

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 com-

pletely 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 throughout 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 |
|-------------------------------------------------------------------|------|
| LIST OF TABLES | vii |
| LIST OF FIGURES | viii |
| CHAPTER | |
| 1 OVERVIEW | 1 |
| 1.1 Introduction and Motivation | 1 |
| 1.2 Literature Survey | 2 |
| 1.3 Contribution of Work: Questions to be addressed | 6 |
| 1.4 Organization of Thesis | 11 |
| 1.5 Summary and Conclusions | 12 |
| 2 Nonlinear Model & Linearization | 14 |
| 2.1 Introduction and Overview | 14 |
| 2.2 Quadrotor Nonlinear Model | 14 |
| 2.2.1 Assumptions | 14 |
| 2.2.2 Kinematics | 15 |
| 2.2.3 Dynamics | 19 |
| 2.3 Actuator Model | 25 |
| 2.4 Linearization | 26 |
| 3 Parameters Measurement and Hardware Implementations | 33 |
| 3.1 Introduction and Overview | 33 |
| 3.2 Airframe Size, Mass and Moment Measurement | 33 |
| 3.2.1 Airframe Size | 33 |
| 3.2.2 Mass and Moment Measurement | 34 |
| 3.3 Propulsion System Test | 36 |
| 3.3.1 Thrust Coefficient and Torque Coefficient Measurement | 37 |

| CHAPTER | Page |
|---------|-------------------------------------------------------------|
| 3.4 | Design of the Flight Controller 41 |
| 3.5 | General Frame Structure of Hardware & Software 43 |
| 4 | Analysis of Linear Model 47 |
| 4.1 | Angular Movement Analysis 47 |
| 4.2 | Vertical Movement Analysis 50 |
| 4.3 | Translational Movement Analysis 51 |
| 4.4 | Summary and Conclusion 52 |
| 5 | Control Design Methods and Simulations 53 |
| 5.1 | Introduction and Overview 53 |
| 5.2 | Angular Movement Cascade Control 53 |
| 5.2.1 | Angular Rate Control 53 |
| 5.2.2 | Attitude Cascade Control 67 |
| 5.3 | Vertical Movement Cascade Control 71 |
| 5.3.1 | Vertical Velocity Control 72 |
| 5.3.2 | Altitude Control 75 |
| 5.4 | Translational Movement Cascade Control 78 |
| 5.4.1 | Translational Velocity Control 79 |
| 5.4.2 | Translational Position Control 83 |
| 5.5 | Summary and Conclusions 85 |
| 6 | Flight State Estimation 86 |
| 6.1 | Onboard Sensor Calibration 86 |
| 6.2 | IIR Low-Pass Filtering 91 |
| 6.3 | Quaternion Based Attitude Estimation 94 |
| 6.4 | Position & Velocity Estimation 97 |

| CHAPTER | Page |
|---------|----------------------------------------------------------------------------------------|
| 7 | Experimental Hardware Results and Comparisons with Simulation Results 99 |
| 7.1 | Attitude Command-response Graph with Simulation Results 99 |
| 7.2 | Altitude Command-response Graph with Simulation Results 100 |
| 7.3 | Translational Position Command-response Graph with Simulation Results 101 |
| 7.4 | Summary and Conclusions 101 |
| 8 | Conclusions and Directions for Future Research 102 |
| 8.1 | Summary of Work 102 |
| 8.2 | Directions for Future Research 103 |
| | REFERENCES 105 |
| | APPENDIX |
| A | MATLAB CODE 108 |
| B | MATLAB Based GUI 136 |
| C | MARK3 Flight Controller Firmware 144 |
| D | Hardware Assembly Instructions & Software Initialization 199 |
| D.1 | Hardware Assembly Instructions 200 |
| D.2 | Software Initialization 202 |

LIST OF TABLES

| Table | Page |
|------------------------------------------------------------------------------|------|
| 3.1 Moment of inertia experiment results | 36 |
| D.1 Items Needed for Assembling the <i>MARK3</i> flight controller | 200 |
| D.2 Items Needed for Assembling the 250mm quadrotor platform | 200 |

LIST OF FIGURES

| Figure | Page |
|---------------------------------------------------------------------|------|
| 1.1 Teensy 3.2 Microcontroller Unit | 6 |
| 1.2 GY-89 10DOF Sensor Module | 7 |
| 1.3 Digi XBee3 Zigbee 3.0 | 8 |
| 1.4 5V to 3.3V Step Down Voltage Regulator | 8 |
| 1.5 250mm quadrotor carbon fiber frame | 9 |
| 1.6 Actuator Set (Propulsion System) | 9 |
| 1.7 An idea to use HTC VIVE trackers to do robot localization | 10 |
| 2.1 Quadrotor Frame Setup | 15 |
| 2.2 Quadrotor Coordinate Diagram | 16 |
| 2.3 Pitch Movement | 20 |
| 2.4 Roll Movement | 21 |
| 2.5 Yaw Movement | 22 |
| 2.6 Full Actuator Dynamics | 30 |
| 2.7 Using Simulink Toolbox | 31 |
| 2.8 Linearized Actuator Dynamics | 31 |
| 3.1 250 quadrotor frame | 34 |
| 3.2 The bifilar pendulum experiment for three body axis | 35 |
| 3.3 Dynamometer Series 1580 | 37 |
| 3.4 Actuator Test Equipment Structure and GUI | 37 |
| 3.5 Thrust vs Motor Rotation Speed Fitting Curve | 38 |
| 3.6 Torque vs Motor Rotation Speed Fitting Curve | 39 |
| 3.7 PWM vs Motor Rotation Speed vs Battery Voltage..... | 40 |
| 3.8 Actuator Pole Identification Curve | 41 |
| 3.9 Block Diagram of Mark3 Flight Controller | 42 |

| Figure | Page |
|----------------------------------------------------------------------|------|
| 3.10 Mark3 Flight Controller Photo | 43 |
| 3.11 The Quadrotor Platform Electrical Architecture | 44 |
| 3.12 The <i>MARK3</i> Flight Controller Software Architecture | 45 |
| 3.13 The Experimental System Hardware Architecture | 46 |
| 4.1 Angular Rate Plant Diagram | 47 |
| 4.2 Angular Rate Movement Bode Plot | 48 |
| 4.3 Motor Speed vs Torque in Body Frame | 49 |
| 4.4 Angular Movement Plant Diagram | 50 |
| 4.5 Vertical Movement Plant Diagram | 50 |
| 4.6 Translational Movement Plant Diagram | 51 |
| 5.1 Close Loop Diagram of Angular Rate Control | 53 |
| 5.2 Open Loop Bode Plot When changing g | 54 |
| 5.3 Open Loop Bode Plot When changing z | 55 |
| 5.4 Sensitivity Bode Plot When changing g | 56 |
| 5.5 Sensitivity Bode Plot When changing z | 56 |
| 5.6 Complementary Sensitivity Bode Plot When changing g | 57 |
| 5.7 Complementary Sensitivity Bode Plot When changing z | 58 |
| 5.8 Input Disturbance to Output Bode Plot When changing g | 59 |
| 5.9 Input Disturbance to Output Bode Plot When changing z | 59 |
| 5.10 Reference to Control Bode Plot When changing g | 60 |
| 5.11 Reference to Control Bode Plot When changing z | 61 |
| 5.12 Sensor Noise to Output Bode Plot When changing g | 62 |
| 5.13 Sensor Noise to Output Bode Plot When changing z | 62 |
| 5.14 Output Disturbance to Control Bode Plot When changing g | 63 |

| Figure | Page |
|--------------------------------------------------------------------------------------------|------|
| 5.15 Output Disturbance to Control Bode Plot When changing z | 64 |
| 5.16 Step Response When changing g | 65 |
| 5.17 Step Response When changing z | 65 |
| 5.18 Angular Rate Control Open Loop Root Locus..... | 67 |
| 5.19 Close Loop Diagram of Attitude Control..... | 68 |
| 5.20 Open Loop Root Locus of Attitude Control..... | 69 |
| 5.21 Complimentary Sensitivity of Attitude Control when k_p changes..... | 69 |
| 5.22 Sensitivity of Attitude Control when k_p changes | 70 |
| 5.23 Sensor Noise to Output of Attitude Control when k_p changes..... | 70 |
| 5.24 Step Response of Attitude Control when k_p changes | 71 |
| 5.25 Close Loop Diagram of Vertical Movement Control | 72 |
| 5.26 Complimentary Sensitivity of Vertical Velocity Control when g changes | 73 |
| 5.27 Sensitivity of Vertical Velocity Control when g changes | 74 |
| 5.28 Sensor Noise to Output of Vertical Velocity Control when g changes ... | 74 |
| 5.29 Step Response of Vertical Velocity Control when g changes..... | 75 |
| 5.30 Open Loop Root Locus of Altitude Control | 76 |
| 5.31 Complimentary Sensitivity of Altitude Control when k_p changes | 76 |
| 5.32 Sensitivity of Altitude Control when k_p changes | 77 |
| 5.33 Sensor Noise to Output of Altitude Control when k_p changes | 77 |
| 5.34 Step Response of Altitude Control when k_p changes | 78 |
| 5.35 Close Loop Diagram of Translational Movement Cascade Control..... | 79 |
| 5.36 Open Loop Root Locus of Translational Velocity Control | 80 |
| 5.37 Complimentary Sensitivity of Translational Velocity Control when g changes | 80 |

| Figure | Page |
|----------------------------------------------------------------------------------------------------------------------|------|
| 5.38 Sensitivity of Translational Velocity Control when g changes | 81 |
| 5.39 Sensor Noise to Output of Translational Velocity Control when g changes | 81 |
| 5.40 Input to Control of Translational Velocity Control when g changes | 82 |
| 5.41 Step Response of Translational Velocity Control when g changes | 82 |
| 5.42 Open Loop Root Locus of Translational Position Control | 83 |
| 5.43 CF of Translational Position Control when k_p changes | 84 |
| 5.44 Sensitivity of Translational Position Control when k_p changes | 84 |
| 5.45 Step Response of Translational Position Control when k_p changes | 85 |
| 6.1 gyroscope p temperature calibration | 86 |
| 6.2 gyroscope q temperature calibration | 87 |
| 6.3 gyroscope r temperature calibration | 87 |
| 6.4 Reference for Sensor Orientation While Performing 6-point Tumble Calibration (ST DT0053 Design tip) | 88 |
| 6.5 Calibrated ACC output vs. Raw output in x axis | 90 |
| 6.6 Calibrated ACC output vs. Raw output in y axis | 90 |
| 6.7 Calibrated ACC output vs. Raw output in z axis | 91 |
| 6.8 Fast Fourier Transform of Gyroscope Output when Motors off | 92 |
| 6.9 Fast Fourier Transform of Gyroscope Output when Motors on | 92 |
| 6.10 Different IIR Filters Spectrum Using MATLAB Toolbox | 93 |
| 6.11 Filtered Gyroscope Zero Output Spectrum with Motors on | 94 |
| 6.12 Complimentary Filter Pitch Angle Estimation | 96 |
| 6.13 Explicit Complimentary Filter Diagram | 97 |
| 6.14 Complimentary Filter Velocity in x axis Estimation | 98 |
| 6.15 Complimentary Filter Position in x axis Estimation | 98 |

| Figure | Page |
|-------------------------------------------------------------|------|
| 7.1 ϕ Step Response with Simulated Data | 99 |
| 7.2 ψ Ramp Response with Simulated Data | 100 |
| 7.3 Altitude Ramp Response with Simulated Data | 100 |
| 7.4 Position Ramp Response with Simulated Data | 101 |
| D.1 <i>MARK3</i> Flight Controller Wiring Instruction | 201 |
| D.2 <i>MARK3</i> Ground Station | 203 |

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 actuator 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 package 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 these questions are critical in order to move substantively 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 $3.3v \sim 5v$ 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).

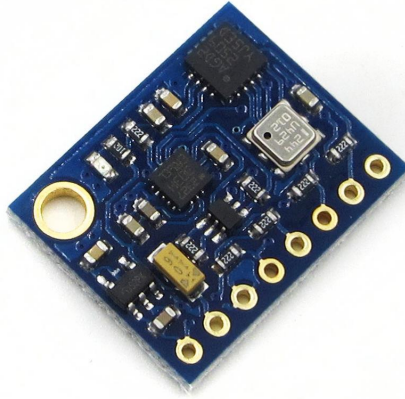


Figure 1.2: GY-89 10DOF Sensor Module

(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 to 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) provides more detailed answers. 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 kinematics, 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.

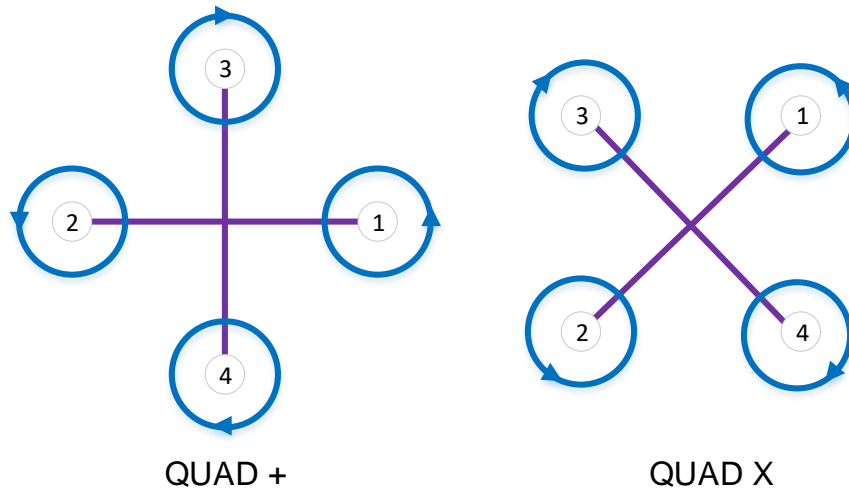


Figure 2.1: Quadrotor Frame Setup

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.

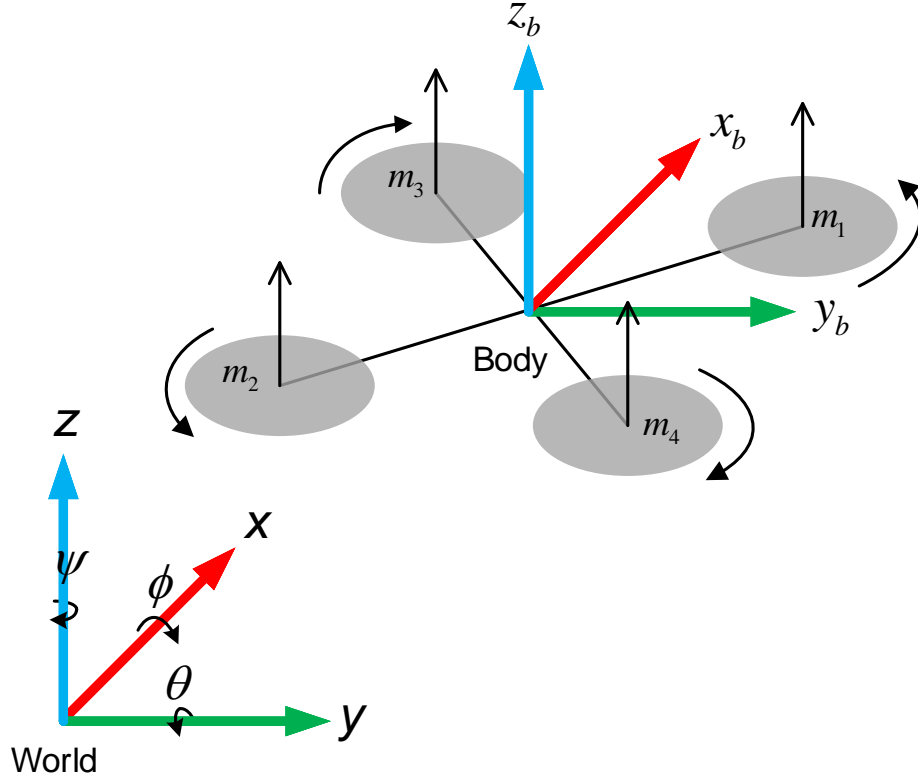


Figure 2.2: Quadrotor Coordinate Diagram

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

- Rotation about z axis is

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

- Rotation about y axis is

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

- Rotation about x axis is

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

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

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

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

$$R_{B \rightarrow E} = R_{E \rightarrow B}^T \quad (2.5)$$

And we have

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

Thus

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = R_{B \rightarrow E} \begin{bmatrix} x_b \\ y_b \\ z_b \end{bmatrix} \quad (2.7)$$

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

$$\begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} \dot{\phi} \\ 0 \\ 0 \end{bmatrix} + R_\phi \begin{bmatrix} 0 \\ \dot{\theta} \\ 0 \end{bmatrix} + R_\phi R_\theta \begin{bmatrix} 0 \\ 0 \\ \dot{\psi} \end{bmatrix} = \Omega_{E \rightarrow B} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \quad (2.8)$$

Then

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

And

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

Finally

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

2.2.3 Dynamics

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

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

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

$$T = T_1 + T_2 + T_3 + T_4 \quad (2.13)$$

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

$$\tau_\psi = m_1 + m_2 - m_3 - m_4 \quad (2.14)$$

The differential 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

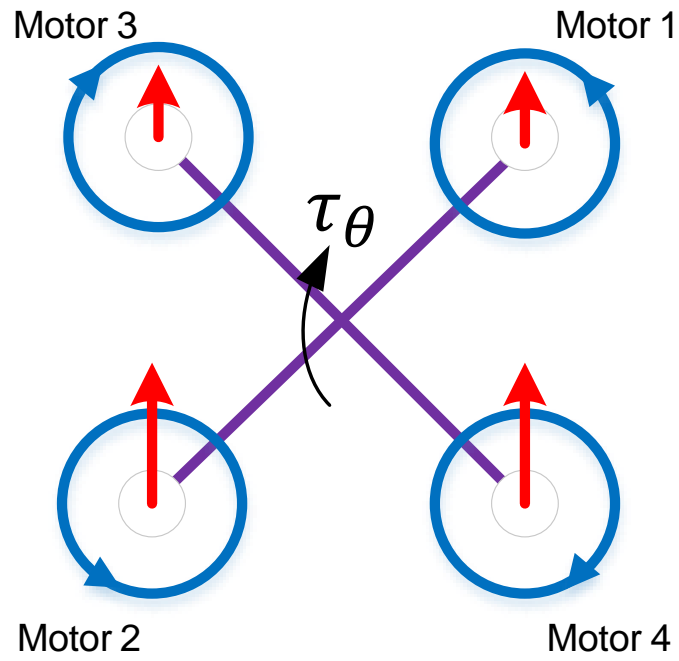


Figure 2.3: Pitch Movement

- **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

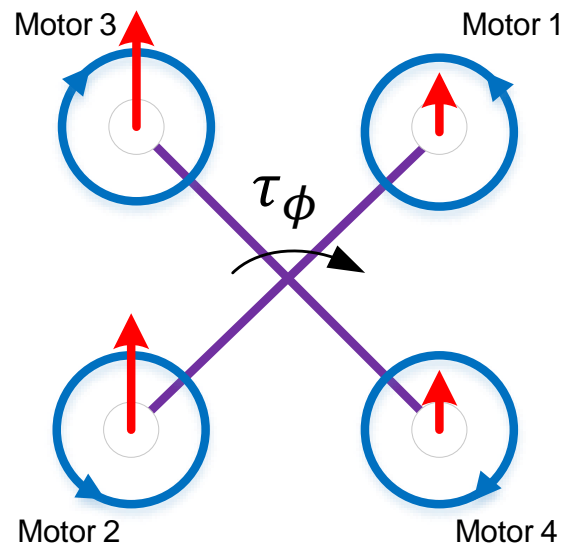


Figure 2.4: Roll Movement

- **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

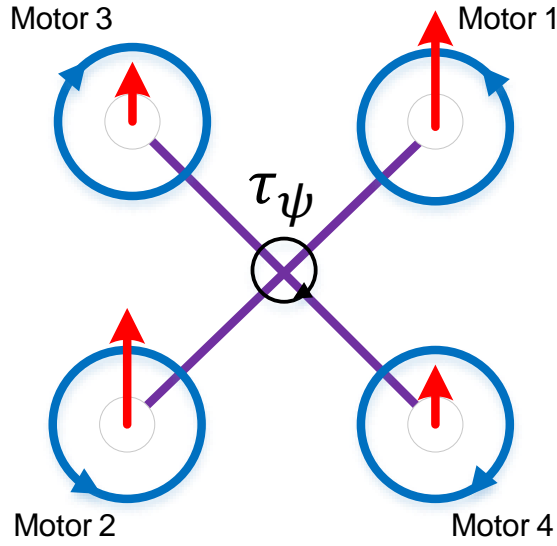


Figure 2.5: Yaw Movement

And we have

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

Similarly

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

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

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

Then the dynamics equations of force is:

$$m \begin{bmatrix} \dot{v}_x \\ \dot{v}_y \\ \dot{v}_z \end{bmatrix} = mg \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix} + R_{B \rightarrow E} T \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} - \begin{bmatrix} K_1 v_x \\ K_2 v_y \\ K_3 v_z \end{bmatrix} \quad (2.18)$$

So position movement of quadrotor can be expressed as:

$$\begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix} = \begin{bmatrix} \dot{v}_x \\ \dot{v}_y \\ \dot{v}_z \end{bmatrix} = \begin{bmatrix} \frac{T}{m}(-\cos(\psi)\sin(\theta)\cos(\phi) - \sin(\psi)\sin(\phi)) - K_1 \dot{x} \\ \frac{T}{m}(\cos(\psi)\sin(\phi) - \sin(\psi)\sin(\theta)\cos(\phi)) - K_2 \dot{y} \\ \frac{T}{m}\cos(\phi)\cos(\theta) - g - K_3 \dot{z} \end{bmatrix} \quad (2.19)$$

The rotation kinematics equation of quadrotor is:

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

And $J = \text{diag}[J_x, J_y, J_z]$ 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:

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

Aerodynamic drag torque is:

$$\tau_{af} = K_{af}\nu \quad (2.22)$$

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

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

Then we have the equations of angular movement of the quadrotor

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

Then we can get the whole nonlinear quadrotor rigid body dynamics

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

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

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

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

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

$$T_i = b\Omega_i^2 \quad (2.28)$$

$$m_i = d\Omega_i^2 \quad (2.29)$$

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

$$L \frac{di}{dt} = u - RI - K_e \Omega_i \quad (2.30)$$

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

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

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.

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

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

$$\dot{\Omega}_i = -A\Omega_i + Bu + C \quad (2.33)$$

Then

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

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.

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

2.4 Linearization

- **Linearization of rigid-body dynamics**

The non-linear rigid-body dynamics is represented as

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

Where

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

And

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

\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

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

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

$$\delta \dot{\mathbf{X}} = A\delta \mathbf{X} + B\delta \mathbf{U} \quad (2.40)$$

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

$$A = \left. \frac{\delta f}{\delta \mathbf{X}} \right|_{\mathbf{x}=\mathbf{x}_{equil}, \mathbf{U}=\mathbf{U}_{equil}} \quad (2.41)$$

$$B = \left. \frac{\delta f}{\delta \mathbf{U}} \right|_{\mathbf{x}=\mathbf{x}_{equil}, \mathbf{U}=\mathbf{U}_{equil}} \quad (2.42)$$

Then we have

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

And

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

Then we have the linearized quadrotor rigid-body dynamics

$$\dot{\mathbf{X}}_{rig} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{v}_x \\ \dot{v}_y \\ \dot{v}_z \\ \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \\ \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} v_x \\ v_y \\ v_z \\ -g\theta - K_1v_x \\ g\phi - K_2v_y \\ \frac{T}{m} - g - K_3v_z \\ p \\ q \\ r \\ \frac{\tau_\phi}{J_x} - K_4p \\ \frac{\tau_\theta}{J_y} - K_5q \\ \frac{\tau_\psi}{J_z} - K_6r \end{bmatrix} = A\mathbf{X}_{rig} + B\mathbf{U}_{rig} \quad (2.45)$$

- Linearization of actuator dynamics

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

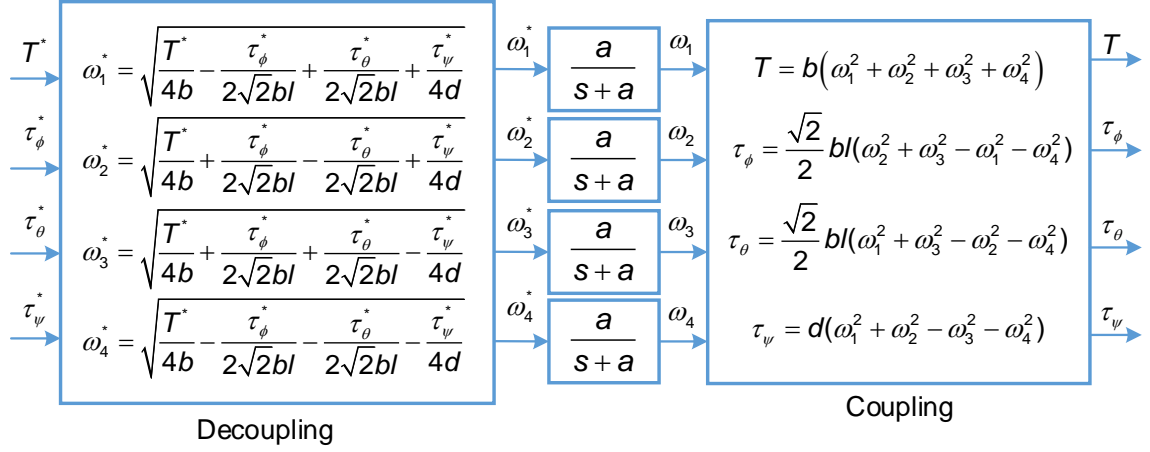


Figure 2.6: Full Actuator Dynamics

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 $\omega_1 = \omega_2 = \omega_3 = \omega_4 = 842.99rad/s$ (Optimization Method: Gradient Descent with Elimination; Algorithm: Active-Set).

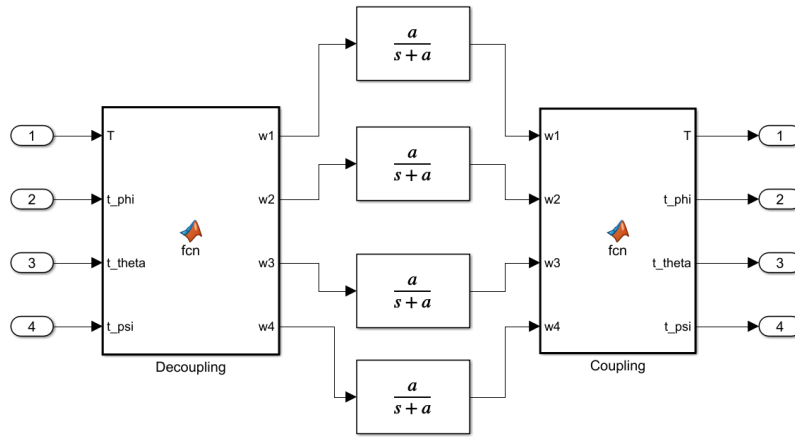


Figure 2.7: Using Simulink Toolbox

And we get the linearized actuator dynamic model in Figure 2.8

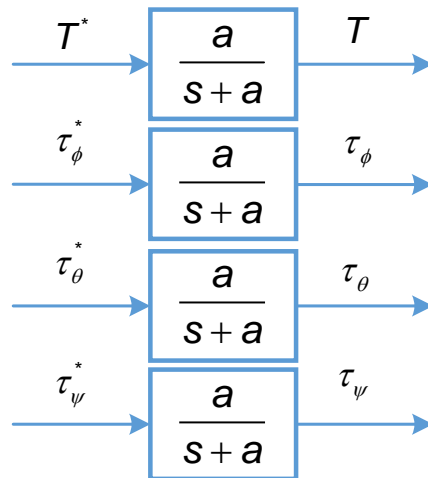


Figure 2.8: Linearized Actuator Dynamics

With $\mathbf{U}_{act}^* = [T^*, \tau_\phi^*, \tau_\theta^*, \tau_\psi^*]^T$, linearized actuator model (state space represen-

tation) can be written as:

$$\dot{\mathbf{U}}_{act} = -a\mathbf{I}_{4 \times 4}\mathbf{U}_{act} + a\mathbf{I}_{4 \times 4}\mathbf{U}_{act}^* \quad (2.46)$$

- **Whole Quadrotor Linearized Model**

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

$$\dot{\mathbf{X}} = \begin{bmatrix} \mathbf{I}_{3 \times 3} & & \mathbf{0}_{3 \times 13} \\ \mathbf{0}_{1 \times 3} & -g & \mathbf{0}_{1 \times 12} \\ \mathbf{0}_{1 \times 4} & g & \mathbf{0}_{1 \times 11} \\ \mathbf{0}_{1 \times 12} & \frac{1}{m} & \mathbf{0}_{1 \times 3} \\ \mathbf{0}_{3 \times 6} & \mathbf{I}_{3 \times 3} & \mathbf{0}_{3 \times 7} \\ \mathbf{0}_{3 \times 9} & \text{diag}[\frac{1}{J_x}, \frac{1}{J_y}, \frac{1}{J_z}] & \mathbf{0}_{3 \times 4} \\ \mathbf{0}_{4 \times 12} & & -a\mathbf{I}_{4 \times 4} \end{bmatrix} \mathbf{X} + \begin{bmatrix} \mathbf{0}_{12 \times 4} \\ a\mathbf{I}_{4 \times 4} \end{bmatrix} \mathbf{U} \quad (2.47)$$

Chapter 3

PARAMETERS MEASUREMENT AND HARDWARE IMPLEMENTATIONS

3.1 Introduction and Overview

This chapter 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 Airframe 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 $l = 0.125m$.

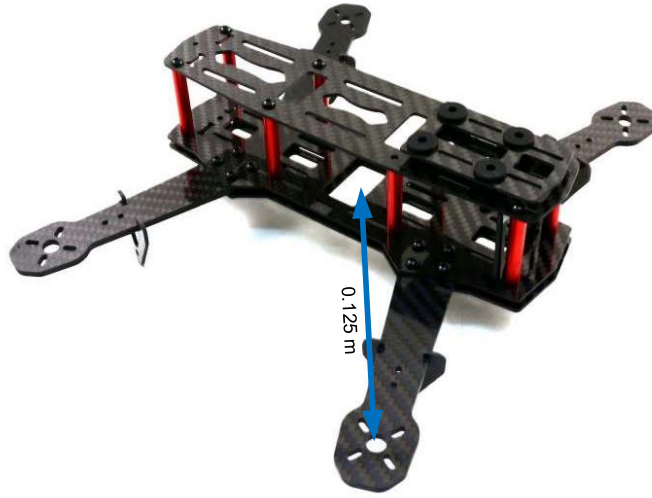


Figure 3.1: 250 quadrotor frame

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 inertia 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

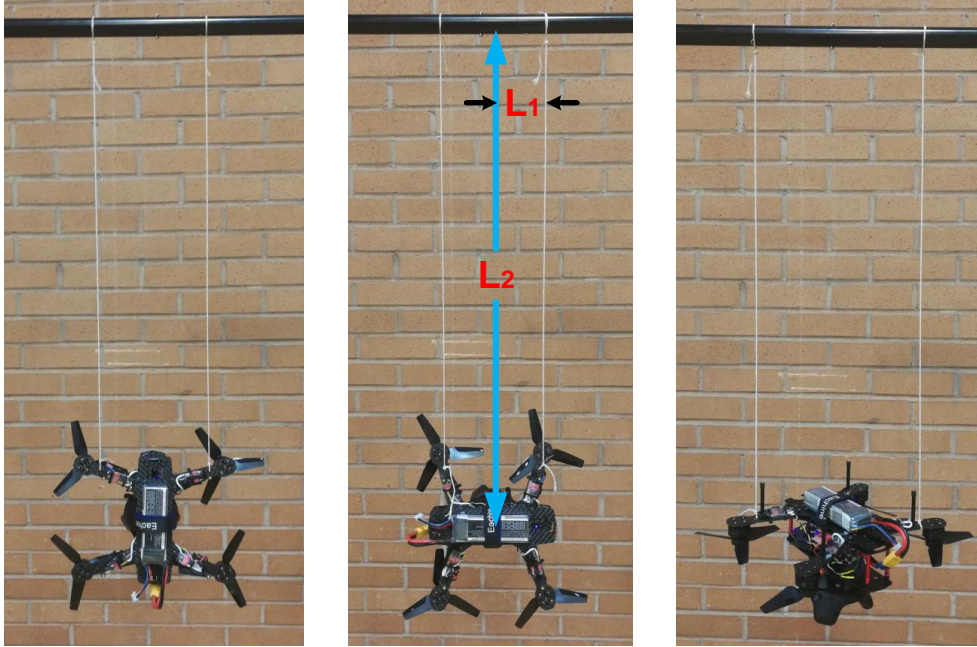


Figure 3.2: The bifilar pendulum experiment for three body axis

the equation used to compute the moment of inertia is

$$J = \frac{mgT^2 L_1^2}{4\pi^2 L_2} \quad (3.1)$$

where T is the period of each oscillation. The free oscillation can be regarded as simple harmonic motion because the 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.

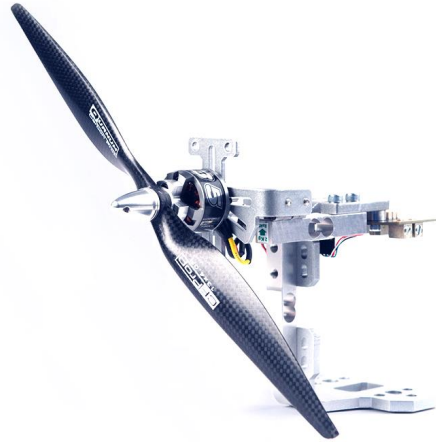


Figure 3.3: Dynamometer Series 1580

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.

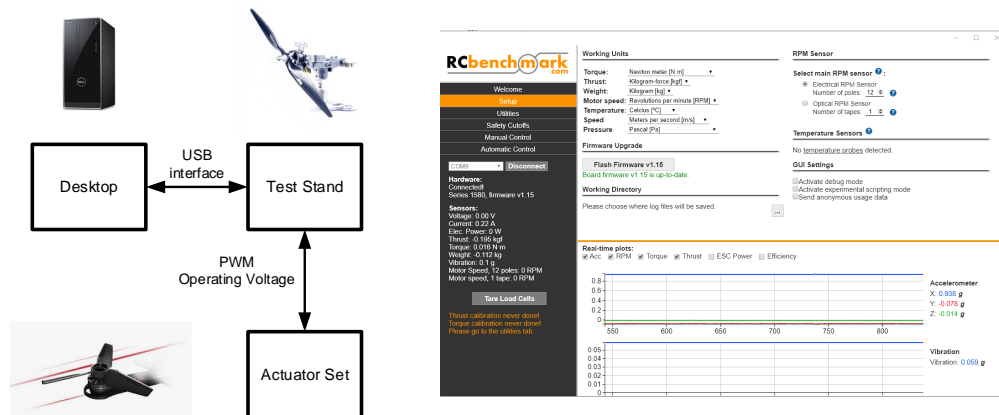


Figure 3.4: Actuator Test Equipment Structure and GUI

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 $b = 1.91 \times 10^{-6} N s^2/rad^2$.

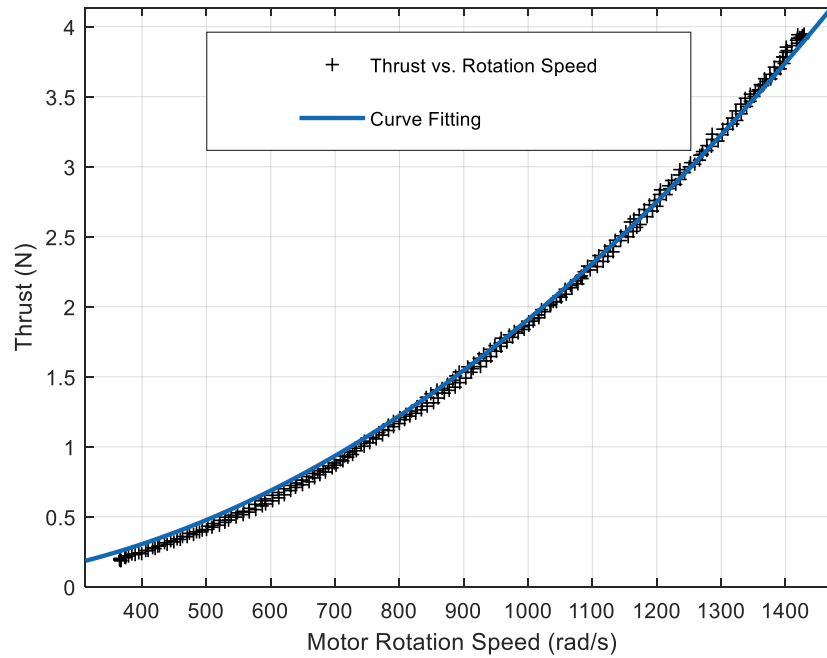


Figure 3.5: Thrust vs Motor Rotation Speed Fitting Curve

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

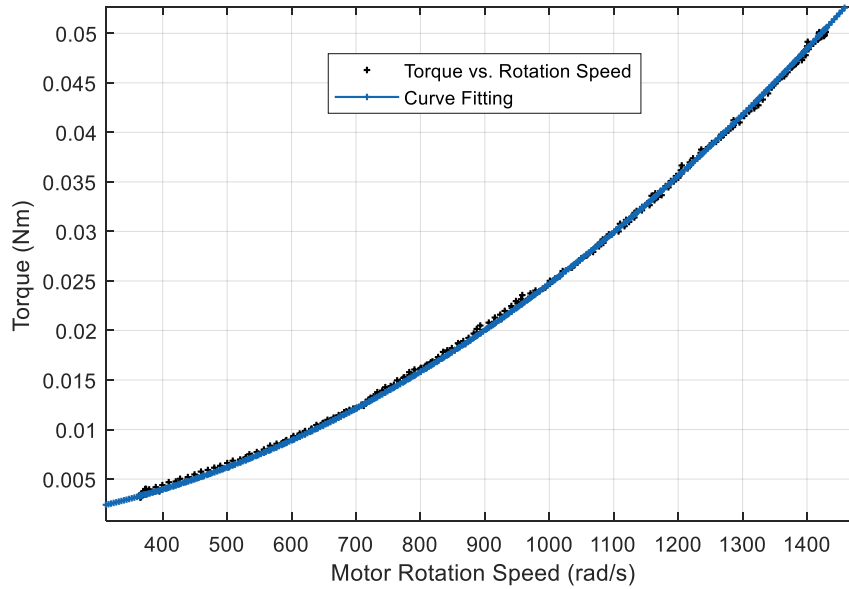


Figure 3.6: Torque vs Motor Rotation Speed Fitting Curve

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 voltages which are controlled by a power supply unit, we can find a linear function to represent the 3-D curve fitting.

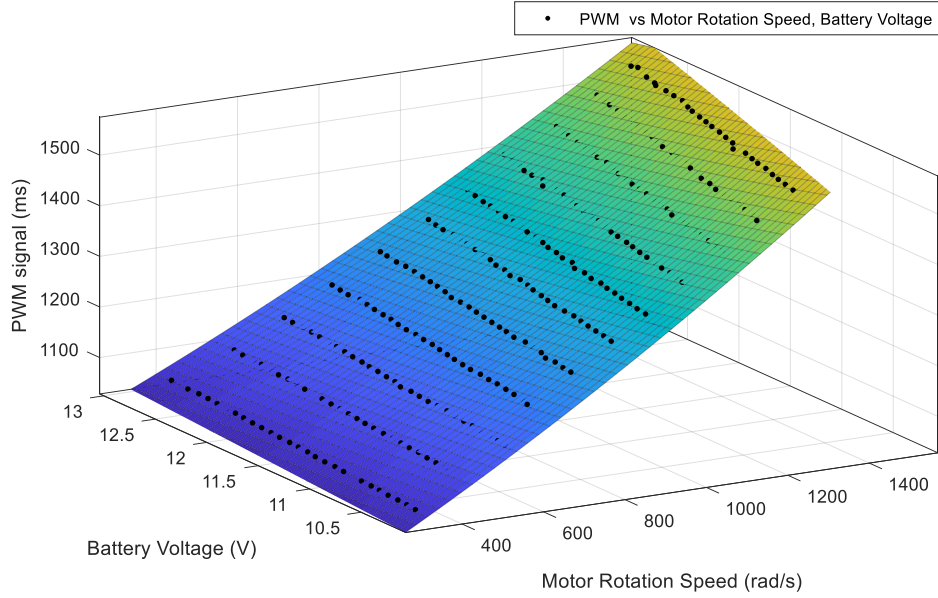


Figure 3.7: PWM vs Motor Rotation Speed vs Battery Voltage

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):

$$\epsilon = \frac{\omega^2 + a_1\omega + a_2}{a_3u_{bat} + a_4} + a_5 \quad (3.2)$$

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.

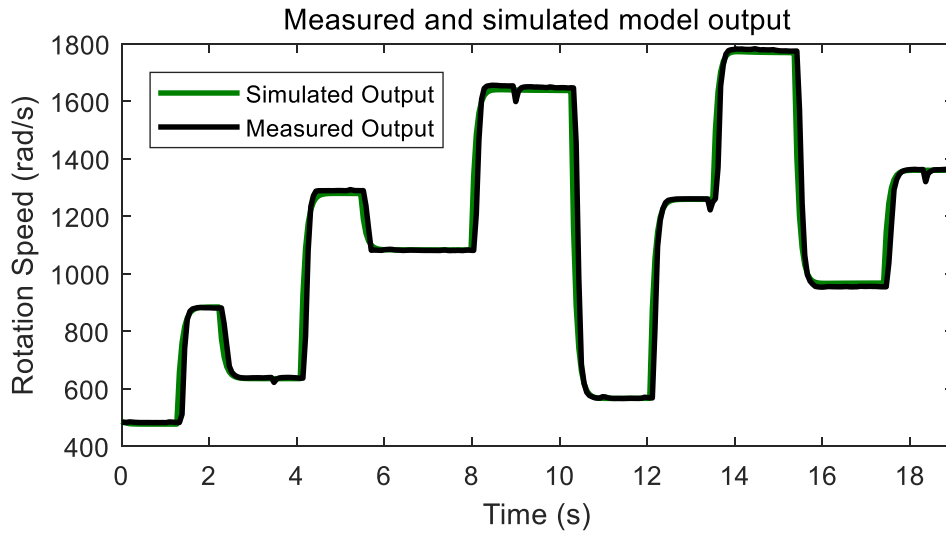


Figure 3.8: Actuator Pole Identification Curve

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.

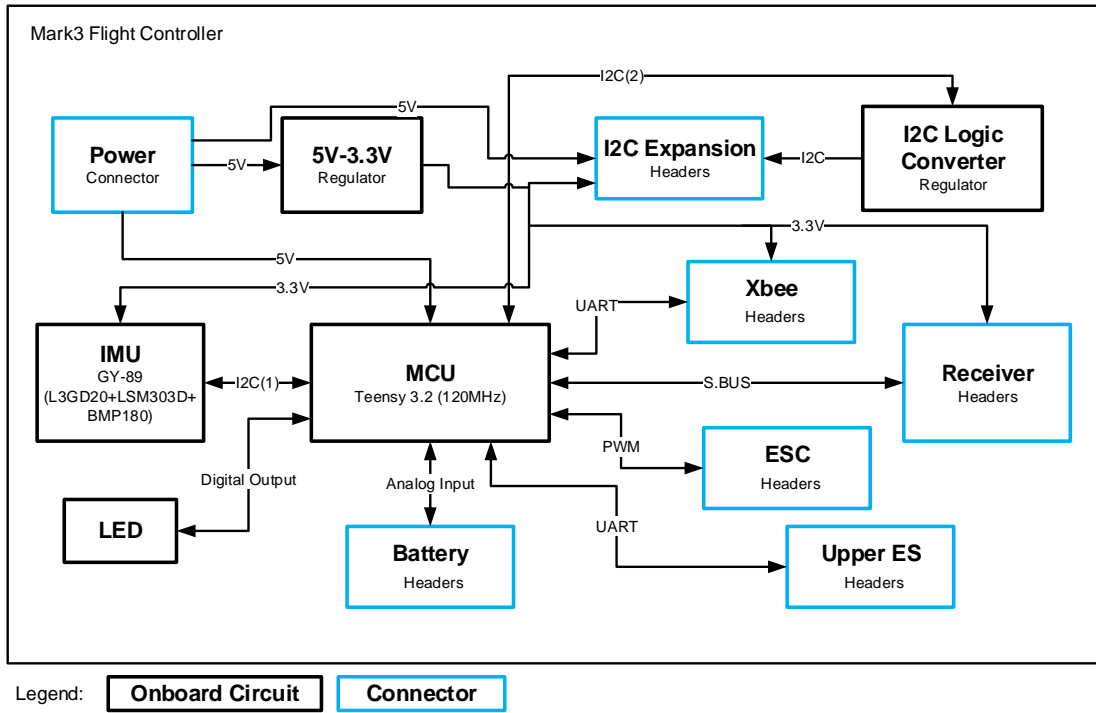


Figure 3.9: Block Diagram of Mark3 Flight Controller

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.

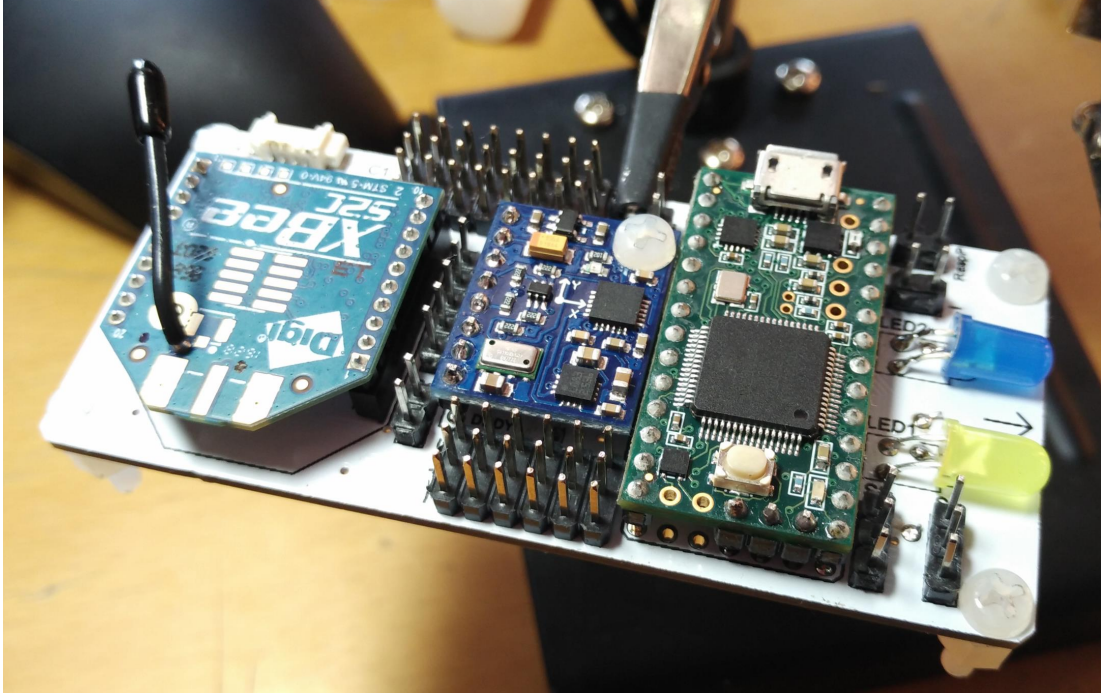


Figure 3.10: Mark3 Flight Controller Photo

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.

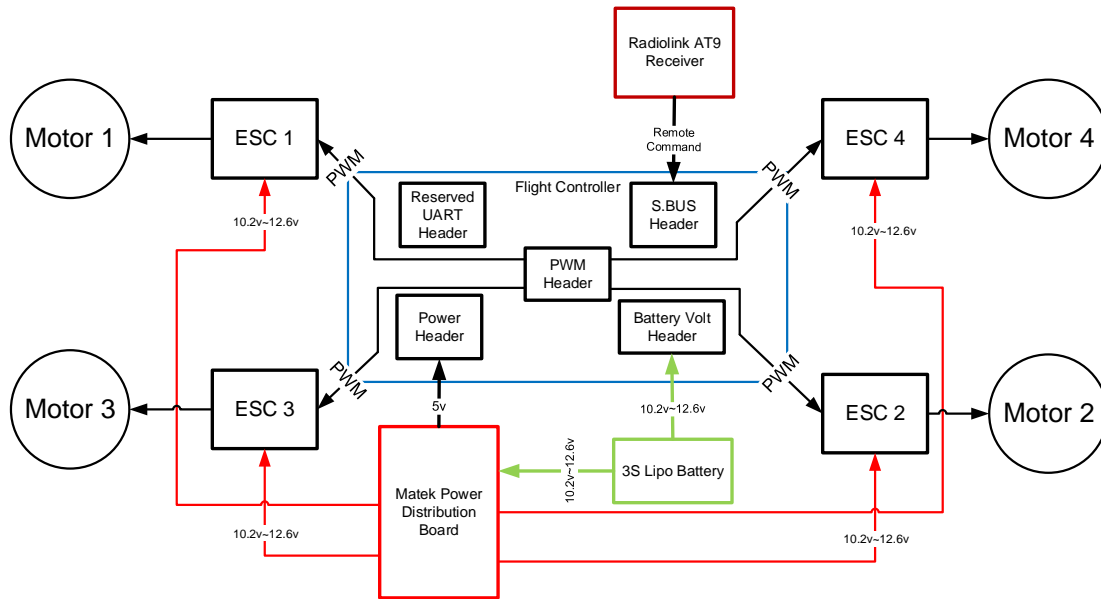


Figure 3.11: The Quadrotor Platform Electrical Architecture

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 Software 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.

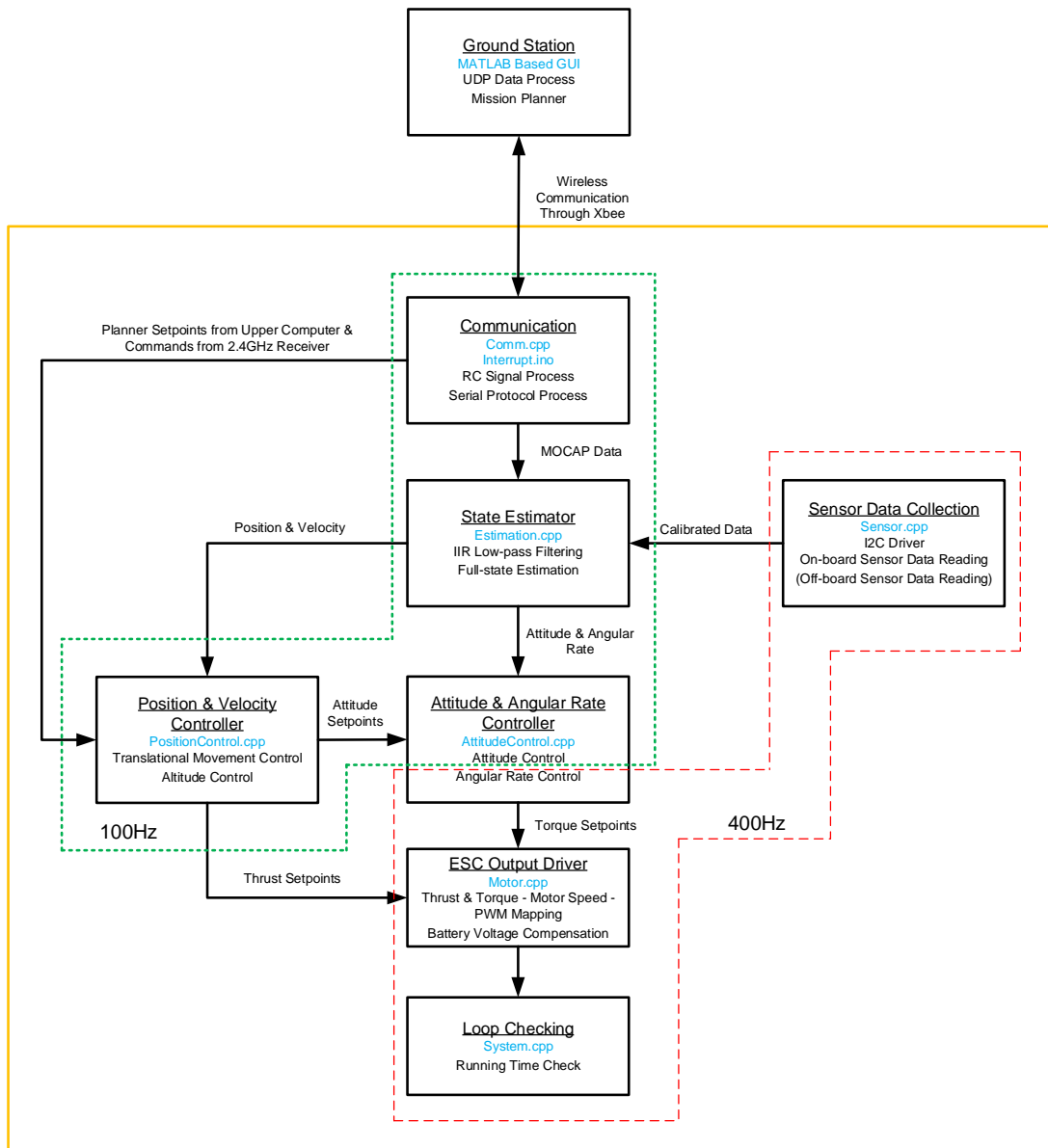


Figure 3.12: The *MARK3* Flight Controller Software Architecture

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.

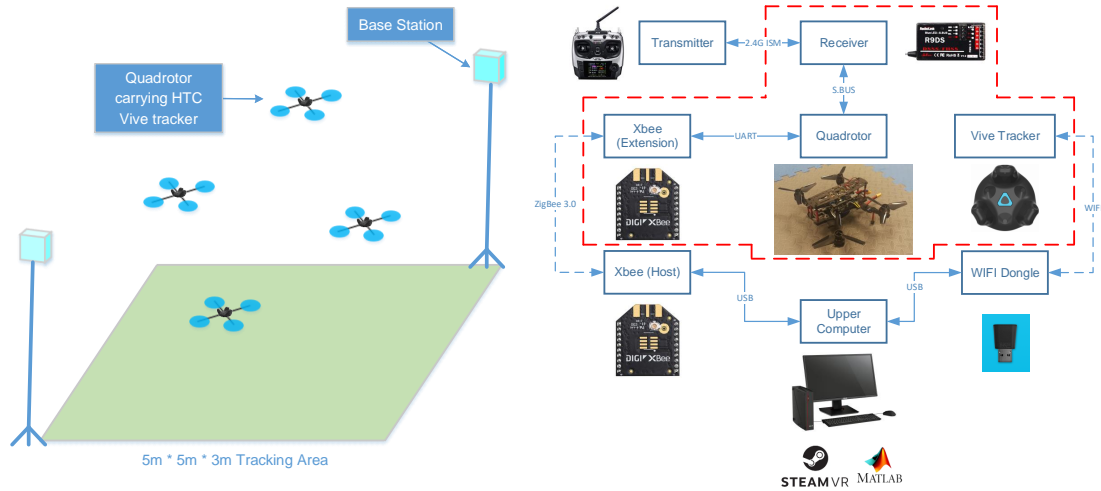


Figure 3.13: The Experimental System Hardware Architecture

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

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

Then

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

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

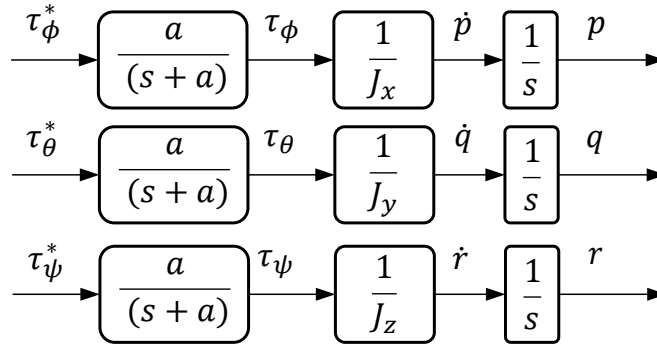


Figure 4.1: Angular Rate Plant Diagram

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

Roll Rate

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

Pitch Rate

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

Yaw Rate

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

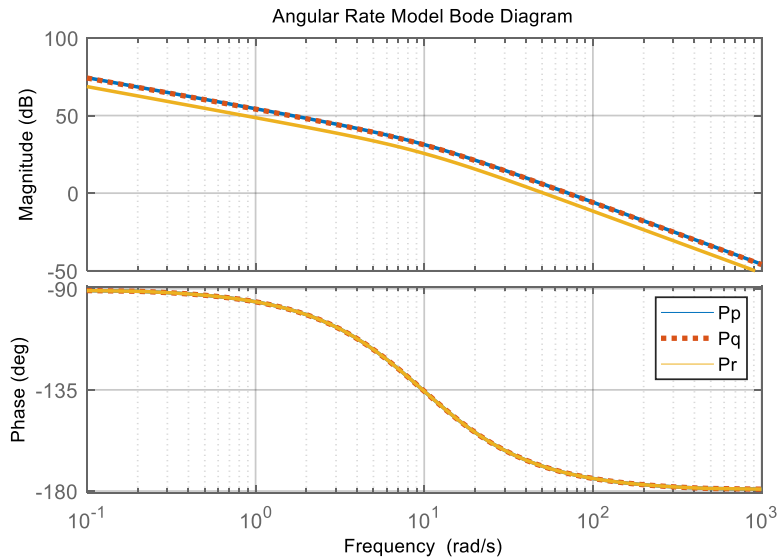


Figure 4.2: Angular Rate Movement Bode Plot

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.

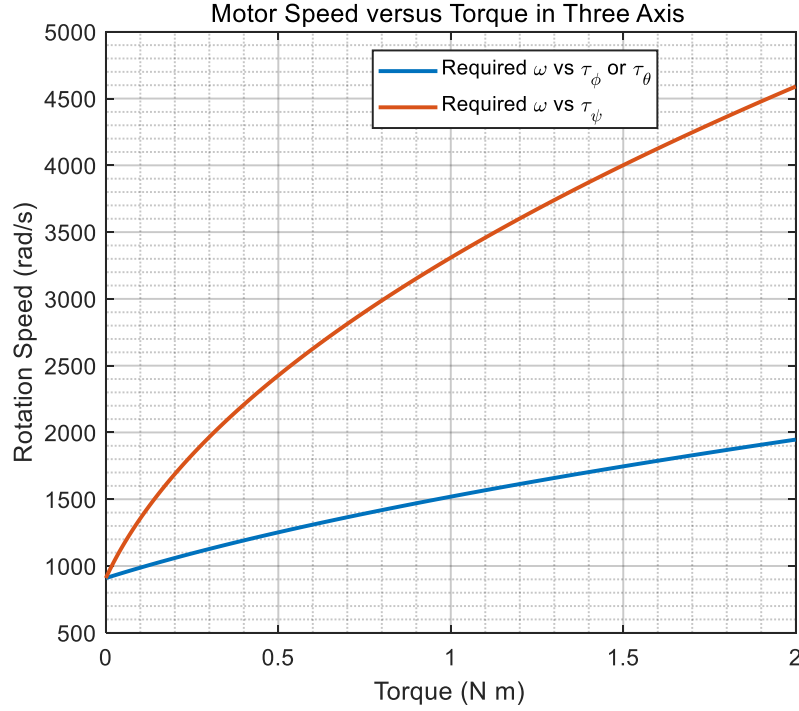


Figure 4.3: Motor Speed vs Torque in Body Frame

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 ($d = 2.47 \times 10^{-8} \text{ Nm s}^2/\text{rad}^2$) is much smaller than the combined actuator thrust factor ($\frac{\sqrt{2}}{2}bl = 1.69 \times 10^{-7} \text{ Nm s}^2/\text{rad}^2$).

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.

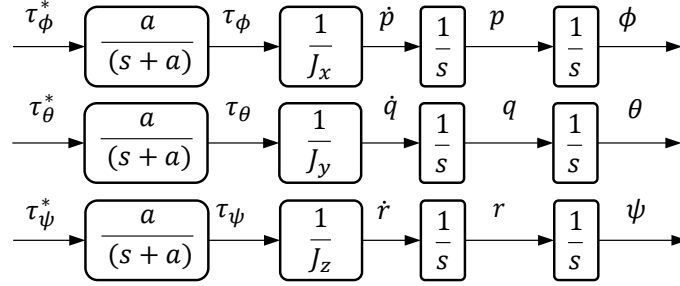


Figure 4.4: Angular Movement Plant Diagram

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

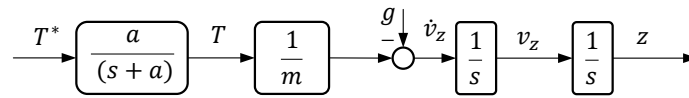


Figure 4.5: Vertical Movement Plant Diagram

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:

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

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

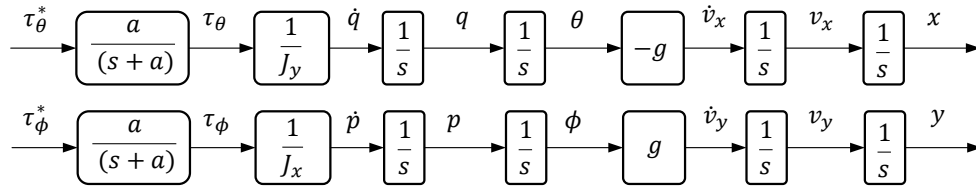


Figure 4.6: Translational Movement Plant Diagram

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:

$$a = \cos(\psi) \quad (4.7)$$

$$b = \sin(\psi) \quad (4.8)$$

$$c_1 = mT\theta^* \quad (4.9)$$

$$c_2 = mT\phi^* \quad (4.10)$$

And finally we have:

$$\phi_{act}^* = a\sin(ac_2 - bc_1) \quad (4.11)$$

and

$$\theta_{act}^* = a\sin(-(ac_1 + bc_2)/\cos(\theta^*)) \quad (4.12)$$

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.

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 trade 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 equipped 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.

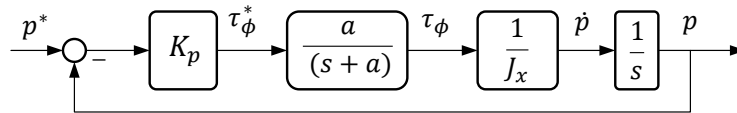


Figure 5.1: Close Loop Diagram of Angular Rate Control

Based on the model obtained from the previous section

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

we design a PD controller

$$K_p = g(s + z) \quad (5.2)$$

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

$$s = 9.79 \quad (5.3)$$

The open loop transfer function L_p is given by

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

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.

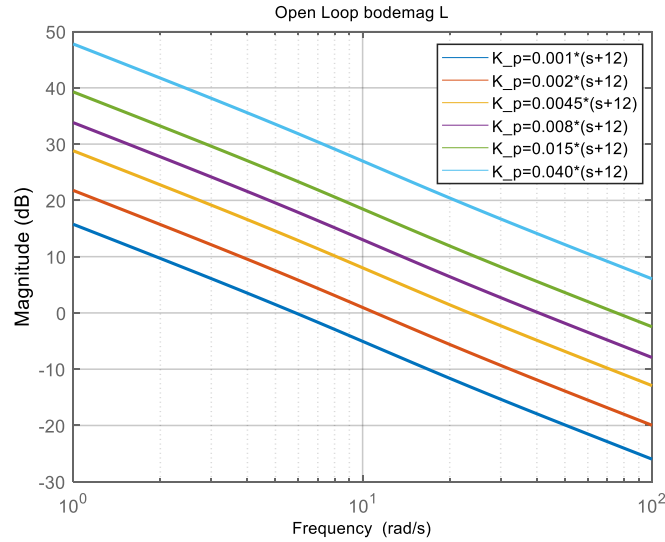


Figure 5.2: Open Loop Bode Plot When changing g

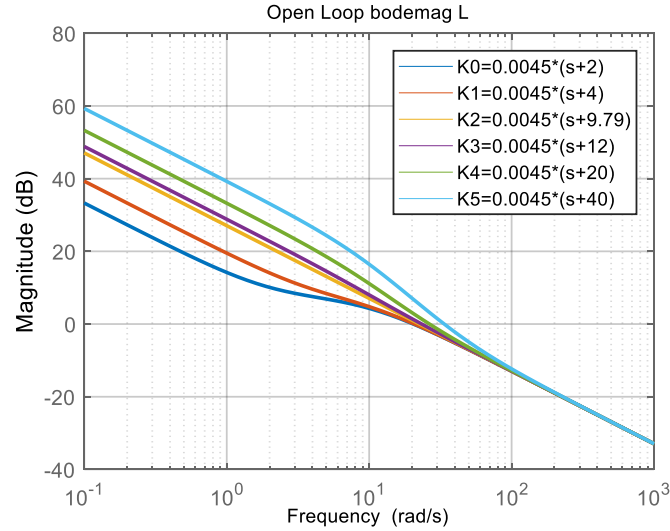


Figure 5.3: Open Loop Bode Plot When changing z

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 doesn'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_{doy}). The sensitivity bode plot is presented in Figure 5.4 and Figure 5.5

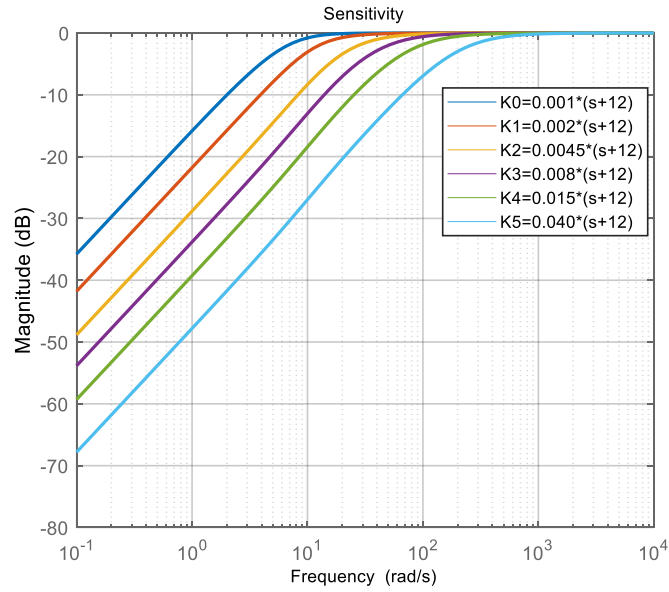


Figure 5.4: Sensitivity Bode Plot When changing g

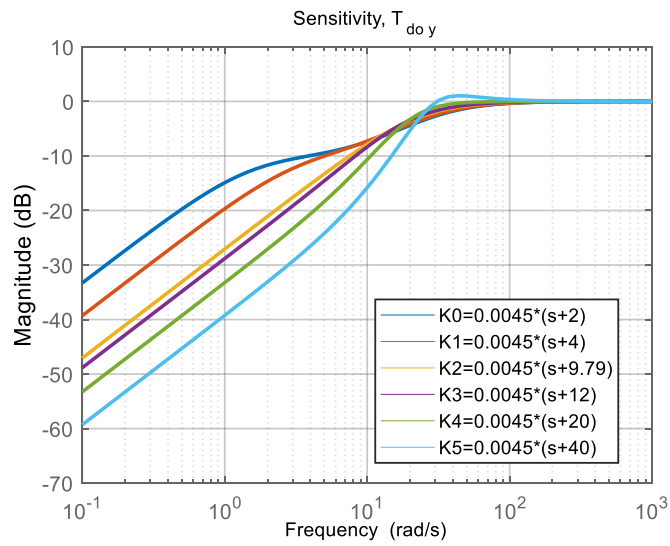


Figure 5.5: Sensitivity Bode Plot When changing z

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

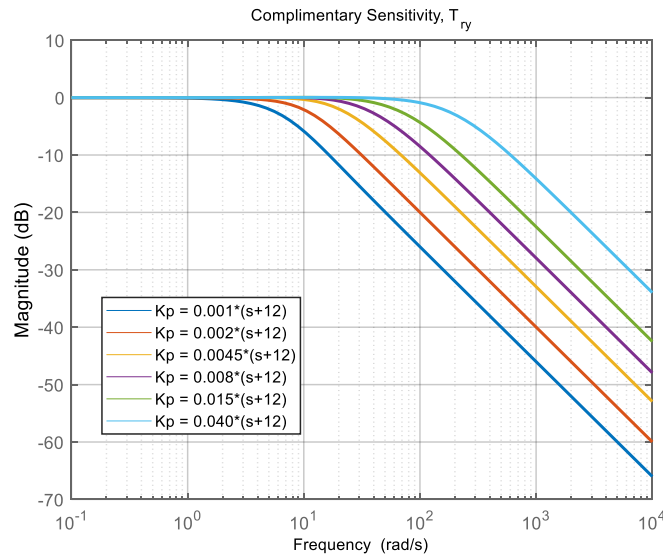


Figure 5.6: Complementary Sensitivity Bode Plot When changing g

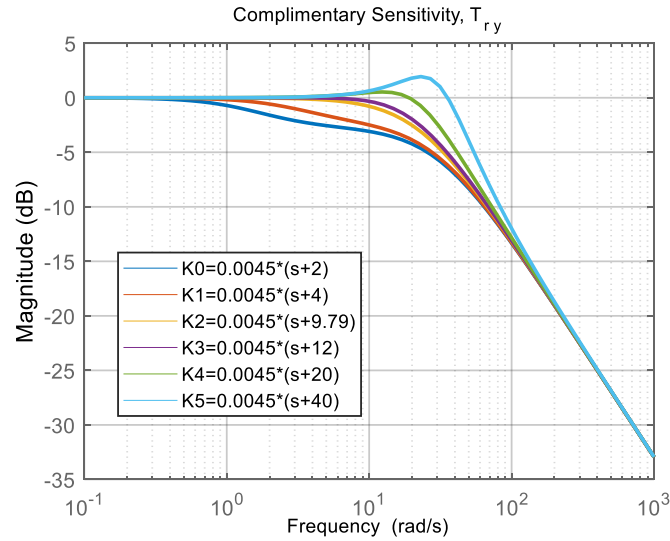


Figure 5.7: Complementary Sensitivity Bode Plot When changing z

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 $T_{d,y}$ (PS). The input disturbance to output bode plot is presented in Figure 5.8 and Figure 5.9

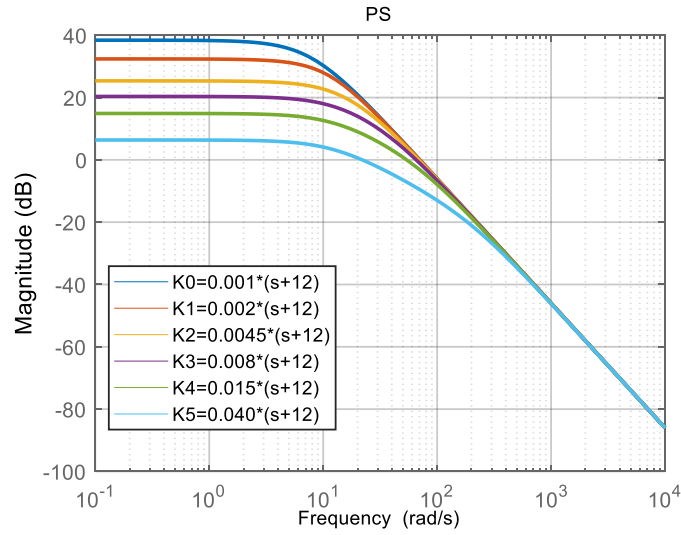


Figure 5.8: Input Disturbance to Output Bode Plot When changing g

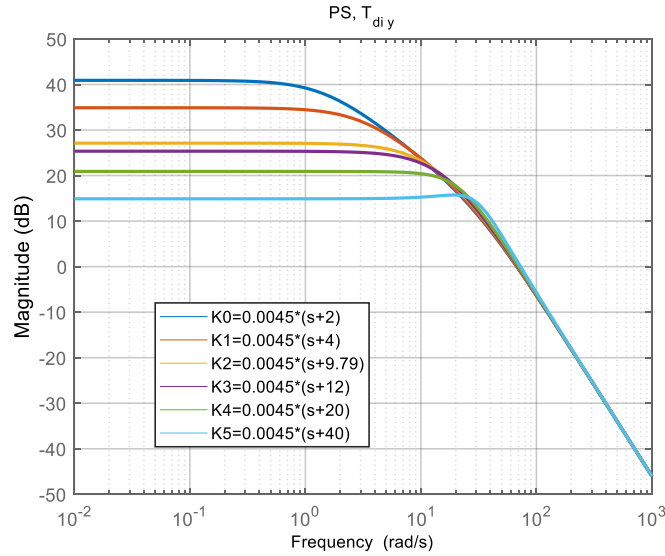


Figure 5.9: Input Disturbance to Output Bode Plot When changing z

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

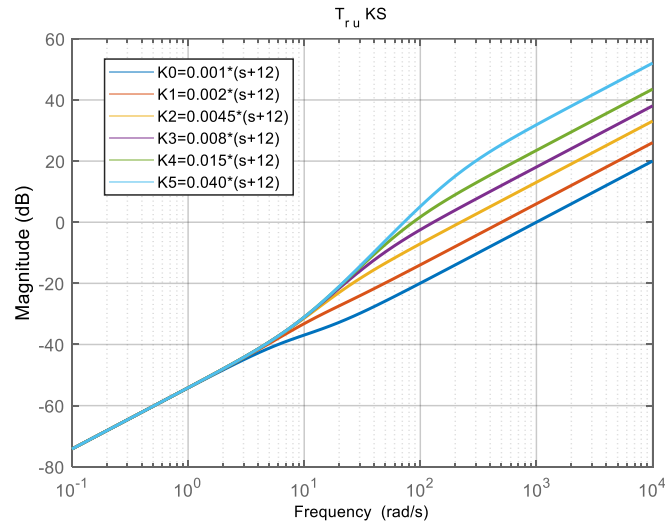


Figure 5.10: Reference to Control Bode Plot When changing g

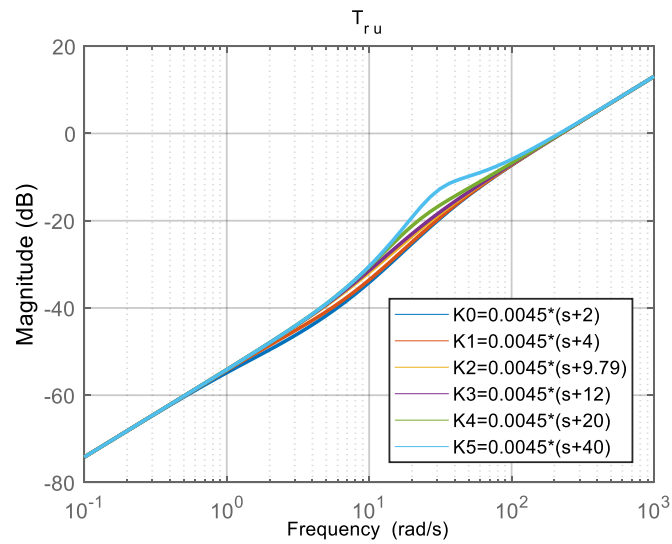


Figure 5.11: Reference to Control Bode Plot When changing z

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} ($T_{d,u}$). The sensor noise to output bode plot is presented in Figure 5.12 and Figure 5.13

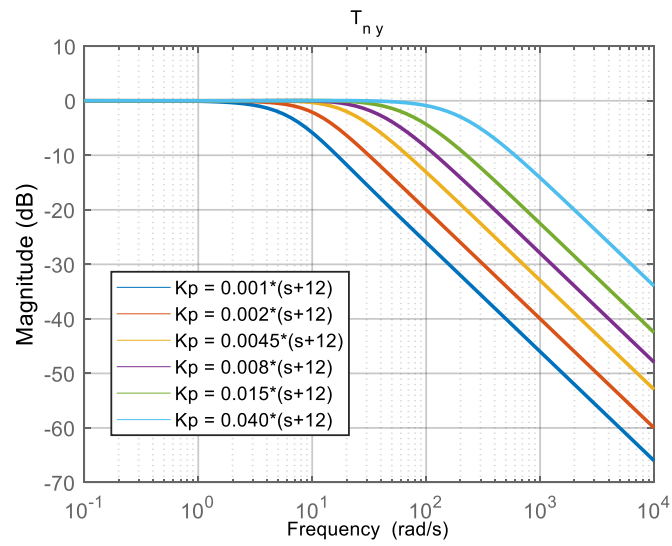


Figure 5.12: Sensor Noise to Output Bode Plot When changing g

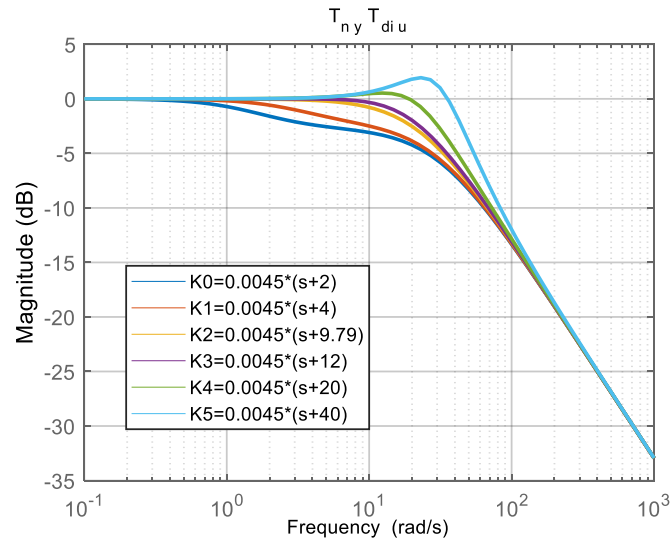


Figure 5.13: Sensor Noise to Output Bode Plot When changing z

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 $T_{d_{ou}}$. The output disturbance to control bode plot is presented in Figure 5.14 and Figure 5.15

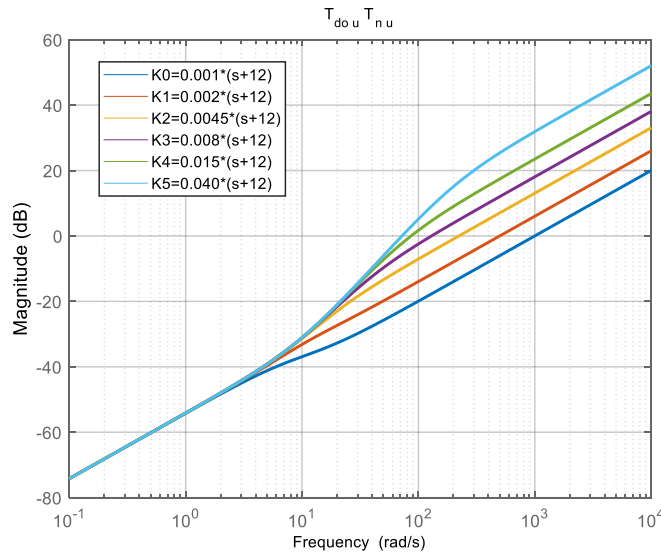


Figure 5.14: Output Disturbance to Control Bode Plot When changing g

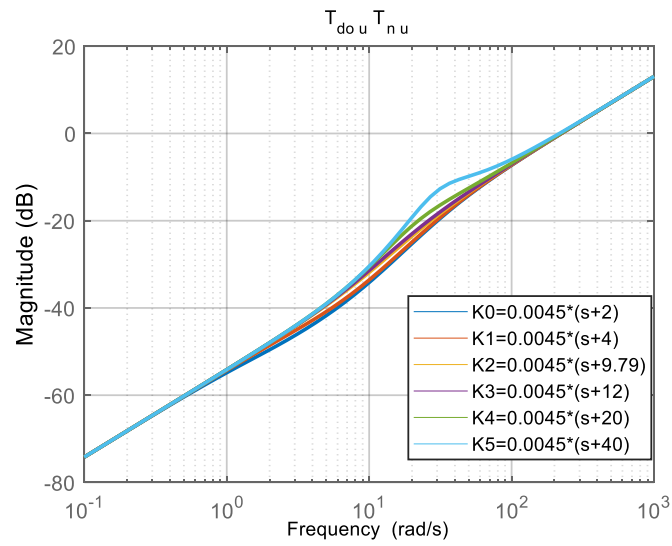


Figure 5.15: Output Disturbance to Control Bode Plot When changing z

Based on the Figures above:

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

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

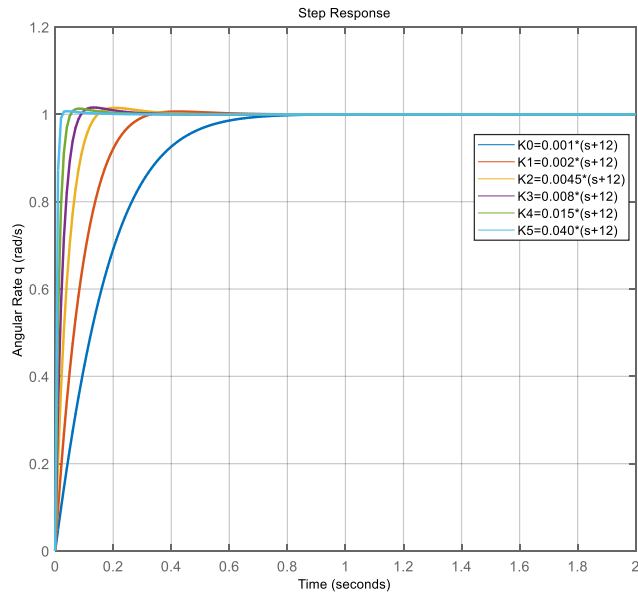


Figure 5.16: Step Response When changing g

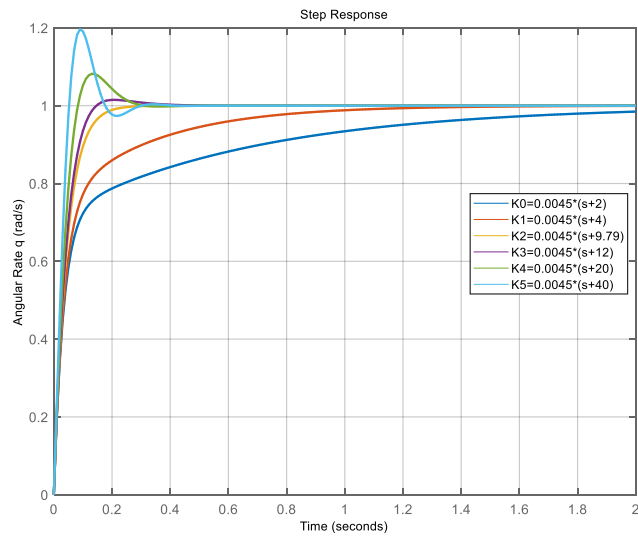


Figure 5.17: Step Response When changing z

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

$$K_{roll\ off1} = \frac{220^2}{(s + 220)^2} \quad (5.5)$$

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 ($\mathbf{J} = \text{diag}[1/J_x, 1/J_y, 1/J_z]^T$, $\nu = [p, q, r]^T$ 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)

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

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

$$\int \mathbf{s}^T \mathbf{Q} \mathbf{s} + \mathbf{u}^T \mathbf{R} \mathbf{u} \quad (5.7)$$

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)

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

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:

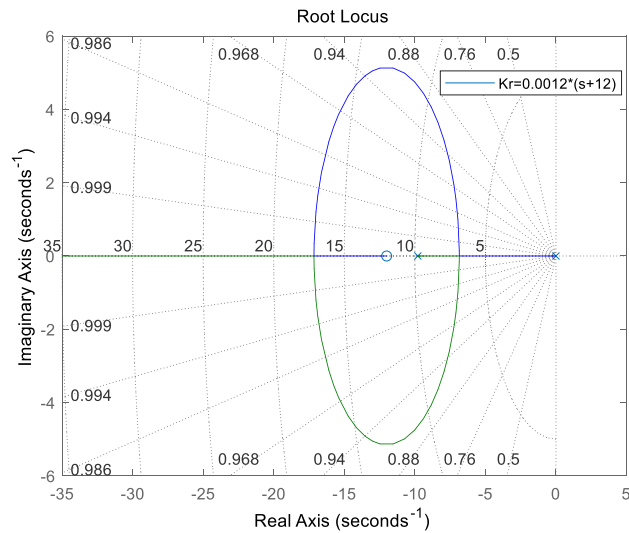


Figure 5.18: Angular Rate Control Open Loop Root Locus

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 describe (ϕ, θ, ψ) attitude control for this quadrotor platform equipped with the *MARK3* flight controller. The attitude of the quadrotor is from attitude state estimation (fusion of gyroscope, accelerometer and MOCAP). The fre-

quency 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.

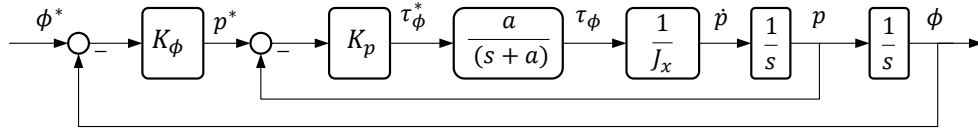


Figure 5.19: Close Loop Diagram of Attitude Control

Based on the close loop structure obtained from the previous section

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

we design a P controller

$$K_\phi = k_p \quad (5.10)$$

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

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

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

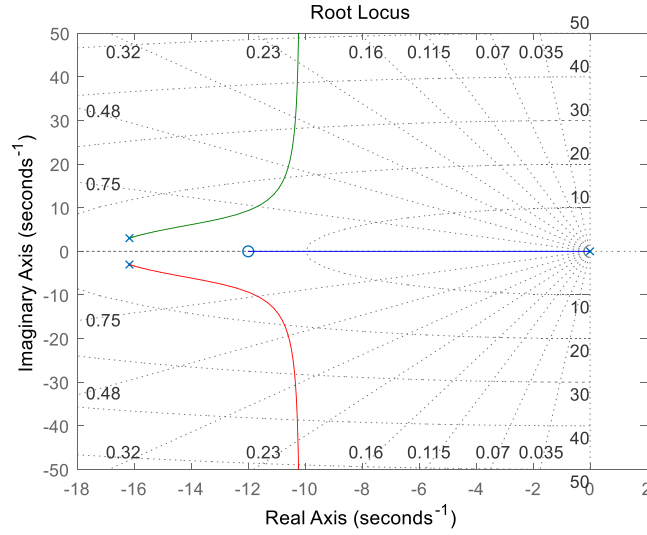


Figure 5.20: Open Loop Root Locus of Attitude Control

The complimentary sensitivity of attitude control is presented in Figure 5.21

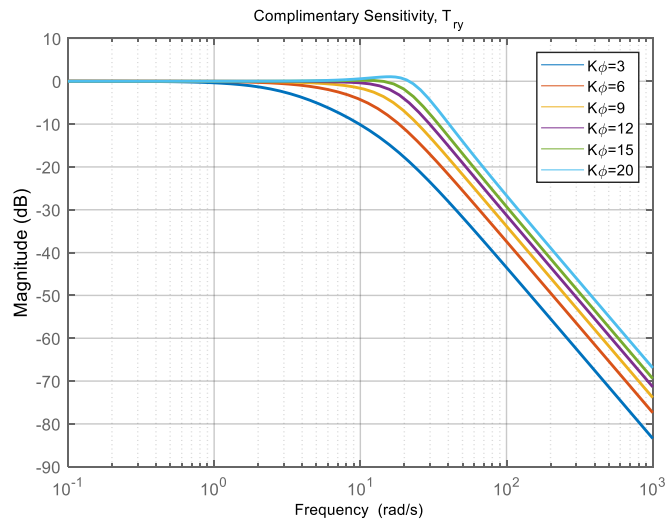


Figure 5.21: Complimentary Sensitivity of Attitude Control when k_p changes

The sensitivity of attitude control is presented in Figure 5.22

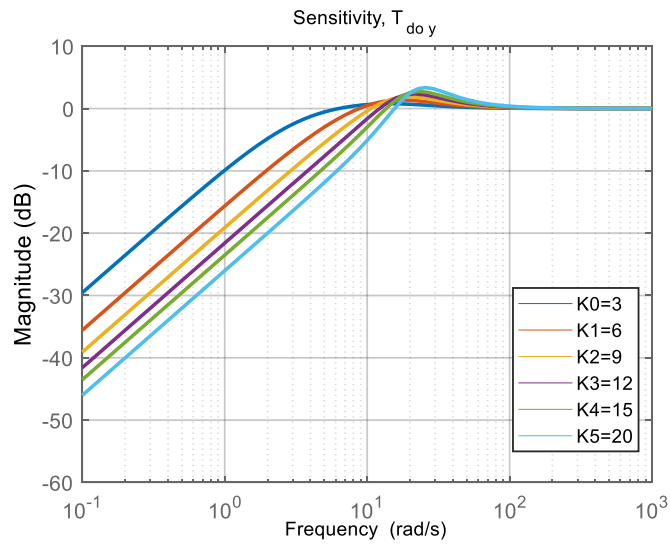


Figure 5.22: Sensitivity of Attitude Control when k_p changes

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

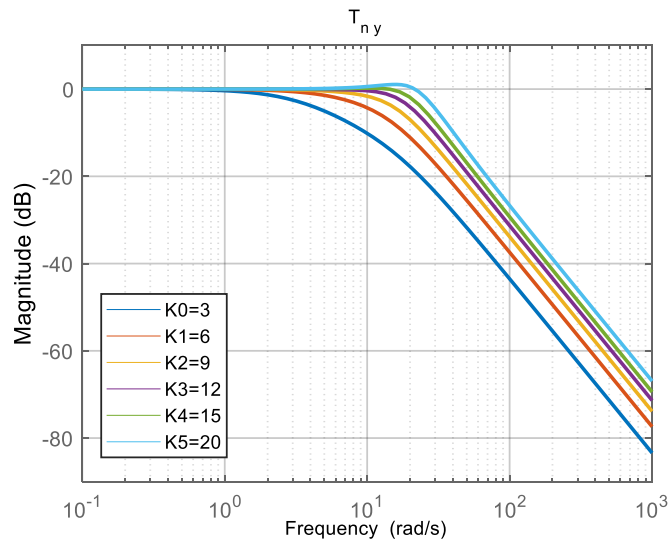


Figure 5.23: Sensor Noise to Output of Attitude Control when k_p changes

The step response of attitude control is presented in Figure 5.24

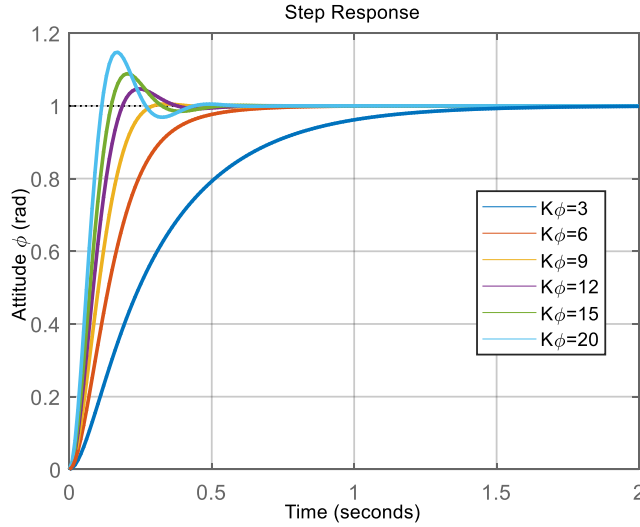


Figure 5.24: Step Response of Attitude Control when k_p changes

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.

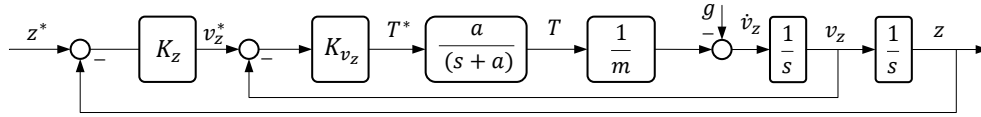


Figure 5.25: Close Loop Diagram of Vertical Movement Control

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

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

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

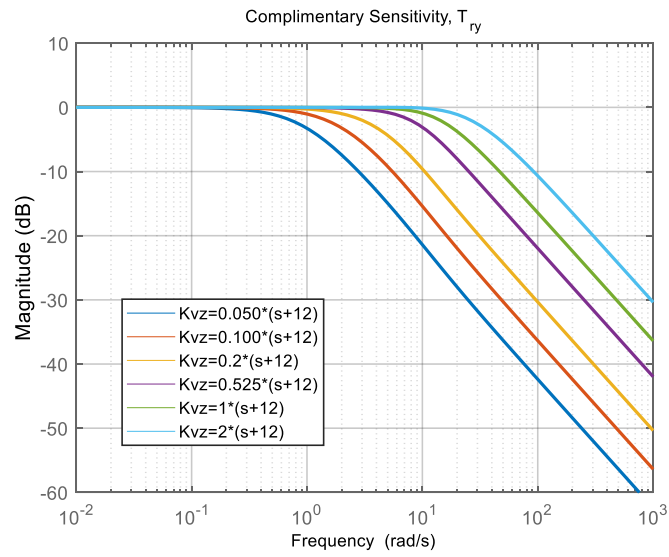


Figure 5.26: Complimentary Sensitivity of Vertical Velocity Control when g changes

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 sensitivity plot presented in Figure 5.27

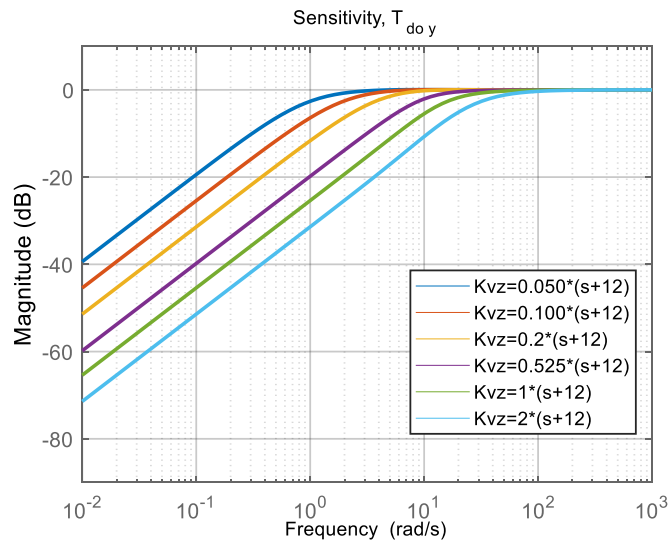


Figure 5.27: Sensitivity of Vertical Velocity Control when g changes

And we have sensor noise to output in Figure 5.28

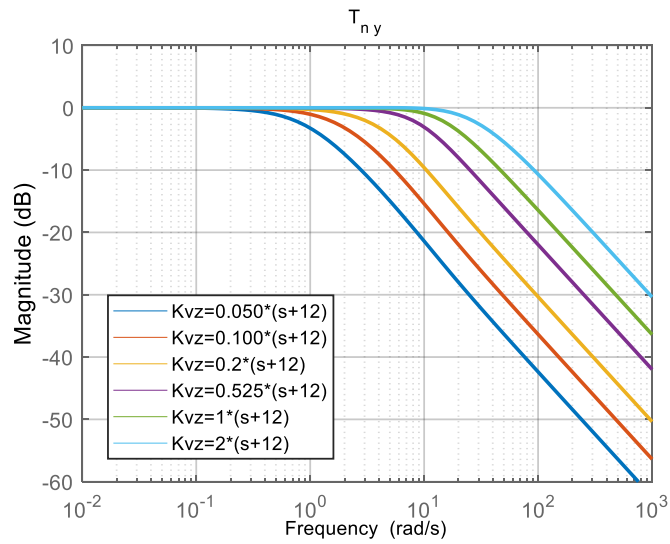


Figure 5.28: Sensor Noise to Output of Vertical Velocity Control when g changes

The step response of attitude control is presented in Figure 5.29

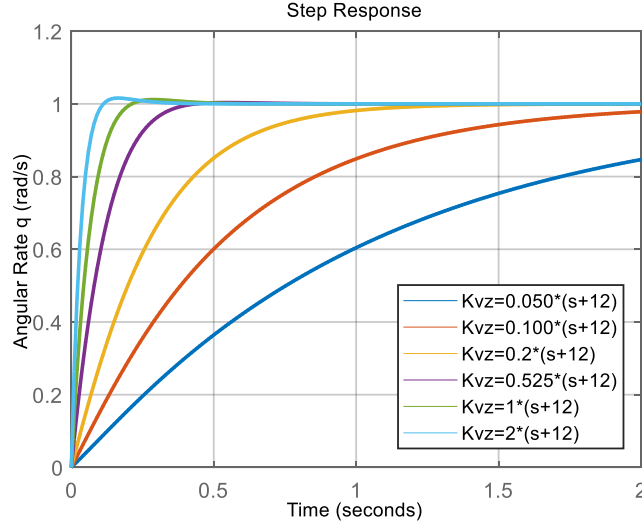


Figure 5.29: Step Response of Vertical Velocity Control when g changes

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

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

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

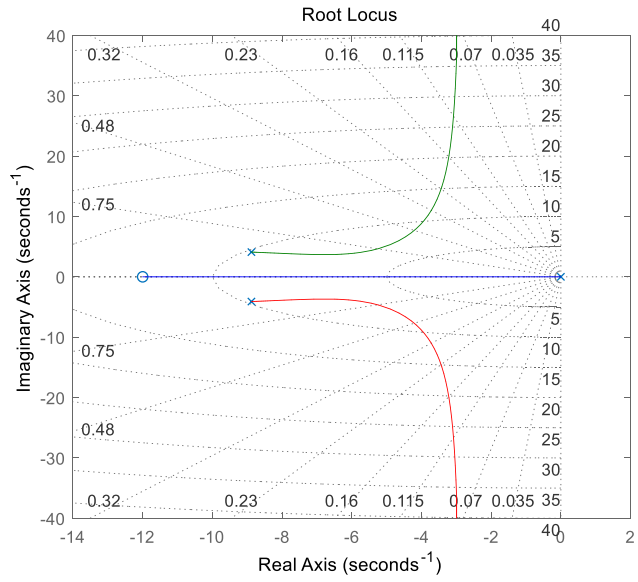


Figure 5.30: Open Loop Root Locus of Altitude Control

The complimentary sensitivity of altitude control is presented in Figure 5.31

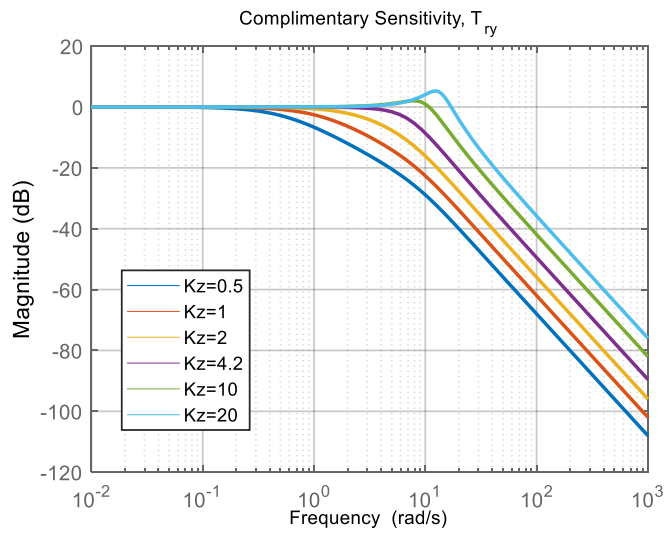


Figure 5.31: Complimentary Sensitivity of Altitude Control when k_p changes

The sensitivity of altitude control is presented in Figure 5.32

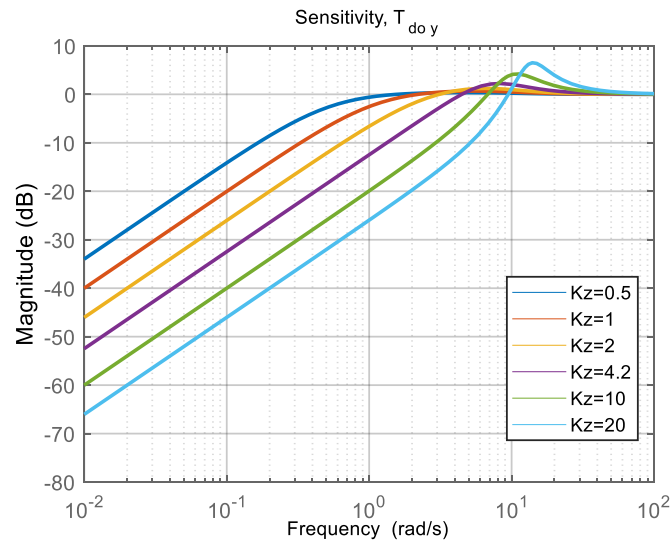


Figure 5.32: Sensitivity of Altitude Control when k_p changes

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

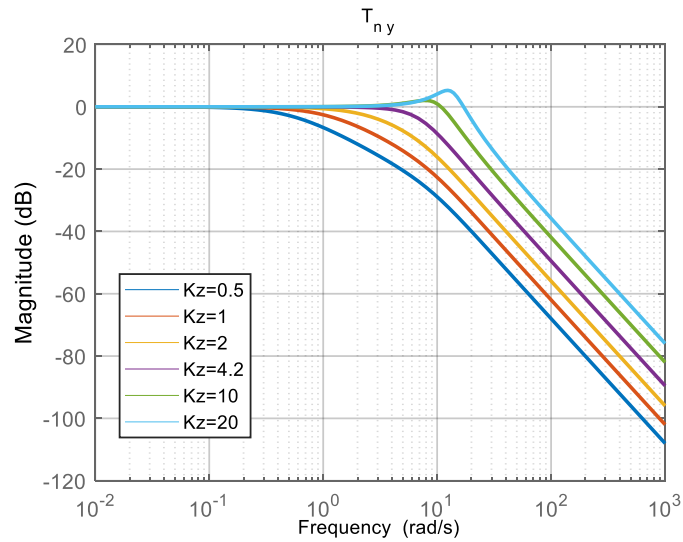


Figure 5.33: Sensor Noise to Output of Altitude Control when k_p changes

The step response of altitude control is presented in Figure 5.34

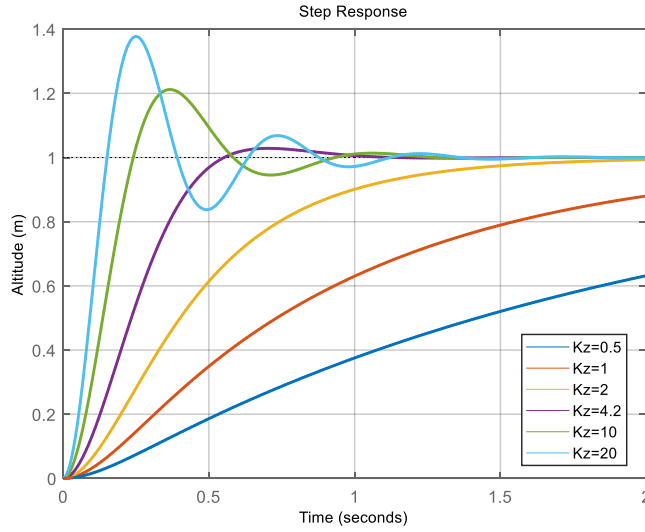


Figure 5.34: Step Response of Altitude Control when k_p changes

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.

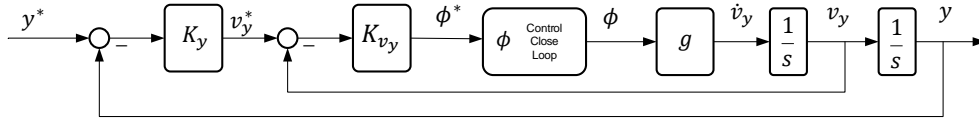


Figure 5.35: Close Loop Diagram of Translational Movement Cascade Control

5.4.1 Translational Velocity Control

In this section, we describe (V_x, V_y) Translational control for this quadrotor platform equipped 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

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

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

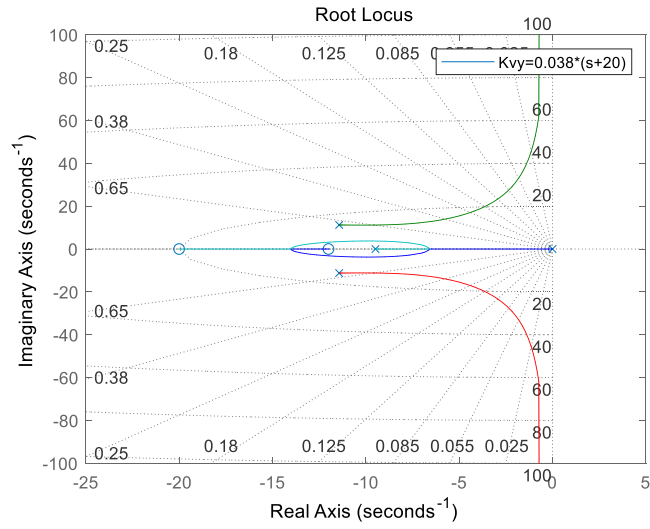


Figure 5.36: Open Loop Root Locus of Translational Velocity Control

The CF of translational velocity control is presented in Figure 5.37

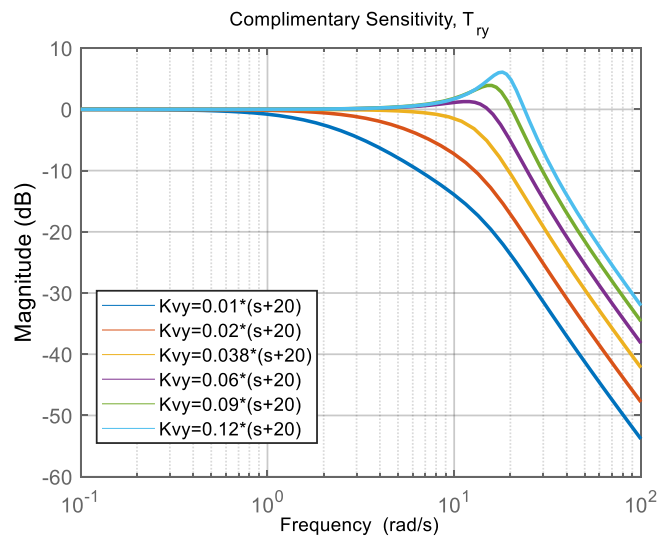


Figure 5.37: Complimentary Sensitivity of Translational Velocity Control when g changes

The sensitivity of translational velocity is presented in Figure 5.38

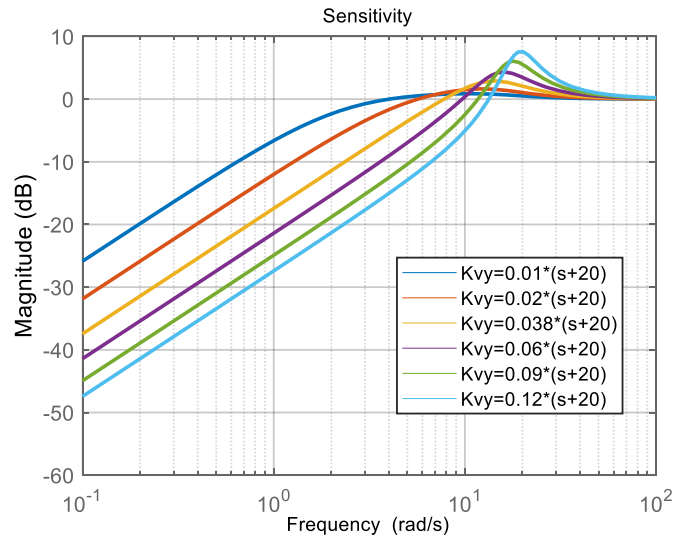


Figure 5.38: Sensitivity of Translational Velocity Control when g changes

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

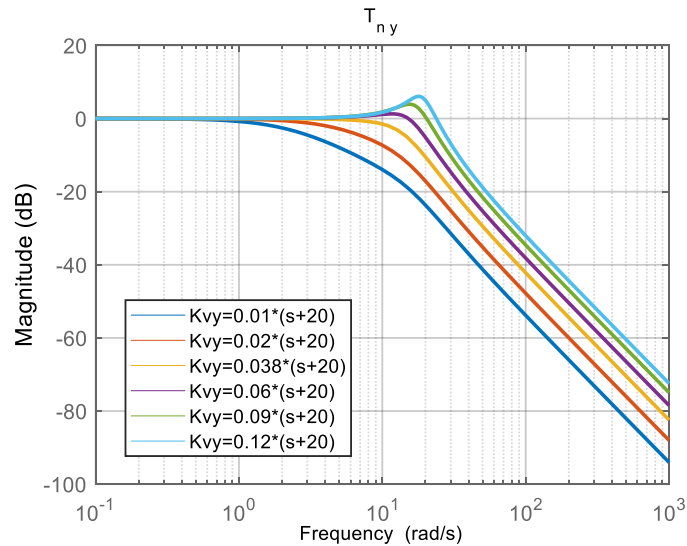


Figure 5.39: Sensor Noise to Output of Translational Velocity Control when g changes

The step response of translational velocity is presented in Figure 5.41

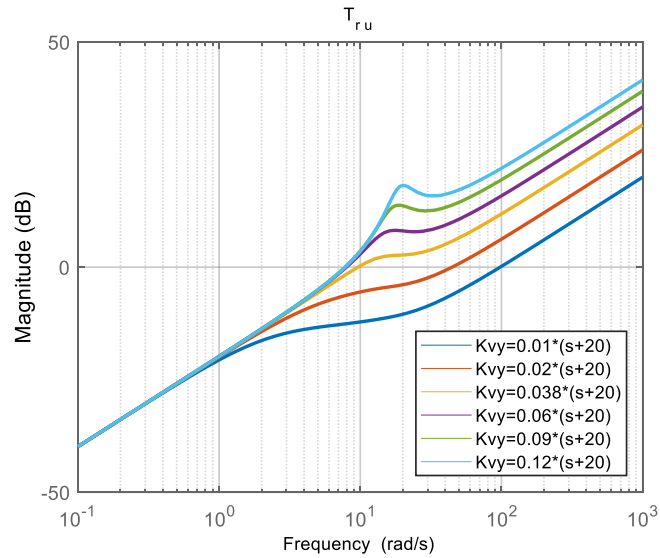


Figure 5.40: Input to Control of Translational Velocity Control when g changes

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

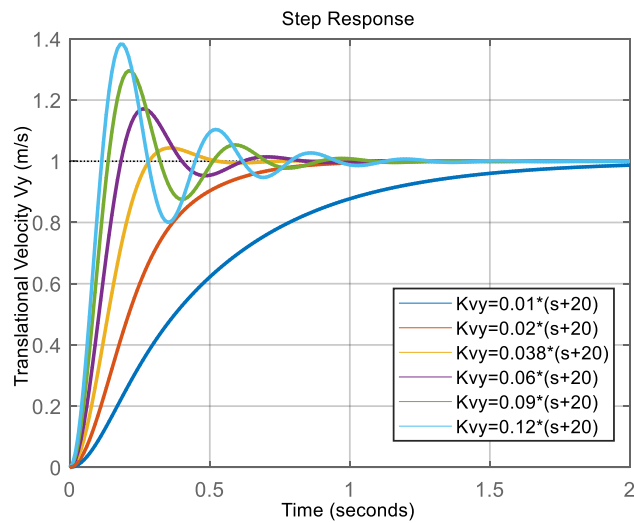


Figure 5.41: Step Response of Translational Velocity Control when g changes

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

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

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 $K_y = 2.0$

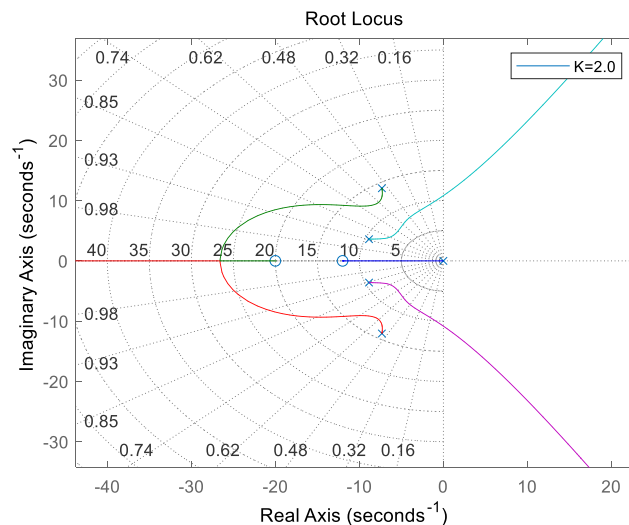


Figure 5.42: Open Loop Root Locus of Translational Position Control

The complimentary sensitivity of position control is presented in Figure 5.43

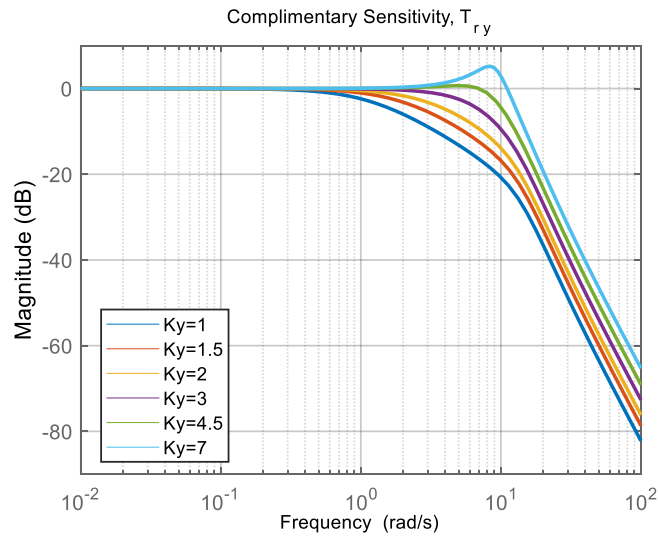


Figure 5.43: CF of Translational Position Control when k_p changes

The sensitivity of position control is presented in Figure 5.43

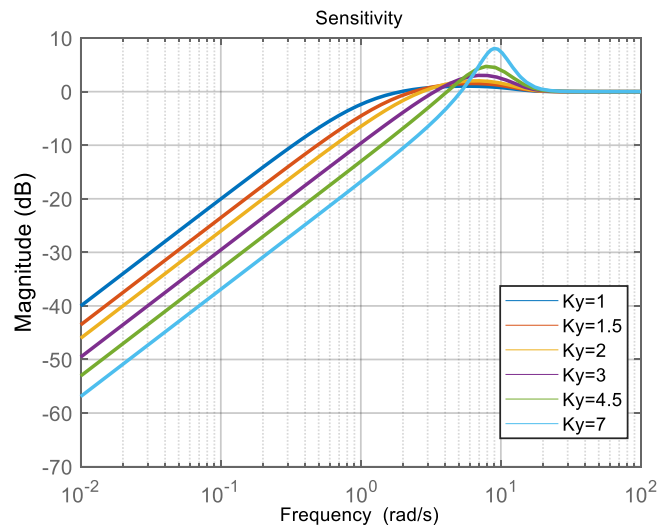


Figure 5.44: Sensitivity of Translational Position Control when k_p changes

The step response of position control is presented in Figure 5.45

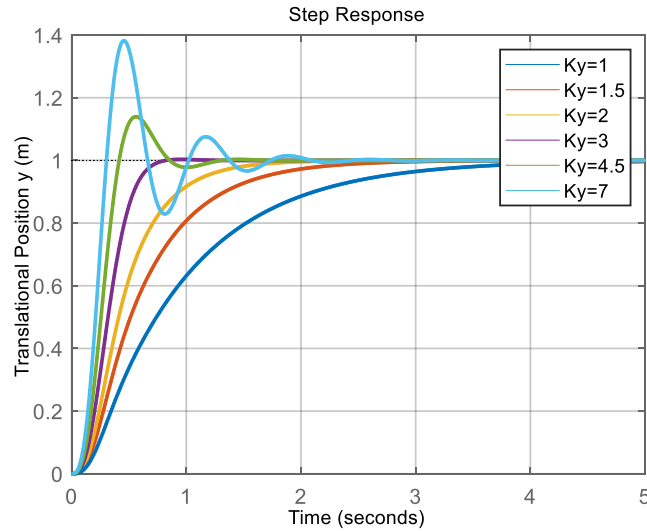


Figure 5.45: Step Response of Translational Position Control when k_p changes

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.

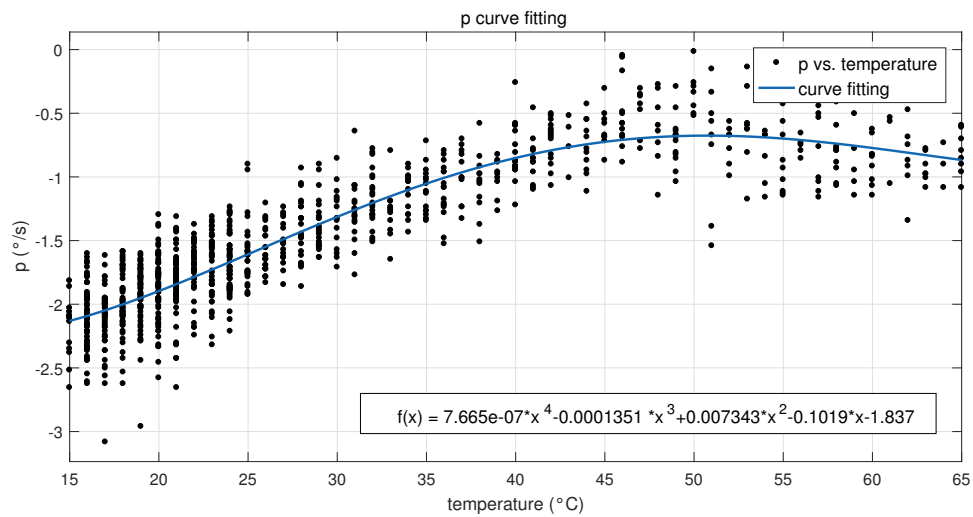


Figure 6.1: gyroscope p temperature calibration

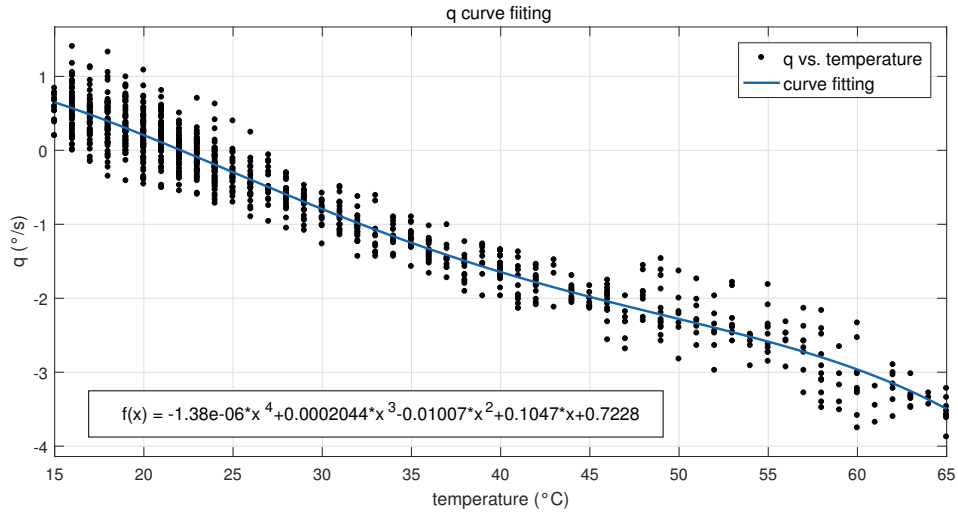


Figure 6.2: gyroscope q temperature calibration

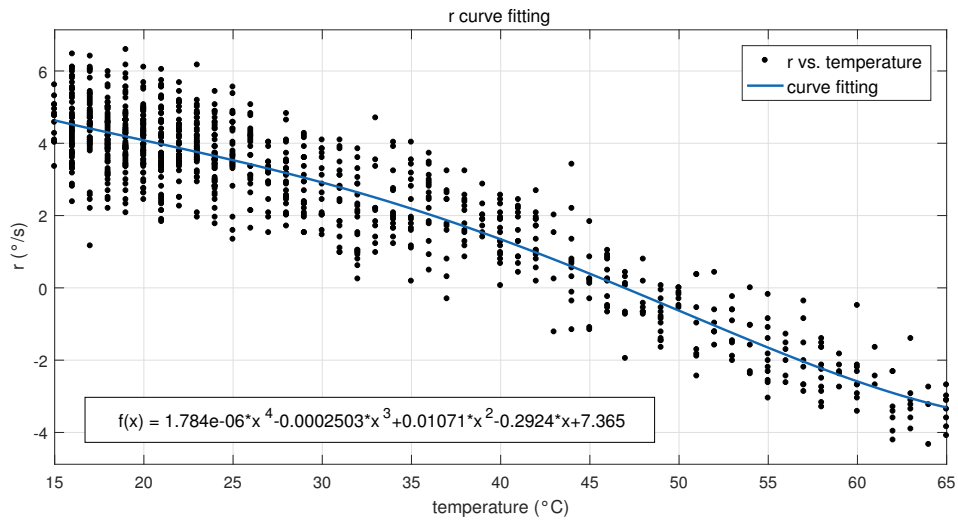


Figure 6.3: gyroscope r temperature calibration

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.

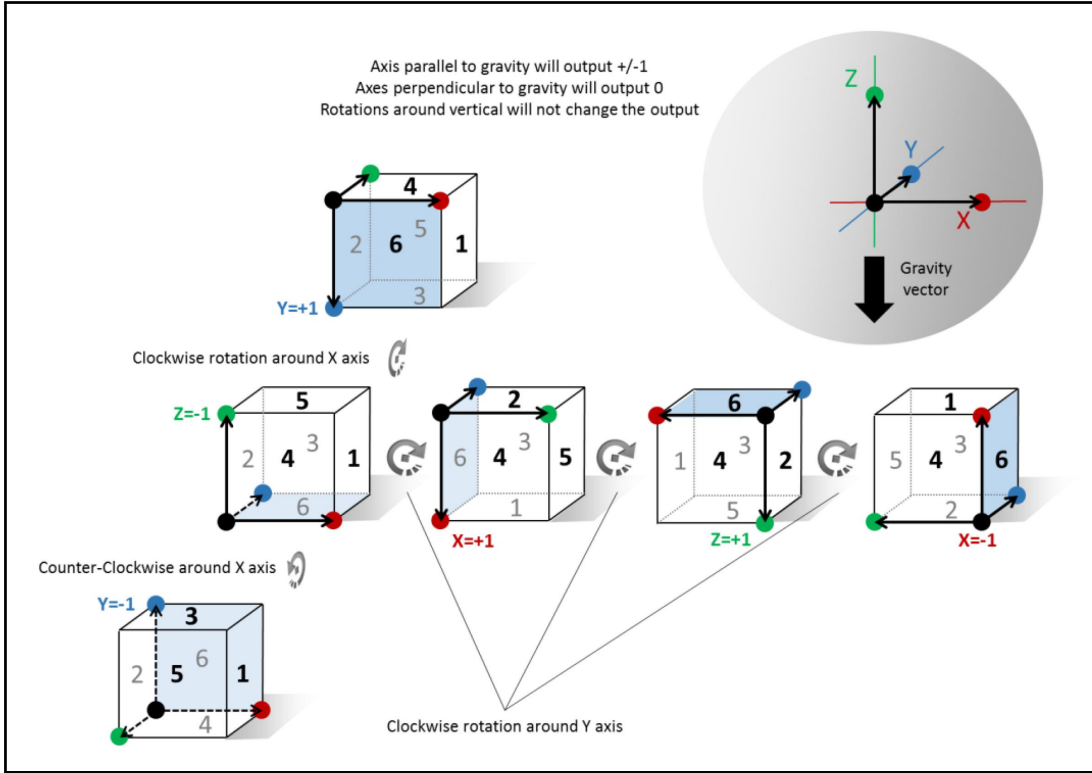


Figure 6.4: Reference for Sensor Orientation While Performing 6-point Tumble Calibration (ST DT0053 Design tip)

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:

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

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

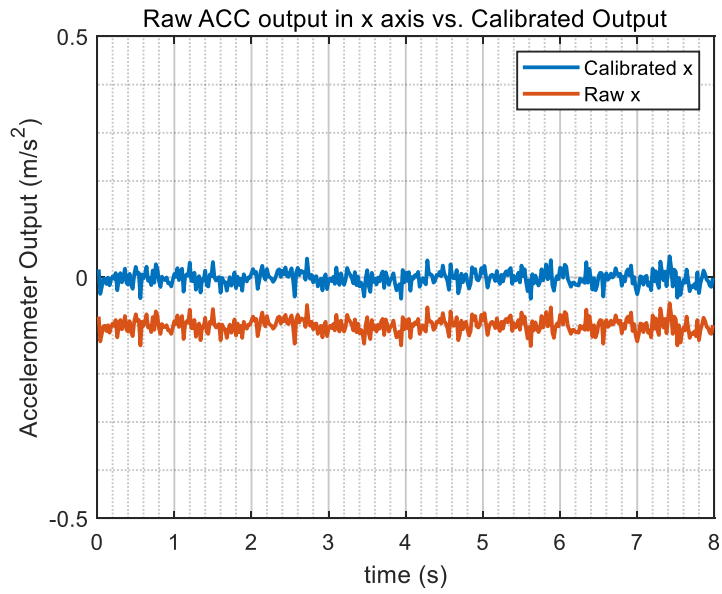


Figure 6.5: Calibrated ACC output vs. Raw output in x axis

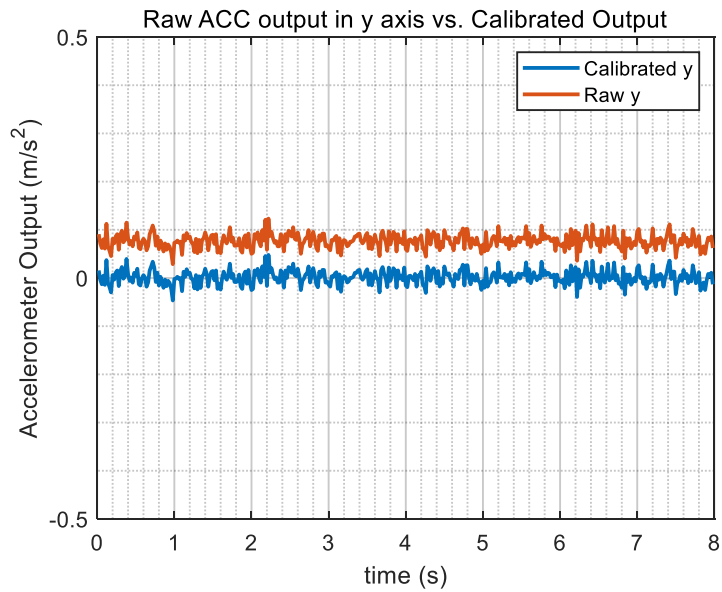


Figure 6.6: Calibrated ACC output vs. Raw output in y axis

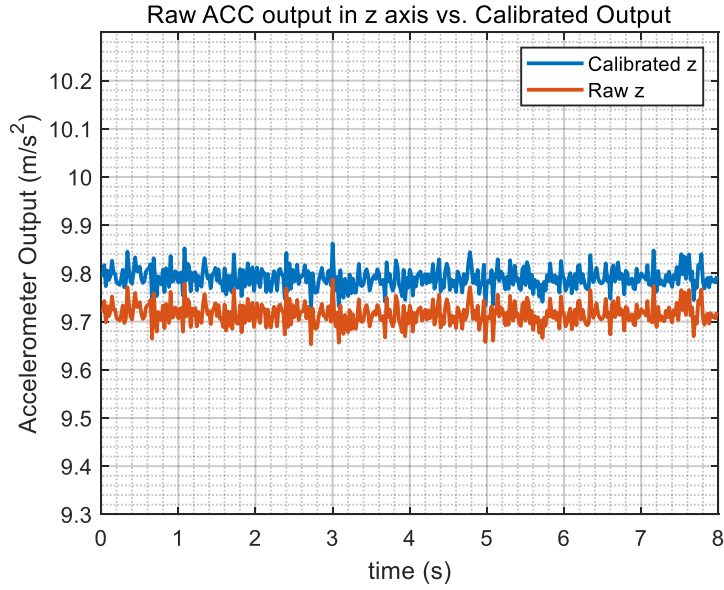


Figure 6.7: Calibrated ACC output vs. Raw output in z axis

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 spectrum.

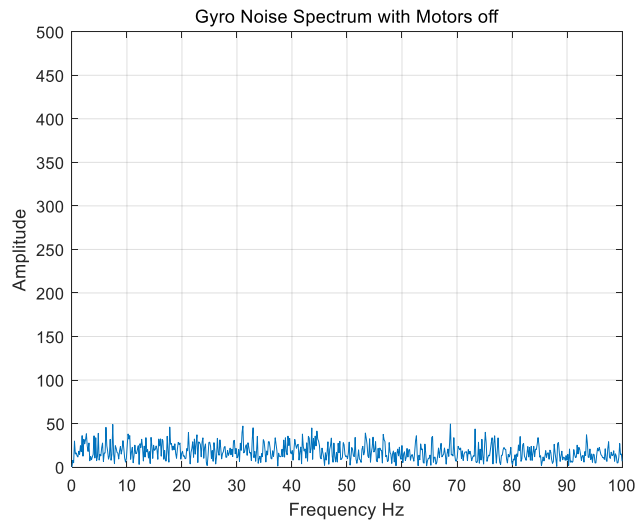


Figure 6.8: Fast Fourier Transform of Gyroscope Output when Motors off

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.

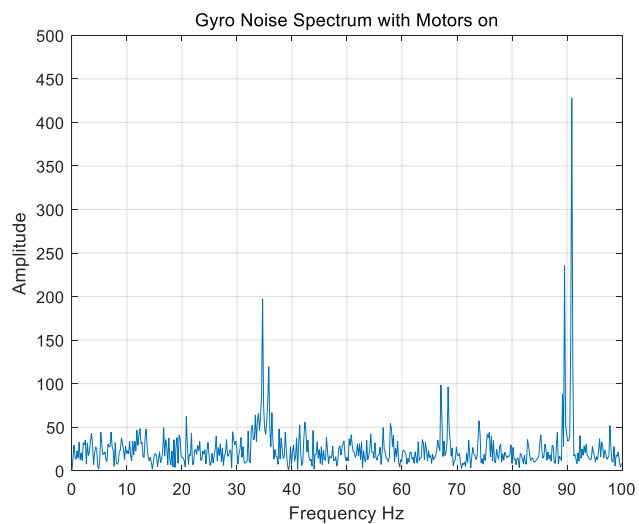


Figure 6.9: Fast Fourier Transform of Gyroscope Output when Motors on

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.

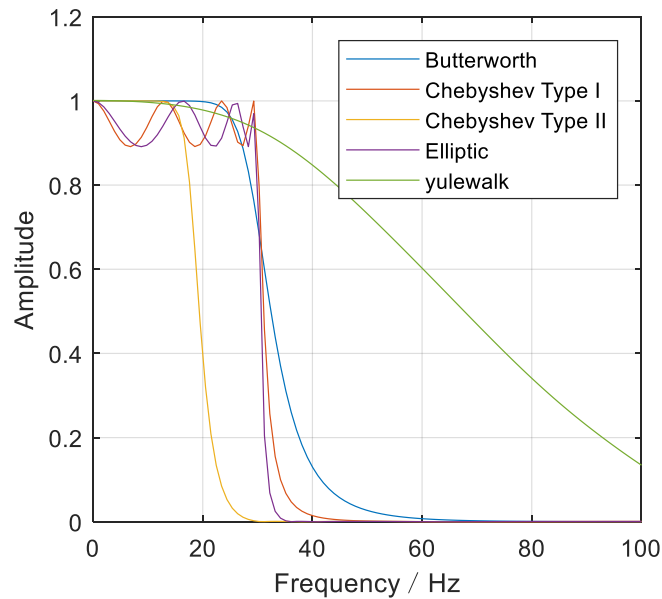


Figure 6.10: Different IIR Filters Spectrum Using MATLAB Toolbox

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

$$H = \frac{35532.45}{s^2 + 377.16s + 35532.45} \quad (6.2)$$

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.

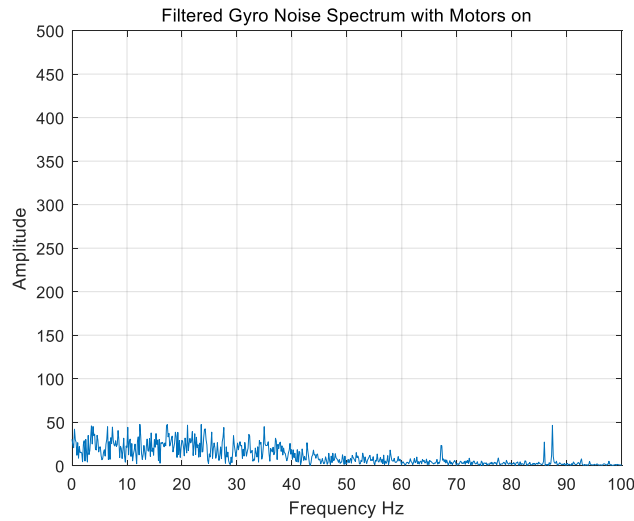


Figure 6.11: Filtered Gyroscope Zero Output Spectrum with Motors on

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]:

$$q = q_0 + q_1i + q_2j + q_3k \quad (6.3)$$

and

$$q = [q_0, q_1, q_2, q_3]^T \quad (6.4)$$

The multiplication of quaternions can be represented as:

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

The rotation matrix from body frame to earth frame can presented using quaternion

$\mathbf{x}_E = R_{B \rightarrow E}(q)\mathbf{x}_B$. And we have

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

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

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

And we have the iteration step

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

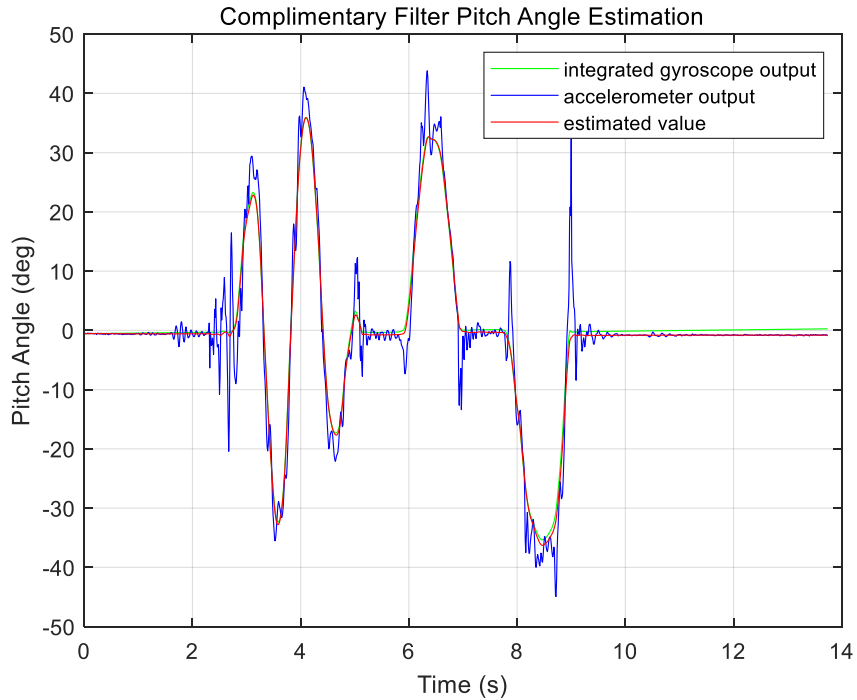


Figure 6.12: Complimentary Filter Pitch Angle Estimation

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.

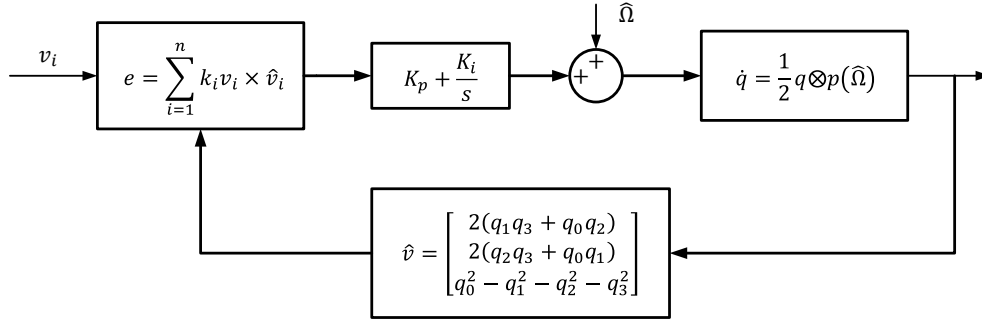


Figure 6.13: Explicit Complimentary Filter Diagram

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

$$v(k+1) = v(k) + a(k)T \quad (6.9)$$

And the position estimation iteration step is based on

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

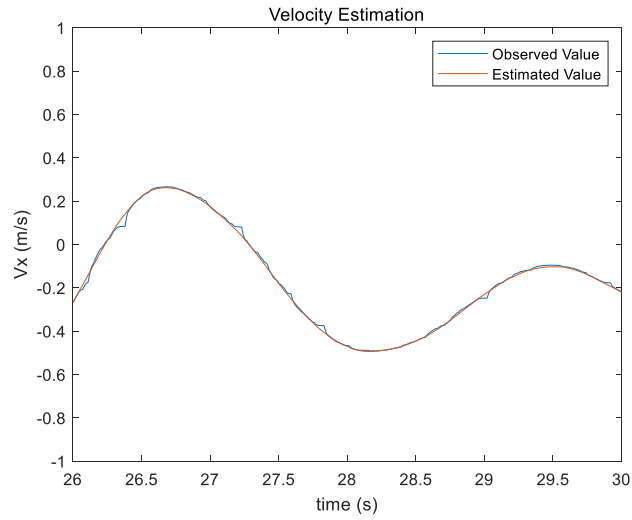


Figure 6.14: Complimentary Filter Velocity in x axis Estimation

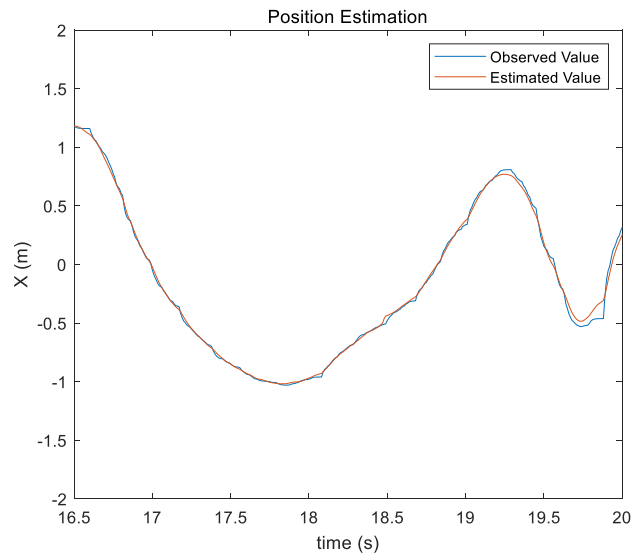


Figure 6.15: Complimentary Filter Position in x axis Estimation

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

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

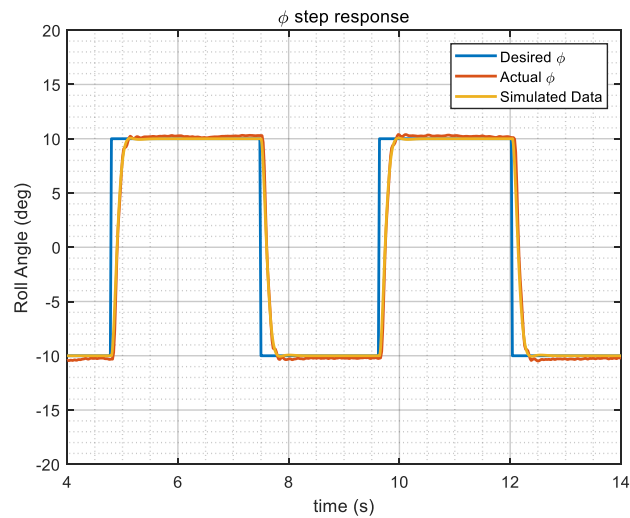


Figure 7.1: ϕ Step Response with Simulated Data

The ramp response of ψ control is presented in Figure 7.2

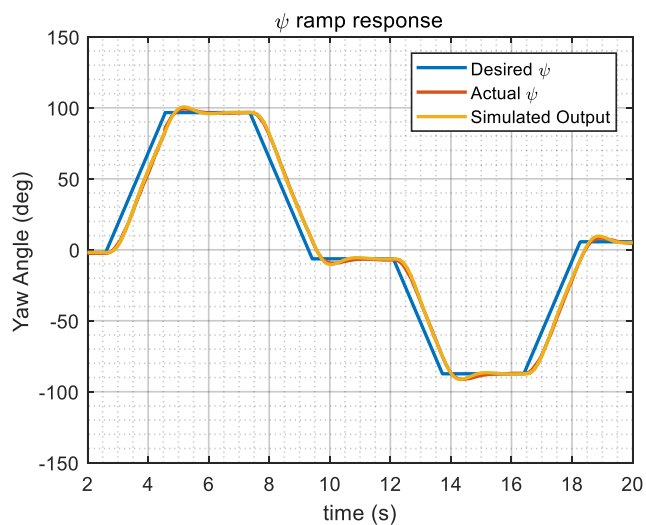


Figure 7.2: ψ Ramp Response with Simulated Data

7.2 Altitude Command-response Graph with Simulation Results

The ramp response of altitude control is presented in Figure 7.3

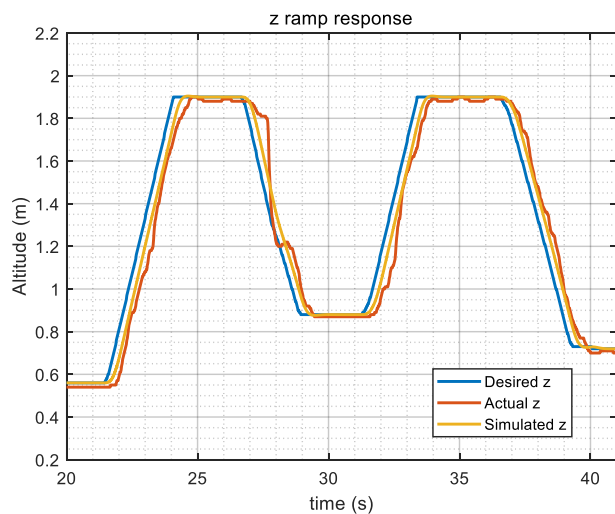


Figure 7.3: Altitude Ramp Response with Simulated Data

7.3 Translational Position Command-response Graph with Simulation Results

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

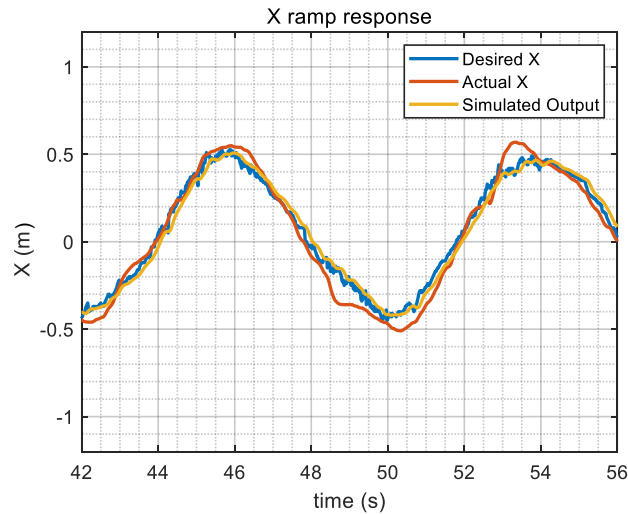


Figure 7.4: Position Ramp Response with Simulated Data

7.4 Summary and Conclusions

This chapter provided comparison 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.

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. **Estimation.** 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 be 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 additional 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, “Termination 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

```
1 close all;
2 clc;
3 clear;
4 %-----Angular Rate Control-----%
5 %% Parameters
6 t=0:0.01:2;
7 J_y = 0.0019536;
8 a = 9.79;
9 s=tf('s');
10 P=1 * a /((J_y*s)*(s+a));
11 %% Controller
12 K0=0.001*(s+12);
13 K1=0.002*(s+12);
14 K2=0.0045*(s+12);
15 K3=0.008*(s+12);
16 K4=0.015*(s+12);
17 K5=0.040*(s+12);
18 %% 1 Plot L = PK
19 L0=P*K0;
20 L1=P*K1;
21 L2=P*K2;
22 L3=P*K3;
23 L4=P*K4;
24 L5=P*K5;
25 figure(1);
26 bodemag(L0);
27 hold on;
28 bodemag(L1);
29 hold on;
30 bodemag(L2);
31 hold on;
32 bodemag(L3);
33 hold on;
34 bodemag(L4);
35 hold on;
36 bodemag(L5);
37 hold off;
38 legend("K_{p}=0.001*(s+12)",...
39        "K_{p}=0.002*(s+12)",...
40        "K_{p}=0.0045*(s+12)",...
41        "K_{p}=0.008*(s+12)",...
42        "K_{p}=0.015*(s+12)",...
43        "K_{p}=0.040*(s+12)");
44 title('Open Loop bodemag L');
45 grid on;
46 grid minor;
47 plot_axis;
48 %% 2 Plot Try (T)
49 T0=minreal(L0/(1+L0));
50 T1=minreal(L1/(1+L1));
51 T2=minreal(L2/(1+L2));
52 T3=minreal(L3/(1+L3));
53 T4=minreal(L4/(1+L4));
54 T5=minreal(L5/(1+L5));
```

```

55 figure(2);
56 bodemag(T0);
57 hold on;
58 bodemag(T1);
59 hold on;
60 bodemag(T2);
61 hold on;
62 bodemag(T3);
63 hold on;
64 bodemag(T4);
65 hold on;
66 bodemag(T5);
67 hold off;
68 legend("K_{p}=0.001*(s+12)",...
69         "K_{p}=0.002*(s+12)",...
70         "K_{p}=0.0045*(s+12)",...
71         "K_{p}=0.008*(s+12)",...
72         "K_{p}=0.015*(s+12)",...
73         "K_{p}=0.040*(s+12)");
74 title('Complimentary Sensitivity, T_{ry}');
75 grid on;
76 grid minor;
77 plot_axis;
78 %% 3 Plot Tdoy (S)
79 S0=1/(1+L0);
80 S1=1/(1+L1);
81 S2=1/(1+L2);
82 S3=1/(1+L3);
83 S4=1/(1+L4);
84 S5=1/(1+L5);
85 figure(3);
86 bodemag(S0);
87 hold on;
88 bodemag(S1);
89 hold on;
90 bodemag(S2);
91 hold on;
92 bodemag(S3);
93 hold on;
94 bodemag(S4);
95 hold on;
96 bodemag(S5);
97 hold off;
98 legend("K_{p}=0.001*(s+12)",...
99         "K_{p}=0.002*(s+12)",...
100        "K_{p}=0.0045*(s+12)",...
101        "K_{p}=0.008*(s+12)",...
102        "K_{p}=0.015*(s+12)",...
103        "K_{p}=0.040*(s+12)");
104 title('Sensitivity');
105 grid on;
106 grid minor;
107 plot_axis;
108 %% 4 Plot Tdiy (PS)
109 PS0=P/(1+L0);
110 PS1=P/(1+L1);
111 PS2=P/(1+L2);

```

```

112 PS3=P/(1+L3);
113 PS4=P/(1+L4);
114 PS5=P/(1+L5);
115 figure(4);
116 bodemag(PS0);
117 hold on;
118 bodemag(PS1);
119 hold on;
120 bodemag(PS2);
121 hold on;
122 bodemag(PS3);
123 hold on;
124 bodemag(PS4);
125 hold on;
126 bodemag(PS5);
127 hold off;
128 legend("K_{p}=0.001*(s+12)",...
129         "K_{p}=0.002*(s+12)",...
130         "K_{p}=0.0045*(s+12)",...
131         "K_{p}=0.008*(s+12)",...
132         "K_{p}=0.015*(s+12)",...
133         "K_{p}=0.040*(s+12)");
134 title('PS');
135 grid on;
136 grid minor;
137 plot_axis;
138 %% 5 Plot Tny Tdiu
139 Tny0= -L0/(1+L0);
140 Tny1= -L1/(1+L1);
141 Tny2= -L2/(1+L2);
142 Tny3= -L3/(1+L3);
143 Tny4= -L4/(1+L4);
144 Tny5= -L5/(1+L5);
145 figure(5);
146 bodemag(Tny0);
147 hold on;
148 bodemag(Tny1);
149 hold on;
150 bodemag(Tny2);
151 hold on;
152 bodemag(Tny3);
153 hold on;
154 bodemag(Tny4);
155 hold on;
156 bodemag(Tny5);
157 hold off;
158 legend("K_{p}=0.001*(s+12)",...
159         "K_{p}=0.002*(s+12)",...
160         "K_{p}=0.0045*(s+12)",...
161         "K_{p}=0.008*(s+12)",...
162         "K_{p}=0.015*(s+12)",...
163         "K_{p}=0.040*(s+12)");
164 title('T_{n y}');
165 grid on;
166 grid minor;
167 plot_axis;
168 %% 6 Plot Tru (KS)

```

```

169 Tru0= K0/(1+L0);
170 Tru1= K1/(1+L1);
171 Tru2= K2/(1+L2);
172 Tru3= K3/(1+L3);
173 Tru4= K4/(1+L4);
174 Tru5= K5/(1+L5);
175
176 figure(6);
177 bodemag(Tru0);
178 hold on;
179 bodemag(Tru1);
180 hold on;
181 bodemag(Tru2);
182 hold on;
183 bodemag(Tru3);
184 hold on;
185 bodemag(Tru4);
186 hold on;
187 bodemag(Tru5);
188 hold off;
189 legend("K_{p}=0.001*(s+12)",...
190        "K_{p}=0.002*(s+12)",...
191        "K_{p}=0.0045*(s+12)",...
192        "K_{p}=0.008*(s+12)",...
193        "K_{p}=0.015*(s+12)",...
194        "K_{p}=0.040*(s+12)");
195 title('T_{r u} KS');
196 grid on;
197 grid minor;
198 plot_axis;
199 %% 7 Plot Tdou Tnu
200 Tdou0= -K0/(1+L0);
201 Tdou1= -K1/(1+L1);
202 Tdou2= -K2/(1+L2);
203 Tdou3= -K3/(1+L3);
204 Tdou4= -K4/(1+L4);
205 Tdou5= -K5/(1+L5);
206 figure(7);
207 bodemag(Tdou0);
208 hold on;
209 bodemag(Tdou1);
210 hold on;
211 bodemag(Tdou2);
212 hold on;
213 bodemag(Tdou3);
214 hold on;
215 bodemag(Tdou4);
216 hold on;
217 bodemag(Tdou5);
218 hold off;
219 legend("K_{p}=0.001*(s+12)",...
220        "K_{p}=0.002*(s+12)",...
221        "K_{p}=0.0045*(s+12)",...
222        "K_{p}=0.008*(s+12)",...
223        "K_{p}=0.015*(s+12)",...
224        "K_{p}=0.040*(s+12)");
225 title('T_{do u} T_{n u}');

```

```

226 grid on;
227 grid minor;
228 plot_axis;
229 %% Plot Step Response
230 figure(8);
231 st0=stepplot(T0,t);
232 hold on;
233 st1=stepplot(T1,t);
234 hold on;
235 st2=stepplot(T2,t);
236 hold on;
237 st3=stepplot(T3,t);
238 hold on;
239 st4=stepplot(T4,t);
240 hold on;
241 st5=stepplot(T5,t);
242 hold off;
243 legend("K_{p}=0.001*(s+12)",...
244        "K_{p}=0.002*(s+12)",...
245        "K_{p}=0.0045*(s+12)",...
246        "K_{p}=0.008*(s+12)",...
247        "K_{p}=0.015*(s+12)",...
248        "K_{p}=0.040*(s+12)");
249 ylabel('Angular Rate q (rad/s)');
250 grid on;
251 grid minor;
252 plot_axis;
253 %% Open Loop Root Locus
254 figure(10);
255 rlocus(T0);
256 grid on;
257 grid minor;
258 legend("K0=0.001*(s+12)");
259 figure(11);
260 rlocus(L1);
261 grid on;
262 grid minor;
263 legend("K1=0.002*(s+12)");
264 figure(12);
265 rlocus(T2);
266 grid on;
267 grid minor;
268 legend("Kp=0.0045*(s+12)");
269 figure(13);
270 rlocus(L3);
271 grid on;
272 grid minor;
273 legend("K3=0.008*(s+12)");
274 figure(14);
275 rlocus(L4);
276 grid on;
277 grid minor;
278 legend("K4=0.015*(s+12)");
279 figure(15);
280 rlocus(L5);
281 grid on;
282 grid minor;

```

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

Attitude Control Simulation File

```
1 close all;
2 clc;
3 clear;
4 %-----Attitude Control-----%
5 %% Parameters
6 t=0:0.01:2;
7 J_y = 0.0019536;
8 a = 9.79;
9 s=tf('s');
10 P2=1 * a /((J_y*s)*(s+a));
11 K2=0.0045*(s+12);
12 P = minreal(P2*K2/(1+P2*K2)/s);
13 %% Controller
14 K0=3;
15 K1=6;
16 K2=9;
17 K3=12;
18 K4=15;
19 K5=20;
20 %% 1 Plot L = PK
21 L0=P*K0;
22 L1=P*K1;
23 L2=P*K2;
24 L3=P*K3;
25 L4=P*K4;
26 L5=P*K5;
27 figure(1);
28 bodemag(L0);
29 hold on;
30 bodemag(L1);
31 hold on;
32 bodemag(L2);
33 hold on;
34 bodemag(L3);
35 hold on;
36 bodemag(L4);
37 hold on;
38 bodemag(L5);
39 hold off;
40 legend("K\phi=3",...
41        "K\phi=6",...
42        "K\phi=9",...
43        "K\phi=12",...
44        "K\phi=15",...
45        "K\phi=20");
46 title('Open Loop bodemag L');
47 grid on;
48 grid minor;
49 plot_axis;
50 %% 2 Plot Try (T)
51 T0=minreal(L0/(1+L0));
52 T1=minreal(L1/(1+L1));
```

```

53 T2=minreal(L2/(1+L2));
54 T3=minreal(L3/(1+L3));
55 T4=minreal(L4/(1+L4));
56 T5=minreal(L5/(1+L5));
57 figure(2);
58 bodemag(T0);
59 hold on;
60 bodemag(T1);
61 hold on;
62 bodemag(T2);
63 hold on;
64 bodemag(T3);
65 hold on;
66 bodemag(T4);
67 hold on;
68 bodemag(T5);
69 hold off;
70 legend("K\phi=3",...
71        "K\phi=6",...
72        "K\phi=9",...
73        "K\phi=12",...
74        "K\phi=15",...
75        "K\phi=20");
76 title('Complimentary Sensitivity, T-ry');
77 grid on;
78 grid minor;
79 plot_axis;
80 %% 3 Plot Tdoy (S)
81 S0=1/(1+L0);
82 S1=1/(1+L1);
83 S2=1/(1+L2);
84 S3=1/(1+L3);
85 S4=1/(1+L4);
86 S5=1/(1+L5);
87 figure(3);
88 bodemag(S0);
89 hold on;
90 bodemag(S1);
91 hold on;
92 bodemag(S2);
93 hold on;
94 bodemag(S3);
95 hold on;
96 bodemag(S4);
97 hold on;
98 bodemag(S5);
99 hold off;
100 legend("K\phi=3",...
101        "K\phi=6",...
102        "K\phi=9",...
103        "K\phi=12",...
104        "K\phi=15",...
105        "K\phi=20");
106 title('Sensitivity, T-do y');
107 grid on;
108 grid minor;
109 plot_axis;

```

```

110 %% 4 Plot Tdiy (PS)
111 PS0=P/(1+L0);
112 PS1=P/(1+L1);
113 PS2=P/(1+L2);
114 PS3=P/(1+L3);
115 PS4=P/(1+L4);
116 PS5=P/(1+L5);
117 figure(4);
118 bodemag(PS0);
119 hold on;
120 bodemag(PS1);
121 hold on;
122 bodemag(PS2);
123 hold on;
124 bodemag(PS3);
125 hold on;
126 bodemag(PS4);
127 hold on;
128 bodemag(PS5);
129 hold off;
130 legend("K\phi=3",...
131         "K\phi=6",...
132         "K\phi=9",...
133         "K\phi=12",...
134         "K\phi=15",...
135         "K\phi=20");
136 title('PS, T- $\{di y\}$ ');
137 grid on;
138 grid minor;
139 plot_axis;
140 %% 5 Plot Tny
141 Tny0= -L0/(1+L0);
142 Tny1= -L1/(1+L1);
143 Tny2= -L2/(1+L2);
144 Tny3= -L3/(1+L3);
145 Tny4= -L4/(1+L4);
146 Tny5= -L5/(1+L5);
147 figure(5);
148 bodemag(Tny0);
149 hold on;
150 bodemag(Tny1);
151 hold on;
152 bodemag(Tny2);
153 hold on;
154 bodemag(Tny3);
155 hold on;
156 bodemag(Tny4);
157 hold on;
158 bodemag(Tny5);
159 hold off;
160 legend("K\phi=3",...
161         "K\phi=6",...
162         "K\phi=9",...
163         "K\phi=12",...
164         "K\phi=15",...
165         "K\phi=20");
166 title('T- $\{n y\}$ ');

```



```

167 grid on;
168 grid minor;
169 plot_axis;
170 %% 6 Plot Tru (KS)
171 Tru0= K0/(1+L0);
172 Tru1= K1/(1+L1);
173 Tru2= K2/(1+L2);
174 Tru3= K3/(1+L3);
175 Tru4= K4/(1+L4);
176 Tru5= K5/(1+L5);
177 figure(6);
178 bodemag(Tru0);
179 hold on;
180 bodemag(Tru1);
181 hold on;
182 bodemag(Tru2);
183 hold on;
184 bodemag(Tru3);
185 hold on;
186 bodemag(Tru4);
187 hold on;
188 bodemag(Tru5);
189 hold off;
190 legend("K\phi=3",...
191         "K\phi=6",...
192         "K\phi=9",...
193         "K\phi=12",...
194         "K\phi=15",...
195         "K\phi=20");
196 title('T_{r u}');
197 grid on;
198 grid minor;
199 plot_axis;
200 %% 7 Plot Tdou Tnu
201 Tdou0= -K0/(1+L0);
202 Tdou1= -K1/(1+L1);
203 Tdou2= -K2/(1+L2);
204 Tdou3= -K3/(1+L3);
205 Tdou4= -K4/(1+L4);
206 Tdou5= -K5/(1+L5);
207 figure(7);
208 bodemag(Tdou0);
209 hold on;
210 bodemag(Tdou1);
211 hold on;
212 bodemag(Tdou2);
213 hold on;
214 bodemag(Tdou3);
215 hold on;
216 bodemag(Tdou4);
217 hold on;
218 bodemag(Tdou5);
219 hold off;
220 legend("K\phi=3",...
221         "K\phi=6",...
222         "K\phi=9",...
223         "K\phi=12",...

```

```

224         "K\phi=15",...
225         "K\phi=20");
226 title('T_{do u} T_{n u}');
227 grid on;
228 grid minor;
229 plot_axis;
230 %% Plot Step Response
231 figure(8);
232 st0=stepplot(T0,t);
233 hold on;
234 st1=stepplot(T1,t);
235 hold on;
236 st2=stepplot(T2,t);
237 hold on;
238 st3=stepplot(T3,t);
239 hold on;
240 st4=stepplot(T4,t);
241 hold on;
242 st5=stepplot(T5,t);
243 hold off;
244 legend("K\phi=3",...
245        "K\phi=6",...
246        "K\phi=9",...
247        "K\phi=12",...
248        "K\phi=15",...
249        "K\phi=20");
250 ylabel('Attitude \phi (rad)');
251 grid on;
252 grid minor;
253 plot_axis;
254 %% Close Loop Pole
255 figure(10);
256 rlocus(L0);
257 grid on;
258 grid minor;
259 legend("K\phi=3");
260 figure(11);
261 rlocus(L1);
262 grid on;
263 grid minor;
264 legend("K\phi=6");
265 figure(12);
266 rlocus(T2);
267 grid on;
268 grid minor;
269 legend("K\phi=9");
270 figure(13);
271 rlocus(L3);
272 grid on;
273 grid minor;
274 legend("K\phi=12");
275 figure(14);
276 rlocus(L4);
277 grid on;
278 grid minor;
279 legend("K\phi=15");
280 figure(15);

```

```

281 rlocus(L5);
282 grid on;
283 grid minor;
284 legend("K\phi=20");

```

Vertical Velocity Control Simulation File

```

1 close all;
2 clc;
3 clear;
4 %-----Vertical Velocity Control-----%
5 %% Parameters
6 t=0:0.01:2;
7 m = 0.645;
8 a = 9.79;
9 s=tf('s');
10 P=1 * a / ((m*s)*(s+a));
11 %% Controller
12 K0=0.050*(s+12);
13 K1=0.100*(s+12);
14 K2=0.2*(s+12);
15 K3=0.525*(s+12);
16 K4=1*(s+12);
17 K5=2*(s+12);
18 %% 1 Plot L = PK
19 L0=P*K0;
20 L1=P*K1;
21 L2=P*K2;
22 L3=P*K3;
23 L4=P*K4;
24 L5=P*K5;
25 figure(1);
26 bodemag(L0);
27 hold on;
28 bodemag(L1);
29 hold on;
30 bodemag(L2);
31 hold on;
32 bodemag(L3);
33 hold on;
34 bodemag(L4);
35 hold on;
36 bodemag(L5);
37 hold off;
38 legend("K0=7.000 + 0.050*s",...
39        "K1=7.000 + 0.200*s",...
40        "K2=7.000 + 0.525*s",...
41        "K3=7.000 + 0.800*s",...
42        "K4=7.000 + 1.000*s",...
43        "K5=7.000 + 2.000*s");
44 title('Open Loop bodemag L');
45 grid on;
46 grid minor;
47 plot_axis;
48 %% 2 Plot Try (T)
49 T0=L0/(1+L0);

```

```

50 T1=L1/(1+L1);
51 T2=L2/(1+L2);
52 T3=L3/(1+L3);
53 T4=L4/(1+L4);
54 T5=L5/(1+L5);
55 figure(2);
56 bodemag(T0);
57 hold on;
58 bodemag(T1);
59 hold on;
60 bodemag(T2);
61 hold on;
62 bodemag(T3);
63 hold on;
64 bodemag(T4);
65 hold on;
66 bodemag(T5);
67 hold off;
68 legend("K0=7.000 + 0.050*s",...
69         "K1=7.000 + 0.200*s",...
70         "K2=7.000 + 0.525*s",...
71         "K3=7.000 + 0.800*s",...
72         "K4=7.000 + 1.000*s",...
73         "K5=7.000 + 2.000*s");
74 grid on;
75 title('Complimentary Sensitivity, T- $\{ry\}$ ');
76 grid minor;
77 plot_axis;
78 %% 3 Plot Tdoy (S)
79 S0=1/(1+L0);
80 S1=1/(1+L1);
81 S2=1/(1+L2);
82 S3=1/(1+L3);
83 S4=1/(1+L4);
84 S5=1/(1+L5);
85 figure(3);
86 bodemag(S0);
87 hold on;
88 bodemag(S1);
89 hold on;
90 bodemag(S2);
91 hold on;
92 bodemag(S3);
93 hold on;
94 bodemag(S4);
95 hold on;
96 bodemag(S5);
97 hold off;
98 legend("K0=7.000 + 0.050*s",...
99         "K1=7.000 + 0.200*s",...
100        "K2=7.000 + 0.525*s",...
101        "K3=7.000 + 0.800*s",...
102        "K4=7.000 + 1.000*s",...
103        "K5=7.000 + 2.000*s");
104 title('Sensitivity, T- $\{do y\}$ ');
105 grid minor;
106 plot_axis;

```

```

107 %% 4 Plot Tdiy (PS)
108 PS0=P/(1+L0);
109 PS1=P/(1+L1);
110 PS2=P/(1+L2);
111 PS3=P/(1+L3);
112 PS4=P/(1+L4);
113 PS5=P/(1+L5);
114 figure(4);
115 bodemag(PS0);
116 hold on;
117 bodemag(PS1);
118 hold on;
119 bodemag(PS2);
120 hold on;
121 bodemag(PS3);
122 hold on;
123 bodemag(PS4);
124 hold on;
125 bodemag(PS5);
126 hold off;
127 legend("K0=7.000 + 0.050*s",...
128         "K1=7.000 + 0.200*s",...
129         "K2=7.000 + 0.525*s",...
130         "K3=7.000 + 0.800*s",...
131         "K4=7.000 + 1.000*s",...
132         "K5=7.000 + 2.000*s");
133 title('PS, T-di y');
134 grid minor;
135 plot_axis;
136 %% 5 Plot Tny
137 Tny0= -L0/(1+L0);
138 Tny1= -L1/(1+L1);
139 Tny2= -L2/(1+L2);
140 Tny3= -L3/(1+L3);
141 Tny4= -L4/(1+L4);
142 Tny5= -L5/(1+L5);
143 figure(5);
144 bodemag(Tny0);
145 hold on;
146 bodemag(Tny1);
147 hold on;
148 bodemag(Tny2);
149 hold on;
150 bodemag(Tny3);
151 hold on;
152 bodemag(Tny4);
153 hold on;
154 bodemag(Tny5);
155 hold off;
156 legend("K0=7.000 + 0.050*s",...
157         "K1=7.000 + 0.200*s",...
158         "K2=7.000 + 0.525*s",...
159         "K3=7.000 + 0.800*s",...
160         "K4=7.000 + 1.000*s",...
161         "K5=7.000 + 2.000*s");
162 title('T-n y');
163 grid minor;

```

```

164 plot_axis;
165 %% 6 Plot Tru (KS)
166 Tru0= K0/(1+L0);
167 Tru1= K1/(1+L1);
168 Tru2= K2/(1+L2);
169 Tru3= K3/(1+L3);
170 Tru4= K4/(1+L4);
171 Tru5= K5/(1+L5);
172 figure(6);
173 bodemag(Tru0);
174 hold on;
175 bodemag(Tru1);
176 hold on;
177 bodemag(Tru2);
178 hold on;
179 bodemag(Tru3);
180 hold on;
181 bodemag(Tru4);
182 hold on;
183 bodemag(Tru5);
184 hold off;
185 legend("K0=7.000 + 0.050*s",...
186        "K1=7.000 + 0.200*s",...
187        "K2=7.000 + 0.525*s",...
188        "K3=7.000 + 0.800*s",...
189        "K4=7.000 + 1.000*s",...
190        "K5=7.000 + 2.000*s");
191 title('T-r u');
192 grid minor;
193 plot_axis;
194 %% 7 Plot Tdou Tnu
195 Tdou0= -K0/(1+L0);
196 Tdou1= -K1/(1+L1);
197 Tdou2= -K2/(1+L2);
198 Tdou3= -K3/(1+L3);
199 Tdou4= -K4/(1+L4);
200 Tdou5= -K5/(1+L5);
201 figure(7);
202 bodemag(Tdou0);
203 hold on;
204 bodemag(Tdou1);
205 hold on;
206 bodemag(Tdou2);
207 hold on;
208 bodemag(Tdou3);
209 hold on;
210 bodemag(Tdou4);
211 hold on;
212 bodemag(Tdou5);
213 hold off;
214 legend("K0=7.000 + 0.050*s",...
215        "K1=7.000 + 0.200*s",...
216        "K2=7.000 + 0.525*s",...
217        "K3=7.000 + 0.800*s",...
218        "K4=7.000 + 1.000*s",...
219        "K5=7.000 + 2.000*s");
220 title('T-do u T-n u');

```

```

221 grid minor;
222 plot_axis;
223 %% Plot Step Response
224 figure(8);
225 st0=stepplot(T0,t);
226 hold on;
227 st1=stepplot(T1,t);
228 hold on;
229 st2=stepplot(T2,t);
230 hold on;
231 st3=stepplot(T3,t);
232 hold on;
233 st4=stepplot(T4,t);
234 hold on;
235 st5=stepplot(T5,t);
236 hold off;
237 legend("K0=7.000 + 0.050*s",...
238        "K1=7.000 + 0.200*s",...
239        "K2=7.000 + 0.525*s",...
240        "K3=7.000 + 0.800*s",...
241        "K4=7.000 + 1.000*s",...
242        "K5=7.000 + 2.000*s");
243 ylabel('Angular Rate q (rad/s)');
244 grid minor;
245 plot_axis;
246 %% Close Loop Pole
247 figure(10);
248 rlocus(T0);
249 grid on;
250 grid minor;
251 legend("K0=7.000 + 0.050*s");
252 figure(11);
253 rlocus(T1);
254 grid on;
255 grid minor;
256 legend("K1=7.000 + 0.200*s");
257 figure(12);
258 rlocus(T2);
259 grid on;
260 grid minor;
261 legend("K2=7.000 + 0.525*s");
262 figure(13);
263 rlocus(T3);
264 grid on;
265 grid minor;
266 legend("K3=7.000 + 0.800*s");
267 figure(14);
268 rlocus(T4);
269 grid on;
270 grid minor;
271 legend("K4=7.000 + 1.000*s");
272 figure(15);
273 rlocus(T5);
274 grid on;
275 grid minor;
276 legend("K5=7.000 + 2.000*s");

```

Altitude Control Simulation File

```
1 close all;
2 clc;
3 clear;
4 %-----Altitude Control-----%
5 %% Parameters
6 t=0:0.01:2;
7 m = 0.645;
8 a = 9.79;
9 s=tf('s');
10 P_in=1 * a /((m*s)*(s+a));
11 K_in=0.525*(s+12);
12 L_in=P_in*K_in;
13 P=minreal(L_in/(1+L_in)/s);
14 %% Controller
15 K0=0.500;
16 K1=1.000;
17 K2=2.000;
18 K3=4.200;
19 K4=10.000;
20 K5=20.000;
21 %% 1 Plot L = PK
22 L0=P*K0;
23 L1=P*K1;
24 L2=P*K2;
25 L3=P*K3;
26 L4=P*K4;
27 L5=P*K5;
28 figure(1);
29 bodemag(L0);
30 hold on;
31 bodemag(L1);
32 hold on;
33 bodemag(L2);
34 hold on;
35 bodemag(L3);
36 hold on;
37 bodemag(L4);
38 hold on;
39 bodemag(L5);
40 hold off;
41 legend("K0=0.5",...
42        "K1=1",...
43        "K2=3",...
44        "K3=5",...
45        "K4=10",...
46        "K5=20");
47 title('Open Loop bodemag L');
48 grid on;
49 grid minor;
50 plot_axis;
51 %% 2 Plot Try (T)
52 T0=minreal(L0/(1+L0));
53 T1=minreal(L1/(1+L1));
54 T2=minreal(L2/(1+L2));
```



```

55 T3=minreal(L3/(1+L3));
56 T4=minreal(L4/(1+L4));
57 T5=minreal(L5/(1+L5));
58 figure(2);
59 bodemag(T0);
60 hold on;
61 bodemag(T1);
62 hold on;
63 bodemag(T2);
64 hold on;
65 bodemag(T3);
66 hold on;
67 bodemag(T4);
68 hold on;
69 bodemag(T5);
70 hold off;legend("K0=0.5",...
71         "K1=1",...
72         "K2=3",...
73         "K3=5",...
74         "K4=10",...
75         "K5=20");
76 title('Complimentary Sensitivity, T-ry');
77 grid minor;
78 plot_axis;
79 %% 3 Plot Tdoy (S)
80 S0=1/(1+L0);
81 S1=1/(1+L1);
82 S2=1/(1+L2);
83 S3=1/(1+L3);
84 S4=1/(1+L4);
85 S5=1/(1+L5);
86 figure(3);
87 bodemag(S0);
88 hold on;
89 bodemag(S1);
90 hold on;
91 bodemag(S2);
92 hold on;
93 bodemag(S3);
94 hold on;
95 bodemag(S4);
96 hold on;
97 bodemag(S5);
98 hold off;legend("K0=0.5",...
99         "K1=1",...
100        "K2=3",...
101        "K3=5",...
102        "K4=10",...
103        "K5=20");
104 title('Sensitivity, T-do y');
105 grid minor;
106 plot_axis;
107 %% 4 Plot Tdiy (PS)
108 PS0=P/(1+L0);
109 PS1=P/(1+L1);
110 PS2=P/(1+L2);
111 PS3=P/(1+L3);

```

```

112 PS4=P/(1+L4);
113 PS5=P/(1+L5);
114 figure(4);
115 bodemag(PS0);
116 hold on;
117 bodemag(PS1);
118 hold on;
119 bodemag(PS2);
120 hold on;
121 bodemag(PS3);
122 hold on;
123 bodemag(PS4);
124 hold on;
125 bodemag(PS5);
126 hold off; legend("K0=0.5",...
127     "K1=1",...
128     "K2=3",...
129     "K3=5",...
130     "K4=10",...
131     "K5=20");
132 title('PS, T-{di y}');
133 grid minor;
134 plot_axis;
135 %% 5 Plot Tny Tdiu
136 Tny0= -L0/(1+L0);
137 Tny1= -L1/(1+L1);
138 Tny2= -L2/(1+L2);
139 Tny3= -L3/(1+L3);
140 Tny4= -L4/(1+L4);
141 Tny5= -L5/(1+L5);
142 figure(5);
143 bodemag(Tny0);
144 hold on;
145 bodemag(Tny1);
146 hold on;
147 bodemag(Tny2);
148 hold on;
149 bodemag(Tny3);
150 hold on;
151 bodemag(Tny4);
152 hold on;
153 bodemag(Tny5);
154 hold off;
155 legend("K0=0.5",...
156     "K1=1",...
157     "K2=3",...
158     "K3=5",...
159     "K4=10",...
160     "K5=20");
161 title('T-{n y} T-{di u}');
162 grid minor;
163 plot_axis;
164 %% 6 Plot Tru (KS)
165 Tru0= K0/(1+L0);
166 Tru1= K1/(1+L1);
167 Tru2= K2/(1+L2);
168 Tru3= K3/(1+L3);

```

```

169 Tru4= K4/(1+L4);
170 Tru5= K5/(1+L5);
171 figure(6);
172 bodemag(Tru0);
173 hold on;
174 bodemag(Tru1);
175 hold on;
176 bodemag(Tru2);
177 hold on;
178 bodemag(Tru3);
179 hold on;
180 bodemag(Tru4);
181 hold on;
182 bodemag(Tru5);
183 hold off;
184 legend("K0=0.5",...
185         "K1=1",...
186         "K2=3",...
187         "K3=5",...
188         "K4=10",...
189         "K5=20");
190 title('T_{r u}');
191 grid minor;
192 plot_axis;
193 %% 7 Plot Tdou Tnu
194 Tdou0= -K0/(1+L0);
195 Tdou1= -K1/(1+L1);
196 Tdou2= -K2/(1+L2);
197 Tdou3= -K3/(1+L3);
198 Tdou4= -K4/(1+L4);
199 Tdou5= -K5/(1+L5);
200 figure(7);
201 bodemag(Tdou0);
202 hold on;
203 bodemag(Tdou1);
204 hold on;
205 bodemag(Tdou2);
206 hold on;
207 bodemag(Tdou3);
208 hold on;
209 bodemag(Tdou4);
210 hold on;
211 bodemag(Tdou5);
212 hold off;
213 legend("K0=0.5",...
214         "K1=1",...
215         "K2=3",...
216         "K3=5",...
217         "K4=10",...
218         "K5=20");
219 title('T_{do u} T_{n u}');
220 grid minor;
221 plot_axis;
222 %% Plot Step Response
223 figure(8);
224 st0=stepplot(T0,t);
225 hold on;

```

```

226 st1=stepplot(T1,t);
227 hold on;
228 st2=stepplot(T2,t);
229 hold on;
230 st3=stepplot(T3,t);
231 hold on;
232 st4=stepplot(T4,t);
233 hold on;
234 st5=stepplot(T5,t);
235 hold off;
236 legend("K0=0.5",...
237         "K1=1",...
238         "K2=3",...
239         "K3=5",...
240         "K4=10",...
241         "K5=20");
242 ylabel('Altitude (m)');
243 grid minor;
244 plot_axis;
245 %% Close Loop Pole
246 figure(10);
247 rlocus(L0);
248 grid on;
249 grid minor;
250 legend("K0=0.5");
251 figure(11);
252 rlocus(L1);
253 grid on;
254 grid minor;
255 legend("K1=1");
256 figure(12);
257 rlocus(L2);
258 grid on;
259 grid minor;
260 legend("K2=3");
261 figure(13);
262 rlocus(L3);
263 grid on;
264 grid minor;
265 legend("Kz=4.2");
266 figure(14);
267 rlocus(L4);
268 grid on;
269 grid minor;
270 legend("K4=10");
271 figure(15);
272 rlocus(L5);
273 grid on;
274 grid minor;
275 legend("K5=20");

```

Translational Control Simulation File

```

1 close all;
2 clc;
3 clear;

```

```

4  %-----Translational Velocity Control-----%
5  %% Parameters
6  t=0:0.01:2;
7  g = 9.8;
8  s=tf('s');
9  J_y = 0.0019536;
10 a = 9.79;
11 P_2=1 * a /((J_y*s)*(s+a));
12 K_2=0.0045*(s+12);
13 P_3 = minreal(P_2*K_2/(1+P_2*K_2)/s);
14 K_3 = 9;
15 L_2=P_3*K_3;
16 T_2=minreal(L_2/(1+L_2));
17 P = minreal(T_2 *g/s);
18 %% Controller
19 K0=0.02*(s+10);
20 K1=0.05*(s+10);
21 K2=0.075*(s+10);
22 K3=0.10*(s+10);
23 K4=0.15*(s+10);
24 K5=0.20*(s+10);
25 %% 1 Plot L = PK
26 L0=minreal(P*K0);
27 L1=minreal(P*K1);
28 L2=minreal(P*K2);
29 L3=minreal(P*K3);
30 L4=minreal(P*K4);
31 L5=minreal(P*K5);
32 figure(1);
33 bodemag(L0);
34 hold on;
35 bodemag(L1);
36 hold on;
37 bodemag(L2);
38 hold on;
39 bodemag(L3);
40 hold on;
41 bodemag(L4);
42 hold on;
43 bodemag(L5);
44 hold off;
45 legend("K0=1.3+0.02*s",...
46        "K1=1.3+0.05*s",...
47        "K2=1.3+0.08*s",...
48        "K3=1.3+0.10*s",...
49        "K4=1.3+0.15*s",...
50        "K5=1.3+0.20*s");
51 title('Open Loop bodemag L');
52 grid on;
53 grid minor;
54 plot_axis;
55 %% 2 Plot Try (T)
56 T0=minreal(L0/(1+L0));
57 T1=minreal(L1/(1+L1));
58 T2=minreal(L2/(1+L2));
59 T3=minreal(L3/(1+L3));
60 T4=minreal(L4/(1+L4));

```

```

61 T5=minreal(L5/(1+L5));
62 figure(2);
63 bodemag(T0);
64 hold on;
65 bodemag(T1);
66 hold on;
67 bodemag(T2);
68 hold on;
69 bodemag(T3);
70 hold on;
71 bodemag(T4);
72 hold on;
73 bodemag(T5);
74 hold off;
75 legend("K0=1.3+0.02*s",...
76        "K1=1.3+0.05*s",...
77        "K2=1.3+0.08*s",...
78        "K3=1.3+0.10*s",...
79        "K4=1.3+0.15*s",...
80        "K5=1.3+0.20*s");
81 title('Complimentary Sensitivity, T- $\{ry\}$ ');
82 grid on;
83 grid minor;
84 plot_axis;
85 %% 3 Plot Tdoy (S)
86 S0=1/(1+L0);
87 S1=1/(1+L1);
88 S2=1/(1+L2);
89 S3=1/(1+L3);
90 S4=1/(1+L4);
91 S5=1/(1+L5);
92 figure(3);
93 bodemag(S0);
94 hold on;
95 bodemag(S1);
96 hold on;
97 bodemag(S2);
98 hold on;
99 bodemag(S3);
100 hold on;
101 bodemag(S4);
102 hold on;
103 bodemag(S5);
104 hold off;
105 legend("K0=1.3+0.02*s",...
106        "K1=1.3+0.05*s",...
107        "K2=1.3+0.08*s",...
108        "K3=1.3+0.10*s",...
109        "K4=1.3+0.15*s",...
110        "K5=1.3+0.20*s");
111 title('Sensitivity, T- $\{do y\}$ ');
112 grid on;
113 grid minor;
114 plot_axis;
115 %% 4 Plot Tdiy (PS)
116 PS0=P/(1+L0);
117 PS1=P/(1+L1);

```

```

118 PS2=P/(1+L2);
119 PS3=P/(1+L3);
120 PS4=P/(1+L4);
121 PS5=P/(1+L5);
122 figure(4);
123 bodemag(PS0);
124 hold on;
125 bodemag(PS1);
126 hold on;
127 bodemag(PS2);
128 hold on;
129 bodemag(PS3);
130 hold on;
131 bodemag(PS4);
132 hold on;
133 bodemag(PS5);
134 hold off;
135 legend("K0=1.3+0.02*s",...
136        "K1=1.3+0.05*s",...
137        "K2=1.3+0.08*s",...
138        "K3=1.3+0.10*s",...
139        "K4=1.3+0.15*s",...
140        "K5=1.3+0.20*s");
141 title('PS, T_{di y}');
142 grid on;
143 grid minor;
144 plot_axis;
145 %% 5 Plot Tny
146 Tny0= -L0/(1+L0);
147 Tny1= -L1/(1+L1);
148 Tny2= -L2/(1+L2);
149 Tny3= -L3/(1+L3);
150 Tny4= -L4/(1+L4);
151 Tny5= -L5/(1+L5);
152 figure(5);
153 bodemag(Tny0);
154 hold on;
155 bodemag(Tny1);
156 hold on;
157 bodemag(Tny2);
158 hold on;
159 bodemag(Tny3);
160 hold on;
161 bodemag(Tny4);
162 hold on;
163 bodemag(Tny5);
164 hold off;
165 legend("K0=1.3+0.02*s",...
166        "K1=1.3+0.05*s",...
167        "K2=1.3+0.08*s",...
168        "K3=1.3+0.10*s",...
169        "K4=1.3+0.15*s",...
170        "K5=1.3+0.20*s");
171 title('T_{n y}');
172 grid on;
173 grid minor;
174 plot_axis;

```

```

175 %% 6 Plot Tru (KS)
176 Tru0= K0/(1+L0);
177 Tru1= K1/(1+L1);
178 Tru2= K2/(1+L2);
179 Tru3= K3/(1+L3);
180 Tru4= K4/(1+L4);
181 Tru5= K5/(1+L5);
182 figure(6);
183 bodemag(Tru0);
184 hold on;
185 bodemag(Tru1);
186 hold on;
187 bodemag(Tru2);
188 hold on;
189 bodemag(Tru3);
190 hold on;
191 bodemag(Tru4);
192 hold on;
193 bodemag(Tru5);
194 legend("K0=1.3+0.02*s",...
195        "K1=1.3+0.05*s",...
196        "K2=1.3+0.08*s",...
197        "K3=1.3+0.10*s",...
198        "K4=1.3+0.15*s",...
199        "K5=1.3+0.20*s");
200 title('T- $\{r u\}$ ');
201 grid on;
202 grid minor;
203 plot_axis;
204 %% 7 Plot Tdou Tnu
205 Tdou0= -K0/(1+L0);
206 Tdou1= -K1/(1+L1);
207 Tdou2= -K2/(1+L2);
208 Tdou3= -K3/(1+L3);
209 Tdou4= -K4/(1+L4);
210 Tdou5= -K5/(1+L5);
211 figure(7);
212 bodemag(Tdou0);
213 hold on;
214 bodemag(Tdou1);
215 hold on;
216 bodemag(Tdou2);
217 hold on;
218 bodemag(Tdou3);
219 hold on;
220 bodemag(Tdou4);
221 hold on;
222 bodemag(Tdou5);
223 hold off;
224 legend("K0=1.3+0.02*s",...
225        "K1=1.3+0.05*s",...
226        "K2=1.3+0.08*s",...
227        "K3=1.3+0.10*s",...
228        "K4=1.3+0.15*s",...
229        "K5=1.3+0.20*s");
230 title('T- $\{do u\}$  T- $\{n u\}$ ');
231 grid on;

```



```

232 grid minor;
233 plot_axis;
234 %% Plot Step Response
235 figure(8);
236 st0=stepplot(T0,t);
237 hold on;
238 st1=stepplot(T1,t);
239 hold on;
240 st2=stepplot(T2,t);
241 hold on;
242 st3=stepplot(T3,t);
243 hold on;
244 st4=stepplot(T4,t);
245 hold on;
246 st5=stepplot(T5,t);
247 hold off;
248 legend("K0=1.3+0.02*s",...
249        "K1=1.3+0.05*s",...
250        "K2=1.3+0.08*s",...
251        "K3=1.3+0.10*s",...
252        "K4=1.3+0.15*s",...
253        "K5=1.3+0.20*s");
254 ylabel('Translational Velocity Vy (m/s)');
255 grid on;
256 grid minor;
257 plot_axis;
258
259 %% Close Loop Pole
260 figure(10);
261 rlocus(T0);
262 grid on;
263 grid minor;
264 legend("K0=1.3+0.01*s");
265 figure(11);
266 rlocus(T1);
267 grid on;
268 grid minor;
269 legend("K1=1.3+0.02*s");
270 figure(12);
271 rlocus(L2);
272 grid on;
273 grid minor;
274 legend("Kvy=0.075*(s+10)s");
275 figure(13);
276 rlocus(T3);
277 grid on;
278 grid minor;
279 legend("K3=1.3+0.10*s");
280 figure(14);
281 rlocus(T4);
282 grid on;
283 grid minor;
284 legend("Kvy=1.3+0.20*s");
285 figure(15);
286 rlocus(T5);
287 grid on;
288 grid minor;

```

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

Translational Position Simulation File

```
1 clc;
2 clear;
3 close all;
4 %-----Translational Position Control-----%
5 %% Parameters
6 t=0:0.01:10;
7 s=tf('s');
8 %% Parameters
9 T2_vy = (149.2*s^2 + 3282*s + 1.79e04)/...
10         (s^4+32.34*s^3+...
11         622.7*s^2+5717*s+1.79e04);
12 P = minreal(T2_vy / s);
13 K0=0.2;
14 K1=0.4;
15 K2=0.8;
16 K3=1.0;
17 K4=2.0;
18 K5=4.0;
19 %% 1 Plot L = PK
20 L0=minreal(P*K0);
21 L1=minreal(P*K1);
22 L2=minreal(P*K2);
23 L3=minreal(P*K3);
24 L4=minreal(P*K4);
25 L5=minreal(P*K5);
26 figure(1);
27 bodemag(L0);
28 hold on;
29 bodemag(L1);
30 hold on;
31 bodemag(L2);
32 hold on;
33 bodemag(L3);
34 hold on;
35 bodemag(L4);
36 hold on;
37 bodemag(L5);
38 hold off;
39 legend("K0=0.2",...
40        "K1=0.4",...
41        "K2=0.8",...
42        "K3=1.0",...
43        "K4=2.0",...
44        "K5=4.0");
45 title('Open Loop bodemag L');
46 grid on;
47 grid minor;
48 plot_axis;
49 %% 2 Plot Try (T)
50 T0=minreal(L0/(1+L0));
51 T1=minreal(L1/(1+L1));
52 T2=minreal(L2/(1+L2));
```

```

53 T3=minreal(L3/(1+L3));
54 T4=minreal(L4/(1+L4));
55 T5=minreal(L5/(1+L5));
56 figure(2);
57 bodemag(T0);
58 hold on;
59 bodemag(T1);
60 hold on;
61 bodemag(T2);
62 hold on;
63 bodemag(T3);
64 hold on;
65 bodemag(T4);
66 hold on;
67 bodemag(T5);
68 hold off;
69 title('Complimentary Sensitivity, T- $\{r y\}$ ');
70 legend("K0=0.2",...
71        "K1=0.4",...
72        "K2=0.8",...
73        "K3=1.0",...
74        "K4=2.0",...
75        "K5=4.0");
76 grid on;
77 grid minor;
78 plot_axis;
79 %% Plot Step Response
80 figure(9);
81 st0=stepplot(T0,t);
82 hold on;
83 st1=stepplot(T1,t);
84 hold on;
85 st2=stepplot(T2,t);
86 hold on;
87 st3=stepplot(T3,t);
88 hold on;
89 st4=stepplot(T4,t);
90 hold on;
91 st5=stepplot(T5,t);
92 hold off;
93 legend("K0=0.2",...
94        "K1=0.4",...
95        "K2=0.8",...
96        "K3=1.0",...
97        "K4=2.0",...
98        "K5=4.0");
99 ylabel('Angular Rate q (rad/s)');
100 grid on;
101 grid minor;
102 plot_axis;

```

APPENDIX B
MATLAB BASED GUI

Mark3 Ground Station

```
1 function varargout = Mark3_GUI(varargin)
2 gui_Singleton = 1;
3 gui_State = struct('gui_Name', mfilename, ...
4 'gui_Singleton', gui_Singleton, ...
5 'gui_OpeningFcn', @Mark3_GUI_OpeningFcn, ...
6 'gui_OutputFcn', @Mark3_GUI_OutputFcn, ...
7 'gui_LayoutFcn', [] , ...
8 'gui_Callback', []);
9 if nargin && ischar(varargin{1})
10     gui_State.gui_Callback = str2func(varargin{1});
11 end
12
13 if nargin
14     [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
15 else
16     gui_mainfcn(gui_State, varargin{:});
17 end
18
19 function Mark3_GUI_OpeningFcn(hObject, eventdata, handles, varargin)
20
21 %fclose(instrfindall);
22 global udp_timer;
23 global udp_timer_2;
24 global drone;
25 drone(1).command_count = 50;
26 drone(1).command = 1234;
27 drone(1).des_x = 0;
28 drone(1).des_y = 0.75;
29 drone(1).des_z = 0.60;
30 drone(1).des_psi = 0;
31 drone(1).polar_psi = 0;
32
33 strRec = '';
34 setappdata(hObject, 'strRec', strRec);
35 udp_timer = 5;
36 udp_timer_2 = 10;
37
38 handles.fid = fopen('roll.txt','a');
39 handles.output = hObject;
40 % Update handles structure
41 guidata(hObject, handles);
42 % UIWAIT makes Mark3_GUI wait for user response (see UIRESUME)
43 % uiwait(handles.figure1);
44
45
46 % — Outputs from this function are returned to the command line.
47 function varargout = Mark3_GUI_OutputFcn(hObject, eventdata, handles)
48
49 varargout{1} = handles.output;
50
51 function Open_Serial_Callback(hObject, eventdata, handles)
52 handles.xbee_pack_drone1=serial('COM4');
53 set(handles.xbee_pack_drone1,'BaudRate',230400,'Parity','none',...
54     'DataBits',8,'StopBits',1,...
```

```

55     'Timeout', 0.01,...
56     'InputBufferSize',2048,...
57     'OutputBufferSize',10240,...
58     'BytesAvailableFcnMode','byte',...
59     'BytesAvailableFcn',{@Xbee_bytes_drone1,handles});
60 fopen(handles.xbee_pack_drone1);
61 handles.timer = timer('Period',0.01,'ExecutionMode',...
62 'FixedRate', 'TimerFcn',{@xbee_pack_drone1-Send,handles});
63 start(handles.timer);
64 guidata(hObject, handles);
65
66 function Xbee_bytes_drone1(obj,eventdata,handles)
67 global udp_timer;
68 strRec = getappdata(handles.figure1, 'strRec');
69 check=get(obj,'BytesAvailable');
70 if check
71     packet = fread(obj,check,'uchar');
72     xbee_length = length(packet);
73     if udp_timer == 5
74         set(handles.xbee_pack_length_drone1,'string',num2str(xbee_length));
75     end
76     pack_string = char(packet);
77     if get(handles.Write_Data,'value')
78         fprintf(handles.fid,'%s',pack_string);
79     end
80     if udp_timer == 5
81         if get(handles.data_disp,'value')
82             set(handles.Serial_Dis,'string',pack_string);
83         end
84     end
85     setappdata(handles.figure1, 'strRec', strRec);
86 end
87
88 function Close_Serial_Callback(hObject, eventdata, handles)
89 stop(handles.timer);
90 delete(handles.timer);
91 stopasynch(handles.xbee_pack_drone1);
92 fclose(handles.xbee_pack_drone1);
93 delete(handles.xbee_pack_drone1);
94 %fclose(instrfindall)
95 % — Executes on button press in pushbutton1.
96 function Open_Udp_Callback(hObject, eventdata, handles)
97 % 184 is the buffer size
98 BUFSIZE = 184;
99 % 5400 is the port number
100 PORT = 5400;
101 handles.udpReceive=udp('127.0.0.1','LocalPort',PORT,...
102     'InputBufferSize',BUFSIZE,'Timeout',Inf);
103 fopen(handles.udpReceive);
104 handles.udpReceive.ReadAsyncMode = 'continuous';
105 set(handles.udpReceive,'DatagramTerminateMode','on');
106 handles.udpReceive.DatagramReceivedFcn=@UDPdataProtocol,handles};
107 guidata(hObject, handles);
108
109
110 % — Executes on button press in pushbutton2.
111 function Close_Udp_Callback(hObject, eventdata, handles)

```

```

112 stopasync(handles.udpReceive);
113 fclose(handles.udpReceive);
114 delete(handles.udpReceive);
115 %close();
116
117 function DataSend_Callback(hObject, eventdata, handles)
118 global drone;
119 str = get(hObject, 'string');
120 if length(str)==4
121     drone(1).command = str2num(str);
122 end
123
124 function DataSend_CreateFcn(hObject, ~, handles)
125 if ispc && isequal(get(hObject,'BackgroundColor'),...
126     get(0,'defaultUiControlBackgroundColor'))
127     set(hObject,'BackgroundColor','white');
128 end
129
130
131 % — Executes on button press in Write_Data.
132 function Write_Data_Callback(hObject, eventdata, handles)
133 % — Executes on button press in send_otus.
134 function Send_Otus_Callback(hObject, eventdata, handles)
135
136 function xbee_pack_drone1_Send(obj,event,handles)
137
138 global drone
139
140 drone(1).data_command = int16(drone(1).command);
141
142 if get(handles.checkbox_plan,'value')
143     drone(1).polar_psi = drone(1).polar_psi + 0.4;
144     if drone(1).polar_psi >= 180
145         drone(1).polar_psi = -180;
146     end
147     drone(1).des_x = 0.75 * sin(drone(1).polar_psi*pi/180);
148     drone(1).des_y = 0.75 * cos(drone(1).polar_psi*pi/180);
149     drone(1).des_z = 0.6;
150     drone(1).des_psi = -drone(1).polar_psi;
151 end
152
153 data_start = [uint8(111),uint8(121),uint8(131)];
154 data_x_check = [uint8(21),uint8(101)];
155 data_y_check = [uint8(22),uint8(102)];
156 data_z_check = [uint8(23),uint8(103)];
157 data_vx_check = [uint8(24),uint8(104)];
158 data_vy_check = [uint8(25),uint8(105)];
159 data_vz_check = [uint8(26),uint8(106)];
160 data_yaw_check = [uint8(27),uint8(107)];
161 data_pitch_check = [uint8(28),uint8(108)];
162 data_roll_check = [uint8(29),uint8(109)];
163
164 data_q0_check = [uint8(31),uint8(111)];
165 data_q1_check = [uint8(32),uint8(112)];
166 data_q2_check = [uint8(33),uint8(113)];
167 data_q3_check = [uint8(34),uint8(114)];
168

```

```

169 data_des_x_check = [uint8(35),uint8(115)];
170 data_des_y_check = [uint8(36),uint8(116)];
171 data_des_z_check = [uint8(37),uint8(117)];
172 data_des_psi_check = [uint8(38),uint8(118)];
173
174 data_command_check = [uint8(30),uint8(110)];
175
176 data_end = [uint8(50),uint8(75),uint8(100)];
177
178 drone(1).array_x=typecast(drone(1).data_x,'uint8');
179 drone(1).array_y=typecast(drone(1).data_y,'uint8');
180 drone(1).array_z=typecast(drone(1).data_z,'uint8');
181
182 drone(1).array_vx=typecast(drone(1).data_vx,'uint8');
183 drone(1).array_vy=typecast(drone(1).data_vy,'uint8');
184 drone(1).array_vz=typecast(drone(1).data_vz,'uint8');
185
186 drone(1).array_roll=typecast(drone(1).data_roll,'uint8');
187 drone(1).array_pitch=typecast(drone(1).data_pitch,'uint8');
188 drone(1).array_yaw=typecast(drone(1).data_yaw,'uint8');
189
190 drone(1).array_q0=typecast(drone(1).data_q0,'uint8');
191 drone(1).array_q1=typecast(drone(1).data_q1,'uint8');
192 drone(1).array_q2=typecast(drone(1).data_q2,'uint8');
193 drone(1).array_q3=typecast(drone(1).data_q3,'uint8');
194
195 data_des_x = single(drone(1).des_x);
196 data_des_y = single(drone(1).des_y);
197 data_des_z = single(drone(1).des_z);
198 data_des_psi = single(drone(1).des_psi);
199
200 drone(1).array_des_x=typecast(data_des_x,'uint8');
201 drone(1).array_des_y=typecast(data_des_y,'uint8');
202 drone(1).array_des_z=typecast(data_des_z,'uint8');
203 drone(1).array_des_psi=typecast(data_des_psi,'uint8');
204
205 drone(1).array_command=typecast(drone(1).data_command,'uint8');
206 %clc;
207
208 packet = [data_start ...
209     data_x_check drone(1).array_x ...
210     data_y_check drone(1).array_y ...
211     data_z_check drone(1).array_z ...
212     data_vx_check drone(1).array_vx ...
213     data_vy_check drone(1).array_vy ...
214     data_vz_check drone(1).array_vz ...
215     data_yaw_check drone(1).array_yaw ...
216     data_pitch_check drone(1).array_pitch ...
217     data_roll_check drone(1).array_roll ...
218     data_q0_check drone(1).array_q0 ...
219     data_q1_check drone(1).array_q1 ...
220     data_q2_check drone(1).array_q2 ...
221     data_q3_check drone(1).array_q3 ...
222     data_des_x_check drone(1).array_des_x ...
223     data_des_y_check drone(1).array_des_y ...
224     data_des_z_check drone(1).array_des_z ...
225     data_des_psi_check drone(1).array_des_psi ...

```



```

226     data_command_check drone(1).array_command ...
227     data_end];
228
229 if get(handles.send_lotus, 'value')
230     fwrite(handles.xbee_pack_drone1, packet, 'uchar');
231 end
232
233
234 % — Executes on button press in pushbutton6.
235 function Exit_Callback(hObject, eventdata, handles)
236 fclose(handles.fid);
237 close();
238
239
240
241 function xbee_pack_drone1_Callback(hObject, eventdata, handles)
242
243 function xbee_pack_drone1_CreateFcn(hObject, eventdata, handles)
244
245 if ispc && isequal(get(hObject, 'BackgroundColor'), ...
246     get(0, 'defaultUiControlBackgroundColor'))
247     set(hObject, 'BackgroundColor', 'white');
248 end
249
250
251 % — Executes on button press in checkbox_plan.
252 function checkbox_plan_Callback(hObject, eventdata, handles)

```

UDP Protocol

```

1 function UDPdataProtocol(obj, event, handles)
2 byte_vector = fread(handles.udpreceive, 1);
3 if mod(length(uint8(byte_vector)), 8) == 0
4     double_vector = typecast(uint8(byte_vector), 'double');
5 end
6 controller_id = strcat(native2unicode(byte_vector(1)), ...
7     native2unicode(byte_vector(2)), ...
8     native2unicode(byte_vector(3)), ...
9     native2unicode(byte_vector(4)), ...
10    native2unicode(byte_vector(5)), ...
11    native2unicode(byte_vector(6)), ...
12    native2unicode(byte_vector(7)), ...
13    native2unicode(byte_vector(8)));
14
15 timestamp = double_vector(2);
16
17 lin_pos_x = double_vector(3);
18 lin_pos_y = double_vector(4);
19 lin_pos_z = double_vector(5);
20
21 lin_vel_x = double_vector(6);
22 lin_vel_y = double_vector(7);
23 lin_vel_z = double_vector(8);
24
25 lin_acc_x = double_vector(9);
26 lin_acc_y = double_vector(10);

```

```

27 lin_acc_z = double_vector(11);
28
29 quaternion_x = double_vector(12);
30 quaternion_y = double_vector(13);
31 quaternion_z = double_vector(14);
32 quaternion_w = double_vector(15);
33
34 ang_vel_x = double_vector(16);
35 ang_vel_y = double_vector(17);
36 ang_vel_z = double_vector(18);
37
38 ang_acc_x = double_vector(19);
39 ang_acc_y = double_vector(20);
40 ang_acc_z = double_vector(21);
41
42 button_1 = double_vector(22);
43 button_2 = double_vector(23);
44
45 global udp_timer;
46
47 global drone;
48
49 if controller_id == '67C87001'
50     drone(1).data_x=single(-lin_pos_z);
51     drone(1).data_y=single(-lin_pos_x);
52     drone(1).data_z=single(-lin_pos_y);
53
54     drone(1).data_vx=single(-lin_vel_z);
55     drone(1).data_vy=single(-lin_vel_x);
56     drone(1).data_vz=single(-lin_vel_y);
57
58     drone(1).quat = [quaternion_w,...
59                     -quaternion_z,...
60                     -quaternion_x,...
61                     quaternion_y];
62     tmp_eul = quat2eul(drone(1).quat) * 57.29578;
63     drone(1).data_yaw=single(tmp_eul(1));
64     drone(1).data_pitch=single(tmp_eul(2));
65     drone(1).data_roll=single(tmp_eul(3));
66
67     drone(1).data_q0 = single(drone(1).quat(1));
68     drone(1).data_q1 = single(drone(1).quat(2));
69     drone(1).data_q2 = single(drone(1).quat(3));
70     drone(1).data_q3 = single(drone(1).quat(4));
71 end
72
73 if udp_timer == 5
74     set(handles.data_show_x, 'string', num2str(drone(1).data_x));
75     set(handles.data_show_y, 'string', num2str(drone(1).data_y));
76     set(handles.data_show_z, 'string', num2str(drone(1).data_z));
77
78     set(handles.data_show_vx, 'string', num2str(drone(1).data_vx));
79     set(handles.data_show_vy, 'string', num2str(drone(1).data_vy));
80     set(handles.data_show_vz, 'string', num2str(drone(1).data_vz));
81
82     set(handles.data_show_roll, 'string', num2str(drone(1).data_roll));
83     set(handles.data_show_pitch, 'string', num2str(drone(1).data_pitch));

```

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

APPENDIX C
MARK3 FLIGHT CONTROLLER FIRMWARE

Mark3 Flight Controller Code - Main Loop

```
1 #include "Copter.h"
2 #include "Interrupt.h"
3 _Copter Copter;
4 SBUS R9DS(Serial2);
5 void setup()
6 {
7   InitComm(); //Teensy 3.2
8   Copter.Copter_Init();
9 }
10 void loop()
11 {
12   Copter.loopClock1 = micros();
13   Copter.Ctrl_timer1++; //100Hz
14   RC_refine();
15   Copter.AHRS();
16   Copter.PEST();
17   Copter.TranslationControl();
18   Copter.AttitudeControl();
19   Copter.AltitudeControl();
20   Copter.InputTransform();
21   Copter.command_Comm();
22   Copter.Copter_Check();
23 }
```

Mark3 Flight Controller Code - Main Library of Definition

```
1 #include <Eigen.h>
2 #include <Eigen/LU>
3 #include "Arduino.h"
4 #include "i2c_t3.h"
5 #include "EEPROM.h"
6 #include <SBUS.h>
7 #include "System.h"
8 #include "Estimation.h"
9 #include "Sensor.h"
10 #include "Motor.h"
11 #include <cmath>
12 #define LED0 13
13 #define LED1 31
14 #define LED2 33
15 #define Attitude_mode 1
16 #define Altitude_mode 2
17 #define loiter_mode 3
18 using namespace Eigen;
19 class _Copter
20 {
21 public:
22   /*-----*/
23   /*Copter.cpp*/
24   unsigned long loopClock1 = 0, loopClock2 = 0;
25   void Copter_Init();
26   void Copter_Check();
27   struct _flag
28   {
29     uint8_t ARMED = 0;
30     uint8_t takingoff = 0;
31     int8_t turnoff = 0;
32     uint8_t CRASHED = 0;
33     uint8_t AltEmergency = 0;
34     int8_t calibratedA = 0;
35     int8_t calibratedG = 0;
36     int8_t calibrationOn = 0;
37     uint8_t mode = 0;
```

```

38     uint8_t momentstart = 0;
39     uint8_t LoiterSwitch = 0;
40 };
41 _flag flag;
42 int16_t gltimer = 0, rltimer = 0;
43 int8_t glch = 10, rlch = 10;
44
45 /*Motor.cpp*/
46 void MotorModel(double omega1, double
47                 omega2, double omega3, double omega4);
48 void MotorRun();
49 void Motor_init();
50 void Motor_stop();
51 void InputTransform();
52 /*Comm.cpp*/
53 uint8_t xbee_data[500];
54 int16_t xbee_length;
55 int16_t xbeetime1 = 0;
56 void command_Comm();
57 void Xbee_comm();
58 void Xbee_Packet();
59 void Xbee_receive(int16_t order, uint16_t stmp);
60 struct _RCsignal
61 {
62     short ROLL;
63     short PITCH;
64     short THROTTLE;
65     short YAW;
66     short MODE;
67     short SWITCH;
68     short CH7;
69     short CH8;
70     short CH9;
71     short CH10;
72 };
73 _RCsignal RCsignal;
74 short Xbee_timer = 0;
75 short Xbee_receive_timer = 0;
76 short comorder; /*---Command---*/
77 short datalength = 0;
78 /*Interrupt.cpp*/
79 struct _IRpulse
80 {
81     int32_t ringbuffer[30];
82     unsigned long timer_up, timer_down;
83 };
84 _IRpulse VRsensor1;
85
86 /*Sensor.cpp*/
87 void InitSensor();
88 uint8_t I2Cwrite(uint8_t SENSOR_ADDRESS,
89                 uint8_t SENSOR_REGISTER,
90                 uint8_t SENSOR_VALUE,
91                 bool sendStop);
92 uint8_t I2CRead(uint8_t SENSOR_ADDRESS,
93                uint8_t SENSOR_REGISTER,
94                uint8_t nbytes);
95 void MPU6050read();
96 void MPU6050ThermalCompensation();
97 void MPU6050Sixpoint();
98 void MPU6050AccCali(uint8_t point);
99 void MPU6050GyroCali();
100 void AccPointRead();
101 float GyroCollection[3] = {0, 0, 0};
102 float gyro_temp, temperature;
103 int16_t mpu_temperature;
104 short GyroCaliFlag = 0;
105 uint8_t i2cData[30];

```

```

106     uint8_t i2c1Data[30];
107     struct _float {
108         float x;
109         float y;
110         float z;
111     };
112     struct _lint16 {
113         short x;
114         short y;
115         short z;
116     };
117     struct _trans {
118         struct _lint16 origin;
119         struct _float filter;
120         struct _float histor;
121         struct _float aftcal;
122         struct _float quietf;
123         struct _float tempcp;
124         struct _float radian;
125     };
126     struct _sensor {
127         struct _trans acc;
128         struct _trans gyro;
129     };
130     _sensor gy89, gy86;
131     struct _Acc-Cali {
132         int16_t acc_calitimer = 0;
133         int32_t acc_calitmpx;
134         int32_t acc_calitmpy;
135         int32_t acc_calitmpz;
136         int16_t accel_raw_ref[6][3];
137         float acc_offset[3];
138         float a[3][3];
139         float T[3][3];
140         float g = 8192; //+-4g
141     };
142     _Acc-Cali Acc-Cali;
143
144     /*AttitudeEstimator.cpp*/
145     float inte_gyro[3];
146     struct _LKF
147     {
148         float p;
149         float q;
150         float p_bias;
151         float q_bias;
152         float p_raw;
153         float q_raw;
154         float acc_roll;
155         float acc_pitch;
156     };
157     _LKF LKF;
158
159     /*-----Position Estimator-----*/
160     union {
161         float f;
162         unsigned long ul;
163     } otus_tmp;
164     union {
165         float f;
166         uint8_t ul[4];
167     } xbee_tmp;
168     struct _Otus
169     {
170         float x;
171         float y;
172         float z;
173         float vx;
174         float vy;

```

```

175     float vz;
176     float yaw;
177     float pitch;
178     float roll;
179     float yaw_sin;
180     float yaw_cos;
181     float q0;
182     float q1;
183     float q2;
184     float q3;
185 };
186 _Otus Otus;
187 void Linear_Kalman_Filter();
188 struct _state {
189     float phi;
190     float theta;
191     float psi;
192     float phi_rad;
193     float theta_rad;
194     float psi_rad;
195     float phi_sin;
196     float theta_sin;
197     float psi_sin;
198     float phi_cos;
199     float theta_cos;
200     float psi_cos;
201     float p;
202     float q;
203     float r;
204     float p_rad;
205     float q_rad;
206     float r_rad;
207     float ax_b;
208     float ay_b;
209     float az_b;
210     float ax;
211     float ay;
212     float az;
213     float vx;
214     float vy;
215     float vz;
216     float x;
217     float y;
218     float z;
219     int16_t comm;
220
221     int16_t takeoff_t = 0;
222     int16_t landing_t = 0;
223     uint8_t standby;
224     uint8_t takeoff;
225     uint8_t flight;
226     uint8_t land;
227     bool flight_state[4];
228     float tmp_U1;
229     bool update_phi = 0;
230     bool update_theta = 0;
231     bool update_psi = 0;
232     //bool update_p;
233     //bool update_q;
234     //bool update_r;
235     bool update_vx = 0;
236     bool update_vy = 0;
237     bool update_vz = 0;
238     bool update_x = 0;
239     bool update_y = 0;
240     bool update_z = 0;
241 };
242 _state state;
243 struct IMU {
244     float phi;
245     float theta;

```



```

246     float psi;
247     float phi_rad;
248     float theta_rad;
249     float psi_rad;
250     float phi_sin;
251     float theta_sin;
252     float psi_sin;
253     float phi_cos;
254     float theta_cos;
255     float psi_cos;
256     float p;
257     float q;
258     float r;
259     float p_rad;
260     float q_rad;
261     float r_rad;
262 };
263 _IMU IMU;
264 struct _PosKF {
265     float P[3][3];
266     float tmp[3][3];
267     float tmp_s[6];
268     float state[3];
269     float a;
270     float v;
271     float ps;
272     float S[2][2];
273     float y[3];
274     float K[3][3];
275     float Q;
276     float R0;
277     float R1;
278 };
279 _PosKF PosX, PosY, PosZ;
280 float q0 = 1.0f;
281 float q1 = 0.0f;
282 float q2 = 0.0f;
283 float q3 = 0.0f;
284 float q3old = 0.0f;
285 float exInt = 0.0;
286 float eyInt = 0.0;
287 float ezInt = 0.0;
288 float twoKp = twoKpDef;
289 float twoKi = twoKiDef;
290 float beta = betaDef;
291 float integralFBx = 0.0f, integralFBy = 0.0f, integralFBz = 0.0f;
292
293 void PEST();
294 void Pos_Kalman(struct _PosKF *P,tmp);
295 void AHRS_Check();
296 void Madgwick_MARG_Update();
297 //void MadgwickAHRSupdateIMU(float gx, float gy, float gz, float ax, float ay, float az);
298 void MahonyAHRSupdate(float gx, float gy, float gz,
299                       float ax, float ay, float az,
300                       float mx, float my, float mz);
301 void MahonyAHRSupdateIMU(float gx, float gy, float gz,
302                          float ax, float ay, float az);
303 void AHRS_filter_init();
304 void KF_init();
305 void AHRS();
306 void State_Reset();
307 void AHRS_filter();
308 void gy86_Dataanl();
309 struct _IIR {
310     float b0;
311     float b1;
312     float b2;
313     float a1;
314     float a2;

```

```

315     float element0;
316     float element1;
317     float element2;
318 };
319 _IIR gyro_IIRx, gyro_IIRy, gyro_IIRz,
320 acc_IIRx, acc_IIRy, acc_IIRz;
321 void IIR_set_cutoff_frequency(float sample_freq,
322                               float cutoff_freq,
323                               struct _IIR *input_IIR);
324 float IIR_filter_apply(float cutoff_freq,
325                        float sample,
326                        struct _IIR *input_IIR);
327 double integral_r = 0;
328
329 /*Control part*/
330 float U1, U2, U3, U4;
331 uint8_t LockYaw = 0;
332 struct _Ztransform {
333     float Input[5] = {0, 0, 0, 0, 0};
334     float Output[5] = {0, 0, 0, 0, 0};
335     float Integral;
336 };
337 _Ztransform Pcon, Qcon, Rcon, Phicon,
338 Thetacon, Psicon, Vzcon, Zcon, Vxpre,
339 Xpre, Vxcon, Xcon, Vypre, Ypre, Vycon, Ycon;
340 struct _Target
341 {
342     float phi;
343     float theta;
344     float psi;
345     float phi_rad;
346     float theta_rad;
347     float sin_phi;
348     float cos_phi;
349     float sin_theta;
350     float cos_theta;
351     float psi_rad;
352     float p;
353     float q;
354     float r;
355     float p_rad;
356     float q_rad;
357     float r_rad;
358     int16_t throttle;
359     float x;
360     float y;
361     float z;
362     float vx;
363     float vy;
364     float vz;
365     float plan_x;
366     float plan_y;
367     float plan_z;
368     float plan_psi;
369 };
370 struct _PIDpram
371 {
372     float Kp;
373     float Ki;
374     float Kd;
375 };
376 _PIDpram pvel, qvel, rvel, phiang, thetaang,
377 psiang, Vzalt, Zalt, Xpos, Ypos, Vxpos, Vypos;
378 _Target Target;
379 uint8_t Ctrl_timer1 = 0;
380 float EstimatedG;
381 void ControlReset();
382 void TranslationControl();
383 void AttitudeControl();
384 void AltitudeControl();

```

```

385     void ZControl();
386     void VzControl();
387     void AngularRateControl();
388     struct _inertia
389     {
390         uint8_t timer_start = 0;
391         unsigned long time_start = 0, time_end = 0, time_count = 0;
392         uint16_t pendulum = 0;
393         float memo = -1000.0;
394         uint8_t xbee_timer = 0;
395     };
396     _inertia inertia;
397     /*System.cpp*/
398     uint16_t run_period;
399     unsigned short time_out = 0;
400     unsigned short battery_warning = 0;
401     void InitControl();
402     void Moment_Check();
403     void Otus_Clear();
404     void Loop_Check();
405     void Timer_Check();
406     void Battery_Check();
407     float Rad(float angle);
408     float Degree(float rad);
409     float data_limitation(float a, float b, float c);
410     float invSqrt(float number);
411     float voltage;
412     float voltageavg;
413
414     /*-----*/
415 private:
416     /*-----*/
417     /*Copter.cpp*/
418
419     /*Motor.cpp*/
420     float PWM1, PWM2, PWM3, PWM4;
421     double omega12, omega22, omega32,
422     omega42, omega1, omega2, omega3, omega4;
423     float pwm_factor = 65535.0 / 2500.0;
424     float PWM;
425     double InputK1;
426     double InputK2;
427     double InputK3;
428     double InputK4;
429
430     /*System.cpp*/
431     unsigned long whole_timer;
432
433     /*-----*/
434 };

```

Mark3 Flight Controller Code - State Estimation

```

1 #include "Copter.h"
2 /*
3 MatrixXf Pos_F(9, 9);
4 MatrixXf Pos_B(9, 9);
5 MatrixXf Pos_H(9, 9);
6 MatrixXf Pos_Q(9, 9);
7 MatrixXf Pos_R(9, 9);
8 MatrixXf Pos_S(9, 9);
9 MatrixXf Pos_K(9, 9);
10 MatrixXf Pos_P(9, 9);
11 MatrixXf Pos_I(9, 9);
12 MatrixXf Pos_State(9, 1);
13 MatrixXf Pos_Y(9, 1);

```

```

14  MatrixXf Pos_U(9, 1);
15  MatrixXf Pos_Z(9, 1);
16  */
17  void _Copter::PEST()
18  {
19      if (Ctrl_timer1 >= 4)
20      {
21          float OtusNorm = invSqrt(Otus.q0 * Otus.q0 + Otus.q1
22 * Otus.q1 + Otus.q2 * Otus.q2 + Otus.q3 * Otus.q3);
23          Otus.q0 *= OtusNorm;
24          Otus.q1 *= OtusNorm;
25          Otus.q2 *= OtusNorm;
26          Otus.q3 *= OtusNorm;
27
28
29          MatrixXf Otus_R(3, 3);
30          MatrixXf Acc_B(3, 1);
31          MatrixXf Acc_I(3, 1);
32
33          float q0_2, q1_2, q2_2, q3_2, q1q2,
34          q0q3, q1q3, q0q2, q2q3, q0q1;
35
36          q0_2 = Otus.q0 * Otus.q0;
37          q1_2 = Otus.q1 * Otus.q1;
38          q2_2 = Otus.q2 * Otus.q2;
39          q3_2 = Otus.q3 * Otus.q3;
40
41          q1q2 = Otus.q1 * Otus.q2;
42          q0q3 = Otus.q0 * Otus.q3;
43          q1q3 = Otus.q1 * Otus.q3;
44          q0q2 = Otus.q0 * Otus.q2;
45          q2q3 = Otus.q2 * Otus.q3;
46          q0q1 = Otus.q0 * Otus.q1;
47
48          Otus_R << q0_2 + q1_2 - q2_2 - q3_2, 2 *
49          (q1q2 - q0q3), 2 * (q1q3 + q0q2),
50          2 * (q1q2 + q0q3), q0_2 - q1_2 + q2_2 -
51          q3_2, 2 * (q2q3 - q0q1),
52          2 * (q1q3 - q0q2), 2 * (q2q3 + q0q1),
53          q0_2 - q1_2 - q2_2 + q3_2;
54
55          Acc_B << gy86.acc.filter.x / 8192.0 * 9.8 ,
56          gy86.acc.filter.y / 8192.0 * 9.8 ,
57          gy86.acc.filter.z / 8192.0 * 9.8;
58          Acc_I = Otus_R * Acc_B;
59
60          state.ax = Acc_I(0, 0);
61          state.ay = -Acc_I(1, 0);
62          state.az = Acc_I(2, 0) - 9.8;
63
64          float CF_a = 0.8, outerT_2 = outerT * outerT;
65          state.vx = (state.vx + outerT * state.ax)
66 * CF_a + Otus.vx * (1 - CF_a);
67          state.vy = (state.vy + outerT * state.ay)
68 * CF_a + Otus.vy * (1 - CF_a);
69          state.vz = (state.vz + outerT * state.az)
70 * CF_a + Otus.vz * (1 - CF_a);
71
72          state.x = (state.x + outerT * state.vx +
73 0.5 * outerT_2 * state.ax) * CF_a + Otus.x * (1 - CF_a);
74          state.y = (state.y + outerT * state.vy +
75 0.5 * outerT_2 * state.ay) * CF_a + Otus.y * (1 - CF_a);
76          state.z = (state.z + outerT * state.vz +
77 0.5 * outerT_2 * state.az) * CF_a + Otus.z * (1 - CF_a);
78      }
79  }
80  void _Copter:: Pos_Kalman(struct _PosKF *P_tmp)

```

```

81 {
82   P_tmp->tmp_s[0] = P_tmp->a / 20000 + P_tmp->state[0] +
83   P_tmp->state[1] / 100 - P_tmp->state[2] / 20000;
84   P_tmp->tmp_s[1] = P_tmp->a / 100 + P_tmp->state[1] -
85   P_tmp->state[2] / 100;
86   P_tmp->tmp_s[2] = P_tmp->state[2];
87
88   P_tmp->state[0] = P_tmp->tmp_s[0];
89   P_tmp->state[1] = P_tmp->tmp_s[1];
90   P_tmp->state[2] = P_tmp->tmp_s[2];
91   P_tmp->tmp[0][0] = P_tmp->P[0][0] +
92   P_tmp->P[0][1] / 100 -
93   P_tmp->P[0][2] / 20000 + P_tmp->P[1][0] / 100 +
94   P_tmp->P[1][1] / 10000 -
95   P_tmp->P[1][2] / 2000000 - P_tmp->P[2][0] / 20000 -
96   P_tmp->P[2][1] / 2000000 +
97   P_tmp->P[2][2] / 400000000;
98   P_tmp->tmp[0][1] = P_tmp->P[0][1] -
99   P_tmp->P[0][2] / 100 +
100  P_tmp->P[1][1] / 100 - P_tmp->P[1][2] / 10000 -
101  P_tmp->P[2][1] / 20000 + P_tmp->P[2][2] / 2000000;
102  P_tmp->tmp[0][2] = P_tmp->P[0][2] +
103  P_tmp->P[1][2] / 100 -
104  P_tmp->P[2][2] / 20000;
105  P_tmp->tmp[1][0] = P_tmp->P[1][0] +
106  P_tmp->P[1][1] / 100 -
107  P_tmp->P[1][2] / 20000 - P_tmp->P[2][0] / 100 -
108  P_tmp->P[2][1] / 10000 + P_tmp->P[2][2] / 2000000;
109  P_tmp->tmp[1][1] = P_tmp->P[1][1] -
110  P_tmp->P[1][2] / 100 -
111  P_tmp->P[2][1] / 100 + P_tmp->P[2][2] / 10000;
112  P_tmp->tmp[1][2] = P_tmp->P[1][2] -
113  P_tmp->P[2][2] / 100;
114  P_tmp->tmp[2][0] = P_tmp->P[2][0] +
115  P_tmp->P[2][1] / 100 - P_tmp->P[2][2] / 20000;
116  P_tmp->tmp[2][1] = P_tmp->P[2][1] -
117  P_tmp->P[2][2] / 100;
118  P_tmp->tmp[2][2] = P_tmp->P[2][2] + P_tmp->Q;
119
120  for (uint8_t i = 0; i < 3; i++)
121    for (uint8_t j = 0; j < 3; j++)
122      P_tmp->P[i][j] = P_tmp->tmp[i][j];
123  /*-----y = Z - H * state;-----*/
124  P_tmp->y[0] = P_tmp->ps - P_tmp->state[0];
125  P_tmp->y[1] = P_tmp->v - P_tmp->state[1];
126  /*-----S = H*P*H' + R;-----*/
127  P_tmp->S[0][0] = P_tmp->P[0][0] + P_tmp->R0;
128  P_tmp->S[0][1] = P_tmp->P[0][1];
129  P_tmp->S[1][0] = P_tmp->P[1][0];
130  P_tmp->S[1][1] = P_tmp->P[1][1] + P_tmp->R1;
131  /*-----pinv(S)-----*/
132  P_tmp->tmp_s[1] = P_tmp->S[0][0] *
133  P_tmp->S[0][0] + P_tmp->S[1][0] * P_tmp->S[1][0];
134  P_tmp->tmp_s[3] = (P_tmp->S[0][0] *
135  P_tmp->S[0][1]) / P_tmp->tmp_s[1];
136  P_tmp->tmp_s[4] = (P_tmp->S[1][0] *
137  P_tmp->S[1][1]) / P_tmp->tmp_s[1];
138
139  P_tmp->tmp_s[5] = P_tmp->S[0][1] -
140  P_tmp->S[0][0] * (P_tmp->tmp_s[3] + P_tmp->tmp_s[4]);
141  P_tmp->tmp_s[0] = P_tmp->S[1][1] -
142  P_tmp->S[1][0] * (P_tmp->tmp_s[3] + P_tmp->tmp_s[4]);
143  P_tmp->tmp_s[2] = P_tmp->tmp_s[5] *
144  P_tmp->tmp_s[5] + P_tmp->tmp_s[0] * P_tmp->tmp_s[0];

```

```

145 if (P_tmp->tmp_s[1] == 0)
146     P_tmp->tmp_s[1] = 0.0000001;
147 if (P_tmp->tmp_s[2] == 0)
148     P_tmp->tmp_s[2] = 0.0000001;
149 P_tmp->tmp[0][0] = P_tmp->S[0][0] /
150 P_tmp->tmp_s[1] -
151 ((P_tmp->tmp_s[5]) * (P_tmp->tmp_s[3] +
152 P_tmp->tmp_s[4])) / P_tmp->tmp_s[2];
153 P_tmp->tmp[0][1] = P_tmp->S[1][0] / P_tmp->tmp_s[1] -
154 ((P_tmp->tmp_s[0]) * (P_tmp->tmp_s[3] +
155 P_tmp->tmp_s[4])) / P_tmp->tmp_s[2];
156 P_tmp->tmp[1][0] = (P_tmp->tmp_s[5]) /
157 P_tmp->tmp_s[2];
158 P_tmp->tmp[1][1] = (P_tmp->tmp_s[0]) /
159 P_tmp->tmp_s[2];
160 /*-----K = P*H^1*pinv(S);-----*/
161 P_tmp->K[0][0] = P_tmp->P[0][0] *
162 P_tmp->tmp[0][0] + P_tmp->P[0][1] * P_tmp->tmp[1][0];
163 P_tmp->K[0][1] = P_tmp->P[0][0] *
164 P_tmp->tmp[0][1] + P_tmp->P[0][1] * P_tmp->tmp[1][1];
165 P_tmp->K[1][0] = P_tmp->P[1][0] *
166 P_tmp->tmp[0][0] + P_tmp->P[1][1] * P_tmp->tmp[1][0];
167 P_tmp->K[1][1] = P_tmp->P[1][0] *
168 P_tmp->tmp[0][1] + P_tmp->P[1][1] * P_tmp->tmp[1][1];
169 P_tmp->K[2][0] = P_tmp->P[2][0] *
170 P_tmp->tmp[0][0] + P_tmp->P[2][1] * P_tmp->tmp[1][0];
171 P_tmp->K[2][1] = P_tmp->P[2][0] *
172 P_tmp->tmp[0][1] + P_tmp->P[2][1] * P_tmp->tmp[1][1];
173 /*-----state = state + K*y;-----*/
174 P_tmp->tmp_s[0] = P_tmp->state[0];
175 P_tmp->tmp_s[1] = P_tmp->state[1];
176 P_tmp->tmp_s[2] = P_tmp->state[2];
177
178 P_tmp->state[0] = P_tmp->tmp_s[0] +
179 P_tmp->K[0][0] * P_tmp->y[0] +
180 P_tmp->K[0][1] * P_tmp->y[1];
181 P_tmp->state[1] = P_tmp->tmp_s[1] +
182 P_tmp->K[1][0] * P_tmp->y[0] +
183 P_tmp->K[1][1] * P_tmp->y[1];
184 P_tmp->state[2] = P_tmp->tmp_s[2] +
185 P_tmp->K[2][0] * P_tmp->y[0] +
186 P_tmp->K[2][1] * P_tmp->y[1];
187 /*-----state = state + K*y;-----*/
188 for (uint8_t i = 0; i < 3; i++)
189     for (uint8_t j = 0; j < 3; j++)
190         P_tmp->tmp[i][j] = P_tmp->P[i][j];
191
192 P_tmp->P[0][0] = - P_tmp->K[0][1] *
193 P_tmp->tmp[1][0] - P_tmp->tmp[0][0] *
194 (P_tmp->K[0][0] - 1);
195 P_tmp->P[0][1] = - P_tmp->K[0][1] *
196 P_tmp->tmp[1][1] - P_tmp->tmp[0][1] *
197 (P_tmp->K[0][0] - 1);
198 P_tmp->P[0][2] = - P_tmp->K[0][1] *
199 P_tmp->tmp[1][2] - P_tmp->tmp[0][2] *
200 (P_tmp->K[0][0] - 1);
201 P_tmp->P[1][0] = -P_tmp->K[1][0] *
202 P_tmp->tmp[0][0] - P_tmp->tmp[1][0] *
203 (P_tmp->K[1][1] - 1);
204 P_tmp->P[1][1] = -P_tmp->K[1][0] *
205 P_tmp->tmp[0][1] - P_tmp->tmp[1][1] *
206 (P_tmp->K[1][1] - 1);
207 P_tmp->P[1][2] = - P_tmp->K[1][0] *
208 P_tmp->tmp[0][2] - P_tmp->tmp[1][2] *

```

```

209 (P_tmp->K[1][1] - 1);
210 P_tmp->P[2][0] = P_tmp->tmp[2][0] -
211 P_tmp->K[2][0] * P_tmp->tmp[0][0] -
212 P_tmp->K[2][1] * P_tmp->tmp[1][0];
213 P_tmp->P[2][1] = P_tmp->tmp[2][1] -
214 P_tmp->K[2][0] * P_tmp->tmp[0][1] -
215 P_tmp->K[2][1] * P_tmp->tmp[1][1];
216 P_tmp->P[2][2] = P_tmp->tmp[2][2] -
217 P_tmp->K[2][0] * P_tmp->tmp[0][2] -
218 P_tmp->K[2][1] * P_tmp->tmp[1][2];
219 }
220 void _Copter::AHRs_Check()
221 {
222     state.p = IMU.p;
223     state.p_rad = IMU.p_rad;
224     state.q = IMU.q;
225     state.q_rad = IMU.q_rad;
226     state.r = IMU.r;
227     state.r_rad = IMU.r_rad;
228
229     if (Ctrl_timer1 >= 4)
230     {
231         /*-----Phi-----*/
232         if (state.update_phi)
233         {
234             if (abs(IMU.phi - Otus.roll) < 5.0)
235                 state.phi = IMU.phi * 0.75 + Otus.roll * 0.25;
236             else
237                 state.phi = IMU.phi * 0.99 + Otus.roll * 0.01;
238         }
239         else
240             state.phi = IMU.phi;
241         state.phi_rad = Rad(state.phi);
242         /*-----Theta-----*/
243         if (state.update_theta)
244         {
245             if (abs(IMU.theta - Otus.pitch) < 5.0)
246                 state.theta = IMU.theta * 0.75 + Otus.pitch * 0.25;
247             else
248                 state.theta = IMU.theta * 0.99 + Otus.pitch * 0.01;
249         }
250         else
251             state.theta = IMU.theta;
252         state.theta_rad = Rad(state.theta);
253
254         state.phi_sin = sin(state.phi_rad);
255         state.theta_sin = sin(state.theta_rad);
256         state.phi_cos = cos(state.phi_rad);
257         state.theta_cos = cos(state.theta_rad);
258
259         float pre_cos , pre_sin , ob_cos , ob_sin , inc , cof = 0.70;
260         inc = (-state.phi_sin / state.theta_cos * state.q +
261 state.phi_cos / state.theta_cos * state.r) * outerT;
262         pre_cos = cos(Rad(state.psi + inc));
263         pre_sin = sin(Rad(state.psi + inc));
264         if (state.update_psi == 1)
265         {
266             ob_cos = cos(Rad(Otus.yaw));
267             ob_sin = sin(Rad(Otus.yaw));
268         }
269         else
270         {
271             ob_cos = pre_cos;
272             ob_sin = pre_sin;
273         }
274         pre_cos = pre_cos * cof + ob_cos * (1 - cof);
275         pre_sin = pre_sin * cof + ob_sin * (1 - cof);

```

```

276
277     state.psi_rad = atan2(pre_sin , pre_cos);
278     state.psi = Degree(state.psi_rad);
279     state.psi_sin = sin(state.psi_rad);
280     state.psi_cos = cos(state.psi_rad);
281 }
282 }
283 void _Copter::AHRS()
284 {
285     AHRS_filter();           //Digital Filter
286     //Linear_Kalman_Filter();
287     IMU.p = gy86.gyro.filter.x / 32.768;
288     IMU.q = gy86.gyro.filter.y / 32.768;
289     IMU.r = gy86.gyro.filter.z / 32.768;
290     IMU.p_rad = Rad(IMU.p);
291     IMU.q_rad = Rad(IMU.q);
292     IMU.r_rad = Rad(IMU.r);
293     inte_gyro[0] += IMU.p * 0.0025;
294     inte_gyro[1] += IMU.q * 0.0025;
295     inte_gyro[2] += IMU.r * 0.0025;
296     Madgwick_MARG_Update();
297     AHRS_Check();
298 }
299 void _Copter::AHRS_filter()
300 {
301     gy86_Dataanl();
302     gy86.gyro.filter.x =
303     IIR_filter_apply(L3GD20_DEFAULT_FILTER_FREQ,
304     gy86.gyro.aftcal.x, &gyro_IIRx);
305     gy86.gyro.filter.y =
306     IIR_filter_apply(L3GD20_DEFAULT_FILTER_FREQ,
307     -gy86.gyro.aftcal.y, &gyro_IIRy);
308     gy86.gyro.filter.z =
309     IIR_filter_apply(L3GD20_DEFAULT_FILTER_FREQ,
310     -gy86.gyro.aftcal.z, &gyro_IIRz);
311
312     gy86.acc.filter.x =
313     IIR_filter_apply(LSM303D_ACCEL_DEFAULT_DRIVER_FILTER_FREQ,
314     gy86.acc.aftcal.x, &acc_IIRx);
315     gy86.acc.filter.y =
316     IIR_filter_apply(LSM303D_ACCEL_DEFAULT_DRIVER_FILTER_FREQ,
317     gy86.acc.aftcal.y, &acc_IIRy);
318     gy86.acc.filter.z =
319     IIR_filter_apply(LSM303D_ACCEL_DEFAULT_DRIVER_FILTER_FREQ,
320     gy86.acc.aftcal.z, &acc_IIRz);
321 }
322 void _Copter::gy86_Dataanl()
323 {
324     MPU6050read();
325 }
326 void _Copter::AHRS_filter_init()
327 {
328     IIR_set_cutoff_frequency(L3GD20_DEFAULT_RATE,
329     L3GD20_DEFAULT_FILTER_FREQ, &gyro_IIRx);
330     IIR_set_cutoff_frequency(L3GD20_DEFAULT_RATE,
331     L3GD20_DEFAULT_FILTER_FREQ, &gyro_IIRy);
332     IIR_set_cutoff_frequency(L3GD20_DEFAULT_RATE,
333     L3GD20_DEFAULT_FILTER_FREQ, &gyro_IIRz);
334
335     IIR_set_cutoff_frequency(LSM303D_ACCEL_DEFAULT_RATE,
336     LSM303D_ACCEL_DEFAULT_DRIVER_FILTER_FREQ, &acc_IIRx);
337     IIR_set_cutoff_frequency(LSM303D_ACCEL_DEFAULT_RATE,
338     LSM303D_ACCEL_DEFAULT_DRIVER_FILTER_FREQ, &acc_IIRy);
339     IIR_set_cutoff_frequency(LSM303D_ACCEL_DEFAULT_RATE,
340     LSM303D_ACCEL_DEFAULT_DRIVER_FILTER_FREQ, &acc_IIRz);

```



```

341 }
342 void _Copter::IIR_set_cutoff_frequency(float sample_freq,
343 float cutoff_freq, struct _IIR *input_IIR)
344 {
345     if (cutoff_freq <= 0.0f) {
346         // no filtering
347         return;
348     }
349     float fr = sample_freq / cutoff_freq;
350     float ohm = tanf(M_PI_F / fr);
351     float c = 1.0f + 2.0f *
352     cosf(M_PI_F / 4.0f) * ohm + ohm * ohm;
353     input_IIR->b0 = ohm * ohm / c;
354     input_IIR->b1 = 2.0f * input_IIR->b0;
355     input_IIR->b2 = input_IIR->b0;
356     input_IIR->a1 = 2.0f * (ohm * ohm - 1.0f) / c;
357     input_IIR->a2 = (1.0f -
358     2.0f * cosf(M_PI_F / 4.0f) *
359     ohm + ohm * ohm) / c;
360 }
361 float _Copter::IIR_filter_apply(float cutoff_freq,
362 float sample, struct _IIR *input_IIR)
363 {
364     if (cutoff_freq <= 0.0f) {
365         // no filtering
366         return sample;
367     }
368     // do the filtering
369     input_IIR->element0 = sample -
370     input_IIR->element1 *
371     input_IIR->a1 - input_IIR->element2 *
372     input_IIR->a2;
373     float output = input_IIR->element0 *
374     input_IIR->b0 +
375     input_IIR->element1 * input_IIR->b1 +
376     input_IIR->element2 * input_IIR->b2;
377
378     input_IIR->element2 = input_IIR->element1;
379     input_IIR->element1 = input_IIR->element0;
380     return output;
381 }
382 void _Copter::State_Reset()
383 {
384     //q0 = Otus.q0;
385     //q1 = Otus.q1;
386     //q2 = Otus.q2;
387     //q3 = Otus.q3;
388
389     integralFBx = 0.0f;
390     integralFBy = 0.0f;
391     integralFBz = 0.0f;
392
393     state.vx = Otus.vx;
394     state.vy = Otus.vy;
395     state.vz = Otus.vz;
396
397     state.x = Otus.x;
398     state.y = Otus.y;
399     state.z = Otus.z;
400
401     state.psi = Otus.yaw;
402 }
403 void _Copter::KF_init()
404 {
405     PosX.Q = 2 / 10;
406     PosY.Q = 2 / 10;
407     PosZ.Q = 2 / 10;
408     PosX.R0 = 0.01;

```

```

409 PosY.R0 = 0.01;
410 PosZ.R0 = 0.01;
411 PosX.R1 = 0.01;
412 PosY.R1 = 0.01;
413 PosZ.R1 = 0.01;
414 }
415 /*-----Complimentary Filter-----*/
416 // Ref: "Nonlinear Complementary Filters on the Special Orthogonal Group"
417 // By Robert Mahoney, published in 2007
418 void _Copter::MahonyAHRUpdate(float gx, float gy,
419 float gz, float ax, float ay, float az,
420 float mx, float my, float mz) {
421     float recipNorm;
422     float q0q0, q0q1, q0q2, q0q3, q1q1,
423     q1q2, q1q3, q2q2, q2q3, q3q3;
424     float hx, hy, bx, bz;
425     float halfvx, halfvy, halfvz,
426     halfwx, halfwy, halfwz;
427     float halfex, halfey, halfez;
428     float qa, qb, qc;
429
430     if ((mx == 0.0f) && (my == 0.0f)
431         && (mz == 0.0f)) {
432         MahonyAHRUpdateIMU(gx, gy, gz, ax, ay, az);
433         return;
434     }
435     if (!(ax == 0.0f) &&
436         (ay == 0.0f) && (az == 0.0f)) {
437
438         // Normalise accelerometer measurement
439         recipNorm = invSqrt(ax * ax +
440         ay * ay + az * az);
441         ax *= recipNorm;
442         ay *= recipNorm;
443         az *= recipNorm;
444
445         // Normalise magnetometer measurement
446         recipNorm = invSqrt(mx * mx +
447         my * my + mz * mz);
448         mx *= recipNorm;
449         my *= recipNorm;
450         mz *= recipNorm;
451
452         q0q0 = q0 * q0;
453         q0q1 = q0 * q1;
454         q0q2 = q0 * q2;
455         q0q3 = q0 * q3;
456         q1q1 = q1 * q1;
457         q1q2 = q1 * q2;
458         q1q3 = q1 * q3;
459         q2q2 = q2 * q2;
460         q2q3 = q2 * q3;
461         q3q3 = q3 * q3;
462
463         hx = 2.0f * (mx * (0.5f - q2q2 - q3q3) +
464         my * (q1q2 - q0q3) + mz * (q1q3 + q0q2));
465         hy = 2.0f * (mx * (q1q2 + q0q3) +
466         my * (0.5f - q1q1 - q3q3) + mz * (q2q3 - q0q1));
467         bx = sqrt(hx * hx + hy * hy);
468         bz = 2.0f * (mx * (q1q3 - q0q2) +
469         my * (q2q3 + q0q1) + mz * (0.5f - q1q1 - q2q2));
470
471         halfvx = q1q3 - q0q2;
472         halfvy = q0q1 + q2q3;
473         halfvz = q0q0 - 0.5f + q3q3;
474         halfwx = bx * (0.5f - q2q2 - q3q3) +
475         bz * (q1q3 - q0q2);
476         halfwy = bx * (q1q2 - q0q3) +

```

```

477     bz * (q0q1 + q2q3);
478     halfwz = bx * (q0q2 + q1q3) +
479     bz * (0.5 f - q1q1 - q2q2);
480
481     //direction and measured direction of field vectors
482     halfex = (ay * halfvz - az * halfvy) +
483     (my * halfwz - mz * halfwy);
484     halfey = (az * halfvx - ax * halfvz) +
485     (mz * halfwx - mx * halfwz);
486     halfez = (ax * halfvy - ay * halfvx) +
487     (mx * halfwy - my * halfwx);
488
489     if (twoKi > 0.0 f) {
490         integralFBx += twoKi * halfex
491         * (1.0 f / sampleFreq);
492         integralFBy += twoKi * halfey
493         * (1.0 f / sampleFreq);
494         integralFBz += twoKi * halfez
495         * (1.0 f / sampleFreq);
496         gx += integralFBx;
497         gy += integralFBy;
498         gz += integralFBz;
499     }
500     else {
501         integralFBx = 0.0 f;
502         integralFBy = 0.0 f;
503         integralFBz = 0.0 f;
504     }
505
506     // Apply proportional feedback
507     gx += twoKp * halfex;
508     gy += twoKp * halfey;
509     gz += twoKp * halfez;
510 }
511
512 // Integrate rate of change of quaternion
513 gx *= (0.5 f * (1.0 f / sampleFreq));
514 gy *= (0.5 f * (1.0 f / sampleFreq));
515 gz *= (0.5 f * (1.0 f / sampleFreq));
516 qa = q0;
517 qb = q1;
518 qc = q2;
519 q0 += (-qb * gx - qc * gy - q3 * gz);
520 q1 += (qa * gx + qc * gz - q3 * gy);
521 q2 += (qa * gy - qb * gz + q3 * gx);
522 q3 += (qa * gz + qb * gy - qc * gx);
523
524 // Normalise quaternion
525 recipNorm = invSqrt(q0 * q0 +
526 q1 * q1 + q2 * q2 + q3 * q3);
527 q0 *= recipNorm;
528 q1 *= recipNorm;
529 q2 *= recipNorm;
530 q3 *= recipNorm;
531 }
532 void _Copter::MahonyAHRUpdateIMU(float gx, float gy,
533 float gz, float ax, float ay, float az) {
534     float recipNorm;
535     float halfvx, halfvy, halfvz;
536     float halfex, halfey, halfez;
537     float qa, qb, qc;
538
539     // Compute feedback only if accelerometer measurement valid (avoids NaN in accelerometer norm
540     if (!(ax == 0.0 f) && (ay == 0.0 f) && (az == 0.0 f))) {
541
542         // Normalise accelerometer measurement
543         recipNorm = invSqrt(ax * ax + ay * ay + az * az);

```

```

544     ax *= recipNorm;
545     ay *= recipNorm;
546     az *= recipNorm;
547
548
549     halfvx = q1 * q3 - q0 * q2;
550     halfvy = q0 * q1 + q2 * q3;
551     halfvz = q0 * q0 - 0.5 f + q3 * q3;
552
553     halfex = (ay * halfvz - az * halfvy);
554     halfey = (az * halfvx - ax * halfvz);
555     halfez = (ax * halfvy - ay * halfvx);
556
557     // Compute and apply integral feedback if enabled
558     if (twoKi > 0.0f) {
559         integralFBx += twoKi * halfex * (1.0f / sampleFreq);
560         integralFBy += twoKi * halfey * (1.0f / sampleFreq);
561         integralFBz += twoKi * halfez * (1.0f / sampleFreq);
562         gx += integralFBx; // apply integral feedback
563         gy += integralFBy;
564         gz += integralFBz;
565     }
566     else {
567         integralFBx = 0.0f; // prevent integral windup
568         integralFBy = 0.0f;
569         integralFBz = 0.0f;
570     }
571
572     // Apply proportional feedback
573     gx += twoKp * halfex;
574     gy += twoKp * halfey;
575     gz += twoKp * halfez;
576 }
577
578 // Integrate rate of change of quaternion
579 gx *= (0.5 f * (1.0 f / sampleFreq));
580 gy *= (0.5 f * (1.0 f / sampleFreq));
581 gz *= (0.5 f * (1.0 f / sampleFreq));
582 qa = q0;
583 qb = q1;
584 qc = q2;
585 q0 += (-qb * gx - qc * gy - q3 * gz);
586 q1 += (qa * gx + qc * gz - q3 * gy);
587 q2 += (qa * gy - qb * gz + q3 * gx);
588 q3 += (qa * gz + qb * gy - qc * gx);
589
590 // Normalise quaternion
591 recipNorm = invSqrt(q0 * q0 +
592 q1 * q1 + q2 * q2 + q3 * q3);
593 q0 *= recipNorm;
594 q1 *= recipNorm;
595 q2 *= recipNorm;
596 q3 *= recipNorm;
597 }

```

Mark3 Flight Controller Code - Communication Process

```

1 #include "Copter.h"
2 void _Copter::command_Comm()
3 {
4     //Counttimer1 = micros();
5     if (RCsignal.THROTTLE > 250 && RCsignal.THROTTLE < 350
6         && RCsignal.YAW < 350 && RCsignal.YAW > 250) {
7         flag.ARMED = 1;
8     }
9     if (flag.ARMED == 1 && RCsignal.THROTTLE < 350

```

```

10     && RCsignal.YAW > 950) {
11     flag.ARMED = 2;
12     ControlReset();
13 }
14 //Stopping the motors: throttle low and yaw right.
15 if (RCsignal.THROTTLE < 350 && RCsignal.YAW > 1650) {
16     flag.ARMED = 0;
17 }
18 if (RCsignal.MODE < 350 && RCsignal.MODE > 0)
19     flag.mode = Attitude_mode;
20 if (RCsignal.MODE < 1050 && RCsignal.MODE > 950)
21     flag.mode = Altitude_mode ;
22 if (RCsignal.MODE < 2000 && RCsignal.MODE > 1650)
23     flag.mode = loiter_mode;
24 if (RCsignal.SWITCH < 350) flag.turnoff = 0;
25 if (RCsignal.SWITCH > 1650) flag.turnoff = 1;
26 Xbee_comm();
27 }
28 void _Copter::Xbee_comm()
29 {
30     /*
31     if (comorder == 1062)
32     {
33         Serial1.print(IMU.p, 3); //Serial.print("      ");
34         //Serial1.print(IMU.q, 3); Serial.print("      ");
35         //Serial1.print(IMU.r, 3); Serial.print("      ");
36         Serial1.println();
37     }
38     */
39     Xbee_timer++;
40     if (Xbee_timer == 8)
41     {
42
43         //Serial.print(state.x, 4); Serial.print('\t ');
44         //Serial.print(state.y, 4); Serial.print('\t ');
45         //Serial.print(state.z, 4); Serial.print('\t ');
46
47         //Serial.print(Otus.x, 4); Serial.print('\t ');
48         //Serial.print(Otus.y, 4); Serial.print('\t ');
49         //Serial.print(Otus.z, 4); Serial.print('\t ');
50         //Serial.println();
51         /*
52         Serial.print(LKF.acc_roll); Serial.print('\t ');
53         Serial.print(LKF.acc_pitch); Serial.print('\t ');
54         Serial.print(Otus.yaw); Serial.print('\t ');
55
56         Serial.print(state.p); Serial.print('\t ');
57         Serial.print(state.q); Serial.print('\t ');
58         Serial.print(state.r); Serial.print('\t ');
59         Serial.println();
60         */
61         //Serial.println();
62         /*
63         if (xbee_length != 0)
64         {
65             Serial.print(Otus.x); Serial.print('\t ');
66             Serial.print(loopClock2); Serial.print('\t ');
67             Serial.print(xbee_length); Serial.print('\t ');
68             for (uint8_t i = 0; i < xbee_length; i++)
69             {
70                 Serial.print(xbee_data[100 + i]); Serial.print(" ");
71             }
72             Serial.println();
73         }
74         */

```

```

75     switch (comorder)
76     {
77     case 1000:
78         Serial1.print(RCsignal.ROLL);
79         Serial1.print('\t');
80         Serial1.print(RCsignal.PITCH);
81         Serial1.print('\t');
82         Serial1.print(RCsignal.THROTTLE);
83         Serial1.print('\t');
84         Serial1.print(RCsignal.YAW);
85         Serial1.print('\t');
86         Serial1.print(RCsignal.MODE);
87         Serial1.print('\t');
88         Serial1.print(RCsignal.SWITCH);
89         Serial1.print('\t');
90         Serial1.print(RCsignal.CH7);
91         Serial1.print('\t');
92         Serial1.print(RCsignal.CH8);
93         Serial1.print('\t');
94         Serial1.print(RCsignal.CH9);
95         Serial1.print('\t');
96         Serial1.print(RCsignal.CH10);
97         Serial1.print('\t');
98         Serial1.print("Arm: ");
99         Serial1.print(flag.ARMED);
100        Serial1.print('\t');
101        Serial1.print("Turn Off: ");
102        Serial1.print(flag.turnoff); Serial1.print('\t');
103        if (flag.mode == 1)
104        {
105            Serial1.print("Attitude mode");
106            Serial1.print('\t');
107        }
108        else if (flag.mode == 2)
109        {
110            Serial1.print("Altitude mode");
111            Serial1.print('\t');
112        }
113        else if (flag.mode == 3)
114        {
115            Serial1.print("loiter mode");
116            Serial1.print('\t');
117        }
118        Serial1.println();
119        break;
120    case 1010:
121        State_Reset();
122        break;
123    /*-----Gyro Test-----*/
124    case 1020: //Calculate Current Offset
125        flag.calibratedG = 0;
126        comorder = 0;
127        inte_gyro[0] = 0;
128        inte_gyro[1] = 0;
129        inte_gyro[2] = 0;
130        break;
131    case 1021:
132        Serial1.print(gy86.acc.origin.x); Serial1.print('\t');
133        Serial1.print(gy86.acc.origin.y); Serial1.print('\t');
134        Serial1.print(gy86.acc.origin.z); Serial1.print('\t');
135        Serial1.print(mpu.temperature); Serial1.print('\t');
136        Serial1.print(temperature); Serial1.print('\t');
137        Serial1.print(gy86.gyro.origin.x); Serial1.print('\t');
138        Serial1.print(gy86.gyro.origin.y); Serial1.print('\t');
139        Serial1.print(gy86.gyro.origin.z); Serial1.print('\t');

```

```

140     Serial1.println();
141     break;
142 case 1022:
143     Serial1.print(gy86.gyro.origin.x); Serial1.print('\t');
144     Serial1.print(gy86.gyro.aftcal.x); Serial1.print('\t');
145     Serial1.println();
146     break;
147 case 1023:
148     Serial1.print(gy86.gyro.origin.x); Serial1.print('\t');
149     Serial1.print(gy86.gyro.origin.y); Serial1.print('\t');
150     Serial1.print(gy86.gyro.origin.z); Serial1.print('\t');
151     Serial1.println();
152     break;
153 case 1024:
154     Serial1.print(gyro_temp); Serial1.print('\t');
155     Serial1.print(gy86.gyro.aftcal.x); Serial1.print('\t');
156     Serial1.print(gy86.gyro.aftcal.y); Serial1.print('\t');
157     Serial1.print(gy86.gyro.aftcal.z); Serial1.print('\t');
158     Serial1.println();
159     break;
160 case 1025:
161     Serial1.print(gy86.gyro.aftcal.x); Serial1.print('\t');
162     Serial1.print(gy86.gyro.filter.x); Serial1.print('\t');
163     Serial1.println();
164     break;
165 case 1026:
166     Serial1.print(gy86.gyro.origin.z); Serial1.print('\t');
167     Serial1.print(gy86.gyro.aftcal.z); Serial1.print('\t');
168     Serial1.println();
169     break;
170 case 1027:
171     Serial1.print(gy86.gyro.filter.x); Serial1.print('\t');
172     Serial1.print(gy86.gyro.filter.y); Serial1.print('\t');
173     Serial1.print(gy86.gyro.filter.z); Serial1.print('\t');
174     Serial1.println();
175     break;
176 case 1028:
177     Serial1.print(inte_gyro[0]); Serial1.print('\t');
178     Serial1.print(inte_gyro[1]); Serial1.print('\t');
179     Serial1.print(inte_gyro[2]); Serial1.print('\t');
180     Serial1.println();
181     break;
182 /*-----Accelerometer Test-----*/
183 case 1031:
184     Serial1.print(gy86.acc.origin.x); Serial1.print('\t');
185     Serial1.print(gy86.acc.origin.y); Serial1.print('\t');
186     Serial1.print(gy86.acc.origin.z); Serial1.print('\t');
187     Serial1.println();
188     break;
189 case 1032:
190     Serial1.print(gy86.acc.aftcal.x); Serial1.print('\t');
191     Serial1.print(gy86.acc.aftcal.y); Serial1.print('\t');
192     Serial1.print(gy86.acc.aftcal.z); Serial1.print('\t');
193     Serial1.println();
194     break;
195 case 1033:
196     Serial1.print(gy86.acc.aftcal.x); Serial1.print('\t');
197     Serial1.print(gy86.acc.aftcal.y); Serial1.print('\t');
198     Serial1.print(gy86.acc.aftcal.z); Serial1.print('\t');
199     Serial1.print(gy86.acc.filter.x); Serial1.print('\t');
200     Serial1.print(gy86.acc.filter.y); Serial1.print('\t');
201     Serial1.print(gy86.acc.filter.z); Serial1.print('\t');
202     Serial1.println();
203     break;
204 case 1034:
205     Serial1.print(gy86.acc.origin.x); Serial1.print('\t');

```

```

206     Serial1.print(gy86.acc.origin.y); Serial1.print('\t');
207     Serial1.print(gy86.acc.origin.z); Serial1.print('\t');
208     Serial1.print(gy86.acc.aftcal.x); Serial1.print('\t');
209     Serial1.print(gy86.acc.aftcal.y); Serial1.print('\t');
210     Serial1.print(gy86.acc.aftcal.z); Serial1.print('\t');
211     Serial1.println();
212     break;
213     /*-----Accelerometer Calibration-----*/
214     case 1041: //level - z + g
215         MPU6050AccCali(4);
216         break;
217     case 1042: //back - z -g
218         MPU6050AccCali(5);
219         break;
220     case 1043: //nose up - x + g
221         MPU6050AccCali(0);
222         break;
223     case 1044: //nose down - x -g
224         MPU6050AccCali(1);
225         break;
226     case 1045: //left wing up - y +g
227         MPU6050AccCali(2);
228         break;
229     case 1046: //left wing down - y -g
230         MPU6050AccCali(3);
231         break;
232     case 1047: //reset
233         AccPointRead();
234         break;
235     case 1048:
236         for (uint8_t point = 0; point < 6; point++)
237             {
238                 Serial1.print(Acc_Cali.accel_raw_ref[point][0]);
239                 Serial1.print('\t');
240                 Serial1.print(Acc_Cali.accel_raw_ref[point][1]);
241                 Serial1.print('\t');
242                 Serial1.print(Acc_Cali.accel_raw_ref[point][2]);
243                 Serial1.print('\t');
244                 Serial1.println();
245             }
246         comorder = 0;
247         break;
248     case 1049:
249         Serial1.println("-----");
250         for (uint8_t i = 0; i < 3; i++)
251             {
252                 for (uint8_t j = 0; j < 3; j++)
253                     {
254                         Serial1.print(Acc_Cali.a[i][j]);
255                         Serial1.print('\t');
256                     }
257                 Serial1.println();
258             }
259         Serial1.println("-----");
260         for (uint8_t i = 0; i < 3; i++)
261             {
262                 for (uint8_t j = 0; j < 3; j++)
263                     {
264                         Serial1.print(Acc_Cali.T[i][j]);
265                         Serial1.print('\t');
266                     }
267                 Serial1.println();
268             }
269         Serial1.println("-----");
270         comorder = 0;
271         break;

```



```

272  /*Attitude Estimation*/
273  case 1051:
274      Serial1.print(state.phi); Serial1.print('\t');
275      Serial1.print(state.theta); Serial1.print('\t');
276      Serial1.print(state.psi); Serial1.print('\t');
277      Serial1.println();
278      break;
279  case 1052:
280      Serial1.print(state.p); Serial1.print('\t');
281      Serial1.print(state.q); Serial1.print('\t');
282      Serial1.print(state.r); Serial1.print('\t');
283      Serial1.println();
284      break;
285  case 1053:
286      Serial1.print(q0); Serial1.print('\t');
287      Serial1.print(q1); Serial1.print('\t');
288      Serial1.print(q2); Serial1.print('\t');
289      Serial1.print(q3); Serial1.print('\t');
290      Serial1.println();
291      break;
292  case 1054:
293      Serial1.print(state.ax); Serial1.print('\t');
294      Serial1.print(state.ay); Serial1.print('\t');
295      Serial1.print(state.az); Serial1.print('\t');
296      Serial1.println();
297      break;
298  /*-----Linear Kalman Filter-----*/
299  case 1061:
300      Serial1.print(LKF.acc_roll); Serial1.print('\t');
301      Serial1.print(LKF.acc_pitch); Serial1.print('\t');
302      Serial1.println();
303      break;
304  case 1062:
305      Serial1.print(state.x, 4); Serial1.print('\t');
306      Serial1.print(Otus.x, 4); Serial1.print('\t');
307      Serial1.println();
308      break;
309  case 1063:
310      Serial1.print(state.y, 4); Serial1.print('\t');
311      Serial1.print(Otus.y, 4); Serial1.print('\t');
312      Serial1.println();
313      break;
314  case 1064:
315      Serial1.print(state.z, 4); Serial1.print('\t');
316      Serial1.print(Otus.z, 4); Serial1.print('\t');
317      Serial1.println();
318      break;
319  case 1065:
320      Serial1.print(state.vx, 4); Serial1.print('\t');
321      Serial1.print(Otus.vx, 4); Serial1.print('\t');
322      Serial1.println();
323      break;
324  case 1066:
325      Serial1.print(state.vy, 4); Serial1.print('\t');
326      Serial1.print(Otus.vy, 4); Serial1.print('\t');
327      Serial1.println();
328      break;
329  case 1067:
330      Serial1.print(state.vz, 4); Serial1.print('\t');
331      Serial1.print(Otus.vz, 4); Serial1.print('\t');
332      Serial1.println();
333      break;
334  case 1101:
335      Serial1.print(U1); Serial1.print('\t');
336      Serial1.print(U2); Serial1.print('\t');
337      Serial1.print(U3); Serial1.print('\t');
338      Serial1.print(U4); Serial1.print('\t');

```

```

339     Serial1.println("—————");
340     break;
341 case 1102:
342     Serial1.print(PWM1); Serial1.print('\t');
343     Serial1.print(PWM2); Serial1.print('\t');
344     Serial1.print(PWM3); Serial1.print('\t');
345     Serial1.print(PWM4); Serial1.print('\t');
346     Serial1.println();
347     break;
348 /*—————P Control—————*/
349 case 1201:
350     Serial1.print(Target.p); Serial1.print('\t');
351     Serial1.print(state.p); Serial1.print('\t');
352     Serial1.println();
353     break;
354 case 1202:
355     Serial1.print(Target.p_rad, 5); Serial1.print('\t');
356     Serial1.print(state.p_rad, 5); Serial1.print('\t');
357     Serial1.println();
358     break;
359 case 1203:
360     Serial1.print("P.Kp = "); Serial1.println(pvel.Kp, 4);
361     Serial1.print("P.Ki = "); Serial1.println(pvel.Ki, 4);
362     Serial1.print("P.Kd = "); Serial1.println(pvel.Kd, 4);
363     comorder = 0;
364     break;
365 case 1204:
366     Serial1.print(U2, 5); Serial1.print('\t');
367     Serial1.print(state.p_rad, 5);
368     Serial1.println();
369     break;
370 /*—————Roll Control—————*/
371 case 1211:
372     Serial1.print(Target.phi); Serial1.print('\t');
373     Serial1.print(state.phi); Serial1.print('\t');
374     Serial1.println();
375     break;
376 case 1212:
377     Serial1.print(Target.phi_rad, 5); Serial1.print('\t');
378     Serial1.print(state.phi_rad, 5); Serial1.print('\t');
379     Serial1.println();
380     break;
381 case 1213:
382     Serial1.print("Phi.Kp = "); Serial1.println(phiang.Kp, 4);
383     comorder = 0;
384     break;
385
386 /*—————q Control—————*/
387 case 1221:
388     Serial1.print(Target.q); Serial1.print('\t');
389     Serial1.print(state.q); Serial1.print('\t');
390     Serial1.println();
391     break;
392 case 1222:
393     Serial1.print(Target.q_rad, 5); Serial1.print('\t');
394     Serial1.print(state.q_rad, 5); Serial1.print('\t');
395     Serial1.println();
396     break;
397 case 1223:
398     Serial1.print("Q.Kp = "); Serial1.println(qvel.Kp, 4);
399     Serial1.print("Q.Ki = "); Serial1.println(qvel.Ki, 4);
400     Serial1.print("Q.Kd = "); Serial1.println(qvel.Kd, 4);
401     comorder = 0;
402     break;
403 case 1224:
404     Serial1.print(U3, 5); Serial1.print('\t');
405     Serial1.print(state.q_rad, 5);

```

```

406     Serial1.println();
407     break;
408     /*-----Pitch Control-----*/
409     case 1231:
410         Serial1.print(Target.theta); Serial1.print('\t');
411         Serial1.print(state.theta); Serial1.print('\t');
412         Serial1.println();
413         break;
414     case 1232:
415         Serial1.print(Target.theta_rad, 5); Serial1.print('\t');
416         Serial1.print(state.theta_rad, 5); Serial1.print('\t');
417         Serial1.println();
418         break;
419     case 1233:
420         Serial1.print("Theta_Kp = "); Serial1.println(thetaang.Kp, 4);
421         comorder = 0;
422         break;
423     /*-----r Control-----*/
424     case 1241:
425         Serial1.print(Target.r); Serial1.print('\t');
426         Serial1.print(state.r); Serial1.print('\t');
427         Serial1.println();
428         break;
429     case 1242:
430         Serial1.print(Target.r_rad, 5); Serial1.print('\t');
431         Serial1.print(state.r_rad, 5); Serial1.print('\t');
432         Serial1.println();
433         break;
434     case 1243:
435         Serial1.print("R_Kp = "); Serial1.println(rvel.Kp, 6);
436         Serial1.print("R_Ki = "); Serial1.println(rvel.Ki, 6);
437         Serial1.print("R_Kd = "); Serial1.println(rvel.Kd, 6);
438         comorder = 0;
439         break;
440     case 1244:
441         Serial1.print(U4, 5); Serial1.print('\t');
442         Serial1.print(state.r_rad, 5);
443         Serial1.println();
444         break;
445     /*-----Yaw Control-----*/
446     case 1251:
447         Serial1.print(Target.psi); Serial1.print('\t');
448         Serial1.print(state.psi); Serial1.print('\t');
449         //Serial1.print(Psicon.Output[0]); Serial1.print('\t');
450         Serial1.println();
451         break;
452     case 1252:
453         Serial1.print(Target.psi_rad, 5); Serial1.print('\t');
454         Serial1.print(state.psi_rad, 5); Serial1.print('\t');
455         Serial1.println();
456         break;
457     case 1253:
458         Serial1.print("Psi_Kp = ");
459         Serial1.println(psiang.Kp, 4);
460         comorder = 0;
461         break;
462     case 1254:
463         Serial1.println(Otus.yaw);
464         break;
465     /*-----Vz Control-----*/
466     case 1261:
467         Serial1.print(Target.vz); Serial1.print('\t');
468         Serial1.print(state.vz); Serial1.print('\t');
469         Serial1.println();
470         break;
471     case 1262:
472         Serial1.print("Z_Kp = "); Serial1.println(Zalt.Kp);

```

```

473     Serial1.print("Vz.Kp = "); Serial1.println(Vzalt.Kp);
474     Serial1.print("Vz.Ki = "); Serial1.println(Vzalt.Ki);
475     Serial1.print("Vz.Kd = "); Serial1.println(Vzalt.Kd);
476     comorder = 0;
477     break;
478 case 1263:
479     Serial1.print(Vzcon.Output[0], 5); Serial1.print('\t');
480     Serial1.print(state.vz, 5);
481     Serial1.println();
482     break;
483 case 1264:
484     Serial1.print(RCsignal.THROTTLE); Serial1.print('\t');
485     Serial1.print(state.vz); Serial1.print('\t');
486     Serial1.print(Target.vz); Serial1.print('\t');
487     Serial1.print(state.z); Serial1.print('\t');
488     Serial1.print(Target.z); Serial1.print('\t');
489     Serial1.print(Vzcon.Output[0]); Serial1.print('\t');
490     Serial1.print(U1);
491     Serial1.println();
492     break;
493 case 1265:
494     Serial1.print(state.flight_state[0]); Serial1.print('\t');
495     Serial1.print(state.flight_state[1]); Serial1.print('\t');
496     Serial1.print(state.flight_state[2]); Serial1.print('\t');
497     Serial1.print(state.flight_state[3]); Serial1.print('\t');
498     Serial1.print(state.takeoff_t); Serial1.print('\t');
499     Serial1.print(state.landing_t); Serial1.print('\t');
500     Serial1.print(U1); Serial1.print('\t');
501     Serial1.println();
502     break;
503 /*-----VxVy Control-----*/
504 case 1271:
505     Serial1.print("X.Kp = "); Serial1.println(Xpos.Kp);
506     Serial1.print("Y.Kp = "); Serial1.println(Ypos.Kp);
507     Serial1.print("Vx.Kp = "); Serial1.println(Vxpos.Kp);
508     Serial1.print("Vx.Kd = "); Serial1.println(Vxpos.Kd);
509     Serial1.print("Vy.Kp = "); Serial1.println(Vypos.Kp);
510     Serial1.print("Vy.Kd = "); Serial1.println(Vypos.Kd);
511     comorder = 0;
512     break;
513 case 1272:
514     Serial1.print(state.vx); Serial1.print('\t');
515     Serial1.print(Target.vx); Serial1.print('\t');
516     Serial1.println();
517     break;
518 case 1273:
519     Serial1.print(state.vy); Serial1.print('\t');
520     Serial1.print(Target.vy); Serial1.print('\t');
521     Serial1.println();
522     break;
523 case 1274:
524     Serial1.print(state.vz); Serial1.print('\t');
525     Serial1.print(Target.vz); Serial1.print('\t');
526     Serial1.println();
527     break;
528 /*-----Input/Output-----*/
529 case 1280:
530     Serial1.print(state.x); Serial1.print('\t');
531     Serial1.print(Target.x); Serial1.print('\t');
532     Serial1.println();
533     break;
534 case 1281:
535     Serial1.print(state.y); Serial1.print('\t');
536     Serial1.print(Target.y); Serial1.print('\t');
537     Serial1.println();
538     break;

```

```

539     case 1282:
540         Serial1.print(state.z); Serial1.print('\t');
541         Serial1.print(Target.z); Serial1.print('\t');
542         Serial1.println();
543         break;
544     case 1283:
545         Serial1.print(Target.phi); Serial1.print('\t');
546         Serial1.print(Target.theta); Serial1.print('\t');
547         Serial1.println();
548         break;
549
550     /*-----Plan-----*/
551     case 1290:
552         Serial1.print(Target.plan_x); Serial1.print('\t');
553         Serial1.print(Target.plan_y); Serial1.print('\t');
554         Serial1.print(Target.plan_z); Serial1.print('\t');
555         Serial1.print(Target.plan_psi); Serial1.print('\t');
556
557         Serial1.print(Target.x); Serial1.print('\t');
558         Serial1.print(Target.y); Serial1.print('\t');
559         Serial1.print(Target.z); Serial1.print('\t');
560         Serial1.print(Target.psi); Serial1.print('\t');
561         Serial1.println();
562         break;
563     case 1301:
564         Serial1.println();
565         Serial1.println("//////////PID PRAM//////////");
566         Serial1.println();
567         Serial1.print("P_Kp = "); Serial1.println(pvel.Kp);
568         Serial1.print("P_Ki = "); Serial1.println(pvel.Ki);
569         Serial1.print("P_Kd = "); Serial1.println(pvel.Kd);
570         Serial1.print("Q_Kp = "); Serial1.println(qvel.Kp);
571         Serial1.print("Q_Ki = "); Serial1.println(qvel.Ki);
572         Serial1.print("Q_Kd = "); Serial1.println(qvel.Kd);
573         delay(10);
574         Serial1.print("R_Kp = "); Serial1.println(rvel.Kp);
575         Serial1.print("R_Ki = "); Serial1.println(rvel.Ki);
576         Serial1.print("R_Kd = "); Serial1.println(rvel.Kd);
577         Serial1.print("Phi_Kp = "); Serial1.println(phiang.Kp);
578         Serial1.print("Theta_Kp = "); Serial1.println(thetaang.Kp);
579         Serial1.print("Psi_Kp = "); Serial1.println(psiang.Kp);
580         delay(10);
581         Serial1.println();
582         Serial1.print("Z_Kp = "); Serial1.println(Zalt.Kp);
583         Serial1.print("Vz_Kp = "); Serial1.println(Vzalt.Kp);
584         Serial1.print("Vz_Ki = "); Serial1.println(Vzalt.Ki);
585         Serial1.print("Vz_Kd = "); Serial1.println(Vzalt.Kd);
586         Serial1.println();
587         Serial1.print("Xpos_Kp = "); Serial1.println(Xpos.Kp);
588         Serial1.print("Vxpos_Kp = "); Serial1.println(Vxpos.Kp);
589         Serial1.print("Vxpos_Ki = "); Serial1.println(Vxpos.Ki);
590         Serial1.print("Vxpos_Kd = "); Serial1.println(Vxpos.Kd);
591         delay(10);
592         Serial1.print("Ypos_Kp = "); Serial1.println(Ypos.Kp);
593         Serial1.print("Vypos_Kp = "); Serial1.println(Vypos.Kp);
594         Serial1.print("Vypos_Ki = "); Serial1.println(Vypos.Ki);
595         Serial1.print("Vypos_Kd = "); Serial1.println(Vypos.Kd);
596         Serial1.println();
597         Serial1.print("Gravity = "); Serial1.println(EstimatedG);
598         Serial1.println("//////////");
599         Serial1.println();
600         comorder = 0;
601         break;
602     case 1411:
603         Serial1.print(Otus.x); Serial1.print('\t');

```

```

604     Serial1.print(Otus.y); Serial1.print('\t');
605     Serial1.print(Otus.z); Serial1.print('\t');
606     Serial1.println();
607     break;
608 case 1412:
609     Serial1.print(Otus.vx); Serial1.print('\t');
610     Serial1.print(Otus.vy); Serial1.print('\t');
611     Serial1.print(Otus.vz); Serial1.print('\t');
612     Serial1.println();
613     break;
614 case 1413:
615     Serial1.print(Otus.roll); Serial1.print('\t');
616     Serial1.print(Otus.pitch); Serial1.print('\t');
617     Serial1.print(Otus.yaw); Serial1.print('\t');
618     Serial1.println();
619     break;
620 case 1414:
621     Serial1.print(q0); Serial1.print('\t');
622     Serial1.print(q1); Serial1.print('\t');
623     Serial1.print(q2); Serial1.print('\t');
624     Serial1.print(q3); Serial1.print('\t');
625     Serial1.print(Otus.q0); Serial1.print('\t');
626     Serial1.print(Otus.q1); Serial1.print('\t');
627     Serial1.print(Otus.q2); Serial1.print('\t');
628     Serial1.print(Otus.q3); Serial1.print('\t');
629     Serial1.println();
630     break;
631 case 1415:
632
633     Serial.print(LKF.acc_roll); Serial.print('\t');
634     Serial.print(LKF.acc_pitch); Serial.print('\t');
635     Serial.print(Otus.yaw); Serial.print('\t');
636
637     Serial.print(state.p); Serial.print('\t');
638     Serial.print(state.q); Serial.print('\t');
639     Serial.print(state.r); Serial.print('\t');
640     Serial.println();
641
642     break;
643 case 1501:
644     for (uint8_t i = 0; i < 30; i++)
645     {
646         Serial1.print(VRsensor1.ringbuffer[i]);
647         Serial1.println('\t');
648     }
649     comorder = 0;
650     break;
651 case 1998:
652     Serial1.print(xbee_length); Serial1.println('\t');
653     break;
654 case 1999:
655     Xbee_Packet();
656     break;
657 case 2001:
658     Serial1.print(voltageavg); Serial1.println('\t');
659     break;
660
661     }
662     Xbee_timer = 0;
663 }
664 }
665 void _Copter::Xbee_Packet()
666 {
667     /*-----Test-----*/
668     uint8_t data_length = 21;
669     uint8_t packet[data_length];
670     packet[0] = 45;

```

```

671 packet[1] = 50;
672 packet[2] = 55;
673
674 packet[3] = 71;
675 xbee_tmp.f = state.phi;
676 packet[4] = xbee_tmp.ul[0];
677 packet[5] = xbee_tmp.ul[1];
678 packet[6] = xbee_tmp.ul[2];
679 packet[7] = xbee_tmp.ul[3];
680
681 packet[8] = 72;
682 xbee_tmp.f = state.theta;
683 packet[9] = xbee_tmp.ul[0];
684 packet[10] = xbee_tmp.ul[1];
685 packet[11] = xbee_tmp.ul[2];
686 packet[12] = xbee_tmp.ul[3];
687
688 packet[13] = 73;
689 xbee_tmp.f = state.theta;
690 packet[14] = xbee_tmp.ul[0];
691 packet[15] = xbee_tmp.ul[1];
692 packet[16] = xbee_tmp.ul[2];
693 packet[17] = xbee_tmp.ul[3];
694
695 packet[18] = 100;
696 packet[19] = 125;
697 packet[20] = 150;
698 Serial1.write(packet, data_length);
699 }
700 void _Copter::Xbee_receive(int16_t order, uint16_t stmp)
701 {
702     float pram = (float)stmp;
703     switch (order)
704     {
705     case 1011:
706         pvel.Kp = pram / 10000;
707         EEPROM.write(10, stmp & 0b11111111);
708         EEPROM.write(11, stmp >> 8);
709         Serial1.print("P_Kp = "); Serial1.print(pvel.Kp);
710         Serial1.println();
711         break;
712     case 1012:
713         pvel.Ki = pram / 1000;
714         EEPROM.write(12, stmp & 0b11111111);
715         EEPROM.write(13, stmp >> 8);
716         Serial1.print("P_Ki = "); Serial1.print(pvel.Ki);
717         Serial1.println();
718         break;
719     case 1013:
720         pvel.Kd = pram / 100000;
721         EEPROM.write(14, stmp & 0b11111111);
722         EEPROM.write(15, stmp >> 8);
723         Serial1.print("P_Kd = "); Serial1.print(pvel.Kd);
724         Serial1.println();
725         break;
726     case 1021:
727         qvel.Kp = pram / 10000;
728         EEPROM.write(18, stmp & 0b11111111);
729         EEPROM.write(19, stmp >> 8);
730         Serial1.print("Q_Kp = "); Serial1.print(qvel.Kp);
731         Serial1.println();
732         break;
733     case 1022:
734
735         qvel.Ki = pram / 1000;
736         EEPROM.write(20, stmp & 0b11111111);

```

```

737     EEPROM.write(21, stmp >> 8);
738     Serial1.print("Q_Ki = "); Serial1.print(qvel.Ki);
739     Serial1.println();
740     break;
741 case 1023:
742
743     qvel.Kd = pram / 100000;
744     EEPROM.write(22, stmp & 0b11111111);
745     EEPROM.write(23, stmp >> 8);
746     Serial1.print("Q_Kd = "); Serial1.print(qvel.Kd);
747     Serial1.println();
748     break;
749 case 1031:
750
751     rvel.Kp = pram / 100000;
752     EEPROM.write(26, stmp & 0b11111111);
753     EEPROM.write(27, stmp >> 8);
754     Serial1.print("R_Kp = "); Serial1.print(rvel.Kp);
755     Serial1.println();
756     break;
757 case 1032:
758
759     rvel.Ki = pram / 10000;
760     EEPROM.write(28, stmp & 0b11111111);
761     EEPROM.write(29, stmp >> 8);
762     Serial1.print("R_Ki = "); Serial1.print(rvel.Ki);
763     Serial1.println();
764     break;
765 case 1033:
766
767     rvel.Kd = pram / 1000000;
768     EEPROM.write(30, stmp & 0b11111111);
769     EEPROM.write(31, stmp >> 8);
770     Serial1.print("R_Kd = "); Serial1.print(rvel.Kd);
771     Serial1.println();
772     break;
773 case 1051:
774
775     phiang.Kp = pram / 100;
776     EEPROM.write(16, stmp & 0b11111111);
777     EEPROM.write(17, stmp >> 8);
778     Serial1.print("Phi_Kp = "); Serial1.print(phiang.Kp);
779     Serial1.println();
780     break;
781 case 1052:
782
783     thetaang.Kp = pram / 100;
784     EEPROM.write(24, stmp & 0b11111111);
785     EEPROM.write(25, stmp >> 8);
786     Serial1.print("Theta_Kp = "); Serial1.print(thetaang.Kp);
787     Serial1.println();
788     break;
789 case 1053:
790
791     psiang.Kp = pram / 100;
792     EEPROM.write(32, stmp & 0b11111111);
793     EEPROM.write(33, stmp >> 8);
794     Serial1.print("Psi_Kp = "); Serial1.print(psiang.Kp);
795     Serial1.println();
796     break;
797 case 1071:
798
799     Zalt.Kp = pram / 100;
800     EEPROM.write(40, stmp & 0b11111111);
801     EEPROM.write(41, stmp >> 8);
802     Serial1.print("Z_Kp = "); Serial1.print(Zalt.Kp);
803     Serial1.println();

```



```

804     break;
805 case 1081:
806
807     Vzalt.Kp = pram / 100;
808     EEPROM.write(42, stmp & 0b11111111);
809     EEPROM.write(43, stmp >> 8);
810     Serial1.print("Vz_Kp = "); Serial1.print(Vzalt.Kp);
811     Serial1.println();
812     break;
813 case 1082:
814
815     Vzalt.Ki = pram / 100;
816     EEPROM.write(44, stmp & 0b11111111);
817     EEPROM.write(45, stmp >> 8);
818     Serial1.print("Vz_Ki = "); Serial1.print(Vzalt.Ki);
819     Serial1.println();
820     break;
821 case 1083:
822
823     Vzalt.Kd = pram / 100;
824     EEPROM.write(46, stmp & 0b11111111);
825     EEPROM.write(47, stmp >> 8);
826     Serial1.print("Vz_Kd = "); Serial1.print(Vzalt.Kd);
827     Serial1.println();
828     break;
829 case 1091:
830
831     Xpos.Kp = pram / 1000;
832     EEPROM.write(50, stmp & 0b11111111);
833     EEPROM.write(51, stmp >> 8);
834     Serial1.print("Xpos_Kp = "); Serial1.print(Xpos.Kp);
835     Serial1.println();
836     break;
837 case 1092:
838
839     Vxpos.Kp = pram / 1000;
840     EEPROM.write(52, stmp & 0b11111111);
841     EEPROM.write(53, stmp >> 8);
842     Serial1.print("Vxpos_Kp = "); Serial1.print(Vxpos.Kp);
843     Serial1.println();
844     break;
845 case 1093:
846
847     Vxpos.Ki = pram / 1000;
848     EEPROM.write(54, stmp & 0b11111111);
849     EEPROM.write(55, stmp >> 8);
850     Serial1.print("Vxpos_Ki = "); Serial1.print(Vxpos.Ki);
851     Serial1.println();
852     break;
853 case 1094:
854
855     Vxpos.Kd = pram / 1000;
856     EEPROM.write(56, stmp & 0b11111111);
857     EEPROM.write(57, stmp >> 8);
858     Serial1.print("Vxpos_Kd = "); Serial1.print(Vxpos.Kd);
859     Serial1.println();
860     break;
861 case 1095:
862
863     Ypos.Kp = pram / 1000;
864     EEPROM.write(60, stmp & 0b11111111);
865     EEPROM.write(61, stmp >> 8);
866     Serial1.print("Ypos_Kp = "); Serial1.print(Ypos.Kp);
867     Serial1.println();
868     break;
869 case 1096:
870

```

```

871     Vypos.Kp = pram / 1000;
872     EEPROM.write(62, stmp & 0b11111111);
873     EEPROM.write(63, stmp >> 8);
874     Serial1.print("Vypos-Kp = "); Serial1.print(Vypos.Kp);
875     Serial1.println();
876     break;
877 case 1097:
878
879     Vypos.Ki = pram / 1000;
880     EEPROM.write(64, stmp & 0b11111111);
881     EEPROM.write(65, stmp >> 8);
882     Serial1.print("Vypos-Ki = "); Serial1.print(Vypos.Ki);
883     Serial1.println();
884     break;
885 case 1098:
886
887     Vypos.Kd = pram / 1000;
888     EEPROM.write(66, stmp & 0b11111111);
889     EEPROM.write(67, stmp >> 8);
890     Serial1.print("Vypos-Kd = "); Serial1.print(Vypos.Kd);
891     Serial1.println();
892     break;
893 case 1100:
894
895     EstimatedG = pram / 1000;
896     EEPROM.write(70, stmp & 0b11111111);
897     EEPROM.write(71, stmp >> 8);
898     Serial1.print("Gravity = "); Serial1.print(EstimatedG);
899     Serial1.println();
900     break;
901 }
902
903 }

```

Mark3 Flight Controller Code - Library of Interrupt

```

1  #include "Arduino.h"
2  void InitComm();
3  void ISR1();
4  void ISR2();
5  /*-----SBUS-----*/
6  void RC_refine();
7  void OtusSerial();
8  void ComOrder();
9  void PramOrder();
10 void OtusMovingAverage();
11 uint16_t channels[16];
12 uint8_t failSafe;
13 uint16_t lostFrames = 0;
14
15 float moving_ave[13][5], sum_ave[13];
16
17 /*-----*/
18
19 /*-----Lighthouse-----*/
20 #define IRsensor1 23
21 #define IRsensor2 22
22 #define IRsensor3 17
23 #define IRsensor4 16
24 #define IRsensor5 24
25 #define IRsensor6 26
26 #define IRsensor7 27
27 #define IRsensor8 28
28 /*-----*/

```

Mark3 Flight Controller Code - Interrupt

```
1 void InitComm()
2 {
3   Serial1.begin(230400);
4   Serial1.begin(230400);
5   R9DS.begin();
6   pinMode(LED0, OUTPUT);
7   pinMode(LED1, OUTPUT);
8   pinMode(LED2, OUTPUT);
9   digitalWrite(LED0, HIGH);
10  digitalWrite(LED1, LOW);
11  digitalWrite(LED2, LOW);
12  Copter.RCsignal.ROLL = 1000;
13  Copter.RCsignal.PITCH = 1000;
14  Copter.RCsignal.THROTTLE = 250;
15  Copter.RCsignal.YAW = 1000;
16  Copter.RCsignal.MODE = 250;
17  Copter.RCsignal.SWITCH = 1700;
18 }
19 void RC_refine()
20 {
21   if (R9DS.read(channels, &failSafe, &lostFrames))
22   {
23     Copter.RCsignal.ROLL = channels[0];
24     Copter.RCsignal.PITCH = channels[1];
25     Copter.RCsignal.THROTTLE = channels[2];
26     Copter.RCsignal.YAW = channels[3];
27     Copter.RCsignal.MODE = channels[4];
28     Copter.RCsignal.SWITCH = channels[5];
29     Copter.RCsignal.CH7 = channels[6];
30     Copter.RCsignal.CH8 = channels[7];
31     Copter.RCsignal.CH9 = channels[8];
32     Copter.RCsignal.CH10 = channels[9];
33   }
34   if (Copter.Ctrl_timer1 == 2)
35   {
36     Copter.Otus_Clear();
37     String comdata = "";
38     Copter.xbeetime1 = 0;
39     while (Serial1.available() > 0)
40       comdata += char(Serial1.read());
41     Copter.xbee_length = comdata.length();
42     for (int i = 0; i < Copter.xbee_length; i++)
43       Copter.xbee_data[100 + i] = comdata[i];
44     if (Copter.xbee_length == 4)
45     {
46       //Copter.comorder = 0;
47       short com_tmp = 0;
48       for (uint8_t i = 0; i < 4; i++)
49         com_tmp = com_tmp * 10 + (comdata[i] - '0');
50       //ComOrder();
51       /*
52        Serial1.println(uint8_t(comdata[0]));
53        Serial1.println(uint8_t(comdata[1]));
54        Serial1.println(uint8_t(comdata[2]));
55        Serial1.println(uint8_t(comdata[3]));
56        Serial1.println(Copter.xbee_length);
57        Serial1.println(Copter.comorder);
58       */
59       //Serial1.flush();
60       if (com_tmp >= 1000 && com_tmp <= 5000)
61         Copter.comorder = com_tmp;
62     }
```

```

63     else if (Copter.xbee_length < 12
64             && Copter.xbee_length > 5
65             && (Copter.xbee_data[100 + 0] == '&'
66             && Copter.xbee_data[100 + 5] == '_'
67             && Copter.xbee_data[100 + Copter.xbee_length - 1] == '*'))
68         PramOrder();
69     else if (Copter.xbee_length >= 20)
70     {
71         OtusSerial();
72         OtusMovingAverage();
73         //Serial1.println(Copter.xbee_length);
74     }
75 }
76 }
77 void OtusMovingAverage()
78 {
79     Copter.Otus.x =
80         Copter.data_limitation(Copter.Otus.x, -10.0, 10.0);
81     Copter.Otus.y =
82         Copter.data_limitation(Copter.Otus.y, -10.0, 10.0);
83     Copter.Otus.z =
84         Copter.data_limitation(Copter.Otus.z, -5.0, 10.0);
85     Copter.Otus.vx =
86         Copter.data_limitation(Copter.Otus.vx, -50.0, 50.0);
87     Copter.Otus.vy =
88         Copter.data_limitation(Copter.Otus.vy, -50.0, 50.0);
89     Copter.Otus.vz =
90         Copter.data_limitation(Copter.Otus.vz, -50.0, 50.0);
91     Copter.Otus.roll =
92         Copter.data_limitation(Copter.Otus.roll, -180.0, 180.0);
93     Copter.Otus.pitch =
94         Copter.data_limitation(Copter.Otus.pitch, -90.0, 90.0);
95     Copter.Otus.yaw =
96         Copter.data_limitation(Copter.Otus.yaw, -180.0, 180.0);
97     Copter.Otus.q0 =
98         Copter.data_limitation(Copter.Otus.q0, -1.0, 1.0);
99     Copter.Otus.q1 =
100        Copter.data_limitation(Copter.Otus.q1, -1.0, 1.0);
101     Copter.Otus.q2 =
102        Copter.data_limitation(Copter.Otus.q2, -1.0, 1.0);
103     Copter.Otus.q3 =
104        Copter.data_limitation(Copter.Otus.q3, -1.0, 1.0);
105
106     Copter.Target.plan_x =
107        Copter.data_limitation(Copter.Target.plan_x, -1.5, 1.5);
108     Copter.Target.plan_y =
109        Copter.data_limitation(Copter.Target.plan_y, -1.5, 1.5);
110     Copter.Target.plan_z =
111        Copter.data_limitation(Copter.Target.plan_z, 0, 2.2);
112     Copter.Target.plan_psi =
113        Copter.data_limitation(Copter.Target.plan_psi, -180, 180);
114     /*-----Moving Average-----*/
115     for (uint8_t j = 0; j < 13 ; j++)
116         sum_ave[j] = sum_ave[j] - moving_ave[j][4] / 5;
117     for (uint8_t i = 4; i > 0; i--)
118         for (uint8_t j = 0; j < 13 ; j++)
119             moving_ave[j][i] = moving_ave[j][i - 1];
120
121     moving_ave[0][0] = Copter.Otus.x;
122     moving_ave[1][0] = Copter.Otus.y;
123     moving_ave[2][0] = Copter.Otus.z;
124
125     moving_ave[3][0] = Copter.Otus.vx;
126     moving_ave[4][0] = Copter.Otus.vy;
127     moving_ave[5][0] = Copter.Otus.vz;
128

```

```

129 moving_ave[6][0] = Copter.Otus.roll;
130 moving_ave[7][0] = Copter.Otus.pitch;
131 moving_ave[8][0] = Copter.Otus.yaw;
132
133 moving_ave[9][0] = Copter.Otus.q0;
134 moving_ave[10][0] = Copter.Otus.q1;
135 moving_ave[11][0] = Copter.Otus.q2;
136 moving_ave[12][0] = Copter.Otus.q3;
137
138 for (uint8_t j = 0; j < 13 ; j++)
139     sum_ave[j] = sum_ave[j] + moving_ave[j][0] / 5;
140
141 Copter.Otus.x = sum_ave[0];
142 Copter.Otus.y = sum_ave[1];
143 Copter.Otus.z = sum_ave[2];
144
145 Copter.Otus.vx = sum_ave[3];
146 Copter.Otus.vy = sum_ave[4];
147 Copter.Otus.vz = sum_ave[5];
148
149 Copter.Otus.q0 = sum_ave[9];
150 Copter.Otus.q1 = sum_ave[10];
151 Copter.Otus.q2 = sum_ave[11];
152 Copter.Otus.q3 = sum_ave[12];
153 }
154 void OtusSerial()
155 {
156     for (int16_t i = 100 + 3; i < 100 + Copter.xbee.length - 5; i++)
157     {
158         switch (Copter.xbee_data[i])
159         {
160             case 21:
161                 if (Copter.xbee_data[i + 1] == 101)
162                 {
163                     if ((Copter.xbee_data[i - 1] == 131
164                         && Copter.xbee_data[i - 2] == 121
165                         && Copter.xbee_data[i - 3] == 111)
166                         && (Copter.xbee_data[i + 6] == 22
167                             && Copter.xbee_data[i + 7] == 102))
168                     {
169                         Copter.state.update_x = 1;
170                         Copter.otus_tmp.ul =
171                             (Copter.xbee_data[i + 5] << 24)
172                             | (Copter.xbee_data[i + 4] << 16)
173                             | (Copter.xbee_data[i + 3] << 8)
174                             | Copter.xbee_data[i + 2];
175                         Copter.Otus.x = Copter.otus_tmp.f;
176                     }
177                 }
178                 break;
179             case 22:
180                 if (Copter.xbee_data[i + 1] == 102)
181                 {
182                     if ((Copter.xbee_data[i - 6] == 21
183                         && Copter.xbee_data[i - 5] == 101)
184                         && (Copter.xbee_data[i + 6] == 23
185                             && Copter.xbee_data[i + 7] == 103))
186                     {
187                         Copter.state.update_y = 1;
188                         Copter.otus_tmp.ul =
189                             (Copter.xbee_data[i + 5] << 24)
190                             | (Copter.xbee_data[i + 4] << 16)
191                             | (Copter.xbee_data[i + 3] << 8)
192                             | Copter.xbee_data[i + 2];
193                         Copter.Otus.y = Copter.otus_tmp.f;
194                     }

```

```

195     }
196     break;
197 case 23:
198     if (Copter.xbee_data[i + 1] == 103)
199     {
200         if ((Copter.xbee_data[i - 6] == 22
201             && Copter.xbee_data[i - 5] == 102)
202             && (Copter.xbee_data[i + 6] == 24
203                 && Copter.xbee_data[i + 7] == 104))
204         {
205             Copter.state.update_z = 1;
206             Copter.otus_tmp.ul =
207                 (Copter.xbee_data[i + 5] << 24)
208                 | (Copter.xbee_data[i + 4] << 16)
209                 | (Copter.xbee_data[i + 3] << 8)
210                 | Copter.xbee_data[i + 2];
211             Copter.Otus.z = Copter.otus_tmp.f;
212         }
213     }
214     break;
215 case 24:
216     if (Copter.xbee_data[i + 1] == 104)
217     {
218         if ((Copter.xbee_data[i - 6] == 23
219             && Copter.xbee_data[i - 5] == 103)
220             && (Copter.xbee_data[i + 6] == 25
221                 && Copter.xbee_data[i + 7] == 105))
222         {
223             Copter.state.update_vx = 1;
224             Copter.otus_tmp.ul =
225                 (Copter.xbee_data[i + 5] << 24)
226                 | (Copter.xbee_data[i + 4] << 16)
227                 | (Copter.xbee_data[i + 3] << 8)
228                 | Copter.xbee_data[i + 2];
229             Copter.Otus.vx = Copter.otus_tmp.f;
230         }
231     }
232     break;
233 case 25:
234     if (Copter.xbee_data[i + 1] == 105)
235     {
236         if ((Copter.xbee_data[i - 6] == 24
237             && Copter.xbee_data[i - 5] == 104)
238             && (Copter.xbee_data[i + 6] == 26
239                 && Copter.xbee_data[i + 7] == 106))
240         {
241             Copter.state.update_vy = 1;
242             Copter.otus_tmp.ul =
243                 (Copter.xbee_data[i + 5] << 24)
244                 | (Copter.xbee_data[i + 4] << 16)
245                 | (Copter.xbee_data[i + 3] << 8)
246                 | Copter.xbee_data[i + 2];
247             Copter.Otus.vy = Copter.otus_tmp.f;
248         }
249     }
250     break;
251 case 26:
252     if (Copter.xbee_data[i + 1] == 106)
253     {
254         if ((Copter.xbee_data[i - 6] == 25
255             && Copter.xbee_data[i - 5] == 105)
256             && (Copter.xbee_data[i + 6] == 27
257                 && Copter.xbee_data[i + 7] == 107))
258         {
259             Copter.state.update_vz = 1;
260             Copter.otus_tmp.ul =

```

```

261         (Copter.xbee_data[i + 5] << 24)
262         | (Copter.xbee_data[i + 4] << 16)
263         | (Copter.xbee_data[i + 3] << 8)
264         | Copter.xbee_data[i + 2];
265     Copter.Otus.vz = Copter.otus_tmp.f;
266 }
267 }
268 break;
269 case 27:
270     if (Copter.xbee_data[i + 1] == 107)
271     {
272         if ((Copter.xbee_data[i - 6] == 26
273             && Copter.xbee_data[i - 5] == 106)
274             && (Copter.xbee_data[i + 6] == 28
275                 && Copter.xbee_data[i + 7] == 108))
276         {
277             Copter.state.update_psi = 1;
278             Copter.otus_tmp.ul =
279                 (Copter.xbee_data[i + 5] << 24)
280                 | (Copter.xbee_data[i + 4] << 16)
281                 | (Copter.xbee_data[i + 3] << 8)
282                 | Copter.xbee_data[i + 2];
283             Copter.Otus.yaw = Copter.otus_tmp.f;
284
285             Copter.Otus.yaw_sin = sin(Copter.Otus.yaw);
286             Copter.Otus.yaw_cos = cos(Copter.Otus.yaw);
287         }
288     }
289     break;
290 case 28:
291     if (Copter.xbee_data[i + 1] == 108)
292     {
293         if ((Copter.xbee_data[i - 6] == 27
294             && Copter.xbee_data[i - 5] == 107)
295             && (Copter.xbee_data[i + 6] == 29
296                 && Copter.xbee_data[i + 7] == 109))
297         {
298             Copter.state.update_theta = 1;
299             Copter.otus_tmp.ul =
300                 (Copter.xbee_data[i + 5] << 24)
301                 | (Copter.xbee_data[i + 4] << 16)
302                 | (Copter.xbee_data[i + 3] << 8)
303                 | Copter.xbee_data[i + 2];
304             Copter.Otus.pitch = Copter.otus_tmp.f;
305         }
306     }
307     break;
308 case 29:
309     if (Copter.xbee_data[i + 1] == 109)
310     {
311         if ((Copter.xbee_data[i - 6] == 28
312             && Copter.xbee_data[i - 5] == 108)
313             && (Copter.xbee_data[i + 6] == 30
314                 && Copter.xbee_data[i + 7] == 110))
315         {
316             Copter.state.update_phi = 1;
317             Copter.otus_tmp.ul =
318                 (Copter.xbee_data[i + 5] << 24)
319                 | (Copter.xbee_data[i + 4] << 16)
320                 | (Copter.xbee_data[i + 3] << 8)
321                 | Copter.xbee_data[i + 2];
322             Copter.Otus.roll = Copter.otus_tmp.f;
323         }
324     }
325     break;
326 case 31:

```

```

327     if (Copter.xbee_data[i + 1] == 111)
328     {
329         if ((Copter.xbee_data[i - 6] == 29
330             && Copter.xbee_data[i - 5] == 109)
331             && (Copter.xbee_data[i + 6] == 32
332                 && Copter.xbee_data[i + 7] == 112))
333         {
334             Copter.otus_tmp.ul =
335                 (Copter.xbee_data[i + 5] << 24)
336                 | (Copter.xbee_data[i + 4] << 16)
337                 | (Copter.xbee_data[i + 3] << 8)
338                 | Copter.xbee_data[i + 2];
339             Copter.Otus.q0 = Copter.otus_tmp.f;
340             Copter.Otus.q0 *= -1;
341         }
342     }
343     break;
344 case 32:
345     if (Copter.xbee_data[i + 1] == 112)
346     {
347         if ((Copter.xbee_data[i - 6] == 31
348             && Copter.xbee_data[i - 5] == 111)
349             && (Copter.xbee_data[i + 6] == 33
350                 && Copter.xbee_data[i + 7] == 113))
351         {
352             Copter.otus_tmp.ul =
353                 (Copter.xbee_data[i + 5] << 24)
354                 | (Copter.xbee_data[i + 4] << 16)
355                 | (Copter.xbee_data[i + 3] << 8)
356                 | Copter.xbee_data[i + 2];
357             Copter.Otus.q1 = Copter.otus_tmp.f;
358             Copter.Otus.q1 *= -1;
359         }
360     }
361     break;
362 case 33:
363     if (Copter.xbee_data[i + 1] == 113)
364     {
365         if ((Copter.xbee_data[i - 6] == 32
366             && Copter.xbee_data[i - 5] == 112)
367             && (Copter.xbee_data[i + 6] == 34
368                 && Copter.xbee_data[i + 7] == 114))
369         {
370             Copter.otus_tmp.ul =
371                 (Copter.xbee_data[i + 5] << 24)
372                 | (Copter.xbee_data[i + 4] << 16)
373                 | (Copter.xbee_data[i + 3] << 8)
374                 | Copter.xbee_data[i + 2];
375             Copter.Otus.q2 = Copter.otus_tmp.f;
376         }
377     }
378     break;
379 case 34:
380     if (Copter.xbee_data[i + 1] == 114)
381     {
382         if ((Copter.xbee_data[i - 6] == 33
383             && Copter.xbee_data[i - 5] == 113)
384             && (Copter.xbee_data[i + 6] == 35
385                 && Copter.xbee_data[i + 7] == 115))
386         {
387             Copter.otus_tmp.ul =
388                 (Copter.xbee_data[i + 5] << 24)
389                 | (Copter.xbee_data[i + 4] << 16)
390                 | (Copter.xbee_data[i + 3] << 8)
391                 | Copter.xbee_data[i + 2];
392             Copter.Otus.q3 = Copter.otus_tmp.f;

```



```

393     }
394   }
395   break;
396 case 35:
397   if (Copter.xbee_data[i + 1] == 115)
398   {
399     if ((Copter.xbee_data[i - 6] == 34
400         && Copter.xbee_data[i - 5] == 114)
401         && (Copter.xbee_data[i + 6] == 36
402             && Copter.xbee_data[i + 7] == 116))
403     {
404       Copter.otus_tmp.ul =
405         (Copter.xbee_data[i + 5] << 24)
406         | (Copter.xbee_data[i + 4] << 16)
407         | (Copter.xbee_data[i + 3] << 8)
408         | Copter.xbee_data[i + 2];
409       Copter.Target.plan_x = Copter.otus_tmp.f;
410     }
411   }
412   break;
413 case 36:
414   if (Copter.xbee_data[i + 1] == 116)
415   {
416     if ((Copter.xbee_data[i - 6] == 35
417         && Copter.xbee_data[i - 5] == 115)
418         && (Copter.xbee_data[i + 6] == 37
419             && Copter.xbee_data[i + 7] == 117))
420     {
421       Copter.otus_tmp.ul =
422         (Copter.xbee_data[i + 5] << 24)
423         | (Copter.xbee_data[i + 4] << 16)
424         | (Copter.xbee_data[i + 3] << 8)
425         | Copter.xbee_data[i + 2];
426       Copter.Target.plan_y = Copter.otus_tmp.f;
427     }
428   }
429   break;
430 case 37:
431   if (Copter.xbee_data[i + 1] == 117)
432   {
433     if ((Copter.xbee_data[i - 6] == 36
434         && Copter.xbee_data[i - 5] == 116)
435         && (Copter.xbee_data[i + 6] == 38
436             && Copter.xbee_data[i + 7] == 118))
437     {
438       Copter.otus_tmp.ul =
439         (Copter.xbee_data[i + 5] << 24)
440         | (Copter.xbee_data[i + 4] << 16)
441         | (Copter.xbee_data[i + 3] << 8)
442         | Copter.xbee_data[i + 2];
443       Copter.Target.plan_z = Copter.otus_tmp.f;
444     }
445   }
446   break;
447 case 38:
448   if (Copter.xbee_data[i + 1] == 118)
449   {
450     if ((Copter.xbee_data[i - 6] == 37
451         && Copter.xbee_data[i - 5] == 117)
452         && (Copter.xbee_data[i + 6] == 30
453             && Copter.xbee_data[i + 7] == 110))
454     {
455       Copter.otus_tmp.ul =
456         (Copter.xbee_data[i + 5] << 24)
457         | (Copter.xbee_data[i + 4] << 16)
458         | (Copter.xbee_data[i + 3] << 8)

```

```

459         | Copter.xbee_data[i + 2];
460         Copter.Target.plan_psi = Copter.otus_tmp.f;
461     }
462 }
463 break;
464 case 30:
465     if (Copter.xbee_data[i + 1] == 110)
466     {
467         if ((Copter.xbee_data[i - 6] == 38
468             && Copter.xbee_data[i - 5] == 118)
469             && (Copter.xbee_data[i + 4] == 50
470                 && Copter.xbee_data[i + 5] == 75
471                 && Copter.xbee_data[i + 6] == 100))
472         {
473             Copter.state.comm =
474                 (Copter.xbee_data[i + 3] << 8)
475                 | Copter.xbee_data[i + 2];
476             if (Copter.state.comm >= 1000
477                 && Copter.state.comm <= 4000)
478                 Copter.comorder = Copter.state.comm;
479             //Serial.println(Copter.state.comm);
480         }
481     }
482     break;
483 }
484 }
485 }
486 void PramOrder()
487 {
488     int16_t order = 0, num;
489     uint16_t stmp = 0;
490     for (num = 1; num <= 4; num++)
491         order = order * 10 + (Copter.xbee_data[100 + num] - '0');
492     for (num = 6; num < (Copter.xbee_length - 1); num++)
493         stmp = stmp * 10 + (Copter.xbee_data[100 + num] - '0');
494     Copter.Xbee_receive(order, stmp);
495 }

```

Mark3 Flight Controller Code - Motor Driver

```

1  #include "Copter.h"
2  void _Copter::Motor_init()
3  {
4      voltageavg = (float)analogRead(A14) * 0.013841;
5
6      pinMode(motor1, OUTPUT);
7      pinMode(motor2, OUTPUT);
8      pinMode(motor3, OUTPUT);
9      pinMode(motor4, OUTPUT);
10
11     analogWriteFrequency(motor1, 400);
12     analogWriteFrequency(motor2, 400);
13     analogWriteFrequency(motor3, 400);
14     analogWriteFrequency(motor4, 400);
15     analogWriteResolution(16);
16
17     /*-----OutFunction.m-----*/
18     InputK1 = 130958.617;
19     InputK2 = 1290232.68;
20     InputK3 = 1728826.63;
21     InputK4 = 10113268.61;
22     Motor_stop();
23 }
24 void _Copter::InputTransform()
25 {

```

```

26  omega12 = InputK1 * U1 - InputK2 * U2 +
27          InputK3 * U3 + InputK4 * U4;
28  omega22 = InputK1 * U1 + InputK2 * U2 -
29          InputK3 * U3 + InputK4 * U4;
30  omega32 = InputK1 * U1 + InputK2 * U2 +
31          InputK3 * U3 - InputK4 * U4;
32  omega42 = InputK1 * U1 - InputK2 * U2 -
33          InputK3 * U3 - InputK4 * U4;
34  if (omega12 < 0) omega12 = 0;
35  if (omega22 < 0) omega22 = 0;
36  if (omega32 < 0) omega32 = 0;
37  if (omega42 < 0) omega42 = 0;
38  omega1 = sqrt(omega12);
39  omega2 = sqrt(omega22);
40  omega3 = sqrt(omega32);
41  omega4 = sqrt(omega42);
42  MotorModel(omega1, omega2, omega3, omega4);
43 }
44 void _Copter::MotorModel(double omega1,
45                          double omega2, double omega3, double omega4)
46 {
47     double param_a = 1166.0, param_b = 5393,
48           param_c = 299600, param_d = 1544, param_e = 894.5;
49     PWM1 = (omega1 * omega1 + param_b * omega1 + param_c) /
50           (param_a * voltageavg + param_d) + param_e;
51     PWM2 = (omega2 * omega2 + param_b * omega2 + param_c) /
52           (param_a * voltageavg + param_d) + param_e;
53     PWM3 = (omega3 * omega3 + param_b * omega3 + param_c) /
54           (param_a * voltageavg + param_d) + param_e;
55     PWM4 = (omega4 * omega4 + param_b * omega4 + param_c) /
56           (param_a * voltageavg + param_d) + param_e;
57     if (PWM1 < 1055) PWM1 = 1055;
58     if (PWM2 < 1055) PWM2 = 1055;
59     if (PWM3 < 1055) PWM3 = 1055;
60     if (PWM4 < 1055) PWM4 = 1055;
61     if (PWM1 > 1550) PWM1 = 1550;
62     if (PWM2 > 1550) PWM2 = 1550;
63     if (PWM3 > 1550) PWM3 = 1550;
64     if (PWM4 > 1550) PWM4 = 1550;
65     MotorRun();
66 }
67 void _Copter::Motor_stop()
68 {
69     PWM = pwm_factor * 950;
70     analogWrite(motor1, PWM);
71     analogWrite(motor2, PWM);
72     analogWrite(motor3, PWM);
73     analogWrite(motor4, PWM);
74 }
75 void _Copter::MotorRun()
76 {
77     if ((!flag.turnoff) && (flag.ARMED == 2))
78     {
79         float inputpwm1 = pwm_factor * PWM1;
80         float inputpwm2 = pwm_factor * PWM2;
81         float inputpwm3 = pwm_factor * PWM3;
82         float inputpwm4 = pwm_factor * PWM4;
83         analogWrite(motor1, inputpwm1);
84         analogWrite(motor2, inputpwm2);
85         analogWrite(motor3, inputpwm3);
86         analogWrite(motor4, inputpwm4);
87     }
88     else
89         Motor_stop();
90 }

```

Mark3 Flight Controller Code - Attitude Control

```

1  #include "Copter.h"
2  void _Copter::AttitudeControl()
3  {
4      if (Ctrl_timer1 >= 4)
5      {
6          /*-----Roll Command-----*/
7          if (flag.mode == Attitude_mode || flag.mode == Altitude_mode)
8          {
9              if (RCsignal.ROLL < 900)
10             {
11                 //if (RCsignal.CH7 < 1000)
12                 Target.phi = (float)(RCsignal.ROLL - 900) / 40;
13                 //else
14                 //Target.phi = -20;
15             }
16             if (RCsignal.ROLL > 1100)
17             {
18                 //if (RCsignal.CH7 < 1000)
19                 Target.phi = (float)(RCsignal.ROLL - 1100) / 40;
20                 //else
21                 //Target.phi = 20;
22             }
23             if (RCsignal.ROLL >= 900 && RCsignal.ROLL <= 1100)
24                 Target.phi = 0;
25             /*-----Pitch Command-----*/
26             if (RCsignal.PITCH < 900)
27             {
28                 //if (RCsignal.CH7 < 1000)
29                 Target.theta = (float)(RCsignal.PITCH - 900) / 40;
30                 //else
31                 //Target.theta = -15;
32             }
33             if (RCsignal.PITCH > 1100)
34             {
35                 //if (RCsignal.CH7 < 1000)
36                 Target.theta = (float)(RCsignal.PITCH - 1100) / 40;
37                 //else
38                 //Target.theta = 15;
39             }
40             if (RCsignal.PITCH >= 900 && RCsignal.PITCH <= 1100)
41                 Target.theta = 0;
42         }
43
44         Target.phi = data_limitation(Target.phi, -25, 25);
45         Target.theta = data_limitation(Target.theta, -25, 25);
46         Target.phi_rad = Rad(Target.phi);
47         Target.theta_rad = Rad(Target.theta);
48         /*-----Yaw Command-----*/
49         if (RCsignal.THROTTLE > 350)
50         {
51             if (LockYaw != 1)
52             {
53                 LockYaw = 1;
54                 Target.psi = state.psi;
55             }
56         }
57         else {
58             if (U1 < 0.5)
59             {
60                 LockYaw = 0;
61                 Target.psi = state.psi;
62             }

```

```

63     }
64     if ((RCsignal.YAW > 1075) || (RCsignal.YAW < 925))
65     {
66         if (RCsignal.YAW > 1075)
67         {
68             //if (RCsignal.CH7 < 1000)
69             Target.psi += ((RCsignal.YAW - 1075) / 250.0f);
70             //else
71             //Target.psi += 0.5;
72         }
73         if (RCsignal.YAW < 925)
74         {
75             //if (RCsignal.CH7 < 1000)
76             Target.psi += ((RCsignal.YAW - 925) / 250.0f);
77             //else
78             //Target.psi -= 0.5;
79         }
80         if (Target.psi > 180.0f) Target.psi -= 360.0f;
81         else if (Target.psi < -180.0f)Target.psi += 360.0f;
82     }
83
84     if (RCsignal.CH8 > 1000)
85         Target.psi = Target.plan_psi;
86
87     Target.psi_rad = Rad(Target.psi);
88     /*-----P Control-----*/
89     //phi
90     Phicon.Input[0] = Target.phi_rad - state.phi_rad;
91     Phicon.Output[0] = phiang.Kp * Phicon.Input[0];
92
93     //theta
94     Thetacon.Input[0] = Target.theta_rad - state.theta_rad;
95     Thetacon.Output[0] = thetaang.Kp * Thetacon.Input[0];
96
97     //psi
98     if ((Target.psi_rad - state.psi_rad) >= M_PI ||
99         (Target.psi_rad - state.psi_rad) < - M_PI)
100    {
101        if (Target.psi_rad > 0 && state.psi_rad < 0)
102            Psicon.Input[0] = (-M_PI - state.psi_rad) +
103            (Target.psi_rad - M_PI);
104        if (Target.psi_rad < 0 && state.psi_rad > 0)
105            Psicon.Input[0] = (M_PI - state.psi_rad) +
106            (Target.psi_rad + M_PI);
107    }
108    else Psicon.Input[0] = Target.psi_rad - state.psi_rad;
109
110    Psicon.Output[0] = psiang.Kp * Psicon.Input[0];
111 }
112 AngularRateControl();
113 }
114 void _Copter::AngularRateControl()
115 {
116     //p
117     Target.p_rad = Phicon.Output[0];
118     Target.p = Degree(Target.p_rad);
119
120     //q
121     Target.q_rad = Thetacon.Output[0];
122     Target.q = Degree(Target.q_rad);
123     //r
124     /*-----Nonlinear Constrain-----*/
125     Target.r_rad = Psicon.Output[0];
126     Target.r = Degree(Target.r_rad);
127     //p

```

```

128 Pcon.Input[0] = Target.p_rad - state.p_rad;
129 Pcon.Output[0] = Pcon.Input[0] * pvel.Kp +
130 (Pcon.Input[0] - Pcon.Input[1]) * pvel.Kd / innerT;
131 Pcon.Output[1] = Pcon.Output[0];
132 Pcon.Input[1] = Pcon.Input[0];
133 //q
134 Qcon.Input[0] = Target.q_rad - state.q_rad;
135 Qcon.Output[0] = Qcon.Input[0] * qvel.Kp +
136 (Qcon.Input[0] - Qcon.Input[1]) * qvel.Kd / innerT;
137 Qcon.Output[1] = Qcon.Output[0];
138 Qcon.Input[1] = Qcon.Input[0];
139 //r
140 Rcon.Input[0] = Target.r_rad - state.r_rad;
141 Rcon.Output[0] = Rcon.Input[0] * rvel.Kp +
142 (Rcon.Input[0] - Rcon.Input[1]) * rvel.Kd / innerT;
143 Rcon.Output[1] = Rcon.Output[0];
144 Rcon.Input[1] = Rcon.Input[0];
145 U2 = Pcon.Output[0];
146 U3 = Qcon.Output[0];
147 U4 = Rcon.Output[0];
148
149 }
150 void _Copter::ControlReset()
151 {
152     Pcon.Integral = 0;
153     Qcon.Integral = 0;
154     Rcon.Integral = 0;
155 }

```

Mark3 Flight Controller Code - Position Control

```

1 #include "Copter.h"
2 void _Copter::TranslationControl()
3 {
4     if (Ctrl_timer1 >= 4)
5     {
6         if ((flag.mode == loiter.mode))
7         {
8             float tmp_roll, tmp_pitch;
9             if (RCsignal.ROLL < 850)
10                 tmp_roll = ((float)(RCsignal.ROLL) - 850) / 100000;
11             if (RCsignal.ROLL > 1150)
12                 tmp_roll = ((float)(RCsignal.ROLL) - 1150) / 100000;
13             if (RCsignal.PITCH < 850)
14                 tmp_pitch = (850 - (float)(RCsignal.PITCH)) / 100000;
15             if (RCsignal.PITCH > 1150)
16                 tmp_pitch = (1150 - (float)(RCsignal.PITCH)) / 100000;
17             if (RCsignal.ROLL <= 1150 && RCsignal.ROLL >= 850)
18                 tmp_roll = 0;
19             if (RCsignal.PITCH <= 1150 && RCsignal.PITCH >= 850)
20                 tmp_pitch = 0;
21             Target.x += (tmp_pitch * state.psi_cos -
22 tmp_roll * state.psi_sin);
23             Target.y += (tmp_roll * state.psi_cos +
24 tmp_pitch * state.psi_sin);
25             Target.x = data_limitation(Target.x, -2.00, 2.00);
26             Target.y = data_limitation(Target.y, -2.00, 2.00);
27
28             if (RCsignal.CH8 > 1000)
29             {
30                 Target.x = Target.plan_x;
31                 Target.y = Target.plan_y;
32             }
33             //X Control;

```

```

34     Xcon.Input[0] = Target.x - state.x;
35     Xcon.Output[0] = Xpos.Kp * Xcon.Input[0];
36     Target.vx = Xcon.Output[0];
37     //Y Control;
38     Ycon.Input[0] = Target.y - state.y;
39     Ycon.Output[0] = Ypos.Kp * Ycon.Input[0];
40     Target.vy = Ycon.Output[0];
41     //Vx Control
42     Vxcon.Input[0] = (Target.vx - state.vx) / 57.3;
43     Vxcon.Output[0] = Vxcon.Input[0] * Vxpos.Kp +
44     (Vxcon.Input[0] - Vxcon.Input[1]) * Vxpos.Kd / outerT;
45     Vxcon.Output[1] = Vxcon.Output[0];
46     //Vy Control
47     Vycon.Input[0] = (Target.vy - state.vy) / 57.3;
48     Vycon.Output[0] = Vycon.Input[0] * Vypos.Kp +
49     (Vycon.Input[0] - Vycon.Input[1]) * Vypos.Kd / outerT;
50     Vycon.Output[1] = Vycon.Output[0];
51
52     float tmp_a, tmp_b, tmp_c1, tmp_c2, tmp_x1, tmp_x2;
53     tmp_a = state.psi_cos;
54     tmp_b = state.psi_sin;
55     tmp_c1 = Vxcon.Output[0] * 0.647 * (U1 * 0.9);
56     tmp_c2 = Vycon.Output[0] * 0.647 * (U1 * 0.9);
57     //Target Phi
58     Target.sin_phi = tmp_a * tmp_c2 - tmp_b * tmp_c1;
59     Target.sin_phi = data_limitation(Target.sin_phi, -1, 1);
60     Target.phi_rad = asin(Target.sin_phi);
61
62     Target.phi = Degree(Target.phi_rad);
63     Target.phi = data_limitation(Target.phi, -15, 15);
64     Target.phi_rad = Rad(Target.phi);
65     //Target Theta
66     Target.cos_phi = cos(Target.phi_rad);
67     Target.sin_theta = -(tmp_a * tmp_c1 + tmp_b * tmp_c2)
68     / Target.cos_phi;
69     Target.sin_theta = data_limitation(Target.sin_theta, -1, 1);
70     Target.theta_rad = asin(Target.sin_theta);
71
72     Target.theta = Degree(Target.theta_rad);
73     Target.theta = data_limitation(Target.theta, -15, 15);
74     Target.theta_rad = Rad(Target.theta);
75 }
76 }
77 }
78 void _Copter::AltitudeControl()
79 {
80     if (Ctrl_timer1 >= 4)
81     {
82         if ((flag.mode == Altitude_mode) || (flag.mode == loiter_mode))
83         {
84             if (RCsignal.THROTTLE < 850)
85             {
86                 if (RCsignal.CH7 < 1000)
87                     Target.z += ((float)(RCsignal.THROTTLE) - 850) / 100000;
88                 else
89                     Target.z -= 0.003;
90             }
91             if (RCsignal.THROTTLE > 1150)
92             {
93                 if (RCsignal.CH7 < 1000)
94                     Target.z += ((float)(RCsignal.THROTTLE) - 1150) / 100000;
95                 else
96                     Target.z += 0.003;
97             }
98             Target.z = data_limitation(Target.z, 0.20, 2.00);

```

```

99     }
100     if (RCsignal.CH8 > 1000)
101         Target.z = Target.plan_z;
102     ZControl();
103     VzControl();
104 }
105 float StickThrust = 0.008193 * RCsignal.THROTTLE - 2.458;
106 if (flag.mode == Attitude_mode)
107 {
108     U1 = StickThrust / (state.theta_cos * state.phi_cos);
109     Target.z = 0;
110     state.flight_state[0] = 1; //stand by
111     state.flight_state[1] = 0; //take off
112     state.flight_state[2] = 0; //flight
113     state.flight_state[3] = 0; //land
114     state.takeoff_t = 0;
115     state.landing_t = 0;
116 }
117 else if ((flag.mode == Altitude_mode) ||
118 (flag.mode == loiter_mode))
119 {
120     uint8_t i, fliorder;
121     float tmp;
122     for (i = 0; i < 4; i++)
123         if (state.flight_state[i])
124             fliorder = (i + 1) * 10;
125     //Serial1.print(fliorder); Serial1.print('\t ');
126     //Serial1.print(state.takeoff_t); Serial1.print('\t ');
127     //Serial1.println();
128     switch (fliorder)
129     {
130     case 10: //-----stand by
131         U1 = 0;
132         Target.x = state.x;
133         Target.y = state.y;
134         if (RCsignal.THROTTLE > 850 && RCsignal.THROTTLE
135 < 1150 && voltageavg >= 10.5)
136         {
137             state.takeoff_t = state.takeoff_t + 2;
138             if (state.takeoff_t > 1000)
139             {
140                 state.takeoff_t = 1000;
141                 state.flight_state[0] = 0; //stand by
142                 state.flight_state[1] = 1; //take off
143                 state.flight_state[2] = 0; //flight
144                 state.flight_state[3] = 0; //land
145             }
146         }
147     else if (RCsignal.THROTTLE < 350)
148         state.takeoff_t = 0;
149     break;
150     case 20: //-----take off
151         if (state.takeoff_t > 0)
152         {
153             state.takeoff_t = state.takeoff_t - 2;
154             if (state.takeoff_t < 0)
155                 state.takeoff_t = 0;
156         }
157     else
158     {
159         state.flight_state[0] = 0; //stand by
160         state.flight_state[1] = 0; //take off
161         state.flight_state[2] = 1; //flight
162         state.flight_state[3] = 0; //land
163         Target.z = 0.35;
164     }

```



```

165     U1 = float(1000 - state.takeoff_t) / 1000 * 6.6;
166     if (RCsignal.THROTTLE < 350)
167     {
168         state.flight_state[0] = 1; //stand by
169         state.flight_state[1] = 0; //take off
170         state.flight_state[2] = 0; //flight
171         state.flight_state[3] = 0; //land
172         U1 = 0;
173     }
174     break;
175     case 30: //-----flight
176         U1 = (EstimatedG + Vzcon.Output[0]) /
177         (state.phi_cos * state.theta_cos);
178         if (voltageavg <= 10.5)
179         {
180             Target.z = 0.20;
181             if (state.z <= 0.25)
182             {
183                 state.landing_t = 2000;
184                 state.flight_state[0] = 0; //stand by
185                 state.flight_state[1] = 0; //take off
186                 state.flight_state[2] = 0; //flight
187                 state.flight_state[3] = 1; //land
188                 state.tmp_U1 = U1;
189             }
190         }
191         if (RCsignal.THROTTLE < 350)
192         {
193             state.landing_t = state.landing_t + 3;
194             if (state.landing_t > 2000)
195                 state.landing_t = 2000;
196             if (state.landing_t == 2000 && state.z <= 0.25)
197             {
198                 state.flight_state[0] = 0; //stand by
199                 state.flight_state[1] = 0; //take off
200                 state.flight_state[2] = 0; //flight
201                 state.flight_state[3] = 1; //land
202                 state.tmp_U1 = U1;
203             }
204         }
205         else if (RCsignal.THROTTLE > 850)
206             state.landing_t = 0;
207         break;
208     case 40: //-----landing
209         state.landing_t--;
210         tmp = float(state.landing_t) / 2000;
211         U1 = tmp * tmp * state.tmp_U1;
212         if (state.landing_t == 0)
213         {
214             state.flight_state[0] = 1; //stand by
215             state.flight_state[1] = 0; //take off
216             state.flight_state[2] = 0; //flight
217             state.flight_state[3] = 0; //land
218             Target.z = 0;
219         }
220         break;
221     }
222 }
223 }
224 void _Copter::ZControl()
225 {
226     Zcon.Input[0] = Target.z - state.z;
227     Zcon.Output[0] = Zalt.Kp * Zcon.Input[0];
228     Target.vz = Zcon.Output[0];
229 }
230 }

```

```

231 void _Copter::VzControl()
232 {
233     if (state.z < 0.35 || voltageavg < 10.5)
234         Target.vz = data_limitation(Target.vz, -0.2, 0.2);
235     Vzcon.Input[0] = Target.vz - state.vz;
236     Vzcon.Output[0] = Vzcon.Input[0] * Vzalt.Kp +
237     (Vzcon.Input[0] - Vzcon.Input[1]) * Vzalt.Kd / outerT;
238     Vzcon.Output[0] = data_limitation(Vzcon.Output[0], -4.0, 4.0);
239     Vzcon.Input[1] = Vzcon.Input[0];
240 }

```

System.cpp Mark3 Flight Controller Code - Sensor Read

```

1  #include "Copter.h"
2  void _Copter::InitSensor()
3  {
4      Wire.begin();
5      Wire.setRate(I2C_RATE_2000);
6      Wire1.begin();
7      Wire1.setRate(I2C_RATE_2000);
8
9      //gy86
10     I2Cwrite(MPU6050_ADDRESS,
11             MPUREG_PWR_MGMT_1, MPU_CLK_SEL_PLLGYROZ, 0);
12     I2Cwrite(MPU6050_ADDRESS,
13             MPUREG_SMPLRT_DIV, 0x07, 0);
14     I2Cwrite(MPU6050_ADDRESS,
15             MPUREG_CONFIG, BITS_DLPF_CFG_42HZ, 0);
16     I2Cwrite(MPU6050_ADDRESS,
17             MPUREG_GYRO_CONFIG, BITS_FS_1000DPS, 0);
18     I2Cwrite(MPU6050_ADDRESS,
19             MPUREG_ACCEL_CONFIG, 0x08, 0);
20     I2Cwrite(MPU6050_ADDRESS,
21             MPUREG_INT_PIN_CFG, 0x02, 0);
22     I2CRead(MPU6050_ADDRESS,
23            MPUREG_WHOAMI, 1);
24     if (i2cData[0] != 0x68)
25         Serial1.println("Error reading sensor");
26     AccPointRead();
27 }
28 void _Copter::AccPointRead()
29 {
30     uint8_t point;
31     for (point = 0; point < 6; point++)
32     {
33         Acc_Cali.accel_raw_ref[point][0] =
34             (EEPROM.read(100 + 6 * point + 2) << 8)
35             | EEPROM.read(100 + 6 * point + 1);
36         Acc_Cali.accel_raw_ref[point][1] =
37             (EEPROM.read(100 + 6 * point + 4) << 8)
38             | EEPROM.read(100 + 6 * point + 3);
39         Acc_Cali.accel_raw_ref[point][2] =
40             (EEPROM.read(100 + 6 * point + 6) << 8)
41             | EEPROM.read(100 + 6 * point + 5);
42     }
43
44     Acc_Cali.acc_offset[0] =
45         (float)(Acc_Cali.accel_raw_ref[0][0] +
46             Acc_Cali.accel_raw_ref[1][0]) / 2.0;
47     Acc_Cali.acc_offset[1] =
48         (float)(Acc_Cali.accel_raw_ref[2][1] +
49             Acc_Cali.accel_raw_ref[3][1]) / 2.0;
50     Acc_Cali.acc_offset[2] =

```

```

51     (float)(Acc_Cali.accel_raw_ref[4][2] +
52           Acc_Cali.accel_raw_ref[5][2]) / 2.0;
53
54     for (point = 0; point < 3; point++)
55         Acc_Cali.a[0][point] =
56             (float)Acc_Cali.accel_raw_ref[0][point] -
57             Acc_Cali.acc_offset[point];
58     for (point = 0; point < 3; point++)
59         Acc_Cali.a[1][point] =
60             (float)Acc_Cali.accel_raw_ref[2][point] -
61             Acc_Cali.acc_offset[point];
62     for (point = 0; point < 3; point++)
63         Acc_Cali.a[2][point] =
64             (float)Acc_Cali.accel_raw_ref[4][point] -
65             Acc_Cali.acc_offset[point];
66 }
67 void _Copter::MPU6050read()
68 {
69     I2CRead(MPU6050_ADDRESS, MPUREG_ACCEL_XOUT_H, 14);
70     gy86.acc.origin.x = ((i2cData[0] << 8) | i2cData[1]);
71     gy86.acc.origin.y = ((i2cData[2] << 8) | i2cData[3]);
72     gy86.acc.origin.z = ((i2cData[4] << 8) | i2cData[5]);
73     mpu_temperature = (i2cData[6] << 8) | i2cData[7];
74     gy86.gyro.origin.x = (i2cData[8] << 8) | i2cData[9];
75     gy86.gyro.origin.y = ((i2cData[10] << 8) | i2cData[11]);
76     gy86.gyro.origin.z = ((i2cData[12] << 8) | i2cData[13]);
77     MPU6050ThermalCompensation();
78     MPU6050Sixpoint();
79 }
80 void _Copter::MPU6050Sixpoint()
81 {
82     gy86.acc.quietf.x = (float)gy86.acc.origin.x -
83                       Acc_Cali.acc_offset[0];
84     gy86.acc.quietf.y = (float)gy86.acc.origin.y -
85                       Acc_Cali.acc_offset[1];
86     gy86.acc.quietf.z = (float)gy86.acc.origin.z -
87                       Acc_Cali.acc_offset[2];
88
89     gy86.acc.aftcal.x = gy86.acc.quietf.x * Acc_Cali.T[0][0] +
90                       gy86.acc.quietf.y * Acc_Cali.T[1][0] +
91                       gy86.acc.quietf.z *
92                       Acc_Cali.T[2][0];
93     gy86.acc.aftcal.y = gy86.acc.quietf.x * Acc_Cali.T[0][1] +
94                       gy86.acc.quietf.y * Acc_Cali.T[1][1] +
95     gy86.acc.quietf.z *
96                       Acc_Cali.T[2][1];
97     gy86.acc.aftcal.z = gy86.acc.quietf.x * Acc_Cali.T[0][2] +
98                       gy86.acc.quietf.y * Acc_Cali.T[1][2] +
99                       gy86.acc.quietf.z *
100                      Acc_Cali.T[2][2];
101 }
102 void _Copter::MPU6050ThermalCompensation()
103 {
104     float tp3, tp2, tp; //, accx, accy, accz;
105
106     temperature = (float) mpu_temperature / 340.00 + 36.53;
107     tp = temperature;
108     if (tp > 55.0) tp = 55.0;
109     if (tp < 22.5) tp = 22.5;
110     tp2 = tp * tp;
111     tp3 = tp2 * tp;
112
113     /*-----Thermal Calibration-----*/
114     gy86.gyro.tempcp.x = 0.000263 * tp3 -
115                       0.03098 * tp2 + 0.03939 * tp - 29.4;

```

```

116 gy86.gyro.tempcp.y = -0.0004279 * tp3 +
117 0.05322 * tp2 - 2.941 * tp + 80.16;
118 gy86.gyro.tempcp.z = 0.0004163 * tp3 -
119 0.0332 * tp2 + 0.6652 * tp + 23.3;
120
121 gy86.gyro.aftcal.x = (float)gy86.gyro.origin.x -
122 gy86.gyro.tempcp.x;
123 gy86.gyro.aftcal.y = (float)gy86.gyro.origin.y -
124 gy86.gyro.tempcp.y;
125 gy86.gyro.aftcal.z = (float)gy86.gyro.origin.z -
126 gy86.gyro.tempcp.z;
127
128 if (flag.calibratedG == 0 && GyroCaliFlag <= 1500)
129 {
130 GyroCaliFlag++;
131 MPU6050GyroCali();
132 }
133 if (flag.calibratedG == 1)
134 {
135 gy86.gyro.aftcal.x = gy86.gyro.aftcal.x -
136 gy86.gyro.radian.x;
137 gy86.gyro.aftcal.y = gy86.gyro.aftcal.y -
138 gy86.gyro.radian.y;
139 gy86.gyro.aftcal.z = gy86.gyro.aftcal.z -
140 gy86.gyro.radian.z;
141 }
142 }
143 void _Copter::MPU6050AccCali(uint8_t point)
144 {
145 /*Point = 0 1 2 3 4 5*/
146 if (Acc_Cali.acc_calitimer == 100)
147 {
148 Acc_Cali.accel_raw_ref[point][0] =
149 Acc_Cali.acc_calitmpx / 100;
150 Acc_Cali.accel_raw_ref[point][1] =
151 Acc_Cali.acc_calitmpy / 100;
152 Acc_Cali.accel_raw_ref[point][2] =
153 Acc_Cali.acc_calitmpz / 100;
154 Acc_Cali.acc_calitmpx = 0;
155 Acc_Cali.acc_calitmpy = 0;
156 Acc_Cali.acc_calitmpz = 0;
157 Acc_Cali.acc_calitimer = 0;
158 comorder = 0;
159 Serial1.print(Acc_Cali.accel_raw_ref[point][0]);
160 Serial1.print('\t');
161 Serial1.print(Acc_Cali.accel_raw_ref[point][1]);
162 Serial1.print('\t');
163 Serial1.print(Acc_Cali.accel_raw_ref[point][2]);
164 Serial1.print('\t');
165 Serial1.println();
166 EEPROM.write(100 + 6 * point + 1,
167 Acc_Cali.accel_raw_ref[point][0] & 0b11111111);
168 EEPROM.write(100 + 6 * point + 2,
169 Acc_Cali.accel_raw_ref[point][0] >> 8);
170 EEPROM.write(100 + 6 * point + 3,
171 Acc_Cali.accel_raw_ref[point][1] & 0b11111111);
172 EEPROM.write(100 + 6 * point + 4,
173 Acc_Cali.accel_raw_ref[point][1] >> 8);
174 EEPROM.write(100 + 6 * point + 5,
175 Acc_Cali.accel_raw_ref[point][2] & 0b11111111);
176 EEPROM.write(100 + 6 * point + 6,
177 Acc_Cali.accel_raw_ref[point][2] >> 8);
178 }
179 else
180 {
181 Acc_Cali.acc_calitmpx += gy86.acc.origin.x;
182 Acc_Cali.acc_calitmpy += gy86.acc.origin.y;

```

```

183     Acc.Cali.acc_calitmpz += gy86.acc.origin.z;
184     Acc.Cali.acc_calitimer++;
185 }
186 }
187 void _Copter::MPU6050GyroCali()
188 {
189     if (GyroCaliFlag > 400 && GyroCaliFlag < 1001)
190     {
191         GyroCollection[0] += gy86.gyro.aftcal.x;
192         GyroCollection[1] += gy86.gyro.aftcal.y;
193         GyroCollection[2] += gy86.gyro.aftcal.z;
194     }
195     if (GyroCaliFlag == 1001)
196     {
197         gy86.gyro.radian.x = GyroCollection[0] / 600;
198         gy86.gyro.radian.y = GyroCollection[1] / 600;
199         gy86.gyro.radian.z = GyroCollection[2] / 600;
200         Serial1.print(GyroCollection[0]); Serial1.print("\t");
201         Serial1.print(GyroCollection[1]); Serial1.print("\t");
202         Serial1.print(GyroCollection[2]); Serial1.print("\t");
203         GyroCaliFlag = 0;
204         GyroCollection[0] = 0;
205         GyroCollection[1] = 0;
206         GyroCollection[2] = 0;
207         flag.calibratedG = 1;
208         Serial1.println("Gyro Offset Calculated");
209         Serial1.print(gy86.gyro.radian.x); Serial1.print("\t");
210         Serial1.print(gy86.gyro.radian.y); Serial1.print("\t");
211         Serial1.print(gy86.gyro.radian.z); Serial1.print("\t");
212     }
213 }
214 /*-----I2C-----*/
215 uint8_t _Copter::I2Cwrite(uint8_t SENSOR_ADDRESS,
216                          uint8_t SENSOR_REGISTER,
217                          uint8_t SENSOR_VALUE,
218                          bool sendStop)
219 {
220     Wire.beginTransmission(SENSOR_ADDRESS);
221     Wire.write(SENSOR_REGISTER);
222     Wire.write(SENSOR_VALUE); //DEVICE_RESET
223     //Wire.endTransmission();
224     uint8_t rcode = Wire.endTransmission(sendStop);
225     if (rcode) {
226         Serial1.print(F("i2cWrite failed: "));
227         Serial1.println(rcode);
228     }
229     return rcode;
230 }
231 uint8_t _Copter::I2CRead(uint8_t SENSOR_ADDRESS, uint8_t
232                        SENSOR_REGISTER, uint8_t nbytes)
233 {
234     uint32_t timeOutTimer;
235     Wire.beginTransmission(SENSOR_ADDRESS);
236     Wire.write(SENSOR_REGISTER);
237     uint8_t rcode = Wire.endTransmission(false);
238     if (rcode) {
239         Serial1.print(F("i2cRead failed: "));
240         Serial1.println(rcode);
241         return rcode;
242         // See: http://arduino.cc/en/Reference/WireEndTransmission
243     }
244     Wire.requestFrom(SENSOR_ADDRESS, nbytes, (uint8_t) true);
245     for (uint8_t i = 0; i < nbytes; i++) {
246         if (Wire.available())
247             i2cData[i] = Wire.read();

```

```

248     else {
249         timeOutTimer = micros();
250         while (((micros() - timeOutTimer) <
251             I2C_TIMEOUT) && !Wire.available());
252         if (Wire.available())
253             i2cData[i] = Wire.read();
254         else {
255             Serial1.println(F("i2cRead timeout"));
256             return 5;
257         }
258     }
259 }
260 return 0; // Success
261 }

```

Mark3 Flight Controller Code - Sensor Address Library

```

1 //GY86
2 #define MPU6050_ADDRESS          0x68
3 #define MPUREG.WHOAMI            0x75
4 #define MPUREG.SMPLRT_DIV        0x19
5 #define MPUREG.CONFIG            0x1A
6 #define MPUREG.GYRO_CONFIG        0x1B
7 #define MPUREG.ACCEL_CONFIG        0x1C
8 #define MPUREG.FIFO_EN            0x23
9 #define MPUREG.INT_PIN_CFG        0x37
10 #define MPUREG.INT_ENABLE         0x38
11 #define MPUREG.INT_STATUS         0x3A
12 #define MPUREG.ACCEL_XOUT_H        0x3B
13 #define MPUREG.ACCEL_XOUT_L        0x3C
14 #define MPUREG.ACCEL_YOUT_H        0x3D
15 #define MPUREG.ACCEL_YOUT_L        0x3E
16 #define MPUREG.ACCEL_ZOUT_H        0x3F
17 #define MPUREG.ACCEL_ZOUT_L        0x40
18 #define MPUREG.TEMP_OUT_H          0x41
19 #define MPUREG.TEMP_OUT_L          0x42
20 #define MPUREG.GYRO_XOUT_H         0x43
21 #define MPUREG.GYRO_XOUT_L         0x44
22 #define MPUREG.GYRO_YOUT_H         0x45
23 #define MPUREG.GYRO_YOUT_L         0x46
24 #define MPUREG.GYRO_ZOUT_H         0x47
25 #define MPUREG.GYRO_ZOUT_L         0x48
26 #define MPUREG.USER_CTRL           0x6A
27 #define MPUREG.PWR_MGMT_1         0x6B
28 #define MPUREG.PWR_MGMT_2         0x6C
29 #define MPUREG.FIFO_COUNTH         0x72
30 #define MPUREG.FIFO_COUNTL         0x73
31 #define MPUREG.FIFO_R_W           0x74
32 // Configuration bits
33 #define BIT_SLEEP                   0x40
34 #define BIT_H_RESET                 0x80
35 #define BITS_CLKSEL                 0x07
36 #define MPU_CLK_SEL_PLLGYROX        0x01
37 #define MPU_CLK_SEL_PLLGYROZ        0x03
38 #define MPU_EXT_SYNC_GYROX          0x02
39 #define BITS_FS_250DPS              0x00
40 #define BITS_FS_500DPS              0x08
41 #define BITS_FS_1000DPS            0x10
42 #define BITS_FS_2000DPS            0x18
43 #define BITS_FS_MASK                0x18
44 #define BITS_DLPF_CFG_256HZ         0x00
45 //Default settings LPF 256Hz/8000Hz sample
46 #define BITS_DLPF_CFG_188HZ         0x01
47 #define BITS_DLPF_CFG_98HZ          0x02
48 #define BITS_DLPF_CFG_42HZ          0x03
49 #define BITS_DLPF_CFG_20HZ          0x04
50 #define BITS_DLPF_CFG_10HZ          0x05
51 #define BITS_DLPF_CFG_5HZ           0x06

```

```

52 #define BITS_DLPF_CFG_2100HZ_NOLPF 0x07
53 #define BITS_DLPF_CFG_MASK          0x07
54 #define BIT_INT_ANYRD_2CLEAR       0x10
55 #define BIT_RAW_RDY_EN              0x01
56 #define BIT_I2C_IF_DIS              0x10
57 #define BIT_INT_STATUS_DATA        0x01
58
59 //GY89
60 #define L3GD20_ADDRESS              (0xD6 >> 1)
61 #define L3G_WHO_AM_I                0x0F
62
63 #define L3G_CTRL_REG1               0x20
64 #define L3G_CTRL_REG2               0x21
65 #define L3G_CTRL_REG3               0x22
66 #define L3G_CTRL_REG4               0x23
67 #define L3G_CTRL_REG5               0x24
68 #define L3G_REFERENCE               0x25
69 #define L3G_OUT_TEMP                0x26
70 #define L3G_STATUS_REG              0x27
71
72 #define L3G_OUT_X_L                  0x28
73 #define L3G_OUT_X_H                  0x29
74 #define L3G_OUT_Y_L                  0x2A
75 #define L3G_OUT_Y_H                  0x2B
76 #define L3G_OUT_Z_L                  0x2C
77 #define L3G_OUT_Z_H                  0x2D
78
79 #define L3G_FIFO_CTRL_REG            0x2E
80 #define L3G_FIFO_SRC_REG            0x2F
81
82 #define L3G_INT1_CFG                  0x30
83 #define L3G_INT1_SRC                  0x31
84 #define L3G_INT1_THS_XH              0x32
85 #define L3G_INT1_THS_XL              0x33
86 #define L3G_INT1_THS_YH              0x34
87 #define L3G_INT1_THS_YL              0x35
88 #define L3G_INT1_THS_ZH              0x36
89 #define L3G_INT1_THS_ZL              0x37
90 #define L3G_INT1_DURATION             0x38
91
92 #define LSM303D_ADDRESS               0b0011101
93 #define LSM303D_CTRL_REG0            0x1F
94 #define LSM303D_CTRL_REG1            0x20
95 #define LSM303D_CTRL_REG2            0x21
96 #define LSM303D_CTRL_REG3            0x22
97 #define LSM303D_CTRL_REG4            0x23
98 #define LSM303D_CTRL_REG5            0x24
99 #define LSM303D_CTRL_REG6            0x25
100 #define LSM303D_CTRL_REG7            0x26
101 #define LSM303D_OUT_X_L_A             0x28
102
103 #define L3GD20_DEFAULT_FILTER_FREQ    30
104 #define L3GD20_DEFAULT_RATE           400
105
106 #define LSM303D_ACCEL_DEFAULT_RANGE_G 16
107 #define LSM303D_ACCEL_DEFAULT_RATE    400
108 #define LSM303D_ACCEL_DEFAULT_ONCHIP_FILTER_FREQ 50
109 #define LSM303D_ACCEL_DEFAULT_DRIVER_FILTER_FREQ 30

```

Mark3 Flight Controller Code - System Check

```

1 #include "Copter.h"
2 void _Copter::InitControl()
3 {
4     pvel.Kp = (EEPROM.read(11) << 8) | EEPROM.read(10); pvel.Kp /= 10000;
5     pvel.Ki = (EEPROM.read(13) << 8) | EEPROM.read(12); pvel.Ki /= 1000;
6     pvel.Kd = (EEPROM.read(15) << 8) | EEPROM.read(14); pvel.Kd /= 100000;
7     phiang.Kp = (EEPROM.read(17) << 8) | EEPROM.read(16); phiang.Kp /= 100;
8

```

```

9   qvel.Kp = (EEPROM.read(19) << 8) |
10  EEPROM.read(18); qvel.Kp /= 10000;
11  qvel.Ki = (EEPROM.read(21) << 8) |
12  EEPROM.read(20); qvel.Ki /= 1000;
13  qvel.Kd = (EEPROM.read(23) << 8) |
14  EEPROM.read(22); qvel.Kd /= 100000;
15  thetaang.Kp = (EEPROM.read(25) << 8) |
16  EEPROM.read(24); thetaang.Kp /= 100;
17
18  rvel.Kp = (EEPROM.read(27) << 8) |
19  EEPROM.read(26); rvel.Kp /= 100000;
20  rvel.Ki = (EEPROM.read(29) << 8) |
21  EEPROM.read(28); rvel.Ki /= 10000;
22  rvel.Kd = (EEPROM.read(31) << 8) |
23  EEPROM.read(30); rvel.Kd /= 1000000;
24  psiang.Kp = (EEPROM.read(33) << 8) |
25  EEPROM.read(32); psiang.Kp /= 100;
26
27  Zalt.Kp = (EEPROM.read(41) << 8) |
28  EEPROM.read(40); Zalt.Kp /= 100;
29  Vzalt.Kp = (EEPROM.read(43) << 8) |
30  EEPROM.read(42); Vzalt.Kp /= 100;
31  Vzalt.Ki = (EEPROM.read(45) << 8) |
32  EEPROM.read(44); Vzalt.Ki /= 100;
33  Vzalt.Kd = (EEPROM.read(47) << 8) |
34  EEPROM.read(46); Vzalt.Kd /= 100;
35
36  Xpos.Kp = (EEPROM.read(51) << 8) |
37  EEPROM.read(50); Xpos.Kp /= 1000;
38  Vxpos.Kp = (EEPROM.read(53) << 8) |
39  EEPROM.read(52); Vxpos.Kp /= 1000;
40  Vxpos.Ki = (EEPROM.read(55) << 8) |
41  EEPROM.read(54); Vxpos.Ki /= 1000;
42  Vxpos.Kd = (EEPROM.read(57) << 8) |
43  EEPROM.read(56); Vxpos.Kd /= 1000;
44
45  Ypos.Kp = (EEPROM.read(61) << 8) |
46  EEPROM.read(60); Ypos.Kp /= 1000;
47  Vypos.Kp = (EEPROM.read(63) << 8) |
48  EEPROM.read(62); Vypos.Kp /= 1000;
49  Vypos.Ki = (EEPROM.read(65) << 8) |
50  EEPROM.read(64); Vypos.Ki /= 1000;
51  Vypos.Kd = (EEPROM.read(67) << 8) |
52  EEPROM.read(66); Vypos.Kd /= 1000;
53
54  EstimatedG = (EEPROM.read(71) << 8) |
55  EEPROM.read(70); EstimatedG /= 1000;
56
57  state.flight_state[0] = 1; //stand by
58  state.flight_state[1] = 0; //take off
59  state.flight_state[2] = 0; //flight
60  state.flight_state[3] = 0; //land
61  state.takeoff_t = 0;
62  state.landing_t = 0;
63 }
64 void _Copter::Loop_Check()
65 {
66   xbee_length = 0;
67   if (Ctrl_timer1 >= 4)
68     Ctrl_timer1 = 0; //—————Check 100Hz
69   Moment_Check();
70   Battery_Check();
71   Timer_Check();
72 }
73 void _Copter::Otus_Clear()

```



```

74 {
75     state.update_phi = 0;
76     state.update_theta = 0;
77     state.update_psi = 0;
78     state.update_vx = 0;
79     state.update_vy = 0;
80     state.update_vz = 0;
81     state.update_x = 0;
82     state.update_y = 0;
83     state.update_z = 0;
84 }
85 void _Copter::Moment_Check()
86 {
87     float temp_gyro;
88     if (comorder >= 1401 && comorder <= 1403)
89     {
90         if (inertia.timer_start == 0)
91         {
92             inertia.timer_start = 1;
93             inertia.time_start = micros();
94         }
95         inertia.time_end = micros();
96         inertia.time_count = inertia.time_end - inertia.time_start;
97
98         if (gy86.gyro.filter.z > 400 && inertia.memo < -400)
99         {
100             inertia.memo = gy86.gyro.filter.z;
101             inertia.pendulum ++;
102         }
103         if (gy86.gyro.filter.z < - 400 && inertia.memo > 400)
104         {
105             inertia.memo = gy86.gyro.filter.z;
106         }
107         if (inertia.xbee_timer == 20)
108         {
109             Serial1.print(inertia.time_count); Serial1.print(" ");
110             Serial1.print(gy86.gyro.filter.z); Serial1.print(" ");
111             Serial1.print(inertia.memo); Serial1.print(" ");
112             Serial1.println(inertia.pendulum);
113             inertia.xbee_timer = 0;
114         }
115         inertia.xbee_timer++;
116     }
117     else
118     {
119         inertia.timer_start = 0;
120         inertia.time_count = 0;
121         inertia.time_end = 0;
122         inertia.time_start = 0;
123         inertia.pendulum = 0;
124         inertia.memo = -1000;
125     }
126 }
127 void _Copter::Battery_Check()
128 {
129     //voltage = (float)analogRead(A14) * 0.019586; //only for quad 2
130     voltage = (float)analogRead(A14) * 0.013841;
131     voltageavg = voltage * 0.005 + voltageavg * 0.995;
132     voltageavg = data_limitation(voltageavg, 9.0, 17.0);
133     if (voltageavg < 10.5)
134         battery_warning = 1;
135     else
136         battery_warning = 0;
137 }
138 void _Copter::Timer_Check()
139 {
140     if (gltimer == 150)

```

```

141 {
142   if (glch > 0)
143   {
144     digitalWrite(LED2, LOW);
145     if (voltageavg < 10.5)
146       digitalWrite(LED1, HIGH);
147     else
148       digitalWrite(LED1, LOW);
149   }
150   else
151   {
152     digitalWrite(LED2, HIGH);
153   }
154   glch *= -1;
155   gltimer = 0;
156 }
157 else
158   gltimer++;
159 run_period = micros() - whole_timer;
160 if (comorder == 2002)
161   Serial1.println(run_period);
162 if (run_period > MainLoopPeriod)
163 {
164   time_out = 1;
165   Serial1.print(run_period);
166   Serial1.println(" <<<<<time out>>>>>");
167 }
168 loopClock2 = micros() - loopClock1;
169 while (micros() - whole_timer < MainLoopPeriod);
170 whole_timer = micros();
171 }
172
173 float _Copter::Rad(float angle)
174 {
175   return (angle * M_PI / 180.0);
176 }
177 float _Copter::Degree(float rad)
178 {
179   return (rad / M_PI * 180.0);
180 }
181 float _Copter::data_limitation(float a, float b, float c)
182 {
183   if (a < b) a = b;
184   if (a > c) a = c;
185   return a;
186 }
187
188 float _Copter::invSqrt(float number) {
189   long i;
190   float x2, y;
191   const float threehalfs = 1.5F;
192
193   x2 = number * 0.5F;
194   y = number;
195   i = * ( long * ) &y;
196   i = 0x5f3759df - ( i >> 1 );
197   y = * ( float * ) &i;
198   y = y * ( threehalfs - ( x2 * y * y ) );
199   return y;
200 }

```

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 300 Ω | 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 |
|--------------------------------|----------|
| <i>MARK3</i> 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

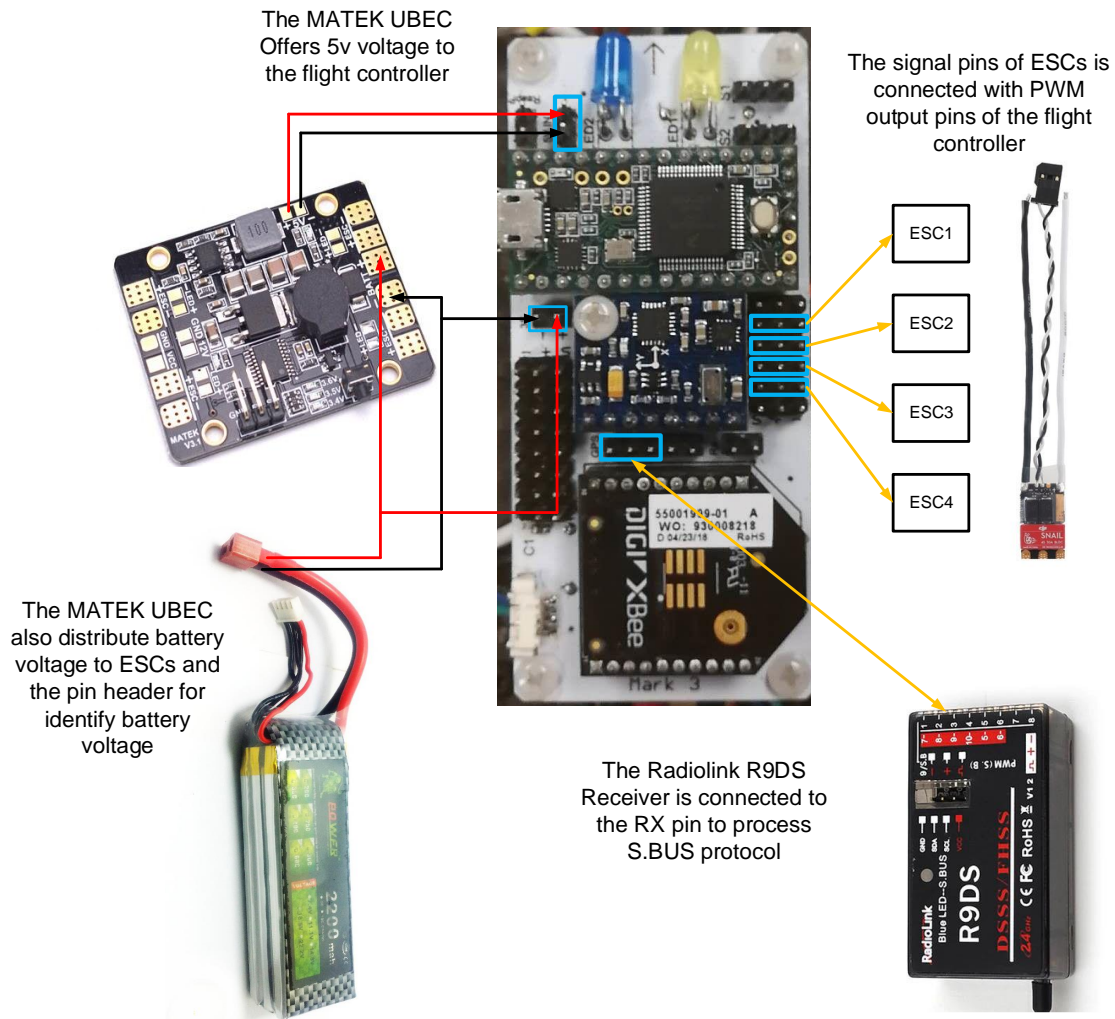


Figure D.1: *MARK3* Flight Controller Wiring Instruction

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)

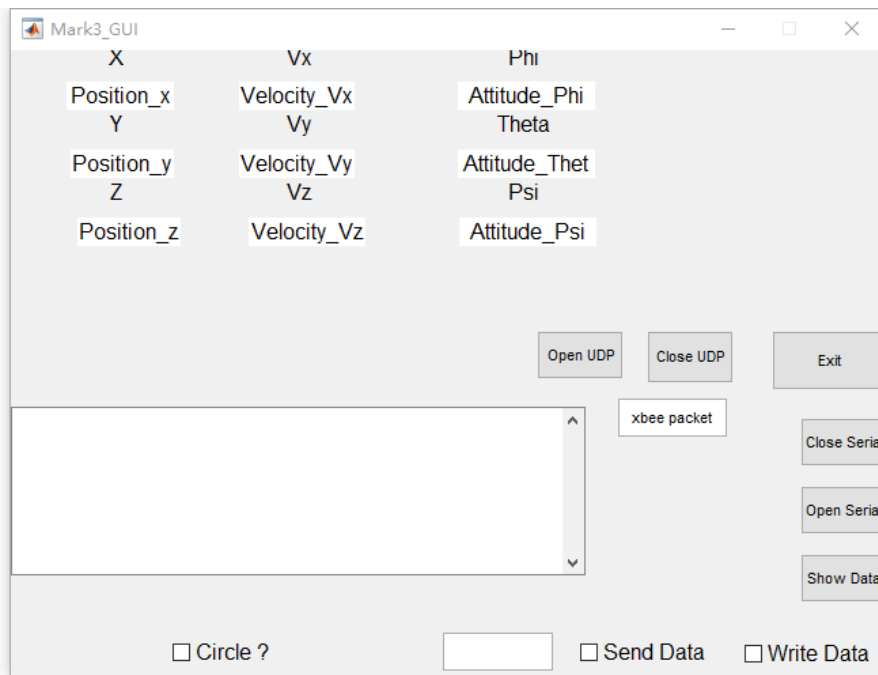


Figure D.2: *MARK3* Ground Station

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.