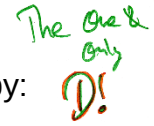
Programmable Flight Controller for Quadcopters

Submitted by: 

Submitted to: Mr. Raymond Mayer

Discipline: Electrical Engineering Technology

Date: 2022-04-22

Declaration of Sole Authorship

I, D!, confirm that this work submitted for assessment is my own and is expressed in my own words. Any uses made within it of the works of any other author, in any form (ideas, equations, figures, texts, tables, programs), are properly acknowledged at the point of use. A list of the references used is included.

Signed: 

Date: April 22, 2022

Acknowledgements

I would like to thank following people who have helped me directly or indirectly for making it possible for me to bring this system to life.

People	Contributions
Mr. Alan, The Captain! and all gardeners at the Hope and Memory Garden, Thunder Bay	For carving the D! out of me.
Mr. Raymond Mayer	• The list is so long that I don't even know where to start. • Briefly, for shaping my approaches to innovate, letting my curiosity run wild, and provide me just enough Johnson's noise to let my internal oscillator oscillate throughout this journey
UE!	• For encouraging me throughout this journey • For helping me setup the test system the very first time. • For helping me in soldering the connectors for battery.
Minh	• For helping me setup the test system. • For helping me find a small mistake in I/O expander PCB
Mr. Kevin	• For teaching me the foundations of telecommunications • And for giving me guidance on work ethics, patience, and honest work.
Mr. Drew	• For telling me his experiences regarding contact debouncing • And for teaching me the foundations of electrical and electronic systems.

Mr. Jamie	• For introducing me to the three ghosts that rule today's Controls world:
	  • Ghost of past -> I Controller
	  • Ghost of present -> P Controller
	  • Ghost of Future -> D Controller

Mr. Ian	• For keeping his cool while I irritated him constantly by knocking on his office door. • For making college resources available to me whenever I needed them.
Confederation College	• For giving me access to every resource I needed to work on this project.

Abstract

This project report describes all the major aspects of a highly programmable and modular flight-controller system that was designed and developed as a part of this project. The report also thoroughly discusses about the flight dynamics, how each sub-systems of the flight controller were created as well as the motive behind creating each sub-system, and what challenges were faced during every phase of the development. Additionally, several images and tables are provided to aid in the discussion of every section of this report.

Contents

Declaration of Sole Authorship ii

Acknowledgements iii

Abstract v

List of Illustrations, Diagrams and Tables ix

1.0 Introduction 1

2.0 Requirements 2

3.0 Degrees of Freedom 3

- 3.1 Frame of Reference 3
- 3.2 Rotational Degrees of Freedom 4
- 3.3 Translational Degrees of Freedom 6
- 3.4 Relationship between Rotational DOF and Translational DOF 8

4.0 Additional System Requirements 10

5.0 System Overview 12

- 5.1 Microcontroller 12
- 5.2 Sensors 13
- 5.3 Indicators 15
- 5.4 Communication devices 16
- 5.5 Mechanical and Electrical subsystem 17
- 5.6 Software 24
- 5.7 The complete system 27

6.0 Challenges 31

- 6.1 Selection of components 32
- 6.3 Software challenges 42
- 6.4 Debugging challenges 46
- 6.5 Tuning challenges 53
- 6.6 Printed Circuit Board (PCB) design challenges 55
- 6.7 “Putting it all together” challenges 59

7.0 Communications 61
7.1 Behavioral command and control channel 62
7.2 Telemetry and flight data 67
8.0 Interfacing Sensors 69
8.1 BMI160 69
8.2 BMM150 69
8.2 BMP388 70
8.3 Interfacing approach 71
8.3 Gravity Series Sensor 10 DOF (BMX160+ BMP388) 74
9.0 Controllers 76
9.1 Manual Controller 76
9.2 Automatic Controller 80
10.0 Setbacks and recommendations 84
10.1 Wrong sensor shipped: MPU 6050 instead of MPU9250 85
10.2 Drifts in altitude readings 85
10.3 System noise, and interfacing issues 86
10.4 Failed attempts in creating manual controllers 86
10.5 Sensor fusion 87
10.6 Receiver issues 87
10.7 Fire in the lab 87
10.8 PWM to servo control signals 87
10.9 Lost shipments and custom investigations 88
10.10 Calibration issues 88
10.11 Failed PCB milling 88
10.12 Failed ESCs in the final stage of testing 88
11.0 Learning Outcomes 89
12.0 Next Steps 90
13.0 References 91
Appendix A: cPPM signals 93

Appendix B: ESC programming 96

Appendix C: Micro electro-mechanical systems 98

Appendix D: Complimentary sensor fusion algorithm 99

Appendix E: Getting altitude from pressure sensor 101

List of Illustrations, Diagrams and Tables

Following table lists the images contained in this report along with their respective page numbers.

Figure	Page No.
Figure 1-1 Simple Multi-input Multi-Output Controller with closed loop feedback	1
Figure 3-1 Orientation of system with respect to its principal axis	3
Figure 5-1 Tiva Launchpad EK-TM4C123GXL	12
Figure 5-2 Wiring diagram of Audio-visual indicators	16
Figure 5-3 Receiver used for this project	16
Figure 5-4 Bluetooth slave module user for this project	17
Figure 5-5 Quadcopter frame used for this project	18
Figure 5-6 Physical dimensions of the frame	18
Figure 5-8 Physical dimensions of the motors used for this project	19
Figure 5-9 Different types of propellers used for this project	20
Figure 5-10 Propellers and motor rotation direction for individual motors within the system	20
Figure 5-11 Connection of single ESC, motor, battery and host microcontroller unit.	21
Figure 5-12 Signals from Microcontroller to ESC	22
Figure 5-13 Signals from ESC to motor	22
Figure 5-14 Power distribution board physical layout	23
Figure 5-15 Schematic diagram of Power distribution board	23
Figure 5-16 User interaction state-machine	25
Figure 5-17 Software stack developed for this system	26
Figure 5-18 Schematic of overall system	27
Figure 5-19 Entire system during development phase of the project	29
Figure 5-20 Entire system after assembling the system	30
Figure 5-21 Entire system during testing phase of the project	30

Figure 6-1 Frame selected for project	33
Figure 6-2 Different types of propellers	34
Figure 6-3 Direction of rotation of each motor	34
Figure 6-4 Type of motor used for this project	35
Figure 6-5 Motor thrust data provided by manufacturer	36
Figure 6-6 Physical dimensions of motors used for this project	37
Figure 6-7 Electronic Speed Controller programmer	38
Figure 6-8 Burnt ESC	39
Figure 6-9 Failed ESC	39
Figure 6-10 Connection diagram of an ESC with battery, motor and host Microcontroller	40
Figure 6-11 PWM signals representing motor speed	41
Figure 6-12 Servo control signals sent to each ESC	41
Figure 6-13 Workflow for developing individual functionality of the system	42
Figure 6-14 Entire software stack developed for this project	44
Figure 6-15 Implementation of Low-pass filter in software	45
Figure 6-16 Data from channel 1 modulated using cPPM encoding	47
Figure 6-17 Data from channel 1=2 modulated using cPPM encoding	48
Figure 6-18 Data from channel 3 modulated using cPPM encoding	48
Figure 6-19 Data from channel 4 modulated using cPPM encoding	49
Figure 6-20 Data from channel 5 modulated using cPPM encoding	49
Figure 6-21 Data from channel 6 modulated using cPPM encoding	50
Figure 6-22 Hidden channel in the radio transmission	50
Figure 6-23 cPPM signals decoded for each radio channel	51
Figure 6-24 I/O expander I2C debugging PCB	51
Figure 6-25 Mobile application configured to send/receive data to/from flight-controller	54
Figure 6-26 Complete system on the breadboard	55
Figure 6-27 Main Flight-controller PCB created for this project	56
Figure 6-28 Flight-controller PCB designed in software (Top layer)	57
Figure 6-29 Flight-controller PCB designed in software (Bottom layer)	57
Figure 6-30 Milled PCB board (Bottom bayer)	58

Figure 6-31 Milled PCB board (Top layer)	58
Figure 6-32 Workflow for developing and testing the drivers of individual sensors	59
Figure 6-33 Hardware fix for Buzzer problem	60
Figure 7-1 Two aspects of communication	61
Figure 7-2 Receiver mounted on the system	62
Figure 7-3 Different radio protocols used for communication	64
Figure 7-4 cPPM signal when channel 3 was at 0%	64
Figure 7-5 Transmitter used for this project	65
Figure 7-6 Bluetooth module mounted on the flight-controller PCB	67
Figure 7-7 Mobile application configured for this project	68
Figure 8-1 BMX160 (BMI160 + BMM150) breakout board	70
Figure 8-2 BMP388 breakout board	70
Figure 8-3 Block diagram of BMX160	71
Figure 8-4 Block diagram of BMP388	72
Figure 8-5 Schematic diagram of Gravity sensor (BMI160 + BMM150 + BMP388)	74
Figure 8-6 Breakout board for Gravity sensor	74
Figure 8-7 BMX160 + BMP388 mounted on the Flight-Controller PCB	75
Figure 9-1 Implementation of control rules in software	78
Figure 9-2 Implementation for setting max roll, pitch and yaw angle in software	78
Figure 9-3 Illustrated behavior of quadcopter when it is commanded to go right and forward at same time	79
Figure 9-4 Block diagram of Automatic controller sub-system with feedback path implemented in this project	81
Figure 9-5 Internals structure of Automatic controller implemented for this project	82
Figure 10-1 Interrupt signal extraction by modifying the breakout of MPU9250	85
Figure A-1 Outputs of the receiver	93

Figure A-2 Signals sent to the ESCs	93
Figure A-3 cPPM signal for channel 3 when the data transmitted was -100% (Lowest value)	94
Figure A-4 cPPM signal for channel 3 when the data transmitted was 0% (Mid value)	94
Figure A-5 cPPM signal for channel 3 when the data transmitted was +100% (Lowest value)	95
Figure A-6 Servo control signals	95
Figure B-1 ESC Programmer used for this project	96
Figure D-1 Block diagram of Complimentary Filter Algorithm implemented for this project	99
Figure D-2 Code listing of Complimentary Filter Sensor Fusion algorithm	100
Figure E-1 Relationship between Pressure and Altitude	101

Following table lists the images contained in this report along with their respective page numbers.

Table	page
Table 3-1 Individual Rotational Degrees of Freedoms and relative motor speeds	4-5
Table 3-2 Individual Translational Degrees of Freedoms and relative motor speeds	6-7
Table 3.3 Relationship between Rotational DOF and Translational DOF	8-9
Table 5-1 Information of each sensor used for this project	13
Table 5-2 Images of IMU sensors used for this system	14
Table 5-3 Block diagrams of IMU sensors used in this system	15
Table 5-4 Transitions conditions for user interaction state-machine	25
Table 5-5 Pin mapping of each external devices with microcontroller	28-29
Table 7-1 Radio channels and the type of data transmitted through them	63
Table 7-2 All Bluetooth commands that were supported during testing phase	68
Table 9-1 Control rules for Manual controller	77
Table 9-2 Types of controllers internal to Automatic controller	81
Table 9-3 Automatic controller Proportional and Integral gains	83
Table B-1 Configuration of each ESCs for this project	97

1.0 Introduction

One of the driving factors of today's technological advancements and innovations in any field is our curiosity and desire to understand that field better. These technological advancements not only allow us to understand a particular aspect of any system, but they also help us to create a better and more reliable systems that interact with real world. We create these systems based on the knowledge we acquire by studying and analyzing the field of interest.

The field of Unmanned Ariel Vehicles (UAVs) is no exception. One of the leading systems in this category is drone technology, which includes systems that can fly very precisely and accurately in the air. These movements are possible because of a complex control system which acts essentially as 'brain' of the drone. The control system gets its inputs from several sensors and actuators including the feedback from multiple outputs and decides how to change one or several outputs such that the system interact with the real world the way its user wants.

A block diagram of two-inputs two-outputs control system with closed loop feedback is shown below. A drone employs similar control system but with a greater number of inputs, outputs, and control systems, which makes sure that the drone does what it is expected to do.

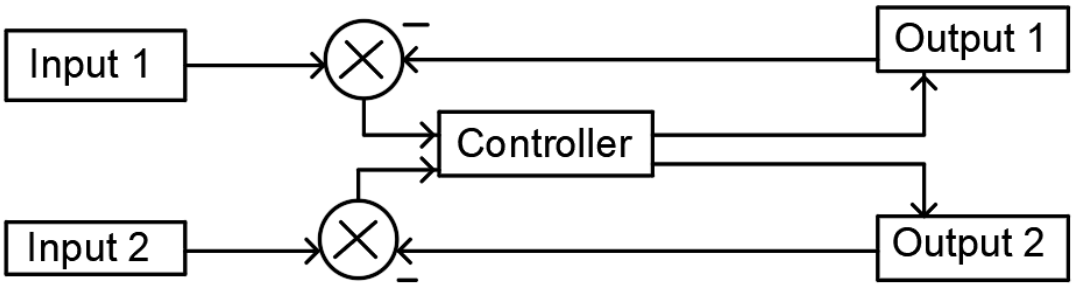


Figure 1-1 Simple Multi-input Multi-Output Controller with closed loop feedback

2.0 Requirements

There are many different types of drones with varying size, types of onboard sensors, number of propellers, types of motors and motor controllers. By having a programmable and configurable flight controller, pilots have more control over the entire system and the system can be easily tuned for specific applications.

Therefore, the main requirement for this project was to build a functional, programmable, and modular flight controller for quadcopters. The reason behind developing such a system was the lack of programmable and configurable off-the-shelf flight-controllers.

Thus, building a working prototype of quadcopter that provides the ability to re-configure the drone dynamics depending on the application was a major goal for this project. Additionally, the system needed to be designed such that it allows easy interfacing and programming of external sensors and works with wide variety of on-board equipment.

The project was also selected based of the complexity and the applications of previously gained knowledge in academic settings. Therefore, the proper balance of knowledge and skills already acquired and the additional knowledge and skills that was needed to create exciting system was very important factor when deciding to pursue this project.

The theoretical knowledge required to create this system included:

- Application of differential equations
- Controls theory
- Telecommunication theory
- Electrical and Electronics system theory
- Fundamentals of physics for three-dimensional motion
- Aerodynamics, power-distribution theory, and thrust-to-weight ratio analysis
- Software and real time embedded system design
- Signal flow analysis

The practical aspects of this project involved:

- Design electrical system for a quadcopter
- Integrate various off-the-shelf components and create a complete working system
- Hands on experience with various tools and methods for creating robust electrical system
- Designing and creating PCBs
- Troubleshooting various systems that are susceptible to signals interference
- Minimizing noise on both hardware and software level.

3.0 Degrees of Freedom

Degrees of freedom (DOF) defines the atomic movements that the drone can perform in real world. These motions are achieved by changing the speed of individual motors such that the net force on the system is acting in the desired direction, thus changing the state of system accordingly.

There are two subcategories of degrees of freedom:

- Rotational DOF
- Translational DOF

Rotational DOF defines the atomic movements of drone when the drone is held fixed but is allowed to rotate *about* its principal axis. Translational DOF defines the movements of drone when it is allowed to move *along* its principal axis.

3.1 Frame of Reference

Before showing the atomic movements the system can perform, it helps to identify how the system is oriented and what are its principal axis. The image below captures how the system is oriented with respect to its principal axis. Please note that the colors of frame arms were modified to easily identify the direction of rotation of individual motors.

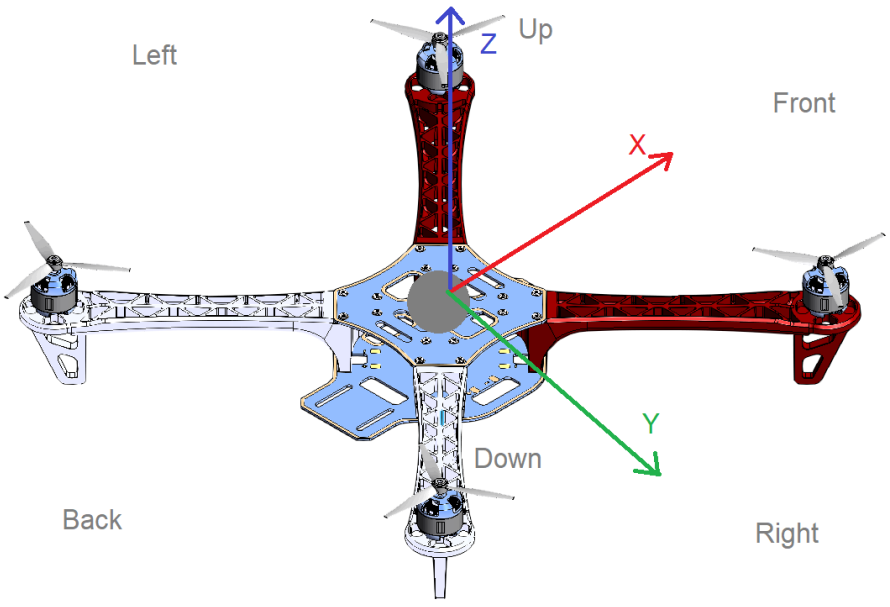


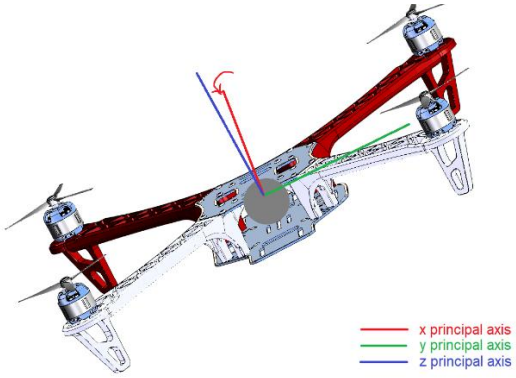
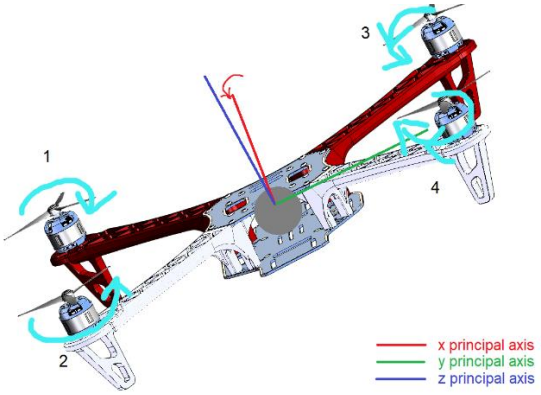
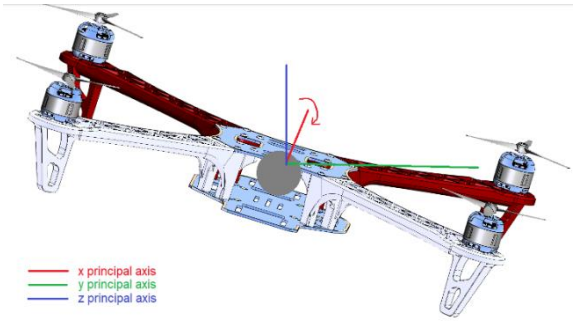
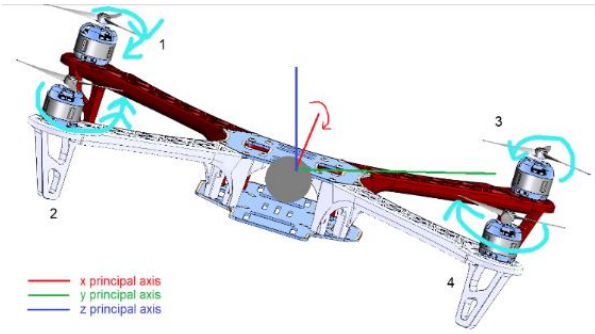
Figure 3-1 Orientation of system with respect to its principal axis

3.2 Rotational Degrees of Freedom

The rotational degrees of freedom this system can perform include:

- Roll (about X-axis).
- Pitch (about Y-axis).
- Yaw (about Z-axis).

The following table shows various rotational DOF that the system can perform. The relative speed of each motor for each rotational motion is also shown.

Sr. No.	Motion Name	Motion	Relative motor speed
1	Roll Left		
2	Roll Right		

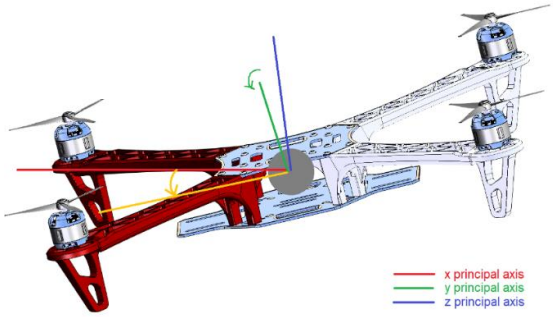
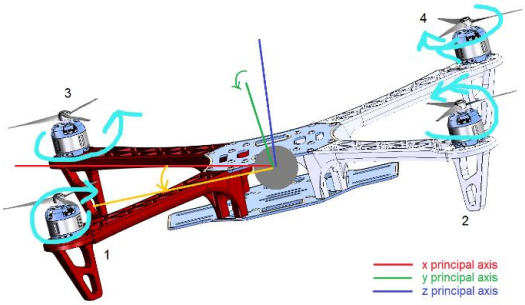
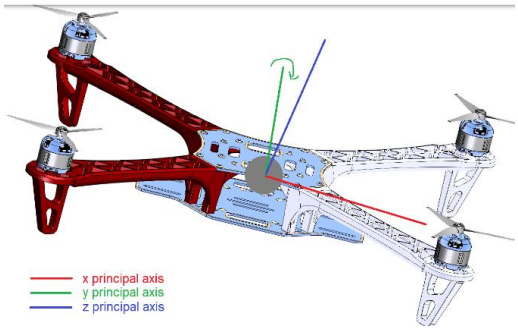
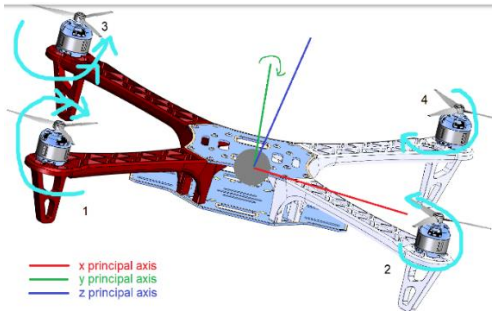
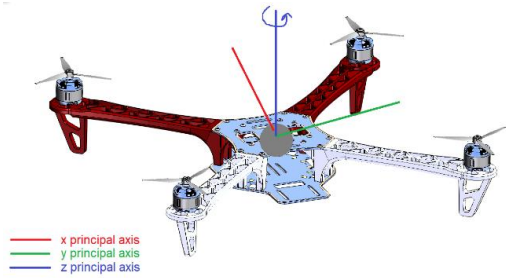
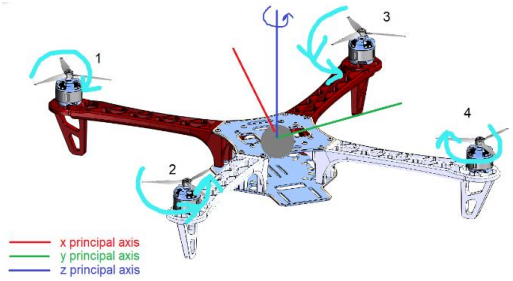
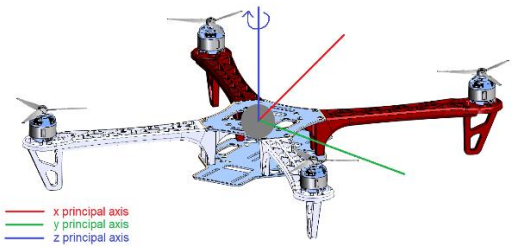
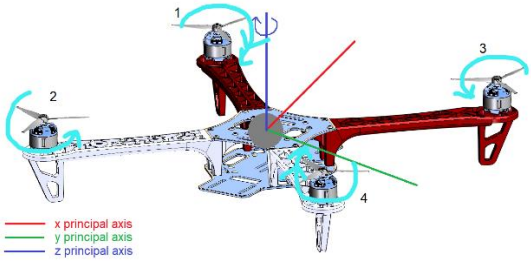
3	Pitch Forward		
4	Pitch Backward		
5	Yaw Left		
6	Yaw Right		

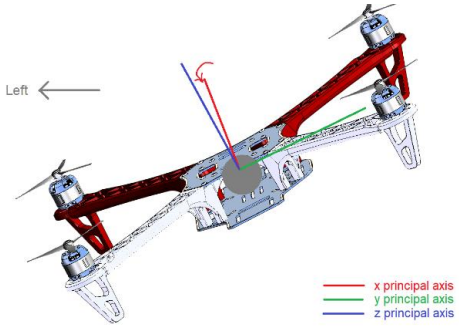
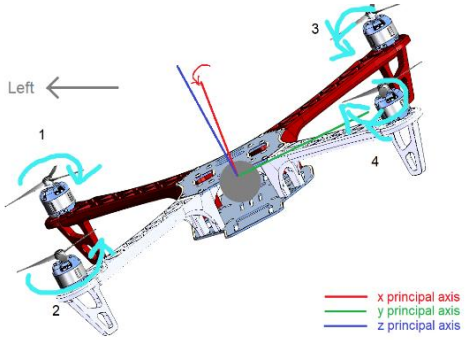
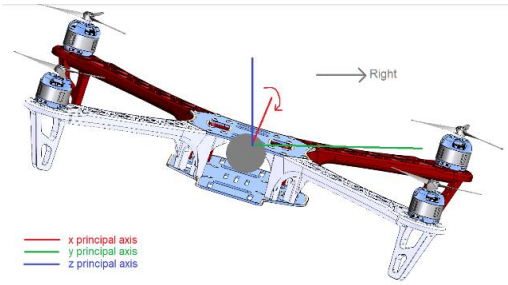
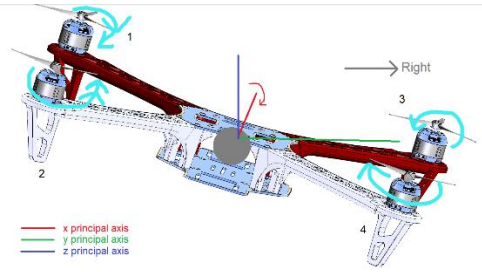
Table 3-1 Individual Rotational Degrees of Freedoms and relative motor speeds

3.3 Translational Degrees of Freedom

The translational degrees of freedom this system can perform include:

- Left-Right (along X-axis).
- Forward-Backward (along Y-axis).
- Up-Down (along Z-axis).

The following table shows various translational DOF that the system can perform. The relative speed of each motor for each rotational motion is also shown.

Sr. No.	Motion Name	Motion	Relative Motor Speed
1	Move Left		
2	Move Right		

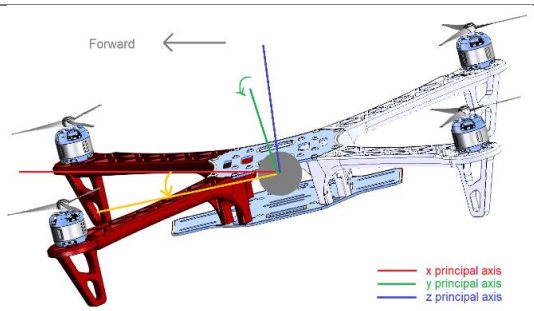
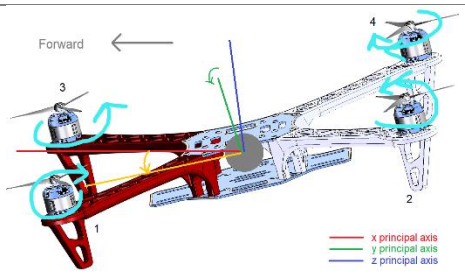
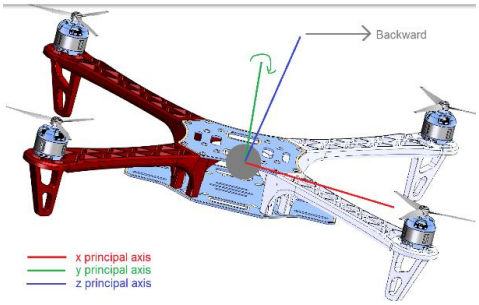
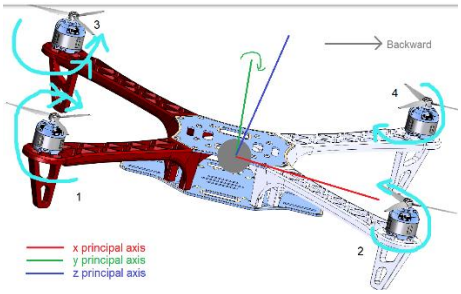
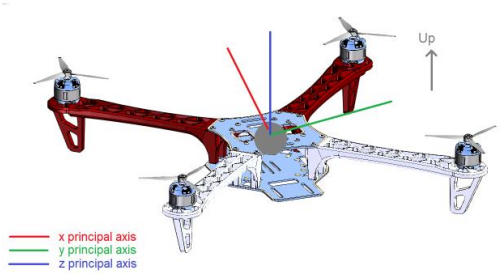
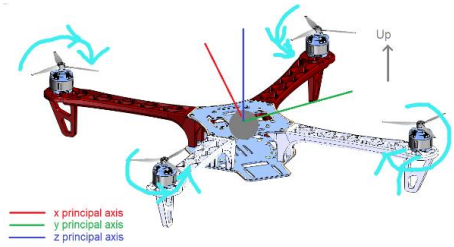
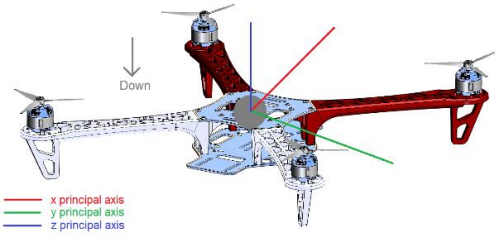
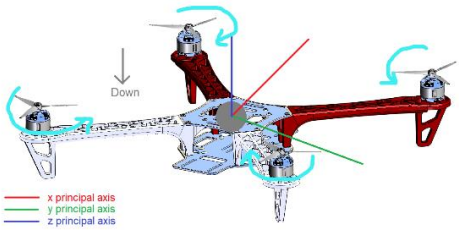
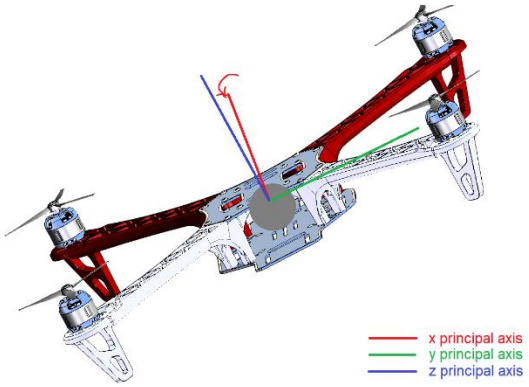
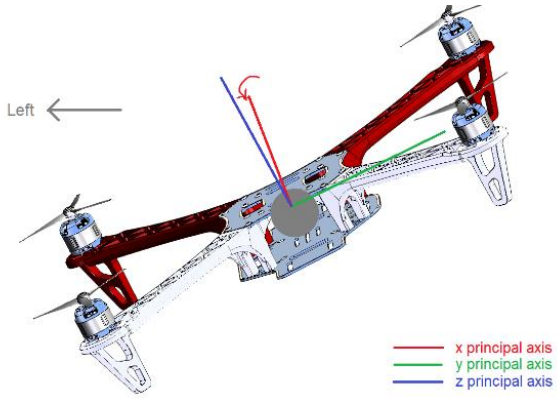
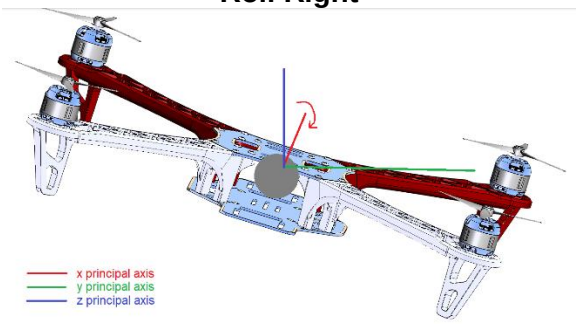
3	Move Forward		
4	Move Backward		
5	Move Up		
6	Move Down		

Table 3-2 Individual Translational Degrees of Freedoms and relative motor speeds

3.4 Relationship between Rotational DOF and Translational DOF

From the previous discussion, following equivalence between the Rotational DOF and Translational DOF can be established:

Rotational DOF	Translational DOF
Roll Left 	**Move Left** 
Roll Right 	**Move Right** 

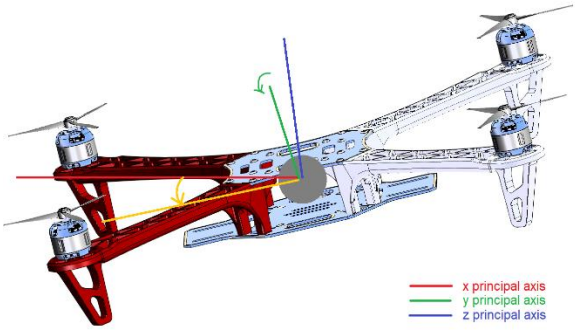
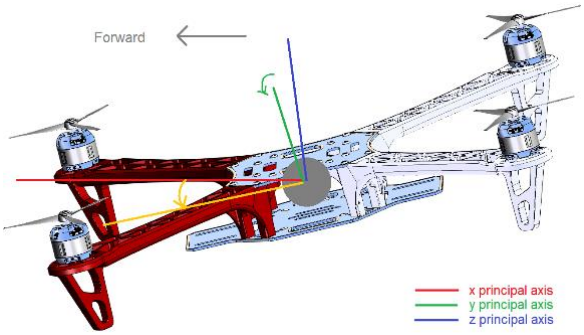
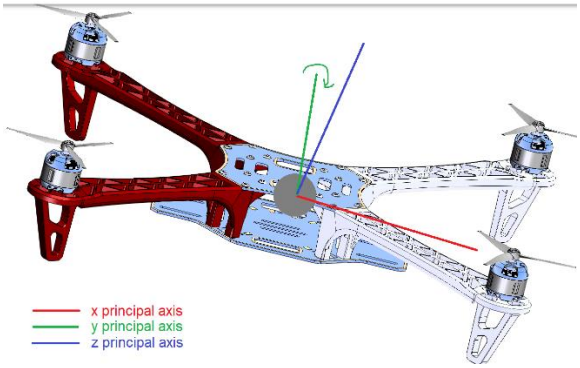
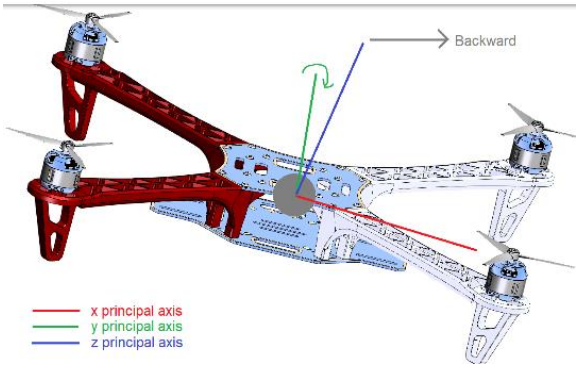
Pitch Forward	Move Forward
	
Pitch Backward	**Move Backward**
	

Table 3.3 Relationship between Rotational DOF and Translational DOF

4.0 Additional System Requirements

Although the main goal for this project was to create a working prototype of a quadcopter along with highly modular and programmable flight controller, there were many other system requirements which ensured that the system is safe to operate, is reliable, does not behave unexpectedly and have appropriate fail-safe routines.

Safety of operator, other people, environment, and the system itself was given the highest priority during each phase of the development of this system. Several safety measures are taken to ensure that the system is safe to operate and does not surprise user during the operation of the system.

Some of these measures to ensure safety of the user include:

- Emergency stop switch (E-stop)
- Forced initial state
- Audio and visual feedback of the system to the user
- Receiver connection link status and integrated safe open fail-over.

A dedicated channel (Channel 6) is used in radio transmission that allows user to stop the flight controller at any time. This switch does not shutdown the entire system, but only the sub-systems that control the flight of the system. Thus, this switch acts as a master control for stopping the motors safely if the user loses control of the system. Additionally, the audio and visual feedback of the system state allows user to know the status of the system.

When a system is first booted up, the software ensures that the initial state of the system is always known, and the software also makes sure that the system does not start the motors suddenly. The operator can start the flight only after following through the sequence of predetermined states to initiate the flight operation.

Because of its complexity, the software within the system employs a comprehensive audio-visual indication system that always indicates the current state of the flight controller. Also, the requirements for detecting receiver link status and taking appropriate actions if the receiver connection is lost were satisfied. For example, landing safely if the receiver fails.

Other important requirements for this project included safety of the system itself. These requirements included:

- RPM tracking for safe landing.
- Gradual RPM reduction when E-Stop is pressed.
- Reconfigurable individual Motor gains and PI Controller gains.
- Re-adjustable Roll and pitch max angles (to make sure the system doesn't go in unstable state.)
- Measures to reduce the effects of system vibrations on main flight controller system.

Additionally, it was a necessity to make the operation of the system a very straightforward process. Therefore, a complete system that is easy for user to use to control various aspect of flights was needed. An internal state machine is used to make the process of controlling the system and changing flight modes easier. This state machine adds a layer of reliability so that the user does not have to do complex transitions to do certain operations. This makes the overall system user friendly, which was also a very important requirement. At any time, the system is in one of the following states:

- Receiver not connected
- Idle (Receiver connected, but E-Stop engaged)
- Calibration
- Manual Control
- Automatic Control

Some other important requirements for this system included:

- Separate wireless command and Telemetry communication channel over Bluetooth.
- Calibration (Zeroing) routine for various sensors.
- Extra power-rails and I2C bus slots for inclusion of additional sensors.

5.0 System Overview

The goal of this section is to provide an overview of the overall system and describe each individual components that make up the entire system. Detailed description of selected components will be discussed in later sections.

5.1 Microcontroller

This component essentially houses majority of the peripherals and circuitry that are required to store and execute the flight controller software. Therefore, this component essentially acts as the brain of the system. The microcontroller used for this project was TM4C123GH6PM by Texas Instruments[1]. Instead of creating a custom breakout board, a development board by Texas Instrument (EK-TM4C123GXL)[2] was used for this project, which allowed rapid prototyping and development of the system.

The reason for selecting this microcontroller was previous experience with this microcontroller and associated development environment (Code Composer Studio)[4]. Another reason for choosing this microcontroller was its wide offering of peripherals that would be necessary to develop such system. The peripherals internal to the microcontroller, which were used for this project include:

- GPIO
- GPTM
- I2C
- UART
- PWM
- SYSTICK

The following image shows the development board used for this project.

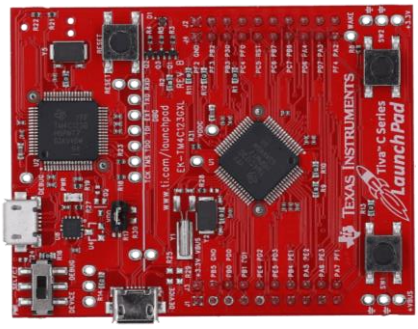


Figure 5-1 Tiva Launchpad EK-TM4C123GXL

5.2 Sensors

The sensors used in this project included Inertial measurement units (IMUs) as well as other sensors to determine the system state (system attitude) and provide the feedback to the controllers. The following table lists each sensors whose support functions were developed in this project and the purpose of each sensor along with their default state.

SR. NO	SENSOR NAME	SENSOR TYPE	DEFAULT STATE
1	BMI160[5]	Accelerometer	• Enabled • Power mode -> Normal Mode • Filter -> Normal mode with 1600 Hz sampling rate • range -> 2G • Communication -> I2C (400 kHZ)
		Gyroscope	• Enabled • Power mode -> Normal Mode • Filter -> Normal mode with 3200 Hz sampling rate • range -> 125 dps (Degrees per second) • Communication -> I2C (400 kHZ)
2	BMM150[6]	Magnetometer	• Enabled • Power mode -> Active Mode • Output Data Rate – 12.5 Hz • Number of repetitions for x/y axis -> 47 • Number of repetitions for z axis -> 83 • Communication -> I2C (400 kHZ)
3	BMP388[7]	Pressure Sensor	• Enabled • Power Status -> Enable • IIR Filter coefficient -> 127 (Max Filtering). • Output Sampling Rate -> x8 oversampling • Output Data Rate -> 50Hz (20 ms) • Communication -> I2C (400 kHZ)
		Temperature Sensor	• Enabled • Power Status -> Enable • IIR Filter coefficient -> 127 (Max Filtering). • Output Sampling Rate -> No Oversampling • Output Data Rate -> 50Hz (20 ms) • Communication -> I2C (400 kHZ)
4	HC-SR04	Ultrasonic sensor	• Disabled
5	MPU9250[9]	Accelerometer, Gyroscope, magnetometer	• Disabled

Table 5-1 Information of each sensor used for this project

The table below shows the images of each sensor that were mounted on the drone:

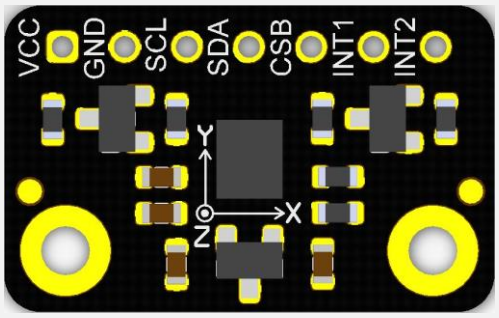
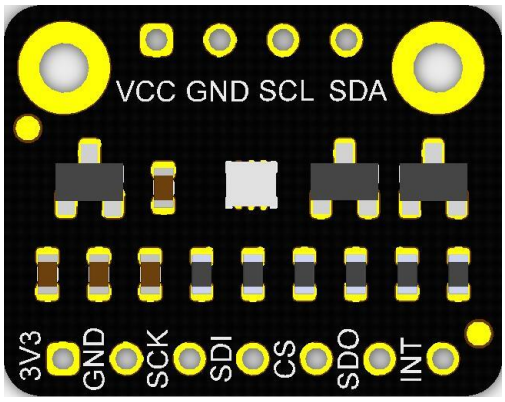
SR. NO.	SENSOR	IMAGE
1	BMX160[8] (BMI160 + BMM150)	
2.	BMP388[7]	

Table 5-2 Images of IMU sensors used for this system

The reasoning behind selecting the above sensors was the power requirements of the sensors as well as their applications in drone technology. Also, the above sensors are highly sensitive and configurable to meet the needs for this project.

The following table contains the block diagram of the above sensors.

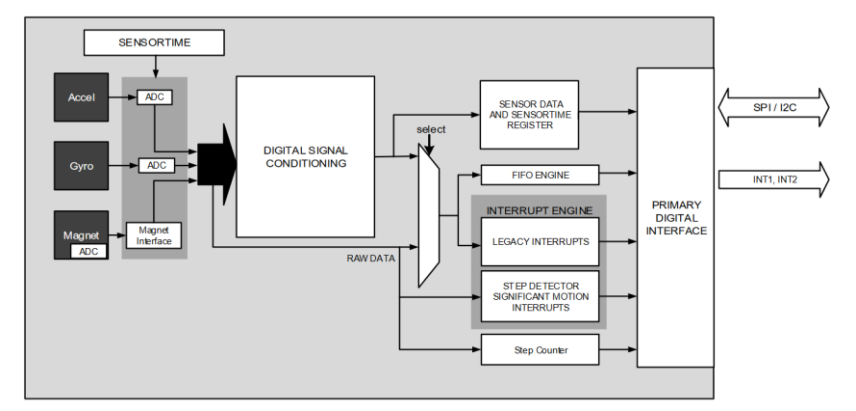
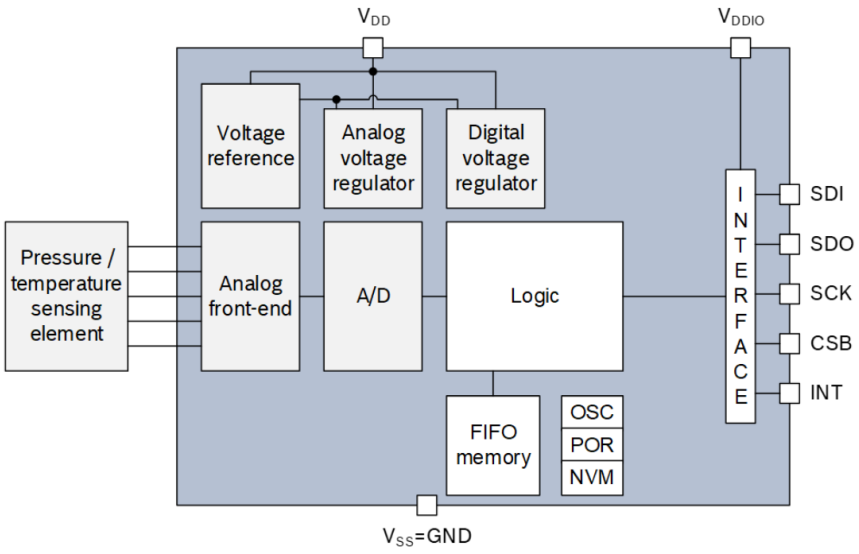
SR. NO.	SENSOR	BLOCK DIAGRAM
1	BMX160[8] (BMI160 + BMM150)	
2.	BMP388[7]	

Table 5-3 Block diagrams of IMU sensors used in this system

For more information on these sensors, please refer to their datasheet. The links to datasheets are attached in the reference section of this report.

5.3 Indicators

To indicate the system state, this project uses the onboard LEDs as well as external sound module. These both devices are accessed by using GPIO peripheral within the microcontroller. The image below shows the wiring of microcontroller and both indicators:

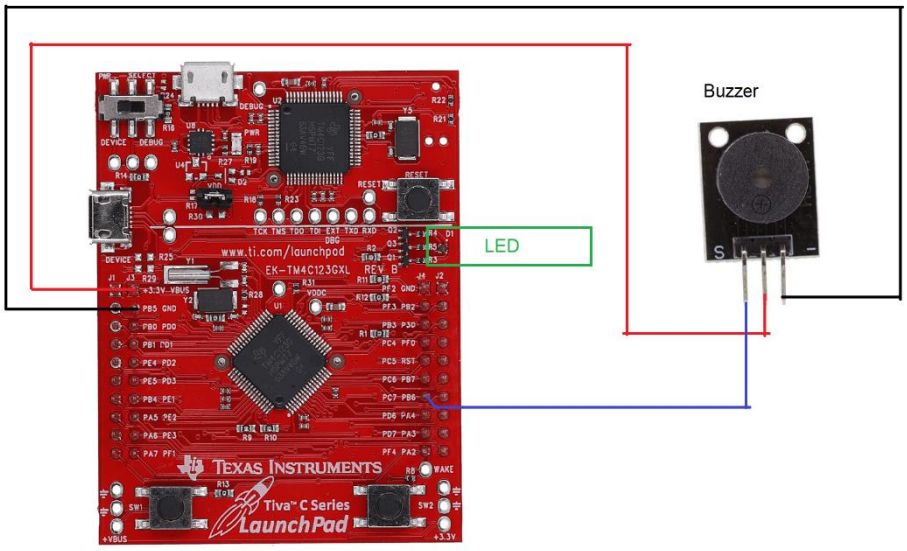


Figure 5-2 Wiring diagram of Audio-visual indicators

5.4 Communication devices

5.4.1 Radio Receiver

This device receives command from operator's transmitter through radio transmission. The receiver used in this project is a combined Pulse Position Modulation receiver, which can work with wireless protocols using cPPM modulation to send data over radio channels. A detailed description of this device is given in later section. The image of receiver is shown below. Further description of cPPM modulation is given in Appendix A.

Figure 5-3 Receiver used for this project [11]

5.4.2 Bluetooth Module

This is a UART-to-Bluetooth slave module that allows the user to receive telemetry data. A user can easily send commands as well as receive information to/from the flight controller system using a terminal application. The application configured for this project was called "Serial Bluetooth Terminal". A later section on device communications

elaborates on this further. The following image shows the Bluetooth module used in this project:

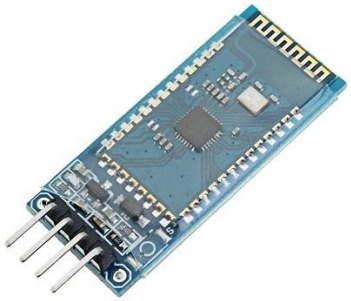


Figure 5-4 Bluetooth slave module user for this project [12]

5.5 Mechanical and Electrical subsystem

This sub-section describes each mechanical and electrical sub-systems that was as a part of the quadcopter.

5.5.1 Frame

The frame is the main support structure for the quadcopter. This is a pure mechanical part with four detachable arms and two lightweight bases (Top and Bottom). The frame used for this project spans approx. 500 mm diagonally and it is suitable for less agile system. The frame was chosen because it is large enough to accommodate the flight controller board and various external sensors used in this project. The frame is also very durable, lightweight, inexpensive, and is easily available on the market.

The following images shows the frame used in this project.

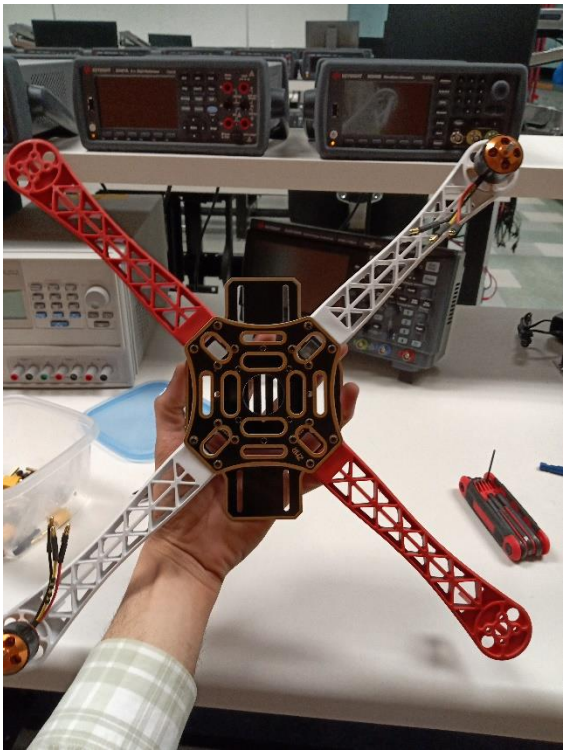


Figure 5-5 Quadcopter frame used for this project

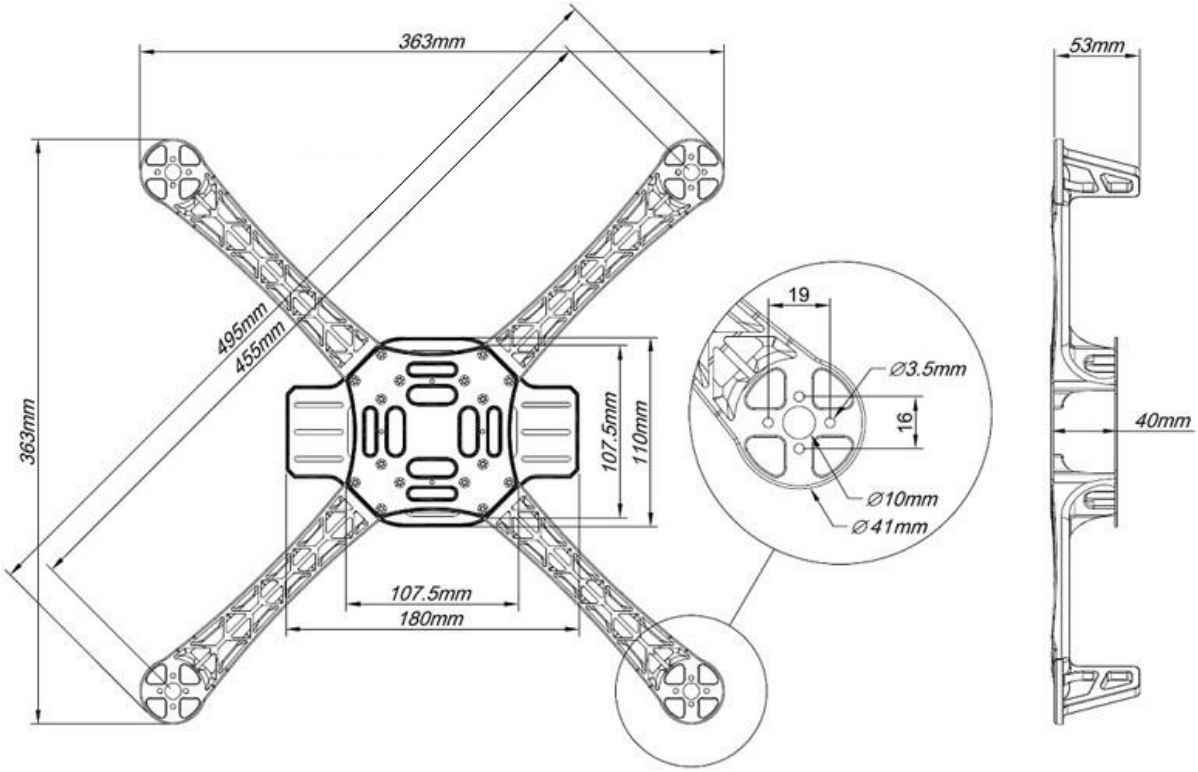


Figure 5-6 Physical dimensions of the frame [15]

5.5.2 Motors

The motors used in this project are Brushless DC (BLDC) motors with 11000 rated RPM at 11.1 V. The motors were selected because of the stator size and thrust test data provided by the manufacturer. It was very important to get the correct stator size motor to make it compatible with the frame of the quadcopter. Further discussion on the motors used in this project is given in a later section.

The following images shows the motors used in this project.

Figure 5-7 Motors used for this project [13]

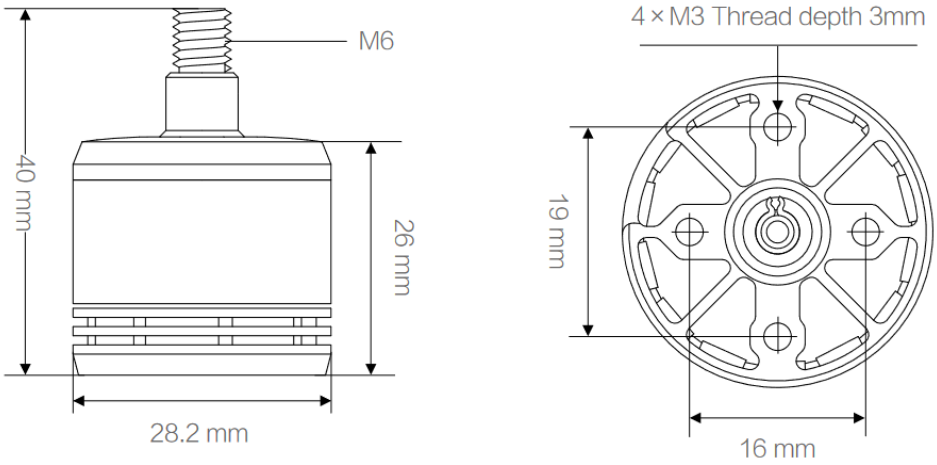


Figure 5-8 Physical dimensions of the motors used for this project [13]

5.5.3 Propellers

The propellers used with this system are 1045 propellers. The diameter of propellers is 10 inches, and the pitch is 4.5 inches. Pitch of propeller is important in determining how fast the system can travel or lift. Diameter and pitch both determine the amount of thrust produced by a given motor/propeller combination.

Propellers are made such that they can be mounted on only specific motor shaft diameter. Therefore, these propellers were chosen to ensure that the system produces enough lift force with the given motor and frame.

Note that the propellers used in this project are not all similar. Two propellers are designed to rotate in *clockwise* direction (CW)and the other two propellers are designed to rotate in *counterclockwise* direction (CCW).

The following diagram shows both types of propellers and how each one is connected to the quadcopter.

Figure 5-9 Different types of propellers used for this project

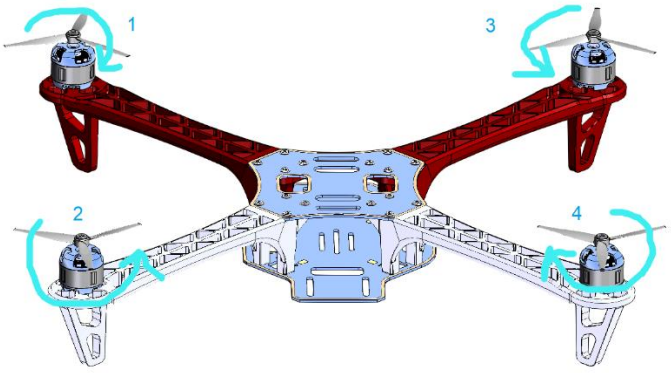


Figure 5-10 Propellers and motor rotation direction for individual motors within the system

5.5.4 Motor Controllers

Since the motors used in this project are BLDC motors, the speed control of the motor is not a straightforward task. The three phases of the motors need to be switched at precise time to control the speed of the motor. Additionally, to determine the RPM of any motor at any given time, the back electromotive force (back emf) generated by that motor needs to be measured. The RPM can then be determined indirectly from back emf.

To make this process simple, BLDC speed controllers, also called *Electronic Speed Controllers* (ESCs) are used for this project. Electronic Speed Controller is an electronic circuit that aids in the process of controlling the BLDC motor by providing the circuitry required for switching three-phases of the BLDC motors at a required rate. These controllers essentially convert DC-to-AC, and they can indirectly determine the speed of the motor by measuring back emf generated by the attached motor.

ESCs employ a microcontroller, which has a firmware inside it which takes care of complex switching algorithm as well as measuring and acting on the back emf. This firmware needs to be configured for each controller to make the controllers aware about the system they will be controlling.

The following images show how a single ESC is connected to a battery, a microcontroller, and a motor:

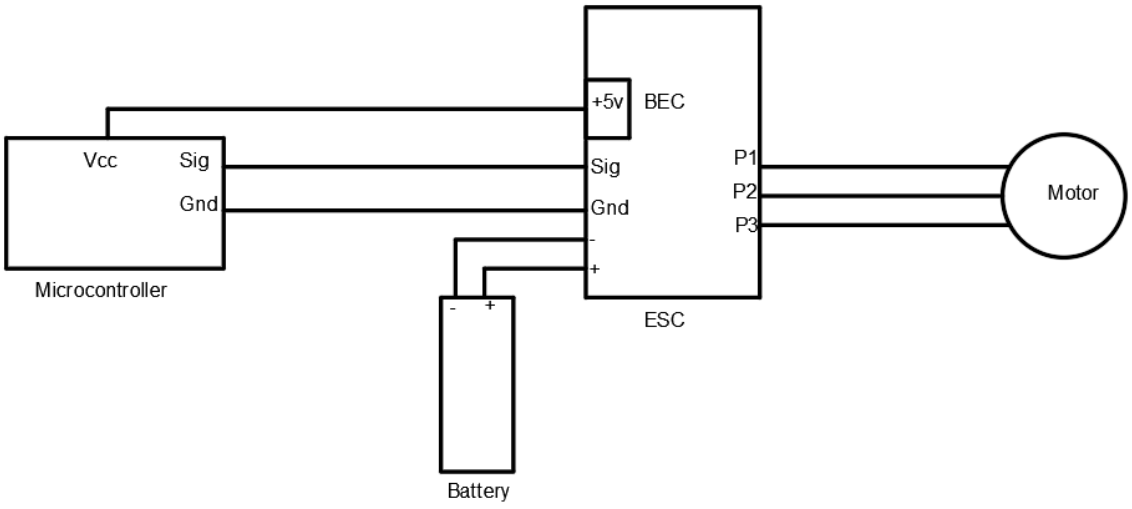


Figure 5-11 Connection of single ESC, motor, battery and host microcontroller unit.

The following image shows the signals from microcontroller to ESC, and from ESC to motor:

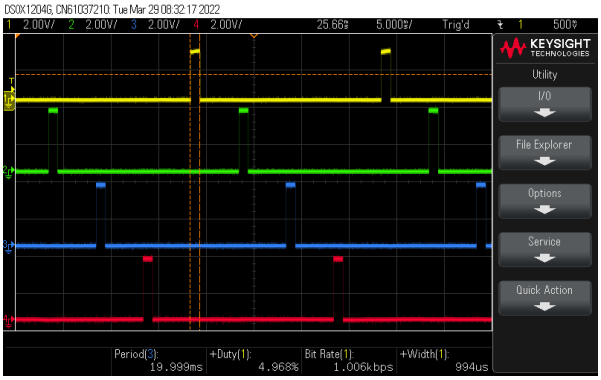


Figure 5-12 Signals from Microcontroller to ESC

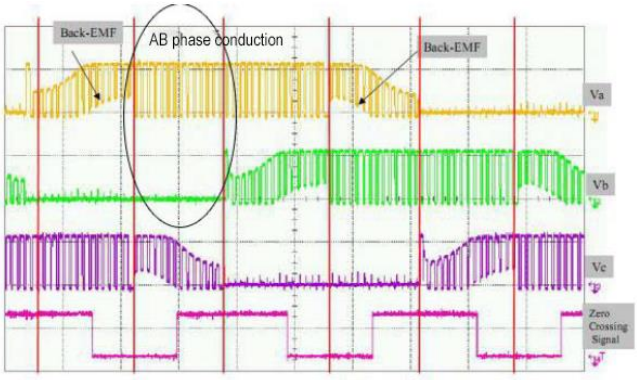


Figure 5-13 Signals from ESC to motor [18]

The controllers used in this project are the Hobby King's 20A UBEC. These controllers were chosen because they also employ an additional circuit, called a *Battery Elimination Circuit* (BEC). This circuit can act as a power source for the flight-controller system and other external sensors. The BEC rating is 3 amps at 5 V. The later section discusses motor controllers further.

5.5.5 Power Distribution Board

The *power distribution board* (PDB) is a small circuit board which eases the connection of multiple ESCs with single power supply. This is essentially the electrical distribution subsystem of the drone, which distributes the required power to each ESCs. This power in turn is used to spin the motor and some of it is used to power the external sensors and the speed controllers itself.

The following images shows the Power distribution board used in this project as well as the schematic diagram that depicts how PDB is connected to four speed controllers (ESCs).

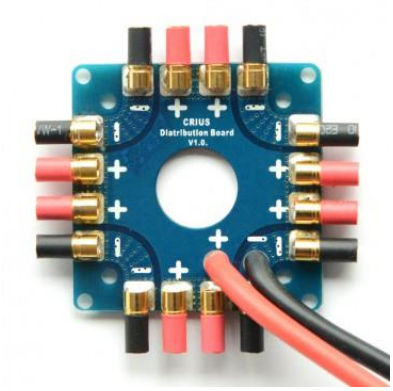


Figure 5-14 Power distribution board physical layout

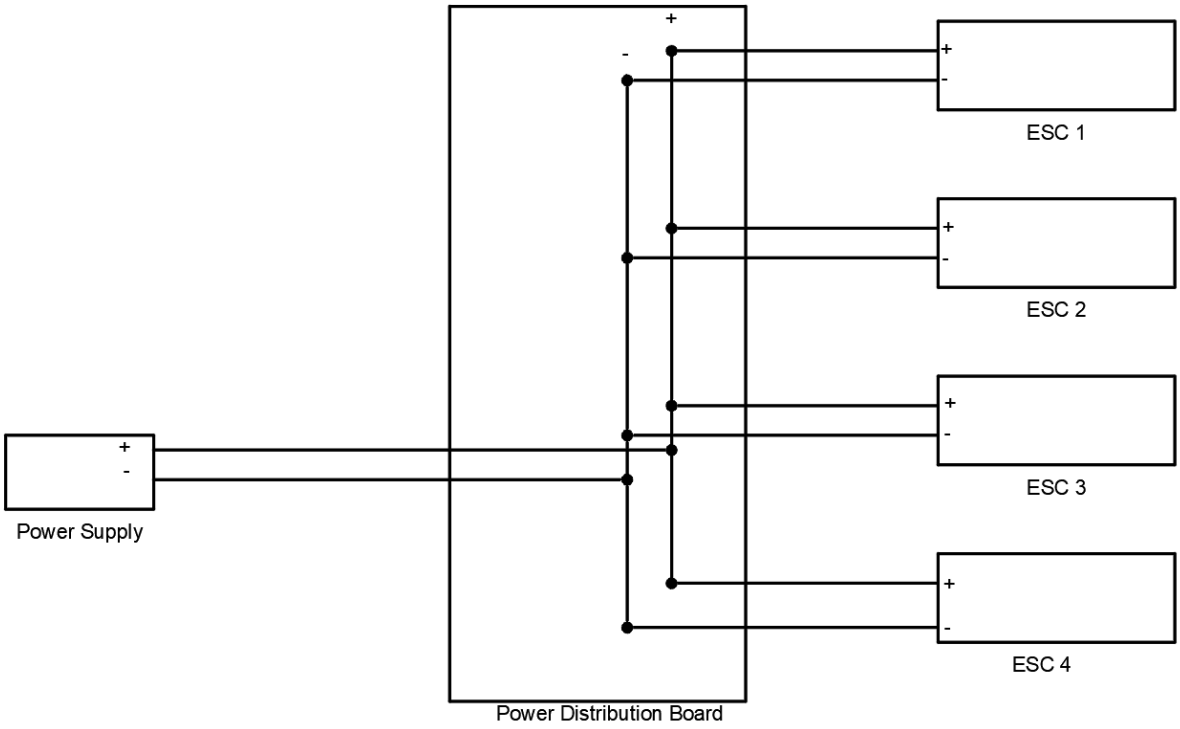


Figure 5-15 Schematic diagram of Power distribution board

5.6 Software

This project included not only writing software for flight controller, but also for all other functionalities, support functions, interfacing and debugging functions. The software needed to be built such that the user should be able to configure wide variety of system settings that essentially determine how the flight controller would behave for a given application.

The approach used to develop the software stack for this project was the Bottom-up approach. For any external hardware device (Sensors, receivers, indicators, etc.), the lower-level drivers were first developed, followed by middle and upper-level APIs for that specific device. All the lower-level code was developed and tested before moving to the upper-level API development.

The software stack architecture developed for this project was inspired from the idea of OSI network model[17] in data communication over computer networks. Each layer above uses the service provided by one or more lower layer. The user layer is the top layer where user interacts with the system. User never directly interacts with the layers below the user layer. Thus, the software system has several abstraction layers that dictate which code sections can interact with each other and how they can share information. This also allowed to enable/disable/bypass any given code section in the software stack very easily during debugging.

There is an internal state machine which allows the operator to interact with drone and change its operational modes. The operator can also tune the system by interacting with one of the software modules residing in user layer, which is responsible for communications with the user. The state-transition diagram of the state machine used to control the drone’s status is shown below. The description of state transition conditions follows the state-transition diagram.

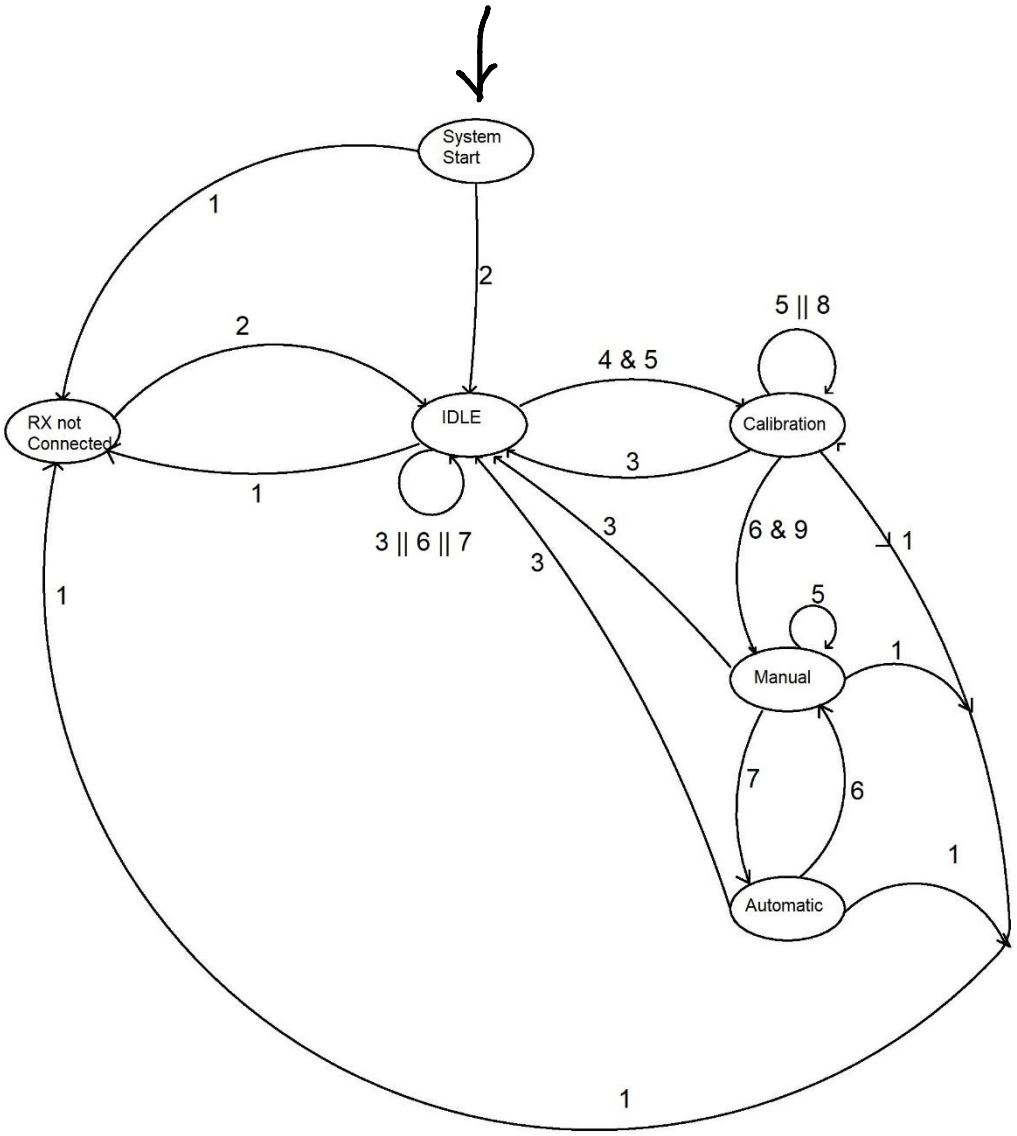


Figure 5-16 User interaction state-machine

Transition Condition		Description
	1	Receiver is not connected
	2	Receiver is connected
	3	E-Stop is engaged
	4	E-Stop is not engaged
	5	Selected Flight Mode is Calibration
	6	Selected Flight Mode is Manual
	7	Selected Flight Mode is Automatic
	8	BMP388 is not Calibrated
	9	BMP388 is Calibrated

Table 5-4 Transitions conditions for user interaction state-machine

There are multiple such code blocks written for this project. The following software stack shows each one of the major code blocks and how they are arranged internally to use each other's services. Each block in this diagram has several support functions and state variables. The complete discussion of this software stack is beyond the scope of this report.

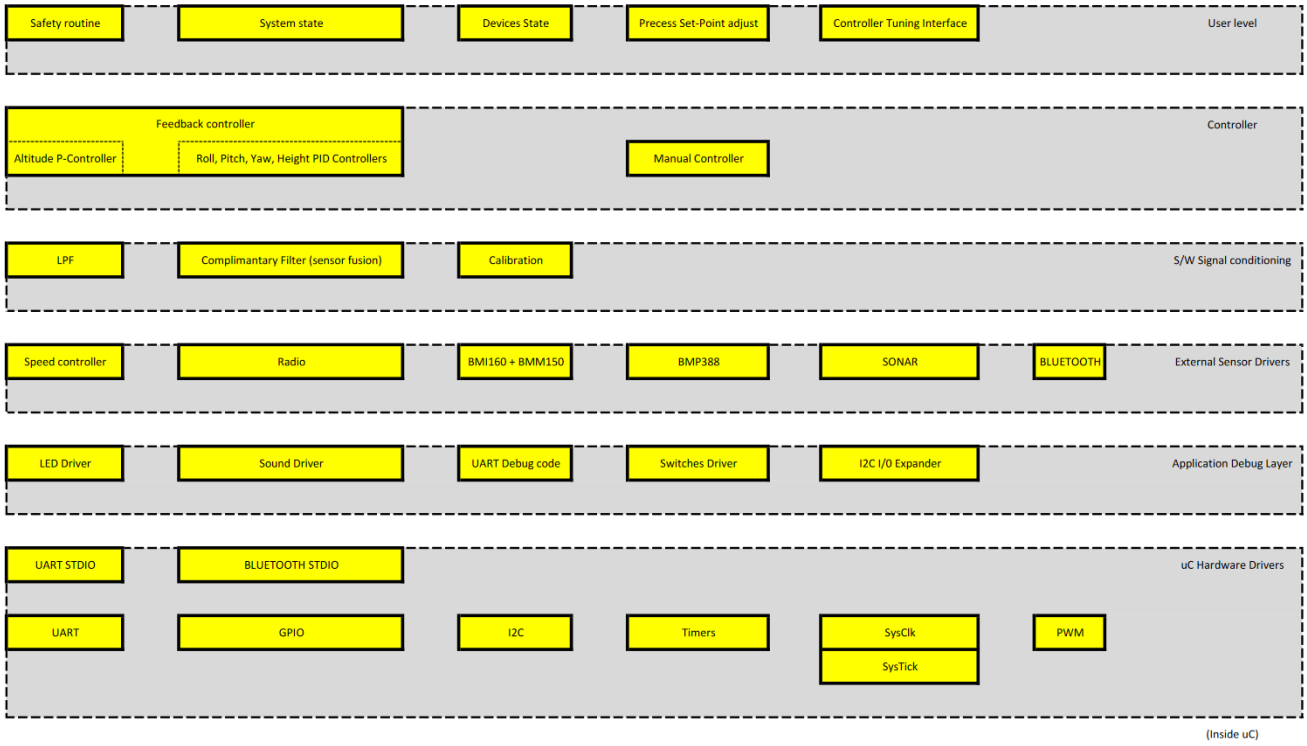


Figure 5-17 Software stack developed for this system

5.7 The complete system

The schematic of overall system developed for this project is shown below:

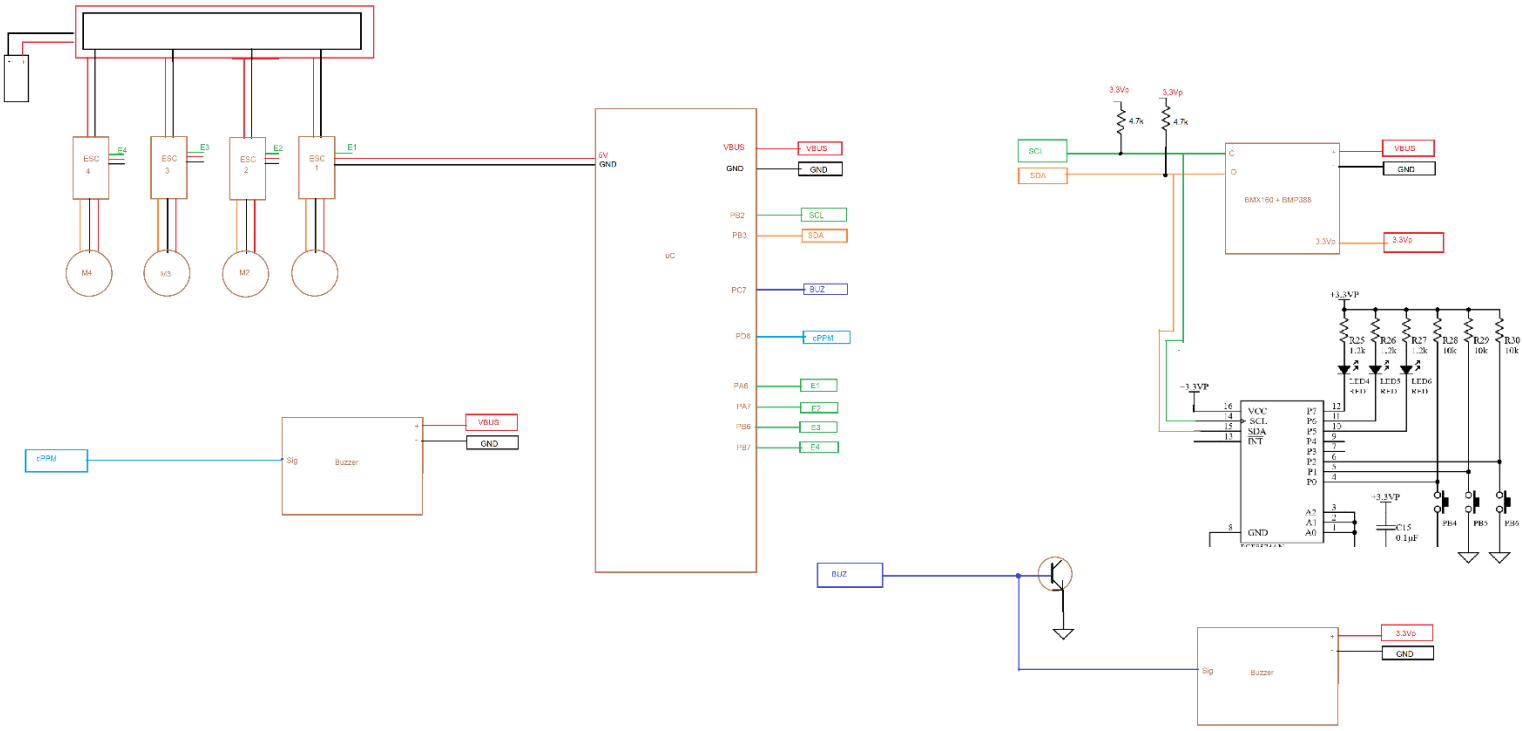


Figure 5-18 Schematic of overall system

The following table shows how each device is connected to the microcontroller:

Device	Pin (From)	Pin (to)	Color
I2C Module 0	SDA	PB3	
	SCL	PB2	
UART Module 0	RX	PA0 (via USB dbg.)	
	TX	PA1 (via USB dbg.)	
BMX160	VCC	3.3 V	
	GND	GND	
	SCL	PB2 (I2C0-SCL)	
	SDA	PB3 (I2C0-SDA)	
	CSB	3.3V	
	INT 1	N/A	
	INT 2	N/A	
BMP388	SDA	PB3 (I2C0-SDA)	
	SCL	PB2 (I2C0-SCL)	
	GND	GND	
	VCC	5 V	
	CS	3.3V	
	SDO	GND	
	INT	PC6 (GPIO)	
Buzzer	Sig	3904(Base) - > PC7 (GPIO)	
	Vcc	3.3V	
	(-)	GND	

		PWM		PPM	
ESCs	Motor 1	PA6		PE2	
	Motor 2	PA7		PE3	
	Motor 3	PB6		PA2	
	Motor 4	PB7		PA3	
Orange RX	Signal (PPM)	PD6			
	+	5V			

	-	GND	
Bluetooth	Vcc	5V	
	GND	GND	
	TXD	PE4 (UART05 RX)	TX
	RXD	PE5 (UART05 TX)	RX

Table 5-5 Pin mapping of each external devices with microcontroller

The images of entire system during development, after assembling and mounting on the testing area is shown below:

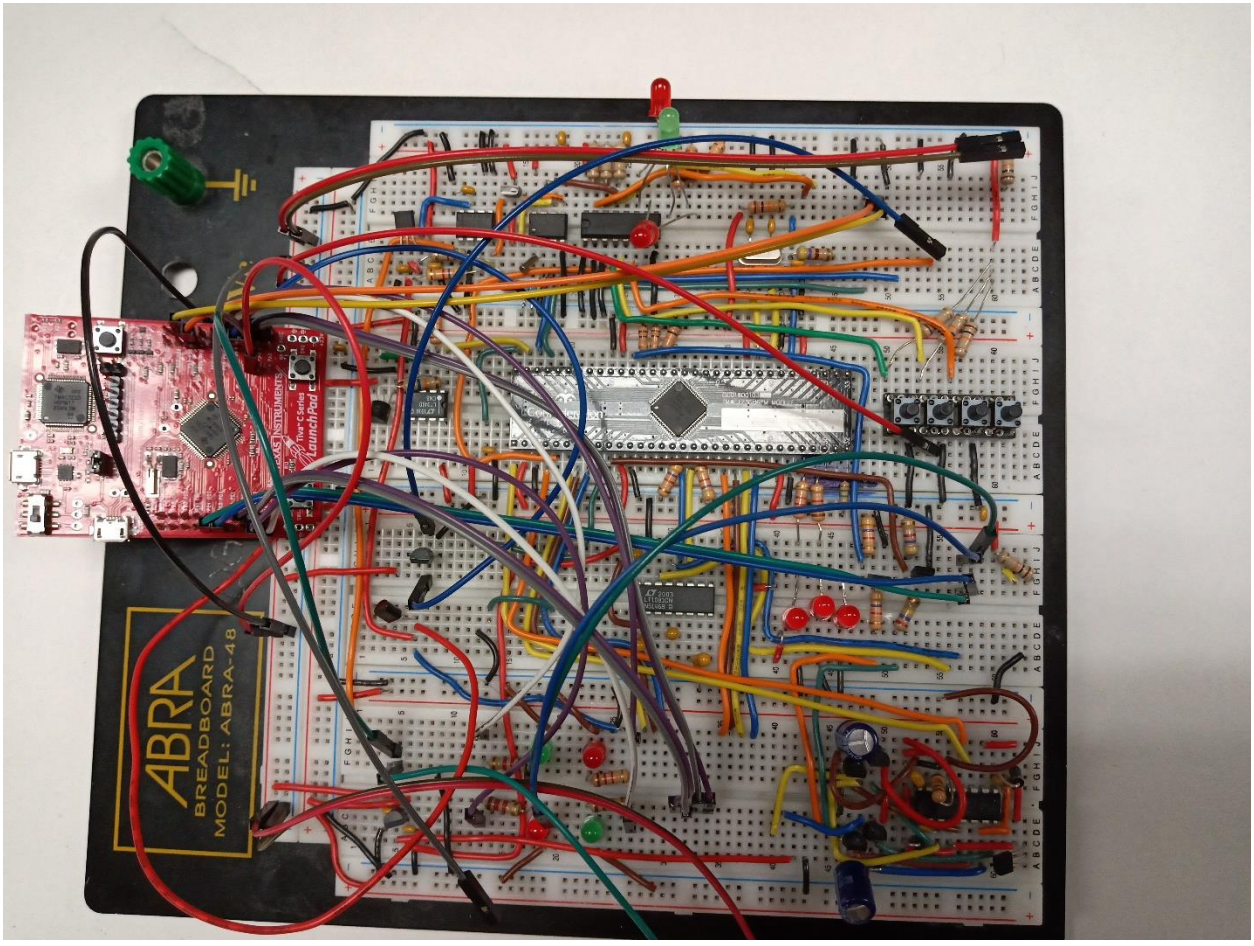


Figure 5-19 Entire system during development phase of the project

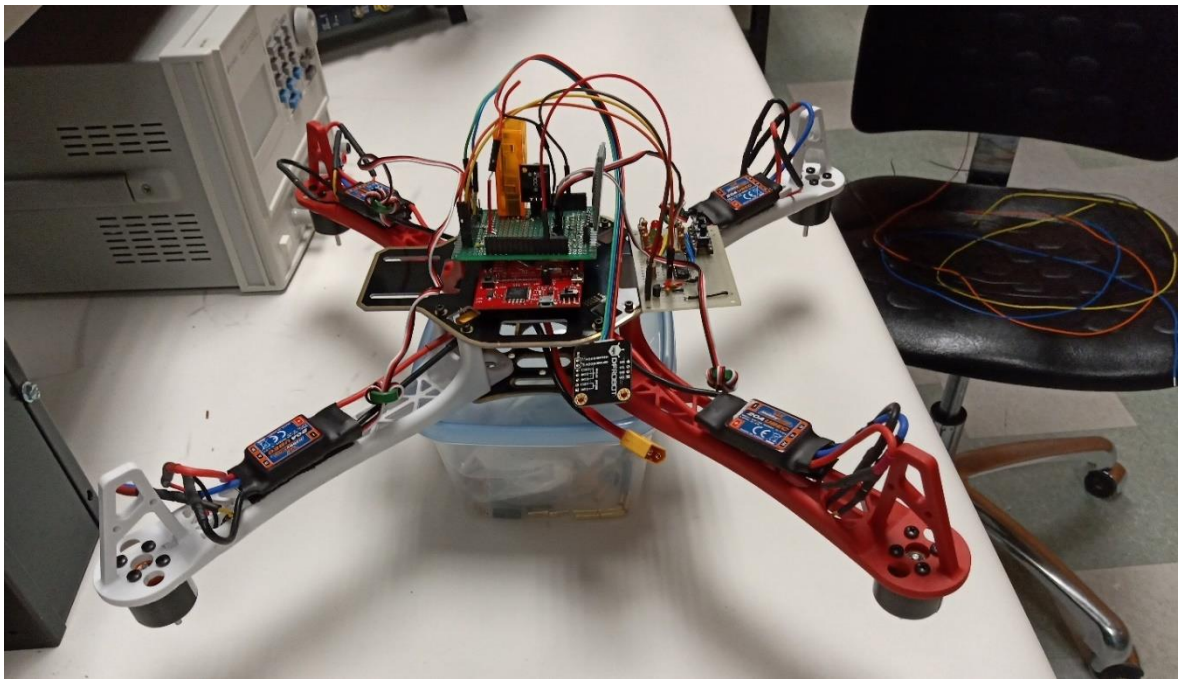


Figure 5-20 Entire system after assembling the system

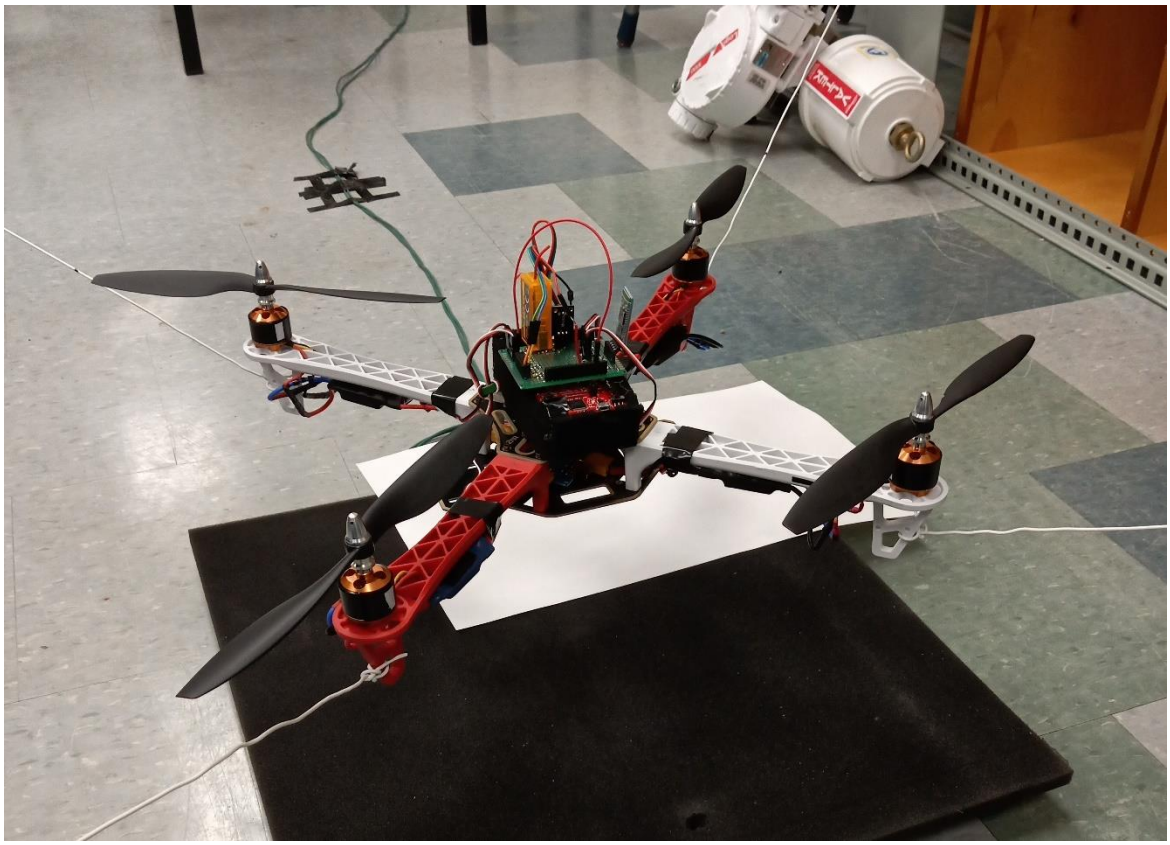


Figure 5-21 Entire system during testing phase of the project

6.0 Challenges

There were many factors to consider even before starting developing the system due to its complexity and size. Since the safety of the user, environment, and the system itself was given highest priority, considerable amount of time and thought went into the process of implementing the sub-systems that ensured the safety aspects of this project.

However, safety was not the only challenge, and this section aims to elaborate on some other challenges that were encountered when trying to bring this system to life.

Following list summarizes the types of challenges encountered during researching, developing, assembling, and testing phases of this project:

- Selection of components
- Software challenges
- Tuning challenges
- “Putting it all together” challenges
- Challenges caused due to noise
- PCB design challenges
- Debugging challenges
- Testing challenges
- Safety challenges

The sub-sections that follow describe each type of challenges in much greater details.

6.1 Selection of components

Much consideration was given on which components were best suited for this project. This subsection focuses on what was the thought process of selecting each type of components used in this project.

The following list summarizes the criteria of all components that were needed to be satisfied before purchasing:

- Easily available
- Within budget
- Small enough to be stable
- Large enough to reduce prototype time
- Thrust to weight calculations

6.1.1 Body component selection

The quadcopter body parts were not chosen randomly. Following selection path was used which ensured that the various components selected will work in harmony with each other:

- Frame
- Propeller size
- Motor stator size, Motor KV, Motor Type, Motor power ratings
- ESC size & power requirements
- Battery
- Electrical System
- Sensors, receivers, transmitter, onboard system state indicators, Wireless communication modules

This path was chosen because manufacturers often address the most suitable next component in the above selection path. For example, the manufacturer of the body frame suggests using either 1042, 1045 or 1047 propellers for this frame. Similarly, the manufacturer of the body frame also describes the type of motor that provides enough thrust to lift the overall system. The manufacturer of the motors also suggests the type of propellers that will fit on the motor shaft, and so on. Therefore, following the selection path was helpful since it eliminated many choices of different components and made the overall process of researching components easier.

6.1.2 Frame

The frame acts like a skeleton of the drone, which holds everything together. The requirements of frame were its size, easy availability, and its weight. Section 5 discussed the frames elected for this project and thus the material won't be repeated here. The following image shows the frame selected for project.

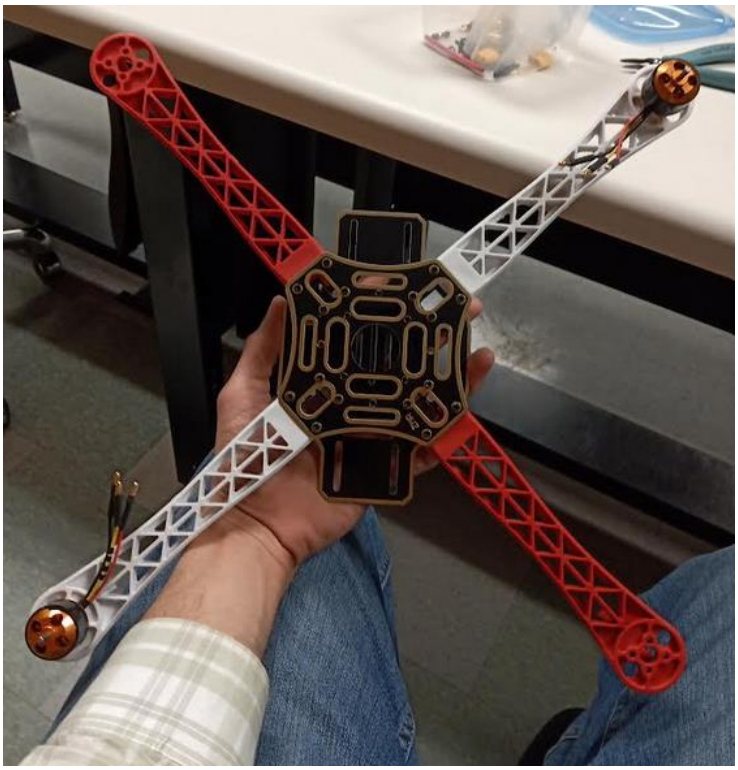


Figure 6-1 Frame selected for project

6.1.3 Propellers

There are two types of propellers. Those which produce lift force when turned clockwise and those which produce lift force when turned counterclockwise.

For quadcopters, there are four motors. Each motor spinning direction needs to be adjusted such that when the corresponding type of propeller is mounted, it creates the thrust as well as it counters the drag force created by the adjacent motors.

For this system, 1045 propeller type was chosen. These propellers were recommended by both the motor manufacturer as well as the frame manufacturer. These also come with various O-rings, that allows them to mount on various motors. An image showing which type of propeller is connected to which motor of the quadcopter is given below:

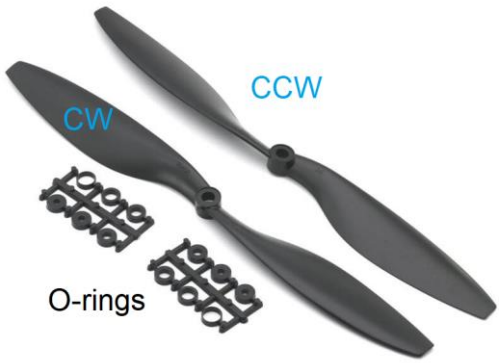


Figure 6-2 Different types of propellers

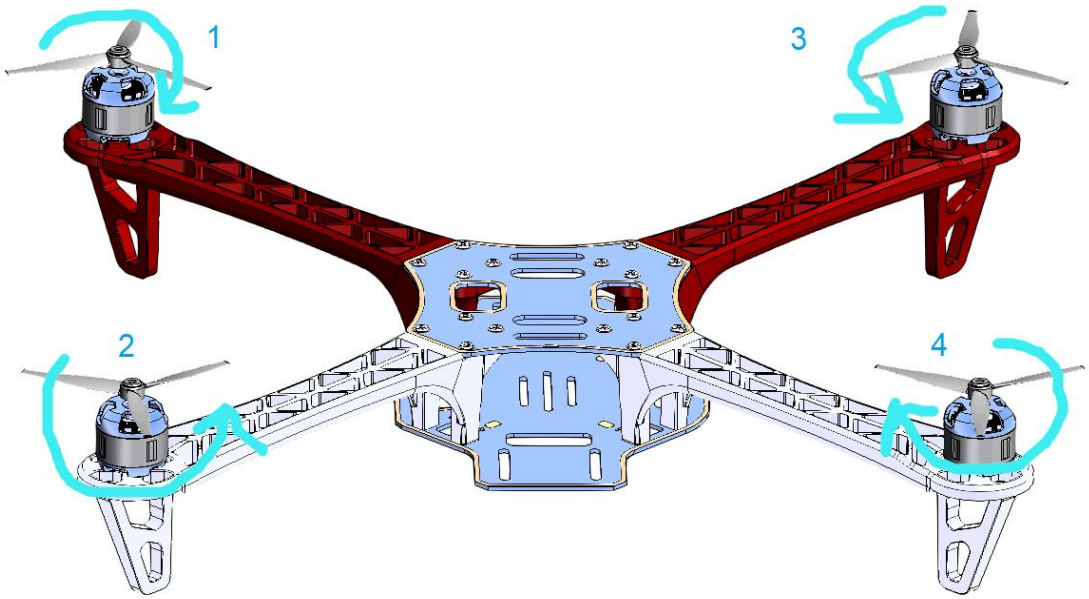


Figure 6-3 Direction of rotation of each motor

Note that the motor spin direction needs to be the same as the propeller's orientation. A motor spinning clockwise should only have a clockwise propeller attached do it and vice versa

6.1.4 Motors

There are several types of motors, but for discussing this project, the discussion that follows only describes Brushless DC motors.

Motors are manufactured differently for different purposes, even for quadcopters. There are many different types of quadcopters in the market, ranging from small racing drones to heavy cargo drones. The requirements of motors for different systems are therefore very different.

Brushless DC motors are mainly categorized into two categories:

I. Inrunner motors
II. Outrunner motors

Inrunner motors are suited for small form factor drones which are meant for agility and speed. *Outrunner motors* are suited when there are heavy loads, and the system needs considerable torque than the torque generated by inrunner motors. Therefore, inrunner motors are considered to create more speedy systems, and outrunner motors are suited for applications where high torque is desired over speed.

Therefore, the motors selected for this system were outrunner motors since the system was too big for inrunner motors to carry.

The motors selected for this project were A2212 1000 KV outrunner motors. These motors spin at the rate of approx. 1000 RPM per Volt. However, when propellers are mounted on the motors, these motors draw large amount of current and are suitable for high power applications. Also, from motor datasheet and the test data made available by manufacturer, each motor can generate about 700 grams of thrust at 11.1 V and when 1045 propeller is attached to the shaft.

The following images shows the images of the motor used in this project as well as the test data from manufacturer.

Figure 6-4 Type of motor used for this project [13]

Motor	Type	kv	Batt	Prop
XXD	**2212**	**1000**	**3S 5200 30C**	**1045**
V	A	W	Thrust (gr)	g/W
12.45	2.00	24.90	**200**	8.03
12.42	2.85	35.40	**250**	7.06
12.39	3.38	41.88	**290**	6.92
12.34	4.39	54.17	**360**	6.65
12.27	6.58	80.74	**465**	5.76
12.23	7.61	93.07	**545**	5.86
12.15	10.01	121.62	**650**	5.34
12.05	12.53	150.99	**750**	4.97
11.98	14.76	176.82	**825**	4.67

Figure 6-5 Motor thrust data provided by manufacturer [13]

6.1.4.1 Thrust to weight calculation

The weight of overall system with battery included was estimated to be approx. 1300 grams.

The thrust produced by one motor, as per motor test data was approx. 700 grams. Thus, total thrust produced by four motors = 2800 grams.

Therefore, the thrust to weight ratio of the system = $\frac{2800}{1300}$ = 2.15

A thrust to weight ratio of 2.15:1 is enough for providing lift force such that the system hovers at about half throttle value. For highly agile systems, thrust to weight ratio ranges anywhere from 4:1 to 7:1[19]. But for this system, a ratio of 2.15:1 is desired since the system is large, and agility is not a primary requirement.

6.1.4.2 Stator size and mounting mechanism

Another challenge for selecting proper motor was its stator size and mounting mechanism. These factors are very important, which ensures that the motor will mount on the frame securely. One issue encountered during mounting of the motors was the screw length was smaller than expected. The screws that were came along with motors were M3x5, however, when mounting on the frame selected, M3x8 mounting screws

were recommended. This issue was resolved by ordering the required screws from other manufacturer.

The following image shows the drawing of the motor from the manufacturer. The left image is a top view, the middle image is the side vide, and the right image is the bottom vies of the motor. Notice that the mounting screws are on the bottom side of the motor.

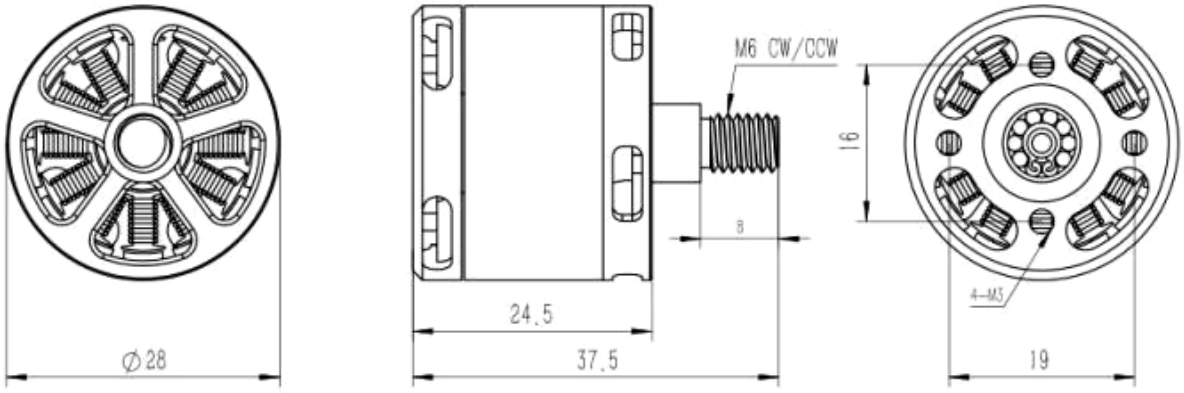


Figure 6-6 Physical dimensions of motors used for this project [13]

Another challenge in selecting the motor was the controls aspect of the motors. Therefore, a way to control the speed of each motor was needed to ensure the correct operation of quadcopter.

6.1.5 Electronic Speed Controller

A BLDC motor speed controller senses the small back emf voltage generated across BLDC motor phase wires and then decides which phase to ground, which phase should be connected to live wire and which phase will be disconnected from the circuit. The switching of phases needs to be synchronized and is done using high power Field Effect Transistors (FETs).

ESCs also employ a microcontroller within them that has a firmware inside it which takes the signal from flight-controller as its input and switched the FETs at a rate determined by the signal from the flight-controller. The firmware inside ESCs needs to be programmed before the system is powered the first time to make sure that the firmware is aware of the system it will be used for controlling.

Following image shows the programmer of ESCs that was used to program each ESC used in this project. The programming of ESC is included in Appendix B.

Figure 6-7 Electronic Speed Controller programmer [16]

6.1.5.1 Challenges with ESCs

ESCs essentially switch large amount of current and are rated for high power applications. Because of the amount of current they normally switch, they can easily catch fire if there is a short circuit in the circuit. For this project, to make sure that there can be no short-circuit outside of ESC circuits, the terminals were attached to appropriate connectors and the conductive ends were insulated using heat shrinks. This insulated layer took care of accidental phase-to-phase faults. Also, the connectors were designed such that the user cannot accidently connect ESC with Power Distribution Board in wrong polarity.

Because of high power requirements, ESCs were most prone to failure. During testing of the ESCs and motors, one ESC caught fire due to phase-to-phase fault across motor connectors. This short circuit caused excessive current to be drawn from the battery and as a result, the ESC caught fire due to excessive heat and failed FET.

The following image shows the failed ESC that caught fire during testing.

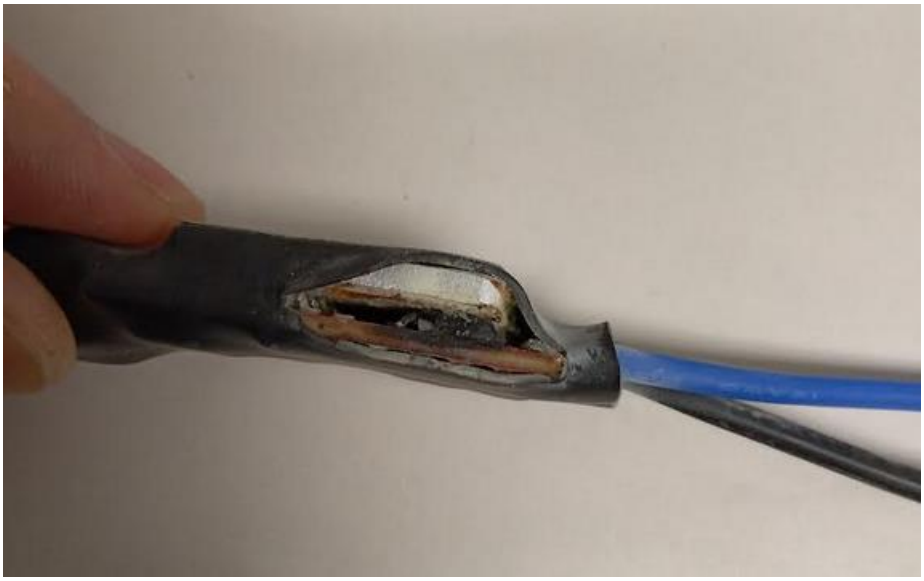


Figure 6-8 Burnt ESC

Another incident of failed ESC occurred during final testing phase of the entire system. This time, the ESC did not catch fire, but just stopped working. The following image shows the ESC that looks just like a normal ESC, but it stops working after indefinite time when connected with flight-controller.

Figure 6-9 Failed ESC

6.1.5.2 Extra circuits within ESC

The ESC selected for this system has an additional circuitry that provides power to additional on-board sensors and electronic devices. This was good because one ESC can provide 3A/5V, which is enough power to the microcontroller and attached sensors and communication devices used for this project.

This additional circuitry is called *Battery Elimination circuit* (BEC). The following schematic shows how one ESC can power the microcontroller. The image was shown previously in section 5.5.4 but is shown here again for easy reference.

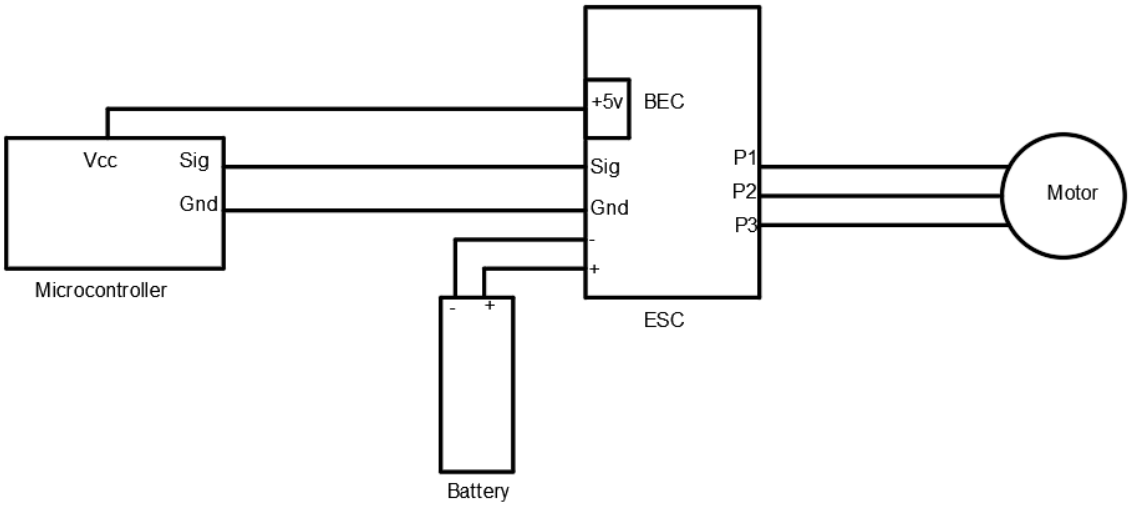


Figure 6-10 Connection diagram of an ESC with battery, motor and host Microcontroller [14]

Since each ESC has its own BEC, the challenge was to isolate the +5V supply from each BEC, but to connect the ground of each BEC with common system ground because ESCs need a reference voltage to act on signals coming from microcontroller.

Therefore, the challenge was to design a PCB that provides isolation for four BECs but is modular such that the user can decide which one of the four BEC to use to power the system.

6.1.5.3 Signals synthesis

Another issue with ESCs, that was realised after several months in development, was that the ESCs used for this project did not work with PWM signals. Instead of that, they understood servo control signals. All the signalling was done using PWM at that point, and the challenge was to recreate the servo signals for each ESCs and disable PWM modules if required.

The synthesis of servo control signals was done using General Purpose Timers (GPTM) peripherals. This is where the layered software stack helped immensely since all it was

needed to be done was to change lower-level code without modifying higher API functions. This allowed the regeneration of the signals that reassembles the servo control signals.

The following images shows the signals that were sent to ESC when PWM was used, and the servo control signals that were recreated and sent to the ESC when the lower-level code was modified to accommodate the new configuration.

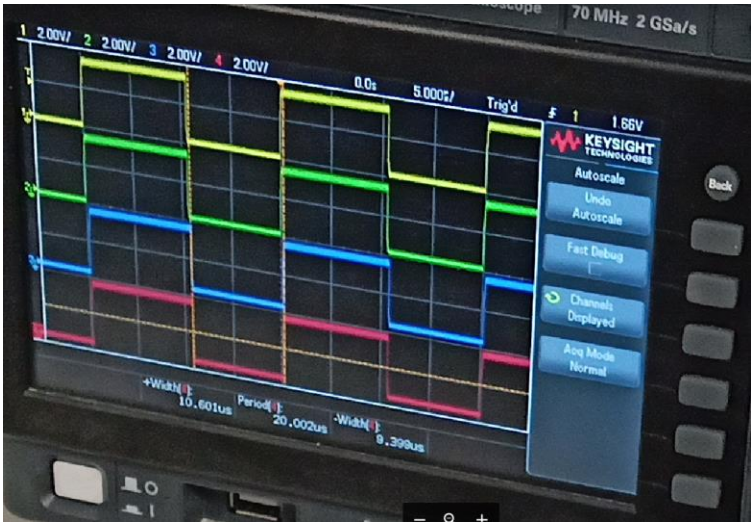


Figure 6-11 PWM signals representing motor speed

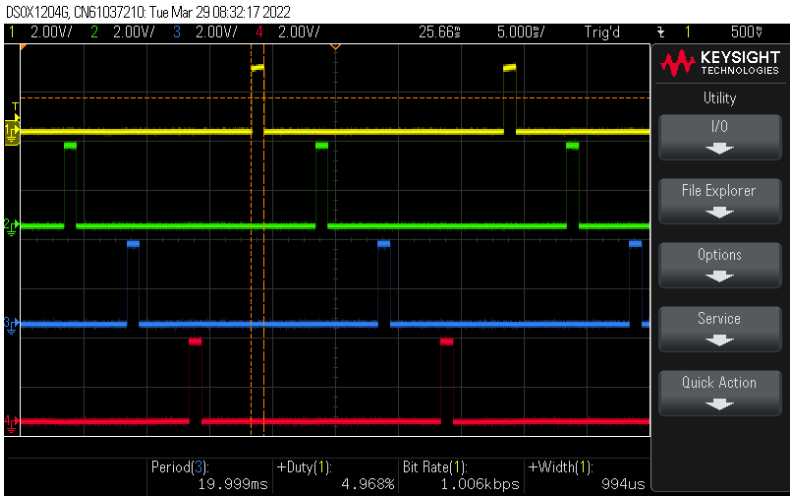


Figure 6-12 Servo control signals sent to each ESC

6.3 Software challenges

Building a system of this size was undoubtedly a challenge by itself. This sub-section discusses some of the challenges encountered while developing the software for this system.

Significant amount of effort was required to keep track of the development of this project. For tracking progress and managing the project files, a version control system called `GIT` was used. This made it easier to track the project files as well as manage multiple versions (branches) of the project.

A simple workflow was established and followed whenever a new functionality of the system was developed and tested. The workflow is graphically demonstrated below.

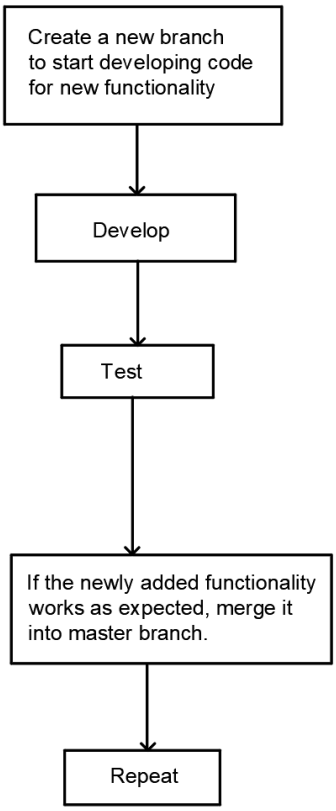


Figure 6-13 Workflow for developing individual functionality of the system

Another challenge encountered during development of control system was the programming of *multi-input multi-output* control system with multiple point of failures and safety criteria using fully interrupt driven approach.

While working with multiple sensors, it was a challenge to create multiple feedback loops from system attitude sensors (IMUs and pressure sensor) and make them work without causing any signal interference. Also, the timing intervals of reading new values from each sensor were different and they needed to be tracked to understand how system was behaving in real world.

Another challenge was to create a sub-system that monitors and reports the state of entire system back to user. This was done by keeping track of multiple variables and structures and displaying them whenever user requested. Similarly, an efficient debugging sub-system was needed for entire system to accelerate development time and minimize troubleshooting time. The section on debugging challenges follows this section.

The following diagram (on next page) depicts the logical structure of the software developed for this project and the hardware used in this project.

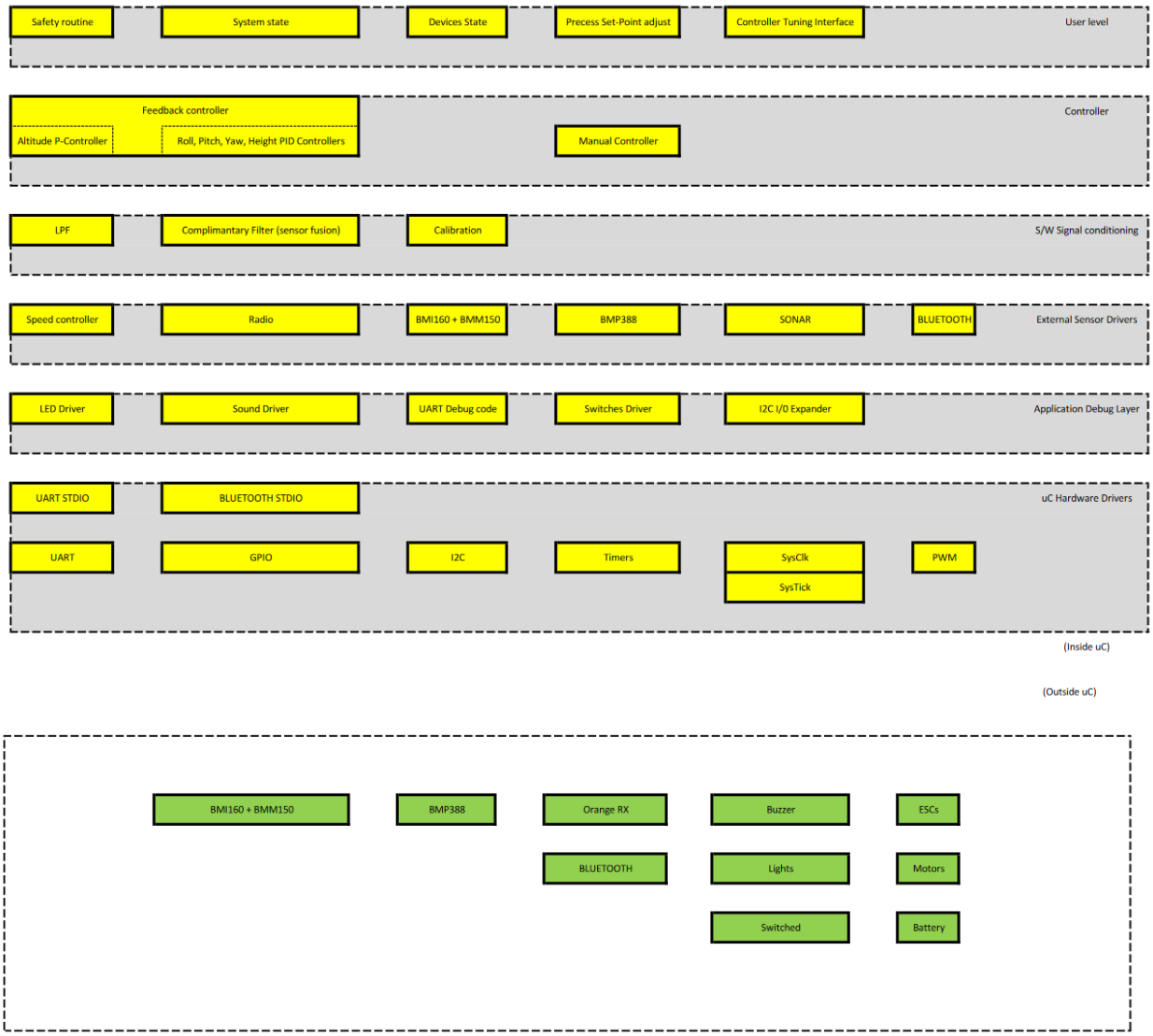


Figure 6-14 Entire software stack developed for this project

Apart from structuring, a need of accurate timing was also very important for this project. The challenge was to use accurate timing peripherals inside the microcontroller to create the accurate timing required for various peripherals and overall system. For this reason, SysTick timer peripheral was used as main software clock and General Purpose Timers were used in creating drivers for radio receivers and for synthesis of servo control signals for ESCs.

Additionally, the noise caused by environment and signal interference was a serious issue to deal with throughout the development of this project. External hardware was used to filter out noise, however there was still a need of creating a software routine that models a first-order low-pass filter. Following code listing is from the project code that implements a first order low-pass filter.

```
24 // Function definitions.
25 /*
26  *
27  *
28  * see http://techteach.no/simview/lowpass_filter/doc/filter_algorithm.pdf
29  *
30  * => New value will be stored in low_pass->prevOutput for next calculation.
31  * => New value will also be returned
32  * => input is the raw input of individual axis (e.g. : x-axis input of accelerometer.)
33  * => dt is the time-step (h).
34  */
35 float applyLPF(LOW_PASS_FILTER *low_pass, float input, float dt) {
36     const float tau = 1.0f/(2.0f*PI*low_pass->Fc); // Time constant
37     const float alpha = dt/(tau + dt);
38
39     // y(n) = y(n-1) + alpha*(u(n) - y(n-1))
40     float output = low_pass->prevOutput + alpha*(input - low_pass->prevOutput);
41     low_pass->prevOutput = output;
42     return output;
43 }
44
```

Figure 6-15 Implementation of Low-pass filter in software

Therefore, several other software challenges encountered throughout development phase of this project. However, they are not discussed in this report to keep the discussion very brief.

6.4 Debugging challenges

The system developed for this project relies on many peripherals internal to the microcontroller. Also, to ensure proper interfacing of external devices such as radio receiver or sensors, it was always required to have a reliable and easy to use debugging system that aided in finding code issues quickly and helped to verify certain aspects of the system, such as register values of a particular sensor.

However, as previously discussed in section 5, a systematic approach was needed to not only debug, but also to quickly modify code with few clicks.

The main challenge behind developing such system is the size and complexity of the software it will be used to debug. The debugging sub-system developed for this project needed to work with several aspects of the flight control system, some of which include:

- motor control
- communications
- data acquisition
- state machines
- calibration
- sensor configuration
- I2C
- Radio signal reception
- etc.

The development of the debugging system was done alongside with the main flight control system. Therefore, both the main flight control system and the debugging subsystem evolved together.

The initial approach in developing this debugging sub-system was to just use comments and polled UART stdio API from Tivaware library[3]. The issue with this debugging approach was that it was not scalable, and it was time consuming to code, modify, analyze, and keep track of comments. Additionally, the polled I/O functions meant extra system resource penalty for the main flight control system. This was obviously not desired.

As the flight control software started to expand in complexity, the debugging sub-system was also modified to use onboard LEDs and switched. Therefore, many support functions for controlling the LEDs were implemented. LEDs were used to indicate a particular issue by flashing in a predetermined pattern. An internal state machine was used for keeping track of number of flashes. Onboard switches were used to trigger a debug code when pressed. This was done to see system response and system state when the switches were pressed. Clearly, this was not the perfect solution since it

required attention of user to count number of flashes, and it was not very useful to just trigger a code at a press of an onboard switch.

A support for Buzzer was soon added to aid in the debugging sub-system. The idea was same as onboard LEDs flashing, however, this time, buzzer also buzzed along with LED flashes. This soon became annoying for other people working around. Therefore, a new approach was needed. Meanwhile, the UART stdio debug code was modified to use fully interrupt driven approach. This was a significant improvement since it enhanced system performance.

The usage of `#ifdef` preprocessor directive solved many debugging issues since it was quick to enable/disable certain code sections using this approach.

For motor controls, PWM modules internal to the microcontroller were used to display the signals that would be sent to ESC. Although ESCs would only understand servo control signals, enabling PWM module while debugging the main flight controllers was very useful to see the control actions of the flight controllers in oscilloscope.

Special software routines were written to debug radio signal reception. GPTM was used to determine the timing of signals received and get the channel data from those signals.

The following images shows how radio signals were detected and how useful information was extracted from the signals using this debugging sub-system.

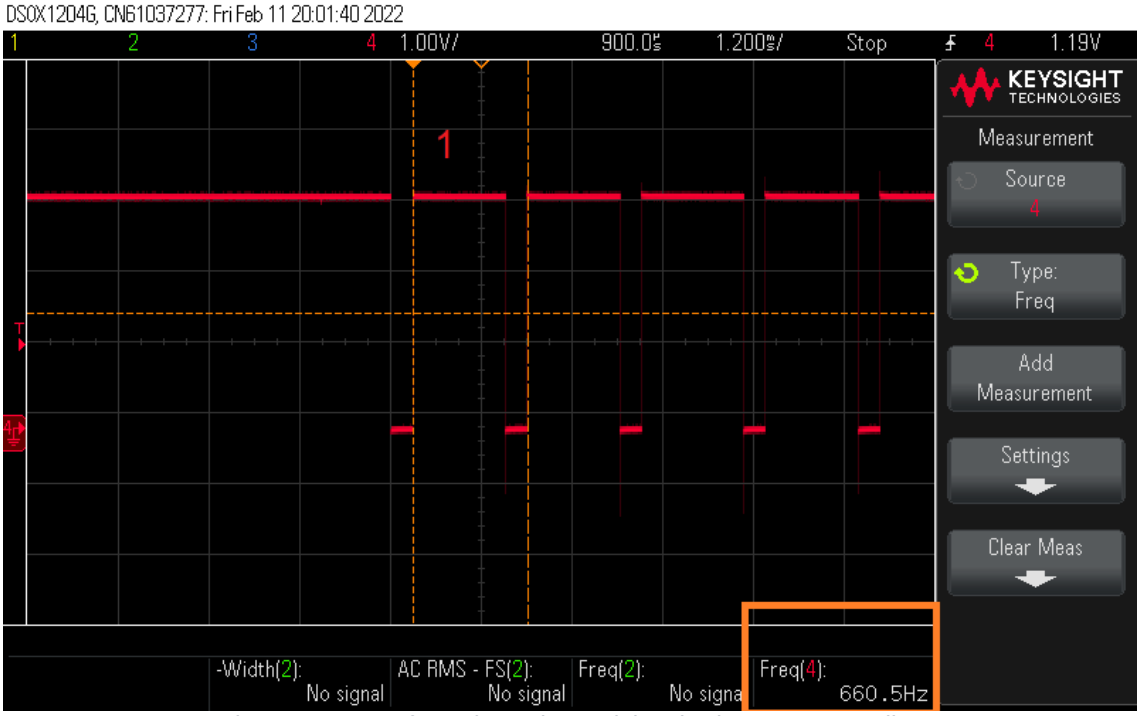


Figure 6-16 Data from channel 1 modulated using cPPM encoding

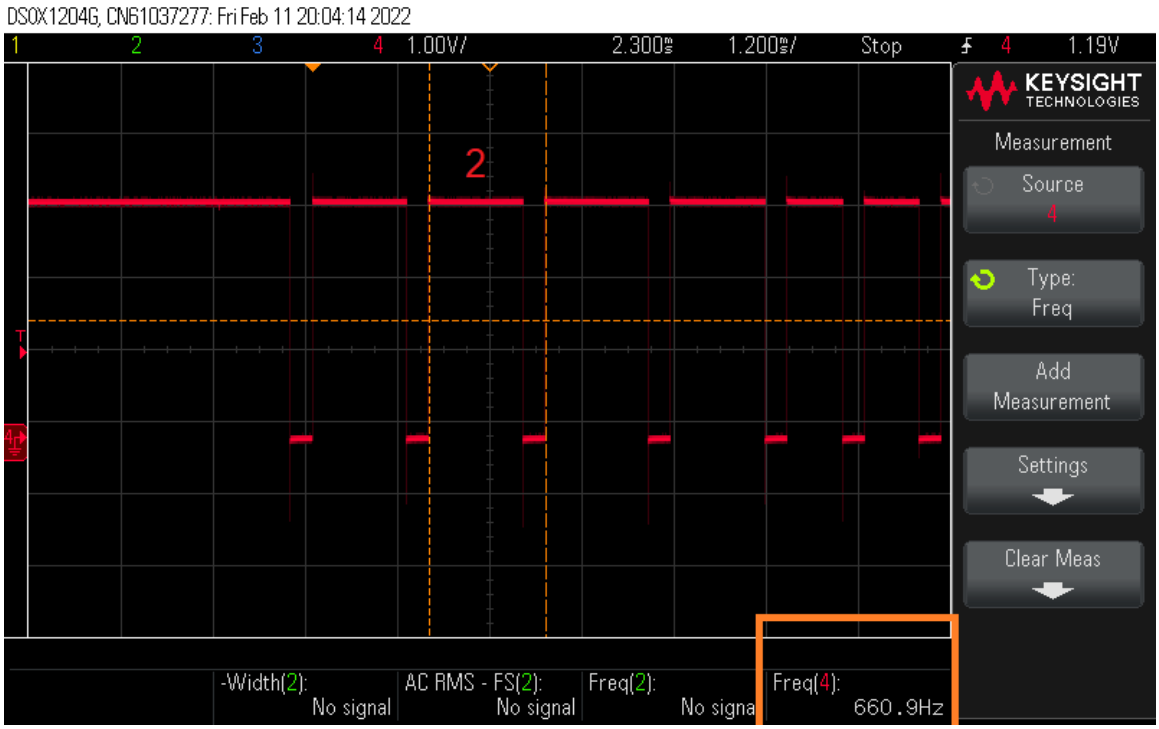


Figure 6-17 Data from channel 1=2 modulated using cPPM encoding

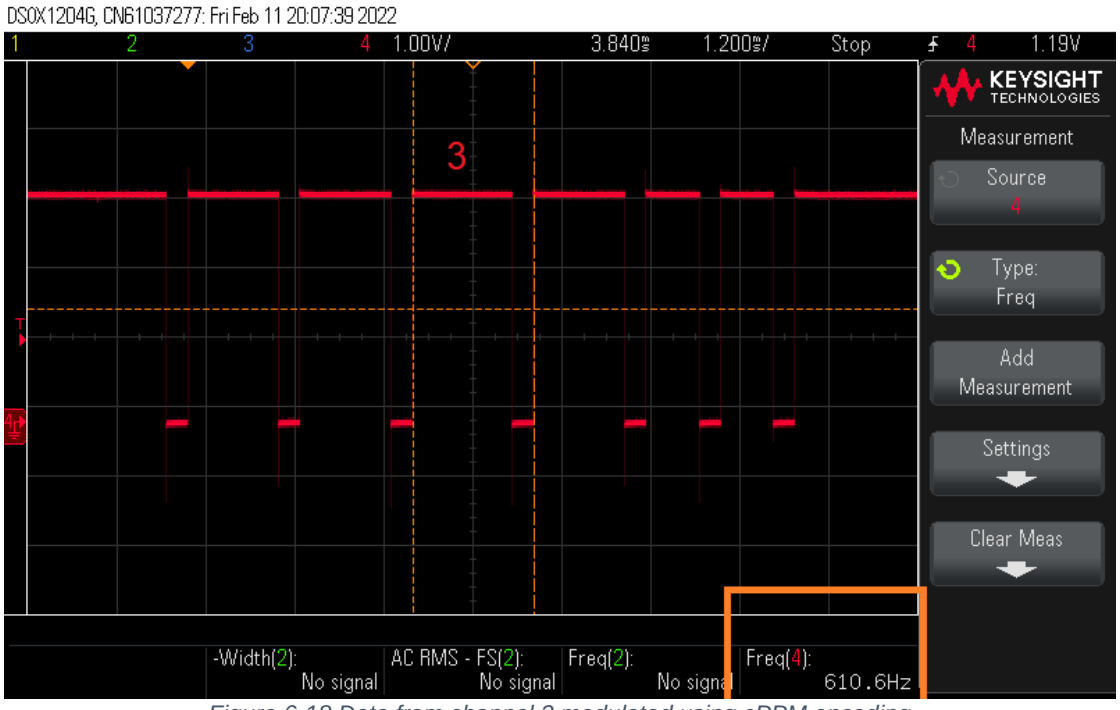


Figure 6-18 Data from channel 3 modulated using cPPM encoding

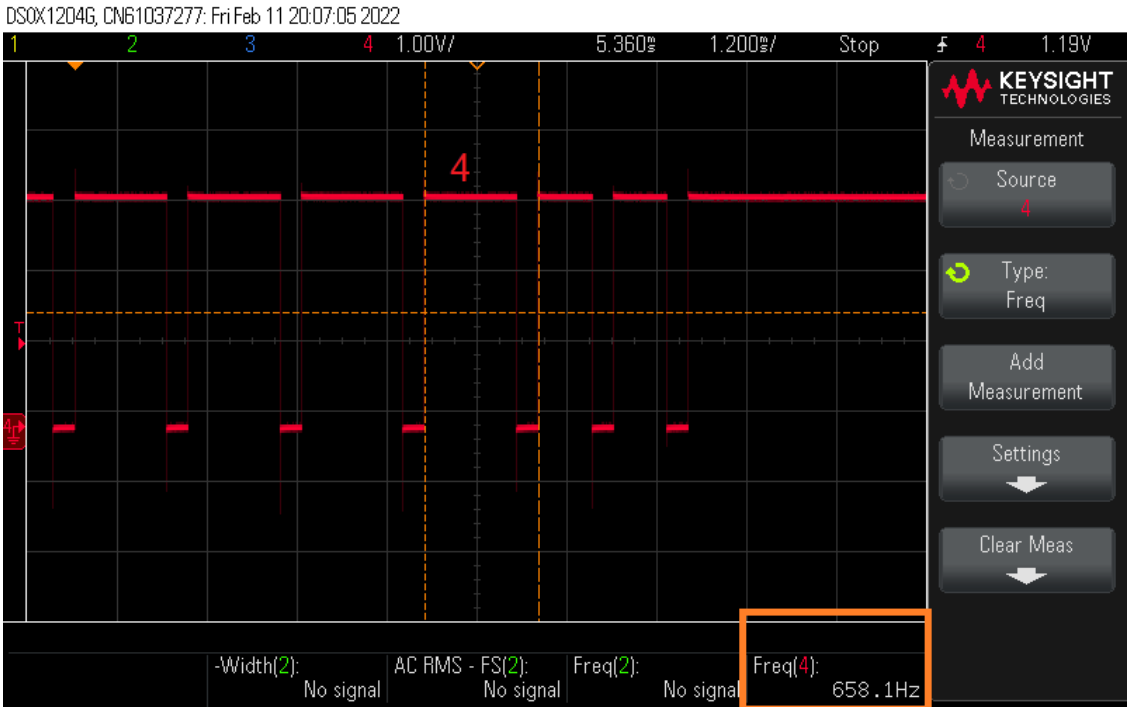


Figure 6-19 Data from channel 4 modulated using cPPM encoding

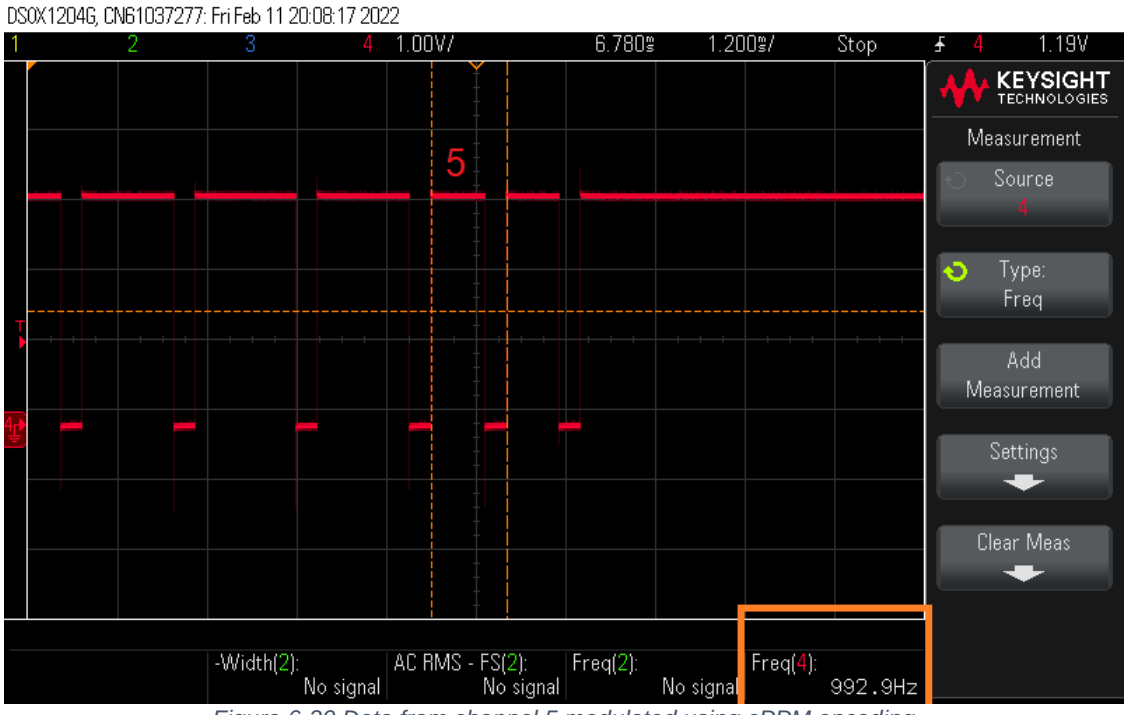


Figure 6-20 Data from channel 5 modulated using cPPM encoding

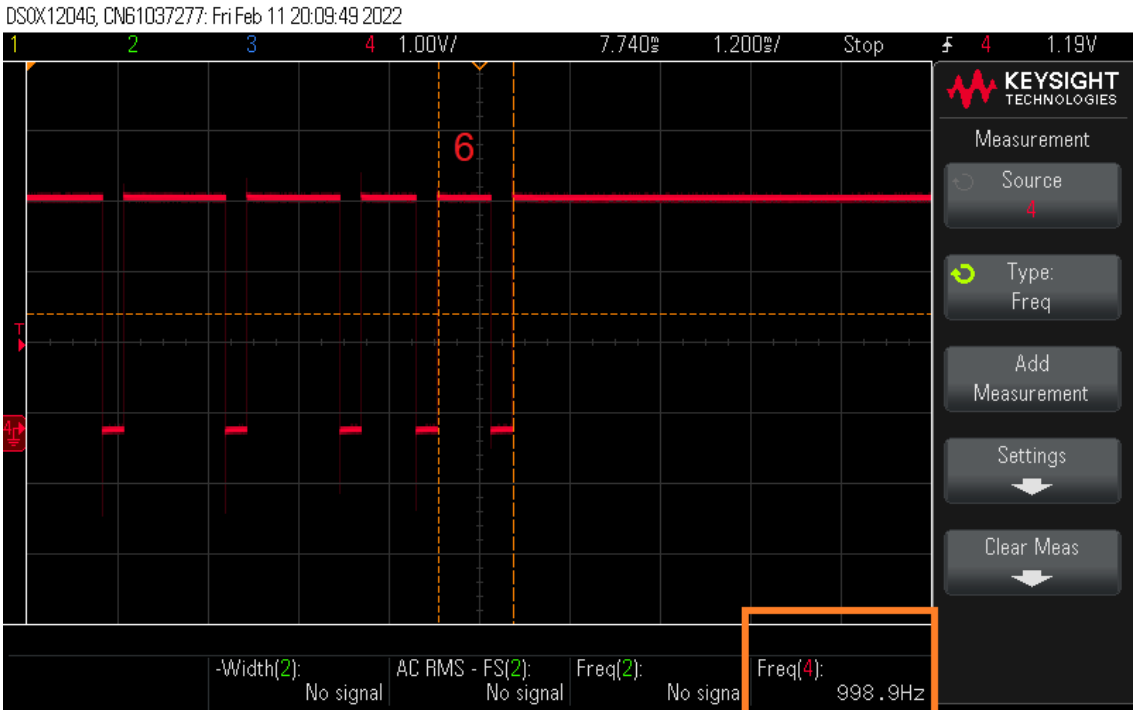


Figure 6-21 Data from channel 6 modulated using cPPM encoding

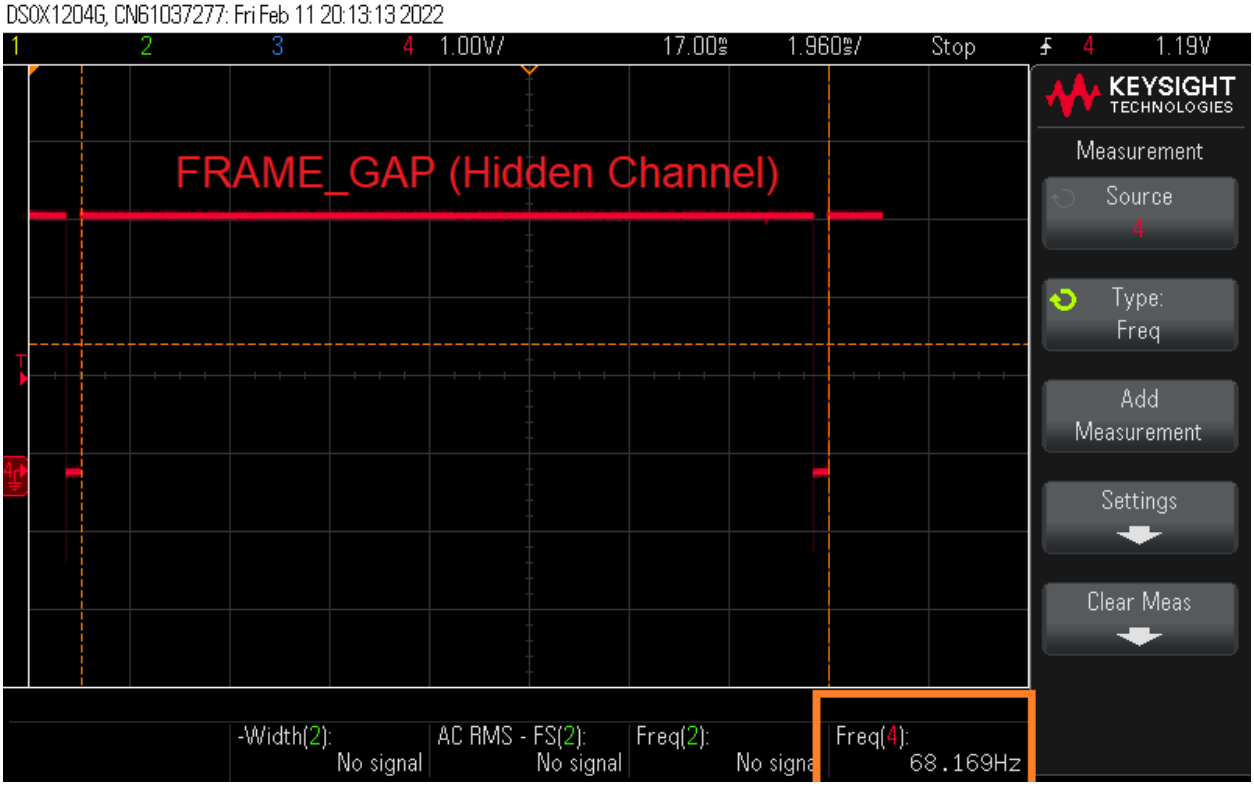


Figure 6-22 Hidden channel in the radio transmission

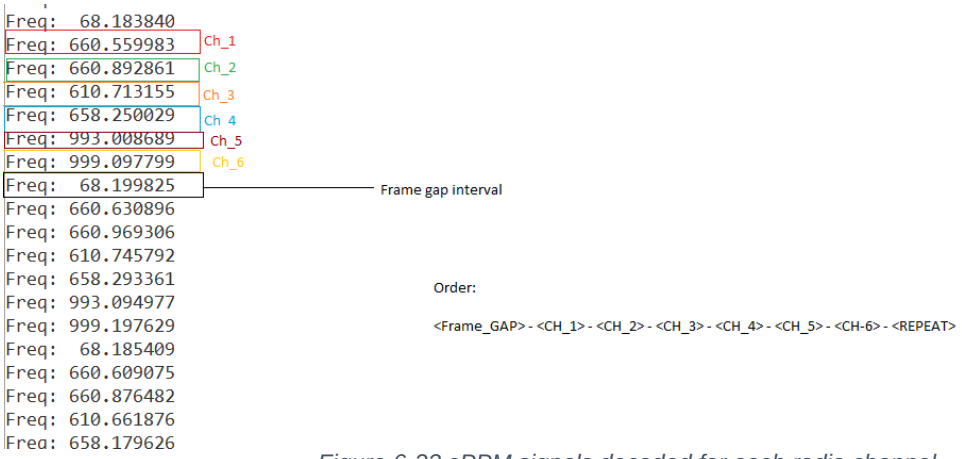


Figure 6-23 cPPM signals decoded for each radio channel

Additionally, a debug channel was added to aid in finding issues with I2C sensors and I2C bus. For this, a I/O Expander PCB was hand soldered, which allowed the interrogation of the I/O expander board through I2C bus. This helped a lot in finding any issues with I2C bus, or when the sensors using I2C were not working properly. The image of this special I2C debugging PCB created for this project is shown below.

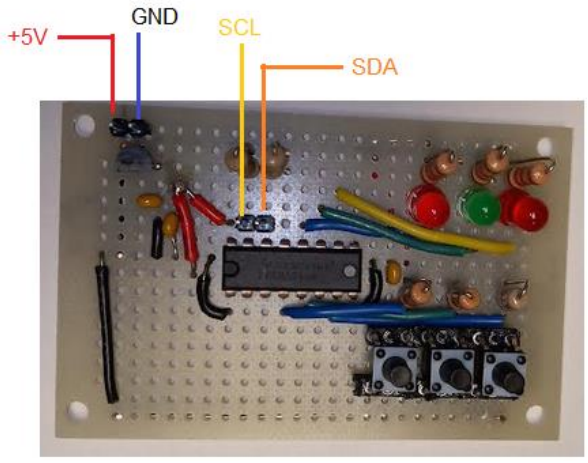


Figure 6-24 I/O expander I2C debugging PCB

The following list summarizes several debug channels used within the debug sub-system developed for this project:

- UART (Fully interrupt driven)
- I2C I/O Expander
- On board LEDs
- On board Switches
- Buzzer
- Bluetooth
- #ifdef preprocessor directive
- PWM module
- Oscilloscope and waveform generator synthesis (GPTM).

6.5 Tuning challenges

The tuning of controllers developed for this project was a very challenging task since the response of the system for each value of tuning parameters needed to be logged. Additionally, there were total of three P+I controllers developed for this project, each one corresponding to Pitch, Roll and Altitude of the system.

Because each controller can adjust multiple outputs, tuning one controller while the other was enabled was a very difficult issue to solve. Also, the method of tuning employed for this project needed to be easy and safe to use.

To be able to tune the system while the system was operating was a challenge. At first, a simple hardcoded solution was used. The programmer would first hardcode the tuning variables and then see the response of the system. Then the programmer would fine tune the system again and repeat the process until the desired system response is observed. This was very cumbersome and time-consuming tuning approach.

Soon, another tuning approach was developed where the programmer would connect a USB to the microcontroller and use the UART module connected to USB data line (UART module 0) to press certain keys in keyboard, which would increment or decrement the value of tuning variables. This was a reasonably good approach when there were no propellers attached and when the power to the system was supplied through the USB.

However, each of the above approaches were prone to failure when the system was operating with propellers attached and with no USB power. Therefore, an additional radio communication channel was added by developing drivers and support functions for UART-to-wireless Bluetooth module. This approach allowed the programmer to send and receive data from the flight controller while the system was operating with propellers as well as with direct supply from battery or external power supply.

The following image shows the console application specifically configured for this project, which can be used to tune P+I controllers or get information about system over Bluetooth radio channel.

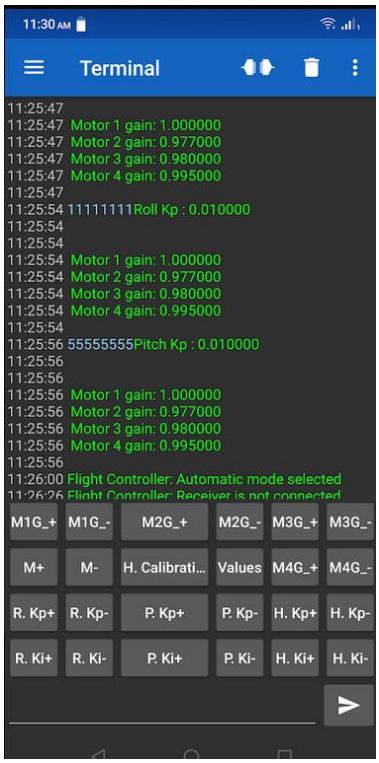


Figure 6-25 Mobile application configured to send/receive data to/from flight-controller. [23]

Additionally, the tuning of maximum angle for pitch, roll, yaw and maximum height for Manual controller is also done via Bluetooth.

Section 7 further discusses the communication channels developed for this project.

6.6 Printed Circuit Board (PCB) design challenges

The need of creating a PCB for this project was realised even before starting the development of the flight controller software. However, the PCB was not needed for creating the software, and therefore a more convenient approach was taken for rapid prototyping of the system and verify the operation of the entire system by using breadboard. The following image shows the complete system on the breadboard:

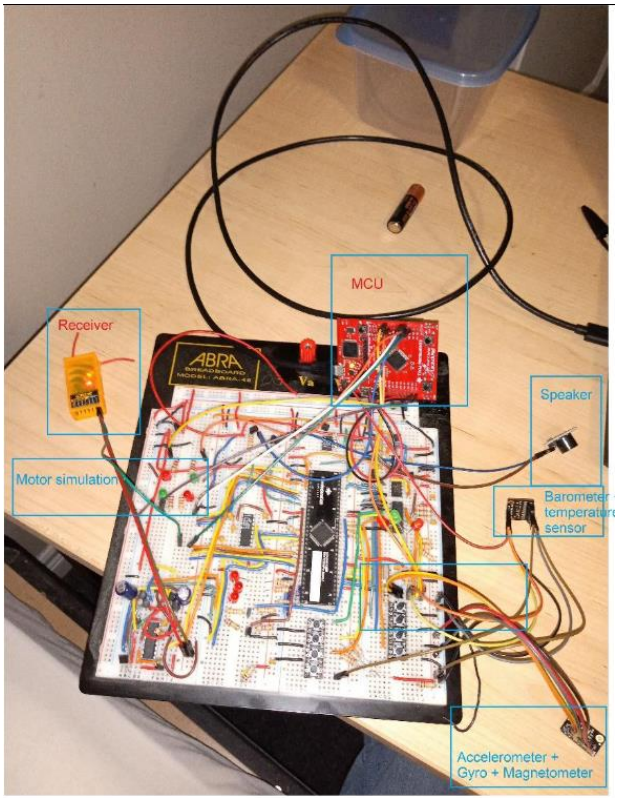


Figure 6-26 Complete system on the breadboard

When the system was working properly, the next step was to assemble the quadcopter frame and mount all the electronic devices on the frame. This was a challenge because the devices had to be connected with jumper cables and there were just too many cables to manage. This was also a safety concern since a wire could get into a way of the propellers and the electronic device may fly away from the system or crash the drone.

To counter this issue, a more modular system was needed to be developed such that the system can be modularized and is compact to a degree that there are no jumper wires in the system for any electronic devices.

Because of these requirements, a small prototype PCB was designed and developed in the lab. This PCB was designed such that the user just plugs the PCB board on top of EK-TM4C123GXL board, then plugs all the sensors, receivers and ESCs at predetermined sockets and headers and the whole system is ready to fly. This allowed a rapid installation of different electronics devices, eliminated all the jumper cables, made the system very compact and light-weight.

The images of the hand-soldered PCB are shown below:

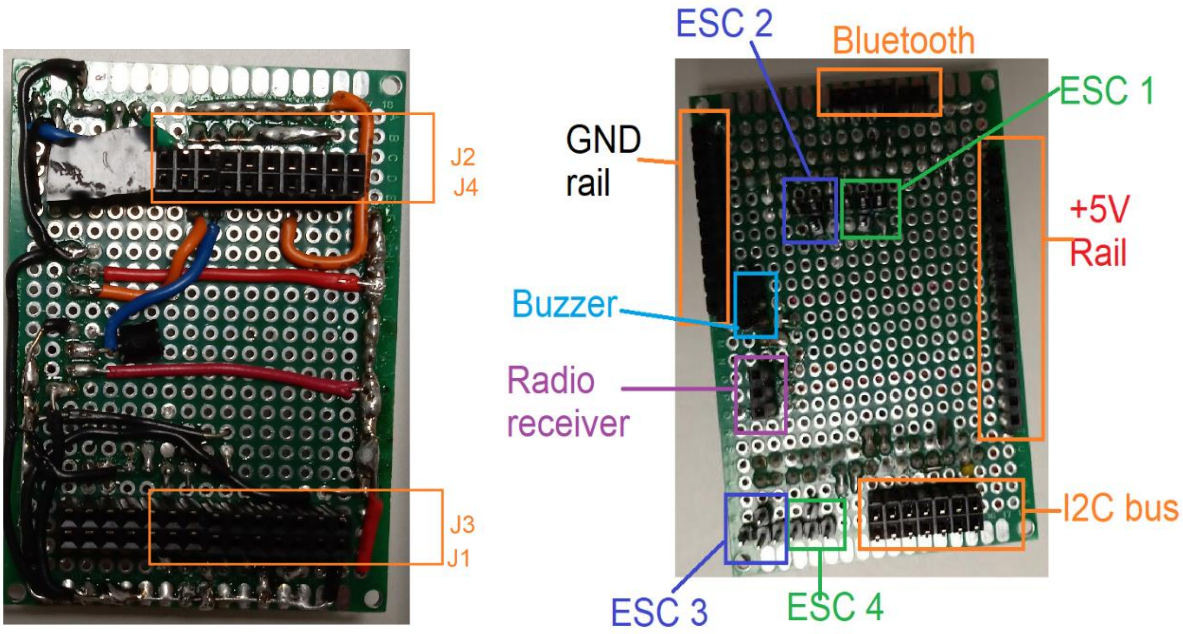


Figure 6-27 Main Flight-controller PCB created for this project

The next challenge was to create a computer design file for this prototype PCB. The reason for doing this was to have a backup for the PCB if the prototype PCB fails during flight. The time took to create the prototype PCB was approx. 14 hours in the lab, which clearly was a very time-consuming process. To ensure that there are backup PCBs available during the testing, and not spend additional time in debugging the hand-soldered PCB, the design files for prototype PCB needed to be created so that the milling machine can mill out the PCB in significantly less time.

The following image shows the image of the designed PCB in Altium software.

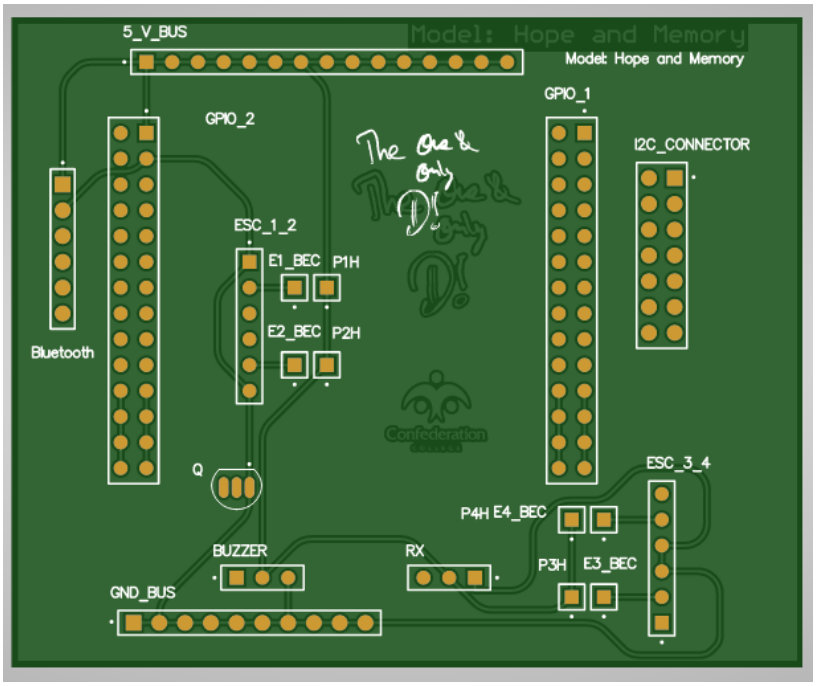


Figure 6-28 Flight-controller PCB designed in software (Top layer)

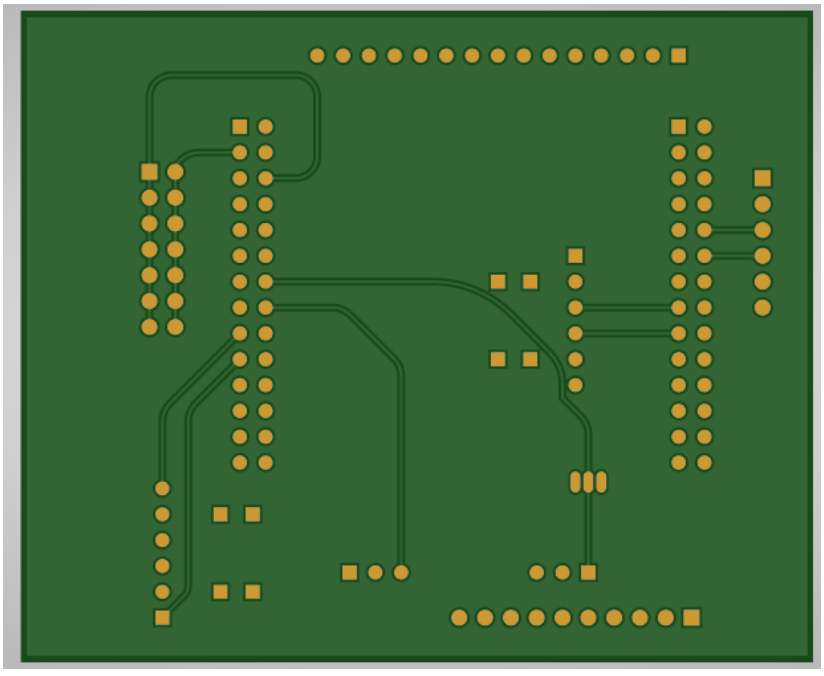


Figure 6-29 Flight-controller PCB designed in software (Bottom layer)

The next step was to mill out the PCBs using the milling machine. The following image shows the outcome of this process:

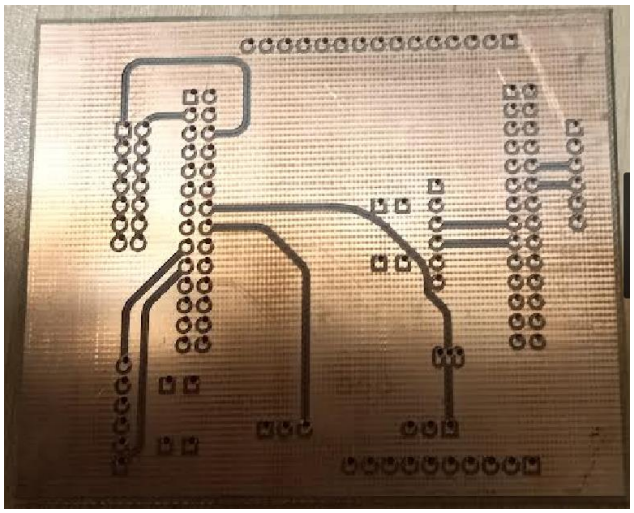


Figure 6-30 Milled PCB board (Bottom bayer)

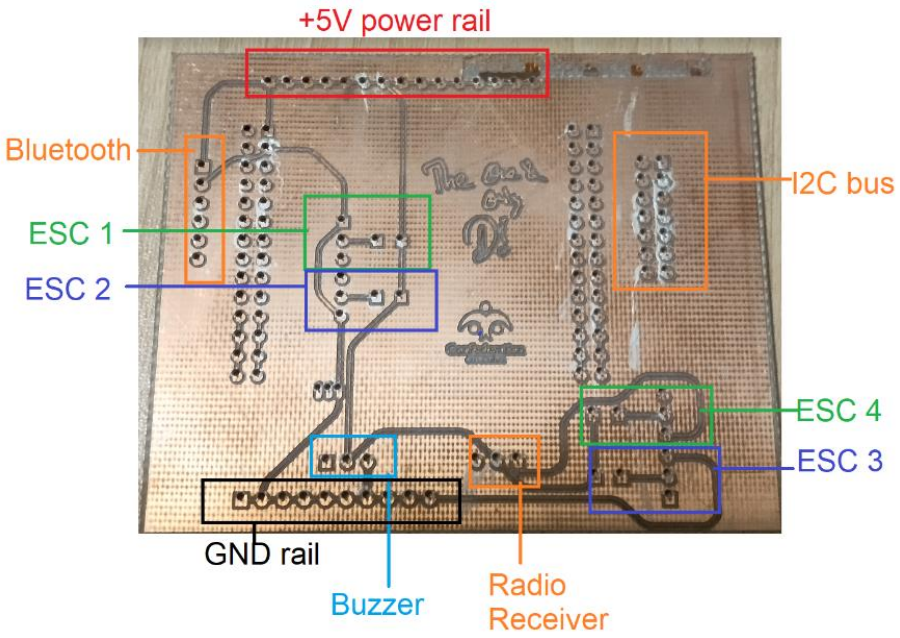


Figure 6-31 Milled PCB board (Top layer)

After milling out the board, it was found that the Z-axis of the milling machine had incorrect offset value, and thus it scratched the surface of the board when drilling the holes. When Proconduct paste was applied, it spread across the board, making the PCB useless for the application. Therefore, the task of milling out the PCB board using milling machine was not successful, and thus all the testing was done by installing the previously designed hand-soldered prototype PCB.

6.7 “Putting it all together” challenges

This sub-section focuses on the challenges and the problems encountered while trying to make everything work in harmony.

Before developing any supporting code for a device, all other unnecessary peripherals and devices were disabled first. Following image captures the approach that was undertaken for developing and testing the drivers of individual sensors, radio receivers, or any other internal peripherals.

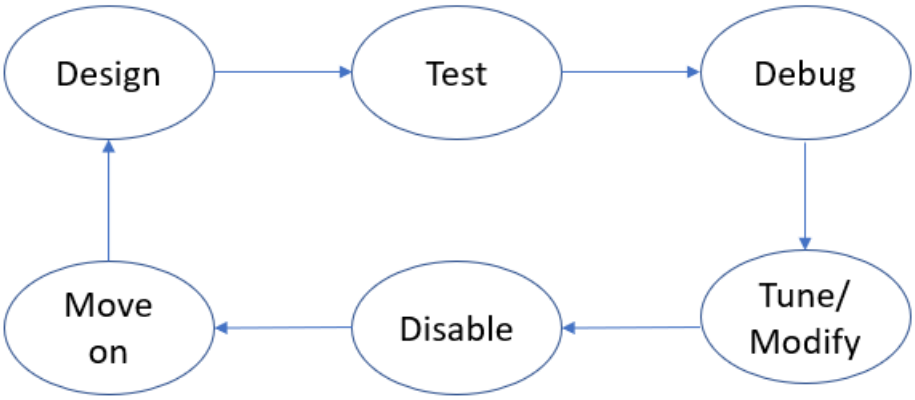


Figure 6-32 Workflow for developing and testing the drivers of individual sensors

Once every sensor, device and internal peripherals were working by themselves, the next major task was to enable them all together and see how the system behaved. The challenge was to make everything work in harmony before assembline the drone frame and mounting motors and ESCs.

When all the components were enabled, the system began to halt at random time intervals and all onboard sensors began to gave false readings. Also, the system began to halt due to I2C peripheral. At first, it appeared that the problem was caused due to buggy implementation of I2C read and write functions. However, everything appeared to be working well when only certain peripherals were enabled.

The troubleshooting of this issue was done by observing various signals from various sensors and devices on oscilloscope. When oscilloscope was used to see the effect of signal interference, it was noticed that there was a 50 kHz signal induced on the I2C bus. This was the frequency of the PWM peripheral, that was required to see how flight controllers adjusted different outputs signals. It was also observed that the power line had voltage drop due to radio receiver’s internal LED.

When the PWM and receiver module were disabled, everything started working well together. The next challenge was to enable PWM and radio receiver module without affecting any other device operation. After refering the datasheet of the microcontroller, it was found that I2C peripheral employs a filter block (Glitch filter), which can be enabled and configured using direct register access or using Tivaware API. Since this project used Tivaware for all low-level code, it was used to configure the I2C master Glitch-filter block and configure it to max setting of 32. Once this was done, entire system started working in harmony with no further false readings from sensors or random system halts.

Another interfacing issue that arose when interfacing everything to the flight controller was with Buzzer module. The symptom of the problem was that the Buzzer produced cracking noise when GPIO was used to send signal to buzzer. However, the Buzzer worked as expected when the signal line was connected directly to 3.3v power rail directly. After connecting potentiomenter to observe how buzzer behaved with varying voltage, it was found that the response of buzzer was non-linear. Therefore, the buzzer needed to be at either around 0.7 V or around 3.3 V. This issue was fixed by modifying the circuit by adding a simple diode across the signal line coming from GPIO peripheral and common ground. The following image shows how this was done:

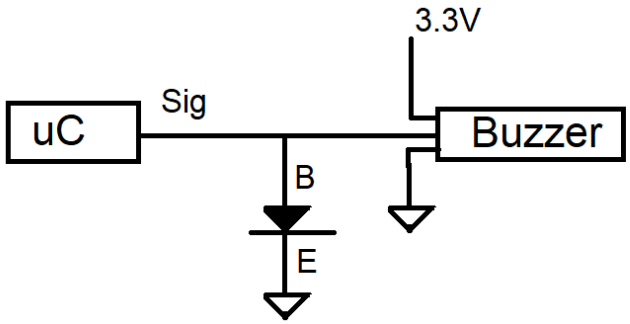


Figure 6-33 Hardware fix for Buzzer problem

7.0 Communications

Communications was an integral part of the system developed for this project, This section briefly describes the different communication channels that the user can use to communicate with the overall system. Appendix A further describes the protocols used by different communication devices that were used for this project.

For this project, there were two aspects of communicating with the flight controller:

- Sending Behavioural commands
- Configuring the system and receiving telemetry and flight data

The following image shows these two aspects of the communications and how the user and the system itself fit into the picture. Each aspect of the communications are described further in sub-sections that follows.

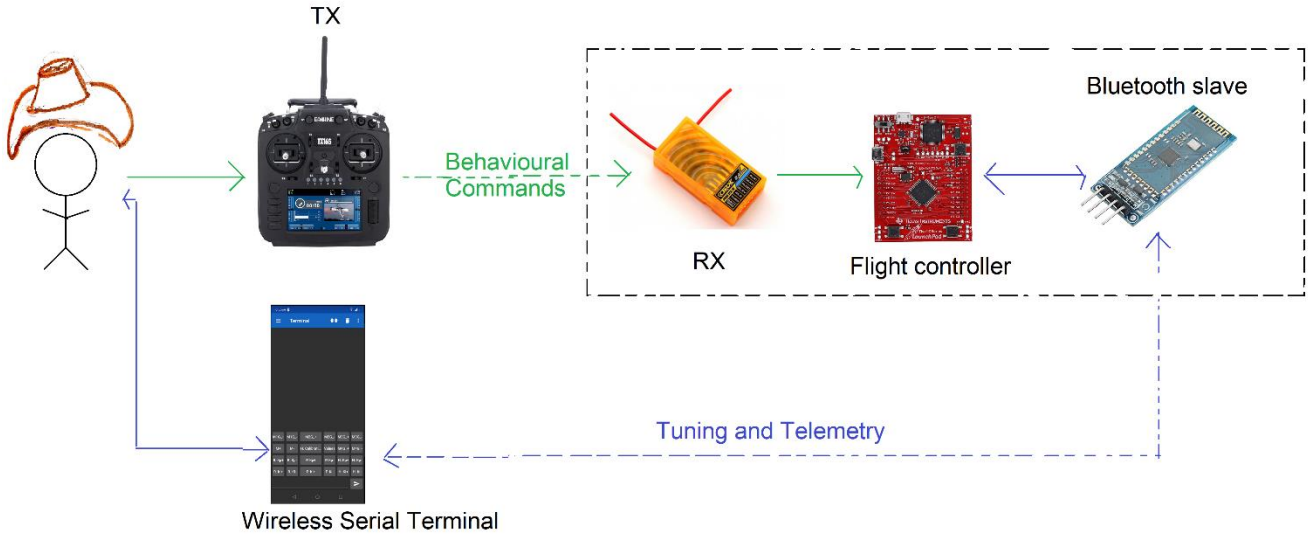


Figure 7-1 Two aspects of communication

7.1 Behavioral command and control channel

This aspect of communication with the flight-controller was concerned with the issue of controlling the behavior of the quadcopter. For this project, following commands needed to be sent to control the behavior of this system:

- GO FORWARD
- GO BACKWARD
- GO LEFT
- GO RIGHT
- GO UP
- GO DOWN
- STOP IMMEDIATELY, BUT SAFE LAND
- SWITCH TO CALIBRATION MODE
- SWITCH TO MANUAL MODE
- SWITCH TO AUTOMATIC MODE
- STOP IF RECEIVER LOST CONNECTION

The devices for this aspect of communication with the system were radio receiver and radio transmitter operating at standard ISM band of 2.4 GHz. These devices are briefly described below.

7.1.1 Receiver

The receiver used in this project was OrangeRX R615X 2.4 GHz receiver. Additionally, the receiver had true diversity antenna system that can receive signals from every direction. The following image shows this receiver and how it was mounted on the drone.

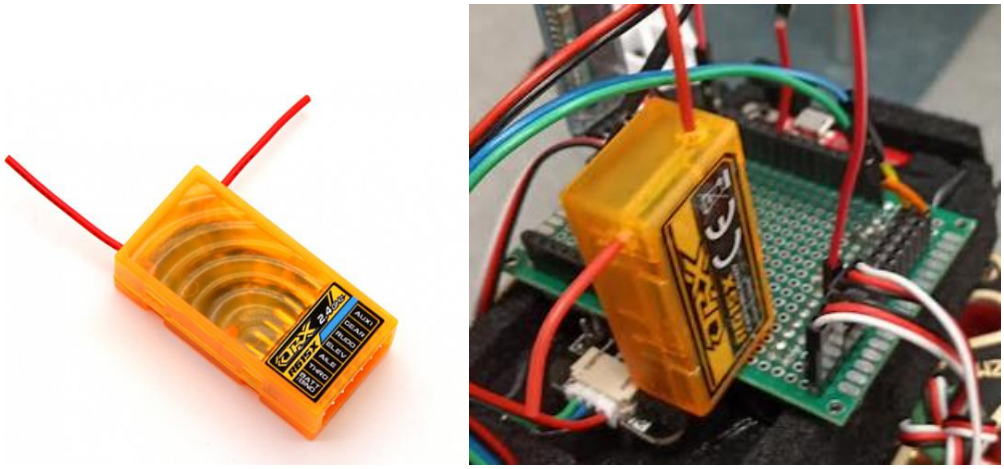


Figure 7-2 Receiver mounted on the system

This receiver communicates with the transmitter using DSMx (x = v2) protocol. The discussion of this protocol is further provided in REF. This receiver can support up to six radio channels, therefore, the transmitter can establish six parallel communication channels with this receiver. For this project, all the six channels were used to control the behavior of the drone. The following table lists each radio channels and what data was being sent over each channel.

Channel No.	Channel Name	Logical Data
1	Aileron	Roll angle
2	Elevator	Pitch angle
3	Throttle	Throttle level
4	Rudder	Yaw angle
5	Mode selector	• 0 % to 40 % -> Calibration or Ideal mode • 41% to 80 % -> Manual mode • 80% to 100% -> Automatic mode
6	Emergency Stop	• 0% to 50% -> Estop Not Engaged • 51% to 100% -> Estop Engaged

Table 7-1 Radio channels and the type of data transmitted through them

Once the transmitter sends the data using DSMx protocol[21], the receiver receives it and creates a separate Pulse width modulation (PWM) signal for each channel as well as it creates a combined Pules position modulation (cPPM) signal. The reason why this receiver was selected was because of this single cPPM signal. Appendix A further describes cPPM signals.

Briefly, cPPM single contains values from each channel of the transmission. Thus, using a single wire from receiver to microcontroller, the system can get data of all six parallel communication channels. This is very helpful in reducing the complexity of the overall hardware setup. Appendix A further describes cPPM signals captured during development of this project. Following image shows different protocols involved for this aspect of communication, as well as the cPPM signals captured from the output of the receiver.

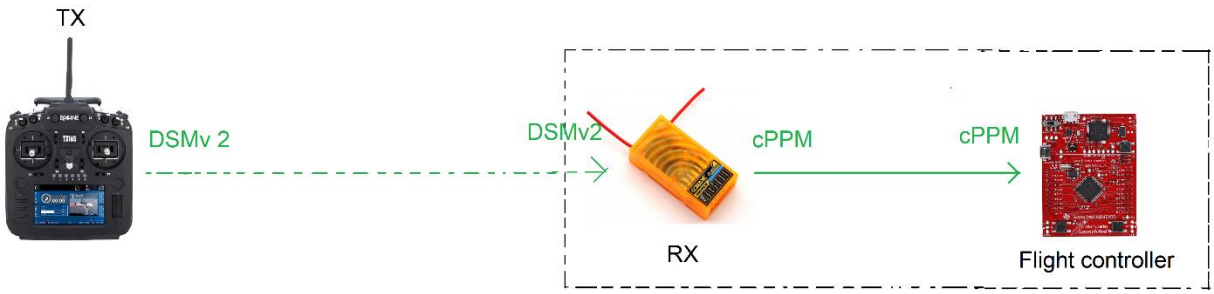


Figure 7-3 Different radio protocols used for communication

For implementing the driver of the receiver, the General-Purpose Timer peripheral was used to decode the individual channel data from the cPPM signal.

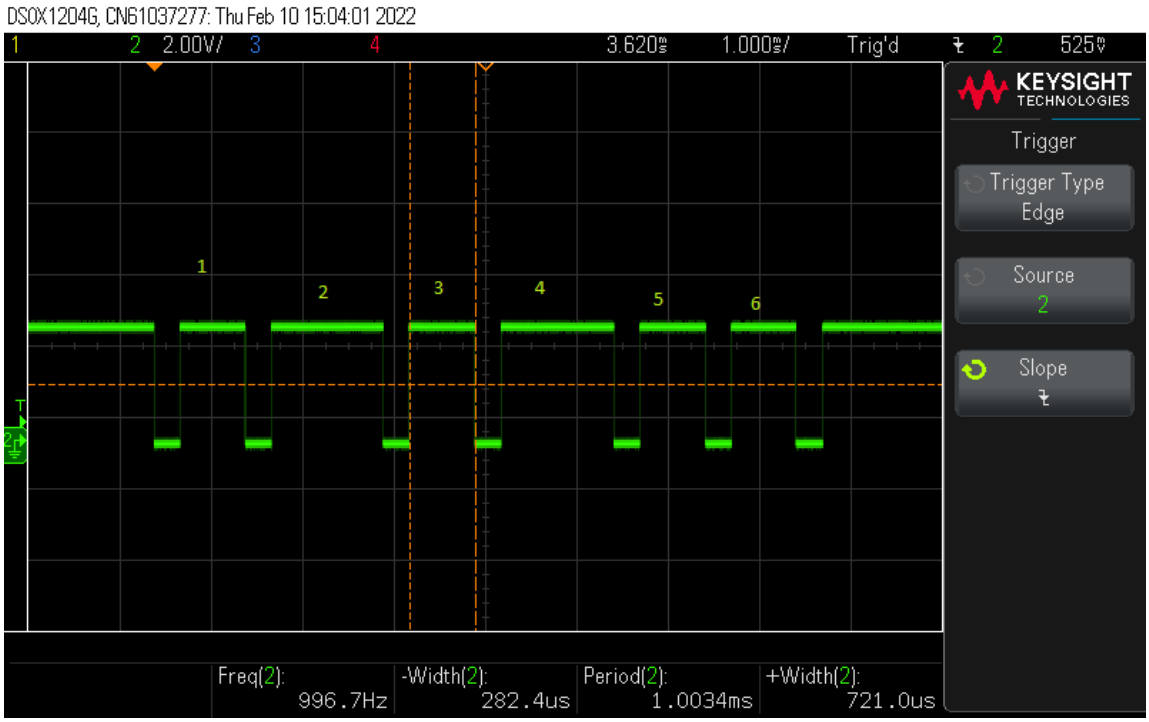


Figure 7-4 cPPM signal when channel 3 was at 0%

7.1.2 Transmitter

The transmitter used for this project was Eachine TS-16S transmitter. The reason for selecting this transmitter was due to its versatility, programmability, and many other features. This transmitter supported transmission of upto 32 physical channels. It also has an integrated multiprotocol chip that can support transmission of radio signals using various protocols, including DSMx (x = v2).

Additionally, the mixing of individual channels can be programmed before they are transmitted, which allowed changing channel orders easily. This transmitter also supported fine trimming of the inputs; thus it can be used to offset the transmitted signals such that the flight-controller gets the correct channel data. This feature saved considerable amount of time when creating the driver for the receiver.

Further, the feature of turn-on locks prevented the accidental flights when the transmitter was powered on. This added an additional layer of safety precaution for the flight system.

The following image shows the transmitter used for this project.

Figure 7-5 Transmitter used for this project

7.1.3 Challenges

This aspect of the project was very critical for the system to succeed. This was because user sent control commands using this sub-system. This section briefly describes few of the challenges that were encountered when creating this communication sub-system.

First, after developing basic support for the receiver, it was found that DSMx protocol used channel hopping technique to reduce errors during transmission of radio signals. It was found that that the order of channels that was modulated by the receiver was not same every time. Thus, a software solution was developed that tracked channel orders and ensured that even if the physical channel order was different, the logical channel order was pointing to their respective physical channel correspondence.

Another challenge that was encountered with this aspect of communication was the interference issue due to other devices using the 2.4 GHz ISM band. This was minimized by designing the PCB such that the devices that used radio signals were not very close to each other. However, no additional measures were taken for this issue due to time constraints.

7.2 Telemetry and flight data

This aspect of communications was concerned with tuning and configuring the entire system while the system was operating. This was also a very critical aspect of this project since this was the only way a user can tune the system to see the control action of the automatic controllers of the drone.

The configuration and tuning of the system were done by using a UART-to-Bluetooth slave module and an Android application developed by XYZ. The user first configures the application to send the commands to the Bluetooth module attached to the flight controller. Once the Bluetooth module receives signals from the user, it converts the signals into UART signals and sends it to the flight controller.

Once the flight control software receives the data from the user, it takes appropriate actions that the user wants and sends the information back to user via Bluetooth module using UART signals. This UART signals are then converted into wireless radio signals by the Bluetooth module and sent to the mobile application. The application then displays the information that was sent by the microcontroller.

Following images shows the Bluetooth module mounted on the flight controller PCB and the screen captures of the mobile application.

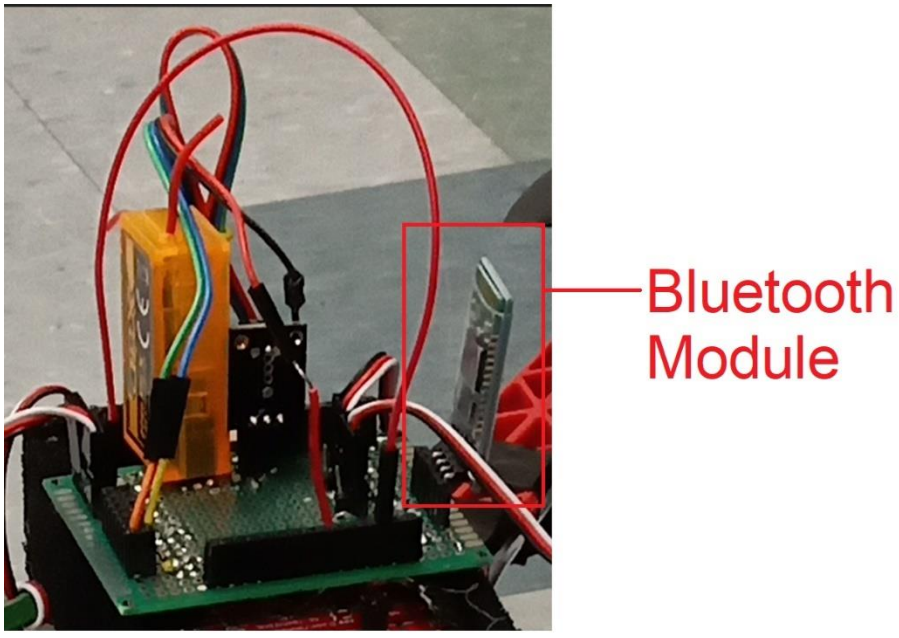


Figure 7-6 Bluetooth module mounted on the flight-controller PCB

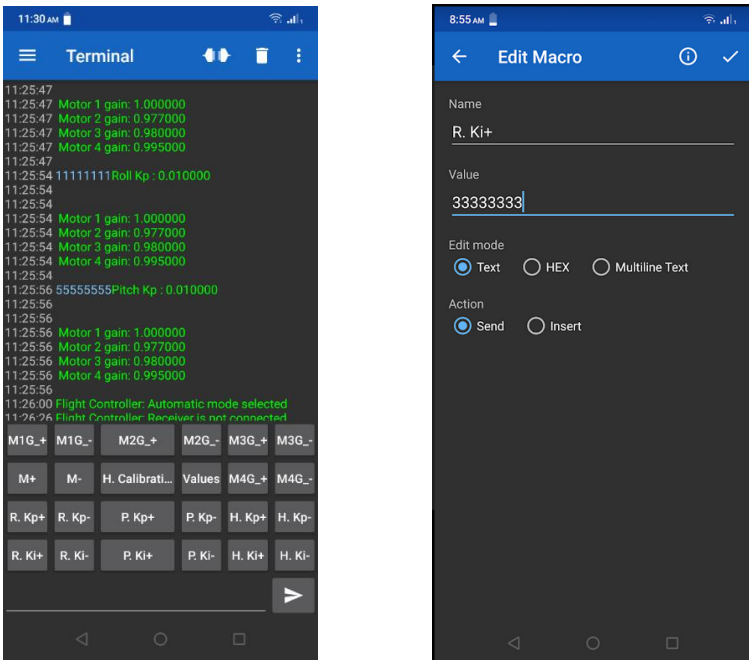


Figure 7-7 Mobile application configured for this project [23]

The following table lists all the commands that was configured when the above image was captured. Please note that the user can very easily add/remove/modify the above commands inside the flight controller software.

Button	Code	Purpose	Button	Code	Purpose
M1G_+	G	Increment Motor 1 gain by 0.001	**R. Kp+**	1	Increment Roll Proportional Gain by 0.01
M1G_-	H	Decrement Motor 1 gain by 0.001	**R. Kp-**	2	Decrement Roll Proportional Gain by 0.01
M2G_+	I	Increment Motor 2 gain by 0.001	**P. Kp+**	5	Increment Pitch Proportional Gain by 0.01
M2G_-	J	Decrement Motor 2 gain by 0.001	**P. Kp-**	6	Decrement Pitch Proportional Gain by 0.01
M3G_+	K	Increment Motor 3 gain by 0.001	**H. Kp+**	9	Increment Height Proportional Gain by 0.01
M3G_-	L	Decrement Motor 3 gain by 0.001	**H. Kp-**	A	Decrement Height Proportional Gain by 0.01
M+	D	Increment Max duty by 0.1	**R. Ki+**	3	Increment Roll Integral Gain by 0.0001
M-	E	Decrement Max duty by 0.1	**R. Ki-**	4	Decrement Roll Integral Gain by 0.0001
H.Calibration	Z	Trigger Pressure Sensor Calibration routine	**P. Ki+**	7	Increment Pitch Integral Gain by 0.0001
Values	X	Display current system attitude (roll, pitch, altitude)	**P. Ki-**	8	Decrement Pitch Integral Gain by 0.0001
M4G_+	M	Increment Motor 4 gain by 0.001	**H. Ki+**	B	Increment Height Integral Gain by 0.0001
M4G_-	N	Decrement Motor 4 gain by 0.001	**H. Ki-**	C	Decrement Height Integral Gain by 0.0001

Table 7-2 All Bluetooth commands that were supported during testing phase

8.0 Interfacing Sensors

The flight controller decides how to control the overall system using signals coming from two sources: The user, and the system attitude sensors. The sensors interfaced with the system senses the system attitude (Roll angle, Pitch angle, Temperature, Altitude and Yaw angle) and these quantities are then fed-back to the flight controller software.

Therefore, the sensors play a very important role in feeding back the process variables that will be then adjusted by the controller. The user simply adjusts the set-point values for each quantity that will be controlled (Roll angle, Pitch angle, altitude and Heading angle).

This section gives an overview of the different types of sensors that were interfaced with the flight-controller developed in this project, the approach that was taken when interfacing individual sensors as well as the challenges that were overcame to get the accurate system attitude data from the sensors. *Section 5.2* provides additional details about the configuration of these sensors.

8.1 BMI160

BMI160 is a low-power IMU (Inertial Measurement Unit), which provides highly accurate acceleration and gyroscopic measurements of the system it is mounted on. It has 16-bit resolution, which provides high resolution for indoor navigation applications such as indoor drone system. For communication with host microcontroller, it supports I2C and SPI protocol. For this project, I2C communication protocol was used to interface this with the TM4C123GH6PM microcontroller.

8.2 BMM150

This is a three-axis magnetometer which allows measurements of magnetic fields around the sensor in all three-perpendicular axis. This sensor was chosen because the device is low-powered and the resolution of the sensor is 0.3 μT, which was appropriate for this project. It has the same communication configuration as BMI160 (I2C - 400kHZ).

The following images shows the *BMX160* breakout board, which houses both BMI160 and BMM150 on same chip.

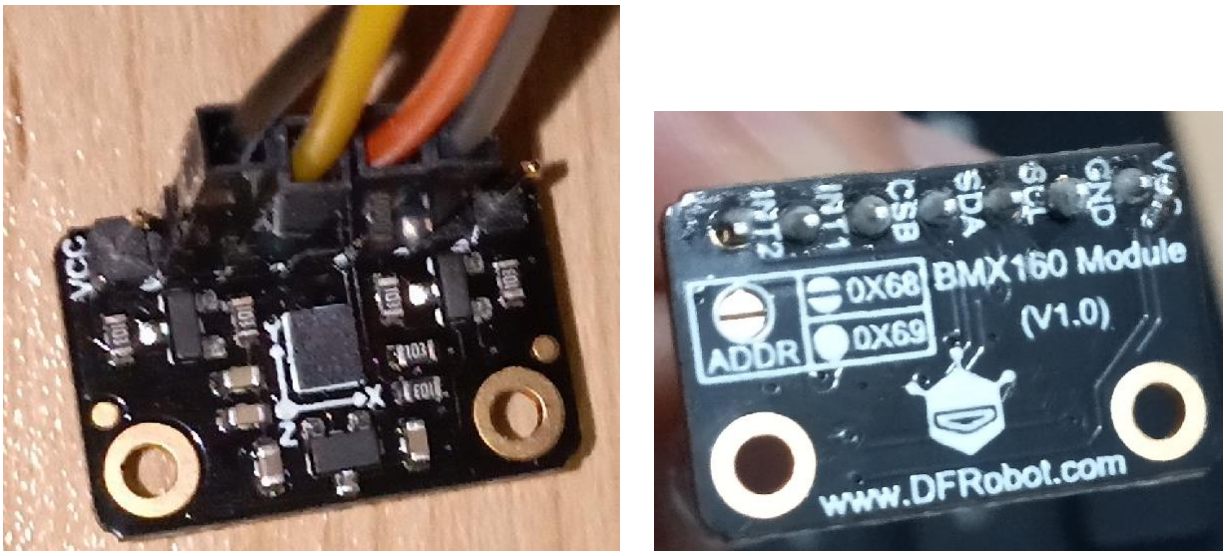


Figure 8-1 BMX160 (BMI160 + BMM150) breakout board

8.2 BMP388

This is a barometric pressure sensor that enables accurate altitude measurements specifically suited for drone applications. It also employs a very accurate temperature sensor which is used when determining altitude of the system based on pressure and temperature of the surrounding space.

The following images shows the BMP388 breakout board which acted as the primary sensor to determine the altitude of the drone.

Figure 8-2 BMP388 breakout board

Section 5.2 provides detailed description of the configuration of these sensors for this project. Each of the above sensors use Micro electro-mechanical systems (MEMS) to determine various system parameters. Appendix C describes MEMS technology further.

8.3 Interfacing approach

For interfacing any sensor, it is best to understand how the sensor does its job, how the sensor chip is laid out logically, and what care must be taken to properly use and configure each sensor.

To understand the sensor chip and the signal process chain better, it is best to refer to manufacturer provided block diagram of that sensor. These diagrams offer a system wide picture of the sensor board and show various sub-systems within the sensor chip that affect how the sensor works.

Therefore, to understand, configure and use the sensors better, the block-diagrams were thoroughly understood before starting the interfacing process. The following diagram shows the block diagram of BMX160 sensor as well as BMP388 sensor that were used in this project:

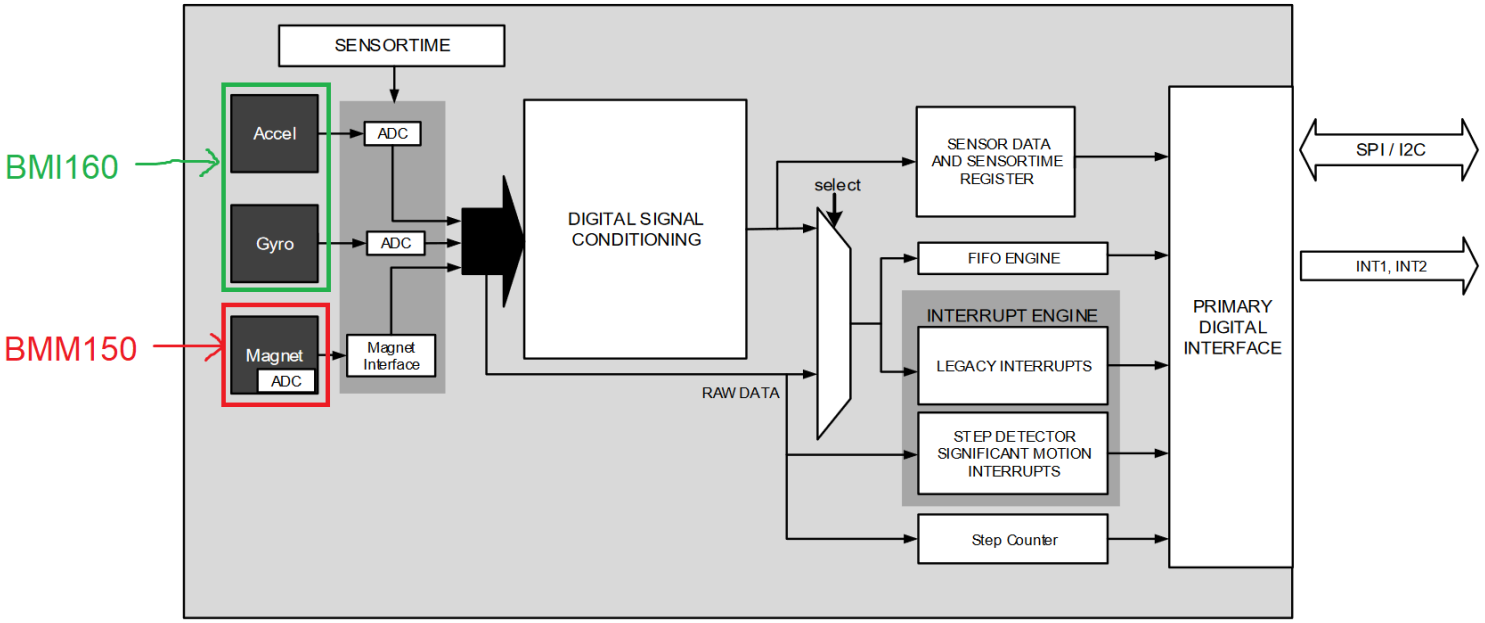


Figure 8-3 Block diagram of BMX160

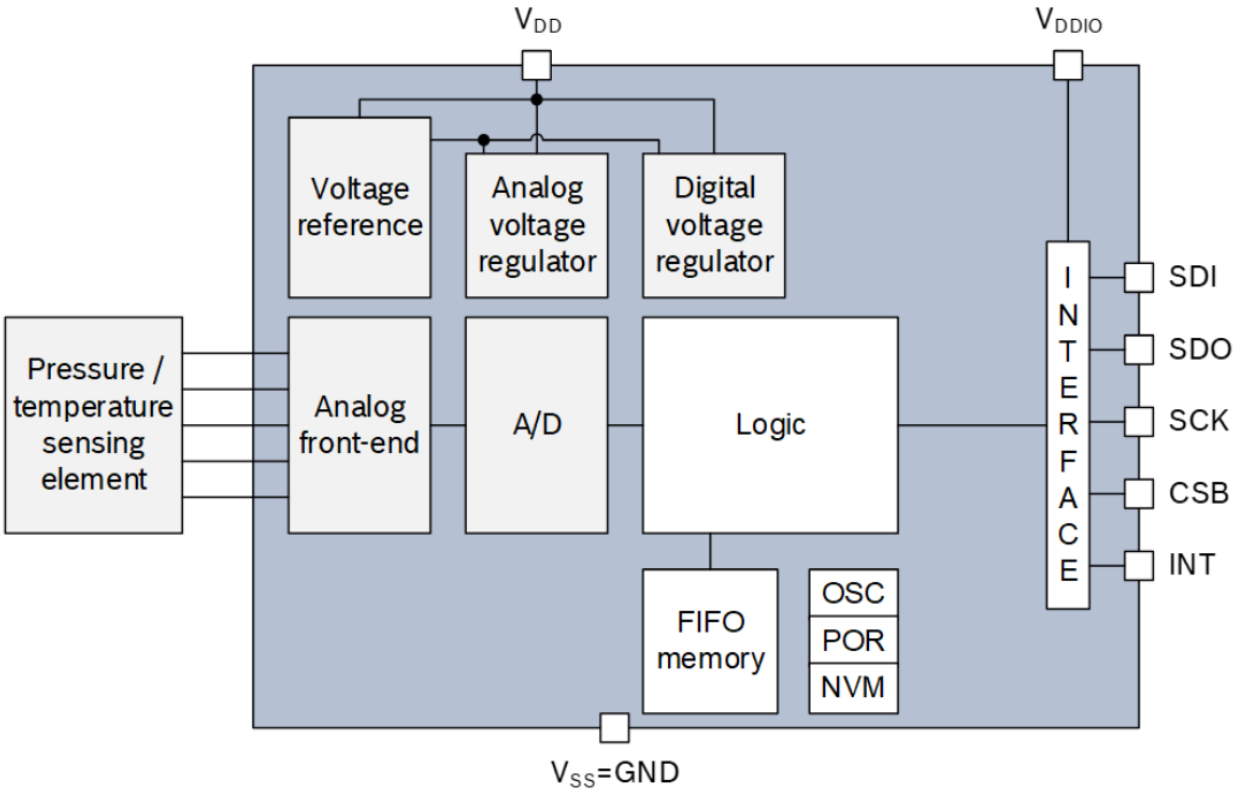


Figure 8-4 Block diagram of BMP388

The next step was to configure appropriate registers to enable/disable certain sub-systems within sensors, as well as configure the sensitivity, power modes, operational modes, filters, interrupts, and various other sub-systems on the sensor chip. *Section 5.2* provides the configuration details of each sensor for this project.

Next, the drivers were developed using bottom-up approach of software engineering. First, the low-level functionalities such as I2C communication routines, configuration routines, routines for identifying I2C devices, debugging routines, etc. were implemented. Then, the middle- and higher-level APIs were developed that were specific for each individual sensor. For example, routine to get pressure reading from BMP388 sensor, routine to get acceleration data from the BMI160 sensor chip, and so on.

The next challenge was to get the system attitude from the sensor data. The issue was that there was no one-to-one correspondence between the sensor data and the system attitude. For example, the roll angle was not being directly measured by the sensors. Instead, the roll angle needed to be determined by combining data from multiple sensors (accelerometer and gyroscope) in a specific manner.

Therefore, to determine the Roll, Pitch and Height of the system, a technique to combine the data from various sensors was needed. The technique to combine different data from multiple different data-sources that was used for this project is called *Sensor Fusion*.

For this project, a modified version of *Complimentary Sensor Fusion algorithm* was developed which mixed data from Accelerometer and Gyroscope to determine the Roll and Pitch angles of the system.

Similarly, the altitude of the system was the result of a sensor fusion algorithm that took pressure and temperature of the surrounding space and determined the altitude of the system. The algorithm to get altitude from pressure and temperature data was provided by the manufacturer of BMP388, and is referenced in REF.

Appendix D describes the Complimentary Sensor Fusion algorithm developed for this project. Appendix E describes how the altitude of the system was estimated using pressure and temperature data.

8.2.1 Calibration

Once the absolute height of the system was determined using the sensor fusion algorithm, the next step was to measure relative height of the system based on some reference altitude.

For this project, calibration of BMP388 sensor involved the implementation of a state machine that would calculate the exponentially weighted moving average of 100 readings from the sensor. Once this state machine was finished calculating the result, the controller would make that reading as its base altitude. This is the new reference point for the system which will be used to determine the relative altitude of the drone.

The state machine was implemented to avoid implementing a polled calibration routine, which would stop execution of the rest of the flight controller system while calibration process was executed. This was obviously not desired.

The calibration of magnetometer was also needed to get the yaw angle with respect to earth's magnetic field. However, the manufacturer did not provide enough details about how the calibration was to be done for magnetometer. As a result, this routine was not implemented due to lack of information about the sensor and the time constraints.

For Accelerometer and Gyroscope, the sensors were factory calibrated and provided very accurate readings if configured properly. Therefore, the calibration of these sensors was not necessary since each pertinent registers were appropriately configured using the low-level APIs developed as a part of this system.

REF describes the exponentially weighted moving average algorithm that was implemented for this project.

8.3 Gravity Series Sensor 10 DOF (BMX160+ BMP388)

During the final assembly stage of this project, another sensor breakout board was used, which included BMI160, BMM160 and BMP388, all in circuit board. This was very useful since all the sensors were accessed by using only four wires. Additionally, this reduced the space requirements on the PCB and ease the management of hardware. Also, no alteration of previously developed software was required since all the sensors were the same.

Following images shows the schematic of Gravity series sensor, and how it was mounted on the flight controller hardware.

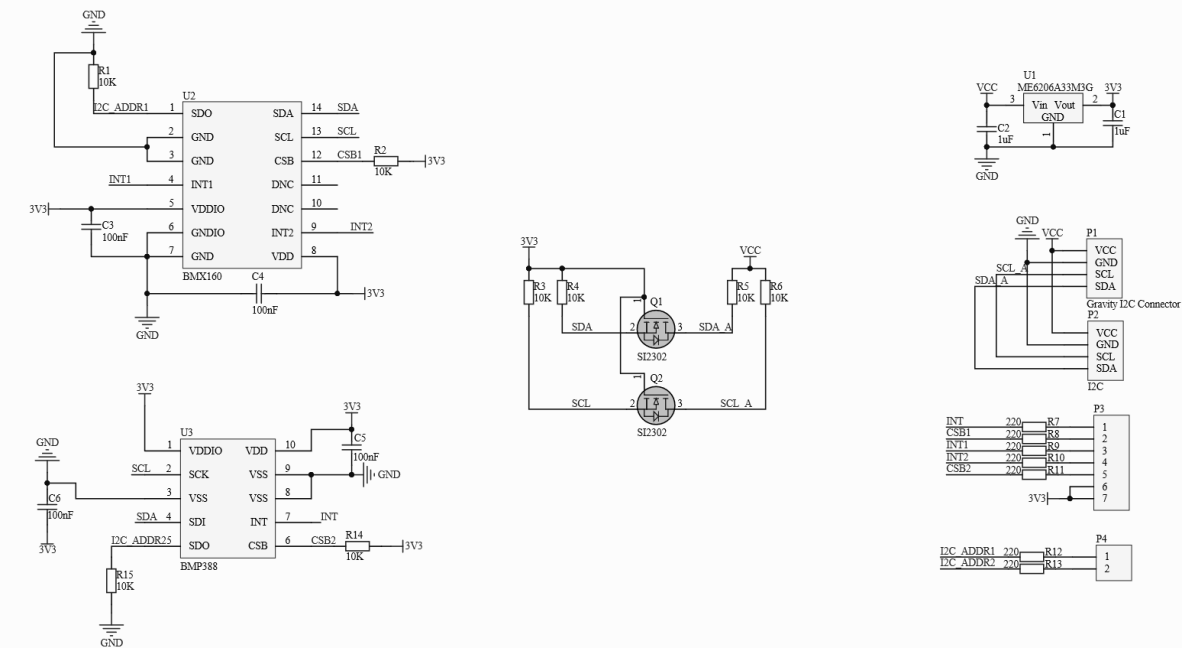


Figure 8-5 Schematic diagram of Gravity sensor (BMI160 + BMM150 + BMP388)

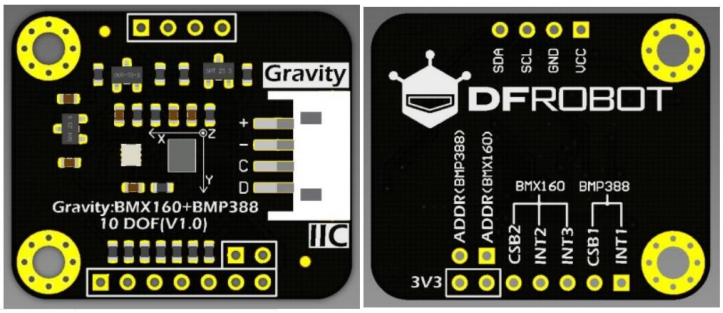


Figure 8-6 Breakout board for Gravity sensor

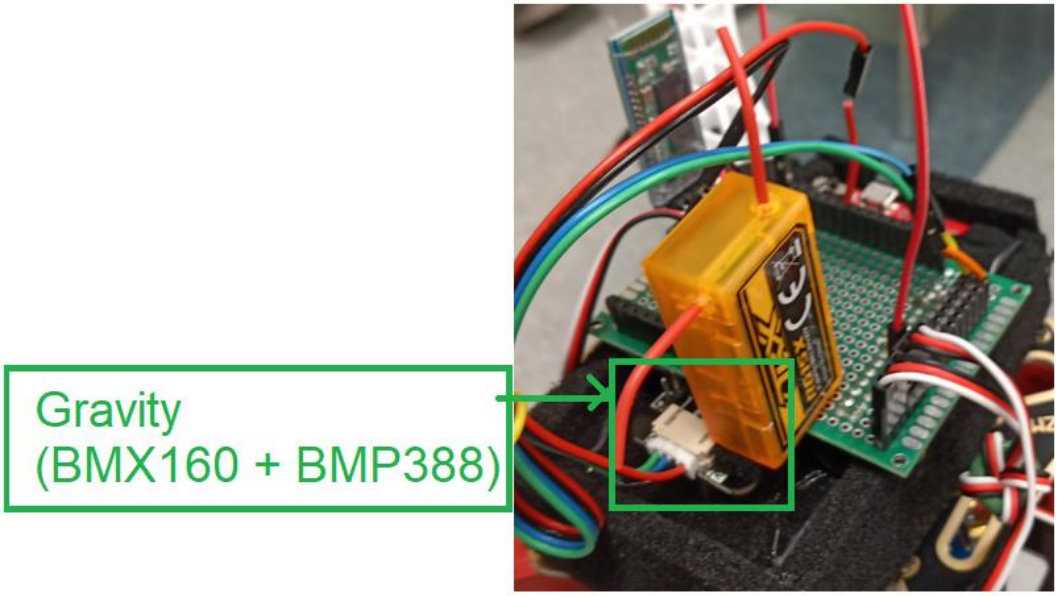


Figure 8-7 BMX160 + BMP388 mounted on the Flight-Controller PCB

9.0 Controllers

There are two flight-controllers developed for this project. The first controller is a manual controller, which is a multi-input multi-output controller. This controller was primarily developed to manually over-ride the automatic controller's actions. This was important for the demonstration of the project since it allowed to show how the drone was behaving for a given group of inputs.

The second controller that was developed for this project was the automatic controller. This controller used the feedback from various sensors and adjusted the system attitude (Roll, Pitch and Height).

The goal of this section is to briefly describe the controllers developed for this project. Detailed discussion of these controllers is omitted.

9.1 Manual Controller

During development of the system, there was a need to see how individual motors reacted based on only the input from the user with no feedback from the sensors. This was necessary to understand drone dynamics and see them in oscilloscope before implementing an automatic controller for this system. Therefore, manual controller was implemented.

The manual controller is arguably the most complex sub-system of the entire flight-controller. This is because the controller has multiple inputs, and controls multiple outputs. Also, the relationships between individual inputs and outputs are not one-to-one. That is, a single input influences multiple outputs, and a single output is influenced by multiple inputs.

The algorithm for manual controller also had to take the maximum roll, pitch, and yaw angles that the system can undergo as its parameters. That is, the controller needed to be configurable such that it allowed the users to determine by how much degree the roll, pitch or yaw of the drone can change. This was important to ensure that the system doesn't turn off any set of motors completely while it is flying.

The following table illustrates individual motor actions when only single input is present. If multiple inputs are present, the motor action is the superposition of the rows of inputs that form the group of present inputs. Therefore, a seamless mixing algorithm was developed for this controller. This mixing algorithm basically determines the speed of each individual motors for any group of inputs. It determines uses this table as its reference.

			MOTORS			
MOTION	Input		M1	M2	M3	M4
UP	Throttle	↑	↑	↑	↑	↑
DOWN	Throttle	↓	↓	↓	↓	↓
RIGHT	Aileron	→	-	-	↓	↓
LEFT	Aileron	←	↓	↓	-	-
FORWARD	Elevator	↑	↓	-	↓	-
BACKWARD	Elevator	↓	-	↓	-	↓
YAW RIGHT	Rudder	→	↓	-	-	↓
YAW LEFT	Rudder	←	-	↓	↓	-

Table 9-1 Control rules for Manual controller

In the above table, for **Input** column the meaning of individual arrow is described below:

↑ means that the value of Throttle or Elevator stick on the transmitter is positive.

↓ means that the value of Throttle or Elevator stick on the transmitter is negative.

→ means that the value of Aileron or Rudder stick on the transmitter is positive.

← means that the value of Aileron or Rudder stick on the transmitter is negative.

For the **Motors**,

↑ means that the speed of the respective motor will be incremented.

↓ means that the speed of the respective motor will be decremented.

For the implementation of manual controller, the table listed above is encoded in a two-dimensional array inside the flight controller software. This makes is very easy for the user to program any different control actions without having to modify the manual controller algorithm. The maximum roll, pitch and yaw angles can also be very easily tuned for any desired angle. The following code snippets shows these configuration setting within the flight-controller software.

```
const bool subtract_Motor[8][4] = { { 0, 0, 0, 0 },      // MOVE_UP [0]
        { 0, 0, 1, 1 },      // MOVE_RIGHT      [1]
        { 1, 0, 1, 0 },      // MOVE_FWD        [2]
        { 1, 0, 0, 1 },      // YAW_RIGHT       [3]
        { 1, 1, 1, 1 },      // MOVE_DOWN       [4]
        { 1, 1, 0, 0 },      // MOVE_LEFT       [5]
        { 0, 1, 0, 1 },      // MOVE_BKD        [6]
        { 0, 1, 1, 0 }       // YAW_LEFT        [7]

};
```

Figure 9-1 Implementation of control rules in software

```
float Channel_Tune_Variable[4] = { 0.2f,     // this is 'a' // Aileron  [0] (K2)
        0.2f,     // this is 'e' // Elevator   [1] (K3)        // K3 = e% of k1
        0.2f,     // this is 'r' // Rudder     [2] (K4)        // K4 = r% of k1
        1.0f      // Throttle   [3] (K1)
        };
```

Figure 9-2 Implementation for setting max roll, pitch and yaw angle in software

Thus, if combination of two inputs is present, each output that is influenced by those inputs will be affected. If a particular output happens to be affected by both inputs, the proportion of change of that output will be greater than the ones which are only affected by only one inputs.

For example, assume that the aileron input is on the right (i.e., the drone is commanded to go right), and the elevator is on the forward (i.e., the drone is commanded to go forward), then, the manual controller needs to take into account regarding which motor is affected by both inputs, which motor is affected by only one of the above inputs, and which motor is affected by none of the above mentioned inputs.

In this case, Motor 3 is affected by both the inputs, Motor 1 is affected by only elevator input, motor 4 is affected by only aileron, and motor 2 is not affected by any input. Thus, the manual controller will calculate the degree by which speed of each motor will be changed.

Following image shows an illustration of the above example.

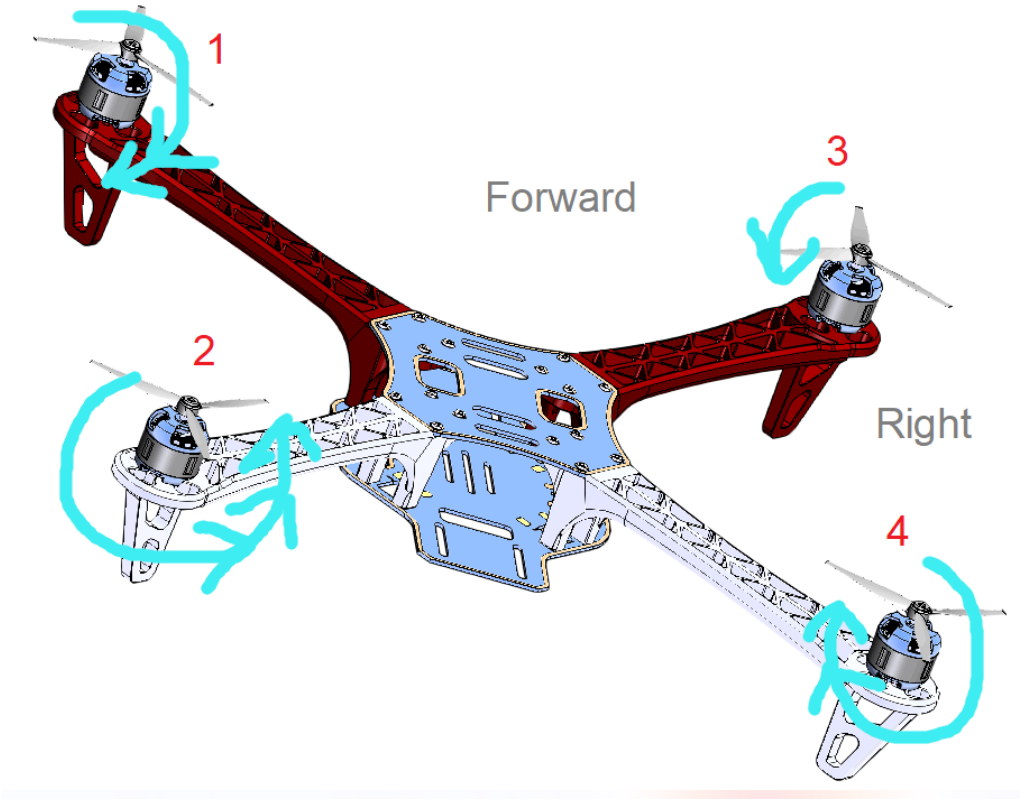


Figure 9-3 Illustrated behavior of quadcopter when it is commanded to go right and forward at same time

9.2 Automatic Controller

The manual controller implemented for this project has a serious flaw: it does not take sensor readings into account. That is, the manual controller has no idea about the system attitude. All it does is controlling outputs based on given set of inputs. This is not desirable since it cannot create a stable system.

For example, even if the motors are similar, each motor is different and has slightly different speed for a given voltage. Therefore, if manual controller is used, it will not know the actual speed of the motors. It will only get the signals from the user and calculate the speed of each motor and send it to motor controllers (ESCs).

For manual controller, the user himself acts as the feedback to the controller. Therefore, it is an open-loop control system This is why it is very difficult, if not impossible, to control the system attitude by using the manual controller.

For this reason, a closed-loop feedback control system, which used on-board sensors as its feedback, was also developed. This was the *automatic controller*.

In an automatic controller, the user does not directly interact with the controller. Instead, the user simply sets the setpoint for any input variable. This setpoint is then compared with the current system attitude, and the result of this comparison is then used by the automatic controller for deciding what control actions to take to get the system to the desired setpoint set by the user.

Following block diagram depicts the overall automatic control system implemented for this project.

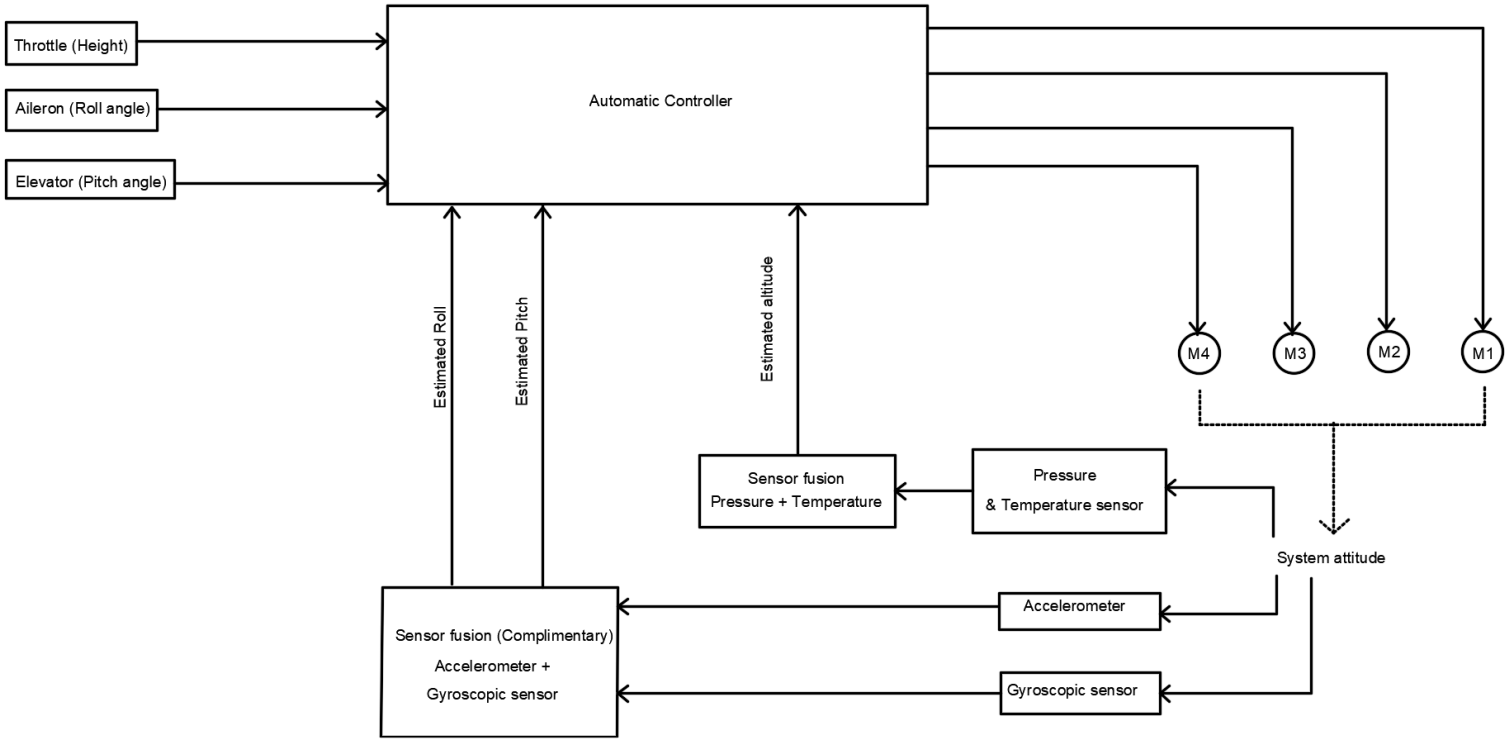


Figure 9-4 Block diagram of Automatic controller sub-system with feedback path implemented in this project

The automatic controller had three sub-controllers, each controlling different aspects of system attitude. Following table lists these sub-controllers as well as the system parameters that are being controlled by each of them:

#	Controller Type	Controlled parameter
1	Proportional + Integral	Roll
2	Proportional + Integral	Pitch
3	Proportional + Integral	Altitude (Height)

Table 9-2 Types of controllers internal to Automatic controller

The output of each of the above controllers determine the degree by which each system output needs to be changed for controlling a particular system attitude parameter. For example, the output of PI controller for Roll will determine the value by which a particular output (speed of motor) should increment, or decrement based on current system roll angle. Similarly, the output of PI controller for Pitch and Altitude will

determine the value by which a particular output should increment, or decrement based on current system pitch angle or altitude level.

The final output action of automatic controller is the accumulation of the results of each individual PI controllers for Roll, Pitch and Altitude. The following block diagram shows these individual PI controllers and how they each determine the final output of the controller.

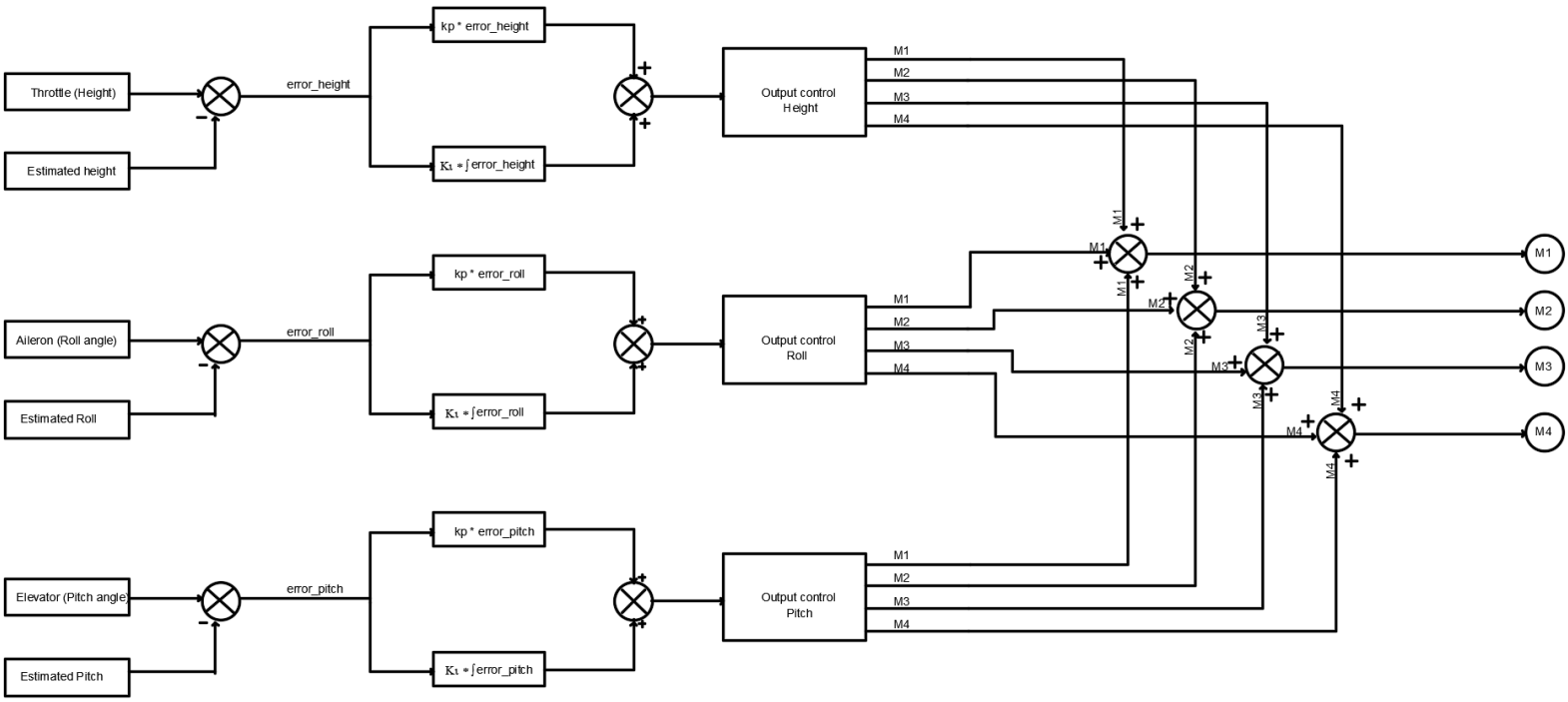


Figure 9-5 Internals structure of Automatic controller implemented for this project

For this system, the control actions for each of the controllers was *reverse action*. Therefore, internal to each controller, the error signal was created by subtracting the process variable from set-point variable. This was important to make sure that the error signal became small overtime.

Each automatic controller was also tuned using Bluetooth communication described previously. The values for each controller constant are listed below.

CONSTANTS	ROLL CONTROLLER	PITCH CONTROLLER	HEIGHT CONTROLLER
KP	0.02	0.03	0.05
KI	0.003	0.01	0.02

Table 9-3 Automatic controller Proportional and Integral gains

The controller for Yaw angle was not implemented because there was no way to determine the absolute heading angle of the system. Also, the proper calibration instruction of magnetic sensor was not provided by the manufacturer.

10.0 Setbacks and recommendations

As much as this project was rewarding, there were also many setbacks encountered throughout the development of this project. From the failed research to the failed test-setup, there were too many hurdles that were thrown on the way in creating a system that flies and controls itself.

The goal of this section is not to list each challenge faced during the development of this project. Instead, the goal of this section is to outline the learning outcomes, and to recommend what could have been done to avoid those issues throughout the development of this project. Following list outlines the major setbacks encountered while creating this system.

- Wrong sensor shipped: MPU 6050 instead of MPU9250
- Drifts in altitude readings
- System noise, and interfacing issues
- Failed attempts in creating manual controllers
- Sensor fusion
- Receiver issues
- Fire in the lab
- PWM to servo control signals
- Lost shipments and custom investigations
- Calibration issues
- Failed PCB milling
- Failed ESCs in the final stage of testing.

The sub-sections that follow briefly describes each issues listed above.

10.1 Wrong sensor shipped: MPU 6050 instead of MPU9250

During the research phase of the development, another IMU sensor was selected to mount on the system. This IMU was MPU 9250 by Invensen technologies. However, after receiving the sensor, and developing I2C communication, and register configuration support functions for the sensors, it was found that the actual chip on the breakout board is a different IMU (MPU 6050), and not the one that was selected. The IMU on that breakout did not have magnetometer, and therefore it was not the desired breakout.

Also, while developing the drivers for MPU 9250, interrupt driven approach was required for I2C. But it was found that the breakout board did not have any pins that was connected to the interrupt pin on the chip. Thus, a hardware solution was required to extract the interrupt signal from the chip to the breakout. It was done by modifying the breakout by cutting the trace on non-required signal that was connected to one of the breakout board's pins and connecting that breakout board's pin with the interrupt pin on the chip.

The following image shows the modified breakout board.

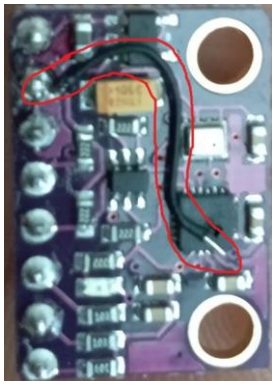


Figure 10-1 Interrupt signal extraction by modifying the breakout of MPU9250

Overall, it was a good learning experience since it involved modifying pre-built PCB, and interrogating chips.

10.2 Drifts in altitude readings

The altitude of the system is not directly sensed by any on-board sensors. Instead, a sensor fusion algorithm is used to fuse the readings from pressure sensor and temperature sensor. The result of this algorithm is the absolute altitude of the chip.

The result of this fusion algorithm fluctuates when either pressure or temperature of the surrounding changes. However, if appropriate measures are taken, the magnitude of the drifts can be minimized considerably.

For example, the oversampling of the pressure sensor and the temperature sensor is done in the system to make the readings more accurate. There is also a hardware IIR

filter on BMP388 barometric pressure sensor chip, which can filter out pressure noise created by sudden changes in pressure (e.g. door closes, someone speaks near the sensor, air flow near pressure, etc.). The signals can be further filtered using software solutions, such as a first order low-pass filter.

This was an interesting challenge since it involved learning about how the barometric sensor worked and how the signal chain should be configured to get the accurate readings from the sensors without any drifts.

10.3 System noise, and interfacing issues

It is very critical that every sensor works properly before trying to make them all work together. The reason is that there might be noise induced due to some other sensors or devices that may interfere with the operation of the device that is being currently worked on.

However, when everything works by itself, the next major move is to harmonize everything and make every device work properly all at the same time. This is where the main challenge for interfacing appeared for this project. It was found that PWM signals were interfering with the I2C sensors, and therefore, the I2C communications were always failing.

Later, it was found that the I2C peripheral on the microcontroller has a I2C filter block (Glitch filter) which can be enabled and configured by accessing appropriate register or by using a Tivaware supported API. Once the I2C filter block was configured, every device and sensors on the system worked as expected.

It was very well realized how noise can bog down the entire system. Therefore, every measure must be taken to ensure that the noise in the system is minimal.

10.4 Failed attempts in creating manual controllers

There were four failed attempts for creating manual controllers. This was due to lack of knowledge regarding the complexity of multi-input multi-output systems.

The challenge was solved by creating mathematical model of the system and the controller itself. Many simulations were done by doing calculations on paper as well as computer software. Thus, this was an interesting challenge to solve because it required a research approach.

10.5 Sensor fusion

Initially, not much thought was given for this system. It was thought to be a simple algorithm, however, it was soon realized that this is perhaps very important aspect of the automatic controllers.

Therefore, the lesson that was learned by passing through this hurdle was to take everything into account, and make sure that the manufacturer provides enough details regarding any device specific routines, such as calibration.

10.6 Receiver issues

For the receiver, channel hopping was the main issue. The challenge was to rearrange the channels in a fixed order, so that the software can then extract the information from each channel.

This was a very good exercise in tracking real-time signals. The knowledge gained when creating this system was very helpful when implementing manual controller for the flight-controller.

10.7 Fire in the lab

This was perhaps the most discouraging setback that was encountered during the testing phase of the system.

It was found that because of exposed terminals, the BLDC motor phases shorted and excessive current flowed through the ESC, which resulted in catching fire.

This was a very good learning outcome since it showed how dangerous it can be when working with electrical system if proper safety precautions are not taken.

This incident could have been avoided if proper care was taken to cover exposed metals of the connectors using heat shrinks.

10.8 PWM to servo control signals

During the last phase of development of this system, it was found that the ESCs would not work with the PWM signals that the flight-controller was sending. Therefore, a way was needed to convert the PWM signals into the signals that were understood by the ESCs. These signals were found to be very common signals for servo control, therefore, the ESCs were using servo control signals and not PWM.

Because of how the software was structured, it was very simple to synthesis the servo control signals from the PWM signals. However, if the software was not structured properly, then it would have been very difficult to modify the system without having any other problems.

Therefore, this challenge showed how proper organization of software can help in a long run.

10.9 Lost shipments and custom investigations

This was not a technical issue. However, this issue was also very important to resolve since the backup parts were needed to ensure that even if somethings went bad during testing, there were enough extra parts to continue the testing.

The learning outcome was to always keep track of extra parts, and order early if needed to make sure that they are there when needed.

10.10 Calibration issues

Due to lack of information, the calibration routine to calibrate magnetometer was not implemented in this project. Therefore, for automatic controller, the yaw angle with respect to the earth's magnetic fields cannot be determined.

It was realized how important it is to have device specific information for developing systems.

10.11 Failed PCB milling

Having a PCB for the system was extremely important, since it made the entire system compact, easy to install and lightweight. However, after creating a prototype PCB by hand, additional PCBs were needed to ensure that if the prototype PCB fails, there will be other PCBs that can be used for testing and demonstration of the system.

When the PCB was designed in software and sent to milling machine for milling it out, it was found that the machine was out of calibration, and the milled-out PCBs got damaged.

Thus, there was only one prototype PCB that was operational. Therefore, during testing, extreme care was taken to make sure that the drone does not drop on ground accidentally and possibly damaging the prototype PCB.

10.12 Failed ESCs in the final stage of testing

Another ESC failed just few days prior to demonstration. This was not a good sign since failing ESCs required complete disassembly of the electrical system that was internal to the drone frame. This process was time consuming, and was not desirable.

This is where having an extra ESC helped. It there was no extra ESC, then the quadcopter would not have made to the demonstration aspect of this project.

11.0 Learning Outcomes

As mentioned in previous section, there were countless challenges encountered throughout the development of this project. These all challenges and setbacks had several learning outcomes, and the goal of this section is not to list each one of them individually.

This section only summarizes all the learning outcomes into one brief statement: How to innovate, solve problems and bring the ideas to life by using the knowledge and skills acquired from previous experiences.

Solving technological challenges is not just about solving problems. Its more about how other people can benefit because of the efforts that was put into solving these challenges. The system developed for this project might not find its place in every house, but the learned outcomes of this project will certainly help in creating the technology that will eventually help to make someone else's life easier and more productive.

12.0 Next Steps

There is certainly more work to be done for calibrating magnetic sensor and creating a complete automatic controller for all degrees of freedoms. Additionally, the support for distance sensor needs to be added to better get the closed proximity distance of the obstacles.

Next steps would be to create a fully autonomous flight control system with vision system and an on-board computer with inter-networking support. This can be used to make the entire system operatable over the internet and extend its ability to establish communication with the operator.

However, this project holds several memories, and therefore a decision will soon be made on whether to keep working on this system or to create another complete system from ground-up to add the above functionalities. If another system is developed, this project is a good candidate to give away to the people who have inspired me in the past.

13.0 References

Following table lists all the resources that were referenced and used for creating this report.

No.	Resource	Description	Link
1	TM4C123GH6PM datasheet	Datasheet for the microcontroller used in this project	link
2	EK-TM4C123GXL datasheet	Datasheet of the development board used for this project	link
3	Tivaware discription	Overview of Tivaware library, which was used for implementing this system	link
4	Code composer studio IDE overview	Integrated Development environment used to develop the software	link
5	BMI160 datasheet	Details of Accelerometer Gyroscopic sensor	link
6	BMM150 datasheet	Details of Magnetometer	link
7	BMP388 datasheet	Details of barometric pressure sensor used for this project	link
8	BMX160 datasheet	Details of IMU breakout board which was mounted on the system	link
9	MPU9250 datasheet	Details of additional IMU that was initially selected	link
10	MPU6050 datasheet	Details of IMU that was not compatible with this project	link
11	OrangeRX description	Details of the radio receiver	link
12	Bluetooth datasheet	Details of the Bluetooth module (HC-06)	link
13	Motors datasheet	Details of BLDC motors used in this project	link
14	ESCs description	Details of Electronic speed controllers	link
15	Frame specification sheet	Details of frame of the quadcopter	link(1) link(2)
16	ESC programmer description	Details of ESC programmer used for this project	link
17	OSI network model description	The primary source of inspiration for creating the layered software stack	link
18	ESC to motor signals image	The signals sent by ESC to control BLDC motor	link
19	Motor thrust to weight description	Information about thrust-to-weight ratio.	link
20	First order Low pass filter blog	Article about mathematical model of first order low-pass filter	link
21	Radio protocols	Article describing various protocols used in RC world	link
22	3D model of quadcopter	Modified 3D model used for illustration purposes for this project (Originally modeled by Nabajit Das)	link
23	Mobile application	Mobile Bluetooth application that was used and configured for this project. (Developed by Kai Morich)	link
24	git-SCM	Version control system that was used extensively to keep track of the source code of this project	link
A-1		Servo control signal image	link

A-2	Servo control information	link
E-1	Graph showing relationship between Pressure and Altitude	link
E-2	Hypsometric formula	link

Appendix A: cPPM signals

The receiver used in in this project was OrangeRX R615, which operated in 2.4 GHz ISM band and supported up to six-radio channels. The signals received are then modulated using both pulse-width modulation and pulse-position modulation techniques.

The cPPM signal is the combined signal containing data of all six pulse-position modulated signals from each radio channel. That is, the receiver converts the parallel data coming from six channels into a serial data using time-multiplexed signals. The following images illustrated the various outputs of the receiver and the signals that are sent to the ESCs.

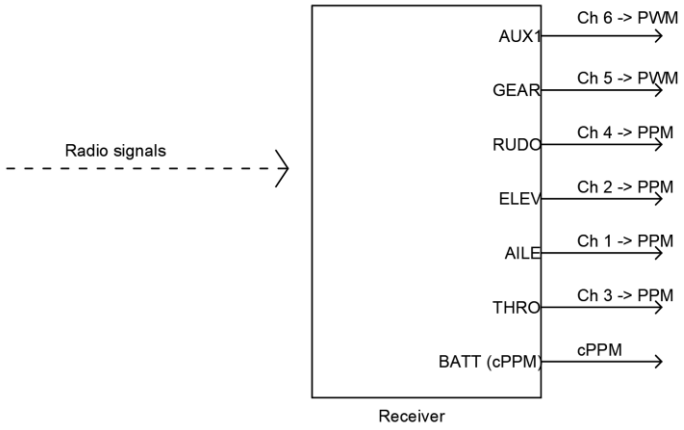


Figure A-1 Outputs of the receiver

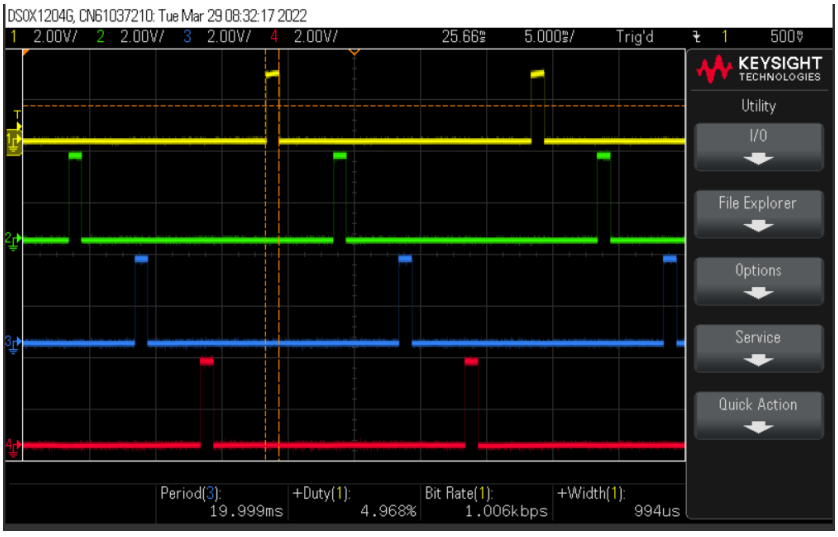


Figure A-2 Signals sent to the ESCs

The following images shows the oscilloscope image captures of cPPM signals when the channel 3 (throttle) was set to -100%, 0% and +100%.

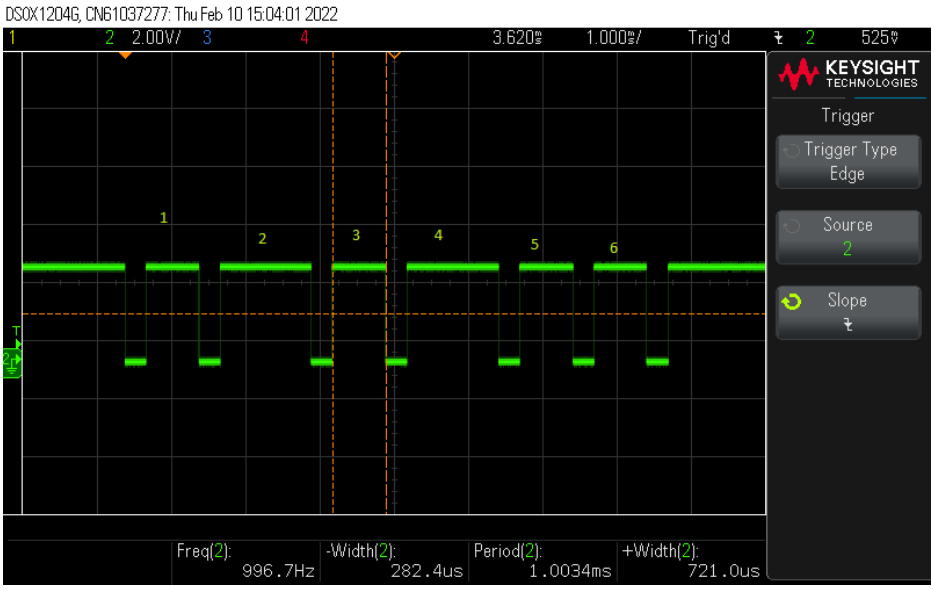


Figure A-3 cPPM signal for channel 3 when the data transmitted was -100% (Lowest value)

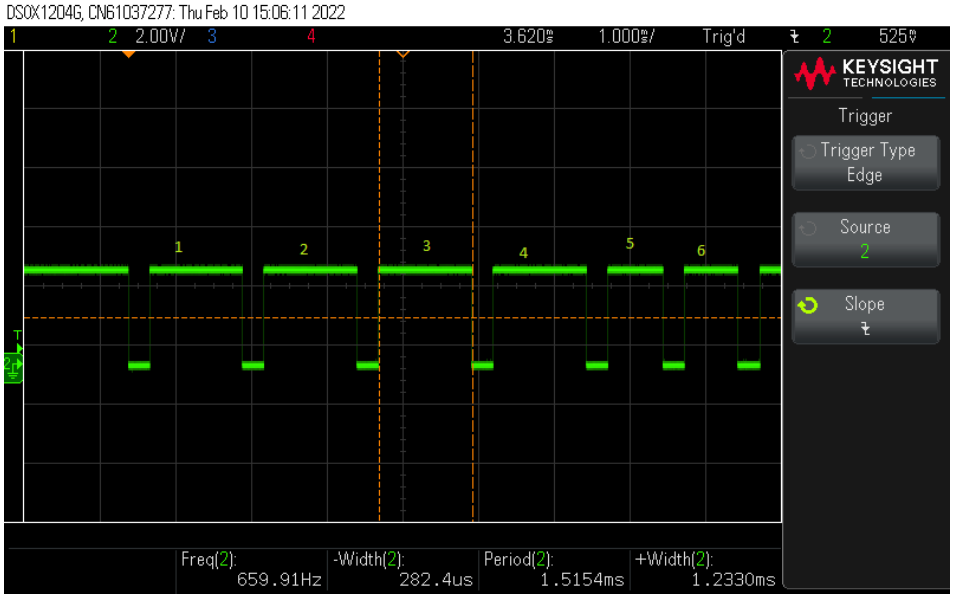


Figure A-4 cPPM signal for channel 3 when the data transmitted was 0% (Mid value)

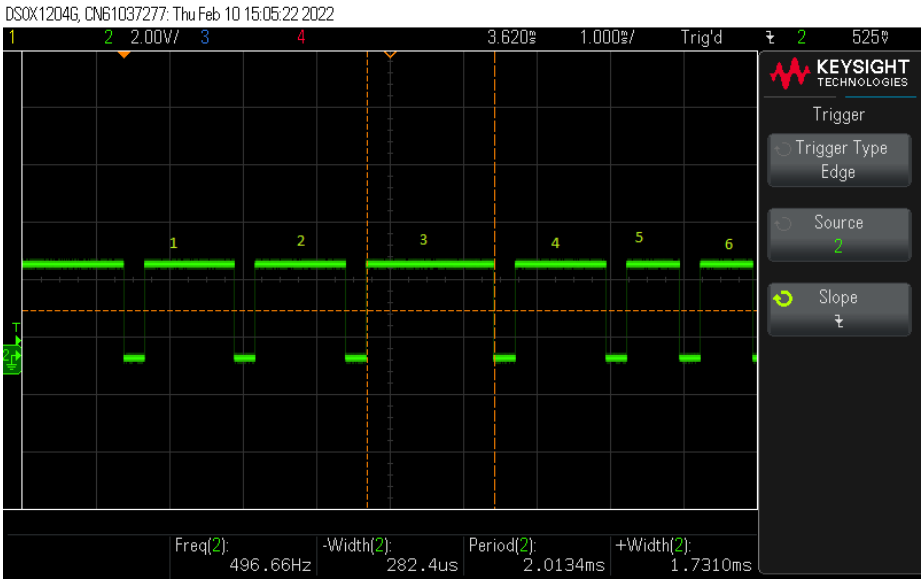


Figure A-5 cPPM signal for channel 3 when the data transmitted was +100% (Lowest value)

From the above images, it can be observed that the period of individual PPM signal is approximately 20 milliseconds. Also, for 0% throttle, the pulse period is approximately 1.5 milliseconds, for -100% throttle, the pulse period is approximately 1 millisecond, and for 100% throttle, the pulse period is approximately 2 milliseconds. This is exactly like a servo control signal. Thus, cPPM signal generated by this receiver is just a combination of six individual channels modulated using servo control signaling protocol.

The image below shows the servo control signals.[A-1, A-2]

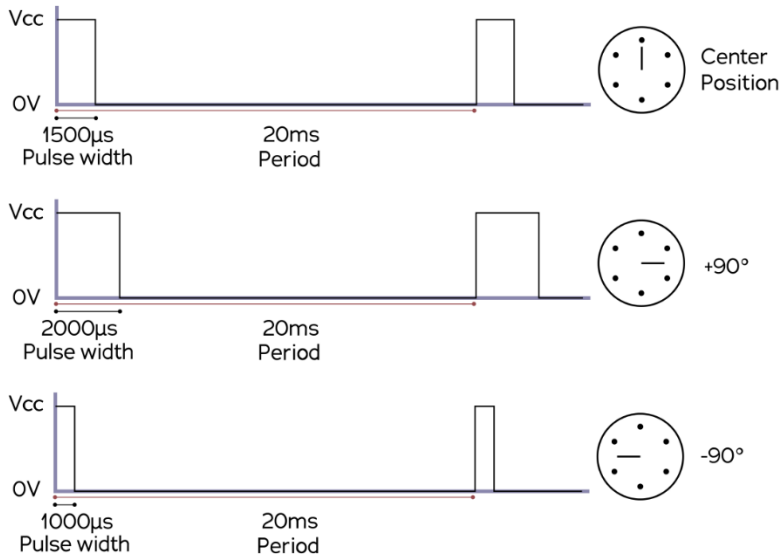


Figure A-6 Servo control signals

Appendix B: ESC programming

Electronic speed controllers (ESCs) control the speed of brushless motors by sensing the back emf generated across the phases of the motor and deciding the rate at which the current switches among three-phases of the motor. They employ a small microcontroller which executes a special firmware that measures back emf, acts on the received signals from host microcontroller (Flight-controller) and changes the speed of the motors by switching the high-power field-effect transistors (FETs) connected on the ESC circuit board.

However, before ESCs can properly do their job, the firmware within them needs to be configured appropriately. The configuration usually involves setting bits inside the ESC microcontroller registers. There are various registers that needs to be configured so that the ESC is used safely and does not catch fire or damage the overall system.

The process of configuring various registers is referred to as *ESC programming*. For this project, a separate ESC programmer was used which took the input from the user, and then configured the registers of the attached ESC's microcontroller. The user simply sets the appropriate switch for a required functionality and the ESC programmer programs the appropriate registers inside the ESC's microcontroller.

The ESC programmer used for this project is shown below.

Figure B-1 ESC Programmer used for this project

The various functions supported by this ESC programmer is given in the table below.

Parameter	Settings	Parameter	Settings
	Signal to ESC	**Protect type**	Reduce
ESC	+ (Vcc)		Stop
	- (Gnd)	**Break**	Off
	+ (Not used)		ON
OPTO	- (Not used)	**Start**	Very soft
Timing	Auto		soft
	7-22		Soft Acc
	22-30	**Battery**	NiMH
Rotation	Normal		Lipo
	Reverse		RPM off
Batt. Protect	2.8 V / 50%		Helic 1
	3.0 V / 65%		Helic 2
	3.2 V / 65%		

Table B-1 Configuration of each ESCs for this project

The configuration of each ESC for this project is highlighted in the above table.

Appendix C: Micro electro-mechanical systems

Micro electro-mechanical systems are very small form factor chips that has microscopic electro-mechanical systems which are arranged in a very specific manner so that the electrical property of the system changes when external force acts on the system in specific manner.

Depending on how the MEMS system is manufactured, the varying electrical property of the system can be its capacitance, resistance, voltage, current or inductance. Most MEMS systems have variable capacitance so that when the force is applied on the system, the capacitance between the suspended plates changes and the magnitude of the force can then be determined by detecting the difference in capacitance.

MEMS technology is a very widely used technology in IMU sensors which determine the magnitude of inertial forces acting on a system. The accelerometer and gyroscopic sensor used for this project used this technology.

Appendix D: Complimentary sensor fusion algorithm

The block diagram of complimentary sensor fusion algorithm that was implemented in this project is shown below:

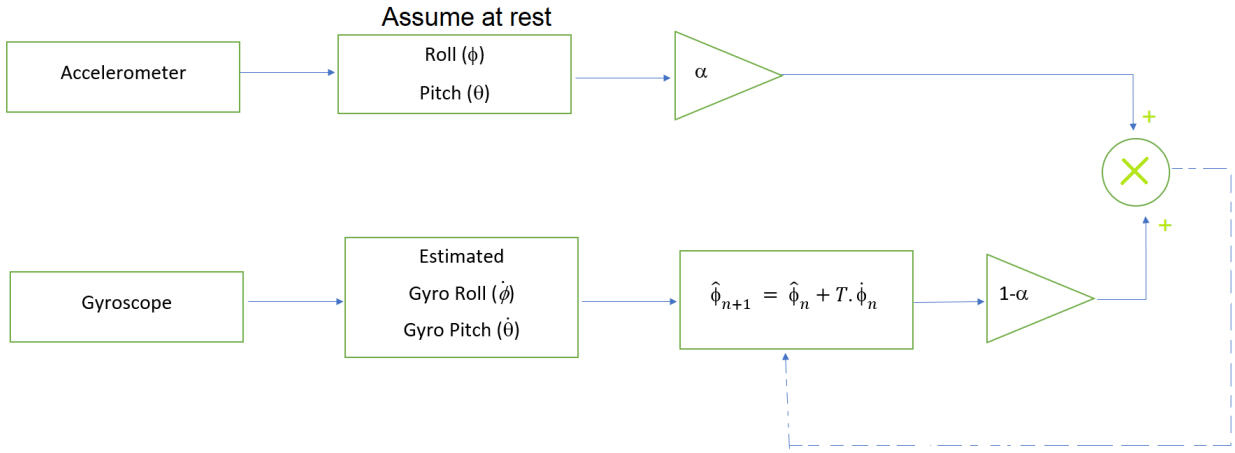


Figure D-1 Block diagram of Complimentary Filter Algorithm implemented for this project

The data from accelerometer is in m/s^2.

First, the Roll (ϕ) and Pitch (θ) is calculated directly from the accelerometer measurements:

$$\phi = \tan^{-1}(\frac{Ay}{Az})$$

$$\theta = \sin^{-1}(\frac{Ax}{9.81})$$

Where (Ax, Ay, Az) is the linear force acting on the accelerometer on principal axes.

Now, assume that gyroscope readings are (Gx, Gy, Gz) in rad/sec.

Therefore, the estimated roll angle ($\dot{\phi}$) and pitch angle ($\dot{\theta}$) can be calculated by using following Euler rotational matrix.

$$\begin{bmatrix}\dot{\phi}\\ \dot{\theta}\end{bmatrix} = \begin{bmatrix}1 & \sin\phi\tan\theta & \cos\phi\tan\theta\\ 0 & \cos\phi & -\sin\phi\end{bmatrix}\begin{bmatrix}Gx\\ Gy\\ Gz\end{bmatrix}$$

The estimated new roll ($\dot{\phi}_{n+1}$)and pitch ($\dot{\theta}_{n+1}$) of the system from the $\dot{\phi}$, $\dot{\theta}$, ϕ and θ using the above algorithm is

$$\dot{\phi}_{n+1} = \phi.\alpha + (1-\alpha)(\dot{\phi} + T.\dot{\phi}_n)$$

$$\dot{\theta}_{n+1} = \theta.\alpha + (1-\alpha)(\dot{\theta} + T.\dot{\theta}_n)$$

Where, α is the constant called complimentary constant such that $0 \leq \alpha \leq 1$

The implementation of the above algorithm in this project is given below. the following code segment is from the `BMX160_updateData()` function in `BMX160.c` file located under `local_include/BMX160/` directory.

```
    // Implimentation of complimantary fliter sensor algorithm.

    // for acceleration, convert g values to ( m / s^2).
    float accDataX_MPS2 = accData.x_LPF * G_TO_MPS2;
    float accDataY_MPS2 = accData.y_LPF * G_TO_MPS2;
    float accDataZ_MPS2 = accData.z_LPF * G_TO_MPS2;

    // get roll(phi) and pitch(theta) from acceleration
    float phiHat_acc_rad = atan2f(accDataY_MPS2, accDataZ_MPS2); // estimated roll in rad
(only from accelerometer)
    float thetaHat_acc_rad = asinf(accDataX_MPS2 / G_TO_MPS2); // estimated pitch in rad (only
from accelerometer)

    // for filtered gyro data, convert it into rad/sec.
    float gyroDataX_RPS = gyroData.x_LPF * DEG_TO_RAD;
    float gyroDataY_RPS = gyroData.y_LPF * DEG_TO_RAD;
    float gyroDataZ_RPS = gyroData.z_LPF * DEG_TO_RAD;

    // Transform the body rates (gyro rates) into Eular rates

    float phiDot_rps = gyroDataX_RPS
            + (sinf(phi_rad) * tanf(theta_rad) * gyroDataY_RPS)
            + (cosf(phi_rad) * tanf(theta_rad) * gyroDataZ_RPS);

    float thetaDot_rps = 0 + (cosf(phi_rad) * gyroDataY_RPS)
            - (sinf(phi_rad) * gyroDataZ_RPS);

    // estimate the angles by fusing the readings.
    phi_rad = COMP_FILTER_ALPHA * phiHat_acc_rad
            + (1 - COMP_FILTER_ALPHA) * (phi_rad + phiDot_rps);

    theta_rad = COMP_FILTER_ALPHA * thetaHat_acc_rad
            + (1 - COMP_FILTER_ALPHA) * (theta_rad + thetaDot_rps); // previous ->
(SAMPLE_TIME_MS / 1000.0f) * thetaDot_rps
```

Figure D-2 Code listing of Complimentary Filter Sensor Fusion algorithm

Appendix E: Getting altitude from pressure sensor

The altitude of any system can be estimated if the pressure exerting on the system and the surrounding temperature of the system is known.

Following graph shows how the pressure on the system and the altitude of the system are related[E-1] :

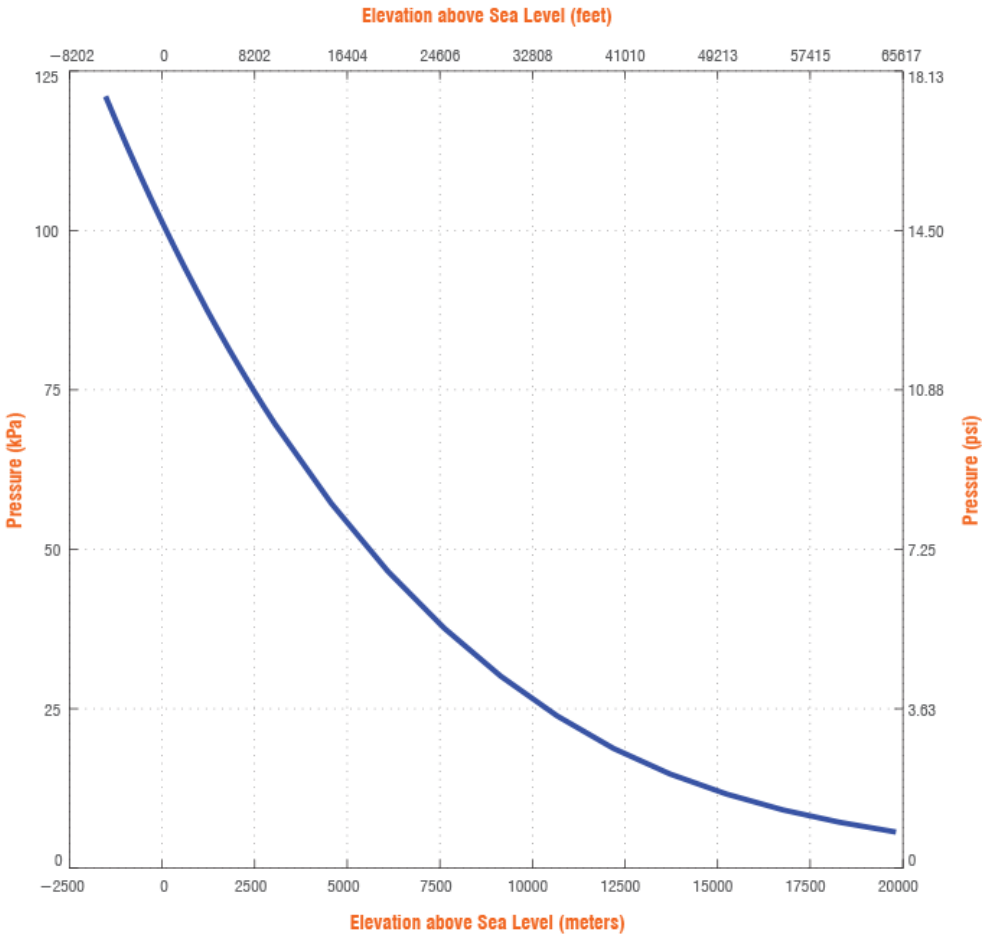


Figure E-1 Relationship between Pressure and Altitude

Hypsometric formula can be used to estimate the above relationship between pressure, temperature, and altitude.[E-2]

For this project, simplified version of this formula was used to estimate the altitude of the system by getting the pressure and the temperature of the surrounding.

$$h = \frac{T_b}{L_b} \cdot \left[\left(\frac{P}{P_b} \right)^{\frac{-R.Lb}{g.M}} - 1 \right]$$

where,

P_b= static pressure (pressure at sea level) [Pa]
T_b= standard temperature (temperature at sea level) [K]
L_b= standard temperature lapse rate [K/m] = -0.0065 [K/m]
h= height about sea level [m]
R= universal gas constant = 8.31432 $\left[\frac{N \cdot m}{mol \cdot K}\right]$
g_0= gravitational acceleration constant = 9.80665 $\left[\frac{m}{s^2}\right]$
M= molar mass of Earth's air = 0.0289644 [kg/mol]

When the standard values are substituted, the formula reduces to:

$$h = \frac{(T + 273.15)}{-0.0065} \cdot \left[\left(\frac{P}{1013} \right)^{0.1902632365} - 1 \right]$$

where

T = Temperature measured by the sensor (BMP388)

P = Pressure measured by the sensor (BMP388)

h = Altitude of the system

For this project, the above equation was used to determine the altitude of the quadcopter.