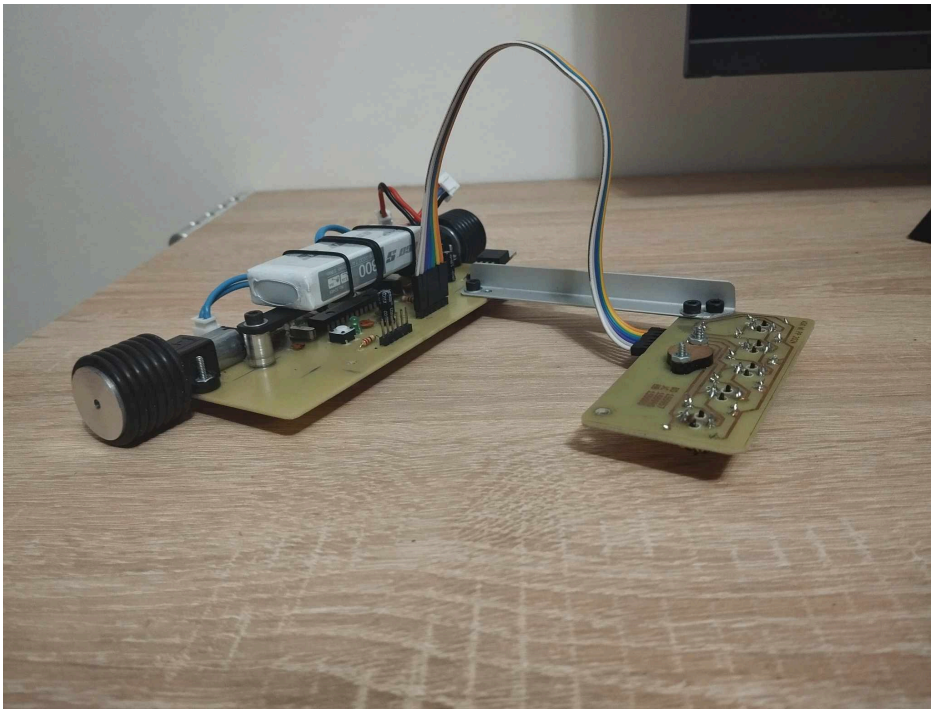
INSTYTUT INFORMATYKI
WYDZIAL INFORMATYKI I TELEKOMUNIKACJI, POLITECHNIKA POZNANSKA

ARCHITEKTURA SYSTEMÓW KOMPUTEROWYCH 2024

prowadzący: dr inż. RAFAŁ KLAUS

Line Follower

drużyna warsztatowa nr 8



Lp.	*Autorzy*	*Indeks*	*nr grupy dziekańskiej*	*Zadania*
1	*Hubert Błaszczyk*	*155085*	*6*	*1. Wytrawienie płyt 2. Realizacja oprogramowania 3. Wykonanie filmu 4. Testowanie i trymowanie systemu*
2	*Jan Cieśla*	*155945*	*4*	*1. Wytrawienie płyt 2. Projekt robota 3. Montaż komponentów 4. Testowanie i trymowanie systemu*
3	*Bartosz Operacz*	*155957*	*1*	*1. Wytrawienie płyt 2. Realizacja oprogramowania 3. Przygotowanie dokumentacji 4. Testowanie i trymowanie systemu*

Poznań, 2024

Wprowadzenie

Celem naszej drużyny było własnoręczne stworzenie robota typu LineFollower, którego zadaniem jest podążanie po znajdującej się na podłodze czarnej linii.

Pomysł zakłada że robot powinien:

- wykrywać znajdującą się pod nim linię,
- mieć możliwość poruszania się po płaskiej powierzchni, skręcania
- posiadać własne zasilanie,
- generować odpowiednie sygnały umożliwiające podążanie za linią,
- móc informować o jego prawidłowym funkcjonowaniu,
- mieć możliwość łatwego programowania z zewnątrz.

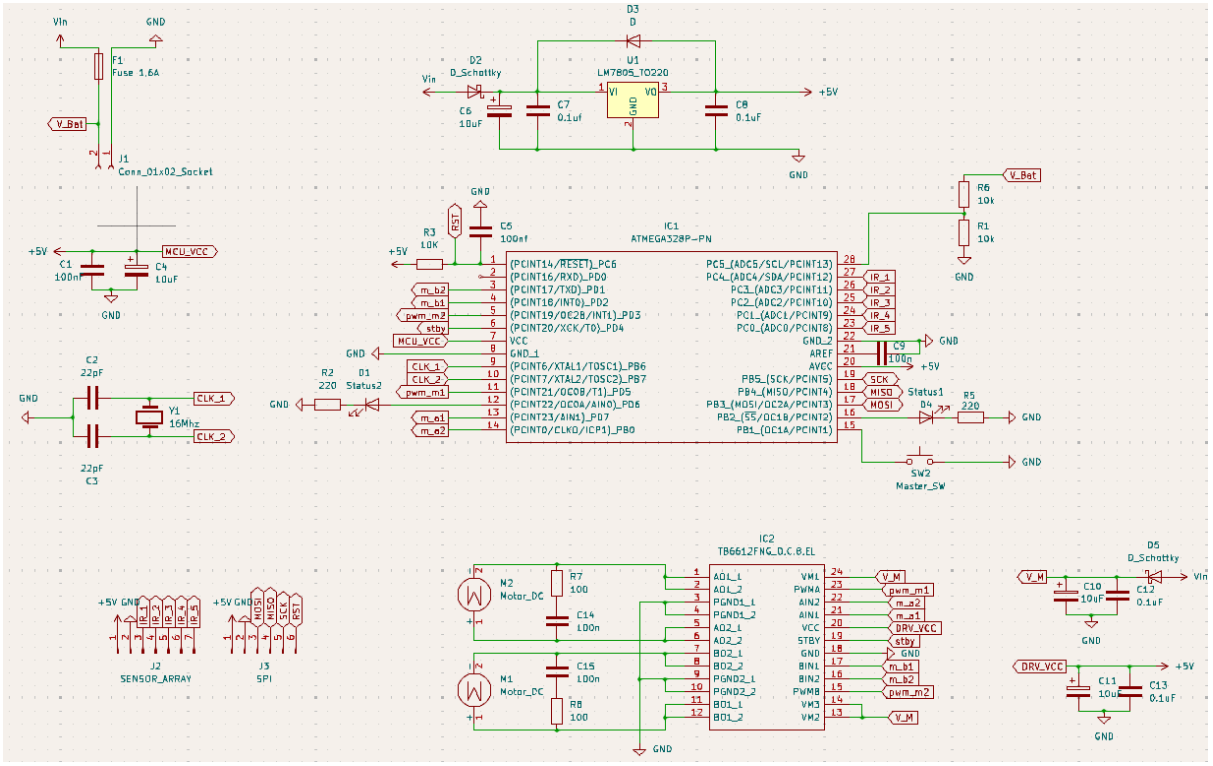
Rozwiązania techniczne

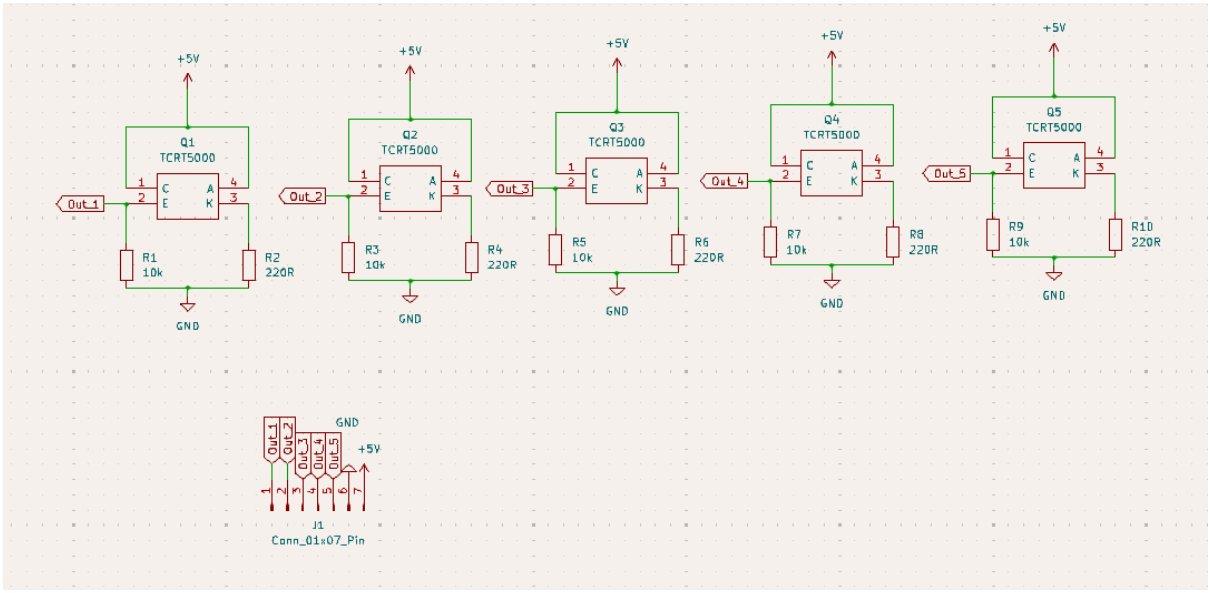
W robocie zastosowano:

- 5 transoptorów odbiciowych **TCRT5000** aby wykrywać czarną linię. Zastosowane ze względu na wysoki dystans działania.
- 2 silniki **N20-BT16** służące za napęd, połączone bezpośrednio do kół robota pozbywając się w ten sposób problemów związanych z przenoszeniem napędu, jednocześnie realizując skręt. 2000 rpm pozwala na jazdę z prędkością ok 2m/s przy zastosowanych przez nas kołach fi24mm, zapewniając zadowalające przyspieszenie.
- dwukanałowy sterownik silników **TB6612FNG** umożliwiający sterowanie silnikami,
- pakiet **Li-Pol 2S 7.4V 20C Dualsky 800mAh** - zasilanie robota. Dobrane ze względu na dobry stosunek wagi do pojemności oraz możliwość dostarczenia prądu o dużym natężeniu.
- stabilizator napięcia **LM7805** wraz z układem zabezpieczenia przed prądem zwrotnym,
- mikrokontroler **ATMEGA328P-PN** taktowany oscylatorem 16MHz,
- dzielnik napięcia aby umożliwić pomiar napięcia pakietu i wykrycie gdy jest on zbyt niski,
- 2 diody led oraz przycisk w celu umożliwienia komunikacji człowiek-robot,
- wyprowadzony interfejs spi służący do programowania robota.
- zabezpieczenie przed odwrotnym podłączeniem zasilania w postaci asymetrycznego gniazda zasilania
- Filtracja zasilania dla układów scalonych, układ zaprojektowany z troską o minimalizację zakłóceń

Opis konstrukcji robota

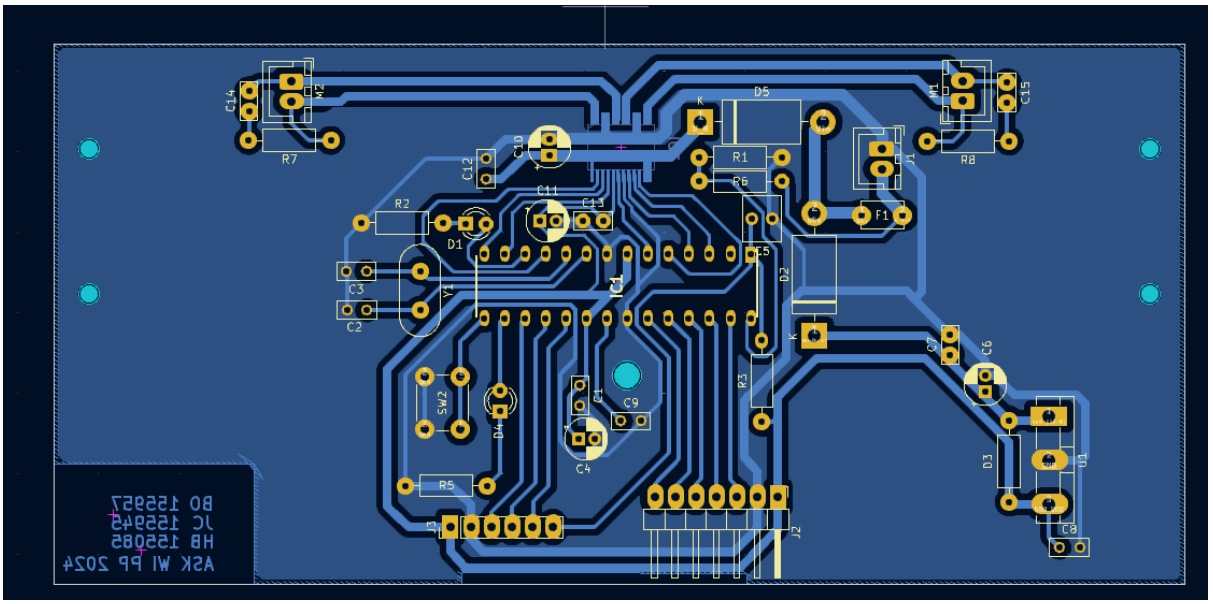
Robot będący połączeniem 2 własnoręcznie wytrawionych płytek przypomina literę “T” gdzie na mniejszej płytce umiejscowionej z przodu robota znajdują się transoptory ułożone w taki sposób by poprawić jego możliwości do wczesnego i efektywnego wykrywania czarnej linii. Cała reszta robota znajduje się w tyle dzięki czemu jego środek ciężkości znajduję się bliżej napędu ułatwiając sterowanie robotem.



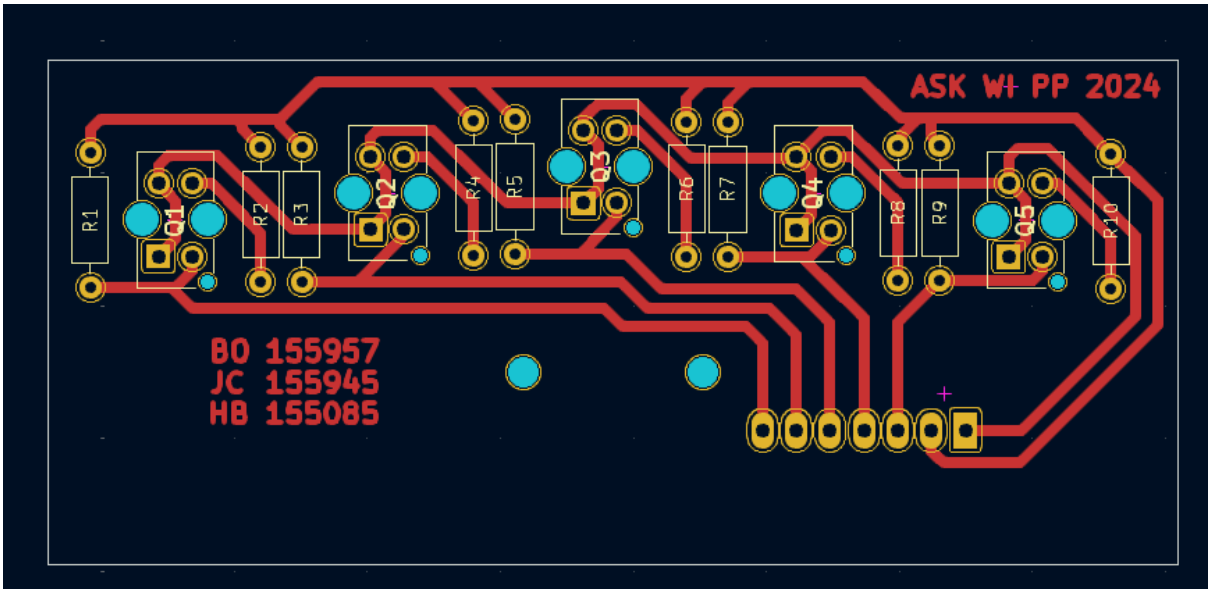


Schematy i opis

Płyta główna:



Płyta czujników:



Opis trawienia:

- Przygotowanie klisz - wydruk layout'u na kalce technicznej lub przezroczystej folii. W celu ograniczenia transparentności klisz połączyliśmy dwie naraz, wymagane jest precyzyjne ustawienie z pomocą znaków ustalających na wydrukach.
- Przygotowanie ciemni (żółte / czerwone światło), roztworów wywoływacza uniwersalnego w proporcji 5.5g/100ml i nadsiarczanu sodu 20g/100ml oraz laminatu z fotorezystem pozytywowym dociętym ok 5mm nad wymiar, należy ogratować krawędzie laminatu. Wywoływacz w temperaturze pokojowej, wytrawiacz należy podgrzać w kąpieli wodnej do ok 50 stopni.
- Zdjęcie folii ochronnej z laminatu, ustawienie kliszy (klisza nie odbita lustrzanie, tonerem do laminatu).
- Naświetlenie płyty światłem UV przez eksperymentalnie dobrany czas (w naszym przypadku 180s). Należy zwrócić uwagę na notę katalogową laminatu i długość fali emitowaną przez naszą lampe, w naszym przypadku duża lampa do paznokci spełniła swoje zadanie. Podczas naświetlania wymagany jest docisk kliszy do laminatu aby uniknąć wybrzuszania się kliszy. (Do przygotowania małego PCB wystarczyła nam ciężka szyba, w przypadku płyty głównej posłużyliśmy się workiem próżniowym i ciśnieniem atmosfery)
- Wywoływanie ok 150-180s w roztworze o temperaturze pokojowej, wymagane mieszanie. Po wyjęciu płytkę należy wypłukać. Za krótko wywoływany rezyst nie dopuści do poprawnego wytrawiania się PCB.
- Wytrawianie ok 10-20min w zależności od rozmiaru płyty i świeżości roztworu, wymagane utrzymywanie b327 w podwyższonej temperaturze. Trawimy do momentu zaniku 100% miedzi z przestrzeni między ścieżkami. Po wyjęciu płytkę należy wypłukać.
- Zmycie pozostałego rezystu acetonem, obcięcie nadwymiarowego laminatu piłą włośnicową.
- Zabezpieczenie PCB lakierem / soldermaską.

Oprogramowanie

W celu zapewnienia odpowiedniego sposobu działania robota wymagane było napisanie oprogramowania, którego zadaniem było odczytywanie wartości sygnałów zwracanych przez transoptory odbiciowe i na ich podstawie generowanie odpowiednich sygnałów do sterownika silników aby robot poruszał się zgodnie z czarną linią.

Wykorzystane biblioteki

W programie skorzystano z biblioteki TB6612 SparkFun ułatwiającej sterowanie silnikami.

```
#include "SparkFun_TB6612.h"
```

Znajdujące się w niej klasa, metody, i procedury, które zostały użyte w programie to:

- klasa Motor

```
Motor::Motor(int In1pin, int In2pin, int PWMpin, int offset, int STBYpin)
{
  In1 = In1pin;
  In2 = In2pin;
  PWM = PWMpin;
  Standby = STBYpin;
  Offset = offset;

  pinMode(In1, OUTPUT);
  pinMode(In2, OUTPUT);
  pinMode(PWM, OUTPUT);
  pinMode(Standby, OUTPUT);
}
```

- drive (wykorzystywana do poruszania konkretnym silnikiem)

```
void Motor::drive(int speed)
{
  digitalWrite(Standby, HIGH);
  speed = speed * Offset;
  if (speed>=0) fwd(speed);
  else rev(-speed);
}
```

  - fwd

```
void Motor::fwd(int speed)
{
    digitalWrite(In1, HIGH);
    digitalWrite(In2, LOW);
    analogWrite(PWM, speed);

}
```

  - rev

```
void Motor::rev(int speed)
{
    digitalWrite(In1, LOW);
    digitalWrite(In2, HIGH);
    analogWrite(PWM, speed);
}
```

- procedura brake (wykorzystywana do hamowania obu silników)

```
void brake(Motor motor1, Motor motor2)
{
  motor1.brake();
  motor2.brake();
}
```

  - metoda brake

```
void Motor::brake()
{
    digitalWrite(In1, HIGH);
    digitalWrite(In2, HIGH);
    analogWrite(PWM,0);
}
```

Zdefiniowane stałe

Dla ułatwienia i zwiększenia czytelności zdefiniowano makra zastępujące identyfikatory odpowiednimi numerami pinów i innych pomocniczych stałych.

```
// Piny do silników
#define AIN1 7
#define AIN2 8
#define PWMA 5

#define BIN1 2
#define BIN2 1
#define PWMB 3

#define STBY 4

// Piny do sensorów
#define S1 A0
#define S2 A1
#define S3 A2
#define S4 A3
#define S5 A4

// Pozostałe piny
#define CN A5       //Czujnik napięcia
#define Led1 10     //Przedni led
#define Led2 6      //Tylni led
#define Button 9
```

```
// Stałe związane z silnikami
#define Max_speed 130.0
#define Min_speed 5
#define Max_acceleration 30

// Stałe związane z wyliczaniem pozycji linii
#define Middle_point 2000
#define Sensor_weight 1000

// PID
#define Kp 0.79
#define Ki 0.0003
#define Kd 0.72

// Pozostałe używane stałe
#define Max_offroad_time 1000
#define Delay_time 1
```

Wykorzystane funkcje

Na potrzeby programu zdefiniowano kilka makr, procedur i funkcji pomocniczych:

- sbi i cbi (wykorzystywane do ustawiania konkretnych bitów rejestrów sfr)

```
#define sbi(sfr, bit) (_SFR_BYTE(sfr) |= _BV(bit))
#define cbi(sfr, bit) (_SFR_BYTE(sfr) &= ~_BV(bit))
```

- start (wykorzystywana podczas włączania robota)

```
void start(){
  // Procedura włączająca robota
  digitalWrite(STBY, HIGH);

  delay(1000);
  digitalWrite(Led1, HIGH);
}
```

- shutdown (wykorzystywana podczas wyłączania robota)

```
void shutdown(){
  // Procedura wyłączająca robota
  brake(motor1, motor2);
  delay(1500);

  digitalWrite(STBY, LOW);

  delay(500);
  digitalWrite(Led1, LOW);
}
```

- calculate_error (wykorzystywana do wyliczania błędu)

```
long calculate_error(long position){
  // Funkcja implementująca regulator PID i zwracająca obliczoną wartość błędu
  proportional = position - Middle_point;
  integral = integral + proportional;
  derivative = proportional - last_proportional;
  last_proportional = proportional;
  error_value = long(proportional * Kp + integral * Ki + derivative * Kd);
  return error_value;
}
```

- calculate_motors_speed (wykorzystywana do wyliczania prędkości obu silników)

```
void calculate_motors_speed(long error_value){
  // Funkcja zwracająca obliczone wartości prędkości silników

  // Ograniczenie wartości błędu do wartości maksymalnej prędkości
  if(error_value < -Max_speed)   error_value = -Max_speed;
  if(error_value > Max_speed)    error_value = Max_speed;

  // W zależności od znaku błędu wybierany jest silnik którego prędkość jest modyfikowana
  if (error_value >= 0){
    right_speed = Max_speed - error_value;
    left_speed = Max_speed;
  }else{
    right_speed = Max_speed;
    left_speed = Max_speed + error_value;
  }
}
```

- drive_motors (wykorzystywana do sterowania silnikami)

```
void drive_motors(int right_speed, int left_speed){
  // Procedura sterująca silnikami robota

  // Ograniczenie wartości prędkości do wartości maksymalnej prędkości lub 0
  if (right_speed > Max_speed)  right_speed = Max_speed;
  if (right_speed < 0)          right_speed = 0;
  if (left_speed > Max_speed)   left_speed = Max_speed;
  if (left_speed < 0)           left_speed = 0;

  // Wyliczenie maksymalnej prędkości silników na podstawie ostatniej prędkości oraz maksymalnego przyśpieszenia
  left_speed_cap = last_left_speed + Max_acceleration;
  right_speed_cap = last_right_speed + Max_acceleration;

  // Ograniczenie limitu prędkości silników
  if(Max_speed < left_speed_cap)      left_speed_cap = Max_speed;
  if(Max_speed < right_speed_cap)     right_speed_cap = Max_speed;

  // Ograniczenie prędkości silników
  if(left_speed_cap < left_speed)     left_speed = left_speed * (left_speed_cap/Max_speed);
  if(right_speed_cap < right_speed)   right_speed = right_speed * (right_speed_cap/Max_speed);

  // Ograniczenie prędkości silników zapobiegające hamowaniu silników
  if(left_speed < Min_speed)    left_speed = Min_speed;
  if(right_speed < Min_speed)   right_speed = Min_speed;

  // Zapamiętywanie wartości prędkości silników
  last_left_speed = left_speed;
  last_right_speed = right_speed;

  // Sterowanie silnikami
  motor2.drive(left_speed);
  motor1.drive(right_speed);
}
```

- calibrate (wykorzystywana do kalibracji sensorów)

```
void calibrate(){
  // Procedura kalibrująca sensory na podstawie podłoża
  start();
  delay(1000);

  // Wstępne odczyty sensorów
  for(int i = 0; i < 5; i++){
    sensors[i] = analogRead(sensor_pins[i]);

    min_values[i] = sensors[i];
    max_values[i] = sensors[i];
  }

  // Kolejne odczyty sensorów w trakcie obracania się robota
  motor1.drive(50);
  motor2.drive(-50);
  for(int j = 0; j < 5000; j++){
    for(int i = 0; i < 5; i++){
      sensors[i] = analogRead(sensor_pins[i]);

      if(sensors[i] < min_values[i])  min_values[i] = sensors[i];
      if(sensors[i] > max_values[i])  max_values[i] = sensors[i];
    }
  }

  shutdown();
}
```

Deklaracja i inicjalizacja zmiennych

Na początku programu są deklarowane i inicjalizowane wszystkie zmienne wymagane do poprawnego działania programu.

```
// Konfiguracja silników
const int offset_A = 1;
const int offset_B = -1;

Motor motor1 = Motor(AIN1, AIN2, PWMA, offset_A, STBY);      //prawy motor
Motor motor2 = Motor(BIN1, BIN2, PWMB, offset_B, STBY);      //lewy motor

// Konfiguracja sensorów
const int sensor_pins[5] = {S1,    S2,   S3,   S4,   S5  };
int min_values[5] =        {0,     0,    0,    0,    0   };
int max_values[5] =        {1023,  1023, 1023, 1023, 1023};
long sensor_mults[5] =     {0,     0,    0,    0,    0   };

// Zmienne do wyliczania pozycji linii
long sensors_sum = 0;
long sensors_average = 0;
long sensors[5] = {0,0,0,0,0};
long position = 0;

// Zmienne do PID
long proportional = 0;
long integral = 0;
long derivative = 0;
long last_proportional = 0;
long error_value = 0;

// Zmienne do sterowania silników
int left_speed = 0;
int right_speed = 0;

int last_left_speed = 0;
int last_right_speed = 0;

float left_speed_cap = 0;
float right_speed_cap = 0;

// Zmienne pomocnicze
float volts = 0;
bool robot_running = false;
bool current_switch_state = HIGH;
bool last_switch_state = HIGH;
int offroad_time = 0;
```

Przygotowanie mikrokontrolera do działania

Następnie dokonywana jest:

- konfiguracja,

```
void setup() {
  // Skonfigurowanie referencyjnego napięcia na wejściach analogowych na 5 V
  analogReference(DEFAULT);

  // Skonfigurowanie zegara do przetwornika analogowo-cyfrowego
  sbi(ADCSRA, ADPS2);
  sbi(ADCSRA, ADPS1);
  cbi(ADCSRA, ADPS0);
```

- ustawiane są wszystkie dostępne piny aby zachowywały się w odpowiedni sposób,

```
  // Ustawienie trybu działania na pinach związanych z sensorami
  for (int i = 0; i < 5; i++) {
    pinMode(sensor_pins[i], INPUT);
  }

  // Ustawienie trybu działania na pozostałych używanych pinach
  pinMode(CN, INPUT);
  pinMode(Led1, OUTPUT);
  pinMode(Led2, OUTPUT);
  pinMode(Button, INPUT_PULLUP);

  // Ustawienie trybu działania na pozostałych nieużywanych pinach
  pinMode(0, INPUT_PULLUP);
  for( int i = 11; i < 14; i++){
    pinMode(i , INPUT_PULLUP);
  }
```

- zostają wyliczone wagi poszczególnych sensorów,

```
  // Wyliczanie wag poszczególnych sensorów
  for (int i = 0; i < 5; i++){
    sensor_mults[i] = i * Sensor_weight;
  }
```

- oraz oba znajdujące się na płytce ledy zostaję wyłączane.

```
  // Wyłączanie ledów znajdujących się na płytce
  digitalWrite(Led1, LOW);
  digitalWrite(Led2, LOW);
}
```

Główna pętla programu

Główna pętla programu składa się z 3 głównych części:

1. Sprawdzenie poziomu naładowania akumulatora

Wykorzystując czujnik napięcia wyliczane jest faktyczne napięcie na akumulatorze które następnie jest porównywane i w przypadku spadku poniżej założonej wartości robot

odpowiednio informuje zapalając odpowiednią diodę, lub w przypadku jeszcze większego odchyłu napięcia program zatrzyma działanie i nie pozwoli na ponowne uruchomienie robota jednocześnie informując o krytycznym poziomie akumulatora migającą diodą led.

```
void loop() {
  // Wyliczanie napięcia akumulatora
  volts = analogRead(CN) * 10.0 / 1023.0;
  if(volts < 7.7){
    // Poniżej minimalnego napięcia baterii
     if(robot_running){
       // Zatrzymanie robota ze względu na rozładowanie akumulatora
       robot_running = false;
       shutdown();
     }

    while(1){
      // Pętla uniemożliwiająca wznowienie działania robota bez odłączenia i podładowania akumulatora
      digitalWrite(Led2, HIGH);
      delay(1000);
      digitalWrite(Led2, LOW);
      delay(1000);
    }
  }else if(volts < 8){
    // Poniżej zalecanego napięcia baterii
    digitalWrite(Led2, HIGH);
  }else{
    // Powyżej zalecanego napięcia baterii
    digitalWrite(Led2, LOW);
  }
```

2. Sekcja związana z przyciskiem

Za pomocą zmiennej przechowującej ostatni stan przycisku jest możliwe wyłapanie przez program momentu jednokrotnego naciśnięcia przycisku a przez powtórne sprawdzenie stanu przycisku możliwe jest rozróżnienie krótszego i dłuższego przytrzymania przycisku dzięki czemu można wybrać czy robot powinien zacząć kalibrację lub odpowiednio zostać włączony lub wyłączony zależnie od jego aktualnego stanu (w trakcie działania robota kalibracja jest niemożliwa). O aktualnym stanie robota informuje dioda led umieszczona w tylnej części robota.

```
// Sprawdzenie stanu przycisku
current_switch_state = digitalRead(Button);
if(current_switch_state == LOW && last_switch_state == HIGH){
  delay(500);
  if((!robot_running) && (digitalRead(Button) == LOW)){
    // Dłuższe przytrzymanie przycisku kiedy robot jest wyłączony atywuje kalibracje robota
    calibrate();
  }else{
    // Krótsze przytrzymanie przycisku włączające lub wyłączające robota
    robot_running = !robot_running;

    if(robot_running){
      start();
    }else{
      shutdown();
    }

    delay(1000);
  }
}
last_switch_state = current_switch_state;
```

3. Sekcja związana z jazdą po linii

Jeżeli robot został włączony przez naciśnięcie przycisku to zaczyna odczytywać wartości z sensorów, które następnie są wykorzystywane do określenia pozycji czarnej linii. Znając pozycję można przystąpić do wyliczanie błędu czyli wartości wskazującej na to jak mocno i w którą stronę należy skręcić aby pozostać na linii. Do wyliczenia tej wartości zastosowano regulator PID. Mając wyliczony błąd można następnie wykorzystać go do obliczenia prędkości silników i następnie odpowiednio wysterować silniki.
Zanim zostaje obliczona pozycja linii następuje sprawdzenie czy robot faktycznie znajduję się w miejscu w którym mógłby śledzić linię. Jeżeli tak nie jest to zamiast wyliczenia nowych prędkości silników, robot porusza się w ten sam sposób jak w ostatniej iteracji oraz jest zwiększana zmienna oznaczająca czas poza trasą a w momencie kiedy przekroczy ona dopuszczalną wartość, robot zostaje automatycznie wyłączony.

```
  if(robot_running){
    // Zerowanie wartości wykorzystywanych do wyliczenia pozycji linii
    sensors_sum = 0;
    sensors_average = 0;

    for (int i = 0; i < 5; i++){
      // Mapowanie wartości sensorów na przedział od 0 do 1000 korzystając z wartości uzyskanych podczas kalibracji
      sensors[i] = map(analogRead(sensor_pins[i]), min_values[i], max_values[i], 0, 1000);
      sensors_average += sensors[i] * sensor_mults[i];
      sensors_sum += sensors[i];
    }

    if(sensors_sum < 4300 && sensors_sum > 500){
      // Robot znajduje się na linii
      offroad_time = 0;

      // Wyliczenie pozycji linii
      position = long(sensors_average / sensors_sum);
      // Wyliczenie błędu
      error_value = calculate_error(position);
      // Wyliczenie prędkości silników
      calculate_motors_speed(error_value);
      // Odpowiednie sterowanie silników
      drive_motors(right_speed, left_speed);
    }else{
      // Robot znajduje się poza linią
      if(offroad_time >= Max_offroad_time){
        // Robot przekroczył czas który może być poza linią więc powinien się wyłączyć
        robot_running = false;
        shutdown();
        offroad_time = 0;
      }else{
        // Robot jeszcze nie przekroczył czasu który może być poza linią
        offroad_time += Delay_time;
      }
    }
  }

  delay(Delay_time);
}
```

Kosztorys

Nazwa	Ilość	Cena	Łącznie	Link
BOTLAND				
silnik dc	2	25.90	51.80	link
wytrawiacz	1	6.50	6.50	link
wywoływacz	1	3.50	3.50	link
mocowania do silników	2	6.95	13.90	link
nakrętki opakowanie	1	3.90	3.90	link
śrubki, nakrętki opakowanie	2	1.40	2.80	link
wiertło (1 mm)	10	0.50	5.00	link
wiertło 0,7mm	10	0.95	9.50	link

Nazwa	Ilość	Cena	Łącznie	Link
kondensator 10uf spolaryzowany	20	0.10	2.00	link
kondensator 22pf	10	0.10	1.00	link
pakiet lipo 2s	2	34.90	69.80	link
kondensator 100nF	10	0.10	1.00	link
kulka ball caster	1	13.00	13.00	link
przewody zestaw	1	6.50	6.50	link
rezystor 10k 5%	10	0.19	1.90	link
rezystor 220R 5%	10	0.19	1.90	link
cyna	1	6.90	6.90	link
plecionka	1	5.90	5.90	link
topnik	1	4.70	4.70	link
kondensator 100uf	15	0.15	2.25	link
dioda led	1	1.90	1.90	link
Rezonator kwarcowy 16MHz	3	0.70	2.10	link
TME				
sterownik silników toshiba tb6612fng	1	11.37	11.37	link
mikrokontroler ATmega328p-pn	1	15.77	15.77	link
laminat	1	26.08	26.08	link
transoptor odbiciowy TCRT5000	5	5.01	25.05	link
stabilizator napięcia LM7805	1	8.50	8.50	link
Dioda 1n5822RL shottky	5	0.88	4.40	link
koła - o-ring 17x3	20	0.39	7.80	link
mikroprzełącznik	4	0.79	3.16	link
styk jst	20	0.37	7.40	link
bezpiecznik polimerowy 1.6A	3	1.45	4.35	link
gniazdo jst xh	10	0.62	6.20	link
wtyk jst xh	10	0.37	3.70	link
INNE				
wiertło 3mm	2	3.00	6.00	-
taśma izolacyjna	1	8.00	8.00	-
brzeszczot	2	5.00	10.00	-

Nazwa	Ilość	Cena	Łącznie	Link
papier ścierny p120	1	2.00	2.00	-
papier ścierny p1000	1	2.00	2.00	-
kątownik aluminiowy	1	10.00	10.00	-
wydruki	3	4.00	12.00	-
aceton	1	5.00	5.00	-
worek próżniowy	1	8.00	8.00	-
ZAPASY DOMOWE				
koła - wałek aluminiowy fi20	1	0.00	0.00	-
dioda 1n4007	1	0.00	0.00	-
rezystor 10k 1%	2	0.00	0.00	-
rezystor 100R 1%	2	0.00	0.00	-
rezystor 4.7k 5%	2	0.00	0.00	-
wiertło 1.5mm	1	0.00	0.00	-
wiertło 3.2mm	1	0.00	0.00	-
kołki mocujące - wałek aluminiowy fi15	1	0.00	0.00	-
płaskownik węglowy	1	0.00	0.00	-
		Całkowita suma:	404.53	

Film

Link do filmu na youtube:
[ASKs 8 - Line Follower]

Inne uwagi

- Kondensator C5 został źle dobrany i finalnie został zdemontowany (uniemożliwiał programowanie mikrokontrolera, za wysoka pojemność uniemożliwia pojawienie się stanu niskiego na pinie RST w odpowiednim czasie. Jego brak nie wpływa negatywnie na działanie systemu. Podczas projektowania został dodany ze względu na obawę przed zakłóceniami spowodowanymi obciążeniem indukcyjnym.

- Błędnie dobrany footprint LM7805

- Montaż układu w obudowie SSOP24 wymagał wysokiej precyzji wykonania PCB. Z tego względu zdecydowaliśmy się na technologię fotochemiczną

- Układ stabilizatora LM7805 powinien być zastąpiony stabilizatorem typu LDO, znacznie wydłużyłoby to czas działania układu na zasilaniu akumulatorowym

- Silniki n20 wykazują się niską jakością wykonania, podczas testowania jeden z użytych wymagał wymiany szczotki.

- uchwyt wiertarski dla wierteł od 1.5mm jest w stanie “złapać” wiertło 0.7mm pogrubione opaską termokurczliwą, jednak obniża to jakość wiercenia

- korzystając z wyższych taktowań zegara wymagane jest obniżenie taktowania przetwornika adc w celu zachowania dokładności.