Modeling, Control and Design of a Quadrotor Platform for Indoor Environments

by

Shi Lu

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

Approved November 2018 by the
Graduate Supervisory Committee:

Armando A. Rodriguez, Chair
Konstantinos Tsakalis
Jennie Si

ARIZONA STATE UNIVERSITY

December 2018

ABSTRACT

Unmanned aerial vehicles (UAVs) are widely used in many applications because of their small size, great mobility and hover performance. This has been a consequence of the fast development of electronics, cheap lightweight flight controllers for accurate positioning and cameras. This thesis describes modeling, control and design of an oblique-cross-quadcopter platform for indoor-environments.

One contribution of the work was the design of a new printed-circuit-board (PCB) flight controller (called $MARK3$). Key features/capabilities are as follows:

(1) a Teensy 3.2 microcontroller with 168MHz overclock –used for communications, full-state estimation and inner-outer loop hierarchical rate-angle-speed-position control, (2) an on-board MEMS inertial-measurement-unit (IMU) which includes an LSM303D (3DOF-accelerometer and magnetometer), an L3GD20 (3DOF-gyroscope) and a BMP180 (barometer) for attitude estimation (barometer/magnetometer not used), (3) 6 pulse-width-modulator (PWM) output pins supports up to 6 rotors (4) 8 PWM input pins support up to 8-channel 2.4 GHz transmitter/receiver for manual control, (5) 2 5V servo extension outputs for other requirements (e.g. gimbals), (6) 2 universal-asynchronous-receiver-transmitter (UART) serial ports - used by flight controller to process data from Xbee; can be used for accepting outer-loop position commands from NVIDIA TX2 (future work), (7) 1 I2C-serial-protocol two-wire port for additional modules (used to read data from IMU at 400 Hz), (8) a 20-pin port for Xbee telemetry module connection; permits Xbee transceiver on desktop PC to send position/attitude commands to Xbee transceiver on quadcopter.

The quadcopter platform consists of the new $MARK3$ PCB Flight Controller, an ATG-250 carbon-fiber frame (250 mm), a DJI Snail propulsion-system (brushless-three-phase-motor, electronic-speed-controller (ESC) and propeller), an HTC VIVE Tracker and RadioLink R9DS 9-Channel 2.4GHz Receiver. This platform is completely compatible with the HTC VIVE Tracking System (HVTS) which has 7ms latency, submillimeter accuracy and a much lower price compared to other millimeter-level tracking systems.

The thesis describes nonlinear and linear modeling of the quadcopter's 6DOF rigid-body dynamics and brushless-motor-actuator dynamics. These are used for hierarchical-classical-control-law development near hover. The HVTS was used to demonstrate precision hover-control and path-following. Simulation and measured flight-data are shown to be similar. This work provides a foundation for future precision multi-quadcopter formation-flight-control.

ACKNOWLEDGMENTS

First, I would like to thank my MS thesis advisor Dr. Armando Antonio Rodriguez for his continuous guidance and assistance on my research. His guidance helped me throughout the important parts of research and thesis writing. I am especially grateful for the support, suggestions, and encouragement he gave me in this thesis.

I take this opportunity to express my gratitude to the members of my thesis committee, Dr. Konstantinos Tsakalis and Dr. Jennie Si for their support. I also thank my parents for their encouragement, attention and support. I am also grateful to my research colleagues who supported me throughput this journey, especially Karan Puttannaiah, Nirangkush Das and Abdullah Altawaitan. They encouraged me a lot throughout this tough task on drone research. Their help has been greatly appreciated.

TABLE OF CONTENTS

Page

APPENDIX

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

Chapter 1

OVERVIEW

1.1 Introduction and Motivation

In the last few decades, quadrotors have been used for many industrial and agricultural applications. The need for UAV with greater maneuverability and hovering ability compared with fixed-wing aircraft has led to a rise in quadrotor research. Research continuously increases the abilities of quadcopters in stability, and maneuverability. Quadrotors are capable of advanced autonomous missions like formation flight and environment exploration. Also quadrotors exhibit a good degree of decoupling, which makes the flight controller design easier than helicopters.

The work of this thesis is the first step of achieving a quadrotor swarm mission. Potential applications can include: manufacturing, transportation, firework display and much more. A flight controller with a teensy 3.2 microcontroller and MEMS sensors is designed to develop a low-cost quadrotor platform that can be used for formation flight. With development of Virtual Reality Entertainment System, we can use low-cost indoor tracking devices (HTC VIVE Virtual Reality System) to do indoor tracking with millimeter-level accuracy instead of expensive motion capture system like VICON and Optitrack.

Plant model of this quadrotor platform within both rigid body dynamics and actuator dynamics is examined. In order to design full- state feedback cascade controllers for a quadrotor, the nonlinear model of rigid body dynamics need to be linearized under the small roll and pitch movement assumption (hovering mode). Additionally, the actuator also requires linearization before controllers designed for each separate system input.

Control design and implementation have high priority in the applications of quadrotors. Many control methods have been proposed for quadrotor control problem, such as PID, LQR, backstepping nonlinear control and sliding mode control. In practice, cascaded feedback control is the most widely used quadrotor control technique providing comparable or even better performance than more complex controllers.

1.2 Literature Survey

To introduce quadrotor modeling, hardware, design, and control, the following literature survey is offered. An approach is made below to indicate what papers or works are most relevant to this thesis. For short, the following works are most relevant for the developments within this thesis

- quadrotor linear control work within: [1] and [2]
- quadrotor modeling work within: [5] and [6]
- quadrotor parameters measurement work within: [7]
- design of quadrotor flight controller and ground station architecture within: [8] and [9]
- quadrotor state estimation within: [11], [14] and [23]

An attempt is made below to provide relevant leading technical details.

- **Quadrotor Modeling** Within this thesis, kinematics, rigid-body dynamics and acutator dynamics are represented as a central focus of the work. Here we assume quadrotor including frame, propulsion system and flight controller as a rigid body. And we assume 4 ESC-motor-propeller sets (propulsion system) are identical. The actuator inputs are voltages and PWM (Pulse Width Modulation) signals. Two motors are rotating in clockwise direction (CW mode) while other two are rotating in counterclockwise direction (CCW mode). The thrust generated by four propellers produces total thrust and torque in roll and pitch movement. The torque generated by four motors produces torque in yaw movement.

Kinematic Model: A kinematic model of quadrotor is presented [5]. Here we use Euler angle to represent roll, pitch and yaw angle on linearized model, modeling analysis and linear control. Quaternion is used to represent attitude on design of nonlinear state estimator for low cost of microcontroller calculation, avoidance of bad use of singular value and prevention of gimbal lock [15].

Dynamic Model: The dynamic model of quadrotor consists of two parts: rigid dynamics and actuator dynamics. For rigid dynamics, we assume the whole quadrotor is a rigid body and the center of frame matches the center of mass. Based on Newton's second law, we can get the rigid dynamics for positional movement and angular movement. For actuator dynamics, we assume all four sets of actuator are identical and the actuator model is an ideal ESC-motor system [2]. From actuator testing, we can get the mapping from PWM signal to desired rotation speed of motors in order to represent the actuator model with a first-order transfer function.

- **Quadrotor Control** The quadrotor control is split into a low-level part for attitude control and a high-level part for position control. The desired orientation and the desired thrust command are outputs of high-level position control. These desired values are inputs of the low-level attitude control and decoupled as the direct command for all four motors.

Low Level Control: The low-level controller is designed for tracking the desired orientation generated from the high-level controller. It is split into angular rate control as inner loop and attitude control as outer loop. The angular rate controller is based on PD control [3] law (Classical control design). It also corresponds to an LQR controller for a dynamical system containing the body rates and body torques as state [4].

High Level Control: The high-level controller consists of translational movement control and vertical movement (altitude control). Both can be split into position control as outer loop and velocity control as inner loop. The reference input of the outer loop is the desired position and the reference input of the inner loop is the desired velocity. The nonlinear constraint must be added to the output of the velocity control in the real flight controller. Here we use P-PD cascade control to perform high-level controller.

- **Design of Quadrotor Flight Controller and Ground Station Architecture**

  Quadrotor Flight Controller Design: The flight controller consists of a Microcontroller Unit, an IMU Module, Power Modules and a Communication Module. It is also a hub offering enough design redundancy for many other important peripherals on the quadrotor like 4 ESCs (Electronic Speed controller), a 2.4Ghz Radio System Receiver, I2C/UART devices, etc. The firmware is programmed to achieve high-level/low-level control, communication process, state estimation and power/device management function.

Ground Station Architecture Design: The ground station consists of motion capture system (HTC VIVE) and a desktop [10]. The desktop will process data from SteamVR API and send flight data pakage and command from mission planner to the flight controller on the quadrotor through the communication module. Besides, it is responsible for monitoring the flight status of the quadrotor using a GUI written in MATLAB.

- **State Estimation**

  Sensor Calibration: The onboard MEMS sensors (accelerometer, gyroscope) have bias and they are sensitive with mechanical noise. Calibration based on sensor dynamics need to be designed to ensure that the collected sensor data is close to real value.

  IIR Filtering: The MEMS sensors require low-pass filtering to reduce the influence of noise during flight. The classic infinite impulse response digital filter is applied to the output of accelerometer and gyroscope.

  Full-State Estimation: A full-state nonlinear complementary filter augmented by the 6-DOF nonlinear model of quadrotor rigid body dynamics is designed as low-cost computing state estimator in the flight controller firmware. The attitude estimator is based on an explicit complimentary filter [12] obtained from accelerometer data which has MOCAP compensation and gyroscope data. The position & velocity estimator is based on a general complementary filter fusing MOCAP data and accelerometer data in world frame.

The literature survey of this thesis are of importance especially to those interested in quadrotor research.

1.3 Contribution of Work: Questions to be addressed

Within this thesis, the following fundamental questions are addressed. When taken collectively, the answers offered below, the details within the thesis, represent a useful contribution to researchers in the field. Moreover, it must be emphasized that answer to thes questions are critical in order to move substantivey toward the research on formation flight.

1. **What does a flight controller consist of?** Referring to popular flight controllers on the market (Multiwii, CC3D, Pixhawk, etc), a flight controller consists of: (1) a Microcontroller Unit that offers enough computing power. (2) a MEMS (micro-electro-mechanical) IMU (Inertial Measurement Unit). (3) a Communication Module or at least a socket for it. (4) a Power Module that gives stable 3.3$v \sim 5v$ voltage. The Mark3 Flight Controller is shown in figure

   (1) *MCU* Teensy 3.2 MCU which can be overclocked over 96Mhz (See Figure 1.1) offers enough computing capacity to execute high-level/low-level control and state estimation with low-cost computing work.

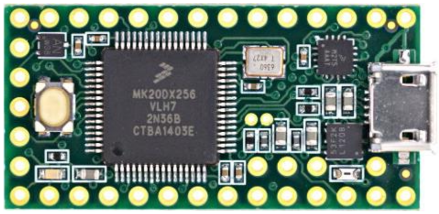


Figure 1.1: Teensy 3.2 Microcontroller Unit

   (2) *IMU* GY-89 10DOF Sensor Module (See Figure 1.2) carrying L3GD20 (Gyroscope), LSM303D (Accelerometer and Magnetometer) and BMP180 (Barometer) is capable of measuring rotation states during flight and giving acceleration

data. The gyroscope gives angular rate and the accelerometer gives resultant force vector of the quadrotor. Currently we are not using the magnetometer and the barometer because of environmental impacts on these sensors and high RMSE (Root-Mean-Square Deviation).

Figure 1.2: GY-89 10DOF Sensor Module

(3) *Communication Module* XBee 3.0 (See Figure 1.3) is the communication module for all protocols including: ZigBee, 802.15.4, DigiMesh, BLE, etc with up to 250 Kbps RF bandwidth and up to 200 ft indoor working range.

Figure 1.3: Digi XBee3 Zigbee 3.0

(4) _Power Module_ The working voltage of MCU, IMU and Communication Module is 3.3v. This step down voltage regulator (See Figure 1.4) gives stable 3.3v power supply.

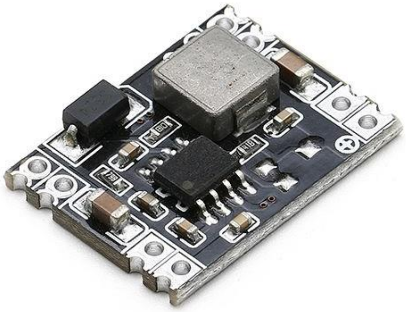


Figure 1.4: 5V to 3.3V Step Down Voltage Regulator

2. **How to choose other frame components?** Here we choose $250mm$ size carbon fiber frame (See Figure 1.5)

Figure 1.5: 250mm quadrotor carbon fiber frame

which offers enough firmness for this platform. Typical actuator sets are designed for this kind of frame from different manufacturers (EMax, T-motor, DJI, etc). An actuator set consists of a propeller, a brushless motor and an ESC (Electronic Speed Controller). A quadrotor has four actuator sets.

Figure 1.6: Actuator Set (Propulsion System)

The most important criterion too choose actuator is the settling time of step response. The Snail Propulsion System shown in (See Figure 1.6) figure gives the shortest settling time compared with other actuator sets we have tested which is illustrated in Chapter 3.

3. **What is the suitable indoor positioning system for a low-cost platform?** Expensive motion capture systems from Optitrack or VICON that provides low latency data with millimeter-level accuracy are the premier solution for UAV and Robotic studies in the labs like UPENN Grasp Lab, Bristol Robotics Lab, etc. As development of the virtual reality entertainment system grows fast, we can use cheap devices to get similar performance to get low latency data with millimeter-level accuracy.

Figure 1.7: An idea to use HTC VIVE trackers to do robot localization

HTC VIVE Virtual Reality System offers a solution to achieve accurate indoor tracking with very low cost compared with other expensive motion capture systems based on cameras and markers. As shown in Figure 1.7, the HTC

VIVE tracker placed on the gaming rifle can be also placed on a quadrotor.

4. **What is a suitable low-level control structure?** The low-level control consists of angular rate control and attitude control. For angular rate control, a simple PD control law suffices (In Chapter 5). It also corresponds to an LQR design considering actuator dynamics. For attitude control, a simple proportional control works based on the assumptions of the quadrotor model (In Chapter 2).

5. **What is a suitable high-level control structure?** The high-level control consists of altitude-vertical-velocity control and translational movement control. The altitude and vertical velocity control has the same structure as angular rate and attitude control. The quadrotor model is linearized at hovering state. So we can use P-PD structure for translational movement control (In Chapter 5).

While partial answers have been provided above, the thesis (when applicable) provedes more detailed anwsers. When taken collectively, the contributions of this thesis are significant - particularly to those interested in developing low-cost platforms for conducting quadrotor research.

1.4 Organization of Thesis

The remainder of the thesis is organized as follows.

- Chapter 2 (page 14) presents nonlinear model and linearization of quadrotor kinematrics, rigid body dynamics and actuator dynamics.
- Chapter 3 (page 33) describes the parameters of quadrotor rigid body dynamics and actuator dynamics measurement. This chapter also describes design of the *MARK3* flight controller and general frame structure of the hardware and software.

- Chapter 4 (page 47 presents analysis of the linearized model including angular movement and positional movement when quadrotor works near hovering mode.

- Chapter 5 (page 53) describes design of low-level control and high-level control of quadrotor hovering mode along with the simulation results.

- Chapter 6 (page 86) introduces full-state estimation based on sensor fusion of accelerometer, gyroscope and HTC VIVE Tracking System.

- Chapter 7 (page 99) presents hardware result of low-level control and high-level control along with the simulation plots.

- Chapter 8 (page 102) summarizes the thesis and presents direction for future robotics research. While much has been accomplished in this thesis, lots remain to be done.

- Appendix A (page 108) contains all MATLAB mfiles used to generate the simulation results for this thesis.

- Appendix B (page 136) contains MATLAB GUI code for UART communication between the quadrotor platform and the upper computer and UDP protocol for receiving data from SteamVR API.

- Appendix C (page 144) contains the firmware for the flight controller.

- Appendix D (page 199) contains hardware assembly instructions & software initialization for this indoor quadrotor platform to show simple indoor flight demo.

1.5 Summary and Conclusions

In this chapter, we provided an overview of the work presented in this thesis and the major contributions. A central contribution of the thesis is a low-cost quadrotor platform which is compatible with HTC VIVE tracking system that can be used for drone formation research. A simple formation demonstration was conducted with two quadrotors using the *MARK3* flight controller and HTC VIVE trackers. The thesis attempts to address most critical modeling, design, and control issues in detail - as needed.

Chapter 2

NONLINEAR MODEL & LINEARIZATION

2.1 Introduction and Overview

In this chapter, we describe the nonlinear model of the quadrotor kinematics, rigid-body dynamics, actuator dynamics, and model linearization. In order to design the control system at equilibrium point, we need to analyze and simplify quadrotor rigid-body dynamics relying on small angle assumptions for roll and pitch movement. For actuator dynamics, we can use a first-order transfer function to reproduce correlation between set-point rotor speed and actual rotor speed.

2.2 Quadrotor Nonlinear Model

2.2.1 Assumptions

The modeling of quadrotor is based on following assumptions.

- The whole quadrotor is a rigid body.
- The quadrotor frame is symmetrical.
- The center of frame matches the center of mass.
- The inertia of motor is small and neglected.
- The range of pitch movement and roll movement is small.

2.2.2 Kinematics

There are two types of quadrotor frame setup. They are 'x' configuration and '+' configuration shown in figure 2.1. While doing pitch or roll movement, the quadrotor with '+' configuration only uses 2 rotors to produce roll movement or pitch movement while the one with 'x' configuration uses all four rotors. We use 'x' configuration to fully use all four rotors for more available torque in roll and pitch movement.

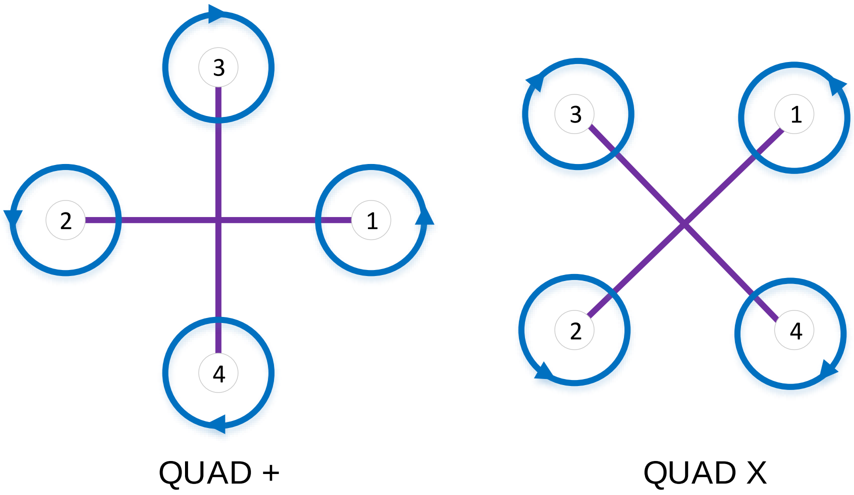


Figure 2.1: Quadrotor Frame Setup

Here we let $\xi = [x, y, z]^T$ to represent position of the quadrotor in the inertial frame. Where x axis points east, y axis points north, and z axis points up. $\xi_b = [x_b, y_b, z_b]^T$ represents position of the quadrotor in the body frame. $\mathbf{V} = [v_x, v_y, v_z]^T$ represents velocity of the quadrotor in the inertial frame. $\Theta = [\phi, \theta, \psi]^T$ represents angular position of the quadrotor in the inertial frame. **Yaw angle**, denoted by ψ, represents rotation along z axis. **Pitch angle**, denoted by θ, represents rotation along y axis. **Roll angle**, denoted by ϕ, represents rotation along x axis. $\nu = [p, q, r]^T$

represents the angular rate of the quadrotor in body frame.

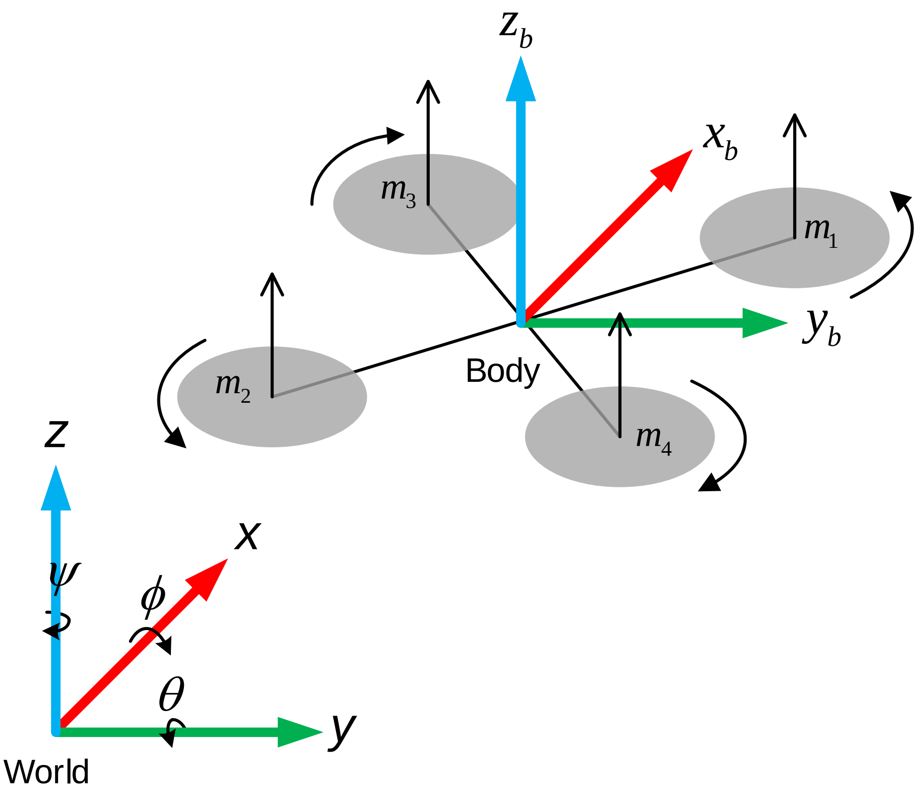


Figure 2.2: Quadrotor Coordinate Diagram

We use rotation matrix [17] based on $Z-Y-X$ Eurler angles to present rigid-body vector that rotates from body frame to inertial frame shown in figure 2.2.

- Rotation about z axis is

$$R_\psi = \begin{bmatrix} cos(\psi) & sin(\psi) & 0 \\ -sin(\psi) & cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.1}$$

- Rotation about y axis is

$$R_\theta = \begin{bmatrix} cos(\theta) & 0 & sin(\theta) \\ 0 & 1 & 0 \\ -sin(\theta) & 0 & cos(\theta) \end{bmatrix} \tag{2.2}$$

- Rotation about x axis is

$$R_\phi = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\phi) & -sin(\phi) \\ 0 & sin(\phi) & cos(\phi) \end{bmatrix} \tag{2.3}$$

Then we have the rotation matrix from body coordinate to inertial coordinate.

$$R_{E\to B} = R_\phi R_\theta R_\psi = \begin{bmatrix} cos(\theta)cos(\psi) & cos(\theta)sin(\psi) & sin(\theta) \\ -cos(\phi)sin(\psi) + cos(\psi)sin(\phi)sin(\theta) & cos(\phi)cos(\psi) + sin(\phi)sin(\theta)sin(\psi) & -cos(\theta)sin(\phi) \\ -cos(\phi)cos(\psi)sin(\theta) - sin(\phi)sin(\psi) & cos(\psi)sin(\phi) - cos(\phi)sin(\theta)sin(\psi) & cos(\phi)cos(\theta) \end{bmatrix} \tag{2.4}$$

The above matrix is orthonormal. So we can have the rotation matrix from body coordinate to inertial coordinate by taking the transpose of $R_{E\to B}$.

$$R_{B\to E} = R^T_{E\to B} \tag{2.5}$$

And we have

$$R_{B\to E} = \begin{bmatrix} cos(\psi)cos(\theta) & -cos(\phi)sin(\psi) + cos(\psi)sin(\phi)sin(\theta) & -cos(\phi)cos(\psi)sin(\theta) - sin(\phi)sin(\psi) \\ cos(\theta)sin(\psi) & cos(\phi)cos(\psi) + sin(\phi)sin(\psi)sin(\theta) & cos(\psi)sin(\phi) - cos(\phi)sin(\psi)sin(\theta) \\ sin(\theta) & -cos(\theta)sin(\phi) & cos(\phi)cos(\theta) \end{bmatrix} \tag{2.6}$$

Thus

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = R_{B\to E} \begin{bmatrix} x_b \\ y_b \\ z_b \end{bmatrix} \tag{2.7}$$

The Euler rates of the quadrotor is same as other aircrafts [18]. It can be used to determine the attitude of the quadrotor. The relation between the euler rates and the body angular rates is

$$\begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} \dot{\phi} \\ 0 \\ 0 \end{bmatrix} + R_\phi \begin{bmatrix} 0 \\ \dot{\theta} \\ 0 \end{bmatrix} + R_\phi R_\theta \begin{bmatrix} 0 \\ 0 \\ \dot{\psi} \end{bmatrix} = \Omega_{E\to B} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \tag{2.8}$$

Then

$$\Omega_{E\to B} = \begin{bmatrix} 1 & 0 & sin(\theta) \\ 0 & cos(\phi) & -sin(\phi)cos(\theta) \\ 0 & sin(\phi) & cos(\theta)cos(\phi) \end{bmatrix} \tag{2.9}$$

And

$$\Omega_{B\to E} = \Omega_{E\to B}^{-1} = \begin{bmatrix} 1 & sin(\phi)tan(\theta) & -cos(\phi)tan(\theta) \\ 0 & cos(\phi) & sin(\phi) \\ 0 & -\frac{sin(\phi)}{cos(\theta)} & \frac{cos(\phi)}{cos(\theta)} \end{bmatrix} \tag{2.10}$$

Finally

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{cases} p + sin(\phi)tan(\theta)q - cos(\phi)tan(\theta)r \\ cos(\phi)q + sin(\phi)r \\ -\frac{sin(\phi)}{cos(\theta)}q + \frac{cos(\phi)}{cos(\theta)}r \end{cases} \tag{2.11}$$

2.2.3 Dynamics

According to Newton's second law of motion, the mass center motion kinematics equation of quadrotor is:

$$\frac{d(m\vec{V})}{dt} = \vec{F} \tag{2.12}$$

Thrust generated by each motor is T_i $(i = 1, 2, 3, 4)$. So the **total thrust** is

$$T = T_1 + T_2 + T_3 + T_4 \tag{2.13}$$

The inverse torque required to generate **yaw moment** is generated by each motor. Where m_i $(i = 1, 2, 3, 4)$. The total inverse torque generated by four motors is

$$\tau_\psi = m_1 + m_2 - m_3 - m_4 \tag{2.14}$$

The diffierential thrust generated by 4 motors generates **pitch moment** and **roll moment**. l is the distance between each motor and the center of the frame.

- **Pitch Movement**

For moving in positive x direction, the rotation speed of motor 1 and 3 is decreased and that of motor 2 and 4 is increased as shown in Figure 2.3

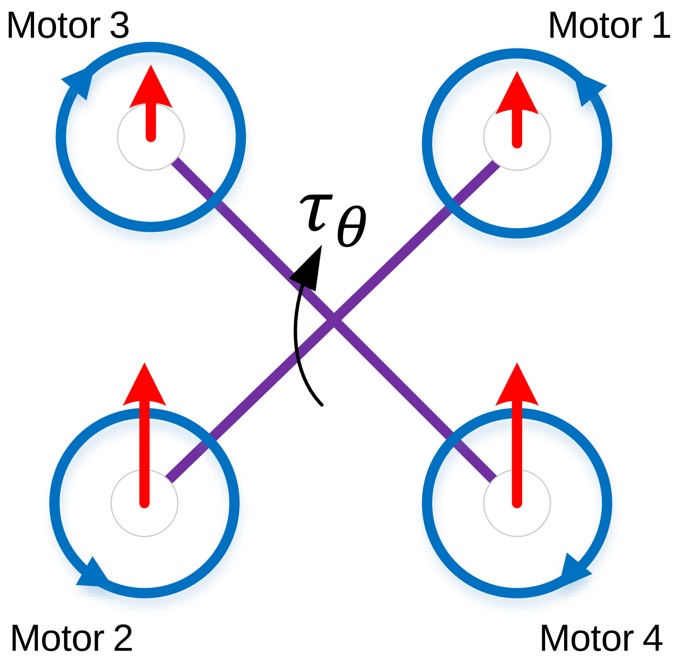


Figure 2.3: Pitch Movement

- **Roll Movement**

For moving in positive y direction, the rotation speed of motor 2 and 3 is decreased and that of motor 1 and 4 is increased as shown in Figure 2.4

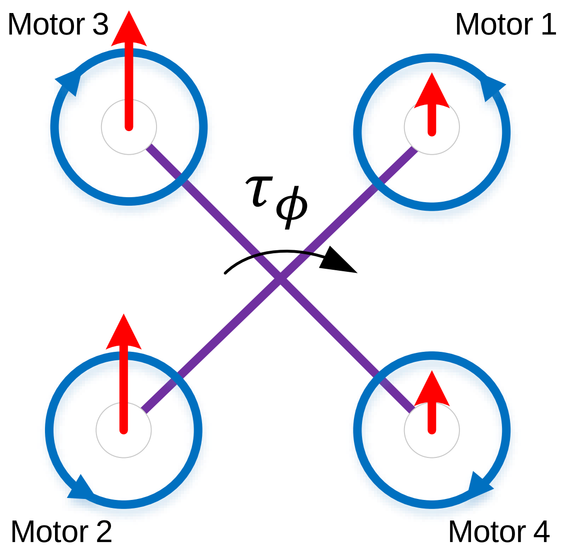


Figure 2.4: Roll Movement

- **Yaw Movement**

For making quadrotor rotate around z axis in body frame, the rotation speed of motor 3 and 4 is decreased and that of motor 1 and 2 is increased as shown in Figure 2.5

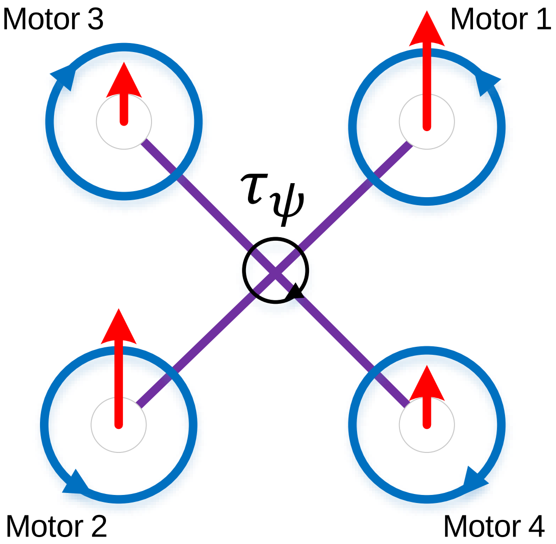


Figure 2.5: Yaw Movement

And we have

$$\tau_\theta = \frac{\sqrt{2}}{2} l(T_1 + T_3 - T_2 - T_4) \tag{2.15}$$

Similarly

$$\tau_\phi = \frac{\sqrt{2}}{2} l(T_2 + T_3 - T_1 - T_4) \tag{2.16}$$

The air drag[19] related to ground coordinate system is

$$\begin{bmatrix} f_x \\ f_y \\ f_z \end{bmatrix} = \begin{bmatrix} K_1\dot{x} \\ K_2\dot{y} \\ K_3\dot{z} \end{bmatrix} \tag{2.17}$$

Then the dynamics equations of force is:

$$m\begin{bmatrix}\dot{v_x}\\ \dot{v_y}\\ \dot{v_z}\end{bmatrix}=mg\begin{bmatrix}0\\0\\-1\end{bmatrix}+R_{B\to E}T\begin{bmatrix}0\\0\\1\end{bmatrix}-\begin{bmatrix}K_1v_x\\K_2v_y\\K_3v_z\end{bmatrix} \tag{2.18}$$

So position movement of quadrotor can be expressed as:

$$\begin{bmatrix}\ddot{x}\\ \ddot{y}\\ \ddot{z}\end{bmatrix}=\begin{bmatrix}\dot{v_x}\\ \dot{v_y}\\ \dot{v_z}\end{bmatrix}=\begin{bmatrix}\frac{T}{m}(-cos(\psi)sin(\theta)cos(\phi)-sin(\psi)sin(\phi))-K_1\dot{x}\\ \frac{T}{m}(cos(\psi)sin(\phi)-sin(\psi)sin(\theta)cos(\phi))-K_2\dot{y}\\ \frac{T}{m}cos(\phi)cos(\theta)-g-K_3\dot{z}\end{bmatrix} \tag{2.19}$$

The rotation kinematics equation of quadrotor is:

$$\frac{d(J\nu)}{dt}=M \tag{2.20}$$

And $J=diag[J_x,J_y,J_z]$ is quadrotor moments of inertia related to 3 axes of body coordinate system. M is the resultant moment applied on the quadrotor. ($M=\dot{H}+\nu\times H$ and $H=J\nu$) External moments mainly consist of body torque and aerodynamic drag torque. Body torque (2.14) generated by rotors is:

$$\begin{bmatrix}\tau_\phi\\ \tau_\theta\\ \tau_\psi\end{bmatrix}=\begin{bmatrix}\frac{\sqrt{2}}{2}l(T_2+T_3-T_1-T_4)\\ \frac{\sqrt{2}}{2}l(T_1+T_3-T_2-T_4)\\ m_1+m_2-m_3-m_4\end{bmatrix} \tag{2.21}$$

Aerodynamic drag torque is:

$$\tau_{af}=K_{af}\nu \tag{2.22}$$

Where $K_af=diag[k_{afx},k_{afy},k_{afz}]$. So dynamics equations of torque is

$$M = \begin{bmatrix} M_x \\ M_y \\ M_z \end{bmatrix} = \begin{bmatrix} \dot{p}J_x + qr(J_z - J_y) \\ \dot{q}J_y + pr(J_x - J_z) \\ \dot{r}J_z + pq(J_y - J_x) \end{bmatrix} = \begin{bmatrix} \tau_\phi - \tau_{afx} \\ \tau_\theta - \tau_{afy} \\ \tau_\psi - \tau_{afz} \end{bmatrix} \tag{2.23}$$

Then we have the equations of angular movement of the quadrotor

$$\begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} \frac{J_y - J_z}{J_x}qr + \frac{\tau_\phi}{J_x} - \frac{\tau_{afx}}{J_x} \\ \frac{J_z - J_x}{J_y}pr + \frac{\tau_\theta}{J_y} - \frac{\tau_{afy}}{J_y} \\ \frac{J_x - J_y}{J_z}pq + \frac{\tau_\psi}{J_z} - \frac{\tau_{afz}}{J_z} \end{bmatrix} \tag{2.24}$$

Then we can get the whole nonlinear quadrotor rigid body dynamics

$$\dot{\mathbf{X}} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{v_x} \\ \dot{v_y} \\ \dot{v_z} \\ \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \\ \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \begin{cases} \dot{x} = v_x \\ \dot{y} = v_y \\ \dot{z} = v_z \\ \dot{v_x} = \frac{T}{m}(-cos(\psi)sin(\theta)cos(\phi) - sin(\psi)sin(\phi)) - K_1 v_x \\ \dot{v_y} = \frac{T}{m}(cos(\psi)sin(\phi) - sin(\psi)sin(\theta)cos(\phi)) - K_2 v_y \\ \dot{v_z} = \frac{T}{m}(cos(\phi)cos(\theta)) - g - K_3 v_z \\ \dot{\phi} = p + sin(\phi)tan(\theta)q - cos(\phi)tan(\theta)r \\ \dot{\theta} = qcos(\phi) + rsin(\phi) \\ \dot{\psi} = -\frac{sin(\phi)}{cos(\theta)}q + \frac{cos(\phi)}{cos(\theta)}r \\ \dot{p} = \frac{(J_y - J_z)}{J_x}qr + \frac{\tau_\phi}{J_x} - K_4 p \\ \dot{q} = \frac{J_z - J_x}{J_y}pr + \frac{\tau_\theta}{J_y} - K_5 q \\ \dot{r} = \frac{J_x - J_y}{J_z}pq + \frac{\tau_\psi}{J_z} - K_6 r \end{cases} \tag{2.25}$$

which is a non-linear system. We need to linearize the quadrotor kinematics and rigid body dynamics at a near-hover state where ϕ, θ and ψ are close to zero.

2.3 Actuator Model

The aerodynamic force and moment are obtained by combining the momentum theory of the blade element. The torque and the force generated by each rotor-propeller are propotional to the square of the propeller speed as

$$T_i = C_T \frac{4\rho_a R^4}{\pi^2} \Omega_i^2 \tag{2.26}$$

$$m_i = C_Q \frac{4\rho_a R^5}{\pi^3} \Omega_i^2 \tag{2.27}$$

Where ω_m is rotor rotation speed,ρ_a is the air density, R is the propeller radius, C_T is the thrust factor, and C_Q is the momentum factor. We simplify these equations as

$$T_i = b\Omega_i^2 \tag{2.28}$$

$$m_i = d\Omega_i^2 \tag{2.29}$$

Assume the voltage input is u, the current is I, and the rotational speed of rotor of each motor is Ω_i. According to Kirchhoff laws, we can get

$$L\frac{di}{dt} = u - RI - K_e\Omega_i \tag{2.30}$$

R is the equivalent resistance of the motor, L is equivalent inductance of the motor, and K_e is voltage coefficients.

torque equilibrium equation during rotation is

$$L\frac{d\Omega_i}{dt} = k_m i - d\Omega_i^2 \tag{2.31}$$

J is the equivalent moment of inertia of motor. τ_m is motor torque and τ_d is loading torque. L is negligible because we are using small brushless motor. Then we can get approximate motor dynamic model.

$$\dot{\Omega}_i = \frac{K_m K_e}{RJ}\Omega_i - \frac{d}{j}\Omega_i^2 + \frac{K_m}{RJ}u \tag{2.32}$$

Use taylor expansion, remove high-order terms, keep the first-order term. we can get the linearized equation.

$$\dot{\Omega}_i = -A\Omega_i + Bu + C \tag{2.33}$$

Then

$$\frac{\Omega(s)}{u} = \frac{z_{vol}}{s+a} \tag{2.34}$$

In practical use, we cannot directly read the voltage as the input of the brushless motor. The ESC of each actuator set only accept PWM signal as command. By curve fitting in Chapter 3, we can have the mapping from PWM signal to desired motor speed Ω^* which includes the mapping from PWM signal to input voltage u and that from input voltage u to desired motor speed Ω^*. The transfer function between desired motor speed Ω^* and actual motor speed Ω can be presented as a first-order low-pass filter.

$$\frac{\Omega(s)}{\Omega^*(s)} = \frac{a}{s+a} \tag{2.35}$$

2.4 Linearization

- **Linearization of rigid-body dynamics**

The non-linear rigid-body dynamics is represented as

$$\dot{\mathbf{X}}_{rig} = f(\mathbf{X}_{rig}, \mathbf{U}_{rig}) \tag{2.36}$$

Where

$$\mathbf{X}_{rig} = \begin{bmatrix} \xi \\ \mathbf{V} \\ \Theta \\ \nu \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ v_x \\ v_y \\ v_z \\ \phi \\ \theta \\ \psi \\ p \\ q \\ r \end{bmatrix} \tag{2.37}$$

And

$$\mathbf{U}_{rig} = \begin{bmatrix} U_1 \\ U_2 \\ U_3 \\ U_4 \end{bmatrix} = \begin{bmatrix} T \\ \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix} \tag{2.38}$$

$\mathbf{X}_{rig}$ represents the state vector and $\mathbf{U}_{rig}$ represents the input vector. Trim Points are $\mathbf{X}_0 = [0, 0, 0, 0, 0, 0, 0, 0, \psi_0, 0, 0, 0]$ and $\mathbf{U}_0 = [mg, 0, 0, 0]^T$. Then we have

$$f(\mathbf{X}_{equil}, \mathbf{U}_{equil}) = 0 \tag{2.39}$$

Based on [20], The linear equation can be represented as

$$\delta\dot{\mathbf{X}} = A\delta\mathbf{X} + B\delta\mathbf{U} \tag{2.40}$$

The characteristic matrix A and the input matrix B can be calculated from

$$A = \frac{\delta f}{\delta \mathbf{X}}|_{\mathbf{X}=\mathbf{X}_{equil},\mathbf{U}=\mathbf{U}_{equil}} \tag{2.41}$$

$$B = \frac{\delta f}{\delta \mathbf{U}}|_{\mathbf{X}=\mathbf{X}_{equil},\mathbf{U}=\mathbf{U}_{equil}} \tag{2.42}$$

Then we have

$$A = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -K_1 & 0 & 0 & 0 & -g & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -K_2 & 0 & g & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -K_3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -K_4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -K_5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -K_6 \end{bmatrix} \tag{2.43}$$

And

$$B = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{m} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & \frac{1}{J_x} & 0 & 0 \\ 0 & 0 & \frac{1}{J_y} & 0 \\ 0 & 0 & 0 & \frac{1}{J_z} \end{bmatrix} \tag{2.44}$$

Then we have the linearized quadrotor rigid-body dynamics

$$\dot{\mathbf{X}}_{rig} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{v_x} \\ \dot{v_y} \\ \dot{v_z} \\ \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \\ \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} v_x \\ v_y \\ v_z \\ -g\theta - K_1 v_x \\ g\phi - K_2 v_y \\ \frac{T}{m} - g - K_3 v_z \\ p \\ q \\ r \\ \frac{\tau_\phi}{J_x} - K_4 p \\ \frac{\tau_\theta}{J_y} - K_5 q \\ \frac{\tau_\psi}{J_z} - K_6 r \end{bmatrix} = A\mathbf{X}_{rig} + B\mathbf{U}_{rig} \tag{2.45}$$

- **Linearization of actuator dynamics**

The full actuator model is shown in Figure 2.6. Obviously we have $\mathbf{U}_{act} = [T, \tau_\phi, \tau_\theta, \tau_\psi]^T$.

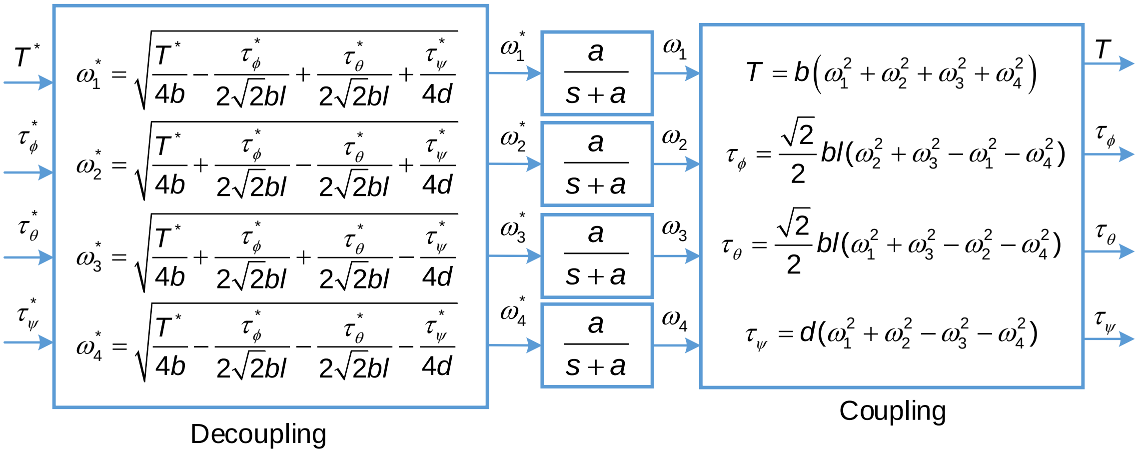


Figure 2.6: Full Actuator Dynamics

Here we use simulink toolbox shown in Figure 2.7 to linearized the nonlinear actuator dynamic model. The trim points are $U_0^* = [mg, 0, 0, 0]^T$, $U_0 = [mg, 0, 0, 0]^T$ and $\omega_1 = \omega_2 = \omega_3 = \omega_4 = 842.99 rad/s$ (Optimization Method: Gradient Descent with Elimination; Algorithm: Active-Set).

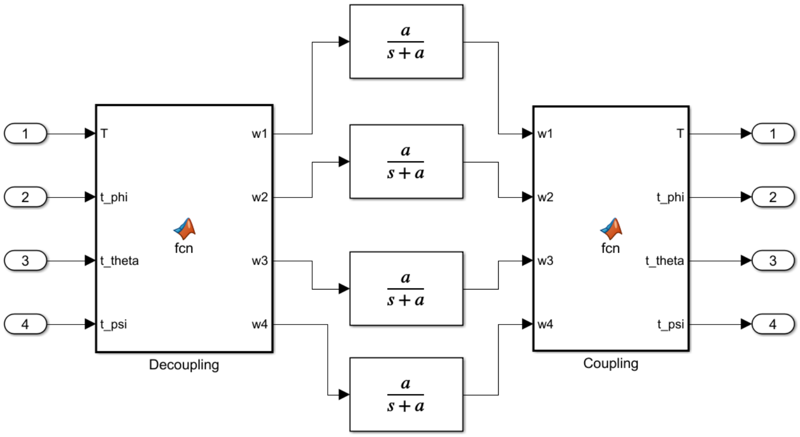


Figure 2.7: Using Simulink Toolbox

And we get the linearized actuator dynamic model in Figure 2.8

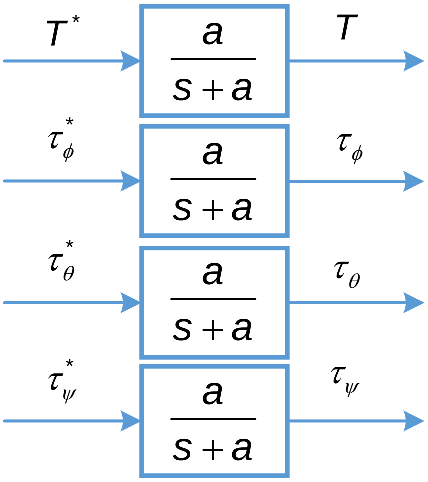


Figure 2.8: Linearized Actuator Dynamics

.

With $\mathbf{U}_{act}^* = [T^*, {\tau_\phi}^*, {\tau_\theta}^*, {\tau_\psi}^*]^T$, linearized actuator model (state space representation) can be written as:

$$\dot{\mathbf{U}}_{act} = -a\mathbf{I}_{4\times4}\mathbf{U}_{act} + a\mathbf{I}_{4\times4}\mathbf{U}^*_{act} \tag{2.46}$$

- **Whole Quadrotor Linearized Model**

Combining the linearized quadrotor rigid body dynamic model and the linearized acutator model. Assuming all drag coefficients are zero, the whole quadrotor linearized model (state space representation) is shown below:

$$\dot{\mathbf{X}} = \begin{bmatrix} \mathbf{I}_{3\times3} & & \mathbf{0}_{3\times13} \\ \mathbf{0}_{1\times3} & -g & \mathbf{0}_{1\times12} \\ \mathbf{0}_{1\times4} & g & \mathbf{0}_{1\times11} \\ \mathbf{0}_{1\times12} & \frac{1}{m} & \mathbf{0}_{1\times3} \\ \mathbf{0}_{3\times6} & \mathbf{I}_{3\times3} & \mathbf{0}_{3\times7} \\ \mathbf{0}_{3\times9} & diag[\frac{1}{J_x}, \frac{1}{J_y}, \frac{1}{J_z}] & \mathbf{0}_{3\times4} \\ \mathbf{0}_{4\times12} & & -a\mathbf{I}_{4\times4} \end{bmatrix} \mathbf{X} + \begin{bmatrix} \mathbf{0}_{12\times4} \\ a\mathbf{I}_{4\times4} \end{bmatrix} \mathbf{U} \tag{2.47}$$

Chapter 3

PARAMETERS MEASUREMENT AND HARDWARE IMPLEMENTATIONS

3.1 Introduction and Overview

This chapter is illustrates measurement of quadrotor rigid-body and actuator parameters with bifilar pendulum and propulsion system test. Also the design of the *MARK3* flight controller and the general frame structure of hardware and software are presented.

3.2 Airframe Size, Mass and Moment Measurement

3.2.1 Airfame Size

Due to the limitation of volume of indoor test area, we choose 250mm quadrotor frame as the quadrotor airframe which is made of carbon fiber with good sturdiness and light weight. As shown in Figure 3.1, the distance from center of frame to the each motor $l = 0.125m$.

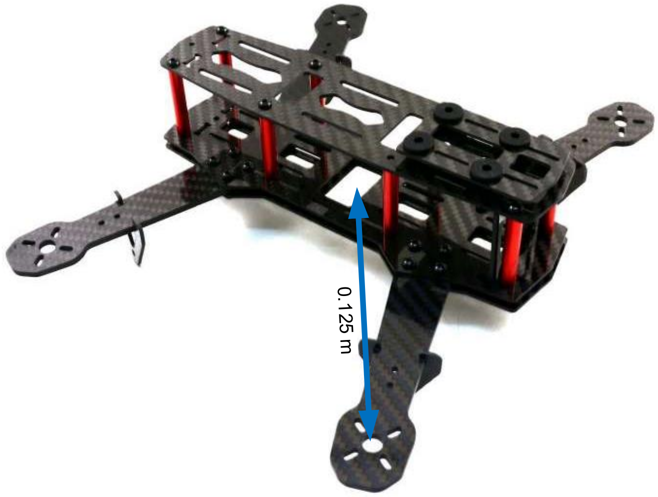


Figure 3.1: 250 quadrotor frame

Apparently, the declared size of the quadrotor on market represents the diagonal distance between the two motors on each arm of the frame which is equivalent to twice of l.

3.2.2 Mass and Moment Measurement

The moment of intertia of the quadrotor is obtained from bifilar pendulum experiment [21], where for each axis, the moment of inertia J, can be computed by measuring the period of twist oscillation with the experimental stand setup shown in figure 3.2

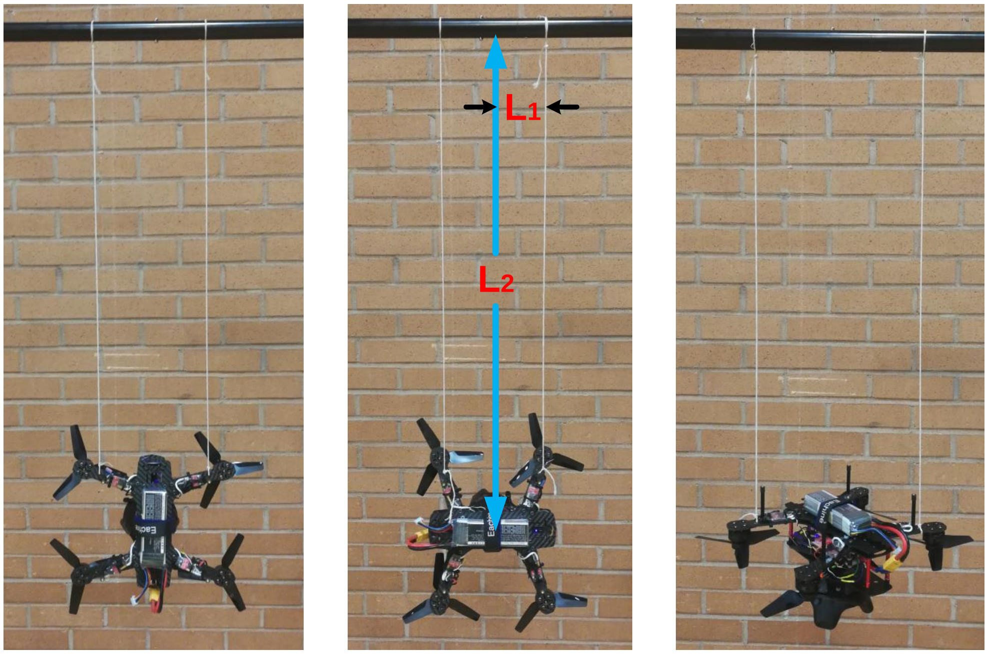


Figure 3.2: The bifilar pendulum experiment for three body axis

the equation used to compute the moment of inertia is

$$J = \frac{mgT^2L_1^2}{4\pi^2 L_2} \tag{3.1}$$

where T is the period of each oscillation. The free oscillation can be regard as simple harmonic motion because amplitude of the swing of the quadrotor rigid body is small. To improve the accuracy, the averaged period from 100 oscillations is obtained from the flight controller board by checking the plus-minus sign of the gyroscope output. And L_1, L_2 is indicated in figure. The result obtained from the bifilar pendulum experiment mass measurement is shown in the Table 3.1.

Parameters	**Definition**	**Nominal Values**
J_x	Moment of Inertia in x axis	0.0019005 $kg * m^2$
J_y	Moment of Inertia in y axis	0.0019536 $kg * m^2$
J_z	Moment of Inertia in z axis	0.0036894 $kg * m^2$
m	Mass	0.551 kg

Table 3.1: Moment of inertia experiment results

Obviously, the moment of inertia in x axis has very small quantitative difference with that in y axis. But that in z axis are much bigger than the others. This will be discussed in Chapter 4.

3.3 Propulsion System Test

Based on Chapter 2, we must know torque coefficient, thrust coefficient and constant of actuator dynamics in the first-order transfer function. With the help of Dynamometer Series 1580 in Figure 3.3 from RCbenchmark, we can get all these numbers easily. Also, the mapping between the desired motor rotation speed and PWM signal is also needed for programming the firmware of *MARK3* flight controller.

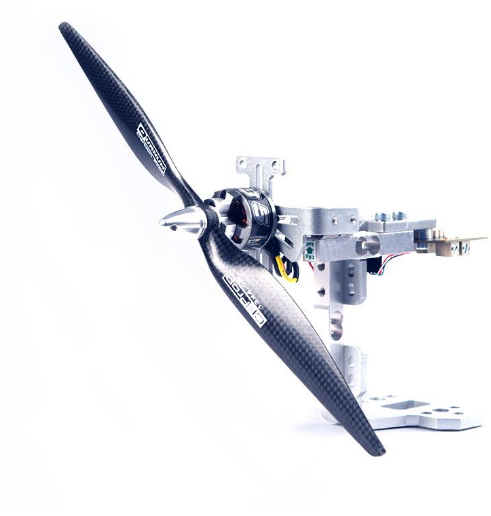
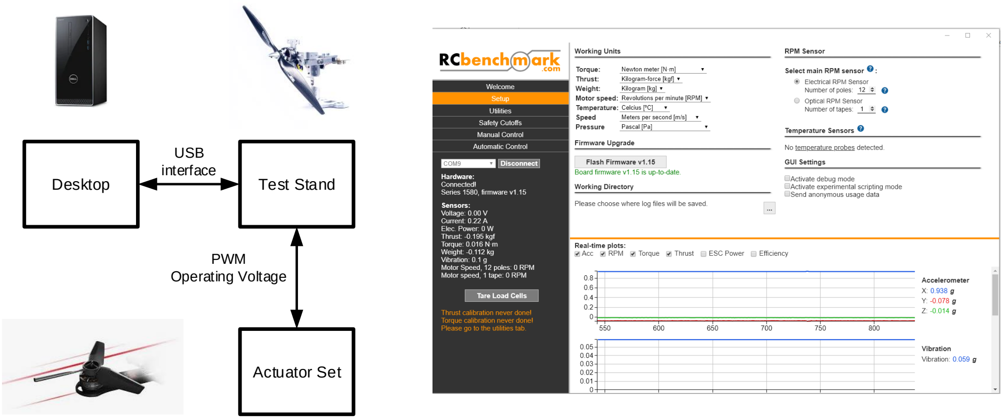


Figure 3.3: Dynamometer Series 1580

3.3.1 *Thrust Coefficient and Torque Coefficient Measurement*

The Dynamometer Series 1580 test stand is connected with an upper computer to transfer data and commands. By changing duty ratio of PWM on the Snail actuator Set, the test stand can adjust motor rotation speed automatically.



Hardware:
Connected!
Series 1580, firmware v1.15

Sensors:
Voltage: 0.00 V
Current: 0.22 A
Elec. Power: 0 W
Thrust: -0.195 kgf
Torque: 0.016 N·m
Weight: -0.112 kg
Vibration: 0.1 g
Motor Speed, 12 poles: 0 RPM
Motor speed, 1 tape: 0 RPM

Thrust calibration never done!
Torque calibration never done!
Please go to the utilities tab.

Working Units
Torque: Newton meter [N·m]
Thrust: Kilogram-force [kgf]
Weight: Kilogram [kg]
Motor speed: Revolutions per minute [RPM]
Temperature: Celcius [°C]
Speed: Meters per second [m/s]
Pressure: Pascal [Pa]

Firmware Upgrade
Board firmware v1.15 is up-to-date.

Working Directory
Please choose where log files will be saved.

RPM Sensor
Select main RPM sensor ?:
● Electrical RPM Sensor
Number of poles: 12
○ Optical RPM Sensor
Number of tapes: 1

Temperature Sensors ?
No temperature probes detected.

GUI Settings
☐ Activate debug mode
☐ Activate experimental scripting mode
☐ Send anonymous usage data

Real-time plots:
☑ Acc ☑ RPM ☑ Torque ☑ Thrust ☐ ESC Power ☐ Efficiency

Accelerometer
X: 0.938 *g*
Y: -0.078 *g*
Z: -0.014 *g*

Vibration
Vibration: 0.059 *g*

Figure 3.4: Actuator Test Equipment Structure and GUI

As shown in Figure 3.4, the structure of the whole test equipment and the software interface are presented. The test stand can identify real-time thrust and torque with three sets of piezoresistive pressure sensors. The rotation speed is read through the change of the voltage on one phase of the brushless motor. Then we can have the thrust coefficient b and the torque coefficient d.

From the curve fitting in Figure 3.5, we get $b = 1.91 \times 10^{-6}\ N\ s^2/rad^2$.

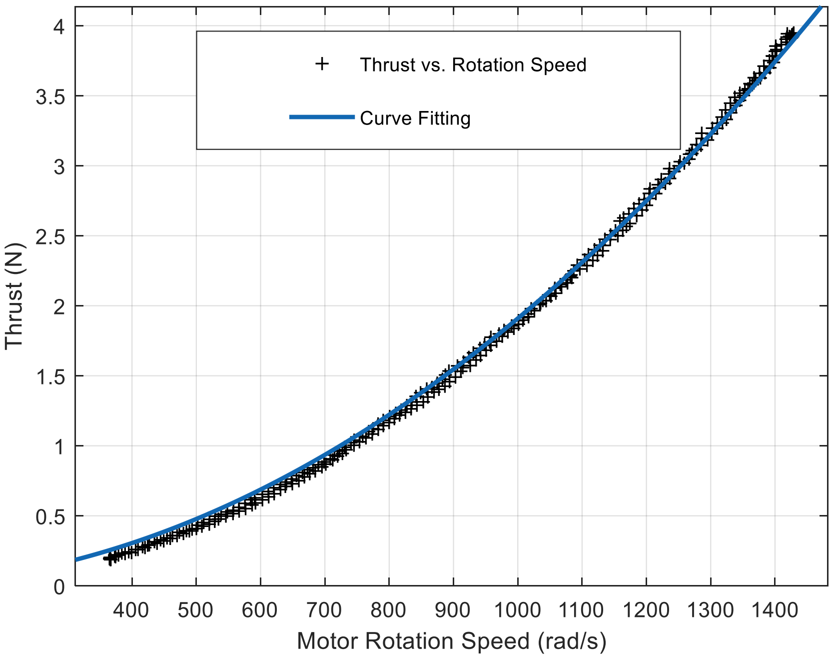


Figure 3.5: Thrust vs Motor Rotation Speed Fitting Curve

From the curve fitting in Figure 3.6, we get $d = 2.47 \times 10^{-8}\ Nm\ s^2/rad^2$.

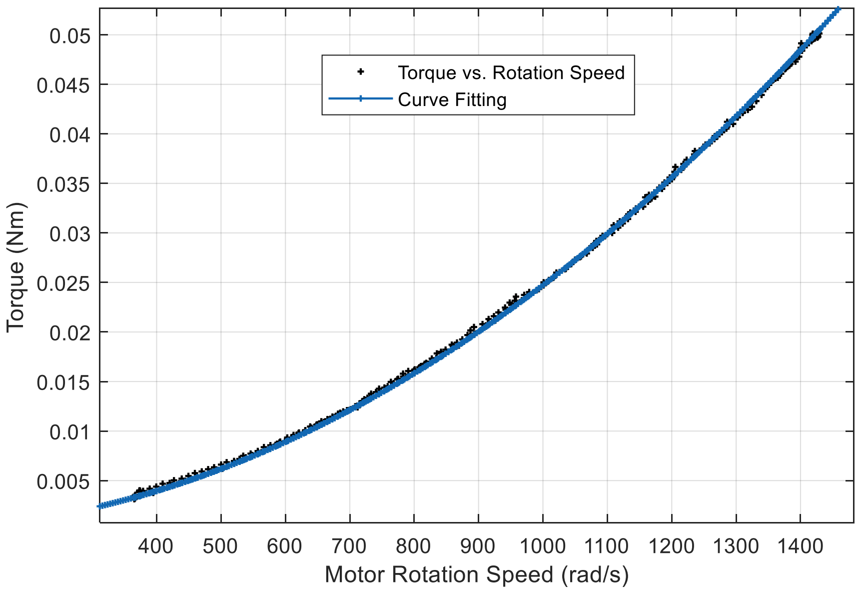


Figure 3.6: Torque vs Motor Rotation Speed Fitting Curve

Then we need to find the mapping between motor rotation speed and PWM signal. According to [16], since we use electronic speed controllers that control the percentage of input voltage, the resulting motor rotation speed for a given PWM command depends on the battery voltage. The voltage compensation is also needed for motor speed mapping. By performing motor speed and PWM mapping identifications with different voltaged wich are controlled by a power supply unit, we can find a linear function to represent the 3-D curve fitting.

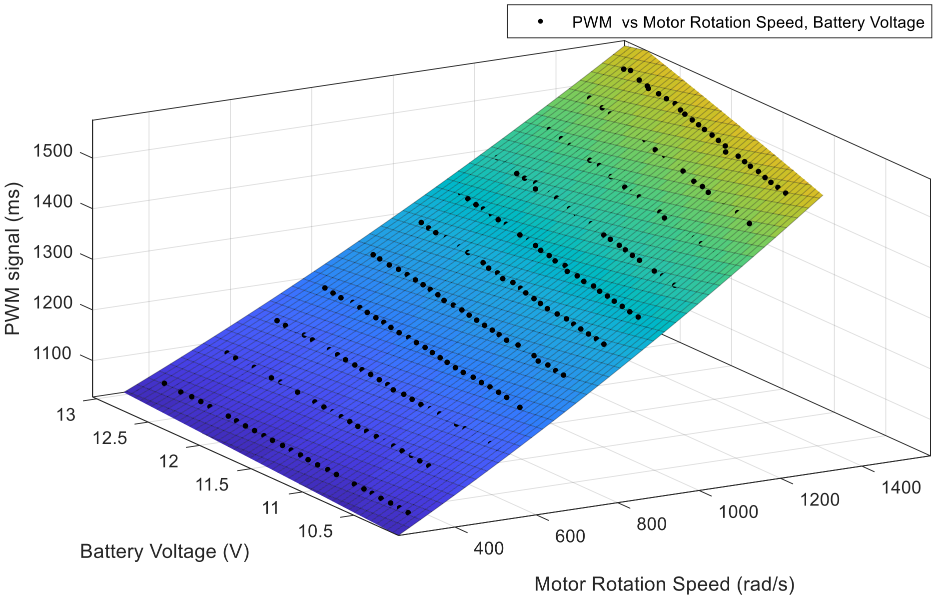


Figure 3.7: PWM vs Motor Rotation Speed vs Battery Voltage

The mapping is presented in Figure 3.7 with the equation (ϵ is PWM signal, ω is motor rotation speed and u_{bat} is battery voltage):

$$\epsilon = \frac{\omega^2 + a_1\omega + a_2}{a_3 u_{bat} + a_4} + a_5 \tag{3.2}$$

From the curve fitting, we have $a_1 = 5393$, $a_2 = 29960$, $a_3 = 1166$, $a_4 = 1544$ and $a_5 = 895$. This PWM-speed mapping with battery compensation ensures the PWM commands are calculated for motors when given the output of controllers in angular rate loop and vertical velocity loop.

To identify the pole of first-order linearized actuator dynamics, step response experiment is performed. The open-loop pole a is estimated using MATLAB Ident toolbox.

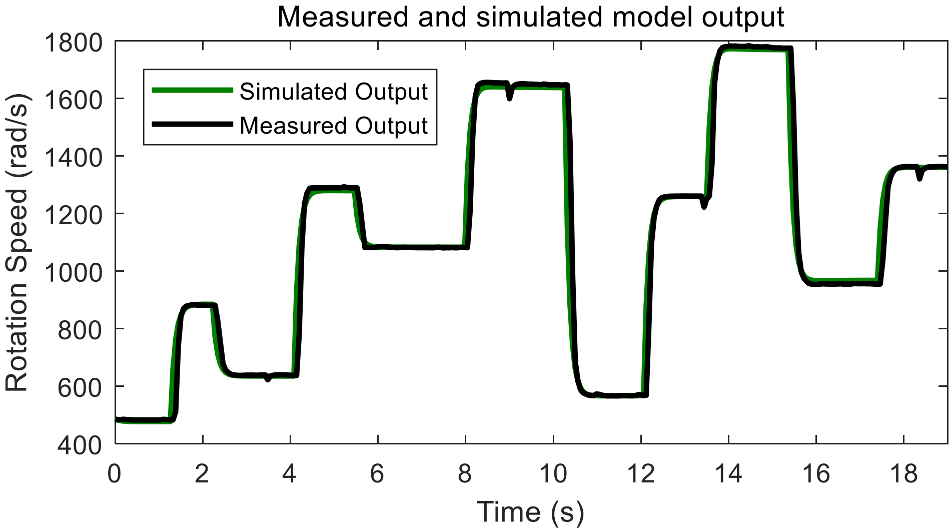


Figure 3.8: Actuator Pole Identification Curve

The identification result is shown in Figure 3.8 when $a = 9.79$ (the pole from the actuator dynamics mentioned in Chapter 2) with 92% fitness. This actuator model based system identification accurately reflects actual characteristic.

3.4 Design of the Flight Controller

The flight controller consists of a Microcontroller Unit, an MEMS sensor board, a Communication Module, and external IOs. The flight controller is responsible for executing communication protocol, state estimation, control and ESC command inputs. To simplify the software design, a microprocessor was needed with sufficient computational power for heavy load of floats computation.

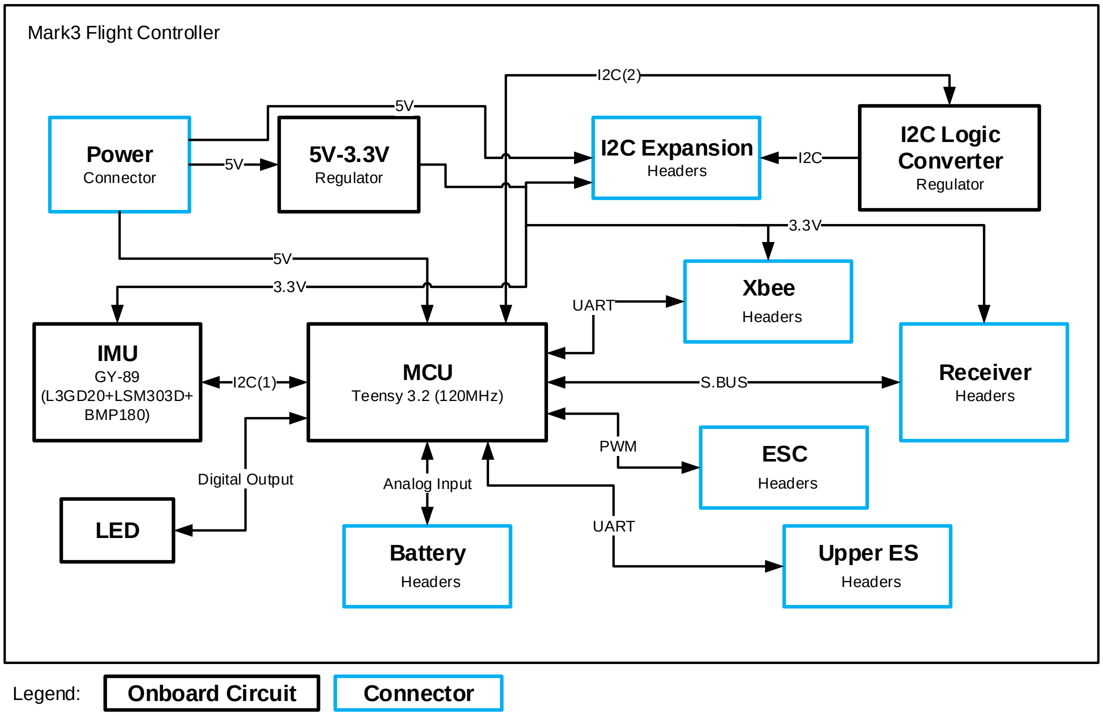


Figure 3.9: Block Diagram of Mark3 Flight Controller

The design of the flight controller is shown in Figure 3.9. The Cortex-M4 MK20DX256 32 bit which can be overclocked to 168MHz offers large flash space, large RAM space, USB interface, low per unit cost and hardware simplicity. The GY-89 10DOF Sensor Board gives enough design redundancy of onboard sensing. The I2C port gives up to 1000Hz data transfer rate to ensure the rate of inner-loop control (angular rate). To ensure the normal power supply of all the on-board modules and peripheral devices, a 5v-3.3v regulator and an I2C logic converter are added. Additional headers are also reserved on this flight controller for more peripheral devices (High-level Embedded System, Other Sensor Boards, etc) in the future. The Schematic Print and the Composite Drawing of the *MARK3* flight controller are presented in Appendix B.

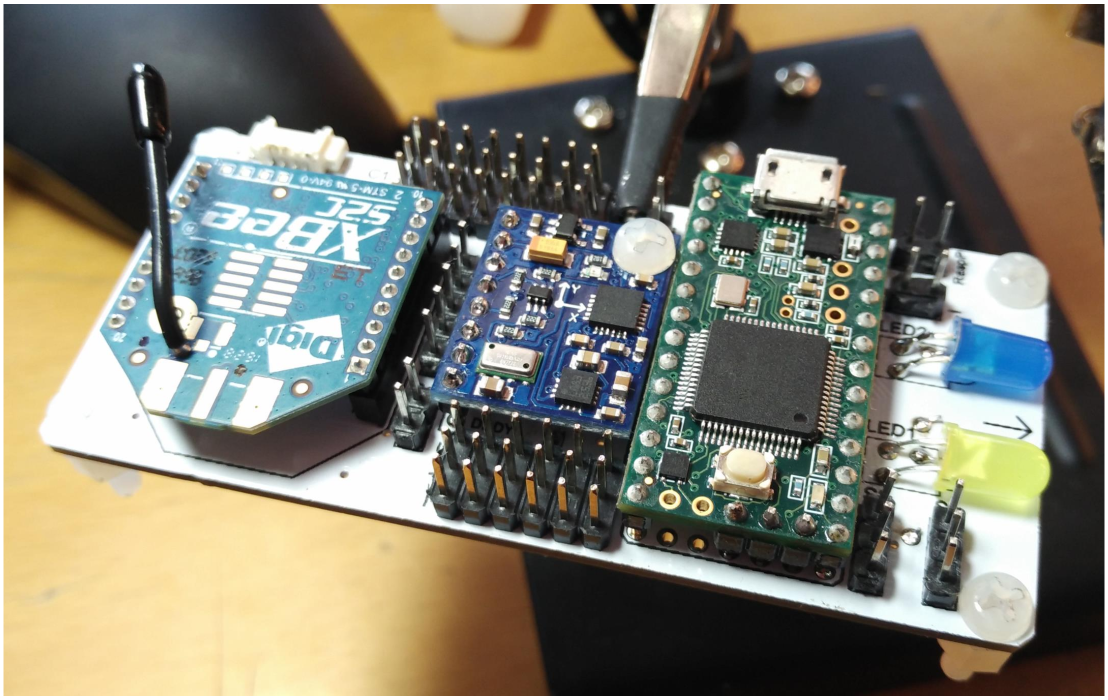


Figure 3.10: Mark3 Flight Controller Photo

The assembled design photo is shown in Figure 3.10

3.5 General Frame Structure of Hardware & Software

The Quadrotor Platform Electrical Architecture. The block diagram of the quadrotor platform electrical architecture is presented in Figure 3.11. Total 4 Electronic Speed Controllers are connected with the pin headers on the *MARK3* flight controller to receive the PWM command from the flight controller. The flight controller is powered by the 5v source on the Matex Power Distribution Board. The Distribution Board also powers all four ESCs with the voltage from the 3S Lipo Battery of which the safe voltage ranges from $10.2v$ to $12.6v$. The flight controller reads battery voltage with a certain header while the $10.2v \sim 12.6v$ battery voltage is being converted to $2.04 \sim 2.52v$ which is suitable for an analog pin of Teensy 3.2

MCU.

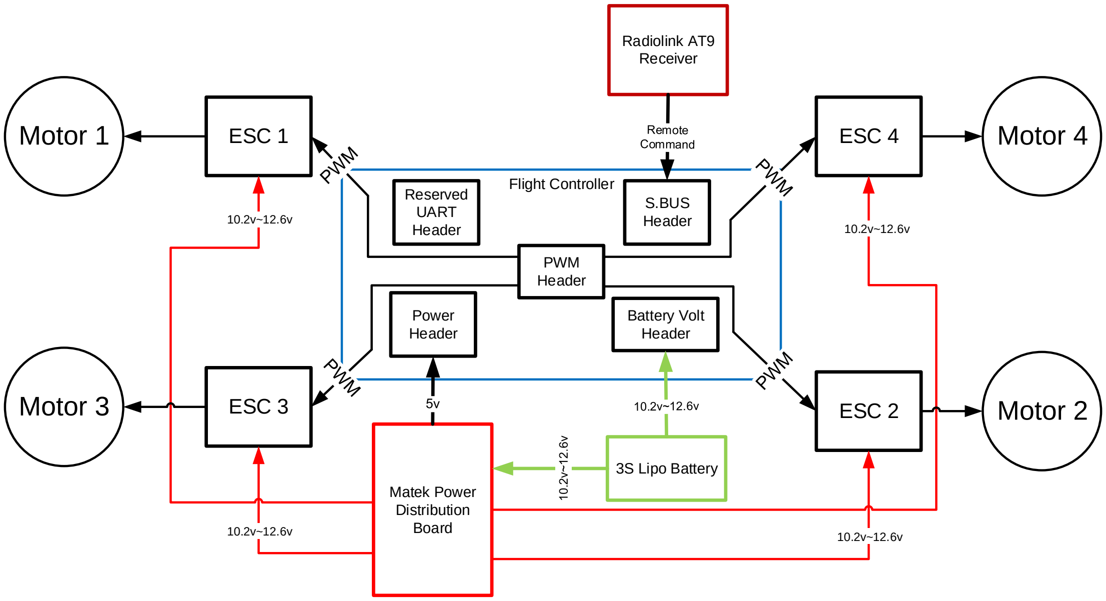


Figure 3.11: The Quadrotor Platform Electrical Architecture

The remote receiver is connected via S.BUS Header to receive and send receiver (RX) and transmitter (TX) protocols. A reserved UART header is for connection between the high-level embedded system which will be used for research in the future.

The Flight Controller Softeware Architecture. The block diagram of the flight controller software architecture is presented in Figure 3.12. The software was implemented in two separate threads. The on-board sensor data collection, angular rate control, ESC output and loop checking are updated at 400 Hz. Communication process, state estimator, position & velocity control and attitude control are updated at 100 Hz where they are distributed in 4 loops of 400-Hz software main loop.

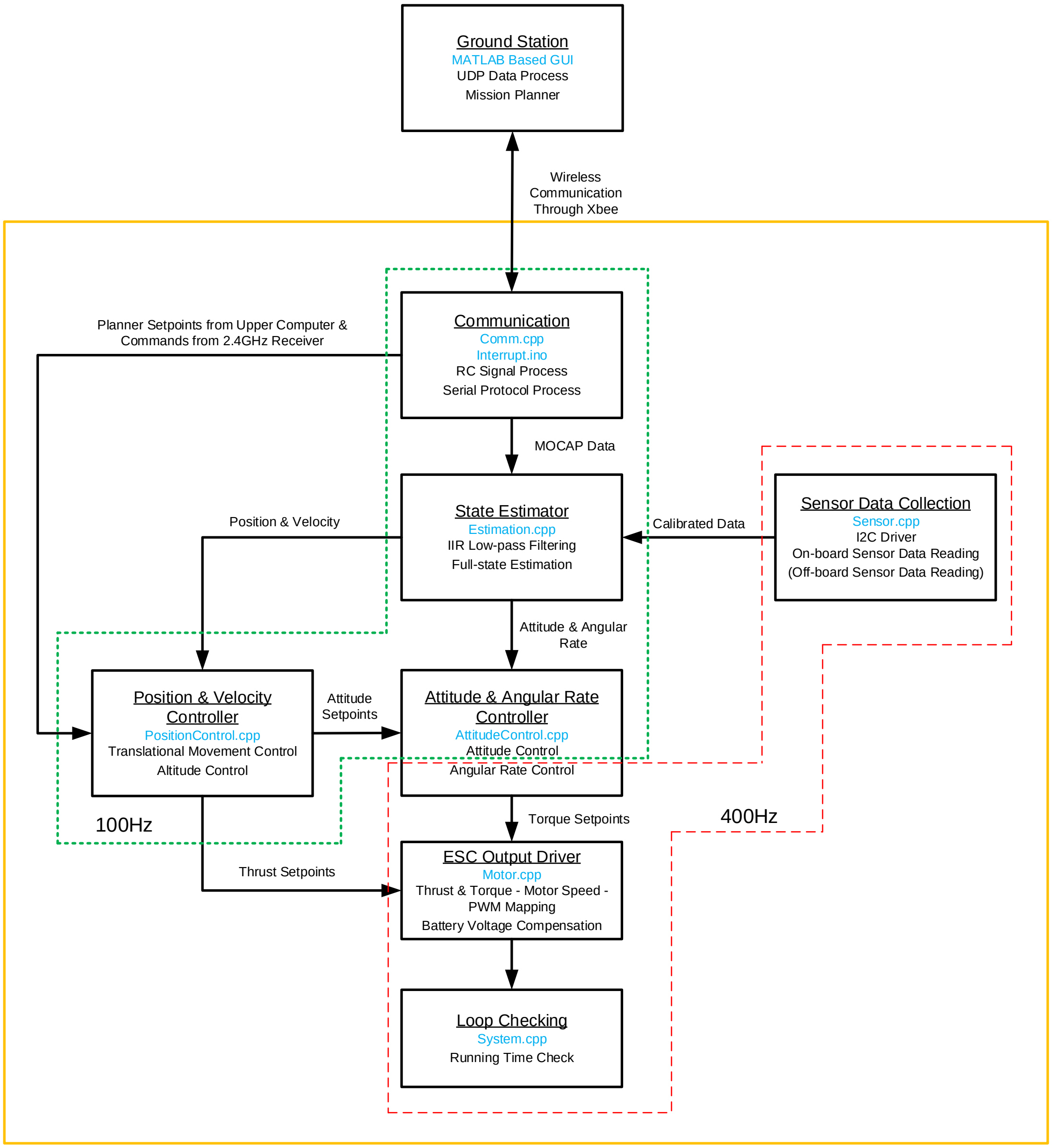


Figure 3.12: The *MARK3* Flight Controller Software Architecture

The Experimental System Hardware Architecture. The block diagram of the experimental system hardware architecture is presented in Figure 3.12. The motion capture system consists of HTC VIVE Tracking System (Two Lighthouse 1.0 Basestaions and some HTC VIVE Trackers) and a high-performance desktop equipped with a CPU not lower than Intel i5-4590 and a GPU not lower than Nvidia GTX970.

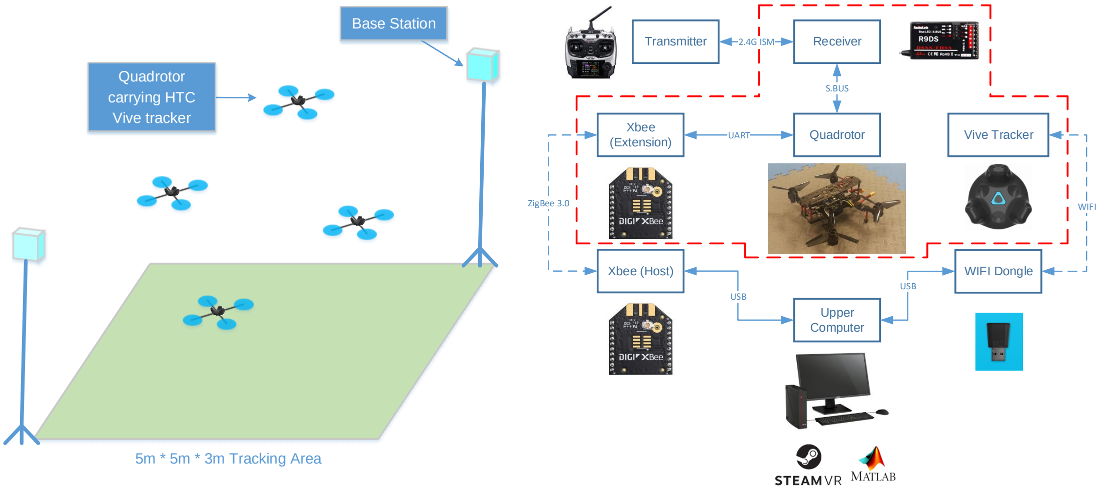


Figure 3.13: The Experimental System Hardware Architecture

The HTC VIVE tracker transmitt full-state data via a WIFI Dongle. Here a GUI Programmed in MATLAB gets data from SteamVR API and send the serial data packet to the host Xbee 3.0 module. Then the extension Xbee 3.0 module on the *MARK3* will receive the serial packet. The transmitter is standby during flight for manual control in case that accident happens (communication loss with the upper computer, collision, etc).

Chapter 4

ANALYSIS OF LINEAR MODEL

4.1 Angular Movement Analysis

Based on Chapter 2, we get the linearized model of the whole quadrotor model including actuator dynamics. Given above, the associated transfer function matrix is given by

$$P = C(sI - A)^{-1}B + D \tag{4.1}$$

Then

$$P_{rate} = \begin{bmatrix} \frac{p}{\tau_\phi^*} & 0 & 0 \\ 0 & \frac{q}{\tau_\theta^*} & 0 \\ 0 & 0 & \frac{r}{\tau_\psi^*} \end{bmatrix} = \begin{bmatrix} \frac{a}{J_x s(s+a)} & 0 & 0 \\ 0 & \frac{a}{J_y s(s+a)} & 0 \\ 0 & 0 & \frac{a}{J_z s(s+a)} \end{bmatrix} \tag{4.2}$$

Clearly the angular rate linearized model is decoupled. We can also get the block diagram of the angular rate model in Figure 4.1

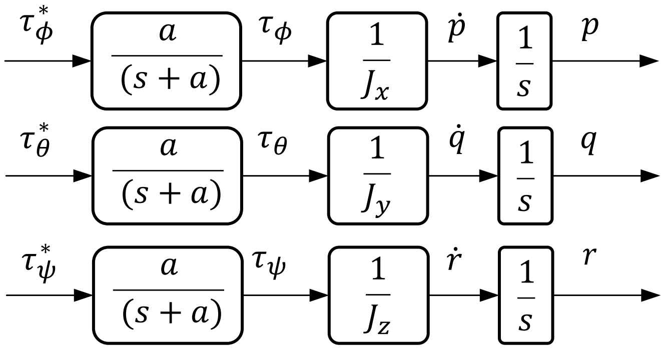


Figure 4.1: Angular Rate Plant Diagram

From the measured parameter values in Chapter 3, we obtain the transfer functions with numerical values below. For these nominal parameter values, we obtian the following numeriacal SISO transfer functions including:

Roll Rate

$$P_p = \frac{p}{\tau_\phi^*} = \frac{5151.28}{s(s+9.79)} \tag{4.3}$$

Pitch Rate

$$P_q = \frac{q}{\tau_\theta^*} = \frac{5011.26}{s(s+9.79)} \tag{4.4}$$

Yaw Rate

$$P_r = \frac{r}{\tau_\psi^*} = \frac{2653.55}{s(s+9.79)} \tag{4.5}$$

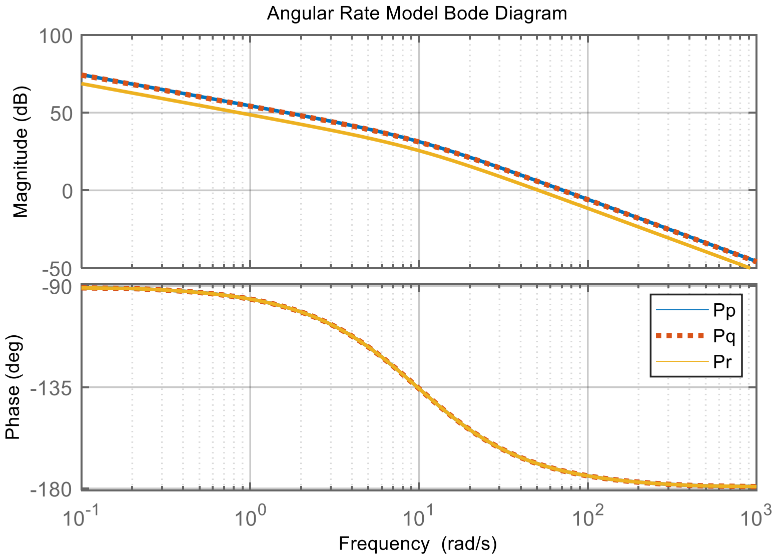


Figure 4.2: Angular Rate Movement Bode Plot

Firstly bode frequency response plot for the angular rate plant is presented in Figure 4.2. Obviously dynamics of pitch angular movement and roll angular movement resemble. To generate same angular acceleration, yaw angular movement requires more torque than that of pitch angular movement and roll angular movement because the moment inertia in x and y body axis is smaller than that in z body axis.

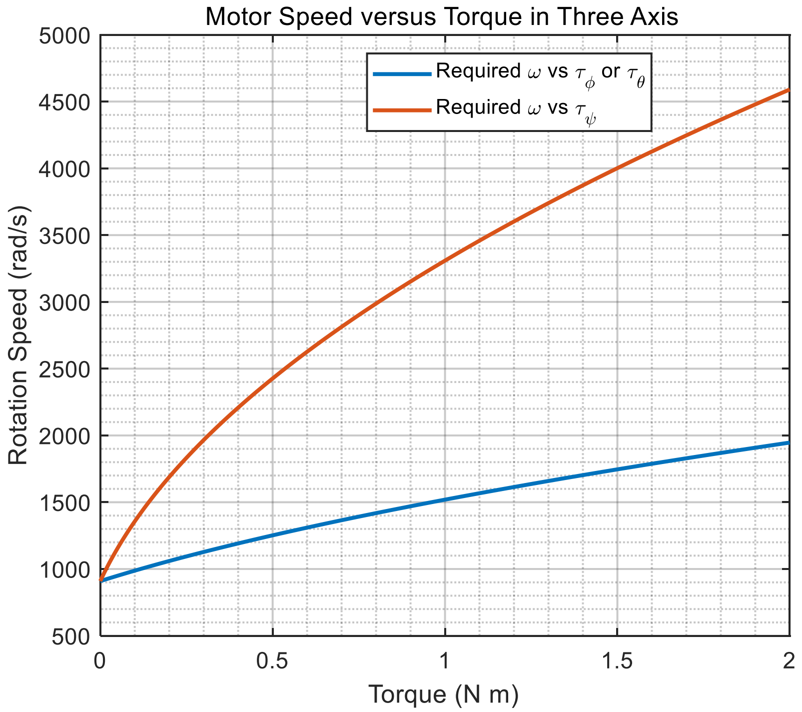


Figure 4.3: Motor Speed vs Torque in Body Frame

Secondly in Figure 4.3, to get same value of body torque from all 4 motors when the quadrotor is near hovering mode ($T = mg$), torque for yaw angular movement requires much more maximum motor speed than that for pitch angular movement and roll angular movement. The reason is that the actuator torque coefficient ($d = 2.47 \times 10^{-8}\ Nm\ s^2/rad^2$) is much smaller than the combined actuator thrust factor ($\frac{\sqrt{2}}{2}bl = 1.69 \times 10^{-7}\ Nm\ s^2/rad^2$).

Based on the above two points, we know that the designed bandwidth for yaw angular movement should be smaller than that for pitch angular movement and roll angular movement.

With an integral item, we can have attitude presented in the whole angular movement block diagram of the linearized model in Figure 4.4.

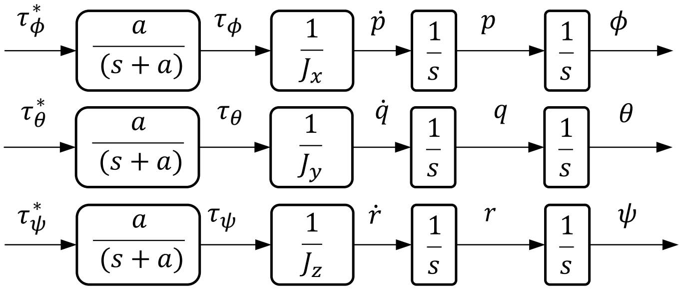


Figure 4.4: Angular Movement Plant Diagram

The control design of angular movement is presented in Chapter 4 with cascade structure. This model is applied in the flight controller firmware when quadrotor flying near hovering mode (ϕ and θ are not big).

4.2 Vertical Movement Analysis

The structure of vertical movement (vertical velocity and altitude) model is same as that of attitude movement shown in Figure 4.5

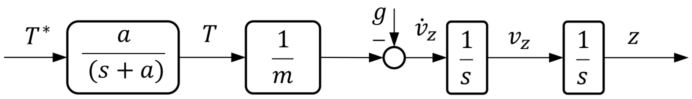


Figure 4.5: Vertical Movement Plant Diagram

The linearized model is designed for simulation. Even though the quadrotor is near hovering mode, we still need to add nonlinear constraint to the vertical velocity controller output in the flight controller firmware [22] due to the nonlinearity in the real world:

$$T^*_{act} = \frac{T^*}{cos(\phi)cos(\theta)} \tag{4.6}$$

where T_{act} is the real input value of motor driver in the firmware.

4.3 Translational Movement Analysis

The structure of translational movement (translational velocity and position) model is presented in Figure 5.35

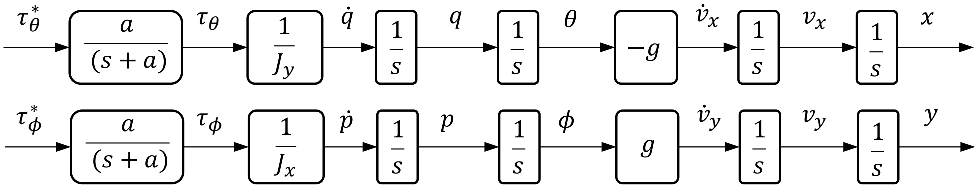


Figure 4.6: Translational Movement Plant Diagram

Similarly the linearized model is designed for simulation. We need to add nonlinear constraint to the translational velocity controller output in the flight controller firmware due to the nonlinearity in the real world. We have:

$$a = cos(\psi) \tag{4.7}$$

$$b = sin(\psi) \tag{4.8}$$

$$c_1 = mT\theta^* \tag{4.9}$$

$$c_2 = mT\phi^* \tag{4.10}$$

And finally we have:

$$\phi^*_{act} = asin(ac_2 - bc_1) \tag{4.11}$$

and

$$\theta^*_{act} = asin(-(ac_1 + bc_2)/cos(\theta^*)) \tag{4.12}$$

where θ^*_{act} and ϕ^*_{act} is the real input value of attitude controller in the firmware.

4.4 Summary and Conclusion

Based the model analysis above, attitude control and vertical movement control will have the similar structure. But the maximum available bandwidth is limited by the actuator output in different channels. In the next chapter, the full-state control design is illustrated based on this model analysis.

Chapter 5

CONTROL DESIGN METHODS AND SIMULATIONS

5.1 Introduction and Overview

In this chapter, we describe how to design controller with full-state feedback for this quadrotor platform equipped with the *MARK3* flight controller. The attitude/angular-rate control has been designed with control parameter trader off. The altitude/vertical-velocity and translational position/velocity are presented and analyzed. The underlying theory for each controller is explained and justified.

5.2 Angular Movement Cascade Control

5.2.1 Angular Rate Control

In this section, we describe (p, q, r) angular rate control for this quadrotor platform equiped with the *MARK3* flight controller. The angular velocity of the quadrotor is from gyroscope. The frequency of angular rate control in the firmware is $400Hz$.

Control Design: PD Controller. We focus on roll angular movement first. The block diagram of close loop angular rate control is shown in Figure 5.1.

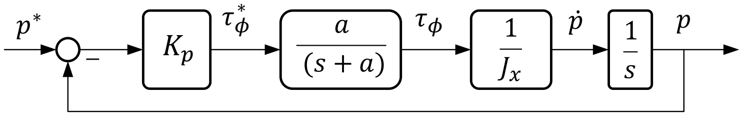


Figure 5.1: Close Loop Diagram of Angular Rate Control

Based on the model obtained from the previous section

$$P_p = \frac{p}{\tau_\phi^*} = \frac{5151.28}{s(s+9.79)} \tag{5.1}$$

we design a PD controller

$$K_p = g(s+z) \tag{5.2}$$

This K_p will be used to generate input of motor driver in the firmware of the *MARK3* flight controller. This PD controller will place the dominant closed loop pole near

$$s = 9.79 \tag{5.3}$$

The open loop transfer function L_p is given by

$$L_p = P_p K_p = g(s+z)\frac{5151.28}{s(s+9.79)} \tag{5.4}$$

In this case $k_p = g$ and $k_d = gz$.

Open Loop L Analysis. Figure 5.2 and Figure 5.3 show the plots of $L_p = P_p K_p$ for specific (g, z) variations.

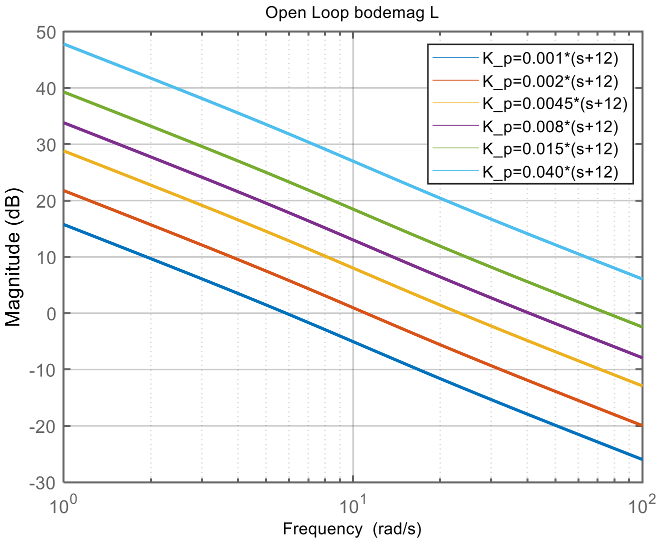


Figure 5.2: Open Loop Bode Plot When changing g

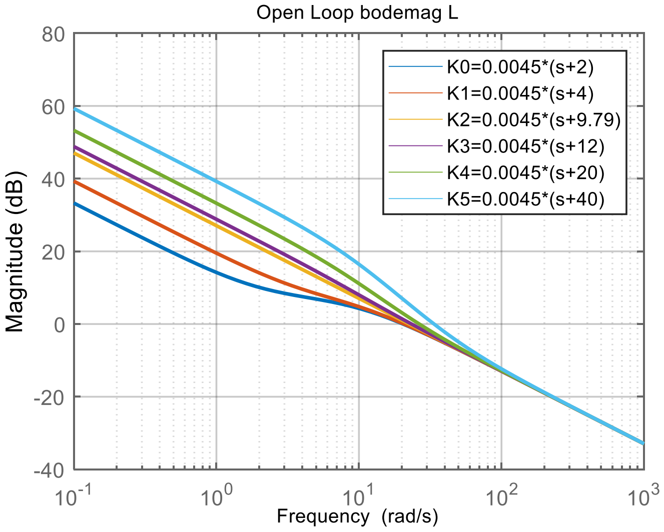


Figure 5.3: Open Loop Bode Plot When changing z

Based on the Figures above:

- increasing g increases magnitude of L and increasing z increase magnitude of L at low frequencies
- increasing g impacts the crossover proportionately and increasing z doesnn't impact the crossover much
- increasing g doesn't impact phase of L and increasing z impacts phase of L clearly at the frequencies near z

Sensitivity ($T_{d_o y}$). The sensitivity bode plot is presented in Figure 5.4 and Figure 5.5

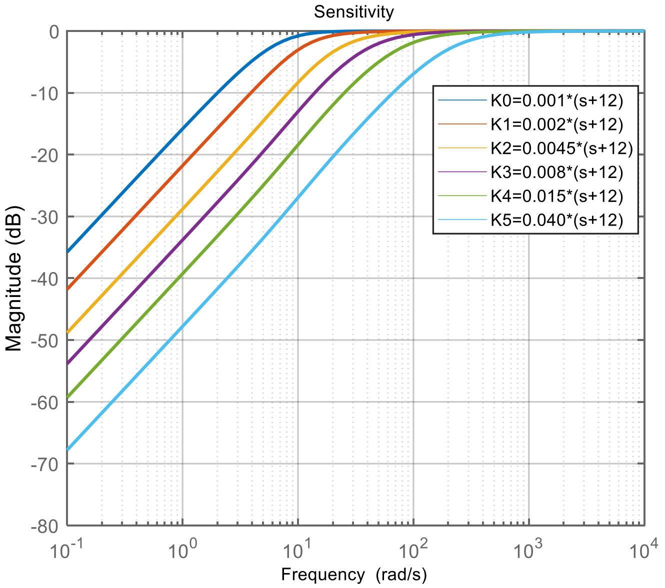


Figure 5.4: Sensitivity Bode Plot When changing g

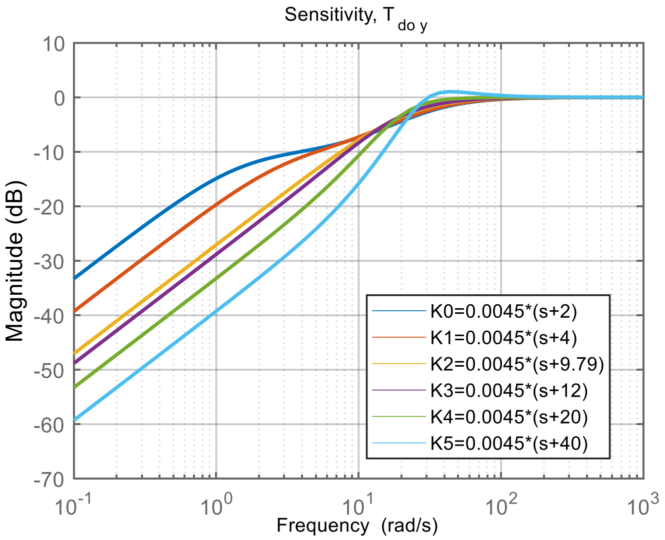


Figure 5.5: Sensitivity Bode Plot When changing z

Based on the Figures above:

- increasing g results smaller magnitude of S at low frequencies
- increasing z results smaller magnitude of S at low frequencies but increases peak sensitivity
- peak sensitivities do not bring much change with increasing z

Complementary Sensitivity (T_{ry}). The complimentary sensitivity bode plot is presented in Figure 5.6 and Figure 5.7

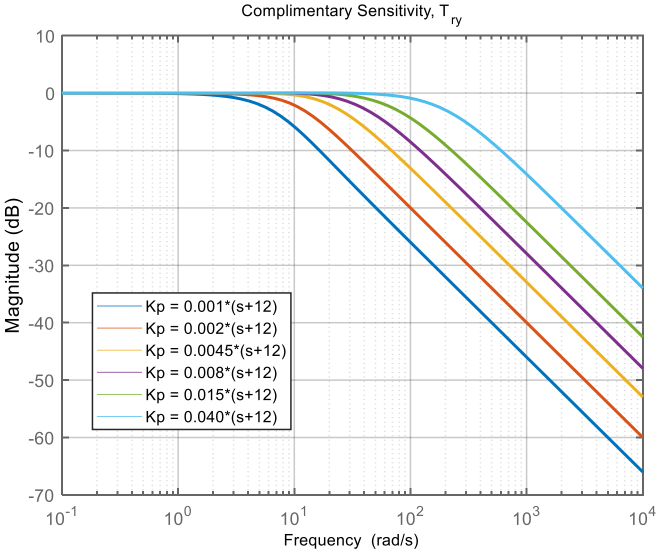


Figure 5.6: Complementary Sensitivity Bode Plot When changing g

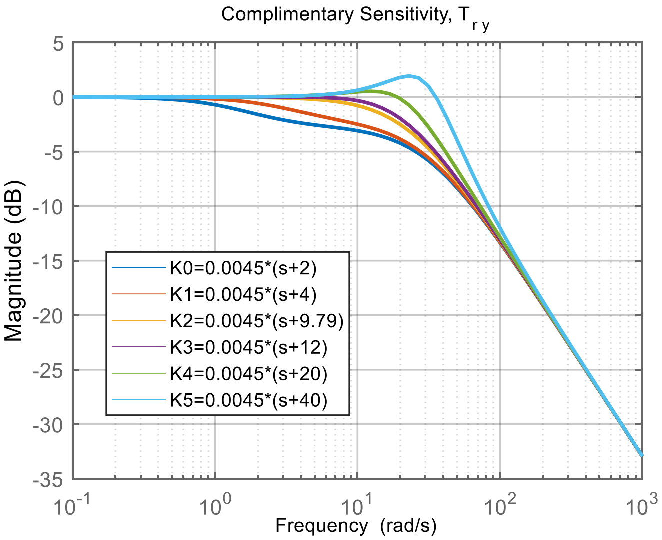


Figure 5.7: Complementary Sensitivity Bode Plot When changing z

Based on the Figures above:

- increasing g results larger bandwidth
- increasing z results larger peak complementary sensitivity

The z value in roll, pitch and yaw angular rate controller is same. As discussed in Chapter 4, the g value for yaw angular rate should be smaller than that of roll and pitch angular movement because of more needed motor speed to generate enough torque for yaw movement.

Input Disturbance to Output T_{d_iy} (PS). The input disturbance to output bode plot is presented in Figure 5.8 and Figure 5.9

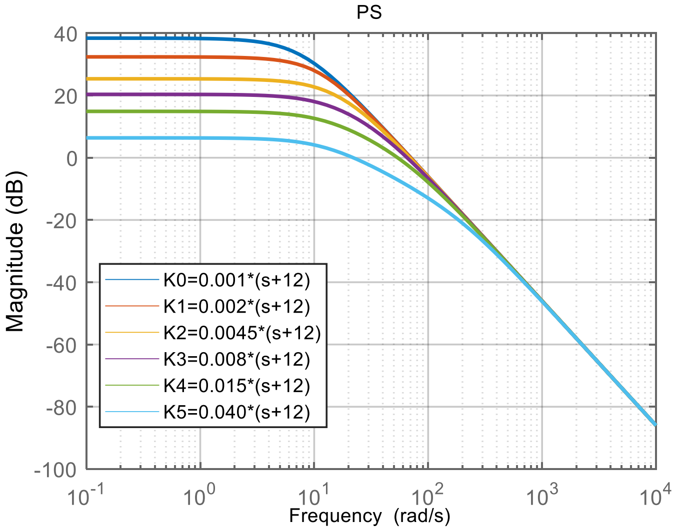


Figure 5.8: Input Disturbance to Output Bode Plot When changing g

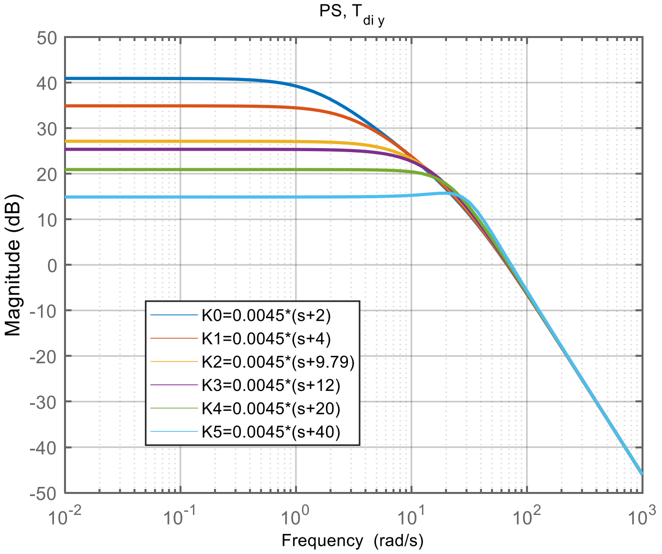


Figure 5.9: Input Disturbance to Output Bode Plot When changing z

Based on the Figures above:

- increasing g reduces magnitude at low frequencies
- increasing z reduces magnitude at low frequencies with a little peak

Reference to Control T_{ru}. The reference to control bode plot is presented in Figure 5.10 and Figure 5.11

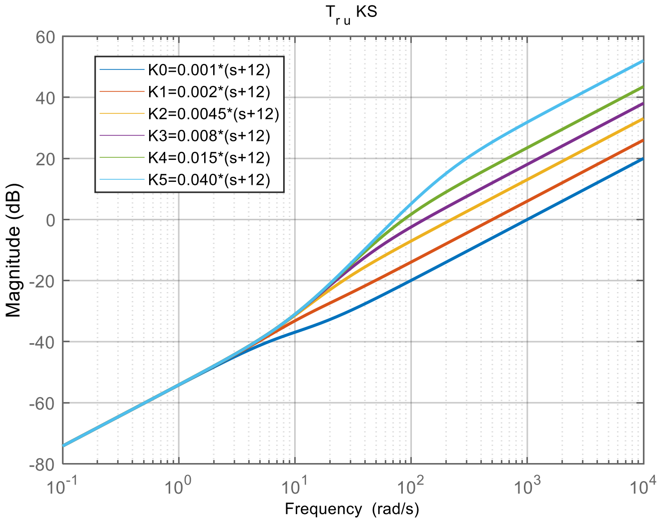


Figure 5.10: Reference to Control Bode Plot When changing g

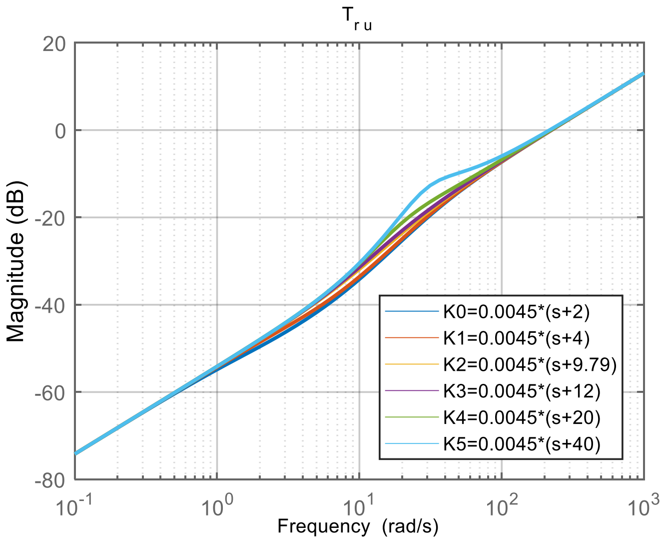


Figure 5.11: Reference to Control Bode Plot When changing z

Based on the Figures above:

- increasing g increases magnitude at high frequencies
- increasing z results larger peak in T_{ru}

Sensor Noise to Output T_{ny} **(**T_{d_iu}**).** The sensor noise to output bode plot is presented in Figure 5.12 and Figure 5.13

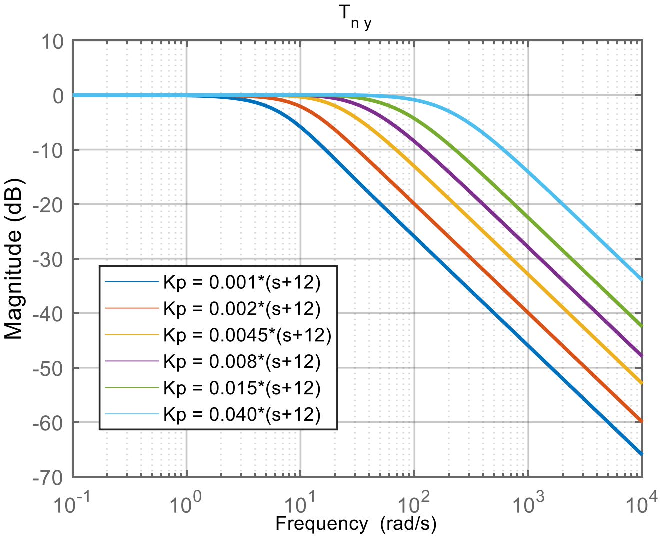


Figure 5.12: Sensor Noise to Output Bode Plot When changing g

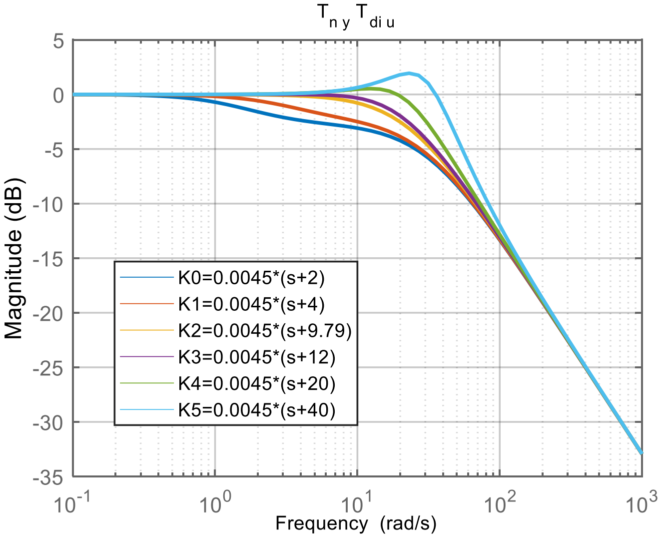


Figure 5.13: Sensor Noise to Output Bode Plot When changing z

Based on the Figures above:

- increasing g increases magnitude at high frequencies
- increasing z results larger peak in T_{ny}

Sensor noise is the main issue limits the angular movement bandwidth. Increasing g brings not only higher bandwidth but also much more noise at higher frequencies. The value $g = 0.0045$ is obtained from actual experiment with the quadrotor fixed on the test stand based on trade-off of complimentary sensitivity and sensor noise to output.

Output Disturbance to Control $T_{d_o u}$. The output disturbance to control bode plot is presented in Figure 5.14 and Figure 5.15

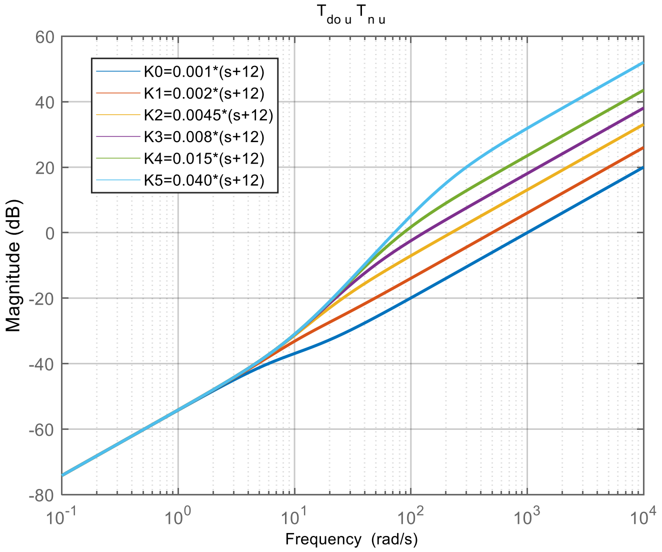


Figure 5.14: Output Disturbance to Control Bode Plot When changing g

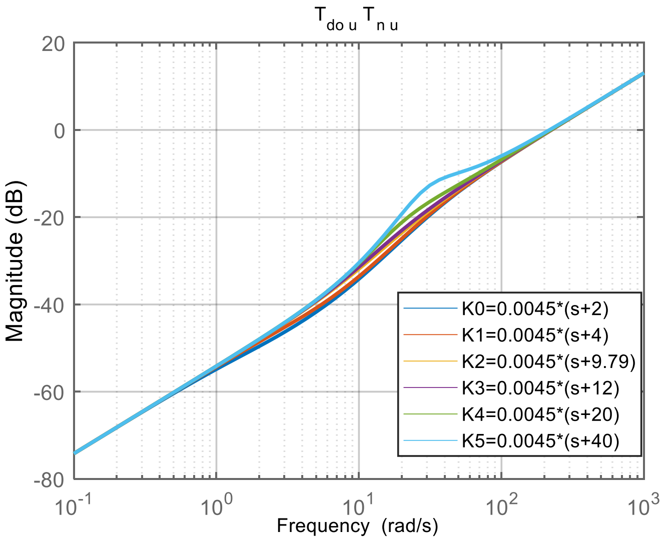


Figure 5.15: Output Disturbance to Control Bode Plot When changing z

Based on the Figures above:

- increasing g increases magnitude at high frequencies in $T_{d_o u}$
- increasing z results larger peak in $T_{d_o u}$

Step Response. The step response is presented in Figure 5.16 and Figure 5.17

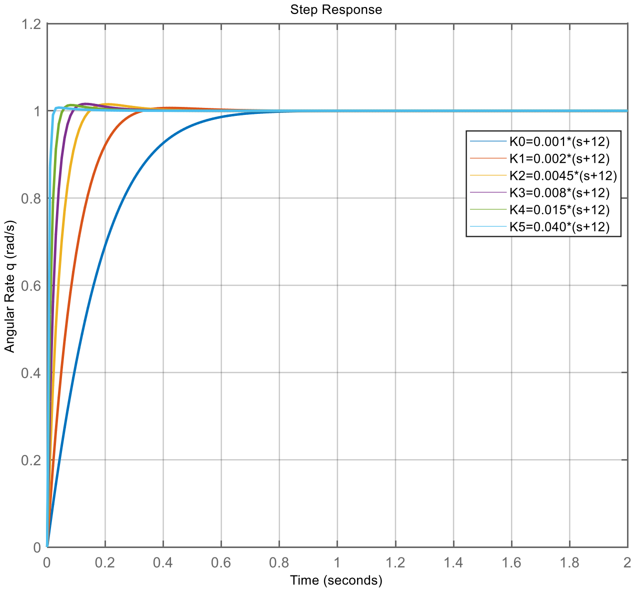


Figure 5.16: Step Response When changing g

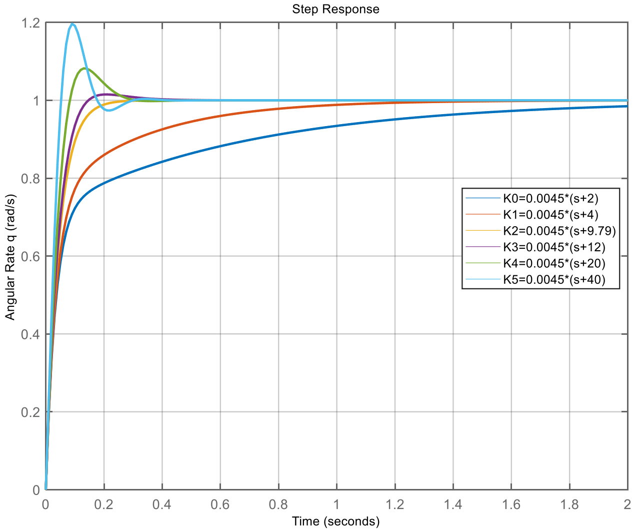


Figure 5.17: Step Response When changing z

Based on the Figures above:

- increasing g reduces settling time
- increasing z reduces rising time but brings more overshoot when z is bigger than the actuator constant $a = 9.79$

Roll-off Term. To reduce effect of high frequency noise in the feedback loop, roll-off term is needed to added with cut-off frequency at 10 times of control bandwidth. The roll-off term is

$$K_{rolloff1} = \frac{220^2}{(s+220)^2} \tag{5.5}$$

The angular rate controller provides good tracking and disturbance rejection. This design corresponds to a LQR controller considering the dynamics of the angular rates and torque in three-axis [4]. Considering a subsystem containing the body rates and body torques as state, it leads to the system like this ($\mathbf{J} = diag[1/J_x, 1/J_y, 1/J_z]^T$, $\nu = [p, q, r]^T$ as angular rate, $\tau = [\tau_\phi, \tau_\theta, \tau_\psi]^T$ as torque in three-axis coordinate system and $\tau^* = [\tau^*_\phi, \tau^*_\theta, \tau^*_\psi]^T$ as desired torque)

$$R_\psi = \begin{bmatrix} \dot{\nu} \\ \dot{\tau} \end{bmatrix} = \begin{bmatrix} \mathbf{0} & \mathbf{J}^{-1} \\ \mathbf{0} & -a\mathbf{I}_3 \end{bmatrix} \begin{bmatrix} \nu \\ \tau \end{bmatrix} + \begin{bmatrix} \mathbf{0} \\ a\mathbf{I}_3 \end{bmatrix} \tau^* \tag{5.6}$$

where an infinite-horizon LQR control law $\mathbf{u} = -\mathbf{K}_{lqr}\mathbf{s}$ that minimizes the cost function

$$\int \mathbf{s}^T\mathbf{Q}\mathbf{s} + \mathbf{u}^T\mathbf{R}\mathbf{u} \tag{5.7}$$

Where $\mathbf{Q}$ is a diagonal weight matrix and $\mathbf{R}$ is the identity matrix. The solution to the formulated LQR problem is a gain matrix of the form (presented with (g, z) in the PD controller)

$$\mathbf{K}_{lqr} = \begin{bmatrix} g_p z_p & 0 & 0 & J_x g_p & 0 & 0 \\ 0 & g_q z_q & 0 & 0 & J_y g_q & 0 \\ 0 & 0 & g_r z_r & 0 & 0 & J_z g_r \end{bmatrix} \tag{5.8}$$

Finally, we choose $K_p = K_q = 0.0045(s + 12)$ as the p and q angular rate controller given that the closed loop poles are at $[(-16.17, 3.02), (-16.17, -3.02)]$ and the only zero is at $(-12, 0)$. The step response settling time is $0.12s$ (1.2% overshoot).

Based on the modeling analysis in Chapter 4, the bandwidth of yaw movement control should be much smaller than that for pitch and roll movement. Here I choose $K_r = 0.0012(s + 12)$ as the r angular rate controller. The open loop root locus for angular rate control is presented in Figure 5.18:

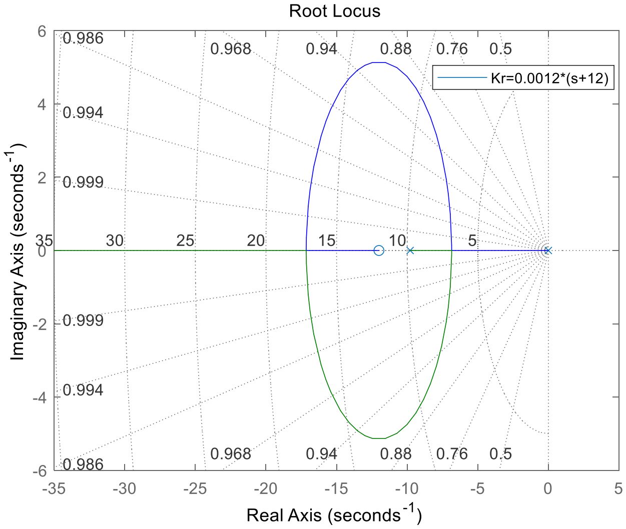


Figure 5.18: Angular Rate Control Open Loop Root Locus

The closed loop poles of yaw angular rate control are at $[(-8.45, 0), (-4.52, 0)]$ and the only zero is at $(-12, 0)$. The step response settling time is $0.93s$ (0% overshoot).

5.2.2 Attitude Cascade Control

In this section, we describle (ϕ, θ, ψ) attitude control for this quadrotor platform equiped with the *MARK3* flight controller. The attitude of the quadrotor is from attitude state estimation (fusion of gyroscope, accelerometer and MOCAP). The frequency of attitude control in the firmware is $100Hz$.

Control Design: Proportional Controller. Pitch and roll control of quadrotor is almost same. The block diagram of close loop angular rate control is shown in Figure 5.19.

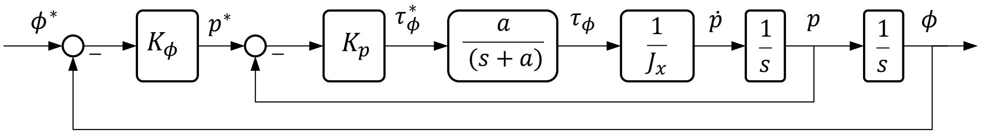


Figure 5.19: Close Loop Diagram of Attitude Control

Based on the close loop structure obtained from the previous section

$$T_{p^*\phi} = \frac{\phi}{p^*} = \frac{22.55s + 276}{s^3 + 32.34s^2 + 270.6s} \tag{5.9}$$

we design a P controller

$$K_\phi = k_p \tag{5.10}$$

This k_p will be used to generate input of angular rate control in the firmware of the *MARK3* flight controller. The open loop transfer function L_p is given by

$$L_\phi = P_\phi K_\phi = K_\phi \frac{22.55s + 276}{s^3 + 32.34s^2 + 270.6s} \tag{5.11}$$

Then we have the open loop root locus of $L = PK$ in Figure 5.20

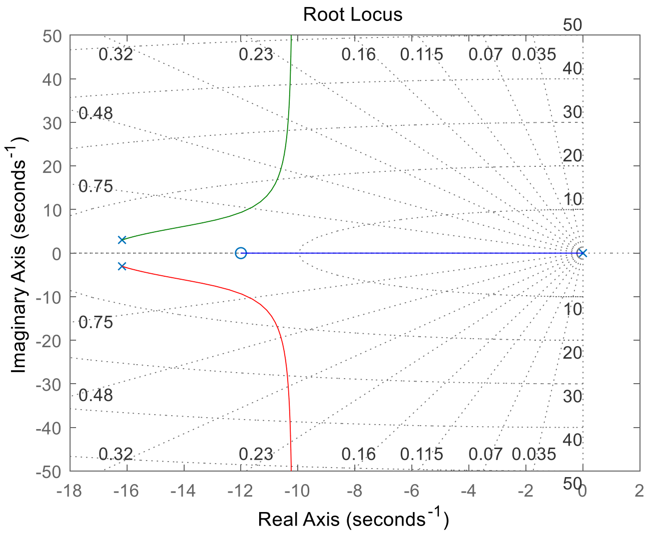


Figure 5.20: Open Loop Root Locus of Attitude Control

The complimentary sensitivity of attitude control is presented in Figure 5.21

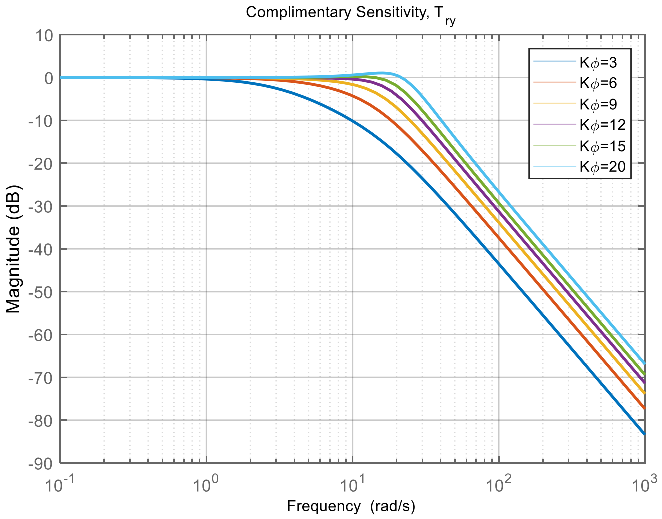


Figure 5.21: Complimentary Sensitivity of Attitude Control when k_p changes

The sensitivity of attitude control is presented in Figure 5.22

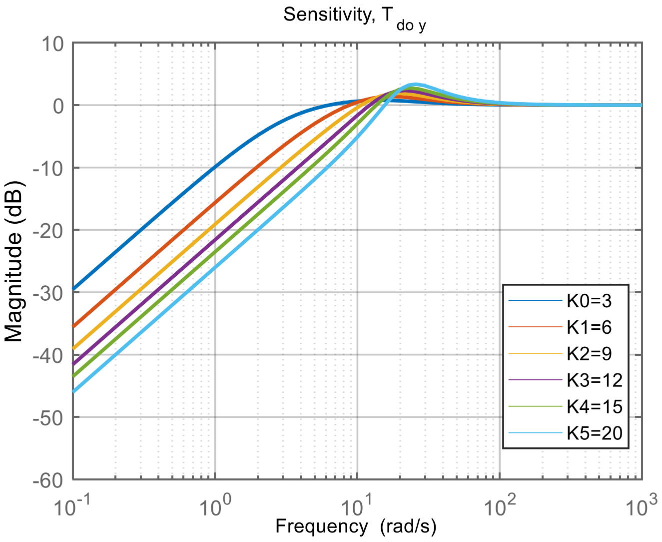


Figure 5.22: Sensitivity of Attitude Control when k_p changes

The sensor noise to output of attitude control is presented in Figure 5.23

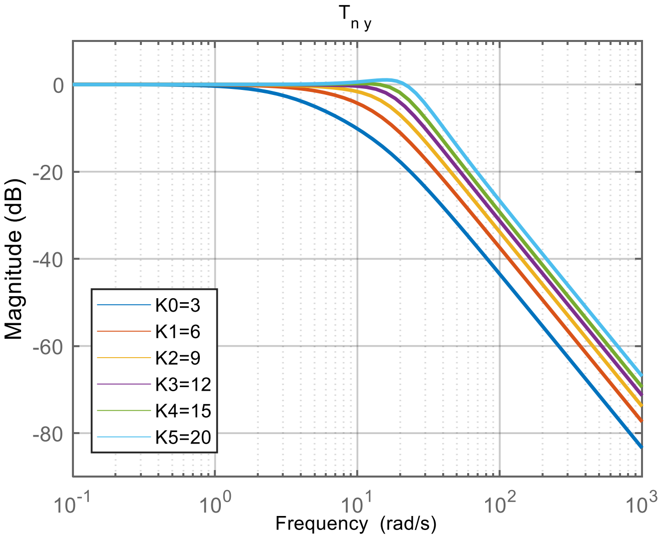


Figure 5.23: Sensor Noise to Output of Attitude Control when k_p changes

The step response of attitude control is presented in Figure 5.24

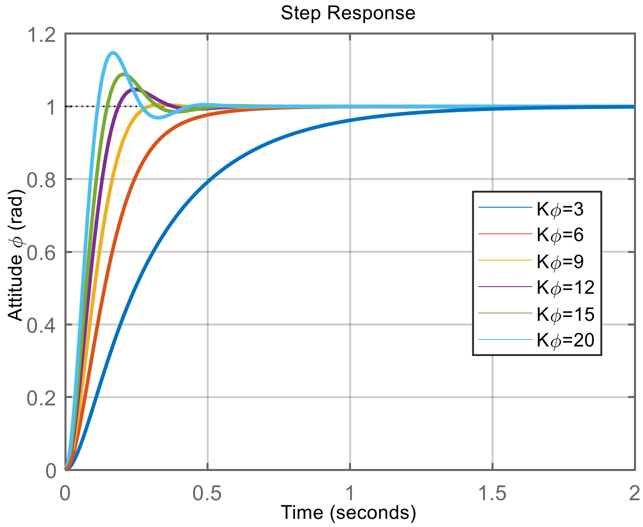


Figure 5.24: Step Response of Attitude Control when k_p changes

From the above figures, we have seen that as the angular rate loop is well tuned, we just need find the largest value of K_ϕ to ensure the possible maximum attitude loop bandwidth. And finally I have $K_\phi = K_\theta = 9$. The step response settling time is $0.26s$ (0% overshoot). The closed loop poles are at $[(-11.43, 11.23), (-11.43, -11.23), (-9.48, 0)]$ and the only zero is at $(-12, 0)$. Similarly, I can have the yaw attitude controller $K_\psi = 2$. And the closed loop poles of yaw control are at $[(-8.95, 0), (-2.01, 2.12), (-2.01, -2, 12)]$ and the only zero is at $(-12, 0)$ with that the settling time of step response is $1.58s$ (0% overshoot).

5.3 Vertical Movement Cascade Control

The vertical movement of quadrotor has the same structure with attitude movement. So we can use the same idea to design vertical movement cascade control. The block diagram is shown in Figure 5.25.

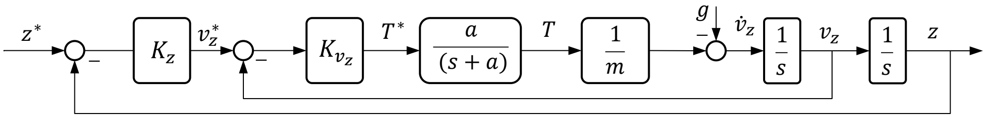


Figure 5.25: Close Loop Diagram of Vertical Movement Control

5.3.1 Vertical Velocity Control

We can use the same idea from design of angular rate control with pole placement to design a PD controller. The vertical velocity of the quadrotor is from state estimation (fusion of accelerometer and MOCAP). The frequency of vertical control in the firmware is $100Hz$. The open loop transfer function L_{v_z} is given by

$$L_{v_z} = P_{v_z} K_{v_z} = g(s+z)\frac{15.18}{s(s+9.79)} \tag{5.12}$$

With $z = 12$, we have the complimentary sensitivity of vertical velocity control is presented in Figure 5.26

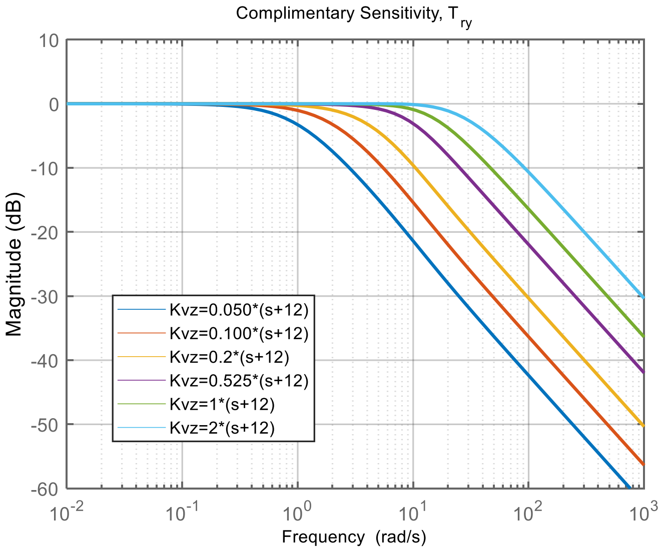


Figure 5.26: Complimentary Sensitivity of Vertical Velocity Control when g changes

It is clear to see the bandwidth of vertical velocity control increases as we increase g and the maximum bandwidth is limited by sensor noise. We can also the sensiticity plot prersented in Figure 5.27

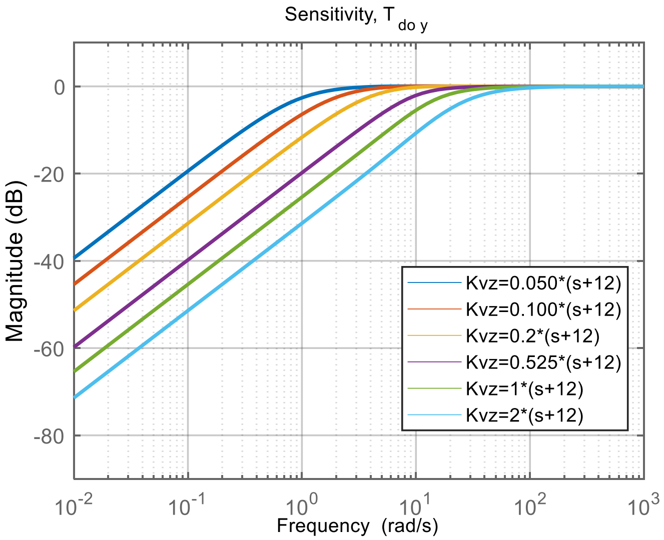


Figure 5.27: Sensitivity of Vertical Velocity Control when g changes

And we have sensor noise to output in Figure 5.28

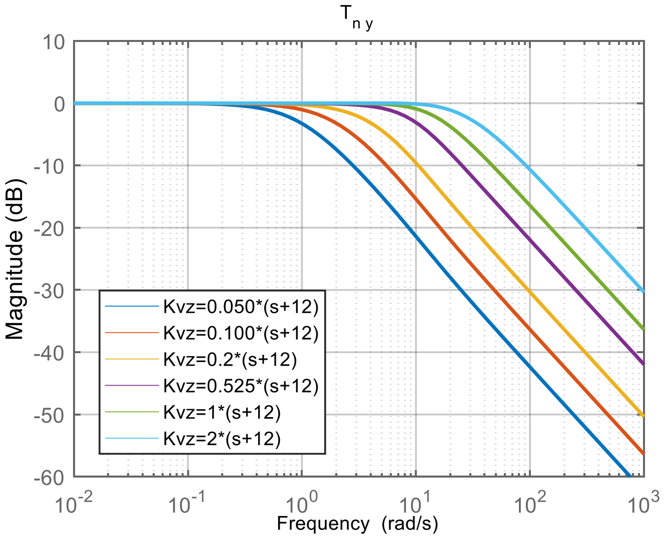


Figure 5.28: Sensor Noise to Output of Vertical Velocity Control when g changes

The step response of attitude control is presented in Figure 5.29

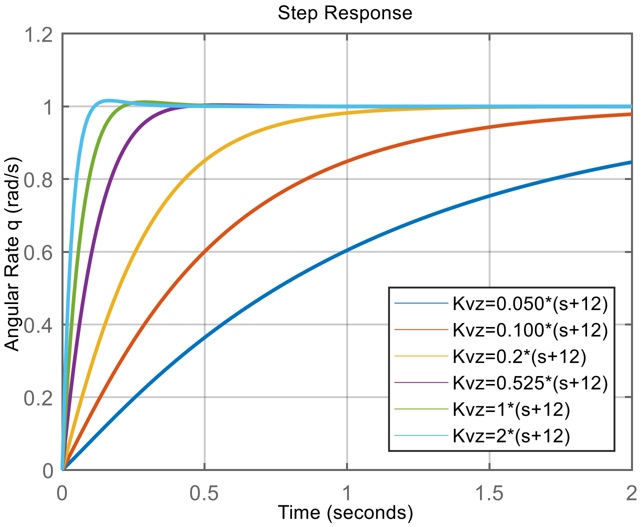


Figure 5.29: Step Response of Vertical Velocity Control when g changes

Finally, we have $K_{v_z} = 0.525(s + 12)$ as the finalized vertical velocity controller. The closed loop poles are at $[(-8.88, 4.10), (-8.88, -4.10)]$ and the only zero is at $(-12, 0)$. The settling time of step response is 0.34s (0.34% overshoot).

5.3.2 Altitude Control

The altitude of the quadrotor is from state estimation (sensor fusion of accelerometer and MOCAP). The frequency of altitude control in the firmware is $100Hz$. The altitude plant transfer function P_z is given by

$$P_z = \frac{7.969s + 95.62}{s^3 + 17.76s^2 + 95.62s} \tag{5.13}$$

Similar as attitude control, we can use proportional control as the altitude controller. We have the open loop root locus of $L = PK$ in Figure 5.30

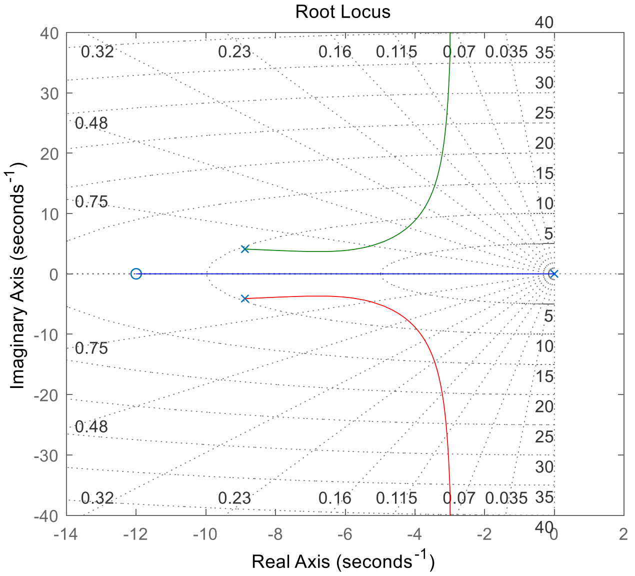


Figure 5.30: Open Loop Root Locus of Altitude Control

The complimentary sensitivity of altitude control is presented in Figure 5.31

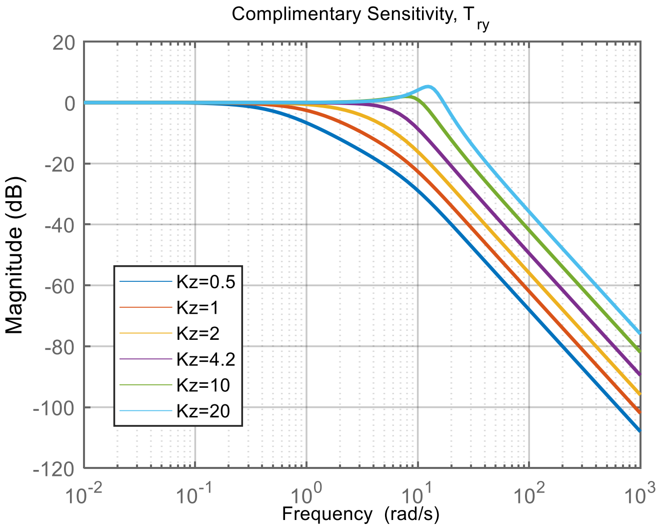


Figure 5.31: Complimentary Sensitivity of Altitude Control when k_p changes

The sensitivity of altitude control is presented in Figure 5.32

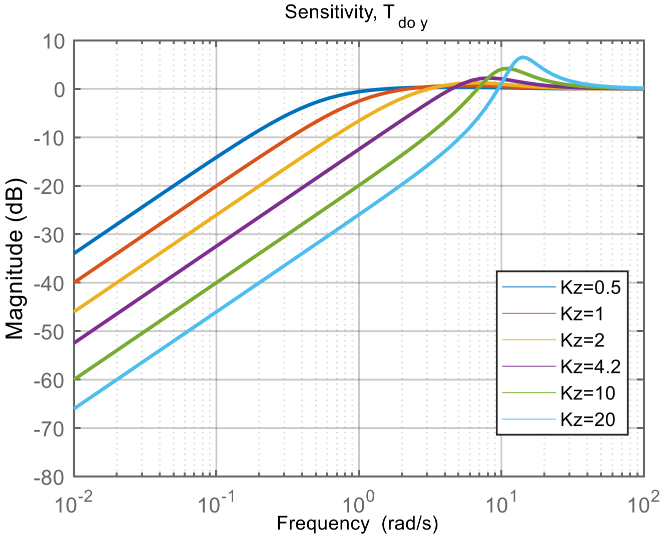


Figure 5.32: Sensitivity of Altitude Control when k_p changes

The sensor noise to output of altitude control is presented in Figure 5.33

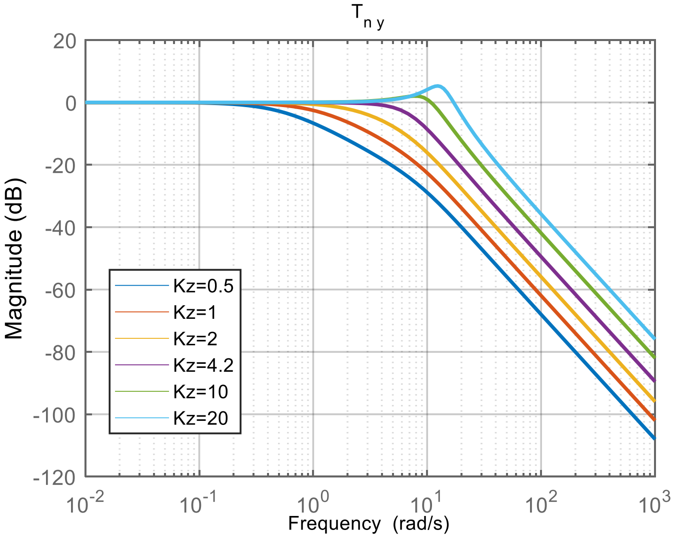


Figure 5.33: Sensor Noise to Output of Altitude Control when k_p changes

The step response of altitude control is presented in Figure 5.34

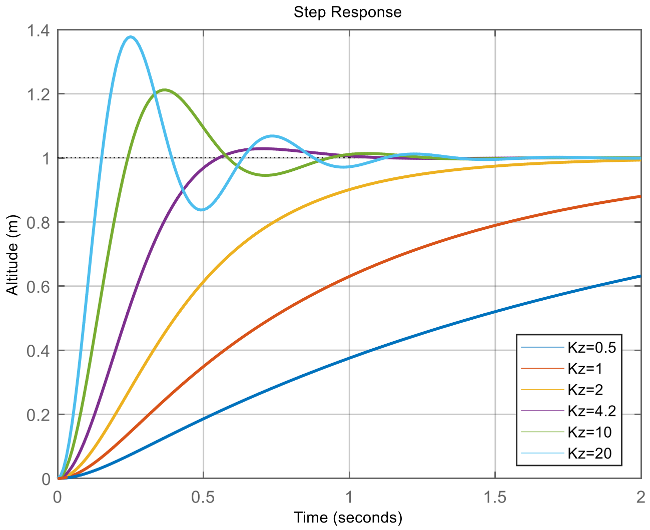


Figure 5.34: Step Response of Altitude Control when k_p changes

From the above figures, we have seen that as the angular rate loop is well tuned, we just need find the largest value of K_z to ensure the possible maximum attitude loop bandwidth. And finally we have $K_z = 4.2$. The closed loop poles are at $(-7.82, 0)$, $(-4.9708, 5.16)$ and $(-4.9708, -5.16)$ and the only zero is at $(-12, 0)$. Besides, the input command of vertical movement control is designed with saturation for flight safety. So design of altitude control law has more conservation than that of attitude control.

5.4 Translational Movement Cascade Control

After the design of attitude cascade control, we can achieve full-state control on translational movement with help of the HTC VIVE Tracking System. The block diagram of translational movement cascade control is shown in Figure 5.35.

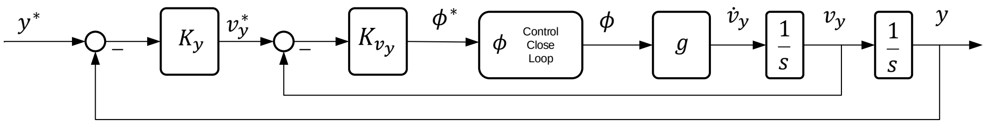


Figure 5.35: Close Loop Diagram of Translational Movement Cascade Control

5.4.1 Translational Velocity Control

In this section, we describe (V_x, V_y) Translational control for this quadrotor platform equiped with the *MARK3* flight controller. The velocity of the quadrotor is from state estimation (sensor fusion of accelerometer and MOCAP). The frequency of translational control in the firmware is $100Hz$.

Control Design: PD Controller. Based on the previous section we have the plant model including attitude control close loop, a $g_{gravity}$ factor and an integral item

$$P_{v_y} = \frac{1989s + 23874}{s^4 + 32.34s^3 + 473.6s^2 + 2435s} \tag{5.14}$$

We have the open loop root locus of $L = PK$ in Figure 5.36 ($K_{vy} = 0.038(s + 20)$)

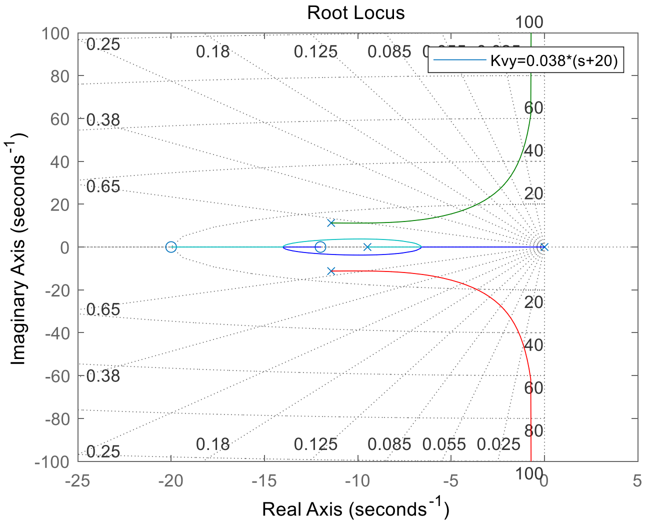


Figure 5.36: Open Loop Root Locus of Translational Velocity Control

The CF of translational velocity control is presented in Figure 5.37

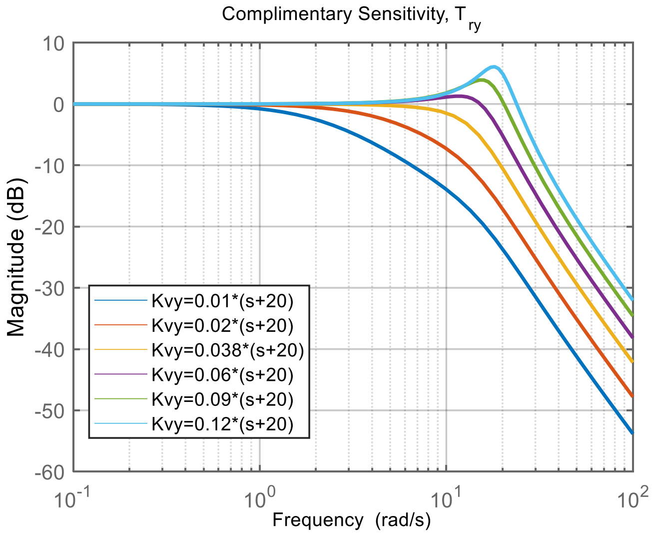


Figure 5.37: Complimentary Sensitivity of Translational Velocity Control when g changes

The sensitivity of translational velocity is presented in Figure 5.38

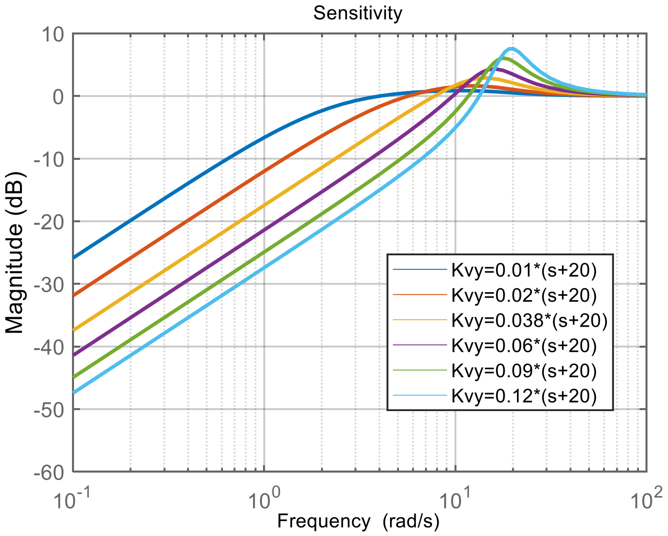


Figure 5.38: Sensitivity of Translational Velocity Control when g changes

The sensor noise to output of translational velocity is presented in Figure 5.39

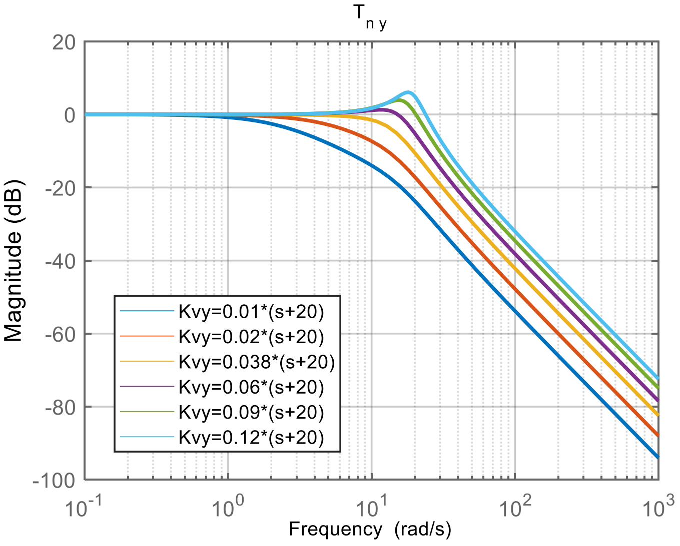


Figure 5.39: Sensor Noise to Output of Translational Velocity Control when g changes

The step response of translational velocity is presented in Figure 5.41

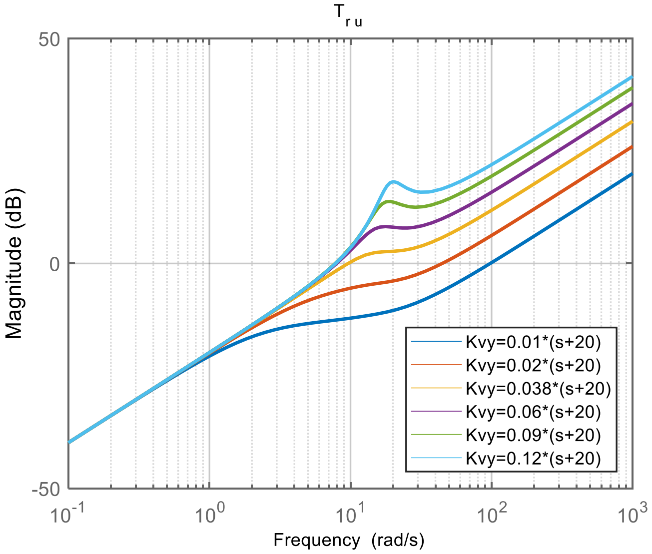


Figure 5.40: Input to Control of Translational Velocity Control when g changes

The input to control of translational velocity is presented in Figure 5.40

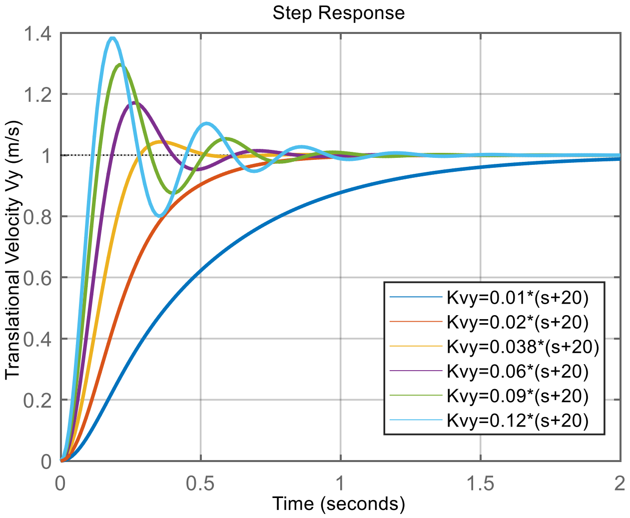


Figure 5.41: Step Response of Translational Velocity Control when g changes

Finally, we pick $K_{v_y} = 0.038(s+20)$ as the finalized translational velocity controller as the translational velocity controller is designed for lower bandwidth compared with attitude control. The closed loop poles are at $(-7.32, 12.05)$, $(-7.32, -12.05)$, $(-8.85, 3.56)$ and $(-8.85, -3.56)$ and zeros are at $(-20, 0)$ and $(-12, 0)$ with settling time of step response is 0.45s (4% overshoot).

5.4.2 Translational Position Control

The translational position of the quadrotor is from state estimation (sensor fusion of accelerometer and MOCAP). The frequency of translational position control in the firmware is $100Hz$. The translational position plant transfer function P_y is given by

$$P_y = \frac{151.2s^2 + 4837s + 36282}{s^5 + 32.34s^4 + 549.1s^3 + 5005s^2 + 36282s} \tag{5.15}$$

Similar as attitude control, we can use proportional control as the translational position controller. We have the open loop root locus of $L = PK$ in Figure 5.42 when $K_y = 2.0$

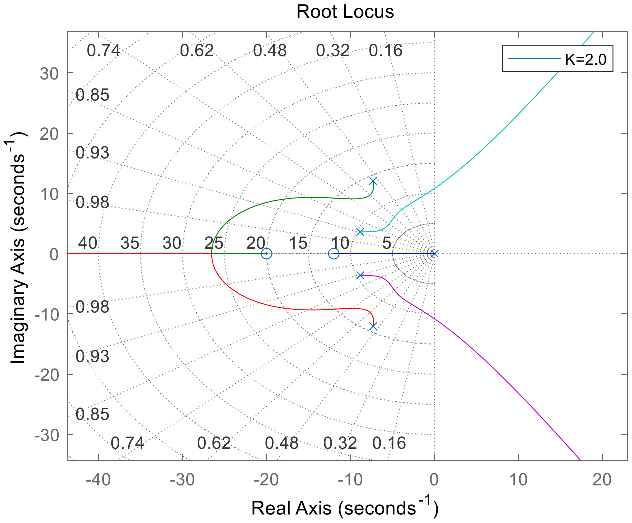


Figure 5.42: Open Loop Root Locus of Translational Position Control

The complimentary sensitivity of position control is presented in Figure 5.43

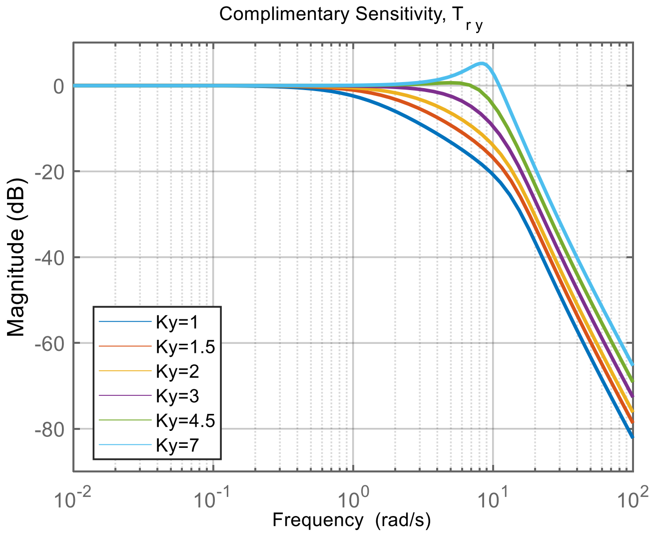


Figure 5.43: CF of Translational Position Control when k_p changes

The sensitivity of position control is presented in Figure 5.43

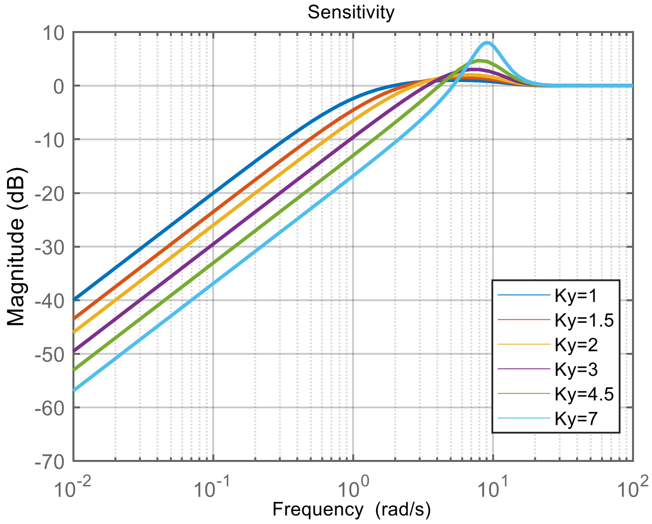


Figure 5.44: Sensitivity of Translational Position Control when k_p changes

The step response of position control is presented in Figure 5.45

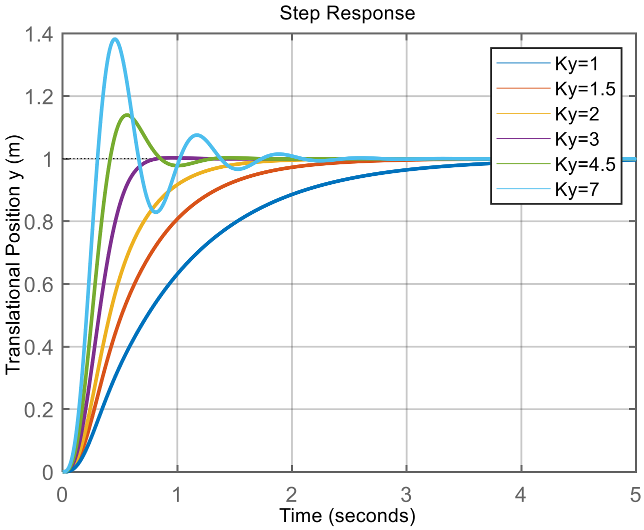


Figure 5.45: Step Response of Translational Position Control when k_p changes

Finally we have the finalized $K_y = 2.0$ with closed loop poles at $(-7.27, \pm 11.18)$, $(-7.41, \pm 3.66)$ and $(-2.99, 0)$ and zeros at $(-20, 0)$ and (-12,0).

5.5 Summary and Conclusions

This Chapter provides a complimentary study for full-state classical control of quadrotor. All control law developments is mainly based on feedback control theory. The actuator and sensor performance limited the bandwidth of the quadrotor near hovering mode.

Chapter 6

FLIGHT STATE ESTIMATION

6.1 Onboard Sensor Calibration

Gyroscope Thermal Calibration

For silicon MEMS gyroscopes with high quality (Q) factors [23] and [24]. In engineering application, it is needed to remove as much of the offset as possible before processing. This is achieved by heating the sensor board to 65°C and cooling the board to 15°C. The curve fitting is done by MATLAB curve fitting toolbox. A 4th degree polynomial was used to describe the gyroscope bias with temperature ranges from 15°C to 65°C. The curve fitting of 3 axis is shown in Figure 6.1, Figure 6.2 and Figure 6.3.

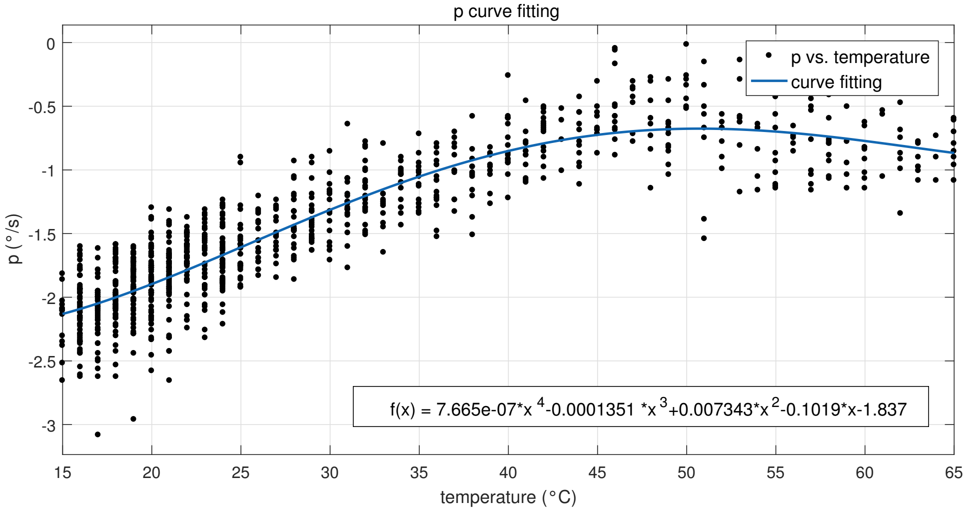


Figure 6.1: gyroscope p temperature calibration

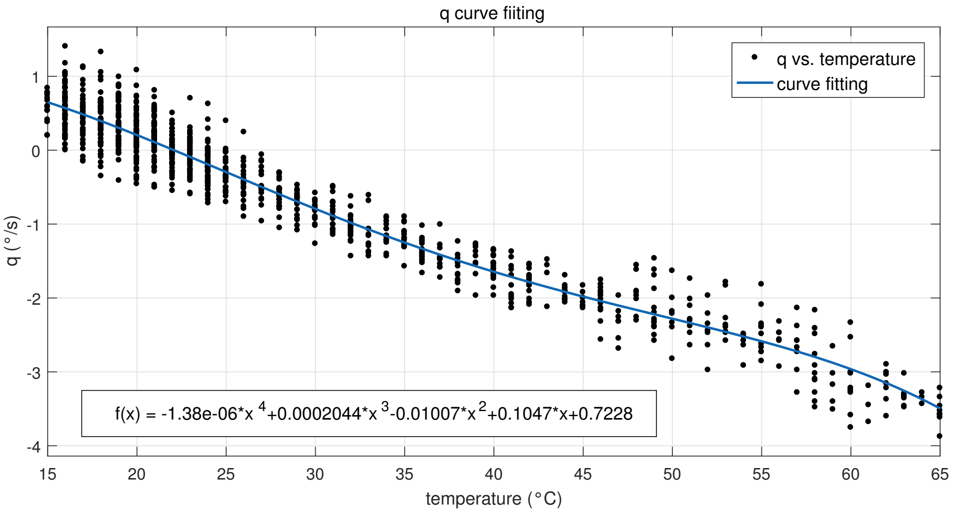


Figure 6.2: gyroscope q temperature calibration

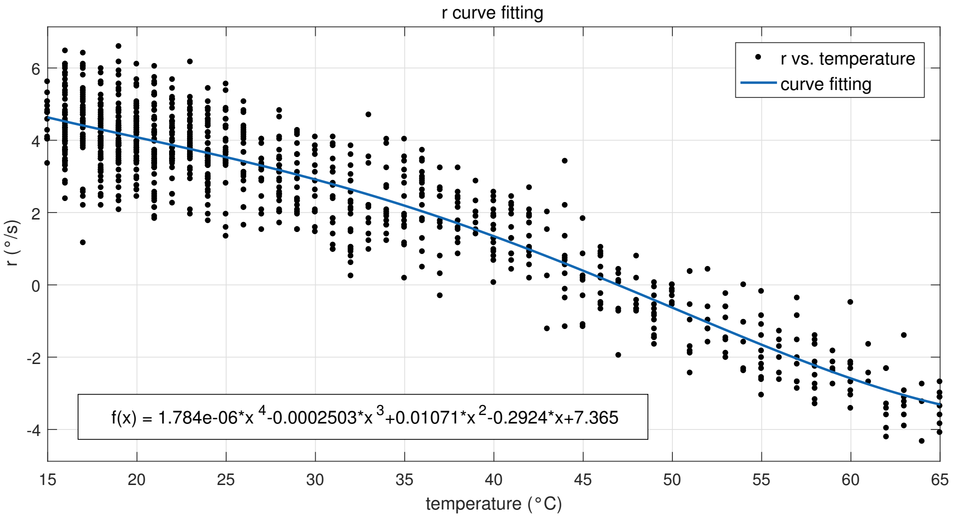


Figure 6.3: gyroscope r temperature calibration

The thermal calibration stabilizes the gyroscope output and remove most of the gyroscope offset.

Accelerometer 6-point Tumble Calibration

Based on [25] and [26], an accelerometer requires calibration via 6-point Tumble calibration in Figure 6.4. This way is easy to use on every MCU.

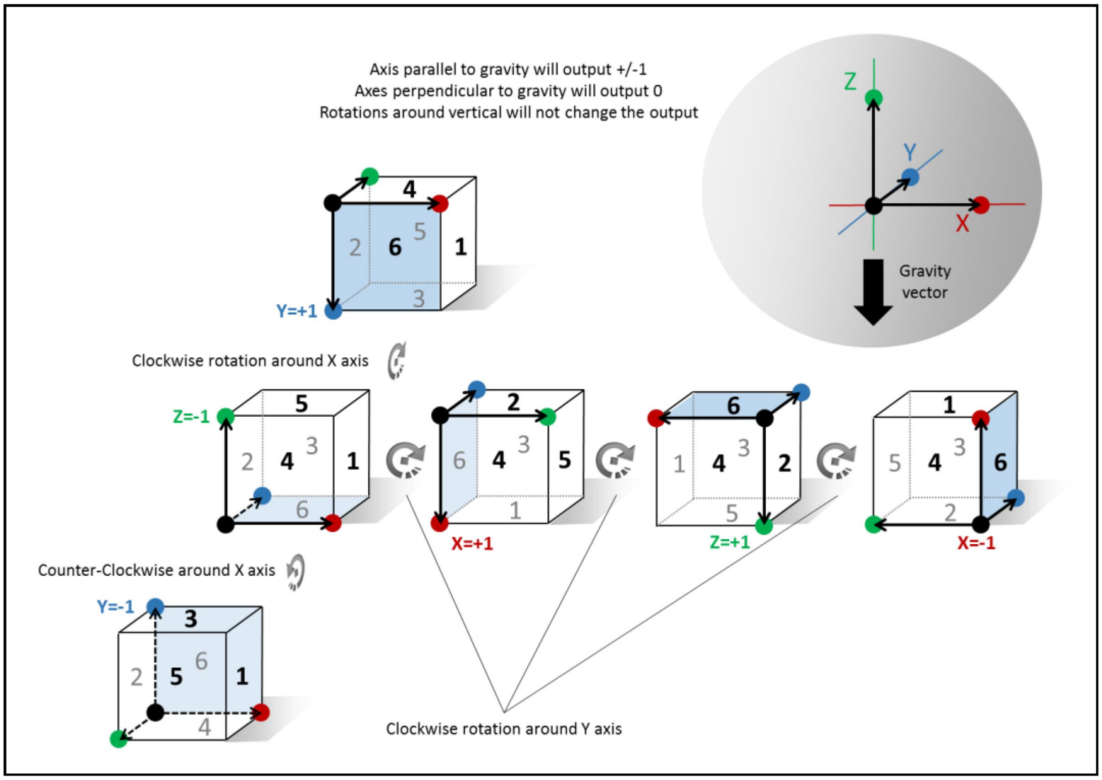


Figure 6.4: Reference for Sensor Orientation While Performing 6-point Tumble Calibration (ST DT0053 Design tip)

The algorithm is described for the particular case of an accelerometer. But it can also be approached on a magnetometer. How true acceleration is related to measured acceleration:

$$\begin{bmatrix} a_{raw,x} \\ a_{raw,y} \\ a_{raw,z} \end{bmatrix} = \begin{bmatrix} g_x & g_{yx} & g_{zx} \\ g_{xy} & g_y & g_{zy} \\ g_{xz} & g_{yz} & g_z \end{bmatrix}_{A_{Cal}} \begin{bmatrix} a_{true,x} \\ a_{true,y} \\ a_{true,z} \end{bmatrix} + \begin{bmatrix} X_{offset} \\ Y_{offset} \\ Z_{offset} \end{bmatrix} \tag{6.1}$$

We need to measure raw accelerometer 3-axis output of 6 points:

Equivalent gravity vector along $+X$ **axis**, $a_{true,1} = [+g, 0, 0]^T$

Equivalent gravity vector along $-X$ **axis**, $a_{true,2} = [-g, 0, 0]^T$

Equivalent gravity vector along $+Y$ **axis**, $a_{true,3} = [0, +g, 0]^T$

Equivalent gravity vector along $-Y$ axis, $a_{true,4} = [0, -g, 0]^T$

Equivalent gravity vector along $+Z$ axis, $a_{true,5} = [0, 0, +g]^T$

Equivalent gravity vector along $-Z$ axis, $a_{true,6} = [0, 0, -g]^T$

Based on the above measurements, we can calculate the offset in each axis:

Offset along X axis, $X_{offset} = \frac{1}{2}(a_{raw,x_1} + a_{raw,x_2})$

Offset along Y axis, $Y_{offset} = \frac{1}{2}(a_{raw,y_3} + a_{raw,y_4})$

Offset along Z axis, $Z_{offset} = \frac{1}{2}(a_{raw,z_5} + a_{raw,z_6})$

The offsets are computed by summing two out of measures listed above. Then the gain matrix (9 numbers in A_{cal}) can be calculated by applying three point (one point from each axis).

After data in all 6 points have been recorded, the offset and the gain matrix will be stored in the flight controller EEPROM. After calibration, we will see that the plane static calibrated accelerometer output is much closer to $[0, 0, g]^T$ shown in Figure 6.5, Figure 6.6 and Figure 6.7

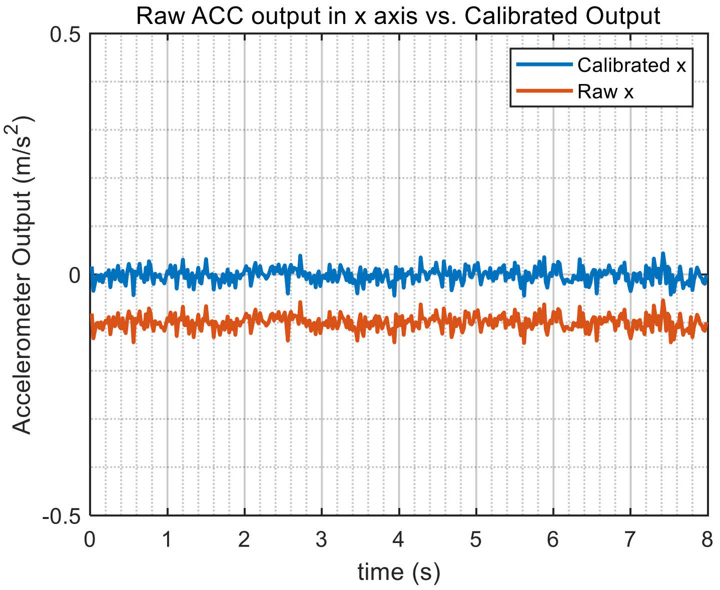


Figure 6.5: Calibrated ACC output vs. Raw output in x axis

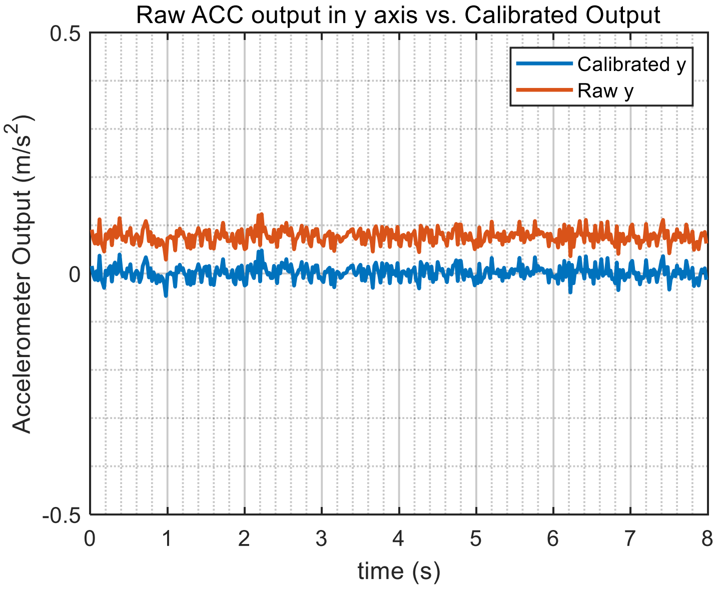


Figure 6.6: Calibrated ACC output vs. Raw output in y axis

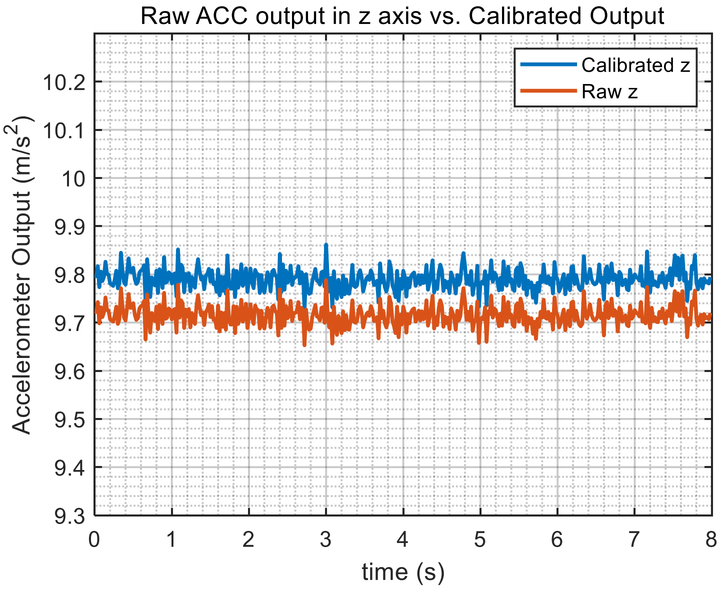


Figure 6.7: Calibrated ACC output vs. Raw output in z axis

The 6-point accelerometer calibration ensures that the resultant force vector of quadrotor in body frame is close to reality. Based on [4], it is not possible to estimate the attitude of a quadrotor in flight without drift by only using IMU measurements. So it is needed to use measurements from the motion capture system.

6.2 IIR Low-Pass Filtering

Based on [27], MEMS (micro-electro-mechanical system) based inertial sensors not only suffer from bias instability, but also noisy output. From Chapter 5, we have seen that noisy sensor feedback limits the quadrotor bandwidth. It is needed to reduce the mechanical noise caused by motors. Here the noise analysis experiment is performed to give the noise specturm.

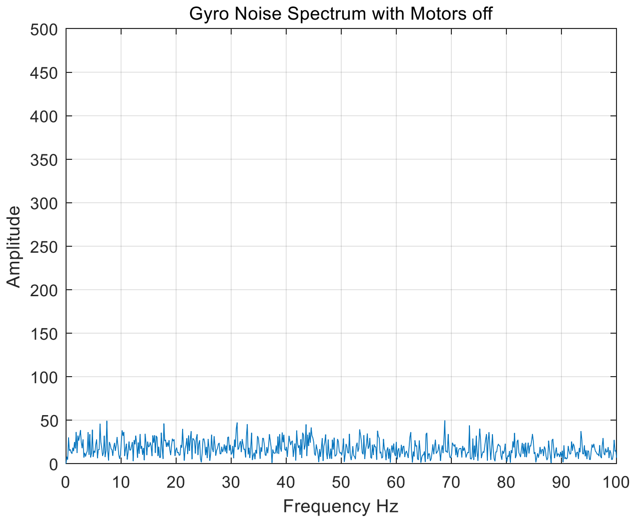


Figure 6.8: Fast Fourier Transform of Gyroscope Output when Motors off

In Figure 6.8, it is clear to see that the zero output of gyroscope is close to white noise (we assume it is gaussian) when the quadrotor holds still on ground (the quadrotor is put on normal foam mats) and motors are off.

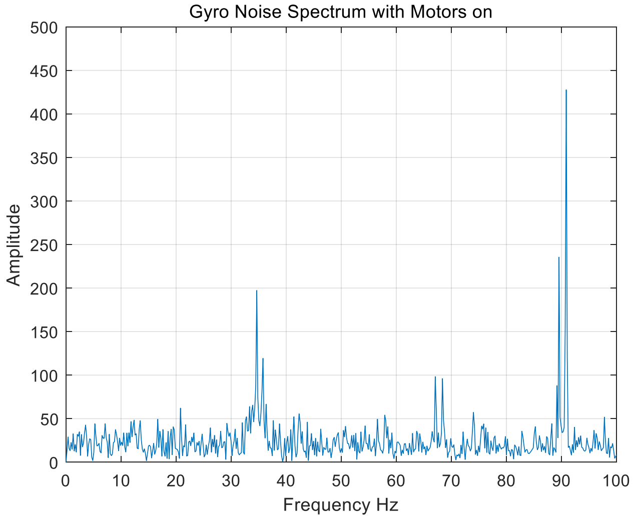


Figure 6.9: Fast Fourier Transform of Gyroscope Output when Motors on

As we turn on all four motors and increase the rotation speed (without propellers) towards $T = mg$ for simulating quadrotor hovering mode. In Figure 6.9, it is clear to see that there are peaks on 35 Hz, 67 Hz and 90 Hz. Accelerometer has similar characteristics with gyroscope. So we need to design a low-pass filter cutting off at 30 Hz for onboard MEME sensors.

IIR (Infinite Impulse Response) filters are the most efficient type of filter to implement in DSP (digital signal processing). MATLAB IIR filter toolbox in Figure 6.10 offers multiple choices of IIR filters.

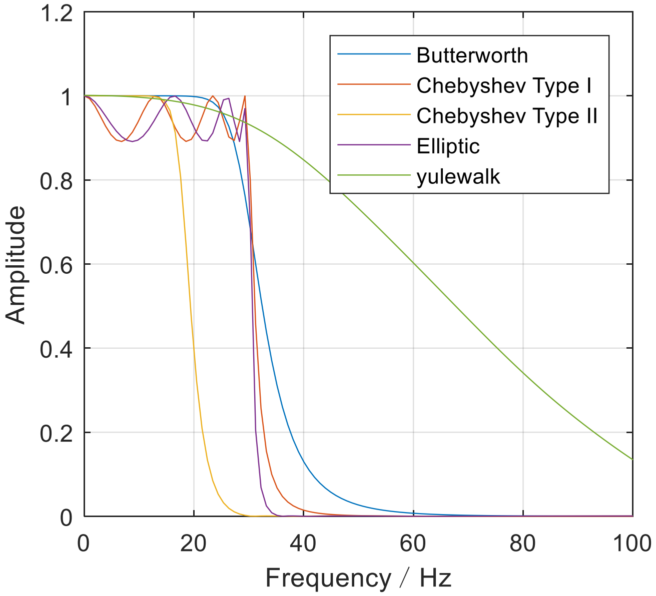


Figure 6.10: Different IIR Filters Spectrum Using MATLAB Toolbox

Here we are using Butterworth filter as the IIR filter to deal with MEMS sensor mechanical noise. We can represent it in laplacian form:

$$H = \frac{35532.45}{s^2 + 377.16s + 35532.45} \tag{6.2}$$

Where the cut-off frequency is $\frac{\sqrt{35532.45}}{2\pi} = 30$ Hz. This IIR filter is implemented in the

MARK3 flight controller. The filtered gyroscope zero output spectrum is presented in Figure 6.11. Compared with Figure 6.9, the noise generated from motors is well filtered.

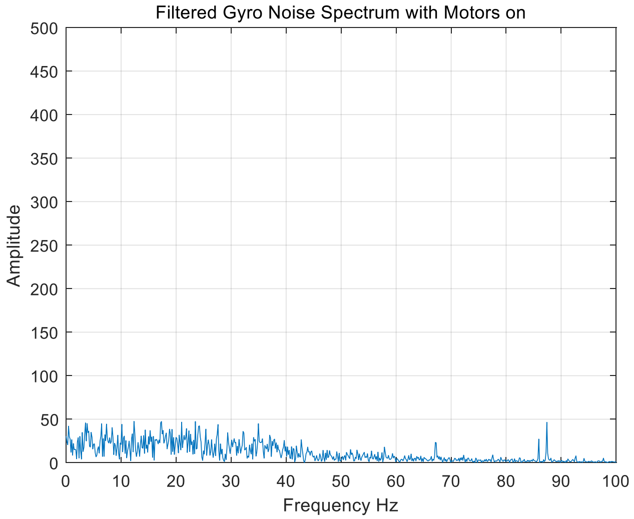


Figure 6.11: Filtered Gyroscope Zero Output Spectrum with Motors on

In the flight controller firmware, the filtered gyroscope output is defined as the angular rate feedback and the filtered accelerometer output is defined as resultant force vector joining the full-state estimation.

6.3 Quaternion Based Attitude Estimation

Euler Angle representation are used to show rotation in the model. This chapter considers the $Z-Y-X$ convention as the rotation order within a rotation around the yaw axis, a rotation around the new pitch axis and a rotation around the new roll axis. This method of rotation representation has two main disadvantages in the flight controller firmware.

Singularities Singularities exist in Euler Angle especially when yaw angle is at

$\pm\pi$, pitch angle is at $\pm 0.5\pi$ and roll angle is at $\pm\pi$. These changes require massive use of conditional statements to attempt to correct when executing sensor fusion algorithm. Besides, high computational cost is needed for huge use of trigonometric function in the math.h library.

Gimbal Lock When the two axes of the quadrotor are driven into a parallel configuration, the rotation system will lose one degree of freedom. This would cause strange effects in the attitude estimation resulting accidents.

Quaternion is widely used in the UAV state estimation and it can be written in two forms [11]:

$$q = q_0 + q_1 i + q_2 j + q_3 k \tag{6.3}$$

and

$$q = [q_0, q_1, q_2, q_3]^T \tag{6.4}$$

The multiplication of quaternions can be represented as:

$$p \otimes q = \begin{bmatrix} p_0 & -p_1 & -p_2 & -p_3 \\ p_1 & p_0 & p_3 & -p_2 \\ p_2 & -p_3 & p_0 & p_1 \\ p_3 & p_2 & -p_1 & p_0 \end{bmatrix} q = p \begin{bmatrix} q_0 & -q_1 & -q_2 & -q_3 \\ q_1 & q_0 & -q_3 & q_2 \\ q_2 & q_3 & q_0 & -q_1 \\ q_3 & -q_2 & q_1 & q_0 \end{bmatrix} \tag{6.5}$$

The rotation matrix from body frame to earth frame can presented using quaternion $\mathbf{x}_E = R_{B\to E}(q)\mathbf{x}_B$. And we have

$$R_{B\to E} = \begin{bmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2(q_1q_2 + q_0q_3) & 2(q_1q_3 - q_0q_2) \\ 2(q_1q_2 - q_0q_3) & q_0^2 - q_1^2 + q_2^2 - q_3^2 & 2(q_2q_3 + q_0q_1) \\ 2(q_1q_3 + q_0q_2) & 2(q_2q_3 - q_0q_1) & q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{bmatrix} \tag{6.6}$$

and the derivative ($[r_x, r_y, r_z]^T$ is the angular rate data collected from gyroscope)

$$\begin{bmatrix} \dot{q}_0 \\ \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 0 & -r_x & -r_y & -r_z \\ r_x & 0 & r_z & -r_y \\ r_y & -r_z & 0 & -r_x \\ r_z & r_y & -r_x & 0 \end{bmatrix} \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix} \tag{6.7}$$

And we have the iteration step

$$q(k+1) = q(k) + \dot{q}(k)T \tag{6.8}$$

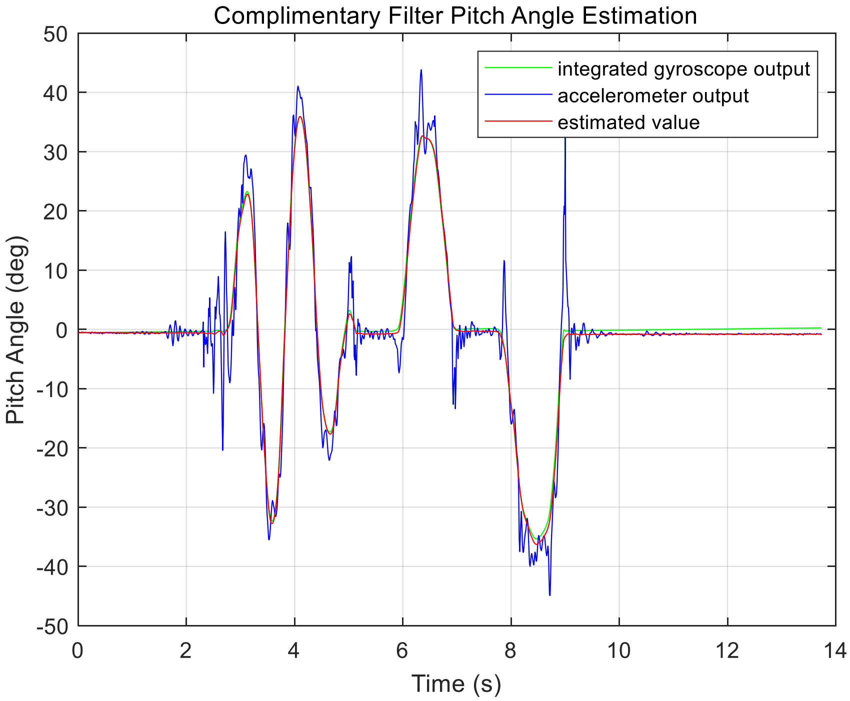


Figure 6.12: Complimentary Filter Pitch Angle Estimation

The Explicit Complimentary Filter is based on [11]. If the gyroscope has no bias on aircrafts, we can use integral term of gyroscope output as attitude feedback value. But even the gyroscope calibration is executed and most of the bias is removed, we still cannot use integral term of gyroscope. By using the explicit complimentary filter in Figure 6.13, we can add accelerometer and MOCAP as reference to make the estimated value is close to real value.

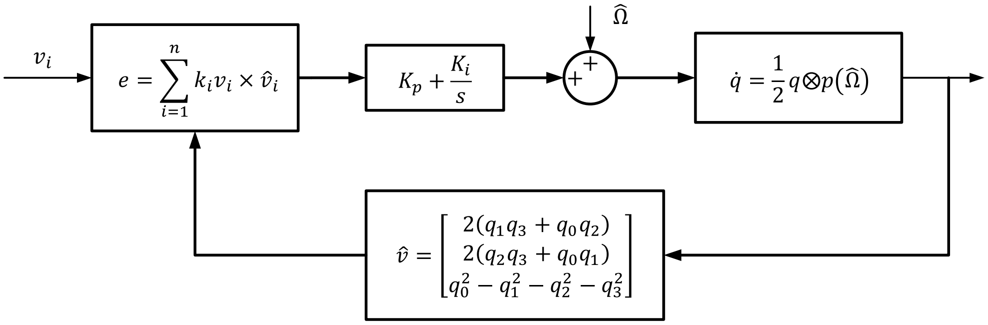


Figure 6.13: Explicit Complimentary Filter Diagram

And the complimentary filter estimation on pitch angle is shown in Figure 6.12. It is clear to see that the attitude estimation removes bias of integral gyroscope output and converges to noisy output accelerometer.

6.4 Position & Velocity Estimation

Even though we have an accurate MOCAP system, we still need to do state estimation to reduce influence of communication signal loss shown in Figure 6.12 and Figure 6.15. It is clear to see the estimated value is reliable when the length of signal loss is small. The velocity estimation iteration step is based on

$$v(k+1) = v(k) + a(k)T \tag{6.9}$$

And the position estimation iteration step is based on

$$x(k+1) = x(k) + v(k)T + \frac{1}{2}a(k)T^2 \tag{6.10}$$

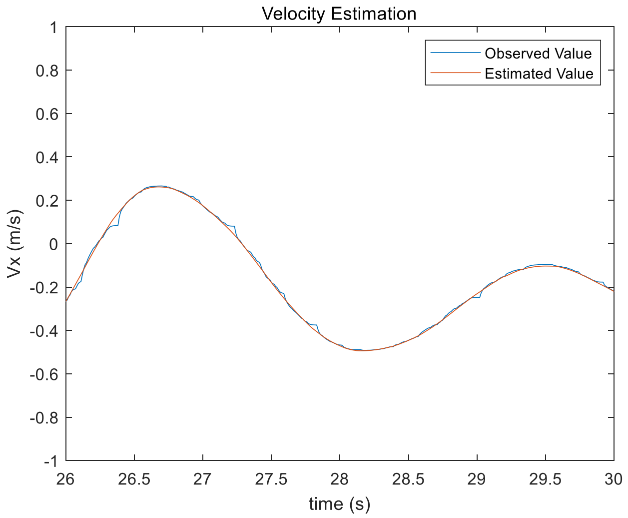


Figure 6.14: Complimentary Filter Velocity in x axis Estimation

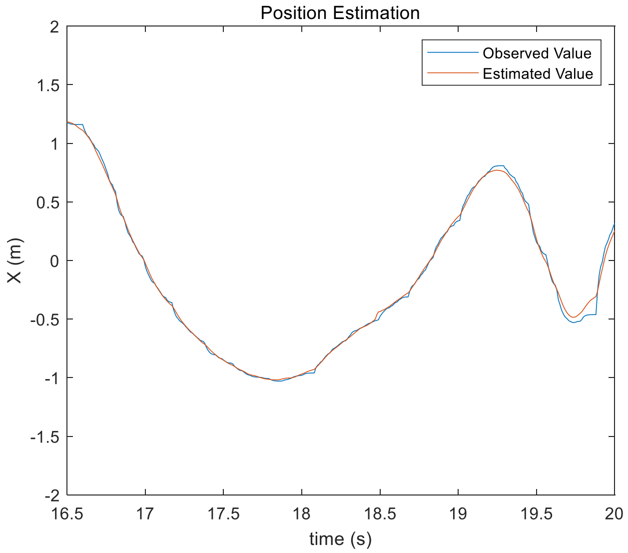


Figure 6.15: Complimentary Filter Position in x axis Estimation

The output from the estimator is used as the feedback for all controllers.

Chapter 7

EXPERIMENTAL HARDWARE RESULTS AND COMPARISONS WITH SIMULATION RESULTS

The simulated data is compared with actual data from flight experiment. The simulated data is generated from Simulink.

7.1 Attitude Command-response Graph with Simulation Results

The step response of ϕ control is presented in Figure 7.1

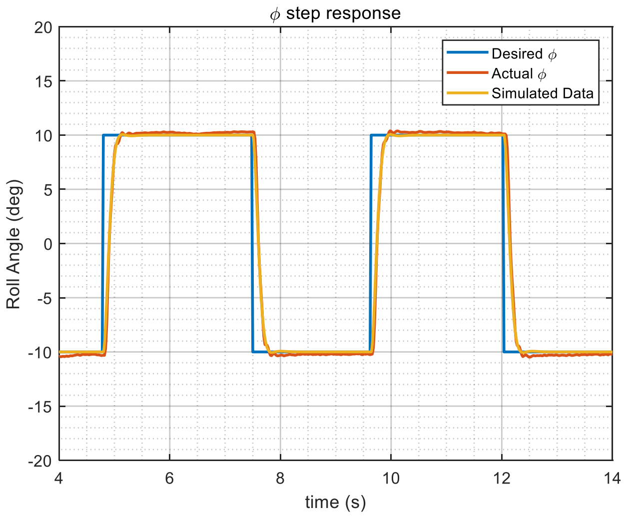


Figure 7.1: ϕ Step Response with Simulated Data

The ramp response of ψ control is presented in Figure 7.2

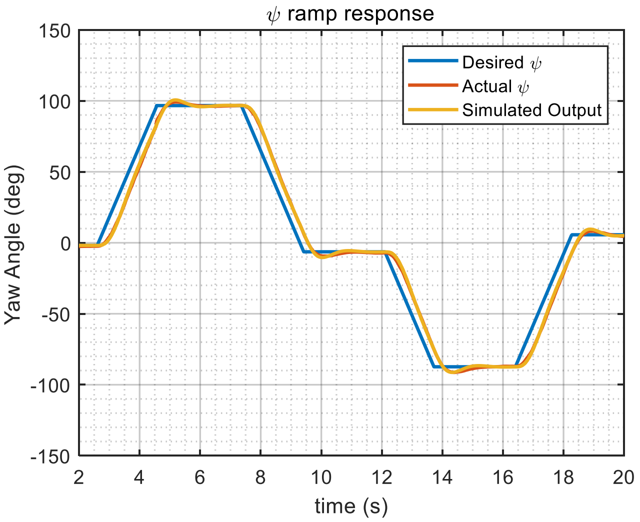


Figure 7.2: ψ Ramp Response with Simulated Data

7.2 Altitude Command-response Graph with Simulation Results

The ramp response of altitude control is presented in Figure 7.3

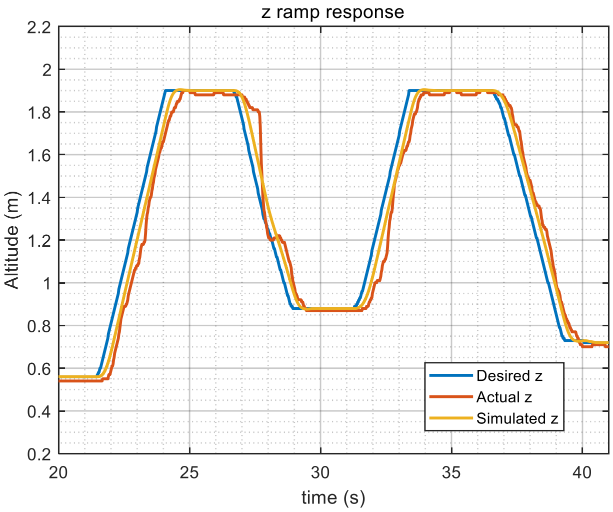


Figure 7.3: Altitude Ramp Response with Simulated Data

7.3 Translational Position Command-response Graph with Simulation Results

The ramp response of translational position control is presented in Figure 7.4

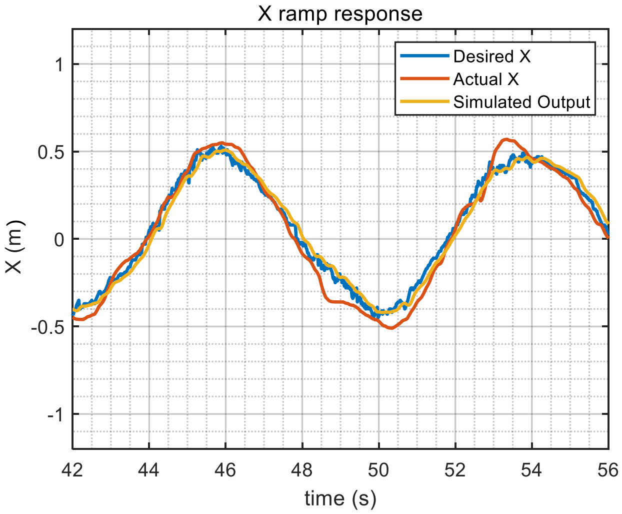


Figure 7.4: Position Ramp Response with Simulated Data

7.4 Summary and Conclusions

This chapter provided comparision of simulation and hardware results. With the HTC VIVE Tracking System and the upper computer GUI, the data is well collected. For inner loop (Attitude Control) we can get good actual response. But for outer loop(Altitude and Translational Position Control), the output is affected by turbulence generated from propeller when the quadrotor is flying near the ground as shown above. Because of the experiment site limitation, we currently can only do flight test small area with HTC VIVE of SteamVR 1.0. In the future, HTC VIVE based on SteamVR 2.0 offering larger tracking area will solve this problem.

Chapter 8

CONCLUSIONS AND DIRECTIONS FOR FUTURE RESEARCH

8.1 Summary of Work

This thesis addressed design, modeling and control of a quadrotor platform for indoor environments. This platform is capable of indoor formation flight. The following summarizes key themes within the thesis.

1. **Design and Implementation of Mark3 Flight Controller.** In this thesis, the *MARK3* flight controller is designed including a 120MHz Teensy 3.2 MCU, a GY-89 10DOF Sensor Board and Xbee 3.0 to offer enough design of redundancy for quadrotor research. The flight controller firmware contains communication processing, full-state estimation, full-state feedback control and system checking function. The *MARK3* flight controller PCB design, code and instructions have been uploaded to `https://github.com/ragewrath/Mark3-Copter-Pilot`.

2. **Overall Indoor Flight Architecture.** A 250mm low-cost platform is designed giving enough payload and agility. The HTC VIVE tracking system is introduced for indoor tracking with submillimeter accuracy. A MATLAB based GUI is designed for communication and receive tracking data from SteamVR API (see Appendix B).

3. **Modeling.** The rigid-body dynamic model and the actuator dynamics model were presented and analyzed. A linearized model with actuator dynamics near hovering model is designed which is useful for the full-state feedback control. The moment parameters are measured with a bifilar pendulum experiment.

The actuator parameters are measured using RCbenchmark 1580 test stand. All parameters are well measured and applied in the control simulation and the flight controller firmware.

4. **Control.** The full-state feedback control design is discussed including attitude cascade control, altitude cascade control and translational movement cascade control. Step response and ramp response experiment is also executed to compare the actual data and simulated data. The whole design shows strong robustness and limitations of the design caused by actuator performance limitation and sensor noise. This part is very useful to researchers pursuing quadrotor developments.

5. **Estimtation.** Then onboard sensor calibration and low-pass filtering is presented to get high quality gyroscope output and accelerometer output. The full-estimation is illustrated in the flight controller firmware with sensor fusion of accelerometer, gyroscope and HTC VIVE tracking system. Attitude state value is from the quaternion based complimentary filter with a PI feedback loop. Position and velocity value is from complimentary filter to get rid of influence of possible signal loss.

8.2 Directions for Future Research

Future work will involve each of the following:

1. **Formation Flight.** With compatibility of HTC VIVE tracking system and Xbee 3.0, multi-quadrotor cooperation can performed and the upper computer collects all the data from quadrotors and HTC VIVE tracking system to determine what to do next.

2. **On-board Sensing.** The *MARK3* flight controller is designed with enough redundancy for on-board sensing. Multiple pin headers for I2C and UART communication ensure the flight controller can be connected addtional devices (GPS, Rangefinders, LIDAR, High-level Embedded System, etc).

3. **Human-Robot Interaction.** The compatibility with HTC VIVE tracking system on this quadrotor platform offers possibility that human-robot interaction uses virtual reality devices to identify gestures and body movements.

4. **Flight in Virtual Reality.** Using the HTC VIVE tracking system offers possibility to fly drones in the most realistic simulation with Unity. In this way, the drone experiences real physics, gets real inertial measurements, but gets photorealistically simulated camera images. This allows researchers and developers to fly their drones in various simulated virtuals environments.

REFERENCES

[1] Pengcheng Wang, Zhihong Man and Zhenwei Cao, “Dynamics modelling and linear control of quadcopter”, International Conference on Advanced Mechatronic Systems, 2016.

[2] Chang Liu, “Design and Implementation of a Mini Quadrotor Control System in GPS Denied Environments”, International Conference on Unmanned Aircraft Systems (ICUAS), 2015.

[3] Jiao Ren, et al, “Cascade PID controller for quadrotor”, IEEE International Conference on Information and Automation (ICIA) , 2016.

[4] Matthias Faessler and Flavio Fontana, “Theory and Math Behind RPG Quadrotor Control”, University of Zurich, 2018.

[5] Tommaso Bresciani, “Modelling, Identification and Control of a Quadrotor Helicopter”, International Conference on Advanced Mechatronic Systems, Lund University, 2008.

[6] Marco Bergamasco and Marco Lovera, “Identification of Linear Models for the Dynamics of a Hovering Quadrotor”, IEEE Transactions on Control Systems Technology, 2014.

[7] M. Elsamanty, et al, “Methodology for Identifying Quadrotor Parameters, Attitude Estimation and Control” IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM), 2013.

[8] Xi Chen, “Flight Controller Design, Automatic Tuning and Performance Evaluation of Quadrotor UAVs” RMIT University, 2017.

[9] Nathan M. Zimmerman, “Flight Control and Hardware Design of Multi-Rotor Systems” Marquette University, 2009.

[10] Fengmin Yu, et al, “Autonomous Flight Control Law for an Indoor UAV Quadrotor” 29th Chinese Control And Decision Conference, 2017.

[11] Matthew Watson, “The Design and Implementation of a Robust AHRS for Integration into a Quadrotor Platform” The University of Sheffield, 2013.

[12] Mark Euston, et al, “A Complementary Filter for Attitude Estimation of a Fixed-Wing UAV” IEEE/RSJ International Conference on Intelligent Robots and Systems, 2008.

[13] Michał Nowicki, et al, “Simplicity or Flexibility? Complementary Filter vs. EKF for Orientation Estimation on Mobile Devices” IEEE 2nd International Conference on Cybernetics, 2015.

[14] Paul Riseborough, “GNC Solutions Small UAS State Estimation” Training presented to ARWORKS, 2017.

[15] Sandhya Rani Chapala, Gangadhara Sai Pirati and Usha Rani. Nelakuditi, “Determination of coordinate transformations in UAVS” Second International Conference on Cognitive Computing and Information Processing (CCIP), 2016.

[16] Matthias Faessler and Flavio Fontana, “Theory and Math Behind RPG Quadrotor Control” University of Zurich, 2018.

[17] James Diebel, “Representing Attitude: Euler Angles, Unit Quaternions, and Rotation Vectors” (2006), Stanford University.

[18] Robert Stengel, “Aircraft Equations of Motion and Flight Path Computation”, Aircraft Flight Dynamics, 2016.

[19] Biao Zhang and Wei Huo “Stabilization of Quadrotor with Air Drag Based on Controlled Lagrangians Method”, Chinese Automation Congress, 2017.

[20] Packard, Poola, Horowitz “Dynamic Systems and Feedback”, 2002.

[21] Paul E. Klopsteg “The Bifilar Pendulum”, Review of Scientific Instruments 1, 3 (1930)

[22] Ezra Tal and Sertac Karaman “Accurate Tracking of Aggressive Quadrotor Trajectories using Incremental Nonlinear Dynamic Inversion and Differential Flatness”, MIT, 2018.

[23] Dunzhu Xia *, Shuling Chen, Shourong Wang and Hongsheng Li, “Microgyroscope Temperature Effects and Compensation-Control Methods”, Sensors, 2009.

[24] I.P. Prikhodko, et al, “Thermal Calibration of Silicon MEMS Gyroscopes”, IMAPS International Conference & Exhibition on Device Packaging, 2012

[25] By Andrea Vitali, “6-point tumble sensor calibration”, STMicroelectronics, 2015

[26] Seong-hoon Peter Won and Farid Golnaraghi, “A Triaxial Accelerometer Calibration Method Using a Mathematical Model”, IEEE Transactions on Instrumentation and Measurement, 2015

[27] Mushfiqul Alam and Jan Rohac, “Adaptive Data Filtering of Inertial Sensors with Variable Bandwidth”, Sensors, 2015

APPENDIX A

MATLAB CODE

Angular Rate Control Simulation File

```
close all;
clc;
clear;
%---------Angular Rate Control---------%
%% Parameters
t=0:0.01:2;
J_y = 0.0019536;
a = 9.79;
s=tf('s');
P=1 * a /((J_y*s)*(s+a));
%% Controller
K0=0.001*(s+12);
K1=0.002*(s+12);
K2=0.0045*(s+12);
K3=0.008*(s+12);
K4=0.015*(s+12);
K5=0.040*(s+12);
%% 1 Plot L = PK
L0=P*K0;
L1=P*K1;
L2=P*K2;
L3=P*K3;
L4=P*K4;
L5=P*K5;
figure(1);
bodemag(L0);
hold on;
bodemag(L1);
hold on;
bodemag(L2);
hold on;
bodemag(L3);
hold on;
bodemag(L4);
hold on;
bodemag(L5);
hold off;
legend("K_{p}=0.001*(s+12)",...
       "K_{p}=0.002*(s+12)",...
       "K_{p}=0.0045*(s+12)",...
       "K_{p}=0.008*(s+12)",...
       "K_{p}=0.015*(s+12)",...
       "K_{p}=0.040*(s+12)");
title('Open Loop bodemag L');
grid on;
grid minor;
plot_axis;
%% 2 Plot Try (T)
T0=minreal(L0/(1+L0));
T1=minreal(L1/(1+L1));
T2=minreal(L2/(1+L2));
T3=minreal(L3/(1+L3));
T4=minreal(L4/(1+L4));
T5=minreal(L5/(1+L5));
```

```
figure(2);
bodemag(T0);
hold on;
bodemag(T1);
hold on;
bodemag(T2);
hold on;
bodemag(T3);
hold on;
bodemag(T4);
hold on;
bodemag(T5);
hold off;
legend("K_{p}=0.001*(s+12)",...
       "K_{p}=0.002*(s+12)",...
       "K_{p}=0.0045*(s+12)",...
       "K_{p}=0.008*(s+12)",...
       "K_{p}=0.015*(s+12)",...
       "K_{p}=0.040*(s+12)");
title('Complimentary Sensitivity, T_{ry}');
grid on;
grid minor;
plot_axis;
%% 3 Plot Tdoy (S)
S0=1/(1+L0);
S1=1/(1+L1);
S2=1/(1+L2);
S3=1/(1+L3);
S4=1/(1+L4);
S5=1/(1+L5);
figure(3);
bodemag(S0);
hold on;
bodemag(S1);
hold on;
bodemag(S2);
hold on;
bodemag(S3);
hold on;
bodemag(S4);
hold on;
bodemag(S5);
hold off;
legend("K_{p}=0.001*(s+12)",...
       "K_{p}=0.002*(s+12)",...
       "K_{p}=0.0045*(s+12)",...
       "K_{p}=0.008*(s+12)",...
       "K_{p}=0.015*(s+12)",...
       "K_{p}=0.040*(s+12)");
title('Sensitivity');
grid on;
grid minor;
plot_axis;
%% 4 Plot Tdiy (PS)
PS0=P/(1+L0);
PS1=P/(1+L1);
PS2=P/(1+L2);
```

```
PS3=P/(1+L3);
PS4=P/(1+L4);
PS5=P/(1+L5);
figure(4);
bodemag(PS0);
hold on;
bodemag(PS1);
hold on;
bodemag(PS2);
hold on;
bodemag(PS3);
hold on;
bodemag(PS4);
hold on;
bodemag(PS5);
hold off;
legend("K_{p}=0.001*(s+12)",...
       "K_{p}=0.002*(s+12)",...
       "K_{p}=0.0045*(s+12)",...
       "K_{p}=0.008*(s+12)",...
       "K_{p}=0.015*(s+12)",...
       "K_{p}=0.040*(s+12)");
title('PS');
grid on;
grid minor;
plot_axis;
%% 5 Plot Tny Tdiu
Tny0= -L0/(1+L0);
Tny1= -L1/(1+L1);
Tny2= -L2/(1+L2);
Tny3= -L3/(1+L3);
Tny4= -L4/(1+L4);
Tny5= -L5/(1+L5);
figure(5);
bodemag(Tny0);
hold on;
bodemag(Tny1);
hold on;
bodemag(Tny2);
hold on;
bodemag(Tny3);
hold on;
bodemag(Tny4);
hold on;
bodemag(Tny5);
hold off;
legend("K_{p}=0.001*(s+12)",...
       "K_{p}=0.002*(s+12)",...
       "K_{p}=0.0045*(s+12)",...
       "K_{p}=0.008*(s+12)",...
       "K_{p}=0.015*(s+12)",...
       "K_{p}=0.040*(s+12)");
title('T_{n y}');
grid on;
grid minor;
plot_axis;
%% 6 Plot Tru (KS)
```

```
Tru0= K0/(1+L0);
Tru1= K1/(1+L1);
Tru2= K2/(1+L2);
Tru3= K3/(1+L3);
Tru4= K4/(1+L4);
Tru5= K5/(1+L5);

figure(6);
bodemag(Tru0);
hold on;
bodemag(Tru1);
hold on;
bodemag(Tru2);
hold on;
bodemag(Tru3);
hold on;
bodemag(Tru4);
hold on;
bodemag(Tru5);
hold off;
legend("K_{p}=0.001*(s+12)",...
       "K_{p}=0.002*(s+12)",...
       "K_{p}=0.0045*(s+12)",...
       "K_{p}=0.008*(s+12)",...
       "K_{p}=0.015*(s+12)",...
       "K_{p}=0.040*(s+12)");
title('T_{r u} KS');
grid on;
grid minor;
plot_axis;
%% 7 Plot Tdou Tnu
Tdou0= -K0/(1+L0);
Tdou1= -K1/(1+L1);
Tdou2= -K2/(1+L2);
Tdou3= -K3/(1+L3);
Tdou4= -K4/(1+L4);
Tdou5= -K5/(1+L5);
figure(7);
bodemag(Tdou0);
hold on;
bodemag(Tdou1);
hold on;
bodemag(Tdou2);
hold on;
bodemag(Tdou3);
hold on;
bodemag(Tdou4);
hold on;
bodemag(Tdou5);
hold off;
legend("K_{p}=0.001*(s+12)",...
       "K_{p}=0.002*(s+12)",...
       "K_{p}=0.0045*(s+12)",...
       "K_{p}=0.008*(s+12)",...
       "K_{p}=0.015*(s+12)",...
       "K_{p}=0.040*(s+12)");
title('T_{do u} T_{n u}');
```

```
grid on;
grid minor;
plot_axis;
%% Plot Step Response
figure(8);
st0=stepplot(T0,t);
hold on;
st1=stepplot(T1,t);
hold on;
st2=stepplot(T2,t);
hold on;
st3=stepplot(T3,t);
hold on;
st4=stepplot(T4,t);
hold on;
st5=stepplot(T5,t);
hold off;
legend("K_{p}=0.001*(s+12)",...
       "K_{p}=0.002*(s+12)",...
       "K_{p}=0.0045*(s+12)",...
       "K_{p}=0.008*(s+12)",...
       "K_{p}=0.015*(s+12)",...
       "K_{p}=0.040*(s+12)");
ylabel('Angular Rate q (rad/s)');
grid on;
grid minor;
plot_axis;
%% Open Loop Root Locus
figure(10);
rlocus(T0);
grid on;
grid minor;
legend("K0=0.001*(s+12)");
figure(11);
rlocus(L1);
grid on;
grid minor;
legend("K1=0.002*(s+12)");
figure(12);
rlocus(T2);
grid on;
grid minor;
legend("Kp=0.0045*(s+12)");
figure(13);
rlocus(L3);
grid on;
grid minor;
legend("K3=0.008*(s+12)");
figure(14);
rlocus(L4);
grid on;
grid minor;
legend("K4=0.015*(s+12)");
figure(15);
rlocus(L5);
grid on;
grid minor;
```

```
legend("K5=0.040*(s+12)");
```

Attitude Control Simulation File

```
close all;
clc;
clear;
%----------Attitude Control----------%
%% Parameters
t=0:0.01:2;
J_y = 0.0019536;
a = 9.79;
s=tf('s');
P2=1 * a /((J_y*s)*(s+a));
K2=0.0045*(s+12);
P = minreal(P2*K2/(1+P2*K2)/s);
%% Controller
K0=3;
K1=6;
K2=9;
K3=12;
K4=15;
K5=20;
%% 1 Plot L = PK
L0=P*K0;
L1=P*K1;
L2=P*K2;
L3=P*K3;
L4=P*K4;
L5=P*K5;
figure(1);
bodemag(L0);
hold on;
bodemag(L1);
hold on;
bodemag(L2);
hold on;
bodemag(L3);
hold on;
bodemag(L4);
hold on;
bodemag(L5);
hold off;
legend("K\phi=3",...
       "K\phi=6",...
       "K\phi=9",...
       "K\phi=12",...
       "K\phi=15",...
       "K\phi=20");
title('Open Loop bodemag L');
grid on;
grid minor;
plot_axis;
%% 2 Plot Try (T)
T0=minreal(L0/(1+L0));
T1=minreal(L1/(1+L1));
```

```
T2=minreal(L2/(1+L2));
T3=minreal(L3/(1+L3));
T4=minreal(L4/(1+L4));
T5=minreal(L5/(1+L5));
figure(2);
bodemag(T0);
hold on;
bodemag(T1);
hold on;
bodemag(T2);
hold on;
bodemag(T3);
hold on;
bodemag(T4);
hold on;
bodemag(T5);
hold off;
legend("K\phi=3",...
       "K\phi=6",...
       "K\phi=9",...
       "K\phi=12",...
       "K\phi=15",...
       "K\phi=20");
title('Complimentary Sensitivity, T_{ry}');
grid on;
grid minor;
plot_axis;
%% 3 Plot Tdoy (S)
S0=1/(1+L0);
S1=1/(1+L1);
S2=1/(1+L2);
S3=1/(1+L3);
S4=1/(1+L4);
S5=1/(1+L5);
figure(3);
bodemag(S0);
hold on;
bodemag(S1);
hold on;
bodemag(S2);
hold on;
bodemag(S3);
hold on;
bodemag(S4);
hold on;
bodemag(S5);
hold off;
legend("K\phi=3",...
       "K\phi=6",...
       "K\phi=9",...
       "K\phi=12",...
       "K\phi=15",...
       "K\phi=20");
title('Sensitivity, T_{do y}');
grid on;
grid minor;
plot_axis;
```

```
%% 4 Plot Tdiy (PS)
PS0=P/(1+L0);
PS1=P/(1+L1);
PS2=P/(1+L2);
PS3=P/(1+L3);
PS4=P/(1+L4);
PS5=P/(1+L5);
figure(4);
bodemag(PS0);
hold on;
bodemag(PS1);
hold on;
bodemag(PS2);
hold on;
bodemag(PS3);
hold on;
bodemag(PS4);
hold on;
bodemag(PS5);
hold off;
legend("K\phi=3",...
       "K\phi=6",...
       "K\phi=9",...
       "K\phi=12",...
       "K\phi=15",...
       "K\phi=20");
title('PS, T_{di y}');
grid on;
grid minor;
plot_axis;
%% 5 Plot Tny
Tny0= -L0/(1+L0);
Tny1= -L1/(1+L1);
Tny2= -L2/(1+L2);
Tny3= -L3/(1+L3);
Tny4= -L4/(1+L4);
Tny5= -L5/(1+L5);
figure(5);
bodemag(Tny0);
hold on;
bodemag(Tny1);
hold on;
bodemag(Tny2);
hold on;
bodemag(Tny3);
hold on;
bodemag(Tny4);
hold on;
bodemag(Tny5);
hold off;
legend("K\phi=3",...
       "K\phi=6",...
       "K\phi=9",...
       "K\phi=12",...
       "K\phi=15",...
       "K\phi=20");
title('T_{n y}');
```

```
grid on;
grid minor;
plot_axis;
%% 6 Plot Tru (KS)
Tru0= K0/(1+L0);
Tru1= K1/(1+L1);
Tru2= K2/(1+L2);
Tru3= K3/(1+L3);
Tru4= K4/(1+L4);
Tru5= K5/(1+L5);
figure(6);
bodemag(Tru0);
hold on;
bodemag(Tru1);
hold on;
bodemag(Tru2);
hold on;
bodemag(Tru3);
hold on;
bodemag(Tru4);
hold on;
bodemag(Tru5);
hold off;
legend("K\phi=3",...
       "K\phi=6",...
       "K\phi=9",...
       "K\phi=12",...
       "K\phi=15",...
       "K\phi=20");
title('T_{r u}');
grid on;
grid minor;
plot_axis;
%% 7 Plot Tdou Tnu
Tdou0= -K0/(1+L0);
Tdou1= -K1/(1+L1);
Tdou2= -K2/(1+L2);
Tdou3= -K3/(1+L3);
Tdou4= -K4/(1+L4);
Tdou5= -K5/(1+L5);
figure(7);
bodemag(Tdou0);
hold on;
bodemag(Tdou1);
hold on;
bodemag(Tdou2);
hold on;
bodemag(Tdou3);
hold on;
bodemag(Tdou4);
hold on;
bodemag(Tdou5);
hold off;
legend("K\phi=3",...
       "K\phi=6",...
       "K\phi=9",...
       "K\phi=12",...
```

```
       "K\phi=15",...
       "K\phi=20");
title('T_{do u} T_{n u}');
grid on;
grid minor;
plot_axis;
%% Plot Step Response
figure(8);
st0=stepplot(T0,t);
hold on;
st1=stepplot(T1,t);
hold on;
st2=stepplot(T2,t);
hold on;
st3=stepplot(T3,t);
hold on;
st4=stepplot(T4,t);
hold on;
st5=stepplot(T5,t);
hold off;
legend("K\phi=3",...
       "K\phi=6",...
       "K\phi=9",...
       "K\phi=12",...
       "K\phi=15",...
       "K\phi=20");
ylabel('Attitude \phi (rad)');
grid on;
grid minor;
plot_axis;
%% Close Loop Pole
figure(10);
rlocus(L0);
grid on;
grid minor;
legend("K\phi=3");
figure(11);
rlocus(L1);
grid on;
grid minor;
legend("K\phi=6");
figure(12);
rlocus(T2);
grid on;
grid minor;
legend("K\phi=9");
figure(13);
rlocus(L3);
grid on;
grid minor;
legend("K\phi=12");
figure(14);
rlocus(L4);
grid on;
grid minor;
legend("K\phi=15");
figure(15);
```

```
rlocus(L5);
grid on;
grid minor;
legend("K\phi=20");
```

Vertical Velocity Control Simulation File

```
close all;
clc;
clear;
%----------Vertical Velocity Control----------%
%% Parameters
t=0:0.01:2;
m = 0.645;
a = 9.79;
s=tf('s');
P=1 * a /((m*s)*(s+a));
%% Controller
K0=0.050*(s+12);
K1=0.100*(s+12);
K2=0.2*(s+12);
K3=0.525*(s+12);
K4=1*(s+12);
K5=2*(s+12);
%% 1 Plot L = PK
L0=P*K0;
L1=P*K1;
L2=P*K2;
L3=P*K3;
L4=P*K4;
L5=P*K5;
figure(1);
bodemag(L0);
hold on;
bodemag(L1);
hold on;
bodemag(L2);
hold on;
bodemag(L3);
hold on;
bodemag(L4);
hold on;
bodemag(L5);
hold off;
legend("K0=7.000 + 0.050*s",...
       "K1=7.000 + 0.200*s",...
       "K2=7.000 + 0.525*s",...
       "K3=7.000 + 0.800*s",...
       "K4=7.000 + 1.000*s",...
       "K5=7.000 + 2.000*s");
title('Open Loop bodemag L');
grid on;
grid minor;
plot_axis;
%% 2 Plot Try (T)
T0=L0/(1+L0);
```

```
T1=L1/(1+L1);
T2=L2/(1+L2);
T3=L3/(1+L3);
T4=L4/(1+L4);
T5=L5/(1+L5);
figure(2);
bodemag(T0);
hold on;
bodemag(T1);
hold on;
bodemag(T2);
hold on;
bodemag(T3);
hold on;
bodemag(T4);
hold on;
bodemag(T5);
hold off;
legend("K0=7.000 + 0.050*s",...
       "K1=7.000 + 0.200*s",...
       "K2=7.000 + 0.525*s",...
       "K3=7.000 + 0.800*s",...
       "K4=7.000 + 1.000*s",...
       "K5=7.000 + 2.000*s");
grid on;
title('Complimentary Sensitivity, T_{ry}');
grid minor;
plot_axis;
%% 3 Plot Tdoy (S)
S0=1/(1+L0);
S1=1/(1+L1);
S2=1/(1+L2);
S3=1/(1+L3);
S4=1/(1+L4);
S5=1/(1+L5);
figure(3);
bodemag(S0);
hold on;
bodemag(S1);
hold on;
bodemag(S2);
hold on;
bodemag(S3);
hold on;
bodemag(S4);
hold on;
bodemag(S5);
hold off;
legend("K0=7.000 + 0.050*s",...
       "K1=7.000 + 0.200*s",...
       "K2=7.000 + 0.525*s",...
       "K3=7.000 + 0.800*s",...
       "K4=7.000 + 1.000*s",...
       "K5=7.000 + 2.000*s");
title('Sensitivity, T_{do y}');
grid minor;
plot_axis;
```

```
%% 4 Plot Tdiy (PS)
PS0=P/(1+L0);
PS1=P/(1+L1);
PS2=P/(1+L2);
PS3=P/(1+L3);
PS4=P/(1+L4);
PS5=P/(1+L5);
figure(4);
bodemag(PS0);
hold on;
bodemag(PS1);
hold on;
bodemag(PS2);
hold on;
bodemag(PS3);
hold on;
bodemag(PS4);
hold on;
bodemag(PS5);
hold off;
legend("K0=7.000 + 0.050*s",...
       "K1=7.000 + 0.200*s",...
       "K2=7.000 + 0.525*s",...
       "K3=7.000 + 0.800*s",...
       "K4=7.000 + 1.000*s",...
       "K5=7.000 + 2.000*s");
title('PS, T_{di y}');
grid minor;
plot_axis;
%% 5 Plot Tny
Tny0= -L0/(1+L0);
Tny1= -L1/(1+L1);
Tny2= -L2/(1+L2);
Tny3= -L3/(1+L3);
Tny4= -L4/(1+L4);
Tny5= -L5/(1+L5);
figure(5);
bodemag(Tny0);
hold on;
bodemag(Tny1);
hold on;
bodemag(Tny2);
hold on;
bodemag(Tny3);
hold on;
bodemag(Tny4);
hold on;
bodemag(Tny5);
hold off;
legend("K0=7.000 + 0.050*s",...
       "K1=7.000 + 0.200*s",...
       "K2=7.000 + 0.525*s",...
       "K3=7.000 + 0.800*s",...
       "K4=7.000 + 1.000*s",...
       "K5=7.000 + 2.000*s");
title('T_{n y}');
grid minor;
```

```
plot_axis;
%% 6 Plot Tru (KS)
Tru0= K0/(1+L0);
Tru1= K1/(1+L1);
Tru2= K2/(1+L2);
Tru3= K3/(1+L3);
Tru4= K4/(1+L4);
Tru5= K5/(1+L5);
figure(6);
bodemag(Tru0);
hold on;
bodemag(Tru1);
hold on;
bodemag(Tru2);
hold on;
bodemag(Tru3);
hold on;
bodemag(Tru4);
hold on;
bodemag(Tru5);
hold off;
legend("K0=7.000 + 0.050*s",...
       "K1=7.000 + 0.200*s",...
       "K2=7.000 + 0.525*s",...
       "K3=7.000 + 0.800*s",...
       "K4=7.000 + 1.000*s",...
       "K5=7.000 + 2.000*s");
title('T_{r u}');
grid minor;
plot_axis;
%% 7 Plot Tdou Tnu
Tdou0= -K0/(1+L0);
Tdou1= -K1/(1+L1);
Tdou2= -K2/(1+L2);
Tdou3= -K3/(1+L3);
Tdou4= -K4/(1+L4);
Tdou5= -K5/(1+L5);
figure(7);
bodemag(Tdou0);
hold on;
bodemag(Tdou1);
hold on;
bodemag(Tdou2);
hold on;
bodemag(Tdou3);
hold on;
bodemag(Tdou4);
hold on;
bodemag(Tdou5);
hold off;
legend("K0=7.000 + 0.050*s",...
       "K1=7.000 + 0.200*s",...
       "K2=7.000 + 0.525*s",...
       "K3=7.000 + 0.800*s",...
       "K4=7.000 + 1.000*s",...
       "K5=7.000 + 2.000*s");
title('T_{do u} T_{n u}');
```

```
grid minor;
plot_axis;
%% Plot Step Response
figure(8);
st0=stepplot(T0,t);
hold on;
st1=stepplot(T1,t);
hold on;
st2=stepplot(T2,t);
hold on;
st3=stepplot(T3,t);
hold on;
st4=stepplot(T4,t);
hold on;
st5=stepplot(T5,t);
hold off;
legend("K0=7.000 + 0.050*s",...
       "K1=7.000 + 0.200*s",...
       "K2=7.000 + 0.525*s",...
       "K3=7.000 + 0.800*s",...
       "K4=7.000 + 1.000*s",...
       "K5=7.000 + 2.000*s");
ylabel('Angular Rate q (rad/s)');
grid minor;
plot_axis;
%% Close Loop Pole
figure(10);
rlocus(T0);
grid on;
grid minor;
legend("K0=7.000 + 0.050*s");
figure(11);
rlocus(T1);
grid on;
grid minor;
legend("K1=7.000 + 0.200*s");
figure(12);
rlocus(T2);
grid on;
grid minor;
legend("K2=7.000 + 0.525*s");
figure(13);
rlocus(T3);
grid on;
grid minor;
legend("K3=7.000 + 0.800*s");
figure(14);
rlocus(T4);
grid on;
grid minor;
legend("K4=7.000 + 1.000*s");
figure(15);
rlocus(T5);
grid on;
grid minor;
legend("K5=7.000 + 2.000*s");
```

Altitude Control Simulation File

```
close all;
clc;
clear;
%----------Altitude Control----------%
%% Parameters
t=0:0.01:2;
m = 0.645;
a = 9.79;
s=tf('s');
P_in=1 * a /((m*s)*(s+a));
K_in=0.525*(s+12);
L_in=P_in*K_in;
P=minreal(L_in/(1+L_in)/s);
%% Controller
K0=0.500;
K1=1.000;
K2=2.000;
K3=4.200;
K4=10.000;
K5=20.000;
%% 1 Plot L = PK
L0=P*K0;
L1=P*K1;
L2=P*K2;
L3=P*K3;
L4=P*K4;
L5=P*K5;
figure(1);
bodemag(L0);
hold on;
bodemag(L1);
hold on;
bodemag(L2);
hold on;
bodemag(L3);
hold on;
bodemag(L4);
hold on;
bodemag(L5);
hold off;
legend("K0=0.5",...
        "K1=1",...
        "K2=3",...
        "K3=5",...
        "K4=10",...
        "K5=20");
title('Open Loop bodemag L');
grid on;
grid minor;
plot_axis;
%% 2 Plot Try (T)
T0=minreal(L0/(1+L0));
T1=minreal(L1/(1+L1));
T2=minreal(L2/(1+L2));
```

```
T3=minreal(L3/(1+L3));
T4=minreal(L4/(1+L4));
T5=minreal(L5/(1+L5));
figure(2);
bodemag(T0);
hold on;
bodemag(T1);
hold on;
bodemag(T2);
hold on;
bodemag(T3);
hold on;
bodemag(T4);
hold on;
bodemag(T5);
hold off;legend("K0=0.5",...
        "K1=1",...
        "K2=3",...
        "K3=5",...
        "K4=10",...
        "K5=20");
title('Complimentary Sensitivity, T_{ry}');
grid minor;
plot_axis;
%% 3 Plot Tdoy (S)
S0=1/(1+L0);
S1=1/(1+L1);
S2=1/(1+L2);
S3=1/(1+L3);
S4=1/(1+L4);
S5=1/(1+L5);
figure(3);
bodemag(S0);
hold on;
bodemag(S1);
hold on;
bodemag(S2);
hold on;
bodemag(S3);
hold on;
bodemag(S4);
hold on;
bodemag(S5);
hold off;legend("K0=0.5",...
        "K1=1",...
        "K2=3",...
        "K3=5",...
        "K4=10",...
        "K5=20");
title('Sensitivity, T_{do y}');
grid minor;
plot_axis;
%% 4 Plot Tdiy (PS)
PS0=P/(1+L0);
PS1=P/(1+L1);
PS2=P/(1+L2);
PS3=P/(1+L3);
```

```
PS4=P/(1+L4);
PS5=P/(1+L5);
figure(4);
bodemag(PS0);
hold on;
bodemag(PS1);
hold on;
bodemag(PS2);
hold on;
bodemag(PS3);
hold on;
bodemag(PS4);
hold on;
bodemag(PS5);
hold off;legend("K0=0.5",...
        "K1=1",...
        "K2=3",...
        "K3=5",...
        "K4=10",...
        "K5=20");
title('PS, T_{di y}');
grid minor;
plot_axis;
%% 5 Plot Tny Tdiu
Tny0= -L0/(1+L0);
Tny1= -L1/(1+L1);
Tny2= -L2/(1+L2);
Tny3= -L3/(1+L3);
Tny4= -L4/(1+L4);
Tny5= -L5/(1+L5);
figure(5);
bodemag(Tny0);
hold on;
bodemag(Tny1);
hold on;
bodemag(Tny2);
hold on;
bodemag(Tny3);
hold on;
bodemag(Tny4);
hold on;
bodemag(Tny5);
hold off;
legend("K0=0.5",...
        "K1=1",...
        "K2=3",...
        "K3=5",...
        "K4=10",...
        "K5=20");
title('T_{n y} T_{di u}');
grid minor;
plot_axis;
%% 6 Plot Tru (KS)
Tru0= K0/(1+L0);
Tru1= K1/(1+L1);
Tru2= K2/(1+L2);
Tru3= K3/(1+L3);
```

```
Tru4= K4/(1+L4);
Tru5= K5/(1+L5);
figure(6);
bodemag(Tru0);
hold on;
bodemag(Tru1);
hold on;
bodemag(Tru2);
hold on;
bodemag(Tru3);
hold on;
bodemag(Tru4);
hold on;
bodemag(Tru5);
hold off;
legend("K0=0.5",...
        "K1=1",...
        "K2=3",...
        "K3=5",...
        "K4=10",...
        "K5=20");
title('T_{r u}');
grid minor;
plot_axis;
%% 7 Plot Tdou Tnu
Tdou0= -K0/(1+L0);
Tdou1= -K1/(1+L1);
Tdou2= -K2/(1+L2);
Tdou3= -K3/(1+L3);
Tdou4= -K4/(1+L4);
Tdou5= -K5/(1+L5);
figure(7);
bodemag(Tdou0);
hold on;
bodemag(Tdou1);
hold on;
bodemag(Tdou2);
hold on;
bodemag(Tdou3);
hold on;
bodemag(Tdou4);
hold on;
bodemag(Tdou5);
hold off;
legend("K0=0.5",...
        "K1=1",...
        "K2=3",...
        "K3=5",...
        "K4=10",...
        "K5=20");
title('T_{do u} T_{n u}');
grid minor;
plot_axis;
%% Plot Step Response
figure(8);
st0=stepplot(T0,t);
hold on;
```

```
st1=stepplot(T1,t);
hold on;
st2=stepplot(T2,t);
hold on;
st3=stepplot(T3,t);
hold on;
st4=stepplot(T4,t);
hold on;
st5=stepplot(T5,t);
hold off;
legend("K0=0.5",...
        "K1=1",...
        "K2=3",...
        "K3=5",...
        "K4=10",...
        "K5=20");
ylabel('Altitude (m)');
grid minor;
plot_axis;
%% Close Loop Pole
figure(10);
rlocus(L0);
grid on;
grid minor;
legend("K0=0.5");
figure(11);
rlocus(L1);
grid on;
grid minor;
legend("K1=1");
figure(12);
rlocus(L2);
grid on;
grid minor;
legend("K2=3");
figure(13);
rlocus(L3);
grid on;
grid minor;
legend("Kz=4.2");
figure(14);
rlocus(L4);
grid on;
grid minor;
legend("K4=10");
figure(15);
rlocus(L5);
grid on;
grid minor;
legend("K5=20");
```

Translational Control Simulation File

```
close all;
clc;
clear;
```

```
%---------Translational Velocity Conrol---------%
%% Parameters
t=0:0.01:2;
g = 9.8;
s=tf('s');
J_y = 0.0019536;
a = 9.79;
P_2=1 * a /((J_y*s)*(s+a));
K_2=0.0045*(s+12);
P_3 = minreal(P_2*K_2/(1+P_2*K_2)/s);
K_3 = 9;
L_2=P_3*K_3;
T_2=minreal(L_2/(1+L_2));
P = minreal(T_2 *g/s);
%% Controller
K0=0.02*(s+10);
K1=0.05*(s+10);
K2=0.075*(s+10);
K3=0.10*(s+10);
K4=0.15*(s+10);
K5=0.20*(s+10);
%% 1 Plot L = PK
L0=minreal(P*K0);
L1=minreal(P*K1);
L2=minreal(P*K2);
L3=minreal(P*K3);
L4=minreal(P*K4);
L5=minreal(P*K5);
figure(1);
bodemag(L0);
hold on;
bodemag(L1);
hold on;
bodemag(L2);
hold on;
bodemag(L3);
hold on;
bodemag(L4);
hold on;
bodemag(L5);
hold off;
legend("K0=1.3+0.02*s",...
       "K1=1.3+0.05*s",...
       "K2=1.3+0.08*s",...
       "K3=1.3+0.10*s",...
       "K4=1.3+0.15*s",...
       "K5=1.3+0.20*s");
title('Open Loop bodemag L');
grid on;
grid minor;
plot_axis;
%% 2 Plot Try (T)
T0=minreal(L0/(1+L0));
T1=minreal(L1/(1+L1));
T2=minreal(L2/(1+L2));
T3=minreal(L3/(1+L3));
T4=minreal(L4/(1+L4));
```

```
T5=minreal(L5/(1+L5));
figure(2);
bodemag(T0);
hold on;
bodemag(T1);
hold on;
bodemag(T2);
hold on;
bodemag(T3);
hold on;
bodemag(T4);
hold on;
bodemag(T5);
hold off;
legend("K0=1.3+0.02*s",...
       "K1=1.3+0.05*s",...
       "K2=1.3+0.08*s",...
       "K3=1.3+0.10*s",...
       "K4=1.3+0.15*s",...
       "K5=1.3+0.20*s");
title('Complimentary Sensitivity, T_{ry}');
grid on;
grid minor;
plot_axis;
%% 3 Plot Tdoy (S)
S0=1/(1+L0);
S1=1/(1+L1);
S2=1/(1+L2);
S3=1/(1+L3);
S4=1/(1+L4);
S5=1/(1+L5);
figure(3);
bodemag(S0);
hold on;
bodemag(S1);
hold on;
bodemag(S2);
hold on;
bodemag(S3);
hold on;
bodemag(S4);
hold on;
bodemag(S5);
hold off;
legend("K0=1.3+0.02*s",...
       "K1=1.3+0.05*s",...
       "K2=1.3+0.08*s",...
       "K3=1.3+0.10*s",...
       "K4=1.3+0.15*s",...
       "K5=1.3+0.20*s");
title('Sensitivity, T_{do y}');
grid on;
grid minor;
plot_axis;
%% 4 Plot Tdiy (PS)
PS0=P/(1+L0);
PS1=P/(1+L1);
```

```
PS2=P/(1+L2);
PS3=P/(1+L3);
PS4=P/(1+L4);
PS5=P/(1+L5);
figure(4);
bodemag(PS0);
hold on;
bodemag(PS1);
hold on;
bodemag(PS2);
hold on;
bodemag(PS3);
hold on;
bodemag(PS4);
hold on;
bodemag(PS5);
hold off;
legend("K0=1.3+0.02*s",...
       "K1=1.3+0.05*s",...
       "K2=1.3+0.08*s",...
       "K3=1.3+0.10*s",...
       "K4=1.3+0.15*s",...
       "K5=1.3+0.20*s");
title('PS, T_{di y}');
grid on;
grid minor;
plot_axis;
%% 5 Plot Tny
Tny0= -L0/(1+L0);
Tny1= -L1/(1+L1);
Tny2= -L2/(1+L2);
Tny3= -L3/(1+L3);
Tny4= -L4/(1+L4);
Tny5= -L5/(1+L5);
figure(5);
bodemag(Tny0);
hold on;
bodemag(Tny1);
hold on;
bodemag(Tny2);
hold on;
bodemag(Tny3);
hold on;
bodemag(Tny4);
hold on;
bodemag(Tny5);
hold off;
legend("K0=1.3+0.02*s",...
       "K1=1.3+0.05*s",...
       "K2=1.3+0.08*s",...
       "K3=1.3+0.10*s",...
       "K4=1.3+0.15*s",...
       "K5=1.3+0.20*s");
title('T_{n y}');
grid on;
grid minor;
plot_axis;
```

```
%% 6 Plot Tru (KS)
Tru0= K0/(1+L0);
Tru1= K1/(1+L1);
Tru2= K2/(1+L2);
Tru3= K3/(1+L3);
Tru4= K4/(1+L4);
Tru5= K5/(1+L5);
figure(6);
bodemag(Tru0);
hold on;
bodemag(Tru1);
hold on;
bodemag(Tru2);
hold on;
bodemag(Tru3);
hold on;
bodemag(Tru4);
hold on;
bodemag(Tru5);
legend("K0=1.3+0.02*s",...
       "K1=1.3+0.05*s",...
       "K2=1.3+0.08*s",...
       "K3=1.3+0.10*s",...
       "K4=1.3+0.15*s",...
       "K5=1.3+0.20*s");
title('T_{r u}');
grid on;
grid minor;
plot_axis;
%% 7 Plot Tdou Tnu
Tdou0= -K0/(1+L0);
Tdou1= -K1/(1+L1);
Tdou2= -K2/(1+L2);
Tdou3= -K3/(1+L3);
Tdou4= -K4/(1+L4);
Tdou5= -K5/(1+L5);
figure(7);
bodemag(Tdou0);
hold on;
bodemag(Tdou1);
hold on;
bodemag(Tdou2);
hold on;
bodemag(Tdou3);
hold on;
bodemag(Tdou4);
hold on;
bodemag(Tdou5);
hold off;
legend("K0=1.3+0.02*s",...
       "K1=1.3+0.05*s",...
       "K2=1.3+0.08*s",...
       "K3=1.3+0.10*s",...
       "K4=1.3+0.15*s",...
       "K5=1.3+0.20*s");
title('T_{do u} T_{n u}');
grid on;
```

```
grid minor;
plot_axis;
%% Plot Step Response
figure(8);
st0=stepplot(T0,t);
hold on;
st1=stepplot(T1,t);
hold on;
st2=stepplot(T2,t);
hold on;
st3=stepplot(T3,t);
hold on;
st4=stepplot(T4,t);
hold on;
st5=stepplot(T5,t);
hold off;
legend("K0=1.3+0.02*s",...
       "K1=1.3+0.05*s",...
       "K2=1.3+0.08*s",...
       "K3=1.3+0.10*s",...
       "K4=1.3+0.15*s",...
       "K5=1.3+0.20*s");
ylabel('Translational Velocity Vy (m/s)');
grid on;
grid minor;
plot_axis;

%% Close Loop Pole
figure(10);
rlocus(T0);
grid on;
grid minor;
legend("K0=1.3+0.01*s");
figure(11);
rlocus(T1);
grid on;
grid minor;
legend("K1=1.3+0.02*s");
figure(12);
rlocus(L2);
grid on;
grid minor;
legend("Kvy=0.075*(s+10)s");
figure(13);
rlocus(T3);
grid on;
grid minor;
legend("K3=1.3+0.10*s");
figure(14);
rlocus(T4);
grid on;
grid minor;
legend("Kvy=1.3+0.20*s");
figure(15);
rlocus(T5);
grid on;
grid minor;
```

```
legend("Kvy=1.3+0.50*s");
```

Translational Position Simulation File

```
clc;
clear;
close all;
%----------Translational Position Control----------%
%% Parameters
t=0:0.01:10;
s=tf('s');
%% Parameters
T2_vy = (149.2*s^2 + 3282*s + 1.79e04)/...
        (s^4+32.34*s^3+...
         622.7*s^2+5717*s+1.79e04);
P = minreal(T2_vy / s);
K0=0.2;
K1=0.4;
K2=0.8;
K3=1.0;
K4=2.0;
K5=4.0;
%% 1 Plot L = PK
L0=minreal(P*K0);
L1=minreal(P*K1);
L2=minreal(P*K2);
L3=minreal(P*K3);
L4=minreal(P*K4);
L5=minreal(P*K5);
figure(1);
bodemag(L0);
hold on;
bodemag(L1);
hold on;
bodemag(L2);
hold on;
bodemag(L3);
hold on;
bodemag(L4);
hold on;
bodemag(L5);
hold off;
legend("K0=0.2",...
       "K1=0.4",...
       "K2=0.8",...
       "K3=1.0",...
       "K4=2.0",...
       "K5=4.0");
title('Open Loop bodemag L');
grid on;
grid minor;
plot_axis;
%% 2 Plot Try (T)
T0=minreal(L0/(1+L0));
T1=minreal(L1/(1+L1));
T2=minreal(L2/(1+L2));
```

```
T3=minreal(L3/(1+L3));
T4=minreal(L4/(1+L4));
T5=minreal(L5/(1+L5));
figure(2);
bodemag(T0);
hold on;
bodemag(T1);
hold on;
bodemag(T2);
hold on;
bodemag(T3);
hold on;
bodemag(T4);
hold on;
bodemag(T5);
hold off;
title('Complimentary Sensitivity, T_{r y}');
legend("K0=0.2",...
       "K1=0.4",...
       "K2=0.8",...
       "K3=1.0",...
       "K4=2.0",...
       "K5=4.0");
grid on;
grid minor;
plot_axis;
%% Plot Step Response
figure(9);
st0=stepplot(T0,t);
hold on;
st1=stepplot(T1,t);
hold on;
st2=stepplot(T2,t);
hold on;
st3=stepplot(T3,t);
hold on;
st4=stepplot(T4,t);
hold on;
st5=stepplot(T5,t);
hold off;
legend("K0=0.2",...
       "K1=0.4",...
       "K2=0.8",...
       "K3=1.0",...
       "K4=2.0",...
       "K5=4.0");
ylabel('Angular Rate q (rad/s)');
grid on;
grid minor;
plot_axis;
```

APPENDIX B

MATLAB BASED GUI

Mark3 Ground Station

```
function varargout = Mark3_GUI(varargin)
gui_Singleton = 1;
gui_State = struct('gui_Name', mfilename, ...
 'gui_Singleton',  gui_Singleton, ...
 'gui_OpeningFcn', @Mark3_GUI_OpeningFcn, ...
 'gui_OutputFcn',  @Mark3_GUI_OutputFcn, ...
 'gui_LayoutFcn',  [] , ...
 'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end

function Mark3_GUI_OpeningFcn(hObject, eventdata, handles, varargin)

%fclose(instrfindall);
global udp_timer;
global udp_timer_2;
global drone;
drone(1).command_count = 50;
drone(1).command = 1234;
drone(1).des_x = 0;
drone(1).des_y = 0.75;
drone(1).des_z = 0.60;
drone(1).des_psi = 0;
drone(1).polar_psi = 0;

strRec = '';
setappdata(hObject, 'strRec', strRec);
udp_timer = 5;
udp_timer_2 = 10;

handles.fid = fopen('roll.txt','a');
handles.output = hObject;
% Update handles structure
guidata(hObject, handles);
% UIWAIT makes Mark3_GUI wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = Mark3_GUI_OutputFcn(hObject, eventdata, handles)

varargout{1} = handles.output;

function Open_Serial_Callback(hObject, eventdata, handles)
handles.xbee_pack_drone1=serial('COM4');
set(handles.xbee_pack_drone1,'BaudRate',230400,'Parity','none',...
    'DataBits',8,'StopBits',1,...
```

```
    'Timeout', 0.01,...
    'InputBufferSize',2048,...
    'OutputBufferSize',10240,...
    'BytesAvailableFcnMode','byte',...
    'BytesAvailableFcn',{@Xbee_bytes_drone1,handles});
fopen(handles.xbee_pack_drone1);
handles.timer = timer('Period',0.01,'ExecutionMode',...
'FixedRate', 'TimerFcn',{@xbee_pack_drone1_Send,handles});
start(handles.timer);
guidata(hObject, handles);

function Xbee_bytes_drone1(obj,eventdata,handles)
global udp_timer;
strRec = getappdata(handles.figure1, 'strRec');
check=get(obj,'BytesAvailable');
if check
    packet = fread(obj,check,'uchar');
    xbee_length = length(packet);
    if udp_timer == 5
  set(handles.xbee_pack_length_drone1,'string',num2str(xbee_length));
    end
    pack_string = char(packet');
    if get(handles.Write_Data,'value')
  fprintf(handles.fid,'%s',pack_string);
    end
    if udp_timer == 5
  if get(handles.data_disp,'value')
set(handles.Serial_Dis,'string',pack_string);
  end
    end
    setappdata(handles.figure1, 'strRec', strRec);
end

function Close_Serial_Callback(hObject, eventdata, handles)
stop(handles.timer);
delete(handles.timer);
stopasync(handles.xbee_pack_drone1);
fclose(handles.xbee_pack_drone1);
delete(handles.xbee_pack_drone1);
%fclose(instrfindall)
% --- Executes on button press in pushbutton1.
function Open_Udp_Callback(hObject, eventdata, handles)
% 184 is the buffer size
BUFSIZE = 184;
% 5400 is the port number
PORT = 5400;
handles.udpReceive=udp('127.0.0.1','LocalPort',PORT,...
    'InputBufferSize',BUFSIZE,'Timeout',Inf);
fopen(handles.udpReceive);
handles.udpReceive.ReadAsyncMode = 'continuous';
set(handles.udpReceive,'DatagramTerminateMode','on');
handles.udpReceive.DatagramReceivedFcn={@UDPdataProtocol,handles};
guidata(hObject, handles);


% --- Executes on button press in pushbutton2.
function Close_Udp_Callback(hObject, eventdata, handles)
```

```
stopasync(handles.udpReceive);
fclose(handles.udpReceive);
delete(handles.udpReceive);
%close();

function DataSend_Callback(hObject, eventdata, handles)
global drone;
str = get(hObject, 'string');
if length(str)==4
    drone(1).command = str2num(str);
end

function DataSend_CreateFcn(hObject, ~, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
    get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in Write_Data.
function Write_Data_Callback(hObject, eventdata, handles)
% --- Executes on button press in send_otus.
function Send_Otus_Callback(hObject, eventdata, handles)

function xbee_pack_drone1_Send(obj,event,handles)

global drone

drone(1).data_command = int16(drone(1).command);

if get(handles.checkbox_plan,'value')
    drone(1).polar_psi = drone(1).polar_psi + 0.4;
    if drone(1).polar_psi >= 180
  drone(1).polar_psi = -180;
    end
    drone(1).des_x = 0.75 * sin(drone(1).polar_psi*pi/180);
    drone(1).des_y = 0.75 * cos(drone(1).polar_psi*pi/180);
    drone(1).des_z = 0.6;
    drone(1).des_psi = -drone(1).polar_psi;
end

data_start = [uint8(111),uint8(121),uint8(131)];
data_x_check = [uint8(21),uint8(101)];
data_y_check = [uint8(22),uint8(102)];
data_z_check = [uint8(23),uint8(103)];
data_vx_check = [uint8(24),uint8(104)];
data_vy_check = [uint8(25),uint8(105)];
data_vz_check = [uint8(26),uint8(106)];
data_yaw_check = [uint8(27),uint8(107)];
data_pitch_check = [uint8(28),uint8(108)];
data_roll_check = [uint8(29),uint8(109)];

data_q0_check = [uint8(31),uint8(111)];
data_q1_check = [uint8(32),uint8(112)];
data_q2_check = [uint8(33),uint8(113)];
data_q3_check = [uint8(34),uint8(114)];

```

```
data_des_x_check = [uint8(35),uint8(115)];
data_des_y_check = [uint8(36),uint8(116)];
data_des_z_check = [uint8(37),uint8(117)];
data_des_psi_check = [uint8(38),uint8(118)];

data_command_check = [uint8(30),uint8(110)];

data_end = [uint8(50),uint8(75),uint8(100)];

drone(1).array_x=typecast(drone(1).data_x,'uint8');
drone(1).array_y=typecast(drone(1).data_y,'uint8');
drone(1).array_z=typecast(drone(1).data_z,'uint8');

drone(1).array_vx=typecast(drone(1).data_vx,'uint8');
drone(1).array_vy=typecast(drone(1).data_vy,'uint8');
drone(1).array_vz=typecast(drone(1).data_vz,'uint8');

drone(1).array_roll=typecast(drone(1).data_roll,'uint8');
drone(1).array_pitch=typecast(drone(1).data_pitch,'uint8');
drone(1).array_yaw=typecast(drone(1).data_yaw,'uint8');

drone(1).array_q0=typecast(drone(1).data_q0,'uint8');
drone(1).array_q1=typecast(drone(1).data_q1,'uint8');
drone(1).array_q2=typecast(drone(1).data_q2,'uint8');
drone(1).array_q3=typecast(drone(1).data_q3,'uint8');

data_des_x = single(drone(1).des_x);
data_des_y = single(drone(1).des_y);
data_des_z = single(drone(1).des_z);
data_des_psi = single(drone(1).des_psi);

drone(1).array_des_x=typecast(data_des_x,'uint8');
drone(1).array_des_y=typecast(data_des_y,'uint8');
drone(1).array_des_z=typecast(data_des_z,'uint8');
drone(1).array_des_psi=typecast(data_des_psi,'uint8');

drone(1).array_command=typecast(drone(1).data_command,'uint8');
%clc;

packet = [data_start ...
    data_x_check drone(1).array_x ...
    data_y_check drone(1).array_y ...
    data_z_check drone(1).array_z ...
    data_vx_check drone(1).array_vx ...
    data_vy_check drone(1).array_vy ...
    data_vz_check drone(1).array_vz ...
    data_yaw_check drone(1).array_yaw ...
    data_pitch_check drone(1).array_pitch ...
    data_roll_check drone(1).array_roll ...
    data_q0_check drone(1).array_q0 ...
    data_q1_check drone(1).array_q1 ...
    data_q2_check drone(1).array_q2 ...
    data_q3_check drone(1).array_q3 ...
    data_des_x_check drone(1).array_des_x ...
    data_des_y_check drone(1).array_des_y ...
    data_des_z_check drone(1).array_des_z ...
    data_des_psi_check drone(1).array_des_psi ...
```

```
    data_command_check drone(1).array_command ...
    data_end];

if get(handles.send_otus,'value')
    fwrite(handles.xbee_pack_drone1, packet, 'uchar');
end


% --- Executes on button press in pushbutton6.
function Exit_Callback(hObject, eventdata, handles)
fclose(handles.fid);
close();



function xbee_pack_drone1_Callback(hObject, eventdata, handles)

function xbee_pack_drone1_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),...
    get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in checkbox_plan.
function checkbox_plan_Callback(hObject, eventdata, handles)
```

UDP Protocol

```
function UDPdataProtocol(obj,event,handles)
byte_vector = fread(handles.udpReceive,1);
if mod(length(uint8(byte_vector)),8) == 0
double_vector =  typecast(uint8(byte_vector), 'double');
end
controller_id = strcat(native2unicode(byte_vector(1)),...
                       native2unicode(byte_vector(2)),...
                       native2unicode(byte_vector(3)),...
                       native2unicode(byte_vector(4)),...
                       native2unicode(byte_vector(5)),...
                       native2unicode(byte_vector(6)),...
                       native2unicode(byte_vector(7)),...
                       native2unicode(byte_vector(8)));

timestamp = double_vector(2);

lin_pos_x = double_vector(3);
lin_pos_y = double_vector(4);
lin_pos_z = double_vector(5);

lin_vel_x = double_vector(6);
lin_vel_y = double_vector(7);
lin_vel_z = double_vector(8);

lin_acc_x = double_vector(9);
lin_acc_y = double_vector(10);
```

```
lin_acc_z = double_vector(11);

quaternion_x = double_vector(12);
quaternion_y = double_vector(13);
quaternion_z = double_vector(14);
quaternion_w = double_vector(15);

ang_vel_x = double_vector(16);
ang_vel_y = double_vector(17);
ang_vel_y = double_vector(18);

ang_acc_x = double_vector(19);
ang_acc_y = double_vector(20);
ang_acc_z = double_vector(21);

button_1 = double_vector(22);
button_2 = double_vector(23);

global udp_timer;

global drone;

if controller_id == '67C87001'
    drone(1).data_x=single(-lin_pos_z);
    drone(1).data_y=single(-lin_pos_x);
    drone(1).data_z=single(-lin_pos_y);

    drone(1).data_vx=single(-lin_vel_z);
    drone(1).data_vy=single(-lin_vel_x);
    drone(1).data_vz=single(-lin_vel_y);

    drone(1).quat = [quaternion_w,...
                    -quaternion_z,...
                    -quaternion_x,...
                     quaternion_y];
    tmp_eul = quat2eul(drone(1).quat) * 57.29578;
    drone(1).data_yaw=single(tmp_eul(1));
    drone(1).data_pitch=single(tmp_eul(2));
    drone(1).data_roll=single(tmp_eul(3));

    drone(1).data_q0 = single(drone(1).quat(1));
    drone(1).data_q1 = single(drone(1).quat(2));
    drone(1).data_q2 = single(drone(1).quat(3));
    drone(1).data_q3 = single(drone(1).quat(4));
end

if udp_timer == 5
set(handles.data_show_x,'string',num2str(drone(1).data_x));
set(handles.data_show_y,'string',num2str(drone(1).data_y));
set(handles.data_show_z,'string',num2str(drone(1).data_z));

set(handles.data_show_vx,'string',num2str(drone(1).data_vx));
set(handles.data_show_vy,'string',num2str(drone(1).data_vy));
set(handles.data_show_vz,'string',num2str(drone(1).data_vz));

set(handles.data_show_roll,'string',num2str(drone(1).data_roll));
set(handles.data_show_pitch,'string',num2str(drone(1).data_pitch));
```

```
set(handles.data_show_yaw,'string',num2str(drone(1).data_yaw));
udp_timer = 1;
else
    udp_timer = udp_timer + 1;
end
```

APPENDIX C

MARK3 FLIGHT CONTROLLER FIRMWARE

Mark3 Flight Controller Code - Main Loop

```
#include ”Copter.h”
#include ”Interrupt.h”
_Copter Copter;
SBUS R9DS(Serial2);
void setup()
{
  InitComm(); //Teensy 3.2
  Copter.Copter_Init();
}
void loop()
{
  Copter.loopClock1 = micros();
  Copter.Ctrl_timer1++;   //100Hz
  RC_refine();
  Copter.AHRS();
  Copter.PEST();
  Copter.TranslationControl();
  Copter.AttitudeControl();
  Copter.AltitudeControl();
  Copter.InputTransform();
  Copter.command_Comm();
  Copter.Copter_Check();
}
```

Mark3 Flight Controller Code - Main Library of Definition

```
#include <Eigen.h>
#include <Eigen/LU>
#include ”Arduino.h”
#include ”i2c_t3.h”
#include ”EEPROM.h”
#include <SBUS.h>
#include ”System.h”
#include ”Estimation.h”
#include ”Sensor.h”
#include ”Motor.h”
#include <cmath>
#define LED0 13
#define LED1 31
#define LED2 33
#define Attitude_mode 1
#define Altitude_mode 2
#define loiter_mode 3
using namespace Eigen;
class _Copter
{
  public:
    /*----------------------------------------*/
    /*Copter.cpp*/
    unsigned long loopClock1 = 0, loopClock2 = 0;
    void Copter_Init();
    void Copter_Check();
    struct _flag
    {
      uint8_t ARMED = 0;
      uint8_t takingoff = 0;
      int8_t  turnoff = 0;
      uint8_t CRASHED = 0;
      uint8_t AltEmergency = 0;
      int8_t calibratedA = 0;
      int8_t calibratedG = 0;
      int8_t calibrationOn = 0;
      uint8_t mode = 0;
```

```
    uint8_t momentstart = 0;
    uint8_t LoiterSwitch = 0;
  };
  _flag flag;
  int16_t gltimer = 0, rltimer = 0;
  int8_t glch = 10, rlch = 10;

  /*Motor.cpp*/
  void MotorModel(double omega1, double
                  omega2, double omega3, double omega4);
  void MotorRun();
  void Motor_init();
  void Motor_stop();
  void InputTransform();
  /*Comm.cpp*/
  uint8_t xbee_data[500];
  int16_t xbee_length;
  int16_t xbeetime1 = 0;
  void command_Comm();
  void Xbee_comm();
  void Xbee_Packet();
  void Xbee_receive(int16_t order, uint16_t stmp);
  struct _RCsignal
  {
    short  ROLL;
    short  PITCH;
    short  THROTTLE;
    short  YAW;
    short  MODE;
    short  SWITCH;
    short  CH7;
    short  CH8;
    short  CH9;
    short  CH10;
  };
  _RCsignal RCsignal;
  short Xbee_timer = 0;
  short Xbee_receive_timer = 0;
  short comorder; /*----Command----*/
  short datalength = 0;
  /*Interrupt.cpp*/
  struct _IRpulse
  {
    int32_t ringbuffer[30];
    unsigned long timer_up, timer_down;
  };
  _IRpulse VRsensor1;

  /*Sensor.cpp*/
  void InitSensor();
  uint8_t I2Cwrite(uint8_t SENSOR_ADDRESS,
                   uint8_t SENSOR_REGISTER,
                   uint8_t SENSOR_VALUE,
                   bool sendStop);
  uint8_t I2CRead(uint8_t SENSOR_ADDRESS,
                  uint8_t SENSOR_REGISTER,
                  uint8_t nbytes);
  void MPU6050read();
  void MPU6050ThermalCompensation();
  void MPU6050Sixpoint();
  void MPU6050AccCali(uint8_t point);
  void MPU6050GyroCali();
  void AccPointRead();
  float GyroCollection[3] = {0, 0, 0};
  float gyro_temp, temperature;
  int16_t mpu_temperature;
  short GyroCaliFlag = 0;
  uint8_t i2cData[30];
```

```
uint8_t i2c1Data[30];
struct _float {
  float x;
  float y;
  float z;
};
struct _lint16 {
  short x;
  short y;
  short z;
};
struct _trans {
  struct _lint16 origin;
  struct _float filter;
  struct _float histor;
  struct _float aftcal;
  struct _float quietf;
  struct _float tempcp;
  struct _float radian;
};
struct _sensor {
  struct _trans acc;
  struct _trans gyro;
};
_sensor gy89, gy86;
struct _Acc_Cali {
  int16_t acc_calitimer = 0;
  int32_t acc_calitmpx;
  int32_t acc_calitmpy;
  int32_t acc_calitmpz;
  int16_t accel_raw_ref[6][3];
  float acc_offset[3];
  float a[3][3];
  float T[3][3];
  float g = 8192; //+-4g
};
_Acc_Cali Acc_Cali;

/*AttitudeEstimator.cpp*/
float inte_gyro[3];
struct _LKF
{
  float p;
  float q;
  float p_bias;
  float q_bias;
  float p_raw;
  float q_raw;
  float acc_roll;
  float acc_pitch;
};
_LKF LKF;

/*--------------Position Estimator--------------*/
union {
  float f;
  unsigned long ul;
} otus_tmp;
union {
  float f;
  uint8_t ul[4];
} xbee_tmp;
struct _Otus
{
  float x;
  float y;
  float z;
  float vx;
  float vy;
```

```
    float vz;
    float yaw;
    float pitch;
    float roll;
    float yaw_sin;
    float yaw_cos;
    float q0;
    float q1;
    float q2;
    float q3;
  };
  _Otus Otus;
  void Linear_Kalman_Filter();
  struct _state {
    float phi;
    float theta;
    float psi;
    float phi_rad;
    float theta_rad;
    float psi_rad;
    float phi_sin;
    float theta_sin;
    float psi_sin;
    float phi_cos;
    float theta_cos;
    float psi_cos;
    float p;
    float q;
    float r;
    float p_rad;
    float q_rad;
    float r_rad;
    float ax_b;
    float ay_b;
    float az_b;
    float ax;
    float ay;
    float az;
    float vx;
    float vy;
    float vz;
    float x;
    float y;
    float z;
    int16_t comm;

    int16_t takeoff_t = 0;
    int16_t landing_t = 0;
    uint8_t standby;
    uint8_t takeoff;
    uint8_t flight;
    uint8_t land;
    bool flight_state[4];
    float tmp_U1;
    bool update_phi = 0;
    bool update_theta = 0;
    bool update_psi = 0;
    //bool update_p;
    //bool update_q;
    //bool update_r;
    bool update_vx = 0;
    bool update_vy = 0;
    bool update_vz = 0;
    bool update_x = 0;
    bool update_y = 0;
    bool update_z = 0;
  };
  _state state;
  struct _IMU {
    float phi;
    float theta;
```

```
  float psi;
  float phi_rad;
  float theta_rad;
  float psi_rad;
  float phi_sin;
  float theta_sin;
  float psi_sin;
  float phi_cos;
  float theta_cos;
  float psi_cos;
  float p;
  float q;
  float r;
  float p_rad;
  float q_rad;
  float r_rad;
};
_IMU IMU;
struct _PosKF {
  float P[3][3];
  float tmp[3][3];
  float tmp_s[6];
  float state[3];
  float a;
  float v;
  float ps;
  float S[2][2];
  float y[3];
  float K[3][3];
  float Q;
  float R0;
  float R1;
};
_PosKF PosX, PosY, PosZ;
float q0 = 1.0f;
float q1 = 0.0f;
float q2 = 0.0f;
float q3 = 0.0f;
float q3old = 0.0f;
float exInt = 0.0;
float eyInt = 0.0;
float ezInt = 0.0;
float twoKp = twoKpDef;
float twoKi = twoKiDef;
float beta = betaDef;
float integralFBx = 0.0f,  integralFBy = 0.0f, integralFBz = 0.0f;

void PEST();
void Pos_Kalman(struct _PosKF *P_tmp);
void AHRS_Check();
void Madgwick_MARG_Update();
//void MadgwickAHRSupdateIMU(float gx,float gy, float gz, float ax,float ay, float az);
void MahonyAHRSupdate(float gx, float gy, float gz,
                      float ax, float ay, float az,
                      float mx, float my, float mz);
void MahonyAHRSupdateIMU(float gx, float gy, float gz,
                         float ax, float ay, float az);
void AHRS_filter_init();
void KF_init();
void AHRS();
void State_Reset();
void AHRS_filter();
void gy86_Dataanl();
struct _IIR {
  float b0;
  float b1;
  float b2;
  float a1;
  float a2;
```

```
  float element0;
  float element1;
  float element2;
};
_IIR gyro_IIRx, gyro_IIRy, gyro_IIRz,
acc_IIRx, acc_IIRy, acc_IIRz;
void IIR_set_cutoff_frequency(float sample_freq,
                              float cutoff_freq,
                              struct _IIR *input_IIR);
float IIR_filter_apply(float cutoff_freq,
                       float sample,
                       struct _IIR *input_IIR);
  double integral_r = 0;

/*Control part*/
  float U1, U2, U3, U4;
  uint8_t LockYaw = 0;
  struct _Ztransform {
    float Input[5] = {0, 0, 0, 0, 0};
    float Output[5] = {0, 0, 0, 0, 0};
    float Integral;
  };
_Ztransform Pcon, Qcon, Rcon, Phicon,
Thetacon, Psicon, Vzcon, Zcon, Vxpre,
Xpre, Vxcon, Xcon, Vypre, Ypre, Vycon, Ycon;
struct _Target
{
  float phi;
  float theta;
  float psi;
  float phi_rad;
  float theta_rad;
  float sin_phi;
  float cos_phi;
  float sin_theta;
  float cos_theta;
  float psi_rad;
  float p;
  float q;
  float r;
  float p_rad;
  float q_rad;
  float r_rad;
  int16_t throttle;
  float x;
  float y;
  float z;
  float vx;
  float vy;
  float vz;
  float plan_x;
  float plan_y;
  float plan_z;
  float plan_psi;
};
struct _PIDpram
{
  float Kp;
  float Ki;
  float Kd;
};
_PIDpram pvel, qvel, rvel, phiang, thetaang,
psiang, Vzalt, Zalt, Xpos, Ypos, Vxpos, Vypos;
_Target Target;
uint8_t Ctrl_timer1 = 0;
float EstimatedG;
void ControlReset();
void TranslationControl();
void AttitudeControl();
void AltitudeControl();
```

```
    void ZControl();
    void VzControl();
    void AngularRateControl();
    struct _inertia
    {
      uint8_t timer_start = 0;
      unsigned long time_start = 0, time_end = 0, time_count = 0;
      uint16_t pendulum = 0;
      float memo = -1000.0;
      uint8_t xbee_timer = 0;
    };
    _inertia inertia;
    /*System.cpp*/
    uint16_t run_period;
    unsigned short time_out = 0;
    unsigned short battery_warning = 0;
    void InitControl();
    void Moment_Check();
    void Otus_Clear();
    void Loop_Check();
    void Timer_Check();
    void Battery_Check();
    float Rad(float angle);
    float Degree(float rad);
    float data_limitation(float a, float b, float c);
    float invSqrt(float number);
    float voltage;
    float voltageavg;

    /*-----------------------------------------*/
  private:
    /*-----------------------------------------*/
    /*Copter.cpp*/

    /*Motor.cpp*/
    float PWM1, PWM2, PWM3, PWM4;
    double omega12, omega22, omega32,
    omega42, omega1, omega2, omega3, omega4;
    float pwm_factor = 65535.0 / 2500.0;
    float PWM;
    double InputK1;
    double InputK2;
    double InputK3;
    double InputK4;

    /*System.cpp*/
    unsigned long whole_timer;

    /*-----------------------------------------*/
};
```

Mark3 Flight Controller Code - State Estimation

```
#include "Copter.h"
/*
  MatrixXf Pos_F(9, 9);
  MatrixXf Pos_B(9, 9);
  MatrixXf Pos_H(9, 9);
  MatrixXf Pos_Q(9, 9);
  MatrixXf Pos_R(9, 9);
  MatrixXf Pos_S(9, 9);
  MatrixXf Pos_K(9, 9);
  MatrixXf Pos_P(9, 9);
  MatrixXf Pos_I(9, 9);
  MatrixXf Pos_State(9, 1);
  MatrixXf Pos_Y(9, 1);
```

```
  MatrixXf Pos_U(9, 1);
  MatrixXf Pos_Z(9, 1);
*/
void _Copter::PEST()
{
  if (Ctrl_timer1 >= 4)
  {
    float OtusNorm = invSqrt(Otus.q0 * Otus.q0 + Otus.q1
    * Otus.q1 + Otus.q2 * Otus.q2 + Otus.q3 * Otus.q3);
    Otus.q0 *= OtusNorm;
    Otus.q1 *= OtusNorm;
    Otus.q2 *= OtusNorm;
    Otus.q3 *= OtusNorm;


    MatrixXf Otus_R(3, 3);
    MatrixXf Acc_B(3, 1);
    MatrixXf Acc_I(3, 1);

    float q0_2, q1_2, q2_2, q3_2, q1q2,
    q0q3, q1q3, q0q2, q2q3, q0q1;

    q0_2 = Otus.q0 * Otus.q0;
    q1_2 = Otus.q1 * Otus.q1;
    q2_2 = Otus.q2 * Otus.q2;
    q3_2 = Otus.q3 * Otus.q3;

    q1q2 = Otus.q1 * Otus.q2;
    q0q3 = Otus.q0 * Otus.q3;
    q1q3 = Otus.q1 * Otus.q3;
    q0q2 = Otus.q0 * Otus.q2;
    q2q3 = Otus.q2 * Otus.q3;
    q0q1 = Otus.q0 * Otus.q1;

    Otus_R << q0_2 + q1_2 - q2_2 - q3_2, 2 *
           (q1q2 - q0q3),  2 * (q1q3 + q0q2),
           2 * (q1q2 + q0q3),  q0_2 - q1_2 + q2_2 -
           q3_2,  2 * (q2q3 - q0q1),
           2 * (q1q3 - q0q2),  2 * (q2q3 + q0q1),
           q0_2 - q1_2 - q2_2 + q3_2;

    Acc_B << gy86.acc.filter.x / 8192.0 * 9.8 ,
    gy86.acc.filter.y / 8192.0 * 9.8 ,
    gy86.acc.filter.z / 8192.0 * 9.8;
    Acc_I =  Otus_R * Acc_B;

    state.ax = Acc_I(0, 0);
    state.ay = -Acc_I(1, 0);
    state.az = Acc_I(2, 0) - 9.8;

    float CF_a = 0.8, outerT_2 = outerT * outerT;
    state.vx = (state.vx + outerT * state.ax)
    * CF_a + Otus.vx * (1 - CF_a);
    state.vy = (state.vy + outerT * state.ay)
    * CF_a + Otus.vy * (1 - CF_a);
    state.vz = (state.vz + outerT * state.az)
    * CF_a + Otus.vz * (1 - CF_a);

    state.x = (state.x + outerT * state.vx +
    0.5 * outerT_2 * state.ax) * CF_a + Otus.x * (1 - CF_a);
    state.y = (state.y + outerT * state.vy +
    0.5 * outerT_2 * state.ay) * CF_a + Otus.y * (1 - CF_a);
    state.z = (state.z + outerT * state.vz +
    0.5 * outerT_2 * state.az) * CF_a + Otus.z * (1 - CF_a);
  }
}
void _Copter:: Pos_Kalman(struct _PosKF *P_tmp)
```

```
{
  P_tmp->tmp_s[0] = P_tmp->a / 20000 + P_tmp->state[0] +
  P_tmp->state[1] / 100 - P_tmp->state[2] / 20000;
  P_tmp->tmp_s[1] = P_tmp->a / 100 + P_tmp->state[1] -
  P_tmp->state[2] / 100;
  P_tmp->tmp_s[2] = P_tmp->state[2];

  P_tmp->state[0] = P_tmp->tmp_s[0];
  P_tmp->state[1] = P_tmp->tmp_s[1];
  P_tmp->state[2] = P_tmp->tmp_s[2];
  P_tmp->tmp[0][0] = P_tmp->P[0][0] +
  P_tmp->P[0][1] / 100 -
  P_tmp->P[0][2] / 20000 + P_tmp->P[1][0] / 100 +
  P_tmp->P[1][1] / 10000 -
  P_tmp->P[1][2] / 2000000 - P_tmp->P[2][0] / 20000 -
  P_tmp->P[2][1] / 2000000 +
  P_tmp->P[2][2] / 400000000;
  P_tmp->tmp[0][1] = P_tmp->P[0][1] -
  P_tmp->P[0][2] / 100 +
  P_tmp->P[1][1] / 100 - P_tmp->P[1][2] / 10000 -
  P_tmp->P[2][1] / 20000 + P_tmp->P[2][2] / 2000000;
  P_tmp->tmp[0][2] = P_tmp->P[0][2] +
  P_tmp->P[1][2] / 100 -
  P_tmp->P[2][2] / 20000;
  P_tmp->tmp[1][0] = P_tmp->P[1][0] +
  P_tmp->P[1][1] / 100 -
  P_tmp->P[1][2] / 20000 - P_tmp->P[2][0] / 100 -
  P_tmp->P[2][1] / 10000 + P_tmp->P[2][2] / 2000000;
  P_tmp->tmp[1][1] = P_tmp->P[1][1] -
  P_tmp->P[1][2] / 100 -
  P_tmp->P[2][1] / 100 + P_tmp->P[2][2] / 10000;
  P_tmp->tmp[1][2] = P_tmp->P[1][2] -
  P_tmp->P[2][2] / 100;
  P_tmp->tmp[2][0] = P_tmp->P[2][0] +
  P_tmp->P[2][1] / 100 - P_tmp->P[2][2] / 20000;
  P_tmp->tmp[2][1] = P_tmp->P[2][1] -
  P_tmp->P[2][2] / 100;
  P_tmp->tmp[2][2] = P_tmp->P[2][2] + P_tmp->Q;

  for (uint8_t i = 0; i < 3; i++)
    for (uint8_t j = 0; j < 3; j++)
      P_tmp->P[i][j] = P_tmp->tmp[i][j];
  /*----------y = Z - H * state;-------------*/
  P_tmp->y[0] = P_tmp->ps - P_tmp->state[0];
  P_tmp->y[1] = P_tmp->v - P_tmp->state[1];
  /*-----------S = H*P*H' + R;-----------*/
  P_tmp->S[0][0] = P_tmp->P[0][0] + P_tmp->R0;
  P_tmp->S[0][1] = P_tmp->P[0][1];
  P_tmp->S[1][0] = P_tmp->P[1][0];
  P_tmp->S[1][1] = P_tmp->P[1][1] + P_tmp->R1;
  /*-----------pinv(S)----------*/
  P_tmp->tmp_s[1] = P_tmp->S[0][0] *
  P_tmp->S[0][0] + P_tmp->S[1][0] * P_tmp->S[1][0];
  P_tmp->tmp_s[3] = (P_tmp->S[0][0] *
  P_tmp->S[0][1]) / P_tmp->tmp_s[1];
  P_tmp->tmp_s[4] = (P_tmp->S[1][0] *
  P_tmp->S[1][1]) / P_tmp->tmp_s[1];

  P_tmp->tmp_s[5] = P_tmp->S[0][1] -
  P_tmp->S[0][0] * (P_tmp->tmp_s[3] + P_tmp->tmp_s[4]);
  P_tmp->tmp_s[0] = P_tmp->S[1][1] -
  P_tmp->S[1][0] * (P_tmp->tmp_s[3] + P_tmp->tmp_s[4]);
  P_tmp->tmp_s[2] = P_tmp->tmp_s[5] *
  P_tmp->tmp_s[5]  + P_tmp->tmp_s[0] * P_tmp->tmp_s[0];
```

```
  if (P_tmp->tmp_s[1] == 0)
    P_tmp->tmp_s[1] = 0.0000001;
  if (P_tmp->tmp_s[2] == 0)
    P_tmp->tmp_s[2] = 0.0000001;
  P_tmp->tmp[0][0] = P_tmp->S[0][0] /
  P_tmp->tmp_s[1] -
  ((P_tmp->tmp_s[5]) * (P_tmp->tmp_s[3] +
  P_tmp->tmp_s[4])) / P_tmp->tmp_s[2];
  P_tmp->tmp[0][1] = P_tmp->S[1][0] / P_tmp->tmp_s[1] -
  ((P_tmp->tmp_s[0]) * (P_tmp->tmp_s[3] +
  P_tmp->tmp_s[4])) / P_tmp->tmp_s[2];
  P_tmp->tmp[1][0] = (P_tmp->tmp_s[5]) /
  P_tmp->tmp_s[2];
  P_tmp->tmp[1][1] = (P_tmp->tmp_s[0]) /
  P_tmp->tmp_s[2];
  /*---------K = P*H'*pinv(S);----------*/
  P_tmp->K[0][0] = P_tmp->P[0][0] *
  P_tmp->tmp[0][0] + P_tmp->P[0][1] * P_tmp->tmp[1][0];
  P_tmp->K[0][1] = P_tmp->P[0][0] *
  P_tmp->tmp[0][1] + P_tmp->P[0][1] * P_tmp->tmp[1][1];
  P_tmp->K[1][0] = P_tmp->P[1][0] *
  P_tmp->tmp[0][0] + P_tmp->P[1][1] * P_tmp->tmp[1][0];
  P_tmp->K[1][1] = P_tmp->P[1][0] *
  P_tmp->tmp[0][1] + P_tmp->P[1][1] * P_tmp->tmp[1][1];
  P_tmp->K[2][0] = P_tmp->P[2][0] *
  P_tmp->tmp[0][0] + P_tmp->P[2][1] * P_tmp->tmp[1][0];
  P_tmp->K[2][1] = P_tmp->P[2][0] *
  P_tmp->tmp[0][0] + P_tmp->P[2][1] * P_tmp->tmp[1][0];
  /*-------------state = state + K*y;----*/
  P_tmp->tmp_s[0] = P_tmp->state[0];
  P_tmp->tmp_s[1] = P_tmp->state[1];
  P_tmp->tmp_s[2] = P_tmp->state[2];

  P_tmp->state[0] = P_tmp->tmp_s[0] +
  P_tmp->K[0][0] * P_tmp->y[0] +
  P_tmp->K[0][1] * P_tmp->y[1];
  P_tmp->state[1] = P_tmp->tmp_s[1] +
  P_tmp->K[1][0] * P_tmp->y[0] +
  P_tmp->K[1][1] * P_tmp->y[1];
  P_tmp->state[2] = P_tmp->tmp_s[2] +
  P_tmp->K[2][0] * P_tmp->y[0] +
  P_tmp->K[2][1] * P_tmp->y[1];
  /*---------state = state + K*y;----------*/
  for (uint8_t i = 0; i < 3; i++)
    for (uint8_t j = 0; j < 3; j++)
      P_tmp->tmp[i][j] = P_tmp->P[i][j];

  P_tmp->P[0][0] = - P_tmp->K[0][1] *
  P_tmp->tmp[1][0] - P_tmp->tmp[0][0] *
  (P_tmp->K[0][0] - 1);
  P_tmp->P[0][1] = - P_tmp->K[0][1] *
  P_tmp->tmp[1][1] - P_tmp->tmp[0][1] *
  (P_tmp->K[0][0] - 1);
  P_tmp->P[0][2] = - P_tmp->K[0][1] *
  P_tmp->tmp[1][2] - P_tmp->tmp[0][2] *
  (P_tmp->K[0][0] - 1);
  P_tmp->P[1][0] = -P_tmp->K[1][0] *
  P_tmp->tmp[0][0] - P_tmp->tmp[1][0] *
  (P_tmp->K[1][1] - 1);
  P_tmp->P[1][1] = -P_tmp->K[1][0] *
  P_tmp->tmp[0][1] - P_tmp->tmp[1][1] *
  (P_tmp->K[1][1] - 1);
  P_tmp->P[1][2] = - P_tmp->K[1][0] *
  P_tmp->tmp[0][2] - P_tmp->tmp[1][2] *
```

```
  (P_tmp->K[1][1] - 1);
  P_tmp->P[2][0] = P_tmp->tmp[2][0] -
  P_tmp->K[2][0] * P_tmp->tmp[0][0] -
  P_tmp->K[2][1] * P_tmp->tmp[1][0];
  P_tmp->P[2][1] = P_tmp->tmp[2][1] -
  P_tmp->K[2][0] * P_tmp->tmp[0][1] -
  P_tmp->K[2][1] * P_tmp->tmp[1][1];
  P_tmp->P[2][2] = P_tmp->tmp[2][2] -
  P_tmp->K[2][0] * P_tmp->tmp[0][2] -
  P_tmp->K[2][1] * P_tmp->tmp[1][2];
}
void _Copter::AHRS_Check()
{
  state.p = IMU.p;
  state.p_rad = IMU.p_rad;
  state.q = IMU.q;
  state.q_rad = IMU.q_rad;
  state.r = IMU.r;
  state.r_rad = IMU.r_rad;

  if (Ctrl_timer1 >= 4)
  {
    /*--------------------Phi--------------------*/
    if (state.update_phi)
    {
      if (abs(IMU.phi - Otus.roll) < 5.0)
        state.phi = IMU.phi * 0.75 + Otus.roll * 0.25;
      else
        state.phi = IMU.phi * 0.99 + Otus.roll * 0.01;
    }
    else
      state.phi = IMU.phi;
    state.phi_rad = Rad(state.phi);
    /*-------------------Theta-------------------*/
    if (state.update_theta)
    {
      if (abs(IMU.theta - Otus.pitch) < 5.0)
        state.theta = IMU.theta * 0.75 + Otus.pitch * 0.25;
      else
        state.theta = IMU.theta * 0.99 + Otus.pitch * 0.01;
    }
    else
      state.theta = IMU.theta;
    state.theta_rad = Rad(state.theta);

    state.phi_sin = sin(state.phi_rad);
    state.theta_sin = sin(state.theta_rad);
    state.phi_cos = cos(state.phi_rad);
    state.theta_cos = cos(state.theta_rad);

    float pre_cos , pre_sin, ob_cos, ob_sin, inc, cof = 0.70;
    inc = (-state.phi_sin / state.theta_cos * state.q +
    state.phi_cos / state.theta_cos * state.r) * outerT;
    pre_cos = cos(Rad(state.psi + inc));
    pre_sin = sin(Rad(state.psi + inc));
    if (state.update_psi == 1)
    {
      ob_cos = cos(Rad(Otus.yaw));
      ob_sin = sin(Rad(Otus.yaw));
    }
    else
    {
      ob_cos = pre_cos;
      ob_sin = pre_sin;
    }
    pre_cos = pre_cos * cof + ob_cos * (1 - cof);
    pre_sin = pre_sin * cof + ob_sin * (1 - cof);
```

```

    state.psi_rad = atan2(pre_sin , pre_cos);
    state.psi = Degree(state.psi_rad);
    state.psi_sin = sin(state.psi_rad);
    state.psi_cos = cos(state.psi_rad);
  }
}
void _Copter::AHRS()
{
  AHRS_filter();         //Digital Filter
  //Linear_Kalman_Filter();
  IMU.p = gy86.gyro.filter.x / 32.768;
  IMU.q = gy86.gyro.filter.y / 32.768;
  IMU.r = gy86.gyro.filter.z / 32.768;
  IMU.p_rad = Rad(IMU.p);
  IMU.q_rad = Rad(IMU.q);
  IMU.r_rad = Rad(IMU.r);
  inte_gyro[0] += IMU.p * 0.0025;
  inte_gyro[1] += IMU.q * 0.0025;
  inte_gyro[2] += IMU.r * 0.0025;
  Madgwick_MARG_Update();
  AHRS_Check();
}
void _Copter::AHRS_filter()
{
  gy86_Dataanl();
  gy86.gyro.filter.x =
  IIR_filter_apply(L3GD20_DEFAULT_FILTER_FREQ,
  gy86.gyro.aftcal.x, &gyro_IIRx);
  gy86.gyro.filter.y =
  IIR_filter_apply(L3GD20_DEFAULT_FILTER_FREQ,
  -gy86.gyro.aftcal.y, &gyro_IIRy);
  gy86.gyro.filter.z =
  IIR_filter_apply(L3GD20_DEFAULT_FILTER_FREQ,
  -gy86.gyro.aftcal.z, &gyro_IIRz);

  gy86.acc.filter.x =
  IIR_filter_apply(LSM303D_ACCEL_DEFAULT_DRIVER_FILTER_FREQ,
  gy86.acc.aftcal.x, &acc_IIRx);
  gy86.acc.filter.y =
  IIR_filter_apply(LSM303D_ACCEL_DEFAULT_DRIVER_FILTER_FREQ,
  gy86.acc.aftcal.y, &acc_IIRy);
  gy86.acc.filter.z =
  IIR_filter_apply(LSM303D_ACCEL_DEFAULT_DRIVER_FILTER_FREQ,
  gy86.acc.aftcal.z, &acc_IIRz);
}
void _Copter::gy86_Dataanl()
{
  MPU6050read();
}
void _Copter::AHRS_filter_init()
{
  IIR_set_cutoff_frequency(L3GD20_DEFAULT_RATE,
  L3GD20_DEFAULT_FILTER_FREQ, &gyro_IIRx);
  IIR_set_cutoff_frequency(L3GD20_DEFAULT_RATE,
  L3GD20_DEFAULT_FILTER_FREQ, &gyro_IIRy);
  IIR_set_cutoff_frequency(L3GD20_DEFAULT_RATE,
  L3GD20_DEFAULT_FILTER_FREQ, &gyro_IIRz);

  IIR_set_cutoff_frequency(LSM303D_ACCEL_DEFAULT_RATE,
  LSM303D_ACCEL_DEFAULT_DRIVER_FILTER_FREQ, &acc_IIRx);
  IIR_set_cutoff_frequency(LSM303D_ACCEL_DEFAULT_RATE,
  LSM303D_ACCEL_DEFAULT_DRIVER_FILTER_FREQ, &acc_IIRy);
  IIR_set_cutoff_frequency(LSM303D_ACCEL_DEFAULT_RATE,
  LSM303D_ACCEL_DEFAULT_DRIVER_FILTER_FREQ, &acc_IIRz);
```

```
}
void _Copter::IIR_set_cutoff_frequency(float sample_freq,
float cutoff_freq, struct _IIR *input_IIR)
{
  if (cutoff_freq <= 0.0f) {
    // no filtering
    return;
  }
  float fr = sample_freq / cutoff_freq;
  float ohm = tanf(M_PI_F / fr);
  float c = 1.0f + 2.0f *
  cosf(M_PI_F / 4.0f) * ohm + ohm * ohm;
  input_IIR->b0 = ohm * ohm / c;
  input_IIR->b1 = 2.0f * input_IIR->b0;
  input_IIR->b2 = input_IIR->b0;
  input_IIR->a1 = 2.0f * (ohm * ohm - 1.0f) / c;
  input_IIR->a2 = (1.0f -
  2.0f * cosf(M_PI_F / 4.0f) *
  ohm + ohm * ohm) / c;
}
float _Copter:: IIR_filter_apply(float cutoff_freq,
float sample, struct _IIR *input_IIR)
{
  if (cutoff_freq <= 0.0f) {
    // no filtering
    return sample;
  }
  // do the filtering
  input_IIR->element0 = sample -
  input_IIR->element1 *
  input_IIR->a1 - input_IIR->element2 *
  input_IIR->a2;
  float output = input_IIR->element0 *
  input_IIR->b0 +
  input_IIR->element1 * input_IIR->b1 +
  input_IIR->element2 * input_IIR->b2;

  input_IIR->element2 = input_IIR->element1;
  input_IIR->element1 = input_IIR->element0;
  return output;
}
void _Copter::State_Reset()
{
  //q0 = Otus.q0;
  //q1 = Otus.q1;
  //q2 = Otus.q2;
  //q3 = Otus.q3;

  integralFBx = 0.0f;
  integralFBy = 0.0f;
  integralFBz = 0.0f;

  state.vx = Otus.vx;
  state.vy = Otus.vy;
  state.vz = Otus.vz;

  state.x = Otus.x;
  state.y = Otus.y;
  state.z = Otus.z;

  state.psi = Otus.yaw;
}
void _Copter::KF_init()
{
  PosX.Q = 2 / 10;
  PosY.Q = 2 / 10;
  PosZ.Q = 2 / 10;
  PosX.R0 = 0.01;
```

```
  PosY.R0 = 0.01;
  PosZ.R0 = 0.01;
  PosX.R1 = 0.01;
  PosY.R1 = 0.01;
  PosZ.R1 = 0.01;
}
/*-----Complimentary Filter-----*/
// Ref: "Nonlinear Complementary Filters on the Special Orthoganol Group"
// By Robert Mahoney, published in 2007
void _Copter::MahonyAHRSupdate(float gx, float gy,
float gz, float ax, float ay, float az,
float mx, float my, float mz) {
  float recipNorm;
  float q0q0, q0q1, q0q2, q0q3, q1q1,
  q1q2, q1q3, q2q2, q2q3, q3q3;
  float hx, hy, bx, bz;
  float halfvx, halfvy, halfvz,
  halfwx, halfwy, halfwz;
  float halfex, halfey, halfez;
  float qa, qb, qc;

  if ((mx == 0.0f) && (my == 0.0f)
  && (mz == 0.0f)) {
    MahonyAHRSupdateIMU(gx, gy, gz, ax, ay, az);
    return;
  }
  if (!((ax == 0.0f) &&
  (ay == 0.0f) && (az == 0.0f))) {

    // Normalise accelerometer measurement
    recipNorm = invSqrt(ax * ax +
    ay * ay + az * az);
    ax *= recipNorm;
    ay *= recipNorm;
    az *= recipNorm;

    // Normalise magnetometer measurement
    recipNorm = invSqrt(mx * mx +
    my * my + mz * mz);
    mx *= recipNorm;
    my *= recipNorm;
    mz *= recipNorm;

    q0q0 = q0 * q0;
    q0q1 = q0 * q1;
    q0q2 = q0 * q2;
    q0q3 = q0 * q3;
    q1q1 = q1 * q1;
    q1q2 = q1 * q2;
    q1q3 = q1 * q3;
    q2q2 = q2 * q2;
    q2q3 = q2 * q3;
    q3q3 = q3 * q3;

    hx = 2.0f * (mx * (0.5f - q2q2 - q3q3) +
    my * (q1q2 - q0q3) + mz * (q1q3 + q0q2));
    hy = 2.0f * (mx * (q1q2 + q0q3) +
    my * (0.5f - q1q1 - q3q3) + mz * (q2q3 - q0q1));
    bx = sqrt(hx * hx + hy * hy);
    bz = 2.0f * (mx * (q1q3 - q0q2) +
    my * (q2q3 + q0q1) + mz * (0.5f - q1q1 - q2q2));

    halfvx = q1q3 - q0q2;
    halfvy = q0q1 + q2q3;
    halfvz = q0q0 - 0.5f + q3q3;
    halfwx = bx * (0.5f - q2q2 - q3q3) +
    bz * (q1q3 - q0q2);
    halfwy = bx * (q1q2 - q0q3) +
```

```
    bz * (q0q1 + q2q3);
    halfwz = bx * (q0q2 + q1q3) +
    bz * (0.5f - q1q1 - q2q2);

    //direction and measured direction of field vectors
    halfex = (ay * halfvz - az * halfvy) +
    (my * halfwz - mz * halfwy);
    halfey = (az * halfvx - ax * halfvz) +
    (mz * halfwx - mx * halfwz);
    halfez = (ax * halfvy - ay * halfvx) +
    (mx * halfwy - my * halfwx);

    if (twoKi > 0.0f) {
      integralFBx += twoKi * halfex
      * (1.0f / sampleFreq);
      integralFBy += twoKi * halfey
      * (1.0f / sampleFreq);
      integralFBz += twoKi * halfez
      * (1.0f / sampleFreq);
      gx += integralFBx;
      gy += integralFBy;
      gz += integralFBz;
    }
    else {
      integralFBx = 0.0f;
      integralFBy = 0.0f;
      integralFBz = 0.0f;
    }

    // Apply proportional feedback
    gx += twoKp * halfex;
    gy += twoKp * halfey;
    gz += twoKp * halfez;
  }

  // Integrate rate of change of quaternion
  gx *= (0.5f * (1.0f / sampleFreq));
  gy *= (0.5f * (1.0f / sampleFreq));
  gz *= (0.5f * (1.0f / sampleFreq));
  qa = q0;
  qb = q1;
  qc = q2;
  q0 += (-qb * gx - qc * gy - q3 * gz);
  q1 += (qa * gx + qc * gz - q3 * gy);
  q2 += (qa * gy - qb * gz + q3 * gx);
  q3 += (qa * gz + qb * gy - qc * gx);

  // Normalise quaternion
  recipNorm = invSqrt(q0 * q0 +
  q1 * q1 + q2 * q2 + q3 * q3);
  q0 *= recipNorm;
  q1 *= recipNorm;
  q2 *= recipNorm;
  q3 *= recipNorm;
}
void _Copter::MahonyAHRSupdateIMU(float gx, float gy,
float gz, float ax, float ay, float az) {
  float recipNorm;
  float halfvx, halfvy, halfvz;
  float halfex, halfey, halfez;
  float qa, qb, qc;

  // Compute feedback only if accelerometer measurement valid (avoids NaN in accelerometer norm
  if (!((ax == 0.0f) && (ay == 0.0f) && (az == 0.0f))) {

    // Normalise accelerometer measurement
    recipNorm = invSqrt(ax * ax + ay * ay + az * az);
```

```
    ax *= recipNorm;
    ay *= recipNorm;
    az *= recipNorm;


    halfvx = q1 * q3 - q0 * q2;
    halfvy = q0 * q1 + q2 * q3;
    halfvz = q0 * q0 - 0.5f + q3 * q3;

    halfex = (ay * halfvz - az * halfvy);
    halfey = (az * halfvx - ax * halfvz);
    halfez = (ax * halfvy - ay * halfvx);

    // Compute and apply integral feedback if enabled
    if (twoKi > 0.0f) {
      integralFBx += twoKi * halfex * (1.0f / sampleFreq);
      integralFBy += twoKi * halfey * (1.0f / sampleFreq);
      integralFBz += twoKi * halfez * (1.0f / sampleFreq);
      gx += integralFBx;  // apply integral feedback
      gy += integralFBy;
      gz += integralFBz;
    }
    else {
      integralFBx = 0.0f; // prevent integral windup
      integralFBy = 0.0f;
      integralFBz = 0.0f;
    }

    // Apply proportional feedback
    gx += twoKp * halfex;
    gy += twoKp * halfey;
    gz += twoKp * halfez;
  }

  // Integrate rate of change of quaternion
  gx *= (0.5f * (1.0f / sampleFreq));
  gy *= (0.5f * (1.0f / sampleFreq));
  gz *= (0.5f * (1.0f / sampleFreq));
  qa = q0;
  qb = q1;
  qc = q2;
  q0 += (-qb * gx - qc * gy - q3 * gz);
  q1 += (qa * gx + qc * gz - q3 * gy);
  q2 += (qa * gy - qb * gz + q3 * gx);
  q3 += (qa * gz + qb * gy - qc * gx);

  // Normalise quaternion
  recipNorm = invSqrt(q0 * q0 +
  q1 * q1 + q2 * q2 + q3 * q3);
  q0 *= recipNorm;
  q1 *= recipNorm;
  q2 *= recipNorm;
  q3 *= recipNorm;
}
```

Mark3 Flight Controller Code - Communication Process

```
#include "Copter.h"
void _Copter::command_Comm()
{
  //Counttimer1 = micros();
  if (RCsignal.THROTTLE > 250 && RCsignal.THROTTLE < 350
      && RCsignal.YAW < 350 && RCsignal.YAW > 250) {
    flag.ARMED = 1;
  }
  if (flag.ARMED == 1 && RCsignal.THROTTLE < 350
```

```
      && RCsignal.YAW > 950)  {
    flag.ARMED = 2;
    ControlReset();
  }
  //Stopping the motors: throttle low and yaw right.
  if (RCsignal.THROTTLE < 350 && RCsignal.YAW > 1650)  {
    flag.ARMED = 0;
  }
  if (RCsignal.MODE < 350 && RCsignal.MODE > 0)
    flag.mode = Attitude_mode;
  if (RCsignal.MODE < 1050 && RCsignal.MODE > 950)
    flag.mode = Altitude_mode ;
  if (RCsignal.MODE < 2000 && RCsignal.MODE > 1650)
    flag.mode = loiter_mode;
  if (RCsignal.SWITCH < 350) flag.turnoff = 0;
  if (RCsignal.SWITCH > 1650) flag.turnoff = 1;
  Xbee_comm();
}
void _Copter::Xbee_comm()
{
  /*
    if (comorder == 1062)
    {
    Serial1.print(IMU.p, 3); //Serial.print("       ");
    //Serial1.print(IMU.q, 3); Serial.print("       ");
    //Serial1.print(IMU.r, 3); Serial.print("       ");
    Serial1.println();
    }
  */
  Xbee_timer++;
  if (Xbee_timer == 8)
  {

    //Serial.print(state.x, 4); Serial.print('\t');
    //Serial.print(state.y, 4); Serial.print('\t');
    //Serial.print(state.z, 4); Serial.print('\t');

    //Serial.print(Otus.x, 4); Serial.print('\t');
    //Serial.print(Otus.y, 4); Serial.print('\t');
    //Serial.print(Otus.z, 4); Serial.print('\t');
    //Serial.println();
    /*
      Serial.print(LKF.acc_roll); Serial.print('\t');
      Serial.print(LKF.acc_pitch); Serial.print('\t');
      Serial.print(Otus.yaw); Serial.print('\t');

      Serial.print(state.p); Serial.print('\t');
      Serial.print(state.q); Serial.print('\t');
      Serial.print(state.r); Serial.print('\t');
      Serial.println();
    */
    //Serial.println();
    /*
      if (xbee_length != 0)
      {
      Serial.print(Otus.x); Serial.print('\t');
      Serial.print(loopClock2); Serial.print('\t');
      Serial.print(xbee_length); Serial.print('\t');
      for (uint8_t i = 0; i < xbee_length; i++)
      {
        Serial.print(xbee_data[100 + i]); Serial.print(" ");
      }
      Serial.println();
      }
    */
```

```
  switch (comorder)
  {
    case 1000:
      Serial1.print(RCsignal.ROLL);
      Serial1.print('\t');
      Serial1.print(RCsignal.PITCH);
      Serial1.print('\t');
      Serial1.print(RCsignal.THROTTLE);
      Serial1.print('\t');
      Serial1.print(RCsignal.YAW);
      Serial1.print('\t');
      Serial1.print(RCsignal.MODE);
      Serial1.print('\t');
      Serial1.print(RCsignal.SWITCH);
      Serial1.print('\t');
      Serial1.print(RCsignal.CH7);
      Serial1.print('\t');
      Serial1.print(RCsignal.CH8);
      Serial1.print('\t');
      Serial1.print(RCsignal.CH9);
      Serial1.print('\t');
      Serial1.print(RCsignal.CH10);
      Serial1.print('\t');
      Serial1.print("Arm: ");
      Serial1.print(flag.ARMED);
      Serial1.print('\t');
      Serial1.print("Turn Off: ");
      Serial1.print(flag.turnoff);Serial1.print('\t');
      if (flag.mode == 1)
      {
        Serial1.print("Attitude mode");
        Serial1.print('\t');
      }
      else if (flag.mode == 2)
      {
        Serial1.print("Altitude mode");
        Serial1.print('\t');
      }
      else if (flag.mode == 3)
      {
        Serial1.print("loiter mode");
        Serial1.print('\t');
      }
      Serial1.println();
      break;
    case 1010:
      State_Reset();
      break;
    /*--------------------Gyro Test----------------------*/
    case 1020: //Calculate Current Offset
      flag.calibratedG = 0;
      comorder = 0;
      inte_gyro[0] = 0;
      inte_gyro[1] = 0;
      inte_gyro[2] = 0;
      break;
    case 1021:
      Serial1.print(gy86.acc.origin.x); Serial1.print('\t');
      Serial1.print(gy86.acc.origin.y); Serial1.print('\t');
      Serial1.print(gy86.acc.origin.z); Serial1.print('\t');
      Serial1.print(mpu_temperature); Serial1.print('\t');
      Serial1.print(temperature); Serial1.print('\t');
      Serial1.print(gy86.gyro.origin.x); Serial1.print('\t');
      Serial1.print(gy86.gyro.origin.y); Serial1.print('\t');
      Serial1.print(gy86.gyro.origin.z); Serial1.print('\t');
```

```
      Serial1.println();
      break;
    case 1022:
      Serial1.print(gy86.gyro.origin.x); Serial1.print('\t');
      Serial1.print(gy86.gyro.aftcal.x); Serial1.print('\t');
      Serial1.println();
      break;
    case 1023:
      Serial1.print(gy86.gyro.origin.x); Serial1.print('\t');
      Serial1.print(gy86.gyro.origin.y); Serial1.print('\t');
      Serial1.print(gy86.gyro.origin.z); Serial1.print('\t');
      Serial1.println();
      break;
    case 1024:
      Serial1.print(gyro_temp); Serial1.print('\t');
      Serial1.print(gy86.gyro.aftcal.x); Serial1.print('\t');
      Serial1.print(gy86.gyro.aftcal.y); Serial1.print('\t');
      Serial1.print(gy86.gyro.aftcal.z); Serial1.print('\t');
      Serial1.println();
      break;
    case 1025:
      Serial1.print(gy86.gyro.aftcal.x); Serial1.print('\t');
      Serial1.print(gy86.gyro.filter.x); Serial1.print('\t');
      Serial1.println();
      break;
    case 1026:
      Serial1.print(gy86.gyro.origin.z); Serial1.print('\t');
      Serial1.print(gy86.gyro.aftcal.z); Serial1.print('\t');
      Serial1.println();
      break;
    case 1027:
      Serial1.print(gy86.gyro.filter.x); Serial1.print('\t');
      Serial1.print(gy86.gyro.filter.y); Serial1.print('\t');
      Serial1.print(gy86.gyro.filter.z); Serial1.print('\t');
      Serial1.println();
      break;
    case 1028:
      Serial1.print(inte_gyro[0]); Serial1.print('\t');
      Serial1.print(inte_gyro[1]); Serial1.print('\t');
      Serial1.print(inte_gyro[2]); Serial1.print('\t');
      Serial1.println();
      break;
    /*-----------Accelerometer Test---------*/
    case 1031:
      Serial1.print(gy86.acc.origin.x); Serial1.print('\t');
      Serial1.print(gy86.acc.origin.y); Serial1.print('\t');
      Serial1.print(gy86.acc.origin.z); Serial1.print('\t');
      Serial1.println();
      break;
    case 1032:
      Serial1.print(gy86.acc.aftcal.x); Serial1.print('\t');
      Serial1.print(gy86.acc.aftcal.y); Serial1.print('\t');
      Serial1.print(gy86.acc.aftcal.z); Serial1.print('\t');
      Serial1.println();
      break;
    case 1033:
      Serial1.print(gy86.acc.aftcal.x); Serial1.print('\t');
      Serial1.print(gy86.acc.aftcal.y); Serial1.print('\t');
      Serial1.print(gy86.acc.aftcal.z); Serial1.print('\t');
      Serial1.print(gy86.acc.filter.x); Serial1.print('\t');
      Serial1.print(gy86.acc.filter.y); Serial1.print('\t');
      Serial1.print(gy86.acc.filter.z); Serial1.print('\t');
      Serial1.println();
      break;
    case 1034:
      Serial1.print(gy86.acc.origin.x); Serial1.print('\t');
```

```
      Serial1.print(gy86.acc.origin.y); Serial1.print('\t');
      Serial1.print(gy86.acc.origin.z); Serial1.print('\t');
      Serial1.print(gy86.acc.aftcal.x); Serial1.print('\t');
      Serial1.print(gy86.acc.aftcal.y); Serial1.print('\t');
      Serial1.print(gy86.acc.aftcal.z); Serial1.print('\t');
      Serial1.println();
      break;
    /*------------Accelerometer Calibration------------*/
    case 1041:    //level - z + g
      MPU6050AccCali(4);
      break;
    case 1042:    //back - z -g
      MPU6050AccCali(5);
      break;
    case 1043:    //nose up - x + g
      MPU6050AccCali(0);
      break;
    case 1044:    //nose down - x -g
      MPU6050AccCali(1);
      break;
    case 1045:    //left wing up - y +g
      MPU6050AccCali(2);
      break;
    case 1046:    //left wing down - y -g
      MPU6050AccCali(3);
      break;
    case 1047:    //reset
      AccPointRead();
      break;
    case 1048:
      for (uint8_t point = 0; point < 6; point++)
      {
        Serial1.print(Acc_Cali.accel_raw_ref[point][0]);
        Serial1.print('\t');
        Serial1.print(Acc_Cali.accel_raw_ref[point][1]);
        Serial1.print('\t');
        Serial1.print(Acc_Cali.accel_raw_ref[point][2]);
        Serial1.print('\t');
        Serial1.println();
      }
      comorder = 0;
      break;
    case 1049:
      Serial1.println("------------------------");
      for (uint8_t i = 0; i < 3; i++)
      {
        for (uint8_t j = 0; j < 3; j++)
        {
          Serial1.print(Acc_Cali.a[i][j]);
          Serial1.print('\t');
        }
        Serial1.println();
      }
      Serial1.println("------------------------");
      for (uint8_t i = 0; i < 3; i++)
      {
        for (uint8_t j = 0; j < 3; j++)
        {
          Serial1.print(Acc_Cali.T[i][j]);
          Serial1.print('\t');
        }
        Serial1.println();
      }
      Serial1.println("------------------------");
      comorder = 0;
      break;
```

```
        /*Attitude Estimation*/
        case 1051:
          Serial1.print(state.phi); Serial1.print('\t');
          Serial1.print(state.theta); Serial1.print('\t');
          Serial1.print(state.psi); Serial1.print('\t');
          Serial1.println();
          break;
        case 1052:
          Serial1.print(state.p); Serial1.print('\t');
          Serial1.print(state.q); Serial1.print('\t');
          Serial1.print(state.r); Serial1.print('\t');
          Serial1.println();
          break;
        case 1053:
          Serial1.print(q0); Serial1.print('\t');
          Serial1.print(q1); Serial1.print('\t');
          Serial1.print(q2); Serial1.print('\t');
          Serial1.print(q3); Serial1.print('\t');
          Serial1.println();
          break;
        case 1054:
          Serial1.print(state.ax); Serial1.print('\t');
          Serial1.print(state.ay); Serial1.print('\t');
          Serial1.print(state.az); Serial1.print('\t');
          Serial1.println();
          break;
        /*---------Linear Kalman Filter-----------*/
        case 1061:
          Serial1.print(LKF.acc_roll); Serial1.print('\t');
          Serial1.print(LKF.acc_pitch); Serial1.print('\t');
          Serial1.println();
          break;
        case 1062:
          Serial1.print(state.x, 4); Serial1.print('\t');
          Serial1.print(Otus.x, 4); Serial1.print('\t');
          Serial1.println();
          break;
        case 1063:
          Serial1.print(state.y, 4); Serial1.print('\t');
          Serial1.print(Otus.y, 4); Serial1.print('\t');
          Serial1.println();
          break;
        case 1064:
          Serial1.print(state.z, 4); Serial1.print('\t');
          Serial1.print(Otus.z, 4); Serial1.print('\t');
          Serial1.println();
          break;
        case 1065:
          Serial1.print(state.vx, 4); Serial1.print('\t');
          Serial1.print(Otus.vx, 4); Serial1.print('\t');
          Serial1.println();
          break;
        case 1066:
          Serial1.print(state.vy, 4); Serial1.print('\t');
          Serial1.print(Otus.vy, 4); Serial1.print('\t');
          Serial1.println();
          break;
        case 1067:
          Serial1.print(state.vz, 4); Serial1.print('\t');
          Serial1.print(Otus.vz, 4); Serial1.print('\t');
          Serial1.println();
          break;
        case 1101:
          Serial1.print(U1); Serial1.print('\t');
          Serial1.print(U2); Serial1.print('\t');
          Serial1.print(U3); Serial1.print('\t');
          Serial1.print(U4); Serial1.print('\t');
```

```
      Serial1.println("---------------------------------");
      break;
    case 1102:
      Serial1.print(PWM1); Serial1.print('\t');
      Serial1.print(PWM2); Serial1.print('\t');
      Serial1.print(PWM3); Serial1.print('\t');
      Serial1.print(PWM4); Serial1.print('\t');
      Serial1.println();
      break;
    /*------------------P Control------------------*/
    case 1201:
      Serial1.print(Target.p); Serial1.print('\t');
      Serial1.print(state.p); Serial1.print('\t');
      Serial1.println();
      break;
    case 1202:
      Serial1.print(Target.p_rad, 5); Serial1.print('\t');
      Serial1.print(state.p_rad, 5); Serial1.print('\t');
      Serial1.println();
      break;
    case 1203:
      Serial1.print("P_Kp =  "); Serial1.println(pvel.Kp, 4);
      Serial1.print("P_Ki =  "); Serial1.println(pvel.Ki, 4);
      Serial1.print("P_Kd =  "); Serial1.println(pvel.Kd, 4);
      comorder = 0;
      break;
    case 1204:
      Serial1.print(U2, 5); Serial1.print('\t');
      Serial1.print(state.p_rad, 5);
      Serial1.println();
      break;
    /*----------------Roll Control----------------*/
    case 1211:
      Serial1.print(Target.phi); Serial1.print('\t');
      Serial1.print(state.phi); Serial1.print('\t');
      Serial1.println();
      break;
    case 1212:
      Serial1.print(Target.phi_rad, 5); Serial1.print('\t');
      Serial1.print(state.phi_rad, 5); Serial1.print('\t');
      Serial1.println();
      break;
    case 1213:
      Serial1.print("Phi_Kp =  "); Serial1.println(phiang.Kp, 4);
      comorder = 0;
      break;

    /*---------------q Control--------------*/
    case 1221:
      Serial1.print(Target.q); Serial1.print('\t');
      Serial1.print(state.q); Serial1.print('\t');
      Serial1.println();
      break;
    case 1222:
      Serial1.print(Target.q_rad, 5); Serial1.print('\t');
      Serial1.print(state.q_rad, 5); Serial1.print('\t');
      Serial1.println();
      break;
    case 1223:
      Serial1.print("Q_Kp =  "); Serial1.println(qvel.Kp, 4);
      Serial1.print("Q_Ki =  "); Serial1.println(qvel.Ki, 4);
      Serial1.print("Q_Kd =  "); Serial1.println(qvel.Kd, 4);
      comorder = 0;
      break;
    case 1224:
      Serial1.print(U3, 5); Serial1.print('\t');
      Serial1.print(state.q_rad, 5);
```

```
          Serial1.println();
          break;
        /*----------------Pitch Control---------------*/
        case 1231:
          Serial1.print(Target.theta); Serial1.print('\t');
          Serial1.print(state.theta); Serial1.print('\t');
          Serial1.println();
          break;
        case 1232:
          Serial1.print(Target.theta_rad, 5); Serial1.print('\t');
          Serial1.print(state.theta_rad, 5); Serial1.print('\t');
          Serial1.println();
          break;
        case 1233:
          Serial1.print("Theta_Kp =  "); Serial1.println(thetaang.Kp, 4);
          comorder = 0;
          break;
        /*-------------------r Control------------------*/
        case 1241:
          Serial1.print(Target.r); Serial1.print('\t');
          Serial1.print(state.r); Serial1.print('\t');
          Serial1.println();
          break;
        case 1242:
          Serial1.print(Target.r_rad, 5); Serial1.print('\t');
          Serial1.print(state.r_rad, 5); Serial1.print('\t');
          Serial1.println();
          break;
        case 1243:
          Serial1.print("R_Kp =  "); Serial1.println(rvel.Kp, 6);
          Serial1.print("R_Ki =  "); Serial1.println(rvel.Ki, 6);
          Serial1.print("R_Kd =  "); Serial1.println(rvel.Kd, 6);
          comorder = 0;
          break;
        case 1244:
          Serial1.print(U4, 5); Serial1.print('\t');
          Serial1.print(state.r_rad, 5);
          Serial1.println();
          break;
        /*----------Yaw Control-----------------*/
        case 1251:
          Serial1.print(Target.psi); Serial1.print('\t');
          Serial1.print(state.psi); Serial1.print('\t');
          //Serial1.print(Psicon.Output[0]); Serial1.print('\t');
          Serial1.println();
          break;
        case 1252:
          Serial1.print(Target.psi_rad, 5); Serial1.print('\t');
          Serial1.print(state.psi_rad, 5); Serial1.print('\t');
          Serial1.println();
          break;
        case 1253:
          Serial1.print("Psi_Kp =  ");
          Serial1.println(psiang.Kp, 4);
          comorder = 0;
          break;
        case 1254:
          Serial1.println(Otus.yaw);
          break;
        /*----------Vz Control---------------*/
        case 1261:
          Serial1.print(Target.vz); Serial1.print('\t');
          Serial1.print(state.vz); Serial1.print('\t');
          Serial1.println();
          break;
        case 1262:
          Serial1.print("Z_Kp =  "); Serial1.println(Zalt.Kp);
```

```
          Serial1.print("Vz_Kp =  "); Serial1.println(Vzalt.Kp);
          Serial1.print("Vz_Ki =  "); Serial1.println(Vzalt.Ki);
          Serial1.print("Vz_Kd =  "); Serial1.println(Vzalt.Kd);
          comorder = 0;
          break;
        case 1263:
          Serial1.print(Vzcon.Output[0], 5); Serial1.print('\t');
          Serial1.print(state.vz, 5);
          Serial1.println();
          break;
        case 1264:
          Serial1.print(RCsignal.THROTTLE); Serial1.print('\t');
          Serial1.print(state.vz); Serial1.print('\t');
          Serial1.print(Target.vz); Serial1.print('\t');
          Serial1.print(state.z); Serial1.print('\t');
          Serial1.print(Target.z); Serial1.print('\t');
          Serial1.print(Vzcon.Output[0]); Serial1.print('\t');
          Serial1.print(U1);
          Serial1.println();
          break;
        case 1265:
          Serial1.print(state.flight_state[0]); Serial1.print('\t');
          Serial1.print(state.flight_state[1]); Serial1.print('\t');
          Serial1.print(state.flight_state[2]); Serial1.print('\t');
          Serial1.print(state.flight_state[3]); Serial1.print('\t');
          Serial1.print(state.takeoff_t); Serial1.print('\t');
          Serial1.print(state.landing_t); Serial1.print('\t');
          Serial1.print(U1); Serial1.print('\t');
          Serial1.println();
          break;
        /*-------------------VxVy Control------------------*/
        case 1271:
          Serial1.print("X_Kp =  "); Serial1.println(Xpos.Kp);
          Serial1.print("Y_Kp =  "); Serial1.println(Ypos.Kp);
          Serial1.print("Vx_Kp =  "); Serial1.println(Vxpos.Kp);
          Serial1.print("Vx_Kd =  "); Serial1.println(Vxpos.Kd);
          Serial1.print("Vy_Kp =  "); Serial1.println(Vypos.Kp);
          Serial1.print("Vy_Kd =  "); Serial1.println(Vypos.Kd);
          comorder = 0;
          break;
        case 1272:
          Serial1.print(state.vx); Serial1.print('\t');
          Serial1.print(Target.vx); Serial1.print('\t');
          Serial1.println();
          break;
        case 1273:
          Serial1.print(state.vy); Serial1.print('\t');
          Serial1.print(Target.vy); Serial1.print('\t');
          Serial1.println();
          break;
        case 1274:
          Serial1.print(state.vz); Serial1.print('\t');
          Serial1.print(Target.vz); Serial1.print('\t');
          Serial1.println();
          break;
        /*------------Input/Output---------------*/
        case 1280:
          Serial1.print(state.x); Serial1.print('\t');
          Serial1.print(Target.x); Serial1.print('\t');
          Serial1.println();
          break;
        case 1281:
          Serial1.print(state.y); Serial1.print('\t');
          Serial1.print(Target.y); Serial1.print('\t');
          Serial1.println();
          break;
```

```
        case 1282:
          Serial1.print(state.z); Serial1.print('\t');
          Serial1.print(Target.z); Serial1.print('\t');
          Serial1.println();
          break;
        case 1283:
          Serial1.print(Target.phi); Serial1.print('\t');
          Serial1.print(Target.theta); Serial1.print('\t');
          Serial1.println();
          break;

        /*--------------Plan----------------*/
        case 1290:
          Serial1.print(Target.plan_x); Serial1.print('\t');
          Serial1.print(Target.plan_y); Serial1.print('\t');
          Serial1.print(Target.plan_z); Serial1.print('\t');
          Serial1.print(Target.plan_psi); Serial1.print('\t');

          Serial1.print(Target.x); Serial1.print('\t');
          Serial1.print(Target.y); Serial1.print('\t');
          Serial1.print(Target.z); Serial1.print('\t');
          Serial1.print(Target.psi); Serial1.print('\t');
          Serial1.println();
          break;
        case 1301:
          Serial1.println();
          Serial1.println("////////////////PID PRAM//////////////");
          Serial1.println();
          Serial1.print("P_Kp =  "); Serial1.println(pvel.Kp);
          Serial1.print("P_Ki =  "); Serial1.println(pvel.Ki);
          Serial1.print("P_Kd =  "); Serial1.println(pvel.Kd);
          Serial1.print("Q_Kp =  "); Serial1.println(qvel.Kp);
          Serial1.print("Q_Ki =  "); Serial1.println(qvel.Ki);
          Serial1.print("Q_Kd =  "); Serial1.println(qvel.Kd);
          delay(10);
          Serial1.print("R_Kp =  "); Serial1.println(rvel.Kp);
          Serial1.print("R_Ki =  "); Serial1.println(rvel.Ki);
          Serial1.print("R_Kd =  "); Serial1.println(rvel.Kd);
          Serial1.print("Phi_Kp =  "); Serial1.println(phiang.Kp);
          Serial1.print("Theta_Kp =  "); Serial1.println(thetaang.Kp);
          Serial1.print("Psi_Kp =  "); Serial1.println(psiang.Kp);
          delay(10);
          Serial1.println();
          Serial1.print("Z_Kp =  "); Serial1.println(Zalt.Kp);
          Serial1.print("Vz_Kp =  "); Serial1.println(Vzalt.Kp);
          Serial1.print("Vz_Ki =  "); Serial1.println(Vzalt.Ki);
          Serial1.print("Vz_Kd =  "); Serial1.println(Vzalt.Kd);
          Serial1.println();
          Serial1.print("Xpos_Kp =  "); Serial1.println(Xpos.Kp);
          Serial1.print("Vxpos_Kp =  "); Serial1.println(Vxpos.Kp);
          Serial1.print("Vxpos_Ki =  "); Serial1.println(Vxpos.Ki);
          Serial1.print("Vxpos_Kd =  "); Serial1.println(Vxpos.Kd);
          delay(10);
          Serial1.print("Ypos_Kp =  "); Serial1.println(Ypos.Kp);
          Serial1.print("Vypos_Kp =  "); Serial1.println(Vypos.Kp);
          Serial1.print("Vypos_Ki =  "); Serial1.println(Vypos.Ki);
          Serial1.print("Vypos_Kd =  "); Serial1.println(Vypos.Kd);
          Serial1.println();
          Serial1.print("Gravity =  "); Serial1.println(EstimatedG);
          Serial1.println("/////////////////////////////////////");
          Serial1.println();
          comorder = 0;
          break;
        case 1411:
          Serial1.print(Otus.x); Serial1.print('\t');
```

```
        Serial1.print(Otus.y); Serial1.print('\t');
        Serial1.print(Otus.z); Serial1.print('\t');
        Serial1.println();
        break;
      case 1412:
        Serial1.print(Otus.vx); Serial1.print('\t');
        Serial1.print(Otus.vy); Serial1.print('\t');
        Serial1.print(Otus.vz); Serial1.print('\t');
        Serial1.println();
        break;
      case 1413:
        Serial1.print(Otus.roll); Serial1.print('\t');
        Serial1.print(Otus.pitch); Serial1.print('\t');
        Serial1.print(Otus.yaw); Serial1.print('\t');
        Serial1.println();
        break;
      case 1414:
        Serial1.print(q0); Serial1.print('\t');
        Serial1.print(q1); Serial1.print('\t');
        Serial1.print(q2); Serial1.print('\t');
        Serial1.print(q3); Serial1.print('\t');
        Serial1.print(Otus.q0); Serial1.print('\t');
        Serial1.print(Otus.q1); Serial1.print('\t');
        Serial1.print(Otus.q2); Serial1.print('\t');
        Serial1.print(Otus.q3); Serial1.print('\t');
        Serial1.println();
        break;
      case 1415:

        Serial.print(LKF.acc_roll); Serial.print('\t');
        Serial.print(LKF.acc_pitch); Serial.print('\t');
        Serial.print(Otus.yaw); Serial.print('\t');

        Serial.print(state.p); Serial.print('\t');
        Serial.print(state.q); Serial.print('\t');
        Serial.print(state.r); Serial.print('\t');
        Serial.println();

        break;
      case 1501:
        for (uint8_t i = 0; i < 30; i++)
        {
          Serial1.print(VRsensor1.ringbuffer[i]);
          Serial1.println('\t');
        }
        comorder = 0;
        break;
      case 1998:
        Serial1.print(xbee_length); Serial1.println('\t');
        break;
      case 1999:
        Xbee_Packet();
        break;
      case 2001:
        Serial1.print(voltageavg); Serial1.println('\t');
        break;

    }
    Xbee_timer = 0;
  }
}
void _Copter::Xbee_Packet()
{
  /*-------------------------Test-------------------------*/
  uint8_t data_length = 21;
  uint8_t packet[data_length];
  packet[0] = 45;
```

```
  packet[1] = 50;
  packet[2] = 55;

  packet[3] = 71;
  xbee_tmp.f = state.phi;
  packet[4] = xbee_tmp.ul[0];
  packet[5] = xbee_tmp.ul[1];
  packet[6] = xbee_tmp.ul[2];
  packet[7] = xbee_tmp.ul[3];

  packet[8] = 72;
  xbee_tmp.f = state.theta;
  packet[9] = xbee_tmp.ul[0];
  packet[10] = xbee_tmp.ul[1];
  packet[11] = xbee_tmp.ul[2];
  packet[12] = xbee_tmp.ul[3];

  packet[13] = 73;
  xbee_tmp.f = state.theta;
  packet[14] = xbee_tmp.ul[0];
  packet[15] = xbee_tmp.ul[1];
  packet[16] = xbee_tmp.ul[2];
  packet[17] = xbee_tmp.ul[3];

  packet[18] = 100;
  packet[19] = 125;
  packet[20] = 150;
  Serial1.write(packet, data_length);
}
void _Copter::Xbee_receive(int16_t order, uint16_t stmp)
{
  float pram = (float)stmp;
  switch (order)
  {
    case 1011:
      pvel.Kp =  pram / 10000;
      EEPROM.write(10, stmp & 0b11111111);
      EEPROM.write(11, stmp >> 8);
      Serial1.print("P_Kp =  "); Serial1.print(pvel.Kp);
      Serial1.println();
      break;
    case 1012:
      pvel.Ki = pram / 1000;
      EEPROM.write(12, stmp & 0b11111111);
      EEPROM.write(13, stmp >> 8);
      Serial1.print("P_Ki =  "); Serial1.print(pvel.Ki);
      Serial1.println();
      break;
    case 1013:
      pvel.Kd = pram / 100000;
      EEPROM.write(14, stmp & 0b11111111);
      EEPROM.write(15, stmp >> 8);
      Serial1.print("P_Kd =  "); Serial1.print(pvel.Kd);
      Serial1.println();
      break;
    case 1021:
      qvel.Kp = pram / 10000;
      EEPROM.write(18, stmp & 0b11111111);
      EEPROM.write(19, stmp >> 8);
      Serial1.print("Q_Kp =  "); Serial1.print(qvel.Kp);
      Serial1.println();
      break;
    case 1022:

      qvel.Ki = pram / 1000;
      EEPROM.write(20, stmp & 0b11111111);
```

```
      EEPROM.write(21, stmp >> 8);
      Serial1.print("Q_Ki =  "); Serial1.print(qvel.Ki);
      Serial1.println();
      break;
    case 1023:

      qvel.Kd = pram / 100000;
      EEPROM.write(22, stmp & 0b11111111);
      EEPROM.write(23, stmp >> 8);
      Serial1.print("Q_Kd =  "); Serial1.print(qvel.Kd);
      Serial1.println();
      break;
    case 1031:

      rvel.Kp = pram / 100000;
      EEPROM.write(26, stmp & 0b11111111);
      EEPROM.write(27, stmp >> 8);
      Serial1.print("R_Kp =  "); Serial1.print(rvel.Kp);
      Serial1.println();
      break;
    case 1032:

      rvel.Ki = pram / 10000;
      EEPROM.write(28, stmp & 0b11111111);
      EEPROM.write(29, stmp >> 8);
      Serial1.print("R_Ki =  "); Serial1.print(rvel.Ki);
      Serial1.println();
      break;
    case 1033:

      rvel.Kd = pram / 1000000;
      EEPROM.write(30, stmp & 0b11111111);
      EEPROM.write(31, stmp >> 8);
      Serial1.print("R_Kd =  "); Serial1.print(rvel.Kd);
      Serial1.println();
      break;
    case 1051:

      phiang.Kp = pram / 100;
      EEPROM.write(16, stmp & 0b11111111);
      EEPROM.write(17, stmp >> 8);
      Serial1.print("Phi_Kp =  "); Serial1.print(phiang.Kp);
      Serial1.println();
      break;
    case 1052:

      thetaang.Kp = pram / 100;
      EEPROM.write(24, stmp & 0b11111111);
      EEPROM.write(25, stmp >> 8);
      Serial1.print("Theta_Kp =  "); Serial1.print(thetaang.Kp);
      Serial1.println();
      break;
    case 1053:

      psiang.Kp = pram / 100;
      EEPROM.write(32, stmp & 0b11111111);
      EEPROM.write(33, stmp >> 8);
      Serial1.print("Psi_Kp =  "); Serial1.print(psiang.Kp);
      Serial1.println();
      break;
    case 1071:

      Zalt.Kp = pram / 100;
      EEPROM.write(40, stmp & 0b11111111);
      EEPROM.write(41, stmp >> 8);
      Serial1.print("Z_Kp =  "); Serial1.print(Zalt.Kp);
      Serial1.println();
```

```
      break;
    case 1081:

      Vzalt.Kp = pram / 100;
      EEPROM.write(42, stmp & 0b11111111);
      EEPROM.write(43, stmp >> 8);
      Serial1.print("Vz_Kp =  "); Serial1.print(Vzalt.Kp);
      Serial1.println();
      break;
    case 1082:

      Vzalt.Ki = pram / 100;
      EEPROM.write(44, stmp & 0b11111111);
      EEPROM.write(45, stmp >> 8);
      Serial1.print("Vz_Ki =  "); Serial1.print(Vzalt.Ki);
      Serial1.println();
      break;
    case 1083:

      Vzalt.Kd = pram / 100;
      EEPROM.write(46, stmp & 0b11111111);
      EEPROM.write(47, stmp >> 8);
      Serial1.print("Vz_Kd =  "); Serial1.print(Vzalt.Kd);
      Serial1.println();
      break;
    case 1091:

      Xpos.Kp = pram / 1000;
      EEPROM.write(50, stmp & 0b11111111);
      EEPROM.write(51, stmp >> 8);
      Serial1.print("Xpos_Kp =  "); Serial1.print(Xpos.Kp);
      Serial1.println();
      break;
    case 1092:

      Vxpos.Kp = pram / 1000;
      EEPROM.write(52, stmp & 0b11111111);
      EEPROM.write(53, stmp >> 8);
      Serial1.print("Vxpos_Kp =  "); Serial1.print(Vxpos.Kp);
      Serial1.println();
      break;
    case 1093:

      Vxpos.Ki = pram / 1000;
      EEPROM.write(54, stmp & 0b11111111);
      EEPROM.write(55, stmp >> 8);
      Serial1.print("Vxpos_Ki =  "); Serial1.print(Vxpos.Ki);
      Serial1.println();
      break;
    case 1094:

      Vxpos.Kd = pram / 1000;
      EEPROM.write(56, stmp & 0b11111111);
      EEPROM.write(57, stmp >> 8);
      Serial1.print("Vxpos_Kd =  "); Serial1.print(Vxpos.Kd);
      Serial1.println();
      break;
    case 1095:

      Ypos.Kp = pram / 1000;
      EEPROM.write(60, stmp & 0b11111111);
      EEPROM.write(61, stmp >> 8);
      Serial1.print("Ypos_Kp =  "); Serial1.print(Ypos.Kp);
      Serial1.println();
      break;
    case 1096:

```

```
      Vypos.Kp = pram / 1000;
      EEPROM.write(62, stmp & 0b11111111);
      EEPROM.write(63, stmp >> 8);
      Serial1.print("Vypos_Kp =  "); Serial1.print(Vypos.Kp);
      Serial1.println();
      break;
    case 1097:

      Vypos.Ki = pram / 1000;
      EEPROM.write(64, stmp & 0b11111111);
      EEPROM.write(65, stmp >> 8);
      Serial1.print("Vypos_Ki =  "); Serial1.print(Vypos.Ki);
      Serial1.println();
      break;
    case 1098:

      Vypos.Kd = pram / 1000;
      EEPROM.write(66, stmp & 0b11111111);
      EEPROM.write(67, stmp >> 8);
      Serial1.print("Vypos_Kd =  "); Serial1.print(Vypos.Kd);
      Serial1.println();
      break;
    case 1100:

      EstimatedG = pram / 1000;
      EEPROM.write(70, stmp & 0b11111111);
      EEPROM.write(71, stmp >> 8);
      Serial1.print("Gravity =  "); Serial1.print(EstimatedG);
      Serial1.println();
      break;
  }

}
```

Mark3 Flight Controller Code - Library of Interrupt

```
#include "Arduino.h"
void InitComm();
void ISR1();
void ISR2();
/*------------SBUS----------*/
void RC_refine();
void OtusSerial();
void ComOrder();
void PramOrder();
void OtusMovingAverage();
uint16_t channels[16];
uint8_t failSafe;
uint16_t lostFrames = 0;

float moving_ave[13][5],sum_ave[13];

/*------------------------------*/

/*----------Lighthouse-----------*/
#define IRsensor1 23
#define IRsensor2 22
#define IRsensor3 17
#define IRsensor4 16
#define IRsensor5 24
#define IRsensor6 26
#define IRsensor7 27
#define IRsensor8 28
/*--------------------------*/
```

Mark3 Flight Controller Code - Interrupt

```
void InitComm()
{
  Serial1.begin(230400);
  Serial1.begin(230400);
  R9DS.begin();
  pinMode(LED0, OUTPUT);
  pinMode(LED1, OUTPUT);
  pinMode(LED2, OUTPUT);
  digitalWrite(LED0, HIGH);
  digitalWrite(LED1, LOW);
  digitalWrite(LED2, LOW);
  Copter.RCsignal.ROLL = 1000;
  Copter.RCsignal.PITCH = 1000;
  Copter.RCsignal.THROTTLE = 250;
  Copter.RCsignal.YAW = 1000;
  Copter.RCsignal.MODE = 250;
  Copter.RCsignal.SWITCH = 1700;
}
void RC_refine()
{
  if (R9DS.read(channels, &failSafe, &lostFrames))
  {
    Copter.RCsignal.ROLL = channels[0];
    Copter.RCsignal.PITCH = channels[1];
    Copter.RCsignal.THROTTLE = channels[2];
    Copter.RCsignal.YAW = channels[3];
    Copter.RCsignal.MODE = channels[4];
    Copter.RCsignal.SWITCH = channels[5];
    Copter.RCsignal.CH7 = channels[6];
    Copter.RCsignal.CH8 = channels[7];
    Copter.RCsignal.CH9 = channels[8];
    Copter.RCsignal.CH10 = channels[9];
  }
  if (Copter.Ctrl_timer1 == 2)
  {
    Copter.Otus_Clear();
    String comdata = "";
    Copter.xbeetime1 = 0;
    while (Serial1.available() > 0)
      comdata += char(Serial1.read());
    Copter.xbee_length = comdata.length();
    for (int i = 0; i < Copter.xbee_length; i++)
      Copter.xbee_data[100 + i] = comdata[i];
    if (Copter.xbee_length == 4)
    {
      //Copter.comorder = 0;
      short com_tmp = 0;
      for (uint8_t i = 0; i < 4; i++)
        com_tmp = com_tmp * 10 + (comdata[i] - '0');
      //ComOrder();
      /*
        Serial1.println(uint8_t(comdata[0]));
        Serial1.println(uint8_t(comdata[1]));
        Serial1.println(uint8_t(comdata[2]));
        Serial1.println(uint8_t(comdata[3]));
        Serial1.println(Copter.xbee_length);
        Serial1.println(Copter.comorder);
      */
      //Serial1.flush();
      if (com_tmp >= 1000 && com_tmp <= 5000)
        Copter.comorder = com_tmp;
    }
```

```
    else if (Copter.xbee_length < 12
            && Copter.xbee_length > 5
            && (Copter.xbee_data[100 + 0] == '&'
                && Copter.xbee_data[100 + 5] == '_'
                && Copter.xbee_data[100 + Copter.xbee_length - 1] == '*'))
      PramOrder();
    else if (Copter.xbee_length >= 20)
    {
      OtusSerial();
      OtusMovingAverage();
      //Serial1.println(Copter.xbee_length);
    }
  }
}
void OtusMovingAverage()
{
  Copter.Otus.x =
    Copter.data_limitation(Copter.Otus.x, -10.0, 10.0);
  Copter.Otus.y =
    Copter.data_limitation(Copter.Otus.y, -10.0, 10.0);
  Copter.Otus.z =
    Copter.data_limitation(Copter.Otus.z, -5.0, 10.0);
  Copter.Otus.vx =
    Copter.data_limitation(Copter.Otus.vx, -50.0, 50.0);
  Copter.Otus.vy =
    Copter.data_limitation(Copter.Otus.vy, -50.0, 50.0);
  Copter.Otus.vz =
    Copter.data_limitation(Copter.Otus.vz, -50.0, 50.0);
  Copter.Otus.roll =
    Copter.data_limitation(Copter.Otus.roll, -180.0, 180.0);
  Copter.Otus.pitch =
    Copter.data_limitation(Copter.Otus.pitch, -90.0, 90.0);
  Copter.Otus.yaw =
    Copter.data_limitation(Copter.Otus.yaw, -180.0, 180.0);
  Copter.Otus.q0 =
    Copter.data_limitation(Copter.Otus.q0, -1.0, 1.0);
  Copter.Otus.q1 =
    Copter.data_limitation(Copter.Otus.q1, -1.0, 1.0);
  Copter.Otus.q2 =
    Copter.data_limitation(Copter.Otus.q2, -1.0, 1.0);
  Copter.Otus.q3 =
    Copter.data_limitation(Copter.Otus.q3, -1.0, 1.0);

  Copter.Target.plan_x =
    Copter.data_limitation(Copter.Target.plan_x, -1.5, 1.5);
  Copter.Target.plan_y =
    Copter.data_limitation(Copter.Target.plan_y, -1.5, 1.5);
  Copter.Target.plan_z =
    Copter.data_limitation(Copter.Target.plan_z, 0, 2.2);
  Copter.Target.plan_psi =
    Copter.data_limitation(Copter.Target.plan_psi, -180, 180);
  /*------------Moving Average------------*/
  for (uint8_t j = 0; j < 13 ; j++)
    sum_ave[j] = sum_ave[j] - moving_ave[j][4] / 5;
  for (uint8_t i = 4; i > 0; i--)
    for (uint8_t j = 0; j < 13 ; j++)
      moving_ave[j][i] = moving_ave[j][i - 1];

  moving_ave[0][0] = Copter.Otus.x;
  moving_ave[1][0] = Copter.Otus.y;
  moving_ave[2][0] = Copter.Otus.z;

  moving_ave[3][0] = Copter.Otus.vx;
  moving_ave[4][0] = Copter.Otus.vy;
  moving_ave[5][0] = Copter.Otus.vz;

```

```
  moving_ave[6][0] = Copter.Otus.roll;
  moving_ave[7][0] = Copter.Otus.pitch;
  moving_ave[8][0] = Copter.Otus.yaw;

  moving_ave[9][0] = Copter.Otus.q0;
  moving_ave[10][0] = Copter.Otus.q1;
  moving_ave[11][0] = Copter.Otus.q2;
  moving_ave[12][0] = Copter.Otus.q3;

  for (uint8_t j = 0; j < 13 ; j++)
    sum_ave[j] = sum_ave[j] + moving_ave[j][0] / 5;

  Copter.Otus.x = sum_ave[0];
  Copter.Otus.y = sum_ave[1];
  Copter.Otus.z = sum_ave[2];

  Copter.Otus.vx = sum_ave[3];
  Copter.Otus.vy = sum_ave[4];
  Copter.Otus.vz = sum_ave[5];

  Copter.Otus.q0 = sum_ave[9];
  Copter.Otus.q1 = sum_ave[10];
  Copter.Otus.q2 = sum_ave[11];
  Copter.Otus.q3 = sum_ave[12];
}
void OtusSerial()
{
  for (int16_t i = 100 + 3; i < 100 + Copter.xbee_length - 5; i++)
  {
    switch (Copter.xbee_data[i])
    {
      case 21:
        if (Copter.xbee_data[i + 1] == 101)
        {
          if ((Copter.xbee_data[i - 1] == 131
               && Copter.xbee_data[i - 2] == 121
               && Copter.xbee_data[i - 3] == 111)
              && (Copter.xbee_data[i + 6] == 22
                  && Copter.xbee_data[i + 7] == 102))
          {
            Copter.state.update_x = 1;
            Copter.otus_tmp.ul =
              (Copter.xbee_data[i + 5] << 24)
              | (Copter.xbee_data[i + 4] << 16)
              | (Copter.xbee_data[i + 3] << 8)
              | Copter.xbee_data[i + 2];
            Copter.Otus.x = Copter.otus_tmp.f;
          }
        }
        break;
      case 22:
        if (Copter.xbee_data[i + 1] == 102)
        {
          if ((Copter.xbee_data[i - 6] == 21
               && Copter.xbee_data[i - 5] == 101)
              && (Copter.xbee_data[i + 6] == 23
                  && Copter.xbee_data[i + 7] == 103))
          {
            Copter.state.update_y = 1;
            Copter.otus_tmp.ul =
              (Copter.xbee_data[i + 5] << 24)
              | (Copter.xbee_data[i + 4] << 16)
              | (Copter.xbee_data[i + 3] << 8)
              | Copter.xbee_data[i + 2];
            Copter.Otus.y = Copter.otus_tmp.f;
          }
```

```
      }
      break;
    case 23:
      if (Copter.xbee_data[i + 1] == 103)
      {
        if ((Copter.xbee_data[i - 6] == 22
             && Copter.xbee_data[i - 5] == 102)
            && (Copter.xbee_data[i + 6] == 24
                && Copter.xbee_data[i + 7] == 104))
        {
          Copter.state.update_z = 1;
          Copter.otus_tmp.ul =
            (Copter.xbee_data[i + 5] << 24)
            | (Copter.xbee_data[i + 4] << 16)
            | (Copter.xbee_data[i + 3] << 8)
            | Copter.xbee_data[i + 2];
          Copter.Otus.z = Copter.otus_tmp.f;
        }
      }
      break;
    case 24:
      if (Copter.xbee_data[i + 1] == 104)
      {
        if ((Copter.xbee_data[i - 6] == 23
             && Copter.xbee_data[i - 5] == 103)
            && (Copter.xbee_data[i + 6] == 25
                && Copter.xbee_data[i + 7] == 105))
        {
          Copter.state.update_vx = 1;
          Copter.otus_tmp.ul =
            (Copter.xbee_data[i + 5] << 24)
            | (Copter.xbee_data[i + 4] << 16)
            | (Copter.xbee_data[i + 3] << 8)
            | Copter.xbee_data[i + 2];
          Copter.Otus.vx = Copter.otus_tmp.f;
        }
      }
      break;
    case 25:
      if (Copter.xbee_data[i + 1] == 105)
      {
        if ((Copter.xbee_data[i - 6] == 24
             && Copter.xbee_data[i - 5] == 104)
            && (Copter.xbee_data[i + 6] == 26
                && Copter.xbee_data[i + 7] == 106))
        {
          Copter.state.update_vy = 1;
          Copter.otus_tmp.ul =
            (Copter.xbee_data[i + 5] << 24)
            | (Copter.xbee_data[i + 4] << 16)
            | (Copter.xbee_data[i + 3] << 8)
            | Copter.xbee_data[i + 2];
          Copter.Otus.vy = Copter.otus_tmp.f;
        }
      }
      break;
    case 26:
      if (Copter.xbee_data[i + 1] == 106)
      {
        if ((Copter.xbee_data[i - 6] == 25
             && Copter.xbee_data[i - 5] == 105)
            && (Copter.xbee_data[i + 6] == 27
                && Copter.xbee_data[i + 7] == 107))
        {
          Copter.state.update_vz = 1;
          Copter.otus_tmp.ul =
```

```
            (Copter.xbee_data[i + 5] << 24)
            | (Copter.xbee_data[i + 4] << 16)
            | (Copter.xbee_data[i + 3] << 8)
            | Copter.xbee_data[i + 2];
          Copter.Otus.vz = Copter.otus_tmp.f;
        }
      }
      break;
    case 27:
      if (Copter.xbee_data[i + 1] == 107)
      {
        if ((Copter.xbee_data[i - 6] == 26
             && Copter.xbee_data[i - 5] == 106)
            && (Copter.xbee_data[i + 6] == 28
                && Copter.xbee_data[i + 7] == 108))
        {
          Copter.state.update_psi = 1;
          Copter.otus_tmp.ul =
            (Copter.xbee_data[i + 5] << 24)
            | (Copter.xbee_data[i + 4] << 16)
            | (Copter.xbee_data[i + 3] << 8)
            | Copter.xbee_data[i + 2];
          Copter.Otus.yaw = Copter.otus_tmp.f;

          Copter.Otus.yaw_sin = sin(Copter.Otus.yaw);
          Copter.Otus.yaw_cos = cos(Copter.Otus.yaw);
        }
      }
      break;
    case 28:
      if (Copter.xbee_data[i + 1] == 108)
      {
        if ((Copter.xbee_data[i - 6] == 27
             && Copter.xbee_data[i - 5] == 107)
            && (Copter.xbee_data[i + 6] == 29
                && Copter.xbee_data[i + 7] == 109))
        {
          Copter.state.update_theta = 1;
          Copter.otus_tmp.ul =
            (Copter.xbee_data[i + 5] << 24)
            | (Copter.xbee_data[i + 4] << 16)
            | (Copter.xbee_data[i + 3] << 8)
            | Copter.xbee_data[i + 2];
          Copter.Otus.pitch = Copter.otus_tmp.f;
        }
      }
      break;
    case 29:
      if (Copter.xbee_data[i + 1] == 109)
      {
        if ((Copter.xbee_data[i - 6] == 28
             && Copter.xbee_data[i - 5] == 108)
            && (Copter.xbee_data[i + 6] == 30
                && Copter.xbee_data[i + 7] == 110))
        {
          Copter.state.update_phi = 1;
          Copter.otus_tmp.ul =
            (Copter.xbee_data[i + 5] << 24)
            | (Copter.xbee_data[i + 4] << 16)
            | (Copter.xbee_data[i + 3] << 8)
            | Copter.xbee_data[i + 2];
          Copter.Otus.roll = Copter.otus_tmp.f;
        }
      }
      break;
    case 31:
```

```
          if (Copter.xbee_data[i + 1] == 111)
          {
            if ((Copter.xbee_data[i - 6] == 29
                 && Copter.xbee_data[i - 5] == 109)
                && (Copter.xbee_data[i + 6] == 32
                    && Copter.xbee_data[i + 7] == 112))
            {
              Copter.otus_tmp.ul =
                (Copter.xbee_data[i + 5] << 24)
                | (Copter.xbee_data[i + 4] << 16)
                | (Copter.xbee_data[i + 3] << 8)
                | Copter.xbee_data[i + 2];
              Copter.Otus.q0 = Copter.otus_tmp.f;
              Copter.Otus.q0 *= -1;
            }
          }
          break;
        case 32:
          if (Copter.xbee_data[i + 1] == 112)
          {
            if ((Copter.xbee_data[i - 6] == 31
                 && Copter.xbee_data[i - 5] == 111)
                && (Copter.xbee_data[i + 6] == 33
                    && Copter.xbee_data[i + 7] == 113))
            {
              Copter.otus_tmp.ul =
                (Copter.xbee_data[i + 5] << 24)
                | (Copter.xbee_data[i + 4] << 16)
                | (Copter.xbee_data[i + 3] << 8)
                | Copter.xbee_data[i + 2];
              Copter.Otus.q1 = Copter.otus_tmp.f;
              Copter.Otus.q1 *= -1;
            }
          }
          break;
        case 33:
          if (Copter.xbee_data[i + 1] == 113)
          {
            if ((Copter.xbee_data[i - 6] == 32
                 && Copter.xbee_data[i - 5] == 112)
                && (Copter.xbee_data[i + 6] == 34
                    && Copter.xbee_data[i + 7] == 114))
            {
              Copter.otus_tmp.ul =
                (Copter.xbee_data[i + 5] << 24)
                | (Copter.xbee_data[i + 4] << 16)
                | (Copter.xbee_data[i + 3] << 8)
                | Copter.xbee_data[i + 2];
              Copter.Otus.q2 = Copter.otus_tmp.f;
            }
          }
          break;
        case 34:
          if (Copter.xbee_data[i + 1] == 114)
          {
            if ((Copter.xbee_data[i - 6] == 33
                 && Copter.xbee_data[i - 5] == 113)
                && (Copter.xbee_data[i + 6] == 35
                    && Copter.xbee_data[i + 7] == 115))
            {
              Copter.otus_tmp.ul =
                (Copter.xbee_data[i + 5] << 24)
                | (Copter.xbee_data[i + 4] << 16)
                | (Copter.xbee_data[i + 3] << 8)
                | Copter.xbee_data[i + 2];
              Copter.Otus.q3 = Copter.otus_tmp.f;
```

```
        }
      }
      break;
    case 35:
      if (Copter.xbee_data[i + 1] == 115)
      {
        if ((Copter.xbee_data[i - 6] == 34
             && Copter.xbee_data[i - 5] == 114)
            && (Copter.xbee_data[i + 6] == 36
                && Copter.xbee_data[i + 7] == 116))
        {
          Copter.otus_tmp.ul =
            (Copter.xbee_data[i + 5] << 24)
            | (Copter.xbee_data[i + 4] << 16)
            | (Copter.xbee_data[i + 3] << 8)
            | Copter.xbee_data[i + 2];
          Copter.Target.plan_x = Copter.otus_tmp.f;
        }
      }
      break;
    case 36:
      if (Copter.xbee_data[i + 1] == 116)
      {
        if ((Copter.xbee_data[i - 6] == 35
             && Copter.xbee_data[i - 5] == 115)
            && (Copter.xbee_data[i + 6] == 37
                && Copter.xbee_data[i + 7] == 117))
        {
          Copter.otus_tmp.ul =
            (Copter.xbee_data[i + 5] << 24)
            | (Copter.xbee_data[i + 4] << 16)
            | (Copter.xbee_data[i + 3] << 8)
            | Copter.xbee_data[i + 2];
          Copter.Target.plan_y = Copter.otus_tmp.f;
        }
      }
      break;
    case 37:
      if (Copter.xbee_data[i + 1] == 117)
      {
        if ((Copter.xbee_data[i - 6] == 36
             && Copter.xbee_data[i - 5] == 116)
            && (Copter.xbee_data[i + 6] == 38
                && Copter.xbee_data[i + 7] == 118))
        {
          Copter.otus_tmp.ul =
            (Copter.xbee_data[i + 5] << 24)
            | (Copter.xbee_data[i + 4] << 16)
            | (Copter.xbee_data[i + 3] << 8)
            | Copter.xbee_data[i + 2];
          Copter.Target.plan_z = Copter.otus_tmp.f;
        }
      }
      break;
    case 38:
      if (Copter.xbee_data[i + 1] == 118)
      {
        if ((Copter.xbee_data[i - 6] == 37
             && Copter.xbee_data[i - 5] == 117)
            && (Copter.xbee_data[i + 6] == 30
                && Copter.xbee_data[i + 7] == 110))
        {
          Copter.otus_tmp.ul =
            (Copter.xbee_data[i + 5] << 24)
            | (Copter.xbee_data[i + 4] << 16)
            | (Copter.xbee_data[i + 3] << 8)
```

```
              | Copter.xbee_data[i + 2];
            Copter.Target.plan_psi = Copter.otus_tmp.f;
          }
        }
        break;
      case 30:
        if (Copter.xbee_data[i + 1] == 110)
        {
          if ((Copter.xbee_data[i - 6] == 38
               && Copter.xbee_data[i - 5] == 118)
              && (Copter.xbee_data[i + 4] == 50
                  && Copter.xbee_data[i + 5] == 75
                  && Copter.xbee_data[i + 6] == 100))
          {
            Copter.state.comm =
              (Copter.xbee_data[i + 3] << 8)
              | Copter.xbee_data[i + 2];
            if (Copter.state.comm >= 1000
                && Copter.state.comm <= 4000)
              Copter.comorder = Copter.state.comm;
            //Serial.println(Copter.state.comm);
          }
        }
        break;
    }
  }
}
void PramOrder()
{
  int16_t order = 0, num;
  uint16_t stmp = 0;
  for (num = 1; num <= 4; num++)
    order = order * 10 + (Copter.xbee_data[100 + num] - '0');
  for (num = 6; num < (Copter.xbee_length - 1); num++)
    stmp = stmp * 10 + (Copter.xbee_data[100 + num] - '0');
  Copter.Xbee_receive(order, stmp);
}
```

Mark3 Flight Controller Code - Motor Driver

```
#include "Copter.h"
void _Copter::Motor_init()
{
  voltageavg = (float)analogRead(A14) * 0.013841;

  pinMode(motor1, OUTPUT);
  pinMode(motor2, OUTPUT);
  pinMode(motor3, OUTPUT);
  pinMode(motor4, OUTPUT);

  analogWriteFrequency(motor1, 400);
  analogWriteFrequency(motor2, 400);
  analogWriteFrequency(motor3, 400);
  analogWriteFrequency(motor4, 400);
  analogWriteResolution(16);

  /*-----------OutFunction.m-----------*/
  InputK1 = 130958.617;
  InputK2 = 1290232.68;
  InputK3 = 1728826.63;
  InputK4 = 10113268.61;
  Motor_stop();
}
void _Copter::InputTransform()
{
```

```
  omega12 = InputK1 * U1 - InputK2 * U2 +
            InputK3 * U3 + InputK4 * U4;
  omega22 = InputK1 * U1 + InputK2 * U2 -
            InputK3 * U3 + InputK4 * U4;
  omega32 = InputK1 * U1 + InputK2 * U2 +
            InputK3 * U3 - InputK4 * U4;
  omega42 = InputK1 * U1 - InputK2 * U2 -
            InputK3 * U3 - InputK4 * U4;
  if (omega12 < 0) omega12 = 0;
  if (omega22 < 0) omega22 = 0;
  if (omega32 < 0) omega32 = 0;
  if (omega42 < 0) omega42 = 0;
  omega1 = sqrt(omega12);
  omega2 = sqrt(omega22);
  omega3 = sqrt(omega32);
  omega4 = sqrt(omega42);
  MotorModel(omega1, omega2, omega3, omega4);
}
void _Copter::MotorModel(double omega1,
                         double omega2, double omega3, double omega4)
{
  double param_a = 1166.0, param_b = 5393,
         param_c = 299600, param_d = 1544, param_e = 894.5;
  PWM1 = (omega1 * omega1 + param_b * omega1 + param_c) /
         (param_a * voltageavg + param_d) + param_e;
  PWM2 = (omega2 * omega2 + param_b * omega2 + param_c) /
         (param_a * voltageavg + param_d) + param_e;
  PWM3 = (omega3 * omega3 + param_b * omega3 + param_c) /
         (param_a * voltageavg + param_d) + param_e;
  PWM4 = (omega4 * omega4 + param_b * omega4 + param_c) /
         (param_a * voltageavg + param_d) + param_e;
  if (PWM1 < 1055) PWM1 = 1055;
  if (PWM2 < 1055) PWM2 = 1055;
  if (PWM3 < 1055) PWM3 = 1055;
  if (PWM4 < 1055) PWM4 = 1055;
  if (PWM1 > 1550) PWM1 = 1550;
  if (PWM2 > 1550) PWM2 = 1550;
  if (PWM3 > 1550) PWM3 = 1550;
  if (PWM4 > 1550) PWM4 = 1550;
  MotorRun();
}
void _Copter::Motor_stop()
{
  PWM = pwm_factor * 950;
  analogWrite(motor1, PWM);
  analogWrite(motor2, PWM);
  analogWrite(motor3, PWM);
  analogWrite(motor4, PWM);
}
void _Copter::MotorRun()
{
  if ((!flag.turnoff) && (flag.ARMED == 2))
  {
    float inputpwm1 = pwm_factor * PWM1;
    float inputpwm2 = pwm_factor * PWM2;
    float inputpwm3 = pwm_factor * PWM3;
    float inputpwm4 = pwm_factor * PWM4;
    analogWrite(motor1, inputpwm1);
    analogWrite(motor2, inputpwm2);
    analogWrite(motor3, inputpwm3);
    analogWrite(motor4, inputpwm4);
  }
  else
    Motor_stop();
}
```

Mark3 Flight Controller Code - Attitude Control

```
#include "Copter.h"
void _Copter::AttitudeControl()
{
  if (Ctrl_timer1 >= 4)
  {
    /*-----------------Roll Command-----------------*/
    if (flag.mode == Attitude_mode || flag.mode == Altitude_mode)
    {
      if (RCsignal.ROLL < 900)
      {
        //if (RCsignal.CH7 < 1000)
        Target.phi = (float)(RCsignal.ROLL - 900) / 40;
        //else
        //Target.phi = -20;
      }
      if (RCsignal.ROLL > 1100)
      {
        //if (RCsignal.CH7 < 1000)
        Target.phi = (float)(RCsignal.ROLL - 1100) / 40;
        //else
        //Target.phi = 20;
      }
      if (RCsignal.ROLL >= 900 && RCsignal.ROLL <= 1100)
        Target.phi = 0;
      /*-----------------Pitch Command-----------------*/
      if (RCsignal.PITCH < 900)
      {
        //if (RCsignal.CH7 < 1000)
        Target.theta = (float)(RCsignal.PITCH - 900) / 40;
        //else
        //Target.theta = -15;
      }
      if (RCsignal.PITCH > 1100)
      {
        //if (RCsignal.CH7 < 1000)
        Target.theta = (float)(RCsignal.PITCH - 1100) / 40;
        //else
        //Target.theta = 15;
      }
      if (RCsignal.PITCH >= 900 && RCsignal.PITCH <= 1100)
        Target.theta = 0;
    }

    Target.phi = data_limitation(Target.phi, -25, 25);
    Target.theta = data_limitation(Target.theta, -25, 25);
    Target.phi_rad = Rad(Target.phi);
    Target.theta_rad = Rad(Target.theta);
    /*-----------------Yaw Command-----------------*/
    if (RCsignal.THROTTLE > 350)
    {
      if (LockYaw != 1)
      {
        LockYaw = 1;
        Target.psi = state.psi;
      }
    }
    else {
      if (U1 < 0.5)
      {
        LockYaw = 0;
        Target.psi = state.psi;
      }
```

```
    }
    if ((RCsignal.YAW > 1075) || (RCsignal.YAW < 925))
    {
      if (RCsignal.YAW > 1075)
      {
        //if (RCsignal.CH7 < 1000)
          Target.psi += ((RCsignal.YAW - 1075) / 250.0f);
        //else
          //Target.psi += 0.5;
      }
      if (RCsignal.YAW < 925)
      {
        //if (RCsignal.CH7 < 1000)
          Target.psi += ((RCsignal.YAW - 925) / 250.0f);
        //else
          //Target.psi -= 0.5;
      }
      if (Target.psi > 180.0f) Target.psi -= 360.0f;
      else if (Target.psi < -180.0f)Target.psi += 360.0f;
    }

    if (RCsignal.CH8 > 1000)
      Target.psi = Target.plan_psi;

    Target.psi_rad = Rad(Target.psi);
    /*--------------------P Control----------------------*/
    //phi
    Phicon.Input[0] = Target.phi_rad - state.phi_rad;
    Phicon.Output[0] = phiang.Kp * Phicon.Input[0];

    //theta
    Thetacon.Input[0] = Target.theta_rad - state.theta_rad;
    Thetacon.Output[0] = thetaang.Kp * Thetacon.Input[0];

    //psi
    if ((Target.psi_rad - state.psi_rad) >= M_PI ||
    (Target.psi_rad - state.psi_rad) < - M_PI)
    {
      if (Target.psi_rad > 0 && state.psi_rad < 0)
      Psicon.Input[0] = (-M_PI - state.psi_rad) +
      (Target.psi_rad - M_PI);
      if (Target.psi_rad < 0 && state.psi_rad > 0)
      Psicon.Input[0] = (M_PI - state.psi_rad) +
      (Target.psi_rad + M_PI);
    }
    else  Psicon.Input[0] = Target.psi_rad - state.psi_rad;

    Psicon.Output[0] = psiang.Kp * Psicon.Input[0];
  }
  AngularRateControl();
}
void _Copter::AngularRateControl()
{
  //p
  Target.p_rad = Phicon.Output[0];
  Target.p = Degree(Target.p_rad);

  //q
 Target.q_rad = Thetacon.Output[0];
  Target.q = Degree(Target.q_rad);
  //r
  /*------------------Nonlinear Constrain----------------*/
   Target.r_rad = Psicon.Output[0];
  Target.r = Degree(Target.r_rad);
  //p
```

```
  Pcon.Input[0] = Target.p_rad - state.p_rad;
  Pcon.Output[0] = Pcon.Input[0] * pvel.Kp +
  (Pcon.Input[0] - Pcon.Input[1]) * pvel.Kd / innerT;
  Pcon.Output[1] = Pcon.Output[0];
  Pcon.Input[1] = Pcon.Input[0];
  //q
  Qcon.Input[0] = Target.q_rad - state.q_rad;
  Qcon.Output[0] = Qcon.Input[0] * qvel.Kp +
  (Qcon.Input[0] - Qcon.Input[1]) * qvel.Kd / innerT;
  Qcon.Output[1] = Qcon.Output[0];
  Qcon.Input[1] = Qcon.Input[0];
  //r
  Rcon.Input[0] = Target.r_rad - state.r_rad;
  Rcon.Output[0] = Rcon.Input[0] * rvel.Kp +
  (Rcon.Input[0] - Rcon.Input[1]) * rvel.Kd / innerT;
  Rcon.Output[1] = Rcon.Output[0];
  Rcon.Input[1] = Rcon.Input[0];
  U2 = Pcon.Output[0];
  U3 = Qcon.Output[0];
  U4 = Rcon.Output[0];

 }
void _Copter::ControlReset()
{
  Pcon.Integral = 0;
  Qcon.Integral = 0;
  Rcon.Integral = 0;
}
```

Mark3 Flight Controller Code - Position Control

```
#include "Copter.h"
void _Copter::TranslationControl()
{
  if (Ctrl_timer1 >= 4)
  {
    if ((flag.mode == loiter_mode))
    {
      float tmp_roll, tmp_pitch;
      if (RCsignal.ROLL < 850)
        tmp_roll = ((float)(RCsignal.ROLL) - 850) / 100000;
      if (RCsignal.ROLL > 1150)
        tmp_roll = ((float)(RCsignal.ROLL) - 1150) / 100000;
      if (RCsignal.PITCH < 850)
        tmp_pitch = (850 - (float)(RCsignal.PITCH)) / 100000;
      if (RCsignal.PITCH > 1150)
        tmp_pitch = (1150 - (float)(RCsignal.PITCH)) / 100000;
      if (RCsignal.ROLL <= 1150 && RCsignal.ROLL >= 850)
        tmp_roll = 0;
      if (RCsignal.PITCH <= 1150 && RCsignal.PITCH >= 850)
        tmp_pitch = 0;
      Target.x += (tmp_pitch * state.psi_cos -
      tmp_roll * state.psi_sin);
      Target.y += (tmp_roll * state.psi_cos +
      tmp_pitch * state.psi_sin);
      Target.x = data_limitation(Target.x, -2.00, 2.00);
      Target.y = data_limitation(Target.y, -2.00, 2.00);

      if (RCsignal.CH8 > 1000)
      {
        Target.x = Target.plan_x;
        Target.y = Target.plan_y;
      }
      //X Control;
```

```
      Xcon.Input[0] = Target.x - state.x;
      Xcon.Output[0] = Xpos.Kp * Xcon.Input[0];
      Target.vx = Xcon.Output[0];
      //Y Control;
      Ycon.Input[0] = Target.y - state.y;
      Ycon.Output[0] = Ypos.Kp * Ycon.Input[0];
      Target.vy = Ycon.Output[0];
      //Vx Control
      Vxcon.Input[0] = (Target.vx - state.vx) / 57.3;
      Vxcon.Output[0] = Vxcon.Input[0] * Vxpos.Kp +
      (Vxcon.Input[0] - Vxcon.Input[1]) * Vxpos.Kd / outerT;
      Vxcon.Output[1] = Vxcon.Output[0];
      //Vy Control
      Vycon.Input[0] = (Target.vy - state.vy) / 57.3;
      Vycon.Output[0] = Vycon.Input[0] * Vypos.Kp +
      (Vycon.Input[0] - Vycon.Input[1]) * Vypos.Kd / outerT;
      Vycon.Output[1] = Vycon.Output[0];

      float tmp_a, tmp_b, tmp_c1, tmp_c2, tmp_x1, tmp_x2;
      tmp_a = state.psi_cos;
      tmp_b = state.psi_sin;
      tmp_c1 = Vxcon.Output[0] * 0.647 * (U1 * 0.9);
      tmp_c2 = Vycon.Output[0] * 0.647 * (U1 * 0.9);
      //Target Phi
      Target.sin_phi = tmp_a * tmp_c2 - tmp_b * tmp_c1;
      Target.sin_phi = data_limitation(Target.sin_phi, -1, 1);
      Target.phi_rad = asin(Target.sin_phi);

      Target.phi = Degree(Target.phi_rad);
      Target.phi = data_limitation(Target.phi, -15, 15);
      Target.phi_rad = Rad(Target.phi);
      //Target Theta
      Target.cos_phi = cos(Target.phi_rad);
      Target.sin_theta = (-(tmp_a * tmp_c1 + tmp_b * tmp_c2))
      / Target.cos_phi;
      Target.sin_theta = data_limitation(Target.sin_theta, -1, 1);
      Target.theta_rad = asin(Target.sin_theta);

      Target.theta = Degree(Target.theta_rad);
      Target.theta = data_limitation(Target.theta, -15, 15);
      Target.theta_rad = Rad(Target.theta);
    }
  }
}
void _Copter::AltitudeControl()
{
  if (Ctrl_timer1 >= 4)
  {
    if ((flag.mode == Altitude_mode) || (flag.mode == loiter_mode))
    {
      if (RCsignal.THROTTLE < 850)
      {
        if (RCsignal.CH7 < 1000)
          Target.z += ((float)(RCsignal.THROTTLE) - 850) / 100000;
        else
          Target.z -= 0.003;
      }
      if (RCsignal.THROTTLE > 1150)
      {
        if (RCsignal.CH7 < 1000)
          Target.z += ((float)(RCsignal.THROTTLE) - 1150) / 100000;
        else
          Target.z += 0.003;
      }
      Target.z = data_limitation(Target.z, 0.20, 2.00);
```

```
    }
    if (RCsignal.CH8 > 1000)
      Target.z = Target.plan_z;
    ZControl();
    VzControl();
  }
  float StickThrust = 0.008193 * RCsignal.THROTTLE - 2.458;
  if (flag.mode == Attitude_mode)
  {
    U1 = StickThrust / (state.theta_cos * state.phi_cos);
    Target.z = 0;
    state.flight_state[0] = 1; //stand by
    state.flight_state[1] = 0; //take off
    state.flight_state[2] = 0; //flight
    state.flight_state[3] = 0; //land
    state.takeoff_t = 0;
    state.landing_t = 0;
  }
  else if ((flag.mode == Altitude_mode) ||
  (flag.mode == loiter_mode))
  {
    uint8_t i, fliorder;
    float tmp;
    for (i = 0; i < 4; i++)
      if (state.flight_state[i])
        fliorder = (i + 1) * 10;
    //Serial1.print(fliorder);Serial1.print('\t');
    //Serial1.print(state.takeoff_t);Serial1.print('\t');
    //Serial1.println();
    switch (fliorder)
    {
      case 10: //--------------stand by
        U1 = 0;
        Target.x = state.x;
        Target.y = state.y;
        if (RCsignal.THROTTLE > 850 && RCsignal.THROTTLE
        < 1150 && voltageavg >= 10.5)
        {
          state.takeoff_t = state.takeoff_t + 2;
          if (state.takeoff_t > 1000)
          {
            state.takeoff_t = 1000;
            state.flight_state[0] = 0; //stand by
            state.flight_state[1] = 1; //take off
            state.flight_state[2] = 0; //flight
            state.flight_state[3] = 0; //land
          }
        }
        else if (RCsignal.THROTTLE < 350)
          state.takeoff_t = 0;
        break;
      case 20: //----------------------------------take off
        if (state.takeoff_t > 0)
        {
          state.takeoff_t = state.takeoff_t - 2;
          if (state.takeoff_t < 0)
            state.takeoff_t = 0;
        }
        else
        {
          state.flight_state[0] = 0; //stand by
          state.flight_state[1] = 0; //take off
          state.flight_state[2] = 1; //flight
          state.flight_state[3] = 0; //land
          Target.z = 0.35;
        }
```

```
        U1 = float(1000 - state.takeoff_t) / 1000 * 6.6;
        if (RCsignal.THROTTLE < 350)
        {
          state.flight_state[0] = 1; //stand by
          state.flight_state[1] = 0; //take off
          state.flight_state[2] = 0; //flight
          state.flight_state[3] = 0; //land
          U1 = 0;
        }
        break;
      case 30: //-----------------------------------flight
        U1 = (EstimatedG + Vzcon.Output[0]) /
        (state.phi_cos * state.theta_cos);
        if (voltageavg <= 10.5)
        {
          Target.z = 0.20;
          if (state.z <= 0.25)
          {
            state.landing_t = 2000;
            state.flight_state[0] = 0; //stand by
            state.flight_state[1] = 0; //take off
            state.flight_state[2] = 0; //flight
            state.flight_state[3] = 1; //land
            state.tmp_U1 = U1;
          }
        }
        if (RCsignal.THROTTLE < 350)
        {
          state.landing_t = state.landing_t + 3;
          if (state.landing_t > 2000)
            state.landing_t = 2000;
          if (state.landing_t == 2000 && state.z <= 0.25)
          {
            state.flight_state[0] = 0; //stand by
            state.flight_state[1] = 0; //take off
            state.flight_state[2] = 0; //flight
            state.flight_state[3] = 1; //land
            state.tmp_U1 = U1;
          }
        }
        else if (RCsignal.THROTTLE > 850)
          state.landing_t = 0;
        break;
      case 40: //---------------------------------landing
        state.landing_t--;
        tmp = float(state.landing_t) / 2000;
        U1 = tmp * tmp * state.tmp_U1;
        if (state.landing_t == 0)
        {
          state.flight_state[0] = 1; //stand by
          state.flight_state[1] = 0; //take off
          state.flight_state[2] = 0; //flight
          state.flight_state[3] = 0; //land
          Target.z = 0;
        }
        break;
    }
  }
}
void _Copter::ZControl()
{
  Zcon.Input[0] = Target.z - state.z;
  Zcon.Output[0] = Zalt.Kp * Zcon.Input[0];
  Target.vz = Zcon.Output[0];

}
```

```cpp
void _Copter::VzControl()
{
  if (state.z < 0.35 || voltageavg < 10.5)
    Target.vz = data_limitation(Target.vz, -0.2, 0.2);
  Vzcon.Input[0] = Target.vz - state.vz;
  Vzcon.Output[0] = Vzcon.Input[0] * Vzalt.Kp +
  (Vzcon.Input[0] - Vzcon.Input[1]) * Vzalt.Kd / outerT;
  Vzcon.Output[0] = data_limitation(Vzcon.Output[0], -4.0, 4.0);
  Vzcon.Input[1] = Vzcon.Input[0];
}
```

System.cpp **Mark3 Flight Controller Code - Sensor Read**

```cpp
#include "Copter.h"
void _Copter::InitSensor()
{
  Wire.begin();
  Wire.setRate(I2C_RATE_2000);
  Wire1.begin();
  Wire1.setRate(I2C_RATE_2000);

  //gy86
  I2Cwrite(MPU6050_ADDRESS,
           MPUREG_PWR_MGMT_1, MPU_CLK_SEL_PLLGYROZ, 0);
  I2Cwrite(MPU6050_ADDRESS,
           MPUREG_SMPLRT_DIV, 0x07, 0);
  I2Cwrite(MPU6050_ADDRESS,
           MPUREG_CONFIG, BITS_DLPF_CFG_42HZ, 0);
  I2Cwrite(MPU6050_ADDRESS,
           MPUREG_GYRO_CONFIG, BITS_FS_1000DPS, 0);
  I2Cwrite(MPU6050_ADDRESS,
           MPUREG_ACCEL_CONFIG, 0x08, 0);
  I2Cwrite(MPU6050_ADDRESS,
           MPUREG_INT_PIN_CFG, 0x02, 0);
  I2CRead(MPU6050_ADDRESS,
          MPUREG_WHOAMI, 1);
  if (i2cData[0] != 0x68)
    Serial1.println("Error reading sensor");
  AccPointRead();
}
void _Copter::AccPointRead()
{
  uint8_t point;
  for (point = 0; point < 6; point++)
  {
    Acc_Cali.accel_raw_ref[point][0] =
      (EEPROM.read(100 + 6 * point + 2) << 8)
      | EEPROM.read(100 + 6 * point + 1);
    Acc_Cali.accel_raw_ref[point][1] =
      (EEPROM.read(100 + 6 * point + 4) << 8)
      | EEPROM.read(100 + 6 * point + 3);
    Acc_Cali.accel_raw_ref[point][2] =
      (EEPROM.read(100 + 6 * point + 6) << 8)
      | EEPROM.read(100 + 6 * point + 5);
  }

  Acc_Cali.acc_offset[0] =
    (float)(Acc_Cali.accel_raw_ref[0][0] +
            Acc_Cali.accel_raw_ref[1][0]) / 2.0;
  Acc_Cali.acc_offset[1] =
    (float)(Acc_Cali.accel_raw_ref[2][1] +
            Acc_Cali.accel_raw_ref[3][1]) / 2.0;
  Acc_Cali.acc_offset[2] =
```

```
    (float)(Acc_Cali.accel_raw_ref[4][2] +
            Acc_Cali.accel_raw_ref[5][2]) / 2.0;

  for (point = 0; point < 3; point++)
    Acc_Cali.a[0][point] =
      (float)Acc_Cali.accel_raw_ref[0][point] -
      Acc_Cali.acc_offset[point];
  for (point = 0; point < 3; point++)
    Acc_Cali.a[1][point] =
      (float)Acc_Cali.accel_raw_ref[2][point] -
      Acc_Cali.acc_offset[point];
  for (point = 0; point < 3; point++)
    Acc_Cali.a[2][point] =
      (float)Acc_Cali.accel_raw_ref[4][point] -
      Acc_Cali.acc_offset[point];
}
void _Copter::MPU6050read()
{
  I2CRead(MPU6050_ADDRESS, MPUREG_ACCEL_XOUT_H, 14);
  gy86.acc.origin.x = ((i2cData[0] << 8) | i2cData[1]);
  gy86.acc.origin.y = ((i2cData[2] << 8) | i2cData[3]);
  gy86.acc.origin.z = ((i2cData[4] << 8) | i2cData[5]);
  mpu_temperature = (i2cData[6] << 8) | i2cData[7];
  gy86.gyro.origin.x = (i2cData[8] << 8) | i2cData[9];
  gy86.gyro.origin.y = ((i2cData[10] << 8) | i2cData[11]);
  gy86.gyro.origin.z = ((i2cData[12] << 8) | i2cData[13]);
  MPU6050ThermalCompensation();
  MPU6050Sixpoint();
}
void _Copter::MPU6050Sixpoint()
{
  gy86.acc.quietf.x = (float)gy86.acc.origin.x -
                      Acc_Cali.acc_offset[0];
  gy86.acc.quietf.y = (float)gy86.acc.origin.y -
                      Acc_Cali.acc_offset[1];
  gy86.acc.quietf.z = (float)gy86.acc.origin.z -
                      Acc_Cali.acc_offset[2];

  gy86.acc.aftcal.x = gy86.acc.quietf.x *  Acc_Cali.T[0][0] +
                      gy86.acc.quietf.y *  Acc_Cali.T[1][0] +
                      gy86.acc.quietf.z *
                      Acc_Cali.T[2][0];
  gy86.acc.aftcal.y = gy86.acc.quietf.x *  Acc_Cali.T[0][1] +
                      gy86.acc.quietf.y *  Acc_Cali.T[1][1] +
gy86.acc.quietf.z *
                      Acc_Cali.T[2][1];
  gy86.acc.aftcal.z = gy86.acc.quietf.x *  Acc_Cali.T[0][2] +
                      gy86.acc.quietf.y *  Acc_Cali.T[1][2] +
                      gy86.acc.quietf.z *
                      Acc_Cali.T[2][2];
}
void _Copter::MPU6050ThermalCompensation()
{
  float tp3, tp2, tp;//, accx, accy, accz;

  temperature = (float) mpu_temperature / 340.00 + 36.53;
  tp = temperature;
  if (tp > 55.0) tp = 55.0;
  if (tp < 22.5) tp = 22.5;
  tp2 = tp * tp;
  tp3 = tp2 * tp;

  /*-------------------Thermal Calibration------------------*/
  gy86.gyro.tempcp.x = 0.000263 * tp3 -
                       0.03098 * tp2 + 0.03939 * tp - 29.4;
```

```
  gy86.gyro.tempcp.y = -0.0004279 * tp3 +
                       0.05322 * tp2 - 2.941 * tp + 80.16;
  gy86.gyro.tempcp.z = 0.0004163 * tp3 -
                       0.0332 * tp2 + 0.6652 * tp + 23.3;

  gy86.gyro.aftcal.x = (float)gy86.gyro.origin.x -
                       gy86.gyro.tempcp.x;
  gy86.gyro.aftcal.y = (float)gy86.gyro.origin.y -
                       gy86.gyro.tempcp.y;
  gy86.gyro.aftcal.z = (float)gy86.gyro.origin.z -
                       gy86.gyro.tempcp.z;

  if (flag.calibratedG == 0 && GyroCaliFlag <= 1500)
  {
    GyroCaliFlag++;
    MPU6050GyroCali();
  }
  if (flag.calibratedG == 1)
  {
    gy86.gyro.aftcal.x = gy86.gyro.aftcal.x -
                         gy86.gyro.radian.x;
    gy86.gyro.aftcal.y = gy86.gyro.aftcal.y -
                         gy86.gyro.radian.y;
    gy86.gyro.aftcal.z = gy86.gyro.aftcal.z -
                         gy86.gyro.radian.z;
  }
}
void _Copter::MPU6050AccCali(uint8_t point)
{
  /*Point = 0 1 2 3 4 5*/
  if (Acc_Cali.acc_calitimer == 100)
  {
    Acc_Cali.accel_raw_ref[point][0] =
      Acc_Cali.acc_calitmpx / 100;
    Acc_Cali.accel_raw_ref[point][1] =
      Acc_Cali.acc_calitmpy / 100;
    Acc_Cali.accel_raw_ref[point][2] =
      Acc_Cali.acc_calitmpz / 100;
    Acc_Cali.acc_calitmpx = 0;
    Acc_Cali.acc_calitmpy = 0;
    Acc_Cali.acc_calitmpz = 0;
    Acc_Cali.acc_calitimer = 0;
    comorder = 0;
    Serial1.print(Acc_Cali.accel_raw_ref[point][0]);
    Serial1.print('\t');
    Serial1.print(Acc_Cali.accel_raw_ref[point][1]);
    Serial1.print('\t');
    Serial1.print(Acc_Cali.accel_raw_ref[point][2]);
    Serial1.print('\t');
    Serial1.println();
    EEPROM.write(100 + 6 * point + 1,
                 Acc_Cali.accel_raw_ref[point][0] & 0b11111111);
    EEPROM.write(100 + 6 * point + 2,
                 Acc_Cali.accel_raw_ref[point][0] >> 8);
    EEPROM.write(100 + 6 * point + 3,
                 Acc_Cali.accel_raw_ref[point][1] & 0b11111111);
    EEPROM.write(100 + 6 * point + 4,
                 Acc_Cali.accel_raw_ref[point][1] >> 8);
    EEPROM.write(100 + 6 * point + 5,
                 Acc_Cali.accel_raw_ref[point][2] & 0b11111111);
    EEPROM.write(100 + 6 * point + 6,
                 Acc_Cali.accel_raw_ref[point][2] >> 8);
  }
  else
  {
    Acc_Cali.acc_calitmpx += gy86.acc.origin.x;
    Acc_Cali.acc_calitmpy += gy86.acc.origin.y;
```

```
    Acc_Cali.acc_calitmpz += gy86.acc.origin.z;
    Acc_Cali.acc_calitimer++;
  }
}
void _Copter::MPU6050GyroCali()
{
  if (GyroCaliFlag > 400 && GyroCaliFlag < 1001)
  {
    GyroCollection[0] += gy86.gyro.aftcal.x;
    GyroCollection[1] += gy86.gyro.aftcal.y;
    GyroCollection[2] += gy86.gyro.aftcal.z;
  }
  if (GyroCaliFlag == 1001)
  {
    gy86.gyro.radian.x = GyroCollection[0] / 600;
    gy86.gyro.radian.y = GyroCollection[1] / 600;
    gy86.gyro.radian.z = GyroCollection[2] / 600;
    Serial1.print(GyroCollection[0]); Serial1.print("\t");
    Serial1.print(GyroCollection[1]); Serial1.print("\t");
    Serial1.print(GyroCollection[2]); Serial1.print("\t");
    GyroCaliFlag = 0;
    GyroCollection[0] = 0;
    GyroCollection[1] = 0;
    GyroCollection[2] = 0;
    flag.calibratedG = 1;
    Serial1.println("Gyro Offset Calculated");
    Serial1.print(gy86.gyro.radian.x); Serial1.print("\t");
    Serial1.print(gy86.gyro.radian.y); Serial1.print("\t");
    Serial1.print(gy86.gyro.radian.z); Serial1.print("\t");
  }
}
/*------------------I2C------------------*/
uint8_t _Copter::I2Cwrite(uint8_t SENSOR_ADDRESS,
                          uint8_t SENSOR_REGISTER,
                          uint8_t SENSOR_VALUE,
                          bool sendStop)
{
  Wire.beginTransmission(SENSOR_ADDRESS);
  Wire.write(SENSOR_REGISTER);
  Wire.write(SENSOR_VALUE);//DEVICE_RESET
  //Wire.endTransmission();
  uint8_t rcode = Wire.endTransmission(sendStop);
  if (rcode) {
    Serial1.print(F("i2cWrite failed: "));
    Serial1.println(rcode);
  }
  return rcode;
}
uint8_t _Copter::I2CRead(uint8_t SENSOR_ADDRESS, uint8_t
                        SENSOR_REGISTER, uint8_t nbytes)
{
  uint32_t timeOutTimer;
  Wire.beginTransmission(SENSOR_ADDRESS);
  Wire.write(SENSOR_REGISTER);
  uint8_t rcode = Wire.endTransmission(false);
  if (rcode) {
    Serial1.print(F("i2cRead failed: "));
    Serial1.println(rcode);
    return rcode;
    // See: http://arduino.cc/en/Reference/WireEndTransmission
  }
  Wire.requestFrom(SENSOR_ADDRESS, nbytes, (uint8_t)true);
  for (uint8_t i = 0; i < nbytes; i++) {
    if (Wire.available())
      i2cData[i] = Wire.read();
```

```
    else {
      timeOutTimer = micros();
      while (((micros() - timeOutTimer) <
              I2C_TIMEOUT) && !Wire.available());
      if (Wire.available())
        i2cData[i] = Wire.read();
      else {
        Serial1.println(F("i2cRead timeout"));
        return 5;
      }
    }
  }
  return 0; // Success
}
```

Mark3 Flight Controller Code - Sensor Address Library

```
//GY86
#define MPU6050_ADDRESS            0x68
#define MPUREG_WHOAMI              0x75
#define MPUREG_SMPLRT_DIV          0x19
#define MPUREG_CONFIG              0x1A
#define MPUREG_GYRO_CONFIG         0x1B
#define MPUREG_ACCEL_CONFIG        0x1C
#define MPUREG_FIFO_EN             0x23
#define MPUREG_INT_PIN_CFG         0x37
#define MPUREG_INT_ENABLE          0x38
#define MPUREG_INT_STATUS          0x3A
#define MPUREG_ACCEL_XOUT_H        0x3B
#define MPUREG_ACCEL_XOUT_L        0x3C
#define MPUREG_ACCEL_YOUT_H        0x3D
#define MPUREG_ACCEL_YOUT_L        0x3E
#define MPUREG_ACCEL_ZOUT_H        0x3F
#define MPUREG_ACCEL_ZOUT_L        0x40
#define MPUREG_TEMP_OUT_H          0x41
#define MPUREG_TEMP_OUT_L          0x42
#define MPUREG_GYRO_XOUT_H         0x43
#define MPUREG_GYRO_XOUT_L         0x44
#define MPUREG_GYRO_YOUT_H         0x45
#define MPUREG_GYRO_YOUT_L         0x46
#define MPUREG_GYRO_ZOUT_H         0x47
#define MPUREG_GYRO_ZOUT_L         0x48
#define MPUREG_USER_CTRL           0x6A
#define MPUREG_PWR_MGMT_1          0x6B
#define MPUREG_PWR_MGMT_2          0x6C
#define MPUREG_FIFO_COUNTH         0x72
#define MPUREG_FIFO_COUNTL         0x73
#define MPUREG_FIFO_R_W            0x74
// Configuration bits
#define BIT_SLEEP                  0x40
#define BIT_H_RESET                0x80
#define BITS_CLKSEL                0x07
#define MPU_CLK_SEL_PLLGYROX       0x01
#define MPU_CLK_SEL_PLLGYROZ       0x03
#define MPU_EXT_SYNC_GYROX         0x02
#define BITS_FS_250DPS             0x00
#define BITS_FS_500DPS             0x08
#define BITS_FS_1000DPS            0x10
#define BITS_FS_2000DPS            0x18
#define BITS_FS_MASK               0x18
#define BITS_DLPF_CFG_256HZ    0x00
//Default settings LPF 256Hz/8000Hz sample
#define BITS_DLPF_CFG_188HZ             0x01
#define BITS_DLPF_CFG_98HZ              0x02
#define BITS_DLPF_CFG_42HZ              0x03
#define BITS_DLPF_CFG_20HZ              0x04
#define BITS_DLPF_CFG_10HZ              0x05
#define BITS_DLPF_CFG_5HZ               0x06
```

```
#define BITS_DLPF_CFG_2100HZ_NOLPF  0x07
#define BITS_DLPF_CFG_MASK          0x07
#define BIT_INT_ANYRD_2CLEAR    0x10
#define BIT_RAW_RDY_EN          0x01
#define BIT_I2C_IF_DIS          0x10
#define BIT_INT_STATUS_DATA     0x01

//GY89
#define L3GD20_ADDRESS    (0xD6 >> 1)
#define L3G_WHO_AM_I      0x0F

#define L3G_CTRL_REG1     0x20
#define L3G_CTRL_REG2     0x21
#define L3G_CTRL_REG3     0x22
#define L3G_CTRL_REG4     0x23
#define L3G_CTRL_REG5     0x24
#define L3G_REFERENCE     0x25
#define L3G_OUT_TEMP      0x26
#define L3G_STATUS_REG    0x27

#define L3G_OUT_X_L       0x28
#define L3G_OUT_X_H       0x29
#define L3G_OUT_Y_L       0x2A
#define L3G_OUT_Y_H       0x2B
#define L3G_OUT_Z_L       0x2C
#define L3G_OUT_Z_H       0x2D

#define L3G_FIFO_CTRL_REG 0x2E
#define L3G_FIFO_SRC_REG  0x2F

#define L3G_INT1_CFG      0x30
#define L3G_INT1_SRC      0x31
#define L3G_INT1_THS_XH   0x32
#define L3G_INT1_THS_XL   0x33
#define L3G_INT1_THS_YH   0x34
#define L3G_INT1_THS_YL   0x35
#define L3G_INT1_THS_ZH   0x36
#define L3G_INT1_THS_ZL   0x37
#define L3G_INT1_DURATION 0x38

#define LSM303D_ADDRESS   0b0011101
#define LSM303D_CTRL_REG0      0x1F
#define LSM303D_CTRL_REG1      0x20
#define LSM303D_CTRL_REG2      0x21
#define LSM303D_CTRL_REG3      0x22
#define LSM303D_CTRL_REG4      0x23
#define LSM303D_CTRL_REG5      0x24
#define LSM303D_CTRL_REG6      0x25
#define LSM303D_CTRL_REG7      0x26
#define LSM303D_OUT_X_L_A      0x28

#define L3GD20_DEFAULT_FILTER_FREQ    30
#define L3GD20_DEFAULT_RATE 400

#define LSM303D_ACCEL_DEFAULT_RANGE_G          16
#define LSM303D_ACCEL_DEFAULT_RATE          400
#define LSM303D_ACCEL_DEFAULT_ONCHIP_FILTER_FREQ  50
#define LSM303D_ACCEL_DEFAULT_DRIVER_FILTER_FREQ  30
```

Mark3 Flight Controller Code - System Check

```
#include "Copter.h"
void _Copter::InitControl()
{
  pvel.Kp = (EEPROM.read(11) << 8) | EEPROM.read(10); pvel.Kp /= 10000;
  pvel.Ki = (EEPROM.read(13) << 8) | EEPROM.read(12); pvel.Ki /= 1000;
  pvel.Kd = (EEPROM.read(15) << 8) | EEPROM.read(14); pvel.Kd /= 100000;
  phiang.Kp = (EEPROM.read(17) << 8) | EEPROM.read(16); phiang.Kp /= 100;

```

```
  qvel.Kp = (EEPROM.read(19) << 8) |
  EEPROM.read(18); qvel.Kp /= 10000;
  qvel.Ki = (EEPROM.read(21) << 8) |
  EEPROM.read(20); qvel.Ki /= 1000;
  qvel.Kd = (EEPROM.read(23) << 8) |
  EEPROM.read(22); qvel.Kd /= 100000;
  thetaang.Kp = (EEPROM.read(25) << 8) |
  EEPROM.read(24); thetaang.Kp /= 100;

  rvel.Kp = (EEPROM.read(27) << 8) |
  EEPROM.read(26); rvel.Kp /= 100000;
  rvel.Ki = (EEPROM.read(29) << 8) |
  EEPROM.read(28); rvel.Ki /= 10000;
  rvel.Kd = (EEPROM.read(31) << 8) |
  EEPROM.read(30); rvel.Kd /= 1000000;
  psiang.Kp = (EEPROM.read(33) << 8) |
  EEPROM.read(32); psiang.Kp /= 100;

  Zalt.Kp = (EEPROM.read(41) << 8) |
  EEPROM.read(40); Zalt.Kp /= 100;
  Vzalt.Kp = (EEPROM.read(43) << 8) |
  EEPROM.read(42); Vzalt.Kp /= 100;
  Vzalt.Ki = (EEPROM.read(45) << 8) |
  EEPROM.read(44); Vzalt.Ki /= 100;
  Vzalt.Kd = (EEPROM.read(47) << 8) |
  EEPROM.read(46); Vzalt.Kd /= 100;

  Xpos.Kp = (EEPROM.read(51) << 8) |
  EEPROM.read(50); Xpos.Kp /= 1000;
  Vxpos.Kp = (EEPROM.read(53) << 8) |
  EEPROM.read(52); Vxpos.Kp /= 1000;
  Vxpos.Ki = (EEPROM.read(55) << 8) |
  EEPROM.read(54); Vxpos.Ki /= 1000;
  Vxpos.Kd = (EEPROM.read(57) << 8) |
  EEPROM.read(56); Vxpos.Kd /= 1000;

  Ypos.Kp = (EEPROM.read(61) << 8) |
  EEPROM.read(60); Ypos.Kp /= 1000;
  Vypos.Kp = (EEPROM.read(63) << 8) |
  EEPROM.read(62); Vypos.Kp /= 1000;
  Vypos.Ki = (EEPROM.read(65) << 8) |
  EEPROM.read(64); Vypos.Ki /= 1000;
  Vypos.Kd = (EEPROM.read(67) << 8) |
  EEPROM.read(66); Vypos.Kd /= 1000;

  EstimatedG = (EEPROM.read(71) << 8) |
  EEPROM.read(70); EstimatedG /= 1000;

  state.flight_state[0] = 1; //stand by
  state.flight_state[1] = 0; //take off
  state.flight_state[2] = 0; //flight
  state.flight_state[3] = 0; //land
  state.takeoff_t = 0;
  state.landing_t = 0;
}
void _Copter::Loop_Check()
{
  xbee_length = 0;
  if (Ctrl_timer1 >= 4)
    Ctrl_timer1 = 0; //----------------Check 100Hz
  Moment_Check();
  Battery_Check();
  Timer_Check();
}
void _Copter::Otus_Clear()
```

```
{
  state.update_phi = 0;
  state.update_theta = 0;
  state.update_psi = 0;
  state.update_vx = 0;
  state.update_vy = 0;
  state.update_vz = 0;
  state.update_x = 0;
  state.update_y = 0;
  state.update_z = 0;
}
void _Copter::Moment_Check()
{
  float temp_gyro;
  if (comorder >= 1401 && comorder <= 1403)
  {
    if (inertia.timer_start == 0)
    {
      inertia.timer_start = 1;
      inertia.time_start = micros();
    }
    inertia.time_end = micros();
    inertia.time_count = inertia.time_end - inertia.time_start;

    if (gy86.gyro.filter.z > 400 && inertia.memo < -400)
    {
      inertia.memo = gy86.gyro.filter.z;
      inertia.pendulum ++;
    }
    if (gy86.gyro.filter.z < - 400 && inertia.memo > 400)
    {
      inertia.memo = gy86.gyro.filter.z;
    }
    if (inertia.xbee_timer == 20)
    {
      Serial1.print(inertia.time_count); Serial1.print("  ");
      Serial1.print(gy86.gyro.filter.z); Serial1.print("  ");
      Serial1.print(inertia.memo); Serial1.print("  ");
      Serial1.println(inertia.pendulum);
      inertia.xbee_timer = 0;
    }
    inertia.xbee_timer++;
  }
  else
  {
    inertia.timer_start = 0;
    inertia.time_count = 0;
    inertia.time_end = 0;
    inertia.time_start = 0;
    inertia.pendulum = 0;
    inertia.memo = -1000;
  }
}
void _Copter::Battery_Check()
{
  //voltage = (float)analogRead(A14) * 0.019586; //only for quad 2
  voltage = (float)analogRead(A14) * 0.013841;
  voltageavg = voltage * 0.005 + voltageavg * 0.995;
  voltageavg = data_limitation(voltageavg, 9.0, 17.0);
  if (voltageavg < 10.5)
    battery_warning = 1;
  else
    battery_warning = 0;
}
void _Copter::Timer_Check()
{
  if (gltimer == 150)
```

```
  {
    if (glch > 0)
    {
      digitalWrite(LED2, LOW);
      if (voltageavg < 10.5)
        digitalWrite(LED1, HIGH);
      else
        digitalWrite(LED1, LOW);
    }
    else
    {
      digitalWrite(LED2, HIGH);
    }
    glch *= -1;
    gltimer = 0;
  }
  else
    gltimer++;
  run_period = micros() - whole_timer;
  if (comorder == 2002)
    Serial1.println(run_period);
  if (run_period > MainLoopPeriod)
  {
    time_out = 1;
    Serial1.print(run_period);
    Serial1.println("   <<<<<time out>>>>>");
  }
  loopClock2 = micros() - loopClock1;
  while (micros() - whole_timer < MainLoopPeriod);
  whole_timer = micros();
}

float _Copter::Rad(float angle)
{
  return (angle * M_PI / 180.0);
}
float _Copter::Degree(float rad)
{
  return (rad / M_PI * 180.0);
}
float _Copter::data_limitation(float a, float b, float c)
{
  if (a < b) a = b;
  if (a > c) a = c;
  return a;
}

float _Copter::invSqrt(float number) {
  long i;
  float x2, y;
  const float threehalfs = 1.5F;

  x2 = number * 0.5F;
  y  = number;
  i  = * ( long * ) &y;
  i  = 0x5f3759df - ( i >> 1 );
  y  = * ( float * ) &i;
  y  = y * ( threehalfs - ( x2 * y * y ) );
  return y;
}
```

APPENDIX D

HARDWARE ASSEMBLY INSTRUCTIONS & SOFTWARE INITIALIZATION

D.1 Hardware Assembly Instructions

The instructions of assembling a 250mm quadrotor platform are provided. Items needed for assembling the *MARK3* flight controller are listed in Table D.1.

Item	**Quantity**
Teensy 3.2 MCU	1
GY-89 Sensor Board	1
Xbee 3.0	1
LED-Blue	1
LED-Yellow	1
1N4001	1
3p 2.54mm Connector	2
2p 2.54mm Connector	4
5p 2.54mm Connector	3
6p Triple Row 2.54mm Connector	1
8p Triple Row 2.54mm Connector	1
8p Double Patch 2.54mm Connector	1
10p 2.00mm Connector	2
Resistor 2k	1
Resistor 10k	1
Resistor 300Ω	2
I2C Logic Converter 3.3v 5v	1
5V to 3.3V Step Down Voltage Regulator	1

Table D.1: Items Needed for Assembling the *MARK3* flight controller

Items needed for assembling the 250mm quadrotor platform are listed in Table D.2.

Item	**Quantity**
MARK3 flight controller	1
DJI Snail Propulsion System	1
DJI 5045 Propeller	4
250mm ATG Carbon-Fiber Frame	1
11.1v 3s Lipo Battery 20c	1
Radiolink R9DS Receiver	1
HTC VIVE Tracker	1
Matek V3 UBEC	1

Table D.2: Items Needed for Assembling the 250mm quadrotor platform

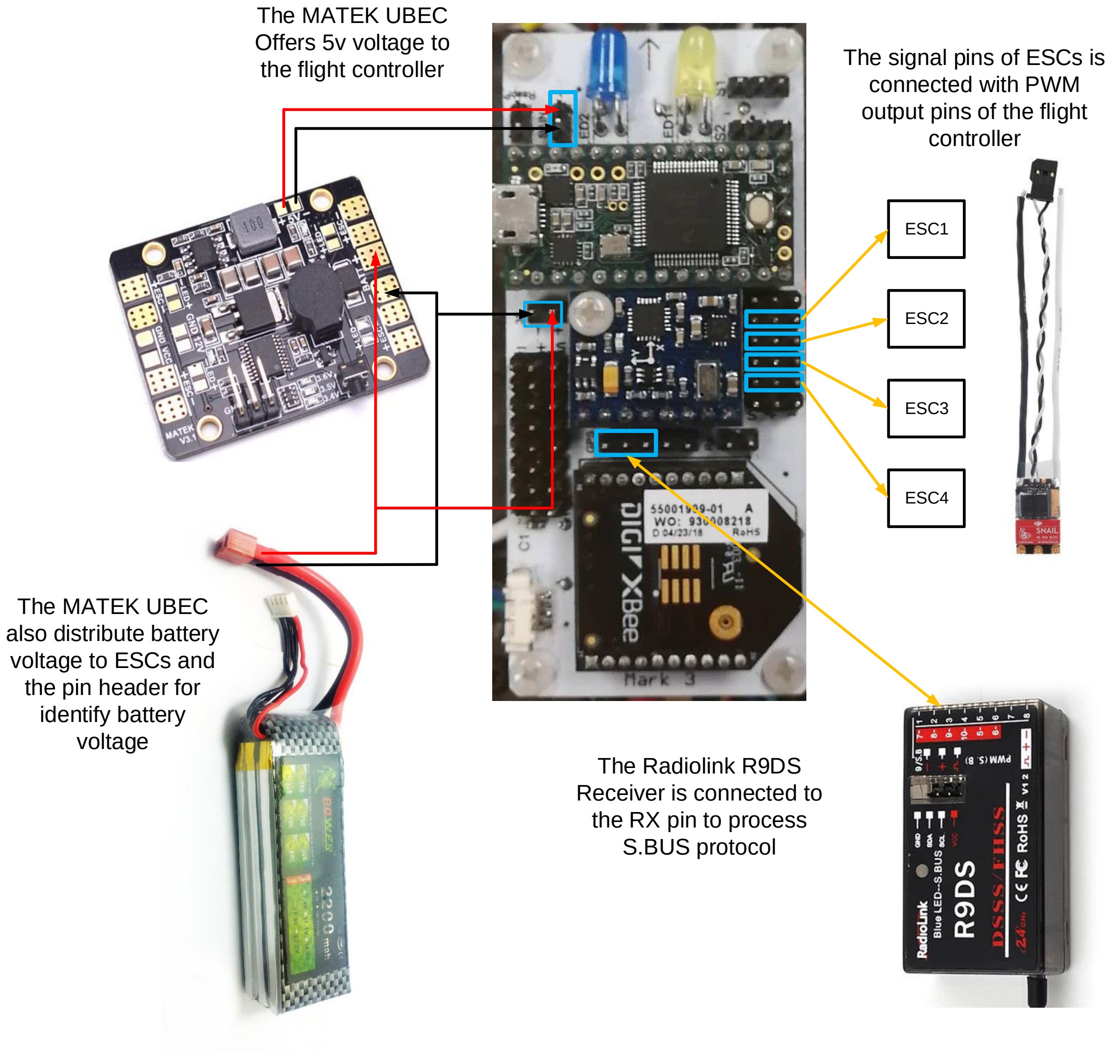


Figure D.1: *MARK3* Flight Controller Wiring Instruction

The flight controller wiring instruction is shown in Figure D.1 with the following steps:

1. *Power Distribution Board & Battery Connection*
   Solder the MATEK V3 UBEC board with all four ESCs and battery XT60 male connector. Two sets of 2.54mm wires need to be soldered for connection of power supply and battery voltage read of the *MARK3* flight controller.
2. *ESC PWM Signal Wire Connection*
   All four ESCs should be powered by the 3s Lipo battery. All four pwm signal wires should be connected with PWM output pin header (the signal pin is on the Left and the ground pin is on the right) on the flight controller with certain sequence.

3. *Receiver Signal Pin Connection*
   The *MARK3* flight controller supports receivers based on S.BUS protocol (the working voltage is $3.3v$). The receiver should be connected with the flight controller RX pin shown in the diagram.

D.2 Software Initialization

1. *Accelerometer 6-point Calibration*
   Based on Chapter 6, the accelerometer requires 6-point Calibration. The following commands is required step by step:

   ```
   1041
   ```

   To perform +z calibration

   ```
   1042
   ```

   To perform -z calibration

   ```
   1043
   ```

   To perform +x calibration

   ```
   1044
   ```

   To perform -x calibration

   ```
   1045
   ```

   To perform +y calibration

   ```
   1046
   ```

   To perform -y calibration
2. *Control Parameters Initialization*
   The control parameters are stored in EEPROM. For the first time use, all control parameters should be typed via serial monitor on arduino:

   ```
   &1011_0950*   //P Kp
   &1012_0000*  //P Ki
   &1013_0405* //P Kd

   &1021_0976* //Q Kp
   &1022_0000* //Q Ki
   &1023_0414* //Q Kd

   &1031_1550* //R Kp
   &1032_000* //R Ki
   &1033_0900* //R Kd

   &1051_1000* //Phi Kp
   &1052_1000* //Theta Kp
   ```

```
&1053_420* //Psi Kp

&1071_270* //Z Kp
&1081_420* //Vz Kp
&1083_036* //Vz Kd

&1091_1250* //X Kp
&1092_0920* //Vx Kp
&1094_0038* //Vx Kd
&1095_1250* //Y Kp
&1096_0920* //Vy Kp
&1098_0038* //Vy Kd
&1100_7550* //G 7000
```

3. *Set up Connection for MATLAB based GUI*
   Both SteamVR client and the MATLAB GUI should be turned on (any other high computing cost process should be killed)

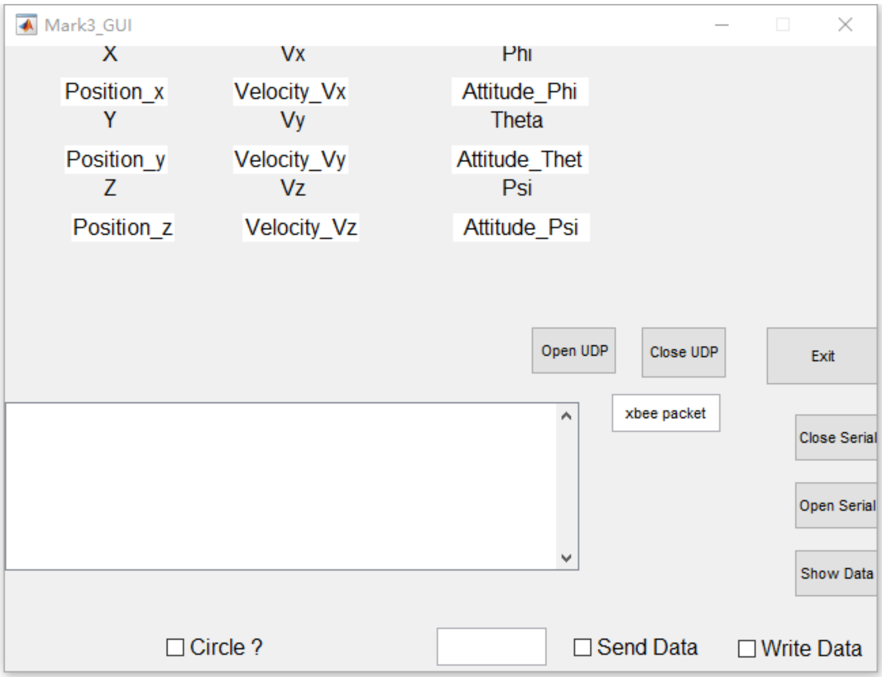


Figure D.2: *MARK3* Ground Station

The interface of MATLAB based GUI is shown in Figure D.2. Turn on the switch for UDP protocol. If the dynamic state value is shown, then turn on the switch for serial protocol. Fill the tick of sending data, then the quadrotor will receive the flight state value from HTC VIVE Tracking System and the command from mission planner (the current mission planner is based on simple circle drawing). By clicking the circle tick, the quadrotor will follow the circle or else it will follow the commands from the transmitter.