Politechnika Wrocławska

Roboty Mobilne - Projekt

Robot klasy LineFollower - „ZACZ”
ETAP III

Wydział: W12N

Prowadzący: Dr inż. Michał Błędowski

Autorzy:

Kamil Winnicki

Oliwier Woźniak

Wrocław, Polska 22 czerwca 2024

Spis treści

1 Cel Projektu

Celem projektu jest zbudowanie robota mobilnego typu LineFollower. Rzeczony robot ma za zadanie w sposób autonomiczny, w jak nakrótszym czasie przejechać tor, wyznaczony za pomocą czarnej linii na białym tle.

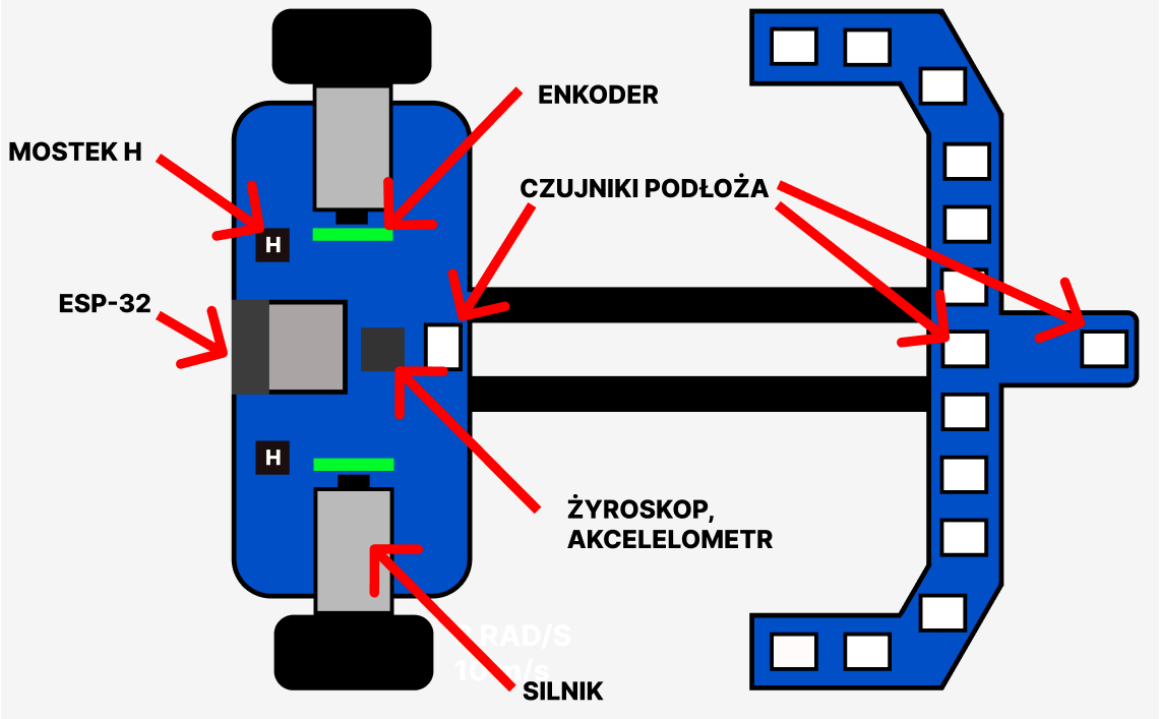
2 Założenia Projektowe

Robot będzię wyposażony w następujące elementy:

- mikrokontroler ESP32-S3,
- zasilanie oparte na akumulatorze Li-Pol 2S,
- dwa koła sterowane różnicowo,
- podwójny mostek H do sterowania silnikami,
- enkodery do pomiaru prędkości obrotowej i pozycji kół,
- żyroskop i akcelelometr do pomiaru obrotu i pozycji,
- 15 analogowych czujników odbiciowych do śledzenia lini pod robotem,
- komparatory do zmiany sygnału analogowego z czujników na cyfrowy,
- sterowanie oparte o regulator PID,

Układ elektroniczny podzielony będzie na dwie płytki PCB:

- płytke główną zawierającej mikrokontroler, żyroskop z akclelometrem, mostki H, enkodery oraz silniki,
- płytke z czujnikami podłoża,



Rysunek 1: Poglądowy rysunek robota

3 Podział pracy na etapy

Z powodu wielopoziomowości problemu jakim jest tworzenie robota typu LineFollower, praca została podzielona na działy zgodnie z poruszanymi dziedzinami nauki.

1. zaprojektowanie i złożenie układu elektronicznego,
2. zaprojektowanie i stworzenie mechaniki robota,
3. napisanie i implementacja programu sterującego.

Każda z tych dziedzin może być realizowany równolegle, jednak testy odbywać się będą dopiero po ukończenia poprzedniego punktu. W związku z powyższym pracę podzieliliśmy na następujące etapy, które zostaną zrealizowane do określonych terminów:

1. stworzenie projektów elektroniki, oraz mechaniki użytej w robocie – 25.04.2024
2. złożenie części fizycznej robota i testy działania – 24.05.2024
3. implementacja algorytmu sterującego na robocie i testy działania – 06.06.2024

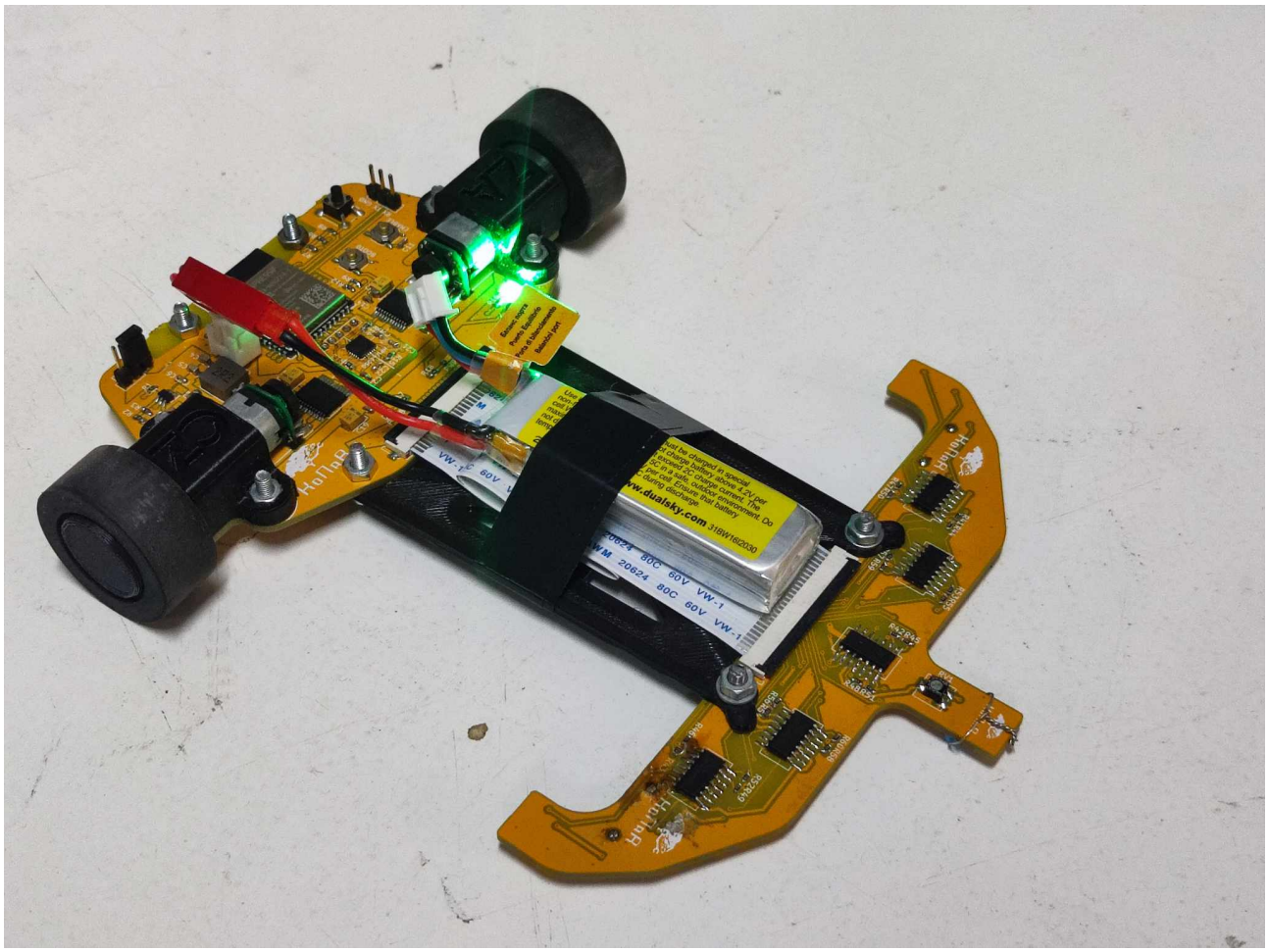
Weryfikacja poszczególnych etapów będzie przebiegała w następujący sposób:

- Etap 1: Schematy elektryczne, oraz projekty mechaniczne zostaną stworzone i dodane do końcowej dokumentacji. Jednoznacznym potwierdzeniem poprawnego wykonania układów będzie przyjęcie ich przez firmę wykonującą płytki PCB.
- Etap 2: Robot będzie w pełni funkcjonalny, będzie w stanie poruszać się na podstawie prostych algorytmów/poleceń zaimplementowanych na układzie sterującym. Jednoznaczną weryfikacją będzie przejechanie przez robota odległość 1 metra po płaskiej powierzchni, wzdłuż względnie prostej linii.
- Etap 3: Robot będzie spełniał cel projektu.

4 Podział pracy na członków

Kamil Winnicki	Oliwier Woźniak
Projekt mechaniczny	**schemat podłączenia żyroskopu i enkoderu**
schemat podłączenia mostków H	**schemat podłączenia enkoderów**
schemat podłączenia ESP-32	**schemat sekcji zasilania**
schemat elektroniczny całego układu	**podłączenie czujników i komparatorów**
projekt głównej płytki PCB	**projekt płytki PCB z czujnikami**
obsługa mostków H	**odczyt z czujników podłoża**
funkcja obsługi żyroskopu i akcelelometru	**funkcja obsługująca enkodery**
implementacja regulatora PID	**implementacja algorytmu sterującego**

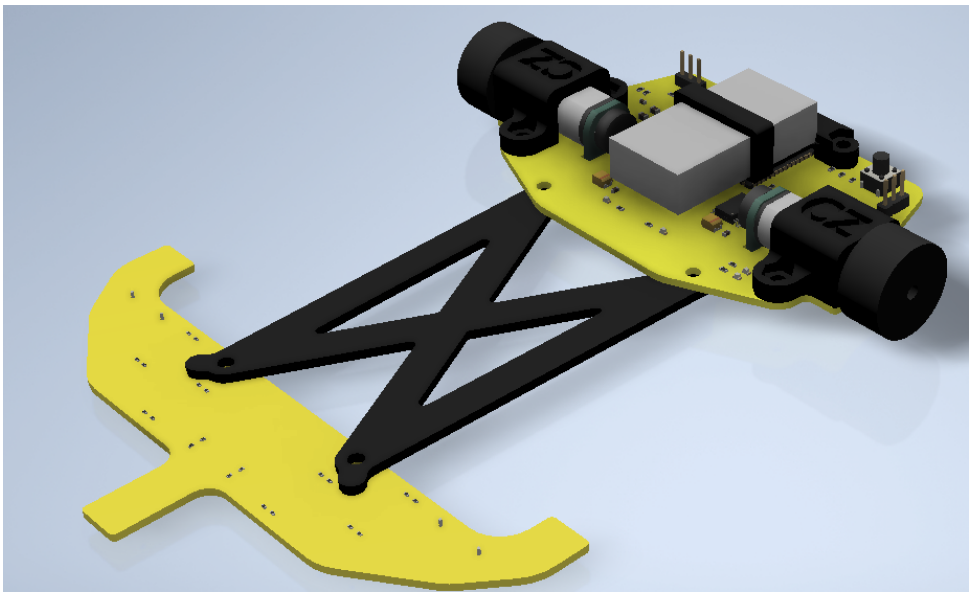
5 Zdjęcia robota



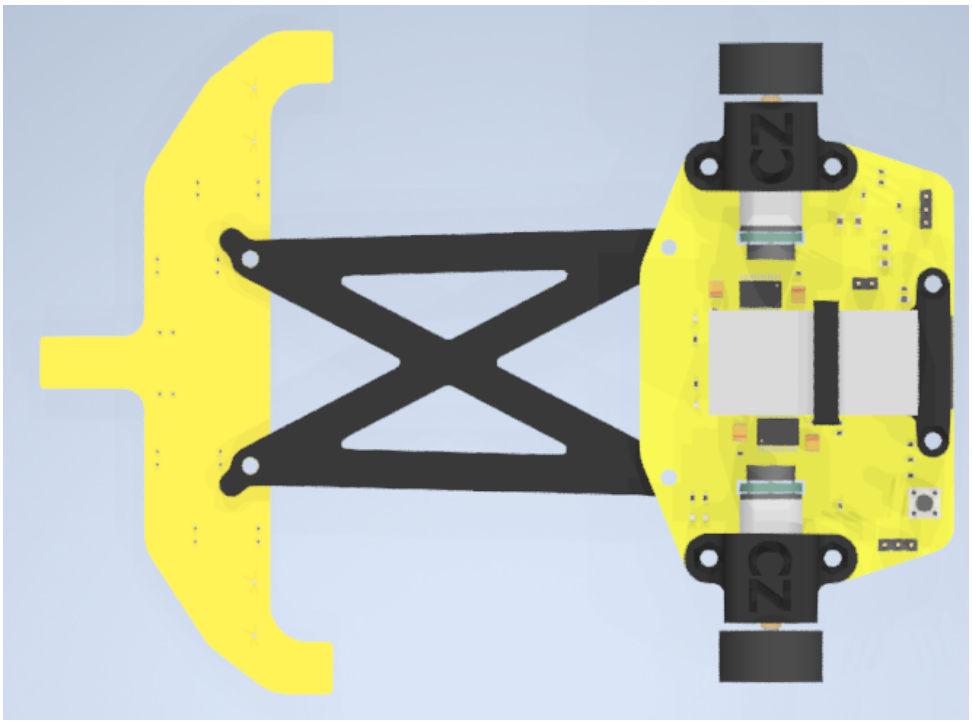
Rysunek 2: Wygląd robota

6 Mechanika

Od strony mechanicznej robot sklada się z dwóch plytek PCB, przedniej plytki czujnikami, i tylnej, glownej plytki do ktorej przymocowane sa silniki oraz bateria. Plytki miedzy sobą są połączone elementem z druku 3D.



Rysunek 3: Projekt mechaniczny



Rysunek 4: Rzut z góry

7 Elektronika

Tak jak zostało wspomniane wcześniej elektronika została rozdzielona na dwie płytki PCB połączone 26 pinową taśmą FFC:

7.1 Płytka główna

Na płytce głównej zostały umieszczone:

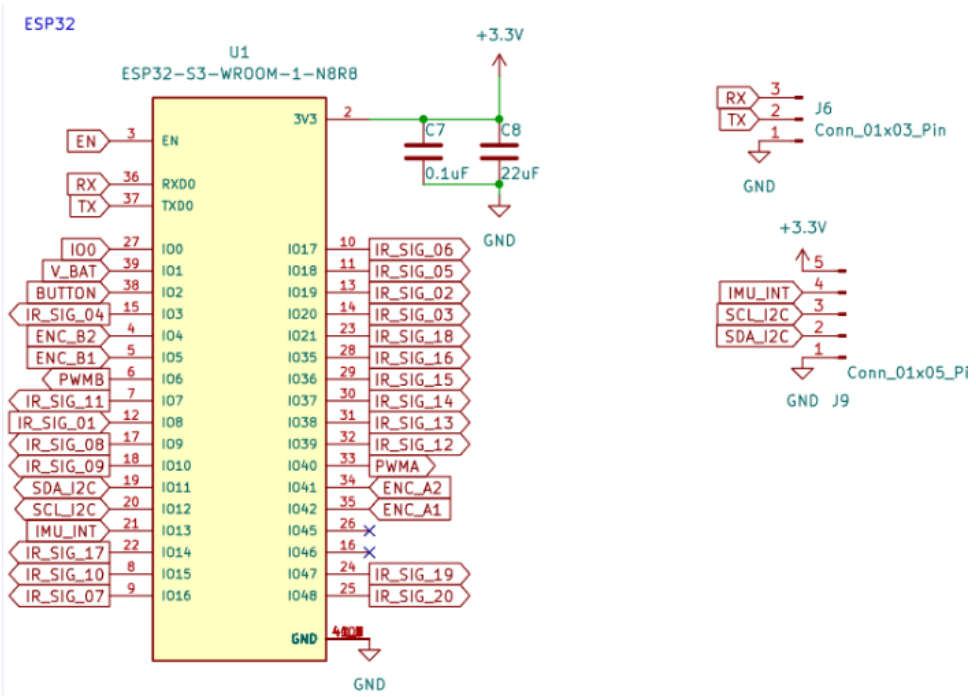
- Mikrokontroler ESP32-S3 z anteną wifi/Bluetooth, diody oraz przyciski
- 2 mostki H - TB6612FNG
- 2 enkodery
- sekcja zasilania oparta na przetwornicy STEP-DOWN
- miejsce na przylutowanie IMU, MPU6050

7.2 Płytka z czujnikami

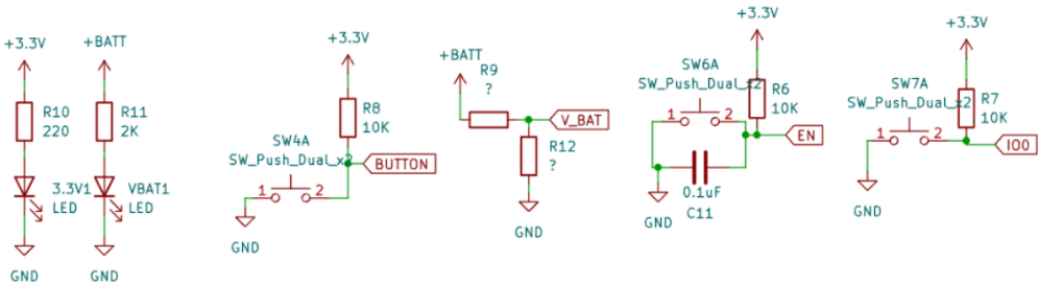
Na płytce z czujnikami zostały umieszczone:

- 20 czujników odbiciowych ITR
- 5 cztero-kanałowych komparatorów
- 2 rezystory THT służące jako ślizgacz

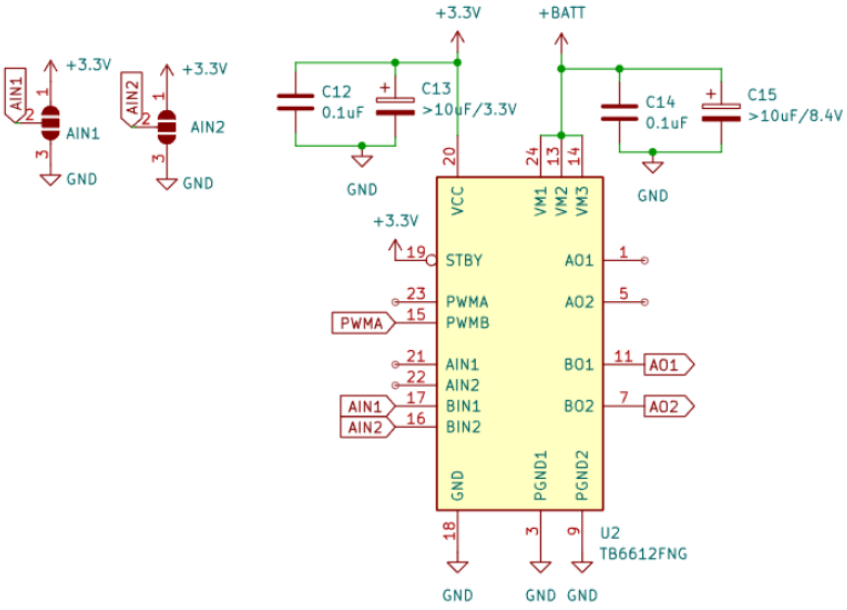
7.3 Schematy



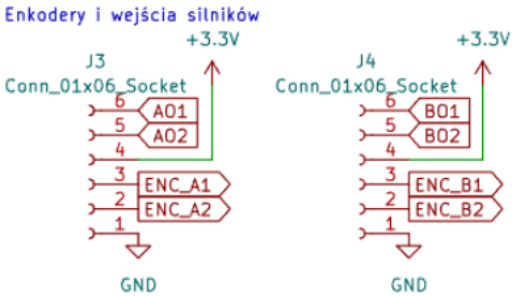
Rysunek 5: Schemat podłączenia ESP32-S3



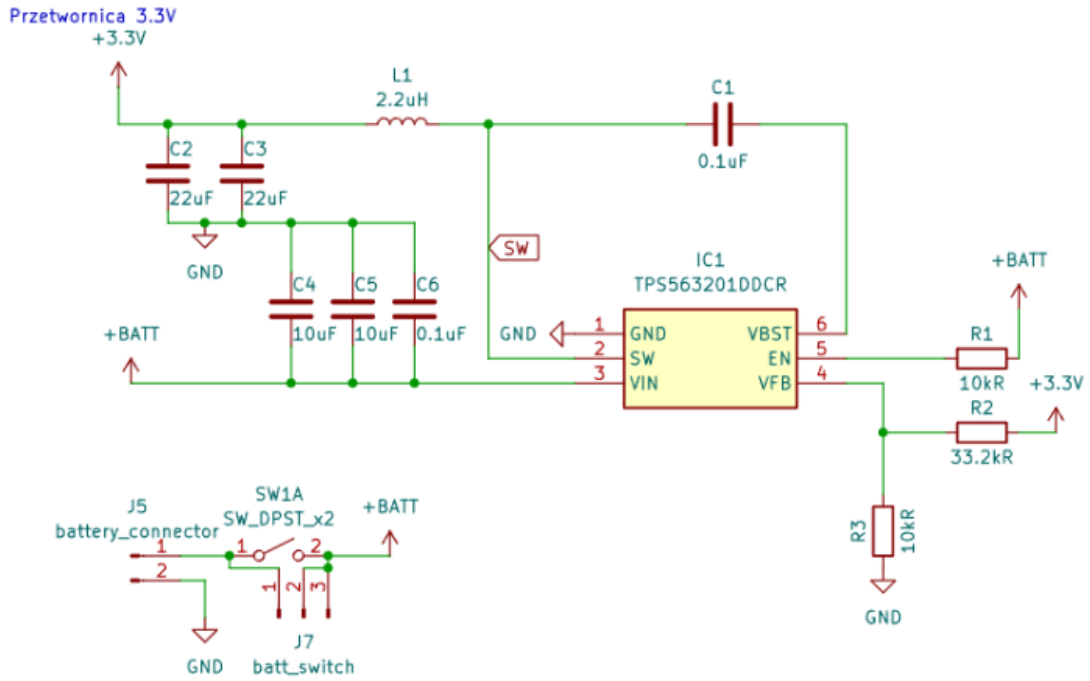
Rysunek 6: Schemat podłączenia przycisków, diód i dzielnika napięcia



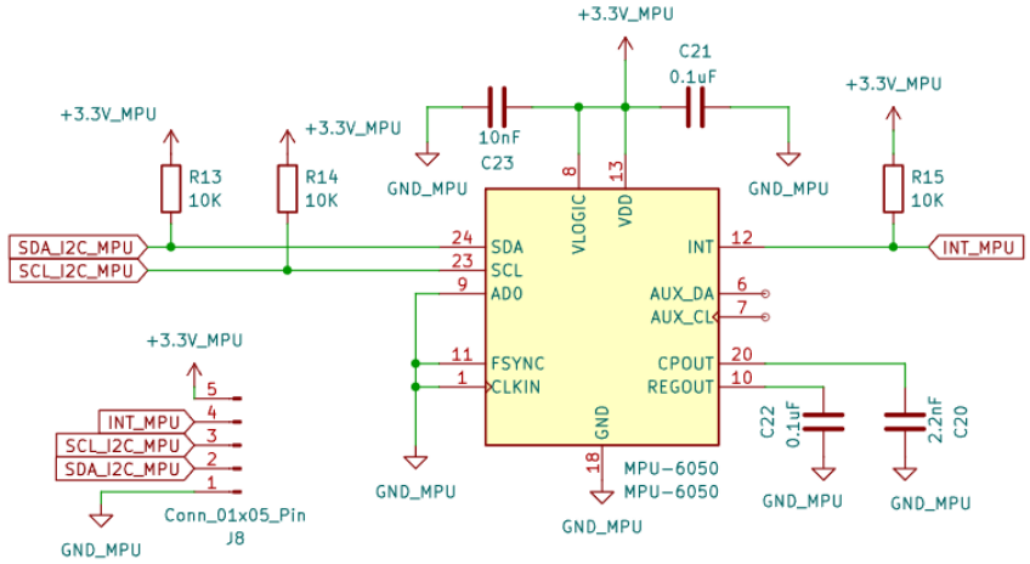
Rysunek 7: Schemat podłączenia jednego z mostków H



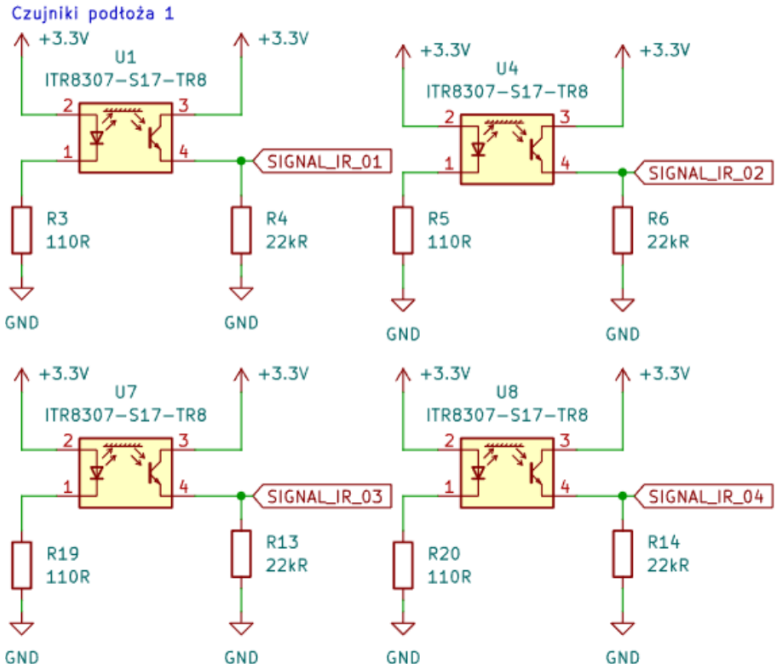
Rysunek 8: Schemat podłączenia enkoderów



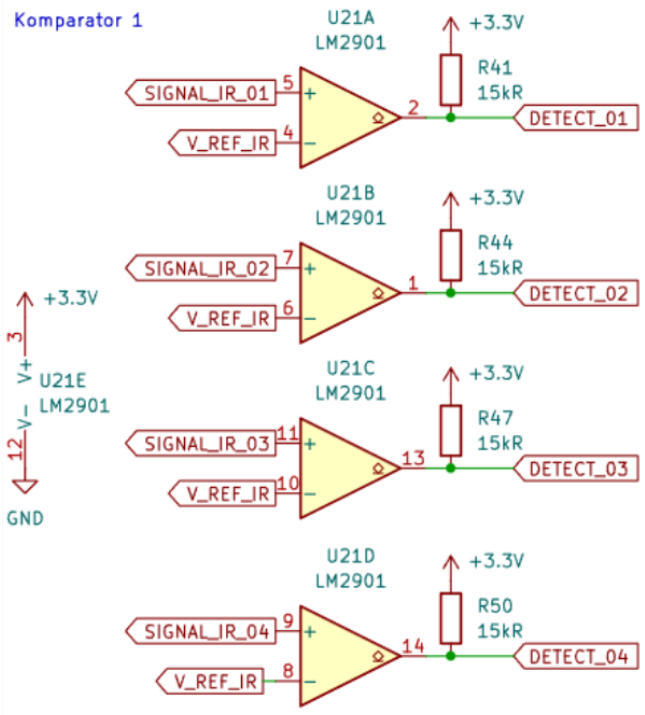
Rysunek 9: Schemat sekcji zasilania



Rysunek 10: Schemat podłączenia MPU6050



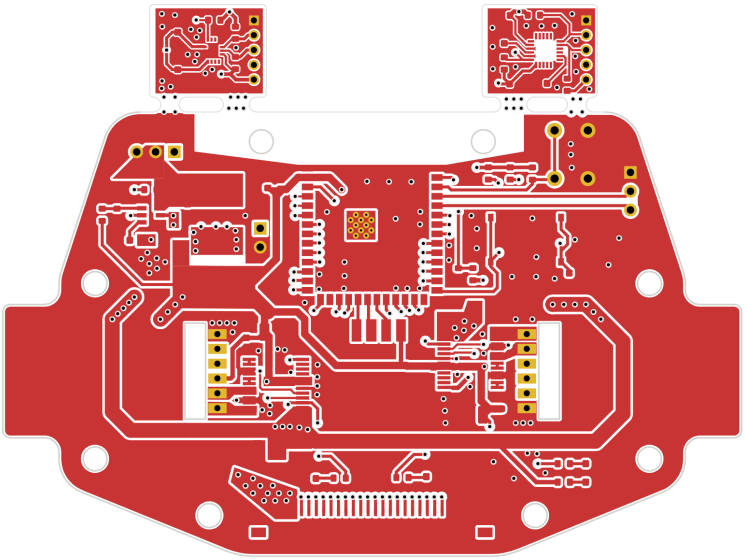
Rysunek 11: Schemat podłączenia czujników odbiciowych



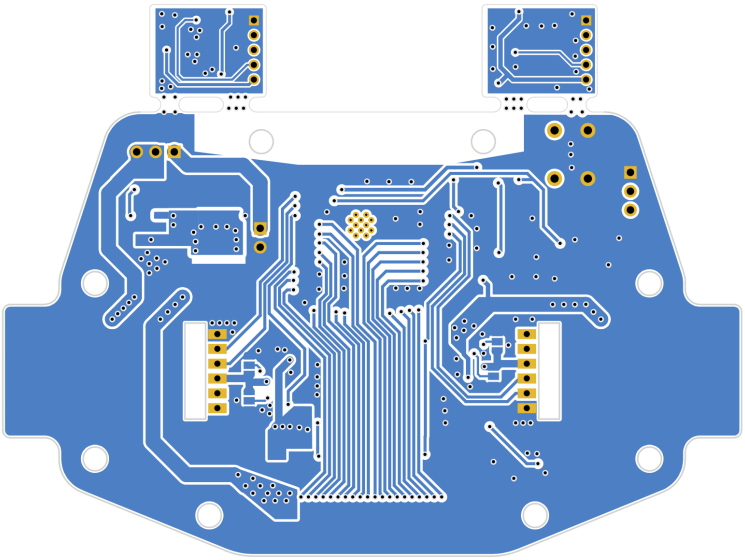
Rysunek 12: Schemat podłączenia komparatora

7.4 Projekty PCB

Płytka główna:

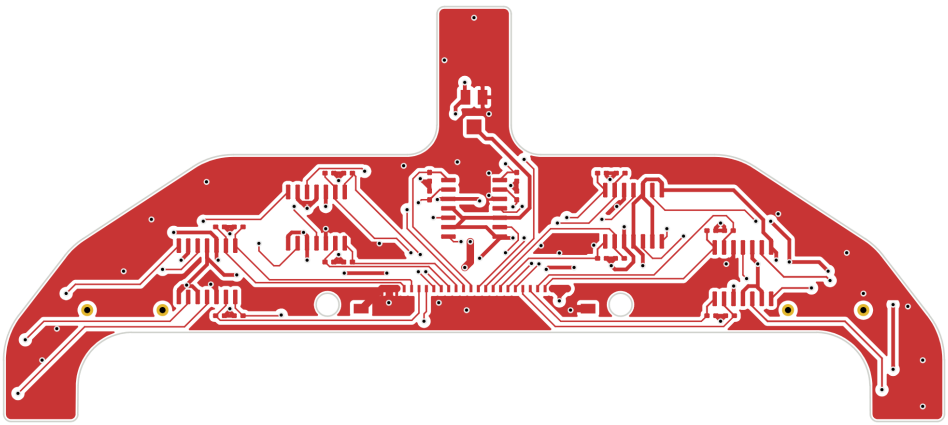


Rysunek 13: Widok PCB z góry

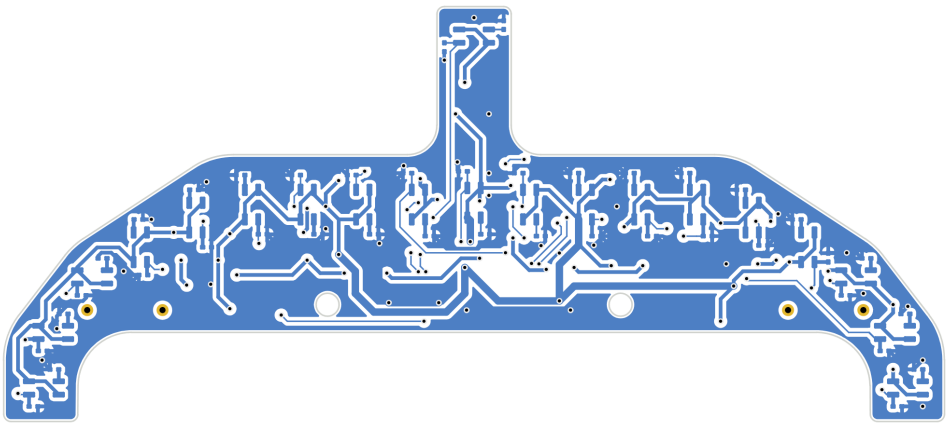


Rysunek 14: Widok PCB z dołu

Płytka z czujnikami

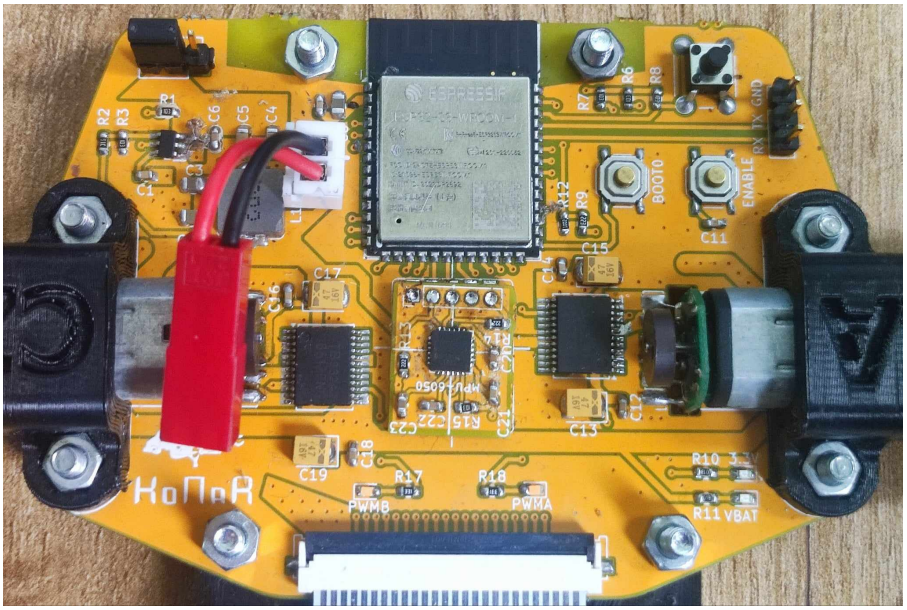


Rysunek 15: Widok PCB z góry

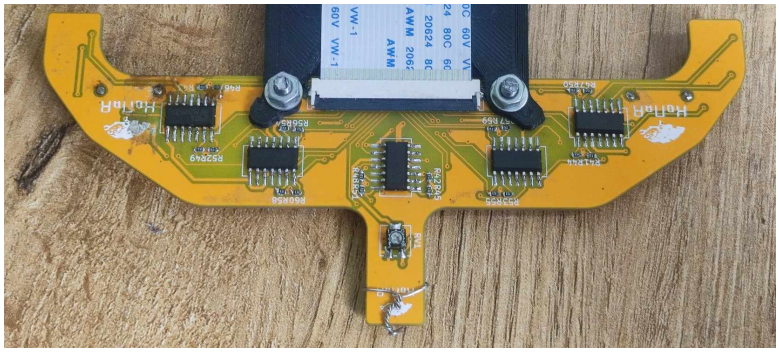


Rysunek 16: Widok PCB z dołu

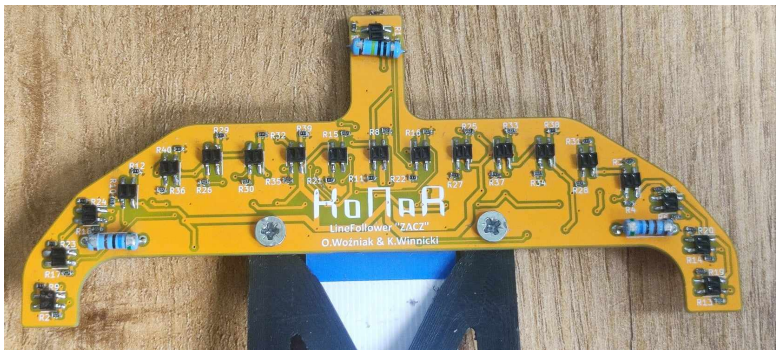
7.5 PCB



Rysunek 17: Płytka główna



Rysunek 18: Płytka z czujnikami - od góry



Rysunek 19: Płytka z czujnikami - od dołu

8 Oprogramowanie

Do zaprogramowania płytki użyliśmy rozszerzenie do VSCode o nazwie platform.io, dało nam to możliwość do szybkiego wgrywania kodu na mikrokontroler. Kod został napisany przy pomocy bibliotek Arduino, które pozwoliły nam na dodanie warstwy abstrakcji do odczytów z różnych czujników i sterowania peryferiami.

8.1 Definicja pinoutu

Na podstawie schematu elektronicznego, dokumentacji mikrokontrolera oraz testów zlokalizowaliśmy połączenia z wszystkimi peryferiami i nadaliśmy im wszystkim kodowe definicje. Pozwala to na zwiększenie efektywności pisania kodu w następnych etapach. Dla ułatwienia zrozumienia przyszłych funkcji, załączamy nazwy i numery pinów w ramce 1.

```
/*INTERFACE*/
#define BATTERY 1
#define BUTTON 2
#define SDA_I2C 11
#define SCL_I2C 12
#define IMU_INT 13

/*LEFT MOTOR*/
#define LEFT_PWM 40
#define LEFT_ENC_2 41
#define LEFT_ENC_1 42

/*RIGHT MOTOR*/

#define RIGHT_PWM 6
#define RIGHT_ENC_2 4
#define RIGHT_ENC_1 5
```

Kod. 1: Definicje pinoutu

8.2 Odczyt z czujników podłoża

Dla czujników podłoża została zdefiniowana specjalna klasa, która pozwala na kompaktową obsługę odczytów. Dzięki takiemu rozwiązaniu jesteśmy w stanie traktować odczyty jako wartości kąta odchylenia robota od podążanej linii oraz odizolować błędy związane z odczytem tej wartości. Definicja klasy znajduje się w ramce 2 poniżej.

```
#ifndef SENSORS_H
#define SENSORS_H

#include <Arduino.h>


class Sensors{
private:
    const uint8_t IR_SENSORS_PINS[20] = {21,14,47,48,36,38,37,35,10,7,15,17,16,9,18,20,8,19,3,39};
    const int8_t SENSORS_WEIGHTS[20] = {-9,-8,-7,-6,-5,-4,-3,-2,-1,0,1,2,3,4,5,6,7,8,9,0}; /*PID
    WEIGHTS*/
    uint8_t measures[20];
    uint8_t last_measures[20];
    int16_t sensors_error;
public:
    Sensors();
    int16_t readSensors();
    const uint8_t * getSensorsPins();
    uint8_t * getSensorsMeasures();
    void printSensorsMeasures();
    uint16_t getSensorsError();
};


#endif
```

Kod. 2: Definicja klasy obslugującej czujniki podłoża

Większość metod w tej klasie pełni funkcję dostępową do danych, w związku z czym nie ma potrzeby ich dokładnego omawiania. Jedną z metod, którą warto lepiej opisać, jest readSensors(), zwracająca kąt odchylenia robota od podążanej linii. Kod znajduje się w ramce 3 poniżej.

```
int16_t Sensors::readSensors(){
    uint8_t founds_counter = 0;
    int16_t weights_sum = 0;
    measures[4] = 0;

    for(int i = 0; i < 20; i++){

        if(i != 4){
        this->measures[i] = !digitalRead(  IR_SENSORS_PINS[i] );
        }

        if(this->measures[i] == 1){
            weights_sum += this->SENSORS_WEIGHTS[i];
            founds_counter++;
        }
    }




    if(founds_counter > 0){ //jezeli linia zostala znaleziona
        for(int i = 0; i < 20; i++){
            last_measures[i] = measures[i];
        }
    }

    if(founds_counter != 0){
        this->sensors_error = weights_sum / founds_counter;
    }else{
        //ćZmieni na graniczne łżpooenie z ostatniego odczytu
        this->sensors_error = 0;
    }

    return this->sensors_error;
}
```

Kod. 3: Funkcja readSensors

Każdy pin odpowiadający za detekcję linii jest sprawdzany w pętli for, dzięki czemu mamy infornację o położeniu linii w tabeli. Następnie, jeśli odczyty wskazują na obecność linii, podliczona zostaje suma wag odczytów. Ta wartość daje nam numeryczną wartość położenia linii względem robota. Została jeszcze dodana pętla zapisująca odczyty do tabeli zawierającej poprzedzające wartości. W przyszłości pomoże nam to zaimplementować funkcję „odnajdywania linii" gdy ta zostanie zgubiona. Zwracany jest błąd regulacji, który jest wartością interpretowaną jako kąt odchylenia robota od podążanej linii.

8.3 Odczyt prędkości ruchu z enkoderów

Dla poprawnej implementacji algorytmu PID potrzebna jest również informacja o prędkości obrotowej kół. Pomimo tego, że tak zaawansowany program nie został zaimplementowany, to obsłużyliśmy działanie enkoderów. Analogicznie jak w przypadku czujników podłoża, została stworzona specjalna klasa do obsługi enkoderów, jej kod znajduje się w ramce 4.

```
#ifndef ENCODER_H
#define ENCODER_H

#include <Arduino.h>
#include <BindArg.h>
#include <math.h>
#define TICK_PER_ROTATION 29

class Encoder
{
private:
    uint8_t outPin_A;
    uint8_t outPin_B;
    int rotations[2];

    bindArgVoidFunc_t interruptGate = nullptr;
    int               interruptNum;

    uint16_t speed; //rad/s


public:
    Encoder();
    ~Encoder();

    //Inicjalizacja Enkodera
    void begin(uint8_t pin_a, uint8_t pin_b);

    //Funkcja ąwewntrz przerwania, ćzmienia adekwatnie do stosowanej metody pomiaru ęśprdkoci
    void update();

    // Funkcja ąprzepisujca śćilo obrotów do poprzedniej komórki w tablicy
    void calc_speed();

    int get_rotations();

    uint16_t get_speed();
};



#endif
```

Kod. 4: Klasa do obsługi enkoderów

Odczyt prędkości obrotowej zrealizowano na dwóch przerwaniach. Pierwsze jest uruchamiane w momencie, kiedy wartość na jednym z pinów enkodera następuje zmiana stanu, jego kod znajduje się w ramce 5. Natomiast drugie uruchamia się okresowo co ustalony czas i przepisuje wartość odczytanych obrotów do poprzedniego indeksu tablicy (ramka 6). Taka implementacja pozwala na dość dokładny, jednak opóźniony odczyt prędkości obrotowej. Ze względu na użycie przerwań do obsługi tych funkcji, muszą być one mało złożone, w związku z tym faktyczną szybkość obrotową wylicza osobna funkcja znajdująca się w ramce 7.

```
void Encoder::update(){
    if (digitalRead(this->outPin_A) == LOW)
    {
        if (digitalRead(this->outPin_B)==LOW)
        {
            ++rotations[1];
        }
        else {
            --rotations[1];
            }
    }
}
```

Kod. 5: Metoda zwiększająca liczbę zliczonych impulsów

Pomimo tego, że robot porusza się jedynie w jednym kierunku, została zaimplementowana obsługa obrotu kół w obie strony, na wypadek działania sił zewnętrznych.

```
void Encoder::calc_speed(){
    rotations[0] = rotations[1];
    rotations[1] = 0;
}
```

Kod. 6: Metoda uruchamiana cyklicznie w celu przesunięcia wartości w tabeli ze zliczonymi impulsami

```
uint16_t Encoder::get_speed(){
    speed = (rotations[0])*20;
    speed = (speed * 2 * M_PI)/TICK_PER_ROTATION;
    return speed;
}
```

Kod. 7: Metoda obliczająca prędkość ruchu na podstawie zliczonych impulsów w ostatnim cyklu

Obie metody spełniają jedną funkcję, jednak ze względu na długość czas wykonywania operacji matematycznych w funkcji $get_speed()$ byliśmy zmuszeni do wydzielenia jej. Użyty mikrokontroler ma bardzo rygorystyczny reżim czasowy, który przerywa działanie przerwań wykonywanych zbyt długo.

8.4 Sterowanie silnikami

Jak już wcześniej zostało wspomniane, silniki są sterowane za pomocą regulatora PID zależnego od odchyłu linii względem robota. Wartość tą uzyskujemy z metody *readSensors*(), klasy obsługującej czujniki podłoża. Ze względu na dużą ilość danych potrzebnych do działania regulatora oraz potrzebę możliwości zmiany wartości regulatora PID podczas działania mirkokontrolera, została utworzona kolejna klasa do jego obsługi. Opis tej klasy znajduje się w ramce 8.

```
#ifndef PID_H
#define PID_H


#include <Arduino.h>
#include "Sensors.h"
#include "defines.h"

#define MAX_PWM_VALUE_CPU 255
#define SAMPLING_TIME 2 //ms

class Regulator{

    private:
    uint8_t baseSpeed;

    int16_t last_ang_err;

    uint8_t leftPWM_value;
    uint8_t rightPWM_value;

    uint8_t leftPWM_percent;
    uint8_t rightPWM_percent;

    uint8_t Kp, Ki, Kd;

    uint32_t sum_I;

    int16_t PID(const int16_t ang_error);
    int16_t integrate(const int16_t curr_value);

    public:

    Regulator();
    void begin();
    void regulator(const int16_t ang_error);

    void set_pid(uint8_t k_p, uint8_t k_i, uint8_t k_d);
    void set_base_speed(uint32_t speed);

    uint8_t get_right_percent();
    uint8_t get_left_percent();

    uint8_t get_right_value();
    uint8_t get_left_value();

};

#endif
```

Kod. 8: Klasa do obsługi regulatora

Klasa ta zapewnia pełną obsługę regulatora, wraz z inicjalizacją silników i ustawianiem wartości sterującej. W celu realizacji tej funkcjonalności wykorzystywane są dwie główne metody: $PID(constint16_t ang_error)$ i $regulator(constint16_t ang_error)$. Pierwsza z nich ustala wartość sterującą na podstawie wewnętrznych wartości regulatora i podanej wartości błędu. Jest ona przedstawiona w ramce 10. Natomiast druga zadaje wartości silników na podstawie wyników działania PID, ta metoda znajduje się w ramce 11. Taki podział funkcji pozwala na zwiększenie czytelności kodu i kompaktowości metod. Klasa regulator ma również inicjalizator w postaci metody begin(), której kod znajduje się w ramce 9

```
void Regulator::begin(){
        pinMode(LEFT_PWM, OUTPUT);
        pinMode(RIGHT_PWM, OUTPUT);
    }
```

Kod. 9: Metoda inicjalizująca regulator

```
int16_t Regulator::PID(const int16_t ang_error){
        uint16_t Proportional,Integral, Differencial;
        if(this->Kp!=0) Proportional = this->Kp*ang_error;
        else Proportional = 0;

        if(this->Ki!=0) Integral = this->Ki*integrate(ang_error);
        else Integral = 0;

        if(this->Kd!=0) Differencial = this->Kd*((ang_error-last_ang_err)/SAMPLING_TIME);
        else Differencial = 0;

        int16_t pid_val = Proportional + Integral + Differencial;

        last_ang_err=ang_error;

        return pid_val;
    }
```

Kod. 10: Metoda wyliczająca wartość PID

```
void Regulator::regulator(const int16_t ang_error){

    int16_t pid_val = PID(ang_error);
    int16_t leftPWM=this->baseSpeed+pid_val, rightPWM=this->baseSpeed-pid_val;
    if (leftPWM<0)  leftPWM=0;
    if (rightPWM<0)  rightPWM=0;
    if (leftPWM> MAX_PWM_VALUE_CPU) leftPWM = MAX_PWM_VALUE_CPU;
    if (rightPWM> MAX_PWM_VALUE_CPU)      rightPWM = MAX_PWM_VALUE_CPU;

    this->leftPWM_percent = (leftPWM * 100)/MAX_PWM_VALUE_CPU;
    this->rightPWM_percent = (rightPWM * 100)/MAX_PWM_VALUE_CPU;
    this->leftPWM_value = leftPWM;
    this->rightPWM_value=rightPWM;

    analogWrite(LEFT_PWM,leftPWM);
    analogWrite(RIGHT_PWM, rightPWM);

    }
```

Kod. 11: Metoda ustawiająca wartość wypełnienia PWM na sterowanych silnikach

9 Podsumowanie

Wszystkie założone funkcjonalności projektu zostały spełnione. Robot nie potrzebuje wprowadzenia większych poprawek w elektronice czy mechanice, wymagane jest jedynie ulepszenie oprogramowania, które przyśpieszy przejazd. Utworzony przez nas robot stanowi bardzo dobrą podstawę do implementacji zaawansowanych algorytmów sterowania robotem typu LineFollower. Obecnie działanie sterownika PID oparte jest tylko o odczyty z czujników podłoża. W przyszłości sterowanie można ulepszyć o wykorzystanie pozostałych sensorów takich jak enkodery oraz 6-cio osiowe IMU. Dodatkowo dzięki nim można zastosować algorytmy mapowania trasy. Przy skorzystaniu z tak utworzonej mapy jesteśmy w stanie znacznie zoptymalizować przejazd robota w taki sposób który znacznie ukróci czas przejazdu.

10 Repozytorium Projektu

`https://github.com/KamilWuu/Line_Follower_ZACZ/tree/main`

Bibliografia

[1] Maurycy Gast. "Rola robotyki w rozrywce". URL: `https://zpe.gov.pl/a/przeczytaj/DUpr2MS6R`.

[2] Jon Evans Jean-Pierre Charras Fabrizio Tappero. *8.0: Polski: Documentation*. URL: `https://docs.kicad.org/8.0/pl/kicad/kicad.html`.

[3] dr.inż. Robert Muszyński. "Prace studenckie". URL: `https://kcir.pwr.edu.pl/~mucha/`.