```
////////////////////////////////////////////////////////////////////////////////////
 //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.
 ////////////////////////////////////////////////////////////////////////////////////



 ////////////////////////////////////////////////////////////////////////////////////
 //The program will start in calibration mode.
 //Send the following characters / numbers via the serial monitor to change the mode
 //
 //r = print receiver signals.
 //a = print quadcopter angles.
 //1 = check rotation / vibrations for motor 1 (right front CCW).
 //2 = check rotation / vibrations for motor 2 (right rear CW).
 //3 = check rotation / vibrations for motor 3 (left rear CCW).
 //4 = check rotation / vibrations for motor 4 (left front CW).
 //5 = check vibrations for all motors together.


 #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

 //Declaring global variables
 byte last_channel_1, last_channel_2, last_channel_3, last_channel_4;
 byte eeprom_data[36], start, data;
 boolean new_function_request,first_angle;
 volatile int receiver_input_channel_1, receiver_input_channel_2, receiver_input_channel_3,
receiver_input_channel_4;
 int esc_1, esc_2, esc_3, esc_4;
 int counter_channel_1, counter_channel_2, counter_channel_3, counter_channel_4;
 int receiver_input[5];
 int loop_counter, gyro_address, vibration_counter;
 int temperature;
 long acc_x, acc_y, acc_z, acc_total_vector[20], acc_av_vector, vibration_total_result;
 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;
```

```
int acc_axis[4], gyro_axis[4];
double gyro_pitch, gyro_roll, gyro_yaw;
float angle_roll_acc, angle_pitch_acc, angle_pitch, angle_roll;
int cal_int;
double gyro_axis_cal[4];

//Setup routine
void setup(){
  Serial.begin(57600);                                                    //Start the serial port.
  Wire.begin();                                                           //Start the wire library as master
  TWBR = 12;                                                              //Set the I2C clock speed to 400kHz.

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

  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.

  for(data = 0; data <= 35; data++)eeprom_data[data] = EEPROM.read(data);          //Read EEPROM for
faster data access

  gyro_address = eeprom_data[32];                                         //Store the gyro address in the variable.

  set_gyro_registers();                                                   //Set the specific gyro registers.

  //Check the EEPROM signature to make sure that the setup program is executed.
  while(eeprom_data[33] != 'J' || eeprom_data[34] != 'M' || eeprom_data[35] != 'B'){
    delay(500);                                                           //Wait for 500ms.
    digitalWrite(12, !digitalRead(12));                                   //Change the led status to indicate error.
  }
  wait_for_receiver();                                                    //Wait until the receiver is active.
  zero_timer = micros();                                                  //Set the zero_timer for the first loop.

  while(Serial.available())data = Serial.read();                          //Empty the serial buffer.
  data = 0;                                                               //Set the data variable back to zero.
}

//Main program loop
void loop(){
  while(zero_timer + 4000 > micros());                                    //Start the pulse after 4000 micro seconds.
  zero_timer = micros();                                                  //Reset the zero timer.
```

```
  if(Serial.available() > 0){
    data = Serial.read();                                          //Read the incomming byte.
    delay(100);                                                    //Wait for any other bytes to come in
    while(Serial.available() > 0)loop_counter = Serial.read();     //Empty the Serial buffer.
    new_function_request = true;                                   //Set the new request flag.
    loop_counter = 0;                                              //Reset the loop_counter variable.
    cal_int = 0;                                                   //Reset the cal_int variable to undo the calibration.
    start = 0;                                                     //Set start to 0.
    first_angle = false;                                           //Set first_angle to false.
    //Confirm the choice on the serial monitor.
    if(data == 'r')Serial.println("Reading receiver signals.");
    if(data == 'a')Serial.println("Print the quadcopter angles.");
    if(data == 'a')Serial.println("Gyro calibration starts in 2 seconds (don't move the quadcopter).");
    if(data == '1')Serial.println("Test motor 1 (right front CCW.)");
    if(data == '2')Serial.println("Test motor 2 (right rear CW.)");
    if(data == '3')Serial.println("Test motor 3 (left rear CCW.)");
    if(data == '4')Serial.println("Test motor 4 (left front CW.)");
    if(data == '5')Serial.println("Test all motors together");

    //Let's create a small delay so the message stays visible for 2.5 seconds.
    //We don't want the ESC's to beep and have to send a 1000us pulse to the ESC's.
    for(vibration_counter = 0; vibration_counter < 625; vibration_counter++){          //Do this loop 625 times
      delay(3);                                                    //Wait 3000us.
      esc_1 = 1000;                                                //Set the pulse for ESC 1 to 1000us.
      esc_2 = 1000;                                                //Set the pulse for ESC 1 to 1000us.
      esc_3 = 1000;                                                //Set the pulse for ESC 1 to 1000us.
      esc_4 = 1000;                                                //Set the pulse for ESC 1 to 1000us.
      esc_pulse_output();                                          //Send the ESC control pulses.
    }
    vibration_counter = 0;                                         //Reset the vibration_counter variable.
  }

  receiver_input_channel_3 = convert_receiver_channel(3);          //Convert the actual receiver
signals for throttle to the standard 1000 - 2000us.
  if(receiver_input_channel_3 < 1025)new_function_request = false;          //If the throttle is in the lowest
position set the request flag to false.


  ////////////////////////////////////////////////////////////////////////////////////
  //Run the ESC calibration program to start with.
  ////////////////////////////////////////////////////////////////////////////////////
  if(data == 0 && new_function_request == false){                  //Only start the calibration mode at
first start.
    receiver_input_channel_3 = convert_receiver_channel(3);        //Convert the actual receiver
signals for throttle to the standard 1000 - 2000us.
    esc_1 = receiver_input_channel_3;                              //Set the pulse for motor 1 equal to the
throttle channel.
    esc_2 = receiver_input_channel_3;                              //Set the pulse for motor 2 equal to the
throttle channel.
    esc_3 = receiver_input_channel_3;                              //Set the pulse for motor 3 equal to the
throttle channel.
    esc_4 = receiver_input_channel_3;                              //Set the pulse for motor 4 equal to the
```

```
throttle channel.
    esc_pulse_output();                                          //Send the ESC control pulses.
  }

  ////////////////////////////////////////////////////////////////////////////////////
  //When user sends a 'r' print the receiver signals.
  ////////////////////////////////////////////////////////////////////////////////////
  if(data == 'r'){
    loop_counter ++;                                             //Increase the loop_counter variable.
    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.

    if(loop_counter == 125){                                     //Print the receiver values when the
loop_counter variable equals 250.
      print_signals();                                           //Print the receiver values on the serial monitor.
      loop_counter = 0;                                          //Reset the loop_counter variable.
    }

    //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;
    //Stopping the motors: throttle low and yaw right.
    if(start == 2 && receiver_input_channel_3 < 1050 && receiver_input_channel_4 > 1950)start = 0;

    esc_1 = 1000;                                                //Set the pulse for ESC 1 to 1000us.
    esc_2 = 1000;                                                //Set the pulse for ESC 1 to 1000us.
    esc_3 = 1000;                                                //Set the pulse for ESC 1 to 1000us.
    esc_4 = 1000;                                                //Set the pulse for ESC 1 to 1000us.
    esc_pulse_output();                                          //Send the ESC control pulses.
  }

  ////////////////////////////////////////////////////////////////////////////////////
  //When user sends a '1, 2, 3, 4 or 5 test the motors.
  ////////////////////////////////////////////////////////////////////////////////////
  if(data == '1' || data == '2' || data == '3' || data == '4' || data == '5'){       //If motor 1, 2, 3 or 4 is selected by the
user.
    loop_counter ++;                                             //Add 1 to the loop_counter variable.
    if(new_function_request == true && loop_counter == 250){     //Wait for the throttle to be set
to 0.
      Serial.print("Set throttle to 1000 (low). It's now set to: ");     //Print message on the serial monitor.
      Serial.println(receiver_input_channel_3);                  //Print the actual throttle position.
      loop_counter = 0;                                          //Reset the loop_counter variable.
    }
    if(new_function_request == false){                           //When the throttle was in the lowest
position do this.
```

```
    receiver_input_channel_3 = convert_receiver_channel(3);                  //Convert the actual receiver signals for throttle to the standard 1000 - 2000us.
    if(data == '1' || data == '5')esc_1 = receiver_input_channel_3;          //If motor 1 is requested set the pulse for motor 1 equal to the throttle channel.
    else esc_1 = 1000;                                                       //If motor 1 is not requested set the pulse for the ESC to 1000us (off).
    if(data == '2' || data == '5')esc_2 = receiver_input_channel_3;          //If motor 2 is requested set the pulse for motor 1 equal to the throttle channel.
    else esc_2 = 1000;                                                       //If motor 2 is not requested set the pulse for the ESC to 1000us (off).
    if(data == '3' || data == '5')esc_3 = receiver_input_channel_3;          //If motor 3 is requested set the pulse for motor 1 equal to the throttle channel.
    else esc_3 = 1000;                                                       //If motor 3 is not requested set the pulse for the ESC to 1000us (off).
    if(data == '4' || data == '5')esc_4 = receiver_input_channel_3;          //If motor 4 is requested set the pulse for motor 1 equal to the throttle channel.
    else esc_4 = 1000;                                                       //If motor 4 is not requested set the pulse for the ESC to 1000us (off).

    esc_pulse_output();                                                      //Send the ESC control pulses.

    //For balancing the propellors it's possible to use the accelerometer to measure the vibrations.
    if(eeprom_data[31] == 1){                                                //The MPU-6050 is installed
      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,6);                                      //Request 6 bytes from the gyro.
      while(Wire.available() < 6);                                           //Wait until the 6 bytes are received.
      acc_x = Wire.read()<<8|Wire.read();                                    //Add the low and high byte to the acc_x variable.
      acc_y = Wire.read()<<8|Wire.read();                                    //Add the low and high byte to the acc_y variable.
      acc_z = Wire.read()<<8|Wire.read();                                    //Add the low and high byte to the acc_z variable.

      acc_total_vector[0] = sqrt((acc_x*acc_x)+(acc_y*acc_y)+(acc_z*acc_z));   //Calculate the total accelerometer vector.

      acc_av_vector = acc_total_vector[0];                                   //Copy the total vector to the accelerometer average vector variable.

      for(start = 16; start > 0; start--){                                   //Do this loop 16 times to create an array of accelrometer vectors.
        acc_total_vector[start] = acc_total_vector[start - 1];               //Shift every variable one position up in the array.
        acc_av_vector += acc_total_vector[start];                            //Add the array value to the acc_av_vector variable.
      }

      acc_av_vector /= 17;                                                   //Divide the acc_av_vector by 17 to get the avarage total accelerometer vector.
```

```
      if(vibration_counter < 20){                                               //If the vibration_counter is less than 20 do this.
        vibration_counter ++;                                                   //Increment the vibration_counter variable.
        vibration_total_result += abs(acc_total_vector[0] - acc_av_vector);          //Add the absolute difference between the avarage vector and current vector to the vibration_total_result variable.
      }
      else{
        vibration_counter = 0;                                                  //If the vibration_counter is equal or larger than 20 do this.
        Serial.println(vibration_total_result/50);                                 //Print the total accelerometer vector divided by 50 on the serial monitor.
        vibration_total_result = 0;                                             //Reset the vibration_total_result variable.
      }
    }
  }
  ////////////////////////////////////////////////////////////////////////////////////
  //When user sends a 'a' display the quadcopter angles.
  ////////////////////////////////////////////////////////////////////////////////////
  if(data == 'a'){

    if(cal_int != 2000){
      Serial.print("Calibrating the gyro");
      //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 % 125 == 0){
          digitalWrite(12, !digitalRead(12));   //Change the led status to indicate calibration.
          Serial.print(".");
        }
        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.
      }
      Serial.println(".");
      //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.
    }
    else{
      ///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.
```

```
      //Let's get the current gyro data.
      gyro_signalen();

      //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[0] = sqrt((acc_x*acc_x)+(acc_y*acc_y)+(acc_z*acc_z));       //Calculate the total accelerometer vector.

      //57.296 = 1 / (3.142 / 180) The Arduino asin function is in radians
      angle_pitch_acc = asin((float)acc_y/acc_total_vector[0])* 57.296;         //Calculate the pitch angle.
      angle_roll_acc = asin((float)acc_x/acc_total_vector[0])* -57.296;         //Calculate the roll angle.

      if(!first_angle){
        angle_pitch = angle_pitch_acc;                                          //Set the pitch angle to the accelerometer angle.
        angle_roll = angle_roll_acc;                                            //Set the roll angle to the accelerometer angle.
        first_angle = true;
      }
      else{
        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.
      }

      //We can't print all the data at once. This takes to long and the angular readings will be off.
      if(loop_counter == 0)Serial.print("Pitch: ");
      if(loop_counter == 1)Serial.print(angle_pitch ,0);
      if(loop_counter == 2)Serial.print(" Roll: ");
      if(loop_counter == 3)Serial.print(angle_roll ,0);
      if(loop_counter == 4)Serial.print(" Yaw: ");
      if(loop_counter == 5)Serial.println(gyro_yaw / 65.5 ,0);

      loop_counter ++;
      if(loop_counter == 60)loop_counter = 0;
    }
  }
}
```

```
//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[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.
  }
}

//Checck if the receiver values are valid within 10 seconds
```

```
 void wait_for_receiver(){
  byte zero = 0;                                                  //Set all bits in the variable zero to 0
  while(zero < 15){                                               //Stay in this loop until the 4 lowest bits are set
    if(receiver_input[1] < 2100 && receiver_input[1] > 900)zero |= 0b00000001;  //Set bit 0 if the receiver pulse
1 is within the 900 - 2100 range
    if(receiver_input[2] < 2100 && receiver_input[2] > 900)zero |= 0b00000010;  //Set bit 1 if the receiver pulse
2 is within the 900 - 2100 range
    if(receiver_input[3] < 2100 && receiver_input[3] > 900)zero |= 0b00000100;  //Set bit 2 if the receiver pulse
3 is within the 900 - 2100 range
    if(receiver_input[4] < 2100 && receiver_input[4] > 900)zero |= 0b00001000;  //Set bit 3 if the receiver pulse
4 is within the 900 - 2100 range
    delay(500);                                                   //Wait 500 milliseconds
  }
 }

 //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 print_signals(){
  Serial.print("Start:");
  Serial.print(start);

  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("  Pitch:");
  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("  Throttle:");
  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);
}

void esc_pulse_output(){
  zero_timer = micros();
  PORTD |= B11110000;                                       //Set port 4, 5, 6 and 7 high at once
  timer_channel_1 = esc_1 + zero_timer;                     //Calculate the time when digital port 4 is set low.
  timer_channel_2 = esc_2 + zero_timer;                     //Calculate the time when digital port 5 is set low.
  timer_channel_3 = esc_3 + zero_timer;                     //Calculate the time when digital port 6 is set low.
  timer_channel_4 = esc_4 + zero_timer;                     //Calculate the time when digital port 7 is set low.

  while(PORTD >= 16){                                       //Execute the loop until digital port 4 to 7 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 4 is set low.
    if(timer_channel_2 <= esc_loop_timer)PORTD &= B11011111;    //When the delay time is expired, digital
port 5 is set low.
    if(timer_channel_3 <= esc_loop_timer)PORTD &= B10111111;    //When the delay time is expired, digital
port 6 is set low.
    if(timer_channel_4 <= esc_loop_timer)PORTD &= B01111111;    //When the delay time is expired, digital
port 7 is set low.
```

```
  }
}

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

  }
}

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