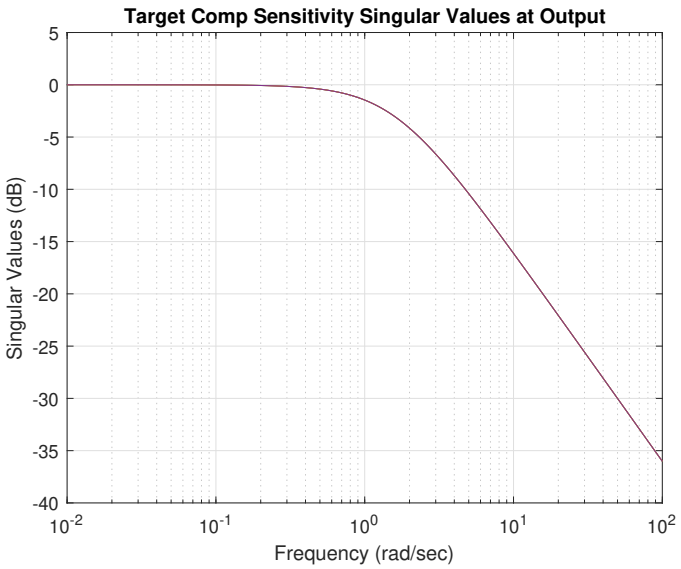


Figure 6.40: Quadrotor Target Complementary Sensitivity T_{KF} Singular Values - LQG/LTRO Design

In the target complementary sensitivity plot, step reference commands r at low frequencies will be followed. The sensitivity plot shows that more clearly. Sensor noise n for frequencies above 15 rad/s will be attenuated by $20dB$. Also, the plot has no bump ($\zeta = 1$), which means there is no overshoot in the output response to a step reference command.

Loop Transfer Recovery at the Output. From [30], we can do Loop Transfer Recovery at the Output (LTRO), by designing a cheap control (small ρ) that will recover the target open loop transfer function matrix $L_o = G_{KF}$ as follows

$$Q = diag(0.1, 0.1, 0.1, 1, 100, 100, 100, 10, 10, 10, 10), \quad R = \rho I_{4\times 4}, \quad \rho = 1 \times 10^{-8} \quad (6.66)$$

where closed loop poles for the system is in Table

Poles	Damping	Frequency (rad/s)
-3.16e+04	1.00e+00	3.16e+04
-1.00e+04	1.00e+00	1.00e+04
-1.00e+04	1.00e+00	1.00e+04
-1.00e+04	1.00e+00	1.00e+04
-1.47e+01	1.00e+00	1.47e+01
-3.24e+00	1.00e+00	3.24e+00
-3.16e+00	1.00e+00	3.16e+00
-1.85e+00 + 1.41e+00i	7.97e-01	2.33e+00
-1.85e+00 - 1.41e+00i	7.97e-01	2.33e+00
-1.58e+00	1.00e+00	1.58e+00
-1.85e+00 + 1.41e+00i	7.97e-01	2.33e+00
-1.85e+00 - 1.41e+00i	7.97e-01	2.33e+00
-1.58e+00	1.00e+00	1.58e+00
-1.58e+00	1.00e+00	1.58e+00
-1.58e+00	1.00e+00	1.58e+00
-4.46e-04	1.00e+00	4.46e-04
2.70e-04	-1.00e+00	2.70e-04
-1.02e-04	1.00e+00	1.02e-04
-9.79e-05	1.00e+00	9.79e-05
-1.31e-08	1.00e+00	1.31e-08
-1.00e-04 + 1.90e-05i	9.82e-01	1.02e-04
-1.00e-04 - 1.90e-05i	9.82e-01	1.02e-04

Table 6.8: Closed Loop Poles for $\lambda_i(A - BG - HC)$ - LQG/LTRO Design

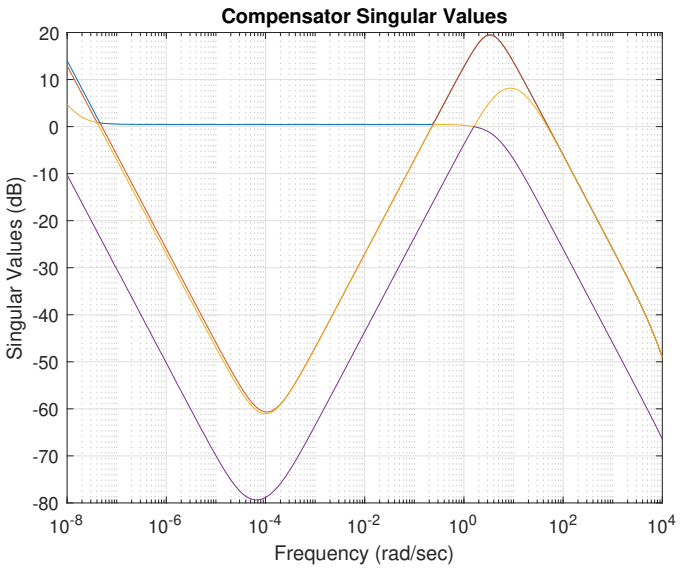


Figure 6.41: Compensator Singular Values - LQG/LTRO Design

In the compensator, singular values plot shows very low magnitudes at low frequencies which will lead to large steady-state error in x, y, and z by doing an SVD at $\omega = 0.0001$ rad/s. The design starts to have an integral action of $-20dB/dec$ slope below 0.0001 rad/s. At high frequencies, the compensator rolls off to attenuate high-frequency signals.

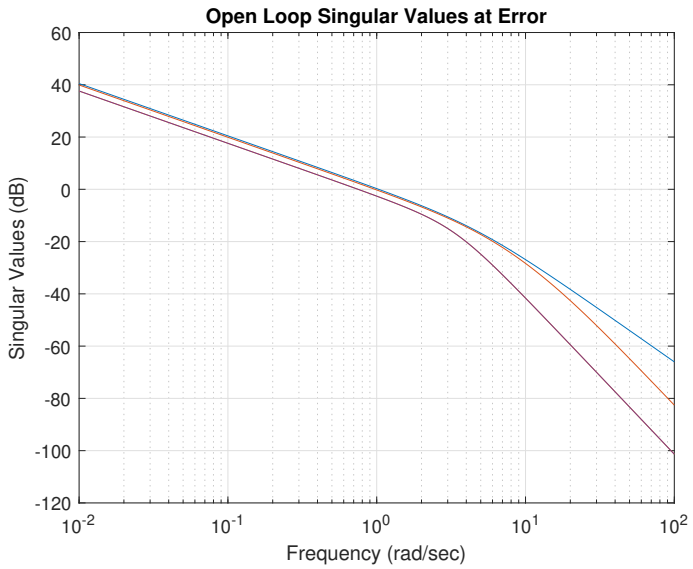


Figure 6.42: Open Loop Singular Values at Error - LQG/LTRO Design

Open Loop Singular Values at Plant Output. As shown in the plot, at low frequencies reference commands r will be followed within a 10% steady state error below 0.0.1 rad/s. Similarly, output disturbances d_o will be attenuated by 20dB (0.1) below 0.1 rad/s. Also, sensor noise n with frequencies above 7 rad/s will be attenuated by 20dB as well.

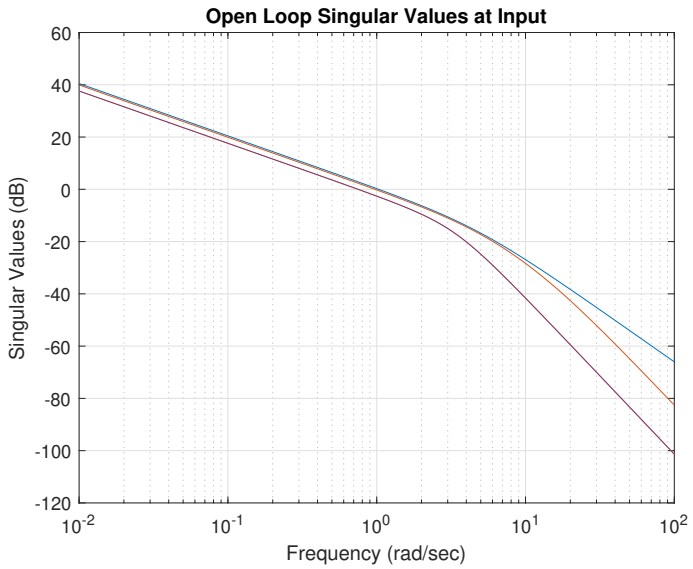


Figure 6.43: Open Loop Singular Values at Input - LQG/LTRO Design

Open Loop Singular Values at Plant Input. The plot shows that input disturbances d_i will be attenuated by 20dB for frequencies below 0.1 rad/s.

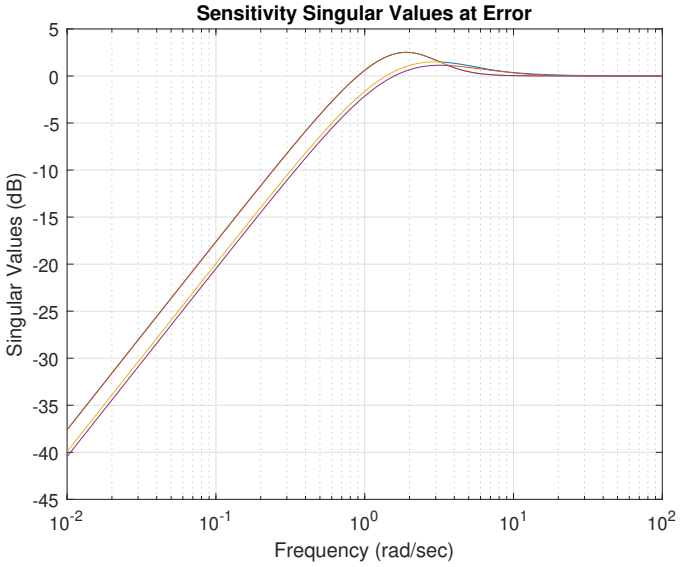


Figure 6.44: Sensitivity Singular Values at Error - LQG/LTRO Design

Sensitivity Singular Values at Plant Output. The plot shows that reference

commands r will be followed within a 10% steady state error for frequencies below 0.03 rad/s. Similarly, output disturbances d_o will be attenuated by 20dB for frequencies below 0.03 rad/s.

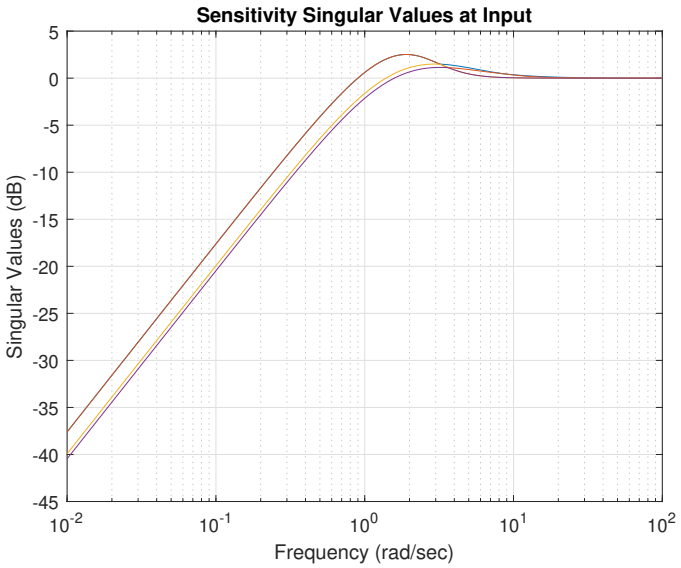


Figure 6.45: Sensitivity Singular Values at Input - LQG/LTRO Design

Sensitivity Singular Values at Plant Input. The plot shows that input disturbances d_i with frequencies below 0.03 rad/s will be attenuated by 20dB.

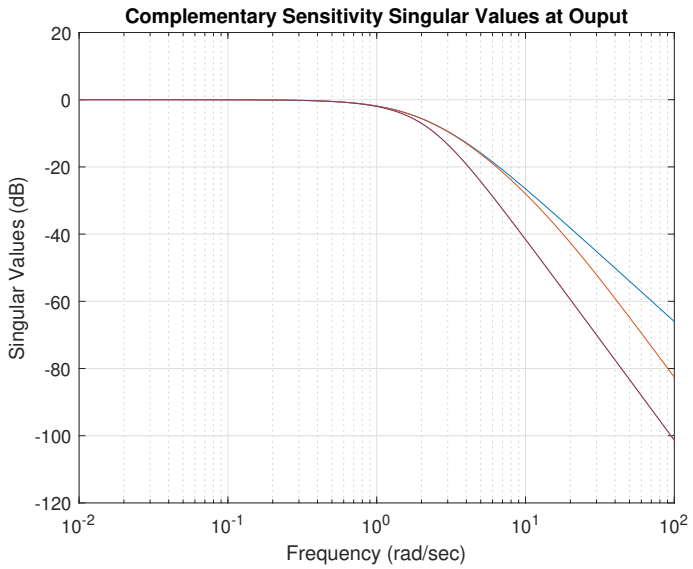


Figure 6.46: Complementary Sensitivity Singular Values at Output - LQG/LTRO Design

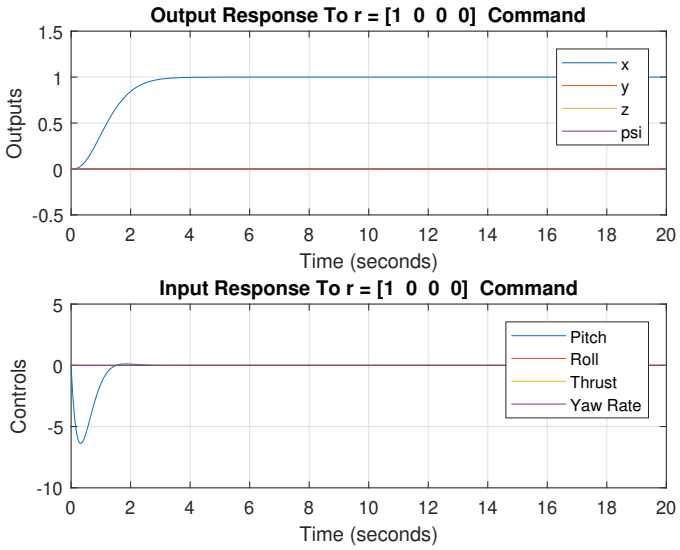


Figure 6.47: Step Response for Reference Command $r = [1\ 0\ 0\ 0]^T$ - LQG/LTRO Design

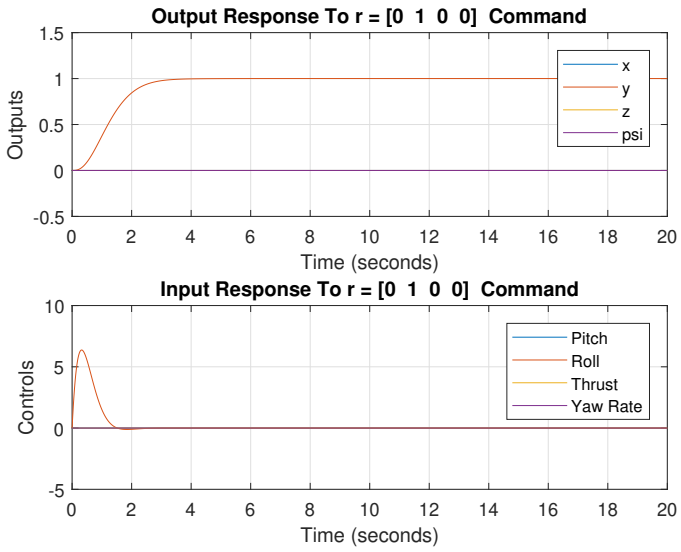


Figure 6.48: Step Response for Reference Command $r = [0\ 1\ 0\ 0]^T$ - LQG/LTRO Design

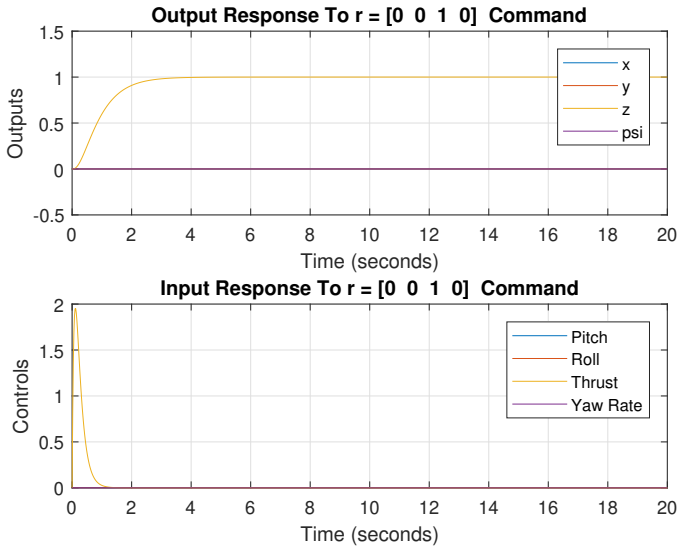


Figure 6.49: Step Response for Reference Command $r = [0\ 0\ 1\ 0]^T$ - LQG/LTRO Design

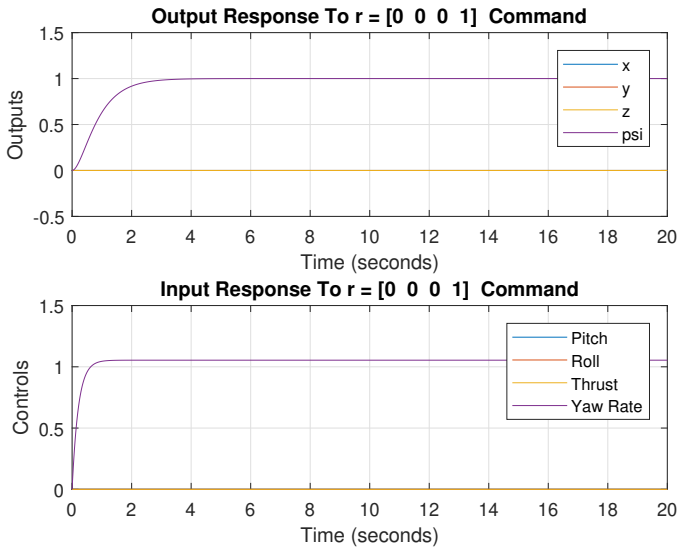


Figure 6.50: Step Response for Reference Command $r = [0\ 0\ 0\ 1]^T$ - LQG/LTRO Design

6.6 Quadrotor Hardware Demonstrations

All hardware demonstrations are done using the LQR controller Design 2. The 3D motion data of positions, velocities, and accelerations are obtained from either HTC Vive Tracking System or OptiTrack Motion Capture System at a rate of 100Hz. Then, the data are processed and filtered on the ground station as well. Then, the ground station computes the commanded roll ϕ, pitch θ, thrust T, and yaw rate r using the nonlinear mapping. The commands are sent through Xbee modules wirelessly from the ground station to the quadrotor at a rate of 100Hz. The following outlines the hardware demonstrations:

1. **Position Control at a Setpoint.** In this demonstration, a step response of 1 meter in the x-axis is applied while keeping the reference z at 1 meter and the rest to zero. The OptiTrack mocap is used to demonstrate the following

experiment. The reference commands are as follow:

$$x^r = 1 \qquad y^r = 0 \qquad z^r = 1$$

$$\dot{x}^r = 0 \qquad \dot{y}^r = 0 \qquad \dot{z}^r = 0$$

$$\ddot{x}^r = 0 \qquad \ddot{y}^r = 0 \qquad \ddot{z}^r = 0$$

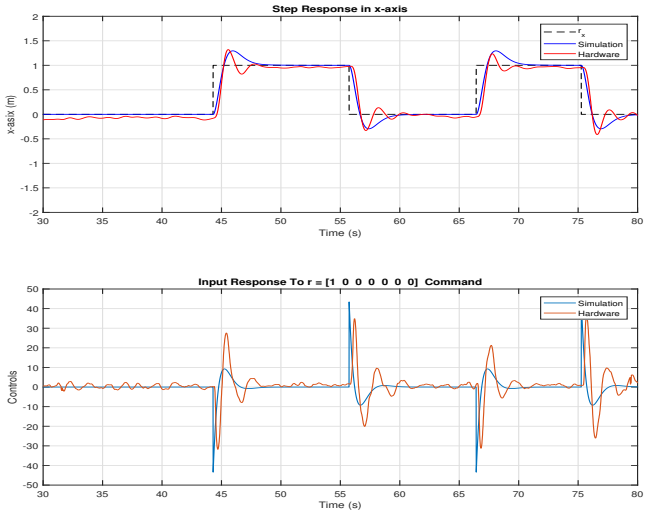


Figure 6.51: Hardware Demonstration for LQ Servo Design: x-axis Step Response for Reference Command $r = [1\ 0\ 0\ 0\ 0\ 0\ 0]^t$ - LQ Servo Design 2

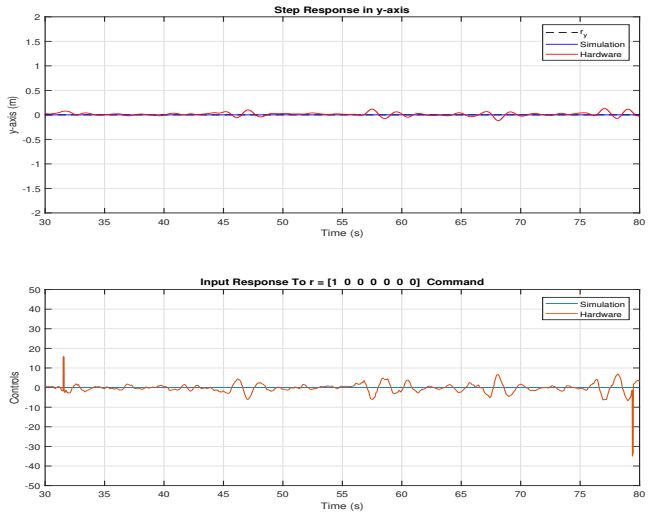


Figure 6.52: Hardware Demonstration for LQ Servo Design: y-axis Step Response for Reference Command $r = [1\ 0\ 0\ 0\ 0\ 0\ 0]^t$ - LQ Servo Design 2

2. **Path Following a Circular Trajectory with $\omega = 0.2$.** In this demonstration, a pre-defined circle trajectory is applied as a trajectory reference command to the quadrotor to follow. The HTC Vive Tracking System is used to demonstrate the following experiment. The velocity of the quadrotor $v = \sqrt{v_x^2 + v_y^2 + v_z^2} \approx 0.12$ m/s. The reference trajectory is as follow:

$$x^r(t) = 0.5sin(0.2t) \qquad y^r(t) = 0.5cos(0.2t) \qquad z^r(t) = 1.0$$

$$\dot{x}^r(t) = 0.5(0.2)cos(0.2t) \qquad \dot{y}^r(t) = -0.5(0.2)sin(0.2t) \qquad \dot{z}^r(t) = 0.0$$

$$\ddot{x}^r(t) = -0.5(0.2)^2sin(0.2t) \qquad \ddot{y}^r(t) = -0.5(0.2)^2cos(0.2t) \qquad \ddot{z}^r(t) = 0.0$$

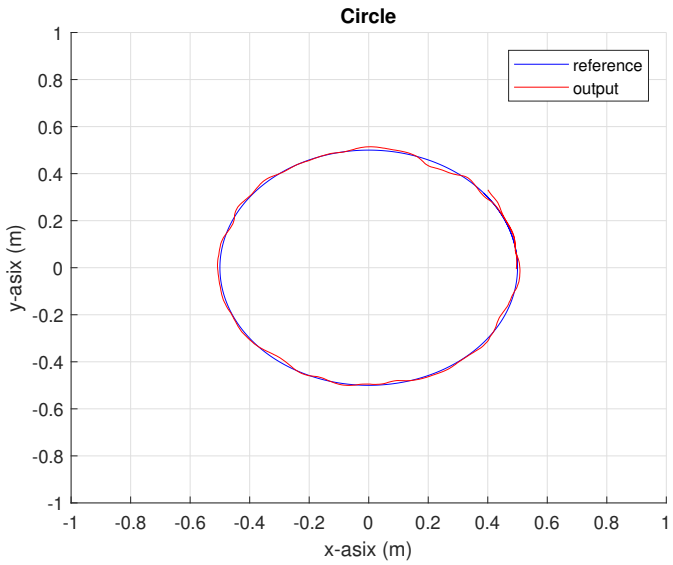


Figure 6.53: Circular Trajectory with $\omega = 0.2$ - LQ Servo Design 2

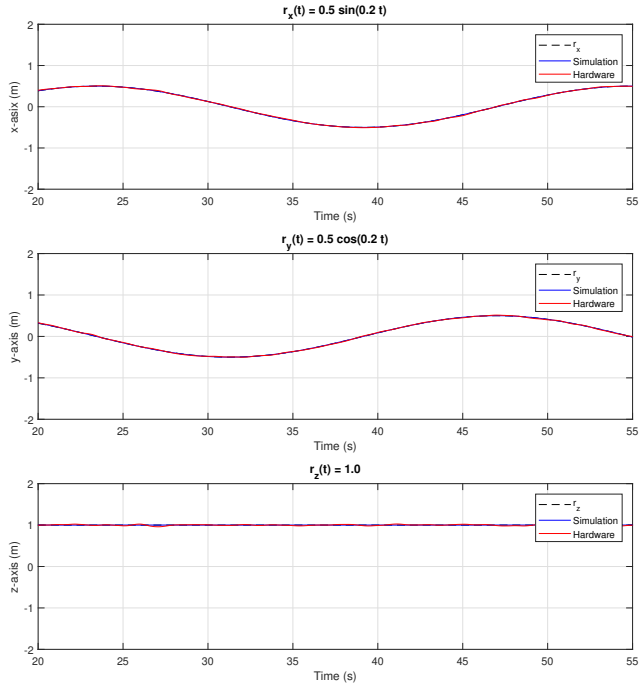


Figure 6.54: x, y, z Resulting from $r_x(t) = 0.5sin(0.2t)$, $r_y(t) = 0.5cos(0.2t)$, $r_z(t) = 1.0$ - LQ Servo Design 2

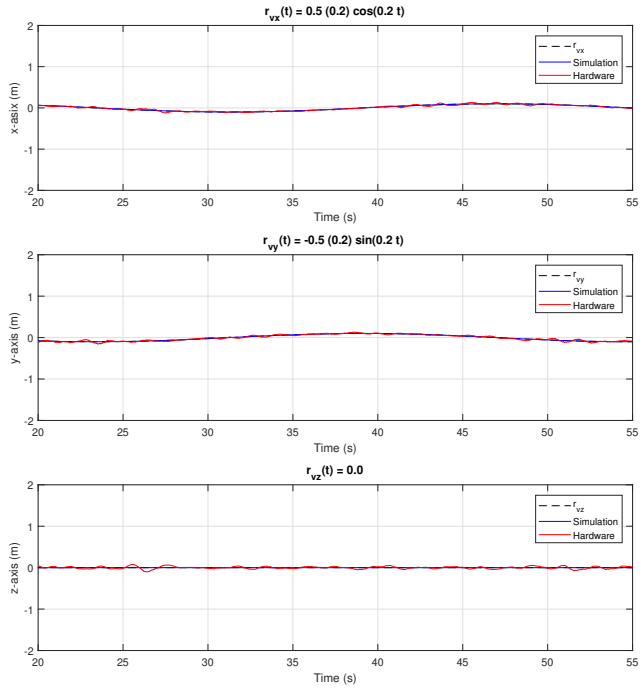


Figure 6.55: v_x, v_y, v_z Resulting from $r_{vx}(t) = 0.5(0.2)cos(0.2t)$, $r_{vy}(t) = -0.5(0.2)sin(0.2t)$, $r_{vz}(t) = 0.0$ - LQ Servo Design 2

3. **Path Following a Circular Trajectory with $\omega = 0.4$.** In this demonstration, a pre-defined circle trajectory is applied as a trajectory reference command to the quadrotor to follow. The OptiTrack mocap is used to demonstrate the following experiment. The velocity of the quadrotor $v = \sqrt{v_x^2 + v_y^2 + v_z^2} \approx 0.9$ m/s. The reference trajectory is as follow:

$$x^r(t) = 3.0sin(0.4t) \qquad y^r(t) = 3.0cos(0.4t) \qquad z^r(t) = 1.0$$

$$\dot{x}^r(t) = 3.0(0.4)cos(0.4t) \qquad \dot{y}^r(t) = -3.0(0.4)sin(0.4t) \qquad \dot{z}^r(t) = 0.0$$

$$\ddot{x}^r(t) = -3.0(0.4)^2sin(0.4t) \qquad \ddot{y}^r(t) = -3.0(0.4)^2cos(0.4t) \qquad \ddot{z}^r(t) = 0.0$$

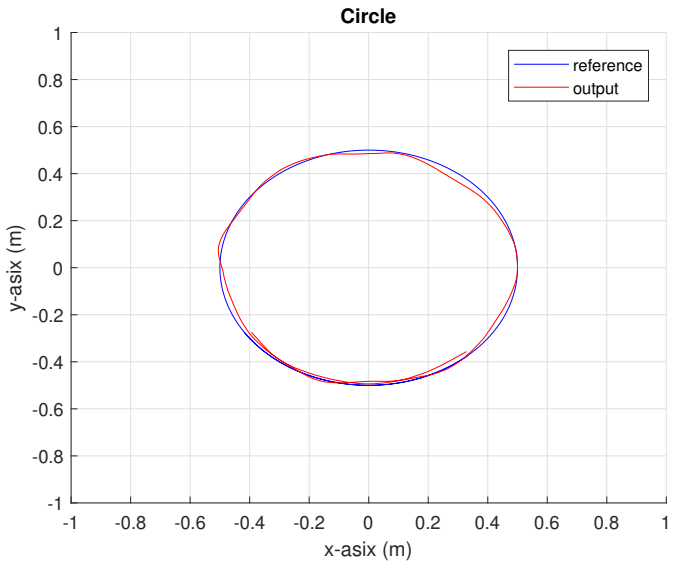


Figure 6.56: Circular Trajectory with $\omega = 0.4$ - LQ Servo Design 2

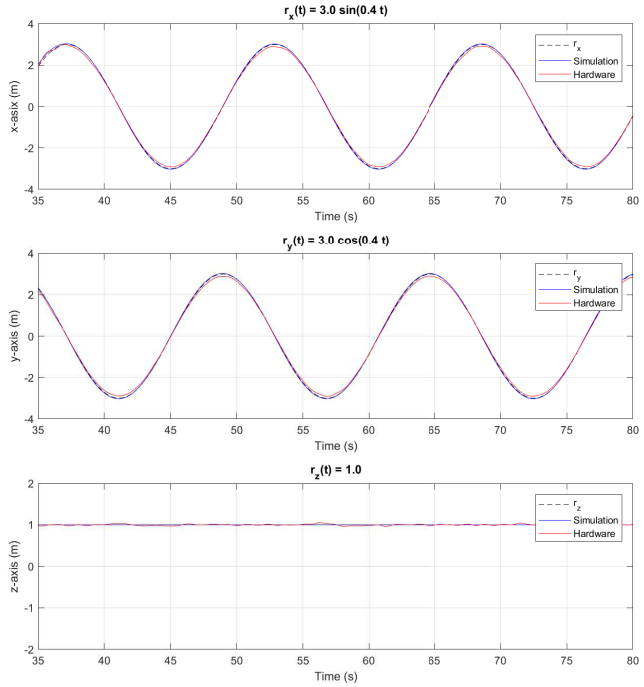


Figure 6.57: x, y, z Resulting from $r_x(t) = 3.0sin(0.4t)$, $r_y(t) = 3.0cos(0.4t)$, $r_z(t) = 1.0$ - LQ Servo Design 2

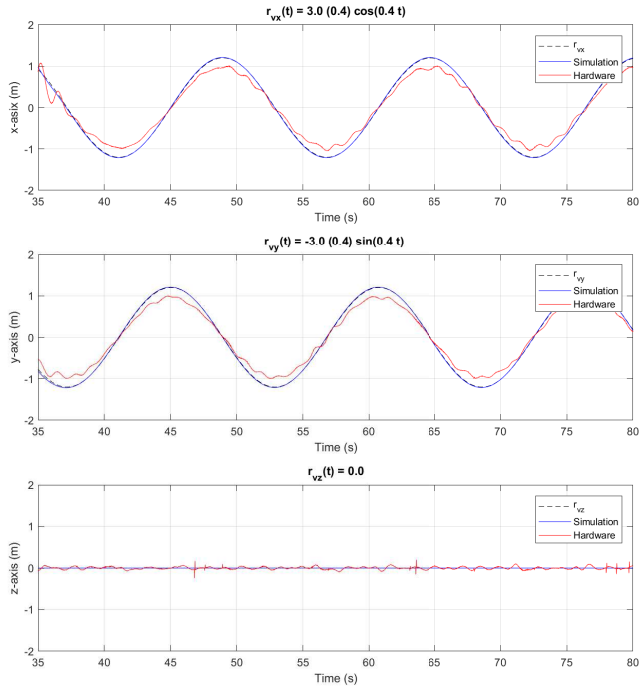


Figure 6.58: v_x, v_y, v_z Resulting from $r_{vx}(t) = 3.0(0.4)cos(0.4t)$, $r_{vy}(t) = -3.0(0.4)sin(0.4t)$, $r_{vz}(t) = 0.0$ - LQ Servo Design 2

4. **Path Following a Circular Trajectory with $\omega = 0.8$.** In this demonstration, a pre-defined circle trajectory is applied as a trajectory reference command to the quadrotor to follow. The OptiTrack mocap is used to demonstrate the following experiment. The velocity of the quadrotor $v = \sqrt{v_x^2 + v_y^2 + v_z^2} \approx 1.8$ m/s. The reference trajectory is as follow:

$$x^r(t) = 3.0sin(0.8t) \qquad y^r(t) = 3.0cos(0.8t) \qquad z^r(t) = 1.0$$

$$\dot{x}^r(t) = 3.0(0.8)cos(0.8t) \qquad \dot{y}^r(t) = -3.0(0.8)sin(0.8t) \qquad \dot{z}^r(t) = 0.0$$

$$\ddot{x}^r(t) = -3.0(0.8)^2sin(0.8t) \qquad \ddot{y}^r(t) = -3.0(0.8)^2cos(0.8t) \qquad \ddot{z}^r(t) = 0.0$$

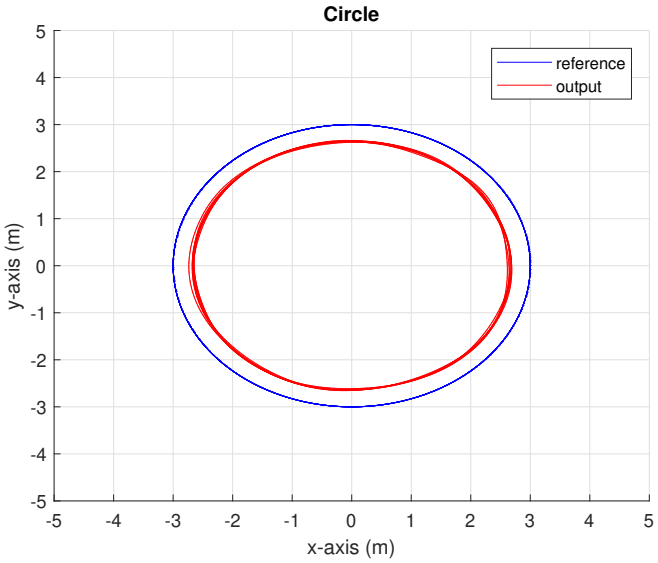


Figure 6.59: Circular Trajectory with $\omega = 0.8$ - LQ Servo Design 2

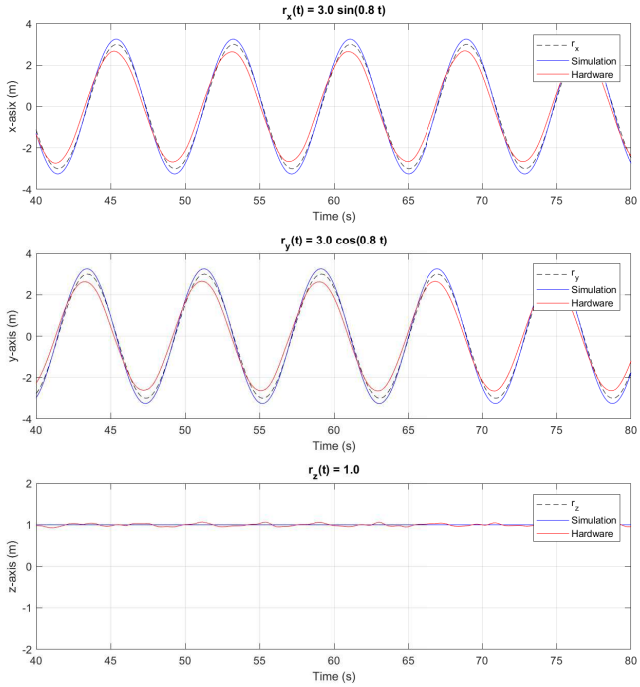


Figure 6.60: x, y, z Resulting from $r_x(t) = 3.0sin(0.8t)$, $r_y(t) = 3.0cos(0.8t)$, $r_z(t) = 1.0$ - LQ Servo Design 2

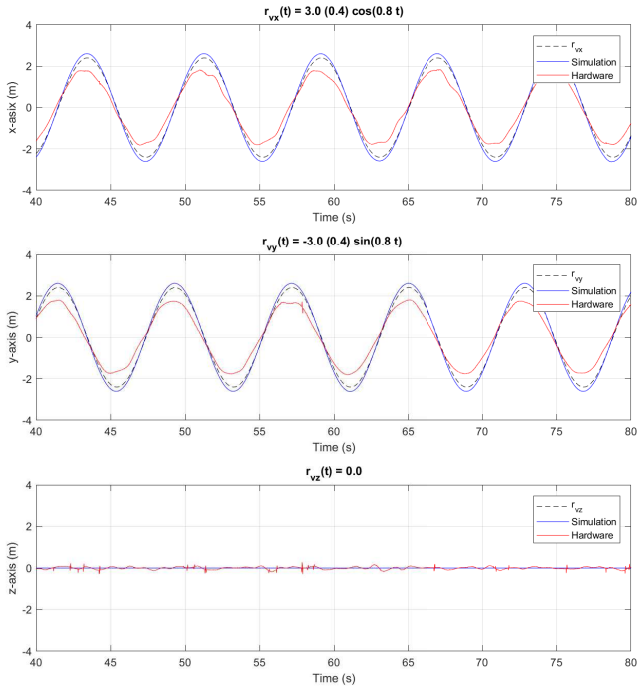


Figure 6.61: v_x, v_y, v_z Resulting from $r_{vx}(t) = 3.0(0.8)cos(0.8t)$, $r_{vy}(t) = -3.0(0.8)sin(0.8t)$, $r_{vz}(t) = 0.0$ - LQ Servo Design 2

5. **Path Following a Circular Trajectory with $\omega = 1.0$.** In this demonstration, a pre-defined circle trajectory is applied as a trajectory reference command to the quadrotor to follow. The HTC Vive Tracking System is used to demonstrate the following experiment. The velocity of the quadrotor $v = \sqrt{v_x^2 + v_y^2 + v_z^2} \approx 0.45$ m/s. The reference trajectory is as follow:

$$x^r(t) = 0.5sin(1.0t) \qquad y^r(t) = 0.5sin(1.0t) \qquad z^r(t) = 1.0$$

$$\dot{x}^r(t) = 0.5(1.0)cos(1.0t) \qquad \dot{y}^r(t) = -0.5(1.0)sin(1.0t) \qquad \dot{z}^r(t) = 0.0$$

$$\ddot{x}^r(t) = -0.5(1.0)^2sin(1.0t) \qquad \ddot{y}^r(t) = -0.5(1.0)^2cos(1.0t) \qquad \ddot{z}^r(t) = 0.0$$

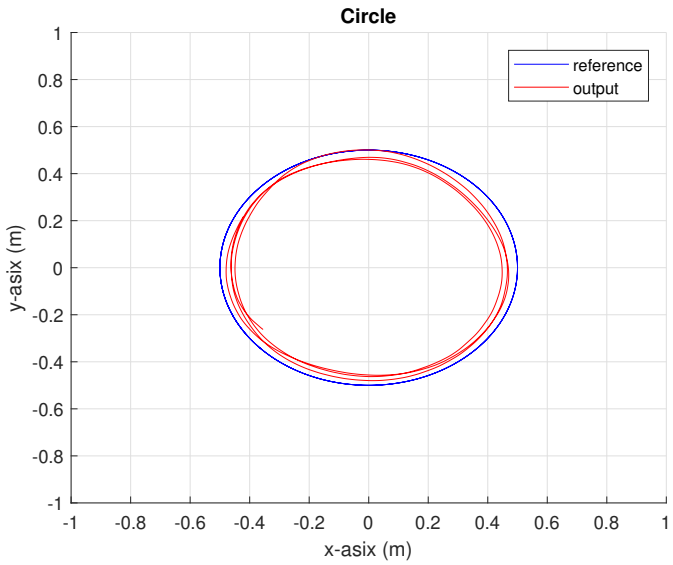


Figure 6.62: Circular Trajectory with $\omega = 1.0$ - LQ Servo Design 2

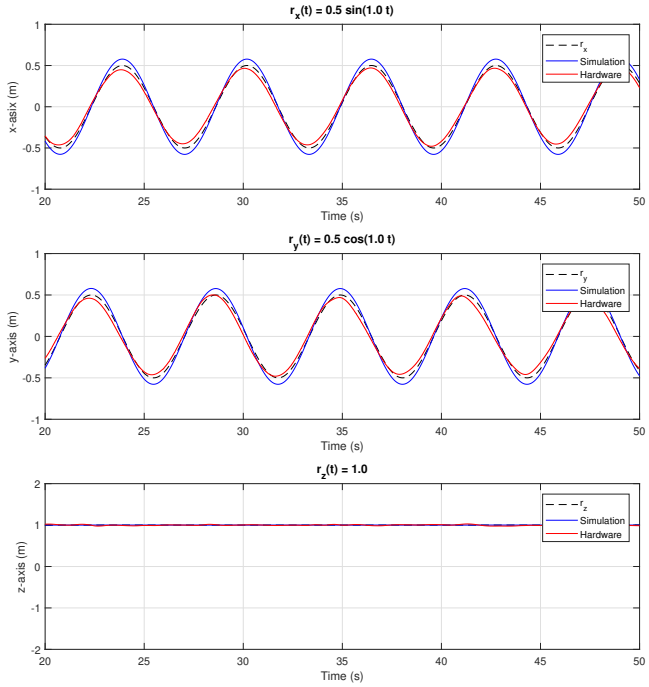


Figure 6.63: x, y, z Resulting from $r_x(t) = 0.5sin(1.0t)$, $r_y(t) = 0.5sin(1.0t)$, $r_z(t) = 1.0$ - LQ Servo Design 2

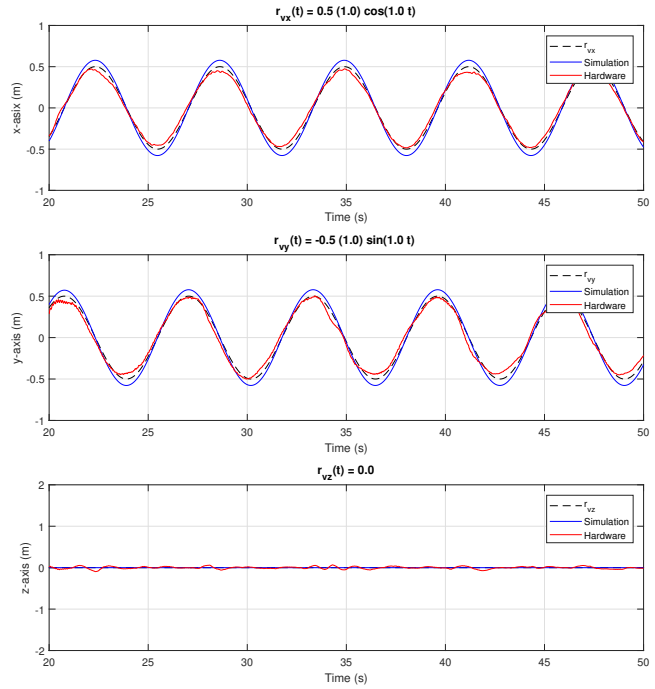


Figure 6.64: v_x, v_y, v_z Resulting from $r_{vx}(t) = 0.5(1.0)cos(1.0t)$, $r_{vy}(t) = -0.5(1.0)sin(1.0t)$, $r_{vz}(t) = 0.0$ - LQ Servo Design 2

6. **Path Following a Lemniscate/Figure Eight Trajectory with $\omega = 1.0$.** In this demonstration, a pre-defined lemniscate or figure eight trajectory is applied as a trajectory reference command to the quadrotor to follow. The HTC Vive Tracking System is used to demonstrate the following experiment. The velocity of the quadrotor $v = \sqrt{v_x^2 + v_y^2 + v_z^2} \approx 0.5$ m/s. The reference trajectory is as follow:

$$x^r(t) = 0.5cos(1.0t) \qquad y^r(t) = 0.5sin(1.0t)\ cos(1.0t) \qquad z^r(t) = 1.0$$

$$\dot{x}^r(t) = -0.5(1.0)sin(1.0t) \qquad \dot{y}^r(t) = 0.5cos(2.0t) \qquad \dot{z}^r(t) = 0.0$$

$$\ddot{x}^r(t) = -0.5(1.0)^2cos(1.0t) \qquad \ddot{y}^r(t) = -sin(2.0t) \qquad \ddot{z}^r(t) = 0.0$$

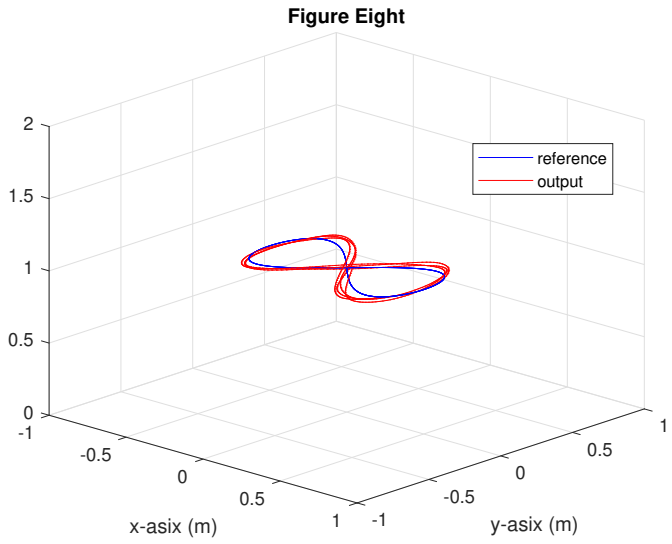


Figure 6.65: Figure Eight Trajectory with $\omega = 1.0$ - LQ Servo Design 2

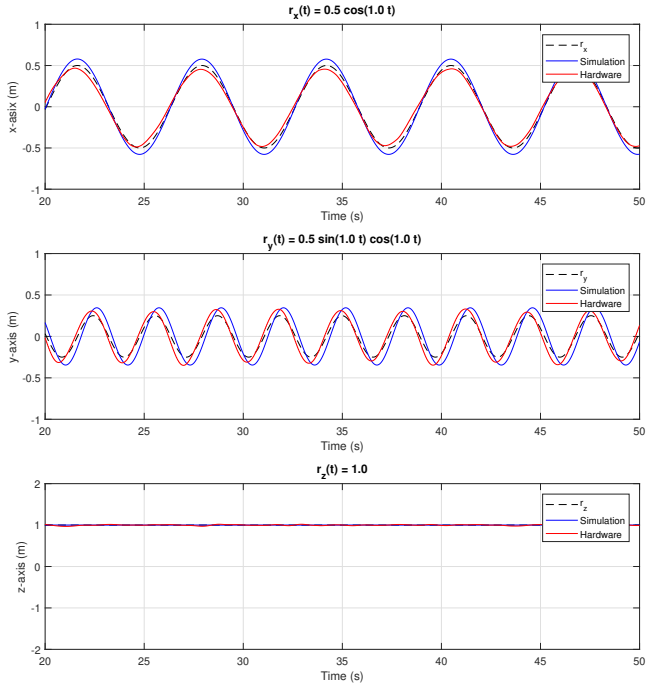


Figure 6.66: x, y, z Resulting from $r_x(t) = 0.5cos(1.0t)$, $r_y(t) = 0.5sin(1.0t)\ cos(1.0t)$, $r_z(t) = 1.0$ - LQ Servo Design 2

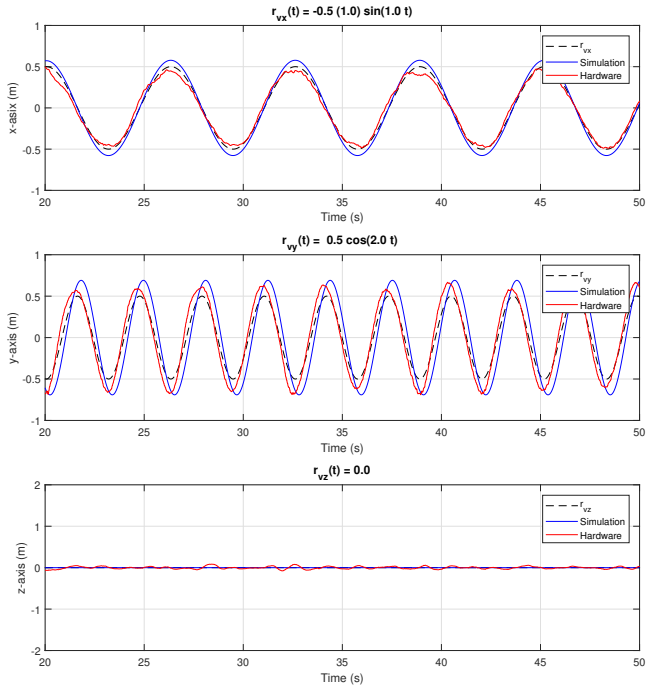


Figure 6.67: v_x, v_y, v_z Resulting from $r_{vx}(t) = -0.5(1.0)sin(1.0t)$, $r_{vy}(t) = 0.5cos(2.0t)$, $r_{vz}(t) = 0.0$ - LQ Servo Design 2

7. **Path Following a Vertical Lemniscate/Figure Eight Trajectory with $\omega = 1.0$.** In this demonstration, a pre-defined lemniscate or figure eight trajectory is applied as a trajectory reference command to the quadrotor to follow. The HTC Vive Tracking System is used to demonstrate the following experiment. The velocity of the quadrotor $v = \sqrt{v_x^2 + v_y^2 + v_z^2} \approx 0.5$ m/s. The reference trajectory is as follow:

$$x^r(t) = 0.0 \quad y^r(t) = 0.5cos(1.0t) \qquad z^r(t) = 1.0 + 0.5sin(1.0t)\ cos(1.0t)$$

$$\dot{x}^r(t) = 0.0 \quad \dot{y}^r(t) = -0.5(1.0)sin(1.0t) \quad \dot{z}^r(t) = 0.5cos(2.0t)$$

$$\ddot{x}^r(t) = 0.0 \quad \ddot{y}^r(t) = -0.5(1.0)^2cos(1.0t) \quad \ddot{z}^r(t) = -sin(2.0t)$$

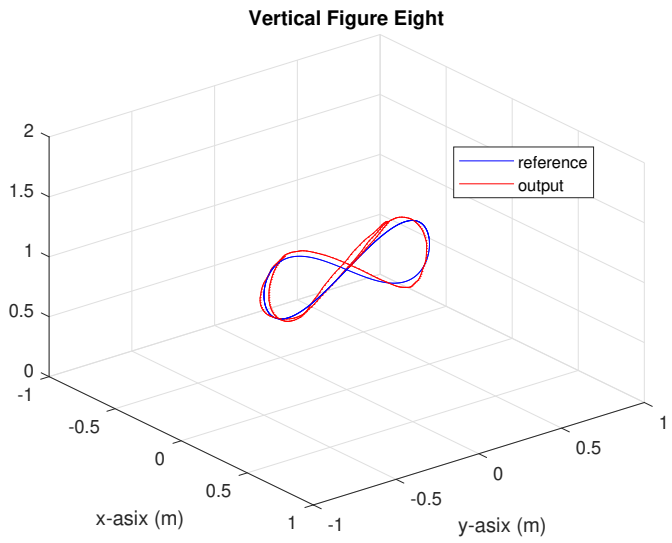


Figure 6.68: Figure Eight Trajectory with $\omega = 1.0$ - LQ Servo Design 2

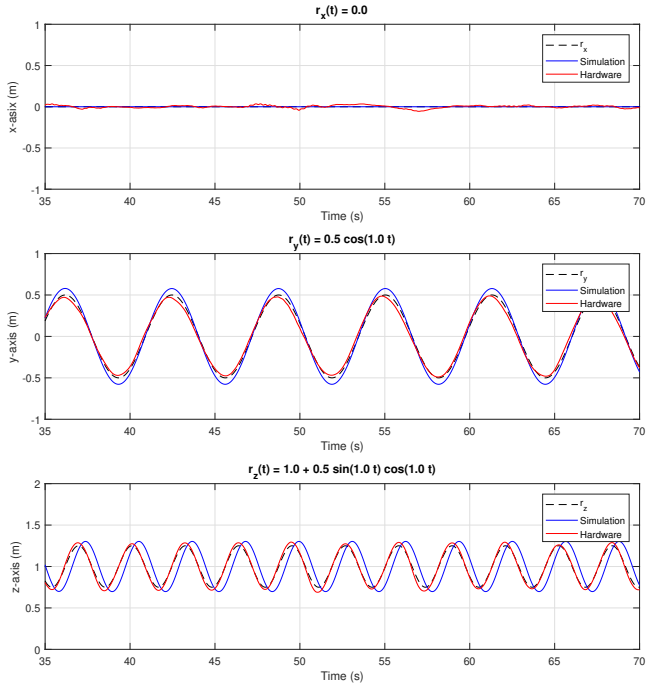


Figure 6.69: x, y, z Resulting from $r_x(t) = 0.0$, $r_y(t) = 0.5cos(1.0t)$, $r_z(t) = 1.0 + 0.5sin(1.0t)\ cos(1.0t)$ - LQ Servo Design 2

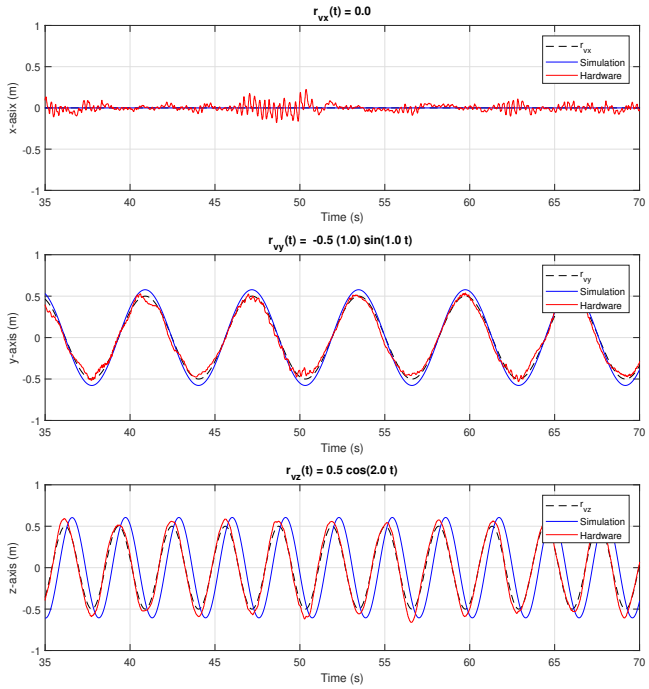


Figure 6.70: v_x, v_y, v_z Resulting from $r_{vx}(t) = 0.0$, $r_{vy}(t) = -0.5(1.0)sin(1.0t)$, $r_{vz}(t) = 0.5cos(2.0t)$ - LQ Servo Design 2

6.7 Summary and Conclusions

In summary, linearization of the nonlinear translational quadrotor dynamics has been shown in two different operating points. From that, we saw the quadrotor linear model can be decoupled if attitude angles ϕ and θ are close to zero (near hover); otherwise, the coupling will show up in the model. The decoupled linear model was considered to control system design in this project. Three control designs were designed: (1) LQ Servo design (2) Weighted $\mathcal{H}^\infty$ Sensitivity Optimization (3) LQG/LTRO design. From the frequency domain and time domain analysis, we see that both LQ design and Weighted $\mathcal{H}^\infty$ Design 2 have similar performance in general.

Chapter 7

MULTIPLE QUADROTOR FORMATION CONTROL USING LEADER-FOLLOWER APPROACH

7.1 Overview

Formation control problems became more attractive in the robotics research field — systems where multiple robots interact with each other to achieve a certain goal called the multi-agent system. More especially, aerial vehicles took great attention in the multi-agent system due to their high maneuverability in all three-dimensional space. They have been used in cooperative application [18], assembling structures [2], and performing dance [19]. From [37], formation control problems can be divided into two sub-categories: (1) Centralized and (2) Decentralized. In the centralized problem, a single unit (e.g., ground station) requires all information from all agents to make a decision. On the other hand, the decentralized problem is where each agent makes a decision locally. In this thesis, we examine the centralized problem where all information is gathered and processed on a central unit (computer ground station). Then, commands are issued to each agent from the ground station accordingly.

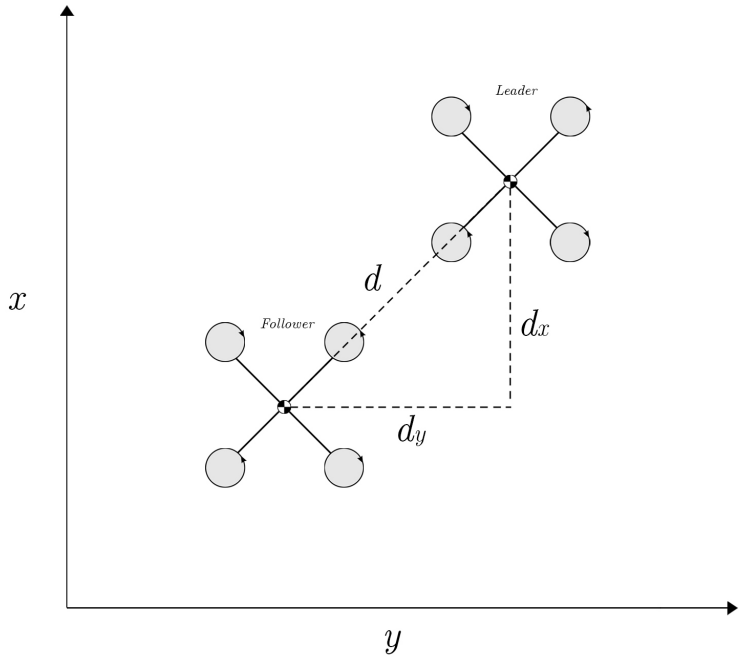


Figure 7.1: Leader-follower (x, y) Axis

Leader-Follower Approach. In the leader-follower approach, the leader quadrotor follows a specified trajectory while the follower quadrotor follows the leader while keeping the separation distance specified beforehand. In this thesis, two quadrotors were experimentally verified to show the leader-follower formation control. There are a couple of assumptions that were made:

- All quadrotors are identical and have the same control system.
- This is a centralized control problem; where all computations are performed on the ground station.
- Only the leader knows the reference trajectory to be followed.

- The followers maintain a constant separation distance in each direction, specified by a constant vector in x, y, z, from the leader.
- The leader has no information about the spacial position, velocity, or acceleration of the followers.

The following is a block diagram that illustration two the leader-follower control for two vehicles. The top loop is the leader quadrotor while the lower loop is the follower quadrotor

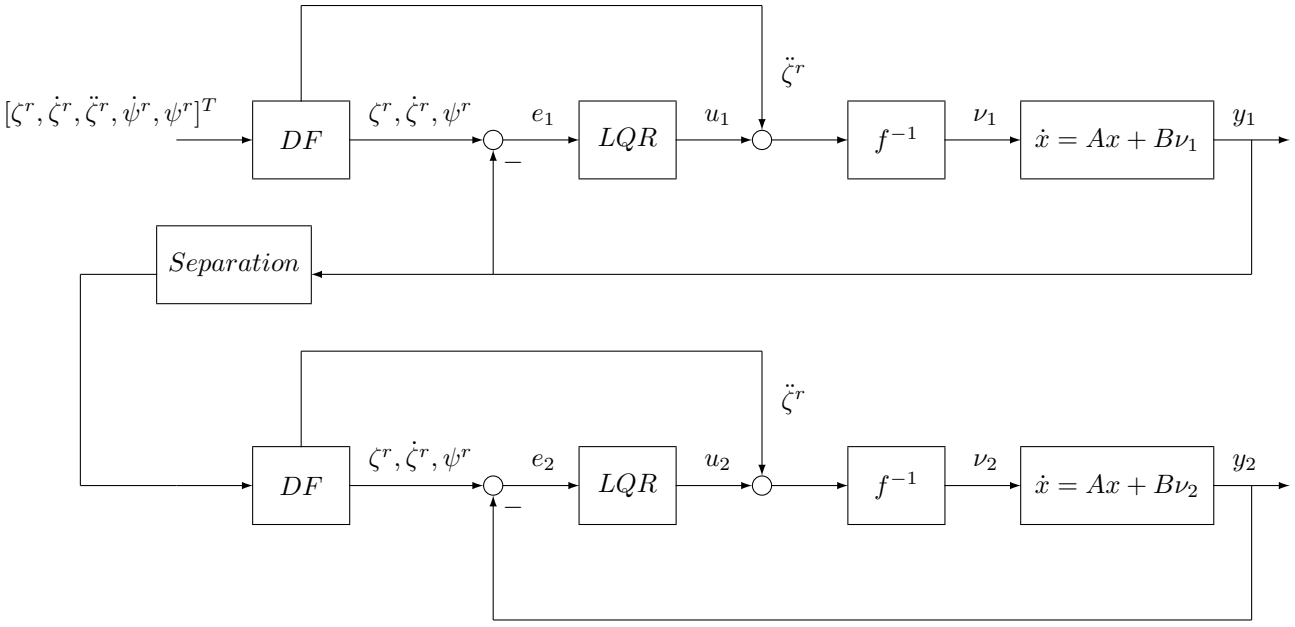


Figure 7.2: Leader-follower Feedback Block Diagram

In the following, the desired leader trajectory is as follows:

$$x_L = x^r \qquad y_L = y^r \qquad z_L = z^r$$

$$\dot{x}_L = \dot{x}^r \qquad \dot{y}_L = \dot{y}^r \qquad \dot{z}_L = \dot{z}^r$$

$$\ddot{x}_L = \ddot{x}^r \qquad \ddot{y}_L = \ddot{y}^r \qquad \ddot{z}_L = \ddot{z}^r$$

while the desired trajectory for the follower is the spatial position, velocity, and acceleration of the leader with the separation in each direction as follows:

$$x_F = x_L - d_x \qquad y_F = y_L - d_y \qquad z_F = z_L - d_z$$
$$\dot{x}_F = \dot{x}_L \qquad \dot{y}_F = \dot{y}_L \qquad \dot{z}_F = \dot{z}_L$$
$$\ddot{x}_F = \ddot{x}_L \qquad \ddot{y}_F = \ddot{y}_L \qquad \ddot{z}_F = \ddot{z}_L$$

Quadrotor Hardware Demonstrations. All hardware demonstrations are done using the LQR controller Design 2. The 3D motion data of positions, velocities, and accelerations are obtained from either HTC Vive Tracking System or OptiTrack Motion Capture System at a rate of 100Hz. Then, the data are processed and filtered on the ground station as well. The ground station computes the commanded roll ϕ, pitch θ, thrust T, and yaw rate r using the nonlinear mapping. The commands are sent through Xbee modules wirelessly from the ground station to each quadrotor at a rate of 100Hz. The following outlines the hardware demonstrations:

1. **Following a Line with a Separation of** 1.5 **meters in the** y**-axis.** In this demonstration, the leader quadrotor is following a reference trajectory along the x-axis while the follower quadrotor is following the leader while maintaining a separation distance of 1.5m in the y-axis. The HTC Vive Tracking System is used to demonstrate the following experiment. The spatial position, velocity, and acceleration of the leader are filtered using a moving average filter and then fed-back to the follower. The leader reference commands are as follow:

$$x^r = 0.5sin(0.6t) \qquad y^r = 0 \qquad z^r = 1$$
$$\dot{x}^r = 0.5(0.6)cos(0.6t) \qquad \dot{y}^r = 0 \qquad \dot{z}^r = 0$$
$$\ddot{x}^r = -0.5(0.6)^2 sin(0.6t) \qquad \ddot{y}^r = 0 \qquad \ddot{z}^r = 0$$

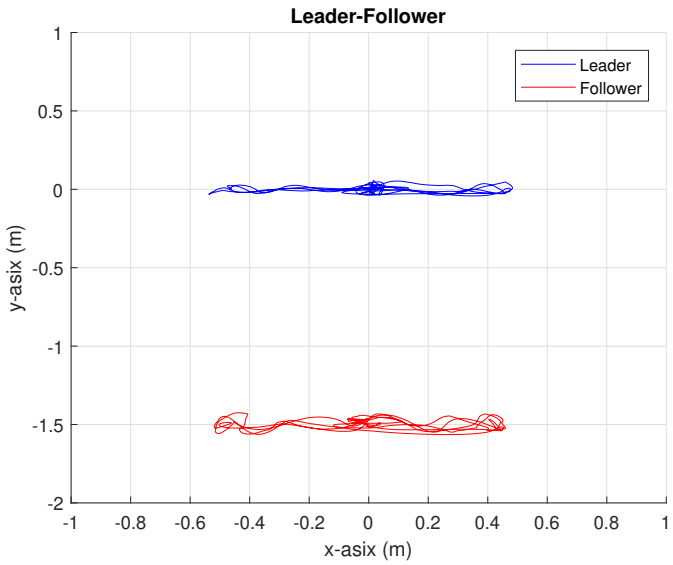


Figure 7.3: Figure Eight Trajectory with $\omega = 1.0$ - Leader-follower

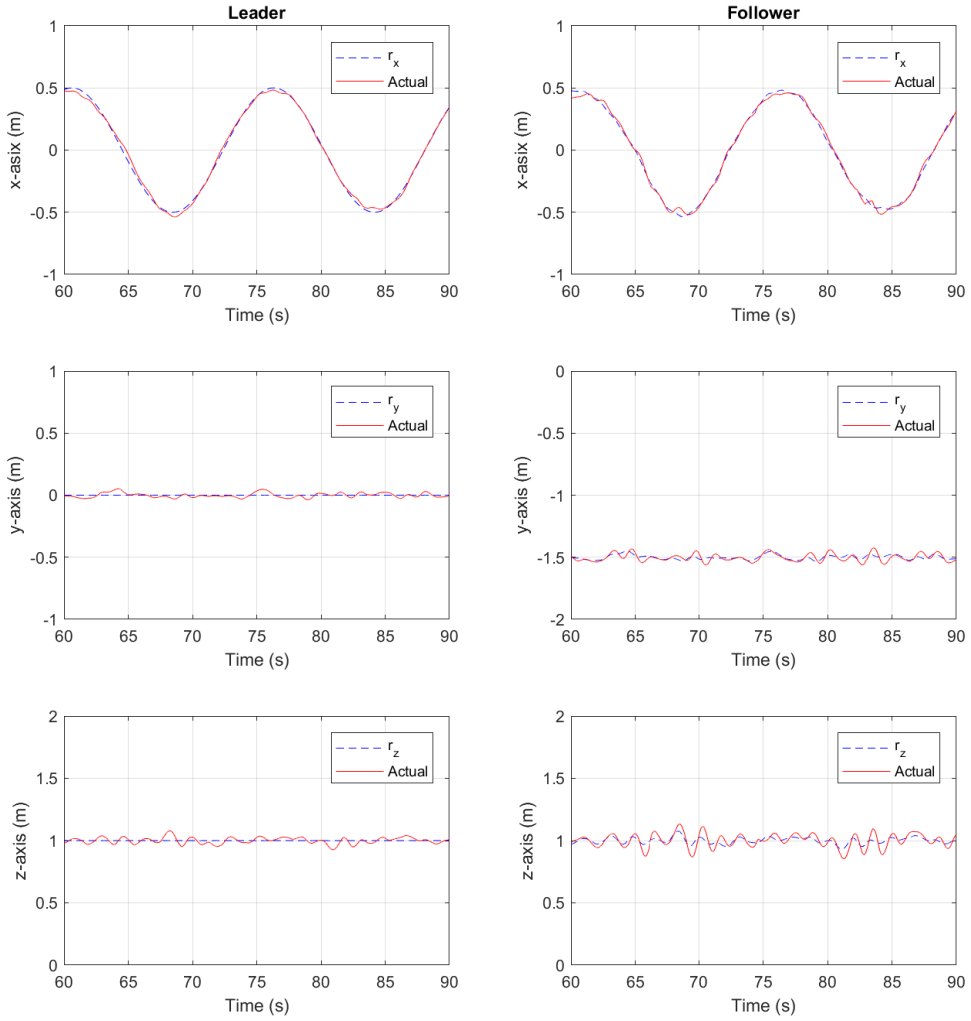


Figure 7.4: x, y, z Resulting from $r_x(t) = 0$, $r_y(t) = 0.5cos(1.0t)$, $r_z(t) = 1.0 + 0.5sin(1.0t)\ cos(1.0t)$ - Leader-follower

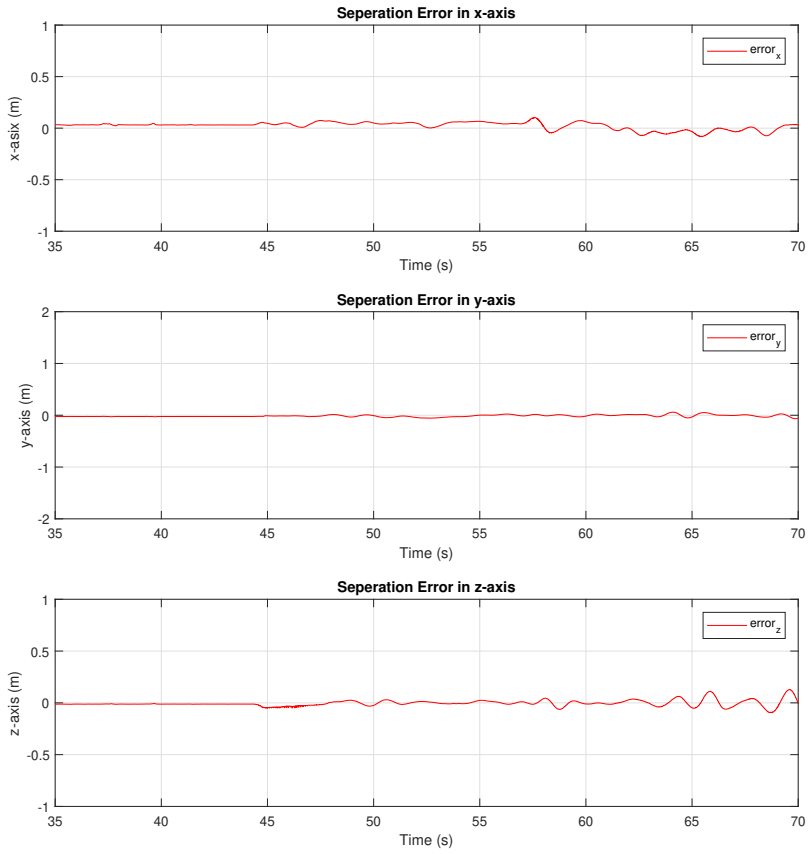


Figure 7.5: x, y, z Error Resulting from $r_x(t) = 0$, $r_y(t) = 0.5cos(1.0t$, $r_z(t) = 1.0 + 0.5sin(1.0t)\ cos(1.0t)$ - Leader-follower

2. **Following a Circular Path with a Separation of** 5.0 **meters in the** y**-axis.** In this demonstration, the leader quadrotor is following a circular reference trajectory when the follower quadrotor is following the leader while maintaining a separation distance of 5.0m in the y-axis. The OptiTrack mocap is used to demonstrate the following experiment. The spatial position, velocity, and acceleration of the leader are filtered using a moving average filter then fed-

back to the follower. The leader reference commands are as follow:

$$x^r = 1.0sin(0.2t) \qquad y^r = 3.0 + 1.0cos(0.2t) \qquad z^r = 1$$

$$\dot{x}^r = 1.0(0.2)cos(0.2t) \qquad \dot{y}^r = -1.0(0.2)sin(0.2t) \qquad \dot{z}^r = 0$$

$$\ddot{x}^r = -1.0(0.2)^2 sin(0.2t) \qquad \ddot{y}^r = -1.0(0.2)^2 cos(0.2t) \qquad \ddot{z}^r = 0$$

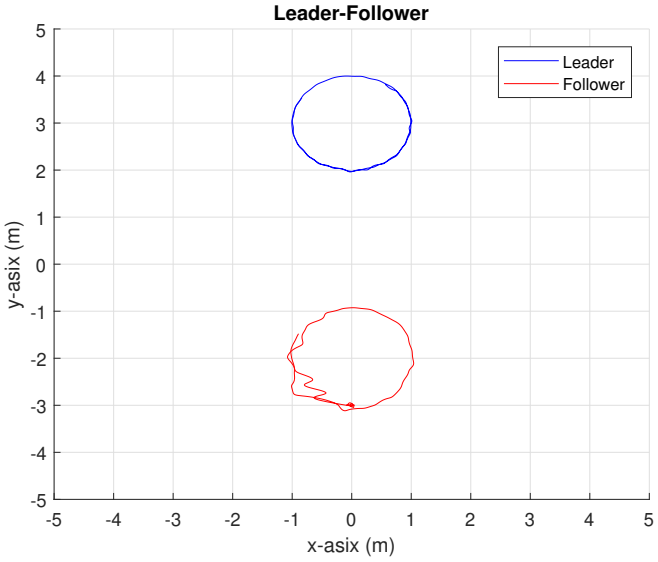


Figure 7.6: Circular Trajectory with $\omega = 0.2$ - Leader-follower

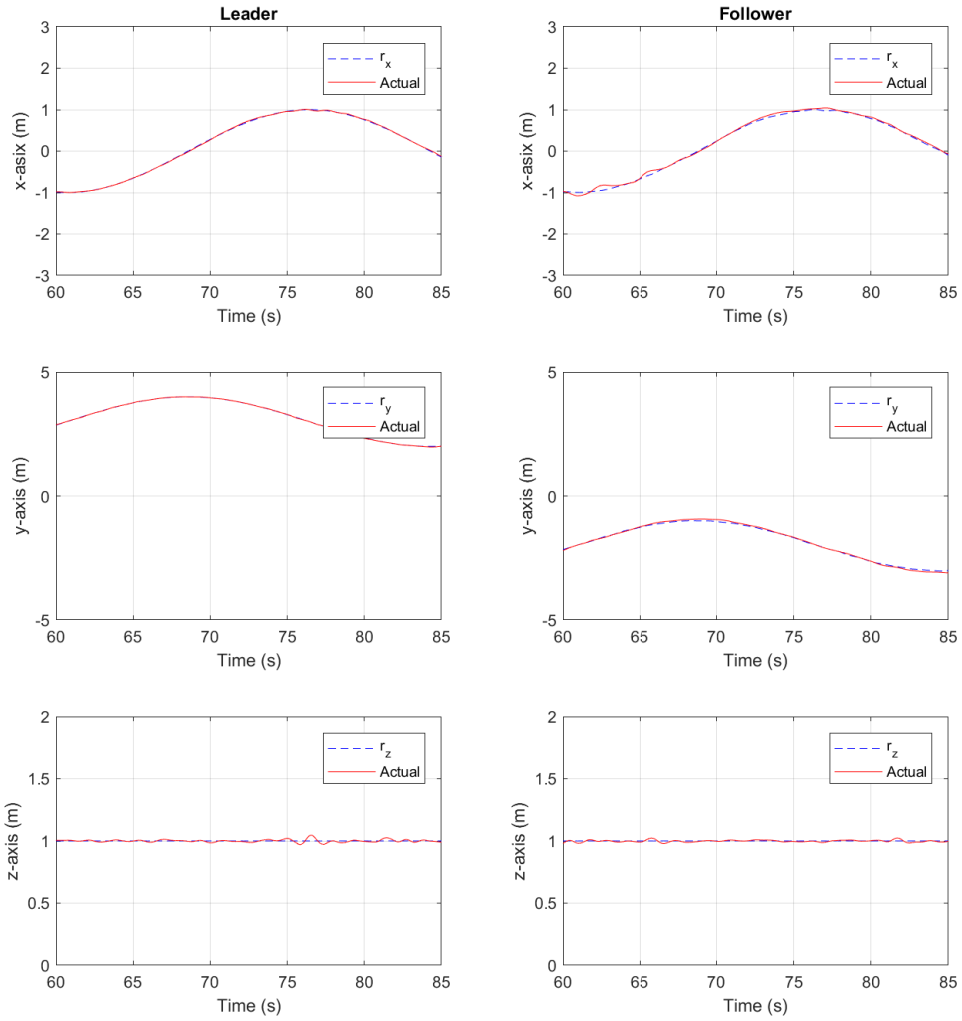


Figure 7.7: x, y, z Resulting from $r_x(t) = 1.0sin(0.2t)$, $r_y(t) = 3.0 + 1.0cos(0.2t)$, $r_z(t) = 1.0$ - Leader-follower

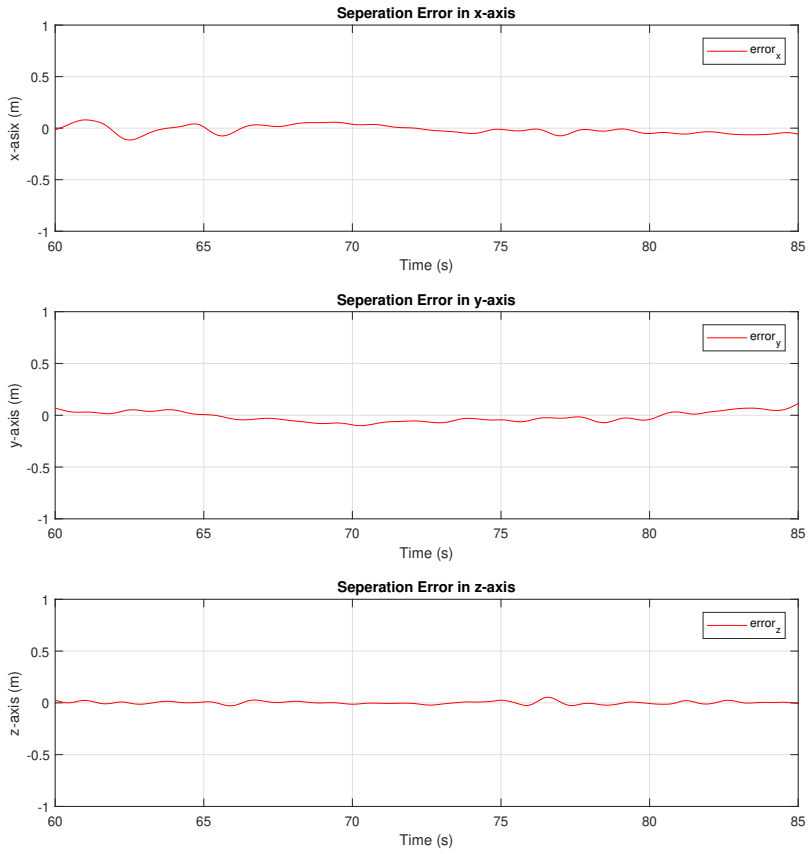


Figure 7.8: x, y, z Error Resulting from $r_x(t) = 1.0sin(0.2t)$, $r_y(t) = 3.0+1.0cos(0.2t)$, $r_z(t) = 1.0$ - Leader-follower

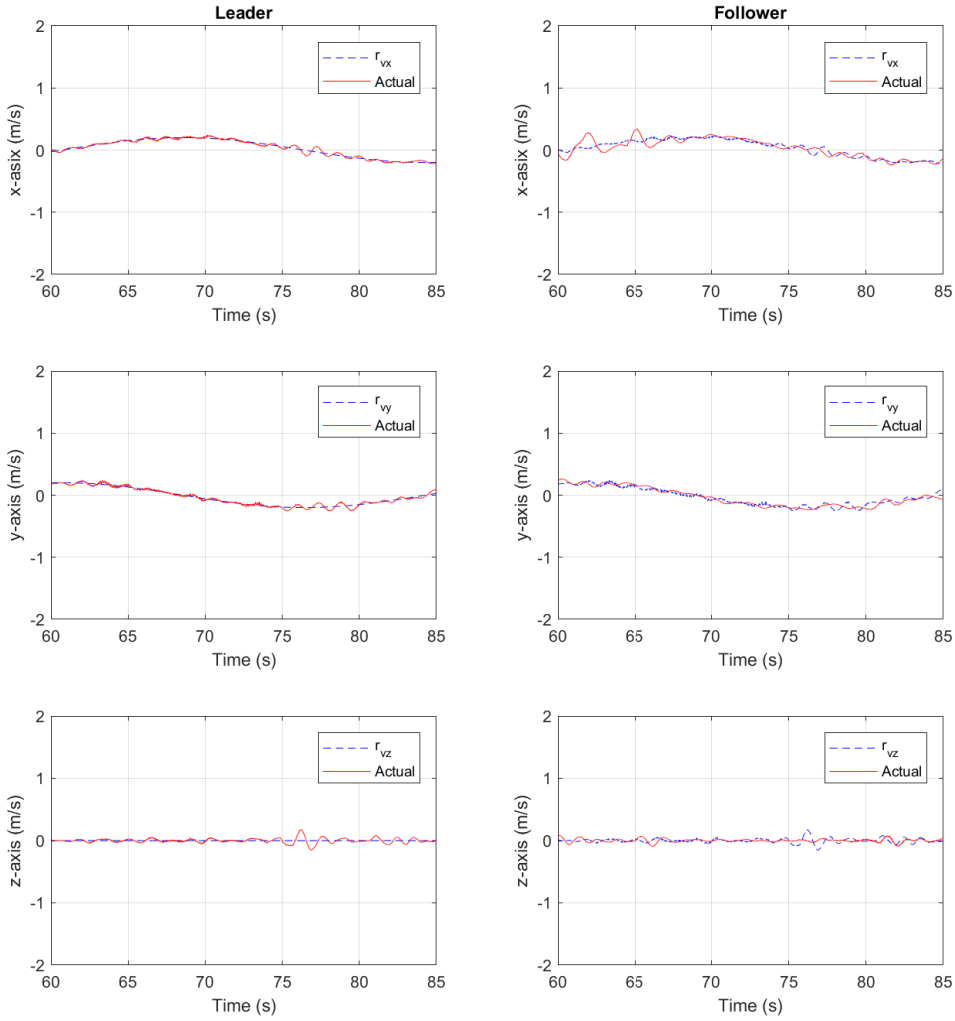


Figure 7.9: v_x, v_y, v_z Error Resulting from $r_x(t) = 1.0sin(0.2t)$, $r_y(t) = 3.0 + 1.0cos(0.2t)$, $r_z(t) = 1.0$ - Leader-follower

Chapter 8

SUMMARY AND DIRECTIONS FOR FUTURE RESEARCH

8.1 Summary

In this thesis, we presented how to build an open-source based platform for quadrotors for future formation control research. We studied the modeling of quadrotor's rotational and translational dynamics with trade-off studies as well. Classical controls were used to design low-level control for rotational dynamics (body rate and attitude). Furthermore, LQ Servo, Weighted $\mathcal{H}^\infty$ Sensitivity Optimization, and LQG/LTRO designs were used to design high-level control for translational dynamics (position and path following control). Simulation and hardware data are shown to be similar. This work is an attempt to design a platform that is going to be a foundation for future research in quadrotor's formation control.

8.2 Directions for Future Research

Future work will involve the following:

- **Onboard Sensing.** Use of more accurate internal measurement unit (IMU) that can provide a better estimation of body rotation rates and attitude angles. In addition to cameras that can perform onboard localization independent of the motion capture system, e.g., GPS, LIDAR, etc.

- **Formation Control.** Design more accurate centralized formation control methods where every agent is aware of the neighbors in addition to collision avoidance between agents.

- **Multi-Agent Cooperation.** Interaction and cooperation between air and ground vehicles to achieve a particular goal using the same platform developed in this thesis.

- **Estimation.** Implement a more accurate estimation for 3D motion data for HTC Vive and OptiTrack. This includes Kalman Filter, Extended Kalman Filter, Particle Filter, etc.

REFERENCES

[1] F. Augugliaro, S. Lupashin, M. Hamer, C. Male, M. Hehn, M. W. Mueller, J. S. Willmann, F. Gramazio, M. Kohler, and R. D'Andrea, "The flight assembled architecture installation: Cooperative construction with flying machines," *IEEE Control Systems Magazine*, vol. 34, no. 4, pp. 46–64, Aug 2014.

[2] H. Durrant-Whyte, N. Roy, and P. Abbeel, *Construction of Cubic Structures with Quadrotor Teams*. MITP, 2012. [Online]. Available: https://ieeexplore-ieee-org.ezproxy1.lib.asu.edu/document/6301066

[3] S. Bouabdallah, P. Murrieri, and R. Siegwart, "Design and control of an indoor micro quadrotor," in *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004*, vol. 5, April 2004, pp. 4393–4398 Vol.5.

[4] M. Y. Amir and V. Abbass, "Modeling of quadrotor helicopter dynamics," in *2008 International Conference on Smart Manufacturing Application*, April 2008, pp. 100–105.

[5] "Flying car and autonomous flight engineer nanodegree." [Online]. Available: https://www.udacity.com/course/flying-car-nanodegree--nd787

[6] S. BOUABDALLAH, "Design and control of quadrotors with application to autonomous flying," Ph.D. Dissertation, EPFL, 2007.

[7] Z. Benic, P. Piljek, and D. Kotarski, "Mathematical modelling of unmanned aerial vehicles with four rotors," *Interdisciplinary Description of Complex Systems - scientific journal*, vol. 14, no. 1, pp. 88–100, 2016. [Online]. Available: https://EconPapers.repec.org/RePEc:zna:indecs:v:14:y:2016:i:1:p:88-100

[8] R. Mahony, V. Kumar, and P. Corke, "Multirotor aerial vehicles: Modeling, estimation, and control of quadrotor," *IEEE Robotics Automation Magazine*, vol. 19, no. 3, pp. 20–32, Sep. 2012.

[9] D. Mellinger, N. Michael, and V. Kumar, "Trajectory generation and control for precise aggressive maneuvers with quadrotors," *The International Journal of Robotics Research*, vol. 31, no. 5, pp. 664–674, 2012. [Online]. Available: https://doi.org/10.1177/0278364911434236

[10] M. Bangura, "Aerodynamics and control of quadrotors," M.S. Thesis, The Australian National University, Canberra, 2017.

[11] A. A. Rodriguez, *Analysis and Design of Feedback Control Systems*. Tempe, AZ: CONTROL3D, L.L.C., 2002.

[12] Z. Lin, "Modeling, design and control of multiple low-cost robotic ground vehicles," M.S. Thesis, Arizona State University, Tempe, AZ, 2015.

[13] M. Faessler, D. Falanga, and D. Scaramuzza, "Thrust mixing, saturation, and body-rate control for accurate aggressive quadrotor flight," *IEEE Robotics and Automation Letters*, vol. 2, pp. 476–482, 2017.

[14] S. Lupashin, M. Hehn, M. W. Mueller, A. Schoellig, M. Sherback, and R. D'Andrea, "A platform for aerial robotics research and demonstration: The flying machine arena," *Mechatronics*, vol. 24, 02 2014.

[15] M. Faessler, F. Fontana, C. Forster, E. Mueggler, M. Pizzoli, and D. Scaramuzza, "Autonomous, vision-based flight and live dense 3d mapping with a quadrotor micro aerial vehicle," *Journal of Field Robotics*, vol. 33, 03 2015.

[16] J. Ferrin, R. Leishman, R. Beard, and T. McLain, "Differential flatness based control of a rotorcraft for aggressive maneuvers," 09 2011, pp. 2688–2693.

[17] J. Vuong, B. Byers, and R. Sharma, "Limitations of differential flatness based control of quadrotor and solutions," 12 2014.

[18] K. Sreenath and V. Kumar, "Dynamics, control and planning for cooperative manipulation of payloads suspended by cables from multiple quadrotor robots," 06 2013.

[19] F. Augugliaro, A. Schoellig, and R. D'Andrea, "Dance of the flying machines: Methods for designing and executing an aerial dance choreography," pp. 96–104, 2013.

[20] D. Mercado-Ravell, R. Castro-Linares, and R. Lozano, "Quadrotors flight formation control using a leader-follower approach," 07 2013, pp. 3858–3863.

[21] E. Abbasi, M. Ghayour, M. Danesh, P. Amiri, and M. Yoosefian, "Formation flight control and path tracking of a multi-quadrotor system in the presence of measurement noise and disturbances," 10 2018.

[22] X. Chen, "Flight controller design, automatic tuning and performance evaluation of quadrotor uavs," Ph.D Dissertation, RMIT University, 2017.

[23] S. Lu, "Modeling, control and design of a quadrotor platform for indoor environments," M.S. Thesis, Arizona State University, Tempe, AZ, 2018.

[24] MAVLink, "Introduction." [Online]. Available: https://mavlink.io/en/

[25] K. Mondal, "Multivariable control of fixed wing aircrafts," M.S. Thesis, Arizona State University, Tempe, AZ, 2015.

[26] Q. Quan, *Introduction to Multicopter Design and Control*, 06 2017.

[27] techSultan, "How does a hobby esc drive a bldc motor?" [Online]. Available: https://electronics.stackexchange.com/questions/355167/how-does-a-hobby-esc-drive-a-bldc-motor

[28] C. H. Wolowicz, J. S. Brown, and W. P. Gilbert, "Similitude requirements and scaling relationships as applied to model testing," 1979.

[29] A. Michael Harrington, "Optimal propulsion system design for a micro quad rotor," 01 2011.

[30] A. A. Rodriguez, *Analysis and Design of Multivariable Feedback Control Systems*. Tempe, AZ: CONTROL3D, L.L.C., 2002.

[31] M. Shafique, "Lecture notes in computer control systems," April 2019.

[32] B. Douglas, "Understanding pid control, part 2: Expanding beyond a simple integral video," Jun 2018. [Online]. Available: https://www.mathworks.com/videos/understanding-pid-control-part-2-expanding-beyond-a-simple-integral-1528310418260.html

[33] R. M. Murray, M. Rathinam, and W. Sluis, "Differential flatness of mechanical control systems: A catalog of prototype systems," in *Proceedings of the 1995 ASME International Congress and Exposition*, 1995.

[34] A. E. C. D. Cunha, "Benchmark: Quadrotor attitude control," in *ARCH14-15. 1st and 2nd International Workshop on Applied veRification for Continuous and Hybrid Systems*, ser. EPiC Series in Computing, G. Frehse and M. Althoff, Eds., vol. 34. EasyChair, 2015, pp. 57–72. [Online]. Available: https://easychair.org/publications/paper/mwnd

[35] W. Dong, G.-Y. Gu, X. Zhu, and H. Ding, "Modeling and control of a quadrotor uav with aerodynamic concepts," in *Proceedings of World Academy of Science, Engineering and Technology*, no. 77. World Academy of Science, Engineering and Technology (WASET), 2013, p. 437.

[36] K. Puttannaiah, "A generalized h-infinity mixed sensitivity convex approach to multivariable control design subject to simultaneous output and input loop-breaking specifications," Ph.D. Dissertation, Arizona State University, Tempe, AZ, 2018.

[37] A. Jiménez, V. García-Díaz, and S. Bolaños, "A decentralized framework for multi-agent robotic systems," *Sensors*, vol. 18, no. 2, p. 417, 2018.

APPENDIX A

MATLAB CODE

MATLAB Version: 9.3.0.9483333 (R2017b).

A.1 High-level Control

```
% Title: Quadrotor Linear Translational Dynamics
% References: http://aar.faculty.asu.edu/classes/eee598S98/eee598c.html
% Control Variables
%     u =  [ delta_theta, pitch angle (rad)
%            delta_phi, roll angle (rad)
%            delta_T, thrust (N)
%            delta_r, yaw rate (rad/s) ]
%
%
% State Variables
%      x = [ x, x-axis position (m)
%            y, y-axis position (m)
%            z, z-axis position (m)
%            vx, velocity in the x-axis (m/s)
%            vy, velocity in the y-axis (m/s)
%            vz, velocity in the z-axis (m/s)
%            psi, yaw angle (rad) ]
%
%
% Output Variables
%      y = [ x, x-axis position (m)
%            y, y-axis position (m)
%            z, z-axis position (m)
%            psi, yaw angle (rad) ]
%
%*****************************************************
% Linearization Around Hover without Drag
% where T_0 = mg (N), phi_0 = theta_0 = psi_0 = 0 (rad)
s = tf('s');
g = 9.81;
m = 0.666;
Ap1 = [zeros(3,3) eye(3,3) zeros(3,1); zeros(4,3) zeros(4,3) ...
    zeros(4,1)];
Bp1 = [zeros(3,3) zeros(3,1); -g 0 0 0; 0 g 0 0; 0 0 1/m 0; 0 0 0 1];
Cp1 = [eye(3,3) zeros(3,4); zeros(1,3) zeros(1,3) 1];
Dp1 = zeros(4,4);

% Changing Units from radians to degrees
r2d = 180/pi;
su  = diag( [ r2d,   r2d,  1,  r2d ] );
sx  = diag( [ 1, 1, 1, 1, 1, 1, r2d ] );
sy  = diag( [ 1, 1 , 1,  r2d ] );
Ap1 = sx*Ap1*inv(sx);
Bp1 = sx*Bp1*inv(su);
Cp1 = sy*Cp1*inv(sx);
Dp1 = sy*Dp1*inv(su);

%*****************************************************
% Linearization Around Hover with Linear Drag
% beta = rho*Cd*vx_e*Ss*
% vx_e = 1 (m/s), Ss = 0.015625 (m^2), rho = 1.225, Cd = 0.47
```

```
beta = 0.0090;
Ap2 = [zeros(3,3) eye(3,3) zeros(3,1); zeros(3,3) -beta*eye(3,3) ...
   zeros(3,1); zeros(1,7)]
Bp2 = [zeros(3,3) zeros(3,1); -g 0 0 0; 0 g 0 0; 0 0 1/m 0; 0 0 0 1];
Cp2 = [eye(3,3) zeros(3,4); zeros(1,3) zeros(1,3) 1];
Dp2 = zeros(4,4);

% Changing Units from radians to degrees
r2d = 180/pi;
su  = diag( [ r2d,   r2d,  1,  r2d ] );
sx  = diag( [ 1, 1, 1, 1, 1, 1, r2d ] );
sy  = diag( [ 1, 1 , 1,  r2d ] );
Ap2 = sx*Ap2*inv(sx);
Bp2 = sx*Bp2*inv(su);
Cp2 = sy*Cp2*inv(sx);
Dp2 = sy*Dp2*inv(su);


% ------------------------------------------------
% Plant Dimensions
%
[ns,nc] = size(Bp1);                  % Number of States, Number of ...
   Controls
[no,~]      = size(Cp1);

% First System with no drag
[ns1,nc1] = size(Bp1);                   % Number of States, Number ...
   of Controls
[no1,~]      = size(Cp1);               % Number of Outputs

% Second System with drag
[ns2,nc2] = size(Bp2);                   % Number of States, Number ...
   of Controls
[no2,~]      = size(Cp2);               % Number of Outputs


% ------------------------------------------------
% Natural Modes: Poles (Eigenvalues), Eigenvectors
%
% First System with no drag
[evec1,eval1] = eig(Ap1)   % evec contains eigenvectors
                       % eval contains poles or eigenvalues
% Second System with drag
[evec2,eval2] = eig(Ap2)


% ------------------------------------------------
% Transmission Zeros
%
% First System with no drag
plantzeros1 = tzero(ss(Ap1,Bp1,Cp1,Dp1))       % transmission zeros
% System has no finite transmission zeros

% Second System with drag
plantzeros2 = tzero(ss(Ap2,Bp2,Cp2,Dp2))       % transmission zeros
% System has no finite transmission zeros
```

```
% ------------------------------------------------
% SYSTEM TRANSFER FUNCTIONS: From u_i to x_j
%
% First System with no drag
Plant_zpk1 = zpk(ss(Ap1,Bp1,Cp1,Dp1)) % Zeros, Poles, and Gains fron ...
    u_i to x_j
% Second System with drag
Plant_zpk2 = zpk(ss(Ap2,Bp2,Cp2,Dp2)) % Zeros, Poles, and Gains fron ...
    u_i to x_j
% ------------------------------------------------
% Controllability
%
% First System with no drag
cm1  = [Bp1 Ap1*Bp1 (Ap1^2)*Bp1  (Ap1^3)*Bp1 (Ap1^4)*Bp1 (Ap1^5)*Bp1 ...
    (Ap1^6)*Bp1]; % Controllability Matrix
rcm1 = rank(cm1)               % Rank of Controllability Matrix
% Second System with drag
cm2  = [Bp2 Ap2*Bp2 (Ap2^2)*Bp2  (Ap2^3)*Bp2 (Ap2^4)*Bp2 (Ap2^5)*Bp2 ...
    (Ap2^6)*Bp2]; % Controllability Matrix
rcm2 = rank(cm2)               % Rank of Controllability Matrix
% ------------------------------------------------
% Observability
%
% First System with no drag
om1  = [Cp1; Cp1*Ap1; Cp1*(Ap1^2); Cp1*(Ap1^3); Cp1*(Ap1^4); ...
    Cp1*(Ap1^5); Cp1*(Ap1^6)]; % Observability Matrix
rom1 = rank(om1)              % Rank of Observability Matrix
% Second System with drag
om2  = [Cp2; Cp2*Ap2; Cp2*(Ap2^2); Cp2*(Ap2^3); Cp2*(Ap2^4); ...
    Cp2*(Ap2^5); Cp2*(Ap2^6)]; % Observability Matrix
rom2 = rank(om2)              % Rank of Observability Matrix

% FREQUENCY RESPONSE: Singular Values
%
% u = [ theta   (rad)     phi   (rad)   T   (N)        r   (rad/s) ]
% x = [ x   (m/s)         y   (m/s)     z   (m/s)      vx   (m/s) ...
          vy   (m/s)    vz   (m/s)     psi (rad)   ]
% y = [ x   (m/s)         y   (m/s)     z   (m/s)      psi (rad)   ]
%
winit  = -4;
wfin   =  1;
nwpts  = 200;
w      = logspace(winit,wfin,nwpts);   % Form vector of ...
    logarithmically spaced freq points
sv     = sigma(ss(Ap1, Bp1, Cp1, Dp1),w);
sv     = 20*log10(sv);
figure; semilogx(w, sv, 'b')
%clear sv
title('Outputs: x, y, z (m), \psi (deg); Inputs: \theta, \phi (deg), ...
    T (N), r (deg/s)')
grid
xlabel('Frequency (rad/sec)')
ylabel('Singular Values (dB)')
hold on
sv     = sigma(ss(Ap2, Bp2, Cp2, Dp2),w);
sv     = 20*log10(sv);
semilogx(w, sv, 'r')
```

```
pause

% PLANT SVD ANALYSIS at Low Frequencies, w = 0.01 (rad/s)
%
% First System with no drag
w0 = 0.01;
P_w0            =  Cp1*inv(w0*eye(7,7)-Ap1)*Bp1;
[udc1,sdc1,vdc1] = svd(P_w0)
% Second System with drag
P_w0            =  Cp2*inv(w0*eye(7,7)-Ap2)*Bp2;
[udc2,sdc2,vdc2] = svd(P_w0)

%% ...
    ************************************************************************
%
% First Design for Linear Quadratic Regulator (LQR) with Integrator
% Augment Plant with Integrators
% For Zero Steady Error to Step Commands
% This follows from the Internal Model Priciple
% State x = [xp  xI]
% where
%       xp is the state
%       xI is the integrator state
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% First System with no drag
A1 = [Ap1 zeros(7,4); Cp1 zeros(4,4)];
B1 = [Bp1; zeros(4,4)];
C1 = [Cp1 zeros(4,4)];
D1 = zeros(4,4);
Cr1 = [0 0 0 1 0 0 0;
       0 0 0 0 1 0 0;
       0 0 0 0 0 1 0];
% Second System with drag
A2 = [Ap2 zeros(7,4); Cp2 zeros(4,4)];
B2 = [Bp2; zeros(4,4)];
C2 = [Cp2 zeros(4,4)];
D2 = zeros(4,4);
Cr2 = [0 0 0 1 0 0 0;
       0 0 0 0 1 0 0;
       0 0 0 0 0 1 0];
% LQR Design Parameters
rho = 0.1;
Q = diag([10,10,10,10,10,10,1,100,100,100,1]);
R = rho * eye(4,4);
[G1, K1, clpoles1] = lqr(A1,B1,Q,R);
[G2, K2, clpoles2] = lqr(A2,B2,Q,R);
%************************************************************************
%
% LQ OPEN LOOP FREQUENCY RESPONSE
%
gy1 = [G1(:,1:3) G1(:,7)];
gr1 = G1(:,4:6);
gz1 = G1(:,8:11);
gy2 = [G2(:,1:3) G2(:,7)];
gr2 = G2(:,4:6);
gz2 = G2(:,8:11);
```

```
% First System with no drag
aol1 = [  Ap1-Bp1*gr1*Cr1   Bp1*gz1;
          zeros(4,7)        zeros(4,4)  ];
bol1 = [  Bp1*gy1;
          eye(4,4)  ];
col1 = [  Cp1   zeros(4,4)  ];
dol1 = zeros(4,4);
ols1 = ss(aol1, bol1, col1, dol1);

% Second System with drag
aol2 = [  Ap2-Bp2*gr2*Cr2   Bp2*gz2;
          zeros(4,7)        zeros(4,4)  ];
bol2 = [  Bp2*gy2;
          eye(4,4)  ];
col2 = [  Cp2   zeros(4,4)  ];
dol2 = zeros(4,4);
ols2 = ss(aol2, bol2, col2, dol2);


w = logspace(-3,3,100);
sv = sigma(ss(A1, B1, G1, 0*ones(4,4)),w);
sv = 20*log10(sv);
figure;semilogx(w, sv, 'b')
%clear sv
title('Open Loop Singular Values: Plant Input')
grid
xlabel('Frequency (rad/sec)')
ylabel('Singular Values (dB)')
hold on
sv = sigma(ss(A2, B2, G2, 0*ones(4,4)),w);
sv = 20*log10(sv);
semilogx(w, sv, 'r')
hold off
pause

w = logspace(-3,3,100);
sv = sigma(ols1,w);
sv = 20*log10(sv);
figure;semilogx(w, sv, 'b')
%clear sv
title('Open Loop Singular Values: Error Signal')
grid
xlabel('Frequency (rad/sec)')
ylabel('Singular Values (dB)')
hold on
sv = sigma(ols2,w);
sv = 20*log10(sv);
semilogx(w, sv, 'r')
pause
%return

%******************************************************************************
%
% LQ CLOSED LOOP FREQUENCY RESPONSE
%
% First System with no drag
acl1 = aol1 - bol1*col1;
```

```
bcl1 = bol1;
ccl1 = col1;
dcl1 = dol1;
cls1 = ss(acl1,bcl1,ccl1,dcl1);
% Second System with drag
acl2 = aol2 - bol2*col2;
bcl2 = bol2;
ccl2 = col2;
dcl2 = dol2;
cls2 = ss(acl2,bcl2,ccl2,dcl2);

% Closed Loop Poles
% First System with no drag
[Wn1,zeta1, clpoles1] = damp(cls1)
% Second System with drag
[Wn2,zeta2, clpoles2] = damp(cls2)

sv = sigma(ss(A1-B1*G1, B1, -G1, eye(4,4)-0*ones(4,4)),w);
sv = 20*log10(sv);
figure;semilogx(w, sv, 'b')
%clear sv
title('LQ  Sensitivity: Plant Input')
grid
xlabel('Frequency (rad/sec)')
ylabel('Singular Values (dB)')
hold on
sv = sigma(ss(A2-B2*G2, B2, -G2, eye(4,4)-0*ones(4,4)),w);
sv = 20*log10(sv);
semilogx(w, sv, 'r')
hold off
pause

sv = sigma(ss(acl1, bcl1, -ccl1, eye(4,4)-dcl1),w);
sv = 20*log10(sv);
figure;semilogx(w, sv, 'b')
%clear sv
title('LQ Sensitivity: Error Signal')
grid
xlabel('Frequency (rad/sec)')
ylabel('Singular Values (dB)')
hold on
sv = sigma(ss(acl2, bcl2, -ccl2, eye(4,4)-dcl2),w);
sv = 20*log10(sv);
semilogx(w, sv, 'r')
pause

sv = sigma(ss(acl1, bcl1, ccl1, dcl1),w);
sv = 20*log10(sv);
figure;semilogx(w, sv, 'b')
%clear sv
title('LQ Complementary Sensitivity: Plant Output')
grid
xlabel('Frequency (rad/sec)')
ylabel('Singular Values (dB)')
hold on
sv = sigma(ss(acl2, bcl2, ccl2, dcl2),w);
sv = 20*log10(sv);
```

```
semilogx(w, sv, 'r')
hold off
pause
%return

sv = sigma(ss(A1-B1*G1, B1, G1, 0*ones(4,4)),w);
sv = 20*log10(sv);
figure;semilogx(w, sv, 'b')
%clear sv
title('LQ Complementary Sensitivity: Plant Input')
grid
xlabel('Frequency (rad/sec)')
ylabel('Singular Values (dB)')
hold on
sv = sigma(ss(A2-B2*G2, B2, G2, 0*ones(4,4)),w);
sv = 20*log10(sv);
semilogx(w, sv, 'r')
hold off
pause

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% CLOSED LOOP COMMAND FOLLOWING
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% First System
%
t = [0:0.02:10];
[y, t, x] = step(cls1,t);

%
% POSITION IN X-AXIS COMMAND
%
% x: r = [1  0  0  0] x-axis position Command
%
figure;
subplot(2,1,1)
plot(t,y(:,:,1))
grid
title('Output Response To r = [1  0  0  0]  Command')
ylabel('Outputs')
xlabel('Time (seconds)')
legend('x', 'y', 'z', 'psi')
%
% Controls: r = [1  0  0  0] x-axis position Command
%
u10 = [-G1 gy1]*[x(:,:,1)'
                 ones(1, size(x(:,:,1)')*[0 1]')
                 0*ones(1, size(x(:,:,1)')*[0 1]')
                 0*ones(1, size(x(:,:,1)')*[0 1]')
                 0*ones(1, size(x(:,:,1)')*[0 1]')];
subplot(2,1,2)
plot(t,u10)
grid
title('Input Response To r = [1  0  0  0]  Command')
ylabel('Controls')
xlabel('Time (seconds)')
legend('Pitch', 'Roll', 'Thrust', 'Yaw Rate')
```

```
pause

%
% POSITION IN Y-AXIS COMMAND
%
% y: r = [0  1  0  0] y-axis position Command
%
figure;
subplot(2,1,1)
plot(t,y(:,:,2))
grid
title('Output Response To r = [0  1  0  0]  Command')
ylabel('Outputs')
xlabel('Time (seconds)')
legend('x', 'y', 'z', 'psi')


%
% Controls: r = [0  1  0  0] y-axis position Command
%
u20 = [-G1 gy1]*[x(:,:,2)'
                 0*ones(1, size(x(:,:,2)')*[0 1]')
                 ones(1, size(x(:,:,2)')*[0 1]')
                 0*ones(1, size(x(:,:,2)')*[0 1]')
                 0*ones(1, size(x(:,:,2)')*[0 1]')];
subplot(2,1,2)
plot(t,u20)
grid
title('Input Response To r = [0  1  0  0]  Command')
ylabel('Pitch, Roll (deg), Thrust (N), Yaw (deg/s)')
xlabel('Time (seconds)')
legend('Pitch', 'Roll', 'Thrust', 'Yaw Rate')
pause

%
% POSITION IN Z-AXIS COMMAND
%
% z: r = [0  0  1  0] z-axis position Command
%
figure;
subplot(2,1,1)
plot(t,y(:,:,3))
grid
title('Output Response To r = [0  0  1  0]  Command')
ylabel('Outputs')
xlabel('Time (seconds)')
legend('x', 'y', 'z', 'psi')
%
% Controls: r = [0  0  1  0] z-axis position Command
%
u30 = [-G1 gy1]*[x(:,:,3)'
                 0*ones(1, size(x(:,:,3)')*[0 1]')
                 0*ones(1, size(x(:,:,3)')*[0 1]')
                 ones(1, size(x(:,:,3)')*[0 1]')
                 0*ones(1, size(x(:,:,3)')*[0 1]')];
subplot(2,1,2)
plot(t,u30)
```

```
grid
title('Input Response To r = [0  0  1  0]  Command')
ylabel('Pitch, Roll (deg), Thrust (N), Yaw (deg/s)')
xlabel('Time (seconds)')
legend('Pitch', 'Roll', 'Thrust', 'Yaw Rate')
pause

%
% YAW COMMAND
%
% Yaw: r = [0  0  0  1] yaw Command
%
figure;
subplot(2,1,1)
plot(t,y(:,:,4))
grid
title('Output Response To r = [0  0  0  1]  Command')
ylabel('Outputs')
xlabel('Time (seconds)')
legend('x', 'y', 'z', 'psi')
%
% Controls: r = [0  0  0  1] Yaw Command
%
u40 = [-G1 gy1]*[x(:,:,4)'
                 0*ones(1, size(x(:,:,4)')*[0 1]')
                 0*ones(1, size(x(:,:,4)')*[0 1]')
                 0*ones(1, size(x(:,:,4)')*[0 1]')
                 ones(1, size(x(:,:,4)')*[0 1]')];
subplot(2,1,2)
plot(t,u40)
grid
title('Input Response To r = [0  0  0  1]  Command')
ylabel('Pitch, Roll (deg), Thrust (N), Yaw (deg/s)')
xlabel('Time (seconds)')
legend('Pitch', 'Roll', 'Thrust', 'Yaw Rate')
pause


%% H-infinity Control design
P = ss(Ap1,Bp1,Cp1,Dp1);

% Bilinear transformation
% Move the poles at the origin to the right-half plane.
% This is to avoid pole-zero cancellation.
% By default, hinfsyn cancels out all stable poles (even near-imaginary
% poles).
p1=-1; p2=-1e20;
[apd,bpd,cpd,dpd]=bilin(Ap1,Bp1,Cp1,Dp1,1,'Sft_jw',[p2 p1]);
Pd=ss(apd,bpd,cpd,dpd);

% ...
    ************************************************************************
% Weights
% Standard first order weights are chosen
% See AAR's book
% Design 1
M11=10; w11=0.15; Eps11=0.001; M12=10; w12=15; Eps12=0.001; M13=10; ...
```

```
    w13=1.5; Eps13=0.001;
W11 = [tf([1/M11 w11], [1 w11*Eps11]) 0 0 0 ;
       0 tf([1/M11 w11], [1 w11*Eps11]) 0 0 ;
       0 0 tf([1/M11 w11], [1 w11*Eps11]) 0 ;
       0 0 0 tf([1/M12 w12], [1 w12*Eps12])];

M21=10; w21=1000; Eps21=0.1; M22=0.01; w22=1000; Eps22=0.1;
W21 = [M21 0 0 0;
       0 M21 0 0;
       0 0 M22 0;
       0 0 0 M21];

M31=10; w31=30; Eps31=0.001; M32=10; w32=150; Eps32=0.001; M33=10; ...
    w33=15; Eps33=0.001;
W31 = [tf([1 w31/M31], [Eps31 w31]) 0 0 0;
       0 tf([1 w31/M31], [Eps31 w31]) 0 0;
       0 0 tf([1 w31/M31], [Eps31 w31]) 0;
       0 0 0 tf([1 w32/M32], [Eps32 w32])];

% Design 2
M11=10; w11=0.15; Eps11=0.001; M12=10; w12=15; Eps12=0.001; M13=10; ...
    w13=1.5; Eps13=0.001;
W12 = [tf([1/M11 w11], [1 w11*Eps11]) 0 0 0 ;
       0 tf([1/M11 w11], [1 w11*Eps11]) 0 0 ;
       0 0 tf([1/M11 w11], [1 w11*Eps11]) 0 ;
       0 0 0 tf([1/M12 w12], [1 w12*Eps12])];

M21=0.001; w21=150; Eps21=0.001; M22=0.01; w22=150; Eps22=0.001;
W22 = [tf([1 w21*M21], [Eps21 w21]) 0 0 0;
       0 tf([1 w21*M21], [Eps21 w21]) 0 0;
       0 0 tf([1 w22*M22], [Eps22 w22]) 0;
       0 0 0 tf([1 w21*M21], [Eps21 w21])] ;

M31=10; w31=30; Eps31=0.001; M32=10; w32=150; Eps32=0.001; M33=10; ...
    w33=15; Eps33=0.001;
W32 = [tf([1 w31/M31], [Eps31 w31]) 0 0 0;
       0 tf([1 w31/M31], [Eps31 w31]) 0 0;
       0 0 tf([1 w31/M31], [Eps31 w31]) 0;
       0 0 0 tf([1 w32/M32], [Eps32 w32])];

% ...
    ************************************************************************
% Design 1 - Generalized plant
GenP=augw(Pd,W11,W21,W31);

% ...
    ************************************************************************
% Design 1 - Obtain controller using Mablab's hinfsyn command
Kd=hinfsyn(GenP);

% ...
    ************************************************************************
% Design 1 - Inverse bilinear transformation
[akd,bkd,ckd,dkd] = ssdata(Kd);
[ak,bk,ck,dk]=bilin(akd,bkd,ckd,dkd,-1,'Sft_jw',[p2 p1]);
K1=ss(ak,bk,ck,dk);
```

```
% ...
    ************************************************************************
% Design 2 - Generalized plant
GenP=augw(Pd,W12,W22,W32);

% ...
    ************************************************************************
% Design 2 - Obtain controller using Mablab's hinfsyn command
Kd=hinfsyn(GenP);

% ...
    ************************************************************************
% Design 2 - Inverse bilinear transformation
[akd,bkd,ckd,dkd] = ssdata(Kd);
[ak,bk,ck,dk]=bilin(akd,bkd,ckd,dkd,-1,'Sft_jw',[p2 p1]);
K2=ss(ak,bk,ck,dk);

% ...
    ************************************************************************
% Form closed loop maps
% f_CLTFM.m is a matlab function for computing OL and CL maps in a ...
    standard
% output feedback structure. This file must be in the current Matlab ...
    folder
[Lo1,Li1,So1,Si1,To1,Ti1,KS1,PS1] = f_CLTFM(P,K1);
[Lo2,Li2,So2,Si2,To2,Ti2,KS2,PS2] = f_CLTFM(P,K2);

% Closed Loop Poles
[Wn1,zeta1, clpoles1] = damp(To1)
[Wn2,zeta2, clpoles2] = damp(To2)

% PLOTS
wvec=logspace(-3,3,10000);

%************************************************************************
%
% OPEN LOOP FREQUENCY RESPONSE
figure;
sigma(Lo1,wvec, 'b');
hold on
sigma(Lo2,wvec, ':r');
title('Open Loop Singular Values: Error Signal')
grid
xlabel('Frequency (rad/sec)')
ylabel('Singular Values (dB)')

figure;
sigma(Li1,wvec, 'b');
hold on
sigma(Li2,wvec, ':r');
title('Open Loop Singular Values: Plant Input')
grid
xlabel('Frequency (rad/sec)')
ylabel('Singular Values (dB)')

%************************************************************************
%
```

```
% CLOSED LOOP FREQUENCY RESPONSE

figure;
sigma(So1,wvec, 'b');
hold on
sigma(So2,wvec, ':r');
title('Sensitivity: Error Signal')
grid
xlabel('Frequency (rad/sec)')
ylabel('Singular Values (dB)')

figure;
sigma(Si1,wvec, 'b');
hold on
sigma(Si2,wvec, ':r');
title('Sensitivity: Plant Input')
grid
xlabel('Frequency (rad/sec)')
ylabel('Singular Values (dB)')

figure;
sigma(To1,wvec, 'b');
hold on
sigma(To2,wvec, ':r');
title('Complementary Sensitivity: Plant Output')
grid
xlabel('Frequency (rad/sec)')
ylabel('Singular Values (dB)')

figure;
sigma(Ti1,wvec, 'b');
hold on
sigma(Ti2,wvec, ':r');
title('Complementary Sensitivity: Plant Input')
grid
xlabel('Frequency (rad/sec)')
ylabel('Singular Values (dB)')


%************************************************************************
%
% CLOSED LOOP TIME RESPONSE

t = [0:0.02:10];
[y1, t, x1] = step(To1,t);
[y2, t, x2] = step(To2,t);
%
% POSITION IN X-AXIS COMMAND
%
% x: r = [1  0  0  0] x-axis position Command
%
figure;
subplot(2,1,1);
plot(t,y1(:,:,1))
hold on
plot(t,y2(:,:,1), ':')
grid
```

```
title('Output Response To r = [1  0  0  0]  Command')
ylabel('Outputs')
xlabel('Time (seconds)')
legend('x', 'y', 'z', 'psi', 'x', 'y', 'z', 'psi')
hold off
%
% Controls: r = [1  0  0  0] x-axis position Command
%
[u1, t] = step(KS1,t);
[u2, t] = step(KS2,t);
u10=u1(:,:,1);
subplot(2,1,2);
plot(t,u10)
hold on
u10=u2(:,:,1);
plot(t,u10, ':')
grid
title('Input Response To r = [1  0  0  0]  Command')
ylabel('Controls')
xlabel('Time (seconds)')
legend('Pitch', 'Roll', 'Thrust', 'Yaw Rate', 'Pitch', 'Roll', ...
   'Thrust', 'Yaw Rate')
hold off
pause

%
% POSITION IN Y-AXIS COMMAND
%
% y: r = [0  1  0  0] y-axis position Command
%
figure;
subplot(2,1,1);
plot(t,y1(:,:,2))
hold on
plot(t,y2(:,:,2), ':')
grid
title('Output Response To r = [1  0  0  0]  Command')
ylabel('Outputs')
xlabel('Time (seconds)')
legend('x', 'y', 'z', 'psi', 'x', 'y', 'z', 'psi')
hold off
%
% Controls: r = [0  1  0  0] y-axis position Command
%
[u1, t] = step(KS1,t);
[u2, t] = step(KS2,t);
u20=u1(:,:,2);
subplot(2,1,2);
plot(t,u20)
hold on
u20=u2(:,:,2);
plot(t,u20, ':')
grid
title('Input Response To r = [0  1  0  0]  Command')
ylabel('Controls')
xlabel('Time (seconds)')
legend('Pitch', 'Roll', 'Thrust', 'Yaw Rate', 'Pitch', 'Roll', ...
```

```
    'Thrust', 'Yaw Rate')
hold off
pause

%
% POSITION IN Z-AXIS COMMAND
%
% z: r = [0  0  1  0] z-axis position Command
%
figure;
subplot(2,1,1);
plot(t,y1(:,:,3))
hold on
plot(t,y2(:,:,3), ':')
grid
title('Output Response To r = [0  0  1  0]  Command')
ylabel('Outputs')
xlabel('Time (seconds)')
legend('x', 'y', 'z', 'psi', 'x', 'y', 'z', 'psi')
hold off
% Controls: r = [0  0  1  0] z-axis position Command
%
[u1, t] = step(KS1,t);
[u2, t] = step(KS2,t);
u30=u1(:,:,3);
subplot(2,1,2);
plot(t,u30)
hold on
u30=u2(:,:,3);
plot(t,u30, ':')
grid
title('Input Response To r = [0  0  1  0]  Command')
ylabel('Controls')
xlabel('Time (seconds)')
legend('Pitch', 'Roll', 'Thrust', 'Yaw Rate', 'Pitch', 'Roll', ...
    'Thrust', 'Yaw Rate')
hold off
pause

%
% YAW COMMAND
%
% Yaw: r = [0  0  0  1] yaw Command
%
figure;
subplot(2,1,1);
plot(t,y1(:,:,4))
hold on
plot(t,y2(:,:,4), ':')
grid
title('Output Response To r = [0  0  0  1]  Command')
ylabel('Outputs')
xlabel('Time (seconds)')
legend('x', 'y', 'z', 'psi', 'x', 'y', 'z', 'psi')
hold off
% Controls: r = [0  0  0  1] yaw Command
%
```

```
[u1, t] = step(KS1,t);
[u2, t] = step(KS2,t);
u40=u1(:,:,4);
subplot(2,1,2);
plot(t,u40)
hold on
u40=u2(:,:,4);
plot(t,u40, ':')
grid
title('Input Response To r = [0   0   0   1]  Command')
ylabel('Controls')
xlabel('Time (seconds)')
legend('Pitch', 'Roll', 'Thrust', 'Yaw Rate', 'Pitch', 'Roll', ...
    'Thrust', 'Yaw Rate')
hold off
pause


%% LQG/LTRO Design
% Design or Form Compensator
%

% Augment Plant Input with Integrators, in order to acheive zero
% steady error to step inputs.
%
% Form Design Plant
ap = Ap1;
bp = Bp1;
cp = Cp1;
dp = Dp1;
a       = [zeros(nc,nc+ns); bp ap];
b       = [eye(nc,nc); zeros(ns,nc)];
c       = [zeros(no,nc) cp];
d       = [zeros(no,nc)];


winit  = -2;
wfin   =  2;
nwpts  = 200;
w      = logspace(winit,wfin,nwpts);   % Form vector of ...
   logarithmically spaced freq points
sv     = sigma(ss(ap, bp, cp, dp),w);
sv     = 20*log10(sv);
figure; semilogx(w, sv)
hold on
sv     = sigma(ss(a, b, c, d),w);
sv     = 20*log10(sv);
semilogx(w, sv, ':r')
%clear sv
title('Design Plant Singular Values')
grid
xlabel('Frequency (rad/sec)')
ylabel('Singular Values (dB)')
pause

%
% Use Kalman Filter to Design Target Loop (At Output)
```

```
%
s = j*0.0001;
ll = inv(cp*inv(s*eye(7)-ap)*bp); % Match at Low Freq
% lh = inv(s*eye(7)-ap)*bp*ll;
lh = cp'*inv(cp*cp');
l  = [ll;lh];

winit  = -2;
wfin   =  2;
nwpts  = 200;
w      = logspace(winit,wfin,nwpts);
gfolsv = sigma(ss(a, l, c, d),w);
gfolsv = 20*log10(gfolsv);
figure; semilogx(w, gfolsv)
%clear sv
title('G_{FOL} Singular Values')
grid
xlabel('Frequency (rad/sec)')
ylabel('Singular Values (dB)')
pause


mu  = 0.4;                          % Set Target Loop Bandwidth ...
   Parameter
%mu = 1;
%mu = 10;
%mu = 20;
x   = are(a',(1/mu)*c'*c,l*l');     % Solve Filter Algebraic Riccati ...
   Equation (FARE)
h   = (1/mu)*x*c';                  % Solve for the Kalman Filter ...
   Gain H


%
% Target Closed Loop Poles and Zeros
%
tclpoles = eig(a-h*c)
tclzeros = tzero(a-h*c,h,c, d)

%
% Target Open Loop SVD at s = j20
%
s                  = j*20
tol_s              = c*inv(s*eye(size(a))-a)*h + d
[tolu, tols, tolv ] = svd(tol_s)

%
% SVD Analysis yields:
%
%         - maximum singular value associated with theta_2 (light link)
%         - minimum singular value associated with theta_1 (heavy link)
%

%
% Target Open Loop Singular Values
%
winit  = -2;
```

```
wfin   =  2;
nwpts  = 200;
w      = logspace(winit,wfin,nwpts);   % Form vector of ...
   logarithmically spaced freq points
sv     = sigma(ss(a,h,c,d),w);
tlsv   = 20*log10(sv);
figure;semilogx(w, tlsv)
title('Target Open Loop Singular Values at Output')
grid
xlabel('Frequency (rad/sec)')
ylabel('Singular Values (dB)')
pause

%
% Target Sensitivity Singular Values
%
winit  = -2;
wfin   =  2;
nwpts  = 200;
w      = logspace(winit,wfin,nwpts);   % Form vector of ...
   logarithmically spaced freq points
sv     = sigma(ss(a-h*c,h,-c,eye(no,no)),w);
sv     = 20*log10(sv);
figure;semilogx(w, sv)
title('Target Sensitivity Singular Values at Output')
grid
xlabel('Frequency (rad/sec)')
ylabel('Singular Values (dB)')
pause

%
% Target Complementary Sensitivity Singular Values
%
winit  = -2;
wfin   =  2;
nwpts  = 200;
w      = logspace(winit,wfin,nwpts);   % Form vector of ...
   logarithmically spaced freq points
sv     = sigma(ss(a-h*c,h,c,d),w);
sv     = 20*log10(sv);
figure;semilogx(w, sv)
title('Target Comp Sensitivity Singular Values at Output')
grid
xlabel('Frequency (rad/sec)')
ylabel('Singular Values (dB)')
pause

%
% Target Closed Loop Singular Values
%
winit  = -2;
wfin   =  2;
nwpts  = 200;
w      = logspace(winit,wfin,nwpts);   % Form vector of ...
   logarithmically spaced freq points
ntacl = a-h*c;
ntbcl = h;
```

```
ntccl = c;
ntdcl = d;
sv     = sigma(ss(ntacl, ntbcl, ntccl, ntdcl),w);
sv     = 20*log10(sv);
figure;semilogx(w, sv)
title('Target Closed Loop Singular Values (r to y)')
grid
xlabel('Frequency (rad/sec)')
ylabel('T_{ry}, Singular Values (dB)')
pause

%
% Target Step Responses
%
tinit = 0;
tinc  = 0.005;
tfin  = 20.0;
t     = [tinit:tinc:tfin]';                    % Vector of uniformly ...
   spaced time points
r1    = [ones(size(t)) zeros(size(t)) zeros(size(t)) zeros(size(t))];
r2    = [zeros(size(t)) ones(size(t)) zeros(size(t)) zeros(size(t))];
r3    = [zeros(size(t)) zeros(size(t)) ones(size(t)) zeros(size(t))];
r4    = [zeros(size(t)) zeros(size(t)) zeros(size(t)) ones(size(t))];



%
% Loop Transfer Recovery at Output
%
q            = diag([0.1, 0.1, 0.1, 1, 100, 100, 100, 10, 10, 10, 10]);
rho          = 1e-9;                           % LQG/LTRO Recovery ...
   Parameter
[g, kp, clp] = lqr(a, b, q, rho*eye(nc,nc));  % Compute Control Gain ...
   Matrix G


% This is a MBC followed by an integrator bannk.
%
ak = [0*ones(nc,no)  g; zeros(ns+nc,no)  a-b*g-h*c];
bk = [zeros(nc,no); h];
ck = [eye(nc,no)  zeros(nc, ns+nc)];
dk = [zeros(no,no)];
%return



%
%*******************************************************************************
%*******************************************************************************
%*******************************************************************************
%
%

kpoles = eig(ak)                                % Compensator Poles
kzeros = tzero(ak, bk, ck, dk)                  % Compensator Zeros

winit  = -8;
```

```
wfin   = 4;
nwpts  = 200;
w      = logspace(winit,wfin,nwpts);
sv     = sigma(ss(ak, bk, ck, dk),w);
sv     = 20*log10(sv);
figure; semilogx(w, sv)
%clear sv
title('Compensator Singular Values')
grid
xlabel('Frequency (rad/sec)')
ylabel('Singular Values (dB)')
pause
%return

%
% Form Open Loop System
%
[al, bl, cl, dl ] = series(ak, bk, ck, dk, ap, bp, cp, dp);

olpoles = eig(al)                          % Open Loop Poles
olzeros = tzero(al,bl,cl,dl)               % Open Loop Zeros

winit   = -2;
wfin    = 2;
nwpts   = 200;
w       = logspace(winit,wfin,nwpts);
sv      = sigma(ss(al, bl, cl, dl), w);
sv      = 20*log10(sv);
figure; semilogx(w, sv, w, tlsv)
%clear sv
title('Open Loop Singular Values at Error (Recovered and Target)')
grid
xlabel('Frequency (rad/sec)')
ylabel('Singular Values (dB)')
pause


figure; semilogx(w, sv)
%clear sv
title('Open Loop Singular Values at Error')
grid
xlabel('Frequency (rad/sec)')
ylabel('Singular Values (dB)')
pause

%return

%
% Form Open Loop System
%
[ali, bli, cli, dli ] = series(ap, bp, cp, dp, ak, bk, ck, dk);

winit   = -2;
wfin    = 2;
nwpts   = 200;
w       = logspace(winit,wfin,nwpts);
sv      = sigma(ss(ali, bli, cli, dli ), w);
```

```
sv      = 20*log10(sv);
figure; semilogx(w, sv)
%clear sv
title('Open Loop Singular Values at Input')
grid
xlabel('Frequency (rad/sec)')
ylabel('Singular Values (dB)')
pause
%return

%
% Form Closed Loop System
%
acl     = al-bl*cl;
bcl     = bl;
ccl     = cl;
dcl     = dl;

clpoles = eig(acl)               % Closed Loop Poles
damp(clpoles)
clzeros = tzero(acl,bcl,ccl,dcl) % Closed Loop Zeros (r to y)

%
% Sensitivity at Error
%
winit   = -2;
wfin    = 2;
nwpts   = 200;
w       = logspace(winit,wfin,nwpts);
sv      = sigma(ss(acl, bcl, -ccl, eye(4)),w);
sv      = 20*log10(sv);
figure; semilogx(w, sv)
%clear sv
title('Sensitivity Singular Values at Error')
grid
xlabel('Frequency (rad/sec)')
ylabel('Singular Values (dB)')
pause
%return


%
% Sensitivity at Input
%
winit   = -2;
wfin    = 2;
nwpts   = 200;
w       = logspace(winit,wfin,nwpts);
sv      = sigma(ss(ali-bli*cli, bli, -cli, eye(4)),w);
sv      = 20*log10(sv);
figure; semilogx(w, sv)
%clear sv
title('Sensitivity Singular Values at Input')
grid
xlabel('Frequency (rad/sec)')
ylabel('Singular Values (dB)')
pause
```

```
%return


%
% Complementary Sensitivity
%
winit   = -2;
wfin    = 2;
nwpts   = 200;
w       = logspace(winit,wfin,nwpts);
sv      = sigma(ss(acl, bcl, ccl, dcl),w);
sv      = 20*log10(sv);
figure; semilogx(w, sv)
%clear sv
title('Complementary Sensitivity Singular Values at Ouput')
grid
xlabel('Frequency (rad/sec)')
ylabel('Singular Values (dB)')
pause
%return

%%
cls = ss(acl, bcl, ccl, dcl);
t = [0:0.02:20];
[y, t, x1] = step(cls,t);
%
% POSITION IN X-AXIS COMMAND
%
% x: r = [1  0  0  0] x-axis position Command
%
figure;
subplot(2,1,1);
plot(t,y(:,:,1))
grid
title('Output Response To r = [1  0  0  0]  Command')
ylabel('Outputs')
xlabel('Time (seconds)')
legend('x', 'y', 'z', 'psi')
%
% Controls: r = [1  0  0  0] x-axis position Command
%
[u, t] = step(ss(ak,bk,ck,dk),t);
u10=u(:,:,1);
subplot(2,1,2);
plot(t,u10)
grid
title('Input Response To r = [1  0  0  0]  Command')
ylabel('Controls')
xlabel('Time (seconds)')
legend('Pitch', 'Roll', 'Thrust', 'Yaw Rate')
pause

%
% POSITION IN Y-AXIS COMMAND
%
% y: r = [0  1  0  0] y-axis position Command
%
```

```
figure;
subplot(2,1,1);
plot(t,y(:,:,2))
grid
title('Output Response To r = [0  1  0  0]  Command')
ylabel('Outputs')
xlabel('Time (seconds)')
legend('x', 'y', 'z', 'psi')
%
% Controls: r = [0  1  0  0] y-axis position Command
%
[u, t] = step(ss(ak,bk,ck,dk),t);
u20=u(:,:,2);
subplot(2,1,2);
plot(t,u20)
grid
title('Input Response To r = [0  1  0  0]  Command')
ylabel('Controls')
xlabel('Time (seconds)')
legend('Pitch', 'Roll', 'Thrust', 'Yaw Rate')
pause

%
% POSITION IN Z-AXIS COMMAND
%
% z: r = [0  0  1  0] z-axis position Command
%
figure;
subplot(2,1,1);
plot(t,y(:,:,3))
grid
title('Output Response To r = [0  0  1  0]  Command')
ylabel('Outputs')
xlabel('Time (seconds)')
legend('x', 'y', 'z', 'psi')
%
% Controls: r = [0  0  1  0] z-axis position Command
%
[u, t] = step(ss(ak,bk,ck,dk),t);
u30=u(:,:,3);
subplot(2,1,2);
plot(t,u30)
grid
title('Input Response To r = [0  0  1  0]  Command')
ylabel('Controls')
xlabel('Time (seconds)')
legend('Pitch', 'Roll', 'Thrust', 'Yaw Rate')
pause

%
% YAW COMMAND
%
% Yaw: r = [0  0  0  1] yaw Command
%
figure;
subplot(2,1,1);
plot(t,y(:,:,4))
```

```
grid
title('Output Response To r = [0  0  0  1]  Command')
ylabel('Outputs')
xlabel('Time (seconds)')
legend('x', 'y', 'z', 'psi')
%
% Controls: r = [0  0  0  1] yaw Command
%
[u, t] = step(ss(ak,bk,ck,dk),t);
u40=u(:,:,4);
subplot(2,1,2);
plot(t,u40)
grid
title('Input Response To r = [0  0  0  1]  Command')
ylabel('Controls')
xlabel('Time (seconds)')
legend('Pitch', 'Roll', 'Thrust', 'Yaw Rate')
pause
```

APPENDIX B

FLIGHT CONTROLLER CODE

B.1 Quadrotor.ino

```
// Author: Abdullah Altawaitan
// Description: Teensy 3.2 code
// Check the Following:
// 1. The voltage readings
// 2. The Baud rate
// 3. The receiver channels
// 4. The control mode
#include "Communication.h"


Quadrotor Quad;
Communication Vive;
SBUS Radiolink(Serial2);

void setup()
{
  Teensy();
  Quad.MotorInit();
  Quad.SensorInit();
  Quad.FilterInit();
  digitalWrite(LEDRed, LOW);
  digitalWrite(LEDGreen, HIGH);
  Serial.println("Now Ready!");
  Quad.loop_timer = micros();
}


void loop()
{
  // outerCounter for 100Hz
  Quad.outerCounter++;
  Receiver();
  Quad.ArmingState();
  Quad.BatteryVoltageCheck();
  Quad.Estimation(1);
  Vive.ROS_Send(&Quad);
  Quad.Receiver();
  Quad.Control(2);
  Quad.DifferentialFlatness();
  Quad.AttitudeControl();
  Quad.GenerateMotorCommands();
  Quad.LoopCounter();
}

void Teensy()
{
  Serial.begin(115200);
  Serial1.begin(115200);
  Radiolink.begin();
  pinMode(LEDRed, OUTPUT);
  pinMode(LEDGreen, OUTPUT);
  digitalWrite(LEDGreen, LOW);
  digitalWrite(LEDRed, LOW);
```

```
  Quad.channel.CH1 = 1000;
  Quad.channel.CH2 = 1000;
  Quad.channel.CH3 = 250;
  Quad.channel.CH4 = 1000;
  Quad.channel.CH5 = 250;
  Quad.channel.CH6 = 250;
  Quad.QuadrotorState = 0;
}

void Receiver()
{
  if (Radiolink.read(Quad.channels, &Quad.failSafe, &Quad.lostFrames))
  {
    Quad.channel.CH1 = Quad.channels[0];
    Quad.channel.CH2 = Quad.channels[1];
    Quad.channel.CH3 = Quad.channels[2];
    Quad.channel.CH4 = Quad.channels[3];
    Quad.channel.CH5 = Quad.channels[4];
    Quad.channel.CH6 = Quad.channels[5];
    Quad.channel.CH7 = Quad.channels[6];
    Quad.channel.CH8 = Quad.channels[7];
    Quad.channel.CH9 = Quad.channels[8];
    Quad.channel.CH10 = Quad.channels[9];
  }
}

void serialEvent1()
{
  Vive.ROS_Receive(&Quad);
}
```

B.2 Quadrotor.h

```
// Author: Abdullah Altawaitan

#include "i2c_t3.h"
#include "EEPROM.h"
#include "SBUS.h"
#include "Arduino.h"
#include "Quaternion.h"
#include "IMU.h"
#include <cmath>
#include <SoftwareSerial.h>


// Teensy 3.2 pins
#define MOTOR1 20
#define MOTOR2 6
#define MOTOR3 21
#define MOTOR4 5
#define CHANNEL1 23
#define CHANNEL2 22
#define CHANNEL3 17
#define CHANNEL4 16
#define CHANNEL5 24
```

```
#define CHANNEL6 26
#define CHANNEL7 27
#define CHANNEL8 28
#define LED 13
#define LEDRed 31
#define LEDGreen 33

// Motor parameters
#define MIN_MOTOR_LEVEL 1055
#define MAX_MOTOR_LEVEL 1550
#define OFF_MOTOR_LEVEL 950


// AHRS
#define twoKi (2.0f * 0.25f)
#define twoKp (2.0f * 0.5f)

// Quadrotor modes
#define START_MODE 0
#define TRANS_MODE 1
#define ARMING_MODE 2
#define TEMP_MODE 3
#define DISARMING_MODE 4

// Quadrotor Parameters
#define mass 0.647
#define armlength 0.1215
#define k_f 0.000001518 // Thrust coefficient
#define k_m 0.00000001843 // Torque coefficient
#define kappa 0.0127 // Ratio between thrust [N] and torque due to ...
   drag [N m] kappa = torque/thrust
#define Ixx 0.002
#define PWM_FACTOR 26.214 // 65535.0 / 2500.0

// Loop
#define FREQ 400
#define dt 0.0025 // Period for 400Hz
#define dtOuter 0.01 // Period for 100Hz
#define dtMicroseconds 2500

// Inner-loop PID parameters
#define kpPQRx 0.1
#define kdPQRx 0.003
#define kiPQRx 0.5
#define kpPQRy 0.1
#define kdPQRy 0.003
#define kiPQRy 0.5
#define kpPQRz 0.1
#define kdPQRz 0.001
#define kiPQRz 0.2
// Outer-loop PID parameters
#define kpx 7
#define kdx 0
#define kix 0
#define kpy 7
#define kdy 0
#define kiy 0
```

```
#define kpz 9
#define kdz 0
#define kiz 0

class Quadrotor
{
public:
  struct _STATE {
    float x, y, z, phi, theta, psi, xdot, ydot, zdot, p, q, r;
    Quaternion quaternion;
  };
  struct _XYZ {
    float x, y, z;
  };
  struct _GYRO {
    _XYZ raw, tempOffset, gyroOffset, calibrated, filtered;
  };
  struct _ACCEL {
    _XYZ raw, afterOffset, calibrated, filtered;
  };
  struct _FILTER {
    float b0, b1, b2, a1, a2, element0, element1, element2;
  };
  struct _Q {
    float w, x, y, z;
  };
  struct _TRIG {
    float phi_sin, theta_sin, psi_sin, phi_cos, theta_cos, psi_cos;
  };
  struct _CH {
    short CH1, CH2, CH3, CH4, CH5, CH6, CH7, CH8, CH9, CH10;
  };
  struct _ERROR {
    float p, q, r, p_prev1, p_prev2, p_prev3, q_prev1, q_prev2, ...
       q_prev3, r_prev1, r_prev2, r_prev3, p_integral, q_integral,  ...
       r_integral;
    float phi, theta, psi, phi_prev1, theta_prev1, psi_prev1, ...
       phi_integral, theta_integral,  psi_integral;
    float xdot, xdot_prev1, xdot_prev2, xdot_prev3, xdot_integral, ...
       ydot, ydot_prev1, ydot_prev2, ydot_prev3, ydot_integral, ...
       zdot, zdot_prev1, zdot_prev2, zdot_prev3, zdot_integral;
    float x, y, z, x_integral, y_integral, z_integral;
  };
  struct _Acc_Cali {
    int16_t accel_raw[6][3], accel_timer = 0;
    int32_t accel_calitmpx, accel_calitmpy, accel_calitmpz;
    float accel_offset[3], a[3][3], T[3][3];
    float g = 8192; // for +-4g range
  };
  struct _CONTROLS {
    float current, prev1, prev2, prev3; // U[n], U[n-1], U[n-2], U[n-3]
  };
  struct _W {
    float input, input_prev1, output, output_prev1; // W = (az + ...
       a)/(z - b) first-order prefilter W = c/(s + c)
  };
```

```
// Sensor variables
IMU imu;
_GYRO gyro;
_ACCEL accel;
_FILTER gyroFilterParameterX, gyroFilterParameterY, ...
    gyroFilterParameterZ, accelFilterParameterX, ...
    accelFilterParameterY, accelFilterParameterZ;
_Acc_Cali Acc_Cali;
_Q q = {1.0, 0.0, 0.0, 0.0};
_XYZ gyroAngle;
_XYZ accelAngle;
int8_t gyro_calibration_done = 0;
int8_t accel_calibration_done = 0;
short gyro_calibration_counter = 0;
float GyroCollection[3] = {0, 0, 0};
float integralFBx = 0.0f,  integralFBy = 0.0f, integralFBz = 0.0f;
Quaternion quat;
float temperature;


// Quadrotor variables
_STATE X;
_STATE Xdes;
_ERROR error;
_TRIG trig;
int16_t throttle;
uint8_t QuadrotorState = START_MODE;
float voltage;
unsigned long loop_timer;
uint8_t blink = 0;
uint8_t blinkCounter = 0;
int flight_mode = 0;

// Control variables
_CONTROLS U1, U2, U3, U4; // Inner loop
_CONTROLS U1des;
_W Wp, Wq, Wr, Wxdot, Wydot, Wzdot; // Prefilter
uint8_t outerCounter = 0;
int8_t control_method = 0;
int control_method_counter = 0;


// Receiver variables
_CH channel;
uint16_t channels[16];
uint8_t failSafe;
uint16_t lostFrames = 0;
bool lastChannel1, lastChannel2, lastChannel3, lastChannel4, ...
    lastChannel5;
unsigned long receiverChannelTimer1, receiverChannelTimer2, ...
    receiverChannelTimer3, receiverChannelTimer4, ...
    receiverChannelTimer5;
float RCYawRate;

// Motor variables
float PWM1, PWM2, PWM3, PWM4;
float volt1, volt2, volt3, volt4;
```

```
float omega1, omega2, omega3, omega4;
float omega1Squared, omega2Squared, omega3Squared, omega4Squared;

// Xbee Zigbee variables
const byte numChars = 32;
char receivedChars[32];
char tempChars[32];
float float1 = 0.0;
float float2 = 0.0;
float float3 = 0.0;
boolean newData = false;
uint8_t Xbee_couter = 0;
uint8_t Xbee_couter2 = 0;
uint16_t DataTimer = 0;

// HTC Vive
uint8_t Xbee_data[500];
int16_t Xbee_length;
short Xbee_command;
float moving_average[13][5], sum_average[13];


// Methods
void MotorInit(void);
void SensorInit(void);
void IMUread(void);
void GyroCalibration(void);
void AccelCalibration(uint8_t point);
void AccelOffsetRead(void);
void FilterInit(void);
void Estimation(int8_t method);
void NonlinearComplementaryFilter(void);
void AHRS(void);
void MahonyAHRS(void);
void MahonyAHRSUpdate(float gx, float gy, float gz, float ax, ...
    float ay, float az);
void ArmingState(void);
void BatteryVoltageCheck(void);
void Receiver(void);
void Control(int8_t method);
void ThrottleControl(void);
void AttitudeControl(void);
void AngularRateControl(void);
void AltitudeControl(void);
void PositionControl(void);
void VelocityControl(void);
void DifferentialFlatness(void);
void GenerateMotorCommands(void);
void MotorRun(void);
void SecondOrderLowPassFilter(float sample_freq, float ...
    cutoff_freq, struct _FILTER *input_IIR);
void LoopCounter(void);
float SecondOrderLowPassFilterApply(float cutoff_freq, float ...
    sample, struct _FILTER *input_IIR);
float CONSTRAIN(float x, float min, float max);
float invSqrt(float number);
```

```
private:

};
```

B.3 Quadrotor.cpp

```
#include "Quadrotor.h"

// Motor initialization
void Quadrotor::MotorInit(void)
{
  voltage = (float)analogRead(A14) * 0.019586;
  pinMode(MOTOR1, OUTPUT);
  pinMode(MOTOR2, OUTPUT);
  pinMode(MOTOR3, OUTPUT);
  pinMode(MOTOR4, OUTPUT);
  analogWriteFrequency(MOTOR1, 400);
  analogWriteFrequency(MOTOR2, 400);
  analogWriteFrequency(MOTOR3, 400);
  analogWriteFrequency(MOTOR4, 400);
  analogWriteResolution(16);

  // Motors initialization
  float PWM = PWM_FACTOR * OFF_MOTOR_LEVEL;
  analogWrite(MOTOR1, PWM);
  analogWrite(MOTOR2, PWM);
  analogWrite(MOTOR3, PWM);
  analogWrite(MOTOR4, PWM);
}

// Sensor initialization and calibration
// Source: Pololu Robotics and Electronics ...
    (https://github.com/pololu/lsm303-arduino)
// Source: Andrea Vitali (DT0053 Design tip - 6-point tumble sensor ...
    calibration)
void Quadrotor::SensorInit(void)
{
  Wire.begin();
  Wire.setRate(I2C_RATE_2000);
  Wire1.begin();
  Wire1.setRate(I2C_RATE_2000);

  // Gyroscope initialization
  imu.Init();

  // Sensor calibration
  Serial.println("Sensor Calibration...");
  Serial.println("Place Quadrotor on level.");
  delay(1000);
  // Accelerometer calibration
  Serial.println("If you enter 'a', the calibration will get started.");
  delay(3000);
  char incomingByte = Serial.read();
```

```
if (incomingByte == 'a')
{
  Serial.println("On Level, and wait for 10 seconds");
  delay(10000);
  while (accel_calibration_done == 0)
  {
    Quadrotor::AccelCalibration(4);
  }
  accel_calibration_done = 0;

  Serial.println("On Back, and wait for 10 seconds");
  delay(10000);
  while (accel_calibration_done == 0)
  {
    Quadrotor::AccelCalibration(5);
  }
  accel_calibration_done = 0;

  Serial.println("Right Wing Up, and wait for 10 seconds");
  delay(10000);
  while (accel_calibration_done == 0)
  {
    Quadrotor::AccelCalibration(3);
  }
  accel_calibration_done = 0;

  Serial.println("Left Wing Up, and wait for 10 seconds");
  delay(10000);
  while (accel_calibration_done == 0)
  {
    Quadrotor::AccelCalibration(2);
  }
  accel_calibration_done = 0;

  Serial.println("Nose Up, and wait for 10 seconds");
  delay(10000);
  while (accel_calibration_done == 0)
  {
    Quadrotor::AccelCalibration(0);
  }
  accel_calibration_done = 0;

  Serial.println("Nose Down, and wait for 10 seconds");
  delay(10000);
  while (accel_calibration_done == 0)
  {
    Quadrotor::AccelCalibration(1);
  }
  accel_calibration_done = 1;
}

Quadrotor::AccelOffsetRead();
accel_calibration_done = 1;
while (gyro_calibration_done == 0 && gyro_calibration_counter <= 1500)
{
  gyro_calibration_counter++;
  GyroCalibration();
```

```
  }
  Serial.println("Sensor Calibration... Done!");
}

// Read Gyroscope Data
// Source: Shi Lu (https://github.com/ragewrath/Mark3-Copter-Pilot)
void Quadrotor::IMUread(void)
{
  imu.read();
  accel.raw.x = imu.a.x;
  accel.raw.y = imu.a.y;
  accel.raw.z = imu.a.z;
  gyro.raw.x = imu.g.x;
  gyro.raw.y = imu.g.y;
  gyro.raw.z = imu.g.z;

  // Sensor Thermal Compensation
  float temp, temp2, temp3;
  temperature = (float) imu.temp / 340.0 + 36.53;
  temp = temperature;
  temp = Quadrotor::CONSTRAIN(temp, 22.5, 55.0);
  temp2 = temp * temp;
  temp3 = temp * temp * temp;
  gyro.tempOffset.x = 0.000263 * temp3 - 0.03098 * temp2 + 0.03939 * ...
     temp - 29.4;
  gyro.tempOffset.y = -0.0004279 * temp3 + 0.05322 * temp2 - 2.941 * ...
     temp + 80.16;
  gyro.tempOffset.z = 0.0004163 * temp3 - 0.0332 * temp2 + 0.6652 * ...
     temp + 23.3;
  gyro.calibrated.x = (gyro.raw.x - gyro.tempOffset.x);
  gyro.calibrated.y = (gyro.raw.y - gyro.tempOffset.y);
  gyro.calibrated.z = (gyro.raw.z - gyro.tempOffset.z);

  if (gyro_calibration_done == 1)
  {
    gyro.calibrated.x = gyro.calibrated.x - gyro.gyroOffset.x;
    gyro.calibrated.y = gyro.calibrated.y - gyro.gyroOffset.y;
    gyro.calibrated.z = gyro.calibrated.z - gyro.gyroOffset.z;

    gyro.filtered.x = Quadrotor::SecondOrderLowPassFilterApply(30.0, ...
       gyro.calibrated.x, &gyroFilterParameterX);
    gyro.filtered.y = Quadrotor::SecondOrderLowPassFilterApply(30.0, ...
       -gyro.calibrated.y, &gyroFilterParameterY);
    gyro.filtered.z = Quadrotor::SecondOrderLowPassFilterApply(30.0, ...
       -gyro.calibrated.z, &gyroFilterParameterZ);

    float p_deg = gyro.filtered.x / 32.768;
    float q_deg = gyro.filtered.y / 32.768;
    float r_deg = gyro.filtered.z / 32.768;

    X.p = p_deg * (PI/180);
    X.q = q_deg * (PI/180);
    X.r = r_deg * (PI/180);

    gyroAngle.x += p_deg * dt;
    gyroAngle.y += q_deg * dt;
    gyroAngle.z += r_deg * dt;
```

```
  }

  if (accel_calibration_done == 1)
  {
    accel.afterOffset.x = (float)accel.raw.x - Acc_Cali.accel_offset[0];
    accel.afterOffset.y = (float)accel.raw.y - Acc_Cali.accel_offset[1];
    accel.afterOffset.z = (float)accel.raw.z - Acc_Cali.accel_offset[2];

    accel.calibrated.x = accel.afterOffset.x *  Acc_Cali.T[0][0] + ...
       accel.afterOffset.y *  Acc_Cali.T[1][0] + ...
       accel.afterOffset.z *  Acc_Cali.T[2][0];
    accel.calibrated.y = accel.afterOffset.x *  Acc_Cali.T[0][1] + ...
       accel.afterOffset.y *  Acc_Cali.T[1][1] + ...
       accel.afterOffset.z *  Acc_Cali.T[2][1];
    accel.calibrated.z = accel.afterOffset.x *  Acc_Cali.T[0][2] + ...
       accel.afterOffset.y *  Acc_Cali.T[1][2] + ...
       accel.afterOffset.z *  Acc_Cali.T[2][2];

    accel.filtered.x = ...
       Quadrotor::SecondOrderLowPassFilterApply(30.0, ...
       accel.calibrated.x, &accelFilterParameterX);
    accel.filtered.y = ...
       Quadrotor::SecondOrderLowPassFilterApply(30.0, ...
       accel.calibrated.y, &accelFilterParameterY);
    accel.filtered.z = ...
       Quadrotor::SecondOrderLowPassFilterApply(30.0, ...
       accel.calibrated.z, &accelFilterParameterZ);
  }
}

// Gyroscope calibration
// Source: Pololu Robotics and Electronics ...
   (https://github.com/pololu/lsm303-arduino)
// Source: Andrea Vitali (DT0053 Design tip - 6-point tumble sensor ...
   calibration)
void Quadrotor::GyroCalibration(void)
{
  if (gyro_calibration_counter > 400 && gyro_calibration_counter < 1001)
  {
    Quadrotor::IMUread();
    GyroCollection[0] += gyro.calibrated.x;
    GyroCollection[1] += gyro.calibrated.y;
    GyroCollection[2] += gyro.calibrated.z;
  }
  if (gyro_calibration_counter == 1001)
  {
    gyro.gyroOffset.x = GyroCollection[0] / 600;
    gyro.gyroOffset.y = GyroCollection[1] / 600;
    gyro.gyroOffset.z = GyroCollection[2] / 600;
    GyroCollection[0] = 0;
    GyroCollection[1] = 0;
    GyroCollection[2] = 0;
    gyro_calibration_done = 1;
  }
}

// Accelerometer Calibration
```

```
// Source: Andrea Vitali (DT0053 Design tip - 6-point tumble sensor ...
    calibration)
// Source: Shi Lu (https://github.com/ragewrath/Mark3-Copter-Pilot)
void Quadrotor::AccelCalibration(uint8_t point)
{
  if (Acc_Cali.accel_timer == 100)
  {
    Acc_Cali.accel_raw[point][0] = Acc_Cali.accel_calitmpx / 100;
    Acc_Cali.accel_raw[point][1] = Acc_Cali.accel_calitmpy / 100;
    Acc_Cali.accel_raw[point][2] = Acc_Cali.accel_calitmpz / 100;
    Acc_Cali.accel_calitmpx = 0;
    Acc_Cali.accel_calitmpy = 0;
    Acc_Cali.accel_calitmpz = 0;
    Acc_Cali.accel_timer = 0;
    EEPROM.write(100 + 6 * point + 1, Acc_Cali.accel_raw[point][0] & ...
       0b11111111);
    EEPROM.write(100 + 6 * point + 2, Acc_Cali.accel_raw[point][0] ...
       >> 8);
    EEPROM.write(100 + 6 * point + 3, Acc_Cali.accel_raw[point][1] & ...
       0b11111111);
    EEPROM.write(100 + 6 * point + 4, Acc_Cali.accel_raw[point][1] ...
       >> 8);
    EEPROM.write(100 + 6 * point + 5, Acc_Cali.accel_raw[point][2] & ...
       0b11111111);
    EEPROM.write(100 + 6 * point + 6, Acc_Cali.accel_raw[point][2] ...
       >> 8);
    accel_calibration_done = 1;
  }
  else
  {
    Quadrotor::IMUread();
    Acc_Cali.accel_calitmpx += accel.raw.x;
    Acc_Cali.accel_calitmpy += accel.raw.y;
    Acc_Cali.accel_calitmpz += accel.raw.z;
    Acc_Cali.accel_timer++;
  }
}

// Accelerometer Offset Calculation
// Source: Andrea Vitali (DT0053 Design tip - 6-point tumble sensor ...
    calibration)
// Source: Shi Lu (https://github.com/ragewrath/Mark3-Copter-Pilot)
void Quadrotor::AccelOffsetRead(void)
{
  uint8_t point;
  for (point = 0; point < 6; point++)
  {
    Acc_Cali.accel_raw[point][0] = (EEPROM.read(100 + 6 * point + 2) ...
       << 8) | EEPROM.read(100 + 6 * point + 1);
    Acc_Cali.accel_raw[point][1] = (EEPROM.read(100 + 6 * point + 4) ...
       << 8) | EEPROM.read(100 + 6 * point + 3);
    Acc_Cali.accel_raw[point][2] = (EEPROM.read(100 + 6 * point + 6) ...
       << 8) | EEPROM.read(100 + 6 * point + 5);
  }

  Acc_Cali.accel_offset[0] = (float)(Acc_Cali.accel_raw[0][0] + ...
     Acc_Cali.accel_raw[1][0]) / 2.0;
```

```
Acc_Cali.accel_offset[1] = (float)(Acc_Cali.accel_raw[2][1] + ...
    Acc_Cali.accel_raw[3][1]) / 2.0;
Acc_Cali.accel_offset[2] = (float)(Acc_Cali.accel_raw[4][2] + ...
    Acc_Cali.accel_raw[5][2]) / 2.0;

Serial.println("------------------------");
Serial.println("Accelerometer Offset: ");
Serial.println("------------------------");
Serial.print(Acc_Cali.accel_offset[0]);
Serial.print(", ");
Serial.print(Acc_Cali.accel_offset[1]);
Serial.print(", ");
Serial.println(Acc_Cali.accel_offset[2]);

for (point = 0; point < 3; point++)
  Acc_Cali.a[0][point] = (float)Acc_Cali.accel_raw[0][point] - ...
      Acc_Cali.accel_offset[point];
for (point = 0; point < 3; point++)
  Acc_Cali.a[1][point] = (float)Acc_Cali.accel_raw[2][point] - ...
      Acc_Cali.accel_offset[point];
for (point = 0; point < 3; point++)
  Acc_Cali.a[2][point] = (float)Acc_Cali.accel_raw[4][point] - ...
      Acc_Cali.accel_offset[point];

Acc_Cali.T[0][0] = (Acc_Cali.g * (Acc_Cali.a[1][1] * ...
    Acc_Cali.a[2][2] - Acc_Cali.a[1][2] * Acc_Cali.a[2][1])) / ...
    (Acc_Cali.a[0][0] * Acc_Cali.a[1][1] * Acc_Cali.a[2][2] - ...
    Acc_Cali.a[0][0] * Acc_Cali.a[1][2] * Acc_Cali.a[2][1] - ...
    Acc_Cali.a[0][1] * Acc_Cali.a[1][0] * Acc_Cali.a[2][2] + ...
    Acc_Cali.a[0][1] * Acc_Cali.a[1][2] * Acc_Cali.a[2][0] + ...
    Acc_Cali.a[0][2] * Acc_Cali.a[1][0] * Acc_Cali.a[2][1] - ...
    Acc_Cali.a[0][2] * Acc_Cali.a[1][1] * Acc_Cali.a[2][0]);
Acc_Cali.T[0][1] = -(Acc_Cali.g * (Acc_Cali.a[0][1] * ...
    Acc_Cali.a[2][2] - Acc_Cali.a[0][2] * Acc_Cali.a[2][1])) / ...
    (Acc_Cali.a[0][0] * Acc_Cali.a[1][1] * Acc_Cali.a[2][2] - ...
    Acc_Cali.a[0][0] * Acc_Cali.a[1][2] * Acc_Cali.a[2][1] - ...
    Acc_Cali.a[0][1] * Acc_Cali.a[1][0] * Acc_Cali.a[2][2] + ...
    Acc_Cali.a[0][1] * Acc_Cali.a[1][2] * Acc_Cali.a[2][0] + ...
    Acc_Cali.a[0][2] * Acc_Cali.a[1][0] * Acc_Cali.a[2][1] - ...
    Acc_Cali.a[0][2] * Acc_Cali.a[1][1] * Acc_Cali.a[2][0]);
Acc_Cali.T[0][2] = (Acc_Cali.g * (Acc_Cali.a[0][1] * ...
    Acc_Cali.a[1][2] - Acc_Cali.a[0][2] * Acc_Cali.a[1][1])) / ...
    (Acc_Cali.a[0][0] * Acc_Cali.a[1][1] * Acc_Cali.a[2][2] - ...
    Acc_Cali.a[0][0] * Acc_Cali.a[1][2] * Acc_Cali.a[2][1] - ...
    Acc_Cali.a[0][1] * Acc_Cali.a[1][0] * Acc_Cali.a[2][2] + ...
    Acc_Cali.a[0][1] * Acc_Cali.a[1][2] * Acc_Cali.a[2][0] + ...
    Acc_Cali.a[0][2] * Acc_Cali.a[1][0] * Acc_Cali.a[2][1] - ...
    Acc_Cali.a[0][2] * Acc_Cali.a[1][1] * Acc_Cali.a[2][0]);

Acc_Cali.T[1][0] = -(Acc_Cali.g * (Acc_Cali.a[1][0] * ...
    Acc_Cali.a[2][2] - Acc_Cali.a[1][2] * Acc_Cali.a[2][0])) / ...
    (Acc_Cali.a[0][0] * Acc_Cali.a[1][1] * Acc_Cali.a[2][2] - ...
    Acc_Cali.a[0][0] * Acc_Cali.a[1][2] * Acc_Cali.a[2][1] - ...
    Acc_Cali.a[0][1] * Acc_Cali.a[1][0] * Acc_Cali.a[2][2] + ...
    Acc_Cali.a[0][1] * Acc_Cali.a[1][2] * Acc_Cali.a[2][0] + ...
    Acc_Cali.a[0][2] * Acc_Cali.a[1][0] * Acc_Cali.a[2][1] - ...
    Acc_Cali.a[0][2] * Acc_Cali.a[1][1] * Acc_Cali.a[2][0]);
```

```
Acc_Cali.T[1][1] = (Acc_Cali.g * (Acc_Cali.a[0][0] * ...
    Acc_Cali.a[2][2] - Acc_Cali.a[0][2] * Acc_Cali.a[2][0])) / ...
    (Acc_Cali.a[0][0] * Acc_Cali.a[1][1] * Acc_Cali.a[2][2] - ...
    Acc_Cali.a[0][0] * Acc_Cali.a[1][2] * Acc_Cali.a[2][1] - ...
    Acc_Cali.a[0][1] * Acc_Cali.a[1][0] * Acc_Cali.a[2][2] + ...
    Acc_Cali.a[0][1] * Acc_Cali.a[1][2] * Acc_Cali.a[2][0] + ...
    Acc_Cali.a[0][2] * Acc_Cali.a[1][0] * Acc_Cali.a[2][1] - ...
    Acc_Cali.a[0][2] * Acc_Cali.a[1][1] * Acc_Cali.a[2][0]);
Acc_Cali.T[1][2] = -(Acc_Cali.g * (Acc_Cali.a[0][0] * ...
    Acc_Cali.a[1][2] - Acc_Cali.a[0][2] * Acc_Cali.a[1][0])) / ...
    (Acc_Cali.a[0][0] * Acc_Cali.a[1][1] * Acc_Cali.a[2][2] - ...
    Acc_Cali.a[0][0] * Acc_Cali.a[1][2] * Acc_Cali.a[2][1] - ...
    Acc_Cali.a[0][1] * Acc_Cali.a[1][0] * Acc_Cali.a[2][2] + ...
    Acc_Cali.a[0][1] * Acc_Cali.a[1][2] * Acc_Cali.a[2][0] + ...
    Acc_Cali.a[0][2] * Acc_Cali.a[1][0] * Acc_Cali.a[2][1] - ...
    Acc_Cali.a[0][2] * Acc_Cali.a[1][1] * Acc_Cali.a[2][0]);

Acc_Cali.T[2][0] = (Acc_Cali.g * (Acc_Cali.a[1][0] * ...
    Acc_Cali.a[2][1] - Acc_Cali.a[1][1] * Acc_Cali.a[2][0])) / ...
    (Acc_Cali.a[0][0] * Acc_Cali.a[1][1] * Acc_Cali.a[2][2] - ...
    Acc_Cali.a[0][0] * Acc_Cali.a[1][2] * Acc_Cali.a[2][1] - ...
    Acc_Cali.a[0][1] * Acc_Cali.a[1][0] * Acc_Cali.a[2][2] + ...
    Acc_Cali.a[0][1] * Acc_Cali.a[1][2] * Acc_Cali.a[2][0] + ...
    Acc_Cali.a[0][2] * Acc_Cali.a[1][0] * Acc_Cali.a[2][1] - ...
    Acc_Cali.a[0][2] * Acc_Cali.a[1][1] * Acc_Cali.a[2][0]);
Acc_Cali.T[2][1] = -(Acc_Cali.g * (Acc_Cali.a[0][0] * ...
    Acc_Cali.a[2][1] - Acc_Cali.a[0][1] * Acc_Cali.a[2][0])) / ...
    (Acc_Cali.a[0][0] * Acc_Cali.a[1][1] * Acc_Cali.a[2][2] - ...
    Acc_Cali.a[0][0] * Acc_Cali.a[1][2] * Acc_Cali.a[2][1] - ...
    Acc_Cali.a[0][1] * Acc_Cali.a[1][0] * Acc_Cali.a[2][2] + ...
    Acc_Cali.a[0][1] * Acc_Cali.a[1][2] * Acc_Cali.a[2][0] + ...
    Acc_Cali.a[0][2] * Acc_Cali.a[1][0] * Acc_Cali.a[2][1] - ...
    Acc_Cali.a[0][2] * Acc_Cali.a[1][1] * Acc_Cali.a[2][0]);
Acc_Cali.T[2][2] = (Acc_Cali.g * (Acc_Cali.a[0][0] * ...
    Acc_Cali.a[1][1] - Acc_Cali.a[0][1] * Acc_Cali.a[1][0])) / ...
    (Acc_Cali.a[0][0] * Acc_Cali.a[1][1] * Acc_Cali.a[2][2] - ...
    Acc_Cali.a[0][0] * Acc_Cali.a[1][2] * Acc_Cali.a[2][1] - ...
    Acc_Cali.a[0][1] * Acc_Cali.a[1][0] * Acc_Cali.a[2][2] + ...
    Acc_Cali.a[0][1] * Acc_Cali.a[1][2] * Acc_Cali.a[2][0] + ...
    Acc_Cali.a[0][2] * Acc_Cali.a[1][0] * Acc_Cali.a[2][1] - ...
    Acc_Cali.a[0][2] * Acc_Cali.a[1][1] * Acc_Cali.a[2][0]);


Serial.println("------------------------");
Serial.println("Accelerometer Calibration: ");
Serial.println("------------------------");
Serial.print(Acc_Cali.T[0][0]);
Serial.print(", ");
Serial.print(Acc_Cali.T[0][1]);
Serial.print(", ");
Serial.println(Acc_Cali.T[0][2]);
Serial.print(Acc_Cali.T[1][0]);
Serial.print(", ");
Serial.print(Acc_Cali.T[1][1]);
Serial.print(", ");
Serial.println(Acc_Cali.T[1][2]);
Serial.print(Acc_Cali.T[2][0]);
```

```
  Serial.print(", ");
  Serial.print(Acc_Cali.T[2][1]);
  Serial.print(", ");
  Serial.println(Acc_Cali.T[2][2]);
  Serial.println("------------------------");

}

// Second Order Low Pass Filter for Gyroscope data
void Quadrotor::FilterInit(void)
{
  Quadrotor::SecondOrderLowPassFilter(400.0, 30.0, ...
      &gyroFilterParameterX);
  Quadrotor::SecondOrderLowPassFilter(400.0, 30.0, ...
      &gyroFilterParameterY);
  Quadrotor::SecondOrderLowPassFilter(400.0, 30.0, ...
      &gyroFilterParameterZ);
  Quadrotor::SecondOrderLowPassFilter(400.0, 30.0, ...
      &accelFilterParameterX);
  Quadrotor::SecondOrderLowPassFilter(400.0, 30.0, ...
      &accelFilterParameterY);
  Quadrotor::SecondOrderLowPassFilter(400.0, 30.0, ...
      &accelFilterParameterZ);
}

// Sensor Estimation Methods
// Method 1: AHRS
// Method 2: Nonlinear Complementary Filter
void Quadrotor::Estimation(int8_t method)
{
  // Choosing one of the methods
  if (method == 1)
  {
    Quadrotor::AHRS();
  }
  else if (method == 2)
  {
    Quadrotor::NonlinearComplementaryFilter();
  }
  else
  {
    Quadrotor::AHRS();
  }
}

// Nonlinear Complementary Filter
// Source: S. Tellex, A. Brown, and S. Lupashin. Estimation for ...
   Quadrotors. June 11, 2018.
void Quadrotor::NonlinearComplementaryFilter(void)
{
  Quadrotor::IMUread();

  accelAngle.x = -atan(accel.calibrated.x / accel.calibrated.z); // ...
      Roll [rad]
  accelAngle.y = atan(accel.calibrated.y / accel.calibrated.z); // ...
      Pitch [rad]
```

```
  _XYZ gyroAngle_rad;
  gyroAngle_rad.x = gyroAngle.x * PI / 180;
  gyroAngle_rad.y = -gyroAngle.y* PI / 180;
  gyroAngle_rad.z = -gyroAngle.z * PI / 180;


  // Nonlinear Complementary Filter using Quaternions
  double tau = (0.90) / (1 - 0.90) * dt;
  quat.q[0] = cos(gyroAngle_rad.x / 2) * cos(gyroAngle_rad.y / 2) * ...
     cos(gyroAngle_rad.z / 2)   + sin(gyroAngle_rad.x / 2)  * ...
     sin(gyroAngle_rad.y / 2) * sin(gyroAngle_rad.z / 2);
  quat.q[1] = -cos(gyroAngle_rad.x / 2) * sin(gyroAngle_rad.y / 2) * ...
     sin(gyroAngle_rad.z / 2)   + cos(gyroAngle_rad.y / 2) * ...
     cos(gyroAngle_rad.z / 2)   * sin(gyroAngle_rad.x / 2);
  quat.q[2] = cos(gyroAngle_rad.x / 2) * cos(gyroAngle_rad.z / 2)   ...
     * sin(gyroAngle_rad.y / 2) + sin(gyroAngle_rad.x / 2)  * ...
     cos(gyroAngle_rad.y / 2) * sin(gyroAngle_rad.z / 2);
  quat.q[3] = cos(gyroAngle_rad.x / 2) * cos(gyroAngle_rad.y / 2) * ...
     sin(gyroAngle_rad.z / 2)   - sin(gyroAngle_rad.x / 2)  * ...
     cos(gyroAngle_rad.z / 2)   * sin(gyroAngle_rad.y / 2);

  Quaternion dq;

  double omega[3] = {gyro.calibrated.x, gyro.calibrated.y, ...
     gyro.calibrated.z};
  double theta = (double)sqrt(omega[0] * omega[0] * dt + omega[1] * ...
     omega[1] * dt + omega[2] * omega[2] * dt);
  if (theta < 1e-8) {
    dq = dq;
  } else {
    dq.q[0] = cos(theta / 2.f);
    dq.q[1] = sin(theta / 2.f) * omega[0] * dt / theta;
    dq.q[2] = -sin(theta / 2.f) * omega[1] * dt / theta;
    dq.q[3] = -sin(theta / 2.f) * omega[2] * dt / theta;
  }

  quat = Quaternion().multiply(dq, quat).normalize();

  // Convert to Euler
  double roll_predicted, pitch_predicted, yaw_predicted;
  roll_predicted = atan2(2 * (quat.q[2] * quat.q[3] + quat.q[0] * ...
     quat.q[1]),  quat.q[0] * quat.q[0] - quat.q[1] * quat.q[1] - ...
     quat.q[2] * quat.q[2] + quat.q[3] * quat.q[3]);
  pitch_predicted = -asin(2 * (quat.q[1] * quat.q[3] - quat.q[0] * ...
     quat.q[2]));
  yaw_predicted = atan2(2 * (quat.q[1] * quat.q[2] + quat.q[0] * ...
     quat.q[3]),  quat.q[0] * quat.q[0] + quat.q[1] * quat.q[1] - ...
     quat.q[2] * quat.q[2] - quat.q[3] * quat.q[3]);


  X.phi = (tau / (tau + dt)) * (roll_predicted) + (dt / (tau + dt)) ...
     * accelAngle.x;
  X.theta = (tau / (tau + dt)) * (pitch_predicted) + (dt / (tau + ...
     dt)) * accelAngle.y;
  X.psi = yaw_predicted;
}
```

```
// Attitude and Heading Reference Systems (AHRS)
// S. Madgwick (http://x-io.co.uk/res/doc/madgwick_internal_report.pdf)
// Source: Shi Lu (https://github.com/ragewrath/Mark3-Copter-Pilot)
void Quadrotor::AHRS(void)
{
  Quadrotor::IMUread();
  Quadrotor::MahonyAHRS();
}

void Quadrotor::MahonyAHRS()
{
  float quat[4], ypr[3], gx, gy, gz, AHRS_val[9];
  AHRS_val[0] = accel.filtered.x / 8192.0;
  AHRS_val[1] = accel.filtered.y / 8192.0;
  AHRS_val[2] = accel.filtered.z / 8192.0;
  AHRS_val[3] = X.p;
  AHRS_val[4] = -X.q;
  AHRS_val[5] = -X.r;

  Quadrotor::MahonyAHRSUpdate(AHRS_val[3], AHRS_val[4], AHRS_val[5], ...
     AHRS_val[0], AHRS_val[1], AHRS_val[2]);

  quat[0] = q.w;
  quat[1] = q.x;
  quat[2] = q.y;
  quat[3] = q.z;

  gx = 2 * (quat[1] * quat[3] - quat[0] * quat[2]);
  gy = 2 * (quat[0] * quat[1] + quat[2] * quat[3]);
  gz = quat[0] * quat[0] - quat[1] * quat[1] - quat[2] * quat[2] + ...
     quat[3] * quat[3];

  ypr[0] = atan2(2 * quat[1] * quat[2] - 2 * quat[0] * quat[3], 2 * ...
     quat[0] * quat[0] + 2 * quat[1] * quat[1] - 1);
  ypr[1] = atan(gx / sqrt(gy * gy + gz * gz));
  ypr[2] = atan(gy / sqrt(gx * gx + gz * gz));;

  X.phi = ypr[2];
  X.theta = ypr[1];
  // When the base stations are off
  if (channel.CH5 < 1600)
  {
    X.psi = ypr[0];
  }
}

void Quadrotor::MahonyAHRSUpdate(float gx, float gy, float gz, float ...
    ax, float ay, float az)
{
  float recipNorm;
  float halfvx, halfvy, halfvz;
  float halfex, halfey, halfez;
  float qa, qb, qc;
  float q0, q1, q2, q3;
  q0 = q.w;
  q1 = q.x;
  q2 = q.y;
```

```
q3 = q.z;

// Compute feedback only if accelerometer measurement valid ...
    (avoids NaN in accelerometer normalisation)
if (!((ax == 0.0f) && (ay == 0.0f) && (az == 0.0f))) {

  // Normalise accelerometer measurement
  recipNorm = invSqrt(ax * ax + ay * ay + az * az);
  ax *= recipNorm;
  ay *= recipNorm;
  az *= recipNorm;

  // Estimated direction of gravity and vector perpendicular to ...
     magnetic flux
  halfvx = q1 * q3 - q0 * q2;
  halfvy = q0 * q1 + q2 * q3;
  halfvz = q0 * q0 - 0.5f + q3 * q3;

  // Error is sum of cross product between estimated and measured ...
     direction of gravity
  halfex = (ay * halfvz - az * halfvy);
  halfey = (az * halfvx - ax * halfvz);
  halfez = (ax * halfvy - ay * halfvx);

  // Compute and apply integral feedback if enabled
  if (twoKi > 0.0f) {
    integralFBx += twoKi * halfex * (1.0f / FREQ);  // integral ...
       error scaled by Ki
    integralFBy += twoKi * halfey * (1.0f / FREQ);
    integralFBz += twoKi * halfez * (1.0f / FREQ);
    gx += integralFBx;  // apply integral feedback
    gy += integralFBy;
    gz += integralFBz;
  }
  else {
    integralFBx = 0.0f; // prevent integral windup
    integralFBy = 0.0f;
    integralFBz = 0.0f;
  }

  // Apply proportional feedback
  gx += twoKp * halfex;
  gy += twoKp * halfey;
  gz += twoKp * halfez;
}

// Integrate rate of change of quaternion
gx *= (0.5f * (1.0f / FREQ));   // pre-multiply common factors
gy *= (0.5f * (1.0f / FREQ));
gz *= (0.5f * (1.0f / FREQ));
qa = q0;
qb = q1;
qc = q2;
q0 += (-qb * gx - qc * gy - q3 * gz);
q1 += (qa * gx + qc * gz - q3 * gy);
q2 += (qa * gy - qb * gz + q3 * gx);
q3 += (qa * gz + qb * gy - qc * gx);
```

```
  // Normalise quaternion
  recipNorm = invSqrt(q0 * q0 + q1 * q1 + q2 * q2 + q3 * q3);
  q0 *= recipNorm;
  q1 *= recipNorm;
  q2 *= recipNorm;
  q3 *= recipNorm;

  q.w = q0;
  q.x = q1;
  q.y = q2;
  q.z = q3;
}

// Quadrotor States of Operation
void Quadrotor::ArmingState(void)
{
  if (QuadrotorState == START_MODE && channel.CH3 < 320 && ...
      channel.CH4 < 320)
  {
    QuadrotorState = TRANS_MODE;
  }
  if (QuadrotorState == TRANS_MODE && channel.CH3 < 320 && ...
      channel.CH4 > 900 && channel.CH4 < 1500)
  {
    QuadrotorState = ARMING_MODE;
    // Reset variables
    error.p_integral = 0;
    error.q_integral = 0;
    error.r_integral = 0;
    error.phi_integral = 0;
    error.theta_integral = 0;
    error.psi_integral = 0;
  }
  if (QuadrotorState == ARMING_MODE && channel.CH3 < 320 && ...
      channel.CH4 < 320)
  {
    QuadrotorState = TEMP_MODE;
  }
  if (QuadrotorState == TEMP_MODE && channel.CH3 < 320 && ...
      channel.CH4 > 900 && channel.CH4 < 1500)
  {
    QuadrotorState = START_MODE;
  }
  if (channel.CH3 < 320 && channel.CH4 > 1600)
  {
    QuadrotorState = DISARMING_MODE;
  }
  if (channel.CH6 > 1000)
  {
    QuadrotorState = DISARMING_MODE;
  }
}

// Checking Battery Voltage
void Quadrotor::BatteryVoltageCheck(void)
{
```

```
float v = (float)analogRead(A14) * 0.019586;
voltage = v * 0.005 + voltage * 0.995;
voltage = Quadrotor::CONSTRAIN(voltage, 9.0, 17.0);
if (voltage < 10.5)
{
  if (blink == 0)
  {
    if (blinkCounter <= 160)
    {
      blinkCounter += 1;
    }
    else
    {
      blinkCounter = 0;
      blink = 1;
    }
    digitalWrite(LEDRed, HIGH);
    digitalWrite(LEDGreen, LOW);
  }
  else
  {
    if (blinkCounter <= 160)
    {
      blinkCounter += 1;
    }
    else
    {
      blinkCounter = 0;
      blink = 0;
    }
    digitalWrite(LEDRed, LOW);
    digitalWrite(LEDGreen, LOW);
  }
}
else
{
  if (blink == 0)
  {
    if (blinkCounter <= 160)
    {
      blinkCounter += 1;
    }
    else
    {
      blinkCounter = 0;
      blink = 1;
    }
    digitalWrite(LEDRed, LOW);
    digitalWrite(LEDGreen, HIGH);
  }
  else
  {
    if (blinkCounter <= 160)
    {
      blinkCounter += 1;
    }
    else
```

```
      {
        blinkCounter = 0;
        blink = 0;
      }
      digitalWrite(LEDRed, LOW);
      digitalWrite(LEDGreen, LOW);
    }
  }
}

// RC Remote Signal Receiver
void Quadrotor::Receiver(void)
{
  if (channel.CH5 < 1600)
  {
    Xdes.phi = 0;
    if (channel.CH1 > 1100)
    {
      float phi_desired_degree = (channel.CH1 - 1100) / 30;
      Xdes.phi = phi_desired_degree * (PI / 180.0);
    }
    else if (channel.CH1 < 900)
    {
      float phi_desired_degree = (channel.CH1 - 900) / 30;
      Xdes.phi = phi_desired_degree * (PI / 180.0);
    }
    Xdes.phi = Quadrotor::CONSTRAIN(Xdes.phi, -0.6, 0.6);

    Xdes.theta = 0;
    if (channel.CH2 > 1100)
    {
      float theta_desired_degree = (channel.CH2 - 1100) / 30;
      Xdes.theta = theta_desired_degree * (PI / 180.0);
    }
    else if (channel.CH2 < 900)
    {
      float theta_desired_degree = (channel.CH2 - 900) / 30;
      Xdes.theta = theta_desired_degree * (PI / 180.0);
    }
    Xdes.theta = Quadrotor::CONSTRAIN(Xdes.theta, -0.6, 0.6);

    RCYawRate = 0;
    if (channel.CH4 > 1100)
    {
      float psi_desired_degree = (channel.CH4 - 1100) / 10;
      RCYawRate = psi_desired_degree * (PI / 180.0);
    }
    else if (channel.CH4 < 900)
    {
      float psi_desired_degree = (channel.CH4 - 900) / 10;
      RCYawRate = psi_desired_degree * (PI / 180.0);
    }
    RCYawRate = Quadrotor::CONSTRAIN(RCYawRate, -1.5, 1.5);
  }
}

// Choosing the control method:
```

```
// 1. Classical PID
// 2. Pole-Placement
void Quadrotor::Control(int8_t method)
{
  control_method = method;
}

// Attitude Control (Outer-loop at 100Hz)
// P Controller
void Quadrotor::AttitudeControl(void)
{
  if (outerCounter >= 4)
  {
    Wp.input = 0;
    Wq.input = 0;
    Wr.input = 0;
    Wp.output = 0;
    Wq.output = 0;
    Wr.output = 0;

    if (QuadrotorState == ARMING_MODE)
    {
      if (channel.CH3 > 320 || (channel.CH5 > 1600 && flight_mode == ...
          1) || (channel.CH5 > 1600 && flight_mode == 2) || ...
          (channel.CH5 > 1600 && flight_mode == 3))
      {
        error.phi = Xdes.phi - X.phi;
        Xdes.p = kpx * error.phi;

        // Prefilter
        Wp.input = Xdes.p;
        Wp.output = (0.6 * Wp.output_prev1) + (0.2 * Wp.input) + ...
            (0.2 * Wp.input_prev1);
        Wp.output = Quadrotor::CONSTRAIN(Wp.output, -6.28, 6.28);

        error.theta = Xdes.theta - X.theta;
        Xdes.q = kpy * error.theta;

        // Prefilter
        Wq.input = Xdes.q;
        Wq.output = (0.6 * Wq.output_prev1) + (0.2 * Wq.input) + ...
            (0.2 * Wq.input_prev1);
        Wq.output = Quadrotor::CONSTRAIN(Wq.output, -6.28, 6.28);

        if (channel.CH5 > 1600) // HTC Vive Base Station
        {
          // Already have a desired yaw rate
        }
        else
        {
          error.psi = Xdes.psi - X.psi;
          (error.psi < -PI ? error.psi+(2*PI) : (error.psi > PI ? ...
              error.psi - (2*PI): error.psi));
          Xdes.r = kpz * error.psi;
          Xdes.r = Quadrotor::CONSTRAIN(Xdes.r, -1, 1);

          if (RCYawRate >= 0.09 || RCYawRate <= -0.09)
```

```
          {
            Xdes.r = RCYawRate;
            Xdes.psi = X.psi;
          }
          if (channel.CH3 < 320)
          {
            Xdes.psi = X.psi;
          }
        }
      }
    }
    Wp.output_prev1 = Wp.output;
    Wp.input_prev1 = Wp.input;
    Wq.output_prev1 = Wq.output;
    Wq.input_prev1 = Wq.input;
  }
  Quadrotor::AngularRateControl();
}

// Angular Rate Control or Body Rate Control (Inner-loop at 400Hz)
// PID Controller (PD Controller is sufficient)
void Quadrotor::AngularRateControl(void)
{
  U2.current = 0;
  U3.current = 0;
  U4.current = 0;
  if (QuadrotorState == ARMING_MODE)
  {
    if (channel.CH3 > 320 || (channel.CH5 > 1600 && flight_mode == ...
        1) || (channel.CH5 > 1600 && flight_mode == 2) || ...
        (channel.CH5 > 1600 && flight_mode == 3))
    {
      if (control_method == 1) // PID with No prefilter
      {
        error.p = Xdes.p - X.p;
        error.p_integral += error.p * dt;
        error.p_integral = Quadrotor::CONSTRAIN(error.p_integral, ...
            -1, 1);
        U2.current = error.p * kpPQRx + (error.p - error.p_prev1) * ...
            kdPQRx / dt + error.p_integral * kiPQRx;
        error.p_prev1 = error.p;

        error.q = Xdes.q - X.q;
        error.q_integral += error.q * dt;
        error.q_integral = Quadrotor::CONSTRAIN(error.q_integral, ...
            -1, 1);
        U3.current = error.q * kpPQRy + (error.q - error.q_prev1) * ...
            kdPQRy / dt + error.q_integral * kiPQRy;
        error.q_prev1 = error.q;

        // Note: Using Radio Control (RC) we control the yaw angular ...
            rate (NOT yaw angle)
        // Base Station is ON
        if (channel.CH5 > 1600)
        {
          error.r = Xdes.r - X.r;
          U4.current = error.r * kpPQRz + (error.r - error.r_prev1) ...
```

```
            * kdPQRz / dt;
          error.r_prev1 = error.r;
        }
        else // Base Station is OFF
        {
          error.r = Xdes.r - X.r;
          error.r_integral += error.r * dt;
          error.r_integral = Quadrotor::CONSTRAIN(error.r_integral, ...
              -1, 1);
          U4.current = error.r * kpPQRz + (error.r - error.r_prev1) ...
              * kdPQRz / dt + error.r_integral * kiPQRz;
          error.r_prev1 = error.r;
        }
      }
      else if (control_method == 2) // PID with prefilter (Design ...
          for Bandwidth and Robustness)
      {
        error.p = Wp.output - X.p;
        U2.current = (2.2 * U2.prev1) - (1.56 * U2.prev2) + (0.36 * ...
            U2.prev3) + (0.0918 * error.p) - (0.08645 * ...
            error.p_prev1) - (0.09174 * error.p_prev2) + (0.08652 * ...
            error.p_prev3);

        error.q = Wq.output - X.q;
        U3.current = (2.2 * U3.prev1) - (1.56 * U3.prev2) + (0.36 * ...
            U3.prev3) + (0.0918 * error.q) - (0.08645 * ...
            error.q_prev1) - (0.09174 * error.q_prev2) + (0.08652 * ...
            error.q_prev3);

        // Note: Using Radio Control (RC) we control the yaw angular ...
            rate (NOT yaw angle)
        // Base Station is ON
        if (channel.CH5 > 1600)
        {
          error.r = Xdes.r - X.r;
          U4.current = error.r * kpPQRz + (error.r - error.r_prev1) ...
              * kdPQRz / dt;
          error.r_prev1 = error.r;
        }
        else // Base Station is OFF
        {
          error.r = Xdes.r - X.r;
          error.r_integral += error.r * dt;
          error.r_integral = Quadrotor::CONSTRAIN(error.r_integral, ...
              -1, 1);
          U4.current = error.r * kpPQRz + (error.r - error.r_prev1) ...
              * kdPQRz / dt + error.r_integral * kiPQRz;
          error.r_prev1 = error.r;
        }
      }
    }
  }
}
U2.current = Quadrotor::CONSTRAIN(U2.current, -2, 2);
U2.prev3 = U2.prev2;
U2.prev2 = U2.prev1;
U2.prev1 = U2.current;
U3.current = Quadrotor::CONSTRAIN(U3.current, -2, 2);
```

```
  U3.prev3 = U3.prev2;
  U3.prev2 = U3.prev1;
  U3.prev1 = U3.current;
  error.p_prev3 = error.p_prev2;
  error.p_prev2 = error.p_prev1;
  error.p_prev1 = error.p;
  error.q_prev3 = error.q_prev2;
  error.q_prev2 = error.q_prev1;
  error.q_prev1 = error.q;
  error.r_prev3 = error.r_prev2;
  error.r_prev2 = error.r_prev1;
  error.r_prev1 = error.r;
}

void Quadrotor::ThrottleControl(void)
{
  if (QuadrotorState == ARMING_MODE && channel.CH3 > 320)
  {
    float Thrust = 0.008193 * channel.CH3 - 2.458;
    U1.current = Thrust / (cos(X.phi)*cos(X.theta));
    U1.current = Quadrotor::CONSTRAIN(U1.current, 0, 15);;
  }
  else
  {
    U1.current = 0;
  }
}

void Quadrotor::DifferentialFlatness(void)
{
  if (channel.CH5 > 1600)
  {
    if (outerCounter >= 4)
    {
      if (QuadrotorState == ARMING_MODE)
      {
        if (flight_mode == 1 || flight_mode == 2 || flight_mode == 3)
        {
          U1des.current = Quadrotor::CONSTRAIN(U1des.current, 0, 15);
          Xdes.phi = Quadrotor::CONSTRAIN(Xdes.phi, -0.6, 0.6);
          Xdes.theta = Quadrotor::CONSTRAIN(Xdes.theta, -0.6, 0.6);
          Xdes.r = Quadrotor::CONSTRAIN(Xdes.r, -2, 2);

          // Assign Thrust Here
          U1.current = U1des.current;
        }
        else if (flight_mode == -1)
        {
          QuadrotorState = DISARMING_MODE;
        }
        else
        {
          U1.current = 0;
          Xdes.phi = 0;
          Xdes.theta = 0;
          Xdes.r = 0;
        }
```

```
      }
    }
  }
  else
  {
    Quadrotor::ThrottleControl();
  }
}

// Generate Motor Commands
// Shi Lu (https://github.com/ragewrath/Mark3-Copter-Pilot)
// Notes: Please read README file
void Quadrotor::GenerateMotorCommands(void)
{
  omega1Squared = 130958.617 * U1.current - 1480900 * U2.current + ...
     1480900 * U3.current + 1290232.68 * U4.current;
  omega2Squared = 130958.617 * U1.current + 1480900 * U2.current - ...
     1480900 * U3.current + 1290232.68 * U4.current;
  omega3Squared = 130958.617 * U1.current + 1480900 * U2.current + ...
     1480900 * U3.current - 1290232.68 * U4.current;
  omega4Squared = 130958.617 * U1.current - 1480900 * U2.current - ...
     1480900 * U3.current - 1290232.68 * U4.current;
  omega1Squared = Quadrotor::CONSTRAIN(omega1Squared, 0, 900000000);
  omega2Squared = Quadrotor::CONSTRAIN(omega2Squared, 0, 900000000);
  omega3Squared = Quadrotor::CONSTRAIN(omega3Squared, 0, 900000000);
  omega4Squared = Quadrotor::CONSTRAIN(omega4Squared, 0, 900000000);
  omega1 = sqrt(omega1Squared);
  omega2 = sqrt(omega2Squared);
  omega3 = sqrt(omega3Squared);
  omega4 = sqrt(omega4Squared);

  // Motor Model
  if (QuadrotorState == ARMING_MODE)
  {
    float param_a = 1166.0, param_b = 5393, param_c = 299600, ...
       param_d = 1544, param_e = 894.5;
    PWM1 = (omega1 * omega1 + param_b * omega1 + param_c) / (param_a ...
       * voltage + param_d) + param_e;
    PWM2 = (omega2 * omega2 + param_b * omega2 + param_c) / (param_a ...
       * voltage + param_d) + param_e;
    PWM3 = (omega3 * omega3 + param_b * omega3 + param_c) / (param_a ...
       * voltage + param_d) + param_e;
    PWM4 = (omega4 * omega4 + param_b * omega4 + param_c) / (param_a ...
       * voltage + param_d) + param_e;
    PWM1 = Quadrotor::CONSTRAIN(PWM1, MIN_MOTOR_LEVEL, MAX_MOTOR_LEVEL);
    PWM2 = Quadrotor::CONSTRAIN(PWM2, MIN_MOTOR_LEVEL, MAX_MOTOR_LEVEL);
    PWM3 = Quadrotor::CONSTRAIN(PWM3, MIN_MOTOR_LEVEL, MAX_MOTOR_LEVEL);
    PWM4 = Quadrotor::CONSTRAIN(PWM4, MIN_MOTOR_LEVEL, MAX_MOTOR_LEVEL);
  }
  if (QuadrotorState != ARMING_MODE)
  {
    PWM1 = OFF_MOTOR_LEVEL;
    PWM2 = OFF_MOTOR_LEVEL;
    PWM3 = OFF_MOTOR_LEVEL;
    PWM4 = OFF_MOTOR_LEVEL;
  }
  if (QuadrotorState == DISARMING_MODE)
```

```
  {
    PWM1 = OFF_MOTOR_LEVEL;
    PWM2 = OFF_MOTOR_LEVEL;
    PWM3 = OFF_MOTOR_LEVEL;
    PWM4 = OFF_MOTOR_LEVEL;
  }

  Quadrotor::MotorRun();
}

// Run Motors
void Quadrotor::MotorRun(void)
{
  float input1 = PWM_FACTOR * PWM1;
  float input2 = PWM_FACTOR * PWM2;
  float input3 = PWM_FACTOR * PWM3;
  float input4 = PWM_FACTOR * PWM4;
  analogWrite(MOTOR1, input1);
  analogWrite(MOTOR2, input2);
  analogWrite(MOTOR3, input3);
  analogWrite(MOTOR4, input4);
}


// Second Order Low Pass Filter
// Source: Leonard Hall (https://github.com/PX4/Firmware) ...
   PX4/Firmware/src/lib/mathlib/math/filter/LowPassFilter2p.cpp
void Quadrotor::SecondOrderLowPassFilter(float sample_freq, float ...
   cutoff_freq, struct _FILTER *input_IIR)
{
  if (cutoff_freq <= 0.0f) {
    // no filtering
    return;
  }
  float fr = sample_freq / cutoff_freq;
  float ohm = tanf(PI / fr);
  float c = 1.0f + 2.0f * cosf(PI / 4.0f) * ohm + ohm * ohm;
  input_IIR->b0 = ohm * ohm / c;
  input_IIR->b1 = 2.0f * input_IIR->b0;
  input_IIR->b2 = input_IIR->b0;
  input_IIR->a1 = 2.0f * (ohm * ohm - 1.0f) / c;
  input_IIR->a2 = (1.0f - 2.0f * cosf(PI / 4.0f) * ohm + ohm * ohm) ...
     / c;
}

// Second Order Low Pass Filter Apply
// Source: Leonard Hall (https://github.com/PX4/Firmware) ...
   PX4/Firmware/src/lib/mathlib/math/filter/LowPassFilter2p.cpp
float Quadrotor::SecondOrderLowPassFilterApply(float cutoff_freq, ...
   float sample, struct _FILTER *input_IIR)
{
  if (cutoff_freq <= 0.0f) {
    // no filtering
    return sample;
  }
  // do the filtering
  input_IIR->element0 = sample - input_IIR->element1 * input_IIR->a1 ...
```

```
      - input_IIR->element2 * input_IIR->a2;
  float output = input_IIR->element0 * input_IIR->b0 + ...
      input_IIR->element1 * input_IIR->b1 + input_IIR->element2 * ...
      input_IIR->b2;

  input_IIR->element2 = input_IIR->element1;
  input_IIR->element1 = input_IIR->element0;

  // return the value.  Should be no need to check limits
  return output;
}

// Constrain Data Between Min and Max Values
float Quadrotor::CONSTRAIN(float x, float min, float max)
{
  if (x < min) x = min;
  if (x > max) x = max;
  return x;
}

// Invert Square Root
float Quadrotor::invSqrt(float number) {
  long i;
  float x2, y;
  const float threehalfs = 1.5F;

  x2 = number * 0.5F;
  y  = number;
  i  = * ( long * ) &y;
  i  = 0x5f3759df - ( i >> 1 );
  y  = * ( float * ) &i;
  y  = y * ( threehalfs - ( x2 * y * y ) );
  return y;
}

void Quadrotor::LoopCounter(void)
{
  if (outerCounter >= 4)
  {
    outerCounter = 0;
  }
  while (micros() - loop_timer < dtMicroseconds);
  // Reset the zero timer
  loop_timer = micros();
}
```

B.4 Communication.h

```
#include <Eigen.h>     // Calls main Eigen matrix class library
#include <Eigen/LU>            // Calls inverse, determinant, LU ...
   decomp., etc.
#include "Quadrotor.h"
#include "mavlink.h"

using namespace Eigen;    // Eigen related statement; simplifies ...
```

```
   syntax for declaration of matrices

class Communication
{
public:
  // Variables
  struct _CommunicationState {
    float thrust, phi, theta, r;
    int mode;
  };
  _CommunicationState CommunicationState;

  // Methods
  void ROS_Send(Quadrotor *quadrotor);
  void ROS_Receive(Quadrotor *quadrotor);
private:
};
```

B.5 Communication.cpp

```
#include "Communication.h"

void Communication::ROS_Send(Quadrotor *quadrotor)
{
    mavlink_message_t mav_msg;
    uint8_t len;
    uint8_t buf[MAVLINK_MAX_PACKET_LEN];
    uint32_t time_mav = micros();
    mavlink_msg_raw_imu_pack(1, 1, &mav_msg, time_mav,
      quadrotor->accel.filtered.x*1000, ...
         quadrotor->accel.filtered.y*1000, ...
         quadrotor->accel.filtered.z*1000,
      quadrotor->X.p*1000, quadrotor->X.q*1000, quadrotor->X.r*1000, ...
         quadrotor->X.phi*1000, quadrotor->X.theta*1000, ...
         quadrotor->voltage*1000);
    len = mavlink_msg_to_send_buffer(buf, &mav_msg);
    Serial1.write(buf, len);
}

void Communication::ROS_Receive(Quadrotor *quadrotor)
{
  mavlink_message_t msg;
  mavlink_status_t status;
  while (Serial1.available())
  {
    uint8_t c = Serial1.read();
    if (mavlink_parse_char(MAVLINK_COMM_0, c, &msg, &status))
    {
      switch (msg.msgid)
      {
        case MAVLINK_MSG_ID_HEARTBEAT:
          mavlink_heartbeat_t hb;
          mavlink_msg_heartbeat_decode(&msg, &hb);
          Serial.print(millis());
          Serial.print("\ncustom_mode: "); ...
```

```
            Serial.println(hb.custom_mode);
          Serial.print("Type: "); Serial.println(hb.type);
          Serial.print("autopilot: "); Serial.println(hb.autopilot);
          Serial.print("mavlink_version: "); ...
            Serial.println(hb.mavlink_version);
          Serial.println();
          break;
        case MAVLINK_MSG_ID_ROLL_PITCH_YAW_SPEED_THRUST_SETPOINT:
          mavlink_roll_pitch_yaw_speed_thrust_setpoint_t des;
          mavlink_msg_roll_pitch_yaw_speed_thrust_setpoint_decode(&msg, ...
            &des);
          CommunicationState.thrust = des.thrust;
          CommunicationState.phi = des.roll_speed; // We are sending ...
            roll angle (rad) NOT body_roll_rate
          CommunicationState.theta = des.pitch_speed; // We are ...
            sending pitch angle (rad) NOT body_pitch_rate
          CommunicationState.r = des.yaw_speed;
          CommunicationState.mode = des.time_boot_ms;
          quadrotor->flight_mode = CommunicationState.mode;
          Serial.println(CommunicationState.thrust);

          if (quadrotor->channel.CH5 > 1600)
          {
            quadrotor->U1des.current = CommunicationState.thrust;
            quadrotor->Xdes.phi = CommunicationState.phi;
            quadrotor->Xdes.theta = CommunicationState.theta;
            quadrotor->Xdes.r = CommunicationState.r;

            quadrotor->U1des.current = ...
              quadrotor->CONSTRAIN(quadrotor->U1des.current, 0, 15);
            quadrotor->Xdes.phi = ...
              quadrotor->CONSTRAIN(quadrotor->Xdes.phi, -1.0, 1.0);
            quadrotor->Xdes.theta = ...
              quadrotor->CONSTRAIN(quadrotor->Xdes.theta, -1.0, 1.0);
            quadrotor->Xdes.r = ...
              quadrotor->CONSTRAIN(quadrotor->Xdes.r, -2, 2);

          }
      }
    }
  }
}
```

APPENDIX C

ROBOT OPERATING SYSTEM (ROS)

C.1 quad_serial.cpp

```
// Author: Abdullah Altawaitan
// Date: April 4, 2019

#include "ros/ros.h"
#include <serial/serial.h>
#include "sensor_msgs/Imu.h"
#include "quad/quad_cmd_msg.h"
#include "mavlink/mavlink.h"

serial::Serial ser;

class quad_serial
{
public:
  ros::NodeHandle n;
  ros::Subscriber sub;
  ros::Publisher pub;
  sensor_msgs::Imu imu;
  quad::quad_cmd_msg quad_cmd;

  quad_serial()
  {
    sub = n.subscribe<quad::quad_cmd_msg>("/quad_cmd", 1000, ...
        &quad_serial::callback, this);
    pub = n.advertise<sensor_msgs::Imu>("quad_serial_imu", 1000);
  }

  ~quad_serial()
  {
    // Empty
  }

  void callback(const quad::quad_cmd_msg::ConstPtr& msg)
  {
    // Serial write
    quad_cmd.mode = msg->mode;
    quad_cmd.phi_des = msg->phi_des;
    quad_cmd.theta_des = msg->theta_des;
    quad_cmd.r_des = msg->r_des;
    quad_cmd.thrust_des = msg->thrust_des;
    mavlink_message_t mav_msg2;
    uint8_t len;
    uint8_t buf2[MAVLINK_MAX_PACKET_LEN];
    ros::Time begin = ros::Time::now();
    mavlink_msg_roll_pitch_yaw_speed_thrust_setpoint_pack(255, 1, ...
        &mav_msg2,
                                    quad_cmd.mode, quad_cmd.phi_des, ...
                                       quad_cmd.theta_des, ...
                                       quad_cmd.r_des, ...
                                       quad_cmd.thrust_des);
    len = mavlink_msg_to_send_buffer(buf2, &mav_msg2);
    ser.write(buf2, len);
  }
```

```
  void serialRead(void)
  {
    // Serial read
    uint8_t buf[MAVLINK_MAX_PACKET_LEN];
    size_t length = ser.read(buf, sizeof(buf));
    if (length > 0)
    {
      mavlink_message_t mav_msg;
        mavlink_status_t status;

      for (int i = 0; i < length; i++)
      {
        if(mavlink_parse_char(MAVLINK_COMM_0, buf[i], &mav_msg, ...
            &status))
        {
          switch (mav_msg.msgid)
          {
            case MAVLINK_MSG_ID_HEARTBEAT:
              mavlink_heartbeat_t hb;
              mavlink_msg_heartbeat_decode(&mav_msg, &hb);
              break;
            case MAVLINK_MSG_ID_RAW_IMU:
              mavlink_raw_imu_t mav_imu;
              mavlink_msg_raw_imu_decode(&mav_msg, &mav_imu);
              imu.linear_acceleration.x = ((float)mav_imu.xacc)/1000.00;
              imu.linear_acceleration.y = ((float)mav_imu.yacc)/1000.00;
              imu.linear_acceleration.z = ((float)mav_imu.zacc)/1000.00;
              imu.angular_velocity.x = ((float)mav_imu.xgyro)/1000.00;
              imu.angular_velocity.y = ((float)mav_imu.ygyro)/1000.00;
              imu.angular_velocity.z = ((float)mav_imu.zgyro)/1000.00;
              imu.orientation.x = ((float)mav_imu.xmag)/1000.00; // ...
                 Roll angle
              imu.orientation.y = ((float)mav_imu.ymag)/1000.00; // ...
                 Pitch angle
              break;
          }
        }
      }
    }
    pub.publish(imu);
  }
};

int main(int argc, char **argv)
{
  ros::init(argc, argv, "quad_serial_node");
  quad_serial quad;

  try
  {
      ser.setPort("/dev/ttyUSB0");
      ser.setBaudrate(115200);
      serial::Timeout to = serial::Timeout::simpleTimeout(1000);
      ser.setTimeout(to);
      ser.open();
  }
```

```
  catch (serial::IOException& e)
  {
      ROS_ERROR_STREAM("Unable to open port ");
      return -1;
  }

  if (ser.isOpen())
  {
      ROS_INFO_STREAM("Serial Port initialized");
  }
  else
  {
      return -1;
  }

  ros::Rate loop_rate(50);

  while (ros::ok())
  {
    quad.serialRead();
    ros::spinOnce();
    loop_rate.sleep();
  }

  ros::shutdown();
  return 0;
}
```

C.2 quad_state.cpp

```
// Author: Abdullah Altawaitan
// Date: April 4, 2019

#include "ros/ros.h"
#include <serial/serial.h>
#include "Eigen/Dense"
#include "math.h"
#include "geometry_msgs/Vector3.h"
#include "nav_msgs/Odometry.h"
#include "sensor_msgs/Imu.h"
#include "tf/LinearMath/Quaternion.h"
#include "tf/LinearMath/Matrix3x3.h"
#include "quad/quad_state_msg.h"
#include "mavlink/mavlink.h"

float x, y, z, vx, vy, vz, qw, qx, qy, qz;
float vx_prev, vy_prev, vz_prev;
float x_offset, y_offset, z_offset;
float sum_average[13];
float moving_average[13][5];
uint8_t initial = 0;
class quad_state
{
public:
  ros::NodeHandle n;
```

```
ros::Subscriber sub;
ros::Subscriber sub2;
ros::Publisher pub;
ros::Publisher pub2;
sensor_msgs::Imu imu;
quad::quad_state_msg state;

quad_state()
{
  sub = n.subscribe<sensor_msgs::Imu>("/quad_serial_imu", 1000, ...
     &quad_state::callback, this);
  sub2 = ...
     n.subscribe<nav_msgs::Odometry>("/vive/LHR_FC64C5CA_odom", ...
     1000, &quad_state::callback2, this);
  pub = n.advertise<quad::quad_state_msg>("quad_state", 1000);
}

~quad_state()
{
  // Empty
}

void callback(const sensor_msgs::Imu::ConstPtr& msg)
{
  imu.linear_acceleration.x = msg->linear_acceleration.x;
  imu.linear_acceleration.y = msg->linear_acceleration.y;
  imu.linear_acceleration.z = msg->linear_acceleration.z;
  imu.angular_velocity.x = msg->angular_velocity.x;
  imu.angular_velocity.y = msg->angular_velocity.y;
  imu.angular_velocity.z = msg->angular_velocity.z;
  imu.orientation.x = msg->orientation.x; // Roll angle
  imu.orientation.y = msg->orientation.y; // Pitch angle

  state.velocity.angular.x = imu.angular_velocity.x;
  state.velocity.angular.y = imu.angular_velocity.y;
  state.velocity.angular.z = imu.angular_velocity.z;
  state.attitude.x = imu.orientation.x;
  state.attitude.y = imu.orientation.y;
}

void callback2(const nav_msgs::Odometry::ConstPtr& msg)
{
  // Transformation
  tf::Quaternion q1(msg->pose.pose.orientation.x,
                    msg->pose.pose.orientation.y,
                    msg->pose.pose.orientation.z,
                    msg->pose.pose.orientation.w);
  tf::Quaternion q2(0.707, 0.000, 0.000, 0.707);
  tf::Matrix3x3 m(q2*q1);
  double roll, pitch, yaw;
  m.getRPY(roll, pitch, yaw);
  state.attitude.z = -(float) yaw; // Yaw (psi) in (rads)
  tf::Quaternion myQuaternion;
  myQuaternion.setRPY(roll, pitch, yaw);
  state.pose.orientation.w = myQuaternion.w();
  state.pose.orientation.x = myQuaternion.x();
  state.pose.orientation.y = myQuaternion.y();
```

```
state.pose.orientation.z = myQuaternion.z();


// Moving Average
for (uint8_t j = 0; j < 13 ; j++)
{
  sum_average[j] = sum_average[j] - moving_average[j][4] / 5;
}

for (uint8_t i = 4; i > 0; i--)
{
  for (uint8_t j = 0; j < 13 ; j++)
  {
    moving_average[j][i] = moving_average[j][i - 1];
  }
}

if (initial == 0)
{
  x_offset = -(msg->pose.pose.position.x);
  y_offset = -(msg->pose.pose.position.z);
  z_offset = msg->pose.pose.position.y;
  initial = initial + 1;
}

// Shift axes
x = -(msg->pose.pose.position.x) - x_offset;
y = -(msg->pose.pose.position.z) - y_offset;
z = msg->pose.pose.position.y - z_offset;

moving_average[0][0] = x;
moving_average[1][0] = y;
moving_average[2][0] = z;
moving_average[3][0] = -(msg->twist.twist.linear.x);
moving_average[4][0] = -(msg->twist.twist.linear.z);
moving_average[5][0] = msg->twist.twist.linear.y;
moving_average[6][0] = state.attitude.x;
moving_average[7][0] = state.attitude.y;
moving_average[8][0] = state.attitude.z;
moving_average[9][0] = state.pose.orientation.w;
moving_average[10][0] = state.pose.orientation.x;
moving_average[11][0] = state.pose.orientation.y;
moving_average[12][0] = state.pose.orientation.z;

for (uint8_t j = 0; j < 13 ; j++)
{
  sum_average[j] = sum_average[j] + moving_average[j][0] / 5;
}

x = sum_average[0];
y = sum_average[1];
z = sum_average[2];
vx = sum_average[3];
vy = sum_average[4];
vz = sum_average[5];
qw = sum_average[9];
qx = sum_average[10];
```

```
    qy = sum_average[11];
    qz = sum_average[12];

    state.acceleration.linear.x = (vx - vx_prev) / (0.01);
    state.acceleration.linear.y = (vy - vy_prev) / (0.01);
    state.acceleration.linear.z = (vz - vz_prev) / (0.01);
    vx_prev = vx;
    vy_prev = vy;
    vz_prev = vz;

    float CF_a = 0.8, dtOuter = 0.01, dtOuter_2 = dtOuter * dtOuter;
    state.velocity.linear.x = (state.velocity.linear.x + dtOuter * ...
       state.acceleration.linear.x) * CF_a + vx * (1 - CF_a);
    state.velocity.linear.y = (state.velocity.linear.y + dtOuter * ...
       state.acceleration.linear.y) * CF_a + vy * (1 - CF_a);
    state.velocity.linear.z = (state.velocity.linear.z + dtOuter * ...
       state.acceleration.linear.z) * CF_a + vz * (1 - CF_a);

    state.pose.position.x = (state.pose.position.x + dtOuter * ...
       state.velocity.linear.x + 0.5 * dtOuter_2 * ...
       state.acceleration.linear.x) * CF_a + x * (1 - CF_a);
    state.pose.position.y = (state.pose.position.y + dtOuter * ...
       state.velocity.linear.y + 0.5 * dtOuter_2 * ...
       state.acceleration.linear.y) * CF_a + y * (1 - CF_a);
    state.pose.position.z = (state.pose.position.z + dtOuter * ...
       state.velocity.linear.z + 0.5 * dtOuter_2 * ...
       state.acceleration.linear.z) * CF_a + z * (1 - CF_a);

    state.x_offset = x_offset;
    state.y_offset = y_offset;
    state.z_offset = z_offset;
    // Publish
    pub.publish(state);
  }

  float invSqrt(float number)
  {
    long i;
    float x2, y;
    const float threehalfs = 1.5F;

    x2 = number * 0.5F;
    y  = number;
    i  = * ( long * ) &y;
    i  = 0x5f3759df - ( i >> 1 );
    y  = * ( float * ) &i;
    y  = y * ( threehalfs - ( x2 * y * y ) );
    return y;
  }

};



int main(int argc, char **argv)
{
  ros::init(argc, argv, "quad_state");
```

```
  quad_state quad;

  ros::Rate loop_rate(100);

  ros::spin();
  ros::shutdown();
  return 0;
}
```

C.3 quad_control.cpp

```
// Author: Abdullah Altawaitan
// Date: April 4, 2019

#include "ros/ros.h"
#include "quad/quad_state_msg.h"
#include "quad/quad_cmd_msg.h"

double error_xi = 0;
double error_yi = 0;
double error_zi = 0;
double error_psii = 0;

class quad_control
{
public:
  ros::NodeHandle n;
  ros::Subscriber sub;
  ros::Subscriber sub2;
  ros::Publisher pub;

  quad::quad_cmd_msg quad_cmd;
  quad::quad_state_msg state;
  quad::quad_state_msg desired;
  float mass = 0.647;

  // Constructor
  quad_control()
  {
    sub = n.subscribe<quad::quad_state_msg>("/quad_state", 1000, ...
       &quad_control::callback, this);
    sub2 = n.subscribe<quad::quad_state_msg>("/quad_trajectory", ...
       1000, &quad_control::callback2, this);
    pub = n.advertise<quad::quad_cmd_msg>("quad_cmd", 1000);
  }
  // Destructor
  ~quad_control()
  {
    // Empty
  }

  void callback(const quad::quad_state_msg::ConstPtr& msg)
  {
    state.pose.position.x = msg->pose.position.x;
    state.pose.position.y = msg->pose.position.y;
```

```
  state.pose.position.z = msg->pose.position.z;
  state.velocity.linear.x = msg->velocity.linear.x;
  state.velocity.linear.y = msg->velocity.linear.y;
  state.velocity.linear.z = msg->velocity.linear.z;
  state.velocity.angular.x = msg->velocity.angular.x;
  state.velocity.angular.y = msg->velocity.angular.y;
  state.velocity.angular.z = msg->velocity.angular.z;
  state.acceleration.linear.x = msg->acceleration.linear.x;
  state.acceleration.linear.y = msg->acceleration.linear.y;
  state.acceleration.linear.z = msg->acceleration.linear.z;
  state.attitude.z = msg->attitude.z;
  state.velocity.angular.z = msg->velocity.angular.z;
}

void callback2(const quad::quad_state_msg::ConstPtr& msg)
{
  desired.pose.position.x = msg->pose.position.x;
  desired.pose.position.y = msg->pose.position.y;
  desired.pose.position.z = msg->pose.position.z;
  desired.velocity.linear.x = msg->velocity.linear.x;
  desired.velocity.linear.y = msg->velocity.linear.y;
  desired.velocity.linear.z = msg->velocity.linear.z;
  desired.acceleration.linear.x = msg->acceleration.linear.x;
  desired.acceleration.linear.y = msg->acceleration.linear.y;
  desired.acceleration.linear.z = msg->acceleration.linear.z;
  desired.attitude.z = msg->attitude.z;
  desired.velocity.angular.z = msg->velocity.angular.z;
  desired.mode = msg->mode;
}

void LQR(void)
{
  // LQR Control
  // Calculate error
  float error_x = desired.pose.position.x - state.pose.position.x;
  float error_y = desired.pose.position.y - state.pose.position.y;
  float error_z = desired.pose.position.z - state.pose.position.z;
  float error_vx = desired.velocity.linear.x - ...
     state.velocity.linear.x;
  float error_vy = desired.velocity.linear.y - ...
     state.velocity.linear.y;
  float error_vz = desired.velocity.linear.z - ...
     state.velocity.linear.z;
  float error_psi = desired.attitude.z - state.attitude.z;

  // Integration
  if (desired.mode != 0)
  {
    error_xi += error_x * 0.01;
    error_yi += error_y * 0.01;
    error_zi += error_z * 0.01;
    error_psii += error_psii * 0.01;
  }
  else
  {
    error_xi = 0;
    error_yi = 0;
```

```
  error_zi = 0;
  error_psii = 0;
}

float g1 = 5.0, g2 = 4.74, g3 = 4.74;
float u1, u2, u3, u4;
u1 = g1 * error_x + g2 * error_vx + g3 * error_xi;
u2 = g1 * error_y + g2 * error_vy + g3 * error_yi;
u3 = 8.0 * error_z + 4.27 * error_vz + 4.86 * error_zi;
u4 = 8.0 * error_psi + 4.86 * error_psii;

// Anti-Windup
double saturated_u1 = CONSTRAIN(u1, -1, 1);
double saturated_u2 = CONSTRAIN(u2, -1, 1);
double saturated_u3 = CONSTRAIN(u3, -5, 5);
double saturated_u4 = CONSTRAIN(u4, -4, 4);
int clamp_x = antiWindup(error_x, u1, saturated_u1);
int clamp_y = antiWindup(error_y, u2, saturated_u2);
int clamp_z = antiWindup(error_z, u3, saturated_u3);
int clamp_psi = antiWindup(error_psi, u4, saturated_u4);

if (clamp_x == 1)
{
  // integral part is set to zero
  error_xi = 0;
  u1 = g1 * error_x + g2 * error_vx;
}
if (clamp_y == 1)
{
  // integral part is set to zero
  error_yi = 0;
  u2 = g1 * error_y + g2 * error_vy;
}
if (clamp_z == 1)
{
  // integral part is set to zero
  error_zi = 0;
  u3 = g1 * error_z + g2 * error_vz;
}
if (clamp_psi == 1)
{
  // integral part is set to zero
  error_psii = 0;
  u4 = g1 * error_psi;
}

// Feedforward
float up_1, up_2, up_3, up_psi;
up_1 = u1 + desired.acceleration.linear.x;
up_2 = u2 + desired.acceleration.linear.y;
up_3 = u3 + desired.acceleration.linear.z + 9.81;
up_psi = u4 + desired.velocity.angular.z;

// Inverse Mapping
float z1, z2, z3;
float thrust_cmd = mass * sqrt(up_1*up_1 + up_2*up_2 + up_3*up_3);
thrust_cmd = CONSTRAIN(thrust_cmd, 0, 15);
```

```
  z1 = (mass/thrust_cmd) * (up_1*cos(state.attitude.z) + ...
     up_2*sin(state.attitude.z));
  z2 = (mass/thrust_cmd) * (-up_1*sin(state.attitude.z) + ...
     up_2*cos(state.attitude.z));
  z3 = (mass/thrust_cmd) * up_3;
  z1 = CONSTRAIN(z1, -1, 1);
  z2 = CONSTRAIN(z2, -1, 1);
  z3 = CONSTRAIN(z3, -1, 1);
  float phi_cmd = asin(z2);
  float theta_cmd = atan(-z1/z3);
  float r_cmd = up_psi*cos(theta_cmd)/cos(phi_cmd) + ...
     state.velocity.angular.y*sin(phi_cmd)/cos(phi_cmd);

  phi_cmd = CONSTRAIN(phi_cmd, -1.0, 1.0);
  theta_cmd = CONSTRAIN(theta_cmd, -1.0, 1.0);
  r_cmd = CONSTRAIN(r_cmd, -1, 1);

  quad_cmd.thrust_des = thrust_cmd;
  quad_cmd.phi_des = phi_cmd;
  quad_cmd.theta_des = theta_cmd;
  quad_cmd.r_des = r_cmd;
  quad_cmd.mode = desired.mode;
  pub.publish(quad_cmd);
}

int antiWindup(float error, float output, float saturatedOutput)
{
  int check1 = 0;
  int check2 = 0;
  int clamp = 0;

  // Check 1
  if (output != saturatedOutput)
  {
    check1 = 1;
  }
  else
  {
    check1 = 0;
  }

  // Check 2
  if (sign(error) != sign(output))
  {
    check2 = 1;
  }
  else
  {
    check2 = 0;
  }

  // Check 3
  if ((check1 * check2) == 1)
  {
    clamp = 1;
  }
  else
```

```
    {
      clamp = 0;
    }
    return clamp;
  }

  int sign(float num)
  {
    if (num > 0) return 1;
    if (num < 0) return -1;
    return 0;
  }

  double CONSTRAIN(double x, double min, double max)
  {
    if (x > max)
    {
      x = max;
    }
    else if (x < min)
    {
      x = min;
    }
    else
    {
      x = x;
    }
    return x;
  }
};

int main(int argc, char **argv)
{
  ros::init(argc, argv, "quad_control");
  quad_control quad;

  ros::Rate loop_rate(100);

  while (ros::ok())
  {
    quad.LQR();
    ros::spinOnce();
    loop_rate.sleep();
  }

  ros::shutdown();
  return 0;
}
```

C.4 quad_trajectory_generation.cpp

```
// Author: Abdullah Altawaitan
// Date: April 4, 2019

#include "ros/ros.h"
```

```
#include "quad/quad_state_msg.h"
#include "std_msgs/Int8.h"

double i = 0;
double k = 0;
double j = 0;
int mode = 0;
double w = 1.0;
class quad_trajectory_generation
{
public:
  ros::NodeHandle n;
  ros::Subscriber sub;
  ros::Publisher pub;
  quad::quad_state_msg state;

  quad_trajectory_generation()
  {
    sub = n.subscribe<std_msgs::Int8>("/quad_gui", 1000, ...
        &quad_trajectory_generation::callback, this);
    pub = n.advertise<quad::quad_state_msg>("quad_trajectory", 1000);
  }

  ~quad_trajectory_generation()
  {
    // Empty
  }

  void callback(const std_msgs::Int8::ConstPtr& msg)
  {
    mode = msg->data;
  }

  void publishGenerateTrajectory(void)
  {
    // Trajectory Generation
    if (mode != 0)
    {
      if (mode == 1)
      {
        state.pose.position.x = 0;
        state.pose.position.y = 0;
        state.pose.position.z = i;
        state.pose.position.z = CONSTRAIN(state.pose.position.z, 0, 1);
        state.velocity.linear.x = 0;
        state.velocity.linear.y = 0;
        state.velocity.linear.z = 0;
        state.acceleration.linear.x = 0;
        state.acceleration.linear.y = 0;
        state.acceleration.linear.z = 0;
        state.attitude.z = 0;
        state.velocity.angular.z = 0;
        state.mode = 1;
        // Increment
        i = i + 0.005;
      }
      else if (mode == 2)
```

```
  {
    state.pose.position.x = 0;
    state.pose.position.y = 0.5 * cos(k * w);
    state.velocity.linear.x = 0;
    state.velocity.linear.y = -0.5 * w * sin(k * w);
    state.acceleration.linear.x = 0;
    state.acceleration.linear.y = -0.5 * (w*w) * cos(k * w);
    state.pose.position.z = 1.0 + 0.5 * sin(k * w) * cos(k * w);
    state.velocity.linear.z = 0.5 * w * cos(2 * k * w);
    state.acceleration.linear.z = -(w*w) * sin(2 * k * w);
    state.attitude.z = 0;
    state.velocity.angular.z = 0;
    state.mode = 2;
    // Increment
    k = k + 0.01;
  }
  else if (mode == 3)
  {
    state.pose.position.x = 0;
    state.pose.position.y = 0;
    state.pose.position.z = 1.0 - j*0.2;
    state.velocity.linear.x = 0;
    state.velocity.linear.y = 0;
    state.velocity.linear.z = 0;
    state.acceleration.linear.x = 0;
    state.acceleration.linear.y = 0;
    state.acceleration.linear.z = 0;
    state.attitude.z = 0;
    state.velocity.angular.z = 0;
    state.mode = 3;
    // Increment
    if (state.pose.position.z > 0)
    {
      j = j + 0.01;
    }
    if (state.pose.position.z < 0.02)
    {
      state.mode = -1;
    }
  }
  else
  {
    state.pose.position.x = 0;
    state.pose.position.y = 0;
    state.pose.position.z = 0;
    state.velocity.linear.x = 0;
    state.velocity.linear.y = 0;
    state.velocity.linear.z = 0;
    state.acceleration.linear.x = 0;
    state.acceleration.linear.y = 0;
    state.acceleration.linear.z = 0;
    state.attitude.z = 0;
    state.velocity.angular.z = 0;
    state.mode = -1;
  }
}
else
```

```
    {
      state.pose.position.x = 0;
      state.pose.position.y = 0;
      state.pose.position.z = 0;
      state.velocity.linear.x = 0;
      state.velocity.linear.y = 0;
      state.velocity.linear.z = 0;
      state.acceleration.linear.x = 0;
      state.acceleration.linear.y = 0;
      state.acceleration.linear.z = 0;
      state.attitude.z = 0;
      state.velocity.angular.z = 0;
      state.mode = 0;
    }
    pub.publish(state);
  }

  double CONSTRAIN(double x, double min, double max)
  {
    if (x > max)
    {
      x = max;
    }
    else if (x < min)
    {
      x = min;
    }
    else
    {
      x = x;
    }
    return x;
  }
};

int main(int argc, char **argv)
{
  ros::init(argc, argv, "quad_trajectory_generation");
  quad_trajectory_generation quad;

  ros::Rate loop_rate(100);

  while (ros::ok())
  {
    quad.publishGenerateTrajectory();
    ros::spinOnce();
    loop_rate.sleep();
  }

  ros::shutdown();
  return 0;
}
```

C.5 quad_log.cpp

```
#include "ros/ros.h"
#include <fstream>
#include "quad/quad_state_msg.h"
#include "quad/quad_cmd_msg.h"
#include "sensor_msgs/Imu.h"

using namespace std;
ofstream datafile;

class quad_log
{
public:
  ros::NodeHandle n;
  ros::Subscriber sub;
  ros::Subscriber sub2;
  ros::Subscriber sub3;
  ros::Subscriber sub4;
  quad::quad_state_msg desired;
  quad::quad_state_msg actual;
  quad::quad_cmd_msg cmd;

  quad_log()
  {
    sub = n.subscribe<quad::quad_state_msg>("/quad_trajectory", ...
        1000, &quad_log::callback, this);
    sub2 = n.subscribe<quad::quad_state_msg>("/quad_state", 1000, ...
        &quad_log::callback2, this);
    sub3 = n.subscribe<quad::quad_cmd_msg>("/quad_cmd", 1000, ...
        &quad_log::callback3, this);
    sub4 = n.subscribe<sensor_msgs::Imu>("/quad_serial_imu", 1000, ...
        &quad_log::callback4, this);
  }

  ~quad_log()
  {
    // Empty
  }

  void callback(const quad::quad_state_msg::ConstPtr& msg)
  {
    desired.pose.position.x = msg->pose.position.x;
    desired.pose.position.y = msg->pose.position.y;
    desired.pose.position.z = msg->pose.position.z;
    desired.velocity.linear.x =  msg->velocity.linear.x;
    desired.velocity.linear.y =  msg->velocity.linear.y;
    desired.velocity.linear.z =  msg->velocity.linear.z;
    desired.acceleration.linear.x = msg->acceleration.linear.x;
    desired.acceleration.linear.y = msg->acceleration.linear.y;
    desired.acceleration.linear.z = msg->acceleration.linear.z;
    desired.attitude.z = msg->attitude.z;
    desired.mode = msg->mode;
  }

  void callback2(const quad::quad_state_msg::ConstPtr& msg)
  {
    actual.pose.position.x = msg->pose.position.x;
    actual.pose.position.y = msg->pose.position.y;
```

```
    actual.pose.position.z = msg->pose.position.z;
    actual.velocity.linear.x =  msg->velocity.linear.x;
    actual.velocity.linear.y =  msg->velocity.linear.y;
    actual.velocity.linear.z =  msg->velocity.linear.z;
    actual.acceleration.linear.x = msg->acceleration.linear.x;
    actual.acceleration.linear.y = msg->acceleration.linear.y;
    actual.acceleration.linear.z = msg->acceleration.linear.z;
    actual.attitude.z = msg->attitude.z;
  }

  void callback3(const quad::quad_cmd_msg::ConstPtr& msg)
  {
    cmd.phi_des = msg->phi_des;
    cmd.theta_des = msg->theta_des;
    cmd.r_des = msg->r_des;
  }

  void callback4(const sensor_msgs::Imu::ConstPtr& msg)
  {
    actual.attitude.x = msg->orientation.x;
    actual.attitude.y = msg->orientation.y;
  }

  void writeData(void)
  {
    datafile << desired.mode << ", " << desired.pose.position.x << ...
        ", " << desired.pose.position.y
     << ", " << desired.pose.position.z << ", " << ...
        desired.velocity.linear.x
     << ", " << desired.velocity.linear.y << ", " << ...
        desired.velocity.linear.z
     << ", " << cmd.phi_des << ", " << cmd.theta_des << ", "
     << desired.attitude.z << ", " << actual.pose.position.x
     << ", " << actual.pose.position.y << ", " << actual.pose.position.z
     << ", " << actual.velocity.linear.x << ", " << ...
        actual.velocity.linear.y
     << ", " << actual.velocity.linear.z << ", " << ...
        actual.attitude.x << ", "
     << actual.attitude.y << ", " << actual.attitude.z << endl;
  }
};

int main(int argc, char **argv)
{
  datafile.open("/home/altwaitan/catkin_ws/src/quad/vehicle1_data.txt");
  ros::init(argc, argv, "quad_log");
  quad_log quad;

  ros::Rate loop_rate(100);


  if (datafile.is_open())
  {
    datafile << "mode" << ", " << "xdes" << ", " << "ydes" << ", " ...
       << "zdes" << ", "
    << "vxdes" << ", " << "vydes" << ", " << "vzdes" << ", "
    << "phides" << ", " << "thetades" << ", " << "psides" << ", "
```

```
    << "x" << ", " << "y" << ", " << "z" << ", " << "vx" << ", "
    << "vy" << ", " << "vz" << ", " << "phi" << ", " << "theta"
    << ", " << "psi" << endl;
    while (ros::ok())
    {
      quad.writeData();
      ros::spinOnce();
      loop_rate.sleep();
    }
  }
  else
  {
    ROS_ERROR_STREAM("Unable to open datafile");
  }
  datafile.close();
  ros::shutdown();
  return 0;
}
```