Rzeszów, 18.05.2024r.

Projekt na zaliczenie przedmiotu “Elektronika dla Informatyków”

Robot do śledzenia linii na bazie Arduino

Michał Kumięga

177114

1EF-DI

1.Wstęp

1.1 Głównym założeniem projektu jest wykorzystanie płytki Arduino do sterowania robotem do śledzenia czarnej linii. Dodatkowo ma być kontrolowany za pomocą Bluetooth.

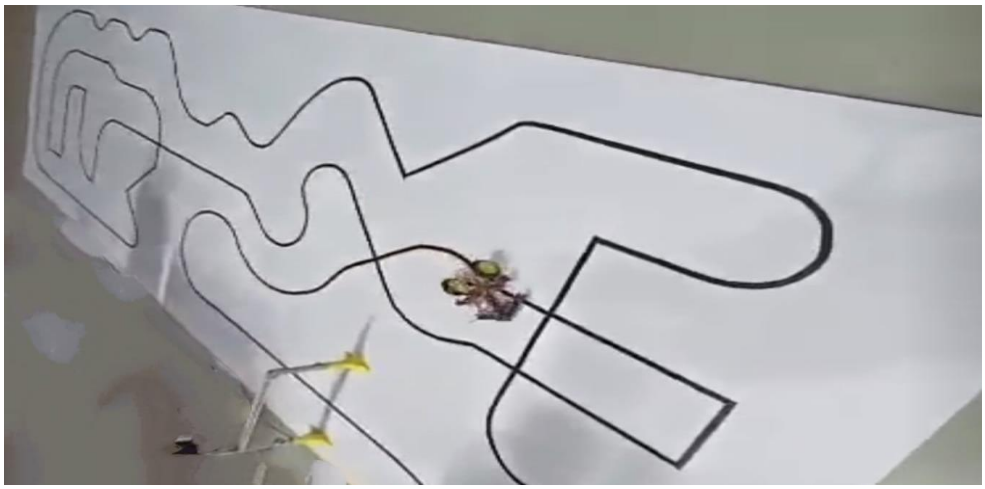
1.2 Czym jest robot klasy line follower?

Są to roboty, gdzie wszystkie jego elementy – elektroniczne i mechaniczne są zoptymalizowane pod kątem jednego zadania: pokonanie trasy z czarnej linii na białym podłożu (lub odwrotnie) w jak najkrótszym czasie.

Robot tej klasy posiada następujące cechy:

- Mała waga, bezwładność
- Duże przyspieszenie
- Zwrotność
- Dokładność
- Możliwość dostosowania do różnych tras
- Zdalny start/stop
- Pełna autonomia ruchu, brak ingerencji w zachowanie maszyny z zewnątrz

1.3 Opis przykładowej trasy



Trasa zbudowana jest najczęściej z białych płyt meblowych lub hdf, na których wyklejona jest czarna linia o szerokości ok. 19mm (szerokość taśmy izolacyjnej).

Na trasie mogą znaleźć się proste linie, skręty 90 stopni, skrzyżowania prostopadłe (które robot najczęściej przejeżdża prosto). Promień każdego zakrętu nie może być mniejszy niż krótszy bok kartki A4.

Na szybkość pokonania trasy wpływa głównie czystość nawierzchni i ilość ciasnych zakrętów.

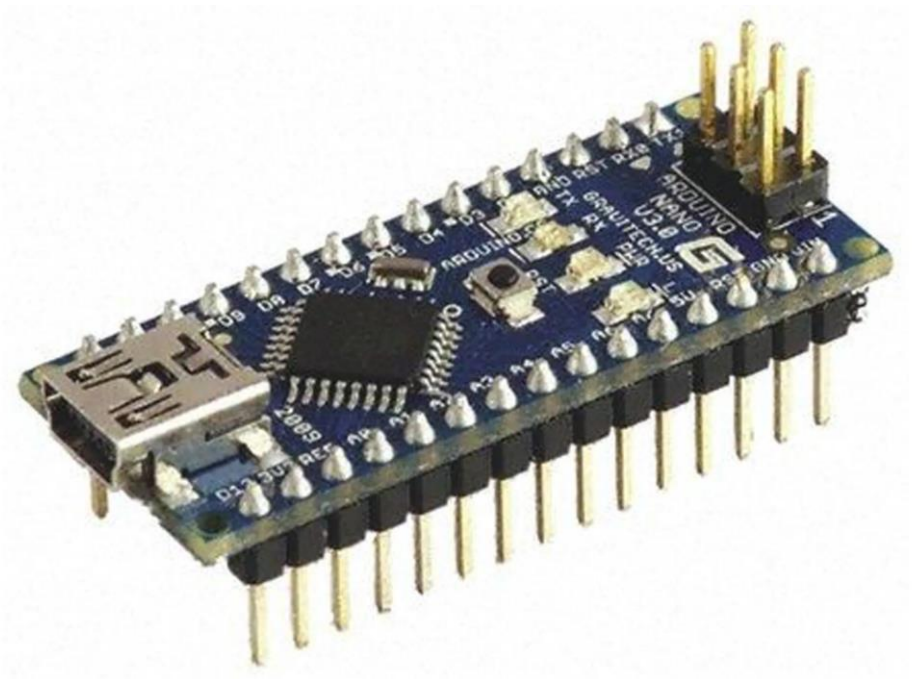
2. Realizacja projektu

2.1 Najważniejsze elementy użyte w projekcie:

- Arduino Nano 16Mhz (ATmega 328P)
- Silniki N20 z zintegrowanymi przykładniami 10:1 3000RPM
- Moduł Bluetooth HC-05
- Listwa z czujnikami odbiciowymi QTR-8RC
- Dwukanałowy sterownik silników DRV8835
- Stabilizator napięcia 5V L7805CV

2.2 Opis elementów:

Arduino Nano



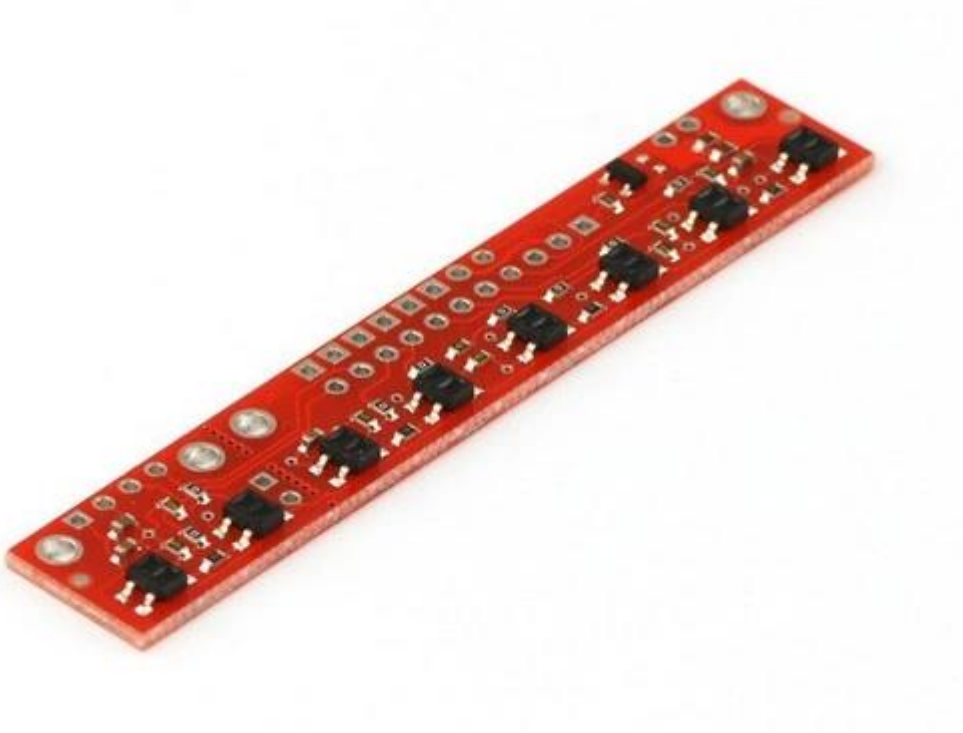
Źródło: https://pl.rs-online.com/web/p/arduino/6961667

Arduno Nano to jedna z najpopularniejszych płytek firmy, zbudowana w oparciu o mikrokontroler ATMEGA328P.

Najważniejsze funkcje pod względem projektu:

- praca na poziomie logicznym 5V, zgodnym z resztą komponentów, jednakże nie odpowiednim do bezpośredniego zasilania silników, ze względu na małe natężenie prądu.
- 6 pinów PWM do sterowania zewnętrznymi komponentami.
- 14 portów wejścia/wyjścia
- Możliwość komunikacji szeregowej
- port USB mini typ B do łatwego wgrywania programów

Listwa odbiciowa QTR-8RC



Źródło: https://botland.com.pl/czujniki-odbiciowe/20-listwa-z-czujnikami-odbiciowymi-qtr-8rc-cyfrowa-pololu-961-5903351249287.html

QTR-8RC:

- Zasilanie +5V lub +3,3V (wykorzystano 5V)
- Prąd zasilania 100mA
- Cyfrowy sygnał wyjściowy
- Dokładne pomiary w odległości 3mm-9mm od podłoża
- Masa 3g

Listwa składa się z 8 par czujników odbiciowych (dioda podczerwona i fototranzystor). Odczyt odbywa się przez pomiar czasu rozładowania kondensatora połączonego z fototranzystorami. Pozwala to na nie korzystanie z przetwornika A/C na płytce Arduino. Dodatkowo producent – firma Pololu stworzyła bibliotekę, która pozwala na łatwe korzystanie z listwy.

Zasada działania:

- Włączenie diód IR
- Ustawienie wyprowadzeń I/O mikrokontrolera
- Odczekanie przynajmniej 10 us
- Ponowne ustawienie wyprowadzeń I/O Arduino
- Pomiar czasu rozładowania kondensatora
- Cykl od nowa

Sterownik DRV8835

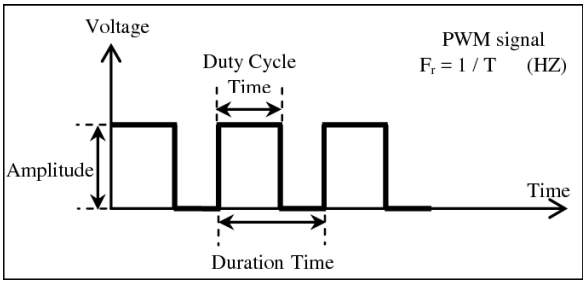
Źródło: https://botland.com.pl/sterowniki-silnikow-dc/851-drv8835-dwukanalowy-sterownik-silnikow-11v12a-pololu-2135-5904422367220.html

Moduł sterownika korzystający z układu DRV8835 firmy Texas Instruments. Jest to dwukanałowy sterownik silników DC.

Cechy:

- Bardzo małe wymiary - mieści się w monecie 5zł
- Może kontrolować napięcie z zakresu 2V-11V
- Pozwala na ciągły pobór prądu do 1,2A na kanał
- 2 kanały - idealne do stworzenia tego projektu
- Zabezpieczenie prądowe i napięciowe
- Praca na napięciu logicznym 5V
- Masa – 0,5g

- sterowanie silnikami za pomocą wypełnienia sygnałem PWM (Pulse Width Modulation).



Źródło: https://www.researchgate.net/figure/PWM-signal-with-its-two-basic-time-periods_fig4_271437313

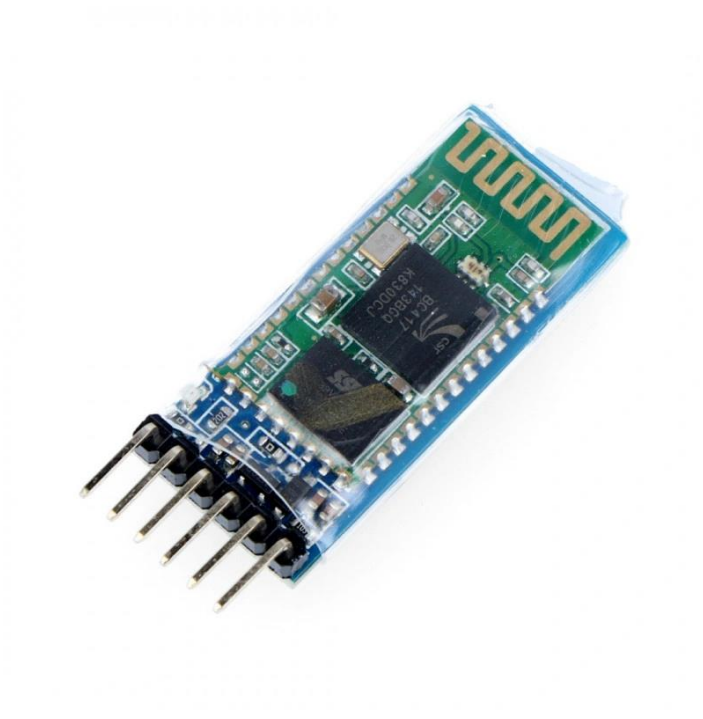
Opis pinów:

- VIN - napięcie zasilania silników
- GND potencjał masy całego układu
- VCC – zasilanie części logicznej układu
- AOUT1, AOUT2 - wyjście kanału 1
- BOUT1, BOUT2 - wyjście kanału 2
- AIN1, AIN2 - wejścia sterujące kanałem 1
- BIN1, BIN2 - wejścia sterujące kanałem 2
- MODE - wejście logiczne – w tym projekcie podłączone do masy, tabela poniżej wyjaśnia jego działanie

xIN1	xIN2	xOUT1	xOUT2	Tryb pracy
0	0	OPEN	OPEN	Wyjścia wyłączone.
0	PWM	L	PWM	Obroty w tył przy prędkości określonej: PWM %
PWM	0	PWM	L	Obroty w przód przy prędkości określonej: PWM %
PWM	1	L	PWM	Obroty w tył przy prędkości określonej: 100% – PWM %
1	PWM	PWM	L	Obroty w przód przy prędkości określonej: 100% – PWM %
1	1	L	L	Hamowanie (wyjścia podłączone do masy)

Źródło: https://botland.com.pl/sterowniki-silnikow-dc/851-drv8835-dwukanalowy-sterownik-silnikow-11v12a-pololu-2135-5904422367220.html

Moduł Bluetooth HC-05



Źródło: https://rc-planeta.pl/pl/komunikacji-wifi/226-modul-bluetooth-hc-05-masterslave-5297773478289.html

Prosty układ Bluetooth korzystający z komunikacji szeregowej UART.

Cechy:

- napięcie zasilania 5V
- Pracuje zarówno na stanach logicznych 3,3V jak i 5V
- Pobór prądu ok. 50mA
- Moc nadawania +4dBm - zasięg ok. 20m
- Standard Bluetooth kompatybilny od 2.0 wzwyż
- Małe wymiary – 37x16mm

Korzystanie:

- Należy na płytce ustawić te samą prędkość transmisji, w tym przypadku 9600 bps
- Podłączamy piny RX i TX do ich odpowiedników na płytce Arduino
- Wysyłamy komendy za pomocą prostej komunikacji szeregowej

Opis pinów:

- Enable/Key - pin używany do przełączania między trybami pracy – komunikacja i AT. Domyślnie znajduje się w ustawieniu komunikacyjnym.
- Vcc – zasilanie modułu +5V
- GND – masa układu
- Tx – pin transmisyjny
- Rx – pin odbierający
- State - podłączony do diody LED w module

Do komunikacji używam prostych komend – start,stop oraz ustawiania poszczególnych wartości algorytmu PID. Do ustawiania wartości wykorzystałem metodę, w której najpierw wysyłam wartość zmiennej 0-100 a następnie jej mnożnik - 10^{0} , 10^{-1} , 10^{-2} itd. Pozwala to na używanie tylko zmiennych typu int, zamiast float.

Silniki N20

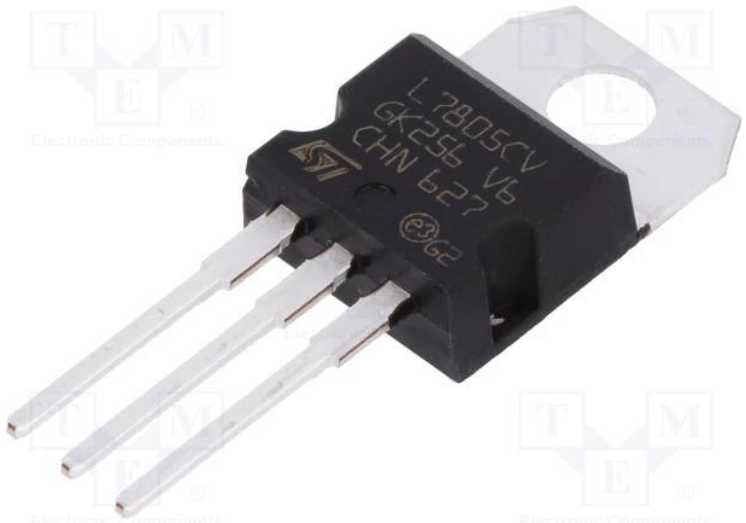
Źródło: https://botland.com.pl/silniki-micro-n20-seria-mp-medium-power/12550-silnik-n20-bt11-micro-51-3000rpm-9v-5904422341183.html

Najpopularniejsze silniki używane do robotów tej klasy używane przez zawodników na całym świecie.

Cechy:

- napięcie zasilania 3-9V
- prędkość bez obciążenia 3000obr/min
- moment obrotowy 0,029Nm (0,3kg*cm)
- przełożenie 10:1
- wał typu D o średnicy 3mm
- mała masa – 10g

Stabilizator napięcia 5V L7805CV



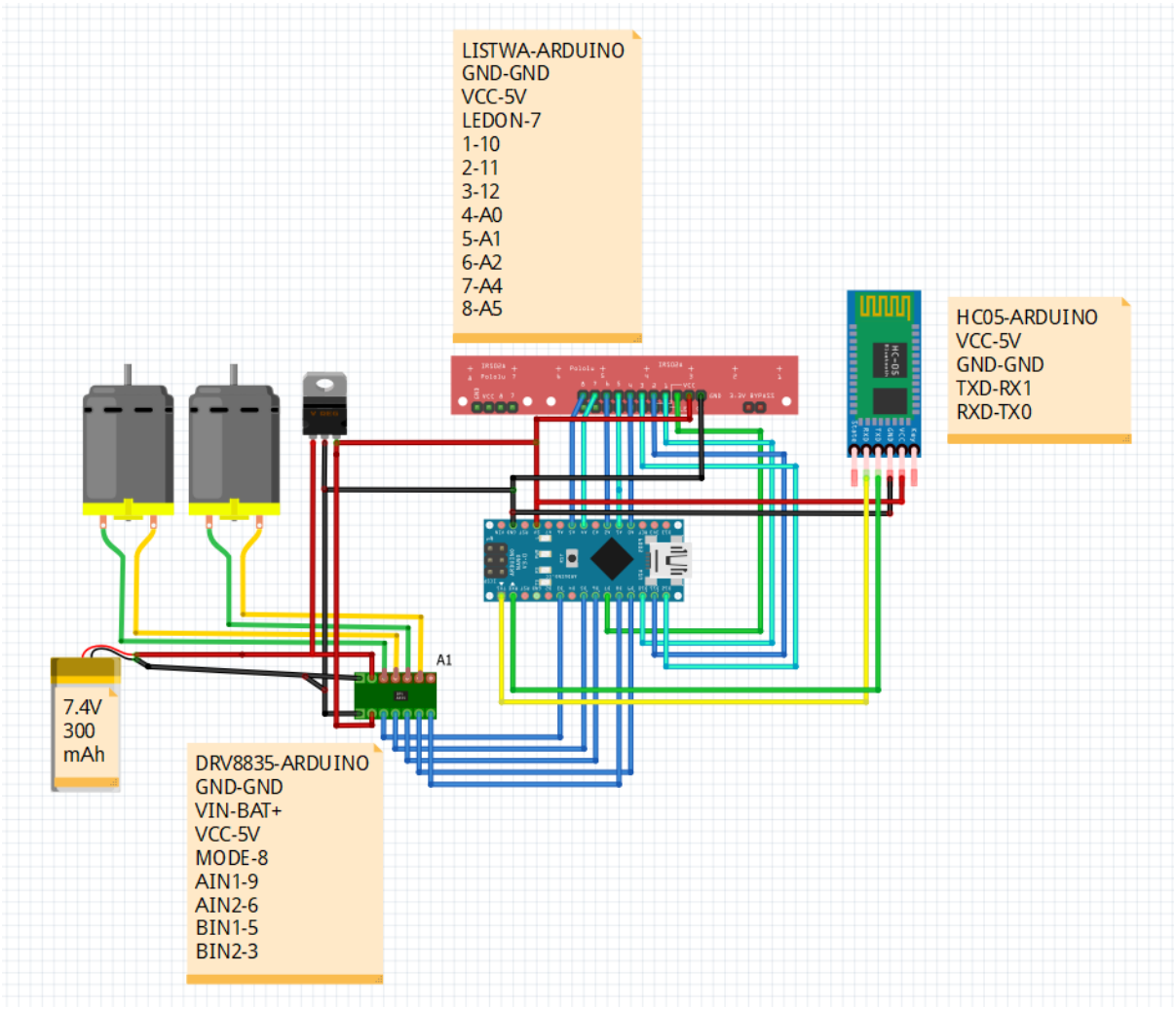
Źródło: https://www.tme.eu/pl/details/l7805cv/stabilizatory-napiecia-nieregulowane/stmicroelectronics/

Cechy:

- napięcie wejściowe 7-25V
- napięcie wyjściowe 5V
- prąd wyjściowy 1,5A

2.2 Budowa

2.2.1 Schemat połączeń



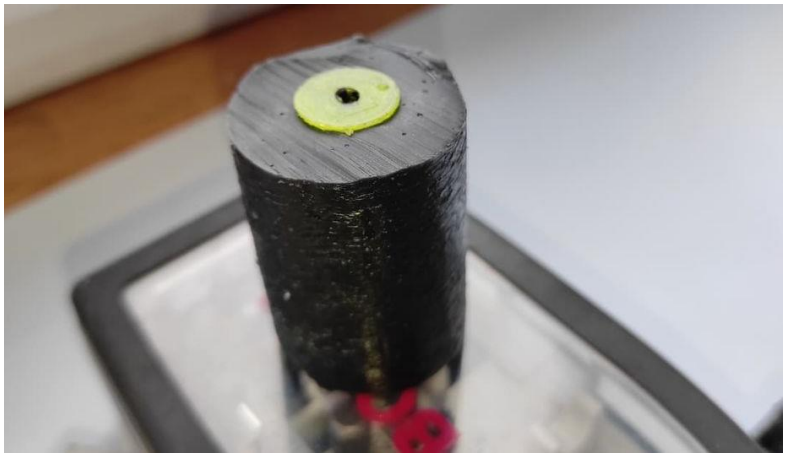
2.2.2 Składanie konstrukcji

Na początku zająłem się kółkami. Odlane zostały z silikonu Gumosil-E firmy SiP.

Mieszanie Gumosilu-E z katalizatorem OL-1.

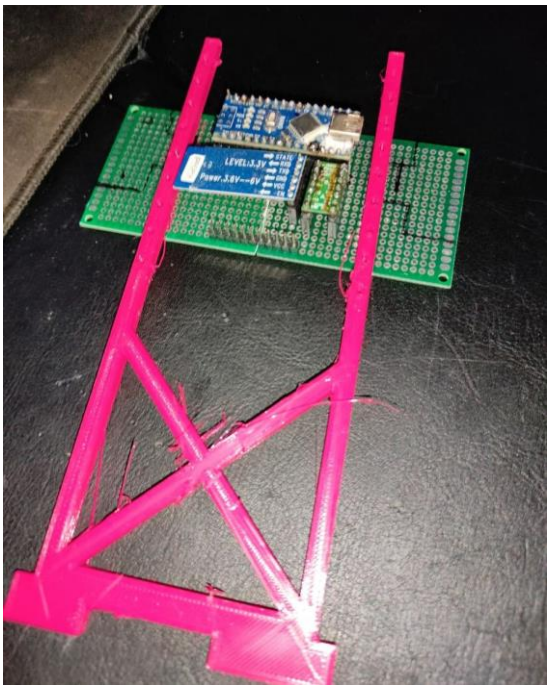
Domowa stacja do odpowietrzania odlewów.

Odpowietrzanie zalanych form przygotowanym silikonem w próżni.



Efekt końcowy po wyschnięciu silikonu (czas schnięcia min. 24h).

Następnie przyszła pora na lutowanie wszystkich elementów względem poprzedniego schematu.

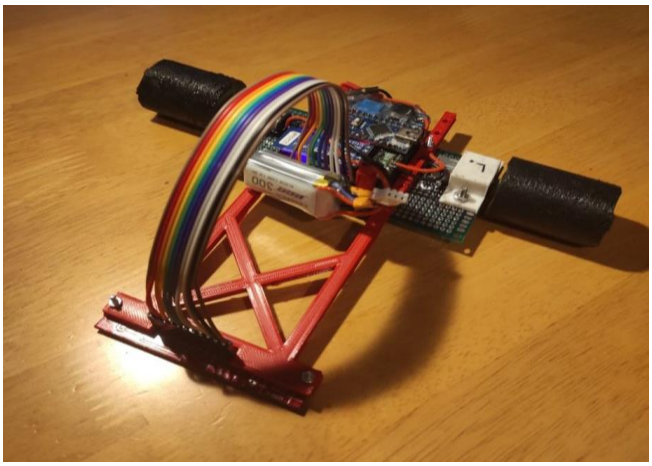


Wstępne ustawienie wszystkich elementów na swoich miejscach.

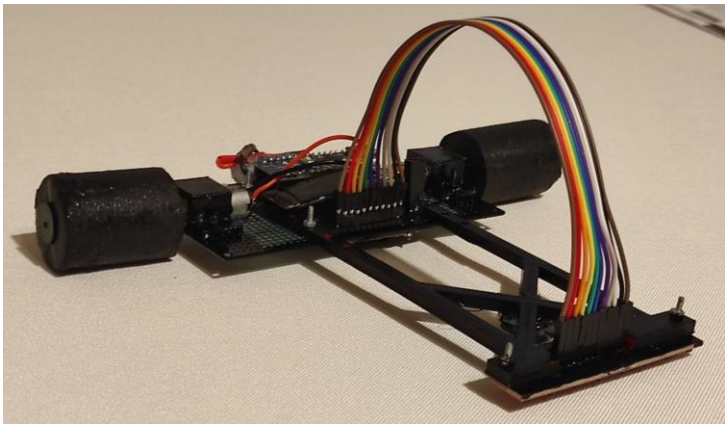
Do przymocowania listwy do reszty konstrukcji wydrukowałem na drukarce 3D ramię, sięgające do przodu robota.

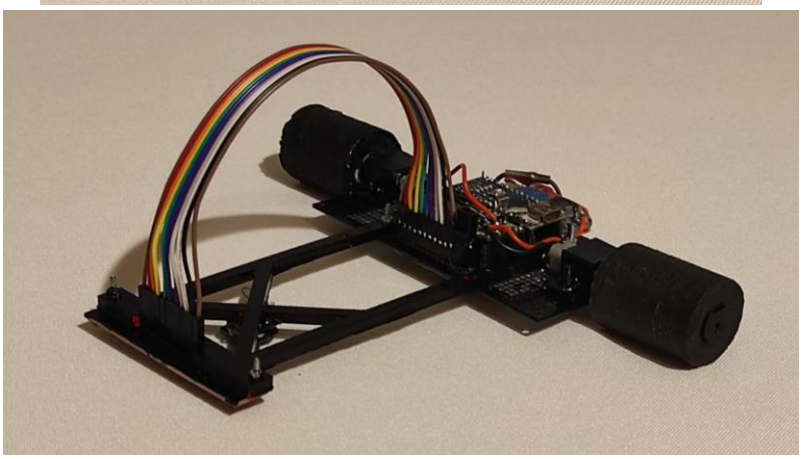
Dokonanie wszystkich połączeń pomiędzy elementami.

Podczas tego procesu, należy zachować szczególną ostrożność, by nigdzie nie spowodować zwarcia. Dodatkowo przewody komunikacyjne pomiędzy modułem HC-05 a Arduino należało poprowadzić w taki sposób, by blisko zamontowany sterownik silników nie powodował zakłóceń.



Skończona konstrukcja.





Finalna pomalowana konstrukcja.

3. Kod

3.1

Najważniejszą częścią robota typu line follower jest algorytm PID (proporcjonalno-całkująco-różniczkujący) i reszta programu sprawiająca, że nasza konstrukcja będzie poprawnie śledzić linię. Oprócz prostego śledzenia linii, należało rozważyć również kilka przypadków szczególnych, gdy robot zgubi linię podczas pokonywania ostrego zakręty czy podczas przejeżdżania skrzyżowania prostopadłego. Do zaprogramowania Arduino Nano wykorzystałem Arduino IDE i język C.

Implementacja algorytmu w kodzie znajduje się poniżej:

Przed rozpoczęciem działania robot kalibruje swoją listwę odbiciową względem podłoża, pozwala to na maksymalną eliminację błędów odczytu czarnej linii na białym podłożu. Firma Pololu uwzględniła tę funkcję w swojej bibliotece QTRSensors. Robot przez 10 sekund pobiera próbki czarnego i białego koloru - światło podczerwone będzie słabiej odbijane przez ciemny kolor niż jasny. Rozpoczęcie jak i zakończenie kalibracji sygnalizowane jest poprzez brzęczyk oraz wbudowaną diodę na płytce - zostały dodane dla lepszej interakcji użytkownika z konstrukcją.

```
void calibration() {
  digitalWrite(LED_BUILTIN, HIGH);
  digitalWrite(A3, HIGH);
  delay(15);
  digitalWrite(A3, LOW);
  for (uint16_t i = 0; i < 400; i++)
  {
    qtr.calibrate();
  }
digitalWrite(LED_BUILTIN, LOW);//to tylko zeby wiedziec kiedy zaczyna i konczy sie kalibracja
  digitalWrite(A3, HIGH);
  delay(15);
  digitalWrite(A3, LOW);
  delay(200);
  digitalWrite(A3, HIGH);
  delay(15);
  digitalWrite(A3, LOW);
}
```

```
void loop() {
  if (Serialq.available()) { //jezeli sie polaczy
    while(Serialq.available() == 0);
    odczyt_wart();
    odczyt();
  }
  if(onoff == 1) {
    speed_set();
  }
  if(onoff == 0) {
    przod(0, 0);
  }
}
```

Pętla główna jest krótka, co pozytywnie wpływa na szybkość działania całego programu. Jeżeli spełniony zostanie warunek Serialq.available() == 0; to robot będzie w tym czasie odczytywać i zapisywać przesyłane wartości. Odebrana może zostać wartość zmiennej onoff, wtedy robot rozpocznie przejazd bądź go zakończy.

```
void speed_set() {
  uint16_t position = qtr.readLineBlack(sensorValues);
  int error = 3500 - position;



  int cnt = 0;
  float sum = 0;
  for (int i = 0; i < 8; i++) {
    if(sensorValues[i] >= threshold) {
      cnt++;
      sum = sum + i;
    }
  }

//przypadki na przekroczenie linii
  if (cnt >= 3) {
    int motorspeeda = 0;
    int motorspeedb = 0;
    int val = sum/cnt;
    if(val < 3.5) {
      //prawo
      //przod(0,100);
      prawo(200, 200);

    }
    if(val > 3.5) {
      //lewo
      //przod(100,0);
      lewo(200, 200);

    }
    if(val == 3.5) {
      cnt = cnt/2;
      uint16_t mini = 1000;
```

Funkcja **speed_set** steruje ruchem robota na podstawie odczytów z czujników liniowych.
Najpierw odczytuje pozycję linii i oblicza błąd względem wartości środkowej. Następnie zlicza aktywne czujniki i oblicza średnią pozycję aktywnych czujników. Jeśli czarna linia nie znajduje się w pozycji środkowej, robot wykonuje odpowiedni skręt w prawo lub lewo, w zależności od średniej pozycji aktywnych czujników.
Jeśli wartość wynosi dokładnie 3.5, wybiera najciemniejszy czujnik w określonym zakresie, by zdecydować o kierunku skrętu. Jeśli aktywnych czujników jest mniej niż trzy, funkcja wywołuje regulację PID, aby skorygować pozycję robota na linii.

```
    uint8_t minpos = 0;
    for (int i = 4 - cnt; i <= 3 + cnt; i++) {
      if (mini > sensorValues[i]) {
        mini = sensorValues[i];
        minpos = i;
      }
    }
    if(minpos < 3.5) {
      //prawo
      //przod(0,100);
      prawo(200, 200);
    }
    if(minpos > 3.5) {
      //lewo
      //przod(100,0);
      lewo(200, 200);
    }
  }
  else {
    PID(error);
  }
}
```

```
void PID(int error) {
  int P = error;
  int I = I + error;
  int D = error - lastError;
  lastError = error;
  Pvalue = (Kp/pow(10,multiP))*P;
  Ivalue = (Ki/pow(10,multiI))*I;
  Dvalue = (Kd/pow(10,multiD))*D;

  float motorspeed = Pvalue + Ivalue + Dvalue;

  int motorspeeda = basespeeda + motorspeed;
  int motorspeedb = basespeedb - motorspeed;

  if (motorspeeda > maxspeeda) {
    motorspeeda = maxspeeda;
  }
  if (motorspeedb > maxspeedb) {
    motorspeedb = maxspeedb;
  }
  if (motorspeeda < minspeeda) {
    motorspeeda = minspeeda;
  }
  if (motorspeedb < minspeedb) {
    motorspeedb = minspeedb;
  }
  speedcontrol(motorspeeda, motorspeedb);
}
```

Funkcja **PID** implementuje algorytm regulacji PID (Proporcjonalno-Całkowo-Różniczkowy) w celu skorygowania prędkości silników robota na podstawie błędu pozycji. Najpierw oblicza składowe P, I i D (proporcjonalną, całkową i różniczkową) błędu, a następnie sumuje ich wartości przeskalowane współczynnikami Kp, Ki i Kd. Wynikowa wartość określa prędkość silników. Funkcja także ogranicza prędkości do maksymalnych i minimalnych wartości oraz ustawia prędkości silników za pomocą funkcji `speedcontrol`.

```
void speedcontrol(int mota, int motb) {
  if (mota >= 0 && motb >= 0) {
    przod(mota, motb);
  }
  if (mota < 0 && motb >= 0) {
    //dreapta
    mota = 0 - mota;
    prawo(mota, motb);

  }
  if (mota >= 0 && motb < 0) {
    //stanga
    motb = 0 - motb;
    lewo(mota, motb);
  }
}
```

Funkcja **speedcontrol** steruje ruchem robota na podstawie podanych prędkości dla dwóch silników. W zależności od znaków prędkości, ustawia robota na jazdę do przodu, skręt w prawo, lub skręt w lewo, wywołując odpowiednie funkcje sterujące (przod, prawo, lewo).

```
void odczyt_wart() {
  val = Serialq.read();
  cnt++;
  v[cnt] = val;
  if (cnt == 2)
    cnt = 0;
}

//przypisywanie otrzymanych pakietow ze stringa
void odczyt() {
  int a = v[1];
  if (a == 1) {
    Kp = v[2];
  }
  if (a == 2) {
    multiP = v[2];
  }
  if (a == 3) {
    Ki = v[2];
  }
  if (a == 4) {
    multiI = v[2];
  }
  if (a == 5) {
    Kd = v[2];
  }
  if (a == 6) {
    multiD = v[2];
  }
  if (a == 7) {
    onoff = v[2];
  }
}
```

Funkcja **odczyt_wart()** odczytuje dane z portu szeregowego i zapisuje je do tablicy v, resetując licznik cnt po osiągnięciu wartości 2. Funkcja odczyt() przypisuje wartości z tablicy v do odpowiednich zmiennych sterujących robotem typu line follower (takich jak Kp, Ki, Kd, ich mnożniki, oraz onoff), na podstawie identyfikatora a w v[1]. Taki typ komunikacji pozwala na uniknięcie przesyłania zmiennych typu float, zamiast tego używanie zmiennych typu int, co pozytywnie wpływa na

```
//konfiguracja sterownika silnikow
void przod(int posa, int posb) {
  digitalWrite(fazaA, LOW);
  digitalWrite(fazaB, HIGH);
  analogWrite(aenbl, posa);
  analogWrite(benbl, posb);
}
void lewo(int posa, int posb) {
  digitalWrite(fazaA, LOW);
  digitalWrite(fazaB, LOW);
  analogWrite(aenbl, posa);
  analogWrite(benbl, posb);
}
void prawo(int posa, int posb) {
  digitalWrite(fazaA, HIGH);
  digitalWrite(fazaB, HIGH);
  analogWrite(aenbl, posa);
  analogWrite(benbl, posb);
}
```

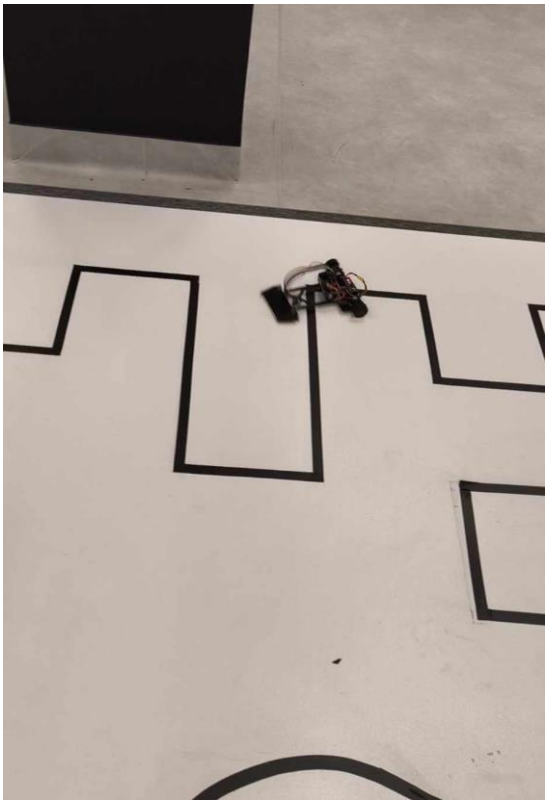
działanie kodu.

Funkcje **`przod()`**, **`lewo()`**, i **`prawo()`** sterują silnikami DC za pomocą sterownika DRV8835. Funkcja `przod()` ustawia fazę A na LOW, fazę B na HIGH oraz prędkości silników `posa` i `posb` do jazdy do przodu. Funkcja `lewo()` ustawia fazę A i fazę B na LOW oraz prędkości silników `posa` i `posb` do skrętu w lewo. Funkcja `prawo()` ustawia fazę A i fazę B na HIGH oraz prędkości silników `posa` i `posb` do skrętu w prawo.

4. Testy

Pierwsze testy zostały przeprowadzone na krótkiej trasie celem sprawdzenia poprawności wykonanych połączeń i napisanego kodu.

Następnie przetestowany został na trasie z edycji 2023 konkursu XChallenge, gdzie dopracowałem parametry algorytmu PID. Najlepszym czasem przejazdu było 19 sekund. Z testów wynikło, że wartość D powinna być ok. 8-11razy większa od wartości P. Wartości I została ustawiona na 0,0001, P na 0,78 a D na 7,5. Wartości zostały dopasowane akurat do mojej konstrukcji oraz nawierzchni.



Pierwszy prawdziwy test przeszedł podczas konkursu Robotic Tournament 2024 odbywającego się 23 marca w Rybniku. Zajął wówczas 4 miejsce z czasem 6,4 sekundy w swojej kategorii. Od czasu konkursu zawodów konstrukcja zostało zoptymalizowana

pod kątem kodu, długości ramienia z czujnikami, masy (mniejsza bateria) i lepszej przyczepności (czyszczenie kółek alkoholem izopropylenowym przed każdym przejazdem).

Trasa

5. Wnioski

Projekt ten wymagał przede wszystkim wzięcia wielu tematów pod uwagę takich jak mechanika, dobór materiałów, dobranie odpowiednich komponentów, projektowanie, rozwiązywanie problemów w trakcie budowy i testowania. Istnieje również dużo możliwości wprowadzenia dodatkowych funkcji, takich jak sterowanie silnikami za pomocą enkoderów a nie wypełnieniem PWM, co przełożyłoby się na dokładniejszą reakcję na błędy. Dodatkowo zaprojektowanie płytki PCB z wszystkimi układami na niej drastycznie zmniejszyłoby masę i moment potrzebny do skręcania. Własny projekt listwy odbiciowej również pomógłby znacząco w wykrywaniu linii, jedynym ograniczeniem byłaby ilość pinów I/O mikrokontrolera. Użycie mikrokontrolera np. STM32F401CCU6 o szybszym zegarze 96MHz wpłynęłoby na większą dokładność w przetwarzaniu danych w porównaniu do Arduino Nano v3.