

California State University, Northridge

Quad Copter Flight

A thesis submitted in partial fulfillment of the requirements

For the degree of Master of Science in

Electrical Engineering

By

Peter O. Basta

May 2012

The graduate project of Peter O. Basta is approved:

---

Dr. Deborah van Alphen

---

Date

---

Dr. Somnath Chattopadhyay

---

Date

---

Dr. Xiaojun Geng, Chair

---

Date

California State University, Northridge

"You must be the change you wish to see in the world." -Gandhi

## TABLE OF CONTENTS

Signature Page .....	ii
Dedication.....	iii
List of Figures.....	vi
List of Tables .....	viii
Abstract.....	ix
Chapter 1. Introduction	
1.1 Purpose and Goals.....	1
1.2 History.....	2
1.3 Prospective Applications.....	4
Chapter 2. Quad Copter Dynamics and Theory	
2.1 Concepts of Quad Copter Flight .....	5
2.2 Dynamics of a Quad Rotor.....	7
2.3 Assumptions.....	9
Chapter 3. Control Theory	
3.1 Closed-Loop Control Systems .....	10
3.2 Classical Control Using PID Method.....	11
3.3 Control Tuning.....	13
3.31 Vertical Position Control .....	14
3.32 Horizontal Position Control .....	15
3.33 Roll and Pitch Position Control .....	15
3.34 Yaw Position Control.....	16
3.4 Rotor Theory.....	16
Chapter 4. Electrical Control Systems	
4.1 The Microcontroller - Arduino Mega 2560 .....	18
4.2 Electronic Speed Controllers .....	20
4.21 Turnigy Plush 18A ESC Modules.....	21
4.3 The Battery Pack.....	22
4.31 Zippy 4000 mAh Battery Pack Module .....	23
4.4 The Gyroscope Transducer .....	24
4.41 ITG3200 Gyro Module .....	24

4.5 The Accelerometer Sensor .....	26
4.51 BMA180 Accelerometer Module .....	26
4.6 The Barometer Transducer - Bosch BMP085 .....	28
4.61 Barometric Pressure in Altimetry .....	28
4.7 The Brushless Motors .....	29
4.71 KDA 20-22L Brushless Motors .....	30
4.8 Software Development General Concepts .....	32
4.81 Operating Modes of Flight.....	32
4.82 Position Flight Control.....	34
4.83 Kalman Filter Design.....	35
Chapter 5. Structural and Mechanical Approaches	
5.1 The Frame of the Quad Copter .....	37
5.2 Motor Alignment and Propeller Balancing .....	38
Chapter 6. Results and Conclusions	
6.1 Results.....	40
6.2 Conclusion .....	40
6.3 Future Work and Closure .....	41
References.....	42
Appendix A. Gyrometer Code Notes .....	44
Appendix B. Motor Code Notes .....	45
Appendix C. Accelerometer Code Notes .....	48
Appendix D. PID Motor Controller Code Notes .....	51

## LIST OF FIGURES

1.1 DeBothezats Quad-Rotor Design, 1922 .....	2
1.2 Bell Boeing Quad Tilt Rotor .....	4
2.1 Quad Copter Torque Pattern and Movement Generality .....	5
2.2 Quad Copter General Inertial Frame Coordinates. ....	6
3.1 Typical Closed Loop Control.....	10
3.2 Typical PID Control Logic. ....	12
3.3 Vertical Position Controller. ....	14
3.4 Horizontal Position Controller.....	15
3.5 Roll Position Controller. ....	16
3.6 Yaw Position Controller .....	16
4.1 Arduino Mega 2560 Microcontroller Board .....	18
4.2 Arduino Mega 2560 Microcontroller Block Diagram .....	19
4.3 Turnigy Plush18A ESC Packs .....	22
4.4 Zippy 4000 mAh Battery .....	23
4.5 ITG 3200 Gyro Board.....	25
4.6 ITG 3200 Gyro Functional Block Diagram. ....	25
4.7 BMA180 Accelerometer Block Diagram. ....	27
4.8 BMA180 Accelerometer Board .....	27
4.9 BMP085 Barometer Board. ....	28
4.10 BMP085 Barometer and Altitude Relationship. ....	29
4.11 KDA 20-22L Brushless Motor.....	30
4.12 Quad Copter Hardware Block Diagram.....	31
4.13 Quad Copter As Constructed .....	31
4.14 Software Architecture Organization. ....	32
4.15 Quad Copter Operating Modes Of Flight. ....	34
4.16 Kalman Filter Recursive Algorithm.....	36

5.1 Image and CAD Model of Aluminum Spar.....	38
5.2 Fiberglass Plates CAD Models.....	38
6.1 Quad Copter In Flight.....	40

## LIST OF TABLES

3.1 Values Set by Zeigler-Nichol Tuning Method.....	14
4.1 Turnigy Plush18A Specifications .....	21
4.2 Zippy 4000 mAh Battery Specifications Table.....	23
4.3 KDA 20-22L Brushless Motor Specifications .....	30



## ABSTRACT

### Quad Copter Flight

By

Peter O. Basta

Master of Science in Electrical Engineering

An unmanned aerial vehicle also known as UAV is an unpiloted aircraft which can either be remotely operated or flown autonomously based on pre-programmed flight plans. Usually these types of vehicles are used in military applications for missions that are too dangerous for manned aircraft. They are also used in a growing number of civil applications such as aerial photography and the transport of various goods.

Rotating wing (or helicopter) UAVs have the advantage above fixed wing UAVs in many ways; they are able to take off and land vertically, making it possible to hover at a fixed point. The design discussed in this report is based on the development of UAV quad rotor helicopter (Quad Copter), hardware, control system and flight dynamics.

The copter is built of electric motor driven rotors, a aluminum and fiberglass frame, an embedded on-board computer, power distribution system and various sensor units. The hardware platform utilized for the on board computer was a ATmega2560 microcontroller, with 54 Digital I/O pins, 256KB of Flash memory, 8KB of SRAM and 4KB of EEPROM with programming done predominately in C++ to express the control commands and overall system.

## **Chapter 1. Introduction**

### **1.1 Purpose and Goals**

Ever since I can remember, I have always shared a wonderful fascination for the concept of flying. This attraction grew from an early age on my first cross country plane trip looking out the window. As I grew my love for flight transitioned into building Model Airplanes, studying the dynamics and properties of lift and eventually and most recently piloting a small aircraft.

I joined the Quad copter graduate project team in early 2011 with the goal of stream lining the copter and assisting with the unmanned flight both in the realm of hardware and software. The importance of unmanned and VTOL crafts (Vertical Take-off and Landing) flight became crucial in its use in the wake of middle eastern revolts and insurgencies at the time.

The Quad Copter presented in this project, is a type of quad rotor helicopter or quadrocopter that is lifted and propelled by four rotors. Early designs in the 1920s and 1930s suffered from poor performance and lack of stability due to limited controls and system integrations. Today with advanced electronics, accurate sensors and control system technology these limitations are becoming more a thing of the past, technology today has allowed these systems to grow more and more appealing due to increased stability and payload capacities.

To overcome these obstacles, complex integration of various sensors would have to be incorporated to allow this configuration of flight to be successful, the quad copter.

The overall goals of this project are the following:

1. Achieve Autonomous Take off and Landing.
2. Streamline Mechanical Construction/ Weight.

3. Attain a workable design that can be improved over time.

The points presented above will be discussed thoroughly within this paper.

## 1.2 History<sup>[1]</sup>

The concept of a quad-rotor aircraft has existed since early in the 20th century. Throughout the 20th century not many unique rotor-craft designs have been developed. The earliest workable designs for a quad-rotor were developed by George DeBothezat and Etienne Oemichen.

The Oemichen's quad-rotor design is the earliest mention of a complete four-rotor hovering vehicle in past history. Oemichen's first design in 1920 failed in the initial attempt to become airborne, thereby requiring Oemichen to add additional lifting power and stability of a helium-filled balloon. After a number of recalculations and redesigns, Oemichen was able to come up with a design that actually was capable of lift off and even established world helicopter flight records of the time, remaining airborne for up to 14 minutes at a time by 1923.

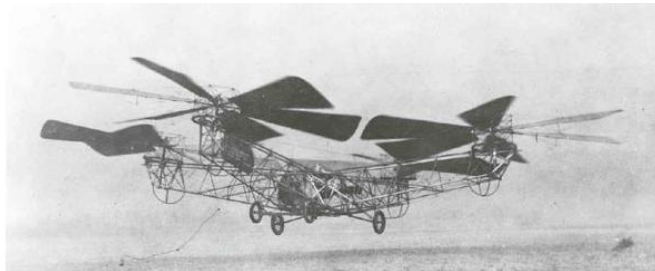


Figure 1.1 DeBothezats Quad-Rotor Design, 1922.

The DeBothezat's design was created for a 1921 contract with the United States Air Corps as seen in Figure 1.1. After working on his design for over 2 years, he was able to develop a fairly capable helicopter, which was able to take on a payload of up to 3 people in addition to the pilot. His design was deemed underpowered, unresponsive and susceptible to reliability issues. In

addition, instead of the calculated 100 meters cruising altitude, his craft was only capable of reaching a height of roughly 5 meters.

The early designs were propelled by additional rotors located somewhere on the rear or the front of the craft, perpendicular to the main rotors. Thus, they are not true quad-rotor designs. It was not until the mid-1950s that a true quad-rotor helicopter flew, which was designed by Marc Adam Kaplan. The prototype first flew in 1956, and did so with great success. The 2200 pounds craft was able to hover and maneuver using its two 90 horsepower motors, each capable of driving all four rotors in backup mode. Control in this case did not call for additional rotors on the sides of the craft, but was obtained by varying the thrust between rotors. This also was the first quad-rotor design that was able to fly successfully forward.

Despite these early proofs-of-concepts, people saw little practical use for quad-rotors. They simply were not competitive with the performance specifications (speed, payload, range, etc.) of more conventional aircrafts. No production contracts were awarded and interest in quad-rotors diminished.

### 1.3 Prospective Applications<sup>[1] [6]</sup>

Quad Copter designs are constantly progressing as the years go on. It wasn't too long ago that designs were limited and constrained, those days are history. The number of projects being undertaken regarding the topic has considerably increased, most of which are for commercial payload, human transport and military use.

Currently Bell Helicopter Textron and Boeing Integrated Defense Systems are doing joint research and development of the Bell Boeing Quad Tilt Rotor, as depicted in Figure 1.2. The initial design consists of four 50-foot rotors powered by V-22 engines. The main role of the Bell Boeing Quad Tilt Rotor will be that of a cargo helicopter with the ability to deliver pallets of supplies or also deploy paratroopers. The first wind tunnel tests were completed in 2006 and the first prototype is expected to be built in 2012, with great anticipation.

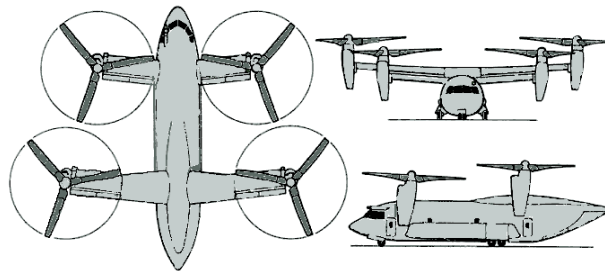


Figure 1.2 Bell Boeing Quad Tilt Rotor.

## Chapter 2. Quad Copter Dynamics and Theory

### 2.1 Concepts of Quad Copter Flight<sup>[3] [4] [5]</sup>

Copters are one of the most complex flying machines due to versatility and maneuverability to perform a number of tasks. Classical helicopters are usually equipped with a main rotor and a tail rotor. However the UAV (Unmanned Air Vehicle) presented in this paper is known as a quad copter.

Quad rotors are symmetrical vehicles with four equally sized rotors at the end of four equal length rods. By making use of multiple rotors it allows for greater thrust and maneuverability. Each of the rotors on the quad-rotor helicopter produces both thrust and torque. Given that the front and rear motors both rotate counter-clockwise and the other two rotate clockwise, resulting in a net torque of zero due to the rotational axis. Lastly, the quad rotors symmetrical design allows for easier control of the overall stability of the aircraft.

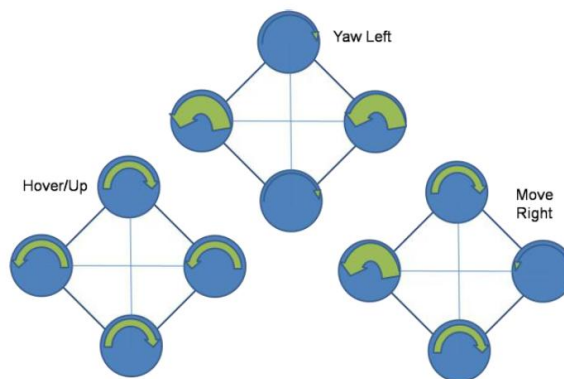


Figure 2.1 Quad Copter Torque Pattern and Movement Generality.

The principle for maintaining an equal rate of change for the two opposing rotors is how the translation of the craft is determined, as shown in Figure 2.1. Due to either a pitch or a roll, the lift force is displaced in the x and y axes, resulting in a horizontal force component that will direct the craft. The altitude of the quad-rotor is altered by changing the rate of all rotors by the same amount.

The UAV representation is obtained by representing the quad copter as a solid body evolving in 3D to one force with three moments (Figure 2.2). The general coordinates of this craft are:

$$q=(x, y, z, \theta, \phi, \psi) \quad [2.1]$$

Where ( x, y, z) denotes the position of the center of the mass of the craft in relation to the frame base and ( $\theta, \phi, \psi$ ) denotes the three angles yaw, pitch and roll (Euler angles) and represent the orientation of the copter.

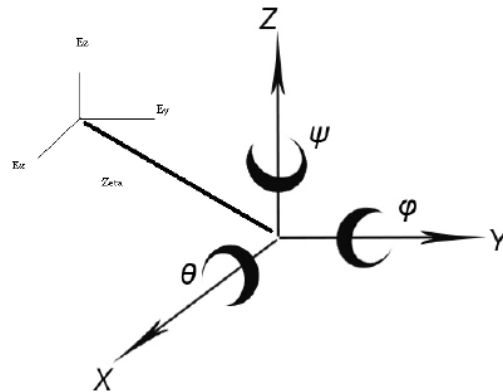


Figure 2.2 Quad Copter General Inertial Frame Coordinates.

## 2.2 Dynamics of a Quad Rotor

The kinematic relations relate the movements and rotations in the earth-fixed inertial reference to the body-fixed reference frame.

The derivatives with respect to time of the angles  $(\psi, \theta, \phi)$  can be expressed in the form:

$$[\dot{\psi} \ \dot{\theta} \ \dot{\phi}]^T = N(\psi, \theta, \phi) \omega \quad [2.2]$$

in which  $\omega = [p \ q \ r]^T$  are the angular velocities with respect to the body reference frame and

$N(\psi, \theta, \phi)$  is the 3x3 matrix given by:

$$N(\psi, \theta, \phi) = \begin{bmatrix} 0 & \sin \phi \sec \theta & \cos \phi \sec \theta \\ 0 & \cos \phi & -\sin \phi \\ 1 & \sin \phi \tan \theta & \cos \phi \tan \theta \end{bmatrix} \quad [2.3]$$

This matrix depends only on  $(\psi, \theta, \phi)$  and is invertible if the boundaries on  $(\psi, \theta, \phi)$  for non singularity hold.

Similarly, the derivative with respect to time of the position  $(x, y, z)$  is given by:

$$[\dot{x} \ \dot{y} \ \dot{z}]^T = V_0 \quad [2.4]$$

where  $V_0 = [u_0 \ v_0 \ w_0]^T$  is the absolute velocity of the quad rotor with respect to an earth-fixed inertial reference frame. Let  $V = [u \ v \ w]^T$  be the absolute velocity of the quad rotor expressed in a body-fixed reference frame.  $V$  and  $V_0$  are related by:

$$V_0 = R(\psi, \theta, \phi) V \quad [2.5]$$



where  $R(\psi, \theta, \phi)$  is the rotation matrix given by:

$$R(\psi, \theta, \phi) = \begin{bmatrix} (\cos \theta \cos \psi) & (\sin \phi \sin \theta \cos \psi - \cos \phi \sin \psi) & (\cos \phi \sin \theta \cos \psi + \sin \phi \sin \psi) \\ (\cos \theta \sin \psi) & (\sin \phi \sin \theta \sin \psi + \cos \phi \cos \psi) & (\cos \phi \sin \theta \sin \psi - \sin \phi \cos \psi) \\ (-\sin \theta) & (\sin \phi \cos \theta) & (\cos \phi \cos \theta) \end{bmatrix} \quad [2.6]$$

After the rotation matrix has been applied, the translational equations of motion with respect to an inertial frame are given by:

$$\begin{aligned} m(\ddot{x}) &= -u \sin(\theta) \\ m(\ddot{y}) &= u \cos(\theta) \sin(\phi) \\ m(\ddot{z}) &= u \cos(\theta) \cos(\phi) - mg \end{aligned} \quad [2.7]$$

Following the Euler-Lagrange equations, the rotational dynamics include torques and Coriolis terms. The coriolis term,  $C(\eta, \dot{\eta}) \dot{\eta}$ , defines the gyroscopic effects on the system when the craft yaws.  $\tau$  represents the vector of torques applied to the system.

$$\begin{aligned} \eta &= (\psi, \theta, \phi) \\ C(\eta, \dot{\eta}) \dot{\eta} &\triangleq J \dot{\eta} - \frac{\delta}{\delta \eta} (\dot{\eta}^T J \dot{\eta}) \\ J \ddot{\eta} + C(\eta, \dot{\eta}) \dot{\eta} &= \tau \\ \tau &= J \ddot{\tau} + C(\eta, \dot{\eta}) \dot{\eta} \end{aligned} \quad [2.8]$$

It follows that

$$\ddot{\eta} = [\ddot{\tau} \psi, \ddot{\tau} \theta, \ddot{\tau} \phi]^T = \ddot{\tau} \quad [2.9]$$

Combined with [2.7] the translational and rotational dynamics can be summarized as

$$\begin{aligned} \ddot{\tau} \psi &= \ddot{\psi} \\ \ddot{\tau} \theta &= \ddot{\theta} \\ \ddot{\tau} \phi &= \ddot{\phi} \end{aligned} \quad [2.10]$$

Where  $x$  and  $y$  are the coordinates in the horizontal plane, and  $z$  is the vertical position and  $\psi, \theta, \phi$  are the yaw, pitch and roll angles respectively.

### **2.3 Assumptions**

It is not possible to create a model that conforms to reality completely. Some assumptions need to be made, in this model the following is assumed:

- The quad rotor structure is rigid and its deformation characteristics will be disregarded.
- The quad rotor structure is symmetrical and material inconsistencies will be disregarded.
- The propellers are rigid and deflections will be ignored.
- The cross products of the inertia matrix can be neglected.
- The ground effect is neglected.

## Chapter 3. Control Theory

### 3.1 Closed-Loop Control Systems

In closed-loop control systems the difference between the actual output and the desired output is fed back to the controller to meet desired system output. Often this difference, known as the error signal is amplified and fed into the controller. The general structure of a closed-loop feedback control system is seen in Figure 3.1. A few examples of feedback control systems are elevators, thermostats, and cruise control in automobiles.

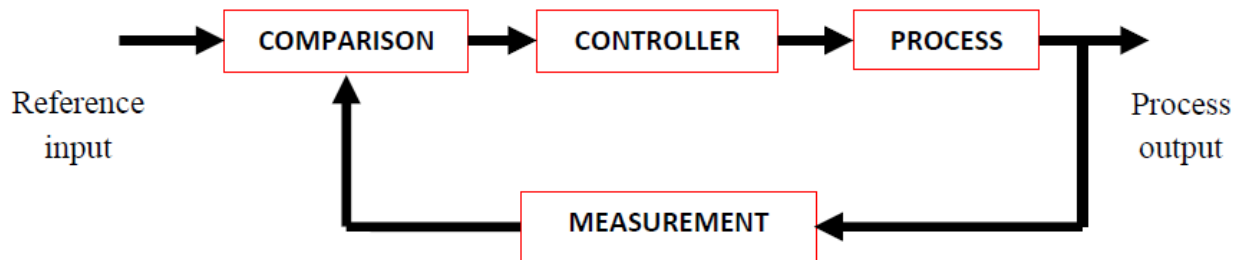


Figure 3.1 Typical Closed Loop Control.

### 3.2 Classical Control using PID Method <sup>[2] [15]</sup>

The PID (Proportional Integral Derivative) control is one of the earlier control methods implemented. Early on its execution was in pneumatic devices in the 1940's, followed by vacuum and solid state analog electronics, before arriving at today's digital use of microprocessors. It offered a simple control structure that was understood by operators and was relatively easy to work with.

The quad rotor will use a PID system, which will be tuned to determine the optimum response and settling time, illustrated in Figure 3.2. The PID controller equation is a closed-loop feedback system which will output a control signal  $u$  and receive feedback from the sensors. The controller will calculate the difference between the desired position and current position, adjusting  $u$  accordingly. The equation for a PID controller is as follows:

$$u = P + I + D \quad [3.1]$$

$$e(t) = e_d(t) - e_a(t)$$

Where  $e_d$  denotes the desired condition,  $e_a$  the actual condition, and  $e(t)$  denotes the actual difference, error, the two at each individual time step.

A PID controller has proportional, integral and derivative terms that can be represented in transfer function form as

$$K(s) = K_p + \frac{K_i}{s} + K_d s \quad [3.2]$$

Where  $K_p$  represents the proportional gain,  $K_i$  represents the integral gain, and  $K_d$  represents the derivative gain, respectively. By tuning these PID controller gains, the controller can provide control actions designed for specific process requirements.

The integral term  $K_i$  is proportional to both, magnitude of the error and the duration of the error.

It (when added to the proportional term) accelerates the movement of the process towards the set

point and often eliminates the remaining steady-state error that may occur with a proportional only controller.

The rate of change of the process error is calculated by determining the differential slope of the error over time (i.e., its first derivative with respect to time). This rate of change in the error is multiplied by the derivative gain  $K_d$ .

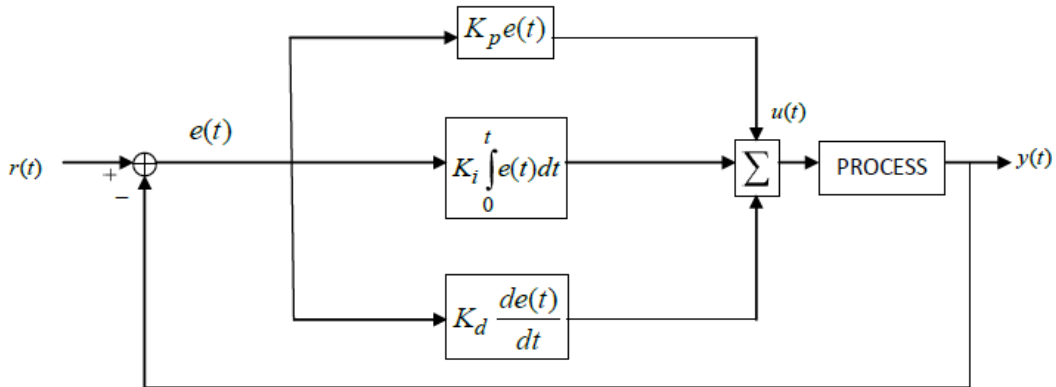


Figure 3.2 Typical PID Control Logic.

Feed-forward control plus feedback control can considerably improve performance over a simple feedback control done whenever there is a major disturbance affecting the system. In idyllic situations, feed-forward control can entirely neutralize the effect of the measured disturbance. Feed-forward control is used along with feedback control as necessary to track set point changes and to curb unmeasured disturbances that are always present in any real process.

For this project, feed-forward control accounts for the behavioral dynamics of the quad copter, such as its momentum and motor response time. The integral term determines the magnitude of the accumulated error by summing the instantaneous error over time. The integral control equation is:

$$I = K_i \int_0^t e(\tau) dt \quad [3.3]$$

Note that  $t$  is replaced with  $\tau$ , which denotes the past time.

The derivative term accounts for the rate at which the error is varying. In addition, by decreasing the rate of change close to the set point reduces overshoot and increases settling time.

$$D = K_d \frac{de}{dt} \quad [3.4]$$

$$P = K_p e(t) \quad [3.5]$$

### 3.3 Control Tuning<sup>[16]</sup>

For tuning of the PID controllers, several tuning algorithms have been developed, such as Ziegler-Nichols and Lambda tuning. The Zeigler-Nichols tuning method is a heuristic method of tuning a PID controller. It was developed by John G. Zeigler and Nathaniel B. Nichols. This tuning process is the response of a plant to unit-step input. If the response results in an S-shaped curve, both time constant and delay time are determined by a differential slope line at the inflection point of the curve.

When iterative tuning is used, the following algorithm can be used to get the desired overall response:

1. Investigate the step response.
2. Add gain  $K_p$  to reduce the rise time  $t_r$  if necessary.
3. Add gain  $K_D$  to improve the maximum overshoot  $M_p$  if necessary.
4. Add gain  $K_I$  to eliminate steady state error  $e_{ss}$  if necessary.
5. Adjust gains till desired overall response is obtained.

The transfer function of this method equals:

$$\frac{c(s)}{u(s)} = \frac{K e^{-Ls}}{Ts+1} \quad [3.6]$$

The values of  $K_p$ ,  $T_i$ , and  $T_d$  are determined by the factors in Table 3.1. This method demonstrates improvement in contrast to traditional manual tuning, but still is difficult to set exact values to critical gain.

Table 3.1 Values Set by Zeigler-Nichol Tuning Method.

Type of Controller	$K_p$	$T_i$	$T_d$
P	T/L	$\infty$	0
PI	0.9 T/L	L/0.3	0
PID	1.2 T/L	2 L	0.5 L

### 3.31 Vertical Position Control

When operating above the ground effect, the control output  $U_1$  is approximately proportional to the vertical acceleration in the body reference frame. To remain at constant height, a large value of  $U_1$  is required to counteract gravity, additionally a PID controller is added to stabilize the motion in the  $z$  direction (Figure 3.3). The control law can be described by:

$$r_1 = -K_{zp}(z - z_d) - K_{zi} \int (z - z_d) \delta t - K_{zd}(\dot{z} - \dot{z}_d)$$

$$U_1 = \frac{r_1 + mg}{\cos \phi \cos \theta} \quad [3.7]$$

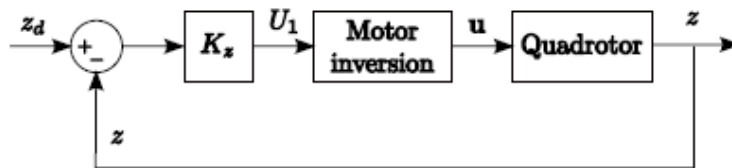


Figure 3.3 Vertical Position Controller [1].

### 3.32 Horizontal Position Control

The horizontal position  $(x, y)$  of the quad rotor is controlled by adjusting the roll and pitch. The total thrust produces a lift acceleration approximately equal to the gravity  $(g)$  in a direction normal to the plane of the rotors (Figure 3.4). A small angle of roll  $\phi$  results therefore in a lateral acceleration  $\ddot{y} \approx \phi g$ .

The commands in  $x$  and  $y$  direction are described by:

$$\begin{aligned} U_x &= -K_{xP}(x - x_d) - K_{xI} \int (x - x_d) \delta t - K_{xD}(\dot{x} - \dot{x}_d) \\ U_y &= -K_{yP}(y - y_d) - K_{yI} \int (y - y_d) \delta t - K_{yD}(\dot{y} - \dot{y}_d) \end{aligned} \quad [3.8]$$

Corrected for the yaw angle they give the commands for the desired roll and pitch angle:

$$\begin{aligned} \phi_d &= -\sin(\psi) U_x + \cos(\psi) U_y \\ \theta_d &= \cos(\psi) U_x + \sin(\psi) U_y \end{aligned} \quad [3.9]$$

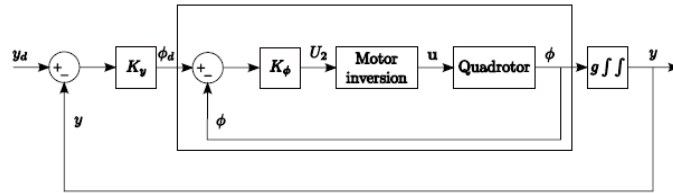


Figure 3.4 Horizontal Position Controller [1].

### 3.33 Roll and Pitch Position Control

Because the quad rotor's layout is symmetrical the roll and pitch control can be assumed independent for small attitude angles (Figure 3.5). Relative to the yaw control a higher bandwidth is required, since the value has a direct relationship to the lateral acceleration in  $x$  and  $y$  direction.

$$U_2 = -K_{\phi P}(\phi - \phi_d) - K_{\phi I} \int (\phi - \phi_d) \delta t - K_{\phi D}(\dot{\phi} - \dot{\phi}_d)$$



$$U_3 = -K_{\theta P}(\theta - \theta_d) - K_{\theta I} \int(\theta - \theta_d) \delta t - K_{\theta D}(\dot{\theta} - \dot{\theta}_d) \quad [3.10]$$

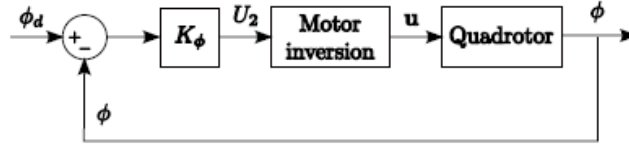


Figure 3.5 Roll Position Controller [1].

### 3.34 Yaw Position Control

The yaw is the least critical of the controls since it has no direct effect on the quad rotor's motion. It can be independently tested and tuned, while having manual control in the remaining channels (Figure 3.6). The PID control law for yaw is described by the following equation:

$$U_4 = -K_{\psi P}(\psi - \psi_d) - K_{\psi I} \int(\psi - \psi_d) \delta t - K_{\psi D}(\dot{\psi} - \dot{\psi}_d) \quad [3.11]$$

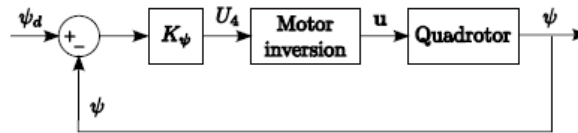


Figure 3.6 Yaw Position Controller [1].

### 3.4 Rotor Theory

Unlike other helicopters which require complex mechanical rotor mechanisms to control the pitch of the blades, a quad-rotor relies solely on differential torque and thrust, and thus uses fixed-pitch blades. One of the goals of the project is to increase total thrust; increasing thrust is accomplished by increasing the velocity of the air being moved by the rotor, or by increasing the amount of air being moved by the rotors.

$$T = \dot{m}(u_9 - u_0) \quad [3.12]$$

Where  $T$  is the thrust,  $\dot{m}$  is the mass airflow, and  $u_9$  and  $u_0$  are the outgoing and incoming velocities, respectively. From a simple physics perspective, using the kinetic energy equation:

$$K = .5mv^2 \quad [3.13]$$

Doubling the thrust by double  $m$  would lead to double the energy required. Doubling the thrust from doubling differential velocity would require four times the energy. The design of propellers is a complex subject, and as there were already dozens of readily available propellers on the market, it was both cost and time effective to select a commercial blade. The propellers are designed for radio control applications, and their airfoils are highly proprietary. The standard measure of propellers is diameter x pitch, where pitch refers to the angle of incidence at  $\frac{3}{4}$  of the radius. Using this angle, the pitch is converted to inches by how far the propeller would move after one revolution if it were “screwed” into a solid substance.

## Chapter 4. Electrical Control Systems

### 4.1 The Microcontroller - Arduino ATmega 2560<sup>[8]</sup>

The microcontroller we decided to utilize in our project was the Arduino Mega 2560, after careful review of several models. Considering that this single part is the heart of the project we selected with special care to characteristics like connectivity, available I/O and broad power integration.

The microcontroller offered all the features we needed to develop the quad copter, with the characteristics we desired. It is based on Atmel's ATmega 2560 microcontroller. The board comes with 54 digital input/output pins, 16 analog inputs, 4 UARTs (hardware serial ports), a 16 MHz crystal oscillator, a USB connector, and a power jack. The operating voltage of the microcontroller is 5V, with an input voltage of 7 to 12V.

The ATmega 2560 has a flash memory of 256 KB, of which 8 KB are reserved for the boot loader. 8 KB of SRAM, 4 KB of EEPROM, and a clock speed of 16 MHz. The Arduino Mega 2560 can be powered through a USB connection or with an external power supply.

Additionally, the Arduino platform was particularly attractive because of its open-source physical computing platform based on a simple i/o board and a development environment that implements the Processing/Wiring and language, seen below in Figure 4.1.



Figure 4.1 Arduino Mega 2560 Microcontroller Board.



## 4.2 Electronic Speed Controllers

An electronic speed control or ESC is a circuit with the purpose to control an electric motor's speed, its direction and possibly also to act as a dynamic brake in some cases. ESCs are often used on electrically powered brushless motors essentially providing an electronically-generated three phase electric power, with a low voltage source.

An ESC interprets control information in a way that varies the switching rate of a network of field effect transistors (FETs), not as mechanical motion as would be the case of a servo. The quick switching of the transistors is what causes the motor itself to emanate its characteristic high-pitched whine, which is especially noticeable at lower speeds. It also allows much smoother and more precise variation of motor speeds in a far more efficient manner than the mechanical type with a resistive coil and moving arm once in common use.

The ESC generally accepts a nominal 50 Hz Pulse Width Modulation (PWM) servo input signal whose pulse width varies from 1ms to 2ms. When supplied with a 1ms width pulse at 50 Hz, the ESC responds by turning off the DC motor attached to its output. A 1.5ms pulse-width input signal results in a 50% duty cycle output signal that drives the motor at approximately 50% speed. When presented with 2.0ms input signal, the motor runs at full speed due to the 100% duty cycle (on constantly) output.

The correct phase varies with the motor rotation, controlled and monitored by the ESC. The orientation of the motor is determined by the back EMF (Electromotive Force). The back EMF is the voltage induced in a motor wire by the magnet spinning past its internal coils. Finally, a PID algorithm in the controller adjusts the PWM to maintain a constant RPM.

Reversing the motor's direction may also be accomplished by switching any two of the three leads from the ESC to the motor.

#### 4.21 Turnigy Plush 18A ESC Modules<sup>[13][14]</sup>

The ESC controller chosen for this project was the Turnigy Plush 18A series, shown in Figure 4.3. Many considerations had to be made in making the appropriate selection. Ideally the ESC controller should be paired to the motor and rotor craft with the following considerations.

1. Temperature and thermal characteristics.
2. Max Current output and Impedence.
3. Needs to be Equipped with a BEC (Battery Eliminator Circuit) to eliminate the need of a second battery.
4. Size and Weight properties.
5. Magnet Rating.

The Plush series Turnigy ESCs offered several outstanding performance features that fulfilled our needs, fast sync timing, a generous current rating and a small size to weight footprint (Table 4.1).

Specifications as follows:

Table 4.1 Turnigy Plush18A Specifications.

Continuous Current Rating	18 Amp
Burst Current Rating	22 Amp
BEC Mode	Linear
BEC	5V/ 2 Amp
LiPo Cells	2-4 Cells
Weight	19 grams
Size	24x45x11mm

Additionally, the speed controller has fixed throttle settings so that the "stop" and "full throttle" points of all the various modes which can be cut through cleanly. The controller produces audible beeps to assist in navigating through the program modes and troubleshooting logs.



Figure 4.3 Turnigy Plush18A ESC Packs.

### 4.3 The Battery Pack

Selecting the proper battery for our rotor copter was a challenging task. Nickel Cadmium (NiCd), Nickel Metal Hydride (NiMH), and Lithium Polymer (LiPo) were common choices with the advantages and disadvantages of each battery pack.

NiCd batteries are reasonably inexpensive, but they have a number of negatives. NiCd batteries need to be fully discharged after each use. If they aren't, they will not discharge to their full potential (capacity) on following discharge cycles, causing the cell to develop what's commonly referred to as a memory. Additionally, the capacity per weight (energy density) of NiCd cells is commonly less than NiMH or LiPo cell types as well. Finally, the Cadmium that is used in the cell is quite destructive to the environment, making disposal of NiCd cells an issue.

NiMH cells have many advantages over their NiCd counterparts. NiMH cell manufacturers are able to offer significantly higher capacities in cells approximately the same size and weight of equivalent NiCd cells. NiMH cells have an advantage when it comes to cell memory as well, as they do not develop the same issues as a result of inappropriate discharge care.

Lithium Polymer (LiPo) cells are one of the newest and most revolutionary battery cells available. LiPo cells maintain a more consistent voltage over the discharge curve when compared to NiCd or NiMH cells. The higher nominal voltage of a single LiPo cell (3.7V vs 1.2V for a typically NiCd or NiMH cell), making it possible to have an equivalent or even higher total

nominal voltage in a much smaller package. LiPo cells typically offer very high capacity for their weight, delivering upwards of twice the capacity for ½ the weight of comparable NiMH cells. Lastly, a LiPo cell battery needs to be carefully monitored during charging since overcharging and the charging of a physically damaged or discharged cell can be a potential fire hazard and possibly even fatal.

LiPo Pro's:

- Highest power/weight ratio.
- Very low self-discharge.
- Less affected by low temperatures than some.

LiPo Con's:

- Intolerant of over-charging.
- Intolerant of over-discharging.
- Significant fire risk.



Figure 4.4 Zippy 4000 mAh

#### 4.31 Zippy 4000mAh Battery Pack Module<sup>[17]</sup>

Considering our copters weight, current load and predicted discharge rates, the Zippy 4000mAh battery pack seemed to fit the bill (Figure 4.4).

The Zippy 4000mAh battery pack offered the following characteristics seen in Table 4.2:

Table 4.2 Zippy 4000 mAh Battery Specifications Table.

Capacity	4000 mAh
Voltage	3 Cell 11.1 V
Discharge	20C Constant & 30C Burst
Weight	306 grams
Dimensions	146x51x22mm
Balance Plug	JST-XH
Discharge Plug	Bullet Connector
Max Charge Rate	2C



To approximately calculate the run time, the following equation was used:

$$\text{Run Time (Mins)} = \text{Battery mAh Rating} \frac{.06}{\text{Motor Load}} \quad [4.1]$$

Considering the average hovering current consumed by the motors is 12Amps per motor. We were able to calculate a min run time of 5 minutes and a max run time of 10 minutes, which was sufficient in our case, since we would be running at reduced speeds for testing with spare batteries.

#### **4.4 The Gyroscope Transducer**

A gyroscope is a device for measuring or maintaining orientation, based on the principles of angular momentum.

A MEMS gyroscope takes the idea of the Foucault pendulum and uses a vibrating element, known as a MEMS (Micro Electro-Mechanical System). These typically are packaged similarly to other integrated circuits and provide either analog or digital outputs. In addition, they are inexpensive and have become widely available today. They provide accurate 3 axis positioning of a craft and are highly reliable over the years with no internal moving parts.

##### **4.41 ITG3200 Gyro Module<sup>[11]</sup>**

The Gyro Sensor chosen for this project was the ITG 3200 by Invensense (Figure 4.5). The ITG-3200 featured a three 16-bit analog-to-digital converters (ADCs) for digitizing the gyro outputs, a user-selectable internal low-pass filter bandwidth, and a Fast-Mode I2C (400kHz) interface as shown in the block diagram below (Figure 4.6). In addition it included an embedded temperature sensor and a 2% accurate internal oscillator.

The ITG-3200 had a wide power supply range of anywhere between 2.1 and 3.6V with a low operating current of 6.5mA. The sensor also featured a interrupt output, and an optional clock input.

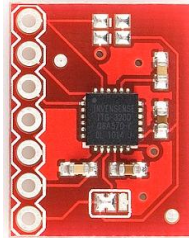


Figure 4.5 ITG 3200 Gyro Board.

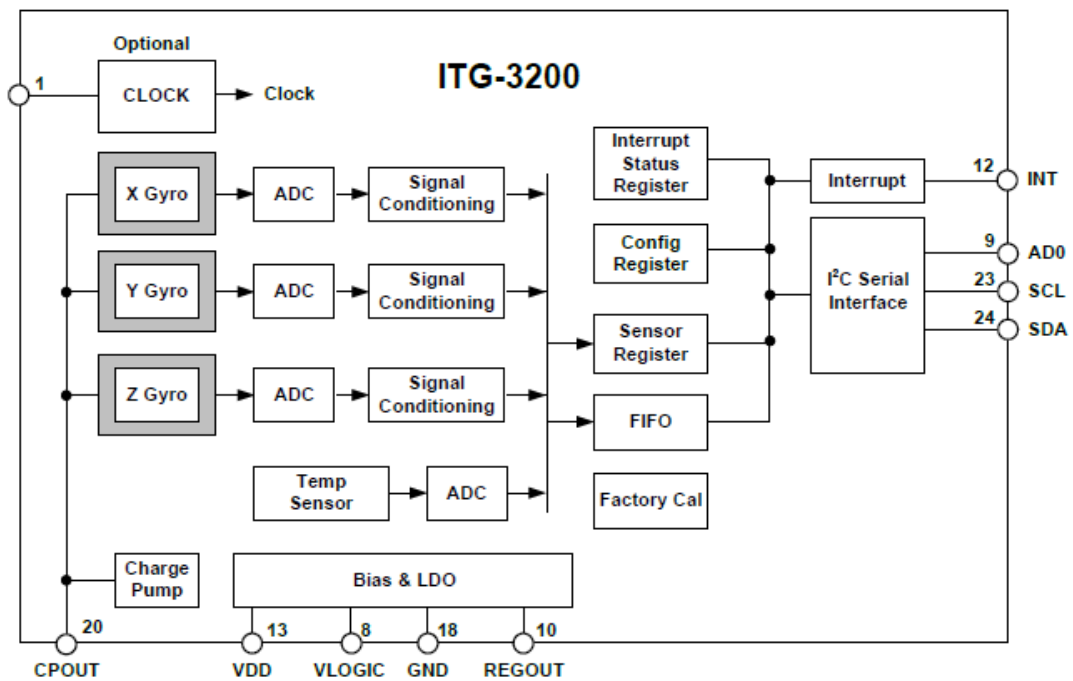


Figure 4.6 ITG 3200 Gyro Functional Block Diagram.

The ITG-3200 detects the rotational rate of the x, y, and z axes for roll, pitch, and yaw respectively. This is achieved through its three independent vibratory MEMS gyroscopes. When

the gyros are rotated about any of the sense axes, it causes a deflection that is detected by a capacitive pickoff.

#### **4.5 The Accelerometer Sensor**

An accelerometer is a device that measures proper acceleration by measuring weight per unit of mass, a quantity of force, or g-force (although it is not a force).

Consequently, by measuring weight, an accelerometer measures the acceleration of the free-fall reference frame (inertial reference frame) relative to itself (the accelerometer).

Single- and multi-axis models of accelerometers are available to detect magnitude and direction of the proper acceleration (or g-force), as a vector quantity, and can be used to sense orientation, coordinate acceleration (so long as it produces g-force or a change in g-force), vibration, shock, and falling (a case where the proper acceleration changes, since it tends toward zero). Micro machined accelerometers are increasingly present in modern electronic devices, to detect the position of the device or provide for the game system input.

Pairs of accelerometers extended over a region of space can be used to detect differences in the proper accelerations of frames of references associated with those points.

##### **4.51 BMA180 Accelerometer Module** <sup>[12]</sup>

The Accelerometer Sensor chosen for this project was the BMA180 by Bosch (Figure 4.8). The Bosch BMA180 three-axis, high performance digital accelerometer was chosen because of its superior variable characteristics. The BMA180 provides a digital 14-bit output signal via a 4-wire SPI or I2C interface. The full-scale measurement range can be set to  $\hat{A}\pm 1g$ , 1.5g, 2g, 3g, 4g, 8g or 16g.

The sensor also has two operating modes consisting of low-noise and low-power. The input power supply voltage should be between 1.62 and 3.6V for VDD and 1.2 to 3.6V for VDDIO. The sensor will typically only consume 650uA in standard mode.

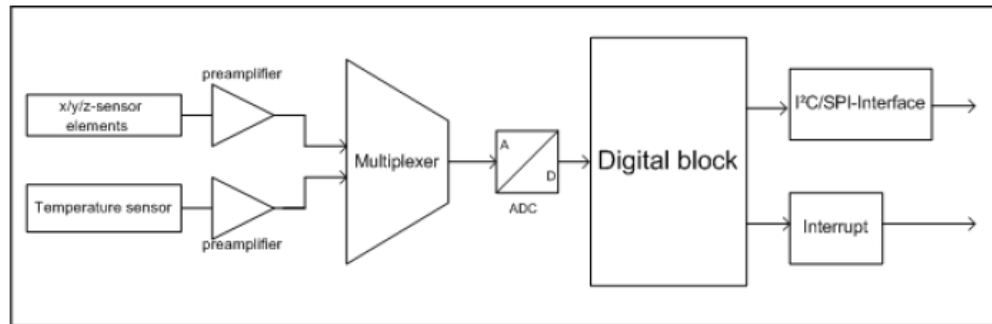


Figure 4.7 BMA180 Accelerometer Block Diagram.

As seen in Figure 4.7 the BMA180 diagram shows:

1. The micromechanical g-sensors elements which measure acceleration in the x,y and z directions.
2. The front end circuit including pre amplifiers and multiplexers.
3. The 14 bit Analog to digital controller.
4. The Interrupt generator and output interface.



Figure 4.8 BMA180 Accelerometer Board.

#### 4.6 The Barometer Transducer - Bosch BMP085<sup>[9]</sup>

A barometer is a scientific instrument used to measure atmospheric pressure (Figure 4.9). We chose Bosch's BMP085 Digital Barometric Pressure sensor for this project. The ultra-low power and low voltage electronics of the BMP085 are ideal for navigation devices. With a low altitude noise of merely 0.25m at fast conversion time, the BMP085 offers superior performance for an application such as ours. It comes equipped with I2C capabilities for easy system integration with microcontrollers and offers a measuring range of 300 to 1100 hPa with an absolute accuracy of down to 0.03 hPa.

This sensor supports a voltage supply between 1.8 and 3.6VDC.



Figure 4.9 BMP085 Barometer Board.

#### 4.61 Barometric Pressure in Altimetry

Barometric pressure has a measurable relationship with altitude, meaning you can use the BMP085 to deduce how high the quad copter has climbed. At sea level the air pressure is on average 1013hPa. The measuring limits of the BMP085 should allow us to measure pressure at elevations anywhere between -1640 to about 29,000 ft above sea level.

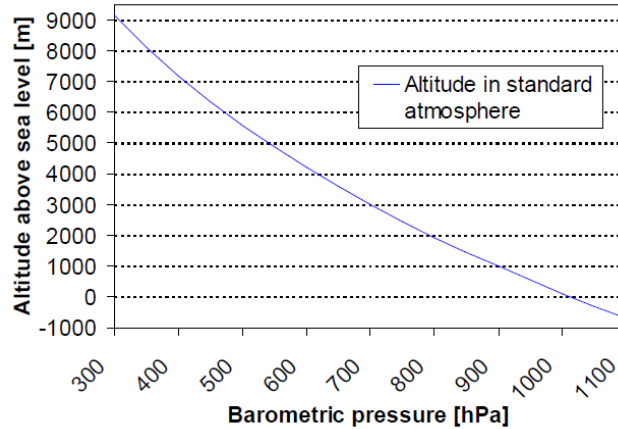


Figure 4.10 BMP085 Barometer and Altitude Relationship.

With the measured pressure  $p$  and the pressure at sea level  $p_0$ , the altitude in meters can be calculated with the international barometric formula.  $p_0$  is the average pressure at sea level (1013hPa), and  $p$  is the pressure that we measured. This relationship is observed in Figure 4.10.

Note that this equation gives you altitude in units of meters.

$$\text{Altitude (meters)} = 44330 \left(1 - \left(\frac{p}{p_0}\right)\right)^{1/5.255} \quad [4.2]$$

#### 4.7 The Brushless Motors

Each of the four rotors comprises of a Brushless DC Motor attached to a propeller. The Brushless motor differs from the conventional Brushed DC Motors in their concept essentially in that the commutation of the input voltage applied to the armature's circuit is done electronically, whereas in the latter, by a mechanical brush. As any rotating mechanical device, it suffers wear during operation, and as a consequence it has a shorter nominal life time than the newer Brushless motors.

In spite of the extra complexity in its electronic switching circuit, the brushless design offers several advantages over its counterpart, to name a few: higher torque/weight ratio, less operational noise, longer lifetime, less generation of electromagnetic interference and much more

power per volume. Virtually limited only by its inherent heat generation, whose transfer to the outer environment usually occurs by conduction.

#### 4.71 KDA20-22L Brushless Motors<sup>[10]</sup>

The KDA Brushless motors were chosen for our copter because of their superior specifications (Table 4.3), weight, power output and availability (Figure 4.11) . Additionally, this motor offered a wide variety of securing provisions which was extra helpful in its mounting and use.

Table 4.3 KDA 20-22L Brushless Motor Specifications.

Kv (rpm/v)	924
Weight	56 grams
Max Current	17 Amps
Max Voltage	11V
Motor Length	32mm
Motor Diameter	28mm
Total Length	46mm



Figure 4.11 KDA 20-22L Brushless Motor.

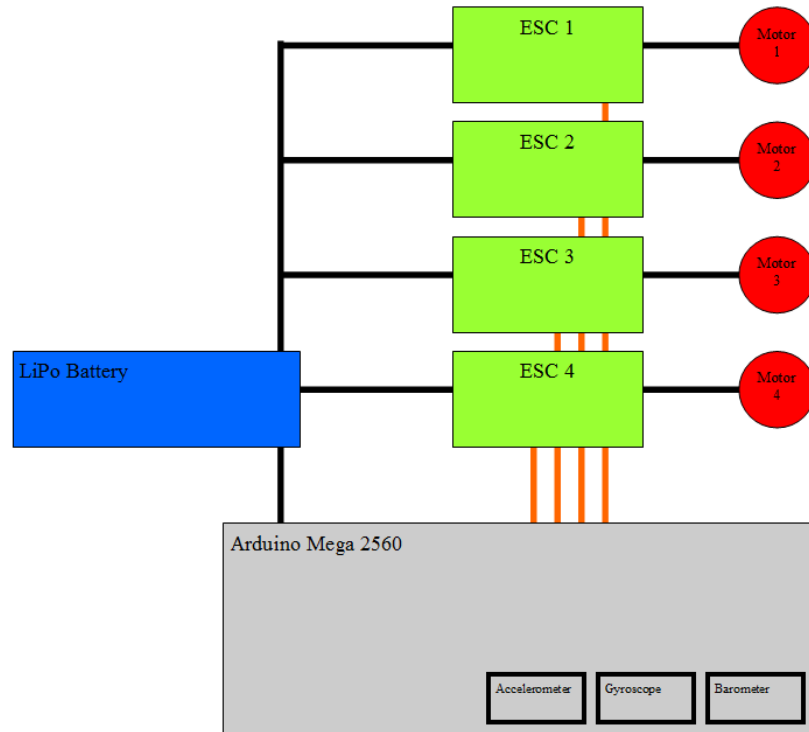


Figure 4.12 Quad Copter Hardware Block Diagram.



Figure 4.13 Quad Copter As Constructed.



## 4.8 Software Development General Concepts

All of the electrical hardware components previously mentioned provide the necessary ground work for the software subsystems to allow the copter to function as desired, as depicted in block diagram Figure 4.12. The software plays one of the most important roles controlling and facilitating the features of the device discussed in this paper. All of the inputs from the various sensors gyroscope, accelerometer and barometric sensor are to be taken and evaluated by the microcontroller to facilitate flight (Figure 4.13).

The goal of this project is to successfully achieve unmanned flight by simply hovering and landing. The quad copter must be able to take off from a stationary position to a hovering state and then land once again using the following organization illustrated in Figure 4.14.

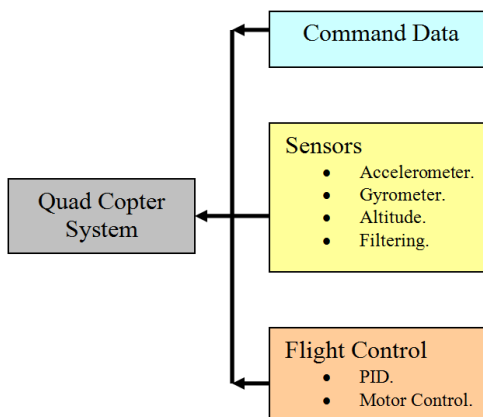


Figure 4.14 Software Architecture Organization.

### 4.81 Operating Modes of Flight

The quad copter has three main operating states known as, "take off", "hover" and "land". These states together form what is known as VTOL (Vertical Takeoff Landing) with characteristics shown in Figure 4.15.

The Take Off stage involves several procedures, which include motor/ESC initializations, data acquisition of the attached sensors and any human input. If after evaluating this data the quadcopter senses that it has not met the necessary criteria it will slowly ramp up the necessary motor thrust until it reaches the desired set point while the microcontroller is verifying the sensor data. When the actual altitude equals the desired position, the quadcopter can now go to hovering mode.

Hovering is the most challenging component of flying any copter. While in the hovering stage, the copter must take into account many factors that assist the craft in steady flight. Among the most common causes for a quadcopter to not perform sufficiently is drifting in the horizontal plane. Movements in all four directions such as forward, back, right, and left, are consequences of drifting, accumulation of small errors in various values leading to uncontrollable oscillations which have to be accounted for.

Landing is the final stage of flight, the landing process is the opposite of the take off phase. The system checks for x and y levels; if they are leveled, the altitude is checked. At this point, the acquired value is checked with the ground level. If the acquired values are not at ground level, thrust from all motors are decreased and decelerated until the ground level is reached, only then will the system halt activity.

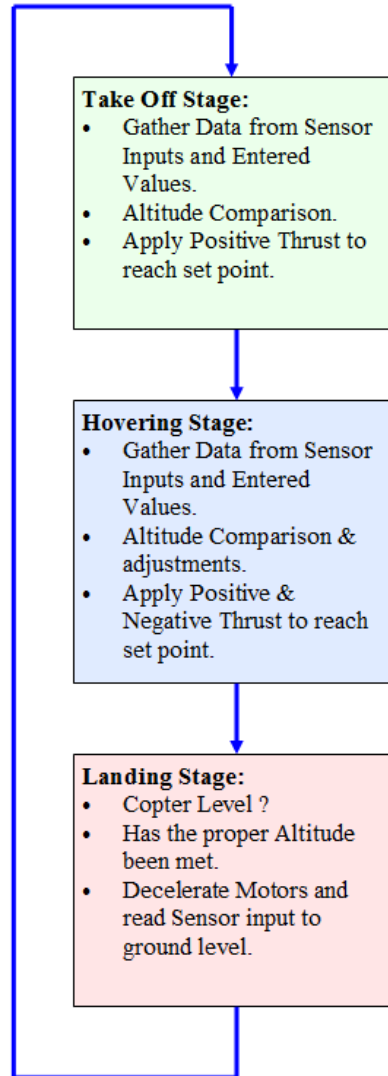


Figure 4.15 Quad Copter Operating Modes Of Flight.

#### 4.82 Position Flight Control

Position control is currently implemented using a PID controller design which actuates the vehicle's roll and pitch as control inputs. Tilting the vehicle in any direction causes a component of the thrust vector to point in that direction, so commanding pitch and roll is directly analogous to commanding accelerations in the X-Y plane. However, the current control implementation has little ability to reject

disturbances from wind and translational velocity effects. For this scale aircraft, even mild winds can cause large disturbances. A key weakness of this and similar position controllers used is the assumption that the velocity of the free stream and attitude control are decoupled. This is in fact only true for very small velocities.

#### 4.83 Kalman Filter Design<sup>[7]</sup>

A Kalman filter is used to reduce the error in tracking. The Kalman filter operates by predicting the next value from the current value and the previous value. This is then corrected once the next measurement is taken to be between the measurement and the predicted value. This reduces the influence of noise on the measurement of the position.

Over time errors in measurements tend to accumulate causing the sensors inefficiencies for long term measurements and thus they tend to drift. The ideal solution to the problem described above would be combining both, gyroscope and accelerometer data values resulting in the offset of the deficiencies of each with the strengths provided by the two sensors.

Kalman filtering is an iterative approach that requires two varying input values. At every iteration, the Kalman filter will change the variables in the linear model, so the output of the model will be closer to the second input (Figure 4.16). For the project, two inputs will consist of the gyroscope and accelerometer data, the model using the gyroscope data looks like:

$$\begin{aligned} P_{k+1} &= P_k(1 - K_k) \\ x_{k+1} &= x_k + K_k(Z_k - X_k) \\ K_k &= P_k(P_k + S)^{-1} \end{aligned} \tag{4.3}$$

Since individual applications are favored and computed, equations for the filter in one dimension are required. Equations [4.3] are reduced for the latter two dimensions. Where,  $K$  is the Kalman gain,  $P$

is the estimation error covariance,  $x$  is the filtered value,  $S$  is the sensor noise, and  $z$  is the measurement.

The process noise  $p$  is reduced gradually, when tuning the filter for optimal performance.

Concurrently, sensor noise  $S$  is increased until a satisfactory speed/noise level is achieved.

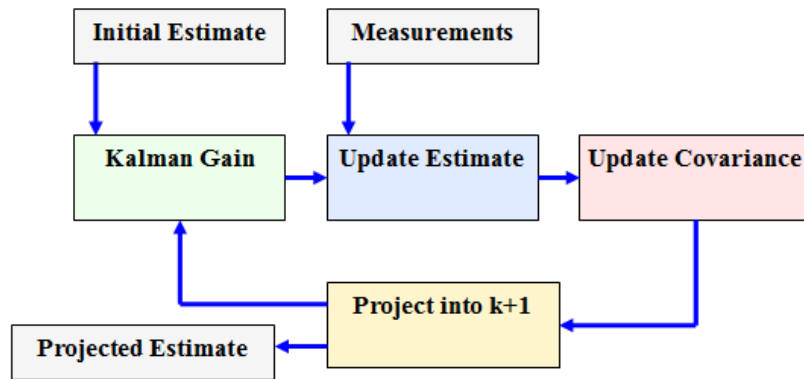


Figure 4.16 Kalman Filter Recursive Algorithm.

## Chapter 5. Structural and Mechanical Approaches

### 5.1 The Frame of the Quad Copter

Typical quad-rotors utilize a four-spar method, with each spar anchored to the central hub like the spokes on a wheel. The frame of the quad copter is composed of a combination of materials chosen for their strength, weight and flexibility.

When designing an autonomous quad-rotor, there are several material options which must be considered. When designing a machine capable of flight, weight must be greatly well thought-out. The materials considered for our project aluminum, plastic, and carbon fiber. Historically, aluminum historically, has been the material of choice for RC helicopters. Aluminum is light and strong, dissipates heat well, and is relatively inexpensive compared to the other available options. Finally, aluminum can develop cracks over time from vibrations.

Plastic absorbs vibration much better than the previously mentioned aluminum. Also, it is fairly durable and will retain its original shape if bent. Plastics are also very inexpensive and light but are difficult to attain in various shapes.

Consequently, we settled on a combinational design utilizing aluminum as well as fiberglass. The four spar rods are made of aluminum tubes of 8x12x1 mm and 325 mm in length shown in Figure 5.1. The rods are milled at the end to accommodate the motor, landing pieces and hardware. At the opposite end, holes were added for the fiberglass plate (~1mm thick) that would eventually house the microcontroller, battery and power distribution plate (Figure 5.2).

The total weight of the frame, without any electronic hardware was about 2.75 lb's and 4.5 lb's fully loaded.



Figure 5.1 Image and CAD Model of Aluminum Spar.

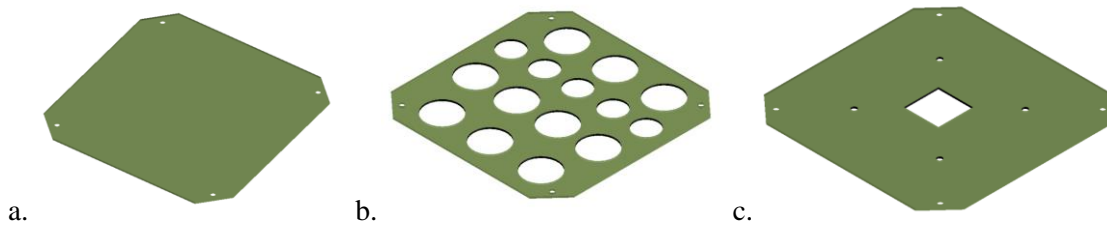


Figure 5.2 Fiberglass Plates CAD Models a. Electrical Hardware Plate b. Heat Dissipater c. Power Distribution plate.

## 5.2 Motor Alignment and Propeller Balancing

A successfully flying device must be perfectly balanced and thus have a properly aligned motor and balanced rotor. By setting the frame of the quad rotor on a flat surface, a T-square was held against the motor mounts. If the mounts were directly against the edge of the square, then the mount would be aligned in a 90 degree angle. With minor alteration the hardware could then be secured to position the motor correctly and square in proper alignment.

Propellers are supposed to operate at high revolutions (RPM) and its balance is also crucial to prevent damages to the design, all collateral/human injury.

Not all propellers are equal, neither do they have a perfect weight distribution, since they are manufactured in large quantities. We chose the Advanced Precision Composites Propeller, LP10047 and LP10047SFP because of availability, fitment and size preferences. To overcome propeller alignment, we rotated the propellers on a fastened spindle shaft by hand and watched for their final resting position. Ideally, blades will end up in a horizontal position, this implies that the propeller has perfect weight distribution.

Initially, we found that all four of our propellers were not balanced. The propeller always ended in a off position, implying that the bottom section of the blade was heavier. With a combination of adding varnish and sanding, we were able to balance all the propellers.



## Chapter 6. Results and Conclusions

### 6.1 Results

At low velocities and minute aerodynamic disturbances, the PID control implemented is sufficient for good tracking of commanded attitude and flight. In general, the controller has proven to be very effective in altitude control, though performance can be improved by enhanced filtering and possibly a more robust means of altitude than our barometer.



Figure 6.1 Quad Copter In Flight.

### 6.2 Conclusion

The common goal of this project was to develop an autonomous quad copter capable of flight and a unit that was sustainable and expandable for future works (Figure 6.1). Sustainability was achieved by using hardware with future expandability in mind and room to enlarge its capabilities, in addition to basic flight.

This project gave us a very extensive view of how different systems all work with one another to make a unified working system. The knowledge in different facets starting from the control

theory, sensor and part analysis, frame hardware and software all had to work hand in hand to implement the final product and succeed.

### **6.3 Future Work and Closure**

Quad rotor helicopters hold a rich future of possibilities in the UAV discipline. Their practical take-off and high payload capabilities make them especially important in the progress of future aircraft technology.

Among the improvements and additions to the project there could be the integration of a Global Positioning System (GPS) to enable autonomous navigation, better suited transducers and the incorporation of a camera system for possible neural network field detection.

## REFERENCES

- [1] Bouabdallah, S. & Siegwart, R. "Design and Control of a Miniature Quadrotor", Advances in Unmanned Aerial Vehicles (2007).
- [2] Gogoi, ManojBouabdallah, S. & Siegwart, R. " Proportional Integral Derivative (PID) Controller Design For Robust Stability of Arbitrary Order Plants with Time Delay and Uncertainty" , 2007.
- [3] Haugen, Fioffmann G. "Dynamics and Control", 2010.
- [4] Hoffmann, G. & Huang, H. "Quadrotor Helicopter Flight Dynamics and Control: Theory and Experiment", 2007.
- [5] Pounds, P. & Mahony, R. "Modeling and Control of a Quad-Rotor Robot", 2007.
- [6] Valavanis, Kimon P. "Advances in Unmanned Aerial Vehicles: State of the Art and the Road to Autonomy". Spring 2007.
- [7] Boyd, Stephen. "The Kalman Filter"  
Accessed: November 2011.  
<http://www.Stanford.edu/class/ee363/kf.pdf>
- [8] Arduino Mega2560.  
Accessed: December 2011.  
<http://www.atmel.com/Images/doc2549.pdf>
- [9] Bosch's BMP085 Datasheet.  
Accessed: December 2011.  
<http://www.bosch-sensortec.com/content/language1/downloads/BST-BMP085-DS000-06.pdf>
- [10] Hacker Style Brushless Outrunner 20-22L.  
Accessed: December 2011.  
[http://www.hobbyking.com/hobbyking/store/uh\\_viewitem.asp?idproduct=4700](http://www.hobbyking.com/hobbyking/store/uh_viewitem.asp?idproduct=4700)
- [11] ITG-3200.  
Accessed: December 2011.  
<http://invensense.com/mems/gyro/documents/PS-ITG-3200-00-01.4.pdf>
- [12] Triple Axis BMA180.  
Accessed: January 2012.  
<http://www.bosch-sensortec.com/content/language1/downloads/BST-BMA180-FL000-03.pdf>
- [13] Turnigy Manual for Brushless Motor Speed Controller.  
Accessed: January 2012.  
[http://www.svensktmodellflyg.se/users/4327/Turnigy\\_Plush\\_and\\_Sentry\\_ESC.pdf](http://www.svensktmodellflyg.se/users/4327/Turnigy_Plush_and_Sentry_ESC.pdf)

- [14] Turnigy Plush 18A Speed Controller.  
Accessed: January 2012.  
[http://www.hobbyking.com/hobbyking/store/uh\\_viewItem.asp?idProduct=4312](http://www.hobbyking.com/hobbyking/store/uh_viewItem.asp?idProduct=4312)
  
- [15] What is PID – Tutorial Overview. Date Accessed: 19 August 2011  
Accessed: January 2012.  
<http://www.expertune.com/tutor.html>
  
- [16] Zeigler-Nichols Rule. Date Accessed 22 August 2011  
Accessed: March 2012.  
<http://www.mstarlabs.com/control/znrule.html>
  
- [17] Zeigler-Nichols Rule. Date Accessed 22 August 2011  
Accessed: April 2012.  
[http://www.hobbyking.com/hobbyking/store/uh\\_viewItem.asp?idProduct=7634](http://www.hobbyking.com/hobbyking/store/uh_viewItem.asp?idProduct=7634)

## APPENDIX A. Gyrometer Code Notes

The following function calls are described:

- initialize() – initializes how the microcontroller reads measurements from the gyro
- measure() – performs sensor measurement from the gyro
- autoZero() – measures the A/D value that corresponds to a zero angular rate
- getFlightData(axis) – returns modified raw sensor data for use in control algorithms
- initialize(roll channel, pitch channel, Z axis channel)
- getRaw(axis) – returns the A/D value centered around zero.
- invert(axis) – execute once to invert the accelerometer axis
- getZero(axis) – returns the raw A/D value that defines zero output from the gyro

Code Fragment:

```
{
  gyroADC[XAXIS] = readShortI2C() - gyroZero[XAXIS];
  gyroADC[YAXIS] = gyroZero[YAXIS] - readShortI2C();
  gyroADC[ZAXIS] = gyroZero[ZAXIS] - readShortI2C();
}
```

```
measureSpecificGyroSum() {
```

```
  for (byte axis = XAXIS; axis <= ZAXIS; axis++) {
    gyroSample[axis] += readShortI2C();
  }
}
```

```
evaluateSpecificGyroRate(int *gyroADC) {
```

```
  gyroADC[XAXIS] = (gyroSample[XAXIS] / gyroSampleCount) - gyroZero[XAXIS];
  gyroADC[YAXIS] = gyroZero[YAXIS] - (gyroSample[YAXIS] / gyroSampleCount);
  gyroADC[ZAXIS] = gyroZero[ZAXIS] - (gyroSample[ZAXIS] / gyroSampleCount);
}
```

```
calibrateGyro() {
```

```
  int findZero[FINDZERO];

  for (byte axis = 0; axis < 3; axis++) {
    for (int i=0; i<FINDZERO; i++) {
      sendByteI2C(ITG3200_ADDRESS, (axis * 2) + ITG3200_MEMORY_ADDRESS);
      findZero[i] = readShortI2C(ITG3200_ADDRESS);
      delay(10);
    }
    gyroZero[axis] = findMedianInt(findZero, FINDZERO);
  }
} #endif
```

## APPENDIX B. Motor Code Notes

The following function calls are described:

- `initialize()` – initializes motor control
- `write()` – commands each motor to values set
- `commandAllMotors(value)` – command all motors to the same value
- `setRemoteCommand(motor, value)` – sets the value sent to motor over serial command
- `getRemoteCommand(motor)` – gets the value sent from a serial command
- `getMotorSlope()` – returns the slope ( $y=mx+b$ ) of equation used to convert PWM to PPM
- `getMotorOffset()` – returns offset ( $y=mx+b$ ) of equation used to convert PWM to PPM
- `setMinCommand(motor, value)` – sets the minimum allowable command to send to the motor
- `setMaxCommand(motor, value)` – sets the maximum allowable command to send to the motor
- `getMotorAxisCommand(motor)` – gets the motor command for the specified axis
- `setMotorCommand(motor, value)` – commands the actual motor command to send to each motor

Code Fragment:

```
#define MOTORPIN0 2
#define MOTORPIN1 3
#define MOTORPIN2 5
#define MOTORPIN3 6
#define MOTORPIN4 7
#define MOTORPIN5 8
#define MOTORPIN6 11
#define MOTORPIN7 12
#else
#define MOTORPIN0 3
#define MOTORPIN1 9
#define MOTORPIN2 10
#define MOTORPIN3 11
#define MOTORPIN4 5
#define MOTORPIN5 6

volatile uint8_t atomicPWM_PIN5_lowState = 0;
volatile uint8_t atomicPWM_PIN5_highState = 0;
volatile uint8_t atomicPWM_PIN6_lowState = 0;
volatile uint8_t atomicPWM_PIN6_highState = 0;

void initializeSoftPWM() {
    TCCR0A = 0; // normal counting mode
    TIMSK0 |= (1<<OCIE0A);
    TIMSK0 |= (1<<OCIE0B);
}
```

```

ISR(TIMERO_COMPA_vect) {
  static uint8_t state = 0;
  if (state == 0) {
    PORTD |= 1<<5; //digital PIN 5 high
    OCR0A+= atomicPWM_PIN5_highState; //250 x 4 microsecons = 1ms
    state = 1;
  } else if (state == 1) {
    OCR0A+= atomicPWM_PIN5_highState;
    state = 2;
  } else if (state == 2) {
    PORTD &= ~(1<<5); //digital PIN 5 low
    OCR0A+= atomicPWM_PIN5_lowState;
    state = 0;
  }
}

ISR(TIMERO_COMPB_vect) { //the same with digital PIN 6 and OCR0B counter
  static uint8_t state = 0;
  if (state == 0) {
    PORTD |= 1<<6;OCR0B+= atomicPWM_PIN6_highState;state = 1;
  } else if (state == 1) {
    OCR0B+= atomicPWM_PIN6_highState;state = 2;
  } else if (state == 2) {
    PORTD &= ~(1<<6);OCR0B+= atomicPWM_PIN6_lowState;state = 0;
  }
}

#endif

initializeMotors(NB_Motors numbers) {
  numberOfMotors = numbers;

  #else
  pinMode(MOTORPIN0, OUTPUT);
  pinMode(MOTORPIN1, OUTPUT);
  pinMode(MOTORPIN2, OUTPUT);
  pinMode(MOTORPIN3, OUTPUT);
  if (numbers == SIX_Motors) {
    pinMode(MOTORPIN4, OUTPUT);
    pinMode(MOTORPIN5, OUTPUT);
    initializeSoftPWM();
  }
  #endif

  commandAllMotors(1000);
}

```

```

writeMotors() {
  analogWrite(MOTORPIN0, motorCommand[MOTOR1] / 8);
  analogWrite(MOTORPIN1, motorCommand[MOTOR2] / 8);
  analogWrite(MOTORPIN2, motorCommand[MOTOR3] / 8);
  analogWrite(MOTORPIN3, motorCommand[MOTOR4] / 8);
  if (numberOfMotors == SIX_Motors) {
    analogWrite(MOTORPIN4, motorCommand[MOTOR5] / 8);
    analogWrite(MOTORPIN5, motorCommand[MOTOR6] / 8);
    #else
    atomicPWM_PIN5_highState = motorCommand[MOTOR6]/8;
    atomicPWM_PIN5_lowState = 255-atomicPWM_PIN5_highState;
    atomicPWM_PIN6_highState = motorCommand[MOTOR5]/8;
    atomicPWM_PIN6_lowState = 255-atomicPWM_PIN6_highState;
  #endif
}
  else if (numberOfMotors == EIGHT_Motors) {
    analogWrite(MOTORPIN4, motorCommand[MOTOR5] / 8);
    analogWrite(MOTORPIN5, motorCommand[MOTOR6] / 8);
    analogWrite(MOTORPIN6, motorCommand[MOTOR7] / 8);
    analogWrite(MOTORPIN7, motorCommand[MOTOR8] / 8);
  }
#endif
}

commandAllMotors(int command) {
  analogWrite(MOTORPIN0, command / 8);
  analogWrite(MOTORPIN1, command / 8);
  analogWrite(MOTORPIN2, command / 8);
  analogWrite(MOTORPIN3, command / 8);
  if (numberOfMotors == SIX_Motors) {
    analogWrite(MOTORPIN4, command / 8);
    analogWrite(MOTORPIN5, command / 8);
    #else
    atomicPWM_PIN5_highState = command/8;
    atomicPWM_PIN5_lowState = 255-atomicPWM_PIN5_highState;
    atomicPWM_PIN6_highState = command/8;
    atomicPWM_PIN6_lowState = 255-atomicPWM_PIN6_highState;
  #endif
}
  else if (numberOfMotors == EIGHT_Motors) {
    analogWrite(MOTORPIN4, command / 8);
    analogWrite(MOTORPIN5, command / 8);
    analogWrite(MOTORPIN6, command / 8);
    analogWrite(MOTORPIN7, command / 8);
  }
#endif
} #endif

```



## APPENDIX C. Accelerometer Code Notes

The following function calls are described:

- initialize() – initializes how the microcontroller reads measurements from the accelerometer
- measure() – performs sensor measurement from the accelerometer
- getFlightData(axis) – returns modified raw sensor data for use in control algorithms
- getRaw(axis) – returns the A/D value centered around zero
- invert(axis) – execute once to invert the accelerometer axis
- getZero(axis) – returns the raw A/D value that defines zero output from the accelerometer
- setZero(axis) – stores a new A/D value that defines zero output from the accelerometer
- getScaleFactor() – returns the scale factor used to convert A/D measurements to G
- angleRad(axis) – returns the angle in radians that is calculated for the desired axis
- angleDeg(axis) – returns the angle in degrees that is calculated for the desired axis

Code Fragment:

```
#ifdef BMA180_ADDRESS_ALTERNATE
#define BMA180_ADDRESS 0x41
#else
#define BMA180_ADDRESS 0x40
#endif
#define BMA180_IDENTITY 0x03
#define BMA180_RESET_REGISTER 0x10
#define BMA180_TRIGGER_RESET_VALUE 0xB6
#define BMA180_ENABLE_WRITE_CONTROL_REGISTER 0x0D
#define BMA180_CONTROL_REGISTER 0x10
#define BMA180_BW_TCS 0x20
#define BMA180_LOW_PASS_FILTER_REGISTER 0x20
#define BMA180_10HZ_LOW_PASS_FILTER_VALUE 0x0F
#define BMA180_1200HZ_LOW_PASS_FILTER_VALUE 0X7F
#define BMA180_OFFSET_REGISTER 0x35
#define BMA180_READ_ROLL_ADDRESS 0x02
#define BMA180_READ_PITCH_ADDRESS 0x04
#define BMA180_READ_YAW_ADDRESS 0x06
#define BMA180_BUFFER_SIZE 6

initializeAccel() {

    if (readWhoI2C(BMA180_ADDRESS) == BMA180_IDENTITY) {
        vehicleState |= ACCEL_DETECTED;
    }

    updateRegisterI2C(BMA180_ADDRESS, BMA180_RESET_REGISTER,
BMA180_TRIGGER_RESET_VALUE);
    delay(10);
}
```

```

    updateRegisterI2C(BMA180_ADDRESS, BMA180_ENABLE_WRITE_CONTROL_REGISTER,
BMA180_CONTROL_REGISTER);           //enable writing to control registers
    sendByteI2C(BMA180_ADDRESS, BMA180_BW_TCS);
    byte data = readByteI2C(BMA180_ADDRESS);
    updateRegisterI2C(BMA180_ADDRESS, BMA180_LOW_PASS_FILTER_REGISTER, data &
BMA180_1200HZ_LOW_PASS_FILTER_VALUE);
    sendByteI2C(BMA180_ADDRESS, BMA180_OFFSET_REGISTER);
    data = readByteI2C(BMA180_ADDRESS);
    data &= 0xF1;
    updateRegisterI2C(BMA180_ADDRESS, BMA180_OFFSET_REGISTER, data);
}

```

```

measureAccel() {

```

```

    sendByteI2C(BMA180_ADDRESS, BMA180_READ_ROLL_ADDRESS);
    Wire.requestFrom(BMA180_ADDRESS, BMA180_BUFFER_SIZE);

    for (byte axis = XAXIS; axis <= ZAXIS; axis++) {
        meterPerSecSec[axis] = (readReverseShortI2C() >> 2) * accelScaleFactor[axis] +
runTimeAccelBias[axis];
    }
}

```

```

measureAccelSum() {

```

```

    sendByteI2C(BMA180_ADDRESS, BMA180_READ_ROLL_ADDRESS);
    Wire.requestFrom(BMA180_ADDRESS, BMA180_BUFFER_SIZE);

    for (byte axis = XAXIS; axis <= ZAXIS; axis++) {
        accelSample[axis] += (readReverseShortI2C() >> 2);
    }
    accelSampleCount++;
}

```

```

evaluateMetersPerSec() {

```

```

    for (byte axis = XAXIS; axis <= ZAXIS; axis++) {
        meterPerSecSec[axis] = (accelSample[axis] / accelSampleCount) * accelScaleFactor[axis] +
runTimeAccelBias[axis];
        accelSample[axis] = 0;
    }
    accelSampleCount = 0;
}

```

```

void computeAccelBias() {

```

```

    for (int samples = 0; samples < SAMPLECOUNT; samples++) {
        measureAccelSum();
    }
}

```

```
    delayMicroseconds(2500);
}

for (byte axis = 0; axis < 3; axis++) {
    meterPerSecSec[axis] = (float(accelSample[axis])/SAMPLECOUNT) * accelScaleFactor[axis];
    accelSample[axis] = 0;
}
accelSampleCount = 0;

runTimeAccelBias[XAXIS] = -meterPerSecSec[XAXIS];
runTimeAccelBias[YAXIS] = -meterPerSecSec[YAXIS];
runTimeAccelBias[ZAXIS] = -9.8065 - meterPerSecSec[ZAXIS];

accelOneG = abs(meterPerSecSec[ZAXIS] + runTimeAccelBias[ZAXIS]);
}

#endif
```

## APPENDIX D. PID Motor Controller Code Notes

Code Fragment:

```
#define RATE_XAXIS_PID_IDX    0
#define RATE_YAXIS_PID_IDX    1
#define ZAXIS_PID_IDX        2
#define ATTITUDE_XAXIS_PID_IDX 3
#define ATTITUDE_YAXIS_PID_IDX 4
#define HEADING_HOLD_PID_IDX  5
#define ATTITUDE_GYRO_XAXIS_PID_IDX 6
#define ATTITUDE_GYRO_YAXIS_PID_IDX 7
#define ALTITUDE_HOLD_PID_IDX  8
#define ZDAMPENING_PID_IDX     9

PIDparameters->previousPIDTime = currentTime;
float error = targetPosition - currentPosition;

PIDparameters->integratedError += error * deltaPIDTime;
PIDparameters->integratedError = constrain(PIDparameters->integratedError, -PIDparameters->windupGuard, PIDparameters->windupGuard);
float dTerm = PIDparameters->D * (currentPosition - PIDparameters->lastPosition) /
(deltaPIDTime * 100);
PIDparameters->lastPosition = currentPosition;
return (PIDparameters->P * error) + (PIDparameters->I * (PIDparameters->integratedError)) +
dTerm;
}

zeroIntegralError() __attribute__((noinline));
zeroIntegralError() {
    for (byte axis = 0; axis <= ATTITUDE_YAXIS_PID_IDX; axis++) {
        PID[axis].integratedError = 0;
        PID[axis].previousPIDTime = currentTime;
    }
}

#endif
```