NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

IMPLEMENTATION OF SIMULINK CONTROLLER DESIGN ON IRIS+ QUADROTOR

by

Wei Zhong Fum

September 2015

Thesis Advisor:	Vladimir Dobrokhodov
Second Reader:	Noel Du Toit

REPORT DOCUMENTATION PAGE		*Form Approved OMB No. 0704–0188*

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** September 2015	**3. REPORT TYPE AND DATES COVERED** Master's Thesis

4. TITLE AND SUBTITLE IMPLEMENTATION OF SIMULINK CONTROLLER DESIGN ON IRIS+ QUADROTOR	**5. FUNDING NUMBERS**
6. AUTHOR(S) Fum, Wei Zhong	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB Protocol number ____N/A____.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited	**12b. DISTRIBUTION CODE**

13. ABSTRACT (maximum 200 words)

The thesis has two primary objectives. First, it develops a high-fidelity 6DOF flight dynamics model of a multi-copter UAV, and uses it for the design and implementation of the linear attitude controller onboard of an industrial quadcopter; both steps are implemented in Simulink. Second, it leverages the weakly joint efforts of MathWorks and the open-source community to build a software setup that enables rapid control software prototyping. This software architecture enables control system design and integration without the need for proficiency in embedded coding that typically utilizes high-level programming languages like C/C++. The higher impact of the dual objective is in advancing methods and tools of verifiable control system design and the embedded code generation that simplifies the V&V process.

The 3DR Iris+ quadrotor, equipped with PX4 "Pixhawk" autopilot, is selected as the primary prototyping platform. The autopilot allows for the real-time execution of an application (attitude controller) that is auto-generated from MatLab/Simulink. This makes the Iris+ quadrotor an ideal platform for rapid flight control prototyping by using MathWork's auto code generation capability.

Ultimately, the developed setup represents a convenient research and development tool that natively bridges the gap between the safety-critical flight control science and flight experimentation technology by "eliminating" the error-prone manual coding of embedded microcontrollers.

14. SUBJECT TERMS quadrotor, 6DOF modeling, rapid control software prototyping, control algorithms	**15. NUMBER OF PAGES** 125
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UU

NSN 7540–01-280-5500 Standard Form 298 (Rev. 2–89) Prescribed by ANSI Std. 239–18

THIS PAGE INTENTIONALLY LEFT BLANK

IMPLEMENTATION OF SIMULINK CONTROLLER DESIGN ON IRIS+ QUADROTOR

Wei Zhong Fum
Civilian, Defence Science and Technology Agency, Singapore
B.Eng, Nanyang Technological University, Singapore, 2008

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN MECHANICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
September 2015

Approved by: Vladimir Dobrokhodov
Thesis Advisor

Noel Du Toit
Second Reader

Garth V. Hobson
Chair, Department of Mechanical and Aerospace Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The thesis has two primary objectives. First, it develops a high-fidelity 6DOF flight dynamics model of a multi-copter UAV, and uses it for the design and implementation of the linear attitude controller onboard of an industrial quadcopter; both steps are implemented in Simulink. Second, it leverages the weakly joint efforts of MathWorks and the open-source community to build a software setup that enables rapid control software prototyping. This software architecture enables control system design and integration without the need for proficiency in embedded coding that typically utilizes high-level programming languages like C/C++. The higher impact of the dual objective is in advancing methods and tools of verifiable control system design and the embedded code generation that simplifies the V&V process.

The 3DR Iris+ quadrotor, equipped with PX4 “Pixhawk” autopilot, is selected as the primary prototyping platform. The autopilot allows for the real-time execution of an application (attitude controller) that is auto-generated from MatLab/Simulink. This makes the Iris+ quadrotor an ideal platform for rapid flight control prototyping by using MathWork’s auto code generation capability.

Ultimately, the developed setup represents a convenient research and development tool that natively bridges the gap between the safety-critical flight control science and flight experimentation technology by “eliminating” the error-prone manual coding of embedded microcontrollers.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

3DR	3D Robotics
6DOF	6 Degree of Freedom
AO	Area of Operations
BLDC	Brushless Direct Current
CG	Center of Gravity
COTS	Commercial off the Shelf
CSV	Comma Separated Values
DCM	Direction Cosine Matrix
EOM	Equations of Motion
HAL	Hardware Abstraction Layer
ISR	Intelligence, Surveillance and Reconnaissance
LiPo	Lithium Polymer
LQR	Linear Quadratic Regulator
MCU	Micro Controller Unit
MEMS	Micro Electro-Mechanical System
PID	Proportional, Integral and Derivative
PSP	Pilot Support Package
PWM	Pulse Width Modulation
PX4FMU	Pixhawk Flight Management Unit
PX4IO	Pixhawk Input Output Board
RC	Remote Control
RTOS	Real Time Operating System
UAV	Unmanned Aerial Vehicle
UGV	Unmanned Ground Vehicle
VTOL	Vertical Take-off and Landing

THIS PAGE INTENTIONALLY LEFT BLANK

EXECUTIVE SUMMARY

The objectives of the thesis are twofold. First, it aims to demonstrate the design of a high-fidelity 6DOF model of flight dynamics of a multi-copter and the implementation of a control algorithm, both developed in Simulink and integrated onto a commercial-off-the-shelf (COTS) quadrotor. Second, the work organizes and compiles a disjoint effort of the open source community to build a software setup that enables rapid control software prototyping. This rapid software development enables the control system design and integration without the need of proficiency in high-level programming language like C/C++. The higher-level utility of the joint objective is in advancing the methods and tools of verifiable control system design and the code generation that lead to easy to validate flight experiment.

As an initial step, the nonlinear 6DoF mathematical model of the quadrotor that represents the dynamics of naturally unstable system would need to be derived for the simulation of the quadrotor's motion and flight dynamics in Simulink. Subsequently, a linearization of the model and the design of a typical linear attitude (like Proportional, Integral and Derivative [PID] controller) was performed based on the quadrotor's mathematical model. Finally, the Simulink-based model of the attitude controller was used to auto-generate the attitude control software application for the COTS autopilot, and its verification were performed in an indoor Vicon facility. The flight experiment utilized different flight scenarios to outline the performance of the newly developed controller.

The COTS quadrotor selected for this thesis is the 3DR Iris+ that comes with the PX4 Pixhawk autopilot to control its flight. The PX4 Pixhawk autopilot features an open-source software architecture that runs on the Nuttx Real Time Operating System (RTOS), which allows the execution of MatLab/Simulink auto generated embedded application on the Pixhawk ARM MCU. As a result, the quadrotor mathematical model and the linear attitude controller that was developed in Simulink can be used on the PX4 autopilot system without the need to directly edit embedded software using low-level programing language. This makes the Iris+ quadrotor an ideal platform for rapid control prototyping using MatLab/Simulink auto coding and validating the design in flight tests.

Upon successful implementation of the Simulink model on the PX4 Pixhawk autopilot system, the quadrotor would be subjected to a series of flight tests to verify its flight performance using the developed attitude controller design. Ultimately, the developed setup represents a convenient research and development tool that natively bridges the gap between the flight control systems design and flight experimentation by “eliminating” the error-prone manual coding for an embedded microcontroller.

ACKNOWLEDGMENTS

Prior to my postgraduate studies at NPS, I was unfamiliar to the world of controls engineering. In light of this, I was very fortunate to be a student in Dr. Vladimir Dobrokhodov's class on basic controls engineering during my first quarter in NPS. This class stirred my interest in controls engineering and eventually influenced my decision to take up a thesis topic related to this field. I could not thank Dr. Vladimir enough for his guidance and mentoring to my learning journey in NPS. I would also like to thank Dr. Noel Du Toit for generously accepting my request to be the second reader and giving valuable advice for this thesis.

During the process of my thesis research, I had encountered several difficulties in implementing the attitude controller designed in Simulink to the Pixhawk autopilot. I would like to thank Adam Polak, PX4 autopilot and the MathWorks team, who had been infinitely patient with my amateurish queries on the software aspect of this thesis. In addition, I also need to thank the 3D Robotics support team for providing timely replies to my queries on the Iris+ quadrotor and its hardware.

Lastly, not forgetting the most important person in my life, I would like to thank my wife, Emily, for her unwavering support during our stay in Monterey. She has been the emotional rock that I can rely on in difficult times during the thesis research and never failed to encourage me to push forth courageously in the pursuit for knowledge.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. BACKGROUND

A quadrotor is an UAV that features two sets of identical propellers that are powered by DC brushless motors to provide the required thrust force and perform maneuvers when it is in flight. Due to its light-weight configuration and inherent instability, quadrotor has better flight maneuverability over fixed wing UAV. More importantly, quadrotors have the capabilities to perform VTOL and hovering in mid-flight. Along with unprecedented simplicity of the mechanical design that eliminates the complexity of the main rotor control of typical helicopter, multirotor UAVs become unique in the class of unmanned systems; one can think of them as Solid State UAVs. The quadrotor's unique flight characteristics coupled with a cheap price of hardware have increased a number of applications in both the military and commercial sectors.

Indeed, online retail giant Amazon [1] saw the potential of expediting delivery order and supplementing manpower shortages through the use of quadrotors to fly autonomously and make deliveries to its customers. Quadrotors can also be conveniently fitted with high-resolution cameras to provide a relatively cheap means for avid photographers and filmmakers alike to capture pictures at high altitude without the need to charter a flight during productions. In military applications, quadrotor is a popular platform in conducting ISR missions especially in urban AO, where quadrotors can be programmed to fly into buildings to perform visual mapping and identify potential threats.

The popularity of the quadrotor extends to the field of amateur hobbyists and research laboratories as well. With its small size, VTOL and hovering capabilities, research laboratories can easily operate quadrotors in an indoor environment. In the recent years, COTS quadrotors such as AR Parrot, 3DR Iris+, Vision+ Phantom, etc., have become increasingly popular and relatively cheap for UAV researches. However, COTS quadrotors typically come with their own hard-coded software and pre-programmed plant model. Although the quadrotor software supports the programming of basic flight functions, significant programming in a low-level programming language (e.g. C/C++) is required to modify the quadrotor’s autopilot embedded software to perform complex flight controls or modify the its mathematical model.

In academic research, the modeling and simulation software Simulink is frequently used in controls engineering courses as a teaching tool to demonstrate system modeling and controller designs. However, implementing Simulink system model and controller design as embedded software into autopilot hardware would require controls engineering students to be versed in low-level programming language; not only is this level of breadth and depth rarely available, but also the objective of the control systems design is very different from the particularities of a specific MCU platform and language implementation. Therefore, a process or method to directly implement the system models and controller designs developed using Simulink on a quadrotor’s autopilot would be an invaluable enabling tool that provides students with an opportunity to test their models and designs on an actual hardware.

Amongst the COTS quadrotors, PX4 open hardware project elaborated in [2] has designed the Pixhawk autopilot system that can be programmed using the PX4 flight stack software [3]. The PX4 flight stack software runs on the Nuttx RTOS and is able to support multiple applications that can be programmed individually. More importantly, PX4 is able to support system models and control algorithms developed using Simulink without for the need to be proficient in high-level programming. This capability allows for a research project to rapidly progress from the modeling and simulation to implementation phase on the actual hardware.

Therefore, the practical objective of this thesis research is to implement the system model and controller design developed in Simulink directly on a quadrotor autopilot. The quadrotor selected in this thesis is the 3DR Iris+ [4] that comes installed with the PX4 autopilot system. In the first part of this thesis, the mathematical model, EOM and the attitude controller design of the 3DR Iris+ quadrotor will be derived and written in the Simulink software. Subsequently, the second part of this thesis shall focus on the implementation of the quadrotor's mathematical model and controller design onto the PX4 autopilot. Finally, the quadrotor's model and controller design will be validated through an actual flight test in three different flight scenarios.

B. LITERATURE REVIEW

As a first step to this thesis research, the modeling and simulation of the Iris+ quadrotor must be performed to determine its flight characteristics and designing its attitude controllers. The derivations of the dynamics equations for a quadrotor model were well elaborated by Boudallah in [5], Beard in [6], Corke in [7], Sidea in [8] and Bresciani in [9]. However, the modelling approaches in these researches were only applicable for a limited set of quadrotor configurations.

The Iris+ quadrotor featured a cross-style configuration and therefore, modifications to the dynamics equations of the plus-style configuration quadrotor were required to obtain the dynamics equations for a quadrotor in cross-style configuration. An approach to model a quadrotor in the cross-style configuration was elaborated by Partovi in [10] for the X650 quadrotor. This approach was suitable and was adopted for the modelling of the Iris+ quadrotor.

For the rapid prototyping of control designs using Simulink, Lizarraga in [11] used the Piccolo autopilot as the hardware and developed an architecture that enables the use of Simulink models on flight control systems, instead of directly programming the hard-coded software on the flight control systems. More recently, Meier presented in [2] the software architecture for the Pixhawk autopilot, which allows applications to be developed and installed onboard the Pixhawk autopilot.

As a result Pixhawk autopilot's versatility, Polak in [12] defined the process for building an application for attitude controllers developed in Simulink and installing the application on a quadrotor with the Pixhawk autopilot. In addition, Polak worked with MathWorks to develop a PSP [13] for developers to make use of the build function in Simulink to generate the C/C++ codes for Simulink models and install the models as applications on the Pixhawk autopilot. The process that was mutually developed by Polak and MathWorks would be further streamlined and used to implement a Simulink attitude controller onboard the Iris+ quadrotor.

C. OVERVIEW OF QUADROTOR TECHNOLOGY

The airframe of a quadrotor generally consists of two beams that are arranged in a cross, or plus, configuration and mounted onto the main body shell that contains the electronics and flight computers (see Figure 1). Two sets of identical propellers (one set rotates in the Clockwise direction, CW, and the other set rotates in the Counter-Clockwise direction, CCW) are installed on DC brushless motors mounted on the edges of each beam.

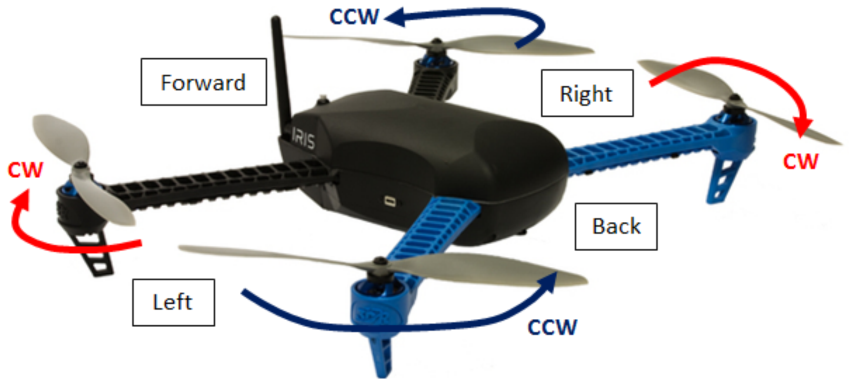


Figure 1 Illustration of Quadrotor Airframe in Cross Configuration (after [4]).

1. Quadrotor Flight Mechanism

As shown in Figure 1, the two sets of propellers mounted on the quadrotor rotate in opposite directions and cancel the net torque acting on the quadrotor.

The quadrotor performs maneuvers in flight by sending Pulse Width Modulation (PWM) signals to the brushless DC motors to vary the rotational speed of the propellers. To lift off from the ground or maneuver vertically, the propellers rotate at the same speed to generate a thrust force to overcome the quadrotor’s weight. To perform a flight maneuver in the horizontal plane, the quadrotor would need to generate a rotating moment by pitching or yawing its body. This is accomplished by varying the rotational speed of each set of propeller.

2. Quadrotor Sensor Systems

A suite of sensor system is required to provide the quadrotor with position and attitude information that are necessary to perform autonomous flights. Most recent of all, advances in MEMS inertial sensor technology now allow for a lightweight navigation unit to be installed on quadrotors in addition to a GPS unit. These sensors provide the quadrotor autopilot with position and attitude information during flight. A barometer or laser range finder is also installed onboard a quadrotor to provide altitude data.

3. Advantages and Disadvantages of Quadrotor/Multi-rotor Technology

With the overview of the multi-rotor technology, a quick analysis of its advantages and disadvantages was summarized in Table 1.

Table 1 Advantages and Disadvantages of Quadrotor.

Advantage	Disadvantages
<u>VTOL & Hovering Capabilities</u> Unlike fixed wing UAV, the unique flight mechanism of the quadrotor allows it to perform VTOL and hovering in flight. These capabilities eliminate the need of a landing strip or a launch and recovery system. <u>Agile Maneuverability</u> By varying the rotational speed of its propellers, the quadrotor is able to generate thrust and moments to perform sharp turns during flight and hover in mid-flight. A fixed wing UAV in contrast makes turns with a larger turning radius. <u>Mechanically Simple</u> Quadrotor uses propeller blades with fixed symmetrical pitch propeller blades and consists of lesser mechanical components compared to conventional helicopters. Therefore, quadrotors are easy to maintain and cheaper to manufacture.	<u>Short Battery Life</u> The battery life of most quadrotors is approximately 20 minutes and is constrained by the charge storage capability of Lithium battery; power density is the fundamental constrain. The short battery life reduces the mission duration of the quadrotor. <u>Under-actuated System</u> A quadrotor is an under-actuated system [14], where multiple actuators are used to perform 6 linear and angular control actions. Therefore, if the symmetry of the actuators action is damaged, it would either no longer be able to perform a maneuver or its control authority might be compromised. <u>Low Payload Capability</u> The payload limit of a medium sized quadrotor (~1.5kg) is typically between 0.8 to 1 lbs. Therefore, the equipment or load that can be carried by quadrotors is not substantial.

D. THESIS OUTLINE

The outline of this thesis is summarized as follows:

Chapter I provides an introduction to quadrotor technology and explains the motivation behind this thesis research; it also provides an overview of the quadrotor components technology.

Chapter II presents an overview of the 3DR Iris+ quadrotor technology, the Pixhawk autopilot and its software architecture.

Chapter III elaborates on the identification of the quadrotor’s configuration and the process of deriving its mathematical model and EOM. In addition, the frames of reference that are used to describe the quadrotor’s position in space and the transformation between the reference frames are also elaborated in this chapter.

Chapter IV describes the equipment and process for measuring the physical specifications of the 3DR Iris+ quadrotor. The physical specifications include the quadrotor’s mass, lengths of its moment arms, mass moment of inertia, and thrust and drag coefficients.

Chapter V gives an overview of the quadrotor flight control principles and the design of the PID controller. In addition, the methodology of tuning the PID controller for stable flight is elaborated in this chapter.

Chapter VI describes the process to implement the PID controller Simulink model as an application for the Pixhawk autopilot using the PX4 flight stack.

Chapter VII presents the analysis of the flight data recorded from the Iris+ flight tests and provides formal review of the attitude controller’s performance.

Chapter VIII draws the conclusion for this thesis and recommends the possible areas that can be looked into for future research work.

THIS PAGE INTENTIONALLY LEFT BLANK

II. 3DR IRIS+ QUADROTOR AND PIXHAWK AUTOPILOT

This chapter provides an overview of the 3DR Iris+ Quadrotor, Pixhawk autopilot hardware and its software architecture. First, this section lists the hardware specifications of the Iris+ quadrotor and the Pixhawk Autopilot. The next section elaborates on the Pixhawk autopilot software architecture and identifies the flight stack software that was used to implement Simulink model to the autopilot.

A. IRIS+ HARDWARE

The Iris+ is a quadrotor that was designed and built by 3D Robotics (3DR) for aerial RC vehicle enthusiasts and hobbyists, and can be fitted with a camera to perform aerial photography (see Figure 2). A LiPo 5100mAh battery supplies up to 12V to the electronics onboard the Iris+, and the four MN2213 950kV DC motors installed on its airframe provide up to 22 minutes of flight time. The rotational speed of the four motors generates a thrust force that is sufficient for the Iris+ quadrotor to overcome its weight and carry a payload of up to 400g. The Iris+ is also equipped with telemetry radio that allows communication with the ground station computer wirelessly, providing real-time flight data and the ability to fly autonomous missions.

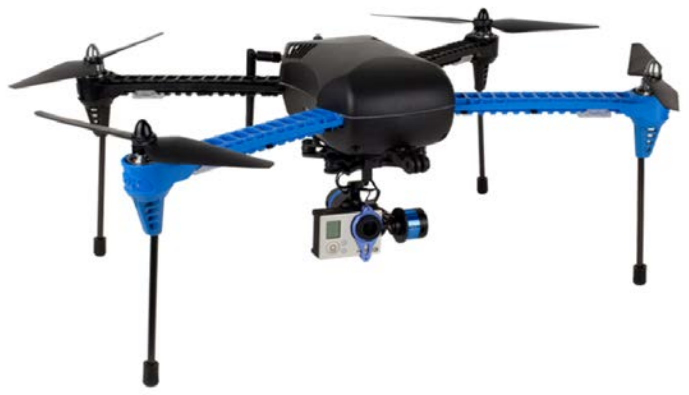


Figure 2 3DR Iris+ Quadrotor (from [4]).

B. PIXHAWK AUTOPILOT

The Pixhawk autopilot (see Figure 3) was designed by the PX4 open-hardware project [15] and combines the Pixhawk Flight Management Unit (PX4FMU) and PX4IO into a single component. It features sensor technology from ST Microelectronics® and a Cortex M4 microprocessor running the NuttX RTOS that allows integrated multi-threading and programming in a Unix/Linux-like environment. In addition, the NuttX RTOS allows for developers to easily implement C/C++ codes onto the Pixhawk autopilot through the building and uploading of applications. The complete specifications of the Pixhawk autopilot are summarized in Table 2.

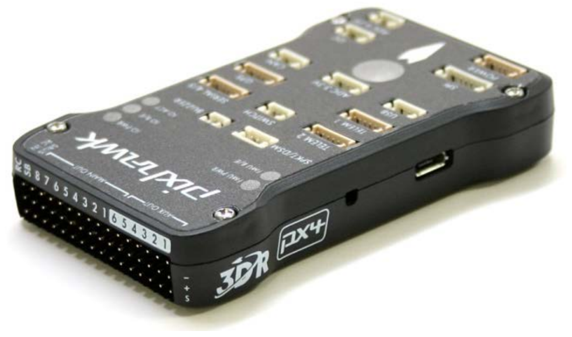


Figure 3 PX4 Pixhawk Autopilot (from [16]).

Table 2 PX4 Pixhawk Hardware Specifications (from [16]).

Processor	32bit STM32F427 Cortex M4 core with FPU 168 MHz 256 KB RAM 2 MB Flash 32 bit STM32F103 failsafe co-processor
Sensors	ST Micro L3GD20H 16 bit gyroscope ST Micro LSM303D 14 bit accelerometer / magnetometer Invensense MPU 6000 3-axis accelerometer/gyroscope MEAS MS5611 barometer
Interfaces	5x UART (serial ports), one high-power capable, 2x with HW flow control 2x CAN (one with internal 3.3V transceiver, one on expansion connector) Spektrum DSM / DSM2 / DSM-X® Satellite compatible input Futaba S.BUS® compatible input and output PPM sum signal input RSSI (PWM or voltage) input I2C SPI 3.3 and 6.6V ADC inputs Internal microUSB port and external microUSB port extension

C. PIXHAWK AUTOPILOT SOFTWARE ARCHITECTURE

The Pixhawk autopilot supports two main flight control software families as described in [3], namely the PX4 flight stack and the APM flight stack, which are both open source software projects. The architecture and framework of both flight control stacks are further elaborated below:

1. PX4 Flight Control Software

The PX4 flight control framework described in [2] consists of three main layers: PX4 flight stack (containing individual applications such as the flight control, state estimation, etc.), PX4 middleware (communications between the applications and drivers) and PX4 drivers (architect specific) as shown in Figure 4. The architecture allows for a modular design since those three layers are naturally separated and can be run independently from each other. In the PX4 flight stack layer, the flight control and state estimation exist as self-contained applications, which can be independently managed at runtime. Therefore, the codes that are generated for the Pixhawk autopilot are highly portable and allow the Pixhawk autopilot to be used for a variety of autonomous vehicles (e.g., fixed wing aerial vehicle, unmanned ground vehicle, etc.). In addition, each application connects to other processes and drivers using a Publisher-Subscriber framework (see Figure 5), allowing for efficient communication between processes and simplifies the process of adding a new application to the system.

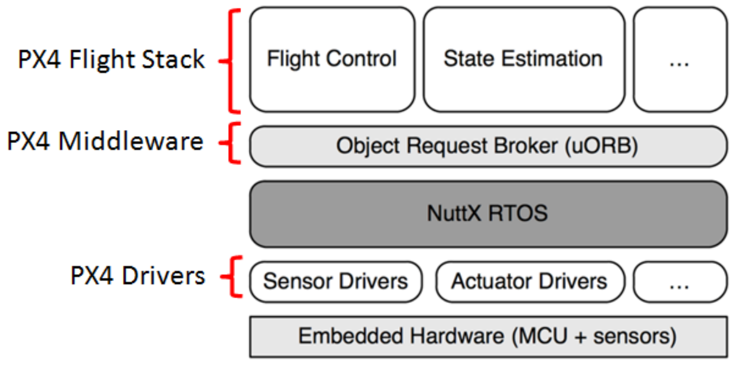


Figure 4 PX4 Flight Control Framework (after [2]).

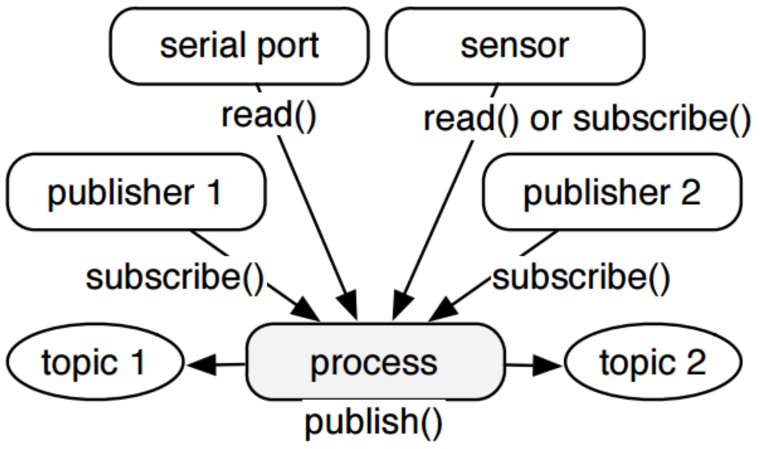


Figure 5 PX4 Application Framework (from [2]).

2. APM Flight Control Framework

The APM flight control software described in [17] was originally designed for the Ardupilot autopilot and was ported as a single application to the PX4 flight control architecture. The APM application can be run on any PX4 control board (i.e. PX4FMU or Pixhawk autopilot) through the PX4 middleware layer (HAL) in the PX4 framework (see Figure 6). When the APM application is selected as the flight control stack on the PX4 control board, it will be executed to replace the PX4 as the main flight controller stack to control the drivers. Therefore, the APM flight control stack functions as a single monolithic application in the PX4 framework with some internal worker threads to execute slower tasks (e.g., data logging) and does behave from the user perspective like the legacy APM hardware.

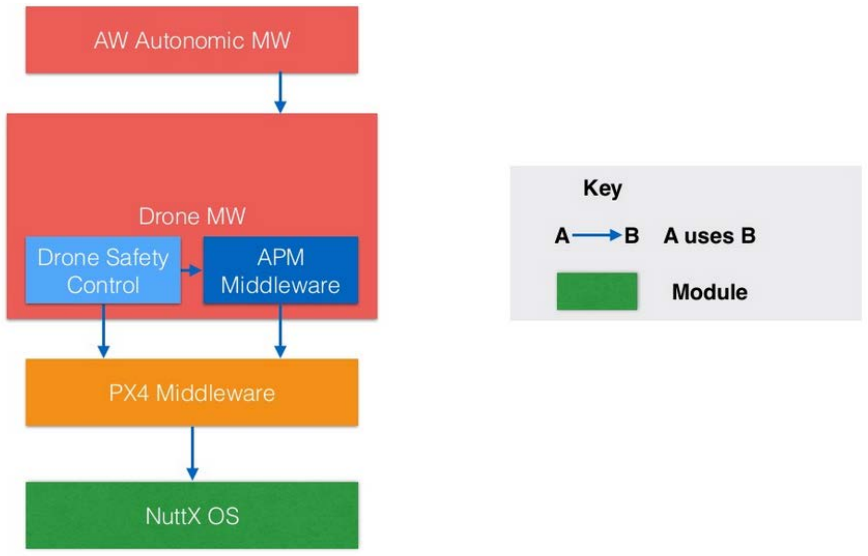


Figure 6 APM Flight Control Application in PX4 Framework (from [18]).

D. FLIGHT STACK SELECTION

Although the Pixhawk autopilot can adequately support both flight stacks, the PX4 flight stack was ultimately chosen as the approach to implement the flight controls in this thesis. This is mainly because of potentially greater flexibility of the PX4 stack and the fact that MathWorks [19] had attempted to lead the open source community effort of bringing the Pixhawk autopilot into academic research; the attempt is not finished yet as the PSP initiated by Mathworks is not officially released yet as of the date of this writing. The PX4 PSP would provide an efficient and convenient means to implement controllers and models designed in Simulink onto actual hardware. The key tools of the PX4 PSP that support this process are as follows.

(1) PX4 Simulink Blocks & Examples

A library of PX4 Simulink blocks were created for the PX4 PSP to interface with the Pixhawk autopilot. In addition, examples of the PX4 Simulink model were also available in the PX4 PSP that can be used for developing the plant or controller of a vehicle.

(2) PX4 Eclipse

The PX4 Eclipse provides the platform to build the application from the generated C/C++ codes of the Simulink model and download it to the Pixhawk autopilot.

(3) TeraTerm Terminal

TeraTerm is a serial terminal program that can connect the user’s computer to the Pixhawk autopilot to manually run the built-in commands using the Nuttx shell. As the Pixhawk autopilot is running on the Nuttx OS, TeraTerm offers a convenient means to access the applications on the Pixhawk autopilot from a computer operating in Windows OS.

III. QUADROTOR MATHEMATICAL MODEL

This chapter documents the process of deriving the quadrotor mathematical model and the development of the equation of motions to describe the Iris+'s movement with respect to a reference coordinate frame. The modeling of a quadrotor is well described in articles [5], [6], [7], [8], [9] and [10], and were used as references for the derivation of the equations found in this chapter. The model and equation of motions are important for predicting the positions of the Iris+ during flight, and used for the controller design in Chapter IV. The sub-sections in this chapter are as follow:

Section A introduces the method in the identification of quadrotor configurations. A typical symmetrical quadrotor can be categorized into two main configurations: 'plus' and 'cross' configurations. This section describes the key characteristics of both configurations and their flight mechanisms.

Section B defines the notations used in the quadrotor mathematical model. This section identifies and consolidates all the notations that are used to ensure consistency in the implementation of the mathematical model to the simulation model.

Section C identifies all the coordinate frames that are used as the reference for the quadrotor's position and movement in space. It is important to identify the coordinate frames as the forces acting on the quadrotor are applied with reference to different coordinate frames.

Section D describes the transformation of the quadrotor's kinematics from one reference frame to the other. In addition, the transformation matrix that is used for describing the positions, position rates, Euler angles and angular rates from one coordinate frame to the other is elaborated in this section.

Section E describes the quadrotor's dynamics by identifying the forces and moments acting on the quadrotor in the various coordinate frames. As the forces and moments are described in different coordinate frames, they cannot be summed directly. Therefore, the transformation method formulated in Section D must be used to transform the forces and moments to a single reference coordinate frame.

Section F formulates the complete system of 6DOF EOM in the Iris+ quadrotor's body coordinate frame using the forces and moments defined in the previous section.

A. IDENTIFICATION OF QUADROTOR CONFIGURATION

A quadrotor consists of four extended arms with four BLDC motors with a fixed-pitch propeller (the propellers are labeled from 1 to 4 in the clockwise direction) attached to them. The motors are arranged to rotate one pair of propellers counter-clockwise and the other pair of propellers in the clockwise direction. With the rotation direction of the four propellers, there are two basic flight configurations that can be adopted by a quadrotor, namely the plus configuration and cross configuration as shown in Figure 7. For a quadrotor in the plus configuration to change its attitude (i.e., roll, pitch or yaw), the rotational speed for two propellers are varied. However, the quadrotor in cross configuration changes its attitude by varying the rotational speed of all four propellers. This gives quadrotors in the cross configuration a higher momentum and therefore, a better maneuverability performance compared to quadrotors in the plus configuration.

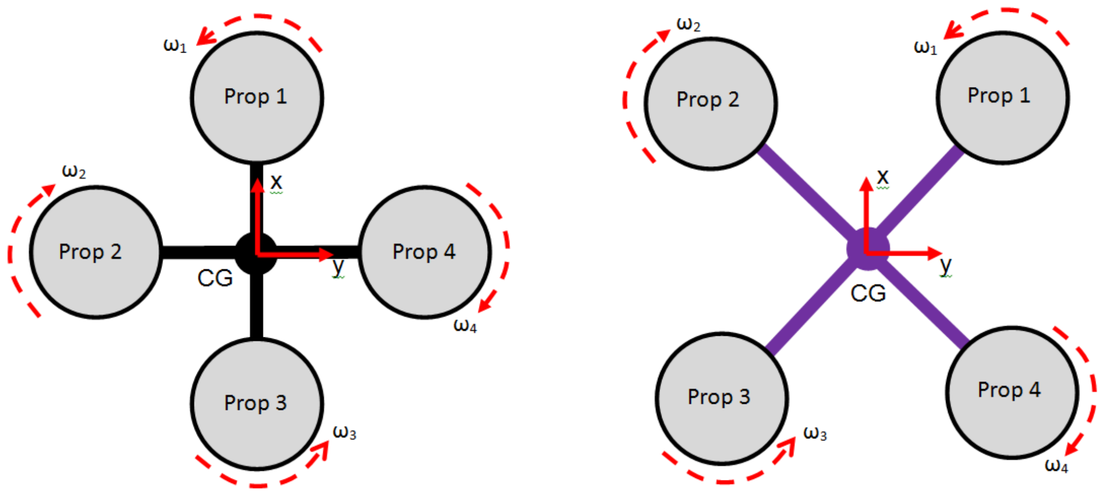


Figure 7 Quadrotor in Plus (+) and Cross (X) Configurations.

1. Plus Configuration Flight Mechanics

For a quadrotor to adopt a plus configuration, its arms are aligned with the quadrotor's body x-axis and y-axis (arranged in the right hand rule orientation). The quadrotor in plus configuration changes the speed of the rotating DC motors to perform a translational or rotational maneuver as shown in Figure 8. By changing the rotational speed of all four propellers by the same amount, thrust (i.e. T) is generated to accelerate the quadrotor along the vertical z-axis. For the quadrotor to perform a roll maneuver, the rotational speed of propeller 2 is increased and the rotational speed of propeller 4 is reduced to generate a torque along the x-axis (i.e. τ_Φ). The concept is similar for the pitch maneuver, where the rotational speed of propeller 1 is increased and the rotational speed of propeller 3 is reduced to generate a torque along the y-axis (i.e. τ_θ). Finally, by applying different speed to each pair of propellers rotating in the same direction, a torque along the z-axis (i.e. τ_ψ) is generated to perform a yaw maneuver.

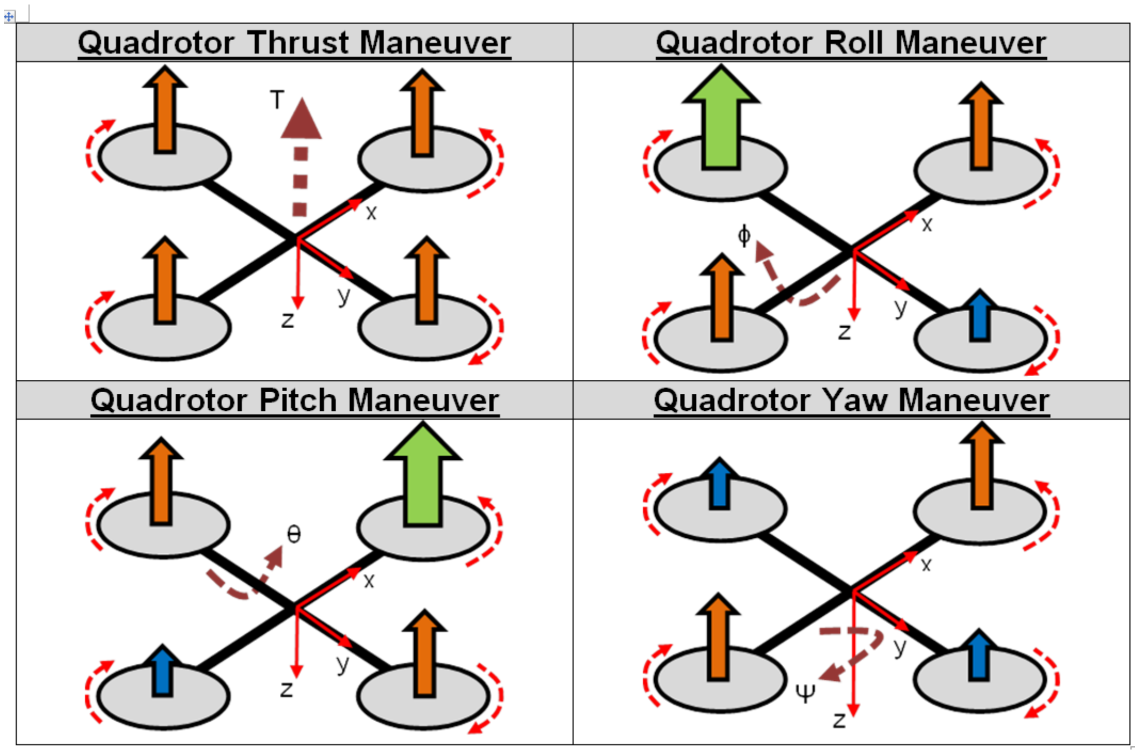


Figure 8 Flight Mechanisms for Quadrotor in Plus Configuration.

2. Cross Configuration Flight Mechanism

Unlike the plus configuration, the body x-axis and y-axis for the quadrotor adopting cross configuration are tilted 45^o with respect to the quadrotor arms. The quadrotor in cross configuration changes the speed of the rotating DC motors to perform a translational or rotational maneuver as shown in Figure 9. Similar to a quadrotor in plus configuration, the quadrotor in cross configuration changes the rotational speed of all four propellers by the same amount, to generate a thrust (i.e. T) and accelerates the quadrotor along the vertical z-axis. For the quadrotor to perform a roll maneuver, the rotational speed of propellers 3 and 4 are increased, while the rotational speed of propellers 1 and 2 are reduced to generate a torque along the x-axis (i.e. τ_{Φ}). The concept is similar for the pitch maneuver, where the rotational speed of propellers 1 and 4 are increased, while the rotational speed of propellers 2 and 3 are reduced to generate a torque along the y-axis (i.e. τ_θ). Finally, by applying different rotational speed to the counter rotating pair of propellers, a torque along the z-axis (i.e. τ_ψ) is generated to perform a yaw maneuver.

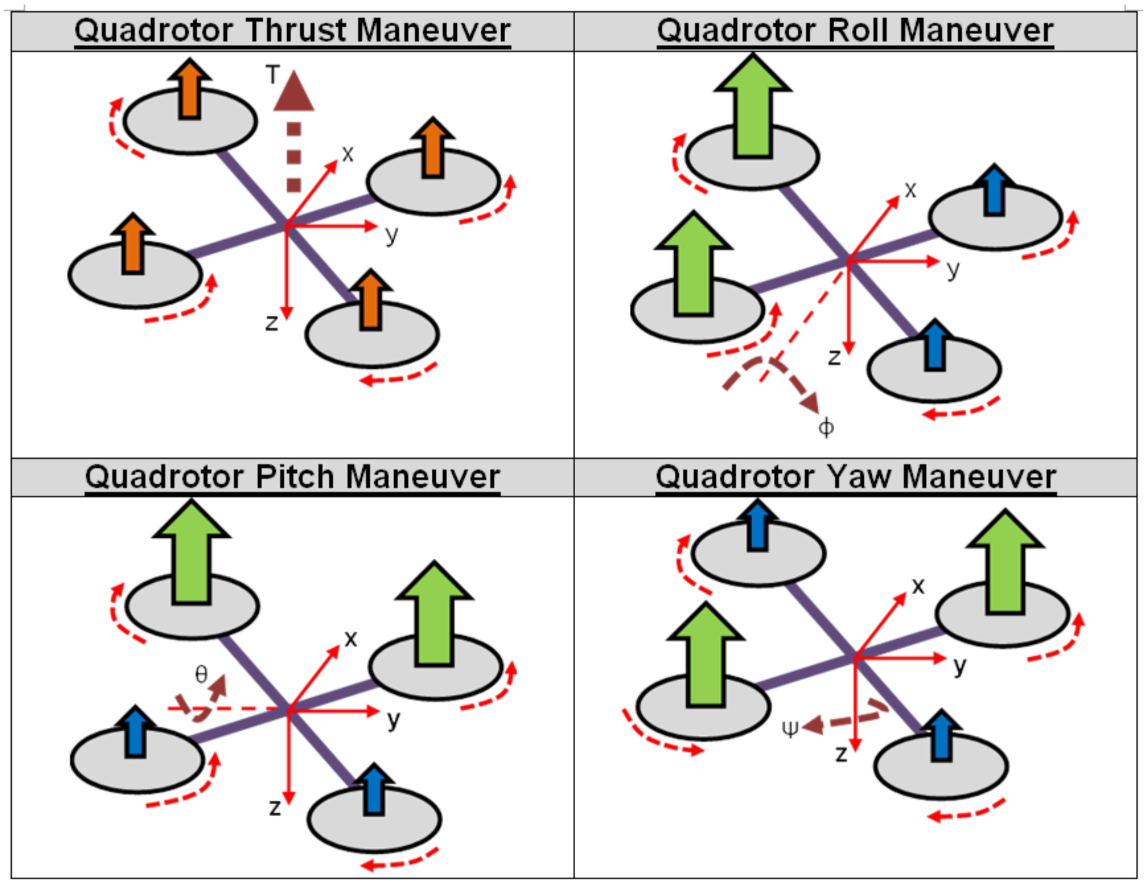


Figure 9 Flight Mechanisms for Quadrotor in Cross Configuration.

3. Notations for Quadrotor Mathematical Model

The notations for the quadrotor’s translational and rotational motions are summarized in Table 3.

Table 3 Notations for Quadrotor Translational & Rotational Motions.

States	Description
x_i	Quadrotor position along the x-axis in the inertia frame.
y_i	Quadrotor position along the y-axis in the inertia frame.
z_i	Quadrotor position along the z-axis in the inertia frame.
$\dot{x}_i$	Quadrotor velocity along the x-axis in the inertia frame.
$\dot{y}_i$	Quadrotor velocity along the y-axis in the inertia frame.
$\dot{z}_i$	Quadrotor velocity along the z-axis in the inertia frame.
$\ddot{x}_i$	Quadrotor acceleration along the x-axis in the inertia frame.
$\ddot{y}_i$	Quadrotor acceleration along the y-axis in the inertia frame.
$\ddot{z}_i$	Quadrotor acceleration along the z-axis in the inertia frame.
x_b	Quadrotor position along the x-axis in the body frame.
y_b	Quadrotor position along the y-axis in the body frame.
z_b	Quadrotor position along the z-axis in the body frame.
x_v	Quadrotor position along the x-axis in the vehicle frame.
y_v	Quadrotor position along the y-axis in the vehicle frame.
z_v	Quadrotor position along the z-axis in the vehicle frame.
u	Quadrotor velocity along the x-axis in the body frame.
v	Quadrotor velocity along the y-axis in the body frame.
w	Quadrotor velocity along the z-axis in the body frame.
Φ	Quadrotor roll angle with reference to inertia frame axis.
θ	Quadrotor pitch angle with reference to inertia frame axis.
Ψ	Quadrotor yaw angle with reference to inertia frame axis.
p	Quadrotor roll rate along the x-axis in the body frame.
q	Quadrotor pitch rate along the y-axis in the body frame.
r	Quadrotor yaw rate along the z-axis in the body frame.

4. Quadrotor Coordinate Frames

To build the quadrotor’s mathematical model, it is important to first define the coordinate frames for describing the quadrotor’s translational and rotational motion (i.e., quadrotor six degrees of freedom, 6DOF). The coordinate frames that are used in the mathematical model are shown in Figure 10 and further elaborated below:

a. Inertial Frame, {i}

The inertial coordinate frame, {i} is a reference fixed frame represented by the unit vector, {i} = $[x_i \ y_i \ z_i]^T$. The x-axis of the inertial frame is pointed to the North, y-axis is pointed to the East and z-axis is pointed to the center of the Earth.

b. Vehicle Frame, {v}

The vehicle coordinate frame, {v} has its origin fixed to the quadrotor CG and is represented by the unit vector, {v} = $[x_v \ y_v \ z_v]^T$. The axes of the vehicle coordinate frame are aligned to the axes of the inertia coordinate frame and do not change even during the quadrotor's rotational motion. The vehicle coordinate frame describes the translational motion of the quadrotor with respect to the inertia coordinate frame on the x-y plane.

c. Body Frame, {b}

Similar to the {v}, the body coordinate frame, {b} has its origin located at the quadrotor's center of gravity (CG) and is represented by the unit vector, {b} = $[x_b \ y_b \ z_b]^T$ that is rigidly attached to the body and thus, rotates with the body. Therefore, the body coordinate frame describes the rotational motion of the quadrotor with respect to {v} as shown in Figure 10. The x-axis of the body coordinates always point out from the front of the quadrotor and the y-axis points to the right of the quadrotor. Finally, the z-axis of the body coordinate frame is pointed down completing the right hand coordinate frame. {v} can be rotated along the z-axis by the yaw angle, Ψ, along the y-axis by the pitch angle, θ and along the x-axis by the roll angle, ϕ.

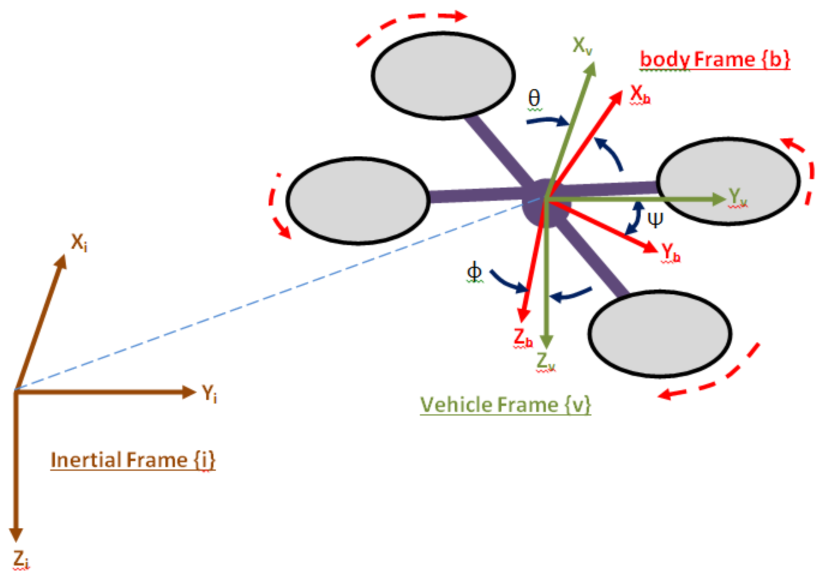


Figure 10 Inertia, Body and Vehicle Coordinate Frames.

B. QUADROTOR KINEMATICS

The quadrotor's linear and angular positions in the body and inertia frames were referenced from [5], [6], [7], [8], [9] and [10]. These equations can be expressed in the vector form as:

$$P^b = \begin{bmatrix} x^b & y^b & z^b \end{bmatrix}^T \quad (1)$$

$$P^i = \begin{bmatrix} x^i & y^i & z^i \end{bmatrix}^T \quad (2)$$

$$\Lambda^b = \begin{bmatrix} \phi & \theta & \psi \end{bmatrix}^T \quad (3)$$

Similarly, the quadrotor linear and angular velocities in the quadrotor's body frame can be expressed in the vector form as:

$$V^b = \begin{bmatrix} u & v & w \end{bmatrix}^T \quad (4)$$

$$\dot{\Lambda} = \begin{bmatrix} p & q & r \end{bmatrix}^T \tag{5}$$

1. DCM Rotation Matrix

The transformation of the quadrotor's angular motion from the body frame to the inertia frame can be described by a rotation matrix, which is also known as DCM. The DCM is the combination of a sequence of rotations where the quadrotor is first rotated along the z-axis (i.e., yaw), followed by a rotation along the y-axis (i.e., pitch) and finally by a rotation along the x-axis (i.e., roll). The rotation matrices for the yaw, pitch and roll are multiplied to obtain equation (6):

$$R_i^b = \begin{bmatrix} c\theta \cdot c\psi & s\psi \cdot c\theta & -s\theta \\ c\psi \cdot s\theta \cdot s\phi - s\psi \cdot c\phi & s\psi \cdot s\theta \cdot s\phi + c\psi \cdot c\phi & c\theta \cdot s\psi \\ c\psi \cdot s\theta \cdot s\phi + s\psi \cdot s\phi & s\psi \cdot s\theta \cdot s\phi - c\psi \cdot c\phi & c\theta \cdot c\phi \end{bmatrix} \tag{6}$$

Where 'c' represents cosine and 's' represents sine of the particular angle.

$$P^b = R_i^b \cdot P^i \tag{7}$$

$$\therefore \begin{bmatrix} x^b \\ y^b \\ z^b \end{bmatrix} = \begin{bmatrix} c\theta \cdot c\psi & s\psi \cdot c\theta & -s\theta \\ c\psi \cdot s\theta \cdot s\phi - s\psi \cdot c\phi & s\psi \cdot s\theta \cdot s\phi + c\psi \cdot c\phi & c\theta \cdot s\phi \\ c\psi \cdot s\theta \cdot s\phi + s\psi \cdot c\phi & s\psi \cdot s\theta \cdot s\phi - c\psi \cdot c\phi & c\theta \cdot c\phi \end{bmatrix} \begin{bmatrix} x^i \\ y^i \\ z^i \end{bmatrix} \tag{8}$$

As the DCM is orthogonal and its rows/columns are linearly independent, the inverse rotation matrix that transforms the quadrotor's angular motion from the body frame to the inertia frame is simply the transpose of the DCM:

$$R_b^i = R_i^{b^T} = \begin{bmatrix} c\theta \cdot c\psi & c\psi \cdot s\theta s\phi - s\psi \cdot c\phi & c\psi \cdot s\theta \cdot c\phi + s\psi \cdot s\phi \\ s\psi \cdot c\theta & s\psi \cdot s\theta \cdot s\phi + c\psi \cdot c\phi & s\psi \cdot s\theta \cdot c\phi - c\psi \cdot s\phi \\ -s\theta & c\theta \cdot s\phi & c\theta \cdot c\phi \end{bmatrix} \tag{9}$$

$$P^i = R_b^i \cdot P^b \tag{10}$$

$$\begin{bmatrix} x^i \\ y^i \\ z^i \end{bmatrix} = \begin{bmatrix} c\theta \cdot c\psi & c\psi \cdot s\theta s\phi - s\psi \cdot c\phi & c\psi \cdot s\theta \cdot c\phi + s\psi \cdot s\phi \\ s\psi \cdot c\theta & s\psi \cdot s\theta \cdot s\phi + c\psi \cdot c\phi & s\psi \cdot s\theta \cdot c\phi - c\psi \cdot s\phi \\ -s\theta & c\theta \cdot s\phi & c\theta \cdot c\phi \end{bmatrix} \begin{bmatrix} x^b \\ y^b \\ z^b \end{bmatrix} \tag{11}$$

2. Rotation Matrix for Quadrotor Angular Velocities

The transformation of the quadrotor's angular velocities from the inertia frame to the body frame was referenced from [20] and can be described by the following equations:

$$\omega^b = R_\phi \cdot R_\theta \cdot R_\psi \begin{bmatrix} 0 \\ 0 \\ \dot{\psi} \end{bmatrix} + R_\phi \cdot R_\theta \begin{bmatrix} 0 \\ \dot{\theta} \\ 0 \end{bmatrix} + R_\phi \begin{bmatrix} 0 \\ 0 \\ \dot{\phi} \end{bmatrix} \tag{12}$$

where: ω^b = angular velocity of quadrotor in body coordinates,

R_ϕ = rotation matrix along the x-axis (i.e., roll),

R_θ = rotation matrix along the y-axis (i.e., pitch),

R_ψ = rotation matrix along the z-axis (i.e., yaw).

$$\therefore \omega_b = \begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} 1 & 0 & -s\theta \\ 0 & c\phi & c\theta \cdot s\phi \\ 0 & -s\phi & c\theta \cdot c\phi \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \tag{13}$$

The transformation of the quadrotor's angular velocities from the body frame coordinate to the inertia frame coordinate can be represented by the following equation:

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & s\phi \frac{s\theta}{c\theta} & c\phi \frac{s\theta}{c\theta} \\ 0 & c\phi & -s\phi \\ 0 & s\phi \frac{1}{c\theta} & c\phi \frac{1}{c\theta} \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{14}$$

C. QUADROTOR DYNAMICS

The following assumptions were undertaken in deriving the quadrotor dynamics:

- The quadrotor's center of gravity coincides with the origin of the body frame.
- The quadrotor is a rigid body.
- The quadrotor is symmetrical with respect to the x and y-axes as described in [5], [6], [7], [8], [9] and [10].
- The quadrotor propellers are rigid.
- The thrust and drag exerted on the quadrotor are proportional to the square of the propellers' angular speed.

1. Gravitational Forces

The gravitational force vector acting on the quadrotor's CG in the inertia coordinate frame can be expressed as:

$$\vec{F}_G^i = \begin{bmatrix} 0 \\ 0 \\ m \cdot g \end{bmatrix} \tag{15}$$

where: m = mass of quadrotor and

g = gravitational acceleration.

The gravitational force acting on the quadrotor's CG in the body frame can be obtained by multiplying the rotation matrix with the gravitational force vector in the inertia coordinate frame:

$$\vec{F}_G^{\,b} = R_i^b \cdot \vec{F}_G^{\,i} \tag{16}$$

$$\therefore \vec{F}_G^{\,b} = \begin{bmatrix} -mg \cdot s\theta \\ mg \cdot c\theta \cdot s\phi \\ mg \cdot c\theta \cdot c\phi \end{bmatrix} \tag{17}$$

2. Gyroscopic Effect

The rotational motion of the propeller-rotor combination generates a gyroscopic effect that acts on the quadrotor in the body coordinate frame. The gyroscopic effect is contributed by the rotor's moment of inertia, the rotor's angular velocity, and the body attitude rate, which can be expressed by equation (18):

$$G^b = I_{rotor}\left(\omega_b \times \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}\right)\Omega = I_{rotor}\left(\begin{bmatrix} p \\ q \\ r \end{bmatrix} \times \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}\right)\Omega \tag{18}$$

where: ω_b is the angular velocity of the quadrotor (body coordinate frame) during the flight,

Ω is the sum of the 4 rotors' rotational velocities (i.e. $\Omega = \omega_1 + \omega_2 + \omega_3 + \omega_4$),

I_{rotor} is the rotor moment of inertia given by

$$I_{rotor} = \left(\frac{1}{4} m_{motor} \cdot r_{motor}{}^2\right) + \left(\frac{1}{12} m_{prop} \cdot L_{prop}{}^2\right),$$

m_{motor} is the motor mass,

r_{motor} is the motor radius

m_{prop} is the propeller mass and

L_{prop} is the propeller length.

$$\therefore G^b = I_{rotor} \begin{bmatrix} q \\ p \\ 0 \end{bmatrix} \Omega \tag{19}$$

Assuming that the attitude control system performs as expected and ideally regulates the angular dynamics to the near hover conditions, the body rates of the UAV become close to zero during the flight. Together with constant and small value of the moment of inertial of the rotor-propeller combination I_{rotor}, the product G^b in the last equation will always be small as long as the roll and pitch attitude rates in ω_b are regulated to near zero values. Therefore, the contribution of the gyroscopic effect to the quadrotor's total moment is very small and can be initially neglected at the first phase of linear control design approach.

3. Aerodynamic Forces

The aerodynamic forces acting on the quadrotor during flight are further elaborated in the sections below:

a. Quadrotor Thrust Force

The thrust from the propellers acting on the quadrotor along the z-axis on the body coordinate frame (i.e., z_b) can be expressed as:

$$T^b = -K_T \left(\omega_1^2 + \omega_2^2 + \omega_3^2 + \omega_4^2 \right) \tag{20}$$

$$T^b = -K_T \cdot U_1 \tag{21}$$

where: K_T is the propeller thrust coefficient and

U_1 is the thrust control input for the propellers' rotation velocity.

b. Quadrotor Roll Moment

The roll moment for the quadrotor along the x-axis on the body coordinate frame in plus and cross configurations can be expressed as:

- Plus Configuration

$$\tau_{\phi} = K_T \cdot l_y \cdot (-\omega_2^2 + \omega_4^2)$$

- Cross Configuration

$$\tau_{\phi} = K_T \cdot l_y \cdot (-\omega_1^2 - \omega_2^2 + \omega_3^2 + \omega_4^2)$$

$$\therefore \tau_{\phi} = K_T \cdot l_y \cdot U_2$$

where: l_y is the length of the moment arm on the body y-axis and

U_2 is the roll control input for the propellers' rotation velocity.

c. Quadrotor Pitch Moment

The pitch moment for the quadrotor along the y-axis on the body coordinate frame in plus and cross configurations can be expressed as:

- Plus Configuration

$$\tau_{\theta} = K_T \cdot l_x \cdot (\omega_1^2 - \omega_3^2) \tag{22}$$

- Cross Configuration

$$\tau_{\theta} = K_T \cdot l_x \cdot \left(\omega_1^2 - \omega_2^2 - \omega_3^2 + \omega_4^2\right) \tag{23}$$

$$\therefore \tau_{\theta} = K_T \cdot l_x \cdot U_3 \tag{24}$$

where: l_x is the length of the moment arm on the body x-axis and

U_3 is the pitch control input for the propellers' rotation velocity.

d. Quadrotor Yaw Moment

The yaw moment for the quadrotor along the z-axis on the body coordinate frame in plus and cross configurations are the same and can be expressed as:

$$\tau_\psi = K_D \cdot \left(\omega_1^2 - \omega_2^2 + \omega_3^2 - \omega_4^2\right) \tag{25}$$

$$\therefore \tau_\psi = K_D \cdot U_4 \tag{26}$$

where: K_D is propellers' drag coefficient and

U_4 is the yaw control input for the propellers' rotation velocity.

e. Summary of Aerodynamic Forces and Moments

The relationship between the aerodynamic forces and the propellers' rotational velocity can be represented in matrix form. The matrices for quadrotor in plus and cross configurations can be expressed as:

- Plus Configuration

$$\begin{bmatrix} T^b \\ \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix} = \begin{bmatrix} -K_T & -K_T & -K_T & -K_T \\ 0 & -K_T \cdot l_y & 0 & K_T \cdot l_y \\ K_T \cdot l_x & 0 & -K_T \cdot l_x & 0 \\ K_D & -K_D & K_D & -K_D \end{bmatrix} \begin{bmatrix} \omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \\ \omega_4^2 \end{bmatrix} \tag{27}$$

- Cross Configuration

$$\begin{bmatrix} T^b \\ \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix} = \begin{bmatrix} -K_T & -K_T & -K_T & -K_T \\ -K_T \cdot l_y & -K_T \cdot l_y & K_T \cdot l_y & K_T \cdot l_y \\ K_T \cdot l_x & -K_T \cdot l_x & -K_T \cdot l_x & K_T \cdot l_x \\ K_D & -K_D & K_D & -K_D \end{bmatrix} \begin{bmatrix} \omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \\ \omega_4^2 \end{bmatrix} \tag{28}$$

D. QUADROTOR EQUATIONS OF MOTION

With the gravitational force and the quadrotor thrust derived in the previous sections, the total force acting on the quadrotor in the body frame is:

$$\sum \vec{F}^b = \vec{F}_G^b + T^b \tag{29}$$

$$\sum \vec{F}^b = \begin{bmatrix} -mg \cdot s\theta \\ mg \cdot c\theta \cdot s\phi \\ mg \cdot c\theta \cdot c\phi \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ -K_T \cdot U_1 \end{bmatrix} = \begin{bmatrix} -mg \cdot s\theta \\ mg \cdot c\theta \cdot s\phi \\ mg \cdot c\theta \cdot c\phi - K_T \cdot U_1 \end{bmatrix} \tag{30}$$

With the quadrotor moments derived in the previous section, the total moment experienced by the quadrotor in the body frame can be expressed as:

$$\sum \vec{M}^b = \tau_\phi + \tau_\theta + \tau_\psi \tag{31}$$

$$\sum \vec{M}^b = \begin{bmatrix} K_T \cdot l_y \cdot U_2 \\ K_T \cdot l_x \cdot U_3 \\ K_D \cdot U_4 \end{bmatrix} = \begin{bmatrix} K_T \cdot l_y \cdot U_2 \\ K_T \cdot l_x \cdot U_3 \\ K_D \cdot U_4 \end{bmatrix} \tag{32}$$

Using the force and moment equations derived in the previous sections, the quadrotor’s 6DOF EOM can be summarized in equations (35) and (36):

$$\sum \vec{F}^{b} = m\begin{bmatrix} \ddot{x}_b \\ \ddot{y}_b \\ \ddot{z}_b \end{bmatrix} = \begin{bmatrix} -mg \cdot s\theta \\ mg \cdot c\theta \cdot s\phi \\ mg \cdot c\theta \cdot c\phi - K_T \cdot U_1 \end{bmatrix} \tag{33}$$

$$\sum \vec{M}^{b} = I\begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} K_T \cdot l_y \cdot U_2 \\ K_T \cdot l_x \cdot U_3 \\ K_D \cdot U_4 \end{bmatrix} \tag{34}$$

$$\begin{bmatrix} \ddot{x}_b \\ \ddot{y}_b \\ \ddot{z}_b \end{bmatrix} = \frac{1}{m}\begin{bmatrix} -mg \cdot s\theta \\ mg \cdot c\theta \cdot s\phi \\ mg \cdot c\theta \cdot c\phi - K_T \cdot U_1 \end{bmatrix} \tag{35}$$

$$\therefore \begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \frac{1}{I}\begin{bmatrix} K_T \cdot l_y \cdot U_2 \\ K_T \cdot l_x \cdot U_3 \\ K_D \cdot U_4 \end{bmatrix} \tag{36}$$

THIS PAGE INTENTIONALLY LEFT BLANK

IV. DETERMINING IRIS+ PHYSICAL PROPERTIES

This chapter describes the process and methodology that were used to determine the physical properties of the Iris+, which are necessary for the precise modeling of its dynamics of motion and its modeling and simulation in Simulink. The sections in this chapter are as follow:

Section A describes the measurement of the quadrotor physical specifications. In addition, the equipment that was used in the measurement of the Iris+ was also listed in this section.

Section B describes the method of using a trifilar pendulum to determine the Iris+’s mass moment of inertia.

Section C describes the experimental method that was used to determine the Iris+’s propeller properties (i.e., thrust and torque coefficients).

A. MEASURING QUADROTOR PROPERTIES

For accurate representation of the quadrotor’s mathematical model, its physical properties need to be measured and determined. A measuring tape and weighing scale (see Figure 11 and Figure 12) were used as the measuring equipment to determine the quadrotor’s weight and length of its moment arms along the x and y-axes (see Figure 13).

Figure 11 Measuring Tape used in Length Measurements.

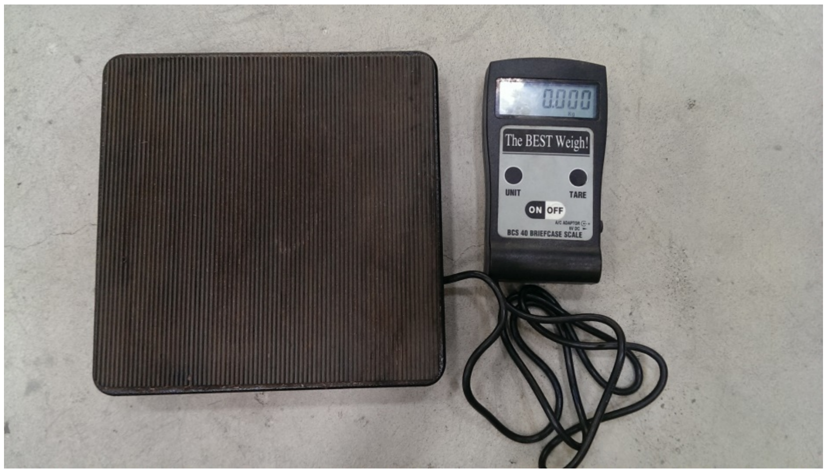


Figure 12 BCS-40 Weighing Scale.

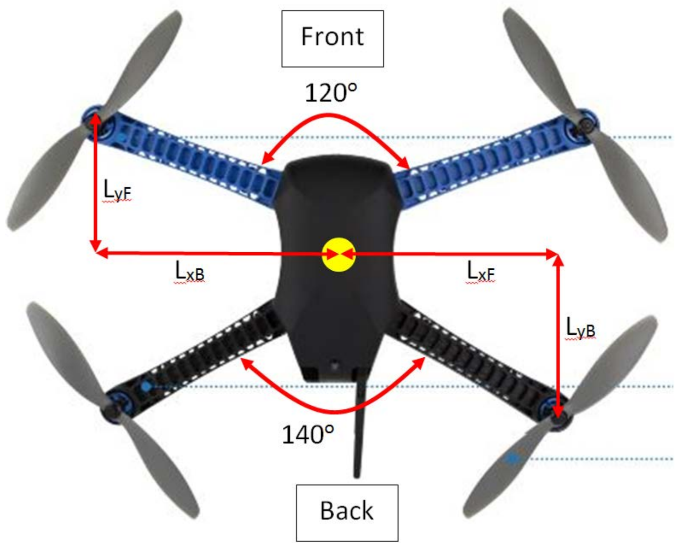


Figure 13 Illustration of Iris+ Moment Arms.

The results from the measurement of the Iris+ quadrotor’s physical specifications are summarized in Table 4.

Table 4 Measurement of Quadrotor Physical Specifications.

Measured Quadrotor Physical Specifications	
Mass, including battery (kg)	1.37
Front Moment Arm Length along x-axis, L_{xF} (m)	0.0923
Front Moment Arm Length along y-axis, L_{yF} (m)	0.2537
Back Moment Arm Length along x-axis, L_{xB} (m)	0.13
Back Moment Arm Length along y-axis, L_{yB} (m)	0.2252

B. DETERMINING IRIS+ MASS MOMENT OF INERTIA

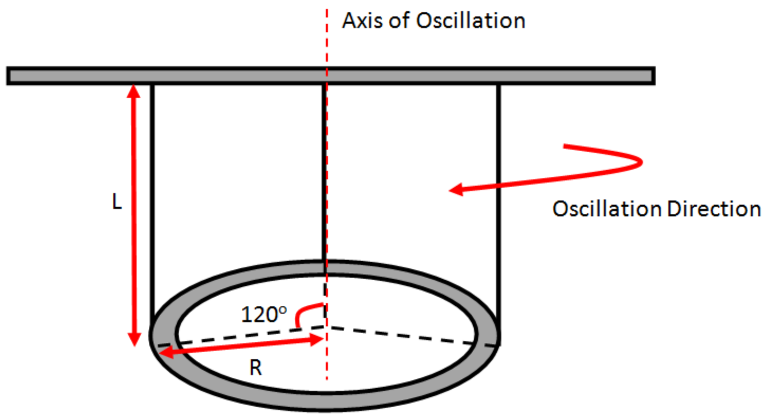
The Iris+ mass moment of inertia can be determined with 2 methods. The first method involves an experimental approach that was described in [21] and [10], where a trifilar pendulum is used to measure the Iris+ oscillations along each of its body axis. Subsequently, the oscillations measured are used to compute the Iris+ mass moment of inertia.

The second method involves an analytical approach, where each component on the Iris+ (e.g., arms, motors, body, etc.) can be approximated as regular shapes (i.e., beam or cylinders). The mass moment of inertia for each of the regular shapes can be found individually and summed together using the parallel axis theorem described in [22] to obtain the mass moment of inertia of the Iris+ with respect to its CG.

Both methods were used to determine the mass moment of inertia for the Iris+ and the procedures are elaborated in the next section. Finally, the mass moment of inertia obtained using both methods will be compared to determine if the differences are significant (i.e., more than 20%).

1. Experimental Method

The trifilar pendulum is an established methodology in determining the mass moment of inertia of objects with irregular shapes that cannot be calculated directly. This methodology was well elaborated in [21] and used to determine the Iris+ mass moment of inertia. To determine an irregular shaped object's mass moment of inertia, the trifilar pendulum holding the object is made to rotate along the z-axis and the period of a single oscillation is measured over three iterations; the mean value was calculated at the end. Subsequently, the period is used to calculate the mass moment of inertia of the irregular shaped object. The setup of the trifilar pendulum comprises of a disc that is hung from the ceiling with 3 wires that are fasten to the disc at an equal distance from each other (see Figure 14). Before the mass moment of inertia of the object can be determined with the trifilar pendulum, the weight of the disc, its radius and the length of the wire would need to be measured first (see Table 6 for the measurement results of the experimental setup).



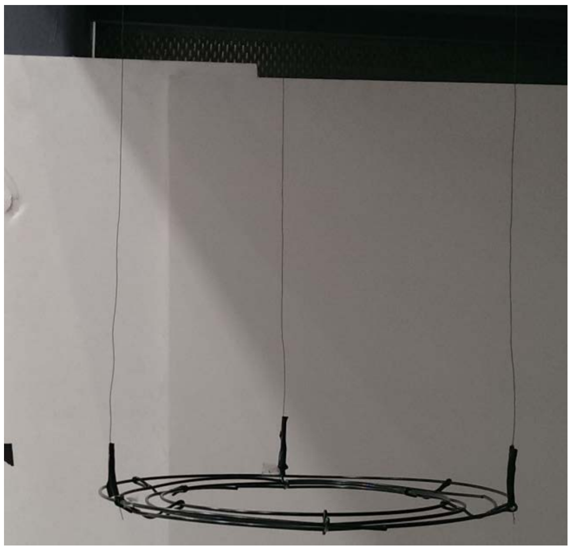


Figure 14 Trifilar Pendulum Setup.

The configuration of the quadrotor's mass moment of inertia along its three principal axes is shown in Figure 15. To determine the quadrotor's mass moment of inertia along any axis (i.e. I_{xx}, I_{yy} and I_{zz}), the quadrotor was placed on the stand and axis of interest was aligned with the trifilar pendulum's axis of oscillation. The 3 configurations of measuring the quadrotor's I along the x-axis, y-axis and z-axis are shown in Figure 16, Figure 17 and Figure 18 respectively. Subsequently, a small angular displacement will be introduced to the trifilar pendulum to rotate the stand holding the quadrotor and the period of 10 rotations/oscillations are measured. The measurement of each period was repeated twice and the total period from the three experimental runs were averaged to reduce the effects of experimental random errors.

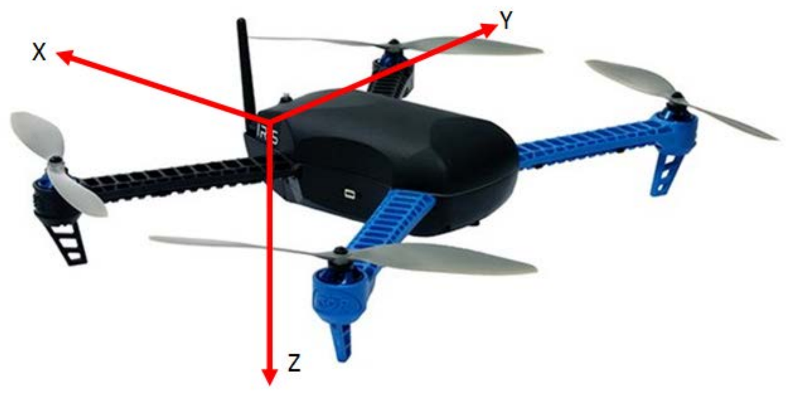


Figure 15 Quadrotor Rotation Axes Configuration.

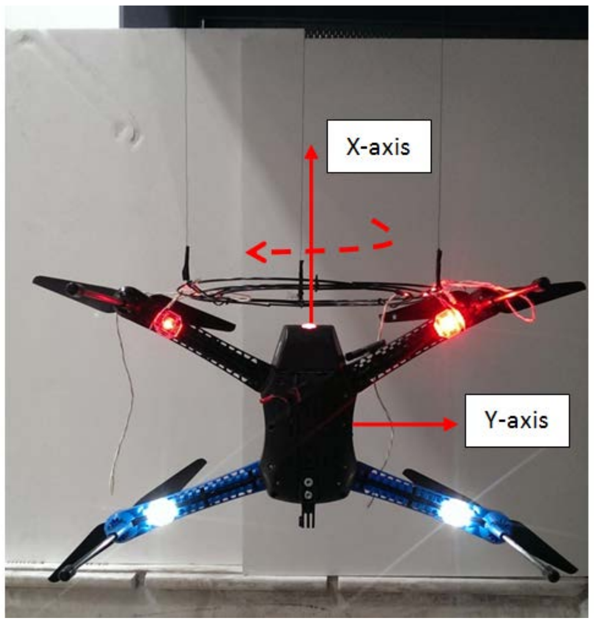


Figure 16 Measurement of Mass Moment of Inertia along x-axis.

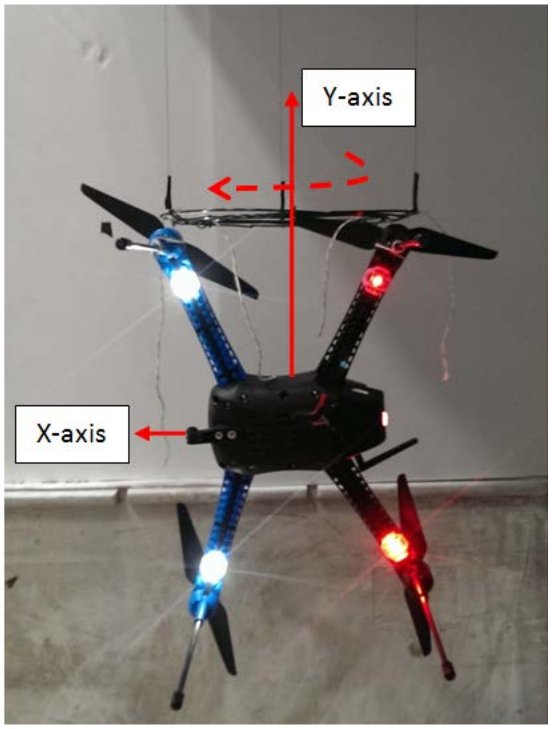


Figure 17 Measurement of Mass Moment of Inertia along y-axis.

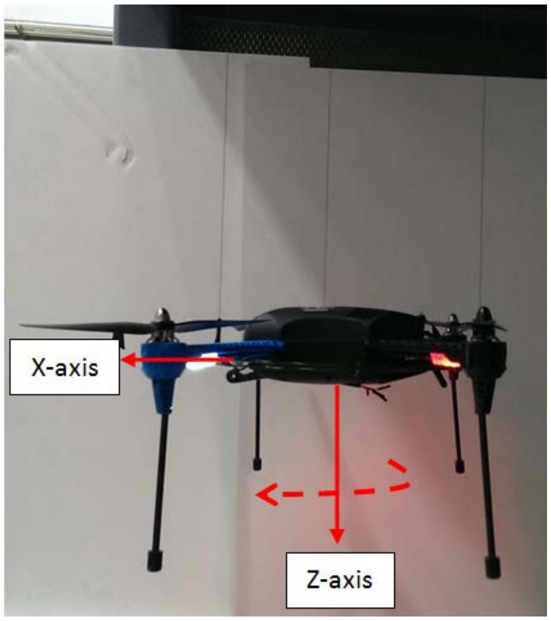


Figure 18 Measurement of Mass Moment of Inertia along z-axis.

With the measured period, the quadrotor mass moment of inertia along each principal axis can be calculated using equation (37):

$$I_{xx,yy,zz} = \frac{W \cdot R^2 \cdot T_{x,y,z}^2}{4 \cdot \pi^2 \cdot L} \tag{37}$$

where: $I_{xx,yy,zz}$ = mass moment of inertia of object in x, y or z-axis,

T = period of one oscillation in s,

W = weight of the disc and quadrotor in kg,

R = radius of disc in m,

L = length of wire suspending disc from ceiling in m.

Equation (37) assumes that the weight and the moment of inertial of the rotating disk that holds the “irregular” body is negligible, therefore their contribution is omitted in (37) for brevity.

The results from the measurement of the periods along each principal axis are summarized in Table 5. Using equation (37), the mass moment of inertia along each principal axis was calculated and summarized in Table 6.

Table 5 Periods of Oscillation in x, y and z Axes.

	10 Oscillations				1 Oscillation
	1st Run	2nd Run	3rd Run	Average	
Period in x-axis, T_x (s^{-1})	15.5	15.5	15.2	15.40	1.54
Period in y-axis, T_y (s^{-1})	11.3	11.3	11.4	11.33	1.13
Period in z-axis, T_z (s^{-1})	17.9	18.0	18.0	17.67	1.77

Table 6 Iris+ Quadrotor Mass Moment of Inertia in x, y and z Axes.

Setup Specifications		Computed Mass Moment of Inertia (kg.m^2)	
Disc Weight (kg)	0.120	x-axis, I_{xx}	0.0219
Disc Radius (m)	0.152	y-axis, I_{yy}	0.0109
Wire Length (m)	0.860	z-axis, I_{zz}`	0.0306

2. Analytical Method

In the analytical method, the individual components of the Iris+ quadrotor can be approximated as the shapes shown in Figure 19 and their dimensions are shown in Figure 7. The dimensions of each component are measured using the measuring tape and weighing scale introduced in Section A.

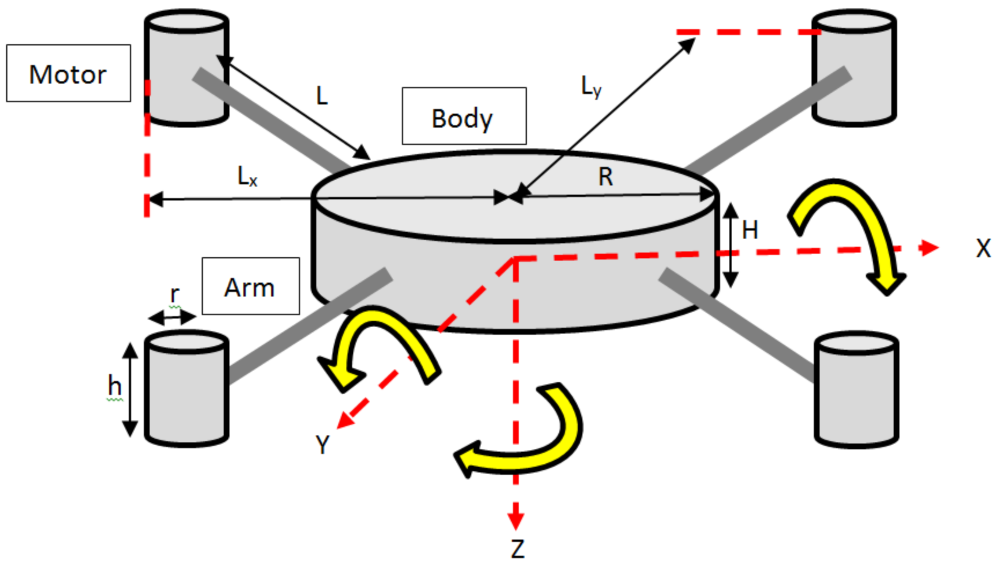


Figure 19 Approximated Shapes and Dimensions for Iris+

Table 7 Dimensions of Iris+ Components

	Mass (kg)	Radius, R (m)	Height, H (m)	Length, L (m)
Body	0.816	0.1	0.07	-
Arm	0.0685	-	-	0.20
Motor	0.070	0.015	0.03	-
Distance between Motor CG and Iris+ CG, L_M		-	-	0.25
Distance between Arm CG and Iris+ CG, L_A		-	-	0.165

The mass moment of inertia for the body and motors can be approximated as cylinders, while the mass moment of inertia of the arm can be approximated as a beam. Therefore, the mass moment of inertia of the Iris+ quadrotor along the x, y and z axes (i.e. I_{xx}, I_{xx} and I_{xx}) can be found using the parallel axis theorem defined in [22] for the approximated shapes, with the following equations:

$$\begin{aligned}I_{xx} &= 4\left(I_{motor}\right) + 2(m_{motor}\cdot L_{yF}{}^{2} + m_{motor}\cdot L_{yB}{}^{2}) + 4(I_{arms} + m_{arms}\cdot\left(\frac{L}{2}\right)^{2}) + I_{body} \\ &= 4\left(\frac{m_{motor}\cdot r^{2}}{4} + \frac{m_{motor}\cdot h^{2}}{12}\right) + 2\left(m_{motor}\cdot L_{yF}{}^{2} + m_{motor}\cdot L_{yB}{}^{2}\right) \\ &+4\left(\frac{m_{arm}\cdot L^{2}}{12} + m_{arm}\cdot\left(\frac{L}{2}\right)^{2}\right) + \left(\frac{m_{body}\cdot R^{2}}{4} + \frac{m_{body}\cdot H^{2}}{12}\right)\end{aligned} \quad (38)$$

$$\begin{aligned}I_{yy} &= 4\left(I_{motor}\right) + 2(m_{motor}\cdot L_{xF}{}^{2} + m_{motor}\cdot L_{xB}{}^{2}) + 4(I_{arms} + m_{arms}\cdot\left(\frac{L}{2}\right)^{2}) + I_{body} \\ &= 4\left(\frac{m_{motor}\cdot r^{2}}{4} + \frac{m_{motor}\cdot h^{2}}{12}\right) + 2\left(m_{motor}\cdot L_{xF}{}^{2} + m_{motor}\cdot L_{xB}{}^{2}\right) \\ &+4\left(\frac{m_{arm}\cdot L^{2}}{12} + m_{arm}\cdot\left(\frac{L}{2}\right)^{2}\right) + \left(\frac{m_{body}\cdot R^{2}}{4} + \frac{m_{body}\cdot H^{2}}{12}\right)\end{aligned} \quad (39)$$

$$\begin{aligned}I_{zz} &= 4(I_{motor} + m_{motor}\cdot L_{M}{}^{2}) + 4\left(I_{arm} + m_{arm}\left(L_{A}\right)^{2}\right) + I_{body} \\ &= 4\left(\frac{m_{motor}\cdot r^{2}}{2} + m_{motor}\cdot L_{M}{}^{2}\right) + 4\left(\frac{m_{arm}}{12}\cdot L^{2} + m_{arm}\cdot L_{A}{}^{2}\right) + \left(\frac{m_{body}\cdot R^{2}}{2}\right)\end{aligned} \quad (40)$$

Substituting the dimensions in Table 7 into the equations (38), (39) and (40), the mass moment of inertia of the Iris+ quadrotor along each axis (found using the analytical method) is:

I_{xx} = 0.0238 kg.m^2,

I_{yy} = 0.00882 kg.m^2,

I_{zz} = 0.0303 kg.m^2.

3. Comparison between Experimental and Analytical Methods

The comparison between the mass moment of inertia obtained using the experimental and analytical methods are shown in Table 8. Comparing between the results obtained from both methods, the percentage difference in the mass moment of inertia are 8.7%, 23.6% and 0.03% for the x, y and z-axes, respectively.

Table 8 Mass Moment of Inertia for Experimental and Analytical Methods

	Experimental (kg.m^2)	Analytical (kg.m^2)	Percentage Difference
I_{xx}	0.0219	0.0238	8.7%
I_{yy}	0.0109	0.00882	23.6%
I_{zz}	0.0306	0.0303	0.03%

It can be seen from Table 8 that the mass moment of inertia in the x and z-axes found using the experimental and analytical methods are relatively small (i.e. 8.7% and 0.03% respectively). The difference for the mass moment of inertia in the y-axis is more pronounced (i.e. 23.6%) and this could be attributed to the overly simplistic representation of the Iris+ distribution of masses. First, the spread angle of the front arms (i.e. 120^o) is different from the separation of the back arms (i.e. 140^o) that leads to the difference in separation of masses along x and y-axes. Next, the simplified representation of the center-body as a cylinder is also a minor contributor to the difference. Nevertheless, the difference between the experimental and analytical method for the y-axis is only slightly over 20%. Therefore, since the mass moment of inertia found using both methods are of the same order of magnitude, the mass moment of inertia that was obtained using the experimental method can be used in the Iris+ quadrotor mathematical model.

C. QUADROTOR PROPELLER COEFFICIENTS

The propellers installed on the Iris+ quadrotor were manufactured by APC and have a dimension of 10” by 4.7” in diameter and pitch respectively (see

Figure 20). A test stand experiment setup that rotates the propeller at a specified speed and measuring the thrust force generated can be used to determine the propeller's thrust coefficient, C_T and torque coefficient, C_Q.

Figure 20 APC 10" by 4.7" Propeller Set (from [23]).

The University of Illinois at Urbana-Champaign (UIUC) had performed a series of experiments as described in [24] and [25] to determine the performance of different small-scale propellers at low Reynolds number. The experimental results for the APC propeller of dimension 10" by 4.7" were summarized in Table 9, while the plots for the thrust and power coefficients against the propeller speed are shown in Figure 21 and Figure 22.

Table 9 APM (10" x 4.7") Propeller Thrust & Power Coefficients at Different Speed (from [26]).

Propeller Rotational Speed (RPM)	Propeller Rotational Speed (rad/s)	Thrust Coefficient, C_T	Power Coefficient, C_P
2377	248.9189	0.1059	0.0431
2676	280.2301	0.1079	0.0437
2947	308.6091	0.1079	0.0437
3234	338.6637	0.1104	0.0444
3494	365.8908	0.1117	0.0450
3762	393.9557	0.1143	0.0460
4029	421.9159	0.1158	0.0466
4319	452.2846	0.1177	0.0474
4590	480.6637	0.1200	0.0484
4880	511.0324	0.1223	0.0494
5147	538.9926	0.1237	0.0500
5417	567.2669	0.1252	0.0508
5715	598.4734	0.1263	0.0513
5960	624.1297	0.1278	0.0520
6226	651.9852	0.1286	0.0524
6226	651.9852	0.1286	0.0531

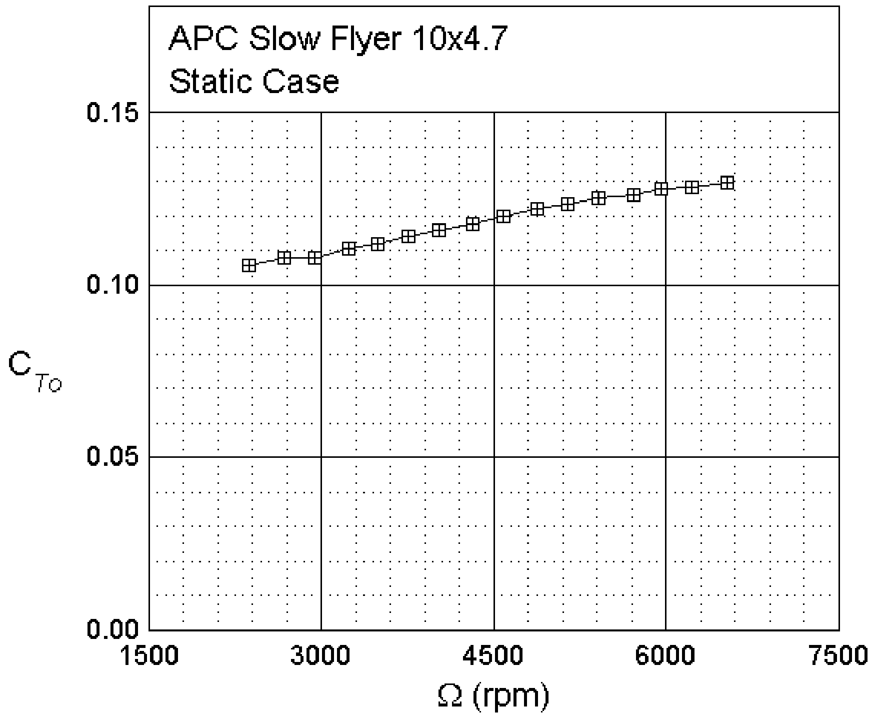


Figure 21 Thrust Coefficient vs Propeller Speed Plot (from [26]).

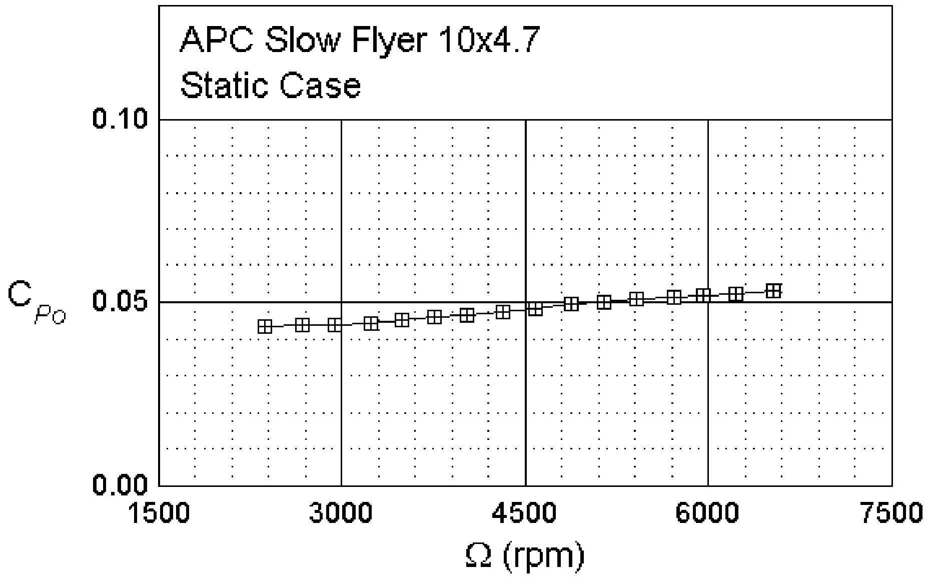


Figure 22 Power Coefficient vs Propeller Speed Plot (from [26]).

The relationship between the thrust generated by a propeller, T_i and the propeller's rotational velocity, ω_i can be expressed in equation (42). Therefore, the thrust generated by a propeller can be found if the propeller's rotational speed and coefficient of thrust is known.

$$T_i = C_T \cdot \rho \cdot D^4 \cdot 4 \cdot \pi^4 \cdot \omega_i^2 \qquad (41)$$

$$\therefore T_i = K_T \cdot \omega_i^2 \qquad (42)$$

where: C_T = propeller thrust coefficient,

ρ = air density,

D = propeller diameter,

K_T = thrust constant representing the product of C_T, ρ and D^4.

The relationship between the torque generated by a propeller, Q_i and the propeller's rotational velocity, ω_i can be expressed in equation (46) as referenced from [25]. Although the torque coefficient, C_Q was not determined directly in the experiments conducted by UIUC, it is related to the power coefficient, C_P shown in equation (44). Therefore, the torque generated by a propeller can be found if the propeller's rotational speed and coefficient of power and coefficient torque are known.

$$Q_i = C_Q \cdot \rho \cdot D^5 \cdot 4 \cdot \pi^2 \cdot \omega_i^2 \tag{43}$$

$$C_Q = \frac{C_P}{2\pi} \tag{44}$$

$$Q_i = \frac{C_P}{2\pi} \cdot \rho \cdot 4 \cdot \pi^2 \cdot \omega_i^2 \tag{45}$$

$$\therefore Q_i = K_Q \cdot \omega_i^2 \tag{46}$$

where: C_Q = propeller torque coefficient,

C_P = propeller power coefficient,

K_Q = torque constant representing the product of C_P, ρ and D.

THIS PAGE INTENTIONALLY LEFT BLANK

V. QUADROTOR FLIGHT CONTROL DESIGN

This chapter presents the process of designing the flight control algorithm for the Iris+ quadrotor based on the mathematical model developed in Chapter III. The PID controller was chosen as the control technique to stabilize (control attitude) the Iris+ model in flight and the method to tune the PID controller is elaborated as well in this chapter. The sections in this chapter are as follow:

Section A provides an overview to the control modeling of the quadrotor and the process for controlling the quadrotor to track a planned trajectory.

Section B gives a brief overview of the PID controller that will be employed on the quadrotor model.

Section C describes the Ziegler-Nichols method that is employed to tune the PID controller for the Iris+ attitude and rates control loops.

A. CONTROL MODELLING OF QUADROTOR

The control modeling of the Iris+ quadrotor is summarized and presented in the block diagram as shown in Figure 23 below. A hierarchical approach that was presented in [7] was used to allocate the control bandwidths required by the control actions in the block diagram: motor mixer allocation, controller blocks. In this approach, higher control bandwidths are allocated to the lower level control actions. Therefore, the motor speed allocation receives the highest control bandwidth as it is located at the lowest level, while the outer loop controller block receives the lowest control allocation, as it is located at the highest level. The functions of each block are further elaborated below.

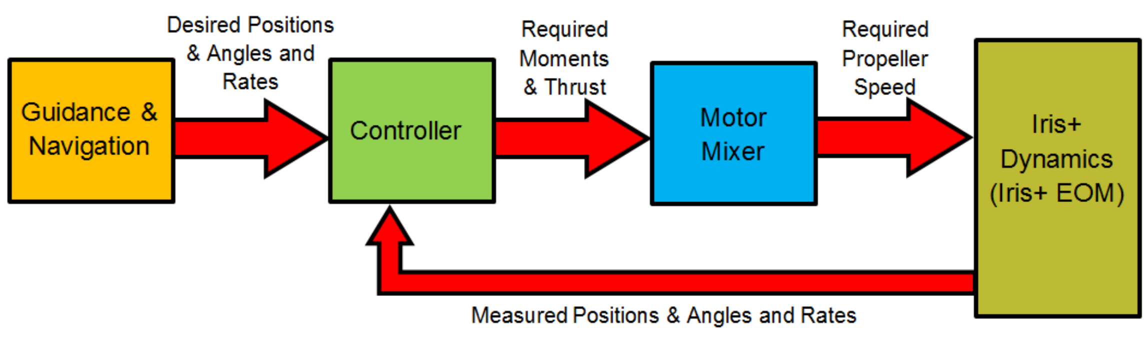

Desired Positions & Angles and Rates
Guidance & Navigation
Controller
Required Moments & Thrust
Motor Mixer
Required Propeller Speed
Iris+ Dynamics (Iris+ EOM)
Measured Positions & Angles and Rates

Figure 23 Block Diagram of Quadrotor Control Model.

1. Trajectory Generator Block

In Simulink, the trajectory generator block computes the desired flight trajectory that needs to be followed by the quadrotor and generates the set of desired positions (i.e., x_d, y_d and z_d) and desired Euler angles (i.e. ϕ_d, θ_d and ψ_d) in the inertial coordinate frame for that trajectory. If a Remote Control (RC) is used to generate flight command during an actual flight, the trajectory generator block will output the desired Euler angles or their rates to the Iris+ quadrotor; the choice depends on the control architecture. The desired positions and Euler angles are used as inputs for the controller block to stabilize the Iris+ in flight during the change in the Iris+ attitude.

2. Controller Block

The control of the quadrotor's position and attitude is accomplished by the design of the feedback controller and the method was well documented in [6] and [7]. As mentioned in the previous chapters, the quadrotor is an under-actuated system. Therefore, to move forward in the 'x' direction, the quadrotor must first change its attitude by pitching downwards to generate a horizontal force from the propellers' thrusts, while maintaining its altitude. Similarly, in order to move laterally in the 'y' direction, the quadrotor must change its attitude by rolling to the right or left while maintaining its altitude. Therefore, the control equations of the quadrotor's position and attitude channels are shown in equations (47) and (48).

The complete derivation of the control equations can be found in Annex A of this thesis.

1. Attitude Control Equations

$$\phi_d = K_{1,y} \cdot \left[R_I^V \left(y_d - y \right) - K_{2,y} \cdot \dot{y}_V \right] \tag{47}$$

$$\theta_d = K_{1,x} \cdot \left[R_I^V \left(x_d - x \right) - K_{2,x} \dot{x}_V \right] \tag{48}$$

where: R_I^V is the rotation matrix that transforms the quadrotor's position from the inertia to vehicle coordinate frame,

$K_{1,x}$ is the proportional gain term for the position in the 'x' direction,

$K_{2,x}$ is the derivative gain term for the velocity in the 'x' direction,

$K_{1,y}$ is the proportional gain term for the position in the 'y' direction,

$K_{2,x}$ is the derivative gain term for the velocity in the 'y' direction.

2. Forces and Moments Control Equations

$$\tau_\phi = K_{P,roll} \cdot (\phi_d - \phi_m) + K_{I,roll} \cdot (\phi_d - \phi_m) + K_{D,roll} \cdot (\dot{\phi}_d - \dot{\phi}_m) \tag{49}$$

$$\tau_\theta = K_{P,\mathrm{pitch}} \cdot (\theta_d - \theta_m) + K_{I,\mathrm{pitch}} \cdot (\theta_d - \theta_m) + K_{D,\mathrm{pitch}} \cdot (\dot{\theta}_d - \dot{\theta}_m) \tag{50}$$

$$\tau_\psi = K_{P,\mathrm{yaw}} \cdot (\psi_d - \psi_m) + K_{I,\mathrm{yaw}} \cdot (\psi_d - \psi_m) + K_{D,\mathrm{yaw}} \cdot (\dot{\psi}_d - \dot{\psi}_m) \tag{51}$$

$$T = K_{P,z}(z_d - z) + K_{I,z}(z_d - z) + K_{\mathrm{D},z}(\dot{z}_d - \dot{z}) + \omega_0 \tag{52}$$

$$\omega_0 = \sqrt{\frac{m \cdot g}{4 \cdot K_T}} \tag{53}$$

where: $K_{P,roll/pitch/yaw}$ is the proportional gain term for the roll, pitch or yaw angles,

$K_{I,roll/pitch/yaw}$ is the Integral gain term for the roll, pitch or yaw angles,

$K_{D,roll/pitch/yaw}$ is the derivative gain term for the roll, pitch or yaw rates,

$K_{P,z}$ is the proportional gain term for the position along the z-axis,

$K_{I,z}$ is the integral gain term for the position along the z-axis,

$K_{D,z}$ is the derivative gain term for the velocity along the z-axis,

ω_0 is the motor rotational speed required to generate a thrust that is equal to the weight of the quadrotor.

3. Motor Mixer Block

The motor mixer block computes the required angular speed of each propeller in order to generate the thrust force and moments (i.e., T^b, τ_ϕ, τ_θ, τ_ψ) to perform a maneuver or changing the quadrotor's attitude in flight. As stated in Chapter III, the thrust and moments of the quadrotor are directly proportional to the square of the propellers' angular speed (i.e., ω_i^2). The proportional relationship between the thrust and moments of the quadrotor and each propeller's angular speed is shown in equation (54), represented by the matrix, M. Therefore, to determine the angular speed required for each propeller in order to generate the thrust force and moment, the inverse of M is first derived and multiplied to the thrust, roll, pitch and yaw moment vector as shown in the equation (55). The calculated angular speed for each propeller will be used as an input for the quadrotor dynamics block.

$$\begin{bmatrix} T^b \\ \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix} = \begin{bmatrix} -K_T & -K_T & -K_T & -K_T \\ -K_T \cdot l_y & -K_T \cdot l_y & K_T \cdot l_y & K_T \cdot l_y \\ K_T \cdot l_x & -K_T \cdot l_x & -K_T \cdot l_x & K_T \cdot l_x \\ K_D & -K_D & K_D & -K_D \end{bmatrix} \begin{bmatrix} \omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \\ \omega_4^2 \end{bmatrix} = \left[M\right] \begin{bmatrix} \omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \\ \omega_4^2 \end{bmatrix} \tag{54}$$

$$\therefore \begin{bmatrix} \omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \\ \omega_4^2 \end{bmatrix} = \left[M\right]^{-1} \begin{bmatrix} T^b \\ \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix} \tag{55}$$

4. Iris+ Quadrotor Dynamics Block

The Iris+ dynamics block consists of the 6DOF EOM that was derived in Chapter III. The required angular speed for each propeller is provided by the motor mixer block to the Iris+ dynamics block as an input and the body response is measured by the sensors onboard the Iris+ as the measured position, Euler angles and their rates. The measured responses are fed back to the controller block and used to determine the error signals for each of the position and Euler angle channel.

B. FLIGHT CONTROLLER IMPLEMENTATION IN SIMULINK

Using the concept formulated in section A.4 of this chapter for the control modeling of the Iris+ quadrotor, a Simulink model was built as shown in Figure 24. The Simulink model is used to tune the PID controller gains and simulate the Iris+ dynamics with the gain values to obtain a first cut flight performance, before using the gains for the Pixhawk autopilot.

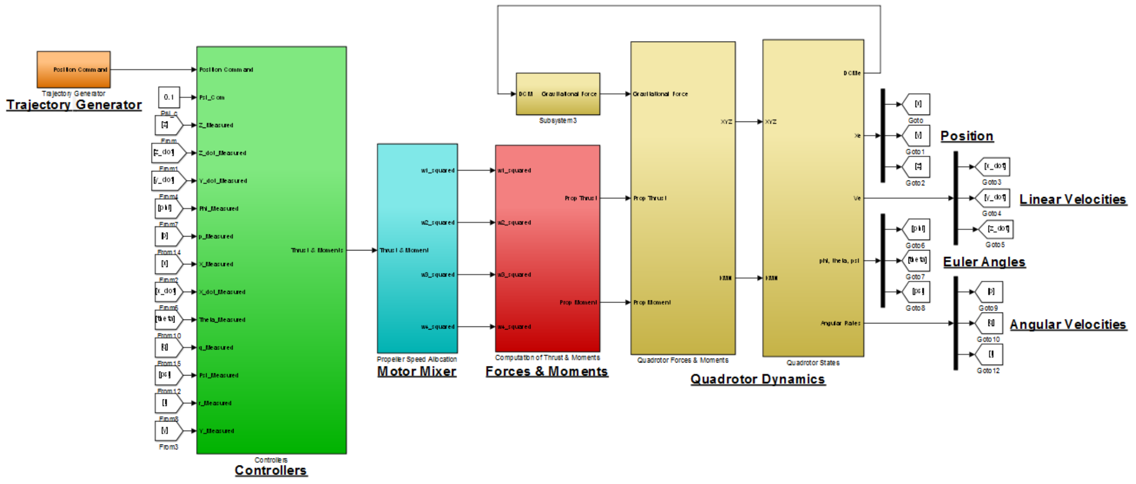


Figure 24 Simulink Model of Iris+ Quadrotor.

C. OVERVIEW OF PID CONTROLLER

The quadrotor model in this thesis uses a PID controller that was elaborated in [27], [28] and [29] to stabilize the Iris+ attitude during flight. A PID controller consists of three tunable gain values: Proportional gain (i.e. K_P), the Integral gain (i.e. K_I) and the Derivative gain (i.e. K_D) as shown in Figure 25. The transfer function of a PID controller can be represented by equation (56).

$$G(S) = K_P + \frac{K_I}{s} + K_D \cdot s \tag{56}$$

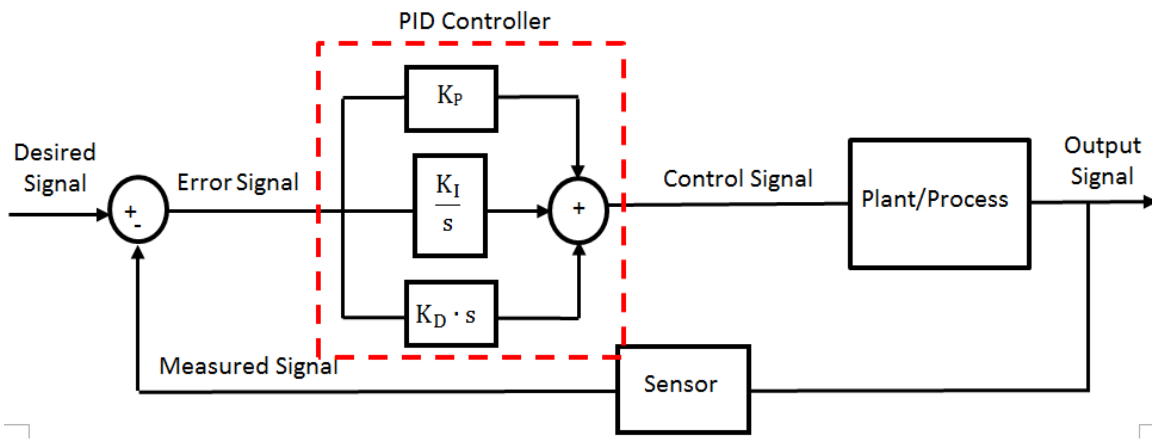


Figure 25 Illustration of System designed with PID Controller.

Each gain in the PID controller can be tuned to modify a particular transient response parameter of the feedback system (see Figure 26) and the effects from increasing each gain value separately is further elaborated below:

1. Proportional Gain, K_P

The K_P value is increased to reduce the time required for the output signal to reach the desired signal (i.e., system response time, t_r). By increasing the K_P value alone in the PID controller, a steady-state error can be reduced and expected to be between the desired signal and the output signal. In addition, setting an overly high K_P value will also propagate any inherent disturbance signal within the system and cause the system to undergo unstable oscillations.

2. Integral Gain, K_I

The K_I value is increased to eliminate the steady-state error of the feedback system. However, as the integral term introduces a pole at the origin of an S-plane plot, the system might become increasingly unstable when the K_I value is increased (i.e., the system will become increasingly oscillatory in the steady-state).

3. Derivative Gain, K_D

The K_D value is increased to reduce the overshoot, M_P and the settling time, t_d of the feedback system's output signal. Although derivative control does not affect the steady-state error directly, it introduces damping to the feedback system. This would allow the system to use a larger K_P value, which would result in improvement to the system's steady-state performance. As described in [27] and [20], "derivative control operates on the rate of change of the actuating error and not the actuating error itself this mode is never used alone." Therefore, K_D gain is generally used in combination with K_P and K_I control actions.

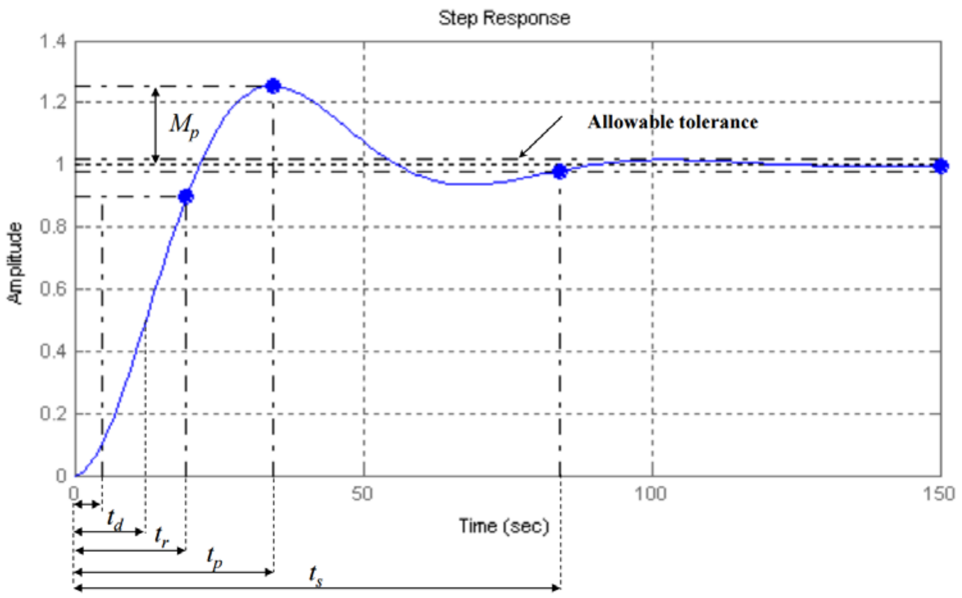


Figure 26 Transient Response for a Feedback System (from [20]).

D. IMPLEMENTATION OF PID CONTROLLERS IN ATTITUDE AND RATES CHANNEL IN SIMULINK

With reference to the concept of a PID controller above, the PID controller was implemented to the attitude and attitude rate control channels for the Iris+ quadrotor in Simulink (see Figure 27 to Figure 32). The PID controllers for the

Iris+ Simulink model will be tuned using the method outlined in the next section to obtain the optimal PID gains for a stabilize flight performance.

1. Attitude Controllers

In linear settings, the outer loop controllers that operate on the Iris+ command attitude rates (i.e. $\dot{\phi}_{com}$, $\dot{\theta}_{com}$ and $\dot{\psi}_{com}$) can be obtained by taking the difference between the Iris+'s command and measured attitude (i.e., error term, $e_{\phi/\theta/\psi}(t)$) and multiplying it with a proportional gain term (i.e. $K_{P,\phi/\theta/\psi}$). The proportional control law equations for the attitude rates are shown in equations (57) through (59) and these equations can be represented in Simulink as shown in Figure 27, Figure 28 and Figure 29:

$$\dot{\phi}_{com} = K_{P,\phi}(\phi_{com} - \phi_{meas}) = K_{P,\phi} \cdot e_{\phi}(t) \tag{57}$$

$$\dot{\theta}_{com} = K_{P,\theta}(\theta_{com} - \theta_{meas}) = K_{P,\theta} \cdot e_{\theta}(t) \tag{58}$$

$$\dot{\psi}_{com} = K_{P,\psi}(\psi_{com} - \psi_{meas}) = K_{P,\psi} \cdot e_{\psi}(t) \tag{59}$$

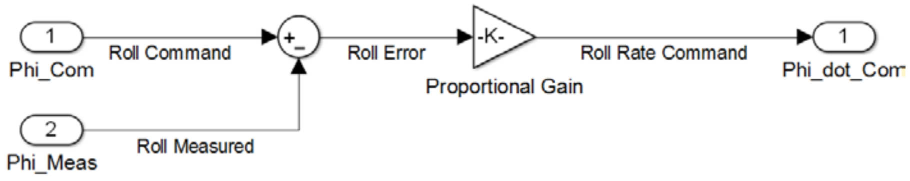


Figure 27 Roll Channel Controller.

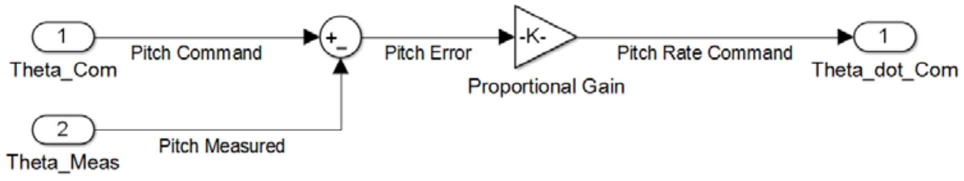


Figure 28 Pitch Channel Controller.

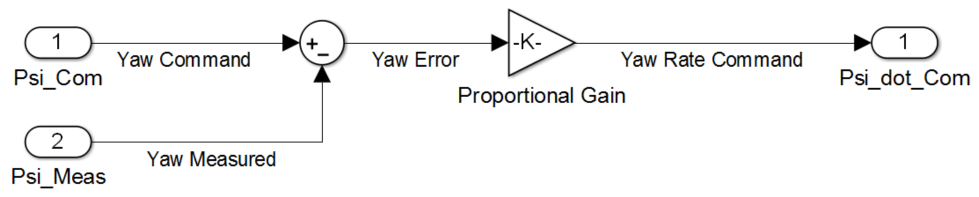


Figure 29 Yaw Channel Controller.

2. Attitude Rate Controllers

In linear settings, the inner loop controller that operates on the Iris+ command moments along the x, y and z-axes (i.e. τ_{roll}, τ_{pitch} and τ_{yaw}) can be obtained by using the difference between the command attitude rates from section 1 and the measured attitude rates from the Iris+ sensors (i.e. $\dot{\phi}_{meas}$, $\dot{\theta}_{meas}$ and $\dot{\psi}_{meas}$). Subsequently, the difference between the command and measured attitude rates were multiplied with proportional, integral and derivative gains (i.e. $K_{P,\phi/\theta/\Psi}$, $K_{I,\phi/\theta/\Psi}$ and $K_{D,\phi/\theta/\Psi}$) and summed to shape the controllers' transient and steady state performance. The control equations for the moments are shown in equations (60) through (62) and these equations can be represented in Simulink as shown in Figure 30, Figure 31 and Figure 32:

$$\tau_{roll} = K_{P,\dot{\phi}}(\dot{\phi}_{com} - \dot{\phi}_{meas}) + \frac{K_{I,\dot{\phi}}}{s}(\dot{\phi}_{com} - \dot{\phi}_{meas}) + \frac{d}{dt}K_{d,\dot{\phi}}(\dot{\phi}_{com} - \dot{\phi}_{meas}) \tag{60}$$

$$\tau_{pitch} = K_{P,\dot{\theta}}(\dot{\theta}_{com} - \dot{\theta}_{meas}) + \frac{K_{I,\dot{\theta}}}{s}(\dot{\theta}_{com} - \dot{\theta}_{meas}) + \frac{d}{dt}K_{d,\dot{\theta}}(\dot{\theta}_{com} - \dot{\theta}_{meas}) \tag{61}$$

$$\tau_{yaw} = K_{P,\dot{\psi}}(\dot{\psi}_{com} - \dot{\psi}_{meas}) + \frac{K_{I,\dot{\psi}}}{s}(\dot{\psi}_{com} - \dot{\psi}_{meas}) + \frac{d}{dt}K_{d,\dot{\psi}}(\dot{\psi}_{com} - \dot{\psi}_{meas}) \tag{62}$$

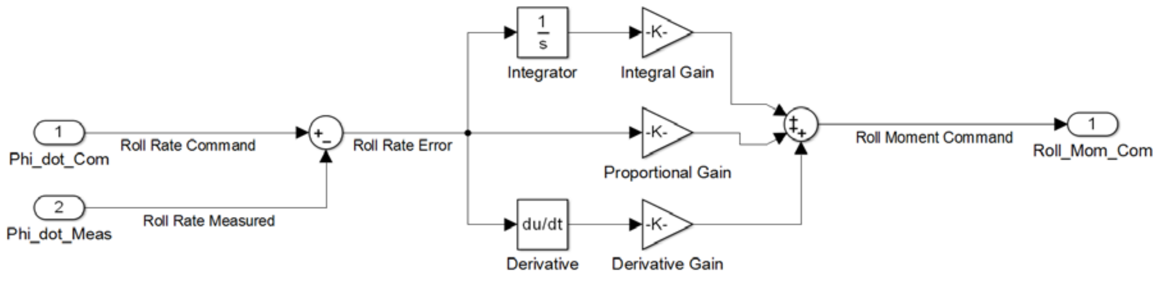


Figure 30 Roll Moment Controller.

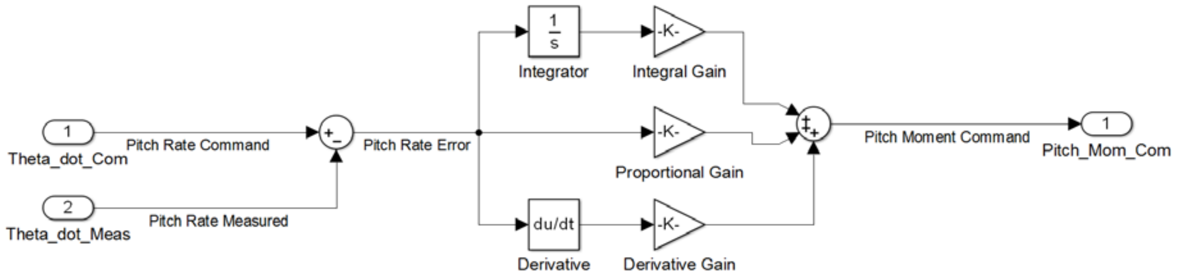


Figure 31 Pitch Moment Controller.

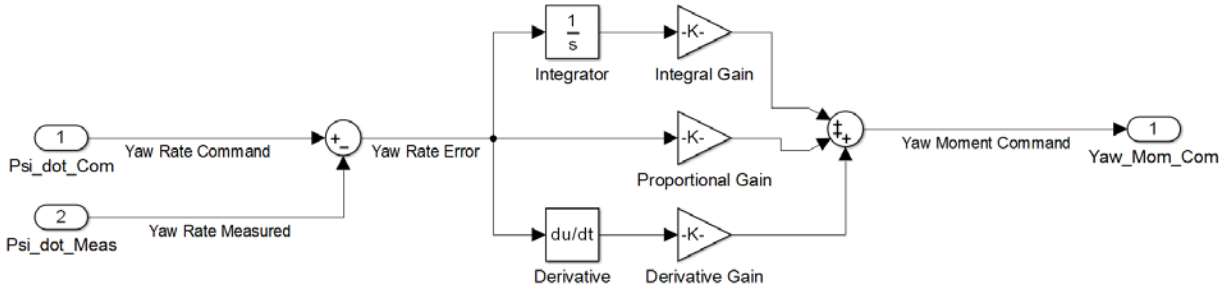


Figure 32 Yaw Moment Controller.

E. PID CONTROLLER TUNING METHOD

With the Simulink model developed for the Iris+ and the PID controllers in the previous section, a systematic approach would need to be adopted to tune the PID controllers and obtain optimal gain values for the Iris+ to attain stabilized flight. The Ziegler-Nichols rule elaborated in [27] and [20] for tuning PID controllers was selected as a suitable method for this thesis to determine the gains of the Iris+ PID controllers. There are two methods for applying the Ziegler-Nichols rule:

1. Ziegler-Nichols First Method

The first method for applying the Ziegler-Nichols rule involves an experiment on the plant or system to obtain the response plot due to a unit step command. If the plant or system does not feature an integrator or dominant complex conjugate poles, the plant or system’s response plot to a unit step response will take the shape of an S curve as shown in Figure 33. Therefore, the first method for the Ziegler-Nichols rule is only valid if the response plot takes the shape of an S. Since this method is restricted by whether the plant or system possess an integrator or dominant complex conjugate poles, the second method was used for tuning the PID controller instead.

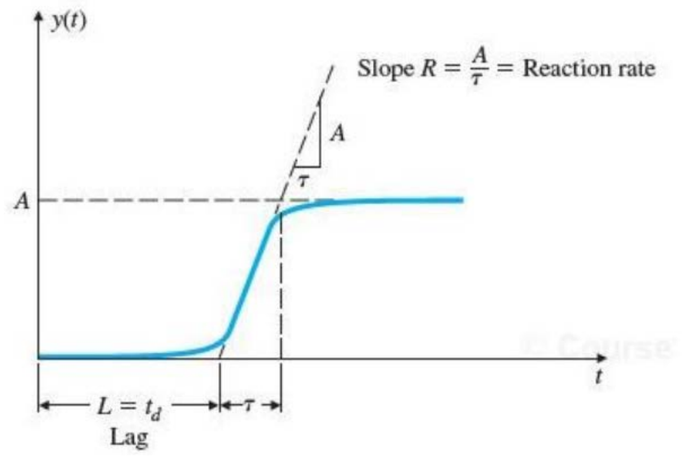


Figure 33 S-Shaped Response Curve to a Unit Step Command (from [20]).

2. Ziegler-Nichols Second Method

The second method for applying the Ziegler-Nichols rule also adopts an experimental approach to determine the PID gain values for a plant or system. In this method, the proportional gain for a control loop (each control loop is tuned individually) is increased to a critical value where the output of the channel exhibits oscillatory behavior. This proportional gain is also known as the critical gain value, K_{cr} that causes the system output to oscillate. The corresponding period of K_{cr} (i.e. P_{cr}) is determined from the plot of the system output. Using the K_{cr} and P_{cr} values found, the gains for the PID controller in each loop can be found using the relation defined in Table 10. The transfer function for the PID controller using the Ziegler-Nichols rule is shown in equation (57).

Table 10 PID Controller Gains using Ziegler-Nichol Methods (from [20]).

Type of Controller	K_P	T_I	T_D
Proportional only	$0.5K_{cr}$	∞	0
Proportional & Integral only	$0.45K_{cr}$	$\frac{1}{12}P_{cr}$	0
Proportional, Integral & Derivative	$0.6K_{cr}$	$0.5P_{cr}$	$0.125P_{cr}$

$$G_C(S) = K_P(1+\frac{1}{T_I \cdot s}+T_D \cdot s) = 0.6 \cdot K_{cr} \cdot (1+\frac{1}{0.5 \cdot P_r \cdot s}+0.125 \cdot P_{cr} \cdot s) \qquad (63)$$

Using the second method for the Ziegler-Nichols rule, the gains for the PID controller in each control loop were determined individually with the Simulink model developed.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. IMPLEMENTATION OF CONTROLLER ON PIXHAWK

This chapter presents the process of implementing the Simulink model of the PID controller that was developed in Chapter V onboard of the Pixhawk autopilot. The outline of this chapter is as follows:

Section A describes the methods that were considered and selected to implement the Simulink model onto the Pixhawk autopilot.

Section B elaborates on using the attitude controller Simulink model found in the PX4 toolchain and the conversion of the model to C/C++ code using the Simulink 'Build' function.

Section C describes the process of using the Eclipse software to build the Simulink controller model into an application and downloading the application to the Pixhawk autopilot.

A. METHODS TO IMPLEMENT SIMULINK MODEL ON PIXHAWK

MathWorks had developed the Simulink coder and Embedded coder to convert Simulink models to C/C++ codes using the 'Build' function in Simulink. The C/C++ codes generated can be used to build an application that is downloaded to the Pixhawk autopilot using the methods described in [13] or [12]. The methods to implement the Simulink model onto the Pixhawk autopilot was elaborated with full details in Appendix B.

B. SETUP OF ATTITUDE CONTROLLER SIMULINK MODEL

To implement the software development starting at a Simulink model and transitioning to the PX4 application that can be run on the autopilot, the PX4 toolchain and PX4 PSP would first need to be downloaded and installed from PX4's website [30] and MathWorks' website [19]. The PX4 toolchain and Pixhawk PSP comes with a library of PX4 Simulink blocks and the various subsystems found in the Iris+ (e.g., controller model, plant model, etc.). As a first step in developing the controller for this thesis, the attitude controller Simulink model found in the PX4 Simulink library can be modified to implement the architecture of the PID controller designed previously in Chapter V (see Figure 34). Alternatively, [12] had developed a wrapper code for the Simulink attitude controller found in the PX4 Simulink library (see Figure 35) that can be modified readily to accept input signals from the sensors found on the Pixahwk autopilot and outputs the required PWM for the Iris+ BLDC motors.

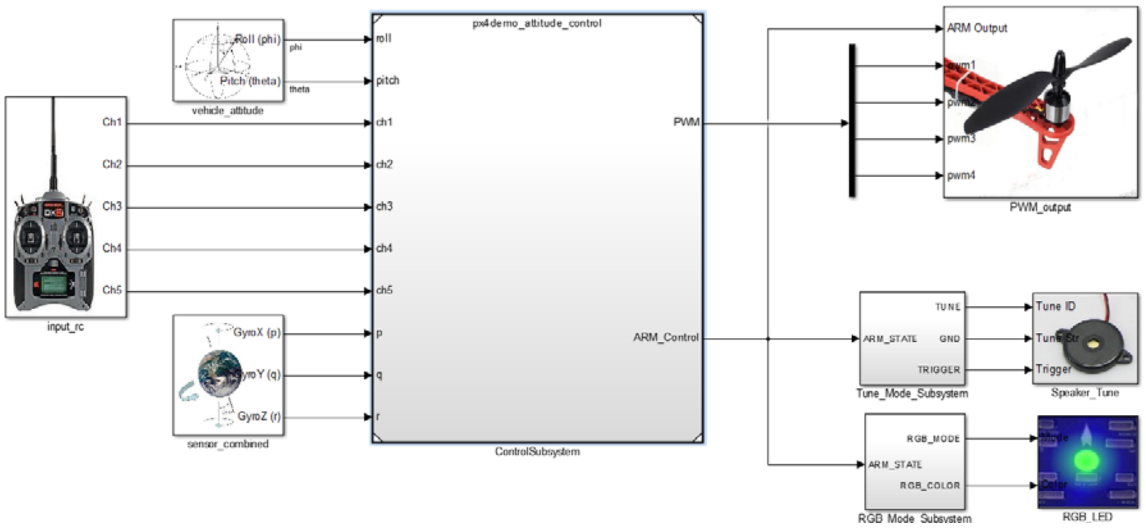


Figure 34 Attitude Control Simulink Model in PX4 Simulink. Example from [13].

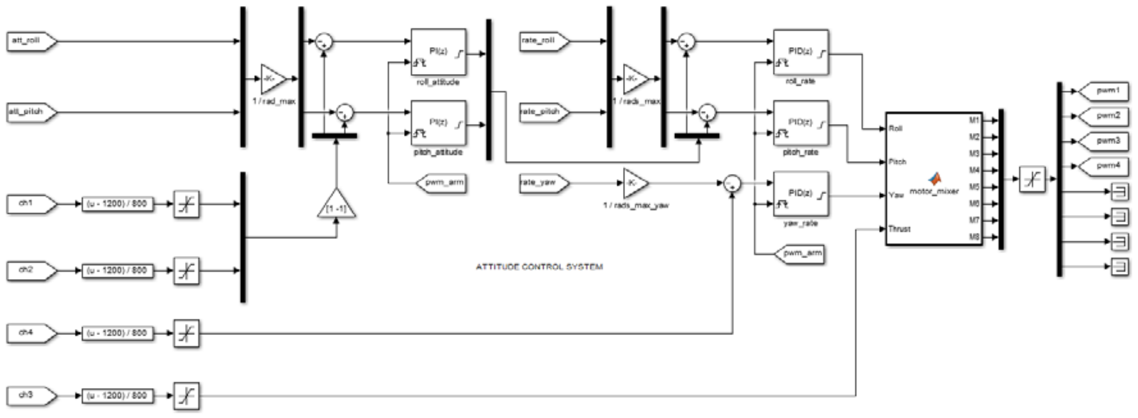


Figure 35 Attitude Control Wrapper (from [12]).

Using the PID gain values found in Chapter V with the Ziegler-Nichols method, the attitude controller Simulink model can be configured to use these PID gain values. In addition, the flight dynamics constraints (e.g., saturation limits of the control laws, the BLDC motors' minimum and maximum PWM values, etc.) of the Iris+ quadrotor would also need to be updated to match the architecture of the newly developed attitude controller model. Finally, the 'Build' function in Simulink can be used to "auto-generate" the updated attitude controller Simulink model into C/C++ codes.

C. BUILDING PIXHAWK APPLICATION IN PX4 ECLIPSE

The process of building an application for the Pixhawk autopilot was documented in both [13] and [12]. This thesis has combined the essential information that are required to build an application for the attitude controller from both sources and elaborated on the process in Appendix B.

Before building the application for the attitude controller, the Iris+ quadrotor's ground station software, QGroundControl would need to be downloaded from [31]. The QGroundControl software adopts the Mavlink communication protocol to support bi-directional command and control of the multi-copter in flight. In addition, it also includes utility function such as the calibration of the Iris+ quadrotor's sensor and interfaces with the Iris+ for logging of the flight data.

The method described in [13] allows the application to be built in Simulink directly and downloaded to the Pixhawk autopilot. In contrast, the method used in [12] uses the PX4 Eclipse software in the PX4 toolchain to build the application from the source code resulted from MathWork's auto-coding and downloading it to the Pixhawk autopilot. The process of using the PX4 Eclipse to build an application for the Pixhawk autopilot was summarized with the flow diagram in Figure 36.

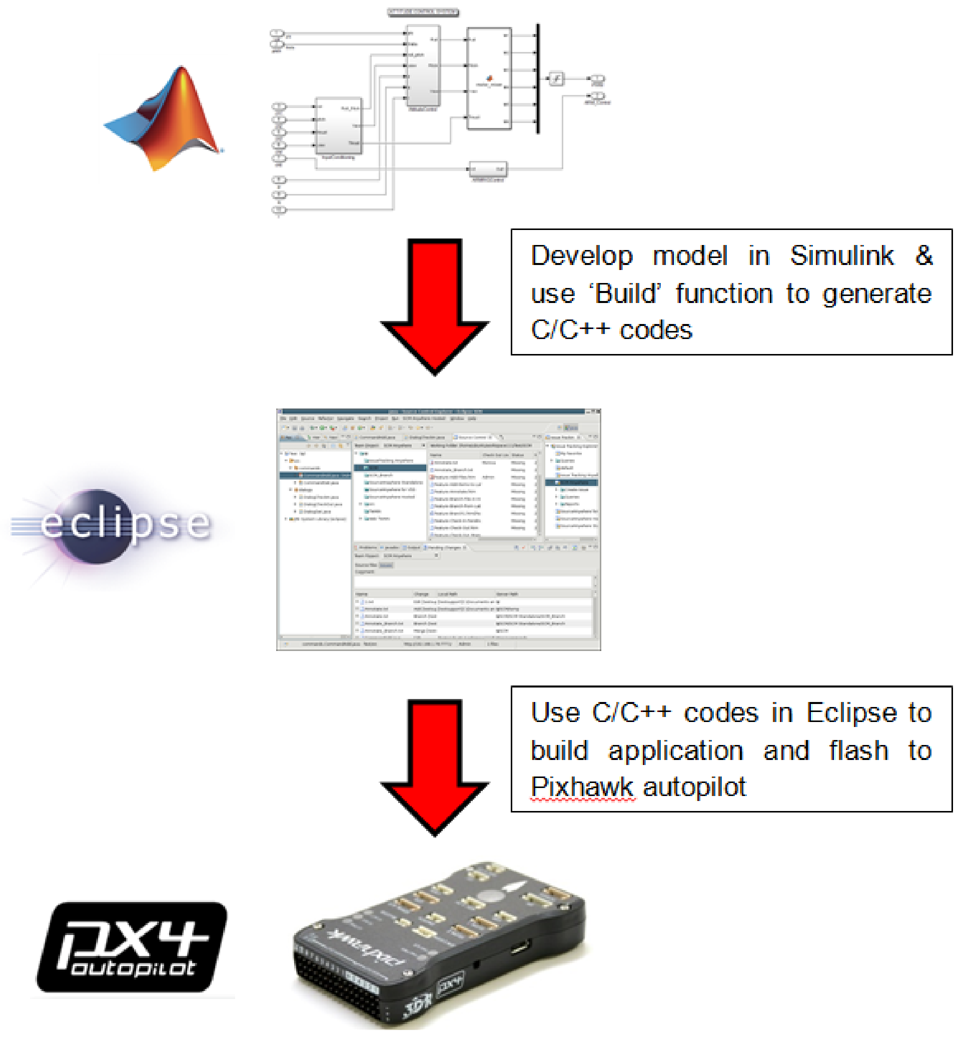

Develop model in Simulink & use 'Build' function to generate C/C++ codes
Use C/C++ codes in Eclipse to build application and flash to Pixhawk autopilot

Figure 36 Process of Building and Downloading Application to Pixhawk Autopilot (after [16] and [19]).

VII. IRIS+ FLIGHT TESTS

After the Simulink attitude controller application was successfully downloaded to the Pixhawk autopilot, a flight test was set up to assess the performance of the Iris+. The flight test was carried out in NPS's Halligan hall indoor flight facilities (see Figure 37), which had been setup to fly multi-copter in a safe environment.

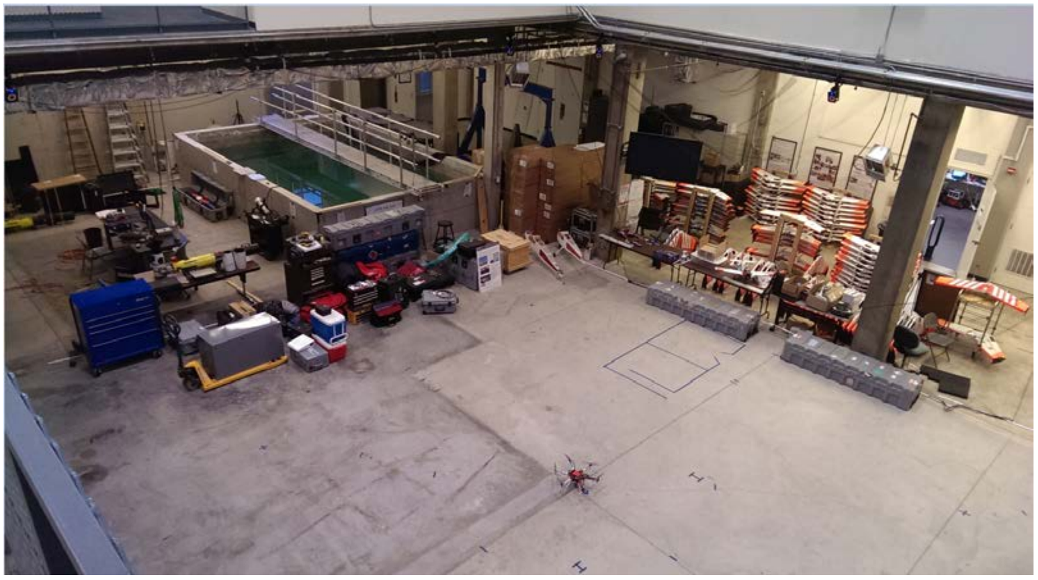


Figure 37 Multi-Copter Indoor Flight Facilities.

A. FLIGHT TEST PROCEDURE

The main objective of the flight test is to analyze the transient and steady-state performance of the PID attitude controllers in the roll, pitch and yaw channels. The steps for the flight test are further elaborated below:

- Step 1

Set up the ground computer to record the flight test data by launching QGroundControl and connecting it to the Iris+ quadrotor with the 3DR radio (set baud rate to 57,600). In QGroundControl, select the 'Analyze' tab on the toolbar and select all the flight data parameters (see Figure 38). Finally, click 'start logging' in the 'Analyze' window to start the flight data recording and save the data as a CSV file in the working directory.

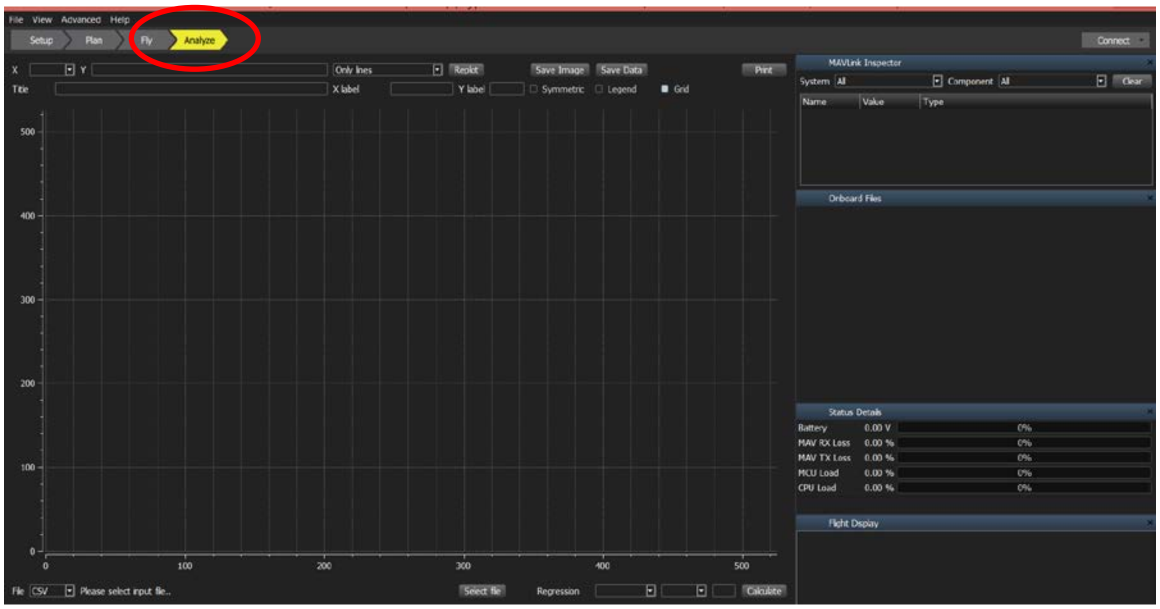


Figure 38   Location of 'Analyze' Tab on QGroundControl.

- Step 2

Once the ground computer is setup to record the flight data, the motors on the Iris+ quadrotor are armed and the copter flies a rectangular trajectory along the circumference of the flight facilities, using manual control from the RC transmitter. During the flight of the Iris+ quadrotor, roll, pitch and yaw commands are transmitted from the RC transmitter to obtain the attitude controller's response.

- Step 3

When the Iris+ quadrotor has landed, disarm the motor with the RC transmitter and select 'stop logging' from the QGroundControl. Develop MATLAB scripts to facilitate quick data analysis. Using the flight data recorded in the CSV file, plot the transient and steady-state curves for the Iris+ attitude and rates channel.

B. FLIGHT TEST RESULTS AND ANALYSIS

A series of flight tests were conducted to determine the transient and steady state performance characteristics of the attitude controller onboard the Iris+. The resulting plot of the flight data is shown in Figure 39. The red line represents the roll magnitude command transmitted by the RC transmitter and the blue line represents the magnitude of roll motion measured by the Iris+ sensor. In the first flight test, it can be seen from the plot for the roll channel exhibits relatively high overshoots throughout the Iris+ flight.

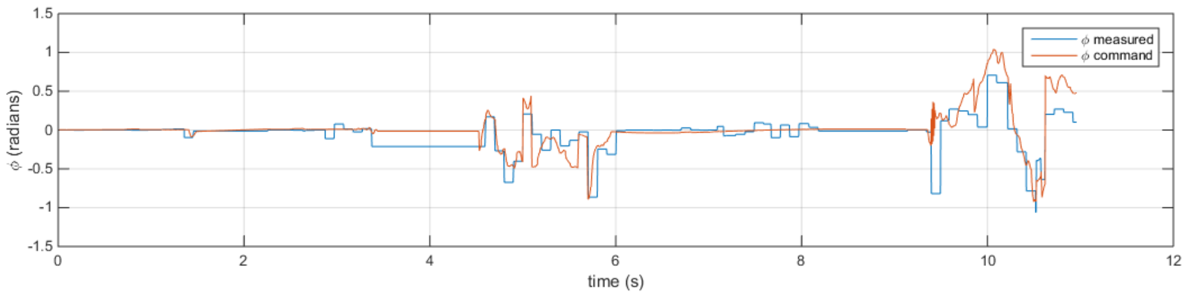


Figure 39 Roll Channel Plot for First Flight.

To improve performance characteristics of the transient and steady state response of the attitude controller, the PID gains in the roll, pitch and yaw channels were tuned. The outcome from the tuning is elaborated in the subsequent sections:

1. Roll and Roll Rate Channels

The plots of the flight data for the roll and roll rate for the duration of 24.6 to 26.6s are shown in Figure 40 and Figure 41 respectively; the red line represents the roll magnitude command transmitted by the RC transmitter and the blue line represents the magnitude of roll measured by the Iris+ sensor. The subplot represents the error between the command and measured roll and roll rate magnitudes.

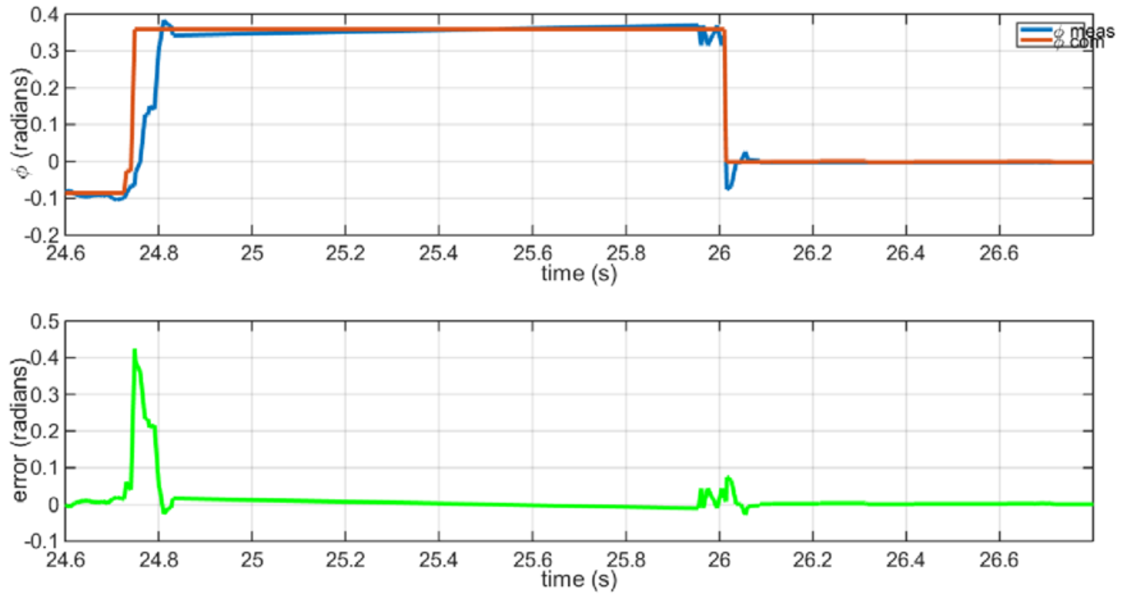


Figure 40 Roll and Error Plots.

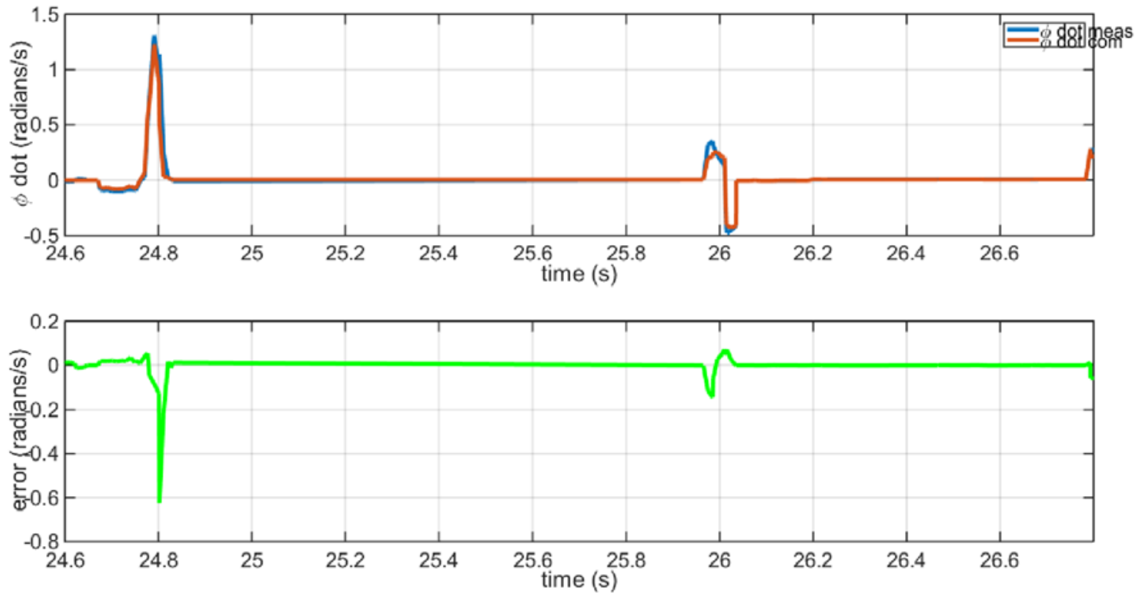


Figure 41 Roll Rate and Error Plots

It can be observed that the outer loop roll channel controller is able to track the roll command with a slight overshoot (i.e., approximately 15%) in the transient phase and exhibits little oscillations in the steady state. The transient and steady state performance of the inner loop roll rate channel is similar. Therefore, the roll and roll rate channels demonstrate reasonable transient and steady state performances to control the Iris+ in flight.

2. Pitch and Pitch Rate Channels

The plots of the flight data for the pitch and pitch rate for the duration of 5.62 to 5.72s are shown in Figure 42 and Figure 43 respectively; the red line represents the pitch magnitude command transmitted by the RC transmitter and the blue line represents the magnitude of pitch motion measured by the Iris+ sensor. The subplot represents the error between the command and measured pitch and pitch rate magnitudes.

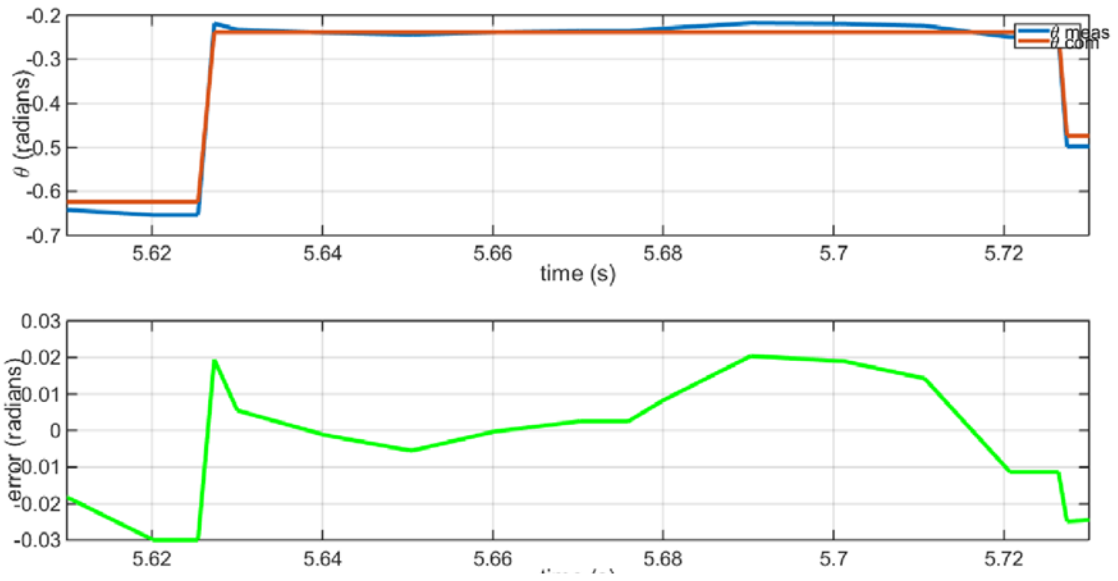


Figure 42 Pitch and Error Plots.

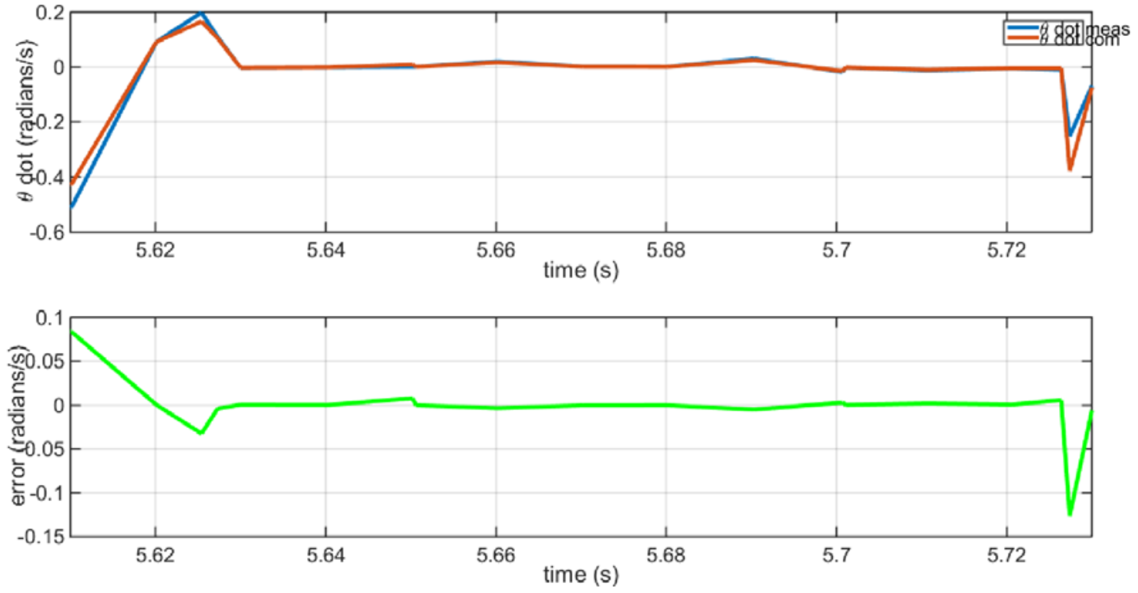


Figure 43 Pitch Rate and Error Plots.

It can be observed that the outer loop pitch channel controller is able to track the roll command with a slight overshoot (i.e., approximately 10%) in the transient phase and exhibits slight oscillations in the steady state. The transient and steady state performance of the inner loop pitch rate channel is similar. Therefore, the pitch and pitch rate channels demonstrate reasonable transient and steady state performances to control the Iris+ in flight.

3. Yaw and Yaw Rate Channels

The plots of the flight data for the yaw and yaw rate for the duration of 21 to 22s are shown in Figure 44 and Figure 45 respectively; the red line represents the yaw magnitude command transmitted by the RC transmitter and the blue line represents the magnitude of pitch motion measured by the Iris+ sensor. The subplot represents the error between the command and measured yaw and yaw rate magnitudes.

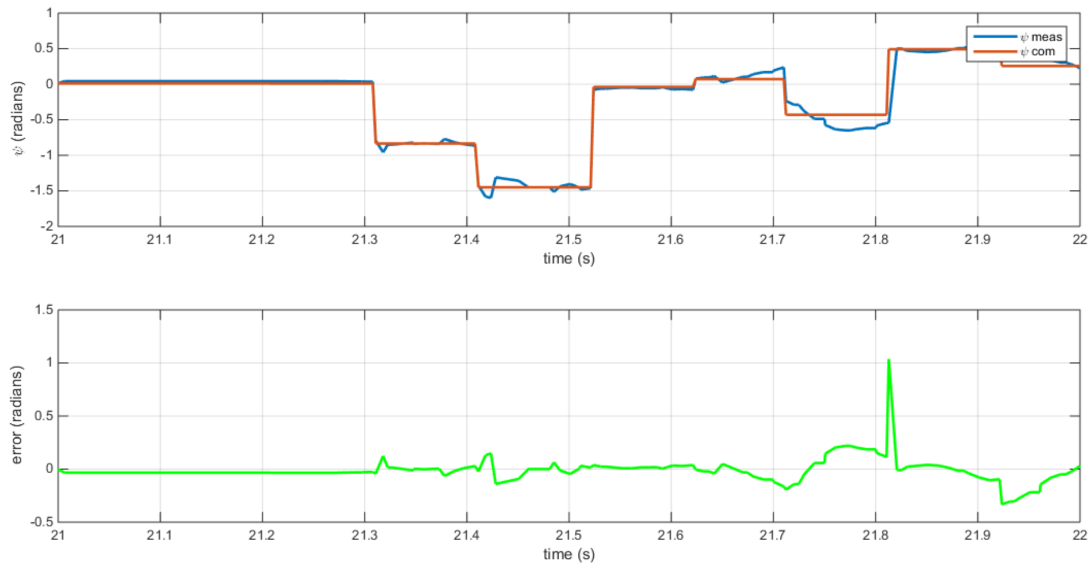


Figure 44 Yaw and Error Plots.

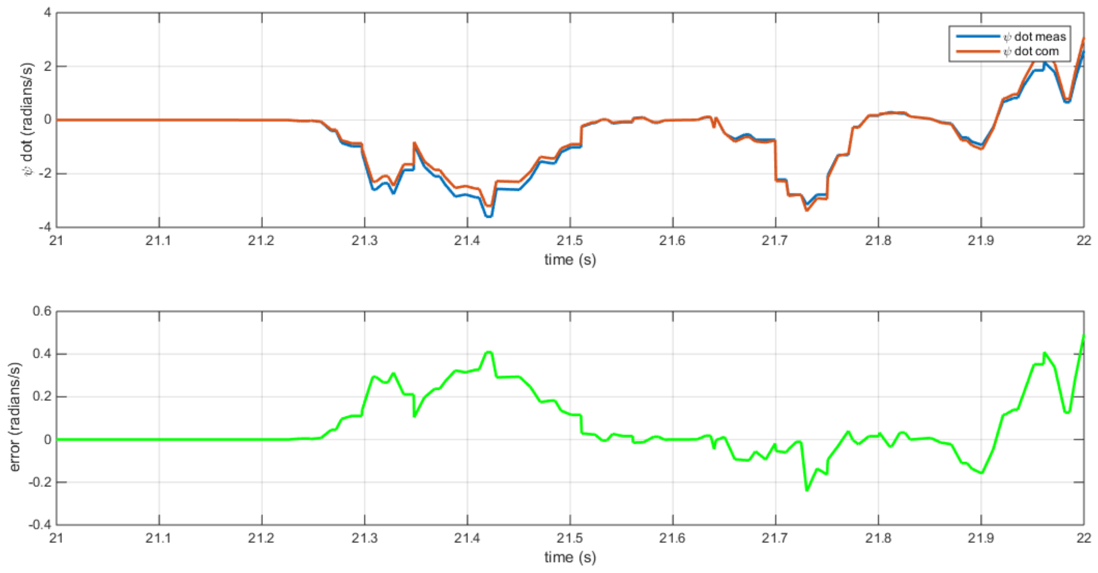


Figure 45 Yaw Rate and Error Plots

It can be observed that the outer loop yaw channel controller is able to track the yaw command with a slight overshoot (i.e., approximately 20%) in the transient phase and exhibits slight oscillations in the steady state. The transient and steady state performance of the inner loop yaw rate channel is similar.

Therefore, the yaw and yaw rate channels demonstrate reasonable transient and steady state performances to control the Iris+ in flight.

VIII. CONCLUSION AND FUTURE WORKS

In this thesis, it was shown that the Iris+ quadrotor flight dynamics can be represented with a mathematical model in Simulink and be used to design an attitude controller to stabilize the Iris+ quadrotor in flight. This thesis also demonstrated the successful implementation of a flight attitude controller developed using Simulink onboard the Pixhawk autopilot by using the Simulink ‘Build’ function and the PX4 Eclipse software, while eliminating the need to be proficient in high level programming language.

Therefore, the methods outlined in this thesis provide control engineering students with an enabling tool to implement their flight controller design that was developed in Simulink directly onto a quadrotor autopilot hardware. The flight controller design can subsequently be put through an actual flight test and the performance of the flight controller can be readily analyzed with the ground control station.

The groundwork was laid by this thesis to implement a flight controller design on actual quadrotor hardware. As a continuation for this thesis, the following areas can be considered by students for future research works:

- Instead of the PID controller used in this thesis, alternatives to the controller design (e.g. LQR) can be implemented on the Iris+ quadrotor to improve its stability and robustness during the flight.
- The GPS signal detected by the Iris+ quadrotor within the indoor flight facility was not always in good condition. Therefore, the Iris+ quadrotor can be integrated to the Vicon motion capture system, which is set up in the indoor flight facility to provide multi-copters with accurate position and attitude data.
- The Pixhawk autopilot was used for a COTS quadrotor in this thesis. Alternate platforms could be considered and modeled in Simulink for use on the Pixhawk autopilots (e.g., fixed wing aircraft, UGV, etc.).

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A. DERIVATION OF CONTROL EQUATIONS

The reference in [7] describes that the flight maneuvers of a quadrotor are performed by keeping constant or varying the rotational speeds of the propellers. For example, to fly forward in the ‘x’ direction, the quadrotor would need to perform a pitch down motion to generate a horizontal thrust force in the ‘x’ direction. The quadrotor’s flight control in the ‘x’, ‘y’ and ‘z’ directions are further elaborated below:

A. MANEUVER ALONG X-AXIS

In order for the quadrotor to fly forward along the x-axis in the vehicle coordinate frame, it would first need to pitch down to generate a force (i.e. $\vec{F}$) that is represented by equation (65). The force can be resolved into the x-component, F_x which is the force that accelerates the quadrotor forward, and the z-component, F_z , which is the upward force that is required to maintain the quadrotor in the desired altitude.

$$\vec{F} = \begin{bmatrix} c\theta & 0 & s\theta \\ 0 & 1 & 0 \\ -s\theta & 0 & c\theta \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ T \end{bmatrix} = \begin{bmatrix} T \cdot s\theta \\ 0 \\ T \cdot c\theta \end{bmatrix} \tag{64}$$

$$\therefore \begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix} = \begin{bmatrix} T \cdot s\theta \\ 0 \\ T \cdot c\theta \end{bmatrix} \tag{65}$$

It is assumed that the pitch angle required to generate the thrust for a forward flight along the x-axis is sufficiently small. Therefore, the x-component of the thrust force can be approximated as:

$$F_x = T \cdot s\theta \approx T \cdot \theta \tag{66}$$

The forward velocity of the quadrotor in the vehicle coordinate frame can be controlled with a proportional control law that is shown in equation (67):

$$F_x = m \cdot K_{f,x} (\dot{x}_d - \dot{x})_V \tag{67}$$

where: m is the quadrotor mass,

$\dot{x}_d$ is the desired velocity along the x-axis in the vehicle reference frame,

$\dot{x}$ is the measured velocity along the x-axis in the vehicle reference frame,

$K_{f,x}$ is the proportional gain.

Combining equations (68) and (69) derived for F_x gives the following equation:

$$T \cdot \theta = m \cdot K_{f,x} \cdot (\dot{x}_d - \dot{x})_V \tag{68}$$

$$\therefore \theta_d = \frac{m \cdot K_{f,x}}{T} (\dot{x}_d - \dot{x})_V \tag{69}$$

where: θ_d is the desired pitch angle to perform the forward flight along the x-axis.

In order for the quadrotor to maintain its altitude in forward flight, upward thrust generated by the propellers would need to be equal to its weight. Therefore, the thrust component, T in equation (69) can be replaced with the quadrotor's weight:

$$\theta_d = \frac{m \cdot K_{f,x}}{T} \left(\dot{x}_d - \dot{x}\right)_V = \frac{m \cdot K_{f,x}}{m \cdot g} \left(\dot{x}_d - \dot{x}\right)_V \tag{70}$$

$$\therefore \theta_d = \frac{K_{f,x}}{g}(\dot{x}_d - \dot{x})_V \qquad (71)$$

Finally, the desired forward velocity of the quadrotor in the inertial coordinate frame, $\dot{x}_d$ is related to the position along the x-axis with the following proportional control law:

$$\dot{x}_d = K_{P,x}(x_d - x) \qquad (72)$$

where: x_d is the desired position of the quadrotor along the x-axis,

x is the measured position of the quadrotor along the x-axis,

$K_{P,x}$ is the proportional gain.

Therefore, the desired pitch angle that is required to perform flight forward along the x-axis is represented by equation (73):

$$\theta_d = K_{1,x} \cdot \left[R_I^V \cdot \left(x_d - x \right) - K_{2,x} \cdot \dot{x}_V \right] \qquad (73)$$

where: R_I^V is the rotation matrix that transforms the position from the inertia to vehicle coordinate frame,

$K_{1,x}$ is the proportional gain term for the position,

$K_{2,x}$ is the proportional gain term for the velocity.

B. MANEUVER ALONG Y-AXIS

Similar to the derivation of the desired pitch angle for the quadrotor to perform a forward flight along the x-axis, the quadrotor would need to roll to a desired angle to generate a lateral force, F_y in order to move the quadrotor side way along the y-axis. The y-component of the force can be found using the rotation matrix for the roll maneuver and is shown in equations (74) and (75):

$$\vec{F} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c\phi & s\phi \\ 0 & -s\phi & c\phi \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ T \end{bmatrix} = \begin{bmatrix} 0 \\ T \cdot s\phi \\ T \cdot c\phi \end{bmatrix} \tag{74}$$

$$\begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix} = \begin{bmatrix} 0 \\ T \cdot s\phi \\ T \cdot c\phi \end{bmatrix} \tag{75}$$

It is assumed that the roll angle to perform the lateral flight along the y-axis is sufficiently small. Therefore, the y-component force can be approximated as:

$$F_y = T \cdot s\phi = T \cdot \phi \tag{76}$$

The lateral velocity of the quadrotor in the vehicle coordinate frame can be controlled with a proportional control law as shown in equation (77):

$$F_y = m \cdot K_{f,y} \left(\dot{y}_d - \dot{y} \right)_V \tag{77}$$

where: $\dot{y}_d$ is the desired velocity along the y-axis in the vehicle reference frame,

$\dot{y}$ is the measured velocity along the y-axis in the vehicle reference frame,

and $K_{f,y}$ is the proportional gain.

Combining the equations (76) and (77) derived gives equation (78):

$$T \cdot \phi = m \cdot K_{f,y} (\dot{y}_d - \dot{y})_V \tag{78}$$

In order for the quadrotor to hover and maintain its altitude, its weight would need to be equal to the upward thrust generated. Therefore, the above equation becomes:

$$\phi = \frac{m \cdot K_{f,y}}{T}\left(\dot{y}_d - \dot{y}\right)_V = \frac{m \cdot K_{f,y}}{m \cdot g}\left(\dot{y}_d - \dot{y}\right)_V \tag{79}$$

$$\therefore \phi = \frac{K_{f,y}}{g}\left(\dot{y}_d - \dot{y}\right)_V \tag{80}$$

Finally, the desired lateral velocity of the quadrotor in the inertial coordinate frame, $\dot{y}_d$ is related to the position along the y-axis with the following proportional control law:

$$\dot{y}_d = K_{P,y}(y_d - y) \tag{81}$$

where: y_d is the desired position of the quadrotor along the y-axis,

y is the measured position of the quadrotor along the y-axis,

$K_{P,y}$ is the proportional gain.

Therefore, the desired roll angle that is required to effect a lateral motion is represented by equation (82):

$$\phi_d = K_{1,y}\left[R_I^V\left(y_d - y\right) - K_{2,y} \cdot \dot{y}_V\right] \tag{82}$$

where: $K_{1,y}$ is the proportional gain term for the position along the y-axis,

$K_{1,y}$ is the proportional gain term for the velocity along the y-axis.

C. MANEUVER ALONG Z-AXIS

In order for the quadrotor to maneuver along the z-axis, the quadrotor's thrust, T would need to be adjusted according to equation (83). The additive term

ω_0 is the rotor speed that is required to generate the amount of thrust that equals the weight of the Iris+ quadrotor. The additive term results in a feedforward controller for the thrust channel, which eliminates the need for high proportional gain values (leads to instability) or an integral gain term (leads to slow response time). Therefore, the quadrotor's altitude control can be implemented with a proportional, derivative controller as a result of the feedforward term:

$$T = K_{P,z}(z_d - z) + K_{D,z}(\dot{z}_d - \dot{z}) + \omega_0 \tag{83}$$

where: $K_{P,z}$ is the proportional gain term for the position along the z-axis,

$K_{D,z}$ is the derivative gain term for the velocity along the z-axis,

ω_0 is the motor rotational speed required to generate a thrust that is equal to the weight of the quadrotor.

APPENDIX B. INSTRUCTIONS FOR BUILDING PIXHAWK APPLICATION

There are two methods designed for building an application for Simulink models and implementing it on the Pixhawk autopilot, as documented in [13] and [12]. This chapter combines the essential information from both documents and summarizes the details for building an application/firmware for the Pixhawk autopilot. The instructions for building the Pixhawk application are further elaborated in the steps below:

A. SOFTWARE INSTALLATION

Before beginning with the process of building an application/firmware, the Pixhawk toolchain and QGroundControl (Version 2.0.3 beta[1]) software would need to be installed. The step-by-step installation process is summarized below:

- Step 1

Ensure that the MATLAB and Simulink installed on your computer meet the requirements in Table 11.

Table 11 Matlab and Simulink Prerequisites (from [13]).

MATLAB & Simulink Requirements
- Version 2014a or later - Contains: i) Simulink Coder ii) Embedded Coder iii) Simulink Aerospace Block Set

[1] It is important to download QGroundControl version 2.0.3 beta, as the latest version of the software does not allow the user to perform sensor calibration after uploading the application onto the Pixhawk autopilot. If the latest version of QGroundControl was used instead, an update of the Pixhawk autopilot with the latest firmware is required to perform the sensor calibration. Thus, the application that was build and uploaded to the Pixhawk would be erased during the firmware update.

- Step 2

Go to the website in [30] to install the PX4 toolchain. Choose the appropriate toolchain installer based on the Operating System (OS) used on your computer (e.g. Windows, Linux, etc.) Download and install the PX4 toolchain for the appropriate MATLAB version (e.g. 'Pixhawk-R2014b-v14_win.exe') in the default root directory (i.e. 'c:\px4').

- Step 3

Download the USB driver for the Pixhawk toolchain if you are using Windows OS from the same website in Step 2.

- Step 4

Update the Pixhawk software by choosing 'PX4 Software Download' from Windows 'Start' menu as shown in Figure 46.

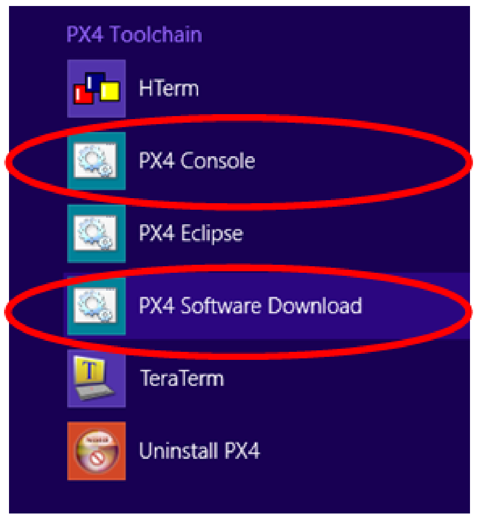


Figure 46 Location of PX4 Software Download & Upgrade.

- Step 5

Once the Pixhawk software is updated, 2 target files would need to be built using the 'PX4 Console' shown in Figure 46. In the PX4 console, change the directory to Firmware by typing 'cd Firmware' and hitting enter.

- Step 6

The next step involves the installation of the Pixhawk PSP to Matlab. Begin by launching the installation program from the installation folder (i.e., pixhawk-R2014a-v14_win-Install.exe) and specify the Matlab and Pixhawk toolchain root directory (i.e. 'c:\'). After the installation is complete, the Pixhawk Simulink blocks and MATLAB Pixhawk toolchain BTI functions are available and can be copied to any Simulink model for use.

- Step 7

Finally, to use the method of building an application outlined in [12], go to the website in this reference and download the 'px4_simulink.zip' folder. Subsequently, extract the contents of the folder to 'c:\px4\' (i.e., the installation location for the PX4 toolchain).

B. BUILD ATTITUDE CONTROLLER APPLICATION

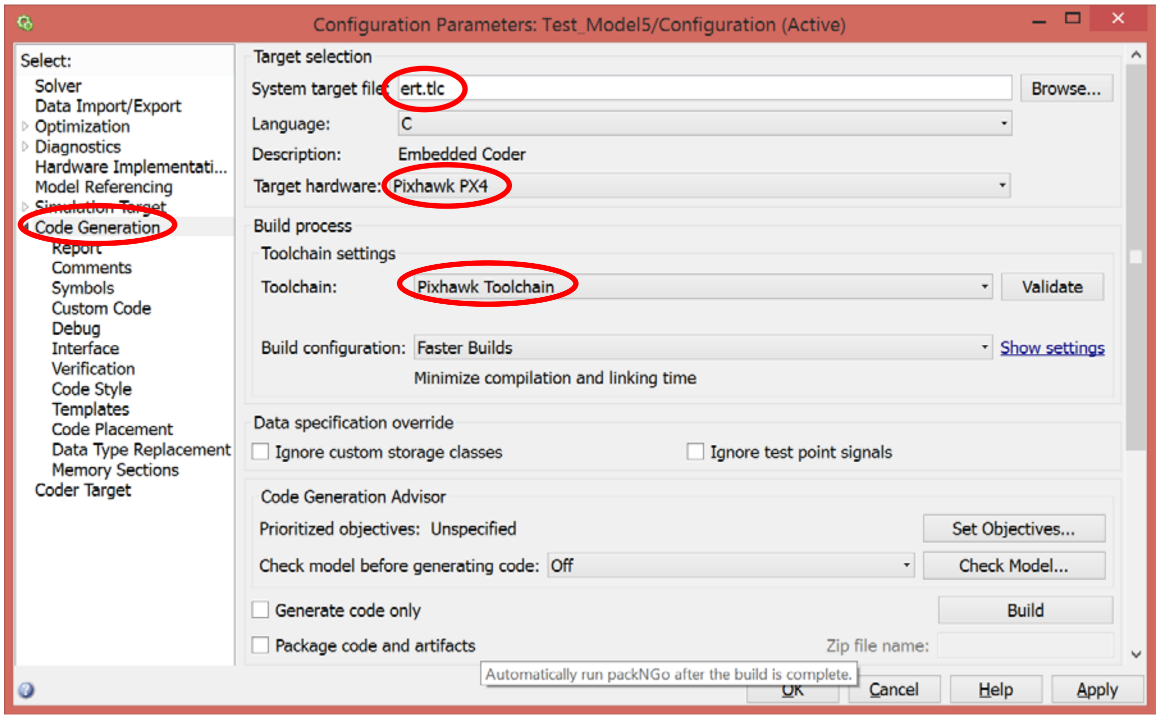
In [13], the process to build the Simulink model to an application can be performed in Simulink entirely, while the process in [12] builds the application with the PX4 Eclipse software. Both processes for building the application are further elaborated below:

1. Build Application in Simulink

The step-by-step process to build the application for the Simulink attitude controller is as follows:

- Step 1

The settings in Simulink would need to be configured before commencing with the building of the application. Select the 'Simulation' tab on the Simulink toolbar and select 'Model Configuration Parameters' (see Figure 47). Subsequently, select 'Code Generation' to set the 'System Target File' field to ert.tlc and the 'Target Hardware' field to Pixhawk PX4. The 'Toolchain' field should automatically become Pixhawk Toolchain.



Target selection

System target file: ert.tlc

Language: C

Description: Embedded Coder

Target hardware: Pixhawk PX4

Build process

Toolchain settings

Toolchain: Pixhawk Toolchain

Build configuration: Faster Builds

Minimize compilation and linking time

Data specification override

☐ Ignore custom storage classes ☐ Ignore test point signals

Code Generation Advisor

Prioritized objectives: Unspecified

Check model before generating code: Off

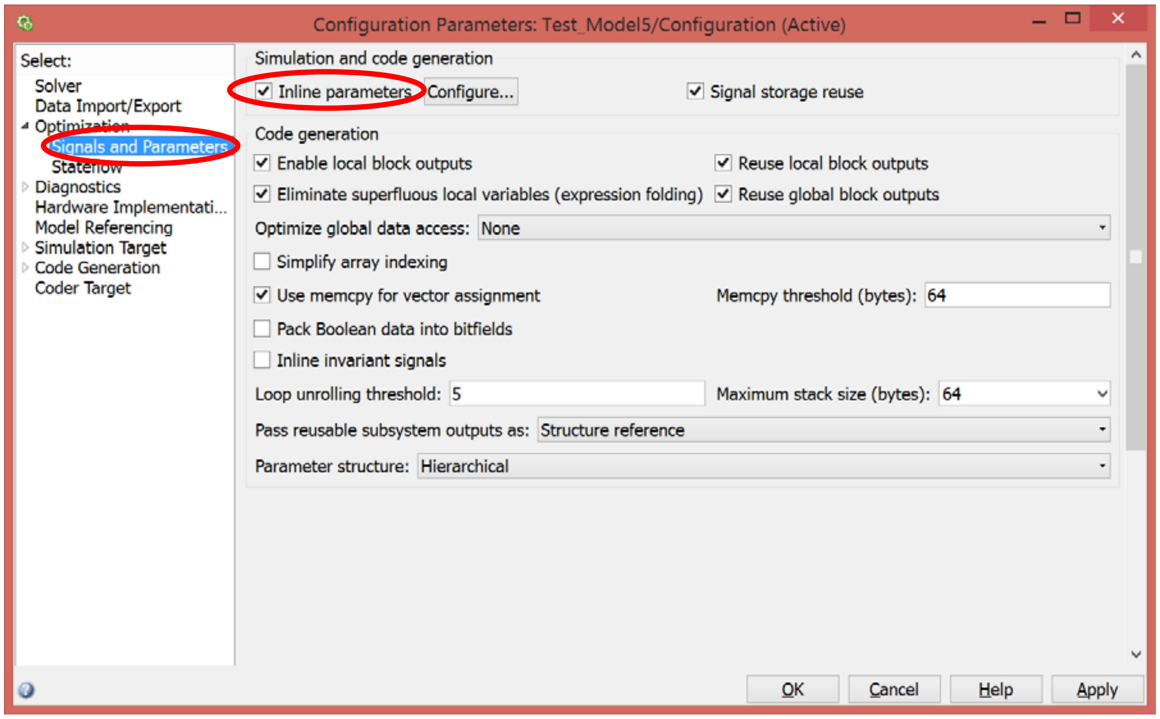
☐ Generate code only

☐ Package code and artifacts Zip file name:

Automatically run packNGo after the build is complete.

Figure 47 Screen Capture of ‘Model Configuration Parameters’.

After setting up the ‘Code Generation’, select the ‘Signals and Parameters’ and check the ‘Inline Parameters’ box as shown in Figure 48. Finally, select the ‘Coder Target’ and go to ‘Build Options’ to select the ‘Build, Load and Run’ option as shown in Figure 49.

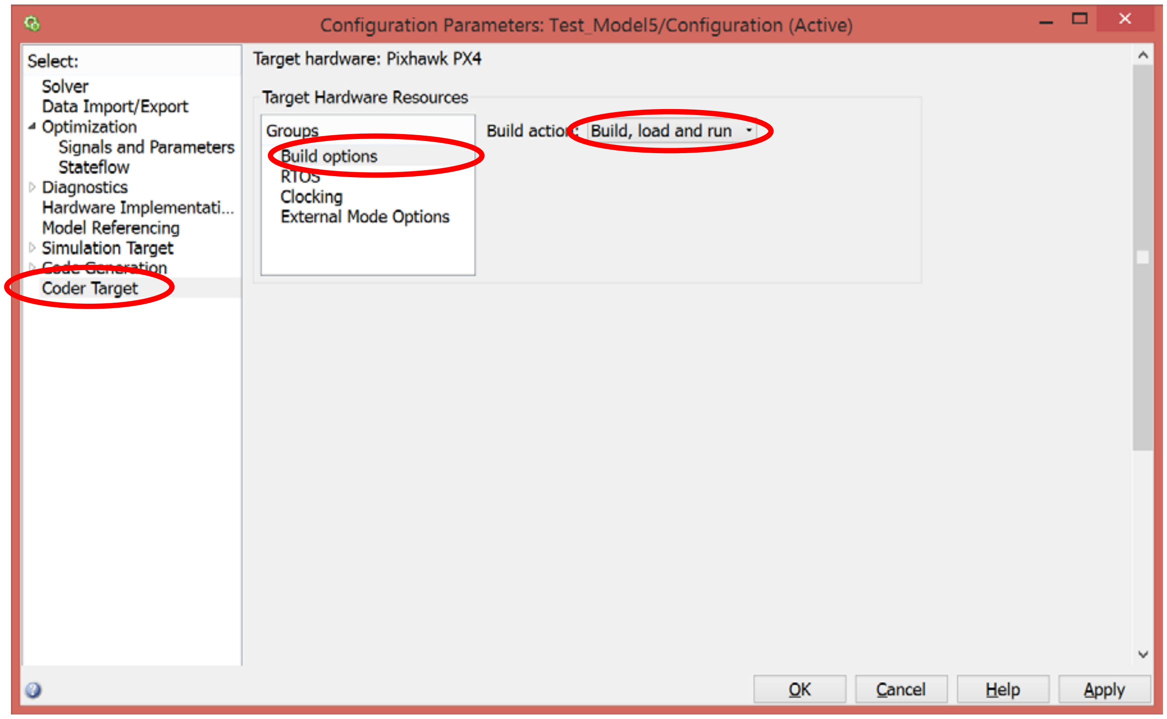

Configuration Parameters: Test_Model5/Configuration (Active)

Select:
- Solver
- Data Import/Export
- Optimization
  - Signals and Parameters
  - Stateflow
- Diagnostics
- Hardware Implementati...
- Model Referencing
- Simulation Target
- Code Generation
- Coder Target

Simulation and code generation
- [✓] Inline parameters Configure...
- [✓] Signal storage reuse

Code generation
- [✓] Enable local block outputs
- [✓] Reuse local block outputs
- [✓] Eliminate superfluous local variables (expression folding)
- [✓] Reuse global block outputs
- Optimize global data access: None
- [ ] Simplify array indexing
- [✓] Use memcpy for vector assignment
- Memcpy threshold (bytes): 64
- [ ] Pack Boolean data into bitfields
- [ ] Inline invariant signals
- Loop unrolling threshold: 5
- Maximum stack size (bytes): 64
- Pass reusable subsystem outputs as: Structure reference
- Parameter structure: Hierarchical

Figure 48 Screen Capture of ‘Signals and Parameters’.


Configuration Parameters: Test_Model5/Configuration (Active)

Select:
- Solver
- Data Import/Export
- Optimization
  - Signals and Parameters
  - Stateflow
- Diagnostics
- Hardware Implementati...
- Model Referencing
- Simulation Target
- Code Generation
- Coder Target

Target hardware: Pixhawk PX4

Target Hardware Resources

Groups
- Build options
- RTOS
- Clocking
- External Mode Options

Build action: Build, load and run

Figure 49 Screen Capture of ‘Coder Target’.

- Step 2

Once the build setting in Simulink is configured, click the ‘Build’ icon on the Simulink toolbar as shown in Figure 50. The build status is reflected on a pop-up window: diagnostic viewer. Once the build process is completed, connect the USB cable from the computer to the Pixhawk autopilot. Finally, click the ‘Code’ icon on the Simulink toolbar and select ‘PX4 PSP: Upload code to PX4FMU’ as shown in. This step will complete the application build process and download the application directly to the Pixhawk autopilot.

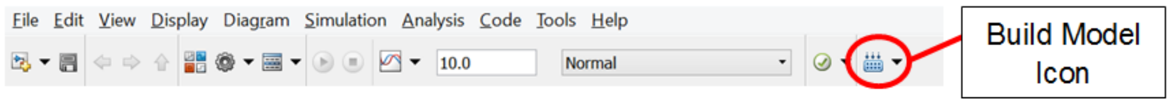


Figure 50 Location of ‘Build’ Icon on Simulink Toolbar.

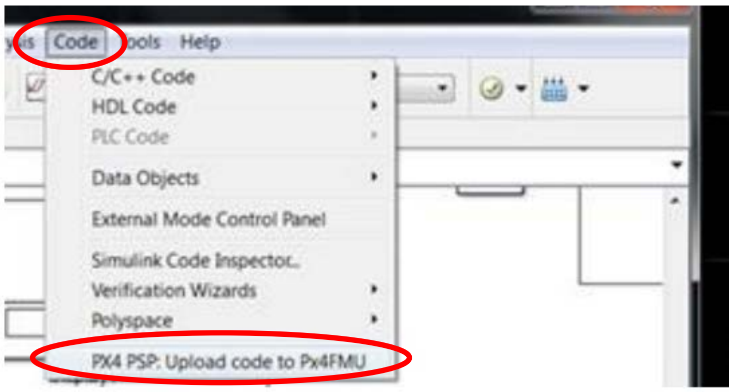

Code
Tools
Help
C/C++ Code
HDL Code
PLC Code
Data Objects
External Mode Control Panel
Simulink Code Inspector...
Verification Wizards
Polyspace
PX4 PSP: Upload code to Px4FMU

Figure 51 Location of ‘Code’ Function on Simulink Toolbar.

- Step 3

The final step of the build process involves copying the ‘rc.txt’ start-up script from the PX4 directory: ‘c:\px4\Firmware\etc’ to the Pixhawk autopilot’s SD card. Rename the ‘rc.txt’ file to ‘rc.txt.simulink’ in the SD card’s root directory.

Upon the Pixhawk autopilot's system startup, it will execute the px4_simulink_app (i.e., the application that was built in step 2).

2. Build Application using PX4 Eclipse

The alternate method to build the Simulink attitude controller application uses the PX4 Eclipse software. Similar to the first method, the Simulink settings for the build function would need to be configured first following steps 1 and 2 outlined in the first method. The subsequent steps to build the applications are as follow:

- Step 1

Launch the PX4 Eclipse software and begin to setup by selecting 'File' from the toolbar and 'New'. Subsequently, select 'Makefile Project with Existing Code' as shown in Figure 52. Once the window for a new project is launched, select 'c:\px4\firmware' as the folder by using the browse function and select 'Cross GCC' as shown in Figure 53.

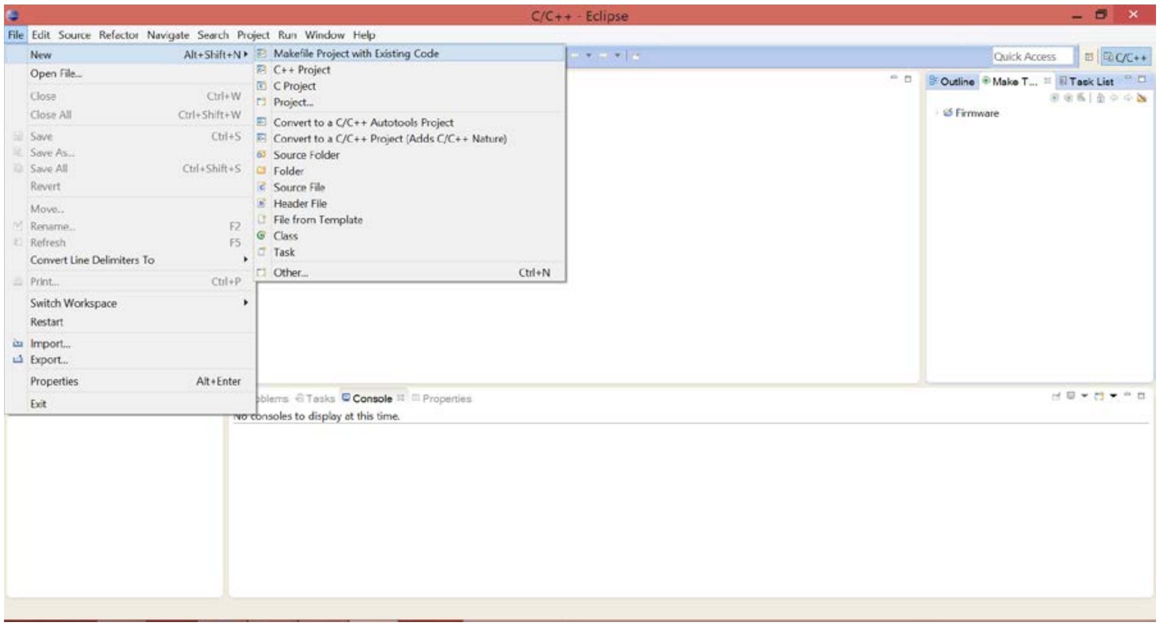
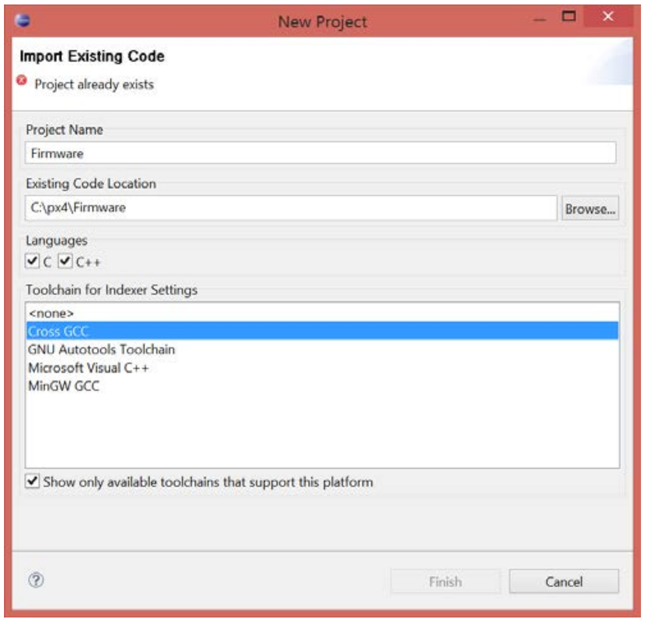


Figure 52 Setup for Application Build in Eclipse.



Import Existing Code

Project already exists

Project Name: Firmware

Existing Code Location: C:\px4\Firmware

Languages: ✓ C ✓ C++

Toolchain for Indexer Settings

<none>
Cross GCC
GNU Autotools Toolchain
Microsoft Visual C++
MinGW GCC

✓ Show only available toolchains that support this platform

Figure 53 Create New Project in Eclipse.

- Step 2

Once the setup in Step 1 is completed, create the following targets in the 'Firmware' folder by clicking on the 'Make Target' icon (see Figure 54):

- 'archives'
- 'all'
- 'distclean'
- 'clean'
- 'upload px4fmu-v1_default' (if using PX4FMU) or 'upload px4fmu-v2_default' (if using Pixhawk)

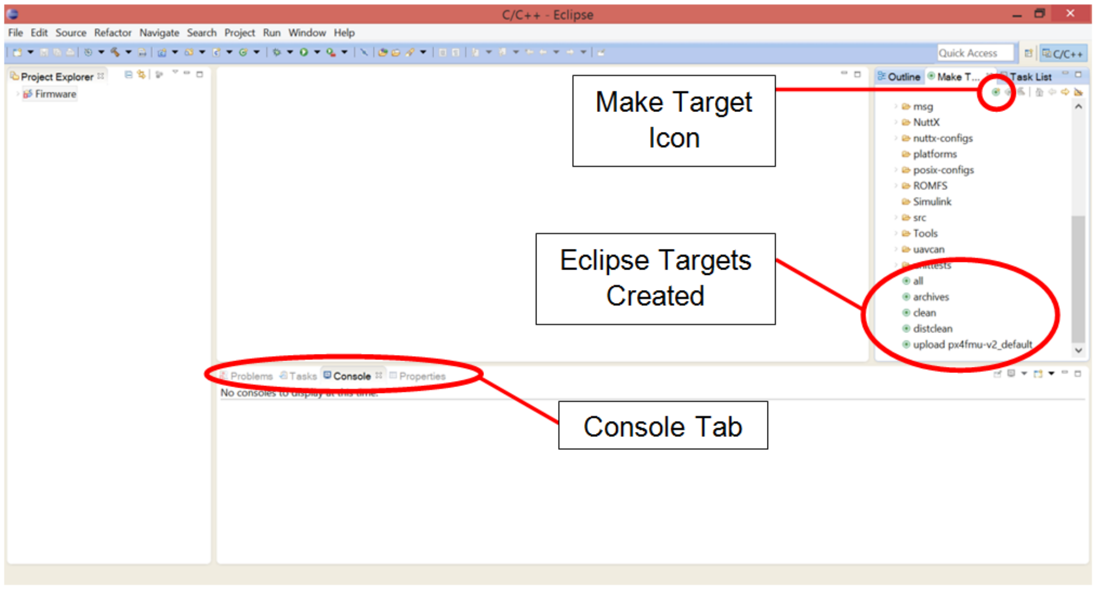


Figure 54 Build Targets for Application in Eclipse.

- Step 3

Check that the targets created in the previous step appear in the workspace on the right of Eclipse’s window as shown in Figure 54. Subsequently, build the following targets one after the other by right clicking on the target and select ‘Build Target’:

- ‘distclean’
- ‘archives’
- ‘all’

The progress of the build process can be monitored under the ‘Console’ tab at the bottom of the Eclipse Window as shown in Figure 54. It should be noted that the build for each target should be completed before proceeding to build the next target.

- Step 4

Once the build for the targets in the previous section is completed, the application is ready to be downloaded to the Pixhawk autopilot. Double click on the ‘upload px4fmu-vX_default’ target. Once the console tab displays the status message ‘Loaded firmware for upload px4fmu-vX_default, waiting for bootloader…’, connect the USB cable from your computer to the Pixhawk autopilot and allow the build target process to complete.

- Step 5

Remove the SD card from the Pixhawk autopilot and delete all the data from the SD card before returning it to the Pixhawk autopilot. Subsequently, launch the QGroundControl software and connect it to the Pixhawk pilot (using USB cable or the 3DR radio). Perform sensor calibration for the Iris+ by going to the ‘Configuration’ tab on the toolbar and selecting ‘sensor calibration’.

- Step 6

Finally, disconnect the Pixhawk autopilot from QGroundControl and remove the SD card from the Pixhawk autopilot. Subsequently, copy the ‘rc.txt’ script from the PX4 directory: ‘c:\px4\Firmware\etc’ to the Pixhawk autopilot’s SD card. Rename the ‘rc.txt’ file to ‘rc.txt.simulink’ in the SD card’s root directory. Upon the Pixhawk autopilot’s system startup, it will execute the px4_simulink_app.

C. UPDATE TO SIMULINK ATTITUDE CONTROLLER MODEL

For the application build process used by [12], the process of updating the Simulink attitude controller model is as follows:

- Step 1

Open the Simulink attitude controller model and set the Simulink workspace to ‘c:\px4\Firmware\src\modules\simulink_app\’. After the changes are applied to the Simulink attitude controller model, click on the ‘Build’ function in Simulink.

- Step 2

Open PX4 Eclipse and build the application by selecting the target ‘all’. It is no longer required to build the targets for ‘distclean’ and ‘archives’ unless the source code of the model was modified.

- Step 3

Finally, upload the application by selecting the target ‘upload px4fmu-v2_default’. Once the console tab displays the status message ‘Loaded firmware for upload px4fmu-vX_default, waiting for bootloader…’, connect the USB cable from your computer to the Pixhawk autopilot and allow the build target process to complete.

D. VERIFICATION OF SIMULINK APPLICATION ON PIXHAWK AUTOPILOT

To verify that the Simulink attitude controller application was correctly downloaded to the Pixhawk autopilot, connect the Iris+ to the TeraTerm terminal using the 3DR radio and setting the baud rate to 57,600.

In TeraTerm terminal, hit the ‘enter’ key once and type in ‘?’ into the ‘nsh’ field. This step will display all the applications that are currently installed onboard the Pixhawk autopilot. Check that the item ‘px4_simulink_app’ can be found in the list of applications.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C. TELEMETRY CONNECTION SETUP AND TROUBLESHOOTING

The Pixhawk autopilot establishes communication link to a computer running the QGroundControl software using the 3DR radio as shown in Figure 55.

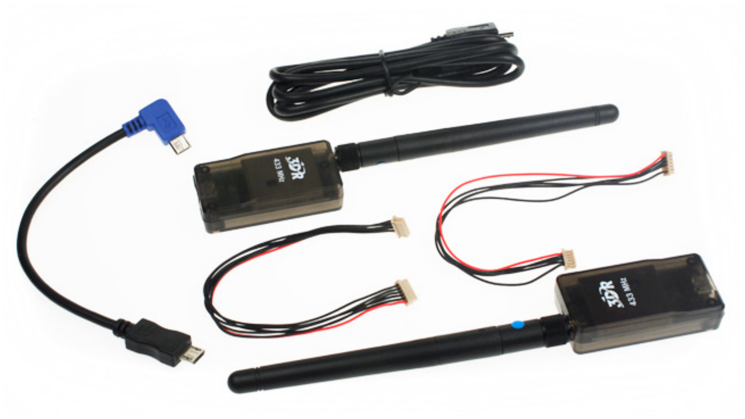


Figure 55 Illustration of 3DR Radio Set (from [32]).

A. SETTING UP 3DR RADIO

The setup of the 3DR radio is relatively straight forward and can be completed in the following steps:

- Step 1

Connect the battery to the Iris+ quadrotor. Subsequently, insert the 3DR radio to the computer and run the QGroundControl software. The QGroundControl should detect the 3DR radio once it is inserted into the computer and automatically assign it a COM port number.

- Step 2

In the QGroundControl, select the COM port detected and set the baud rate to 57,600 in the top right corner of the window as shown in Figure 56. Click on the connect button and the Iris+ will be connected to QGroundControl.

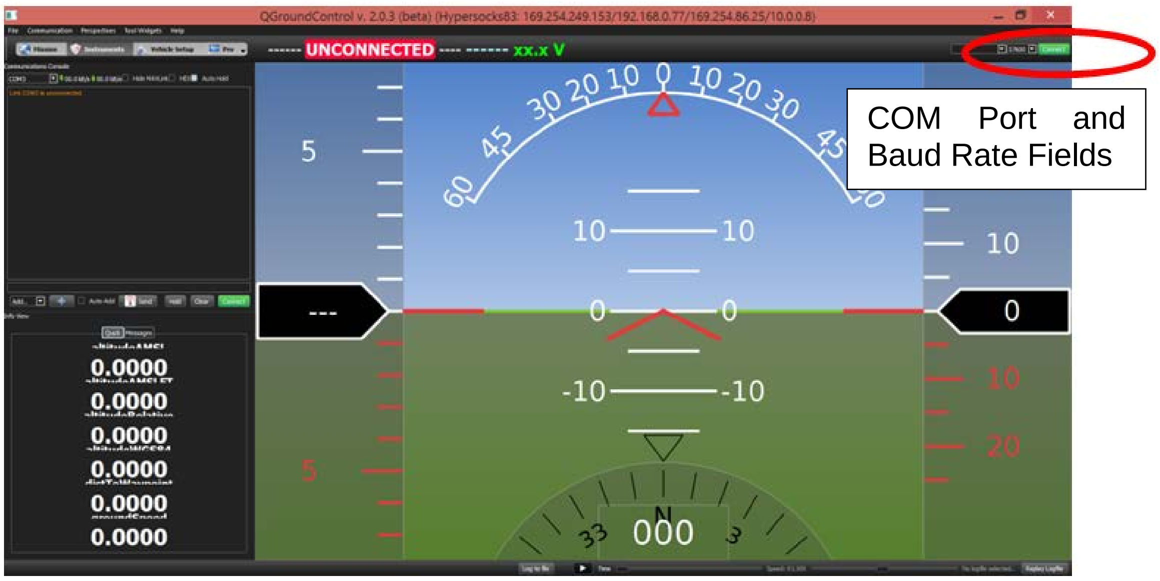


Figure 56 Setting COM Port and Baud Rate.

B. TROUBLESHOOTING 3DR RADIO

In the event that the connection between the Iris+ and 3DR radio fails, the following should be undertaken to troubleshoot the connection problem:

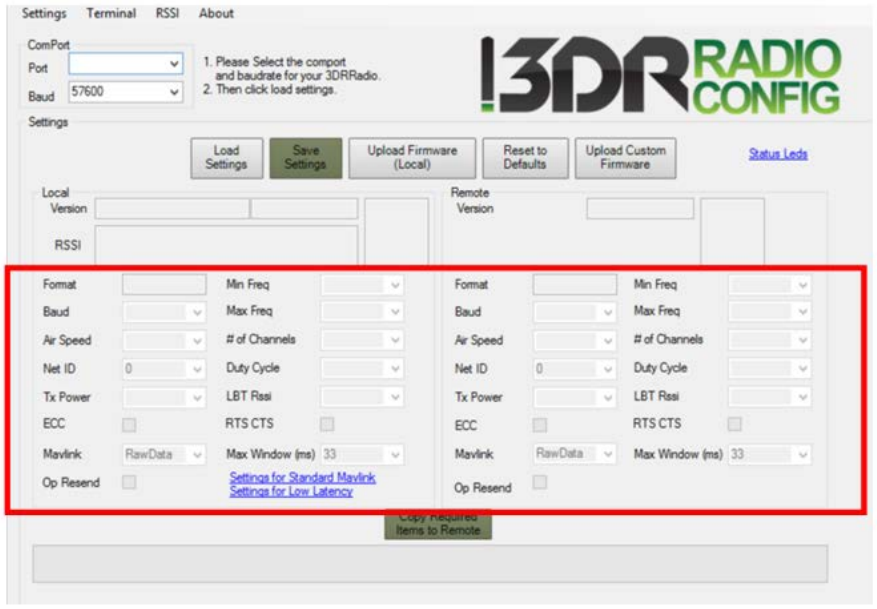
1. Check Baud Rates

The baud rate for 3DR radio connection with the Iris+ quadrotor should always be set as 57,600 and not 115,200 (which is used for direct wired serial connection between the Pixhawk autopilot and the QGroundControl).

2. Check Radio Settings

If the correct COM port and baud rate were selected on QGroundControl and connection with Iris+ still cannot be established, download the 3DR Radio Configuration Utility software [33]. Subsequently, plug the ground station's 3DR Radio to the computer and run the 3DR Radio Configuration software; make sure that the serial cable is unplugged and there is no serial stream entering the radio. Select the load settings tab and 3DR radio's parameters should populate the section below the tabs (see Figure 57). Finally, connect the computer with the

3DR radio installed onboard the Iris+ (see Figure 58) using the USB cable provided and select the load settings tab. The parameters provided should be the same as the ground station's 3DR radio. If there are parameters that are different, amend the values accordingly to the ground station's parameters and select the save settings tab.



ComPort
Port
Baud 57600

1. Please Select the comport and baudrate for your 3DRRadio.
2. Then click load settings.

Settings

Local
Version
RSSI

Remote
Version

Local				Remote			
Format		Min Freq		Format		Min Freq	
Baud		Max Freq		Baud		Max Freq	
Air Speed		# of Channels		Air Speed		# of Channels	
Net ID	0	Duty Cycle		Net ID	0	Duty Cycle	
Tx Power		LBT Rssi		Tx Power		LBT Rssi	
ECC	☐	RTS CTS	☐	ECC	☐	RTS CTS	☐
Mavlink	RawData	Max Window (ms)	33	Mavlink	RawData	Max Window (ms)	33
Op Resend	☐			Op Resend	☐		

Figure 57 Screen Capture of 3DR Radio Configuration Window.

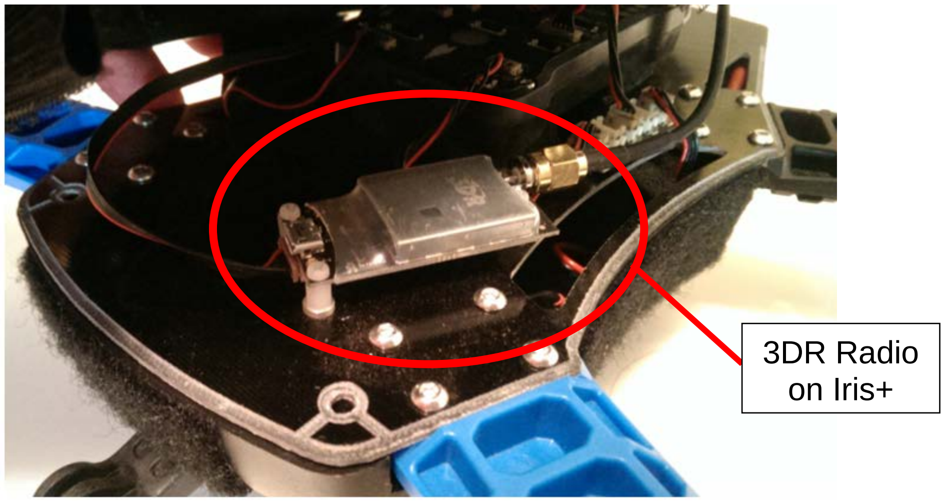


Figure 58 Installation Location of 3DR Radio on Iris+.

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

[1] Amazon Prime air. (n.d.). Amazon. [Online]. Available: http://www.amazon.com/b?node=8037720011. Accessed Mar. 5, 2015.

[2] L. Meier, D. Honegger and M. Pollefeys, “PX4: A node-based multithreaded open source robotics framework for deeply embedded platforms,” in *ICRA (Int. Conf. on Robotics and Automation)*, Seattle, WA, 2015.

[3] L. Meier. (n.d.). Software choice for Pixhawk hardware, PX4 autopilot. [Online]. Available: https://pixhawk.org/choice. Accessed Mar. 9, 2015.

[4] 3DR Product, Iris+. (n.d.). 3D Robotics. [Online]. Available: https://store.3drobotics.com/products/iris. Accessed Mar. 10, 2015.

[5] S. Bouabdallah, “Design and control of quadrotors with application to autonomous flying,” M.S. thesis, Dept. Sci and Eng., EPFL, Lausanne, Switzerland, 2007.

[6] W. R. Beard. (2008, Oct. 3). Quadrotor dynamics and control. [Online]. Available: http://rwbclasses.groups.et.byu.net/lib/exe /fetch.php?media=quadrotor:beardsquadrotornotes.pdf

[7] P. Corke, *Robotics, Vision and Control*. Berlin: Springer, 2013, pp. 79–90.

[8] A. G. Sidea, R. Y. Brogaard, N. A. Andersen and O. Ravn, “General model and control of an n rotor helicopter,” in *J. Phys.: Conf.*, Series 570, 2014. pp. 052004.

[9] T. Bresciani, “Modelling, identification and control of a quadrotor helicopter,” M.S. thesis, Dept. Automatic Control, Lund Univ., Lund, Sweden, 2008.

[10] A. R. Partovi, Z. Y. K. Ang, H. Lin and G. Cai, “Development of a cross style quadrotor,” in *AIAA Guidance, Navigation, and Control Conf.*, Minneapolis, MN, 2012, pp. 2012-4780.

[11] I. L. Mariano, V. Dobrokhodov, H. G. Elkaim, R. Curry and I. Kaminer, “Simulink based hardware-in-the-loop simulator for rapid prototyping of UAV control algorithms,” in *AIAA Infotech@Aerospace Conf.*, Seattle, WA, pp. 2009-1843.

[12] A. Polak. (2014, Jan. 21). PX4 development kit for Simulink. [Online]. Available: http://polakiumengineering.org/px4-development-kit-for-simulink/

[13] S. Kuznicki and D. Lee. (2015, Apr. 22). Pixhawk Pilot Support Package (PSP) user guide, MathWorks, [Online]. Available: https://www.mathworks.com/hardware-support/forms/pixhawk-downloads-conf.html?confirmation_page

[14] A. A. Mian and W. Daobo, “Nonlinear flight control strategy for an underactuated quadrotor aerial robot,” in *IEEE Intl. Conf.*, Sanya, China, 2008. pp. 938–942.

[15] L. Meier. (n.d.). PX4 overview, PX4 autopilot. [Online]. Available: http://www.pixhawk.org/start. Accessed Mar. 3, 2015.

[16] 3DR product, Pixhawk autopilot. (n.d.). 3D Robotics. [Online]. Available: https://store.3drobotics.com/products/3dr-pixhawk. Accessed Mar. 15, 2015.

[17] APM open source autopilot. (n.d.). Adrupilot. [Online]. Available: http://dev.ardupilot.com/wiki/license-gplv3/. Accessed Mar. 20, 2015.

[18] M. Torres. (2015, Mar. 10). Autonomous drones architecture - initial proposal. [Online]. Available: http://www.slideshare.net/mariohct/drones-architecture

[19] Pixhawk support. (n.d.). MathWorks. [Online]. Available: https://www.mathworks.com/hardware-support/forms/pixhawk-downloads.html. Accessed May. 20, 2015.

[20] "Performance specifications," class notes for Missile Flight and Control, Dept. of Mechanical and Aerospace Engineering, Naval Postgraduate School, Monterey, CA, spring 2015.

[21] H. Cyril and A. G. Piersol, *Harris’s Shock and Vibration Handbook*, vol. 5, New York: McGraw-Hill, 2002, p. 38.

[22] A. Ugural and S. Fenster, *Advanced Mechanics of Materials and Applied Elasticity*, 5th ed. Upper Saddle River, NJ: Prentice Hall, 2012. pp 645–652.

[23] 3DR product, Iris+ propellers. (n.d.). 3D Robotics. [Online]. Available: https://store.3drobotics.com/products/iris-plus-propellers. Accessed Jun. 10, 2015.

[24] J. B. Brandt and M. S. Selig, “Propeller performance data at low Reynolds numbers,” in *49th AIAA Aerospace Sciences Meeting*, Orlando, 2011. pp. 2011-1255.

[25] R. W. Deters, G. K. Ananda and M. S. Selig, “Reynolds number effects on the performance of small-scale propellers,” in *32nd AIAA Applied Aerodynamics Conf.*, Atlanta, GA, 2014. pp. 2014-2151.

[26] G. Anada. (2015, Feb. 03). UIUC propeller data site. [Online]. Available: http://m-selig.ae.illinois.edu/props/volume-1/propDB-volume-1.html

[27] K. Ogata, *Modern Control Engineering*, 5th ed. Upper Saddle River, NJ: Prentice Hall, 2010. pp. 567–595.

[28] S. D. Hanford, L.N. Long, J.F. Horn, “A small semi-autonomous rotary-wing unmanned air vehicle,” presented at AIAA InfoTech@Aerospace Conf., Washington, DC, 2005.

[29] S. Bouabdallah, A. Noth and R. Siegwart, “PID vs LQR control techniques applied to an indoor micro quadrotor,” in *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems*, Sendai, Japan, 2004, pp. 2451–2456.

[30] L. Meier. (n.d.). PX4 toolchain installation, PX4 autopilot. [Online]. Available: http://www.pixhawk.org/dev/toolchain_installation. Accessed May. 20 2015.

[31] QGroundControl download. (n.d.). QGroundControl. [Online]. Available: http://qgroundcontrol.org/downloads. Accessed May. 20, 2015.

[32] 3DR product: 3DR radio. (n.d.). 3D Robotics. [Online]. Available: https://store.3drobotics.com/products/3dr-radio-set. Accessed July. 19, 2015.

[33] DIY drones forum. (n.d.). DIY drones. [Online]. Available: http://diydrones.com/forum/topics/3dr-radios-will-not-connect-for-initial-setup. Accessed Aug. 6, 2015.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California