SETUP CODE:

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

//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

//Declaring Global Variables

byte last\_channel\_1, last\_channel\_2, last\_channel\_3, last\_channel\_4;

byte lowByte, highByte, type, gyro\_address, error, clockspeed\_ok;

byte channel\_1\_assign, channel\_2\_assign, channel\_3\_assign, channel\_4\_assign;

byte roll\_axis, pitch\_axis, yaw\_axis;

byte receiver\_check\_byte, gyro\_check\_byte;

volatile int receiver\_input\_channel\_1, receiver\_input\_channel\_2, receiver\_input\_channel\_3, receiver\_input\_channel\_4;

int center\_channel\_1, center\_channel\_2, center\_channel\_3, center\_channel\_4;

int high\_channel\_1, high\_channel\_2, high\_channel\_3, high\_channel\_4;

int low\_channel\_1, low\_channel\_2, low\_channel\_3, low\_channel\_4;

int address, cal\_int;

unsigned long timer, timer\_1, timer\_2, timer\_3, timer\_4, current\_time;

float gyro\_pitch, gyro\_roll, gyro\_yaw;

float gyro\_roll\_cal, gyro\_pitch\_cal, gyro\_yaw\_cal;

//Setup routine

void setup(){

  pinMode(12, OUTPUT);

  //Arduino (Atmega) pins default to inputs, so they don't need to be explicitly declared as inputs

  PCICR |= (1 << PCIE0);    // set PCIE0 to enable PCMSK0 scan

  PCMSK0 |= (1 << PCINT0);  // set PCINT0 (digital input 8) to trigger an interrupt on state change

  PCMSK0 |= (1 << PCINT1);  // set PCINT1 (digital input 9)to trigger an interrupt on state change

  PCMSK0 |= (1 << PCINT2);  // set PCINT2 (digital input 10)to trigger an interrupt on state change

  PCMSK0 |= (1 << PCINT3);  // set PCINT3 (digital input 11)to trigger an interrupt on state change

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

  Serial.begin(57600);      //Start the serial connetion @ 57600bps

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

}

//Main program

void loop(){

  //Show the YMFC-3D V2 intro

  intro();

  Serial.println(F(""));

  Serial.println(F("==================================================="));

  Serial.println(F("System check"));

  Serial.println(F("==================================================="));

  delay(1000);

  Serial.println(F("Checking I2C clock speed."));

  delay(1000);

  TWBR = 12;                      //Set the I2C clock speed to 400kHz.

  #if F\_CPU == 16000000L          //If the clock speed is 16MHz include the next code line when compiling

    clockspeed\_ok = 1;            //Set clockspeed\_ok to 1

  #endif                          //End of if statement

  if(TWBR == 12 && clockspeed\_ok){

    Serial.println(F("I2C clock speed is correctly set to 400kHz."));

  }

  else{

    Serial.println(F("I2C clock speed is not set to 400kHz. (ERROR 8)"));

    error = 1;

  }

  if(error == 0){

    Serial.println(F(""));

    Serial.println(F("==================================================="));

    Serial.println(F("Transmitter setup"));

    Serial.println(F("==================================================="));

    delay(1000);

    Serial.print(F("Checking for valid receiver signals."));

    //Wait 10 seconds until all receiver inputs are valid

    wait\_for\_receiver();

    Serial.println(F(""));

  }

  //Quit the program in case of an error

  if(error == 0){

    delay(2000);

    Serial.println(F("Place all sticks and subtrims in the center position within 10 seconds."));

    for(int i = 9;i > 0;i--){

      delay(1000);

      Serial.print(i);

      Serial.print(" ");

    }

    Serial.println(" ");

    //Store the central stick positions

    center\_channel\_1 = receiver\_input\_channel\_1;

    center\_channel\_2 = receiver\_input\_channel\_2;

    center\_channel\_3 = receiver\_input\_channel\_3;

    center\_channel\_4 = receiver\_input\_channel\_4;

    Serial.println(F(""));

    Serial.println(F("Center positions stored."));

    Serial.print(F("Digital input 08 = "));

    Serial.println(receiver\_input\_channel\_1);

    Serial.print(F("Digital input 09 = "));

    Serial.println(receiver\_input\_channel\_2);

    Serial.print(F("Digital input 10 = "));

    Serial.println(receiver\_input\_channel\_3);

    Serial.print(F("Digital input 11 = "));

    Serial.println(receiver\_input\_channel\_4);

    Serial.println(F(""));

    Serial.println(F(""));

  }

  if(error == 0){

    Serial.println(F("Move the throttle stick to full throttle and back to center"));

    //Check for throttle movement

    check\_receiver\_inputs(1);

    Serial.print(F("Throttle is connected to digital input "));

    Serial.println((channel\_3\_assign & 0b00000111) + 7);

    if(channel\_3\_assign & 0b10000000)Serial.println(F("Channel inverted = yes"));

    else Serial.println(F("Channel inverted = no"));

    wait\_sticks\_zero();

    Serial.println(F(""));

    Serial.println(F(""));

    Serial.println(F("Move the roll stick to simulate left wing up and back to center"));

    //Check for throttle movement

    check\_receiver\_inputs(2);

    Serial.print(F("Roll is connected to digital input "));

    Serial.println((channel\_1\_assign & 0b00000111) + 7);

    if(channel\_1\_assign & 0b10000000)Serial.println(F("Channel inverted = yes"));

    else Serial.println(F("Channel inverted = no"));

    wait\_sticks\_zero();

  }

  if(error == 0){

    Serial.println(F(""));

    Serial.println(F(""));

    Serial.println(F("Move the pitch stick to simulate nose up and back to center"));

    //Check for throttle movement

    check\_receiver\_inputs(3);

    Serial.print(F("Pitch is connected to digital input "));

    Serial.println((channel\_2\_assign & 0b00000111) + 7);

    if(channel\_2\_assign & 0b10000000)Serial.println(F("Channel inverted = yes"));

    else Serial.println(F("Channel inverted = no"));

    wait\_sticks\_zero();

  }

  if(error == 0){

    Serial.println(F(""));

    Serial.println(F(""));

    Serial.println(F("Move the yaw stick to simulate nose right and back to center"));

    //Check for throttle movement

    check\_receiver\_inputs(4);

    Serial.print(F("Yaw is connected to digital input "));

    Serial.println((channel\_4\_assign & 0b00000111) + 7);

    if(channel\_4\_assign & 0b10000000)Serial.println(F("Channel inverted = yes"));

    else Serial.println(F("Channel inverted = no"));

    wait\_sticks\_zero();

  }

  if(error == 0){

    Serial.println(F(""));

    Serial.println(F(""));

    Serial.println(F("Gently move all the sticks simultaneously to their extends"));

    Serial.println(F("When ready put the sticks back in their center positions"));

    //Register the min and max values of the receiver channels

    register\_min\_max();

    Serial.println(F(""));

    Serial.println(F(""));

    Serial.println(F("High, low and center values found during setup"));

    Serial.print(F("Digital input 08 values:"));

    Serial.print(low\_channel\_1);

    Serial.print(F(" - "));

    Serial.print(center\_channel\_1);

    Serial.print(F(" - "));

    Serial.println(high\_channel\_1);

    Serial.print(F("Digital input 09 values:"));

    Serial.print(low\_channel\_2);

    Serial.print(F(" - "));

    Serial.print(center\_channel\_2);

    Serial.print(F(" - "));

    Serial.println(high\_channel\_2);

    Serial.print(F("Digital input 10 values:"));

    Serial.print(low\_channel\_3);

    Serial.print(F(" - "));

    Serial.print(center\_channel\_3);

    Serial.print(F(" - "));

    Serial.println(high\_channel\_3);

    Serial.print(F("Digital input 11 values:"));

    Serial.print(low\_channel\_4);

    Serial.print(F(" - "));

    Serial.print(center\_channel\_4);

    Serial.print(F(" - "));

    Serial.println(high\_channel\_4);

    Serial.println(F("Move stick 'nose up' and back to center to continue"));

    check\_to\_continue();

  }

  if(error == 0){

    //What gyro is connected

    Serial.println(F(""));

    Serial.println(F("==================================================="));

    Serial.println(F("Gyro search"));

    Serial.println(F("==================================================="));

    delay(2000);

    Serial.println(F("Searching for MPU-6050 on address 0x68/104"));

    delay(1000);

    if(search\_gyro(0x68, 0x75) == 0x68){

      Serial.println(F("MPU-6050 found on address 0x68"));

      type = 1;

      gyro\_address = 0x68;

    }

    if(type == 0){

      Serial.println(F("Searching for MPU-6050 on address 0x69/105"));

      delay(1000);

      if(search\_gyro(0x69, 0x75) == 0x68){

        Serial.println(F("MPU-6050 found on address 0x69"));

        type = 1;

        gyro\_address = 0x69;

      }

    }

    if(type == 0){

      Serial.println(F("Searching for L3G4200D on address 0x68/104"));

      delay(1000);

      if(search\_gyro(0x68, 0x0F) == 0xD3){

        Serial.println(F("L3G4200D found on address 0x68"));

        type = 2;

        gyro\_address = 0x68;

      }

    }

    if(type == 0){

      Serial.println(F("Searching for L3G4200D on address 0x69/105"));

      delay(1000);

      if(search\_gyro(0x69, 0x0F) == 0xD3){

        Serial.println(F("L3G4200D found on address 0x69"));

        type = 2;

        gyro\_address = 0x69;

      }

    }

    if(type == 0){

      Serial.println(F("Searching for L3GD20H on address 0x6A/106"));

      delay(1000);

      if(search\_gyro(0x6A, 0x0F) == 0xD7){

        Serial.println(F("L3GD20H found on address 0x6A"));

        type = 3;

        gyro\_address = 0x6A;

      }

    }

    if(type == 0){

     Serial.println(F("Searching for L3GD20H on address 0x6B/107"));

      delay(1000);

      if(search\_gyro(0x6B, 0x0F) == 0xD7){

        Serial.println(F("L3GD20H found on address 0x6B"));

        type = 3;

        gyro\_address = 0x6B;

      }

    }

    if(type == 0){

      Serial.println(F("No gyro device found!!! (ERROR 3)"));

      error = 1;

    }

    else{

      delay(3000);

      Serial.println(F(""));

      Serial.println(F("==================================================="));

      Serial.println(F("Gyro register settings"));

      Serial.println(F("==================================================="));

      start\_gyro(); //Setup the gyro for further use

    }

  }

  //If the gyro is found we can setup the correct gyro axes.

  if(error == 0){

    delay(3000);

    Serial.println(F(""));

    Serial.println(F("==================================================="));

    Serial.println(F("Gyro calibration"));

    Serial.println(F("==================================================="));

    Serial.println(F("Don't move the quadcopter!! Calibration starts in 3 seconds"));

    delay(3000);

    Serial.println(F("Calibrating the gyro, this will take +/- 8 seconds"));

    Serial.print(F("Please wait"));

    //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 % 100 == 0)Serial.print(F("."));                //Print dot to indicate calibration.

      gyro\_signalen();                                           //Read the gyro output.

      gyro\_roll\_cal += gyro\_roll;                                //Ad roll value to gyro\_roll\_cal.

      gyro\_pitch\_cal += gyro\_pitch;                              //Ad pitch value to gyro\_pitch\_cal.

      gyro\_yaw\_cal += gyro\_yaw;                                  //Ad yaw value to gyro\_yaw\_cal.

      delay(4);                                                  //Wait 3 milliseconds before the next loop.

    }

    //Now that we have 2000 measures, we need to devide by 2000 to get the average gyro offset.

    gyro\_roll\_cal /= 2000;                                       //Divide the roll total by 2000.

    gyro\_pitch\_cal /= 2000;                                      //Divide the pitch total by 2000.

    gyro\_yaw\_cal /= 2000;                                        //Divide the yaw total by 2000.

    //Show the calibration results

    Serial.println(F(""));

    Serial.print(F("Axis 1 offset="));

    Serial.println(gyro\_roll\_cal);

    Serial.print(F("Axis 2 offset="));

    Serial.println(gyro\_pitch\_cal);

    Serial.print(F("Axis 3 offset="));

    Serial.println(gyro\_yaw\_cal);

    Serial.println(F(""));

    Serial.println(F("==================================================="));

    Serial.println(F("Gyro axes configuration"));

    Serial.println(F("==================================================="));

    //Detect the left wing up movement

    Serial.println(F("Lift the left side of the quadcopter to a 45 degree angle within 10 seconds"));

    //Check axis movement

    check\_gyro\_axes(1);

    if(error == 0){

      Serial.println(F("OK!"));

      Serial.print(F("Angle detection = "));

      Serial.println(roll\_axis & 0b00000011);

      if(roll\_axis & 0b10000000)Serial.println(F("Axis inverted = yes"));

      else Serial.println(F("Axis inverted = no"));

      Serial.println(F("Put the quadcopter back in its original position"));

      Serial.println(F("Move stick 'nose up' and back to center to continue"));

      check\_to\_continue();

      //Detect the nose up movement

      Serial.println(F(""));

      Serial.println(F(""));

      Serial.println(F("Lift the nose of the quadcopter to a 45 degree angle within 10 seconds"));

      //Check axis movement

      check\_gyro\_axes(2);

    }

    if(error == 0){

      Serial.println(F("OK!"));

      Serial.print(F("Angle detection = "));

      Serial.println(pitch\_axis & 0b00000011);

      if(pitch\_axis & 0b10000000)Serial.println(F("Axis inverted = yes"));

      else Serial.println(F("Axis inverted = no"));

      Serial.println(F("Put the quadcopter back in its original position"));

      Serial.println(F("Move stick 'nose up' and back to center to continue"));

      check\_to\_continue();

      //Detect the nose right movement

      Serial.println(F(""));

      Serial.println(F(""));

      Serial.println(F("Rotate the nose of the quadcopter 45 degree to the right within 10 seconds"));

      //Check axis movement

      check\_gyro\_axes(3);

    }

    if(error == 0){

      Serial.println(F("OK!"));

      Serial.print(F("Angle detection = "));

      Serial.println(yaw\_axis & 0b00000011);

      if(yaw\_axis & 0b10000000)Serial.println(F("Axis inverted = yes"));

      else Serial.println(F("Axis inverted = no"));

      Serial.println(F("Put the quadcopter back in its original position"));

      Serial.println(F("Move stick 'nose up' and back to center to continue"));

      check\_to\_continue();

    }

  }

  if(error == 0){

    Serial.println(F(""));

    Serial.println(F("==================================================="));

    Serial.println(F("LED test"));

    Serial.println(F("==================================================="));

    digitalWrite(12, HIGH);

    Serial.println(F("The LED should now be lit"));

    Serial.println(F("Move stick 'nose up' and back to center to continue"));

    check\_to\_continue();

    digitalWrite(12, LOW);

  }

  Serial.println(F(""));

  if(error == 0){

    Serial.println(F("==================================================="));

    Serial.println(F("Final setup check"));

    Serial.println(F("==================================================="));

    delay(1000);

    if(receiver\_check\_byte == 0b00001111){

      Serial.println(F("Receiver channels ok"));

    }

    else{

      Serial.println(F("Receiver channel verification failed!!! (ERROR 6)"));

      error = 1;

    }

    delay(1000);

    if(gyro\_check\_byte == 0b00000111){

      Serial.println(F("Gyro axes ok"));

    }

    else{

      Serial.println(F("Gyro exes verification failed!!! (ERROR 7)"));

      error = 1;

    }

  }

  if(error == 0){

    //If all is good, store the information in the EEPROM

    Serial.println(F(""));

    Serial.println(F("==================================================="));

    Serial.println(F("Storing EEPROM information"));

    Serial.println(F("==================================================="));

    Serial.println(F("Writing EEPROM"));

    delay(1000);

    Serial.println(F("Done!"));

    EEPROM.write(0, center\_channel\_1 & 0b11111111);

    EEPROM.write(1, center\_channel\_1 >> 8);

    EEPROM.write(2, center\_channel\_2 & 0b11111111);

    EEPROM.write(3, center\_channel\_2 >> 8);

    EEPROM.write(4, center\_channel\_3 & 0b11111111);

    EEPROM.write(5, center\_channel\_3 >> 8);

    EEPROM.write(6, center\_channel\_4 & 0b11111111);

    EEPROM.write(7, center\_channel\_4 >> 8);

    EEPROM.write(8, high\_channel\_1 & 0b11111111);

    EEPROM.write(9, high\_channel\_1 >> 8);

    EEPROM.write(10, high\_channel\_2 & 0b11111111);

    EEPROM.write(11, high\_channel\_2 >> 8);

    EEPROM.write(12, high\_channel\_3 & 0b11111111);

    EEPROM.write(13, high\_channel\_3 >> 8);

    EEPROM.write(14, high\_channel\_4 & 0b11111111);

    EEPROM.write(15, high\_channel\_4 >> 8);

    EEPROM.write(16, low\_channel\_1 & 0b11111111);

    EEPROM.write(17, low\_channel\_1 >> 8);

    EEPROM.write(18, low\_channel\_2 & 0b11111111);

    EEPROM.write(19, low\_channel\_2 >> 8);

    EEPROM.write(20, low\_channel\_3 & 0b11111111);

    EEPROM.write(21, low\_channel\_3 >> 8);

    EEPROM.write(22, low\_channel\_4 & 0b11111111);

    EEPROM.write(23, low\_channel\_4 >> 8);

    EEPROM.write(24, channel\_1\_assign);

    EEPROM.write(25, channel\_2\_assign);

    EEPROM.write(26, channel\_3\_assign);

    EEPROM.write(27, channel\_4\_assign);

    EEPROM.write(28, roll\_axis);

    EEPROM.write(29, pitch\_axis);

    EEPROM.write(30, yaw\_axis);

    EEPROM.write(31, type);

    EEPROM.write(32, gyro\_address);

    //Write the EEPROM signature

    EEPROM.write(33, 'J');

    EEPROM.write(34, 'M');

    EEPROM.write(35, 'B');

    //To make sure evrything is ok, verify the EEPROM data.

    Serial.println(F("Verify EEPROM data"));

    delay(1000);

    if(center\_channel\_1 != ((EEPROM.read(1) << 8) | EEPROM.read(0)))error = 1;

    if(center\_channel\_2 != ((EEPROM.read(3) << 8) | EEPROM.read(2)))error = 1;

    if(center\_channel\_3 != ((EEPROM.read(5) << 8) | EEPROM.read(4)))error = 1;

    if(center\_channel\_4 != ((EEPROM.read(7) << 8) | EEPROM.read(6)))error = 1;

    if(high\_channel\_1 != ((EEPROM.read(9) << 8) | EEPROM.read(8)))error = 1;

    if(high\_channel\_2 != ((EEPROM.read(11) << 8) | EEPROM.read(10)))error = 1;

    if(high\_channel\_3 != ((EEPROM.read(13) << 8) | EEPROM.read(12)))error = 1;

    if(high\_channel\_4 != ((EEPROM.read(15) << 8) | EEPROM.read(14)))error = 1;

    if(low\_channel\_1 != ((EEPROM.read(17) << 8) | EEPROM.read(16)))error = 1;

    if(low\_channel\_2 != ((EEPROM.read(19) << 8) | EEPROM.read(18)))error = 1;

    if(low\_channel\_3 != ((EEPROM.read(21) << 8) | EEPROM.read(20)))error = 1;

    if(low\_channel\_4 != ((EEPROM.read(23) << 8) | EEPROM.read(22)))error = 1;

    if(channel\_1\_assign != EEPROM.read(24))error = 1;

    if(channel\_2\_assign != EEPROM.read(25))error = 1;

    if(channel\_3\_assign != EEPROM.read(26))error = 1;

    if(channel\_4\_assign != EEPROM.read(27))error = 1;

    if(roll\_axis != EEPROM.read(28))error = 1;

    if(pitch\_axis != EEPROM.read(29))error = 1;

    if(yaw\_axis != EEPROM.read(30))error = 1;

    if(type != EEPROM.read(31))error = 1;

    if(gyro\_address != EEPROM.read(32))error = 1;

    if('J' != EEPROM.read(33))error = 1;

    if('M' != EEPROM.read(34))error = 1;

    if('B' != EEPROM.read(35))error = 1;

    if(error == 1)Serial.println(F("EEPROM verification failed!!! (ERROR 5)"));

    else Serial.println(F("Verification done"));

  }

  if(error == 0){

    Serial.println(F("Setup is finished."));

    Serial.println(F("You can now calibrate the esc's and upload the YMFC-AL code."));

  }

  else{

   Serial.println(F("The setup is aborted due to an error."));

   Serial.println(F("Check the Q and A page of the YMFC-AL project on:"));

   Serial.println(F("www.brokking.net for more information about this error."));

  }

  while(1);

}

//Search for the gyro and check the Who\_am\_I register

byte search\_gyro(int gyro\_address, int who\_am\_i){

  Wire.beginTransmission(gyro\_address);

  Wire.write(who\_am\_i);

  Wire.endTransmission();

  Wire.requestFrom(gyro\_address, 1);

  timer = millis() + 100;

  while(Wire.available() < 1 && timer > millis());

  lowByte = Wire.read();

  address = gyro\_address;

  return lowByte;

}

void start\_gyro(){

  //Setup the L3G4200D or L3GD20H

  if(type == 2 || type == 3){

    Wire.beginTransmission(address);                             //Start communication with the gyro with the address found during search

    Wire.write(0x20);                                            //We want to write to register 1 (20 hex)

    Wire.write(0x0F);                                            //Set the register bits as 00001111 (Turn on the gyro and enable all axis)

    Wire.endTransmission();                                      //End the transmission with the gyro

    Wire.beginTransmission(address);                             //Start communication with the gyro (adress 1101001)

    Wire.write(0x20);                                            //Start reading @ register 28h and auto increment with every read

    Wire.endTransmission();                                      //End the transmission

    Wire.requestFrom(address, 1);                                //Request 6 bytes from the gyro

    while(Wire.available() < 1);                                 //Wait until the 1 byte is received

    Serial.print(F("Register 0x20 is set to:"));

    Serial.println(Wire.read(),BIN);

    Wire.beginTransmission(address);                             //Start communication with the gyro  with the address found during search

    Wire.write(0x23);                                            //We want to write to register 4 (23 hex)

    Wire.write(0x90);                                            //Set the register bits as 10010000 (Block Data Update active & 500dps full scale)

    Wire.endTransmission();                                      //End the transmission with the gyro

    Wire.beginTransmission(address);                             //Start communication with the gyro (adress 1101001)

    Wire.write(0x23);                                            //Start reading @ register 28h and auto increment with every read

    Wire.endTransmission();                                      //End the transmission

    Wire.requestFrom(address, 1);                                //Request 6 bytes from the gyro

    while(Wire.available() < 1);                                 //Wait until the 1 byte is received

    Serial.print(F("Register 0x23 is set to:"));

    Serial.println(Wire.read(),BIN);

  }

  //Setup the MPU-6050

  if(type == 1){

    Wire.beginTransmission(address);                             //Start communication with the gyro

    Wire.write(0x6B);                                            //PWR\_MGMT\_1 register

    Wire.write(0x00);                                            //Set to zero to turn on the gyro

    Wire.endTransmission();                                      //End the transmission

    Wire.beginTransmission(address);                             //Start communication with the gyro

    Wire.write(0x6B);                                            //Start reading @ register 28h and auto increment with every read

    Wire.endTransmission();                                      //End the transmission

    Wire.requestFrom(address, 1);                                //Request 1 bytes from the gyro

    while(Wire.available() < 1);                                 //Wait until the 1 byte is received

    Serial.print(F("Register 0x6B is set to:"));

    Serial.println(Wire.read(),BIN);

    Wire.beginTransmission(address);                             //Start communication with the gyro

    Wire.write(0x1B);                                            //GYRO\_CONFIG register

    Wire.write(0x08);                                            //Set the register bits as 00001000 (500dps full scale)

    Wire.endTransmission();                                      //End the transmission

    Wire.beginTransmission(address);                             //Start communication with the gyro (adress 1101001)

    Wire.write(0x1B);                                            //Start reading @ register 28h and auto increment with every read

    Wire.endTransmission();                                      //End the transmission

    Wire.requestFrom(address, 1);                                //Request 1 bytes from the gyro

    while(Wire.available() < 1);                                 //Wait until the 1 byte is received

    Serial.print(F("Register 0x1B is set to:"));

    Serial.println(Wire.read(),BIN);

  }

}

void gyro\_signalen(){

  if(type == 2 || type == 3){

    Wire.beginTransmission(address);                             //Start communication with the gyro

    Wire.write(168);                                             //Start reading @ register 28h and auto increment with every read

    Wire.endTransmission();                                      //End the transmission

    Wire.requestFrom(address, 6);                                //Request 6 bytes from the gyro

    while(Wire.available() < 6);                                 //Wait until the 6 bytes are received

    lowByte = Wire.read();                                       //First received byte is the low part of the angular data

    highByte = Wire.read();                                      //Second received byte is the high part of the angular data

    gyro\_roll = ((highByte<<8)|lowByte);                         //Multiply highByte by 256 (shift left by 8) and ad lowByte

    if(cal\_int == 2000)gyro\_roll -= gyro\_roll\_cal;               //Only compensate after the calibration

    lowByte = Wire.read();                                       //First received byte is the low part of the angular data

    highByte = Wire.read();                                      //Second received byte is the high part of the angular data

    gyro\_pitch = ((highByte<<8)|lowByte);                        //Multiply highByte by 256 (shift left by 8) and ad lowByte

    if(cal\_int == 2000)gyro\_pitch -= gyro\_pitch\_cal;             //Only compensate after the calibration

    lowByte = Wire.read();                                       //First received byte is the low part of the angular data

    highByte = Wire.read();                                      //Second received byte is the high part of the angular data

    gyro\_yaw = ((highByte<<8)|lowByte);                          //Multiply highByte by 256 (shift left by 8) and ad lowByte

    if(cal\_int == 2000)gyro\_yaw -= gyro\_yaw\_cal;                 //Only compensate after the calibration

  }

  if(type == 1){

    Wire.beginTransmission(address);                             //Start communication with the gyro

    Wire.write(0x43);                                            //Start reading @ register 43h and auto increment with every read

    Wire.endTransmission();                                      //End the transmission

    Wire.requestFrom(address,6);                                 //Request 6 bytes from the gyro

    while(Wire.available() < 6);                                 //Wait until the 6 bytes are received

    gyro\_roll=Wire.read()<<8|Wire.read();                        //Read high and low part of the angular data

    if(cal\_int == 2000)gyro\_roll -= gyro\_roll\_cal;               //Only compensate after the calibration

    gyro\_pitch=Wire.read()<<8|Wire.read();                       //Read high and low part of the angular data

    if(cal\_int == 2000)gyro\_pitch -= gyro\_pitch\_cal;             //Only compensate after the calibration

    gyro\_yaw=Wire.read()<<8|Wire.read();                         //Read high and low part of the angular data

    if(cal\_int == 2000)gyro\_yaw -= gyro\_yaw\_cal;                 //Only compensate after the calibration

  }

}

//Check if a receiver input value is changing within 30 seconds

void check\_receiver\_inputs(byte movement){

  byte trigger = 0;

  int pulse\_length;

  timer = millis() + 30000;

  while(timer > millis() && trigger == 0){

    delay(250);

    if(receiver\_input\_channel\_1 > 1750 || receiver\_input\_channel\_1 < 1250){

      trigger = 1;

      receiver\_check\_byte |= 0b00000001;

      pulse\_length = receiver\_input\_channel\_1;

    }

    if(receiver\_input\_channel\_2 > 1750 || receiver\_input\_channel\_2 < 1250){

      trigger = 2;

      receiver\_check\_byte |= 0b00000010;

      pulse\_length = receiver\_input\_channel\_2;

    }

    if(receiver\_input\_channel\_3 > 1750 || receiver\_input\_channel\_3 < 1250){

      trigger = 3;

      receiver\_check\_byte |= 0b00000100;

      pulse\_length = receiver\_input\_channel\_3;

    }

    if(receiver\_input\_channel\_4 > 1750 || receiver\_input\_channel\_4 < 1250){

      trigger = 4;

      receiver\_check\_byte |= 0b00001000;

      pulse\_length = receiver\_input\_channel\_4;

    }

  }

  if(trigger == 0){

    error = 1;

    Serial.println(F("No stick movement detected in the last 30 seconds!!! (ERROR 2)"));

  }

  //Assign the stick to the function.

  else{

    if(movement == 1){

      channel\_3\_assign = trigger;

      if(pulse\_length < 1250)channel\_3\_assign += 0b10000000;

    }

    if(movement == 2){

      channel\_1\_assign = trigger;

      if(pulse\_length < 1250)channel\_1\_assign += 0b10000000;

    }

    if(movement == 3){

      channel\_2\_assign = trigger;

      if(pulse\_length < 1250)channel\_2\_assign += 0b10000000;

    }

    if(movement == 4){

      channel\_4\_assign = trigger;

      if(pulse\_length < 1250)channel\_4\_assign += 0b10000000;

    }

  }

}

void check\_to\_continue(){

  byte continue\_byte = 0;

  while(continue\_byte == 0){

    if(channel\_2\_assign == 0b00000001 && receiver\_input\_channel\_1 > center\_channel\_1 + 150)continue\_byte = 1;

    if(channel\_2\_assign == 0b10000001 && receiver\_input\_channel\_1 < center\_channel\_1 - 150)continue\_byte = 1;

    if(channel\_2\_assign == 0b00000010 && receiver\_input\_channel\_2 > center\_channel\_2 + 150)continue\_byte = 1;

    if(channel\_2\_assign == 0b10000010 && receiver\_input\_channel\_2 < center\_channel\_2 - 150)continue\_byte = 1;

    if(channel\_2\_assign == 0b00000011 && receiver\_input\_channel\_3 > center\_channel\_3 + 150)continue\_byte = 1;

    if(channel\_2\_assign == 0b10000011 && receiver\_input\_channel\_3 < center\_channel\_3 - 150)continue\_byte = 1;

    if(channel\_2\_assign == 0b00000100 && receiver\_input\_channel\_4 > center\_channel\_4 + 150)continue\_byte = 1;

    if(channel\_2\_assign == 0b10000100 && receiver\_input\_channel\_4 < center\_channel\_4 - 150)continue\_byte = 1;

    delay(100);

  }

  wait\_sticks\_zero();

}

//Check if the transmitter sticks are in the neutral position

void wait\_sticks\_zero(){

  byte zero = 0;

  while(zero < 15){

    if(receiver\_input\_channel\_1 < center\_channel\_1 + 20 && receiver\_input\_channel\_1 > center\_channel\_1 - 20)zero |= 0b00000001;

    if(receiver\_input\_channel\_2 < center\_channel\_2 + 20 && receiver\_input\_channel\_2 > center\_channel\_2 - 20)zero |= 0b00000010;

    if(receiver\_input\_channel\_3 < center\_channel\_3 + 20 && receiver\_input\_channel\_3 > center\_channel\_3 - 20)zero |= 0b00000100;

    if(receiver\_input\_channel\_4 < center\_channel\_4 + 20 && receiver\_input\_channel\_4 > center\_channel\_4 - 20)zero |= 0b00001000;

    delay(100);

  }

}

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

void wait\_for\_receiver(){

  byte zero = 0;

  timer = millis() + 10000;

  while(timer > millis() && zero < 15){

    if(receiver\_input\_channel\_1 < 2100 && receiver\_input\_channel\_1 > 900)zero |= 0b00000001;

    if(receiver\_input\_channel\_2 < 2100 && receiver\_input\_channel\_2 > 900)zero |= 0b00000010;

    if(receiver\_input\_channel\_3 < 2100 && receiver\_input\_channel\_3 > 900)zero |= 0b00000100;

    if(receiver\_input\_channel\_4 < 2100 && receiver\_input\_channel\_4 > 900)zero |= 0b00001000;

    delay(500);

    Serial.print(F("."));

  }

  if(zero == 0){

    error = 1;

    Serial.println(F("."));

    Serial.println(F("No valid receiver signals found!!! (ERROR 1)"));

  }

  else Serial.println(F(" OK"));

}

//Register the min and max receiver values and exit when the sticks are back in the neutral position

void register\_min\_max(){

  byte zero = 0;

  low\_channel\_1 = receiver\_input\_channel\_1;

  low\_channel\_2 = receiver\_input\_channel\_2;

  low\_channel\_3 = receiver\_input\_channel\_3;

  low\_channel\_4 = receiver\_input\_channel\_4;

  while(receiver\_input\_channel\_1 < center\_channel\_1 + 20 && receiver\_input\_channel\_1 > center\_channel\_1 - 20)delay(250);

  Serial.println(F("Measuring endpoints...."));

  while(zero < 15){

    if(receiver\_input\_channel\_1 < center\_channel\_1 + 20 && receiver\_input\_channel\_1 > center\_channel\_1 - 20)zero |= 0b00000001;

    if(receiver\_input\_channel\_2 < center\_channel\_2 + 20 && receiver\_input\_channel\_2 > center\_channel\_2 - 20)zero |= 0b00000010;

    if(receiver\_input\_channel\_3 < center\_channel\_3 + 20 && receiver\_input\_channel\_3 > center\_channel\_3 - 20)zero |= 0b00000100;

    if(receiver\_input\_channel\_4 < center\_channel\_4 + 20 && receiver\_input\_channel\_4 > center\_channel\_4 - 20)zero |= 0b00001000;

    if(receiver\_input\_channel\_1 < low\_channel\_1)low\_channel\_1 = receiver\_input\_channel\_1;

    if(receiver\_input\_channel\_2 < low\_channel\_2)low\_channel\_2 = receiver\_input\_channel\_2;

    if(receiver\_input\_channel\_3 < low\_channel\_3)low\_channel\_3 = receiver\_input\_channel\_3;

    if(receiver\_input\_channel\_4 < low\_channel\_4)low\_channel\_4 = receiver\_input\_channel\_4;

    if(receiver\_input\_channel\_1 > high\_channel\_1)high\_channel\_1 = receiver\_input\_channel\_1;

    if(receiver\_input\_channel\_2 > high\_channel\_2)high\_channel\_2 = receiver\_input\_channel\_2;

    if(receiver\_input\_channel\_3 > high\_channel\_3)high\_channel\_3 = receiver\_input\_channel\_3;

    if(receiver\_input\_channel\_4 > high\_channel\_4)high\_channel\_4 = receiver\_input\_channel\_4;

    delay(100);

  }

}

//Check if the angular position of a gyro axis is changing within 10 seconds

void check\_gyro\_axes(byte movement){

  byte trigger\_axis = 0;

  float gyro\_angle\_roll, gyro\_angle\_pitch, gyro\_angle\_yaw;

  //Reset all axes

  gyro\_angle\_roll = 0;

  gyro\_angle\_pitch = 0;

  gyro\_angle\_yaw = 0;

  gyro\_signalen();

  timer = millis() + 10000;

  while(timer > millis() && gyro\_angle\_roll > -30 && gyro\_angle\_roll < 30 && gyro\_angle\_pitch > -30 && gyro\_angle\_pitch < 30 && gyro\_angle\_yaw > -30 && gyro\_angle\_yaw < 30){

    gyro\_signalen();

    if(type == 2 || type == 3){

      gyro\_angle\_roll += gyro\_roll \* 0.00007;              //0.00007 = 17.5 (md/s) / 250(Hz)

      gyro\_angle\_pitch += gyro\_pitch \* 0.00007;

      gyro\_angle\_yaw += gyro\_yaw \* 0.00007;

    }

    if(type == 1){

      gyro\_angle\_roll += gyro\_roll \* 0.0000611;          // 0.0000611 = 1 / 65.5 (LSB degr/s) / 250(Hz)

      gyro\_angle\_pitch += gyro\_pitch \* 0.0000611;

      gyro\_angle\_yaw += gyro\_yaw \* 0.0000611;

    }

    delayMicroseconds(3700); //Loop is running @ 250Hz. +/-300us is used for communication with the gyro

  }

  //Assign the moved axis to the orresponding function (pitch, roll, yaw)

  if((gyro\_angle\_roll < -30 || gyro\_angle\_roll > 30) && gyro\_angle\_pitch > -30 && gyro\_angle\_pitch < 30 && gyro\_angle\_yaw > -30 && gyro\_angle\_yaw < 30){

    gyro\_check\_byte |= 0b00000001;

    if(gyro\_angle\_roll < 0)trigger\_axis = 0b10000001;

    else trigger\_axis = 0b00000001;

  }

  if((gyro\_angle\_pitch < -30 || gyro\_angle\_pitch > 30) && gyro\_angle\_roll > -30 && gyro\_angle\_roll < 30 && gyro\_angle\_yaw > -30 && gyro\_angle\_yaw < 30){

    gyro\_check\_byte |= 0b00000010;

    if(gyro\_angle\_pitch < 0)trigger\_axis = 0b10000010;

    else trigger\_axis = 0b00000010;

  }

  if((gyro\_angle\_yaw < -30 || gyro\_angle\_yaw > 30) && gyro\_angle\_roll > -30 && gyro\_angle\_roll < 30 && gyro\_angle\_pitch > -30 && gyro\_angle\_pitch < 30){

    gyro\_check\_byte |= 0b00000100;

    if(gyro\_angle\_yaw < 0)trigger\_axis = 0b10000011;

    else trigger\_axis = 0b00000011;

  }

  if(trigger\_axis == 0){

    error = 1;

    Serial.println(F("No angular motion is detected in the last 10 seconds!!! (ERROR 4)"));

  }

  else

  if(movement == 1)roll\_axis = trigger\_axis;

  if(movement == 2)pitch\_axis = trigger\_axis;

  if(movement == 3)yaw\_axis = trigger\_axis;

}

//This routine is called every time input 8, 9, 10 or 11 changed state

ISR(PCINT0\_vect){

  current\_time = micros();

  //Channel 1=========================================

  if(PINB & B00000001){                                        //Is input 8 high?

    if(last\_channel\_1 == 0){                                   //Input 8 changed from 0 to 1

      last\_channel\_1 = 1;                                      //Remember current input state

      timer\_1 = current\_time;                                  //Set timer\_1 to current\_time

    }

  }

  else if(last\_channel\_1 == 1){                                //Input 8 is not high and changed from 1 to 0

    last\_channel\_1 = 0;                                        //Remember current input state

    receiver\_input\_channel\_1 = current\_time - timer\_1;         //Channel 1 is current\_time - timer\_1

  }

  //Channel 2=========================================

  if(PINB & B00000010 ){                                       //Is input 9 high?

    if(last\_channel\_2 == 0){                                   //Input 9 changed from 0 to 1

      last\_channel\_2 = 1;                                      //Remember current input state

      timer\_2 = current\_time;                                  //Set timer\_2 to current\_time

    }

  }

  else if(last\_channel\_2 == 1){                                //Input 9 is not high and changed from 1 to 0

    last\_channel\_2 = 0;                                        //Remember current input state

    receiver\_input\_channel\_2 = current\_time - timer\_2;         //Channel 2 is current\_time - timer\_2

  }

  //Channel 3=========================================

  if(PINB & B00000100 ){                                       //Is input 10 high?

    if(last\_channel\_3 == 0){                                   //Input 10 changed from 0 to 1

      last\_channel\_3 = 1;                                      //Remember current input state

      timer\_3 = current\_time;                                  //Set timer\_3 to current\_time

    }

  }

  else if(last\_channel\_3 == 1){                                //Input 10 is not high and changed from 1 to 0

    last\_channel\_3 = 0;                                        //Remember current input state

    receiver\_input\_channel\_3 = current\_time - timer\_3;         //Channel 3 is current\_time - timer\_3

  }

  //Channel 4=========================================

  if(PINB & B00001000 ){                                       //Is input 11 high?

    if(last\_channel\_4 == 0){                                   //Input 11 changed from 0 to 1

      last\_channel\_4 = 1;                                      //Remember current input state

      timer\_4 = current\_time;                                  //Set timer\_4 to current\_time

    }

  }

  else if(last\_channel\_4 == 1){                                //Input 11 is not high and changed from 1 to 0

    last\_channel\_4 = 0;                                        //Remember current input state

    receiver\_input\_channel\_4 = current\_time - timer\_4;         //Channel 4 is current\_time - timer\_4

  }

}

//Intro subroutine

void intro(){

  Serial.println(F("==================================================="));

  delay(1500);

  Serial.println(F(""));

  Serial.println(F("Your"));

  delay(500);

  Serial.println(F("  Multicopter"));

  delay(500);

  Serial.println(F("    Flight"));

  delay(500);

  Serial.println(F("      Controller"));

  delay(1000);

  Serial.println(F(""));

  Serial.println(F("YMFC-AL Setup Program"));

  Serial.println(F(""));

  Serial.println(F("==================================================="));

  delay(1500);

  Serial.println(F("For support and questions: www.brokking.net"));

  Serial.println(F(""));

  Serial.println(F("Have fun!"));

}

CALLIBRATION CODE:

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

//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.

}

SETUP CODE:

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

//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 = 1.3;               //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;

int cal\_int, start, gyro\_address;

int receiver\_input[5];

int temperature;

int acc\_axis[4], gyro\_axis[4];

float roll\_level\_adjust, pitch\_level\_adjust;

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;

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

//Setup routine

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

void setup(){

  //Serial.begin(57600);

  //Copy the EEPROM data for fast access data.

  for(start = 0; start <= 35; start++)eeprom\_data[start] = EEPROM.read(start);

  start = 0;                                                                //Set start back to zero.

  gyro\_address = eeprom\_data[32];                                           //Store the gyro address in the variable.

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

  TWBR = 12;                                                                //Set the I2C clock speed to 400kHz.

  //Arduino (Atmega) 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 |= B00110000;                                                        //Configure digital poort 12 and 13 as output.

  //Use the led on the Arduino for startup indication.

  digitalWrite(12,HIGH);                                                    //Turn on the warning led.

  //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(10);

  //The flight controller needs the MPU-6050 with gyro and accelerometer

  //If setup is completed without MPU-6050 stop the flight controller program

  if(eeprom\_data[31] == 2 || eeprom\_data[31] == 3)delay(10);

  set\_gyro\_registers();                                                     //Set the specific gyro registers.

  for (cal\_int = 0; cal\_int < 1250 ; cal\_int ++){                           //Wait 5 seconds before continuing.

    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.

    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 throtle 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 correcion 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)

    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.

  }

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

  //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[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.

}

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

//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

  }

}

THE END