Design & Developing of a Quadrotor Drone

Prepared by:

Nazmul Haque
ID: 2011-1-55-012

Shahida Arobi Sarna
ID: 2011-1-55-010

Project Supervisor:

Dr. Md. Habibur Rahman
Professor
Dept. of Electrical and Electronic Engineering
University Of Dhaka

East West University
Department of Electronics & Communication Engineering

DECLARATION

We hereby declare that this Project is our original work. We also declare that no part of this work has been submitted elsewhere partially or fully for the award of any other degree or diploma. Any material reproduced in this project has been properly acknowledged. Requisite references are quoted to support my work.

Signature:

----------	----------
Nazmul Haque	Shahida Arobi Sarna
ID: 2011-1-55-012	ID: 2011-1-55010

Signature of Supervisor	Signature of Dept. Chairperson
----------	----------
Dr. Md. Habibur Rahman Professor Dept. of Electrical and Electronic Engineering University Of Dhaka	**Dr. M. Mofazzal Hossin** Professor & Chairperson Dept. of Electronics &Communication Engineering East West University

APPROVAL

The Project titled as “Design and developing of a microcontroller based Quadrotor Drone” has been submitted to the following respected members of the Board of Examiners of the Faculty of Engineering for partial fulfillment of the requirements for the degree of Bachelor of Science in Electronics & Telecommunications Engineering by the following students and has been accepted as satisfactory.

Nazmul Haque
ID: 2011-1-55012

Shahida Arobi Sarna
ID: 2011-1-55-010

Dr. Md. Habibur Rahman
Professor
Dept. of Electrical and Electronic Engineering
University Of Dhaka

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGEMENTS

Many people deserve our thanks for their help in completing this project. We would like to thank our department for giving us this chance to do this project. We want to express our thanks and deep appreciation to our advisor Dr. Md. Habibur Rahman as he has exhausted all his knowledge and time by following up our daily progress and encouraging advices and even by sharing on the troubles.

We would like to extend our thanks to the laboratory staff of ECE Dept. for their fast response and cooperation with us to get some materials we need for our case.

We have special acknowledgment for our group members for their understanding each other and hard working from the beginning up to the end.

Finally, we would like to thank the entire person who involve with this projects for their invaluable help and professionalism during this project.

ABSTRACT

All over the world drone is most important subject in robotics. Now a day drone are used for research, military and commercially. In these circumstances, we are trying to make a quadrotor drone. This quadrotor drone in RF remote controlled capable. The drone fly by four individual propeller system and those are controlled from ardiuno. On the air gyro calculate the directional change so drone could sable itself.

Chapter 1

INTRODUCTION

1.1 Drone

The fundamental difference between the terms "**drone**" and "**quadcopter**" is one of characterization – **drone** is the general term used for all unmanned aerial vehicles, though **quadcopter** identifies with a particular set of **drones** with four engines that make lift for vertical takeoff through their propellers

1.2 Quadrotor Drone

A Quadrotor Drone, also called a quadrotor helicopter or quadcopter or quadrotor, is a multirotor helicopter that is lifted and propelled by four rotors. Quadcopters are classified as rotorcraft, as opposed to fixed-wing aircraft, because their lift is generated by a set of rotors (vertically oriented propellers).

Quadcopters generally use two pairs of identical fixed pitched propellers; two clockwise (CW) and two counter-clockwise (CCW). These use independent variation of the speed of each rotor to achieve control. By changing the speed of each rotor it is possible to specifically generate a desired total thrust; to locate for the centre of thrust both laterally and longitudinally; and to create a desired total torque, or turning force.

Quadcopters differ from conventional helicopters which use rotors which are able to vary the pitch of their blades dynamically as they move around the rotor hub. In the early days of flight, quadcopters (then referred to as 'quadrotors') were seen as possible solutions to some of the persistent problems in vertical flight; torque-induced control issues (as well as efficiency issues originating from the tail rotor, which generates no useful lift) can be eliminated by counter-rotation and the relatively short blades are much easier to construct

1.3 Applications

Quadcopters are a useful tool for university researchers to test and evaluate new ideas in a number of different fields, including flight control theory, navigation, real time systems, and robotics. In recent years many universities have shown quadcopters performing increasingly complex aerial maneuvers. Swarms of quadcopters can hover in mid-air. There are numerous advantages to using quadcopters as versatile test platforms. They are relatively cheap, available in a variety of sizes and their simple mechanical design means that they can be built and maintained by amateurs. Due to the multi-disciplinary nature of operating a quadcopter, academics from a number of fields need to work together in order to make significant improvements to the way quadcopters perform. Quadcopter unmanned

aerial vehicles are used for surveillance and reconnaissance by military and law enforcement agencies, as well as search and rescue missions in urban environments. One such example is the Aeryon Scout, created by Canadian company Aeryon Labs, which is a small UAV that can quietly hover in place and use a camera to observe people and objects on the ground. The largest use of quadcopters in the USA has been in the field of aerial imagery. Quadcopter UAVs are suitable for this job because of their autonomous nature and huge cost savings. In the USA, the legality of the use of remotely controlled aircraft for commercial purposes has been a matter of debate. Quadcopter projects are typically collaborations between computer science, electrical engineering and mechanical engineering specialists.

1.4 Aim of the Project

- To design a quadrotor drone
- Test for its functionality
- To design the control system with low cost components

Chapter 2

Throry of Quadrotor Drone

2.1 Flight dynamics

Each rotor produces both a thrust and torque about its center of rotation, as well as a drag force opposite to the vehicle's direction of flight. If all rotors are spinning at the same angular velocity, with rotors one and three rotating clockwise and rotors two and four counterclockwise, the net aerodynamic torque, and hence the angular acceleration about the yaw axis, is exactly zero, which implies that the yaw stabilizing rotor of conventional helicopters is not needed. Yaw is induced by mismatching the balance in aerodynamic torques (i.e., by offsetting the cumulative thrust commands between the counter-rotating blade pairs).

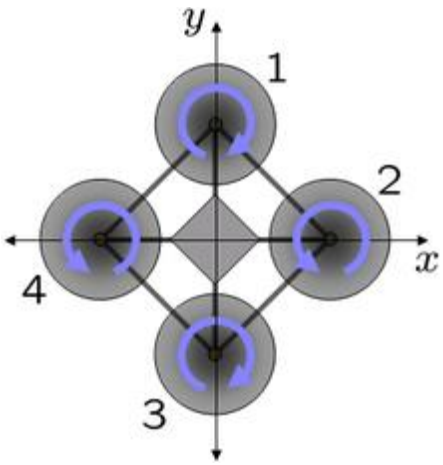


Figure 2.1 rotation

Fig: 1 Schematic of reaction torques on each motor of a quadcopter aircraft, due to spinning rotors. Rotors 1 and 3 spin in one direction, while rotors 2 and 4 spin in the opposite direction, yielding opposing torques for control.

A quadrotor hovers or adjusts its altitude by applying equal thrust to all four rotors.

Figure 2. 2 equal thrust

Figure 2.3 yaw

A quadrotor adjusts its yaw by applying more thrust to rotors rotating in one direction.

Figure 2.4 pitch or roll

A quadrotor adjusts its pitch or roll by applying more thrust to one rotor and less thrust to its diametrically opposite rotor.

2.2 Vortex ring state

Figure 2.5 Vortex ring state

Small quadcopters are subject to normal rotorcraft aerodynamics, including vortex ring state. The vortex ring state, also known as settling with power, it is common to all rotorcraft. It's encountered when a rotorcraft descends vertically too quickly. From this point on, let's assume that the rotorcraft is a quadcopter. The quadcopter's propeller blades may descend into the turbulent downwash beneath the craft. If this occurs, the blades lose some lift, causing an even faster descent into the downwash. A vortex also starts to form in a circular ring (the Vortex Ring) around the blade's path of rotation. This vortex sucks turbulent air from beneath the blades to the top of the blades. Applying throttle just increases the vortex ring, eventually causing total loss of lift from the blades.

2.3 Mechanical structure

The main mechanical components needed for construction are the frame, propellers (either fixed-pitch or variable-pitch), and the electric motors. For best performance and simplest control algorithms, the motors and propellers should be placed equidistant. Recently, carbon fiber composites have become popular due to their light weight and structural stiffness. The electrical components needed to construct a working quadcopter are similar to those needed for a modern RC helicopter. They are the electronic speed control module, on-board computer or controller board, and battery. Typically, a hobby transmitter is also used to allow for human input.

2.4 Autonomous flight

Quadcopters and other multicopters often can fly autonomously. Many modern flight controllers use software that allows the user to mark "way-points" on a map, to which the quadcopter will fly and perform tasks, such as landing or gaining altitude. Other flight applications include crowd control between several quadcopters where visual data from the device is used to predict where the crowd will move next and in turn direct the quadcopter to the next corresponding waypoint.

Chapter 3

THEORY BEHIND THE PROJECT

3.1 component list

1. Arduino UNO R3 Board
2. L3G4200D 3-axis gyro
3. SparkFun Logic Level Converter
4. 2.4G Fly-Sky FS- CT6B 6-Channel Transmitter+Receiver
5. LiPo 3-cell (30c) battery
6. 20A ESC
7. 14000kv brushless motor

3.2 Arduino UNO R3 Board

There are many varieties of Arduino boards that can be used for different purposes. The Arduino UNO components are:

Figure 3.1 Arduino UNO R3 Board

3.2.1 Power -USB / Barrel Jack

Our Arduino board needs a way to be connected to a power source. The Arduino UNO can be powered from a USB cable coming from your computer or a wall power supply that is terminated in a barrel jack. In the picture above the USB connection is labeled and the barrel jack is labeled .The USB connection is also how you will load code onto your Arduino board.

3.2.2 Pins (5V, 3.3V, GND, Analog, Digital, PWM, AREF)

The pins of Arduino are the places where connect wires to construct a circuit. The Arduino has several different kinds of pins, each of which is labeled on the board and used for different functions.

- **5V**: The 5V pin supplies 5 volts of power. Most of the simple components used with the Arduino run happily off of 5 or 3.3 volts.
- **GND**: Full name is Ground. There are several GND pins on the Arduino, any of which can be used to ground circuit.
- **Analog**: The area of pins under the 'Analog In' label (A0 through A5 on the UNO) is Analog In pins. These pins can read the signal from an analog sensor and convert it into a digital value that we can read.
- **Digital**: Across from the analog pins are the digital pins (0 through 13 on the UNO). These pins can be used for both digital input and digital output (like powering an LED).
- **PWM**: The digital pins (3, 5, 6, 9, 10, and 11) on the UNO are the PWM (~) pins. These pins act as normal digital pins, but can also be used for something called Pulse-Width Modulation (PWM).
- **AREF**: Stands for Analog Reference. It is sometimes used to set an external reference voltage (between 0 and 5 Volts) as the upper limit for the analog input pins.

3.2.3 Power LED Indicator

Just beneath and to the right of the word "UNO" on circuit board, there's a tiny LED next to the word 'ON'. This LED should light up whenever plug Arduino into a power source. If this light doesn't turn on, there's a good chance something is wrong.

3.2.4 Reset Button

The Arduino has a reset button. Pushing it will temporarily connect the reset pin to ground and restart any code that is loaded on the Arduino. This can be very useful if code doesn't repeat, but we want to test it multiple times. Unlike the original Nintendo however, blowing on the Arduino doesn't usually fix any problems.

3.2.5 TX RX LEDs

TX is short for transmit, RX is short for receive. In our case, there are two places on the Arduino UNO where TX and RX appear once by digital pins 0 and 1, and a second time next to the TX and RX indicator LEDs. These LEDs will give us some nice visual indications whenever Arduino is receiving or transmitting data.

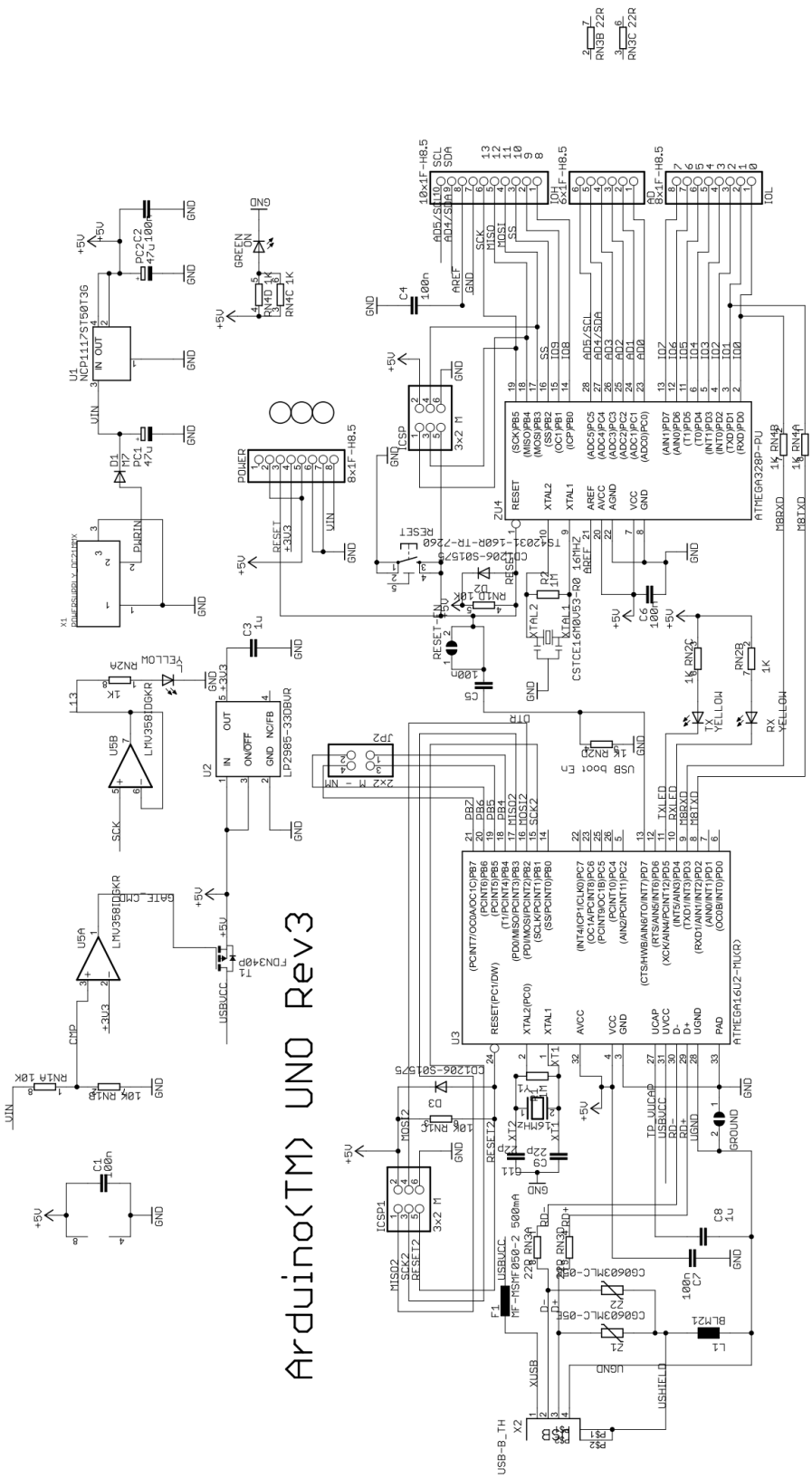
3.2.6 Main IC

The black thing with all the metal legs is an IC, or Integrated Circuit. . The main IC on the Arduino is slightly different from board type to board type, but is usually from the ATmega line of IC's from the ATMEL Company. This can be important, as may need to know the IC type before loading up a new program from the Arduino software. This information can usually be found in writing on the top side of the IC.

3.2.7 Voltage Regulator

The voltage regulator is not actually something interacting with on the Arduino. But it is potentially useful to know that it is there and what it's for. It controls the amount of voltage that is let into the Arduino board. It will turn away an extra voltage that might harm the circuit.

3.2.8 Schematic Diagram



3.3 Getting started with Arduino Software

First download and install the Arduino IDE for Mac, Linux or Windows from arduino.cc. Windows users also need to install a driver. Connect your board via USB, launch the Arduino application and select Arduino Uno from the tools to board menu. Open the sketch File. Open
Examples: 01.Basics: Blink. Click the toolbar button to upload it to your board.

3.3.1 The Integrated Development Environment (IDE)

Every microcontroller needs software to be programmed. The Arduino board is not a case apart. It has its own integrated development environment (IDE).It is free and everyone can download it from its official website using either the Windows, Mac OS X or Linux platform. That allows Arduino Board to gain more users and it also helps it to grow.

3.3.2 IDE Parts

- Compile: Before program "code" can be sent to the board, it needs to be converted into instructions that the board understands. This process is called Compiling.
- Stop: This stops the compilation process.
- Create new Sketch: This opens a new window to create news ketch.
- Open Existing Sketch: This loads a sketch from a file on our computer.
- Save Sketch: This saves the changes to the sketch.
- Upload to Board: This compiles and then transmits over the USB cable to our board.
- Serial Monitor: Until this point when our programs (sketches) didn't work, we just pulled out our hair and tried harder.
- Tab Button: This lets you create multiple files in your sketch. This is for more advanced programming than we will do in this class.
- Sketch Editor: This is where write or edit sketches
- Text Console: This shows you what the IDE is currently doing and is also where error messages display if make a mistake in typing program.
- Line Number: This shows what line number your cursor is on.

3.4 L3G4200D 3-axis gyro

The L3G4200D board is a low-power three-axis angular rate sensor module, provides I2C/SPI digital output interface. The L3G4200D board features I2C pinheader on one side, and I2C connector on the opposite side; Hence, it's more flexible to connect the board to your development system; The board also supports I2C cascading, allowing the use of multi module connected to the I2C bus at the same time by connecting the pinheader and connector

Figure 3.3 gyro L3G4200D

3.5 SparkFun Logic Level Converter

If needed to connect a 3.3V device to a 5V system, The [SparkFun] bi-directional logic level converter is a small device that safely steps down 5V signals to 3.3V AND steps up 3.3V to 5V at the same time. This level converter also works with 2.8V and 1.8V devices. Each level converter has the capability of converting 4 pins on the high side to 4 pins on the low side with two inputs and two outputs provided for each side.

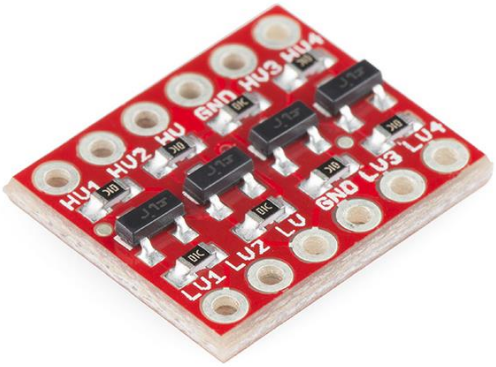


Figure 3.4 Logic Level Converter

The level converter is very easy to use. The board needs to be powered from the two voltages sources (high voltage and low voltage) that your system is using. High voltage (5V for example) to the 'HV' pin, low voltage (3.3V for example) to 'LV', and ground from the system to the 'GND' pin.

3.6 Transmitter and Receiver

FS-R6B Receiver. 2 * Pcs Original FlySky FS-R6B 2.4Ghz 6CH Receiver. Brand Name: Flysky. Model type: Heli/Airplane. Frequency: 2.4G. Package size: 10 * 10 * 2 cm / 4 * 4 * 0.8 in.

FLY SKY 2.4G FS-CT6B 6 CH Channel Radio Model RC Transmitter Receiver Control. Model No: FS-CT6B. Mode Type :Airplane, Helicopter, Glider. Transmitter/Receiver: 2.4GHz. 4 Type (Airplane, Heli90, Heli1)

Figure 3.5 Tx and Rx

3.7 LiPo battery

Minimum Capacity: 2200mAh
(True 100% Capacity)
Configuration: 3S1P / 11.1v / 3Cell
Constant Discharge: 20C
Peak Discharge (10sec): 30C

Pack Weight: 185g
Pack Size: 103 x 33 x 24mm
Charge Plug: JST-XH
Discharge Plug: XT60

Figure 3.6 Battery

3.8 20A ESC

HobbyWing SkyWalker 20A (Linear BEC) Brushless ESC for Aircraft and Heli. ESC is an electronic circuit with the purpose to vary an electric motor's speed, its direction and possibly also to act as a dynamic brake. ESCs are often used on electrically powered radio controlled models, with the variety most often used for brushless motors essentially

providing an electronically generated three-phase electric power low voltage source of energy for the motor.
An ESC can be a stand-alone unit which plugs into the receiver's throttle control channel or incorporated into the receiver itself, as is the case in most toy-grade R/C vehicles. Some R/C manufacturers that install proprietary hobby-grade electronics in their entry-level vehicles, vessels or aircraft use onboard electronics that combine the two on a single circuit board

Figure 3.7 ESC

3.9 14000kv brushless motor

A brushless DC motor is essentially a dc motor without the mechanical commutation of the brushed dc motor. BLDC motors are powered by direct current and have electronic commutation systems instead of the mechanical brushes and commutators used in brushed dc motors.

Figure 3.8 brushless motor

Chapter 4

DESIGN, ANALYSIS AND IMPLEMENTATION

4.1 Introduction

In this project helps those people who interested to build something with Arduino. To Design a project include into two parts, one is hardware design and another part is software design. We use ESC, brushless motor, lipo battery, gyro, level converter and voltage divider circuit for the hardware design and we connected these components with microcontroller. Arduino microcontroller is more suitable for establishing a new project including robotics. Arduino software is downloaded from www.arduino.cc and C/C++ programmable language is used. Many examples are given in the ardiono.cc and this software is easy to usage.

4.2 Hardware design

The whole system design is divided into two parts to design quadrotor drone. One is body design which is mechanical side and circuit design. Finally, the quarotor drone is formed a complete integrated system. In this project Arduino development board is more efficient.

4.2.1 Full Circuit Design

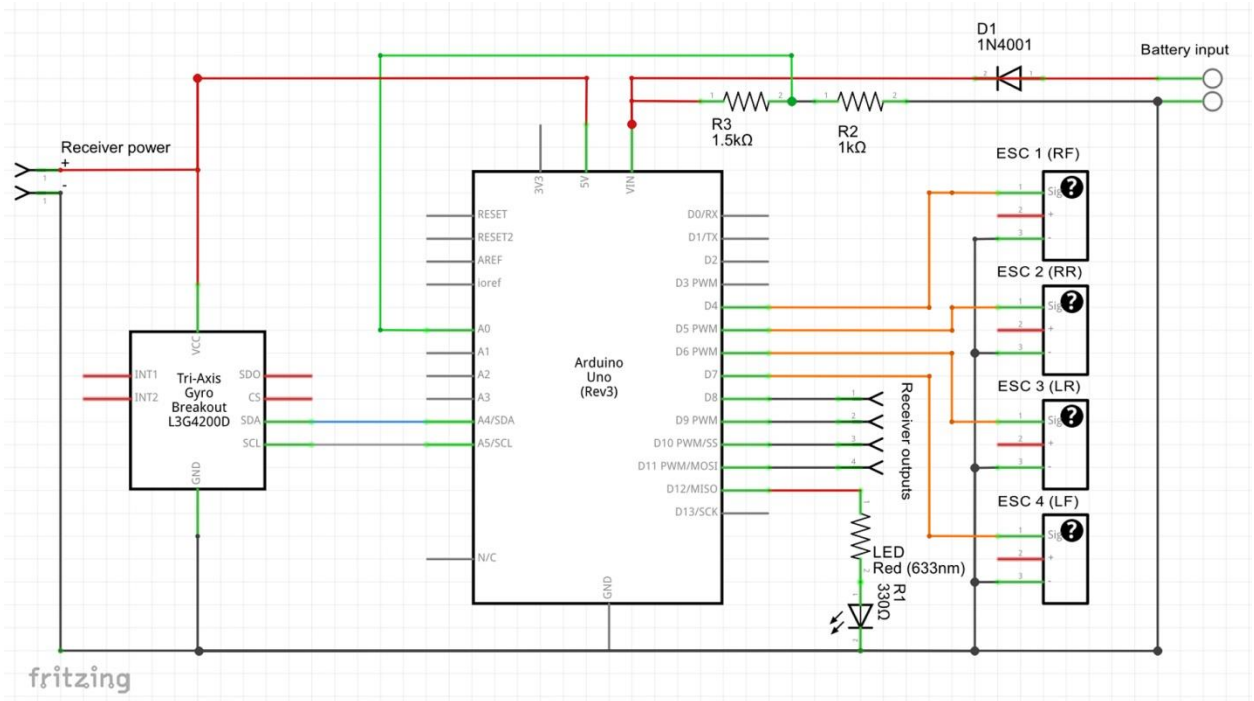


Figure 3.9 Full Circuit Design

4.2.2 Receiver configuration

It is a most complex part in this project. In here there is circuit diagram of receiver configuration. This receiver has 6 channel output. We use 4 individual channels for this drone project. Channel 1 for Pitch, channel 2 for Roll, channel 3 for throttle and channel 4 for YEW control. On receiver channel 1, 2, 3 and 4 signal out pin individually connected to ardiuno pin 8, 9, 10 and 11. And all GND are connected to GNG. Receiver VCC connected to ardiuno +5v out pin and GND to GND.

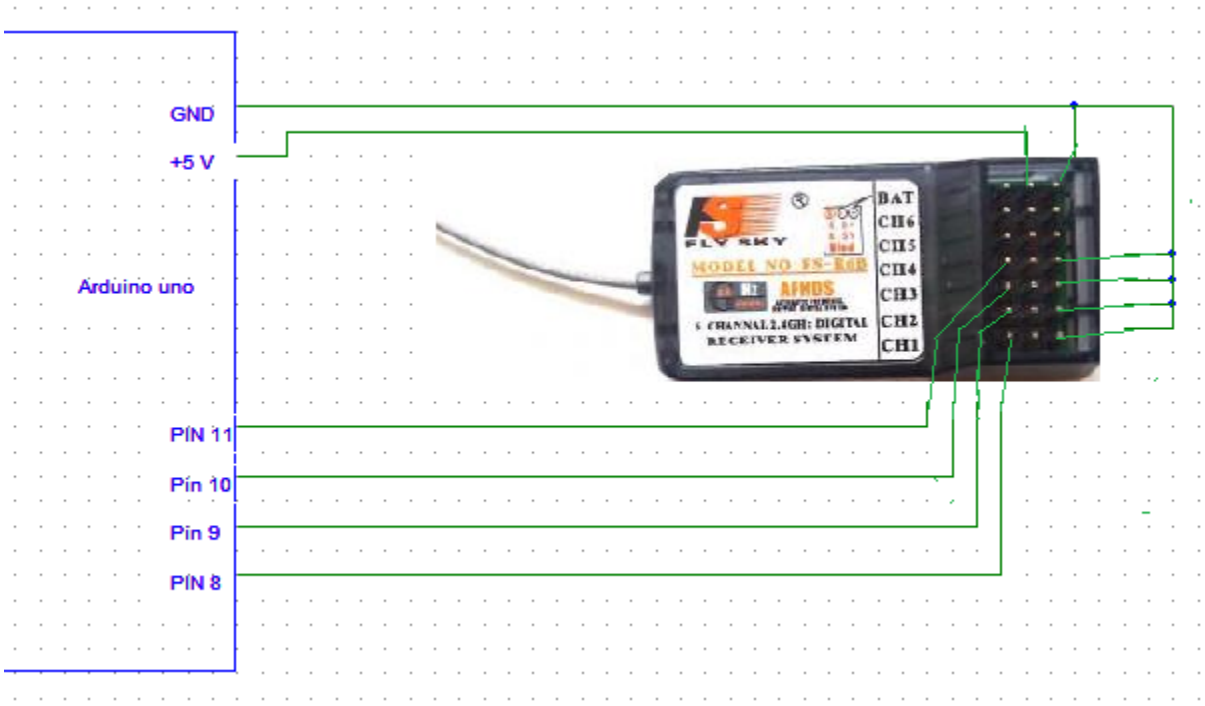


Figure 4.1 Receiver configuration

Receiver code is given to appendix [P1]. After upload the code on monitor mode we can read the every channel data individually. Generally receiver receives 1000 to 2000 micro second plus. In here we receive 1025 micro second plus for channel lowest value and 1504 micro second plus is middle value and 1956 micro second is highest value.

4.2.3 Gyro configuration

Gyro configuration is a important part for the drone. Gyro data is analyzed by ardiuno. It helps to stabilize the drone position on air. Gyro sends data for X, Y, Z direction. Gyro is connected with SDA and SCL pin to ardiuno analog pin A04 and A05 respectively. VCC and GND are connected to ardiuno +5v and GND respectively.

There are two different code used for gyro configure. Appendix [P2] and [2p] given to [P1] for devise test for gyro. After uploading the code and monitor mode the gyro model number is shown. And second code [P3], after uploading code and monitor mode X, Y, Z directional change value is shown. All value will zero (0) for zero directional change. Negative and positive value will show for directional change.

Ardiuno input/output signal voltage is around 0 to 5 volt. So ardino can communicate with those devises which have 0 to 5 volt input/output signal level. In this project we use a gyro which is 3 volt devise. So its I/O signal level is 0 to around 3 volt. For this reson, if we connect the gyro with drdiuno, the gyro will burn out. So we use logic level converter circuit. These circuits pull down 5 volt signal to 3 volt for gyro and pull up 3 volt to 5 volt signal for ardino. This circuit has 6 individual channels. We need 2 of them. The cofigure circuit is given below.

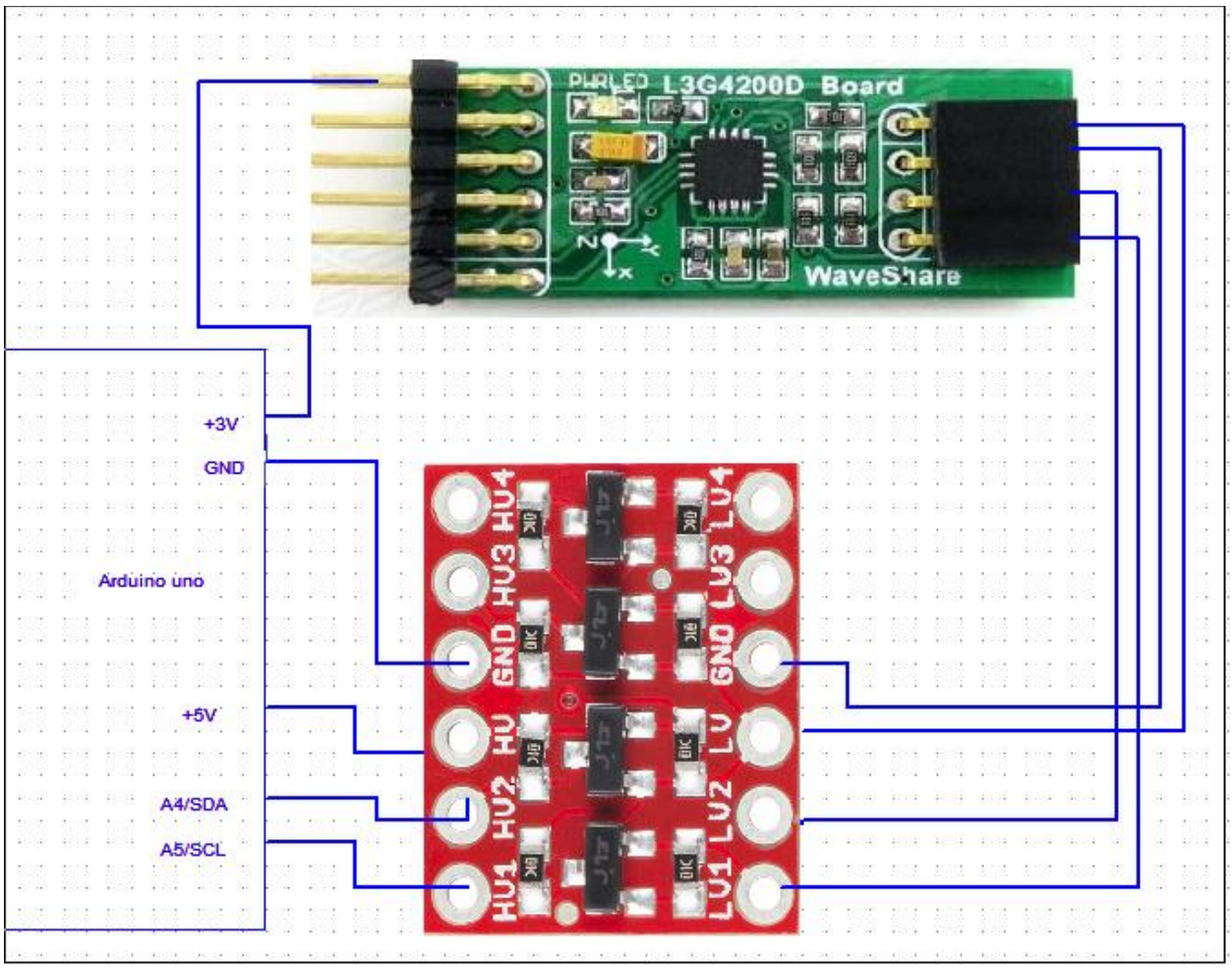


Figure 4.2Gyro configuration

4.2.4 ESC configuration

ESC means electronic speed controller. Generally it supplies voltage for run brushless motor with high current flow. It receive PWM signal from ardiuno and convert it to voltage feed. ESC powered from lipo (lithium polymer battery). ESC signal pin connected to pin change interrupt pin of ardiuno, digital pin 4 or 5 or 6 or 7. The three phase voltage out plug directly connected to thee brushless motor. For motor connection there is no specific configuration, just need connection.

In here ESC test circuit diagram is given blow. For motor run test ESC need PWM signal. In here we connect ESC signaling pin directly to channel-3 signal out pin of RF receiver module. Because RF receiver produce PWM signal. RF receiver powered from ESC, built-in voltage regulator provide +5V to RF receiver module. ESC is powered from lipo (lithium polymer battery), which is +11.1V.

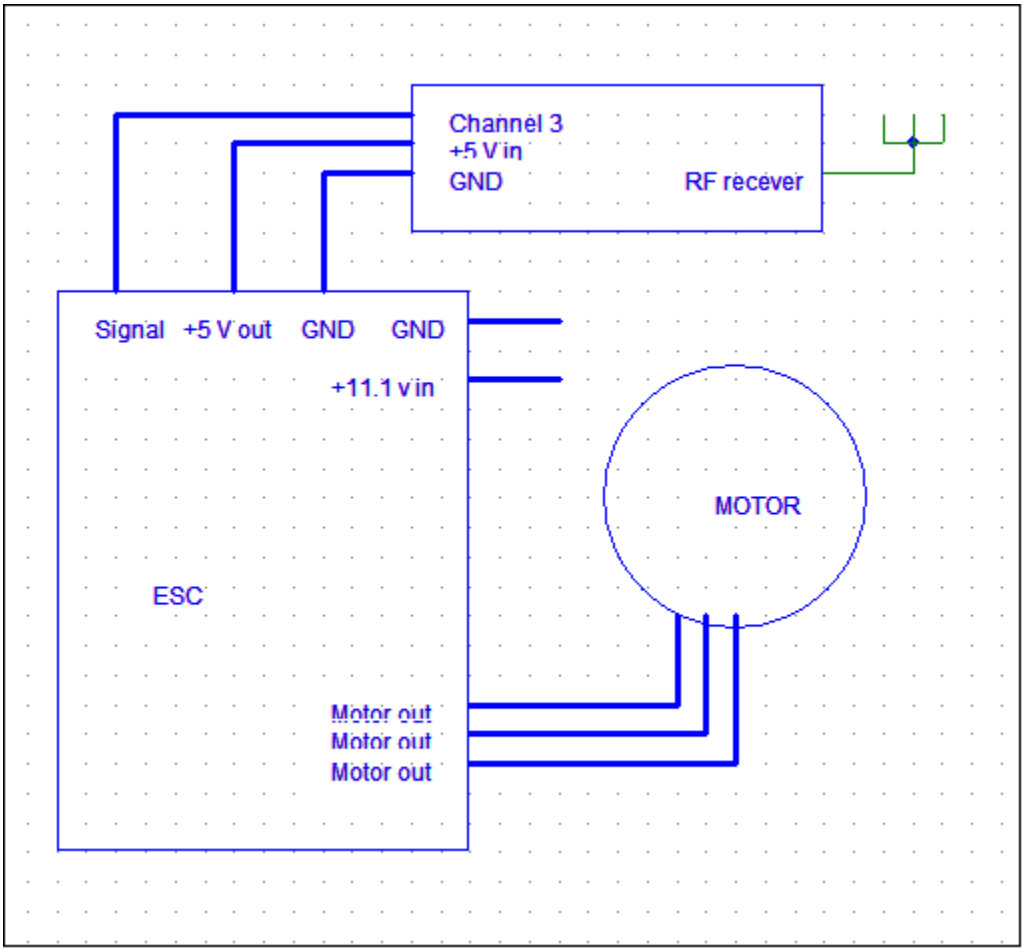


Figure 4.3 ESC configuration

Now from the transmitter, when we up the throttle stick, motor is starting to rotate. For highest throttle positions the motor rotate with highlight speed with 14000 rpm.

4.2.5 Voltage reference circuit

Quadrotor drone consume high current so lithium polymer battery goes down quickly. So ardiuno need to know battery level. Voltage reference circuit given below. Analog A0 pin read 4.8 to 5 volt to detect as battery level high. And below 4.0 volt battery is critically low.

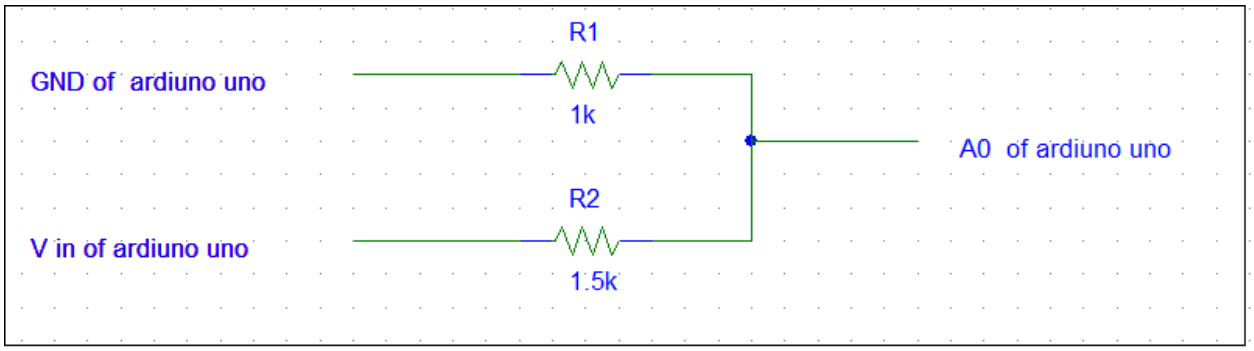


Figure 4.4 Voltage reference circuit

4.3 Development of the Whole System

After completing the whole work, drone run properly. The total circuitry performed well. When drone is switch is on there are need a few second to calibrating total system. There is a statue LED, when LED is shut down whole system is ready to launch.

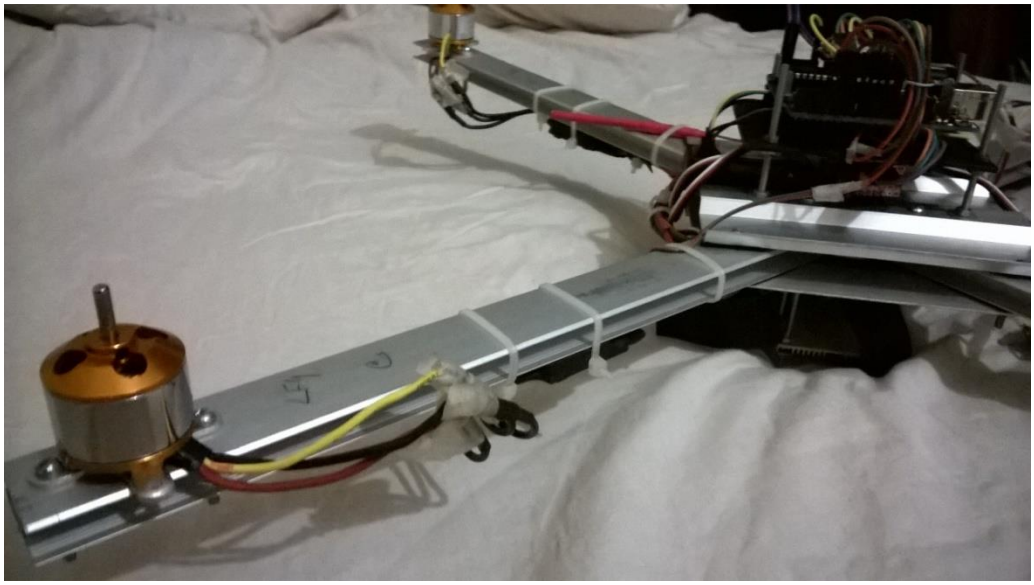


Figure 4.5 Whole System 1

Figure 4.6 Whole System 2

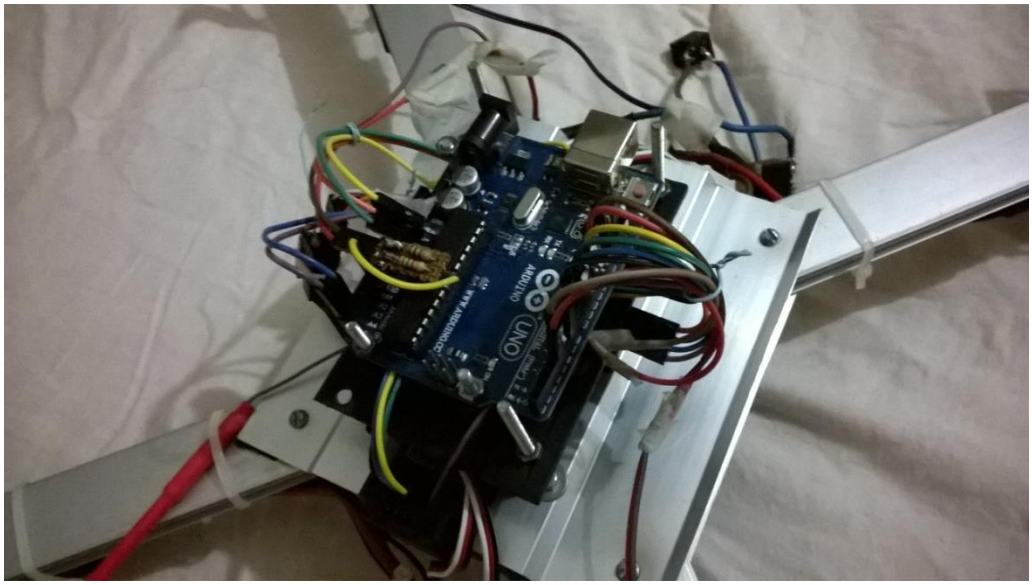


Figure 4.7 Whole System 3

Chapter 5

CONCLUSION

5.1 Future work scope

The prototypes built around the development and implementation of this project have proven to be invaluable in testing how robust the simulation and controller models are in real flight scenarios. Several updates were made between the creation of the drone system.. On the drone system, a lipo needs to be replaced and the vibration dampening system needs some further work. Once more funding is secured, these parts and changes can be implemented and further testing can proceed. Until such time, development and testing will be limited to a simulation environment.

5.2 Conclusion

The overall objective of this project endeavor was to approach the design of a man-portable drone from a systems engineering standpoint. The focus was to be on the system and its use as a whole, not isolating any one or two subsystems or using an external environment to facilitate the drone movement in the terrain. The derivation of the modeling equations and the raw implementation of a simulation model and controller allowed for the understanding of the physical characteristics that dictate the behavior of the drone. The platform is comprised of several subsystems, each of which have been studied in-depth to understand how the various subsystems can work together for synergistic benefit. These systems include the avionics, sensors, actuators, chassis, and user control architecture. The test flight experiments performed using the drone build prototype indicates that successful flight is possible with adequate funding. Although faulty sensors and subsystem components paired with a lack of continued funding kept this current prototype confined to the test bench, small tweaks to the avionics loop and the addition of lipo promise to make for a stable, autonomous platform.

APPENDIX

Programming code for receiver test [P1]

```
//Declaring Variables
byte last_channel_1, last_channel_2, last_channel_3, last_channel_4;
int receiver_input_channel_1, receiver_input_channel_2, receiver_input_channel_3, receiver_input_channel_4;
unsigned long timer_1, timer_2, timer_3, timer_4;

//Setup routine
void setup(){
  //Arduino (Atmega) pins default to inputs, so they don't need to be explicitly declared as inputs
  PCICR |= (1 << PCIE0);    // set PCIE0 to enable PCMSK0 scan
  PCMSK0 |= (1 << PCINT0);  // set PCINT0 (digital input 8) to trigger an interrupt on state change
  PCMSK0 |= (1 << PCINT1);  // set PCINT1 (digital input 9)to trigger an interrupt on state change
  PCMSK0 |= (1 << PCINT2);  // set PCINT2 (digital input 10)to trigger an interrupt on state change
  PCMSK0 |= (1 << PCINT3);  // set PCINT3 (digital input 11)to trigger an interrupt on state change
  Serial.begin(9600);
}

//Main program loop
void loop(){
  delay(250);
  print_signals();
}

//This routine is called every time input 8, 9, 10 or 11 changed state
ISR(PCINT0_vect){
  //Channel 1=========================================
  if(last_channel_1 == 0 && PINB & B00000001 ){       //Input 8 changed from 0 to 1
    last_channel_1 = 1;                               //Remember current input state
    timer_1 = micros();                               //Set timer_1 to micros()
  }
  else if(last_channel_1 == 1 && !(PINB & B00000001)){  //Input 8 changed from 1 to 0
    last_channel_1 = 0;                               //Remember current input state
    receiver_input_channel_1 = micros() - timer_1;     //Channel 1 is micros() - timer_1
  }
  //Channel 2=========================================
  if(last_channel_2 == 0 && PINB & B00000010 ){       //Input 9 changed from 0 to 1
    last_channel_2 = 1;                               //Remember current input state
    timer_2 = micros();                               //Set timer_2 to micros()
  }
  else if(last_channel_2 == 1 && !(PINB & B00000010)){  //Input 9 changed from 1 to 0
    last_channel_2 = 0;                               //Remember current input state
    receiver_input_channel_2 = micros() - timer_2;     //Channel 2 is micros() - timer_2
  }
  //Channel 3=========================================
  if(last_channel_3 == 0 && PINB & B00000100 ){       //Input 10 changed from 0 to 1
    last_channel_3 = 1;                               //Remember current input state
    timer_3 = micros();                               //Set timer_3 to micros()
  }
  else if(last_channel_3 == 1 && !(PINB & B00000100)){  //Input 10 changed from 1 to 0
```

```
    last_channel_3 = 0;                         //Remember current input state
    receiver_input_channel_3 = micros() - timer_3;     //Channel 3 is micros() - timer_3
  }
  //Channel 4=========================================
  if(last_channel_4 == 0 && PINB & B00001000 ){       //Input 11 changed from 0 to 1
    last_channel_4 = 1;                         //Remember current input state
    timer_4 = micros();                         //Set timer_4 to micros()
  }
  else if(last_channel_4 == 1 && !(PINB & B00001000)){  //Input 11 changed from 1 to 0
    last_channel_4 = 0;                         //Remember current input state
    receiver_input_channel_4 = micros() - timer_4;     //Channel 4 is micros() - timer_4
  }
}
//Subroutine for displaying the receiver signals
void print_signals(){
  Serial.print("Roll:");
  if(receiver_input_channel_1 - 1480 < 0)Serial.print("<<<");
  else if(receiver_input_channel_1 - 1520 > 0)Serial.print(">>>");
  else Serial.print("-+-");
  Serial.print(receiver_input_channel_1);

  Serial.print("  Nick:");
  if(receiver_input_channel_2 - 1480 < 0)Serial.print("^^^");
  else if(receiver_input_channel_2 - 1520 > 0)Serial.print("vvv");
  else Serial.print("-+-");
  Serial.print(receiver_input_channel_2);

  Serial.print("  Gas:");
  if(receiver_input_channel_3 - 1480 < 0)Serial.print("vvv");
  else if(receiver_input_channel_3 - 1520 > 0)Serial.print("^^^");
  else Serial.print("-+-");
  Serial.print(receiver_input_channel_3);

  Serial.print("  Yaw:");
  if(receiver_input_channel_4 - 1480 < 0)Serial.print("<<<");
  else if(receiver_input_channel_4 - 1520 > 0)Serial.print(">>>");
  else Serial.print("-+-");
  Serial.println(receiver_input_channel_4);
}
```

Programming code for Gyro device [P2]

```
#include <Wire.h> //Include the Wire.h library so we can communicate with the gyro

byte lowByte;
unsigned long timer;
int adress;

//Setup routine
void setup() {
  Wire.begin();             //Start the I2C as master
```

```
  Serial.begin(9600);         //Start the serial connetion @ 9600bps
  delay(250);                 //Give the gyro time to start
}
//Main program
void loop() {
  if (adress == 0) {
    Serial.println("Searching for divice");
    for (adress = 0; adress < 255; adress ++) {
      Wire.beginTransmission(adress);
      Wire.write(0x0F);
      Wire.endTransmission();
      Wire.requestFrom(adress, 1);
      timer = millis() + 100;
      while (Wire.available() < 1 && timer > millis());
      lowByte = Wire.read();
      if (lowByte == 211) {
        Serial.println("");
        Serial.print("Sensor L3G4200 found @ adress:");
        Serial.println(adress);
        adress = 256;
      }
      else if (lowByte == 215) {
        Serial.println("");
        Serial.print("Sensor L3GD20H found @ adress:");
        Serial.println(adress);
        adress = 256;
      }
      else Serial.print(".");
    }
    if (adress == 255) {
      Serial.println("");
      Serial.println("divice not found!");
    }
  }
}
```

Programming code for Gyro test [P3]

```
#include <Wire.h> //Include the Wire.h library so we can communicate with the gyro

//Declaring variables
int cal_int;
unsigned long UL_timer;
double gyro_pitch, gyro_roll, gyro_yaw;
double gyro_roll_cal, gyro_pitch_cal, gyro_yaw_cal;
byte highByte, lowByte;

//Setup routine
void setup(){
  Wire.begin();                           //Start the I2C as master
  Serial.begin(9600);                       //Start the serial connetion @ 9600bps
```

```
  //The gyro is disabled by default and needs to be started
  Wire.beginTransmission(105);                  //Start communication with the gyro (adress 1101001)
  Wire.write(0x20);                             //We want to write to register 20
  Wire.write(0x0F);                             //Set the register bits as 00001111 (Turn on the gyro and enable all axis)
  Wire.endTransmission();                       //End the transmission with the gyro
  Wire.beginTransmission(105);                  //Start communication with the gyro (adress 1101001)
  Wire.write(0x23);                             //We want to write to register 23
  Wire.write(0x80);                             //Set the register bits as 10000000 (Block Data Update active)
  Wire.endTransmission();                       //End the transmission with the gyro


  delay(250);                                   //Give the gyro time to start

  //Let's take multiple samples so we can determine the average gyro offset
  Serial.print("Starting calibration...");          //Print message
  for (cal_int = 0; cal_int < 2000 ; cal_int ++){    //Take 2000 readings for calibration
    gyro_signalen();                            //Read the gyro output
    gyro_roll_cal += gyro_roll;                 //Ad roll value to gyro_roll_cal
    gyro_pitch_cal += gyro_pitch;               //Ad pitch value to gyro_pitch_cal
    gyro_yaw_cal += gyro_yaw;                   //Ad yaw value to gyro_yaw_cal
    if(cal_int%100 == 0)Serial.print(".");          //Print a dot every 100 readings
    delay(4);                                   //Wait 4 milliseconds before the next loop
  }
  //Now that we have 2000 measures, we need to devide by 2000 to get the average gyro offset
  Serial.println(" done!");                     //2000 measures are done!
  gyro_roll_cal /= 2000;                        //Divide the roll total by 2000
  gyro_pitch_cal /= 2000;                       //Divide the pitch total by 2000
  gyro_yaw_cal /= 2000;                         //Divide the yaw total by 2000

}
//Main program
void loop(){
  delay(250);                                   //Wait 250 microseconds for every loop
  gyro_signalen();                              //Read the gyro signals
  print_output();                               //Print the output
}

void gyro_signalen(){
  Wire.beginTransmission(105);                  //Start communication with the gyro (adress 1101001)
  Wire.write(168);                              //Start reading @ register 28h and auto increment with every read
  Wire.endTransmission();                       //End the transmission
  Wire.requestFrom(105, 6);                     //Request 6 bytes from the gyro
  while(Wire.available() < 6);                  //Wait until the 6 bytes are received
  lowByte = Wire.read();                        //First received byte is the low part of the angular data
  highByte = Wire.read();                       //Second received byte is the high part of the angular data
  gyro_roll = ((highByte<<8)|lowByte);          //Multiply highByte by 256 and ad lowByte
  if(cal_int == 2000)gyro_roll -= gyro_roll_cal;    //Only compensate after the calibration
  lowByte = Wire.read();                        //First received byte is the low part of the angular data
  highByte = Wire.read();                       //Second received byte is the high part of the angular data
  gyro_pitch = ((highByte<<8)|lowByte);         //Multiply highByte by 256 and ad lowByte
  gyro_pitch *= -1;                             //Invert axis
  if(cal_int == 2000)gyro_pitch -= gyro_pitch_cal;  //Only compensate after the calibration
```

```
  lowByte = Wire.read();                    //First received byte is the low part of the angular data
  highByte = Wire.read();                   //Second received byte is the high part of the angular data
  gyro_yaw = ((highByte<<8)|lowByte);       //Multiply highByte by 256 and ad lowByte
  gyro_yaw *= -1;                           //Invert axis
  if(cal_int == 2000)gyro_yaw -= gyro_yaw_cal;      //Only compensate after the calibration
}

void print_output(){
  Serial.print("Pitch:");
  if(gyro_pitch >= 0)Serial.print("+");
  Serial.print(gyro_pitch/57.14286,0);              //Convert to degree per second
  if(gyro_pitch/57.14286 - 2 > 0)Serial.print(" NoU");
  else if(gyro_pitch/57.14286 + 2 < 0)Serial.print(" NoD");
  else Serial.print(" ---");
  Serial.print("  Roll:");
  if(gyro_roll >= 0)Serial.print("+");
  Serial.print(gyro_roll/57.14286,0);               //Convert to degree per second
  if(gyro_roll/57.14286 - 2 > 0)Serial.print(" RwD");
  else if(gyro_roll/57.14286 + 2 < 0)Serial.print(" RwU");
  else Serial.print(" ---");
  Serial.print("  Yaw:");
  if(gyro_yaw >= 0)Serial.print("+");
  Serial.print(gyro_yaw/57.14286,0);                //Convert to degree per second
  if(gyro_yaw/57.14286 - 2 > 0)Serial.println(" NoR");
  else if(gyro_yaw/57.14286 + 2 < 0)Serial.println(" NoL");
  else Serial.println(" ---");
}
```

Programming code for ESC output test [P4]

```
//Declaring Variables
byte last_channel_1, last_channel_2, last_channel_3, last_channel_4;
int receiver_input_channel_1, receiver_input_channel_2, receiver_input_channel_3, receiver_input_channel_4;
int counter_channel_1, counter_channel_2, counter_channel_3, counter_channel_4, start;
unsigned long timer_channel_1, timer_channel_2, timer_channel_3, timer_channel_4, esc_timer,
esc_loop_timer;
unsigned long zero_timer, timer_1, timer_2, timer_3, timer_4, current_time;

//Setup routine
void setup() {
  DDRD |= B11110000;                        //Configure digital poort 4, 5, 6 and 7 as output
  DDRB |= B00010000;                        //Configure digital poort 12 as output
  //Arduino Uno pins default to inputs, so they don't need to be explicitly declared as inputs

  PCICR |= (1 << PCIE0);                    // set PCIE0 to enable PCMSK0 scan
  PCMSK0 |= (1 << PCINT0);                  // set PCINT0 (digital input 8) to trigger an interrupt on state
change
  PCMSK0 |= (1 << PCINT1);                  // set PCINT1 (digital input 9)to trigger an interrupt on state
change
  PCMSK0 |= (1 << PCINT2);                  // set PCINT2 (digital input 10)to trigger an interrupt on state
change
  PCMSK0 |= (1 << PCINT3);                  // set PCINT3 (digital input 11)to trigger an interrupt on state
```

```
change

  ///Wait until the receiver is active and the throtle is set to the lower position.
  while (receiver_input_channel_3 < 990 || receiver_input_channel_3 > 1020 || receiver_input_channel_4 < 1500) {
    start ++;                                  //While waiting increment start whith every loop.
    //We don't want the esc's to be beeping annoyingly. So let's give them a 1000us puls while waiting for the receiver inputs.
    PORTD |= B11110000;                        //Set digital poort 4, 5, 6 and 7 high.
    delayMicroseconds(1000);                   //Wait 1000us (We can use delayMicroseconds because the receiver interrupt routine is not used).
    PORTD &= B00001111;                        //Set digital poort 4, 5, 6 and 7 low.
    delay(3);                                  //Wait 3 milliseconds before the next loop.
    if (start == 125) {                        //Every 125 loops (500ms).
      digitalWrite(12, !digitalRead(12));      //Change the led status.
      start = 0;                               //Start again at 0.
    }
  }
  start = 0;
  digitalWrite(12, LOW);                       //Turn off the led.
  zero_timer = micros();                       //Set the zero_timer for the first loop.
}

//Main program loop
void loop() {
  while (zero_timer + 4000 > micros());        //Start the pulse after 4000 micro seconds.
  zero_timer = micros();                       //Reset the zero timer.
  PORTD |= B11110000;
  timer_channel_1 = receiver_input_channel_3 + zero_timer;  //Calculate the time when digital port 8 is set low.
  timer_channel_2 = receiver_input_channel_3 + zero_timer;  //Calculate the time when digital port 9 is set low.
  timer_channel_3 = receiver_input_channel_3 + zero_timer;  //Calculate the time when digital port 10 is set low.
  timer_channel_4 = receiver_input_channel_3 + zero_timer;  //Calculate the time when digital port 11 is set low.

  while (PORTD >= 16) {                        //Execute the loop until digital port 8 til 11 is low.
    esc_loop_timer = micros();                 //Check the current time.
    if (timer_channel_1 <= esc_loop_timer)PORTD &= B11101111; //When the delay time is expired, digital port 8 is set low.
    if (timer_channel_2 <= esc_loop_timer)PORTD &= B11011111; //When the delay time is expired, digital port 9 is set low.
    if (timer_channel_3 <= esc_loop_timer)PORTD &= B10111111; //When the delay time is expired, digital port 10 is set low.
    if (timer_channel_4 <= esc_loop_timer)PORTD &= B01111111; //When the delay time is expired, digital port 11 is set low.
  }
}

//This routine is called every time input 8, 9, 10 or 11 changed state
ISR(PCINT0_vect) {
  current_time = micros();
  //Channel 1=========================================
  if (PINB & B00000001) {                      //Is input 8 high?
```

```
  if (last_channel_1 == 0) {                    //Input 8 changed from 0 to 1
    last_channel_1 = 1;                         //Remember current input state
    timer_1 = current_time;                     //Set timer_1 to current_time
  }
 }
 else if (last_channel_1 == 1) {                //Input 8 is not high and changed from 1 to 0
  last_channel_1 = 0;                           //Remember current input state
  receiver_input_channel_1 = current_time - timer_1;       //Channel 1 is current_time - timer_1
 }
 //Channel 2=========================================
 if (PINB & B00000010 ) {                       //Is input 9 high?
  if (last_channel_2 == 0) {                    //Input 9 changed from 0 to 1
    last_channel_2 = 1;                         //Remember current input state
    timer_2 = current_time;                     //Set timer_2 to current_time
  }
 }
 else if (last_channel_2 == 1) {                //Input 9 is not high and changed from 1 to 0
  last_channel_2 = 0;                           //Remember current input state
  receiver_input_channel_2 = current_time - timer_2;       //Channel 2 is current_time - timer_2
 }
 //Channel 3=========================================
 if (PINB & B00000100 ) {                       //Is input 10 high?
  if (last_channel_3 == 0) {                    //Input 10 changed from 0 to 1
    last_channel_3 = 1;                         //Remember current input state
    timer_3 = current_time;                     //Set timer_3 to current_time
  }
 }
 else if (last_channel_3 == 1) {                //Input 10 is not high and changed from 1 to 0
  last_channel_3 = 0;                           //Remember current input state
  receiver_input_channel_3 = current_time - timer_3;       //Channel 3 is current_time - timer_3
 }
 //Channel 4=========================================
 if (PINB & B00001000 ) {                       //Is input 11 high?
  if (last_channel_4 == 0) {                    //Input 11 changed from 0 to 1
    last_channel_4 = 1;                         //Remember current input state
    timer_4 = current_time;                     //Set timer_4 to current_time
  }
 }
 else if (last_channel_4 == 1) {                //Input 11 is not high and changed from 1 to 0
  last_channel_4 = 0;                           //Remember current input state
  receiver_input_channel_4 = current_time - timer_4;       //Channel 4 is current_time - timer_4
 }
}
```

Combination of all Programming code [P5]

```
#include <Wire.h> //Include the Wire.h library so we can communicate with the gyro.

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//////////////
//PID gain and limit settings
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//////////////
```

```
float pid_p_gain_roll = 1.3;              //Gain setting for the roll P-controller (1.3)
float pid_i_gain_roll = 0.05;             //Gain setting for the roll I-controller (0.3)
float pid_d_gain_roll = 15;               //Gain setting for the roll D-controller (15)
int pid_max_roll = 400;                   //Maximum output of the PID-controller (+/-)

float pid_p_gain_pitch = pid_p_gain_roll;  //Gain setting for the pitch P-controller.
float pid_i_gain_pitch = pid_i_gain_roll;  //Gain setting for the pitch I-controller.
float pid_d_gain_pitch = pid_d_gain_roll;  //Gain setting for the pitch D-controller.
int pid_max_pitch = pid_max_roll;         //Maximum output of the PID-controller (+/-)

float pid_p_gain_yaw = 4.0;               //Gain setting for the pitch P-controller. //4.0
float pid_i_gain_yaw = 0.02;              //Gain setting for the pitch I-controller. //0.02
float pid_d_gain_yaw = 0.0;               //Gain setting for the pitch D-controller.
int pid_max_yaw = 400;                    //Maximum output of the PID-controller (+/-)

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//Declaring Variables
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
byte last_channel_1, last_channel_2, last_channel_3, last_channel_4;
int receiver_input_channel_1, receiver_input_channel_2, receiver_input_channel_3, receiver_input_channel_4;
int counter_channel_1, counter_channel_2, counter_channel_3, counter_channel_4, loop_counter;
int esc_1, esc_2, esc_3, esc_4;
int throttle, battery_voltage;
unsigned long timer_channel_1, timer_channel_2, timer_channel_3, timer_channel_4, esc_timer, esc_loop_timer;
unsigned long timer_1, timer_2, timer_3, timer_4, current_time;
int cal_int, start;
unsigned long loop_timer;
double gyro_pitch, gyro_roll, gyro_yaw;
double gyro_roll_cal, gyro_pitch_cal, gyro_yaw_cal;
byte highByte, lowByte;

float pid_error_temp;
float pid_i_mem_roll, pid_roll_setpoint, gyro_roll_input, pid_output_roll, pid_last_roll_d_error;
float pid_i_mem_pitch, pid_pitch_setpoint, gyro_pitch_input, pid_output_pitch, pid_last_pitch_d_error;
float pid_i_mem_yaw, pid_yaw_setpoint, gyro_yaw_input, pid_output_yaw, pid_last_yaw_d_error;

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//Setup routine
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void setup(){

  Wire.begin();                                //Start the I2C as master.

  DDRD |= B11110000;                           //Configure digital poort 4, 5, 6 and 7 as output.
  DDRB |= B00110000;                           //Configure digital poort 12 and 13 as output.
  //Arduino (Atmega) pins default to inputs, so they don't need to be explicitly declared as inputs.

  //Use the led on the Arduino for startup indication
  digitalWrite(12,HIGH);                       //Turn on the warning led.
```

```
  delay(3000);                                        //Wait 2 second befor continuing.

  Wire.beginTransmission(105);                        //Start communication with the gyro (adress 1101001)
  Wire.write(0x20);                                   //We want to write to register 1 (20 hex)
  Wire.write(0x0F);                                   //Set the register bits as 00001111 (Turn on the gyro and enable all
axis)
  Wire.endTransmission();                             //End the transmission with the gyro

  Wire.beginTransmission(105);                        //Start communication with the gyro (adress 1101001)
  Wire.write(0x23);                                   //We want to write to register 4 (23 hex)
  Wire.write(0x90);                                   //Set the register bits as 10010000 (Block Data Update active & 500dps
full scale)
  Wire.endTransmission();                             //End the transmission with the gyro

  delay(250);                                         //Give the gyro time to start.

  //Let's take multiple gyro data samples so we can determine the average gyro offset (calibration).
  for (cal_int = 0; cal_int < 2000 ; cal_int ++){           //Take 2000 readings for calibration.
    if(cal_int % 15 == 0)digitalWrite(12, !digitalRead(12));  //Change the led status to indicate calibration.
    gyro_signalen();                                  //Read the gyro output.
    gyro_roll_cal += gyro_roll;                       //Ad roll value to gyro_roll_cal.
    gyro_pitch_cal += gyro_pitch;                     //Ad pitch value to gyro_pitch_cal.
    gyro_yaw_cal += gyro_yaw;                         //Ad yaw value to gyro_yaw_cal.
    //We don't want the esc's to be beeping annoyingly. So let's give them a 1000us puls while calibrating the gyro.
    PORTD |= B11110000;                               //Set digital poort 4, 5, 6 and 7 high.
    delayMicroseconds(1000);                          //Wait 1000us.
    PORTD &= B00001111;                               //Set digital poort 4, 5, 6 and 7 low.
    delay(3);                                         //Wait 3 milliseconds before the next loop.
  }
  //Now that we have 2000 measures, we need to devide by 2000 to get the average gyro offset.
  gyro_roll_cal /= 2000;                              //Divide the roll total by 2000.
  gyro_pitch_cal /= 2000;                             //Divide the pitch total by 2000.
  gyro_yaw_cal /= 2000;                               //Divide the yaw total by 2000.

  PCICR |= (1 << PCIE0);                              //Set PCIE0 to enable PCMSK0 scan.
  PCMSK0 |= (1 << PCINT0);                            //Set PCINT0 (digital input 8) to trigger an interrupt on state
change.
  PCMSK0 |= (1 << PCINT1);                            //Set PCINT1 (digital input 9)to trigger an interrupt on state
change.
  PCMSK0 |= (1 << PCINT2);                            //Set PCINT2 (digital input 10)to trigger an interrupt on state
change.
  PCMSK0 |= (1 << PCINT3);                            //Set PCINT3 (digital input 11)to trigger an interrupt on state
change.

    //We don't want the esc's to be beeping annoyingly. So let's give them a 1000us puls while waiting for the
receiver inputs.
    delayMicroseconds(1000);                          //Wait 1000us.
    PORTD &= B00001111;                               //Set digital poort 4, 5, 6 and 7 low.
    delay(3);                                         //Wait 3 milliseconds before the next loop.
    if(start == 125){                                 //Every 125 loops (500ms).
      digitalWrite(12, !digitalRead(12));             //Change the led status.
      start = 0;                                      //Start again at 0.
    }
```

```
  }
  start = 0;                                   //Set start back to 0.

  //Load the battery voltage to the battery_voltage variable.
  //65 is the voltage compensation for the diode.
  //12.6V equals ~5V @ Analog 0.
  //12.6V equals 1023 analogRead(0).
  //1260 / 1023 = 1.2317.
  //The variable battery_voltage holds 1050 if the battery voltage is 10.5V.
  battery_voltage = (analogRead(0) + 65) * 1.2317;

  //When everything is done, turn off the led.
  digitalWrite(12,LOW);                        //Turn off the warning led.
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//////////////
//Main program loop
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//////////////
void loop(){
  //Let's get the current gyro data and scale it to degrees per second for the pid calculations.
  gyro_signalen();
  gyro_roll_input = (gyro_roll_input * 0.8) + ((gyro_roll / 57.14286) * 0.2);          //Gyro pid input is deg/sec.
  gyro_pitch_input = (gyro_pitch_input * 0.8) + ((gyro_pitch / 57.14286) * 0.2);       //Gyro pid input is deg/sec.
  gyro_yaw_input = (gyro_yaw_input * 0.8) + ((gyro_yaw / 57.14286) * 0.2);             //Gyro pid input is deg/sec.

    start = 2;
    //Reset the pid controllers for a bumpless start.
    pid_i_mem_roll = 0;
    pid_last_roll_d_error = 0;
    pid_i_mem_pitch = 0;
    pid_last_pitch_d_error = 0;
    pid_i_mem_yaw = 0;
    pid_last_yaw_d_error = 0;
  }
  //Stopping the motors: throttle low and yaw right.
  if(start == 2 && receiver_input_channel_3 < 1050 && receiver_input_channel_4 > 1950)start = 0;

  //The PID set point in degrees per second is determined by the roll receiver input.
  //In the case of deviding by 3 the max roll rate is aprox 164 degrees per second ( (500-8)/3 = 164d/s ).
  pid_roll_setpoint = 0;
  //We need a little dead band of 16us for better results.
  if(receiver_input_channel_1 > 1508)pid_roll_setpoint = (receiver_input_channel_1 - 1508)/3.0;
  else if(receiver_input_channel_1 < 1492)pid_roll_setpoint = (receiver_input_channel_1 - 1492)/3.0;

  //The PID set point in degrees per second is determined by the pitch receiver input.
  //In the case of deviding by 3 the max pitch rate is aprox 164 degrees per second ( (500-8)/3 = 164d/s ).
  pid_pitch_setpoint = 0;
  //We need a little dead band of 16us for better results.
  if(receiver_input_channel_2 > 1508)pid_pitch_setpoint = (receiver_input_channel_2 - 1508)/3.0;
  else if(receiver_input_channel_2 < 1492)pid_pitch_setpoint = (receiver_input_channel_2 - 1492)/3.0;

  //The PID set point in degrees per second is determined by the yaw receiver input.
```

```
  //In the case of deviding by 3 the max yaw rate is aprox 164 degrees per second ( (500-8)/3 = 164d/s ).
  pid_yaw_setpoint = 0;
  //We need a little dead band of 16us for better results.
  if(receiver_input_channel_3 > 1050){ //Do not yaw when turning off the motors.
    if(receiver_input_channel_4 > 1508)pid_yaw_setpoint = (receiver_input_channel_4 - 1508)/3.0;
    else if(receiver_input_channel_4 < 1492)pid_yaw_setpoint = (receiver_input_channel_4 - 1492)/3.0;
  }
  //PID inputs are known. So we can calculate the pid output.
  calculate_pid();

  //The battery voltage is needed for compensation.
  //A complementary filter is used to reduce noise.
  //0.09853 = 0.08 * 1.2317.
  battery_voltage = battery_voltage * 0.92 + (analogRead(0) + 65) * 0.09853;

  //Turn on the led if battery voltage is to low.
  if(battery_voltage < 1050 && battery_voltage > 600)digitalWrite(12, HIGH);

  throttle = receiver_input_channel_3;                    //We need the throttle signal as a base signal.

  if (start == 2){                                        //The motors are started.
    if (throttle > 1800) throttle = 1800;                 //We need some room to keep full control at full throttle.
    esc_1 = throttle - pid_output_pitch + pid_output_roll - pid_output_yaw; //Calculate the pulse for esc 1 (front-
right - CCW)
    esc_2 = throttle + pid_output_pitch + pid_output_roll + pid_output_yaw; //Calculate the pulse for esc 2 (rear-
right - CW)
    esc_3 = throttle + pid_output_pitch - pid_output_roll - pid_output_yaw; //Calculate the pulse for esc 3 (rear-left
- CCW)
    esc_4 = throttle - pid_output_pitch - pid_output_roll + pid_output_yaw; //Calculate the pulse for esc 4 (front-
left - CW)

    if (battery_voltage < 1240 && battery_voltage > 800){   //Is the battery connected?
      esc_1 += esc_1 * ((1240 - battery_voltage)/(float)3500);  //Compensate the esc-1 pulse for voltage drop.
      esc_2 += esc_2 * ((1240 - battery_voltage)/(float)3500);  //Compensate the esc-2 pulse for voltage drop.
      esc_3 += esc_3 * ((1240 - battery_voltage)/(float)3500);  //Compensate the esc-3 pulse for voltage drop.
      esc_4 += esc_4 * ((1240 - battery_voltage)/(float)3500);  //Compensate the esc-4 pulse for voltage drop.
    }

    if (esc_1 < 1200) esc_1 = 1200;                       //Keep the motors running.
    if (esc_2 < 1200) esc_2 = 1200;                       //Keep the motors running.
    if (esc_3 < 1200) esc_3 = 1200;                       //Keep the motors running.
    if (esc_4 < 1200) esc_4 = 1200;                       //Keep the motors running.

    if(esc_1 > 2000)esc_1 = 2000;                         //Limit the esc-1 pulse to 2000us.
    if(esc_2 > 2000)esc_2 = 2000;                         //Limit the esc-2 pulse to 2000us.
    if(esc_3 > 2000)esc_3 = 2000;                         //Limit the esc-3 pulse to 2000us.
    if(esc_4 > 2000)esc_4 = 2000;                         //Limit the esc-4 pulse to 2000us.
  }

  else{
    esc_1 = 1000;                                         //If start is not 2 keep a 1000us pulse for ess-1.
    esc_2 = 1000;                                         //If start is not 2 keep a 1000us pulse for ess-2.
    esc_3 = 1000;                                         //If start is not 2 keep a 1000us pulse for ess-3.
```

```
  esc_4 = 1000;                                          //If start is not 2 keep a 1000us pulse for ess-4.
 }

 //All the information for controlling the motor's is available.
 //The refresh rate is 250Hz. That means the esc's need there pulse every 4ms.
 while(micros() - loop_timer < 4000);                    //We wait until 4000us are passed.
 loop_timer = micros();                                  //Set the timer for the next loop.

 PORTD |= B11110000;                                     //Set digital outputs 4,5,6 and 7 high.
 timer_channel_1 = esc_1 + loop_timer;                   //Calculate the time of the faling edge of the esc-1
pulse.
 timer_channel_2 = esc_2 + loop_timer;                   //Calculate the time of the faling edge of the esc-2
pulse.
 timer_channel_3 = esc_3 + loop_timer;                   //Calculate the time of the faling edge of the esc-3
pulse.
 timer_channel_4 = esc_4 + loop_timer;                   //Calculate the time of the faling edge of the esc-4
pulse.

 while(PORTD >= 16){                                     //Stay in this loop until output 4,5,6 and 7 are low.
  esc_loop_timer = micros();                             //Read the current time.
  if(timer_channel_1 <= esc_loop_timer)PORTD &= B11101111;          //Set digital output 4 to low if the time is
expired.
  if(timer_channel_2 <= esc_loop_timer)PORTD &= B11011111;          //Set digital output 5 to low if the time is
expired.
  if(timer_channel_3 <= esc_loop_timer)PORTD &= B10111111;          //Set digital output 6 to low if the time is
expired.
  if(timer_channel_4 <= esc_loop_timer)PORTD &= B01111111;          //Set digital output 7 to low if the time is
expired.
 }
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//////////////
//This routine is called every time input 8, 9, 10 or 11 changed state
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//////////////
ISR(PCINT0_vect){
 current_time = micros();
 //Channel 1=========================================
 if(PINB & B00000001){                                   //Is input 8 high?
  if(last_channel_1 == 0){                               //Input 8 changed from 0 to 1
   last_channel_1 = 1;                                   //Remember current input state
   timer_1 = current_time;                               //Set timer_1 to current_time
  }
 }
 else if(last_channel_1 == 1){                           //Input 8 is not high and changed from 1 to 0
  last_channel_1 = 0;                                    //Remember current input state
  receiver_input_channel_1 = current_time - timer_1;     //Channel 1 is current_time - timer_1
 }
 //Channel 2=========================================
 if(PINB & B00000010 ){                                  //Is input 9 high?
  if(last_channel_2 == 0){                               //Input 9 changed from 0 to 1
   last_channel_2 = 1;                                   //Remember current input state
```

```
    timer_2 = current_time;                          //Set timer_2 to current_time
   }
  }
  else if(last_channel_2 == 1){                      //Input 9 is not high and changed from 1 to 0
   last_channel_2 = 0;                               //Remember current input state
   receiver_input_channel_2 = current_time - timer_2;        //Channel 2 is current_time - timer_2
  }
  //Channel 3=========================================
  if(PINB & B00000100 ){                             //Is input 10 high?
   if(last_channel_3 == 0){                          //Input 10 changed from 0 to 1
    last_channel_3 = 1;                              //Remember current input state
    timer_3 = current_time;                          //Set timer_3 to current_time
   }
  }
  else if(last_channel_3 == 1){                      //Input 10 is not high and changed from 1 to 0
   last_channel_3 = 0;                               //Remember current input state
   receiver_input_channel_3 = current_time - timer_3;        //Channel 3 is current_time - timer_3

  }
  //Channel 4=========================================
  if(PINB & B00001000 ){                             //Is input 11 high?
   if(last_channel_4 == 0){                          //Input 11 changed from 0 to 1
    last_channel_4 = 1;                              //Remember current input state
    timer_4 = current_time;                          //Set timer_4 to current_time
   }
  }
  else if(last_channel_4 == 1){                      //Input 11 is not high and changed from 1 to 0
   last_channel_4 = 0;                               //Remember current input state
   receiver_input_channel_4 = current_time - timer_4;        //Channel 4 is current_time - timer_4
  }
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//////////////
//Subroutine for reading the gyro
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//////////////
void gyro_signalen(){
  Wire.beginTransmission(105);                       //Start communication with the gyro (adress 1101001)
  Wire.write(168);                                   //Start reading @ register 28h and auto increment with every read
  Wire.endTransmission();                            //End the transmission
  Wire.requestFrom(105, 6);                          //Request 6 bytes from the gyro
  while(Wire.available() < 6);                       //Wait until the 6 bytes are received
  lowByte = Wire.read();                             //First received byte is the low part of the angular data
  highByte = Wire.read();                            //Second received byte is the high part of the angular data
  gyro_roll = ((highByte<<8)|lowByte);               //Multiply highByte by 256 (shift left by 8) and ad lowByte
  if(cal_int == 2000)gyro_roll -= gyro_roll_cal;     //Only compensate after the calibration
  lowByte = Wire.read();                             //First received byte is the low part of the angular data
  highByte = Wire.read();                            //Second received byte is the high part of the angular data
  gyro_pitch = ((highByte<<8)|lowByte);              //Multiply highByte by 256 (shift left by 8) and ad lowByte
  gyro_pitch *= -1;                                  //Invert axis
  if(cal_int == 2000)gyro_pitch -= gyro_pitch_cal;   //Only compensate after the calibration
  lowByte = Wire.read();                             //First received byte is the low part of the angular data
```

```
  highByte = Wire.read();                                   //Second received byte is the high part of the angular data
  gyro_yaw = ((highByte<<8)|lowByte);                       //Multiply highByte by 256 (shift left by 8) and ad lowByte
  gyro_yaw *= -1;                                           //Invert axis
  if(cal_int == 2000)gyro_yaw -= gyro_yaw_cal;              //Only compensate after the calibration
}

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//////////////
//Subroutine for calculating pid outputs
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//////////////
//The PID controllers are explained in part 5 of the YMFC-3D video session:
//www.youtube.com/watch?v=JBvnB0279-Q

void calculate_pid(){
  //Roll calculations
  pid_error_temp = gyro_roll_input - pid_roll_setpoint;
  pid_i_mem_roll += pid_i_gain_roll * pid_error_temp;
  if(pid_i_mem_roll > pid_max_roll)pid_i_mem_roll = pid_max_roll;
  else if(pid_i_mem_roll < pid_max_roll * -1)pid_i_mem_roll = pid_max_roll * -1;

  pid_output_roll = pid_p_gain_roll * pid_error_temp + pid_i_mem_roll + pid_d_gain_roll * (pid_error_temp -
pid_last_roll_d_error);
  if(pid_output_roll > pid_max_roll)pid_output_roll = pid_max_roll;
  else if(pid_output_roll < pid_max_roll * -1)pid_output_roll = pid_max_roll * -1;

  pid_last_roll_d_error = pid_error_temp;

  //Pitch calculations
  pid_error_temp = gyro_pitch_input - pid_pitch_setpoint;
  pid_i_mem_pitch += pid_i_gain_pitch * pid_error_temp;
  if(pid_i_mem_pitch > pid_max_pitch)pid_i_mem_pitch = pid_max_pitch;
  else if(pid_i_mem_pitch < pid_max_pitch * -1)pid_i_mem_pitch = pid_max_pitch * -1;

  pid_output_pitch = pid_p_gain_pitch * pid_error_temp + pid_i_mem_pitch + pid_d_gain_pitch *
(pid_error_temp - pid_last_pitch_d_error);
  if(pid_output_pitch > pid_max_pitch)pid_output_pitch = pid_max_pitch;
  else if(pid_output_pitch < pid_max_pitch * -1)pid_output_pitch = pid_max_pitch * -1;

  pid_last_pitch_d_error = pid_error_temp;

  //Yaw calculations
  pid_error_temp = gyro_yaw_input - pid_yaw_setpoint;
  pid_i_mem_yaw += pid_i_gain_yaw * pid_error_temp;
  if(pid_i_mem_yaw > pid_max_yaw)pid_i_mem_yaw = pid_max_yaw;
  else if(pid_i_mem_yaw < pid_max_yaw * -1)pid_i_mem_yaw = pid_max_yaw * -1;


  pid_last_yaw_d_error = pid_error_temp;
}
```

REFERENCES

1. Introduction to Arduino by Alan G. Smith, September 30, 2011,available in www.introtoarduino.com
2. History of Modern Computing byPaul E. Ceruzzi, Boston, MS: MIT Press
3. The Electronics Handbook by J. C. Whitaker, 1996, CRC Press
4. Arduino and kinect projects by Enrique Ramos Melgar and Ciriaco Castro Diez
5. Beginning Arduino by Michael McRoberts ,2nd Edition
6. Beginning C for Arduino,Ph.D. Jack Purdum, Copyright © 2012 by Jack Purdum, Available in http://www.apress.com/bulk-sales.
7. Photodiodes - From Fundamentals to Applications, Edited by Ilgu Yun, ISBN 978-953-51-0895-5, Publisher: InTech, Chapters published December 19, 2012
8. Arduino - Software. 2013. Arduino - Software. [ONLINE] Available at: http://arduino.cc/en/Main/Software.
9. http://arduino.cc/
10. http://www.webopedia.com
11. https://learn.sparkfun.com
12. http://www.ukessays.com
13. http://www.kenleung.ca/portfolio/arduino-a-brief-history-3/
14. http://www.thefinancialexpress-bd.com/old/more.php?news_id=93501&date=2012-01-12
15. www.tigerprints.clemson.edu/cgi/viewcontent.cgi?article=1088&context=all_theses
16. www.osioptoelectronics.com
17. http://www.radio-electronics.com/info/data/semicond/photo_diode/operation-theory.php
18. http://electronics.stackexchange.com/questions/86470
19. http://www.technologystudent.com/elec1/opamp1.htm
20. research.cs.tamu.edu/prism/lectures/iss/iss_l5.pdf
21. eleceng.dit.ie/frank/msp430/microcontrollers
22. http://www.engineersgarage.com/electronic-components
23. www.uknowledge.uky.edu/cgi/viewcontent.cgi?article=1097&context=gradschool_theses
24. http://shop.evilmadscientist.com/productsmenu/partsmenu/509
25. http://www.pcbheaven.com/wikipages/How_Relays_Work/