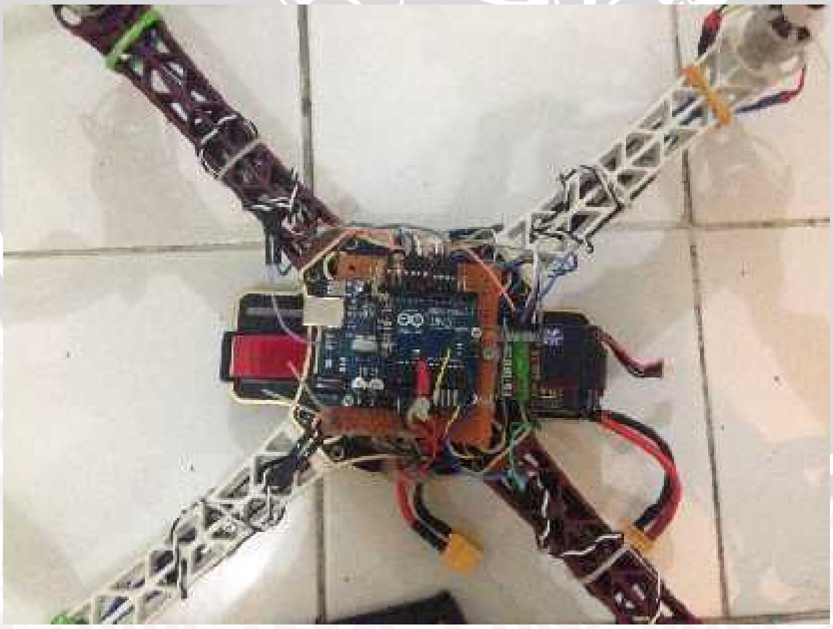
LAMPIRAN I

FOTO ALAT

FOTO ALAT

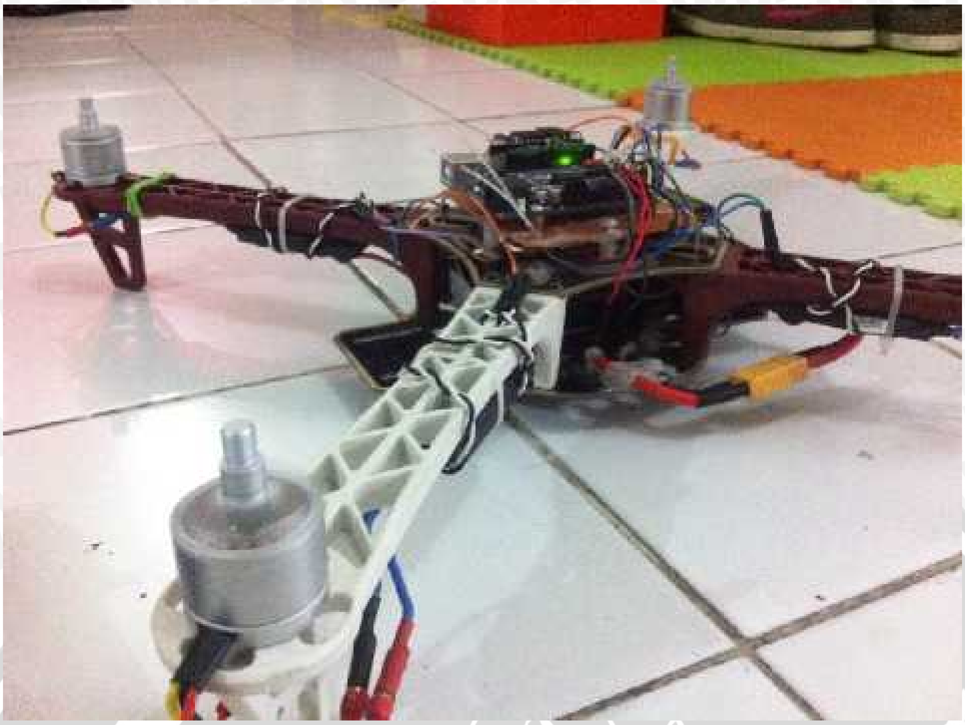
Gambar alat dari samping



Gmbar alat dari atas

Gambar alat dari samping dengan *remote control*

Gambar tampilan layar pengaturan *remote control*

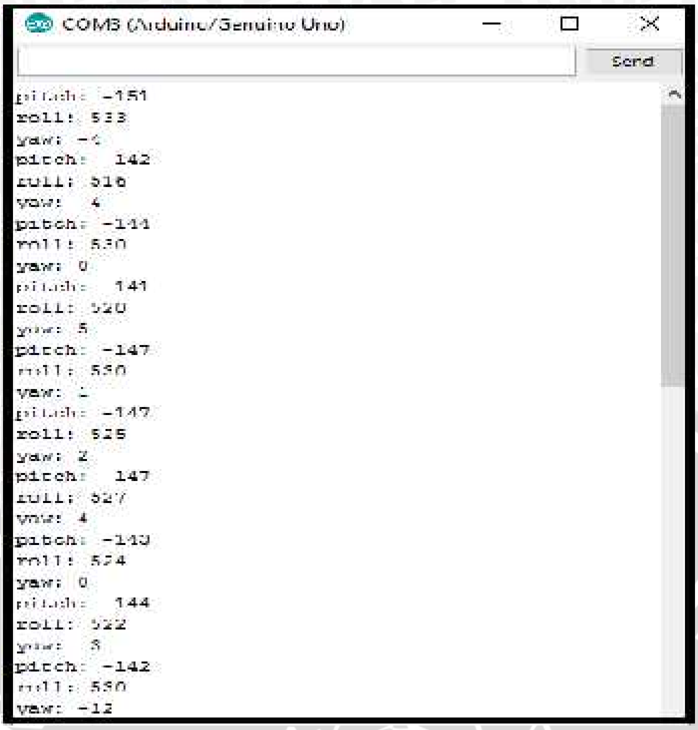


Gambar alat dari samping

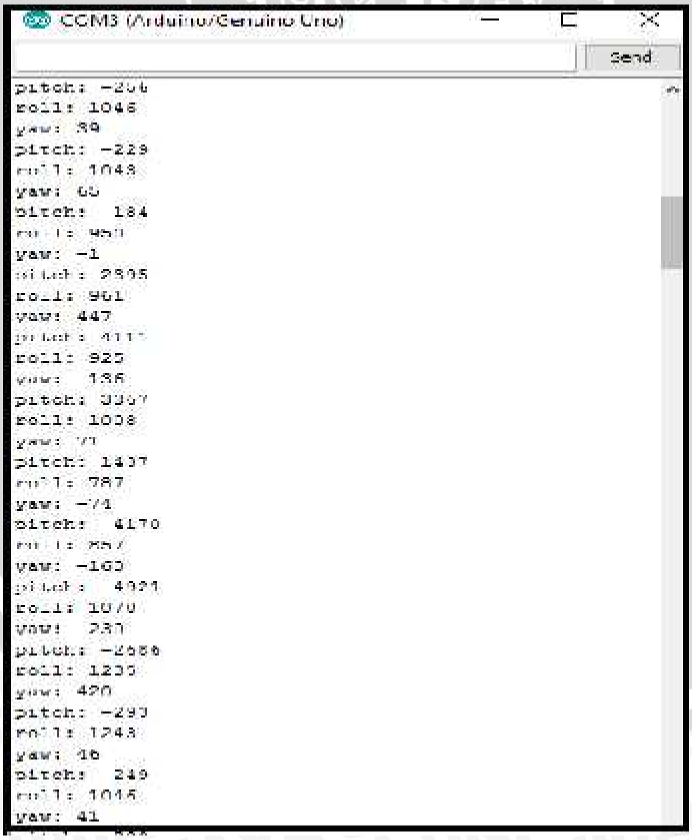
Gambar alat saat aktif

LAMPIRAN II

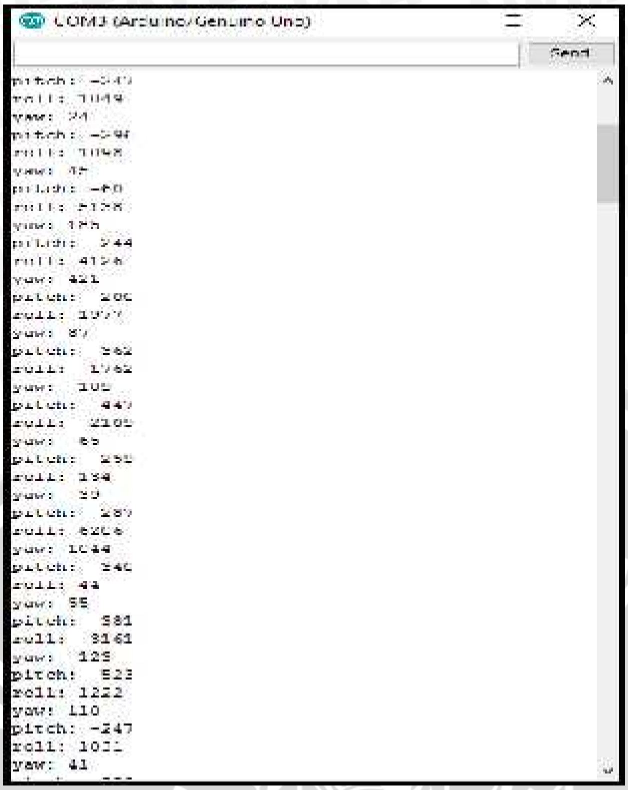
HASIL PENGUJIAN GYROSCOPE



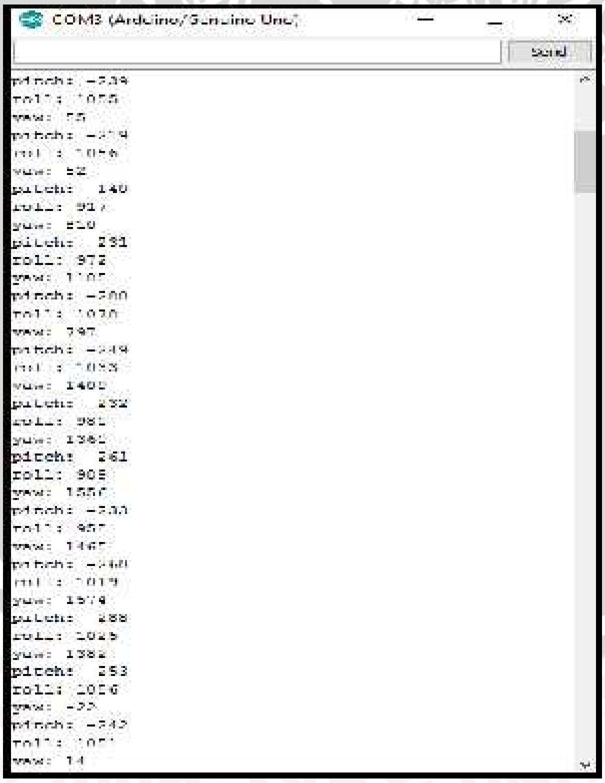
Data Mentah Modul Sensor Masing-masing Sumbu Saat Kondisi Diam



Data Mentah Modul Sensor saat Pergerakan *Pitch*



Data Mentah Sensor saat Pergerakan *Roll*



Data Mentah Sensor saat Pergerakan *Yaw*

LAMPIRAN III

LISTING PROGRAM

PROGRAM PENGUJIAN SENSOR *GYROSCOPE*

```
#include <Wire.h> //Include the Wire.h library so we can communicate with the gyro
#include <SoftwareSerial.h>// librari untuk Serial
//<<kalo pake bluetooth coment software nya dihapus>>
//Declaring variables
int cal_int;
unsigned long UL_timer;
double gyro_pitch, gyro_roll, gyro_yaw;
double gyro_roll_cal, gyro_pitch_cal, gyro_yaw_cal;
byte highByte, lowByte;
SoftwareSerial blutut(0,1); // RX, TX
//Setup routine
void setup(){
 Wire.begin();
 blutut.begin(9600);                  //Start the blutut connetion @ 9600bps
//blutut.begin(9600);
 //The gyro is disabled by default and needs to be started
 Wire.beginTransmission(107);         //Start communication with the gyro (adress 1101011)
 Wire.write(0x20);                    //We want to write to register 20
 Wire.write(0x0F);                    //Set the register bits as 00001111 (Turn on the gyro and enable all axis)
 Wire.endTransmission();              //End the transmission with the gyro
 Wire.beginTransmission(107);         //Start communication with the gyro (adress 1101001)
 Wire.write(0x23);                    //We want to write to register 23
 Wire.write(0x80);                    //Set the register bits as 10000000 (Block Data Update active)
 Wire.endTransmission();              //End the transmission with the gyro
 delay(250);                          //Give the gyro time to start
 //Let's take multiple samples so we can determine the average gyro offset
 blutut.print("Starting calibration...");       //Print message
 for (cal_int = 0; cal_int < 2000 ; cal_int ++){    //Take 2000 readings for calibration
  gyro_signalen();                    //Read the gyro output
```

```
    gyro_roll_cal += gyro_roll;                    //Ad roll value to gyro_roll_cal
    gyro_pitch_cal += gyro_pitch;                  //Ad pitch value to gyro_pitch_cal
    gyro_yaw_cal += gyro_yaw;                      //Ad yaw value to gyro_yaw_cal
    if(cal_int%100 == 0)blutut.print(".");         //Print a dot every 100 readings
    delay(4);                                      //Wait 4 milliseconds before the next loop
  }
  //Now that we have 2000 measures, we need to devide by 2000 to get the average gyro offset
  blutut.println(" done!");                        //2000 measures are done!
  gyro_roll_cal /= 2000;                           //Divide the roll total by 2000
  gyro_pitch_cal /= 2000;                          //Divide the pitch total by 2000
  gyro_yaw_cal /= 2000;                            //Divide the yaw total by 2000
  }
//Main program
void loop(){
  delay(250);                                      //Wait 250 microseconds for every loop
  gyro_signalen();                                 //Read the gyro signals
  print_output();                                  //Print the output
}
void gyro_signalen(){
  Wire.beginTransmission(107);                     //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(107, 6);                        //Request 6 bytes from the gyro
  while(Wire.available() < 6);                     //Wait until the 6 bytes are received
  lowByte = Wire.read();                           //First received byte is the low part of the angular data
  highByte = Wire.read();                          //Second received byte is the high part of the angular data
  gyro_roll = ((highByte<<8)|lowByte);             //Multiply highByte by 256 and ad lowByte
  if(cal_int == 2000)gyro_roll -= gyro_roll_cal;   //Only compensate after the calibration
  lowByte = Wire.read();                           //First received byte is the low part of the angular data
  highByte = Wire.read();                          //Second received byte is the high part of the angular data
  gyro_pitch = ((highByte<<8)|lowByte);            //Multiply highByte by 256 and ad lowByte
```

```
  gyro_pitch *= -1;                              //Invert axis
  if(cal_int == 2000)gyro_pitch -= gyro_pitch_cal;   //Only compensate after the calibration
  lowByte = Wire.read();                         //First received byte is the low part of the angular data
  highByte = Wire.read();                        //Second received byte is the high part of the angular data
  gyro_yaw = ((highByte<<8)|lowByte);            //Multiply highByte by 256 and ad lowByte
  gyro_yaw *= -1;                                //Invert axis
  if(cal_int == 2000)gyro_yaw -= gyro_yaw_cal;       //Only compensate after the calibration
}
void print_output(){
  blutut.print("Pitch:");
  if(gyro_pitch >= 0)blutut.print("+");
  blutut.print(gyro_pitch/57.14286,0);           //Convert to degree per second
  if(gyro_pitch/57.14286 - 2 > 0)blutut.print(" NoU");
  else if(gyro_pitch/57.14286 + 2 < 0)blutut.print(" NoD");
  else blutut.print(" ---");
  blutut.print("  Roll:");
  if(gyro_roll >= 0)blutut.print("+");
  blutut.print(gyro_roll/57.14286,0);            //Convert to degree per second
  if(gyro_roll/57.14286 - 2 > 0)blutut.print(" RwD");
  else if(gyro_roll/57.14286 + 2 < 0)blutut.print(" RwU");
  else blutut.print(" ---");
  blutut.print("  Yaw:");
  if(gyro_yaw >= 0)blutut.print("+");
  blutut.print(gyro_yaw/57.14286,0);   //Convert to degree per second
  if(gyro_yaw/57.14286 - 2 > 0)blutut.println(" NoR");
  else if(gyro_yaw/57.14286 + 2 < 0)blutut.println(" NoL");
  else blutut.println(" ---");
}
```

PRGORAM UTAMA

```
//#include <SoftwareSerial.h>            (untuk blutooth)
#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 = 1.2;              //Gain setting for the roll P-controller (1.3)
float pid_i_gain_roll = 0.05;             //Gain setting for the roll I-controller (0.05)
float pid_d_gain_roll = 10;               //Gain setting for the roll D-controller (15)
int pid_max_roll = 400;                   //Maximum output of the PID-controller (+/-)
float pid_p_gain_pitch = pid_p_gain_roll;  //Gain setting for the pitch P-controller.
float pid_i_gain_pitch = pid_i_gain_roll;  //Gain setting for the pitch I-controller.
float pid_d_gain_pitch = pid_d_gain_roll;  //Gain setting for the pitch D-controller.
int pid_max_pitch = pid_max_roll;          //Maximum output of the PID-controller (+/-)
float pid_p_gain_yaw = 4;                 //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;                 //Gain setting for the pitch D-controller. //0
int pid_max_yaw = 400;                    //Maximum output of the PID-controller (+/-)
//SoftwareSerial Serial (0,1); // RX, TX (untuk blutooth)
////////////////////////////////////////////////////////////////////////////////////////////////////////////
//Declaring global variables
////////////////////////////////////////////////////////////////////////////////////////////////////////////
byte last_channel_1, last_channel_2, last_channel_3, last_channel_4;
byte eeprom_data[36];
byte highByte, lowByte;
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];
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[4], 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;
///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//Setup routine
///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void setup(){
  //Serial.begin(57600);
  //Read EEPROM for fast access data.
  for(start = 0; start <= 35; start++)eeprom_data[start] = EEPROM.read(start);
  gyro_address = eeprom_data[32];                          //Store the gyro address in the variable.
  Wire.begin();                                            //Start the I2C as master.
  Serial.begin(9600);
  //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.
  //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);
  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).
  Serial.print("Starting calibration...");          //Print message
  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.
  Serial.println(" done!");
  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;
  }
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//Main program loop
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void loop(){
 //delay(250);                                 //Wait 250 microseconds for every loop
  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.

//Let's get the current gyro data and scale it to degrees per second for the pid calculations.

gyro_signalen();

gyro_roll_input = (gyro_roll_input * 0.8) + ((gyro_roll / 57.14286) * 0.2); //Gyro pid input is deg/sec.

gyro_pitch_input = (gyro_pitch_input * 0.8) + ((gyro_pitch / 57.14286) * 0.2); //Gyro pid input is deg/sec.

gyro_yaw_input = (gyro_yaw_input * 0.8) + ((gyro_yaw / 57.14286) * 0.2); //Gyro pid input is deg/sec.

//UNTUK MENYALAKAN MOTOR : 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;

//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;

}

//UNTUK STOP MOTOR : throttle low and yaw right.

if(start == 2 && receiver_input_channel_3 < 1050 && receiver_input_channel_4 > 1950)start = 0;

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

//In the case of deviding by 3 the max roll rate is aprox 164 degrees per second ((500-8)/3 = 164d/s).

pid_roll_setpoint = 0;

//We need a little dead band of 16us for better results.

if(receiver_input_channel_1 > 1508)pid_roll_setpoint = (receiver_input_channel_1 - 1508)/3.0;

```
else if(receiver_input_channel_1 < 1492)pid_roll_setpoint = (receiver_input_channel_1 - 1492)/3.0;
  //The PID set point in degrees per second is determined by the pitch receiver input.
//In the case of deviding by 3 the max pitch rate is aprox 164 degrees per second ( (500-8)/3 = 164d/s ).
pid_pitch_setpoint = 0;
//We need a little dead band of 16us for better results.
if(receiver_input_channel_2 > 1508)pid_pitch_setpoint = (receiver_input_channel_2 - 1508)/3.0;
else if(receiver_input_channel_2 < 1492)pid_pitch_setpoint = (receiver_input_channel_2 - 1492)/3.0;
  //The PID set point in degrees per second is determined by the yaw receiver input.
//In the case of deviding by 3 the max yaw rate is aprox 164 degrees per second ( (500-8)/3 = 164d/s ).
pid_yaw_setpoint = 0;
//We need a little dead band of 16us for better results.
if(receiver_input_channel_3 > 1050){ //Do not yaw when turning off the motors.
  if(receiver_input_channel_4 > 1508)pid_yaw_setpoint = (receiver_input_channel_4 - 1508)/3.0;
  else if(receiver_input_channel_4 < 1492)pid_yaw_setpoint = (receiver_input_channel_4 - 1492)/3.0;
}
//PID inputs are known. So we can calculate the pid output.
calculate_pid();
  //The battery voltage is needed for compensation.
//A complementary filter is used to reduce noise.
//0.09853 = 0.08 * 1.2317.
battery_voltage = battery_voltage * 0.92 + (analogRead(0) + 65) * 0.09853;
  //Turn on the led if battery voltage is to low.
if(battery_voltage < 1030 && battery_voltage > 600)digitalWrite(12, HIGH);
  throttle = receiver_input_channel_3;                    //We need the throttle signal as a base signal.
  if (start == 2){                                        //The motors are started.
  if (throttle > 1800) throttle = 1800;                   //We need some room to keep full control at full throttle.
  esc_1 = throttle - pid_output_pitch + pid_output_roll - pid_output_yaw; //Calculate the pulse for esc 1 (front-right - CCW)
  esc_2 = throttle + pid_output_pitch + pid_output_roll + pid_output_yaw; //Calculate the pulse for esc 2 (rear-right - CW)
  esc_3 = throttle + pid_output_pitch - pid_output_roll - pid_output_yaw; //Calculate the pulse for esc 3 (rear-left - CCW)
```

```
esc_4 = throttle - pid_output_pitch - pid_output_roll + pid_output_yaw; //Calculate the pulse for esc 4 (front-left - CW)

if (battery_voltage < 1240 && battery_voltage > 800){          //Is the battery connected?

  esc_1 += esc_1 * ((1240 - battery_voltage)/(float)3500);          //Compensate the esc-1 pulse for voltage drop.

  esc_2 += esc_2 * ((1240 - battery_voltage)/(float)3500);          //Compensate the esc-2 pulse for voltage drop.

  esc_3 += esc_3 * ((1240 - battery_voltage)/(float)3500);          //Compensate the esc-3 pulse for voltage drop.

  esc_4 += esc_4 * ((1240 - battery_voltage)/(float)3500);          //Compensate the esc-4 pulse for voltage drop.

}

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

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

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

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

  if(esc_1 > 2000)esc_1 = 2000;                      //Limit the esc-1 pulse to 2000us.

  if(esc_2 > 2000)esc_2 = 2000;                      //Limit the esc-2 pulse to 2000us.

  if(esc_3 > 2000)esc_3 = 2000;                      //Limit the esc-3 pulse to 2000us.

  if(esc_4 > 2000)esc_4 = 2000;                      //Limit the esc-4 pulse to 2000us.

 }

 else{

  esc_1 = 1000;                              //If start is not 2 keep a 1000us pulse for ess-1.

  esc_2 = 1000;                              //If start is not 2 keep a 1000us pulse for ess-2.

  esc_3 = 1000;                              //If start is not 2 keep a 1000us pulse for ess-3.

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

 }

  //All the information for controlling the motor's is available.

 //The refresh rate is 250Hz. That means the esc's need there pulse every 4ms.

 while(micros() - loop_timer < 4000);                    //We wait until 4000us are passed.

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

 PORTD |= B11110000;                              //Set digital outputs 4,5,6 and 7 high.

 timer_channel_1 = esc_1 + loop_timer;                     //Calculate the time of the faling edge of the esc-1 pulse.
```

```
  timer_channel_2 = esc_2 + loop_timer;                    //Calculate the time of the faling edge of the esc-2 pulse.
  timer_channel_3 = esc_3 + loop_timer;                    //Calculate the time of the faling edge of the esc-3 pulse.
  timer_channel_4 = esc_4 + loop_timer;                    //Calculate the time of the faling edge of the esc-4 pulse.
    while(PORTD >= 16){                                    //Stay in this loop until output 4,5,6 and 7 are low.
    esc_loop_timer = micros();                             //Read the current time.
    if(timer_channel_1 <= esc_loop_timer)PORTD &= B11101111;    //Set digital output 4 to low if the time is expired.
    if(timer_channel_2 <= esc_loop_timer)PORTD &= B11011111;    //Set digital output 5 to low if the time is expired.
    if(timer_channel_3 <= esc_loop_timer)PORTD &= B10111111;    //Set digital output 6 to low if the time is expired.
    if(timer_channel_4 <= esc_loop_timer)PORTD &= B01111111;    //Set digital output 7 to low if the time is expired.
  }
 //set_gyro_registers();
 //gyro_signalen();                 //Read the gyro signals
 print_output();                    //Print the output
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//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
  }
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//Subroutine for reading the gyro
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void gyro_signalen(){
  //Read the L3G4200D or L3GD20H
  if(eeprom_data[31] == 2 || eeprom_data[31] == 3){
    Wire.beginTransmission(gyro_address);                          //Start communication with the gyro (adress
1101001)
    Wire.write(168);                                               //Start reading @ register 28h and auto increment with every
read
    Wire.endTransmission();                                        //End the transmission
    Wire.requestFrom(gyro_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_axis[1] = ((highByte<<8)|lowByte);                        //Multiply highByte by 256 (shift left by 8) and
ad lowByte
    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_axis[2] = ((highByte<<8)|lowByte);                        //Multiply highByte by 256 (shift left by 8) and
ad lowByte
    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_axis[3] = ((highByte<<8)|lowByte);                        //Multiply highByte by 256 (shift left by 8) and
ad lowByte
  }
```

```
    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];
  if(eeprom_data[28] & 0b10000000)gyro_roll *= -1;
  gyro_pitch = gyro_axis[eeprom_data[29] & 0b00000011];
  if(eeprom_data[29] & 0b10000000)gyro_pitch *= -1;
  gyro_yaw = gyro_axis[eeprom_data[30] & 0b00000011];
  if(eeprom_data[30] & 0b10000000)gyro_yaw *= -1;
  }
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//Subroutine for calculating pid outputs
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
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 L3GD20H
  if(eeprom_data[31] == 3){
```

```
   Wire.beginTransmission(gyro_address);                        //Start communicationwith 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(gyro_address);                   //Start communication 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.

   //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(0x23);                                    //Start reading @ register 0x23
   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() != 0x90){                              //Check if the value is 0x90
     digitalWrite(12,HIGH);                              //Turn on the warning led
     while(1)delay(10);                                 //Stay in this loop for ever
   }
  }
}
void print_output(){
  Serial.print("Pitch:");
  if(gyro_axis[2] >= 0)Serial.print("+");
  Serial.print(gyro_axis[2]/57.14286,0);               //Convert to degree per second
  if(gyro_axis[2]/57.14286 - 2 > 0)Serial.print(" NoU");
  else if(gyro_axis[2]/57.14286 + 2 < 0)Serial.print(" NoD");
```

```
  else Serial.print(" ---");
  Serial.print("  Roll:");
  if(gyro_axis[1] >= 0)Serial.print("+");
  Serial.print(gyro_axis[1]/57.14286,0);              //Convert to degree per second
  if(gyro_axis[1]/57.14286 - 2 > 0)Serial.print(" RwD");
  else if(gyro_axis[1]/57.14286 + 2 < 0)Serial.print(" RwU");
  else Serial.print(" ---");
  Serial.print("  Yaw:");
  if(gyro_axis[3]>= 0)Serial.print("+");
  Serial.print(gyro_axis[3]/57.14286,0);              //Convert to degree per second
  if(gyro_axis[3]/57.14286 - 2 > 0)Serial.println(" NoR");
  else if(gyro_axis[3]/57.14286 + 2 < 0)Serial.println(" NoL");
  else Serial.println(" ---");
}
```

LAMPIRAN IV

DATASHEET