

```

////////////////////////////////////
//Terms of use
////////////////////////////////////
//THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
//IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
//FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
//AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
//LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
//OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
//THE SOFTWARE.
////////////////////////////////////
//Safety note
////////////////////////////////////
//Always remove the propellers and stay away from the motors unless you
//are 100% certain of what you are doing.
////////////////////////////////////

#include <Wire.h>                //Include the Wire.h library so we can communicate with the gyro.
#include <EEPROM.h>              //Include the EEPROM.h library so we can store information onto the
EEPROM

////////////////////////////////////
//PID gain and limit settings
////////////////////////////////////
float pid_p_gain_roll = 0;      //Gain setting for the roll P-controller
float pid_i_gain_roll = 0.04;   //Gain setting for the roll I-controller
float pid_d_gain_roll = 18.0;   //Gain setting for the roll D-controller
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 (+/-)

boolean auto_level = true;      //Auto level on (true) or off (false)

////////////////////////////////////
//Declaring global variables
////////////////////////////////////
byte last_channel_1, last_channel_2, last_channel_3, last_channel_4;
byte eeprom_data[36];
byte highByte, lowByte;
volatile 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;

```

```
long acc_x, acc_y, acc_z, acc_total_vector;
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;
unsigned long loop_timer;
double gyro_pitch, gyro_roll, gyro_yaw;
double gyro_axis_cal[4];
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;
float angle_roll_acc, angle_pitch_acc, angle_pitch, angle_roll;
boolean gyro_angles_set;
```

[illegible]

```

    delayMicroseconds(1000);           //Wait 1000us.
    PORTD &= B00001111;               //Set digital poort 4, 5, 6 and 7 low.
    delayMicroseconds(3000);           //Wait 3000us.
}

//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_axis_cal[1] += gyro_axis[1];           //Ad roll value to gyro_roll_cal.
    gyro_axis_cal[2] += gyro_axis[2];           //Ad pitch value to gyro_pitch_cal.
    gyro_axis_cal[3] += gyro_axis[3];           //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_axis_cal[1] /= 2000;           //Divide the roll total by 2000.
gyro_axis_cal[2] /= 2000;           //Divide the pitch total by 2000.
gyro_axis_cal[3] /= 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.

//Wait until the receiver is active and the throttle is set to the lower position.
while(receiver_input_channel_3 < 990 || receiver_input_channel_3 > 1020 || receiver_input_channel_4 <
1400){
    receiver_input_channel_3 = convert_receiver_channel(3);           //Convert the actual receiver signals for
throttle to the standard 1000 - 2000us
    receiver_input_channel_4 = convert_receiver_channel(4);           //Convert the actual receiver signals for
yaw to the standard 1000 - 2000us
    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.
    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;

loop_timer = micros();                    //Set the timer for the next loop.

//When everything is done, turn off the led.
digitalWrite(12,LOW);                     //Turn off the warning led.
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//Main program loop
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void loop(){

    //65.5 = 1 deg/sec (check the datasheet of the MPU-6050 for more information).
    gyro_roll_input = (gyro_roll_input * 0.7) + ((gyro_roll / 65.5) * 0.3); //Gyro pid input is deg/sec.
    gyro_pitch_input = (gyro_pitch_input * 0.7) + ((gyro_pitch / 65.5) * 0.3); //Gyro pid input is deg/sec.
    gyro_yaw_input = (gyro_yaw_input * 0.7) + ((gyro_yaw / 65.5) * 0.3);    //Gyro pid input is deg/sec.

    //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
    //This is the added IMU code from the videos:
    //https://youtu.be/4BoIE8YQwM8
    //https://youtu.be/j-kE0AMEWy4
    //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

    //Gyro angle calculations
    //0.0000611 = 1 / (250Hz / 65.5)
    angle_pitch += gyro_pitch * 0.0000611; //Calculate the traveled pitch angle and add this
to the angle_pitch variable.
    angle_roll += gyro_roll * 0.0000611;   //Calculate the traveled roll angle and add this to
the angle_roll variable.

    //0.000001066 = 0.0000611 * (3.142(PI) / 180degr) The Arduino sin function is in radians
    angle_pitch -= angle_roll * sin(gyro_yaw * 0.000001066); //If the IMU has yawed transfer the roll
angle to the pitch angel.
    angle_roll += angle_pitch * sin(gyro_yaw * 0.000001066); //If the IMU has yawed transfer the pitch
angle to the roll angel.

    //Accelerometer angle calculations
    acc_total_vector = sqrt((acc_x*acc_x)+(acc_y*acc_y)+(acc_z*acc_z)); //Calculate the total accelerometer
vector.

    if(abs(acc_y) < acc_total_vector){ //Prevent the asin function to produce a NaN

```

```

    angle_pitch_acc = asin((float)acc_y/acc_total_vector)* 57.296;    //Calculate the pitch angle.
}
if(abs(acc_x) < acc_total_vector){    //Prevent the asin function to produce a NaN
    angle_roll_acc = asin((float)acc_x/acc_total_vector)* -57.296;    //Calculate the roll angle.
}

//Place the MPU-6050 spirit level and note the values in the following two lines for calibration.
angle_pitch_acc -= 0.0;    //Accelerometer calibration value for pitch.
angle_roll_acc -= 0.0;    //Accelerometer calibration value for roll.

angle_pitch = angle_pitch * 0.9996 + angle_pitch_acc * 0.0004;    //Correct the drift of the gyro pitch
angle with the accelerometer pitch angle.
angle_roll = angle_roll * 0.9996 + angle_roll_acc * 0.0004;    //Correct the drift of the gyro roll angle
with the accelerometer roll angle.

pitch_level_adjust = angle_pitch * 15;    //Calculate the pitch angle correction
roll_level_adjust = angle_roll * 15;    //Calculate the roll angle correction

if(!auto_level){    //If the quadcopter is not in auto-level mode
    pitch_level_adjust = 0;    //Set the pitch angle correction to zero.
    roll_level_adjust = 0;    //Set the roll angle correction to zero.
}

//For starting the motors: throttle low and yaw left (step 1).
if(receiver_input_channel_3 < 1050 && receiver_input_channel_4 < 1050)start = 1;
//When yaw stick is back in the center position start the motors (step 2).
if(start == 1 && receiver_input_channel_3 < 1050 && receiver_input_channel_4 > 1450){
    start = 2;

    angle_pitch = angle_pitch_acc;    //Set the gyro pitch angle equal to the accelerometer
pitch angle when the quadcopter is started.
    angle_roll = angle_roll_acc;    //Set the gyro roll angle equal to the accelerometer
roll angle when the quadcopter is started.
    gyro_angles_set = true;    //Set the IMU started flag.

    //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;

```

```

else if(receiver_input_channel_1 < 1492)pid_roll_setpoint = receiver_input_channel_1 - 1492;

pid_roll_setpoint -= roll_level_adjust;           //Subtract the angle correction from the
standardized receiver roll input value.
pid_roll_setpoint /= 3.0;                         //Divide the setpoint for the PID roll controller by 3 to
get angles in degrees.

//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;
else if(receiver_input_channel_2 < 1492)pid_pitch_setpoint = receiver_input_channel_2 - 1492;

pid_pitch_setpoint -= pitch_level_adjust;         //Subtract the angle correction from the
standardized receiver pitch input value.
pid_pitch_setpoint /= 3.0;                       //Divide the setpoint for the PID pitch controller by 3
to get angles in degrees.

//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;
}

calculate_pid();                                //PID inputs are known. So we can calculate the pid
output.

//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 < 1000 && 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)

```
/*Serial.println("a");  
Serial.println(esc_1);  
Serial.println(esc_2);  
Serial.println(esc_3);  
Serial.println(esc_4);  
Serial.println("b");  
Serial.println(pid_output_pitch);  
Serial.println(pid_output_roll);  
Serial.println(pid_output_yaw);  
Serial.println(throttle);*/
```

```
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 < 1100) esc_1 = 1100;           //Keep the motors running.  
if (esc_2 < 1100) esc_2 = 1100;           //Keep the motors running.  
if (esc_3 < 1100) esc_3 = 1100;           //Keep the motors running.  
if (esc_4 < 1100) esc_4 = 1100;           //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.  
}
```

```
/*  
if(start == 2){  
Serial.println("c");  
Serial.print(battery_voltage,0);  
Serial.print(";");  
Serial.print(gyro_roll,0);  
Serial.print(";");  
Serial.print(gyro_pitch,0);  
Serial.print(";");  
Serial.println(gyro_yaw,0);
```

```

}*/

/////////////////////////////////////////////////////////////////
//Creating the pulses for the ESC's is explained in this video:
//https://youtu.be/fqEkVcqxtU8
/////////////////////////////////////////////////////////////////

//!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
//Because of the angle calculation the loop time is getting very important. If the loop time is
//longer or shorter than 4000us the angle calculation is off. If you modify the code make sure
//that the loop time is still 4000us and no longer! More information can be found on
//the Q&A page:
//!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

if(micros() - loop_timer > 4050)digitalWrite(12, HIGH);          //Turn on the LED if the loop time
exceeds 4050us.

//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.

//There is always 1000us of spare time. So let's do something usefull that is very time consuming.
//Get the current gyro and receiver data and scale it to degrees per second for the pid calculations.
gyro_signalen();

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. This is used to read the receiver signals.

```


//More information about this subroutine can be found in this video:

//<https://youtu.be/bENjl1KQbvo>

//

```
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[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[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[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[4] = current_time - timer_4;
        //Channel 4 is current_time - timer_4.
    }
}
```

```

/////////////////////////////////////////////////////////////////
//Subroutine for reading the gyro
/////////////////////////////////////////////////////////////////
void gyro_signalen(){
  //Read the MPU-6050
  if(eeprom_data[31] == 1){
    Wire.beginTransmission(gyro_address);                                //Start communication with the gyro.
    Wire.write(0x3B);                                                    //Start reading @ register 43h and auto increment with
    every read.
    Wire.endTransmission();                                              //End the transmission.
    Wire.requestFrom(gyro_address,14);                                  //Request 14 bytes from the gyro.

    receiver_input_channel_1 = convert_receiver_channel(1);             //Convert the actual receiver signals for
    pitch to the standard 1000 - 2000us.
    receiver_input_channel_2 = convert_receiver_channel(2);             //Convert the actual receiver signals for
    roll to the standard 1000 - 2000us.
    receiver_input_channel_3 = convert_receiver_channel(3);             //Convert the actual receiver signals for
    throttle to the standard 1000 - 2000us.
    receiver_input_channel_4 = convert_receiver_channel(4);             //Convert the actual receiver signals for
    yaw to the standard 1000 - 2000us.

    while(Wire.available() < 14);                                       //Wait until the 14 bytes are received.
    acc_axis[1] = Wire.read()<<8|Wire.read();                          //Add the low and high byte to the acc_x
    variable.
    acc_axis[2] = Wire.read()<<8|Wire.read();                          //Add the low and high byte to the acc_y
    variable.
    acc_axis[3] = Wire.read()<<8|Wire.read();                          //Add the low and high byte to the acc_z
    variable.
    temperature = Wire.read()<<8|Wire.read();                          //Add the low and high byte to the temperature
    variable.
    gyro_axis[1] = Wire.read()<<8|Wire.read();                          //Read high and low part of the angular data.
    gyro_axis[2] = Wire.read()<<8|Wire.read();                          //Read high and low part of the angular data.
    gyro_axis[3] = Wire.read()<<8|Wire.read();                          //Read high and low part of the angular data.
  }

  if(cal_int == 2000){
    gyro_axis[1] -= gyro_axis_cal[1];                                    //Only compensate after the calibration.
    gyro_axis[2] -= gyro_axis_cal[2];                                    //Only compensate after the calibration.
    gyro_axis[3] -= gyro_axis_cal[3];                                    //Only compensate after the calibration.
  }
  gyro_roll = gyro_axis[eeprom_data[28] & 0b00000011];                //Set gyro_roll to the correct axis that
  was stored in the EEPROM.
  if(eeprom_data[28] & 0b10000000)gyro_roll *= -1;                    //Invert gyro_roll if the MSB of EEPROM
  bit 28 is set.
  gyro_pitch = gyro_axis[eeprom_data[29] & 0b00000011];                //Set gyro_pitch to the correct axis that
  was stored in the EEPROM.
  if(eeprom_data[29] & 0b10000000)gyro_pitch *= -1;                    //Invert gyro_pitch if the MSB of
  EEPROM bit 29 is set.
  gyro_yaw = gyro_axis[eeprom_data[30] & 0b00000011];                //Set gyro_yaw to the correct axis that
  was stored in the EEPROM.
  if(eeprom_data[30] & 0b10000000)gyro_yaw *= -1;                    //Invert gyro_yaw if the MSB of
  EEPROM bit 30 is set.

```

```

    acc_x = acc_axis[eprom_data[29] & 0b00000011];           //Set acc_x to the correct axis that was
stored in the EEPROM.                                       //Invert acc_x if the MSB of EEPROM bit
    if(eprom_data[29] & 0b10000000)acc_x *= -1;             //Set acc_y to the correct axis that was
29 is set.                                                  //Invert acc_y if the MSB of EEPROM bit
    acc_y = acc_axis[eprom_data[28] & 0b00000011];           //Set acc_z to the correct axis that was
stored in the EEPROM.                                       //Invert acc_z if the MSB of EEPROM bit
    if(eprom_data[28] & 0b10000000)acc_y *= -1;             //Set acc_z to the correct axis that was
28 is set.                                                  //Invert acc_z if the MSB of EEPROM bit
    acc_z = acc_axis[eprom_data[30] & 0b00000011];           //Set acc_z to the correct axis that was
stored in the EEPROM.                                       //Invert acc_z if the MSB of EEPROM bit
    if(eprom_data[30] & 0b10000000)acc_z *= -1;             //Invert acc_z if the MSB of EEPROM bit
30 is set.
}

```

```

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

```

```

//Subroutine for calculating pid outputs

```

```

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

```

```

//The PID controllers are explained in part 5 of the YMFC-3D video session:

```

```

//https://youtu.be/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_output_yaw = pid_p_gain_yaw * pid_error_temp + pid_i_mem_yaw + pid_d_gain_yaw * (pid_error_temp
- pid_last_yaw_d_error);
if(pid_output_yaw > pid_max_yaw)pid_output_yaw = pid_max_yaw;
else if(pid_output_yaw < pid_max_yaw * -1)pid_output_yaw = pid_max_yaw * -1;

pid_last_yaw_d_error = pid_error_temp;
}

//This part converts the actual receiver signals to a standardized 1000 – 1500 – 2000 microsecond value.
//The stored data in the EEPROM is used.
int convert_receiver_channel(byte function){
    byte channel, reverse;                                //First we declare some local variables
    int low, center, high, actual;
    int difference;

    channel = eeprom_data[function + 23] & 0b00000111;    //What channel corresponds with the
specific function
    if(eeprom_data[function + 23] & 0b10000000)reverse = 1;    //Reverse channel when most
significant bit is set
    else reverse = 0;    //If the most significant is not set there is no reverse

    actual = receiver_input[channel];    //Read the actual receiver value for the
corresponding function
    low = (eeprom_data[channel * 2 + 15] << 8) | eeprom_data[channel * 2 + 14]; //Store the low value for the
specific receiver input channel
    center = (eeprom_data[channel * 2 - 1] << 8) | eeprom_data[channel * 2 - 2]; //Store the center value for the
specific receiver input channel
    high = (eeprom_data[channel * 2 + 7] << 8) | eeprom_data[channel * 2 + 6]; //Store the high value for the
specific receiver input channel

    if(actual < center){    //The actual receiver value is lower than the center value
        if(actual < low)actual = low;    //Limit the lowest value to the value that was
detected during setup
        difference = ((long)(center - actual) * (long)500) / (center - low);    //Calculate and scale the actual value to
a 1000 - 2000us value
        if(reverse == 1)return 1500 + difference;    //If the channel is reversed
        else return 1500 - difference;    //If the channel is not reversed
    }
    else if(actual > center){    //The actual receiver value is higher than
the center value
        if(actual > high)actual = high;    //Limit the lowest value to the value that was
detected during setup
        difference = ((long)(actual - center) * (long)500) / (high - center);    //Calculate and scale the actual value to
a 1000 - 2000us value
        if(reverse == 1)return 1500 - difference;    //If the channel is reversed
        else return 1500 + difference;    //If the channel is not reversed
    }
    else return 1500;
}

```

```

void set_gyro_registers(){
  //Setup the MPU-6050
  if(eeprom_data[31] == 1){
    Wire.beginTransmission(gyro_address);                                //Start communication with the address found
    during search.
    Wire.write(0x6B);                                                    //We want to write to the PWR_MGMT_1 register (6B
    hex)
    Wire.write(0x00);                                                    //Set the register bits as 00000000 to activate the gyro
    Wire.endTransmission();                                              //End the transmission with the gyro.

    Wire.beginTransmission(gyro_address);                                //Start communication with the address found
    during search.
    Wire.write(0x1B);                                                    //We want to write to the GYRO_CONFIG register
    (1B hex)
    Wire.write(0x08);                                                    //Set the register bits as 00001000 (500dps full scale)
    Wire.endTransmission();                                              //End the transmission with the gyro

    Wire.beginTransmission(gyro_address);                                //Start communication with the address found
    during search.
    Wire.write(0x1C);                                                    //We want to write to the ACCEL_CONFIG register
    (1A hex)
    Wire.write(0x10);                                                    //Set the register bits as 00010000 (+/- 8g full scale
    range)
    Wire.endTransmission();                                              //End the transmission with the gyro

    //Let's perform a random register check to see if the values are written correct
    Wire.beginTransmission(gyro_address);                                //Start communication with the address found
    during search
    Wire.write(0x1B);                                                    //Start reading @ register 0x1B
    Wire.endTransmission();                                              //End the transmission
    Wire.requestFrom(gyro_address, 1);                                    //Request 1 bytes from the gyro
    while(Wire.available() < 1);                                         //Wait until the 6 bytes are received
    if(Wire.read() != 0x08){                                              //Check if the value is 0x08
      digitalWrite(12,HIGH);                                              //Turn on the warning led
      while(1)delay(10);                                                 //Stay in this loop for ever
    }

    Wire.beginTransmission(gyro_address);                                //Start communication with the address found
    during search
    Wire.write(0x1A);                                                    //We want to write to the CONFIG register (1A hex)
    Wire.write(0x03);                                                    //Set the register bits as 00000011 (Set Digital Low
    Pass Filter to ~43Hz)
    Wire.endTransmission();                                              //End the transmission with the gyro

  }
}

```