

ELEKTRONIKA II

Projekt Line Followera Enhanced

Mateusz Grudzień
gr 24

Politechnika Warszawska

1. Zalożenia projektu

Celem projektu była budowa robota spełniającego wymagania regulaminów zawodów typu Line Follower Enhanced. Głównymi problemami jakie powinny zostać rozwiązane są:

- podążanie po linii
- detekcja przeszkód na trasie
- omijanie przeszkody i bezpieczny powrót na trasę
- wykrywanie zmian oświetlenia
- pokonywanie skrzyżowań, oraz przerw w linii

2. Najważniejsze użyte części

Jako płytę sterującą wybrałem Arduino MEGA 2560 oparte na mikrokontrolerze Atmega 2560, ze względu na większą ilość wejść analogowych niż w przypadku mniejszych modeli z rodziny Arduino, oraz łatwiejszą obsługę niż w przypadku wykorzystania samego mikrokontrolera.

Do komunikacji wykorzystałem popularny moduł bluetooth HC-05 z interfejsem UART.

W celu dokładnego sterowania obrotom robota użyłem modułu akcelerometru z żyroskopem MPU6050 wykorzystującego interfejs I2C.

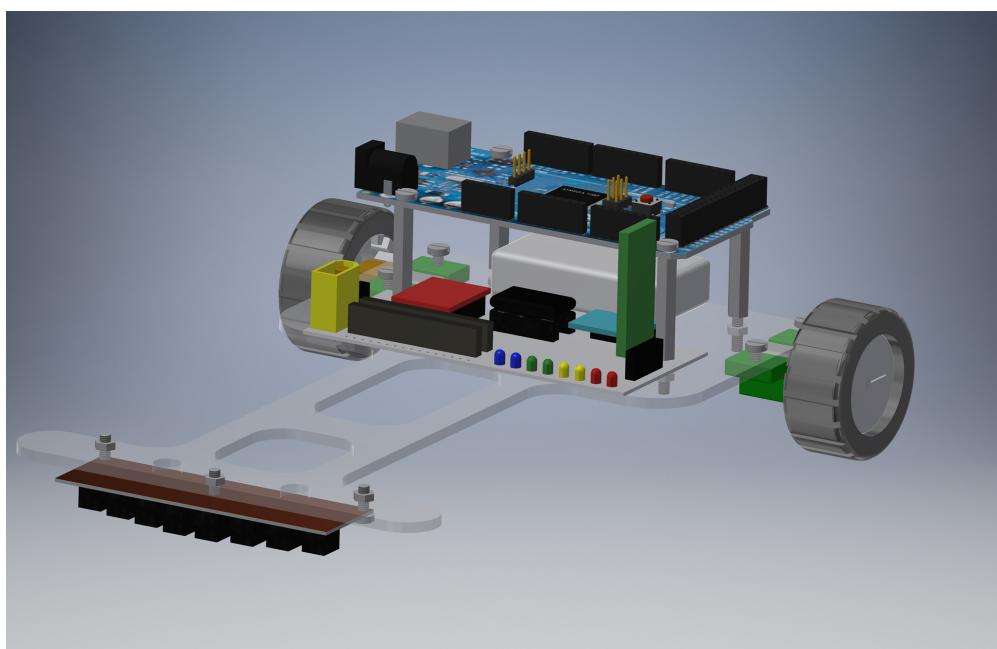
Do wysterowania diod sygnalizacyjnych użyłem ekspandera PCF8547P sterownego przy użyciu interfejsu I2C.

Za napęd posłużyły silniki firmy Mabuchi model N20 z reduktorem 1:30, sterowane przez mostek H TB6612FNG.

Za odczyt położenia linii odpowiada 8 transceptorów odbiorników CNY-70.

Wszystkie elementy elektroniczne zostały umieszczone na własnoręcznie wykonanych płytach PCB, natomiast za szkielet konstrukcji służy płyta z PMMA.

Płytki PCB zostały zaprojektowane w programie Eagle, oraz została wykonany model całego robota w Inventorze na podstawie którego wykonana została dokumentacja do cięcia PMMA laserem, oraz druku uchwytów na silniki na drukarce 3D, rendery znajdują się na dysku internetowym.



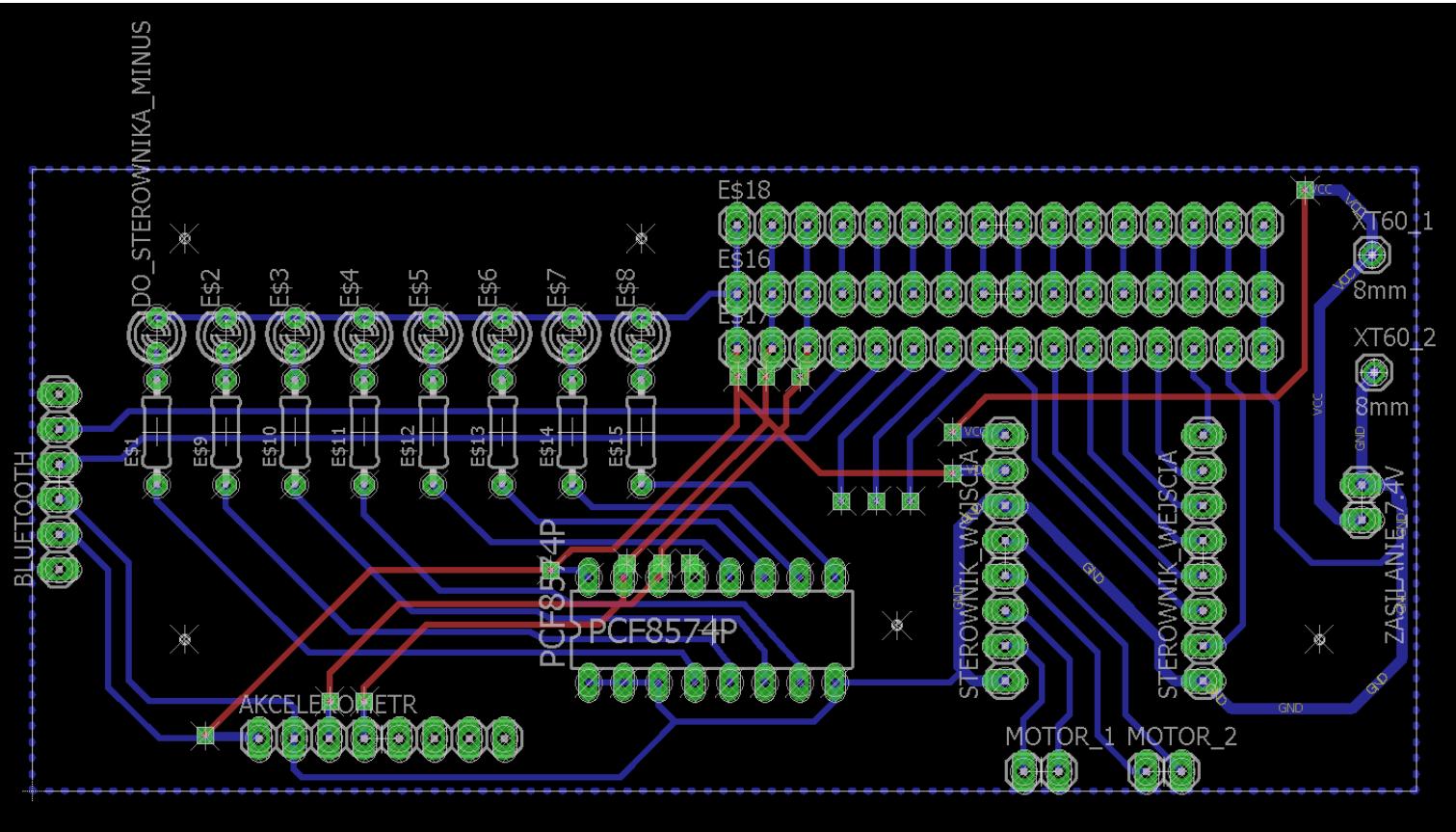
3. Kosztorys projektu

Mikrokontroler - Arduino Mega 2560	---->	30.20 zł
Mostek H - TB6612FNG	---->	5.82 zł
Silniki - Mabuchi N20 6V 300RPM	---->	2 x 11.25 zł
Koła - 43*19*3mm K346	---->	2 x 3.72 zł
Expander - PCF8574P	---->	1.50 zł
Moduł bluetooth - HC-05	---->	13.30 zł
Akcelerometr + żyroskop - MPU-6050	---->	6.30 zł
Diody 3mm	---->	8 x 0.04 zł
Czujnik ultradźwiękowy - HC - SR04	---->	2.90 zł
Fototranzystory - CNY-70	---->	8 x 1.24 zł
Fotorezystor	---->	0.02 zł
Buzzer z generatorem 5V	---->	0.42 zł
Rezystory 25 szt	---->	25 x 0.08 zł
Tranzystor BD137	---->	0.70 zł
Listwy golpin	---->	3.50 zł
Przewody - PCF8574P	---->	5.00 zł
Szkło akrylowe - 0.0286 m ² * 75zł	---->	2.15 zł
Laminat miedziany + płytki uniwersalne	---->	3.00 zł
SUMA		116.99 zł

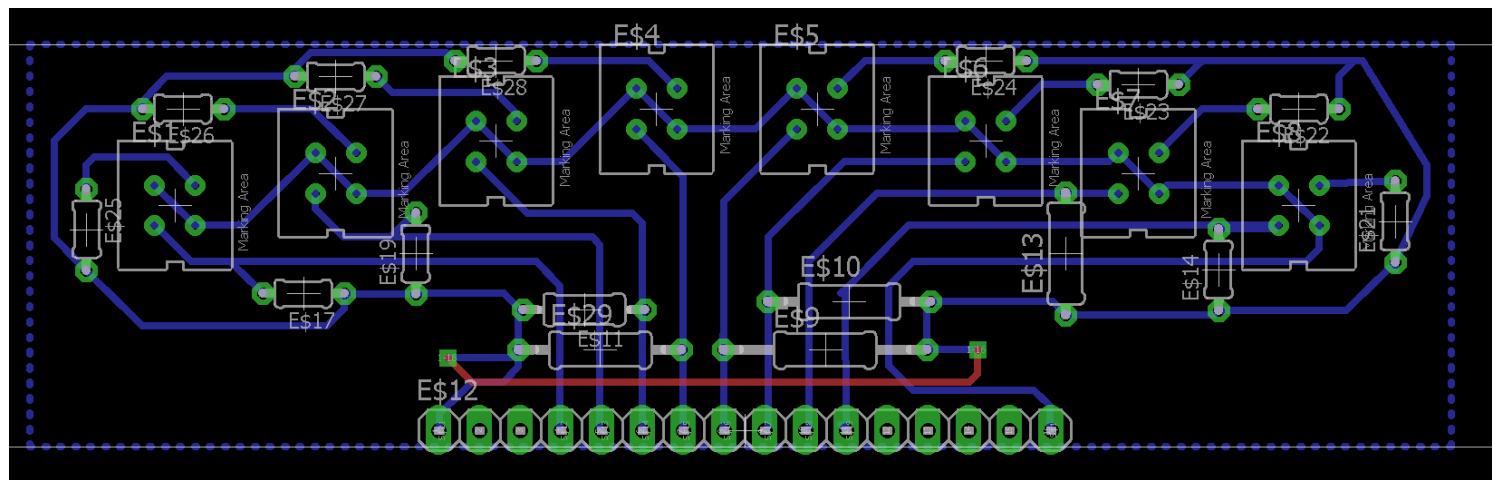
4. Schemat płytki PCB

Obrazy w większym rozmiarze, oraz zdjęcia z trawienia i montażu znajdują się na podanym w pkt 7 dysku.

Schemat ścieżek płyty głównej:

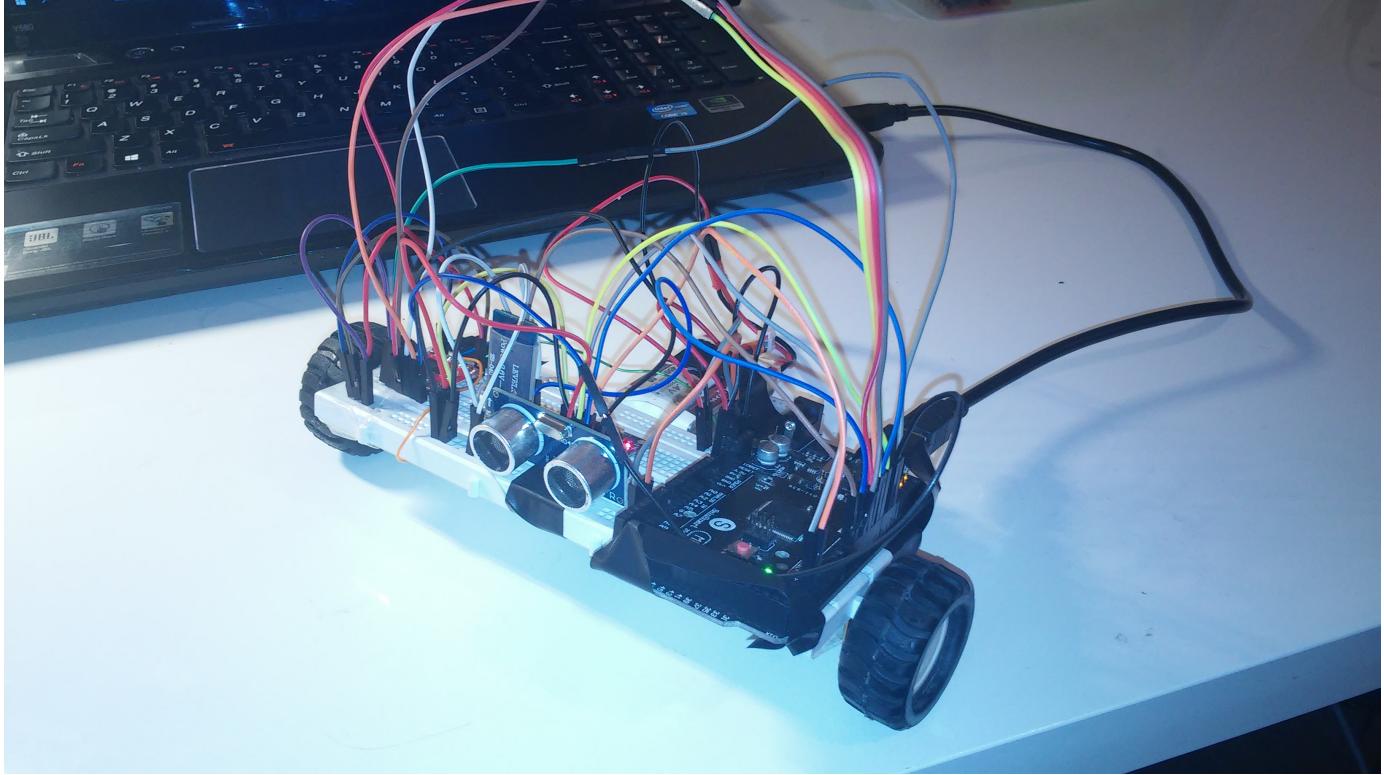


Schemat ścieżek płytka transceptorów:

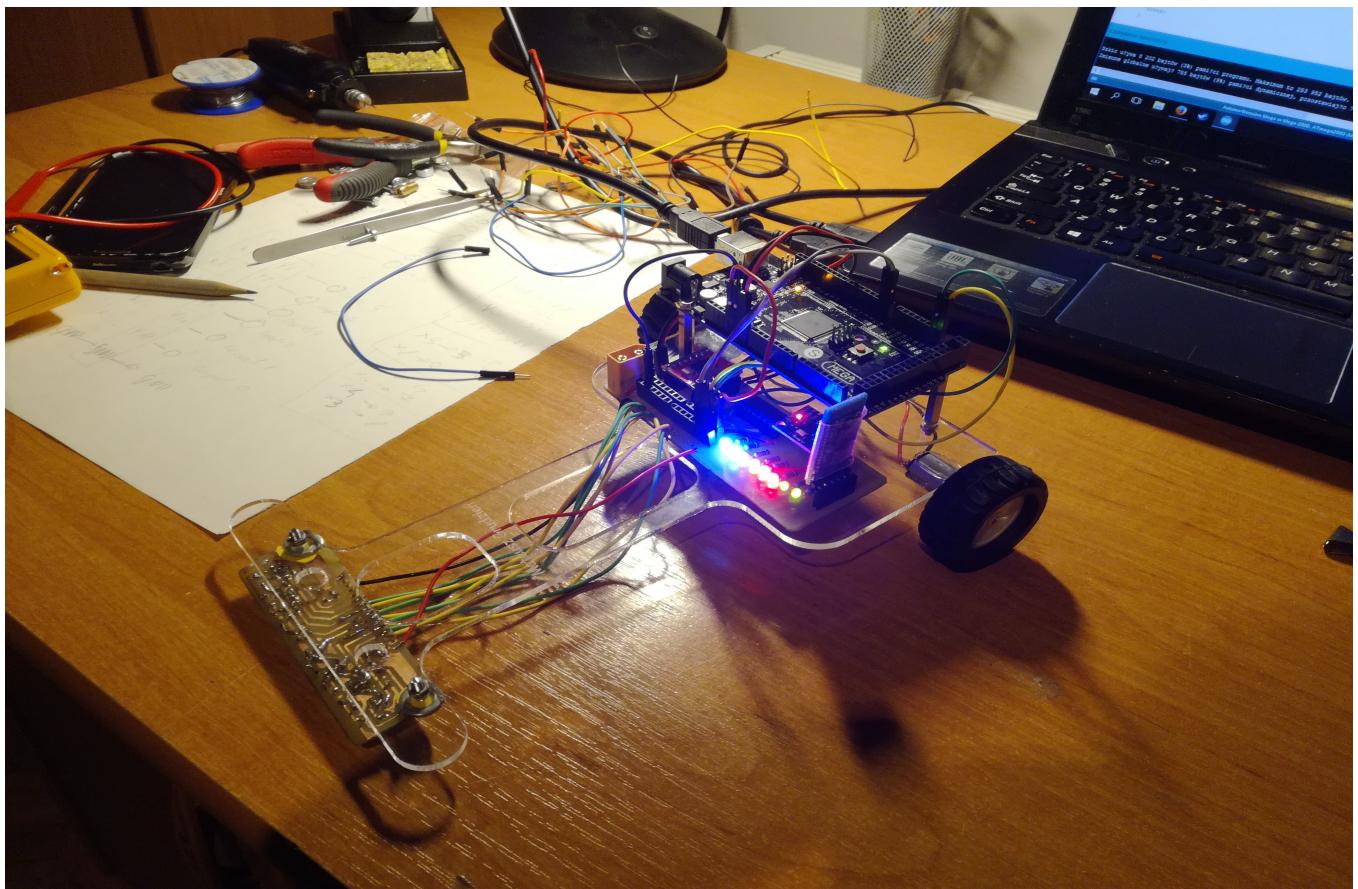


5. Wykonanie

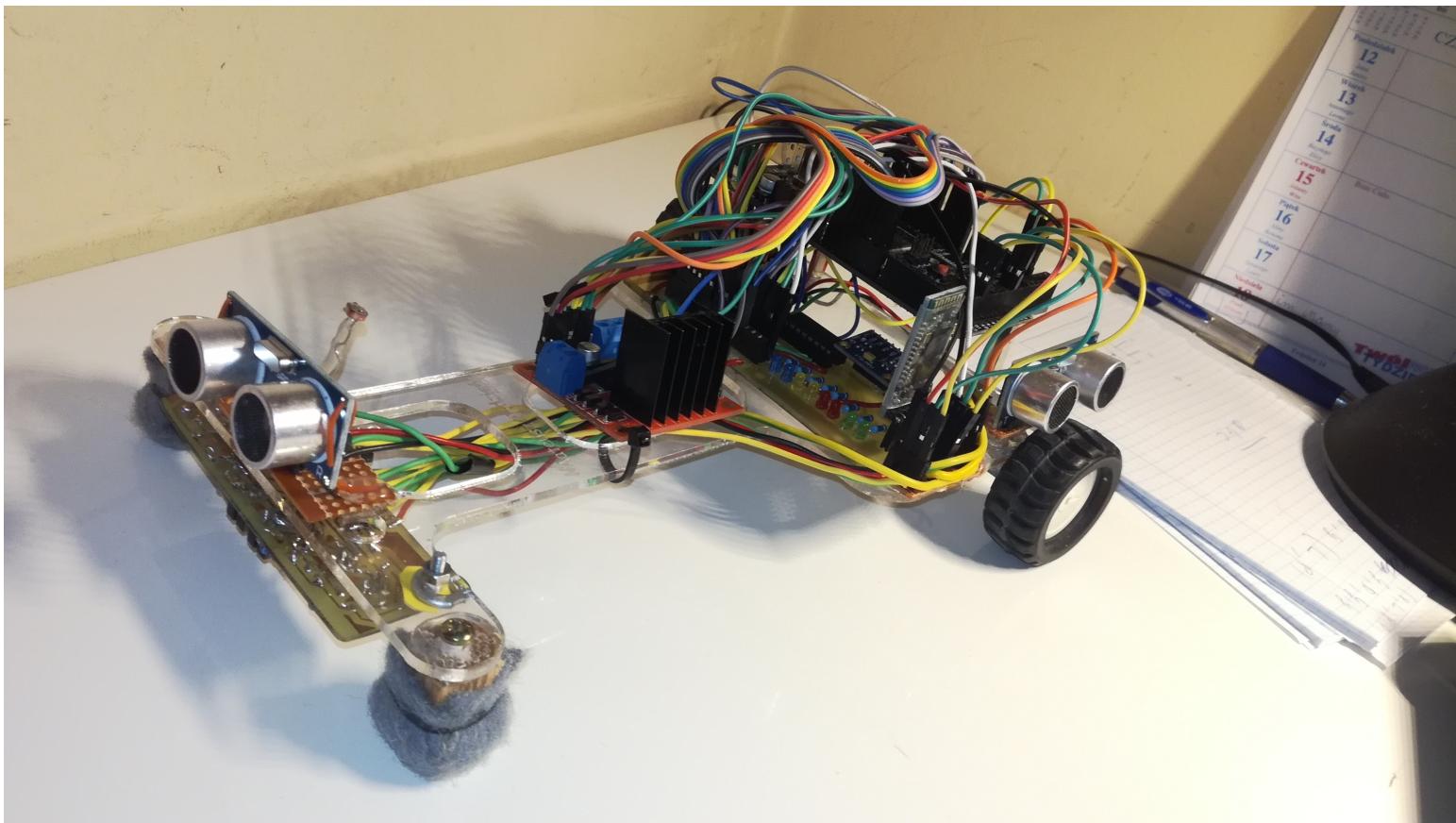
Pierwszy prototyp:



Wersja początkowa:



Wersja ostateczna:



Więcej zdjęć z wykonania znajduje się na dysku.

6. Kod programu sterującego

Większość kodu wraz z komentarzami kolorem zielonym odnośnie kluczowych miejsc programu. Całość znajduje się na dysku internetowym.

//-----

```
#include <Wire.h>          //Biblioteka do obsługi komunikacji przez I2C
#include <EEPROM.h>         //Biblioteka do obsługi pamięci nieulotnej EEPROM
#include <PCF8574.h>        //Biblioteka do obsługi ekspandera pinów cyfrowych
#include <MPU6050.h>         //Biblioteka do obsługi żyroskopu
```

```
#define baud_rate 9600
#define expander_adress 0x20
```

```
//Definicje pinów wykorzystywanych w programie
#define R_PWM 2                  //prawy PWM
#define L_PWM 3                  //lewym PWM
#define RR 5                     //prawy silnik do tyłu
```

```

#define RF 6           //prawy silnik do przodu
#define LF 8           //lewy silnik do przodu
#define LR 9           //lewy silnik do tyłu
#define sensor_transistor 11 //tranzystor włączający płytę z transoptoram
#define buzzer 12        //buzzer
#define ultrasonic_front_trig 51 //przedni czujnik odległości
#define ultrasonic_front_echo 50
#define ultrasonic_left_trig 38 //lewego czujnika odległości
#define ultrasonic_left_echo 39
#define ultrasonic_right_trig 34 //prawego czujnika odległości
#define ultrasonic_right_echo 35
#define optoresistor 31      //fotorezystor
//Definicja czasów kolejnych obrotów przy omijaniu przeszkód bez użycia żyroskopu
#define time_first 500
#define time_second 500
#define time_third 550
#define time_fourth 700
#define time_fifth 400
#define time_1 550
#define time_2 750

//Zmienne do obsługi silników wraz z opisem miejsc w pamięci EEPROM
short kat_1 = 70;          // EEPROM 4
short kat_2 = 70;          // EEPROM 5
short kat_3 = 40;          // EEPROM 6

short lewy = 200;          // EEPROM 1
short prawy = 200;          // EEPROM 2
float mnoznik_zawracania = 100; // EEPROM 3

short lewo_zwrot = 180;
short prawo_zwrot = 180;

//Odległość od której robot wykrywa przeszkodę na trasie
short odleglosc = 10;      // EEPROM 7

//Definicja uchwytów do ekspandera i żyroskopu
PCF8574 expander;
MPU6050 mpu;

//-----
//Funkcja konfiguracyjna
void setup()
{
    //Rozpoczęcie transmisji szeregowej po UART
    Serial.begin(baud_rate);
    Serial.println("Checking for errors...");

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

```

```
//Pierwsze miejsce w pamięci EEPROM jest bajtem kontrolnym decyduje o tym czy są już zapisane poprawne dane. Po jego wyzerowaniu użytkownik będzie poproszony o wprowadzenie nowych parametrów a poprzednie zostaną nadpisane
```

```
//EEPROM.write(0, 100); //Do resetowania
```

```
//Wczytywanie danych od użytkownika jeśli jest to pierwsze uruchomienie lub reset ustawień podano tylko kilka przykładowych
```

```
if(EEPROM.read(0) != 99)
```

```
{
```

```
    int tmp = 300;
```

```
    Serial.println("Podaj predkosc silnika lewego (0 - 255): ");
```

```
    do
```

```
{
```

```
    tmp = wczytaj_liczbe();
```

```
    delay(50);
```

```
}while(!(tmp <= 255 && tmp >= 1));
```

```
    EEPROM.write(1, tmp);
```

```
    tmp = 300;
```

```
    Serial.println("Podaj predkosc silnika prawego (0 - 255): ");
```

```
    do
```

```
{
```

```
    tmp = wczytaj_liczbe();
```

```
    delay(50);
```

```
}while(!(tmp <= 255 && tmp >= 1));
```

```
    EEPROM.write(2, tmp);
```

```
    tmp = 300;
```

```
    Serial.println("Podaj odleglosc wyzwalania przeszkody (3 - 50): ");
```

```
    do
```

```
{
```

```
    tmp = wczytaj_liczbe();
```

```
    delay(50);
```

```
}while(!(tmp <= 50 && tmp >= 1));
```

```
    EEPROM.write(7, tmp);
```

```
//Zapisanie informacji iż w pamięci są już poprawne dane
```

```
EEPROM.write(0, 99);
```

```
}
```

```
//Odczyt wszystkich parametrów z pamięci EEPROM
```

```
lewy = EEPROM.read(1);
```

```
prawy = EEPROM.read(2);
```

```
mnoznik_zawracania = EEPROM.read(3);
```

```
kat_1 = EEPROM.read(4);
```

```
kat_2 = EEPROM.read(5);
```

```
kat_3 = EEPROM.read(6);
```

```
odleglosc = EEPROM.read(7);
```

```

//Wypisanie odczytanych parametrów
Serial.print("Predkosc lewy: ");
Serial.println(lewy);
Serial.print("Predkosc prawy: ");
Serial.println(prawy);

mnoznik_zawracania = mnoznik_zawracania / 100;
Serial.print("Mnoznik: ");
Serial.println(mnoznik_zawracania);

lewo_zwrot = mnoznik_zawracania*lewy;
prawo_zwrot = mnoznik_zawracania*prawy;

Serial.print("Zwrot lewo: ");
Serial.println(lewo_zwrot);

Serial.print("Zwrot prawo: ");
Serial.println(prawo_zwrot);

Serial.print("Kat 1: ");
Serial.println(kat_1);

Serial.print("Kat 2: ");
Serial.println(kat_2);

Serial.print("Kat 3: ");
Serial.println(kat_3);

Serial.print("Odleglosc wyzwalania przeszkody: ");
Serial.println(odlegosc);

///////////////////////////////
//Zainicjowanie komunikacji z ekspanderem po I2C
Wire.beginTransmission(expander_adress);
byte error = Wire.endTransmission();

//Sprawdzenie poprawności połączenia
if (error = 2)
{
    expander.begin(expander_adress);
    Serial.print("Connected to \"PCF8574\" at address 0x");
    Serial.print(expander_adress, HEX);
    Serial.println(" via I2C");
}
else
{
    Serial.print("Unable to connect to \"PCF8574\" at adress 0x");
    Serial.print(expander_adress, HEX);
    Serial.println(", check connection!");
}

```

```
//Definicja pinów jako wyjście oraz ustawienie stanu wysokiego, co powoduje iż diody nie będą się
 świecić ze względu na ich odwrotną polaryzację w stosunku do typowego łączenia. Jest to
 spowodowane konstrukcją ekspandera który przy standardowej polaryzacji tej wspólny biegun
 ujemny diod oraz dodatnie biegony sterujące osiąga jedynie kilkadziesiąt mA wydajności
 prądowej , natomiast przy zastosowanej polaryzacji wynosi ona kilkaset mA
expander.pinMode(0, OUTPUT);
expander.pinMode(1, OUTPUT);
expander.pinMode(2, OUTPUT);
expander.pinMode(3, OUTPUT);
expander.pinMode(4, OUTPUT);
expander.pinMode(5, OUTPUT);
expander.pinMode(6, OUTPUT);
expander.pinMode(7, OUTPUT);

expander.digitalWrite(0, HIGH);
expander.digitalWrite(1, HIGH);
expander.digitalWrite(2, HIGH);
expander.digitalWrite(3, HIGH);
expander.digitalWrite(4, HIGH);
expander.digitalWrite(5, HIGH);
expander.digitalWrite(6, HIGH);
expander.digitalWrite(7, HIGH);

////////////////////////////////////////////////////////////////////////
```

```
//Próba nawiązania połączenia z modułem żyroskopu po I2C
if(!mpu.begin(MPU6050_SCALE_2000DPS, MPU6050_RANGE_2G))
{
    Serial.println("Unable to connect to \"MPU6050\" at address 0x61, check connection!");
    delay(500);
}
else
{
    Serial.println("Connected to MPU6050 at address 0x61 via I2C");
    // Kalibracja żyroskopu
    mpu.calibrateGyro();
    // Ustawienie czułości
    mpu.setThreshold(3);
}
```

```
////////////////////////////////////////////////////////////////////////
```

```
//Definicja, oraz ustawienie stanów pinów bezpośrednio na płytce Arduino
pinMode(L_PWM, OUTPUT);
pinMode(R_PWM, OUTPUT);
pinMode(RF, OUTPUT);
pinMode(RR, OUTPUT);
pinMode(LF, OUTPUT);
pinMode(LR, OUTPUT);

pinMode(sensor_transistor, OUTPUT);
```

```
pinMode(buzzer, OUTPUT);

pinMode(ultrasonic_front_trig, OUTPUT);
pinMode(ultrasonic_front_echo, INPUT);
pinMode(ultrasonic_left_trig, OUTPUT);
pinMode(ultrasonic_left_echo, INPUT);
pinMode(ultrasonic_right_trig, OUTPUT);
pinMode(ultrasonic_right_echo, INPUT);

digitalWrite(L_PWM, LOW);
digitalWrite(R_PWM, LOW);
digitalWrite(RF, LOW);
digitalWrite(RR, LOW);
digitalWrite(LF, LOW);
digitalWrite(LR, LOW);

digitalWrite(sensor_transistor, LOW);
digitalWrite(buzzer, LOW);

digitalWrite(ultrasonic_front_trig, LOW);
digitalWrite(ultrasonic_front_echo, LOW);
digitalWrite(ultrasonic_left_trig, LOW);
digitalWrite(ultrasonic_left_echo, LOW);
digitalWrite(ultrasonic_right_trig, LOW);
digitalWrite(ultrasonic_right_echo, LOW);
```

```
//////////
```

//Sygnalizacja użytkownikowi końca konfiguracji

```
for(int i = 0 ; i < 4 ; i++)
{
    expander.digitalWrite(i, LOW);
    delay(250);
}

for(int i = 7 ; i > 3 ; i--)
{
    expander.digitalWrite(i, LOW);
    delay(250);
}

delay(500);
tone(buzzer, 5000, 100);
delay(150);
tone(buzzer, 5000, 100);
delay(150);
tone(buzzer, 5000, 100);
delay(500);
expander.set();

Serial.println("READY");
}
```

```

//Główna pętla sterująca
void loop()
{
    short tmp = 10;
    //Wypisanie menu dla użytkownika
    Serial.println("[1] Start LF");
    Serial.println("[2] Start LFE");
    Serial.println("[3] Start LFE_G");
    Serial.println("[4] Freeride");
    Serial.println("[5] Read MPU6050 <--- change baud rate to 115200 before");
    Serial.println("[6] Blink PCF8574");
    Serial.println("[7] Check opto sensors");
    Serial.println("[8] Line scanner");
    Serial.println("[9] Settings");
    Serial.println("Input 0 to return to this menu");
    //Wczytanie odpowiedzi użytkownika
    do
    {
        tmp = Serial.read();
        delay(50);
    }while(tmp != '1' && tmp != '2' && tmp != '3' && tmp != '4' && tmp != '5' && tmp != '6' &&
tmp != '7' && tmp != '8' && tmp != '9');

    //Zainicjowanie odpowiedniej funkcji w zależności od wyboru użytkownika
    switch(tmp)
    {
        case '1':
        {
            //Jazda w trybie tylko odczytu z transoptorów
            LFS();

            break;
        }

        case '2':
        {
            //Jazda w trybie z wykorzystaniem również innych czujników, ale bez żyroskopu
            LFE();

            break;
        }

        case '3':
        {
            //Jazda w trybie z wykorzystaniem innych czujników w tym żyroskopu
            LFE_G();

            break;
        }

        case '4':
        {

```

```
//Jazda w trybie sterownia przez UART
freeride();

    break;
}

case '5':
{
    //Sprawdzenie odczytów z żyroskopu
    read_MP6050();

    break;
}

case '6':
{
    //Sprawdzenie działania diod LED
    blink_expander();

    break;
}

case '7':
{
    //Odczyt z transceptorów do UART
    check_sensors();

    break;
}

case '8':
{
    //Odczyt z transceptorów do LED
    line_scanner();

    break;
}

case '9':
{
    //Zmiana ustawień i parametrów
    settings();

    break;
}
}

delay(100);
}
```

//-----

```
//Funkcja realizująca jazdę w trybie standardowym – bez omijania przeszkód, kod wygląda tak samo jak w funkcji LFE_G() tylko bez odczytu z żyroskopu i detekcji przeszkody – linijki 384 do 553 w kodzie na dysku
void LFS(){}
//-----
//Funkcja realizująca jazdę w trybie rozszerzonym o detekcję przeszkód, ale bez odczytu z żyroskopu, reszta tak jak w funkcji LFE_G() – linijki 557 do 757 w kodzie na dysku
void LFE(){}
//-----
//Główna funkcja sterująca pojazdem w trybie jazdy po linii z detekcją przeszkód oraz ich omijania z wykorzystaniem wskazań żyroskopu
void LFE_G()
{
    //Sygnalizacja startu dla użytkownika
    Serial.println("Starting...");

    //Wyzerowanie używanych zmiennych
    short counter = 0;
    char tmp = '1';
    bool czujniki[8] = {0,0,0,0,0,0,0,0};
    short last = 1; // 0 - lewo, 1 - srodek, 2 - prawo

    //Włączenie płytki z transoptorami
    digitalWrite(sensor_transistor, HIGH); //Włączenie płytki z czujnikami

    //Sygnalizacja startu dla użytkownika
    delay(1000);
    tone(buzzer, 5000, 500);
    delay(600);
    tone(buzzer, 5000, 500);
    delay(600);
    tone(buzzer, 5000, 500);
    delay(500);
    Serial.println("READY!");

    do
    {
        //Sprawdzenie komendy od użytkownika
        tmp = Serial.read();
        counter++;

        //Odczyt danych z transoptorów
        for(int i = 0 ; i < 8 ; i++)
        {
            int level = analogRead(i);
            //Porównanie odczytu z ustalonym poziomem logicznej jedynki
            if(level >700)
            {

```

```

        czujniki[i] = 1;
    }
else
{
    czujniki[i] = 0;
}
}

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



//Jeśli co najmniej 7 na 8 czujników wykryło linię oznacza to, iż robot przejeżdża przez
//skrzyżowanie i należy jechać dalej prosto
if(czujniki[0]+czujniki[1]+czujniki[2]+czujniki[3]+czujniki[4]+czujniki[5]+czujniki[6]+czujniki[7]
>= 7)
{
    wpisz_silniki(lewy,250,1,0,1,0);
    continue;
}

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



//Co 3 krok sprawdzana jest droga przed pojazdem
if(counter > 2)
{
    //Zerowanie licznika
    counter = 0;
    //Odczyt z czujnika ultradźwiękowego
    int distance = read_ultrasonic(ultrasonic_front_trig, ultrasonic_front_echo);

    //Porównanie odczytanej odległości z ustaloną minimalną bezpieczną
    if(distance > 0 && distance < odleglosc)
    {
        //Zainicjowanie funkcji omijającej przeszkodę
        avoid_obstacle_w_g();

        //Ustawienie zmiennej opisującej ostatnie położenie linii na położenie ani prawe ani lewe –
        //wtedy robot będzie jechał prosto aż napotka trasę
        last = 1;
        //Nowy odczyt do dalszego przetwarzania
        for(int i = 0 ; i < 8 ; i++)
        {
            czujniki[i] = 0;
        }
    }
}

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

```

```
//Wypisanie aktualnego stanu logicznego transceptorów odczytanego przez program na diody LED
expander.set();
if(czujniki[0] == 1)
{
    expander.digitalWrite(4, LOW);
}
if(czujniki[1] == 1)
{
    expander.digitalWrite(5, LOW);
}
if(czujniki[2] == 1)
{
    expander.digitalWrite(6, LOW);
}
if(czujniki[3] == 1)
{
    expander.digitalWrite(7, LOW);
}
if(czujniki[4] == 1)
{
    expander.digitalWrite(3, LOW);
}
if(czujniki[5] == 1)
{
    expander.digitalWrite(2, LOW);
}
if(czujniki[6] == 1)
{
    expander.digitalWrite(1, LOW);
}
if(czujniki[7] == 1)
{
    expander.digitalWrite(0, LOW);
}
```

//////////

```
//Wyłączenie silników
wpisz_silniki(0, 0, 0, 0, 0, 0);
```

```
//Główna decyzja o wyjściu na silniki. Czujniki są ponumerowane od 0 do 7 od lewej do prawej.
Priorytet przy decyzji mają czujniki najbardziej zewnętrzne, jeśli linia zostanie tam wykryta jeden z
silników zostanie zatrzymany.
```

```
if(czujniki[7])
{
    wpisz_silniki(0, prawy, 1, 0, 1, 0);
}
else
{
    if(czujniki[0] )
    {
```

```
wpisz_silniki(lewy, 0, 1, 0, 1, 0);
}
else
{
    if(czujniki[3] || czujniki[4])
    {
        wpisz_silniki(lewy, prawy, 1, 0, 1, 0);
    }
    else
    {
        if(czujniki[2] || czujniki[1])
        {
            wpisz_silniki(lewy, prawo_zwrot, 1, 0, 1, 0);
        }
        else
        {
            if(czujniki[5] || czujniki[6])
            {
                wpisz_silniki(lewo_zwrot, prawy, 1, 0, 1, 0);
            }
            else
            {
                wpisz_silniki(0,0,0,0,0,0);
            }
        }
    }
}
```

```
//Blok opisujący zachowanie się robota w przypadku nie wykrycia linii. Brany jest tu pod uwagę parametr last który opisuje gdzie ostatnio widziana była linia
if(!(czujniki[0]|| czujniki[1] || czujniki[2]|| czujniki[3] || czujniki[4]|| czujniki[5] || czujniki[6]|| czujniki[7]))
{
    //Ostatnio z lewej - obrót wokół własnej osi w lewo
    if(last == 0)
    {
        wpisz_silniki(lewo_zwrot, prawo_zwrot, 0, 1, 1, 0);
    }
    //Nie przekroczone czujników na brzegu – czyli robot pokonuje przerwę w trasie, zachowana jest jazda w kierunku na wprost
    if(last == 1)
    {
        wpisz_silniki(lewy,prawy,1,0,1,0);
    }
    //Ostatnio z prawej - obrót wokół własnej osi w prawo
    if(last == 2)
    {
        wpisz_silniki(lewo_zwrot, prawo_zwrot, 1, 0, 0, 1);
    }
}
```

```
}
```

```
//////////
```

```
//Blok zapisujący położenie linii w razie gdyby następny odczyt nie wskazał jej położenia
if(czujniki[7])
{
    last = 0;
}
else
{
    if(czujniki[0] )
    {
        last = 2;
    }
    else
    {
        if(czujniki[5] || czujniki[6])
        {
            last = 0;
        }
        else
        {
            if(czujniki[2] || czujniki[1])
            {
                last = 2;
            }
            else
            {
                if(czujniki[3] || czujniki[4])
                {
                    last = 1;
                }
            }
        }
    }
}
```

```
//////////
```

```
delay(15);
}while(tmp != '0');
```

```
//Wyłączenie silników
wpisz_silniki(0, 0, 0, 0, 0, 0);
//Odcięcie zasilania od płytki z transitorami
digitalWrite(sensor_transistor, LOW);
}
```

```
//-----
```

```
//Funkcja realizująca tryb jazdy przy sterowaniu ręcznym przez UART
void freeride()
{
//Sygnalizacja startu dla użytkownika
Serial.println("Starting...");
char tmp = '1';
delay(1000);
tone(buzzer, 5000, 500);
delay(600);
tone(buzzer, 5000, 500);
delay(600);
tone(buzzer, 5000, 500);
delay(500);
Serial.println("READY!");

do
{
//Wyzerowanie stanu silników
wpisz_silniki(0, 0, 0, 0, 0, 0);
//Odczyt polecenia użytkownika
if(Serial.available() > 0)
{
char data = Serial.read();

//Decyzja o kierunku jazdy
switch(data)
{
case 'w':
{
wpisz_silniki(lewy, prawy, 1, 0, 1, 0);

break;
}

case 's':
{
wpisz_silniki(lewy, prawy, 0, 1, 0, 1);

break;
}

case 'a':
{
wpisz_silniki(lewo_zwrot, prawo_zwrot, 0, 1, 1, 0);

break;
}

case 'd':
{
wpisz_silniki(lewo_zwrot, prawo_zwrot, 1, 0, 0, 1);
}
```

```

        break;
    }

    case '0':
    {
        tmp = 0;

        break;
    }
    //Odczekanie ustalonego okresu wysyłki danych przez urządzenie sterujące
    delay(50);
}

}while(tmp != '0');

//Zatrzymanie silników
wpisz_silniki(0, 0, 0, 0, 0, 0);
}

//-----
//Funkcja realizująca odczyt i wypisanie danych o obrocie z modułu MPU 6050
void read_MP6050()
{
    //Zasygnalizowanie startu
    Serial.println("Starting...");
    bool tmp = 1;

    //Zerowanie zmiennych do odczytu
    unsigned long timer = 0;
    //Doświadczalnie dobrany krok odczytu
    float timeStep = 0.01;

    float pitch = 0;
    float roll = 0;
    float yaw = 0;

    //Kalibracja i odczekanie na ustanie błędnych odczytów
    mpu.calibrateGyro();
    delay(3000);

    do
    {
        //Sprawdzenie poleceń użytkownika
        tmp = Serial.read();

        //Odczytanie czasu pracy mikrokontrolera
        timer = millis();

        Vector norm = mpu.readNormalizeGyro();

```

```

//Obliczenie nowego odczytu kątów
pitch = pitch + norm.YAxis * timeStep;
roll = roll + norm.XAxis * timeStep;
yaw = yaw + norm.ZAxis * timeStep;

//Wypisanie kątów dla użytkownika
Serial.print(" Pitch = ");
Serial.print(pitch);
Serial.print(" Roll = ");
Serial.print(roll);
Serial.print(" Yaw = ");
Serial.println(yaw);

delay((timeStep*1000) - (millis() - timer));

}while(tmp != '0');
}

//-----
//Funkcja realizująca sprawdzenie działania diod LED linijki od 1073 do 1092 w kodzie z dysku
void blink_expander() {}

//-----
//Funkcja realizująca odczyt z transoptorów i wypisującą ich aktualny stan poprzez UART np. do
//konsoli w telefonie
void check_sensors()
{
    //Zerowanie odczytów czujników
    bool czujniki[8] = {0,0,0,0,0,0,0,0};

    //Zasignalizowanie użytkownikowi startu
    Serial.println("Starting...");

    //Przygotowanie zmiennej do odczytu sygnału od użytkownika
    char tmp = '1';

    //Włączenie zasilania płytki czujników
    digitalWrite(sensor_transistor, HIGH);
    Serial.println("READY!");

    do
    {
        //Sprawdzenie czy nie została wysłana komenda zakończenia pracy
        tmp = Serial.read();
        //Odczyt stanu transoptorów
        for(int i = 0 ; i < 8 ; i++)
        {
            int level = analogRead(i);
            if(level >700)
            {

```

```

        czujniki[i] = 1;
    }
else
{
    czujniki[i] = 0;
}
}

for(int i = 0 ; i < 8 ; i++)
{
    Serial.print(czujniki[i]);
    Serial.print(" ");
}
Serial.println(" ");
delay(250);

}while(tmp != '0');

//Odcięcie zasilania płytki transoptorów
digitalWrite(sensor_transistor, LOW);
}

//-----
//Funkcja realizująca skanowanie linii czyli test poprawności funkcjonowania transoptorów, na
robocie znajduje się 8 diod LED reprezentujących stan 8 czujników, można dzięki temu w prosty
sposób stwierdzić czy w danym miejscu trasy są problemy z odczytem
void line_scanner()
{
    //Zerowanie odczytów czujników
    bool czujniki[8] = {0,0,0,0,0,0,0,0};

    //Zasignalizowanie użytkownikowi startu
    Serial.println("Starting...");
    //Przygotowanie zmiennej do odczytu sygnału od użytkownika
    char tmp = '1';

    //Włączenie zasilania płytki czujników
    digitalWrite(sensor_transistor, HIGH);
    Serial.println("READY!");

    do
    {
        //Sprawdzenie czy nie została wysłana komenda zakończenia pracy
        tmp = Serial.read();
        //Odczyt stanu transoptorów
        for(int i = 0 ; i < 8 ; i++)
        {
            int level = analogRead(i);
            //Porównanie z doświadczalnie wyznaczonym poziomem wykrywania linii
            if(level >700)
            {

```

```

        czujniki[i] = 1;
    }
else
{
    czujniki[i] = 0;
}
}

//Wyświetlenie odpowiedniej kombinacji LED przy użyciu biblioteki ekspandera
if(czujniki[0] == 1)
{
    expander.digitalWrite(4, LOW);
}
if(czujniki[1] == 1)
{
    expander.digitalWrite(5, LOW);
}
if(czujniki[2] == 1)
{
    expander.digitalWrite(6, LOW);
}
if(czujniki[3] == 1)
{
    expander.digitalWrite(7, LOW);
}
if(czujniki[4] == 1)
{
    expander.digitalWrite(3, LOW);
}
if(czujniki[5] == 1)
{
    expander.digitalWrite(2, LOW);
}
if(czujniki[6] == 1)
{
    expander.digitalWrite(1, LOW);
}
if(czujniki[7] == 1)
{
    expander.digitalWrite(0, LOW);
}

delay(50);

//Wyłączenie wszystkich diod poprzez stan wysoki
expander.set();

}while(tmp != '0');

//Odcięcie zasilania płytki transoptorów
digitalWrite(sensor_transistor, LOW);
}

```

```
//-----  
  
//Funkcja obsługująca zmianę ustawień poprzez interfejs UART np. przy użyciu terminalu w  
telefonie i połączenia bluetooth  
void settings()  
{  
    bool wyjscie = 1;  
    bool czy = 1;  
    char tmp;  
  
    do  
    {  
        //Wypisanie opcji dla użytkownika  
        Serial.println("[1] Wypisz aktualne");  
        Serial.println("[2] Zmien predkosc");  
        Serial.println("[3] Zmien kat");  
        Serial.println("[4] Zmien odleglosc wyzwalania przeskody");  
  
        czy = 1;  
        wyjscie = 1;  
  
        //Czekam na dane przychodzące z UART tak długo aż będą poprawne  
        do  
        {  
            if(Serial.available() > 0)  
            {  
                tmp = Serial.read();  
                if(tmp == '0' || tmp == '1' || tmp == '2' || tmp == '3' || tmp == '4')  
                    czy = 0;  
            }  
        }while(czy);  
  
        switch(tmp)  
        {  
            case '0':  
            {  
                wyjscie = 0;  
                break;  
            }  
            case '1':  
            {  
                {}//Wypisanie danych o aktualnych ustawieniach prędkości, kąta, odległości linijki 1240 do  
                // 1271 w kodzie na dysku  
                break;  
            }  
  
            //Wczytanie od użytkownika nowych parametrów silników  
            case '2':  
            {  
                //Zresetowanie bajtu kontrolnego  
                EEPROM.write(0, 100);  
            }  
        }  
    }  
}
```

```

if(EEPROM.read(0) != 99)
{
    //Pobranie danych od użytkownika i zapisanie ich do pamięci EEPROM
    int tmp = 300;
    Serial.println("Podaj predkosc silnika lewego (0 - 255): ");
    do
    {
        tmp = wczytaj_liczbe();
        delay(50);
    }while(!(tmp <= 255 && tmp >= 1));
    //Zapis
    EEPROM.write(1, tmp);

    tmp = 300;
    Serial.println("Podaj predkosc silnika prawego (0 - 255): ");
    do
    {
        tmp = wczytaj_liczbe();
        delay(50);
    }while(!(tmp <= 255 && tmp >= 1));
    //Zapis
    EEPROM.write(2, tmp);

    tmp = 300;
    Serial.println("Podaj mnoznik predkosci do zawracania (0 - 100): ");
    do
    {
        tmp = wczytaj_liczbe();
        delay(50);
    }while(!(tmp <= 100 && tmp >= 1));
    //Zapis
    EEPROM.write(3, tmp);

    //Zapis do bajtu kontrolnego informacji o wprowadzeniu danych
    EEPROM.write(0, 99);
}

//Odczyt nowych danych z pamięci nieulotnej EEPROM
lewy = EEPROM.read(1);
prawy = EEPROM.read(2);
mnoznik_zawracania = EEPROM.read(3);

//Wypisanie dla użytkownika potwierdzenia wprowadzonych danych
Serial.print("Predkosc lewy: ");
Serial.println(lewy);

Serial.print("Predkosc prawy: ");
Serial.println(prawy);

mnoznik_zawracania = mnoznik_zawracania / 100;
Serial.print("Mnoznik: ");

```

```

Serial.println(mnoznik_zawracania);

lewo_zwrot = mnoznik_zawracania*lewy;
prawo_zwrot = mnoznik_zawracania*prawy;

Serial.print("Zwrot lewo: ");
Serial.println(lewo_zwrot);

Serial.print("Zwrot prawo: ");
Serial.println(prawo_zwrot);

//Czekanie na akcję użytkownika
while(!(Serial.available()));

//Czyszczenie ewentualnych pozostałości w buforze
for(int i = 0 ; i < 50 ; i++)
    Serial.read();

break;
}

//Zapis nowych danych dla żyroskopu
case '3':
{
    //Reset bajtu kontrolnego
    EEPROM.write(0, 100);

    if(EEPROM.read(0) != 99)
    {
        //Wczytanie nowych danych od użytkownika i ich zapis do EEPROM
        int tmp = 300;
        Serial.println("Podaj kat 1 (1 - 180): ");
        do
        {
            tmp = wczytaj_liczbe();
            delay(50);
        }while(!(tmp <= 180 && tmp >= 1));
        //Zapis
        EEPROM.write(4, tmp);

        tmp = 300;
        Serial.println("Podaj kat 2 (1 - 180): ");
        do
        {
            tmp = wczytaj_liczbe();
            delay(50);
        }while(!(tmp <= 180 && tmp >= 1));
        //Zapis
        EEPROM.write(5, tmp);

        tmp = 300;
        Serial.println("Podaj kat 3 (1 - 180): ");
    }
}

```

```
do
{
    tmp = wczytaj_liczbe();
    delay(50);
}while(!(tmp <= 180 && tmp >= 1));
//Zapis
EEPROM.write(6, tmp);

//Zapis do bajtu kontrolnego informacji o wprowadzeniu danych
EEPROM.write(0, 99);
}

//Odczyt nowych parametrów z pamięci nieulotnej EEPROM
kat_1 = EEPROM.read(4);
kat_2 = EEPROM.read(5);
kat_3 = EEPROM.read(6);

//Wypisanie potwierdzenia wprowadzonych danych dla użytkownika
Serial.print("Kat 1: ");
Serial.println(kat_1);

Serial.print("Kat 2: ");
Serial.println(kat_2);

Serial.print("Kat 3: ");
Serial.println(kat_3);

//Czekanie na akcję użytkownika
while(!Serial.available()));

//Czyszczenie ewentualnych pozostałości w buforze
for(int i = 0 ; i < 50 ; i++)
    Serial.read();

break;
}

//Wprowadzenie nowej odległości wykrywania przeszkody
case '4':
{
    //Reset bajtu kontrolnego
    EEPROM.write(0, 100);

    if(EEPROM.read(0) != 99)
    {
        //Wczytanie nowych danych od użytkownika i ich zapis do EEPROM
        int tmp = 300;
        Serial.println("Podaj odleglosc wyzwalania przeszkody (3 - 50): ");
        do
        {
            tmp = wczytaj_liczbe();
            delay(50);
        }
        while(tmp < 3 || tmp > 50);
        EEPROM.write(0, tmp);
    }
}
```

```

}while(!(tmp <= 50 && tmp >= 1));
//Zapis
EEPROM.write(7, tmp);

//Zapis do bajtu kontrolnego informacji o wprowadzeniu danych
EEPROM.write(0, 99);
}

//Odczyt nowych parametrów z pamięci nieulotnej EEPROM
odleglosc = EEPROM.read(7);

//Wypisanie potwierdzenia wprowadzonych danych dla użytkownika
Serial.print("Odleglosc wyzwalania przeszkody: ");
Serial.println(odleglosc);

//Czekanie na akcję użytkownika
while(!(Serial.available()));

//Czyszczenie ewentualnych pozostałości w buforze
for(int i = 0 ; i < 50 ; i++)
    Serial.read();

    break;
}
}
}while(wyjscie);
}

//-----
//Funkcja wypisująca żądany stan pracy silników
void wpisz_silniki(short l_pwm, short r_pwm, bool lf, bool lr, bool rf, bool rr)
{
//Nadanie sygnałów PWM
analogWrite(L_PWM, l_pwm);
analogWrite(R_PWM, r_pwm);
//Informacja o kierunku pracy
digitalWrite(LF, lf);
digitalWrite(LR, lr);
digitalWrite(RF, rf);
digitalWrite(RR, rr);
}

//-----
//Funkcja odczytująca dane z wybranego czujnika ultradźwiękowego odległości, jako parametry
//podane zostały piny odpowiadające za nadanie sygnału i jego nasłuch
int read_ultrasonic(short trig, short echo)
{
int czas, dist;
//Nadanie krótkiego sygnału przez czujnik
digitalWrite(trig, HIGH);

```

```

delayMicroseconds(1000);
//Wyłączenie nadawania sygnału
digitalWrite(trig, LOW);
//Czekam na powrót fali do urządzenia i zapisuje czas jaki minął
czas = pulseIn(echo, HIGH);
//Doświadczalnie wyznaczony wzór na odległość
dist = (czas/2) / 29.1;

return dist;
}

//-----
//Funkcja omijająca przeszkodę bez wykorzystania żyroskopu, opiera się na doświadczalnym
ustalaniu czasu obrotu, co należy wykonywać przy każdej zmianie powierzchni po której robot
porusza się
void avoid_obstacle_w_o_g()
{
    //Obrót w prawo
    wpisz_silniki(lewo_zwrot, prawo_zwrot, 1, 0, 0, 1);
    delay(time_first);
    //Przejechanie ustalonej odległości
    wpisz_silniki(lewo_zwrot, prawo_zwrot, 1, 0, 1, 0);
    delay(time_second);
    //Obrót w lewo
    wpisz_silniki(lewo_zwrot, prawo_zwrot, 0, 1, 1, 0);
    delay(time_third);
    //Przejechanie ustalonej odległości
    wpisz_silniki(lewo_zwrot, prawo_zwrot, 1, 0, 1, 0);
    delay(time_fourth);
    //Obrót w lewo
    wpisz_silniki(lewo_zwrot, prawo_zwrot, 0, 1, 1, 0);
    delay(time_fiveth);
    //Powrót na trasę
    wpisz_silniki(lewo_zwrot, prawo_zwrot, 1, 0, 1, 0);

    //Zerowanie odczytu transceptorów
    bool czujniki[8] = {0,0,0,0,0,0,0,0};
    //Czekanie aż któryś z czujników zobaczy linię
    do
    {
        for(int i = 0 ; i < 8 ; i++)
        {
            int level = analogRead(i);
            if(level >700)
            {
                czujniki[i] = 1;
            }
            else
            {
                czujniki[i] = 0;
            }
        }
    }
}

```

```
    }
}while(!(czujniki[0]|| czujniki[1] || czujniki[2]|| czujniki[3] || czujniki[4]|| czujniki[5] || czujniki[6]|| czujniki[7]));
}
```

```
//-----
```

```
//Funkcja uruchamiająca sekwencję omijania przeszkody z użyciem żyroskopu, w tym czasie pomijane są dane z transoptorów, przeszkoda jest zawsze omijana z prawej strony
```

```
void avoid_obstacle_w_g()
```

```
{
```

```
    //Obrót w prawo
```

```
    wpisz_silniki(lewo_zwrot, prawo_zwrot, 1, 0, 0, 1);
```

```
    wait_X_degrees(1, kat_1);
```

```
    //Przejechanie ustalonej odległości
```

```
    wpisz_silniki(lewo_zwrot, prawo_zwrot, 1, 0, 1, 0);
```

```
    delay(time_1);
```

```
    //Obrót w lewo
```

```
    wpisz_silniki(lewo_zwrot, prawo_zwrot, 0, 1, 1, 0);
```

```
    wait_X_degrees(0, kat_2);
```

```
    //Przejechanie ustalonej odległości
```

```
    wpisz_silniki(lewo_zwrot, prawo_zwrot, 1, 0, 1, 0);
```

```
    delay(time_2);
```

```
    //Obrót w lewo
```

```
    wpisz_silniki(lewo_zwrot, prawo_zwrot, 0, 1, 1, 0);
```

```
    wait_X_degrees(0, kat_3);
```

```
    //Powrót na trasę
```

```
    wpisz_silniki(lewo_zwrot, prawo_zwrot, 1, 0, 1, 0);
```

```
//Zerowanie odczytu transoptorów
```

```
bool czujniki[8] = {0,0,0,0,0,0,0};
```

```
//Czekanie aż któryś z czujników zobaczy linię
```

```
do
```

```
{
```

```
    for(int i = 0 ; i < 8 ; i++)
```

```
{
```

```
        int level = analogRead(i);
```

```
        if(level >700)
```

```
{
```

```
            czujniki[i] = 1;
```

```
}
```

```
        else
```

```
{
```

```
            czujniki[i] = 0;
```

```
}
```

```
        delayMicroseconds(100);
```

```
}
```

```
}while(!(czujniki[0]|| czujniki[1] || czujniki[2]|| czujniki[3] || czujniki[4]|| czujniki[5] || czujniki[6]|| czujniki[7]));
```

```
}
```

```
//-----
```

```

//Funkcja oczekująca na obrót o ilość stopni podanych jako zmienna degree, w kierunku podanym
//jako zmienna typu bool (0 – lewo, 1 - prawo)
void wait_X_degrees(bool lr, int degree)
{
    //Ustalenie kroku czasowego odczytu – dobrane doświadczalnie
    float timeStep = 0.02; //ZMIENIĆ NA 0.05 PRZED WŁĄCZENIEM SERIAL PORTU
    //Wyzerowanie zmiennych do odczytu
    unsigned long timer = 0;
    float pitch = 0;
    float roll = 0;
    float yaw = 0;

    //Liczenie w prawo
    if(lr)
    {
        do
        {
            //Odczyt aktualnego czasu pracy procesora
            timer = millis();

            //Odczyt i wypisanie obrotu w osi Z
            Vector norm = mpu.readNormalizeGyro();
            yaw = yaw + norm.ZAxis * timeStep;

            Serial.println(yaw);

            delay((timeStep*1000) - (millis() - timer));

        }while(yaw > -degree);
    }
    //Liczenie w lewo
    else
    {
        do
        {
            //Odczyt aktualnego czasu pracy procesora
            timer = millis();

            //Odczyt i wypisanie obrotu w osi Z
            Vector norm = mpu.readNormalizeGyro();
            yaw = yaw + norm.ZAxis * timeStep;

            Serial.println(yaw);

            delay((timeStep*1000) - (millis() - timer));

        }while(yaw < degree);
    }
}

//Funkcja do wczytania liczby, linijki od 1633 do 1645 w kodzie na dysku
int wczytaj_liczbe() {

```

7. Linki do materiałów dodatkowych – zdjęcia, filmy, kod programu

Dodatkowe materiały znajdują się na dysku Google nie wymagającym logowania:

<https://drive.google.com/open?id=0Bwza1RiWruz9X2R2bVgxeVh2VzA>

8. Wnioski

Wykonany robot spełnia początkowe założenia projektu, oraz został pozytywnie oceniony podczas rekrutacji do Koła Naukowego Robotyków przy wydziale MEiL. Jest w stanie spełnić wymagania stawiane konstrukcjom na zawodach typu Line Follower Enhanced, podąża po linii omija przeszkodę na trasie z dużą niezawodnością, radzi sobie przy napotkaniu skrzyżowań, oraz przerw w trasie, jest również niewrażliwy na zmiany oświetlenia.

Platforma ze względu na zastosowanie tanich komponentów cechuje się niską prędkością przejazdów, jednak jest wystarczająca do nauki różnych metod sterowania tego typu robotami. Wykorzystanie żyroskopu uniezależniło użytkownika od konieczności zmiany parametrów opóźnień przy zmianie podłoża na którym porusza się robot, innym rozwiązaniem mogło być zastosowanie enkoderów z tarczami impulsowymi na każde z kół, co jednak nie daje stu procentowej pewności w przypadku utraty przyczepności.

W celu przygotowania platformy do profesjonalnego startu w zawodach należałyby wykonać płytę PCB będącą jednocześnie elementem mocującym komponenty elektroniczne jak i mechaniczne, co zwiększyłoby prędkość pojazdu, oraz zmniejszyło jego gabaryty. Mniejsza masa pozytywnie wpłynęłaby na bezwładność pojazdu, która powoduje jego nie zatrzymywanie się dokładnie w momencie gdy żyroskop wskaże odpowiedni kąt, co powoduje konieczność doświadczalnej korekcji kątów. Użyte transoptory cechują się bardzo dobrą niezawodnością wykrywania linii, dobrane parametry rezystorów umożliwiają powtarzalny pomiar zupełnie niezależny od zewnętrznych warunków oświetlenia spotykanych na zawodach z odległości około 17mm. Ich zasadniczą wadą jest rozmiar który będzie przeszkodą przy miniaturyzacji. Zastosowany algorytm sterujący cechuje się olbrzymią prostotą i nie sprawdzi się w przypadku szybszych konstrukcji, jednak w przypadku opisywanego wykonania spełnia w pełni swoje zadanie, nie zauważono żadnych oscylacji, ani problemów z utrzymywaniem toru jazdy. Zastosowany sposób możliwości zmiany danych poprzez np. komunikację z telefonem przez bluetooth znacznie usprawnia testy dzięki możliwości zmiany parametrów „w locie”.

Część z przyrządów zamontowanych na platformie nie została w pełni wykorzystana. Możliwe jest rozszerzenie funkcjonalności o wykrywanie wjazdu na pochylnie które bywają przeszkodami na niektórych zawodach, oraz istnieje możliwość wykorzystania bocznych czujników odległości do kontroli szerokości przeszkody podczas jej omijania.