Design, Development, and Analysis of an LQG Controller for MAV Stability and Control Using Visual and INS Sensor Fusion

a project presented to
The Faculty of the Department of Aerospace Engineering
San Jose State University

in partial fulfillment of the requirements for the degree
Master of Science in Aerospace Engineering

by

Aaron Mandeville

December 2021

approved by

Professor Long Lu

Faculty Advisor

Style Definition: TOC 2: Tab stops: 0.76", Left + 6.49", Right,Leader: …

Style Definition: TOC 1

Abstract

Design, Development, and Analysis of an LQG Controller for MAV Stability and Control Using Visual and INS Sensor Fusion

Aaron Mandeville

The work in this report serves to design and analyze an optimal linear quadratic regular (LQR) flight controller for the purpose of controlling a quadrotor aircraft. Work was done to characterize the 6 degree of freedom dynamics of the vehicle, after which a non-linear model was created in Simulink. The results of the non-linear model are comparable to documented literature when given a specific control input. After the model is linearized about the hover state condition, the linearized state-space is formulated. Comparison between the non-linear plant models display favorable and matching results. An AHRS complementary filter is designed for the purpose of estimating attitude and heading angles. The hardware for the physical quadcopter plant is assembled and custom Simulink s-functions are written to receive sensor data. Two Kalman filters are developed which estimate the x, y, and z position and velocity from the PMW3901 optical flow sensor and the HC-SR04 ultrasonic sensor; both of which are fused with the accelerometer to estimate the states. These estimates are used along with a developed optimal LQR controller to close the control loop. Initial tests of the flight controller embedded in rotorS simulator in Gazebo prove to be promising. The simulation displays the efficacy of the controller in controlling the attitude and position of the quadcopter. After simulating the controller in Gazebo, the controller logic is embedded into an Arduino Due and tested on the quadcopter that was constructed. The quadcopter fails to hover most likely due to the low electronic speed controller sample rate of 50 Hz, however suggestions on how to fix this issue are presented.

Deleted: ¶
The Designated Project Advisor(s) Approves the Project Titled ¶
¶
DESIGN, DEVELOPMENT, AND ANALYSIS OF AN LQG CONTROLLER FOR MAV STABILITY AND CONTROL USING VISUAL AND INS SENSOR FUSION¶
¶
by ¶
¶
Aaron Mandeville¶
¶
APPROVED FOR THE DEPARTMENT OF AEROSPACE ENGINEERING ¶
SAN JOSÉ STATE UNIVERSITY ¶
¶
December 2021¶
¶
¶
Prof. Long Lu→→Department of Aerospace Engineering Advisor¶
Page Break
¶

Deleted: BSTRACT

Acknowledgements

I would like to thank Professor Lu whose interest and expertise in control systems has led to my own curiosity and pursuit of knowledge in the field. I would also like to thank my family for supporting me on every path that I decide to take.

Deleted: ACKNOWLEDGEMENTS

Deleted: k

Table of Contents

Deleted: iv

List of Tables

Deleted: LIST OF TABLES

Formatted: Centered

Deleted: Table 2.1 – Summary of simulation parameter values from [19]→13¶
Table 4.1 – Closed Loop Pole Locations of the Quadrotor States→29¶
Table 5.1 – MATLAB System Identification Transfer Function Results→39¶

List of Figures

Deleted: LIST OF FIGURES

Nomenclature

Symbol	Definition	Units
ϕ	Roll angle	*rad*
$\dot{\phi}$	Roll angle rate	*rad/s*
θ	Pitch angle	*rad*
$\dot{\theta}$	Pitch angle rate	*rad/s*
ψ	Yaw angle	*rad*
$\dot{\psi}$	Yaw angle rate	*rad/s*
z	Altitude	*m*
$\dot{z}$	Vertical velocity	*m/s*
x	Forward direction	*m*
$\dot{x}$	Forward direction velocity	*m/s*
y	Sideways direction	*m*
$\dot{y}$	Sideways direction velocity	*m/s*
k	Propeller thrust constant	
ω_i	Propeller angular velocity	*rad/s*
R_b^E	Rotation matrix from body to inertial	
g	Gravity	m/s^2
C_D	Drag coefficient	
l	Length of quadcopter arm	*m*
b	Thrust constant	
c	Drag torque constant	
τ_ϕ	Roll moment	*Nm*
τ_θ	Pitch moment	*Nm*
τ_ψ	Yaw moment	*Nm*
u_1	Collective thrust control	*N*
u_2	Roll control	*Nm*
u_3	Pitch control	*Nm*
u_4	Yaw control	*Nm*
M_{gb}	Body gyroscopic effect	*Nm*
M_{gp}	Propeller gyroscopic effect	*Nm*
J_r	Quadcopter inertia matrix	$kg*m^2$
Ω	Body angular velocity	*rad/s*
ζ	Position vector	*m*
η	Rotation vector	*rad*
I_M	Propeller inertia	$kg*m^2$
I_{xx}	Moment of inertia about x axis	$kg*m^2$
I_{yy}	Moment of inertia about y axis	$kg*m^2$
I_{zz}	Moment of inertia about z axis	$kg*m^2$
A_x	Drag constant about x axis	*kg/s*
A_y	Drag constant about y axis	*kg/s*
A_z	Drag constant about z axis	*kg/s*

Formatted: Font: Bold

Co	Controllability Matrix	
Ob	Observability Matrix	
J_{LQR}	LQR Cost Function	
K	LQR Gain Matrix	
Q	LQR State weight Matrix	
R	LQR Control Weight Matrix	
q	Quaternion	
$g(x, u, \Delta t)$	State transition matrix	
w_k	Zero-mean Gaussian process noise vector	
v_k	Zero-mean Gaussian measurement noise vector	
b_a	Accelerometer bias	m/s^2
θ_x	Aperture angle offset	*rad*
θ_{px}	Camera aperture angle	*rad*
Δn	Optical flow pixel velocity	*pixels/second*
N_x	Number of pixels in camera focal plane	*pixels*

Chapter 1 - Introduction

1.1.Motivation

Micro aerial vehicles (MAVs) are small fixed-wing or rotor aircraft that are used in applications such as surveillance or navigation in hazardous environments [1]. Commonly, these aircraft are developed in the form of quadrotor aircraft, as these have the benefit of vertical takeoff and landing (VTOL) as well as possessing agility and hover capabilities. These systems have become quite popular in the last decade, and a large amount of research is arising where these vehicles are seeing a lot of use, especially in the domain of autonomous flight. With the rise of micro-electro-mechanical IMUs (MEMS) these MAVs can be built in increasingly smaller form factors allowing for cheaper costs of entry in the field of autonomous flight research. This opens the door to many hobbyists and student researchers that want to use the quadrotor platform for experiments.

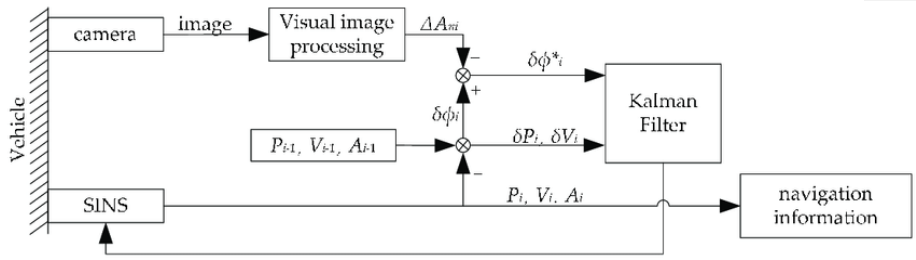


Figure 1.1 – Block diagram of a basic Visual-Inertial Navigation System [2]

Especially prevalent in the field of autonomous quadcopter research is the use of a small downward-facing onboard camera for autonomous visual-inertial navigation systems (VINS). These systems function by estimating velocity using computer vision techniques such as optical flow. The estimates are then fed into a filter (typically a Kalman filter) and fused with state estimates from the onboard inertial navigation system which results in both accurate and robust state estimates. A simple block diagram of a VINS architecture is shown in Figure 1.1. In this case, the strapdown inertial navigation system (SINS) and the attitude state estimates from the visual image processing unit are fused with a Kalman filter to produce the required navigation information. This fused navigation information helps to reduce random drift errors in the SINS gyroscope, and thus also reduces accumulated navigation errors during continuous operation where the inertial-navigation system would otherwise be used alone [2]. This type of visual-inertial navigation is especially important in GPS-denied environments, such as indoor flight or heavily concealed areas where GPS cannot penetrate, such as forests or dense cities.

Though state estimation can be performed as a stand-alone calculation, it is beneficial to use these accurate state estimates for loop closure of autonomous control systems. Loop closure is an important part controlling any system, especially an inherently unstable system such as a quadrotor aircraft. As the name suggests, a quadrotor aircraft consists of four motors which drive

four propellers. Since the quadrotor has six degrees of freedom and only four actuators, it is a underactuated system. Inherently, this system is unstable, complex, and the rotational and translational dynamics are coupled [3]. However, the system is still controllable. As such, state estimates from a visual-inertial navigation system can be used to produce state error estimates which can be fed through a closed-loop flight controller to allow for quadrotor stability and control. The robustness and effectiveness of the control system is related to the quality of the data that is being fed from the navigation and state estimation systems. The use of these two systems in conjunction has led to many deep investigations in autonomous flight, as well as led to the development and creation of companies which use the VINS architecture for their navigation and control solutions.

Applications for research involving autonomous quadrotor navigation using a VINS include autonomous commercial transportation, package delivery, target-following camera, and autonomous surveying of unmapped target areas. Several companies, such as A^3, Kitty Hawk, and Skydio, are developing navigation systems for autonomous VTOL aircraft which use computer vision solutions for loop closure. These systems are scalable and open the door for many new industries in the future.

Deleted: ¶
¶

1.2. Literature Review

Typical aircraft systems heavily utilize GPS which provides acceptable position measurements for most uses. However, GPS errors can grow rather large and are affected by several errors, such as ionosphere and troposphere delays, and can experience complete dropout when a system is navigating through tunnels, parking garages, canyons, or other areas where there is no direct upward path to the sky [4]. GPS also suffers from low sample rates which can make it difficult to use this measurement data for a closed-loop control system that uses state feedback. To solve this issue, most systems typically use both GPS as well as an onboard inertial navigation system.

These systems can measure position, velocity, and acceleration at high sample rates, and the states are all estimated through dead reckoning, which determines the states of a moving system by recursively measuring the states with respect to their initial values [5]. Combined with an IMU these MEMS systems provide information about the full six degrees of freedom of the system. Though convenient in size and weight, as stated in the previous section, these MEMS suffer from severe drift, bias, and are prone to the vibrational perturbations of MAV motors and other driving mechanisms [6]. Because of this, it can be convenient, and sometimes necessary, to fuse measurements from the inertial navigation system (INS) with GPS measurements. Fusing these measurements together provides the robustness of the GPS measurements, while also providing high-frequency updates through the INS measurements. The GPS works to correct for the INS drift and bias by re-initializing the initial conditions, in a sense. Though this method of state estimation works great indoors, as stated previously, errors will start to accumulate if a system is operating in a GPS-denied environment [5]. Consequently, it may become necessary to fuse the INS measurements with another form of measurement that can provide the position-fixing that is usually provided by the GPS. Recent research has focused heavily on the use of onboard monocular cameras to provide position fixing of the system.

Visual measurements from an on-board camera can be used to produce accurate attitude and position state estimations which can then be fused with the information from the INS and fed back in a closed-loop form [6, 7]. These sensors are cheap, lightweight, and passive which makes them desirable for sensor fusion in the state estimation of MAVs or other small vehicles.[8] The sensor fusion can either be tightly or loosely coupled. Tightly coupled sensor fusion approaches seek to directly fuse the visual and inertial measurements within a single process which results in a higher overall accuracy. Comparatively, with loosely coupled sensor fusion, each navigation system (the INS and visual state estimation system) infers the motion of the system, and the result is then fused together to infer the overall motion constraints of the system [6]. Regardless of whether the fusion technique is tightly or loosely coupled, any system that fuses together INS data and visual data is called a visual-inertial navigation system (VINS).

1.2.1. Vision-Based Navigation Systems

The visual state estimates that are used in VINS are generated using computer vision algorithms. These algorithms function by detecting and then tracking features (such as edges or corners) to calculate the distance the feature moves on the camera focal plane. Once the distance is estimated, using the intrinsic and extrinsic properties of the camera, the physical motion of the system can be calculated by converting distance traveled in the focal plane coordinate frame to inertial frame coordinates. It is important that the features that are being tracked are good features, as the effectiveness and accuracy of vision-based navigations systems relies heavily on the accuracy of the feature tracking algorithms [9]. As such, a lot of time has been spent developing robust and accurate feature detectors and trackers for use in vision-based systems. There are several different algorithms that can be used for feature detecting and tracking, however, commonly used is the Shi-Tomasi corners detector and the Kanade-Lucas-Tomasi (KLT) feature tracker.

VINS and vision-aided stability and control is a popular solution for closed-loop navigation in GPS-denied environments such as indoors, underwater, or in mountainous regions, as the visual measurement updates can be used for position fixing of the inertial sensors. In [10], a vision-aided stabilization system is development which greatly reduces MAV translational drift over time. In typical systems, an IMU is used to maintain level attitude flight by correcting for small biases and errors in the pitch, roll, and yaw measurements, however, the IMU cannot correct for small drifts in translational motion. With the presence of wind gusts and air drafts, the position of the MAV may begin to deviate from the desired location. As such, just as the IMU is used to correct for the attitude of the MAV, there needs to be a system that can correct for drift in the position of the MAV. The system works by using the Harris corner detection algorithm to track feature movement between consecutive images from a downward-facing camera. From these tracked features, deviations in the x and y axes can be calculated using sum of absolute differences template matching, and then sent to the IMU for correction. This type of vision-aided stabilization is powerful and computationally inexpensive and offers many benefits for MAV systems which have tight weight limitations.

More advanced algorithms can estimate and correct for MAV attitude, position, velocity, and IMU sensor bias using a sensor fusion formulation for correction. Typically, the fusion technique is developed in the form of an extended Kalman filter (EKF) or unscented Kalman

filter (UKF) [5, 10]. One of the earliest forms of this type of navigation solution is the multi-state constraint Kalman filter (MSCKF). The algorithm propagates the MAV state using quaternion-based inertial dynamics and uses an EKF for the state update. Many other algorithms have been built upon the MSCKF, such as the square root inverse sliding window filter (SR-ISWF) and the optimal state constraint (OSC)-EKF.

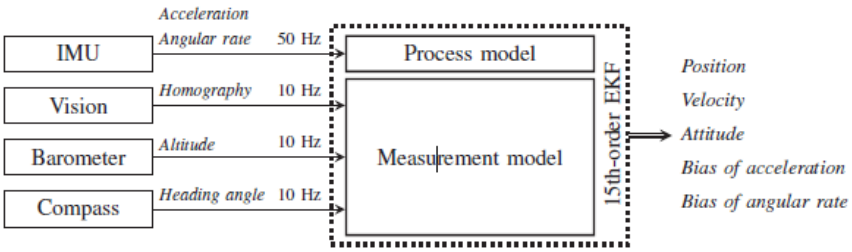


Figure 1.2 – Block diagram of a vision-aided navigation system which uses an EKF to fuse image homography measurements with IMU, magnetometer, and barometer measurements. [11]

In [11], a robust method is presented which performs autonomous UAV stabilization in the absence of GPS. Homgraphy decomposition is used to retrieve navigational information which is buried in the image homography measurements, namely the rotation and translation matrices of the current state. An EKF architecture is used to fuse the image homography measurements with onboard IMU, magnetometer, and barometer data. Figure 1.2 shows a simplified diagram of the EKF used in [11]. An EKF process model is used which takes image homography, IMU, magnetometer, and barometer measurements as inputs, and outputs position velocity and attitude measurements with reduced bias and drift. The resulting fused data provides more accurate state estimates when compared to those of a system which does not use vision-aided navigation. These state estimates can be used in a flight controller to allow for a system that is more stable and has a higher resistance to drift.

Many flight controller architectures may be used for low-level control of a MAV that utilizes VINS. These control schemes include the following: PID, linear quadratic regulator/Gaussian (LQR/LQG), sliding mode, backstepping, feedback linearization, adaptive, robust, optical, L1, H∞, fuzzy logic, and artificial neural network [12]. Each one of these controllers has pros and cons that come with their implementation. Work done with LQR controllers shows very capable stability and control of a MAV, however, this control method lacks robustness once the aircraft deviates from its flight path. Compared to a traditional PID controller, the LQR controller performs significantly better due to modeling the whole quadrotor attitude and translational dynamics as coupled compared to the simplified and decentralized non-optimal dynamics and control methods used with the PID controller [13]. The LQR controller performs well even when subject to wind and other environmental disturbances [14, 15]. An LQG controller with integral action shows promising results for attitude stabilization of a MAV during hover. The purpose of the integral action is to reject disturbances in the input or reference signals. It helps to ensure that there is unitary closed-loop gain. LQG is a beneficial control

scheme, as the system does not require complete state implementation since the Kalman filter reconstructs the missing states. This can prove beneficial for systems which are missing certain sensors and cannot measure specific states.

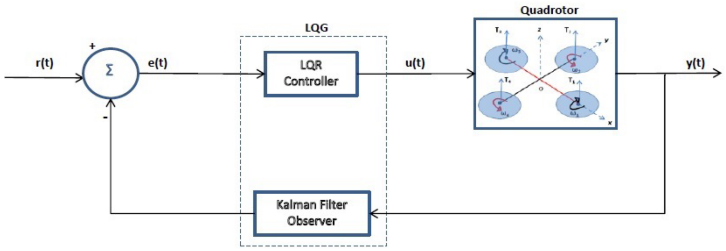


Figure 1.3 – Simplified block diagram of LQG controller for quadrotor control [12].

1.2.2. LQG Controller

An LQG controller is one that fuses together a LQR controller with a state observer, typically in the form of a Kalman filter. Compared to an LQR controller, the LQG controller is advantageous as it can be used for systems which do not have complete state measurements or systems which have a large amount of sensor noise [16]. The Kalman filter observer can be used to smooth out the data. Another benefit of this type of system is it is fault tolerant. Assuming the dynamics have been correctly linearized, an LQG controller can provide state feedback even if a particular sensor which measures a state has been damaged or has malfunctioned. By nature of the controller architecture, an LQG controller provides an optimal solution to a control problem. The controller does this by minimizing a desired performance index. The specifics of this topic will be expanded on in later chapters, as the focus of this project will include LQG controller design.

Effective use of an LQR or LQG controller first requires the linearization of the highly non-linear quadrotor dynamics. The complex dynamics of the system can be broken apart and linearized into multiple smaller subsystems, as demonstrated in [17] where six total LQR controllers were used for the balancing control. In this approach, each body angle is controlled with cascaded LQR and integral LQR controllers. Results prove promising for both steady-state and transient control. Other modelling approaches focus on controlling the system using a full twelve-state representation of the dynamics, which includes drag forces as well as external torque effects [1]. Though simple and effective, linearization does have limitations, and errors begin to increase as the system deviates from its linearization point. For systems that require quick movements and fast flight, non-linear modelling and control approaches can offer a more robust and accurate solution [18].

1.3. Research Goals

The goal of this research is to create an optimal control strategy for a quadrotor aircraft which uses monocular video fused with inertial navigation system sensors for closed-loop feedback control.

The system will be able to operate in a GPS-denied environment and maintain stability and tracking during autonomous operations. Ideally, the system will also possess a forward-facing camera which aids in localization and tracking using computer vision techniques. Based on the information gathered from the literature review, an LQG controller will be used for stabilization and control of the quadrotor. A MEMS device which has an IMU, accelerometer, and magnetometer will be used for measuring translational and rotational states. A barometer can also be used for estimating height. Along with the MEMS device, the quadrotor will be fit with a downward facing camera which will use optical flow and other computer vision techniques to estimate velocity in the x and y directions. This velocity estimate will be used to estimate other states variables. The estimation technique that will be used will be a standard Kalman filter, with some investigations into EKF as well. Comparisons between the standard LQG controller and the controller which uses visual state feedback will be done to quantify the effectiveness of the state estimation technique.

1.4. Methodology

Below is a breakdown of the different steps that must be completed to produce the final system. First steps include analysis and modelling of the non-linear 6DOF system followed by the calculation and modelling of the linearized state-space system Analysis will be carried out to prove the linearized model accurately captures the quadrotor dynamics at the chosen equilibrium point. After verification of the dynamics, an LQG controller will be developed for 6DOF stabilization and control. Once this is achieved, work will begin on constructing a quadrotor testbed which the flight controller can be installed into. After, software will be developed for the VINS system which will provide guidance and navigation inputs for the flight computer that was developed. Lastly, the project will conclude with tests and comparisons between the system operating on INS versus VINS. A detailed explanation of each of the project goals can be read below.

1.) Calculation of the 6DOF MAV Dynamics

To linearize and model any system, a deep understanding of the vehicle dynamics is needed. Ideally before creating a control architecture, it is important to reduce the second-order vehicle dynamics into a state-space representation of first-order equations. Quadrotors are capable of roll motion, pitch motion, yaw motion, horizontal motion, and vertical motion. All these motions must be modeled correctly to achieve an accurate state space representation of the system.

2.) Development of the 6DOF LQG Controller for MAV

Once the vehicle dynamics have been determined, work can be started on developing a control architecture for the MAV. Initially, a simple attitude and altitude hold autopilot can be developed, and from there, the controller scheme can be extended to perform more complicated actions. This initial controller will just use measurement feedback from the on-board MEMS unit to estimate states to prove functionality of the system without visual state estimation.

3.) Develop the MAV Testbed

An important part of testing is having a platform to test the control schemes on. As such, parts need to be chosen which can be used to control a physical system. Specifically, a processor, flight stack for communication with ground and controller networks, inertial measurement sensors, and MAV hardware all need to be purchased and assembled to create a bed on which the flight software can be uploaded to.

4.) Software Development of the VINS

After uploading the INS-only control scheme to the MAV control board and proving functionality, the next step is to begin work on programming a control system which fuses both the INS and the visual camera for state estimation which will be used for feedback in the LQG control architecture. First attempts will try and use the camera to calculate horizontal velocity, which can then be used to estimate the odometry of the MAV through integration of the acquired velocity. Other iterations of the design will also attempt to investigate using the camera to estimate rotation rates to try and smooth out the angular body rate measurements that will come from the IMU. The fusion of the INS and camera will contribute to the MAV attitude stability, while also allowing for position estimation and tracking.

5.) Final testing and Comparison of INS and VINS

Once both systems are developed, it will be interesting to compare the performance of the two. Specifically, estimated attitude and positions will be compared to the commanded reference signals to quantify the error in these state estimations. Additionally, the INS and VINS will be compared to each other to quantify performance error between the two.

The designed LQG flight controller requires the development of an autopilot controller as well as a means of fusing multiple sensor data together. Once these algorithms have been developed and simulated, work will begin on applying the control system to an actual quadrotor aircraft. This will include purchasing and assembling the necessary parts of the flight controller, as well as acquiring the correct framework of which the control algorithm can be uploaded to. It is assumed that the controller will need to be tuned for actual flight conditions, so some time will need to be spent fixing errors and fine-tuning parameters to optimize the controller and minimize the state estimation error.

Chapter 2 - Non-Linear Quadrotor Dynamics

2.1.Frame of Reference

The dynamic model of the quadrotor is presented in this chapter. As with modelling any dynamic system, it is important to determine the frames of reference associated with the vehicle. As is standard with most aircraft, the orientation of the quadcopter will be given in respect to the *NED* (North-East-down) World reference frame. This reference frame is fixed to the Earth and is defined with the positive x-axis pointing true North, the positive z-axis points to the center of the Earth, and the y axis completes the right-handed convention and points East. An image of the NED reference frame can be seen in Figure 2.1 below.

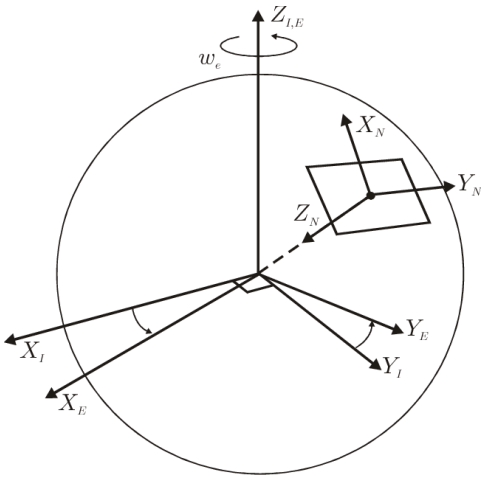


Figure 2.1 – Representation of the NED reference frame.

The body frame of the quadcopter is defined with the origin at the center of gravity of the body, the positive x-axis points out of the nose of the aircraft, the positive z-axis points down, and the positive y-axis completes the triad and is defined by the right-hand rule. With this frame, right-handed rotations about x-axis, y-axis, and z-axis give positive roll (ϕ), pitch (θ), and yaw (φ) angles, respectively. The Euler angles can be used to relate the NED frame to the body frame through the a *Z-Y-X* (*R(φ)-R(θ)-R(ϕ)*) rotation sequence which is defined by the following equation:

$$R^{E/b} = \begin{bmatrix} c\psi c\theta & c\psi s\theta s\phi - s\psi c\phi & s\psi s\phi + c\psi s\theta c\phi \\ s\psi c\theta & c\psi\phi + s\psi s\theta s\phi & s\psi c\theta c\phi - c\psi s\phi \\ -s\theta & c\theta s\phi & c\theta c\phi \end{bmatrix} \quad (2.1)$$

where *cx* and *sx* represent *cos(x)* and *sin(x)*, respectively and $\eta = [\phi\ \theta\ \psi]^T$ is the vector of Euler angles.

2.2. Control Inputs and Equations of Motion

A quadcopter consists of four blades, with one set of clockwise- rotating blades and second set of counter-clockwise rotating blades. While there are many quadcopter body configurations, the quadcopter being studied in this research is one that is in a plus configuration. A description of the system can be seen in Figure 2.2 below.

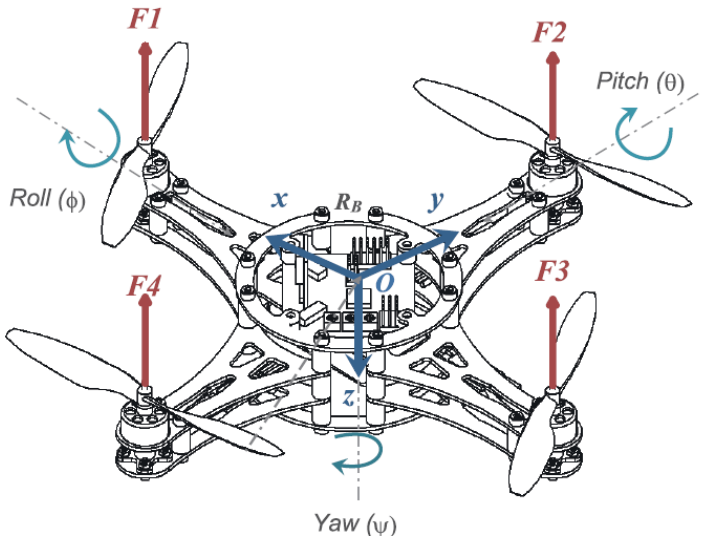


Figure 2.2 – Diagram of quadcopter configuration [1].

2.2.1. Forces and Torques

Regardless of body configuration, a quadrotor has four control inputs which can be used to control the six degrees of freedom of the body. As stated previously, this system is underactuated, so rotational and translational dynamics are coupled. The first control input, u_1, controls the thrust that the quadcopter generates. The thrust generated by the i[th] motor can be expressed in the quadcopter body frame as follows [1, 22]:

$$F_i = k\omega_i^2 \tag{2.2}$$

where k is the propeller lift coefficient and ω_i is the angular velocity of the i[th] motor. The total thrust in the quadcopter body frame is defined though equation (2.3) below.

$$F_T^b = \begin{bmatrix} 0 \\ 0 \\ \sum_{i=1}^{4} k\omega_i^2 \end{bmatrix} \tag{2.3}$$

In the inertial frame, the force of gravity can be expressed as:

$$F_g^E = m \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} \tag{2.4}$$

where m is the mass of the quadcopter system. Assuming that the speed of the quadcopter is slow and the aircraft is small, the drag force acting on the quadcopter body can be modelled as a viscous resistance which is expressed through equation (2.5):

$$F_d^E = \begin{bmatrix} C_{Dx}\dot{x} \\ C_{Dy}y \\ C_{Dz}z \end{bmatrix} \tag{2.5}$$

where C_{Dx}, C_{Dy}, and C_{Dy} represent the drag coefficients of the quadcopter body and x, y, z are the velocities of the body in the x, y, and z axes, respectively.

The quadcopter body is subject to three attitude torques due to the differences in the angular velocities of the four actuators. For a plus configuration body, the pitching torque is a function of the difference between the force generated by the first motor, and the force generated by the third motor. The rolling torque is a function of the difference of the force generated by the fourth motor, and the force generated by the second motor. Lastly, the yawing torque is a function of the difference between the clockwise-rotating motors and the counter-clockwise rotating motors. Equations of the roll, pitch, and yaw body torques are expressed as follows:

$$\tau_\phi = lb(\omega_4^2 - \omega_2^2) \tag{2.6}$$

$$\tau_\theta = lb(\omega_1^2 - \omega_3^2) \tag{2.7}$$

$$\tau_\psi = c(\omega_1^2 - \omega_2^2 + \omega_3^2 - \omega_4^2) \tag{2.8}$$

where l is defined as the distance from the center of each rotor to the center of gravity of the body, and c is a scaling factor for a motor-propeller set, defined in [20]. Vectorizing these equations produces the following equation which expresses the four control inputs as a function of motor angular velocities:

$$\begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} = \begin{bmatrix} k & k & k & k \\ 0 & -lb & 0 & lb \\ lb & 0 & -lb & 0 \\ c & -c & c & -c \end{bmatrix} \begin{bmatrix} \omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \\ \omega_4^2 \end{bmatrix} \tag{2.9}$$

Along with the body torques, there are also two gyroscopic effect torques due to the rotation of the quadrotor body (M_{gb}) and the rotation of the propellers (M_{gp}). These equations are shown as, respectively,

$$M_{gb} = -\Omega \times J\Omega \tag{2.10}$$

$$M_{gp} = \sum_{i=1}^{4} J_r \left(\Omega \times \begin{bmatrix} 0 \\ 0 \\ (-1)^{i+1}\omega_i \end{bmatrix}\right) \tag{2.11}$$

where J_r is the rotor inertia, ω_i is the angular velocity of the i[th] motor, $J = diag[J_x,\ J_y,\ J_z]$ is the inertia matrix of the quadrotor, and Ω denotes the angular velocity of the quadrotor in the body-fixed reference frame.

2.2.2. Newton-Euler Modelling Method

The six equations of motion which describe the quadrotor motion can be derived through the Newton-Euler equations. In the inertial frame, the only accelerations that are acting on the system are acceleration due to gravity, acceleration due to the quadrotor motors, and acceleration due to drag on the quadrotor body, shown below:

$$m\ddot{\zeta} = F_g + F_T + F_D \tag{2.12}$$

$$m\ddot{\zeta} = m\begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} - \left\{R^{E/b}\begin{bmatrix} 0 \\ 0 \\ u_1 \end{bmatrix}\right\} - \begin{bmatrix} C_{Dx}\dot{x} \\ C_{Dy}\dot{y} \\ C_{Dz}\dot{z} \end{bmatrix} \tag{2.13}$$

where $\zeta = [x, y, z]$ is the quadcopter position vector, and $R^{E/b}$ represents the rotation from body-fixed coordinate frames to the Earth-fixed frame. Multiplying out the equations produces the resulting equations for the x, y, and z accelerations in the inertial frame:

$$\ddot{x} = -\frac{1}{m}(s\phi s\psi + c\phi s\theta c\psi)u_1 - \frac{C_{Dx}\dot{x}}{m} \tag{2.14}$$

$$\ddot{y} = -\frac{1}{m}(c\phi s\theta s\psi - s\phi c\psi)u_1 - \frac{C_{Dy}\dot{y}}{m} \tag{2.15}$$

$$\ddot{z} = g - \frac{1}{m}(c\phi c\theta)u_1 - \frac{C_{Dz}\dot{z}}{m} \tag{2.16}$$

The rotational acceleration equations can be found in the same fashion. For this system, the three main torques acting on the body are the control torques due to actuator action, a body gyro effect torque, and a propeller gyro effect torque. By summing these equations together, a solution for the total torque acting on the quadrotor body can be found:

$$J\dot{n} = \tau_a + M_{gb} + M_{gp} \tag{2.17}$$

$$Jn = \begin{bmatrix} \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix} + -\Omega \times J\Omega + \sum_{i=1}^{4} J_r \left(\Omega \times \begin{bmatrix} 0 \\ 0 \\ (-1)^{i+1}\omega_i \end{bmatrix}\right) \tag{2.18}$$

where n is the angular acceleration vector. Multiplying out the terms produces the following result for the angular acceleration in each of the three axes:

$$\dot{\phi} = -\frac{\theta\psi(J_{zz} - J_{yy})}{J_{xx}} + \frac{\theta\Omega_p J_r}{J_{xx}} + \frac{u_2}{J_{xx}} \tag{2.19}$$

$$\theta = \frac{\phi\psi(J_{zz} - J_{xx})}{J_{yy}} - \frac{\phi\Omega_p J_r}{J_{yy}} + \frac{u_3}{J_{yy}} \tag{2.20}$$

$$\dot{\psi} = \frac{\theta\phi(J_{xx} - J_{yy})}{J_{zz}} + \frac{u_4}{J_{zz}} \tag{2.21}$$

where $\Omega_p = \omega_1 - \omega_2 + \omega_3 - \omega_4$.

After deducing the six equations of motion which describe the translational and rotational dynamics of the quadcopter, a non-linear state space representation can be formed. Taking equation (2.22) as the state space vector, the state space can be formed as follows:

$$X = \left(\phi, \dot{\phi}, \theta, \dot{\theta}, \psi, \dot{\psi}, z, \dot{z}, x, \dot{x}, y, \dot{y}\right)^T \in \mathbb{R}^{12} \tag{2.22}$$

$$X = f(X, u) = \begin{cases} x_1 = x_2 \\ x_2 = \frac{x_4 x_6(J_{yy} - J_{zz})}{J_{xx}} + \frac{\Omega_p J_r x_4}{J_{xx}} + \frac{u_2}{J_{xx}} \\ x_3 = x_4 \\ x_4 = \frac{x_2 x_6(J_{zz} - J_{xx})}{J_{yy}} - \frac{\Omega_p J_r x_2}{J_{yy}} + \frac{u_3}{J_{yy}} \\ x_5 = x_6 \\ x_6 = \frac{x_2 x_4(J_{xx} - J_{yy})}{J_{zz}} + \frac{u_4}{J_{zz}} \\ x_7 = x_8 \\ x_8 = g - \frac{1}{m} c x_3 c x_1 u_1 - \frac{C_{Dz}}{m} x_8 \\ x_9 = x_{10} \\ x_{10} = -\frac{1}{m}(c x_1 s x_3 + s x_1 s x_5) u_1 - \frac{C_{Dx}}{m} x_{10} \\ x_{11} = x_{12} \\ x_{12} = -\frac{1}{m}(s x_3 s x_5 c x_1 - c x_5 s x_1) u_1 - \frac{C_{Dy}}{m} x_{12} \end{cases} \tag{2.23}$$

2.3. Non-Linear Open-Loop Simulink Model and Results

After designing the non-linear plant in Simulink, it is important to compare the results obtained to those from a published paper. Since the plant is open loop and a controller has yet to be designed, the control inputs will be in the form of motor angular velocities. For the test, the control input from [19] was used along with identical quadcopter parameters, shown in table 2.1.

Table 2.1 – Summary of simulation parameter values from [19]

Parameter	Description	Value	Unit
g	Gravity	9.81	m/s^2
m	System mass	0.468	kg
l	Length from motor to center of mass	0.225	m
k	Thrust constant	$2.980 * 10^{-6}$	
b	Drag constant	$1.140 * 10^{-7}$	
I_M	Propeller inertia	$3.357 * 10^{-5}$	$kg * m^2$
I_{xx}	Quadcopter inertia about x-axis	$4.856 * 10^{-3}$	$kg * m^2$
I_{yy}	Quadcopter inertia about y-axis	$4.856 * 10^{-3}$	$kg * m^2$
I_{zz}	Quadcopter inertia about z-axis	$8.801 * 10^{-3}$	$kg * m^2$
A_x	Drag constant about x-axis	0.25	kg/s
A_y	Drag constant about y-axis	0.25	kg/s
A_z	Drag constant about z-axis	0.25	kg/s

Formatted Table

The control inputs for each motor can be seen in the Figure 3.1 below. The first 0.5 seconds of the control input commands each of the four motors to change their speeds in a sine wave motion. Since the first control input controls the upwards thrust of the quadcopter, as seen in equation (2.3), it is expected that the quadcopter will increase and then decrease altitude. Next, motors two and four were commanded to change, while one and three were left the same. Looking at equation (2.5), the expected output is a net torque in the rolling direction. The third command consists of the same behavior as the last, however motors two and four stay constant while motors one and three experience changes. Based on equation (2.6), this should create a net pitching torque. Lastly, a net yawing torque (2.7) is induced by changing the speed of the rotating (one and three) and counter-rotating (two and four) motor pairs.

Since the plant is open loop and a controller has yet to be designed, the control inputs will be in the form of motor angular velocities. the results were compared with the results obtained from [19]. As seen in figure 3.2 below, the results are similar.

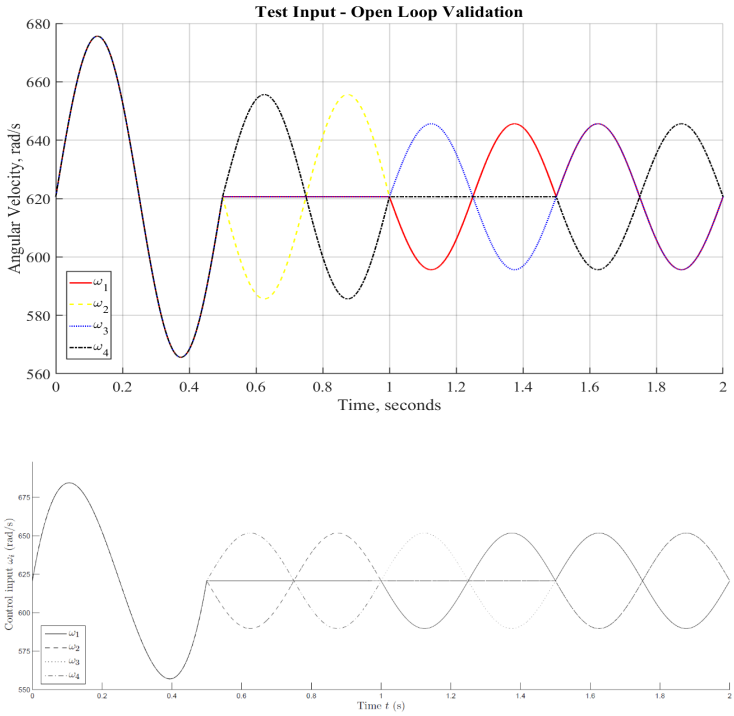


Figure 2.3 – Angular velocities of the four quadcopter motors used for validation from [19].

Once the desired control is input to the non-linear plant, values for the resulting translational and rotational positions can be obtained. Looking at Figure 2.4, it is seen that in response to the first control input u_1, the quadcopter increases in altitude, and subsequently decreases in altitude back to zero. The changes in control inputs $u_2, u_3,$ and u_4 cause the roll, pitch, and yaw angles to change which consequently causes the x and y positions to change since they translational and rotational dynamics of the quadcopter are coupled. Comparing the plots generated to those from [19], Figure 2.4 shows the quadcopter experiences comparable translational changes when subject to the test control inputs.

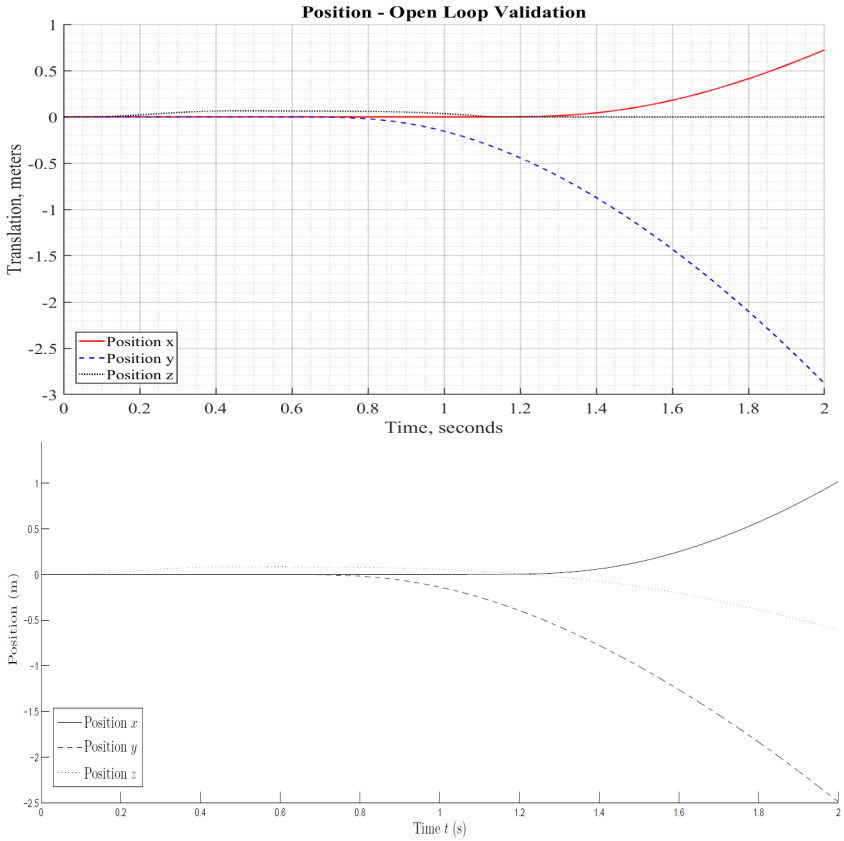


Figure 2.4 – Comparison of quadcopter displacement with [19] given the specified control input.

When comparing the quadcopter Euler angles to those presented in the literature, it is shown that the change in the angles is comparable to that of the literature. Increasing all motors by the same amount has no effect on the Euler angles, as apparent by the Figure 2.5. Alternatively, increasing motors two and four causes an increase in the roll angle, which causes an acceleration in the negative y direction and an increase in the roll angle of 25 degrees. The same behavior is seen with the pitch angle which increases as motors one and three increase, which causes a net acceleration in the positive x direction and increases the pitch angle by about 20 degrees. Increasing motors one and three and decreasing motors two and four causing a change in the quadcopter yaw angle of around 5 degrees. The accelerations in the x, y, z directions and the rotational accelerations are halted by decreasing the motor speeds in a similar fashion.

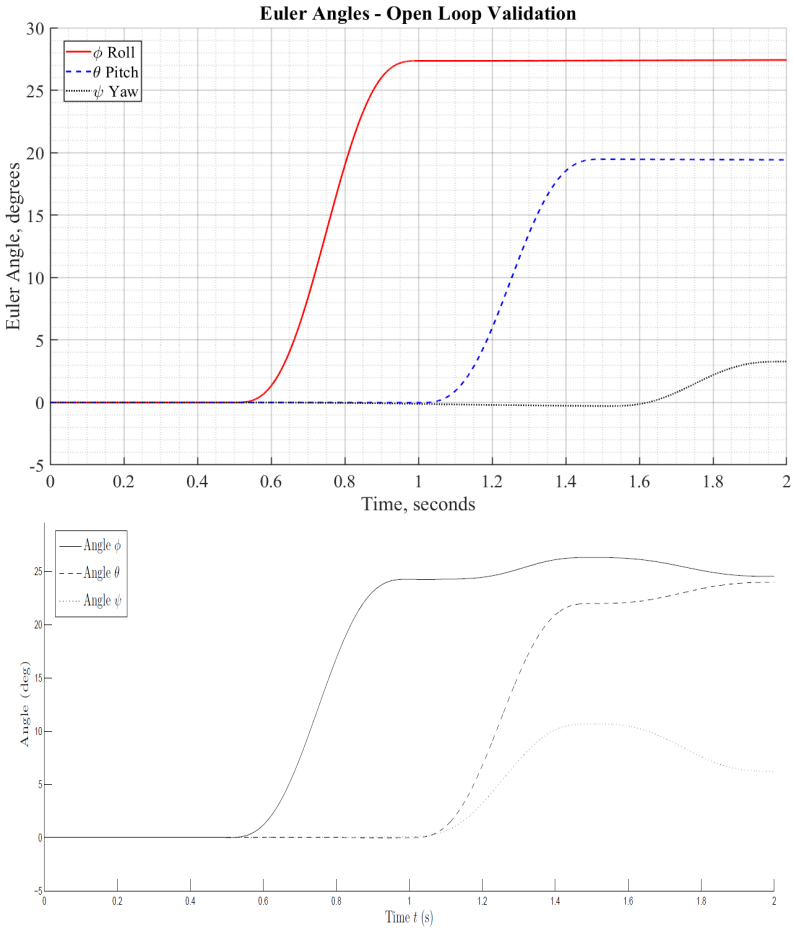


Figure 2.5 – Comparison of quadcopter Euler angles with [19] given the specified control input.

Overall, the results obtained from the non-linear open-loop simulation are comparable to the results from benchmarked data. This shows that the model that was created is sufficient for capturing the non-linear dynamics of the quadcopter. Chapter 3 will cover linearization of the non-linear state space around a chosen trim condition.

Chapter 3 - Linearized Quadcopter Dynamics

3.1. Linearization

Though the non-linear dynamics of the quadcopter have been successfully captured, linear control methods cannot be applied to the system until the dynamics have been linearized around a specified trim condition. As such, it is important to derive the linearized state-space of the quadcopter. For the purposes of this project, the trim condition that will be used to linear the model will be one where the quadcopter is in a stable hover at a set altitude, z. In this condition, the linear and rotational velocity of the quadcopter body are zero, as well as the attitude angles. Since the rotational and translational dynamics of the system are coupled, it is necessary that the attitude angles are zero so that there is no net acceleration in the x and y axes. Though the yaw angle ψ is decoupled from the translational dynamics, this angle was also set to zero for simplicity. The trimmed state-space is shown below with equation (3.1):

$$X^{*} = (0, 0, 0, 0, 0, 0, z, 0, x, 0, y, 0)^{T} \in \mathbb{R}^{12} \tag{3.1}$$

To achieve this desired trim condition, the control vector **u** must be set such that

$$\sum F_z = mg = u_1 = k(\omega_1^2 + \omega_2^2 + \omega_3^2 + \omega_4^2). \tag{3.2}$$

As such, the trim condition is achieved and maintained with the constant control input value (3.3) [21]:

$$\begin{matrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{matrix} = \begin{matrix} mg \\ 0 \\ 0 \\ 0 \end{matrix}. \tag{3.3}$$

The equilibrium operating conditions (x_0, u_0) is then defined as follows [1]:

$$(x_0, u_0) = \begin{cases} x_1, x_2, x_3, x_4, x_5, x_6, x_8, x_{10}, x_{12} = 0 \\ x_7, x_9, x_{11} = constant \\ u_1^0 = mg \\ u_2^0, u_3^0, u_4^0 = 0 \end{cases} \tag{3.4}$$

Solving for the motor speed ω_i provides the motor speed that is necessary to counteract the weight of the quadcopter and achieve a hover condition. For the benchmark test case, the motor speed for each of the four motors required to achieve a hover state is $620.62\ rad/s$.

After deciding on the trim condition, the non-linear state-space model is then linearized about a constant position and attitude state. The non-linear model $f(X, u)$ will be linearized with the theory of small oscillations, where the sine of an angle is approximated as the angle itself, and the cosine of an angle is approximated as unity [21]. It is important to note that this linearized approximation of the non-linear model is only accurate within small deviations from the trim conditions. If states stray too far from the trim condition, errors and inaccuracies will

begin to accumulate and the linearized model will no longer be valid. Applying the small angle approximation to the non-linear model results in the following set of linearized equations:

$$\dot{X} = f(X,u) = \begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = \dfrac{\Delta u_2}{J_{xx}} \\ \dot{x}_3 = x_4 \\ \dot{x}_4 = \dfrac{\Delta u_3}{J_{yy}} \\ \dot{x}_5 = x_6 \\ \dot{x}_6 = \dfrac{\Delta u_4}{J_{zz}} \\ \dot{x}_7 = x_8 \\ \dot{x}_8 = -\dfrac{\Delta u_1}{m} - \dfrac{C_{Dz}}{m} x_8 \\ \dot{x}_9 = x_{10} \\ \dot{x}_{10} = -g x_3 - \dfrac{C_{Dx}}{m} x_{10} \\ \dot{x}_{11} = x_{12} \\ \dot{x}_{12} = g x_1 - \dfrac{C_{Dy}}{m} x_{12} \end{cases} . \quad (3.5)$$

The linearized equations can then be put in a linearized state-space form which is in the form of equation (3.6) with the matrices associated of the linear system given by equations (3.7), (3.1.8), (3.9) and (3.10):

$$\begin{cases} \dot{x} = \boldsymbol{A}x + \boldsymbol{B}u \\ y = \boldsymbol{C}x + \boldsymbol{D}u \end{cases} \quad (3.6)$$

$$\boldsymbol{A} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\frac{C_{Dz}}{m} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & -g & 0 & 0 & 0 & 0 & 0 & 0 & -\frac{C_{Dx}}{m} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ g & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\frac{C_{Dy}}{m} \end{bmatrix} \in \mathbb{R}^{12x12} \tag{3.7}$$

$$\boldsymbol{B} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & \frac{1}{Ixx} & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{Iyy} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{Izz} \\ -\frac{1}{m} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \in \mathbb{R}^{12x4} \tag{3.8}$$

$$\boldsymbol{C} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \in \mathbb{R}^{12x12} \quad (3.9)$$

$$\boldsymbol{D} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \in \mathbb{R}^{12x4} \quad (3.10)$$

3.2. Linear Open-Loop Simulink Model and Results

Once the model has been linearized, it is necessary to compare the linear state outputs to the non-linear outputs given the same control input. For this test, the control input from Figure 2.3 is used as input into the linear and non-linear plants. The linearized model plant can be seen in Figure 3.1.

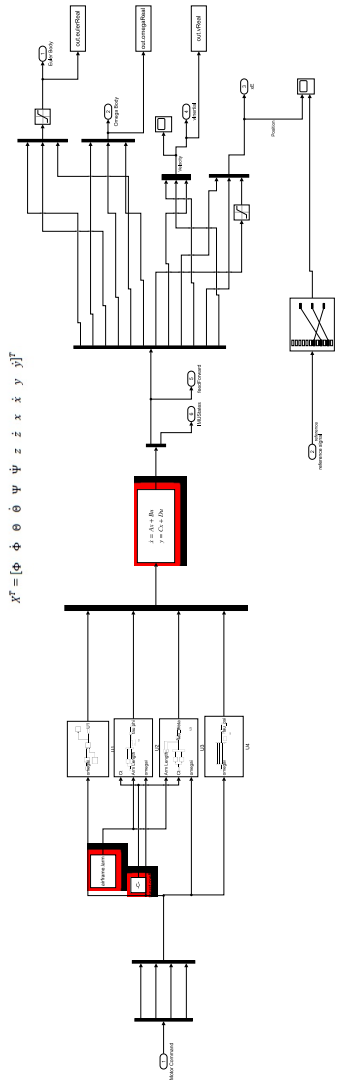


Figure 3.1 – Linearized quadcopter plant.

Below, Figures 3.2 and 3.3 show a comparison between the outputs of the linear and non-linear plants for both the displacement and Euler angles.

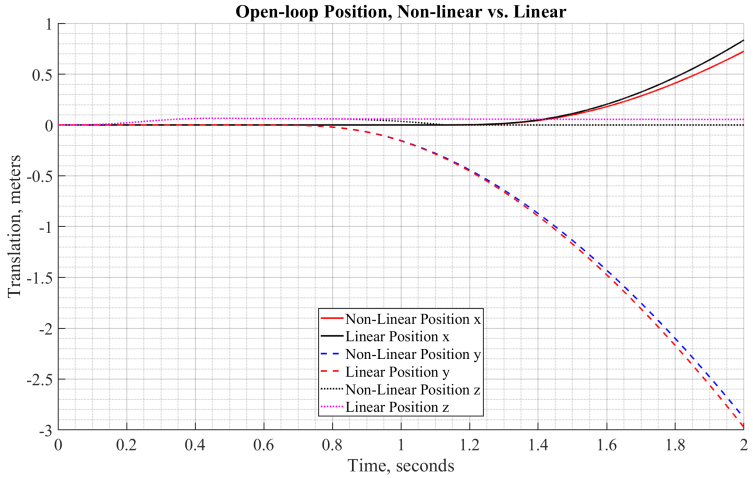


Figure 3.2 – Comparison of quadcopter displacement between non-linear and linear quadcopter plant.

Looking at the displacement plot (Figure 3.2), it is seen that there are slight differences between the two plants. This behavior is expected since the complicated and coupled dynamics of the quadcopter have been simplified and reduced to a set of linear, first-order equations. Though there is a discrepancy between the two displacement outputs, the figure shows that this error is minimal considering the amount of simplification that has been done.

Unlike the displacement equations which are coupled with the rotation angles of the quadcopter, the Euler angles can be calculated independently from the position of the quadcopter. This is evident in Figure 3.3 where there is no difference between the linear and non-linear state calculations. Though there are no differences for this test case, it should be known that this will not be true for all cases. More complicated control inputs where the quadcopter must perform complicated dynamics could prove to unveil differences between the linear and non-linear state calculations.

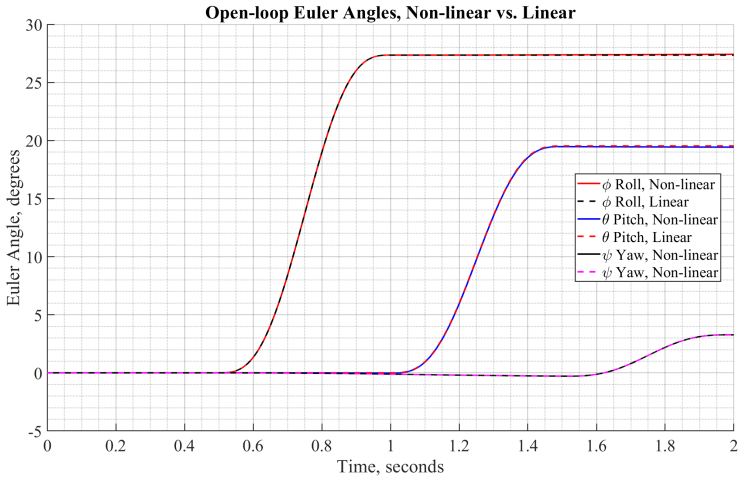


Figure 3.3 – Comparison of quadcopter Euler angles between non-linear and linear quadcopter plant.

3.3. Open-Loop Stability Analysis

When analyzing the stability of the open-loop system, it is important to check the pole and zero locations of the transfer functions of the system. The poles of the states of a system provide information on the stability of that state. Poles that lie on the left half side of the s-plane are stable, whereas those on the right half side are unstable. Further, poles that contain imaginary parts indicate that the state is oscillatory in nature, where those that are real indicate that the system state does not oscillate.

After checking the pole locations of each of the attitude and position states against their respective control inputs, it was found that each pole resides at the origin of the s-plane and they do not contain any imaginary parts. This behavior can be seen in Figures 3.4-3.9 below.

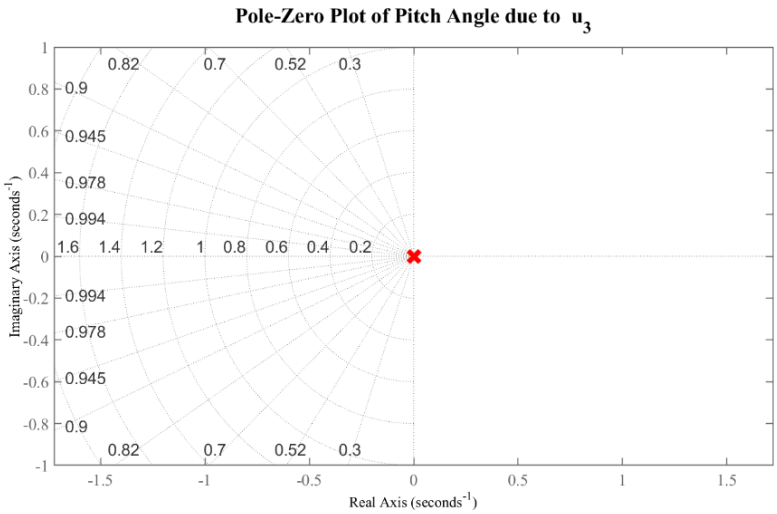


Figure 3.4 – Pole zero plot of pitch due to control input u_3.

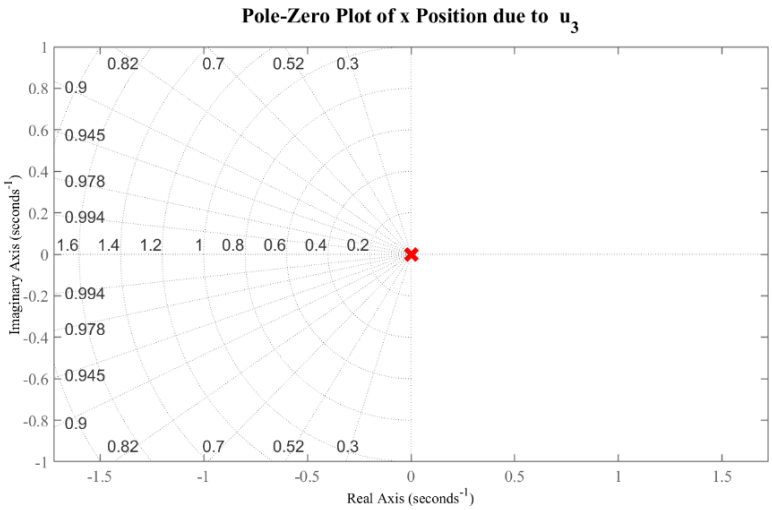


Figure 3.5 – Pole zero plot of x position due to control input u_3.

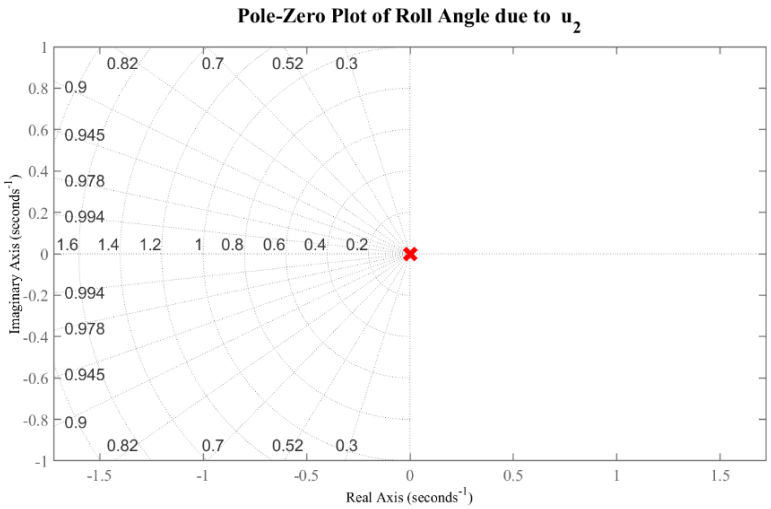


Figure 3.6 – Pole zero plot of roll angle due to control input u_2.

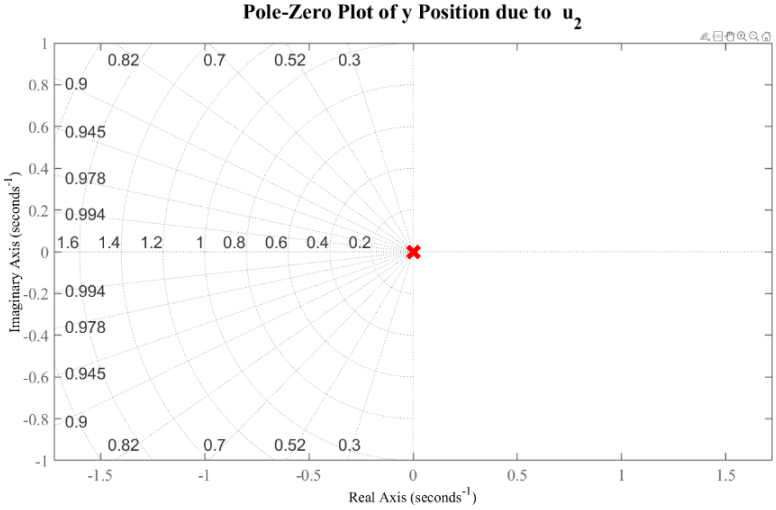


Figure 3.7 – Pole zero plot of y position due to control input u_2.

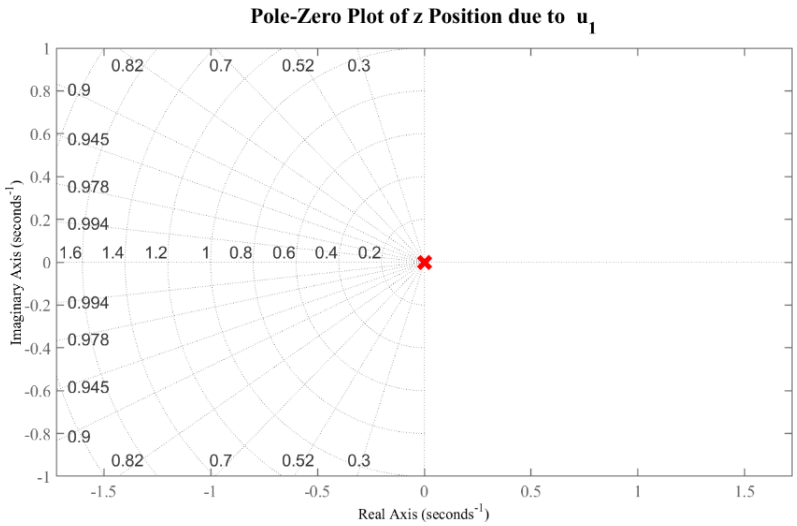


Figure 3.8 – Pole zero plot of z position due to control input u_1.

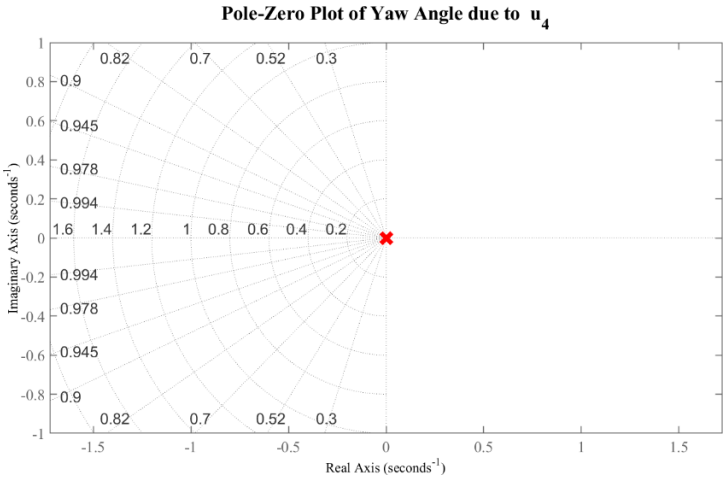


Figure 3.9 – Pole zero plot of yaw angle due to control input u_4.

Multiple poles which all reside at the origin of the s-plane is indicative that the system is unstable [22]. Since the criteria for stability is a pole location that is negative, the quadrotor system will need a closed-loop controller to push the poles of the states towards the left-half plane.

Chapter 4 - Control System Design and Closed Loop Stability Analysis

4.1. Controllability and Observability

Before designing a controller, it is necessary to verify that the system is fully controllable. For the purposes of designing an LQR controller with a complementary filter used for angular state estimation, it is also necessary to check that the system is fully observable. To verify these two requirements, the controllability and observability matrices must be full rank. The controllability and observability matrices are defined below by equations (4.1) and (4.2), respectively:

$$Co = [B\ AB\ A^2B\ \dots A^{n-1}B] \tag{4.1}$$
$$Ob = [C\ CA\ CA^2\ \dots CA^{n-1}]^T \tag{4.2}$$

where n corresponds to the number of system states.

To satisfy controllability and observability, the two matrices defined must be full rank. Specifically, the rank of the controllability and observability matrices must be equal to the number of states, *n*. Using that MATLAB *ctrb()* and *obsv()* functions along with the *rank()* function, it was found that each of the matrices has a rank of 12 which is full rank, thus the system is fully controllable and observable.

4.2. Optimal Control System Design with a Complementary Filter

Based on the open-loop stability analysis that was presented in Chapter 3, it was determined that the system is marginally stable and will require a control system to effectively stabilize and track control inputs. It was decided an optimal control system will be used that has sufficient noise rejection capabilities since it is expected that the onboard sensors of the MAV will be quite noisy. As such, this section will discuss the design and implementation of an optimal Linear Quadratic Regulator (LQR) controller for full state stabilization and control.

Using the linearized model that was developed in Chapter 3, an optimal LQR controller can be designed. The state-space form used for the modelling approach for the controller design is given by [1, 23]:

$$\begin{cases} x = \boldsymbol{A}x + \boldsymbol{B}u + v \\ y = \boldsymbol{C}x + \boldsymbol{D}u + \omega \end{cases} \tag{4.3}$$

where v and ω are defined as the disturbance and measurement noise processes. Typically, these noise inputs are assumed to be zero-mean Gaussian stochastic processes with covariances Q_n and R_n, respectively. The control objective of the LQR is to design an optimal state-feedback controller in the form of:

$$u = -\boldsymbol{K}\boldsymbol{x} \tag{4.4}$$

such that the performance index given by equation (4.5) is minimized [14]:

$$J_{LQR} = \int_0^\infty x^T Q x + u^T R u \, dt \tag{4.5}$$

where x is the state vector, u is the input vector, Q is the state weighting matrix, and R is the control cost matrix.

The solution to of the above problem is achieved according to the Separation Theorem [1], where the steps to the solution are defined as follows:

- Design a state estimator which denoises or estimates the state $\hat{\boldsymbol{x}}$ of the state x.
- Calculate the state-feedback controller in the form of equation (4.4), where $\boldsymbol{K}$ is the optimal state-feedback controller gain.

The value of the gain matrix $\boldsymbol{K}$ is calculated by solving the reduced matrix Ricatti equation for matrix P, as defined by equation (4.6) [24]:

$$A^T P + PA - PBR^{-1}B^T P + Q = 0 \tag{4.6}$$

where P is a real, symmetric, positive matrix. Once found, the gain $\boldsymbol{K}$ can be found as follows:

$$\boldsymbol{K} = R^{-1}B^T P. \tag{4.7}$$

Thus, the optimal control input is found to be:

$$u = -R^{-1}B^T P\hat{\boldsymbol{x}} = -\boldsymbol{K}\hat{\boldsymbol{x}} \tag{4.8}$$

The LQR controller with a complementary filter for quadcopter stabilization is solved using the MATLAB environment. Specifically, the function *lqr()* is used to calculate the value of the controller gain matrix $\boldsymbol{K}$. As stated previously, the value of this matrix is dependent on the value of the Q and R matrices. To replicate results from literature and confirm correct design of the controller, the values of the Q and R matrices were taken from [1], which are used to calculate the gain matrix:

$$Q = 2 \times 10^{-1}\boldsymbol{I}_{12} \tag{4.9}$$

$$R = \begin{bmatrix} 10^{-2} & 0 & 0 & 0 \\ 0 & 10 & 0 & 0 \\ 0 & 0 & 10 & 0 \\ 0 & 0 & 0 & 10 \end{bmatrix} \tag{4.10}$$

4.3. Closed Loop Stability Analysis

After implementing the LQR controller into the system, the poles and zeros of the system are analyzed to characterize the stability of the system. Analyzing the poles of the closed loop system revealed that each pole resides on the left half of the s-plane, meaning that the system is completely stable. Four of the poles contain negative imaginary parts which is evident of states which are stable, yet oscillatory. The location of the twelve state poles can be seen in Figure 4.1 below, with specific values of the poles presented in Table 4.1.

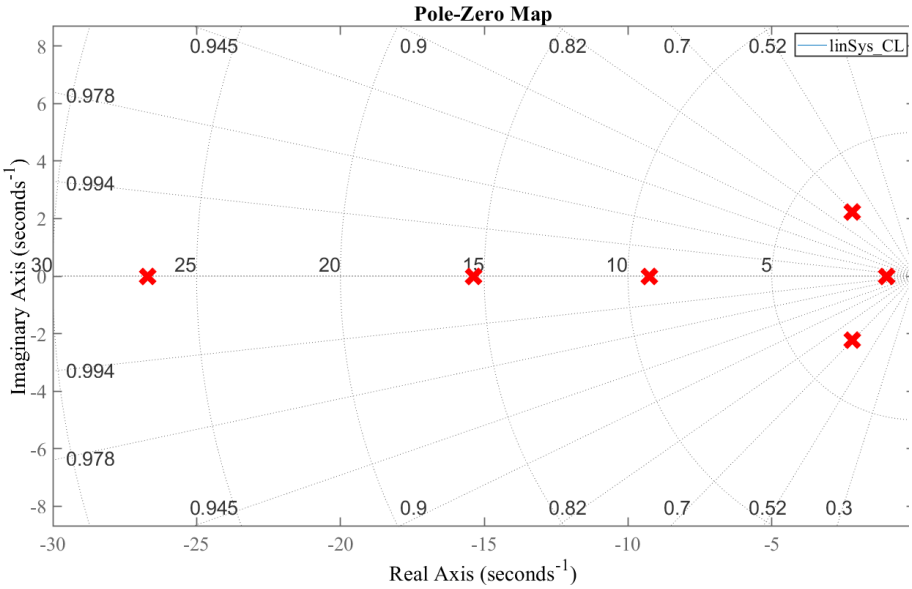


Figure 4.1 – Closed loop pole locations of the system with LQR control and full state feedback.

Table 4.1 – Closed loop pole locations of the quadrotor states

State	Pole		State	Pole
ϕ	-26.71		z	-2.21+2.22i
ϕ	-26.71		z	-2.21-2.22i
θ	-15.37		x	-1.00
θ	-9.26		x	-1.00
ψ	-2.21+2.22i		y	-1.00
ψ	-2.21-2.22i		y	-1.00

It is evident that the LQR controller is successful in stabilizing the quadcopter. Further tuning of the controller could prove to push some of the poles further into left-half plane which would increase the stability of the system.

4.4. Linearized Closed Loop Results and Analysis

After calculating the LQR gain matrix, the controller was designed in Simulink assuming full twelve state feedback response. The controller layout can be seen in Figure 4.2.

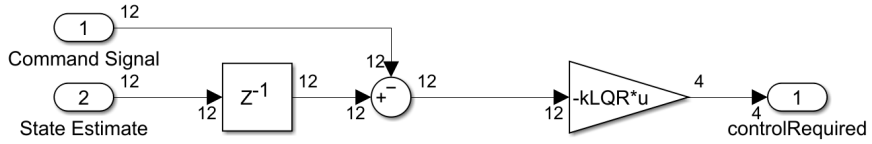


Figure 4.2 – LQR Controller Design.

To test the performance of the designed LQR controller, the following setpoints were chosen for the controller inputs:

$$X_{reference} = \begin{bmatrix} x_{ref} \\ y_{ref} \\ z_{ref} \\ \psi_{ref} \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \\ 1 \\ 1 \end{bmatrix}. \quad (4.11)$$

The setpoints can be seen as red step functions in the following figures. Given a positional step input, it is seen in Figure 4.1 that the quadcopter responds well and tracks the reference signal in about 5 seconds. It is important to note that the tracking errors in the *x* and *y* positions are due to the filtering technique that is being used for the attitude angles. Since this preliminary version of the state estimation uses a complementary filter to estimate the roll and pitch angles, the errors are larger when compared to those from the benchmarked data which uses a Kalman filter to estimate the attitude angles. The tracking error occurs because errors in the attitude angles lead to errors in the position response due to the coupled nature of the quadcopter states. Further iterations of the controller and state estimation technique will seek to minimize the errors by reducing the attitude state estimation error.

Figure 4.3 displays a comparison of the Euler angles between the results from this project and the benchmark results from [1]. It is apparent that there is a difference in the noise levels of the signals. This is because the filtering techniques of the complementary filter are not satisfactory enough to produce comparable results with the same measurement and process noise levels that are presented in [1]. Once a Kalman filter or another more robust filter has been

implemented, the noise levels of the measurement signals will be increased to match the benchmark data. Though a complementary filter is used to estimate roll and pitch angles, the filter is not able to estimate the yaw angle. Currently, the yaw angle is estimated by integrating the yaw angular velocity rates. This explains the accumulated error that is seen in Figure 4.4 for the yaw angle estimation. A magnetometer will be used in further iterations of this project to pull down the state error back to acceptable levels.

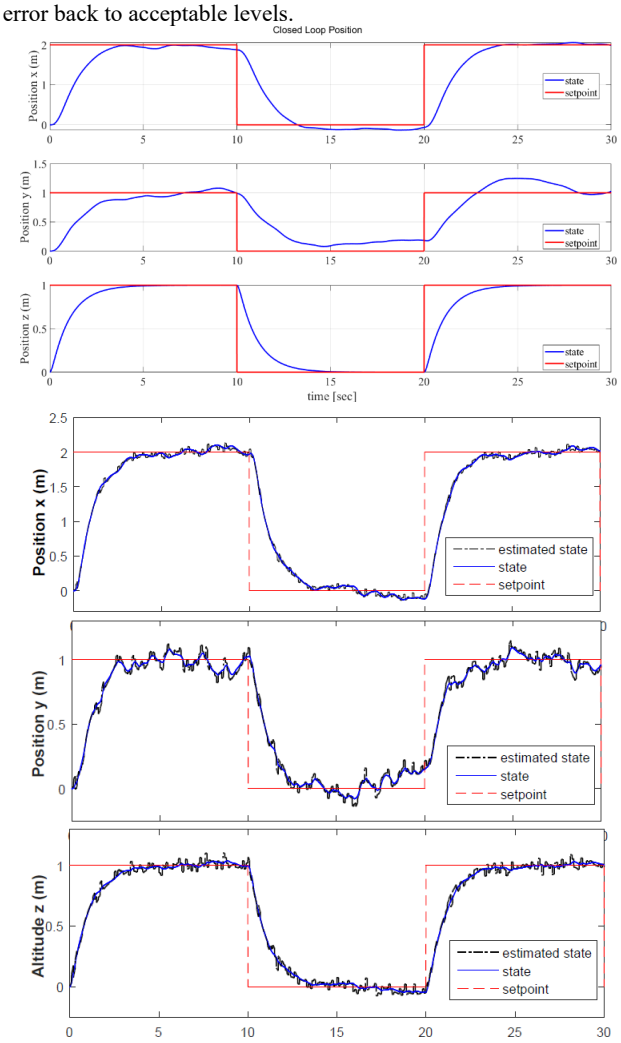


Figure 4.3 – LQR position response of the quadrotor vs. benchmark results.

Figures 4.5 and 4.6 show the velocity and angular velocity responses compared to the benchmark data. The results match well with [1], however as stated before, the noise levels are different between the two. Regardless, the state responses are along the same magnitude as the benchmark.

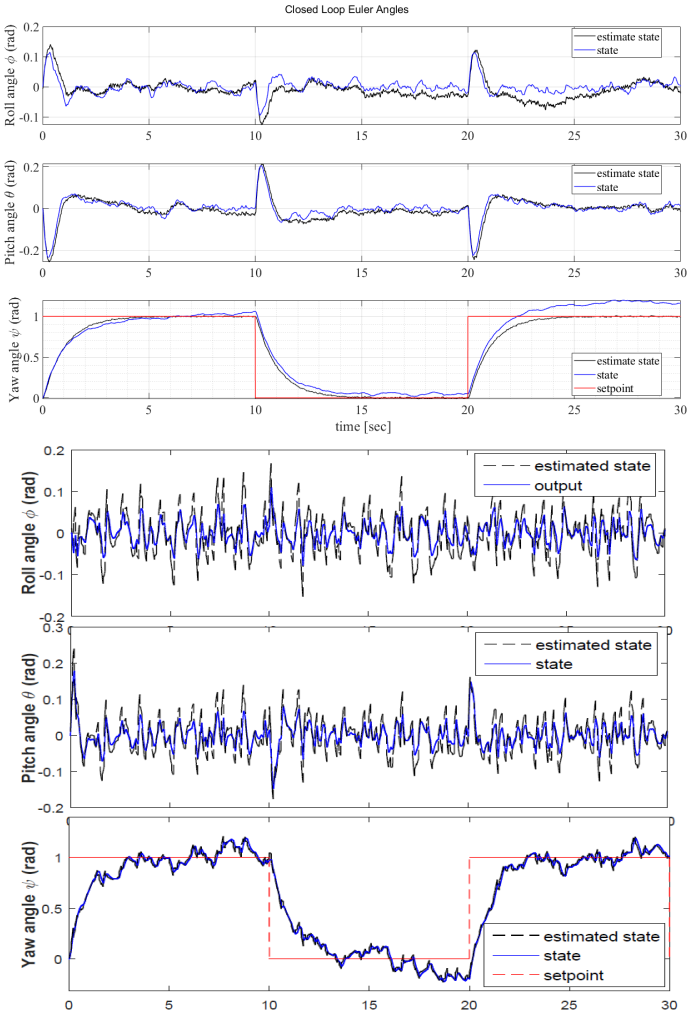


Figure 4.4 – LQR attitude response of the quadrotor vs. benchmark results.

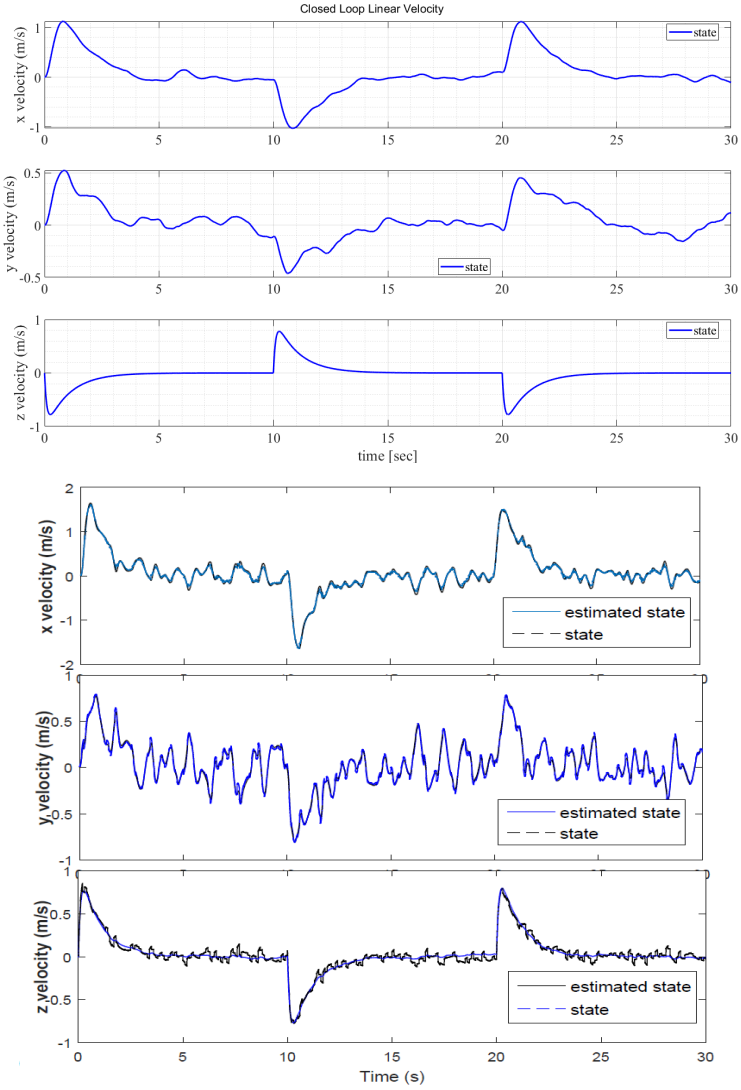
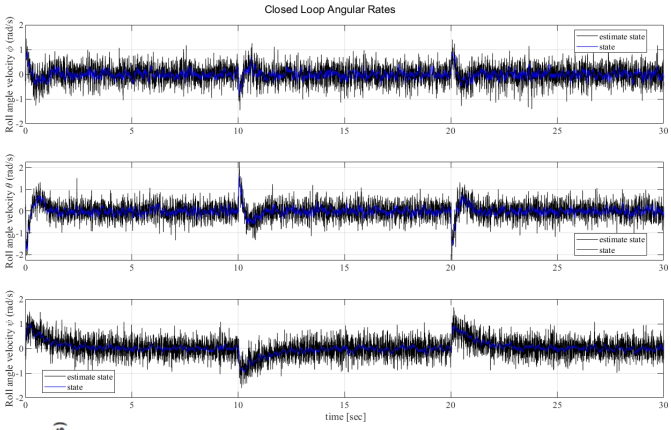


Figure 4.5 – LQR velocity response of the quadrotor vs. benchmark results.



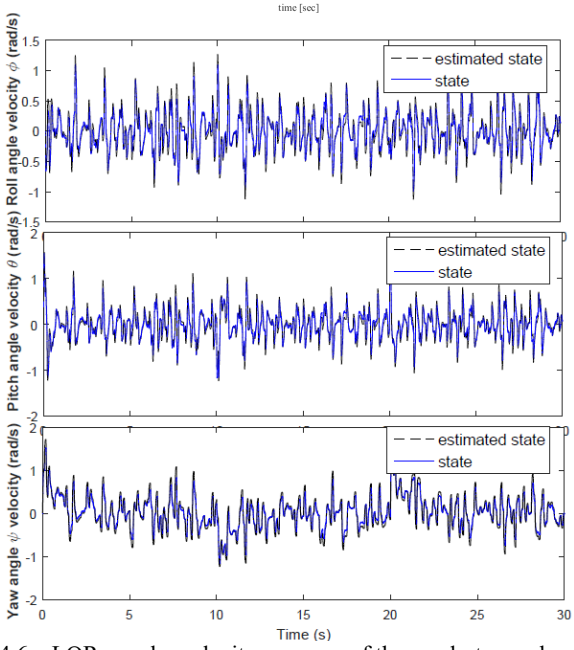


Figure 4.6 – LQR angular velocity response of the quadrotor vs. benchmark results.

The control inputs required to track the reference signal are seen below in Figure 4.7, along with a comparison to the benchmark data. It is seen that the magnitude of the control inputs align are the same between the two sets of data. This proves that the LQR controller is

responding correctly to the control inputs. It should be noted that there is a slight difference in control input u_1, as there is no noise in the data generated from this study whereas the benchmark data does have noise. The reasoning behind this is that the state z is not yet being estimated at this point of the project, so the state can be thought of as "perfect". Further iterations of this project will estimate *x, y,* and *z* positions along with their associated velocities which should cause the data to align better when compared with the benchmark.

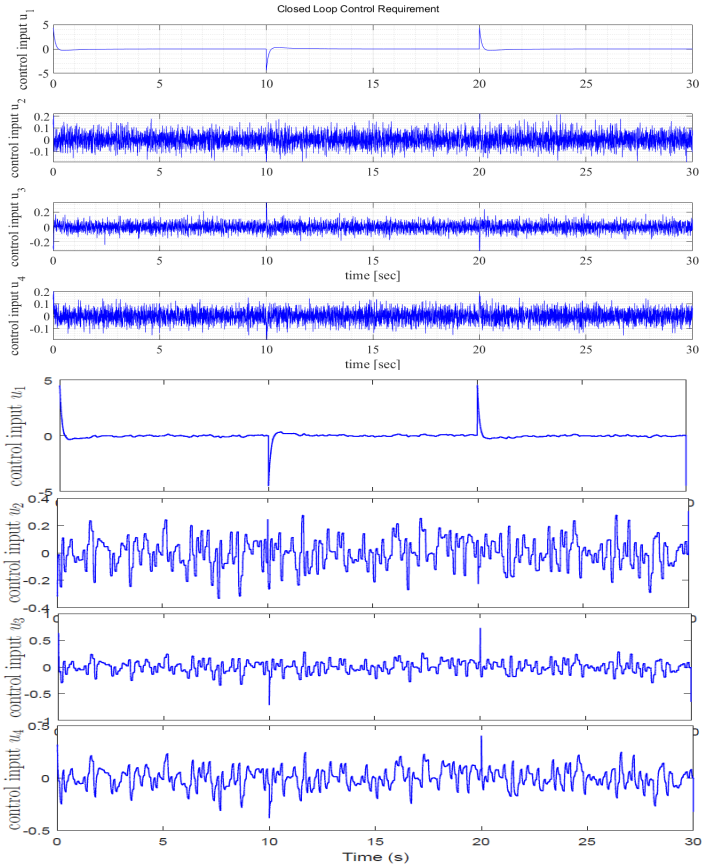


Figure 4.7 – LQR control input response of the quadrotor vs. benchmark results.

Something to take note of with this data is that the control inputs $u_2, u_3,$ and u_4 presented in Figure 4.7 respond at an extremely high frequency. The reasoning behind this is that the actuator dynamics have yet to be modeled, so the control inputs can respond unboundedly in a fashion that is not realistic to how a real actuator would respond. Further work will be done to

include a Simulink block which attempts to accurately capture the dynamics of the actuator in the form of a closed-loop state space system.

Chapter 5 - Motor Dynamics

5.1. Mapping Pulse Width Modulation to Motor Velocity

Currently, it is assumed that the motor control inputs calculated by the flight controller can be any value. There are no saturation values or dynamics present that model the control inputs as they would behave in a real-world setting. For example, the motors are not restricted to one direction and can instantaneously switch from positive to negative rotation to achieve the desired forces and torques. These control inputs can be expressed as being "ideal"; the motors are able to deliver any output that is requested from the flight controller. Though this behavior works from a pure mathematical standpoint, this behavior is neither representative nor acceptable for a physical implementation of the motors.

A solution to this behavior is presented in [28] where the quadcopter actuator dynamics are modelled by a first order differential equation which relates the rotor speed to the command speed. The equation for the first order dynamics is represented by

$$\boldsymbol{\omega} = k_m\left(\omega_i^{des} - \omega_i\right). \tag{5.1}$$

The motor gain k_m is found to be $20s^{-1}$ in this reference, however it is expected that this value will be different depending on the motors that are being used. It is worth noting that in [25] the maximum motor angular velocities are saturated between 1200rpm and 7800rpm. These values are useful for setting realistic bounds on the minimum and maximum angular velocities of the motors. A comparison between the motor speed before and after implementing the motors dynamics is presented in Figure 5.1.

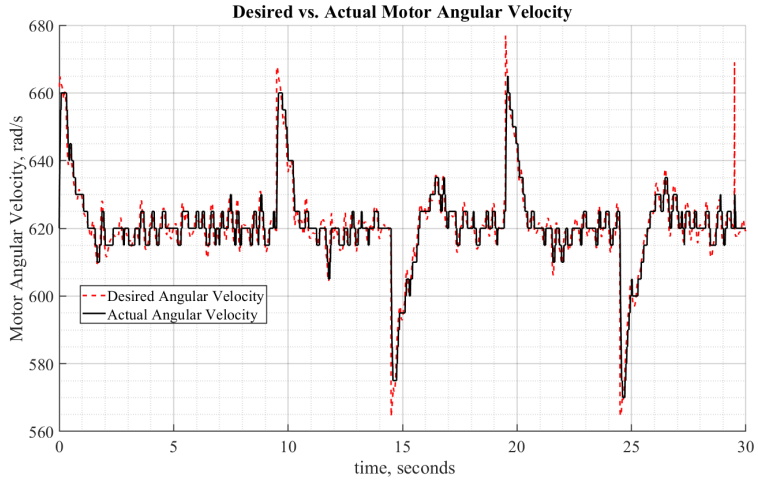


Figure 5.1 – Comparison between desired motor angular velocity and actual angular velocity after being pushed through the first order differential equation.

As shown in the figure, the motor angular velocities shown in the black line are more realistic in terms of motor capabilities. Along with adding saturation limits, a quantization interval was added to smooth out the angular velocity outputs of the motors. This is important because it is not desirable to have high frequency commands sent to the motors electronic speed controllers during flight on a physical quadcopter plant.

5.2. Transfer Function Calculation

The motor outputs presented in the previous section are fine for purely simulation purposes since the simulated quadcopter plant receives motor angular velocities as inputs when updating the state equations. Realistically, however, a motor cannot take a direct angular velocity command to drive the motor speed unless there is some sort of encoder which translates desired angular velocity to the voltage that must be output to the motor. As such, to understand the relationship between desired motor speed and the voltage that must be delivered, a transfer function can be derived that relates the voltage, in the form of PWM output of the microcontroller, to the angular velocity.

To derive the transfer function, the MATLAB system identification toolbox was used which identifies systems given input and output data. The data that was used for this process was collected by measuring the angular velocity of the rotor given a PWM duty cycle using a hand-held laser tachometer. These values were recorded by hand and imported into MATLAB for analysis. Attempts were made to use a photointerrupter to automatically measure the angular velocity of the rotors, however this proved difficult without a sufficient test stand to mount the motor to. The devices used to collect data along with the data collected from this process can be seen in figures 5.2 and 5.3.

Figure 5.2 – Laser tachometer used to collect the propeller angular velocity data

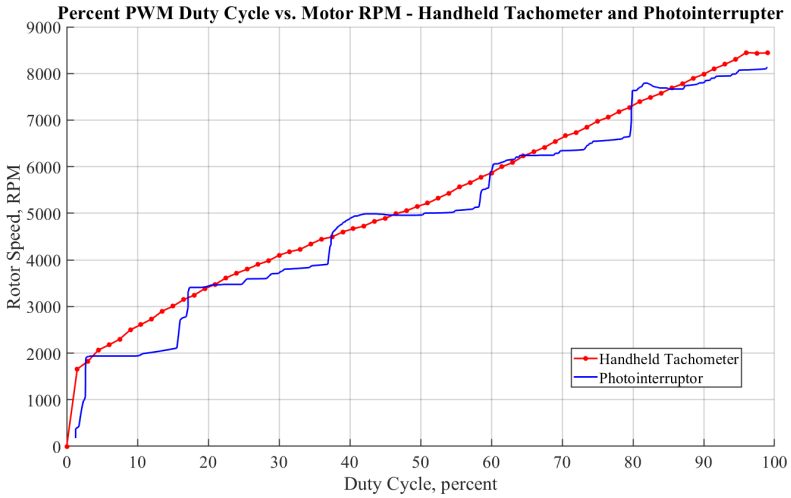


Figure 5.3 – Results of the tachometer data collected from quadrotor propeller.

As shown, the collected data displays a linear relationship between percent duty cycle and the rotor speed. As such, a transfer function can be developed which expresses this relationship.

Since the input to the microcontroller is a PWM signal, the transfer function developed in the system identification toolbox is constructed such that the motor angular velocities (calculated by the LQR controller) are the input, and the required PWM percent duty cycle is the output. This PWM output is fed to the PWM input pins on the Arduino board. Using the MATLAB system identification toolbox and using two poles and two zeros, the transfer function relating desired motor speed to required PWM output was produced, as displayed in Table 5.1.

Table 5.1 – MATLAB system identification transfer function results

Poles	Zeros	Transfer Function (RPM in, Duty Cycle out)	Data Fit
2	2	$G(s) = \frac{(9.959 * 10^{-3})s^2 + (9.918 * 10^{-4})s + 2.325 * 10^{-5}}{s^2 + 0.0957s + 0.001805}$	98.3%

After finding the transfer function, results were tested in Simulink to see if the transfer function responded as expected. Shown in Figure 5.4 is PWM response given a set of inputs. These values are plotted with the associated motor angular velocity that is required.

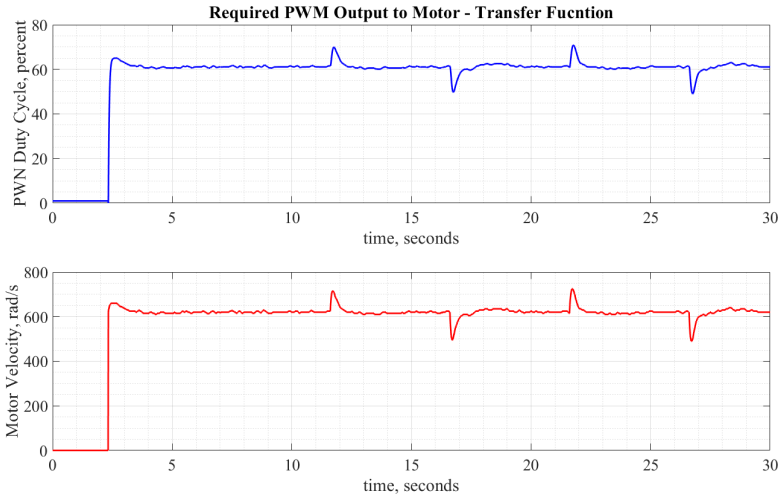


Figure 5.4 – Required motor angular velocity compared to the required PWM duty cycle calculated by the derived transfer function.

It is shown in the figure that the PWM output operates around the 60% point, which makes sense as this is the PWM output that is required to maintain an angular velocity of 620 rad/s which is the required rotor velocity to achieve a steady hover condition.

Chapter 6 -State Estimation

Deleted: ¶

6.1. Attitude Estimation

Chapter 4 demonstrated the quadcopter response with the LQR controller using a complementary filter as means to estimate the attitude of the quadcopter. Though this is a quick implementation and produces satisfactory results for the roll and pitch angles given small movements, the filter is not robust enough to estimate the attitude angles given large movements or disturbances and does not effectively calculate the yaw angle of the vehicle. As stated previously, the only way to calculate the yaw angle is through direct integration of the z-axis reading of the gyroscope, since the accelerometer cannot be fused with the yaw rate to produce a filtered estimate. There are several ways to solve this problem, and for the purposes of this paper due to the hardware selection, an AHRS (attitude and heading reference system) filter was chosen to estimate the attitudes of the quadcopter.

6.1.1. Attitude and Heading Reference System

The AHRS filter is a predictor-corrector filter which utilizes data from a 3-axis accelerometer, gyroscope, and magnetometer to produce accurate body angle estimates as well as bias-corrected attitude rate estimates. The filter works by first predicting the attitude angles using the gyroscope alone, and then corrects for the errors in the attitude estimates using the accelerometer and magnetometer. The accelerometer corrects for the gravity vector, like the way a standard complementary filter works, and the magnetometer corrects for the yaw angle by using Earth's magnetic field vector to orient the yaw angle correctly.

Justa Et al. [29] provide an open-source codebase for the AHRS filter which has been conveniently ported to Simulink by Mathworks. A block diagram of the filtering process is displayed in Figure 5.5. For the purposes of this project, the filter was not coded by hand, but instead the Simulink AHRS block was used to produce the AHRS attitude estimates. As such, the details of how the filter functions will be covered minimally, however, further detail regarding the filter can be found at [26].

The steps of the AHRS filter are as follows:

i. Calculate dynamic rotation change (q)
ii. Predict integration (q_{pred})
iii. Calculate accelerometer reference vector (a_{Ref}, acc_{pred})
iv. Calculate magnetometer reference vector (m_{Ref}. mag_{pred})
v. Calculate deviation of prediction and measurement (mag_{Dev}. acc_{Dev})
vi. Calculate correction quaternion
vii. Fuse quaternions together
viii. Apply the correction step

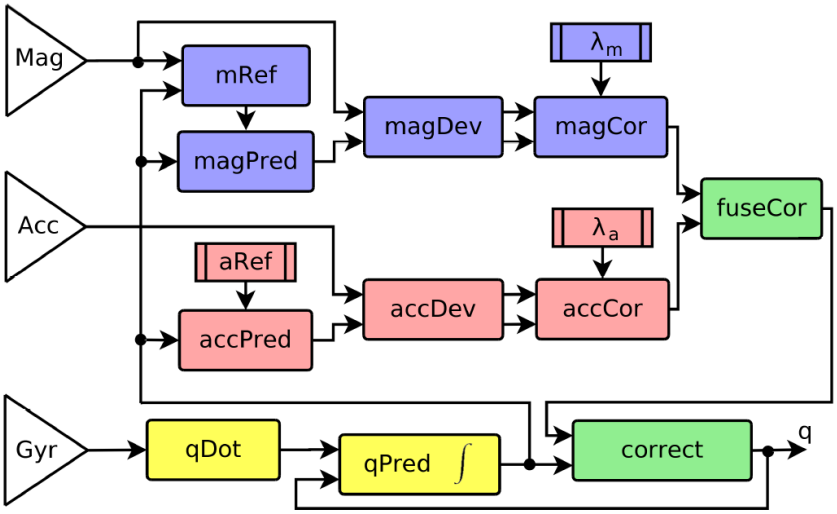


Figure 6.1 – Fast AHRS filter presented by Justa Et al. [26].

The attitude estimates produced by the AHRS filter are significantly more accurate than the complementary filter, as the slow drift of the attitude estimates is correct is corrected towards equilibrium due to the presence of the magnetometer.

6.2. Altitude Estimation

Since the LQR controller is linearized around the hover condition, it is important to be able to accurately estimate the altitude and vertical velocity of the quadcopter. Direct measurements of altitude can be made through a multitude of different sensors such as a barometric altimeter, ultrasound sensor, time of flight sensor, a range-finding laser sensor, and others. For the sake of convenience and cost, it was decided to use a barometric altimeter and ultrasounds sensor to estimate the altitude of the quadcopter. These sensors are widely available and are easy to setup. Both use I^2C protocol to send data to the microcontroller bus.

While these sensors produce direction altitude measurements, they suffer from noise and frequency dropout due to bad readings. Also, while these sensors can measure vertical position, they cannot measure vertical velocity. To measure vertical velocity, a Kalman filter was developed which uses an accelerometer and ultrasound sensor to predict both the vertical position and velocity of the quadcopter.

6.2.1. Altitude Kalman Filter

The Kalman filter is a state prediction technique which, given an input, produces an optimal prediction of the state at the next time step. The predicted state is assumed to reside in

some Gaussian distribution from the actual state. Specifically, the algorithm seeks to minimize the error between the state estimate and the actual state towards zero. It is important to note that the Kalman filter is a linear process, so the state transition and observation models are assumed to be linear. Given the state matrix x_t and the control inputs u_t, a state transition matrix $g(x_t, u_t, \Delta t)$ is calculated which relates turn current step x_t to the state estimation in the next time step, x_{t+1}. The process of calculating the state transition matrix is as follows:

$$x_t = \begin{bmatrix} z \\ z \end{bmatrix}, \tag{6.1}$$

$$u_t = [z] \tag{6.2}$$

$$g(x_t, u_t, \Delta t) = \begin{bmatrix} x_{t,z} + x_{t,z}\Delta t \\ x_{t,z} + u_{t,z}\Delta t \end{bmatrix} \tag{6.3}$$

Taking the Jacobian, A, of (6.13), the following matrix is calculated:

$$A = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix}. \tag{6.4}$$

Taking the control matrix, B, as:

$$B = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \tag{6.5}$$

The linear state transition equation in matrix form can be found as:

$$g(x_t, u_t, \Delta t) = Ax_t + Bu_t + \mathrm{w_k} \tag{6.6}$$

$$g(x_t, u_t, \Delta t) = \begin{bmatrix} 1 & \Delta t \\ 0 & 0 \end{bmatrix} \begin{bmatrix} z \\ z \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} [z] + \mathrm{w_k} \tag{6.7}$$

where $\mathrm{w_k}$ is the zero-mean Gaussian process noise vector.

Because the model describes a linear process, the state transition equations can be written as a set of linear differential equations, as shown in (6.7).

Now that the state transition equation has been calculated, it is also necessary to calculate the measurement model. The measurement model of a Kalman filter describes the measurement process that occurs each time step for the given state matrix. As stated previously, direct altitude measurements are being made from the ultrasound sensor, and no sensor is available which can measure the vertical velocity of the system. The altitude measurements are corrupted by Gaussian white noise. The measurement function of the system, $h(x_t)$, can be described as follows:

$$h(x_t) = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} z \\ z \end{bmatrix} + \mathrm{v_k} \tag{6.8}$$

where $\mathrm{v_k}$ is described as the zero mean Gaussian measurement noise.

Equation (6.18) shows that the model only has altitude measurements available, which is expected since the only sensor in question directly outputs the altitude of the quadcopter.

The last variables missing for the implementation of the Kalman filter are *Q,* the covariance of the process noise, and *R*, the covariance of the measurement noise. For the implementation of the Kalman filter for altitude estimation, the values of *Q* and *R* were chosen as displayed below:

$$Q = 1 \tag{6.9}$$

$$R = 0.2 \tag{6.10}$$

The matrices shown in the previous equation are used in the Simulink Kalman filter block to provide estimation of the vertical velocity and position of the quadcopter.

6.3. Position and Velocity Estimation

Formatted: Normal

Deleted: ¶

While calculation of the vertical position and velocity of the quadcopter is simple to implement, calculating the lateral and longitudinal position and velocity proves to be more complicated. The reasoning behind this is that:

- X and Y positions and velocities are coupled with the body angle ϕ and θ.
- Besides GPS, there are few simple techniques which can provide position or velocity measurements on the x-y plane.

Typically, GPS is used in a Kalman filter along with an accelerometer to estimate the planar position and velocity of a quadcopter. While this solution is simple and robust, it will not work for the scope of this project as GPS does not work indoors. As such, another sensor will need to be utilized in the Kalman filter to produce estimations of the planar positions and velocities.

While there are many sensors that can be used other than GPS, it was decided that a downward-facing monocular camera be used to provide velocity updates to the flight controller. These velocity updates are used in a Kalman filter for x-y state estimation.

6.3.1. Velocity Kalman Filter

The Kalman filter implemented in this model is based on [28] which uses a similar sensor suite to the one used in this work. In [28], readings from an optical flow sensor and an ultrasound sensor are fused to produce estimations of the body velocity $\boldsymbol{v}_b$ of the quadcopter on all 3 axes, the altitude $\boldsymbol{z}_b$, and the accelerometer measurement biases $\boldsymbol{b}_a$. The model presented in [28] is assumed to be linear, which makes implementation of the Kalman filter simpler. However, it should be noted that in most cases and extended Kalman filter is used with the optical flow measurements to produce a more accurate and robust estimation of the planar velocities and positions.

Following the same process as the previous section, the Kalman filter is setup as follows:

$$x_t = [x \quad y \quad z \quad z \quad b_{a,x} \quad b_{a,y} \quad b_{a,z}]^T \tag{6.11}$$

Where $x, y, and\ z$ are the quadcopter body velocities, z is the altitude of the quadcopter, and $b_{a,x}, b_{a,y}, and\ b_{a,z}$ are the x, y, and z accelerometer biases. The control input to the Kalman filter u_t, is shown in the equation below:

$$u_t = \begin{bmatrix} (x + r_{31}g)\Delta t & (y + r_{32}g)\Delta t & (z + r_{33}g)\Delta t & (z + r_{33}g)\frac{\Delta t^2}{2} & 0 & 0 & 0 \end{bmatrix}^T \quad (6.12)$$

where $r_{31}, r_{32}, and\ r_{33}$ represent the first, second, and third columns of the rotation matrix from the inertial frame to the body frame, respectively. The state transition equation is given by

$$g(x_t, u_t, \Delta t) = \begin{bmatrix} x_t - \mathrm{b_{a,x}}\Delta t + (x + r_{31}g)\Delta t \\ y_t - \mathrm{b_{a,y}}\Delta t + (y + r_{32}g)\Delta t \\ z_t - \mathrm{b_{a,z}}\Delta t + (z + r_{33}g)\Delta t \\ z - z_t\Delta t + (z + r_{33}g)\frac{\Delta t^2}{2} \\ b_{a,x} \\ b_{a,y} \\ b_{a,z} \end{bmatrix}. \quad (6.13)$$

The Jacobian of the state transition matrix is found to be represented by the following matrix

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 & -\Delta t & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & -\Delta t & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & -\Delta t \\ 0 & 0 & -\Delta t & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}. \quad (6.14)$$

And the control matrix, B, is calculated at the following:

$$= \begin{bmatrix} \Delta\mathrm{t} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \Delta t & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \Delta t & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{\Delta t^2}{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}. \quad (6.15)$$

Thus, the linear state transition equation in matrix form can be expressed in the form of:

$$g(x_t, u_t, \Delta t) = Ax_t + Bu_t + \mathrm{w_k} \quad (6.16)$$

where $\mathrm{w_k}$ is the zero-mean Gaussian process noise vector.

The measurement function of the system can be described by the following equation:

$$h(x_t) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ z \end{bmatrix} + \mathrm{v_k} \tag{6.17}$$

where $v_x \; and \; v_y$ are the velocities of the camera in the x and y direction, respectively, and z is the altitude reading provided by the ultrasound sensor.

For this Kalman filter model to work, the values of $v_x \; and \; v_y$ must be calculated. Since the optical flow sensor only outputs pixel velocity, further calculations must be done to convert the pixel velocities to velocity in the body frame. The calculations presented are derived from [29] which presents an algorithm for calculating the velocity of the quadcopter body given pixel velocities.

To calculate the velocity in the body frame given direct outputs from the optical flow sensors, the angular offsets about the x and y body axes must first be calculated. Given a camera with aperture angles $\theta_{px} \; and \; \theta_{py}$ the angular offsets about the x and y axes can be expressed through the following set of equations:

$$\theta_x = \frac{\theta_{px}}{\Delta t N_y} \Delta \mathrm{n_y}, \qquad \theta_y = \frac{\theta_{py}}{\Delta t N_x} \Delta \mathrm{n_x} \tag{6.18}$$

where N_x, N_y are the number of pixels on the camera image plane in the x and y directions, and $\Delta n_x, \Delta n_y$ are the pixel velocities that are returned from querying the optical flow sensor. Once the angular offsets are known, velocity in the x and y body directions can be calculated through the following relationship:

$$x = \frac{\theta_{py}}{\Delta t N_x} \Delta \mathrm{n_x} - z\omega_{B,y}, \qquad y = \frac{\theta_{px}}{\Delta t N_y} \Delta \mathrm{n_y} - z\omega_{B,x}. \tag{6.19}$$

Here, z is defined as the altitude of the quadcopter, $\omega_{B,x} \; and \; \omega_{B,y}$ are the angular body rates of the quadcopter in the x and y axes, respectively.

Now that the velocity of the quadcopter in the x and y direction is known, these values, along with the matrices that were developed previously, are used in the Simulink linear Kalman filter block to further estimate the x and y velocities of the quadcopter. The x and y positions are calculated through direct integration of the velocity estimations.

Chapter 7 - Ros and Gazebo System Modelling

Deleted: A

7.1.Methodology

While the simulation developed in Simulink provides a decent estimation of the effectiveness of the LQR controller, it is not all encompassing and does not consider the real physics and dynamics of the rotors. What is meant by this is that the rotation of the rotors does not generate lift in the Simulink model, but instead the control outputs that are calculated from the LQR controller are essentially just applied directly to the quadcopter plant.

As an alternative to Simulink, a simulation platform was developed using the RotorS package for Gazebo [30]. This package allows for the simulation of real rotor physics by calculating the lift and drag characteristics of the rotors that are spinning in the model. This simulation package is also useful for the fact that sensor objects can be placed on the quadcopter model to simulate the data acquisition of sensors that would be present on a physical quadcopter plant. This makes testing flight control and state estimation algorithms convenient.

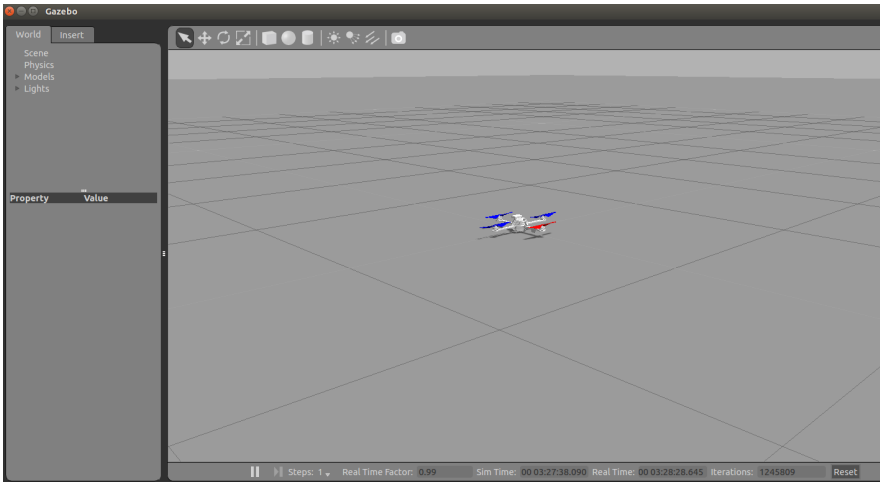


Figure 7.1 – Quadcopter plant implemented in Gazebo.

Since Gazebo interfaces with ROS, the ROS support package for MATLAB was used to create a node which represents the flight controller. The flight controller node is programmed to subscribe to the /imu topic to receive IMU data which is used in the flight control algorithm. Once the required motor speed control inputs are calculated from the flight controller, those values are sent to the /command/motor_speed topic to directly control the motor speeds in the simulation. Ground truth position and velocity data is received from the quadcopter simulation by subscribing to the /ground_truth/transform and the ground_truth/odometry ROS topics. This data is compared to the desired position commands to quantify the effectiveness of the flight controller.

Deleted: _

Deleted: _

Deleted: _

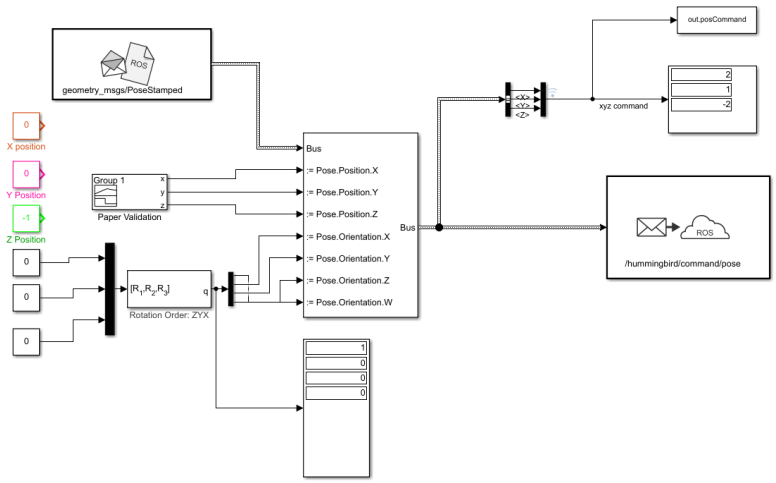


Figure 7.2 – Simulink file used to send commands to ROS topic as well as receive ground truth data from the quadcopter.

7.2. Results

Of interest when analyzing the data from the simulations performed in Gazebo is the position tracking capabilities of the quadcopter. It should be stated beforehand that the simulations performed in Gazebo utilized an old flight control algorithm which only uses a complementary filter for attitude estimation and does not have any means of estimating x-y position and velocity. These states are taken as unity feedback in the state estimation process. Shown in Figure 7.1 are the x, y, and z tracking capabilities of the LQR controller running in Gazebo given a reference signal, shown in red.

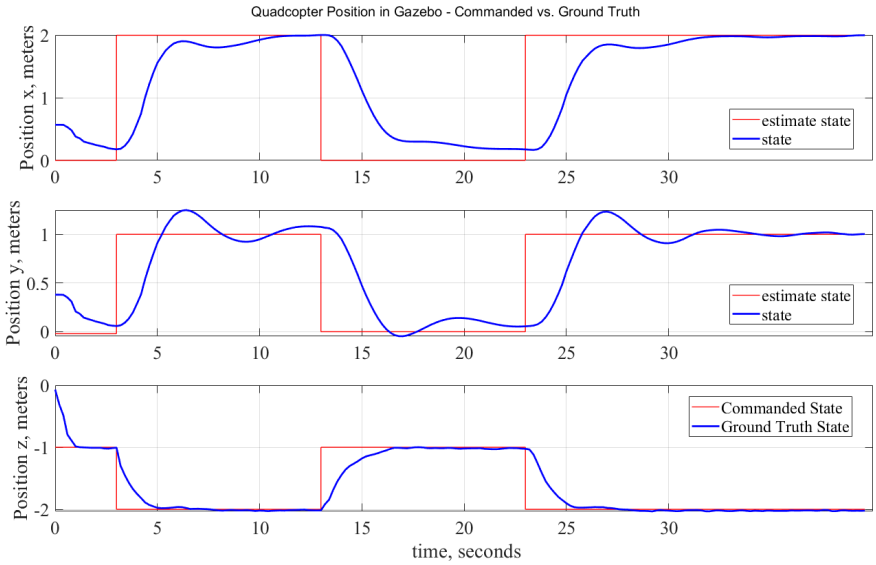


Figure 7.3 – LQR position tracking performance when simulated in RotorS Gazebo platform.

The quadcopter exhibits excellent performance when tracking a reference signal along the vertical axis. This makes sense because (i.) the quadcopter was linearized around the hover condition, so it is expected to perform well when given a stable altitude reference signal to track, and (ii.) the dynamics of the motion of the quadcopter along the z-axis are decoupled from the attitude angles as opposed to the x and y positions which are directly coupled to the attitude dynamics. Future studies will be conducted to analyze the effectiveness of the flight control when utilizing the AHRS algorithm and Kalman filters discussed previously.

Chapter 8 - Hardware Set-Up

To analyze the performance of the flight control system in a realistic setting, a physical quadcopter was built which the flight controller was mounted to. The goal of constructing the quadcopter is to produce similar results as were displayed in the simulation runs. The various parts of the quadcopter that were chosen are outline in the following sections.

8.1.Processor

The processor chosen to build and compile the code onto is the Arduino Due microcontroller. The Arduino Due, pictured below, is a 32-bit microcontroller based off the ARM-Cortex-M3 CPU. It was chosen for this project due to the 32-bit core which enables an 84 MHz processing speed – more than enough for the scope of this project.

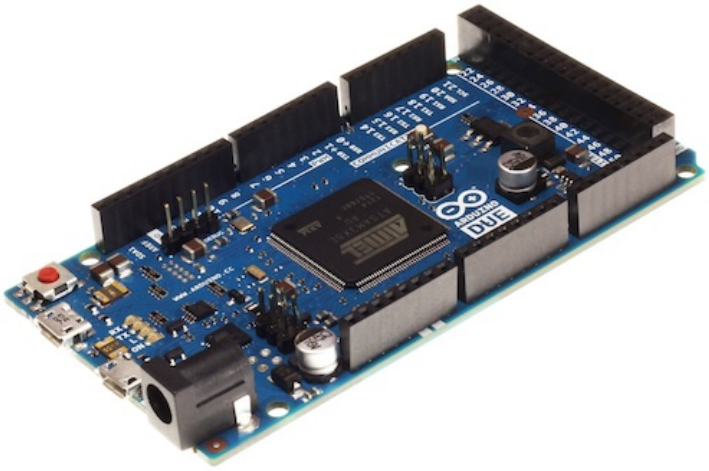


Figure 8.1 – Arduino Due used as the flight controller for this project.

Also noteworthy are the 12 PWM ports that are available on the Due along with the I^2C and SPI communication busses. This microprocessor provides more than enough capabilities for this project and leaves room for future additions as well.

8.2. Communication

To send and receive commands to and from the flight controller, a set of XBee Bluetooth communication devices are used. These devices communicate over 802.15.4-ZigBee & BLE protocols to enable serial communications between the quadcopter and the ground station. The serial signal is received and transmitted through Arduino UART. The ground station uses an XBee which mates with a USB dongle that can be connected to any computer. This allows for wireless control and communication with the quadcopter during flight indoors and outdoors.

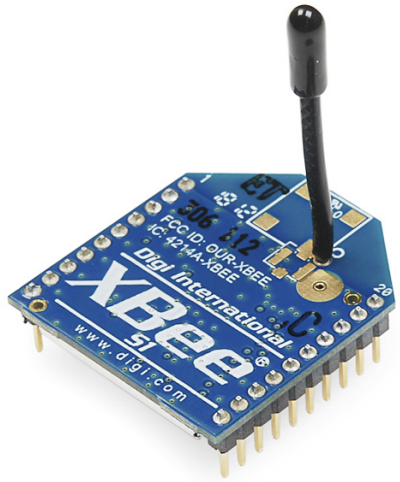


Figure 8.2 – Image of XBee used for wireless communication in this project.

8.3. Battery

The battery chosen for this project is a 4000 mAh 3S Lipo battery. The 3S battery provides the necessary power to the electronic speed controllers which control the motors. The battery provides sufficient power to the quadcopter without being too heavy.

Figure 8.3 – Lipo battery used to power the ESCs. .

8.4.Motors and ESC

The motors chosen for this project are the EMAX 2213 935 kV brushless DC motors. At full power when mounted with an 8045 propeller blade, each one of these motors can produce around 500 grams of thrust for a combined total thrust from all four motors of around 2 kg. Given that the quadcopter weighs a total of 1.17 kg, the thrust-to-weight ratio of the system is around 1.7 which is sufficient.

To drive the motors, a set of SimonK 30A ESCs was chosen. These ESC modules are commanded by a PWM signal and exhibit a response time of up to 490 Hz

Figure 8.4 – EMAX 2213 935 kV motors used in this project. The motors each have a 8045 blade mounted on top.

8.5. Sensor Suite

The sensors are one of the most important components of the quadcopter, as these are what will enable state estimation during flight. It was decided that the quadcopter is to be equipped with an IMU, ultrasonic sensor, barometer, and optical flow sensor. This sensor suite enables state estimation of all 12 states which meets the full-state feedback criteria that is necessary for the LQR controller to function.

8.6. IMU

The IMU that communicates with the Arduino Due is the MPU9250 9-DoF MARG. This chip contains a 3-axis accelerometer, gyroscope, and magnetometer; all of which are necessary for use of the AHRS attitude estimation algorithm.

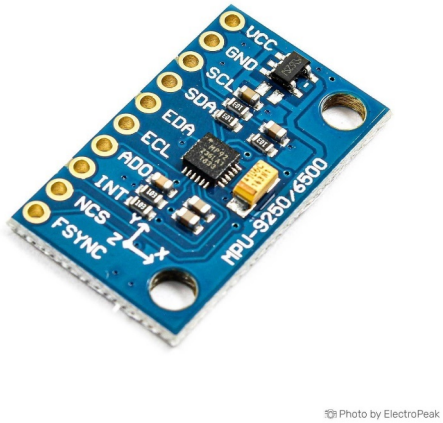


Figure 8.5 – MPU9250 MARG Module.

Conveniently, Simulink has a block which can be used to communicate with the MPU9250 sensor during simulation.

8.7. Ultrasonic Sensor

The ultrasonic sensor chosen is the widely used HC-SR04 sensor. This sensor is easy to use and provides reasonable altitude estimates. It should be noted that the readings from this sensor exhibit frequency dropouts, however this is not concerning because the logic developed in the state estimation techniques of the flight controller should be able to handle these sporadic dropouts.

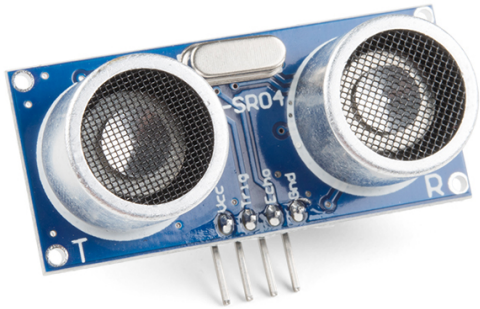


Figure 8.6 – HC-SR04 ultrasonic sensor which is used to provide altitude readings to the flight controller.

8.8. Optical Flow Sensor

The last sensor in this discussion is the PMW3901 optical flow sensor. This sensor is conveniently sized and can be placed below the quadcopter to provide optical flow estimates in the x and y directions. This sensor was chosen because it can be run directly from an Arduino board through SPI protocol and performs all the optical flow calculations onboard which means the quadcopter does not need a flight computer to handle the calculations.

Figure 8.7 – PMW3901 Optical Flow Sensor.

The optical flow estimates of this chip are regarded as highly accurate which is beneficial to the Kalman filtering process used to estimate x-y position and velocity of the quadcopter.

Chapter 9 - Hardware Simulation

After constructing the physical model of the quadcopter, flight tests were done to test the effectiveness of the control system. Making the transition from simulation to implementation is not always smooth, and several issues were encountered which resulted in the inability to successfully hover the quadcopter. The issues are outlined in the following sections – some of which were fixed - and possible solutions are recommended.

9.1. Motor Vibrations

Due to the nature of MEMS, measurements are extremely sensitive to noise and any small vibrations. Inherently, a quadcopter operating at hover thrust conditions will produce large vibrations due to the motors spinning at over 700 rad/s. These vibrations are due to the imbalances in the propellers and motors themselves. An example of the effect of the motor vibrations is displayed below in figure 9.1.

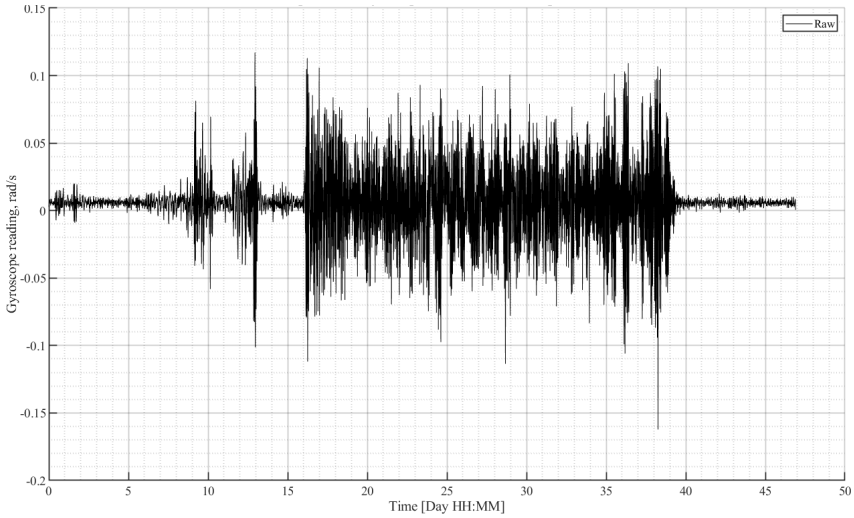


Figure 9.1 – Vibrations caused by motors spinning.

As shown by the data, turning on the motors – even without propellers on – causes vibrations in the IMU which are six times greater than the gyroscope noise. As one can imagine, this is an issue because poor angle rate measurements produced from the IMU will corrupt the attitude estimates. The LQR control takes direct feedback from the attitude estimates and uses the attitude error to produce a corresponding control torque to keep the quadcopter stable in hover. If the attitude estimates are corrupted with noise from the vibration of the motors and propellers, the quadcopter will not be able to perform.

A solution to this problem is the implementation of a discrete low-pass filter. During hover, the propellers spin at around 700 rad/s which corresponds to a frequency of about 110 Hz. Since the data from the IMU is output at a sample rate of 200 Hz, the frequency of the rotors spinning will corrupt the measurements. However, since it is known that the frequency of the noise is 110 Hz, a low pass filter with a cutoff frequency below 110 Hz should get rid of a lot of the noise. Figure 9.2 displays the IMU measurements taken with the motors running after the low pass filter has been implemented.

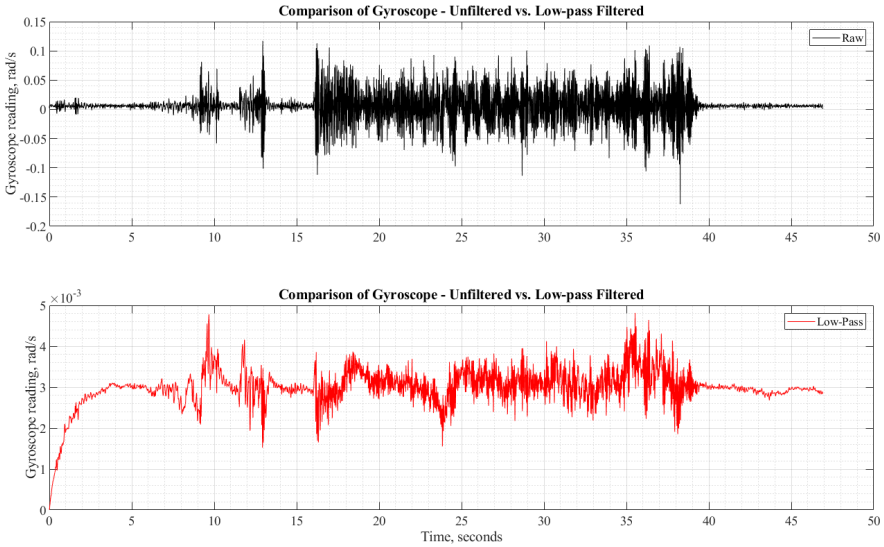


Figure 9.2 – IMU Measurement with low pass filter.

It is seen that most of the motor high frequency noise is filtered out and the angle estimates contain minimal noise in comparison to the previous estimation.

9.2. Propeller Thrust Coefficient

The propeller thrust coefficient is unique to each propeller geometry and determines how quick the quadcopter propellers must spin to produce the desired thrust that the control system requests. It is important that this value is accurate, as inaccuracies of the coefficient will cause the motors to spin too quickly or too slowly. When conducting hover tests, the quadcopter had difficulty reaching the propeller angular velocity values required for hover. It was determined that the issue was an incorrect assumption of the propeller thrust coefficient. Effectively, the value of this coefficient was too high which assumed that the propeller was more effective than it is. Typically, the value of this coefficient is found through experimentation using a thrust stand.

However, since this sort of test rig was not available, the propeller thrust coefficient was instead estimated.

As described previously in equation (2.2.2), the thrust of a propeller is a function of both the angular velocity of the rotor as well as the propeller thrust coefficient. This relation can be reworked as follows to solve for the propeller thrust coefficient k:

$$k_{propeller} = m_{quad} g/(4\omega_{hover}^2) \quad (9.1).$$

The motors of the quadcopter were increased until the vehicle began to lift off the ground. At this point, the angular speed of the rotors was recorded, and the corresponding propeller thrust coefficient was calculated. Using this experiment, the propeller thrust coefficient was found to be 6.1152e-06. This value is consistent with values that are calculated for similar propellers. Upon testing the hover capabilities using this new thrust coefficient, it was found that the quadcopter begins to hover very close to the value that is expected.

9.3. Battery Voltage

The control system presented in this paper works by linearizing about a stability point. Specifically, it was chosen to trim the quadcopter about the hover condition. To do this, the angular velocity of the propellers at which the quadcopter begins to hover was recorded, and this value was used to trim the u_1 thrust input of the LQR controller. While this process works in theory and in simulation, it had poor results in the hardware implementation of the control system.

As the motors spin, the voltage of the battery begins to drop. This drop in voltage causes the propellers to spin slower for a given PWM value. In turn, this essentially causes the constant hover trim force to not be sufficient to drive the motors to achieve hover. This phenomenon is not an issue for a fully charged battery, but eventually the quadcopter will not be able to hover as the battery begins to drain.

A potential fix to this issue is to model the voltage drain of the battery. In turn, as the voltage drains, a higher trim thrust will need to be sent to the flight control system to maintain steady hover

9.4. Motor ESC Sample Rate

To control the motors, the Simulink support package for Arduino was used. This package contains standard Arduino libraries such as Servo.h which provide block functions to use for controlling motors, for example. An image of the block is displayed in figure 9.3. The standard servo write block in Simulink outputs a pulse width modulation pulse at a value between 1000 microseconds and 2000 microseconds, corresponding to fully off, and fully on, respectively. Any value between the range of 1000 and 2000 microseconds will allow for a motor that is attached to a PWM pin on the Arduino Due to be throttled. While this block successfully outputs the required PWM pulse width, there are many limitations in this function, as it only operates at a command frequency of 50 Hz. For many purposes, 50 Hz is plenty enough to provide the

required control outputs to a plant. In practice, however, it was found that the 50 Hz limitation results in a large lag in the control response of the quadcopter.

As stated previously, the flight controller calculates the required forces and torques and corresponding actuator speeds at a rate of 200 Hz. The Simulink and Gazebo models were tested with the 200 Hz flight controller and performed exceedingly well. When making the transition from the simulations to real hardware, it was discovered that the 50 Hz control signal output by the standard servo write block essentially limits the control loop to operate at a rate of 50 Hz as well. While the required forces and torques are calculated at a rate of 200 Hz, a rate transition must occur between the flight control loop and the actuator output loop which restricts the flight controller to also operate at 50 Hz only. This was found to be a big issue as the controller was operating at a quarter of the intended rate. This caused instability when attempting to hover as the dynamics of the quadcopter change quickly and the 50 Hz update rate is much too slow to deliver the required control inputs.

While there are many solutions to this issue, the main solution is to stray away from the Arduino Servo library. Based on the SimonK ESC datasheet, the ESC can take an input of up to 500 Hz – 10 times faster than what the Arduino Servo library outputs. To reach this 500 Hz mark (or even just 200 Hz which is the update rate of the control loop), an external library can be written in C++ which drives the PWM values that the Arduino outputs. Once this is done, the motor outputs can be delivered at rates higher than 50 Hz which should greatly increase the stability of the vehicle when attempting to hover.

Figure 9.3 – Servo write block used to control motors.

Chapter 10 - Conclusion And Future Work

10.1. Conclusion

The work presented in this paper covers the modelling and simulation of an open loop non-linear quadcopter plant which displayed promising results when compared to benchmarked data. After, the model was linearized in Chapter 3 and the results from the linearized model were compared to the results from the non-linear plant. The results matched with error that was within reason for linearization. Chapter 4 covers the LQR controller design and provides a closed-loop analysis as well as an analysis of the state response given reference inputs. These results were compared to results from [1].

State estimation approaches are presented in the form of Kalman filters and an AHRS filter. Two Kalman filters are utilized – one to estimate vertical position and velocity by using accelerometer and ultrasonic sensor measurements, and one to estimate x-y position and velocity using the PMW3901 optical flow sensor and accelerometer. The AHRS filter uses the accelerometer and gyroscope data to provide unbiased and accurate attitude and heading estimates. The state estimation techniques work well and track closely to reality. These state estimates are implemented in the feedback loop and used in the LQR control loop to calculate the required thrust and corresponding motor speeds needed to stabilize the quadcopter.

The efficacy of the state estimation and control techniques were demonstrated in Gazebo, which is a real-time physics engine. The rotorS plugin for Gazebo was used which aims to simulate realistic lift and drag characteristics of spinning rotors. The LQR control and state estimation techniques were tested in Gazebo and displayed promising results. Data shows that the states track closely to the command setpoint, and the quadcopter only oscillates minimally. The flight controller can take in commands which require small deviations from equilibrium, however large x-y position commands cause instability as the LQR controller attempts to quickly converge on the setpoint value. This behavior can be mitigated by reducing LQR gains or imposing saturation limits on the maximum roll and pitch angles that the quadcopter can perform.

An overview of the parts selected for building the quadcopter is presented. The Arduino Due is used at the processor for the flight controller due to the processing speed and 32-bit CPU. Though steady flight was not achieved, the shortcomings were explained and suggestions for fixing the issues were offered. Despite the inability to successfully implement a physical model of the quadcopter, the work presented in this paper presents simulation techniques and processes which can be used for future iterations of this design as well as the design of others.

10.2. Future Work

Though the LQR controller design has much success in stabilizing the system in simulation, work still needs to be done in terms of making the quadcopter simulation more accurate. Specifically, it was found that to accurately achieve and maintain hover, it is necessary to model the voltage drain of the battery so that the PWM output applied to the motors can be increased as voltage drops.

For this design, the attitude angles and heading are being estimated using an AHRS filter. It would be an interesting study to compare the results to those estimated using an extended Kalman filter. It is possible that an EKF design will boast quicker execution times which helps to stabilize the quadcopter to a higher degree.

As outline in the results section of this report, the quadcopter was unable to achieve a stable hover. Works needs to be done towards fixing the issues presented by implementing the suggestions that were mentioned in Chapter 9. Based on the performance of the flight controller in the Gazebo simulator, it is expected that once these issues are resolved, the quadcopter should be able to hover successfully.

References

[1] Fessi, R., and Bouallègue, S., "Modeling and Optimal LQG Controller Design for a Quadrotor UAV," Research Laboratory in Automatic Control, University of Tunis El Manar, Tunis, Tunisia, 2016.

[2] Guang, X., Gao, Y., Leung, H., Liu, P., and Li, G., "An autonomous vehicle navigation system based on inertial and visual sensors," *Sensors*, Vol. 18, 2018, p. 2952.

[3] Spong, M.W., "Partial feedback linearization of underactuated mechanical systems," *IEEE*, Vol. 1, 1994, pp. 314-321.

[4] Quinchia, A., Falco, G., Falletti, E., Dovis, F., and Ferrer, C., "A comparison between different error modeling of MEMS applied to GPS/INS Integrated Systems," *Sensors*, Vol. 13, 2013, pp. 9549–9588.

[5] El-Sheimy, N., and Youssef, A., "Inertial sensors technologies for navigation applications: state of the art and future trends," *Satellite Navigation*, Vol. 1, No. 1, 2020, pp. 1-21.

[6] G. Huang, "Visual-Inertial Navigation: A Concise Review," *2019 International Conference on Robotics and Automation (ICRA)*, Montreal, Canada, 2019, pp. 9572-9582.

[7] Ly Dat Minh, and Cheolkeun Ha, "Modeling and control of quadrotor MAV using vision-based measurement," *International Forum on Strategic Technology 2010*, *IEEE*, Ulsan, South Korea, 2010, pp. 70-75.

[8] Z. Lianhua, L. Hao, S. Zongying, "Velocity estimation and control of 3-DOF lab helicopter based on optical flow," *2013 9th Asian Control Conference (ASCC)*, Istanbul, Turkey, 2013, pp. 1-6.

[9] Jianbo Shi, and Tomasi, "Good features to track," *1994 Proceedings of IEEE Conference on Computer Vision and Pattern Recognition,* Seattle, WA, 1994, pp. 593-600.

[10] S. G. Fowers, D. -J. Lee, B. J. Tippetts, "Vision Aided Stabilization and the Development of a Quad-Rotor Micro UAV," *2007 International Symposium on Computational Intelligence in Robotics and Automation*, Jacksonville, FL, 2007, pp. 143-148.

[11] Zhao, S., Zhao, S., Lin, F., "Vision-aided Estimation of Attitude, Velocity, and Inertial Measurement Bias for UAV Stabilization," *Journal of Intelligent & Robotic Systems*, Vol. 81, No. 3, 2016, pp. 531-549.

[12] Zulu, A., and John, S., "A Review of Control Algorithms for Autonomous Quadrotors," *Open Journal of Applied Sciences*, Vol. 4, No. 14, 2014, pp. 547-556.

Commented [RA1]: Location?

[13] Silveira, A. S., Silva, A. F., Real, J. A. F., and Silva, O. F., “Centralized multivariable LQG control system for longitudinal and lateral speed hold autopilot for the AR.DRONE 2.0 Quadcopter,” *Proceedings XXII Congresso Brasileiro de Automática*, Brasil, 2018.

[14] Wang, W., Ma, H., and Sun, C., "Control System Design for Multi-Rotor MAV," *Journal of Theoretical and Applied Mechanics*, Vol. 51, 2013, pp. 1027.

[15] T. Chakraborty, and A. Kumar, "Adaptive Sensor Fusion and Propeller Thrust Equation Based Digital Control System for UAV," *2019 IEEE Bombay Section Signature Conference (IBSSC),* Mumbai, India*,* 2019, pp. 1-6.

[16] Green, M, Limebeer, D.J.N., "Sensors," *Linear and Robust Control,"* 1st ed., Mineola, New York, 1995, pp. 320.

[17] Joukhadar, A., Hasan, I., Alsabbagh, A., "Integral Lqr-Based 6dof Autonomous Quadrocopter Balancing System Control," *International Journal of Advanced Research in Artificial Intelligence*, Vol. 4, No. 5, 2015.

[18] Musa, S.," Techniques for Quadcopter Modelling & Design: A Review," *Journal of Unmanned System Technology*, May 2018.

[19] Luukkonen, T., "Modelling and control of quadcopter," Independent Research Project, School of Science, Aalto University, Espoo, Sweden, 2011.

[20] B. Erginer, and E. Altug, "Modeling and PD Control of a Quadrotor VTOL Vehicle," *2007 IEEE Intelligent Vehicles Symposium*, Istanbul, Turkey, 2007, pp. 894-899

[21] Sabatino, F., “Quadrotor control: modeling, nonlinear control design, and simulation,” Master’s Degree Project, Department of Electrical Engineering, KTH Royal Institute of Technology in Stockholm, Stockholm, Sweden, 2016.

[22] Gopalakrishnan Eswar, “Quadcopter flight mechanics model and control algorithms,” Master’s Thesis, Department of Control Engineering, Luleå University of Technology, Luleå, Sweden, 2016.

[23] Kurak, S., and Hodzic, M., “Control and estimation of a quadcopter dynamical model,” Periodicals of Engineering and Natural Sciences (PEN), Vol. 6, 2018, p. 63.

[24] Lyu, H., "Multivariable control of a rolling spider drone," Master’s Thesis, Electrical, Computer, and Biomedical Engineering, University of Rhode Island, South Kingstown, Rhode Island, 2017.

[25] Mellinger, D., Shomin, M., and Kumar, V., "Control of quadrotors for robust perching and landing," *Proceedings of the International Powered Lift Conference*, Philadelphia, Pennsylvania, 2010, pp. 205-225.

[26] Justa, J., Šmídl, V., and Hamáček, A., "Fast AHRS Filter for Accelerometer, Magnetometer, and Gyroscope Combination with Separated Sensor Corrections," *Sensors,* Basel, Switzerland, Vol. 20, No. 14, 2020, pp. 3824.

[27] Deng, H., Arif, U., Yang, K., "Global optical flow-based estimation of velocity for multicopters using monocular vision in GPS-denied environments," *Optik* (Stuttgart), Vol. 219, 2020, pp. 164923.

[28] Lefeber, E., Greiff, M., and Robertsson, A., "Filtered Output Feedback Tracking Control of a Quadrotor UAV," IFAC PapersOnLine, Vol. 53, No. 2, 2020, pp. 5764-5770.

[29] Furrer, F., Burri, M., Achtelik, M., and Siegwart, R., "RotorS—a modular gazebo MAV Simulator Framework," *Studies in Computational Intelligence*, Jan. 2016, pp. 595–625.

[30] Weinstein, A., Cho, A., Loianno, G., "Visual Inertial Odometry Swarm: An Autonomous Swarm of Vision-Based Quadrotors," *IEEE Robotics and Automation Letters,* Vol. 3, No. 3, 2018, pp. 1801-1807.

[31] Guardeño, R., López, M. J., and Sánchez, V. M., "MIMO PID controller tuning method for quadrotor based on LQR/LQG theory," *Robotics*, Vol. 8, 2019, p. 36.