First Flight Drone

For First Time Flyers

Thomas Jandebeur, Electrical Engineering

Sponsored by CECS, University of Evansville

Project Advisor: Dr. Mark Mitchell

April 26, 2019

Evansville, Indiana

Acknowledgements

A big thanks to Dr. Mitchell for being my advisor and allowing me to use the mounting board from the optics lab for the initial flight testing and a very special thanks to Jeff for helping me create the aluminum mounting bracket.

Table of Contents

List of Figures

List of Tables

I. Introduction

With the popularity of Unmanned Arial Vehicles, UAVs or more commonly called Drones, quickly rising, more people are wanting to learn how to fly. This comes with inherent risks no matter where or what someone is flying. Learning to fly indoors can be easier as dangers such as wind and trees are no longer a problem and cold weather and rain are not a concern. New issues, however, arise from tight spaces with walls and ceilings becoming the new danger. I built a drone using off the shelf parts with a custom flight controller to allow for easy indoor flight. I originally planned to add distance sensors but ultimately ran out of time due to complications with the tuning process.

II. Design Approach

I decided to keep the design simple by using standard parts and focusing on just creating the flight controller. This was done to prevent having to balance weight and power and to avoid being caught in a cycle attempting to balance the two. Doing so allowed me to complete the project and have a flyable drone.

A. Hardware Design

The drone is using a 250 frame (250mm between opposite motors) made from carbon fiber for increased durability and reduced weight. This is paired with 2300kv 2204 brushless motors with matching 15A Electronic Speed Controllers (ESCs) and 5030 dual blade propellers. Everything is powered through a Matek Power Distribution Board (PDB) connected to a TATTU 4S 45C 1300mAh Lithium Polymer (LiPo) battery that is monitored by LiPo voltage monitor. The flight controller is programmed onto a STM32F407G-DISC1 microcontroller utilizing an Adafruit 9-DOF IMU. Everything is controlled by the Flysky FS-i6X 2.4GHz Remote Control (RC) transmitter with matching receiver. Figure 1 to the left is the basic flowchart for the hardware listed above. The Discovery board was chosen due to my previous experience with the board and its abundance of available timers and ports for the most flexibility. The RC transmitter is using Mode 2, the most common and popular of the four RC modes, which places throttle/yaw on the left stick and pitch/roll on the right stick.

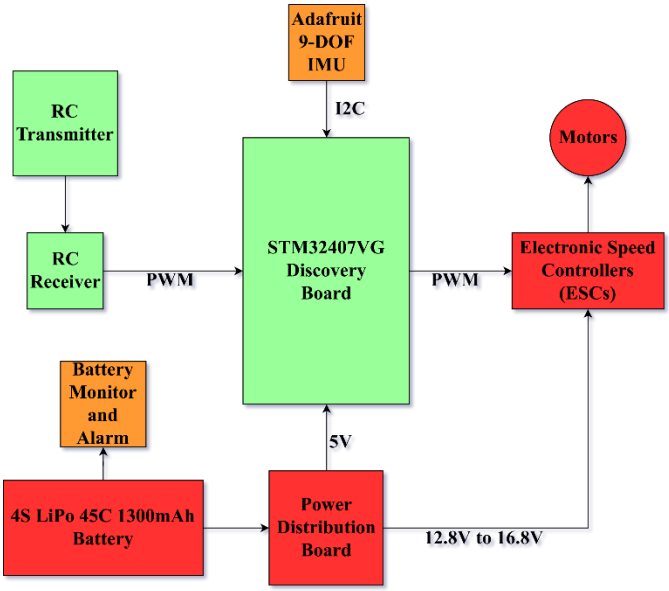

Figure 1 Basic hardware flow chat

Inputs							
	Channel 1	Channel 2	Channel 3	Channel 4	Channel 5	Channel 6	I^2C
Purpose	Roll	Pitch	Throttle	Yaw	Arm & Kill	Mode	IMU
Timer	T5C2	T3C2	T3C3	T3C4	T3C1/T5C1	T5C3	N/A
Port	A1	B5	C8	C9	B4/A0	A2	B10/B11
Pulse	PWM		Period: 20ms		High Time: 1-2ms		5µs CLK

Table 1 Microcontroller Inputs

Outputs				
	Motor A	Motor B	Motor C	Motor D
Position	Front-Left	Front-Right	Back-Left	Back-Right
Timer	T4C1	T4C2	T4C3	T4C4
Port	B6	B7	D14	D15
PWM	Period: 500µs		High Time: 125-250µs	

Table 2 Microcontroller Outputs

The RC reciever has a 6-channel output using standard PWM for RC shown in Table 1 above. The Arm and Kill switch is connected to the first channel of both input timers due to a characteristic of the timers that requires channel one to have to the PWM signal with the highest pulse width or the other three channels on that timer will overflow. The IMU is wrapped in aluminium foil to shield against interference as any signal over 2MHz near the board causes the crystal to lock up and freezes the I^2C protocol stopping the flight controller. It is mounted to the center of the frame with nylon nuts with rubber washers to isolate the sensors from high-frequency vibrations from the motors. The microcontroller is mounted to an aluminum bracket cut from a small sheet of thin aluminum, folded and attached to the frame using standoffs. The front of the bracket is also used as a battery holder, along with a velcro strap, to hold the battery down to the frame with the standoffs acting as a backstop and marker so the battery is always in the same position when replaced. The microcontroller is also attached to two breakout boards, one on each side, to allow it to be easily replaced as well as closed-cell foam on top of the microcontroller for additional padding to prevent damage during a crash.

B. Software Design

For this project, I had to teach myself the I^2C protocol and how to implement it. After discovering that interference was causing the IMU to freeze, I implemented a counter that would detect if the IMU was no longer responding and restart the I^2C. There are three modes of flight as shown in Figure 2 on the next page. Expert, or more commonly called ‘Acro Mode,’ uses only the gyroscope and the Rate PID controller with the stick inputs controlling how fast the drone rotates. Intermediate, also called ‘Angle Mode,’ uses the accelerometer with the Angle PID controller as well as the gyroscope and Rate PID controller with the pitch and roll inputs controlling how far the drone rotates. Yaw control in intermediate still only uses the Rate PID controller. Beginner is the same as Intermediate but with pitch and roll more limited in how far

each can rotate as well as having the throttle range limited to half. Each axis, pitch, yaw, and roll, has individual Rate PID controllers with separate constants. Pitch and roll also have separate

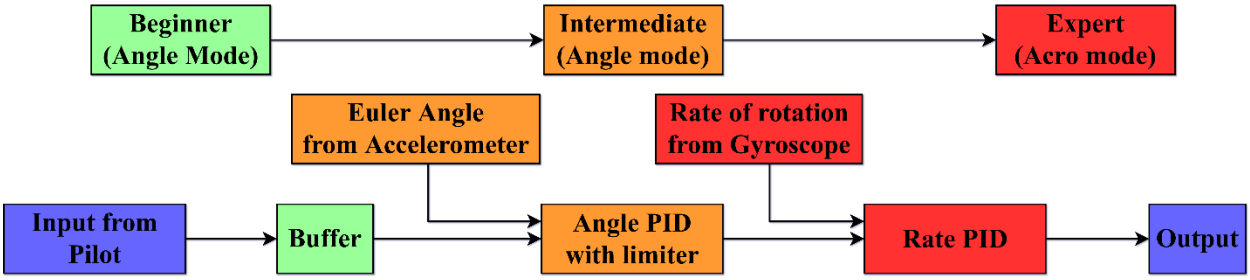

Figure 2 PID Flowchart

Angle PID controllers. Each axis also has individual sensitivity settings with pitch and roll having different sensitivity settings for both Intermediate and Beginner. The Gyroscope has three variables, one for each axis, to correct for drift and the Accelerometer has two variables, one for pitch and one for roll, to correct for offset. The brushless motors also have a mininum throttle to stay in sync with the ESCs. If the input falls below this limit followed by a jump above it then the motor no longer spins properly and stays out of sync until the mininum throttle is matched again. All of these account for a total of 28 individual variables, some affected by or even dependent on another, that I had to tune seperatly. This was the bulk of my project; finding the sweet spot for all of them. This compounded by the fact that every time something physically changes on the drone the PID constants change. I had to completely restart tuning from scratch twice; once when I broke the original mounting bracet and replaced it with the aluminum bracket, and once when I fixed a back-and-forth drifting issue that would cause the drone to crash by increasing the gain on the IMU inputs.

III. Results

I created a drone that anyone with zero flight experience and minimal instruction can fly indoors. It has three modes of flight, Beginner, Intermediate, and Expert, designed to allow someone to learn both Acro and Angle control mode and fly any drone. The drone is durable and can survive many crashes and can be flown both indoors and outdoors.

Figure 3 Final Product

Appendix A

```
#include "stm32f407vg.h"
#include <float.h>

int main()
{
	{//timer/port/I2C ini
	{//timer/port clock
	RCC_AHB1ENR |= (1 << 3);			//port D clock
	RCC_AHB1ENR |= (1 << 2);			//port C clock
	RCC_AHB1ENR |= (1 << 1);			//port B clock
	RCC_AHB1ENR |= 1;				//port A clock
	RCC_APB1ENR |= (1 << 3);			//timer 5 clock
	RCC_APB1ENR |= (1 << 2);			//timer 4 clock
	RCC_APB1ENR |= (1 << 1);			//timer 3 clock
	RCC_APB1ENR |= (1 << 22);			//enable I2C2
	RCC_APB1RSTR |= (1 << 22);			//reset I2C2
	RCC_APB1RSTR &= ~(1 << 22);
	NVICISER0 |= (1 << 30);
	}
	{//timer 4
	TIM4_DIER |= 1;					//Update interrupt enabled
	TIM4_DIER |= (1 << 6);				//Trigger interrupt enabled
	TIM4_CR1 |= (1 <<7);				//enable auto-reload preload
	TIM4_CR2 |= 0x20;
	TIM4_PSC = 0;					//Timer 4 prescaler
	TIM4_ARR = 8000;				//Timer 4 auto reload every 2.5ms; 1000 tics = 1ms
	TIM4_CCMR2 |= 0x6868;				//Output compare 4 mode 3, preload enabled
	TIM4_CCMR1 |= 0x6868;
	TIM4_SCMR |= 0x0080;
	TIM4_CCER |= 0x1111;				//Capture/Compare output enable
	TIM4_CCR1 = 250;
	TIM4_CCR2 = 250;
	TIM4_CCR3 = 250;
	TIM4_CCR4 = 250;				//output value
	TIM4_EGR |= 1;					//timer reset
	TIM4_CR1 |= 1;					//enable timer
	}
	{//timer 3
	TIM3_DIER |= 1;					//Update interrupt enabled
	TIM3_DIER |= (1 << 6);				//Trigger interrupt enabled
	TIM3_DIER |= (1 << 1);				//Trigger interrupt enabled
	TIM3_CR1 |= (1 <<7);				//enable auto-reload preload
	TIM3_PSC =15;					//Timer 3 prescaler
	TIM3_ARR = 20000;				//Timer 3 auto reload every 20ms; 1000 tics = 1ms
	TIM3_CCMR2 |= 0x0101;				//Output compare pwm input
	TIM3_CCMR1 |= 0x0101;
	TIM3_CCER |= 0x3333;				//Capture/Compare input
	TIM3_SCMR |= 0x0044;
```

```
TIM3_CR1 |= 1;                          //enable timer
}
{//timer 5
TIM5_DIER |= 1;                         //Update interrupt enabled
TIM5_DIER |= (1 << 6);                  //Trigger interrupt enabled
TIM5_DIER |= (1 << 2);                  //Trigger interrupt enabled
TIM5_CR1 |= (1 <<7);                    //enable auto-reload preload
TIM5_PSC =15;                           //Timer 5 prescaler
TIM5_ARR = 20000;                       //Timer 5 auto reload every 20ms; 1000 tics = 1ms
TIM5_CCMR2 |= 0x0101;                   //Output compare pwm input
TIM5_CCMR1 |= 0x0101;
TIM5_CCER |= 0x3333;                    //Capture/Compare input
TIM5_SCMR |= 0x0044;
TIM5_CR1 |= 1;                          //enable timer
}
{//port A
GPIOA_PUPDR &= 0x00000000;              //clear
GPIOA_PUPDR |= 0x0000A000;
GPIOA_AFRL &= 0xFFFF0000;
GPIOA_AFRL |= 0x00002222;               //PA0-3 AF2
GPIOA_MODER &= 0xFFFF0F00;
GPIOA_MODER |= 0x000050AA;              //PA0-3 AF
}
{//port B
GPIOB_PUPDR &= 0x00000000;              //clear
GPIOB_PUPDR |= 0x000A0000;
GPIOB_AFRL &= 0x0000FFFF;
GPIOB_AFRL |= 0x22220000;               //PB4-7 AF2
GPIOB_MODER &= 0xFFF000FF;
GPIOB_MODER |= 0x0005AA00;              //PB4-7
GPIOB_MODER |= 0x00A00000;              //PB10-11
GPIOB_OTYPER |= 0x00000C00;             //PB10-11 Open Drain
GPIOB_AFRH |= 0x00004400;               //PB10-11 AF4
}
{//port C
GPIOC_PUPDR &= 0x00000000;              //clear
GPIOC_PUPDR |= 0x0000000A;
GPIOC_AFRH &= 0xFFFFFF00;
GPIOC_AFRH |= 0x00000022;               //PC8,9 AF2
GPIOC_MODER &= 0xFFF0FFF0;
GPIOC_MODER |= 0x000A0005;
}
{//port D
GPIOD_PUPDR &= 0x00000000;
GPIOD_PUPDR |= 0x00A04800;
GPIOD_AFRH &= 0x00FFFFFF;
GPIOD_AFRH |= 0x22000000;               //PD15, 14 AF2
GPIOD_MODER &= 0x0F0F00F0;
GPIOD_MODER |= 0xA0501401;              //PD14,15AF; 5,6,10,11 ground
}

}
```

```
//Variables
volatile unsigned char GyrXL, GyrXM, GyrYL, GyrYM, GyrZL, GyrZM, EulXL, EulXM, EulYL, EulYM;
volatile signed short  GyrPitch, GyrRoll, GyrYaw, EulRoll, EulPitch;
volatile float RollF, PitchF, GyrPitchF, GyrRollF, GyrYawF;
volatile unsigned short MotorA = 0, MotorB = 0, MotorC = 0, MotorD = 0;

volatile float ThrottleOutput = 0;
volatile float PitchRateCurrent = 0, PitchRateDesired = 0, PitchRateError = 0, PitchRateErrorOld = 0,
PitchRateIntegral = 0, PitchRateDerivative = 0, PitchRateOutput = 0;
volatile float RollRateCurrent = 0, RollRateDesired = 0, RollRateError = 0, RollRateErrorOld = 0,
RollRateIntegral = 0, RollRateDerivative = 0, RollRateOutput = 0;
volatile float YawRateCurrent = 0, YawRateDesired = 0, YawRateError = 0, YawRateErrorOld = 0,
YawRateIntegral = 0, YawRateDerivative = 0, YawRateOutput = 0;
volatile float PitchAngleCurrent = 0, PitchAngleDesired = 0, PitchAngleError = 0, PitchAngleErrorOld = 0,
PitchAngleIntegral = 0, PitchAngleDerivative = 0, PitchAngleOutput = 0;
volatile float RollAngleCurrent = 0, RollAngleDesired = 0, RollAngleError = 0, RollAngleErrorOld = 0,
RollAngleIntegral = 0, RollAngleDerivative = 0, RollAngleOutput = 0;

volatile const float PitchRateKp = .003, PitchRateKi = .009, PitchRateKd = .0000518;
volatile const float RollRateKp = .002, RollRateKi = .005, RollRateKd = .00004295;
volatile const float YawRateKp = .2, YawRateKi = .001, YawRateKd = 0;
volatile const float PitchAngleKp = 2.5, PitchAngleKi = 0, PitchAngleKd = .01;
volatile const float RollAngleKp = 2.5, RollAngleKi = 0, RollAngleKd = .01;

volatile const float PitchRateSens = 250, RollRateSens = 250, YawRateSens = 10, dt = .002;
volatile float PitchAngleSens = 0, RollAngleSens = 0;
volatile const float EulPitchTrim = -70, EulRollTrim = -28;
volatile const float GyrPitchTrim = 50, GyrRollTrim = -100, GyrYawTrim = 90;
volatile short PitchInput = 0, RollInput = 0, ModeSelect = 0;
volatile int c = 0, maxC = 0;

restart:
{//I2C
I2C2_CR1 &= 0x0000;                     //disable i2c
I2C2_CR2 |= 0x070A;                     //10MHz clk, enable interrupts A
I2C2_CCR |= 0x0032;                     //5us clk periods 0032 4B  14
I2C2_TRISE &= 0x0000;
I2C2_TRISE |= 0x0006;                   //500ns rise time 6
I2C2_FLTR |= 0x0010;                    //enable analog filter
I2C2_OAR1 |= 0x4055;                    //address = 0101010
I2C2_CR1 |= 0x0401;                     //enable i2c and ack
}

{//IMU ini
I2C2_CR1 |= (1 << 8);                   //Start bit
while((I2C2_SR1 & 0x0001) != 0x0001);   //wait for start bit
I2C2_DR = (0x28<<1);                    //sensor address LSB=0 to transmit
while((I2C2_SR1 & 0x0002) != 0x0002);   //wait for address sent
if(I2C2_SR2 != 1);                      //read to clear ADDR
I2C2_DR = 0x3D;                         //OPR_MODE reg
while((I2C2_SR1 & (1<<7)) != (1<<7));   //wait for TxE
```

```
I2C2_DR = 0xC;                          //NDOF mode
while((I2C2_SR1 & (1<<7)) != (1<<7));   //wait for TxE
I2C2_CR1 |= (1 << 9);                   //Stop bit
}

while(1)
{
        {//I2C
                if(c>maxC)
                        maxC = c;
                c = 0;
                        I2C2_CR1 |= (1<<10);                    //enable ack
                        I2C2_CR1 |= (1 << 8);                   //Start bit
                        while((I2C2_SR1 & 0x0001) != 0x0001)    //wait for start bit
                        {
                                c++;
                                if(c > 500)
                                        goto restart;
                        }
                        c = 0;
                        I2C2_DR = (0x28<<1);                    //sensor address LSB=0 to transmit
                        while((I2C2_SR1 & 0x0002) != 0x0002)    //wait for address sent
                        {
                                c++;
                                if(c > 500)
                                        goto restart;
                        }
                        c = 0;
                        if(I2C2_SR2 != 1);                      //read to clear ADDR
                        I2C2_DR = 0x14;                         //GYR DATA X L reg
                        while((I2C2_SR1 & (1<<7)) != (1<<7))    //wait for TxE
                        {
                                c++;
                                if(c > 500)
                                        goto restart;
                        }
                        c = 0;
                        I2C2_CR1 |= (1 << 8);                   //Start bit
                        while((I2C2_SR1 & 0x0001) != 0x0001)    //wait for start bit
                        {
                                c++;
                                if(c > 500)
                                        goto restart;
                        }
                        c = 0;
                        I2C2_DR = ((0x28<<1) + 1);          //temp sensor address LSB=1 to recieve
                        while((I2C2_SR1 & 0x0002) != 0x0002)    //wait for address sent
                        {
                                c++;
                                if(c > 500)
                                        goto restart;
                        }
                        c = 0;
```

```
if(I2C2_SR2 == 0x0001);                         //read to clear ADDR
while((I2C2_SR1 & 0x0040) != 0x0040)                    //wait for RxNE
{
        c++;
        if(c > 500)
                goto restart;
}
GyrXL = I2C2_DR;
while((I2C2_SR1 & 0x0040) != 0x0040)                    //wait for RxNE
{
        c++;
        if(c > 500)
                goto restart;
}
c = 0;
GyrXM = I2C2_DR;
while((I2C2_SR1 & 0x0040) != 0x0040)                    //wait for RxNE
{
        c++;
        if(c > 500)
                goto restart;
}
c = 0;
GyrYL = I2C2_DR;
while((I2C2_SR1 & 0x0040) != 0x0040)                    //wait for RxNE
{
        c++;
        if(c > 500)
                goto restart;
}
c = 0;
GyrYM = I2C2_DR;
while((I2C2_SR1 & 0x0040) != 0x0040)                    //wait for RxNE
{
        c++;
        if(c > 500)
                goto restart;
}
c = 0;
GyrZL = I2C2_DR;
I2C2_CR1 &= 0xFBFF;                             //Nack
I2C2_CR1 |= (1 << 9);                           //Stop bit
while((I2C2_SR1 & 0x0040) != 0x0040)            //wait for RxNE
{
        c++;
        if(c > 500)
                goto restart;
}
c = 0;
GyrZM = I2C2_DR;

I2C2_CR1 |= (1<<10);                            //enable ack
I2C2_CR1 |= (1 << 8);                           //Start bit
```

```
while((I2C2_SR1 & 0x0001) != 0x0001)        //wait for start bit
{
        c++;
        if(c > 500)
                goto restart;
}
c = 0;
I2C2_DR = (0x28<<1);                        //sensor address LSB=0 to transmit
while((I2C2_SR1 & 0x0002) != 0x0002)        //wait for address sent
{
        c++;
        if(c > 500)
                goto restart;
}
c = 0;
if(I2C2_SR2 != 1);                          //read to clear ADDR
I2C2_DR = 0x1C;                             //Acc DATA X L reg
while((I2C2_SR1 & (1<<7)) != (1<<7))        //wait for TxE
{
        c++;
        if(c > 500)
                goto restart;
}
c = 0;
I2C2_CR1 |= (1 << 8);                       //Start bit
while((I2C2_SR1 & 0x0001) != 0x0001)        //wait for start bit
{
        c++;
        if(c > 500)
                goto restart;
}
c = 0;
I2C2_DR = ((0x28<<1) + 1);          //temp sensor address LSB=1 to recieve
while((I2C2_SR1 & 0x0002) != 0x0002)        //wait for address sent
{
        c++;
        if(c > 500)
                goto restart;
}
c = 0;
if(I2C2_SR2 == 0x0001);                     //read to clear ADDR
while((I2C2_SR1 & 0x0040) != 0x0040)        //wait for RxNE
{
        c++;
        if(c > 500)
                goto restart;
}
c = 0;
EulXL = I2C2_DR;
while((I2C2_SR1 & 0x0040) != 0x0040)                //wait for RxNE
{
        c++;
        if(c > 500)
```

```
                                        goto restart;
                                }
                                c = 0;
                                EulXM = I2C2_DR;
                                while((I2C2_SR1 & 0x0040) != 0x0040)                    //wait for RxNE
                                {
                                        c++;
                                        if(c > 500)
                                                goto restart;
                                }
                                c = 0;
                                EulYL = I2C2_DR;
                                I2C2_CR1 &= 0xFBFF;                             //Nack
                                I2C2_CR1 |= (1 << 9);                           //Stop bit
                                while((I2C2_SR1 & 0x0040) != 0x0040)        //wait for RxNE
                                {
                                        c++;
                                        if(c > 500)
                                                goto restart;
                                }
                                c = 0;
                                EulYM = I2C2_DR;
                        }
                                GyrPitch = ((GyrXM << 8) + GyrXL) + GyrPitchTrim;
                                GyrRoll = (GyrYM << 8) + GyrYL + GyrRollTrim;
                                GyrYaw = (GyrZM << 8) + GyrZL + GyrYawTrim;
                                EulRoll = -(((EulXM << 8) + EulXL) + EulRollTrim);
                                EulPitch = -(((EulYM << 8) + EulYL) + EulPitchTrim);
                                RollF = (float)EulRoll*32;
                                PitchF = (float)EulPitch*32;
                                GyrPitchF = (float)GyrPitch*32; //16
                                GyrRollF = (float)GyrRoll*32;
                                GyrYawF = -(float)GyrYaw;

                ModeSelect = TIM5_CCR3;

                PitchInput = TIM3_CCR2;
                RollInput = TIM5_CCR2;

                if(ModeSelect > 1750)                                   //Acro Mode
                {
                        PitchRateCurrent = GyrPitchF;
                        PitchRateDesired = ((-((int)PitchInput - 1500) - -500) * ((PitchRateSens * 500) -
(PitchRateSens * -500)) / (500 - -500)) + (PitchRateSens * -500);
                        PitchRateError = PitchRateDesired - PitchRateCurrent;
                        PitchRateIntegral = PitchRateIntegral + (PitchRateError * dt);
                        PitchRateDerivative = (PitchRateError - PitchRateErrorOld) / dt;
                        PitchRateOutput = (PitchRateKp * PitchRateError) + (PitchRateKi * PitchRateIntegral) +
(PitchRateKd * PitchRateDerivative);
                        if(PitchRateOutput > 2000)
                                PitchRateOutput = 2000;
                        else if(PitchRateOutput < - 2000)
                                PitchRateOutput = -2000;
```

```
                        PitchRateErrorOld = PitchRateError;

                        RollRateCurrent = GyrRollF;
                        RollRateDesired = ((((int)RollInput - 1500) - -500) * ((RollRateSens * 500) - (RollRateSens
* -500)) / (500 - -500)) + (RollRateSens * -500);
                        RollRateError = RollRateDesired - RollRateCurrent;
                        RollRateIntegral = RollRateIntegral + (RollRateError * dt);
                        RollRateDerivative = (RollRateError - RollRateErrorOld) / dt;
                        RollRateOutput = (RollRateKp * RollRateError) + (RollRateKi * RollRateIntegral) +
(RollRateKd * RollRateDerivative);
                        if(RollRateOutput > 2000)
                                RollRateOutput = 2000;
                        else if(RollRateOutput < - 2000)
                                RollRateOutput = -2000;
                        RollRateErrorOld = RollRateError;

                        YawRateCurrent = GyrYawF;
                        YawRateDesired = ((((int)TIM3_CCR4 - 1500) - -500) * ((YawRateSens * 500) -
(YawRateSens * -500)) / (500 - -500)) + (YawRateSens * -500);
                        YawRateError = YawRateDesired - YawRateCurrent;
                        YawRateIntegral = YawRateIntegral + (YawRateError * dt);
                        YawRateDerivative = (YawRateError - YawRateErrorOld) / dt;
                        YawRateOutput = (YawRateKp * YawRateError) + (YawRateKi * YawRateIntegral) +
(YawRateKd * YawRateDerivative);
                        if(YawRateOutput > 2000)
                                YawRateOutput = 2000;
                        else if(YawRateOutput < - 2000)
                                YawRateOutput = -2000;
                        YawRateErrorOld = YawRateError;

                        ThrottleOutput = (((TIM3_CCR3 - 1000) * (4000 - 2000)) / (2000 - 1000)) + 2000;
                }

                if(ModeSelect < 1750)                           //Angle Mode
                {
                        if(ModeSelect < 1250)                   //Beginner Mode
                        {
                                PitchAngleSens = 30;
                                RollAngleSens = 30;
                        }
                        else
                        {
                                PitchAngleSens = 75;
                                RollAngleSens = 75;
                        }
                        PitchAngleCurrent = PitchF;
                        PitchAngleDesired = ((-((int)PitchInput - 1500) - -500) * ((PitchAngleSens * 500) -
(PitchAngleSens * -500)) / (500 - -500)) + (PitchAngleSens * -500);
                        PitchAngleError = PitchAngleDesired - PitchAngleCurrent;
                        PitchAngleIntegral = PitchAngleIntegral + (PitchAngleError * dt);
                        PitchAngleDerivative = (PitchAngleError - PitchAngleErrorOld) / dt;
                        PitchAngleOutput = (PitchAngleKp * PitchAngleError) + (PitchAngleKi *
PitchAngleIntegral) + (PitchAngleKd * PitchAngleDerivative);
```

```
                        PitchAngleErrorOld = PitchAngleError;

                        RollAngleCurrent = RollF;
                        RollAngleDesired = ((((int)RollInput - 1500) - -500) * ((RollAngleSens * 500) -
(RollAngleSens * -500)) / (500 - -500)) + (RollAngleSens * -500);
                        RollAngleError = RollAngleDesired - RollAngleCurrent;
                        RollAngleIntegral = RollAngleIntegral + (RollAngleError * dt);
                        RollAngleDerivative = (RollAngleError - RollAngleErrorOld) / dt;
                        RollAngleOutput = (RollAngleKp * RollAngleError) + (RollAngleKi * RollAngleIntegral) +
(RollAngleKd * RollAngleDerivative);
                        RollAngleErrorOld = RollAngleError;

                        PitchRateCurrent = GyrPitchF;
                        PitchRateDesired = PitchAngleOutput;
                        PitchRateError = PitchRateDesired - PitchRateCurrent;
                        PitchRateIntegral = PitchRateIntegral + (PitchRateError * dt);
                        PitchRateDerivative = (PitchRateError - PitchRateErrorOld) / dt;
                        PitchRateOutput = (PitchRateKp * PitchRateError) + (PitchRateKi * PitchRateIntegral) +
(PitchRateKd * PitchRateDerivative);
                        if(PitchRateOutput > 2000)
                                PitchRateOutput = 2000;
                        else if(PitchRateOutput < - 2000)
                                PitchRateOutput = -2000;
                        PitchRateErrorOld = PitchRateError;

                        RollRateCurrent = GyrRollF;
                        RollRateDesired = RollAngleOutput;
                        RollRateError = RollRateDesired - RollRateCurrent;
                        RollRateIntegral = RollRateIntegral + (RollRateError * dt);
                        RollRateDerivative = (RollRateError - RollRateErrorOld) / dt;
                        RollRateOutput = (RollRateKp * RollRateError) + (RollRateKi * RollRateIntegral) +
(RollRateKd * RollRateDerivative);
                        if(RollRateOutput > 2000)
                                RollRateOutput = 2000;
                        else if(RollRateOutput < - 2000)
                                RollRateOutput = -2000;
                        RollRateErrorOld = RollRateError;

                        YawRateCurrent = GyrYawF;
                        YawRateDesired = ((((int)TIM3_CCR4 - 1500) - -500) * ((YawRateSens * 500) -
(YawRateSens * -500)) / (500 - -500)) + (YawRateSens * -500);
                        YawRateError = YawRateDesired - YawRateCurrent;
                        YawRateIntegral = YawRateIntegral + (YawRateError * dt);
                        YawRateDerivative = (YawRateError - YawRateErrorOld) / dt;
                        YawRateOutput = (YawRateKp * YawRateError) + (YawRateKi * YawRateIntegral) +
(YawRateKd * YawRateDerivative);
                        if(YawRateOutput > 2000)
                                YawRateOutput = 2000;
                        else if(YawRateOutput < - 2000)
                                YawRateOutput = -2000;
                        YawRateErrorOld = YawRateError;

                        ThrottleOutput = (((TIM3_CCR3 - 1000) * (4000 - 2000)) / (2000 - 1000)) + 2000;
```

```
		if(ModeSelect < 1250)				//Beginner Mode
		{
			ThrottleOutput = (((TIM3_CCR3 - 1000) * (3500 - 2000)) / (2000 - 1000)) + 2000;
		}
}

MotorA = ThrottleOutput + PitchRateOutput + RollRateOutput - YawRateOutput + 100;
MotorB = ThrottleOutput + PitchRateOutput - RollRateOutput + YawRateOutput + 100;
MotorC = ThrottleOutput - PitchRateOutput + RollRateOutput + YawRateOutput + 100;
MotorD = ThrottleOutput - PitchRateOutput - RollRateOutput - YawRateOutput + 100;

if(MotorA < 2100)
		MotorA = 2100;
else if(MotorA > 3980)
		MotorA = 3980;
if(MotorB < 2100)
		MotorB = 2100;
else if(MotorB > 3980)
		MotorB = 3980;
if(MotorC < 2100)
		MotorC = 2100;
else if(MotorC > 3980)
		MotorC = 3980;
if(MotorD < 2100)
		MotorD = 2100;
else if(MotorD > 3980)
		MotorD = 3980;


if(TIM3_CCR1 < 1500)			//Kill Switch
{
		MotorA = 1850;
		MotorB = 1850;
		MotorC = 1850;
		MotorD = 1850;
		PitchAngleIntegral = 0;
		RollAngleIntegral = 0;
		PitchRateIntegral = 0;
		RollRateIntegral = 0;
		YawRateIntegral = 0;
}

if(ThrottleOutput < 2050)		//Prevent Integral Windup while on the ground
{
		PitchAngleIntegral = 0;
		RollAngleIntegral = 0;
		PitchRateIntegral = 0;
		RollRateIntegral = 0;
		YawRateIntegral = 0;
}

TIM4_CCR1 = MotorA;
TIM4_CCR2 = MotorB;
```

```
                TIM4_CCR3 = MotorD;
                TIM4_CCR4 = MotorC;
                GPIOD_ODR = GPIOD_ODR ^ (1);                    //Looptime test

        }
}

void TIM5_IRQHandler()
{
        TIM5_SR &= 0xFFFE;                                      //clear interrupt flag
        TIM5_CR1 |= 1;
}

void TIM4_IRQHandler()
{
        TIM4_SR &= 0xFFFE;                                      //clear interrupt flag
        TIM4_CR1 |= 1;
}

void TIM3_IRQHandler()
{
        TIM3_SR &= 0xFFFE;                                      //clear interrupt flag
        TIM3_CR1 |= 1;
}
```