BBM434 GÖMÜLÜ SİSTEMLER LAB. PROJESİ

Emre DOĞAN 21426864

PROJE TANIMI

Arduino ve ESP32 tabanlı multikopter ve kumanda.

GEREKSİNİMLER

4 x Emax MT2204 2300KV	Fırçasız DC Motor (2CW - 2 CCW)
4 x Emax SimonK 12A	ESC
Gens Ace 2700Mah 11.1V 25C 3s1p	Li-Po Batarya
QA250 Tipi Şase	

Kontrolcü:

ESP WROOM-32	
GY-91 MPU9250 + BMP280	10DOF 9 Eksen IMU (Jiroskop, İvmeölçer, Pusula, Barometre)
NRF24L01 + PA + LNA SMA Anten	2.4 GHz Kablosuz Haberleşme Modülü
ACS711EX	Akım Sensörü -31/+31 Amper
4 x ST D17 N06 9727	MOSFET Transistor
2 x Step Down Voltaj Regülatörü 3A (3.3V - 5V)	
20 x RGB SMD Led	

İsteğe Bağlı Kullanılabilecek Diğer Modüller:

GY-NEO6MV2	GPS Modülü - Uçuş Kontrol Sistem GPS′i
DRF7020D20	Dorji RF Alıcı Verici Modül (20dBm 433MHz)

Kumanda:

Arduino Nano 328	
NRF24L01 + PA + LNA SMA Anten	2.4 GHz Kablosuz Haberleşme Modülü
2S 7.4V 1300mAh 25C	Li-Po Batarya

2 x 2 Eksen Joystick
2 x 2 Yollu Toggle Switch
2 x Potansiyometre
2 x RGB SMD Led

İsteğe Bağlı Kullanılabilecek Diğer Modüller:

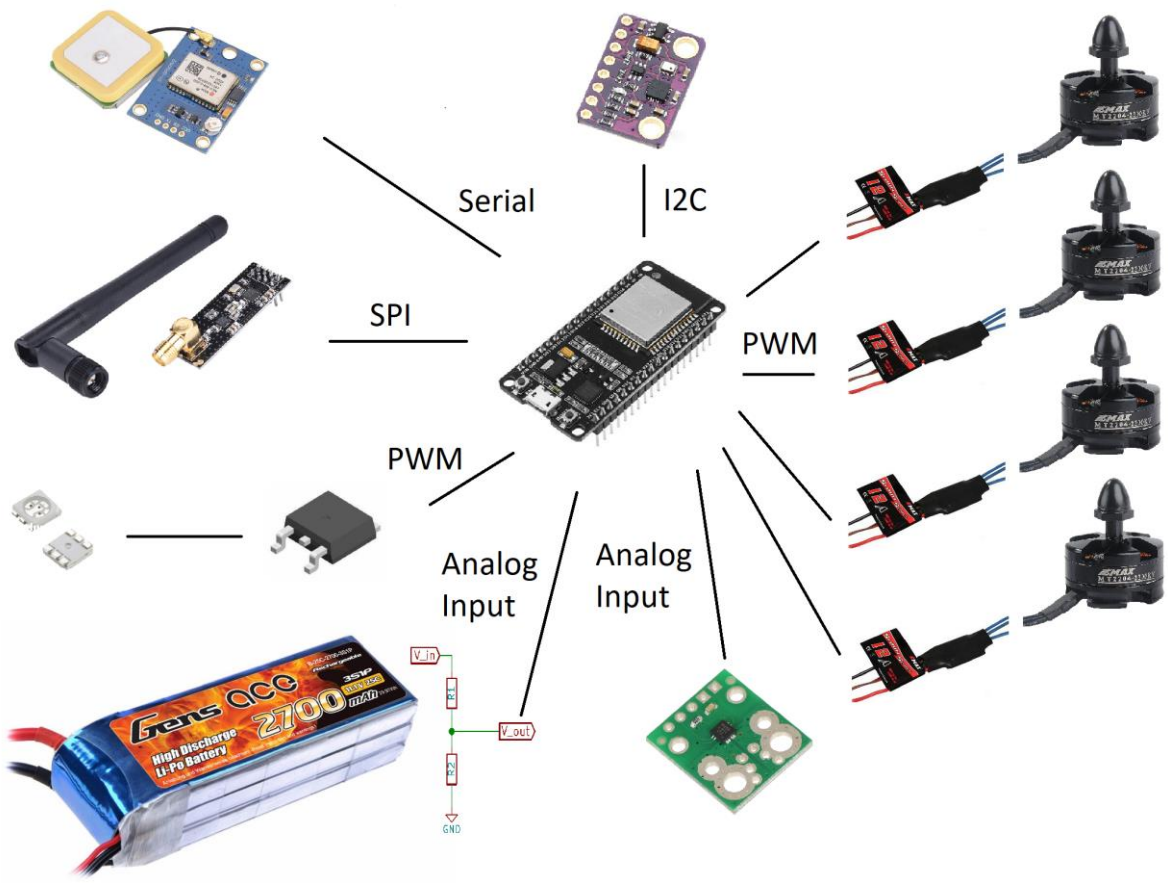
Arduino Nano 328	
GY-NEO6MV2	GPS Modülü - Uçuş Kontrol Sistem GPS′i
GY-521 MPU6050	6 Eksen IMU (Jirokop,İvmeölçer)
HC06	Bluetooth-Serial Modül
SD Kart Modülü	
Hoparlör	
Devreli Buzzer 5v-12v 12mm	

Devre Elemanları:

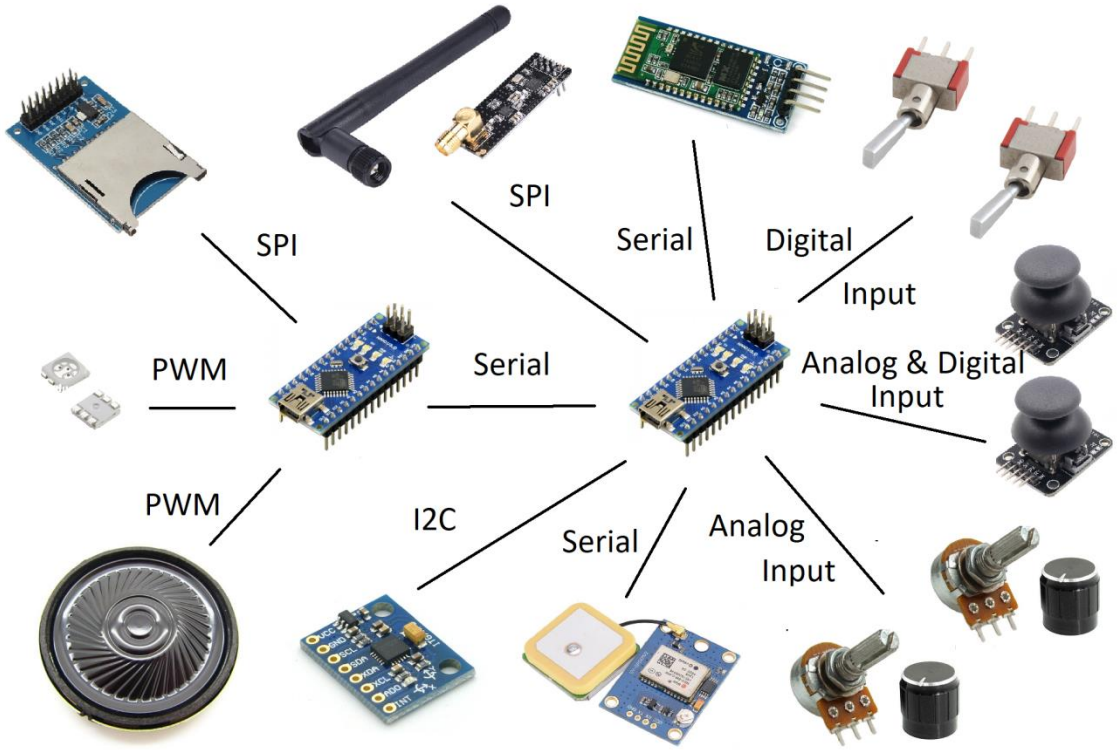
0R, 100R, 220R, 330R, 470R, 1K, 4.7K, 10K, 100K	1206 ¼ SMD Direnç
100NF 50V 10% x7R	1206 SMD Kondansatör
100UF	16V Kondansatör
1000UF	16V Kondansatör
MG	TVS Diyot

GENEL BAKIŞ

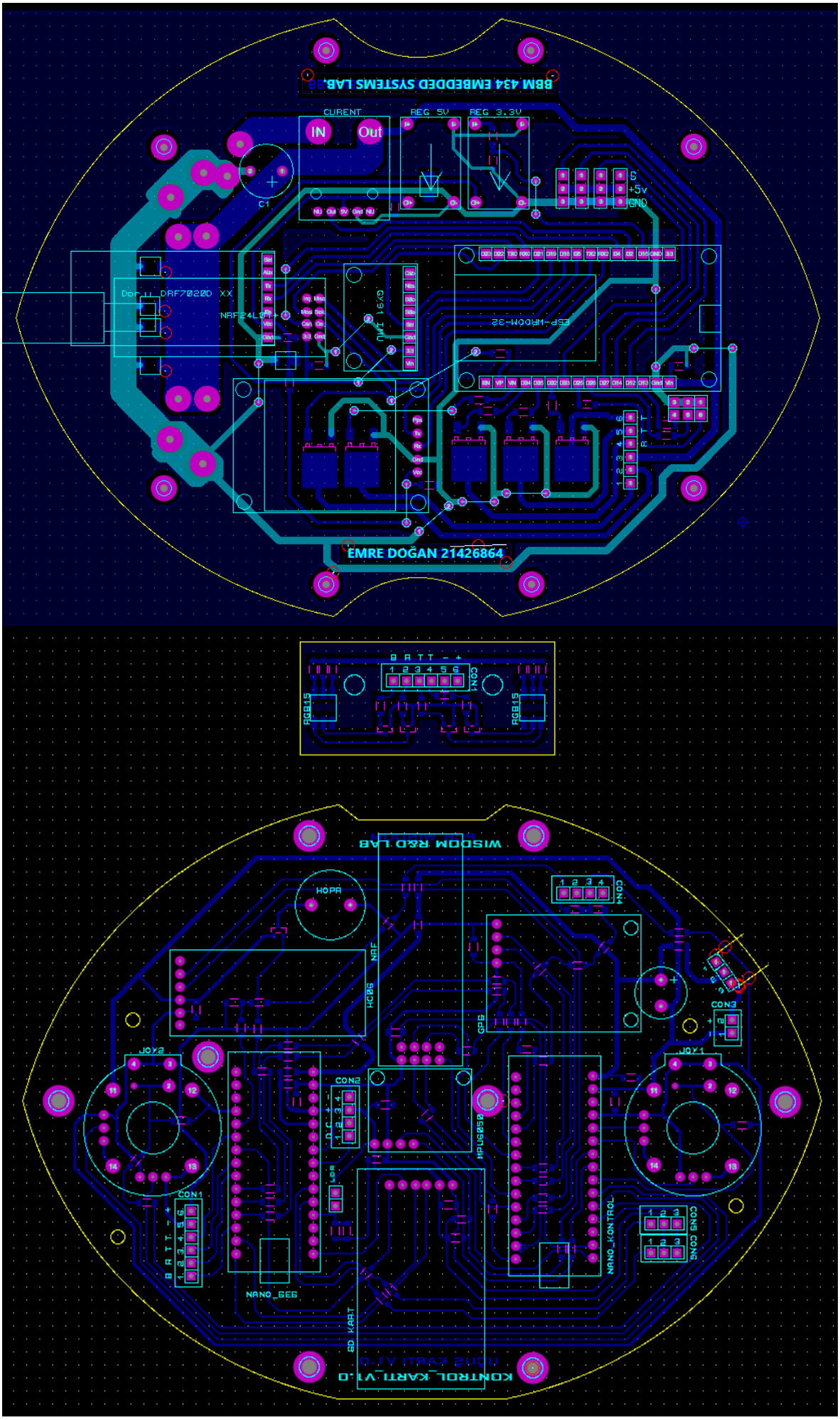
Kontrolcü:



Kumanda:



TASARIM


BBM 434 EMBEDDED SYSTEMS LAB
CURENT
IN
Out
REG 5V
REG 3.3V
C1
5
+5v
GND
EMRE DOĞAN 21426864
BATT - +
CON1
RGB15
RGB15
WISDOM R&D LAB
HOPR
NRF
HC05
GPS
CON4
CON3
JOY2
JOY1
CON2
MPU6050
LCD
NANO_GEB
SD KART
NANO_KONTROL
CONS
CONS
KONTROL_KARTI_V1.0

YAZILIM

Geliştirme ortamı olarak Arduino IDE 1.8.5 kullanılmıştır.

ESP WROOM-32 ’yi Arduino IDE ’si üzerinden programlamak için:

https://github.com/espressif/arduino-esp32/blob/master/docs/arduino-ide/windows.md

Gerekli Kütüphaneler

Kontolrcü:

Servo.H	
SPI.h	
nRF24L01.h	
RF24.h	https://github.com/nRF24/RF24
Wire.h	
MPU9250_asukiaaa.h	https://github.com/asukiaaa/MPU9250_asukiaaa

Kumanda:

SPI.h

nRF24L01.h

RF24.h

String.h

Wire.h

math.h

ESC Kalibrasyonu

```
#include <Servo.h>


#define MIN_PULSE_LENGTH 1000 // Minimum pulse length in µs
#define MAX_PULSE_LENGTH 2000 // Maximum pulse length in µs

Servo motA, motB, motC, motD;

char data;

void setup() {

  Serial.begin(9600);

  motA.attach(12, 10, MIN_PULSE_LENGTH, MAX_PULSE_LENGTH);
  motB.attach(14, 11, MIN_PULSE_LENGTH, MAX_PULSE_LENGTH);
  motC.attach(27, 12, MIN_PULSE_LENGTH, MAX_PULSE_LENGTH);
  motD.attach(33, 13, MIN_PULSE_LENGTH, MAX_PULSE_LENGTH);

  displayInstructions();

}
```

```
void loop() {

  if (Serial.available()) {

    data = Serial.read();

    switch (data) {

      // 0
      case 48 : Serial.println("Sending minimum throttle");
                motA.writeMicroseconds(MIN_PULSE_LENGTH);
                motB.writeMicroseconds(MIN_PULSE_LENGTH);
                motC.writeMicroseconds(MIN_PULSE_LENGTH);
                motD.writeMicroseconds(MIN_PULSE_LENGTH);
      break;

      // 1
      case 49 : Serial.println("Sending maximum throttle");
                motA.writeMicroseconds(MAX_PULSE_LENGTH);
                motB.writeMicroseconds(MAX_PULSE_LENGTH);
                motC.writeMicroseconds(MAX_PULSE_LENGTH);
                motD.writeMicroseconds(MAX_PULSE_LENGTH);
      break;

      // 2
      case 50 : Serial.print("Running test in 3");
                delay(1000);
                Serial.print(" 2");
                delay(1000);
                Serial.println(" 1...");
                delay(1000);
                test();
      break;

      }
  }


}

/**
 * Test function: send min throttle to max throttle to each ESC.
 */
void test()
{
    for (int i = MIN_PULSE_LENGTH; i <= MAX_PULSE_LENGTH; i += 5) {
        Serial.print("Pulse length = ");
        Serial.println(i);

        motA.writeMicroseconds(i);
        motB.writeMicroseconds(i);
        motC.writeMicroseconds(i);
        motD.writeMicroseconds(i);

        delay(200);
    }

    Serial.println("STOP");
    motA.writeMicroseconds(MIN_PULSE_LENGTH);
    motB.writeMicroseconds(MIN_PULSE_LENGTH);
    motC.writeMicroseconds(MIN_PULSE_LENGTH);
    motD.writeMicroseconds(MIN_PULSE_LENGTH);
}

/**
```

```
 * Displays instructions to user
 */
void displayInstructions()
{
    Serial.println("READY - PLEASE SEND INSTRUCTIONS AS FOLLOWING :");
    Serial.println("\t0 : Send min throttle");
    Serial.println("\t1 : Send max throttle");
    Serial.println("\t2 : Run test function\n");
}
```

Demo 1

Throttle

Kumanda:

```
#include  <SPI.h>
#include "nRF24L01.h"
#include "RF24.h"
#include <String.h>
#include <Wire.h>
#include <math.h>

const int MPU=0x68;
int16_t AcX,AcY,AcZ,Tmp;
int8_t PITCH, ROLL;
int16_t THROTTLE, YAW;
long THROTTLE_CAL=0, YAW_CAL=0, PITCH_CAL=0, ROLL_CAL=0;
int16_t HOWER;
void writeRawData();

void IMU(){
  Wire.beginTransmission(MPU); /* I2C haberleÅŸmesi yapÄ±lacak kart seÃ§ildi */
  Wire.write(0x3B); /* 0x3B adresindeki register'a ulaÅŸÄ±ldÄ± */
  Wire.endTransmission(false);
  Wire.requestFrom(MPU,14,true);/* 14 BYTE'lÄ±k veri istendi */
  AcX=Wire.read()<<8|Wire.read();
  AcY=Wire.read()<<8|Wire.read();
  AcZ=Wire.read()<<8|Wire.read(); // Tmp=Wire.read()<<8|Wire.read();



  AcX = map(AcX,0,16000,0,250);
  AcY = map(AcY,0,16000,0,250);
  AcZ = map(AcZ,0,16000,0,250);

}


RF24 radio(8, 7);
const uint64_t pipe = 0xE8E8F0F0E1LL;
int THRESHOLD = 1500;
bool condition = 0;
bool flag = 0;

struct dataStruct{
  int16_t throttle;
  int16_t yaw;
  int8_t pitch;
  int8_t roll;
  int16_t pot;
  bool switch_l;
  bool switch_r;
  bool button_l;
```

```
  bool button_r;
  bool k_status;
}data;

void setup(void) {

  Wire.begin();
  Wire.beginTransmission(MPU);
  Wire.write(0x6B);
  Wire.write(0); /* MPU-6050 Ã§alÄ±ÅŸtÄ±rÄ±ldÄ± */
  Wire.endTransmission(true); /* I2C haberleÅŸmesi baÅŸlatÄ±ldÄ± ve MPU-6050'nin ilk
ayarlarÄ± yapÄ±ldÄ± */

  delay(100);
  Serial.begin(115200);

  pinMode(A0, INPUT); /* sol joy buton */
  pinMode(A1, INPUT); /* sol joy x yaw */
  pinMode(A2, INPUT); /* sol joy y throttle */
  pinMode(A3, INPUT); /* sağ joy buton */
  pinMode(A6, INPUT); /* sağ joy x roll */
  pinMode(A7, INPUT); /* sağ joy y pitch */
  pinMode(A4, INPUT); /* imu SDA pin4 */
  pinMode(A5, INPUT); /* imu SCA pin5 */
  pinMode(2 , INPUT); /* switch sağ */
  pinMode(3 , INPUT); /* switch sol */
  pinMode(4 , INPUT);

  for(int i=0;i<200;i++){
    THROTTLE_CAL    += analogRead(A2);
    YAW_CAL         += analogRead(A1);
    PITCH_CAL       += analogRead(A7);
    ROLL_CAL        += analogRead(A6);
    }

  THROTTLE_CAL    = THROTTLE_CAL/200;
  YAW_CAL         = YAW_CAL/200;
  PITCH_CAL       = PITCH_CAL/200;
  ROLL_CAL        = ROLL_CAL/200;




  Serial.println("Kalibrasyon tamamlandi, kumanda hazir.");

  while( digitalRead(3)!=0 || digitalRead(2)!=0 ||
        digitalRead(A0)!=1 || digitalRead(4)!=1 ){
    Serial.println("Kumandayi acmak icin sag switch 0 konumunda iken joystick
                    butonlarina basiniz!");
    }

  radio.begin();
  radio.setChannel(2);
  radio.setPayloadSize(13);
  radio.setDataRate(RF24_1MBPS);
  radio.openWritingPipe(pipe);

}

void loop(void) {

  if(analogRead(A2)-THROTTLE_CAL>2)
    THROTTLE    =  map(analogRead(A2),THROTTLE_CAL,670,1500,2000); // yukseklik
  else
    THROTTLE    =  map(analogRead(A2),100,THROTTLE_CAL,100,1500);  // yukseklik
```

```
if(analogRead(A1)-YAW_CAL>2)
  YAW         = -map(analogRead(A1),YAW_CAL,620,0,180);          // eksen
else
  YAW         = -map(analogRead(A1),60,YAW_CAL,-180,0);          // eksen

if(analogRead(A7)-PITCH_CAL>2)
  PITCH       =  map(analogRead(A7),PITCH_CAL,680,0,30);         // ileri-geri
else
  PITCH       =  map(analogRead(A7),150,PITCH_CAL,-30,0);        // ileri-geri

if(analogRead(A6)-ROLL_CAL>2)
  ROLL        = -map(analogRead(A6),ROLL_CAL,680,0,30);          // sag-sol
else
  ROLL        = -map(analogRead(A6),150,ROLL_CAL,-30,0);         // sag-sol


if( (condition == 1) && (flag == 1))
  THROTTLE = THROTTLE;
else if( (condition ==1) && (flag == 0) && (THROTTLE > THRESHOLD) )
  THROTTLE = THRESHOLD;
else
  THROTTLE = THROTTLE-495;

if( (condition == 0) && (THROTTLE > THRESHOLD) )
  condition = 1;

if( (condition == 1) && (THROTTLE < 1050) ){
  flag = 1;
  data.k_status = flag;
}

if(THROTTLE<1000) THROTTLE  = 1000;
if(THROTTLE>2000) THROTTLE  = 2000;
if(YAW<-180)      YAW       = -180;
if(YAW>180)       YAW       = 180;
if(PITCH<-30)     PITCH     = -30;
if(PITCH>30)      PITCH     = 30;
if(ROLL<-30)      ROLL      = -30;
if(ROLL>30)       ROLL      = 30;

if(abs(YAW)<2)    YAW       = 0;
if(abs(PITCH)<2)  PITCH     = 0;
if(abs(ROLL)<2)   ROLL      = 0;

//writeRawData();

if(digitalRead(2)==0){
  HOWER = THROTTLE;
  data.throttle = THROTTLE;
}else{
  data.throttle = HOWER;
  IMU();
  Serial.print(AcX); Serial.print(" ");
  Serial.print(AcY); Serial.print(" ");
  Serial.println(AcZ);
}

data.yaw = YAW;
data.pitch = PITCH;
data.roll = ROLL;
data.pot = analogRead(A3);
data.switch_l = digitalRead(3);
data.switch_r = digitalRead(2);
data.button_l = digitalRead(A0);
```

```
  data.button_r = digitalRead(4);
  Serial.println("send ...");
  radio.write(&data, sizeof(data));
  delay(10);

  Serial.print(data.throttle);
  Serial.print("\t");
  Serial.print(data.yaw);
  Serial.print("\t");
  Serial.print(data.pitch);
  Serial.print("\t");
  Serial.print(data.roll);
  Serial.print("\t");
  Serial.print(data.pot);
  Serial.print("\t");
  Serial.print(data.switch_l);
  Serial.print("\t");
  Serial.print(data.switch_r);
  Serial.print("\t");
  Serial.print(data.button_l);
  Serial.print("\t");
  Serial.print(data.button_r);
  Serial.print("\t");
  Serial.print(data.k_status);
  Serial.println();


}
```

Kontrolcü:

```
#include<Servo.h>
#include <SPI.h>
#include "nRF24L01.h"
#include "RF24.h"

// use first channel of 16 channels (started from zero)
#define LEDC_CHANNEL_0     0
#define LEDC_CHANNEL_1     1
#define LEDC_CHANNEL_2     2
#define LEDC_CHANNEL_3     3

// use 8 bit precission for LEDC timer
#define LEDC_TIMER_8_BIT  8

// use 5000 Hz as a LEDC base frequency
#define LEDC_BASE_FREQ     1000
#define ESC_BASE_FREQ      50

// fade LED PIN (replace with LED_BUILTIN constant for built-in LED)
#define BAT_RED            13  //BATARYA
#define BAT_GREEN          32  //BATARYA
#define BLUE               26  //UCUS
#define GREEN              25  //UCUS

#define ESC_PIN_1              12
#define ESC_PIN_2              14
#define ESC_PIN_3              27
#define ESC_PIN_4              33

Servo ESC_1;
Servo ESC_2;
Servo ESC_3;
Servo ESC_4;
```

```
int brightness = 10;    // how bright the LED is
int fadeAmount = 1;    // how many points to fade the LED by
int pwm = 1000;
char Serialdata;
bool led_status = 0;
bool drone_status = 0;

// Arduino like analogWrite
// value has to be between 0 and valueMax

void ledcAnalogWrite(uint8_t channel, uint32_t value, uint32_t valueMax = 255) {
  // calculate duty, 8191 from 2 ^ 13 - 1
  uint32_t duty = (255 / valueMax) * min(value, valueMax);
  // write duty to LEDC
  ledcWrite(channel, duty);
}

uint32_t NUM = 5;

struct dataStruct{
  int16_t throttle;
  int16_t yaw;
  int8_t pitch;
  int8_t roll;
  int16_t pot;
  bool switch_l;
  bool switch_r;
  bool button_l;
  bool button_r;
  bool k_status;
}data;

RF24 radio(5,4);
const uint64_t pipe = 0xE8E8F0F0E1LL;
unsigned long timer = 0;

void setup() {

  Serial.begin(115200);
  radio.begin();
  radio.setChannel(2);
  radio.setPayloadSize(13);
  radio.setDataRate(RF24_1MBPS);
  radio.openReadingPipe(1,pipe);
  radio.startListening();

  ESC_1.attach(ESC_PIN_1, 10, 1000, 2000);
  ESC_2.attach(ESC_PIN_2, 11, 1000, 2000);
  ESC_3.attach(ESC_PIN_3, 12, 1000, 2000);
  ESC_4.attach(ESC_PIN_4, 13, 1000, 2000);

  // Setup timer and attach timer to a led pin

  ledcSetup(LEDC_CHANNEL_0, LEDC_BASE_FREQ, LEDC_TIMER_8_BIT);
  ledcAttachPin(BAT_RED, LEDC_CHANNEL_0);

  ledcSetup(LEDC_CHANNEL_1, LEDC_BASE_FREQ, LEDC_TIMER_8_BIT);
  ledcAttachPin(BAT_GREEN, LEDC_CHANNEL_1);

  ledcSetup(LEDC_CHANNEL_2, LEDC_BASE_FREQ, LEDC_TIMER_8_BIT);
  ledcAttachPin(BLUE, LEDC_CHANNEL_2);

  ledcSetup(LEDC_CHANNEL_3, LEDC_BASE_FREQ, LEDC_TIMER_8_BIT);
  ledcAttachPin(GREEN, LEDC_CHANNEL_3);
}
```

```
void loop() {

  ESC_1.writeMicroseconds(pwm);
  ESC_2.writeMicroseconds(pwm);
  ESC_3.writeMicroseconds(pwm);
  ESC_4.writeMicroseconds(pwm);

  if(led_status == 0){
    ledcAnalogWrite(LEDC_CHANNEL_0, 250-brightness);
    ledcAnalogWrite(LEDC_CHANNEL_1, brightness);
    ledcAnalogWrite(LEDC_CHANNEL_2, 250-map(pwm,1000,2000,250,5));
    ledcAnalogWrite(LEDC_CHANNEL_3, map(pwm,1000,2000,250,5));
  }else{
    ledcAnalogWrite(LEDC_CHANNEL_0, brightness);
    ledcAnalogWrite(LEDC_CHANNEL_1, 250-brightness);
    ledcAnalogWrite(LEDC_CHANNEL_2, 250-map(pwm,1000,2000,250,5));
    ledcAnalogWrite(LEDC_CHANNEL_3, map(pwm,1000,2000,250,5));
  }

  brightness = brightness + fadeAmount;

  if (brightness <= 0 || brightness >= 50) {
    fadeAmount = -fadeAmount;
  }

  if (radio.available()){
    radio.read(&data, sizeof(data));
    Serial.print(data.throttle);
    Serial.print(" ");
    Serial.print(data.yaw);
    Serial.print(" ");
    Serial.print(data.pitch);
    Serial.print(" ");
    Serial.print(data.roll);
    Serial.print(" ");
    Serial.print(data.pot);
    Serial.print("\t");
    Serial.print(data.switch_l);
    Serial.print("\t");
    Serial.print(data.switch_r);
    Serial.print("\t");
    Serial.print(data.button_l);
    Serial.print("\t");
    Serial.print(data.button_r);
    Serial.print("\t");
    Serial.print(data.k_status);
    Serial.println();
    delay(10);
    led_status = 1;
    if(data.k_status == 1)
      drone_status = 1;
    if(drone_status==data.k_status)
      pwm = data.throttle;
    timer = 0;
  }
  else{
    timer += 20-brightness/5;
    delay(20-brightness/5);
  //Serial.println("No radio available");
  }

  if(timer>500){
    led_status = 0;
  }
}
```

Demo 2

PID Kontrolü dahil edildi.

Kumanda:

```
#include  <SPI.h>
#include "nRF24L01.h"
#include "RF24.h"
#include <String.h>
#include <Wire.h>
#include <math.h>

const int MPU=0x68;
int16_t AcX,AcY,AcZ,Tmp;
int8_t PITCH, ROLL;
int16_t THROTTLE, YAW;
long THROTTLE_CAL=0, YAW_CAL=0, PITCH_CAL=0, ROLL_CAL=0;
int16_t HOWER;
void writeRawData();

void IMU(){
  Wire.beginTransmission(MPU); /* I2C haberleÅŸmesi yapÄ±lacak kart seÃ§ildi */
  Wire.write(0x3B); /* 0x3B adresindeki register'a ulaÅŸÄ±ldÄ± */
  Wire.endTransmission(false);
  Wire.requestFrom(MPU,14,true);/* 14 BYTE'lÄ±k veri istendi */
  AcX=Wire.read()<<8|Wire.read();
  AcY=Wire.read()<<8|Wire.read();
  AcZ=Wire.read()<<8|Wire.read(); // Tmp=Wire.read()<<8|Wire.read();
  AcX = map(AcX,0,16000,0,250);
  AcY = map(AcY,0,16000,0,250);
  AcZ = map(AcZ,0,16000,0,250);

}


RF24 radio(8, 7);
const uint64_t pipe = 0xE8E8F0F0E1LL;
int THRESHOLD = 1500;
bool condition = 0;
bool flag = 0;

struct dataStruct{
  int16_t throttle;
  int16_t yaw;
  int8_t pitch;
  int8_t roll;
  int16_t pot;
  bool switch_l;
  bool switch_r;
  bool button_l;
  bool button_r;
  bool k_status;

}data;

void setup(void) {

  Wire.begin();
  Wire.beginTransmission(MPU);
  Wire.write(0x6B);
  Wire.write(0); /* MPU-6050 Ã§alÄ±ÅŸtÄ±rÄ±ldÄ± */
  Wire.endTransmission(true); /* I2C haberleÅŸmesi baÅŸlatÄ±ldÄ± ve MPU-6050'nin ilk
ayarlarÄ± yapÄ±ldÄ± */
```

```
  delay(100);
  Serial.begin(115200);

  pinMode(A0, INPUT); /* sol joy buton */
  pinMode(A1, INPUT); /* sol joy x yaw */
  pinMode(A2, INPUT); /* sol joy y throttle */
  pinMode(A3, INPUT); /* sağ joy buton */
  pinMode(A6, INPUT); /* sağ joy x roll */
  pinMode(A7, INPUT); /* sağ joy y pitch */
  pinMode(A4, INPUT); /* imu SDA pin4 */
  pinMode(A5, INPUT); /* imu SCA pin5 */
  pinMode(2 , INPUT); /* switch sağ */
  pinMode(3 , INPUT); /* switch sol */
  pinMode(4 , INPUT);

  for(int i=0;i<200;i++){
    THROTTLE_CAL    += analogRead(A2);
    YAW_CAL         += analogRead(A1);
    PITCH_CAL       += analogRead(A7);
    ROLL_CAL        += analogRead(A6);
    }

  THROTTLE_CAL    = THROTTLE_CAL/200;
  YAW_CAL         = YAW_CAL/200;
  PITCH_CAL       = PITCH_CAL/200;
  ROLL_CAL        = ROLL_CAL/200;

  Serial.println("Kalibrasyon tamamlandi, kumanda hazir.");

  while( digitalRead(3)!=0 || digitalRead(2)!=0 ||
        digitalRead(A0)!=1 || digitalRead(4)!=1 ){
    Serial.println("Kumandayi acmak icin sag switch 0 konumunda iken joystick
                    butonlarina basiniz!");
    }

  radio.begin();
  radio.setChannel(2);
  radio.setPayloadSize(13);
  radio.setDataRate(RF24_1MBPS);
  radio.openWritingPipe(pipe);

}

void loop(void) {

  if(analogRead(A2)-THROTTLE_CAL>2)
    THROTTLE    =  map(analogRead(A2),THROTTLE_CAL,670,1500,2000); // yukseklik
  else
    THROTTLE    =  map(analogRead(A2),100,THROTTLE_CAL,100,1500);  // yukseklik

  if(analogRead(A1)-YAW_CAL>2)
    YAW         = -map(analogRead(A1),YAW_CAL,620,0,180);          // eksen
  else
    YAW         = -map(analogRead(A1),60,YAW_CAL,-180,0);          // eksen

  if(analogRead(A7)-PITCH_CAL>2)
    PITCH       =  map(analogRead(A7),PITCH_CAL,680,0,30);         // ileri-geri
  else
    PITCH       =  map(analogRead(A7),150,PITCH_CAL,-30,0);        // ileri-geri

  if(analogRead(A6)-ROLL_CAL>2)
    ROLL        = -map(analogRead(A6),ROLL_CAL,680,0,30);          // sag-sol
  else
    ROLL        = -map(analogRead(A6),150,ROLL_CAL,-30,0);         // sag-sol
```

```
if( (condition == 1) && (flag == 1))
  THROTTLE = THROTTLE;
else if( (condition ==1) && (flag == 0) && (THROTTLE > THRESHOLD) )
  THROTTLE = THRESHOLD;
else
  THROTTLE = THROTTLE-495;

if( (condition == 0) && (THROTTLE > THRESHOLD) )
  condition = 1;

if( (condition == 1) && (THROTTLE < 1050) ){
  flag = 1;
  data.k_status = flag;
}

if(THROTTLE<1000) THROTTLE  = 1000;
if(THROTTLE>2000) THROTTLE  = 2000;
if(YAW<-180)      YAW       = -180;
if(YAW>180)       YAW       = 180;
if(PITCH<-30)     PITCH     = -30;
if(PITCH>30)      PITCH     = 30;
if(ROLL<-30)      ROLL      = -30;
if(ROLL>30)       ROLL      = 30;

if(abs(YAW)<2)    YAW       = 0;
if(abs(PITCH)<2)  PITCH     = 0;
if(abs(ROLL)<2)   ROLL      = 0;

//writeRawData();

if(digitalRead(2)==0){
  HOWER = THROTTLE;
  data.throttle = THROTTLE;
}else{
  data.throttle = HOWER;
  IMU();
  Serial.print(AcX); Serial.print(" ");
  Serial.print(AcY); Serial.print(" ");
  Serial.println(AcZ);
}

data.yaw = YAW;
data.pitch = PITCH;
data.roll = ROLL;
data.pot = analogRead(A3);
data.switch_l = digitalRead(3);
data.switch_r = digitalRead(2);
data.button_l = digitalRead(A0);
data.button_r = digitalRead(4);
Serial.println("send ...");
radio.write(&data, sizeof(data));
delay(10);

Serial.print(data.throttle);
Serial.print("\t");
Serial.print(data.yaw);
Serial.print("\t");
Serial.print(data.pitch);
Serial.print("\t");
Serial.print(data.roll);
Serial.print("\t");
Serial.print(data.pot);
Serial.print("\t");
Serial.print(data.switch_l);
```

```
  Serial.print("\t");
  Serial.print(data.switch_r);
  Serial.print("\t");
  Serial.print(data.button_l);
  Serial.print("\t");
  Serial.print(data.button_r);
  Serial.print("\t");
  Serial.print(data.k_status);
  Serial.println();


}
```

Kontrolcü:

```
#include<Servo.h>
#include <SPI.h>
#include "nRF24L01.h"
#include "RF24.h"

//-------------MPU-------------
#include <Wire.h>
#include <MPU9250_asukiaaa.h>

#ifdef _ESP32_HAL_I2C_H_
#define SDA_PIN 21
#define SCL_PIN 22
#endif

MPU9250 mySensor;
uint8_t sensorId;

//------------------------------
#define PI acos(-1) // 3.14.....

#define YAW      0
#define PITCH    1
#define ROLL     2
#define THROTTLE 3

#define X           0     // X axis
#define Y           1     // Y axis
#define Z           2     // Z axis
#define SSF_GYRO    65.5  // Sensitivity Scale Factor of the gyro from datasheet
#define SSF_ACC     8.192 // Sensitivity Scale Factor of the accelerometer from
datasheet

// ---------------- Controller variables ------------------------------------
#define LEDC_CHANNEL_0     0
#define LEDC_CHANNEL_1     1
#define LEDC_CHANNEL_2     2
#define LEDC_CHANNEL_3     3
#define LEDC_TIMER_16_BIT  8
#define LEDC_BASE_FREQ     1000
#define ESC_BASE_FREQ      50
#define BAT_RED            13  //BATARYA
#define BAT_GREEN          32  //BATARYA
#define BLUE               26  //UCUS
#define GREEN              25  //UCUS

#define ESC_PIN_1              33 // Front Left
#define ESC_PIN_2              14 // Front Right
#define ESC_PIN_3              27 // Rear Left
#define ESC_PIN_4              12 // Rear Right
```

```
Servo ESC_1;
Servo ESC_2;
Servo ESC_3;
Servo ESC_4;

int brightness = 10;    // how bright the LED is
int fadeAmount = 1;    // how many points to fade the LED by
int pwm = 1000;
char Serialdata;
bool led_status = 0;
bool drone_status = 0;

void ledcAnalogWrite(uint8_t channel, uint32_t value, uint32_t valueMax = 255) {
  // calculate duty, 8191 from 2 ^ 13 - 1
  uint32_t duty = (255 / valueMax) * min(value, valueMax);

  // write duty to LEDC
  ledcWrite(channel, duty);
}

uint32_t NUM = 5;

struct dataStruct{
  int16_t throttle;
  int16_t yaw;
  int8_t pitch;
  int8_t roll;
  int16_t pot;
  bool switch_l;
  bool switch_r;
  bool button_l;
  bool button_r;
  bool k_status;
}data;

RF24 radio(5,4);
const uint64_t pipe = 0xE8E8F0F0E1LL;
unsigned long timer = 0;

//-------------------------------------------------------------------------

void setupMpu9250Registers();
void calibrateMpu9250();

// ---------------------- MPU variables ------------------------------------
int gyro_x, gyro_y, gyro_z;
long acc_x, acc_y, acc_z, acc_total_vector;
long gyro_x_cal, gyro_y_cal, gyro_z_cal;
float angle_pitch, angle_roll;
int angle_pitch_buffer, angle_roll_buffer;
boolean set_gyro_angles;
float angle_roll_acc, angle_pitch_acc;

// ---------------------- Variables for servo signal generation ------------
unsigned long loop_timer;
unsigned long now, difference;

unsigned long pulse_length_esc1 = 1000,
        pulse_length_esc2 = 1000,
        pulse_length_esc3 = 1000,
        pulse_length_esc4 = 1000;

// ------------- Global variables used for PID automation -----------------
```

```
float errors[3];                    // Measured errors (compared to instructions) :
[Yaw, Pitch, Roll]
float error_sum[3]      = {0, 0, 0}; // Error sums (used for integral component) :
[Yaw, Pitch, Roll]
float previous_error[3] = {0, 0, 0}; // Last errors (used for derivative component) :
[Yaw, Pitch, Roll]
float measures[3]       = {0, 0, 0}; // Angular measures : [Yaw, Pitch, Roll]
// ------------------------------------------------------------------------

/**
 * Setup configuration
 */
void setup() {

  Serial.begin(115200);
  setupMpu9250Registers();
  calibrateMpu9250();

  radio.begin();
  radio.setChannel(2);
  radio.setPayloadSize(13);
  radio.setDataRate(RF24_1MBPS);
  radio.openReadingPipe(1,pipe);
  radio.startListening();

  ESC_1.attach(ESC_PIN_1, 10, 1000, 2000);
  ESC_2.attach(ESC_PIN_2, 11, 1000, 2000);
  ESC_3.attach(ESC_PIN_3, 12, 1000, 2000);
  ESC_4.attach(ESC_PIN_4, 13, 1000, 2000);

  ledcSetup(LEDC_CHANNEL_0, LEDC_BASE_FREQ, LEDC_TIMER_16_BIT);
  ledcAttachPin(BAT_RED, LEDC_CHANNEL_0);

  ledcSetup(LEDC_CHANNEL_1, LEDC_BASE_FREQ, LEDC_TIMER_16_BIT);
  ledcAttachPin(BAT_GREEN, LEDC_CHANNEL_1);

  ledcSetup(LEDC_CHANNEL_2, LEDC_BASE_FREQ, LEDC_TIMER_16_BIT);
  ledcAttachPin(BLUE, LEDC_CHANNEL_2);


  ledcSetup(LEDC_CHANNEL_3, LEDC_BASE_FREQ, LEDC_TIMER_16_BIT);
  ledcAttachPin(GREEN, LEDC_CHANNEL_3);

  #ifdef _ESP32_HAL_I2C_H_ // For ESP32
    Wire.begin(SDA_PIN, SCL_PIN); // SDA, SCL
  #else
    Wire.begin();
  #endif

}

void loop() {

    if(led_status == 0){
    ledcAnalogWrite(LEDC_CHANNEL_0, 250-brightness);
    ledcAnalogWrite(LEDC_CHANNEL_1, brightness);
    ledcAnalogWrite(LEDC_CHANNEL_2, 250-map(pwm,1000,2000,250,5));
    ledcAnalogWrite(LEDC_CHANNEL_3, map(pwm,1000,2000,250,5));
  }else{
    ledcAnalogWrite(LEDC_CHANNEL_0, brightness);
    ledcAnalogWrite(LEDC_CHANNEL_1, 250-brightness);
    ledcAnalogWrite(LEDC_CHANNEL_2, 250-map(pwm,1000,2000,250,5));
    ledcAnalogWrite(LEDC_CHANNEL_3, map(pwm,1000,2000,250,5));
  }
```

```
brightness = brightness + fadeAmount;

// reverse the direction of the fading at the ends of the fade:100-65400
if (brightness <= 0 || brightness >= 50) {
  fadeAmount = -fadeAmount;
}

if (radio.available()){

  radio.read(&data, sizeof(data));
  Serial.print(data.throttle);
  Serial.print("\t");
  Serial.print(data.yaw);
  Serial.print("\t");
  Serial.print(data.pitch);
  Serial.print("\t");
  Serial.print(data.roll);
  Serial.print("\t");
  Serial.print(data.pot);
  Serial.print("\t");
  Serial.print(data.switch_l);
  Serial.print("\t");
  Serial.print(data.switch_r);
  Serial.print("\t");
  Serial.print(data.button_l);
  Serial.print("\t");
  Serial.print(data.button_r);
  Serial.print("\t");
  Serial.print(data.k_status);
  Serial.println();
  delay(2);

  led_status = 1;
  if(data.k_status == 1)
    drone_status = 1;
  if(drone_status==data.k_status)
    pwm = data.throttle;
  timer = 0;


}else{
  timer += 20-brightness/5;
  delay(20-brightness/5);
}

if(timer>500){
  led_status = 0;
  ESC_1.writeMicroseconds(1000);
  ESC_2.writeMicroseconds(1000);
  ESC_3.writeMicroseconds(1000);
  ESC_4.writeMicroseconds(1000);
  data.throttle=1000;
  data.yaw=0;
  data.pitch=0;
  data.roll=0;
  data.pot=0;
}


  // 1. First, read angular values from MPU-6050

  readSensor();

  convertRawValues();
```

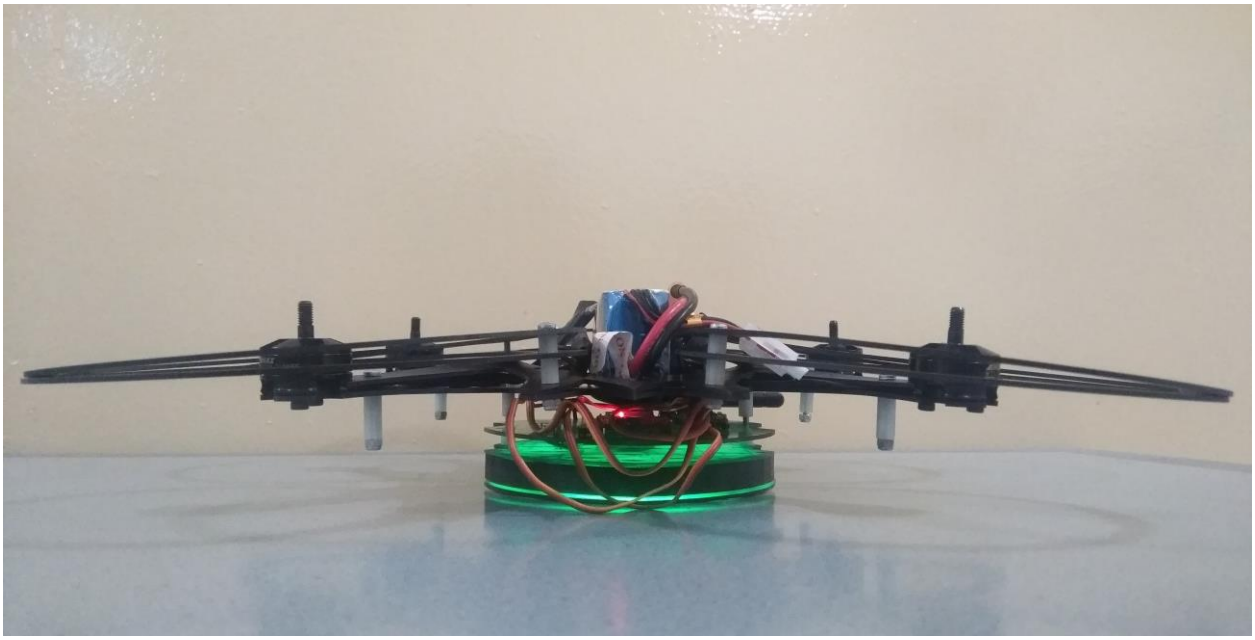
```
    // 3. Calculate errors comparing received instruction with measures
    calculateErrors();

    // 4. Calculate motors speed with PID controller
    automation();

    // 5. Apply motors speed
    applyMotorSpeed();
}

void applyMotorSpeed() {

  Serial.print("\tSpeed:");
  Serial.print("\t1: ");
  Serial.print(pulse_length_esc1);
  Serial.print("\t2: ");
  Serial.print(pulse_length_esc2);
  Serial.print("\t3: ");
  Serial.print(pulse_length_esc3);
  Serial.print("\t4: ");
  Serial.println(pulse_length_esc4);

  ESC_1.writeMicroseconds(pulse_length_esc1);
  ESC_2.writeMicroseconds(pulse_length_esc2);
  ESC_3.writeMicroseconds(pulse_length_esc3);
  ESC_4.writeMicroseconds(pulse_length_esc4);
}

void readSensor() {

  mySensor.accelUpdate();
  acc_x = mySensor.accelX();
  acc_y = mySensor.accelY();
  acc_z = mySensor.accelZ();
  Serial.print("\taccelX: " + String(acc_x));
  Serial.print("\taccelY: " + String(acc_y));
  Serial.print("\taccelZ: " + String(acc_z));

  mySensor.gyroUpdate();
  gyro_x = mySensor.gyroX();
  gyro_y = mySensor.gyroY();
  gyro_z = mySensor.gyroZ();
  Serial.print("\tgyroX: " + String(gyro_x));
  Serial.print("\tgyroY: " + String(gyro_y));
  Serial.print("\tgyroZ: " + String(gyro_z));

}

void convertRawValues() {
    gyro_x -= gyro_x_cal;                                                //Subtract
the offset calibration value from the raw gyro_x value
    gyro_y -= gyro_y_cal;                                                //Subtract
the offset calibration value from the raw gyro_y value
    gyro_z -= gyro_z_cal;                                                //Subtract
the offset calibration value from the raw gyro_z value

    //Gyro angle calculations
    //0.0000611 = 1 / (250Hz / 65.5)
    angle_pitch += gyro_x * 0.0000611;                                   //Calculate
the traveled pitch angle and add this to the angle_pitch variable
    angle_roll += gyro_y * 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
```

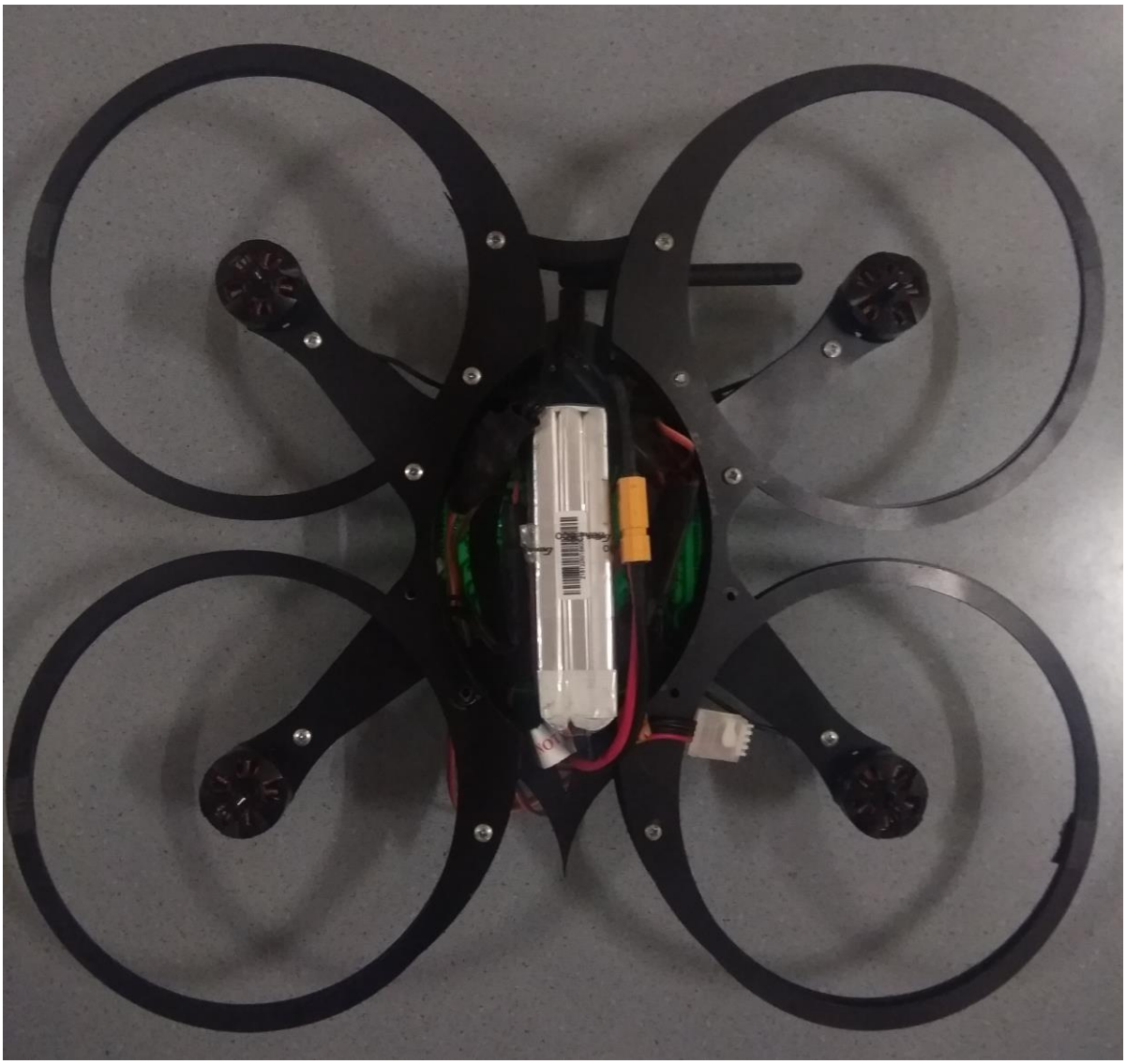
```
    angle_pitch += angle_roll * sin(gyro_z * 0.000001066);               //If the IMU
has yawed transfer the roll angle to the pitch angel
    angle_roll -= angle_pitch * sin(gyro_z * 0.000001066);               //If the IMU
has yawed transfer the pitch angle to the roll angel

    //Accelerometer angle calculations
    acc_total_vector = sqrt((acc_x*acc_x)+(acc_y*acc_y)+(acc_z*acc_z));  //Calculate
the total accelerometer vector
    //57.296 = 1 / (3.142 / 180) The Arduino asin function is in radians
    angle_pitch_acc = asin((float)acc_y/acc_total_vector)* 57.296;       //Calculate
the pitch angle
    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

    if (set_gyro_angles) {                                                    //If the
IMU is already started
        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
    } else {                                                                  //At
first start
        angle_pitch = angle_pitch_acc;                                        //Set the
gyro pitch angle equal to the accelerometer pitch angle
        angle_roll = angle_roll_acc;                                          //Set the
gyro roll angle equal to the accelerometer roll angle
        set_gyro_angles = true;                                               //Set the
IMU started flag
    }

    //To dampen the pitch and roll angles a complementary filter is used
    measures[ROLL]  = measures[ROLL] * 0.9 + angle_pitch * 0.1;   //Take 90% of the
output pitch value and add 10% of the raw pitch value
    measures[PITCH] = measures[PITCH] * 0.9 + angle_roll * 0.1;      //Take 90% of
the output roll value and add 10% of the raw roll value
    measures[YAW]   = gyro_z / SSF_GYRO;
}

void automation() {
    float Kp[3]       = {3, 5, 5};  //P coefficients in that order : Yaw, Pitch, Roll
    float Ki[3]       = {0.0, 0, 0};//I coefficients in that order : Yaw, Pitch, Roll
    float Kd[3]       = {0, 0, 0};  //D coefficients in that order : Yaw, Pitch, Roll
    float deltaErr[3] = {0, 0, 0};  // Error deltas in that order :  Yaw, Pitch, Roll
    float yaw         = 0;
    float pitch       = 0;
    float roll        = 0;

    // Initialize motor commands with throttle
    pulse_length_esc1 = data.throttle;
    pulse_length_esc2 = data.throttle;
    pulse_length_esc3 = data.throttle;
    pulse_length_esc4 = data.throttle;

    // Do not calculate anything if throttle is 0
    if (data.throttle >= 1012) {
        // Calculate sum of errors : Integral coefficients
```

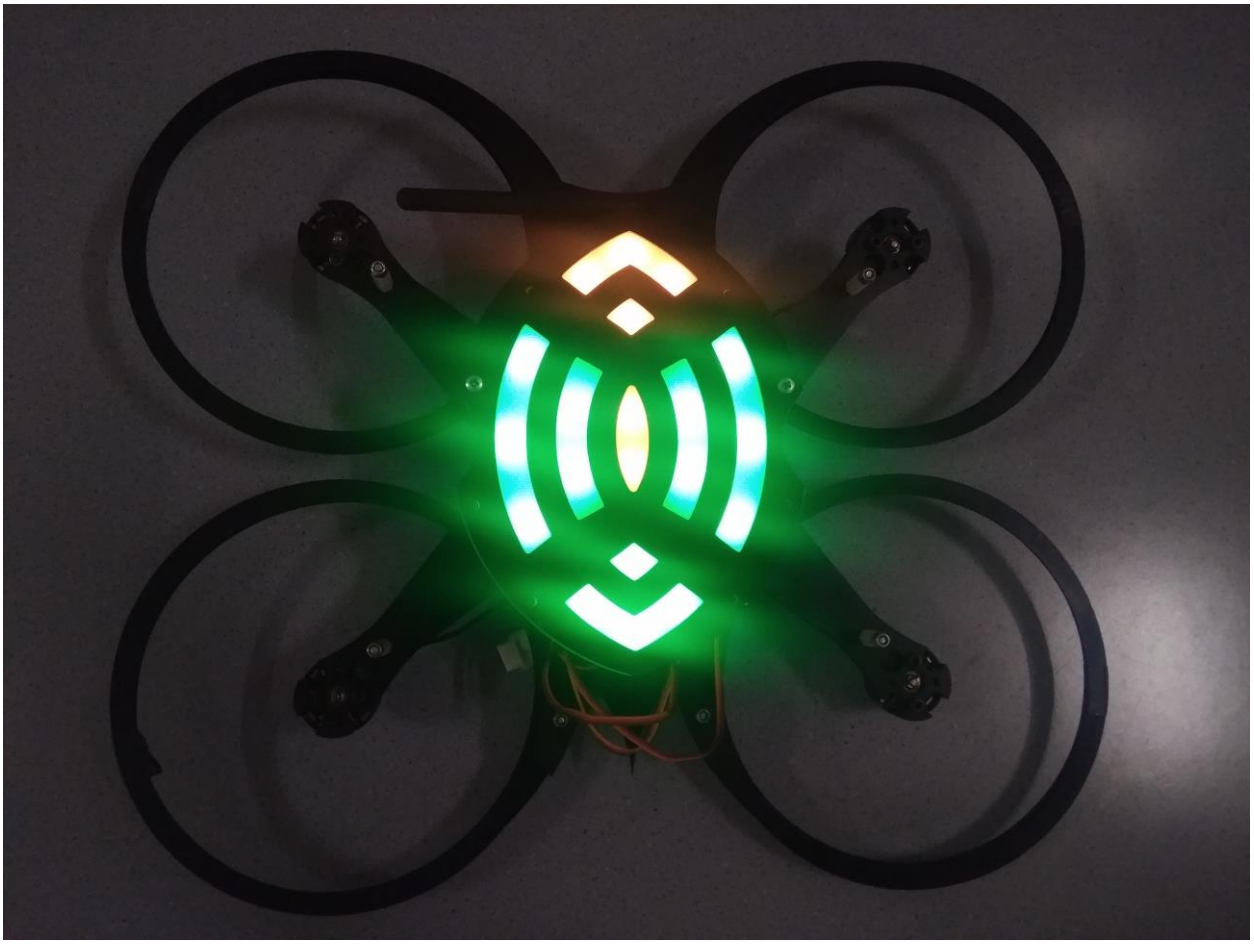
```
        error_sum[YAW] += errors[YAW];
        error_sum[PITCH] += errors[PITCH];
        error_sum[ROLL] += errors[ROLL];

        // Calculate error delta : Derivative coefficients
        deltaErr[YAW] = errors[YAW] - previous_error[YAW];
        deltaErr[PITCH] = errors[PITCH] - previous_error[PITCH];
        deltaErr[ROLL] = errors[ROLL] - previous_error[ROLL];

        // Save current error as previous_error for next time
        previous_error[YAW] = errors[YAW];
        previous_error[PITCH] = errors[PITCH];
        previous_error[ROLL] = errors[ROLL];

        yaw = (errors[YAW] * Kp[YAW]) + (error_sum[YAW] * Ki[YAW]) + (deltaErr[YAW] *
Kd[YAW]);
        pitch = (errors[PITCH] * Kp[PITCH]) + (error_sum[PITCH] * Ki[PITCH]) +
(deltaErr[PITCH] * Kd[PITCH]);
        roll = (errors[ROLL] * Kp[ROLL]) + (error_sum[ROLL] * Ki[ROLL]) +
(deltaErr[ROLL] * Kd[ROLL]);

        // Yaw - Lacet (Z axis)
        pulse_length_esc1 -= yaw;
        pulse_length_esc4 -= yaw;
        pulse_length_esc3 += yaw;
        pulse_length_esc2 += yaw;

        // Pitch - Tangage (Y axis)
        pulse_length_esc1 += pitch;
        pulse_length_esc2 += pitch;
        pulse_length_esc3 -= pitch;
        pulse_length_esc4 -= pitch;

        // Roll - Roulis (X axis)
        pulse_length_esc1 -= roll;
        pulse_length_esc3 -= roll;
        pulse_length_esc2 += roll;
        pulse_length_esc4 += roll;
    }

    // Maximum value
    if (pulse_length_esc1 > 2000) pulse_length_esc1 = 2000;
    if (pulse_length_esc2 > 2000) pulse_length_esc2 = 2000;
    if (pulse_length_esc3 > 2000) pulse_length_esc3 = 2000;
    if (pulse_length_esc4 > 2000) pulse_length_esc4 = 2000;

    // Minimum value
    if (pulse_length_esc1 < 1000) pulse_length_esc1 = 1000;
    if (pulse_length_esc2 < 1000) pulse_length_esc2 = 1000;
    if (pulse_length_esc3 < 1000) pulse_length_esc3 = 1000;
    if (pulse_length_esc4 < 1000) pulse_length_esc4 = 1000;
}


/**
 * Calculate errors of Yaw, Pitch & Roll: this is simply the difference between the
measure and the command.
 *
 * @return void
 */
void calculateErrors() {

    errors[YAW]   = measures[YAW]   - data.yaw;
    errors[PITCH] = measures[PITCH] - data.pitch;
    errors[ROLL]  = measures[ROLL]  - data.roll;
```

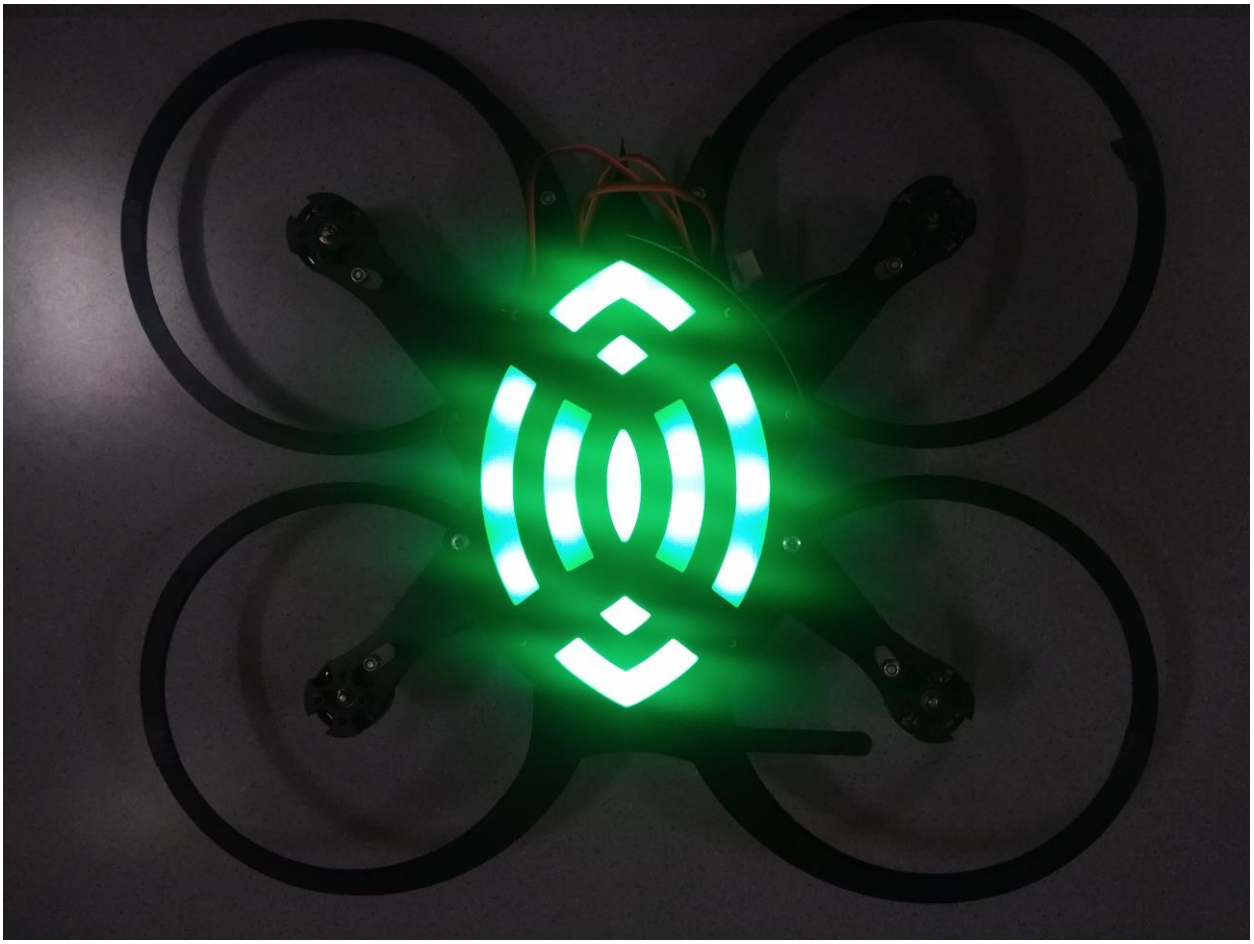
```
}

void setupMpu9250Registers() {

  mySensor.setWire(&Wire);
  mySensor.beginAccel();
  mySensor.beginGyro();
  mySensor.beginMag();
  sensorId = mySensor.readId();
  Serial.print("SendorId: ");
  Serial.println(sensorId);

}

void calibrateMpu9250(){


for (int cal_int = 0; cal_int < 2000; cal_int++) {  //Run this code 2000 times
        readSensor();         //Read the raw acc and gyro data from the MPU-6050
        gyro_x_cal += gyro_x; //Add the gyro x-axis offset to the gyro_x_cal variable
        gyro_y_cal += gyro_y; //Add the gyro y-axis offset to the gyro_y_cal variable
        gyro_z_cal += gyro_z; //Add the gyro z-axis offset to the gyro_z_cal variable
        delay(3);             //Delay 3us to simulate the 250Hz program loop
    }
gyro_x_cal /= 2000;//Divide the gyro_x_cal variable by 2000 to get the avarage offset
gyro_y_cal /= 2000;//Divide the gyro_y_cal variable by 2000 to get the avarage offset
gyro_z_cal /= 2000;//Divide the gyro_z_cal variable by 2000 to get the avarage offset

Serial.println("Calibration Done");

}
```

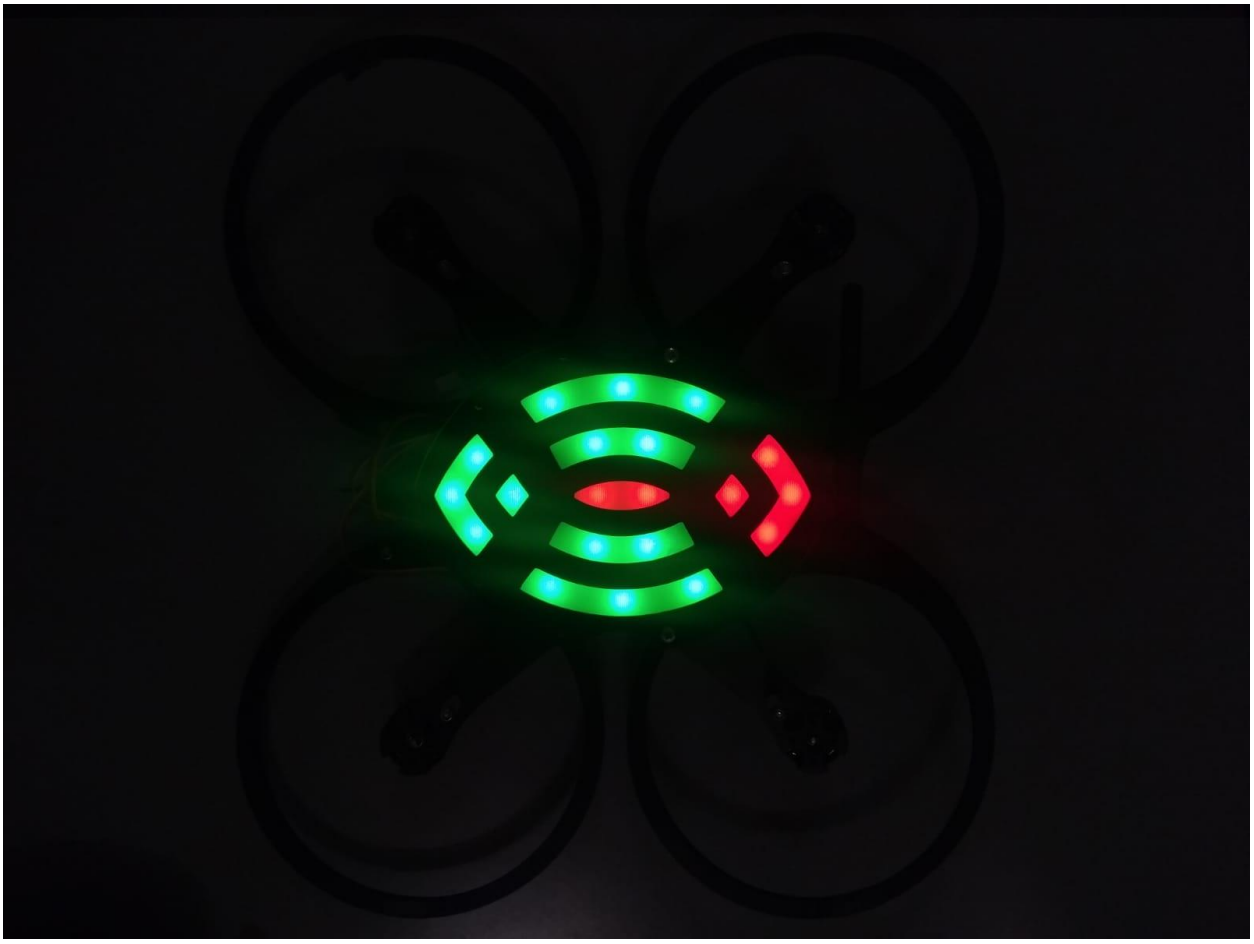
PROJE GÖRSELLERİ

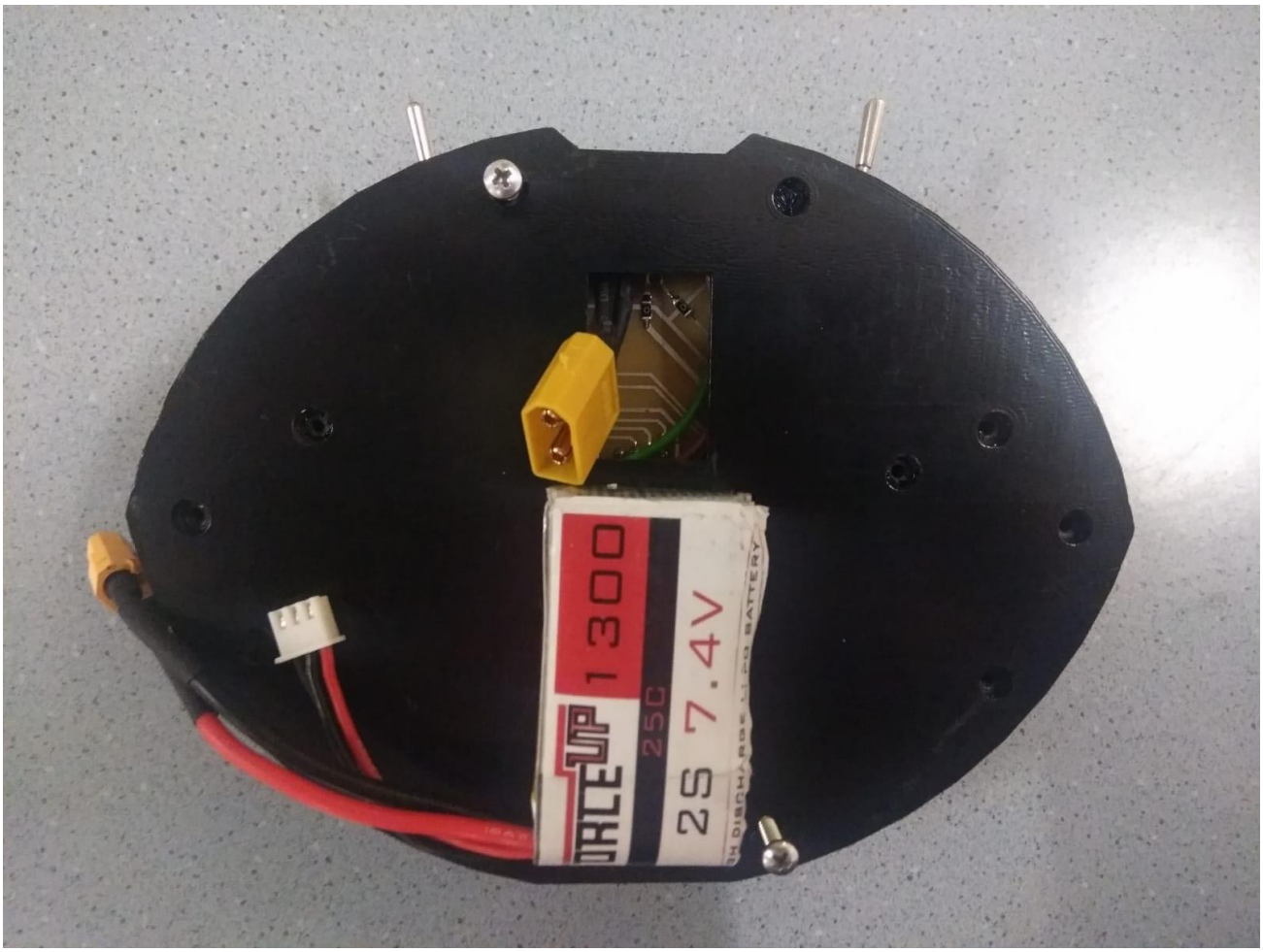











1300
25C
2S 7.4V