

Politechnika Warszawska

W Y D Z I A Ł E L E K T R O N I K I
I T E C H N I K I N F O R M A C Y J N Y C H



Instytut Automatyki i Informatyki Stosowanej
na kierunku Informatyka
w specjalności Inżynieria Oprogramowania

Wstęp do Robotyki - Laboratoria

Bartłomiej Krawczyk

Numer albumu 310774

Mateusz Brzozowski

Numer albumu 310608

WARSZAWA 2023

Spis treści

1. Zadania	3
1.1. Zadanie 1 - Podążanie wzdłuż linii (Linefollower)	3
1.2. Zadanie 2 - Transporter	3
2. Założenia	4
3. Przygotowanie do pracy	4
4. Realizacja	7
4.1. Mechanika	7
4.2. Schemat	7
4.3. Podstawa matematyczna	8
4.4. Rozwiążanie bazujące na PID	8
4.4.1. Wpływ parametrów PID	8
4.5. Dobieranie parametrów	8
4.6. Wyniki	9
4.6.1. Wnioski	9
4.7. Line Follower	9
4.7.1. Parametry	9
4.8. Transporter	10
4.9. Tor	11
5. Budowa robota	12
5.1. Zawody	12
5.1.1. Robot - iteracja I	12
5.1.2. Robot - iteracja II	14
5.2. Line Follower	16
5.3. Transporter	16
5.4. Działający robot	16
6. Implementacja	17
6.1. Kod bazowy	17
6.2. Kod Line Follower'a	18
6.2.1. Kod "Naiwny"	18
6.2.2. Kod bazujący na PID	19
6.3. Kod Transporter'a	20
Spis rysunków	26
Spis tabel	26
Fragmenty kodu	27

1. Zadania

Na pierwszych zajęciach prowadzący przekazał nam instrukcję z zadaniami jakie mieliśmy wykonać w trakcie trwania laboratoriów, następnie szczegółowo wytłumaczył nam na czym polegają poszczególne zadania.

1.1. Zadanie 1 - Podążanie wzduż linii (Linefollower)

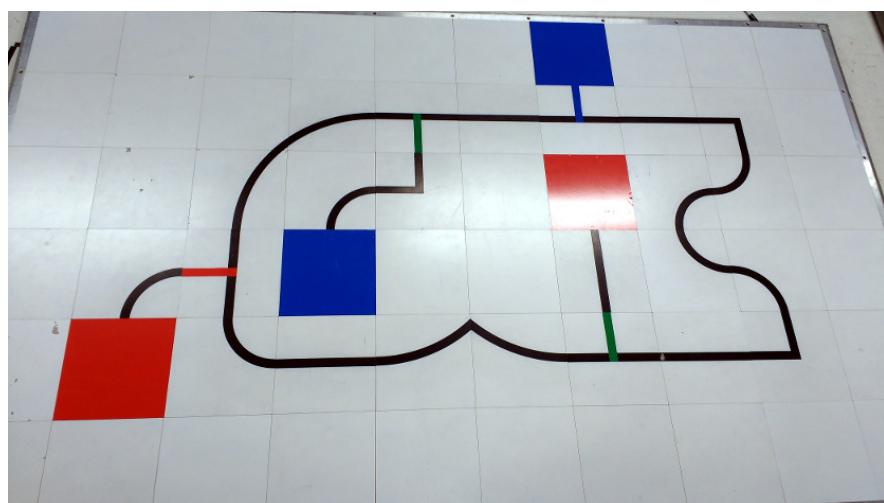
Zadaniem robota jest przejechanie całej trasy po wyznaczonej linii.



Rysunek 1.1. Przykładowa trasa do podążania za wyznaczoną linią

1.2. Zadanie 2 - Transporter

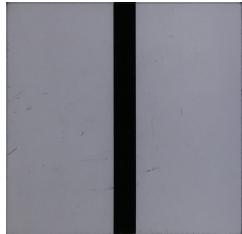
Zadaniem robota jest przetransportowanie obiektów z punktów bazowych do punktów docelowych. Punkt bazowy i punkt docelowy oznaczone są przez kolorowe elementy planszy. Rozwidlenie do opowiedniego koloru jest zaznaczone na czarnej linii trasy.



Rysunek 1.2. Przykładowa plansza do zadania Transporter

2. Założenia

Trasa do zadania pierwszego jest czarną linią na białym tle. Linia składa się nie tylko z prostych odcinków, lecz na drodze mogą znajdować się różnego rodzaju trudne zakręty, skrzyżowania do pokonania, takie jak np. ostre zakręty, zaokrąglone zakaty, skrzyżowania, zakrzywione linie.



Rysunek 2.1. Prosta linia



Rysunek 2.2. Ostry zakręt



Rysunek 2.3. Zaokrąglony zakręt



Rysunek 2.4. Skrzyżowanie



Rysunek 2.5. Zakrzywiona linia

W ramach drugiego zadania trasa dodatkowo składa się z skrzyżowań odpowiednia oznaczonych wybranym kolorem. Kolory mogą się powtarzać, dlatego należy zapamiętywać stan robota. Trasa może dodatkowo zawierać np. kilka czerwonych skrzyżowań, zielonego skrzyżowania, czerwonej platformy, zielonej platformy z dłuższym dojazdem.

3. Przygotowanie do pracy

Przed przystąpieniem do rozwiązywania zadań otrzymaliśmy od prowadzącego laboratorium części robota LEGO Mindstorms Ev3, a także klocki LEGO, które posłużyły nam do zbudowania naszego robota.



Rysunek 2.6. Czerwone skrzyżowanie



Rysunek 2.7. Zielone skrzyżowanie



Rysunek 2.8. Czerwona platforma



Rysunek 2.9. Zielona platforma



Rysunek 3.1. Otrzymane elementy robota LEGO Mindstorms Ev3

3. Przygotowanie do pracy

Poszczególne elementy posiadają różne funkcjonalności, które po złożeniu w całość pomogły nam rozwiązać zadania.

Do zadania pierwszego wykorzystaliśmy następujące elementy robota takie jak:

Tabela 3.1. Otrzymane elementy robota - zadanie 1

Element	Zastosowanie	Zdjęcie
główna jednostka sterująca	Uruchamia program, odbiera i nadaje sygnały do poszczególnych czujników i silników	
2 x silnik napędowy do kół	Do silników były przymocowane koła, które umożliwiły poruszanie się robota	
2 x czujnik światła	Wykrywanie linii i kolorów jakie znajdowały się pod robotem	
przycisk	Uruchamianie i zatrzymywanie robota	

Posłużyły nam one do wykrywania linii i poruszania się wzdłuż niej. Jednakże w przypadku rozwiązywania kolejnego zadania, przedstawione elementy okazały się niewystarczające, ponieważ oprócz śledzenia musieliszy jeszcze wykrywać obiekt, podnosić go i opuszczać. Dlatego dodatkowo w przypadku rozwiązywania zadania 2 dołożyliśmy następujące elementy:

Tabela 3.2. Otrzymane elementy robota - zadanie 2

Element	Zastosowanie	Zdjęcie
serwomechanizm	Podnoszenie i opuszczanie obiektu	

Element	Zastosowanie	Zdjęcie
czujnik odległości	Wykrywanie w jakiej odległości znajduje się obiekt	

4. Realizacja

Początek pracy poświęciliśmy na zapoznanie się z wykładami udostępnionymi w ramach wykładów z przedmiotu Wstęp Do Robotyki. Dzięki temu zgłębiliśmy matematyczne podstawy tego w jaki sposób może poruszać się pojazd kołowy.

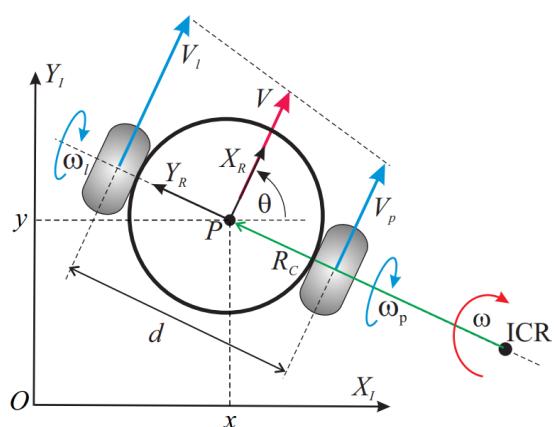
4.1. Mechanika

Robot z napędem różnicowym - dwa niezależnie napędzane koła stałe na jednej osi.

W celu zmiany położenia modyfikowaliśmy prędkości kół. Założyliśmy jedną prędkość podstawową do przodu oraz dodatkowo modyfikowaliśmy skręt odpowiednio modyfikując składowe prędkości poszczególnych kół.

Prędkość naszego robota do przodu wynikała ze średniej prędkości obu kół, a prędkość obrotu zależała od różnicy między prędkościami kół.

4.2. Schemat



Rysunek 4.1. Schemat

4.3. Podstawa matematyczna

- v_p, v_l - prędkość liniowa prawego, lewego koła
- ω_p, ω_l - prędkość kątowa prawego, lewego koła
- v, ω - prędkość liniowa i kątowa robota
- R_C - chwilowy promień skrętu robota
- d - rozstaw kół
- r - promień koła

$$v = \frac{v_l + v_p}{2}$$

$$\omega = \frac{v_l - v_p}{d}$$

$$R_C = \frac{v}{\omega} = \frac{d(v_l + v_p)}{2(v_l - v_p)}$$

4.4. Rozwiążanie bazujące na PID

4.4.1. Wpływ parametrów PID

- **Parametr P**
 - parametr brany z największą wagą
 - oznaczał chwilową różnicę w poziomie odbijanego światła odbieranego przez czujniki
- **Parametr I**
 - parametr brany z najmniejszą wagą (ponieważ integral był bardzo dużą liczbą w porównaniu do error oraz derivative)
 - integral przechowywał historię kilku ostatnich iteracji programu, przez co powodował, że jak wyjechaliśmy jedynie w niewielkim stopniu to skręt był niewielki, jednak gdy przez dłuższy czas czujniki wykrywały linię to poziom skrętu zwiększał się
- **Parametr D**
 - parametr wynikał z chwilowej zmiany między poziomem lewego i prawego czujnika
 - parametr miał największy wpływ w przypadku ostrych skrętów
 - parametr liczył się jedynie przy zmianie między ostatnimi błędami

4.5. Dobieranie parametrów

Parametry PID, które zastosowaliśmy w naszym robocie, zostały dobrane metodą inżynierską. Zastosowaliśmy podejście iteracyjne, w którym kolejne wartości parametrów były ustalane na podstawie wyników testów oraz obserwacji zachowania robota.

W zależności od rodzaju zadania, dla którego był przeznaczony robot, dobieraliśmy parametry PID w inny sposób. Na przykład, podczas zawodów głównie skupialiśmy się na zwiększaniu prędkości przy okazji odpowiednio modyfikując parametr D, ponieważ tor nie posiadał ostrych zakrętów. Zwiększałyśmy również parametr I, aby nasz robot na odcinkach prostych mógł osiągnąć jak największą prędkość. W przypadku parametrów D i I staraliśmy się dobrze odpowiednią wartość na podstawie doświadczenia.

W zadaniu Line Follower skupiliśmy się na dostosowaniu wartości parametrów P i D, ponieważ było tam dużo ostrych zakrętów. Zaczęliśmy od wartości, które sprawdziły się podczas zawodów, a następnie stosowaliśmy iteracyjną metodę doboru wartości parametrów, aż do osiągnięcia idealnych wartości, które okazały się takie same jak wartości początkowe. W tym przypadku jedyną zmianą, jaką wprowadziliśmy, było zmniejszenie prędkości ruchu robota.

W ostatnim zadaniu pozostawiliśmy parametry prawie takie same jak w poprzednim zadaniu, jednakże zmniejszyliśmy prędkość ruchu robota. W przypadku tego zadania dodatkowo musieliśmy wyznaczyć ilość obrotów kół jakie musi wykonać nasz robot aby wykonać pełen obrót o 360 stopni, tak aby w prosty sposób dokonywać obrotów w prawo/lewo, a także zawracania.

4.6. Wyniki

Tabela 4.1. Wyniki zawodów

Team	Round 1	Round 2	Round 3	Round 4	Round 5
Parostatek	-	28.01	-	-	29.77

4.6.1. Wnioski

- najczęściej było dobranie parametrów PID, tak aby robot jeździł z zadowalającą prędkością

4.7. Line Follower

4.7.1. Parametry

Zmniejszona prędkość względem zawodów, żeby wyrobić się na ostrych zakrętach: Podstawowe PID

Fragment kodu 4.1. Zmodyfikowane parametry - zadanie śledzenia linii

```

1 MIN_FORWARD_SPEED = 10
2 MAX_FORWARD_SPEED = 20
3
4 FORWARD_SPEED_CORRECTION = (

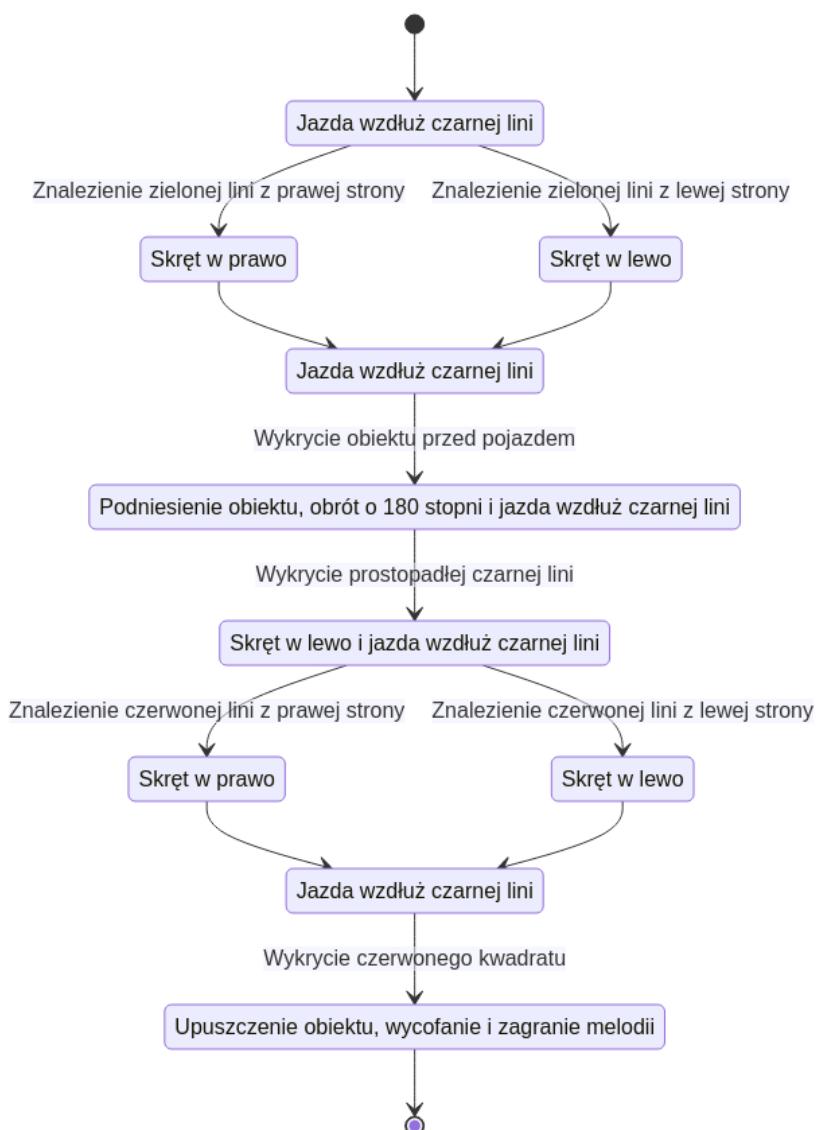
```

4. Realizacja

```
5      (MAX_FORWARD_SPEED - MIN_FORWARD_SPEED) / MAX_FORWARD_SPEED
6 )
7
8 CONSTANT_P = 4.0
9 CONSTANT_I = 0.01
10 CONSTANT_D = 4.0
11
12 HISTORY_LOSS = 0.5
13
14 AMPLIFIER = 0.25
```

4.8. Transporter

Aby przetransportować obiekt z jednego miejsca na drugi musielibyśmy na początku przeanalizować wszystkie możliwe stany, jakie mogą wystąpić w trakcie zadania. Tak więc, przygotowaliśmy schemat stanowy, który przedstawia to jak ma zachowywać się w danym momencie.



4.9. Tor

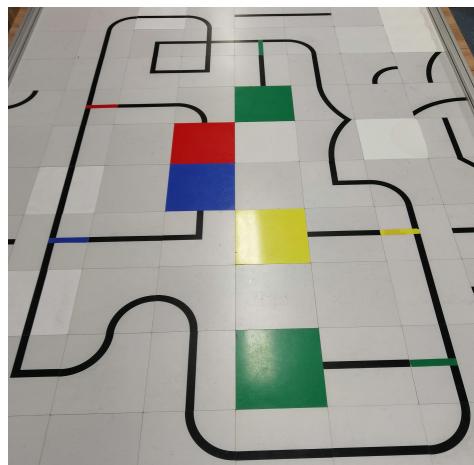
Do każdego rodzaju zadania z jakim musiał zmierzyć się nasz robot, przygotowywany był odpowiedni tor.



Rysunek 4.2. Tor na zawody



Rysunek 4.3. Tor - podążanie za linią



Rysunek 4.4. Trasa dla transportera

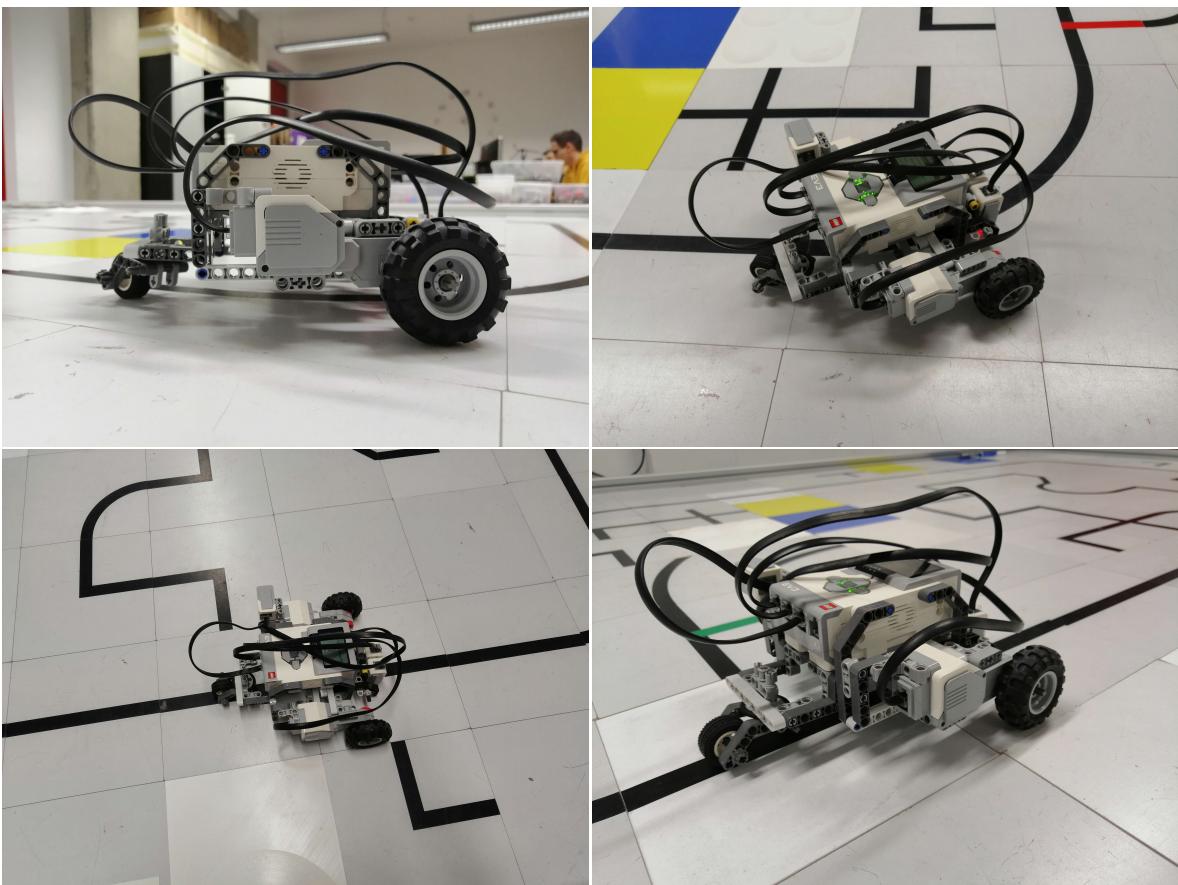
5. Budowa robota

5.1. Zawody

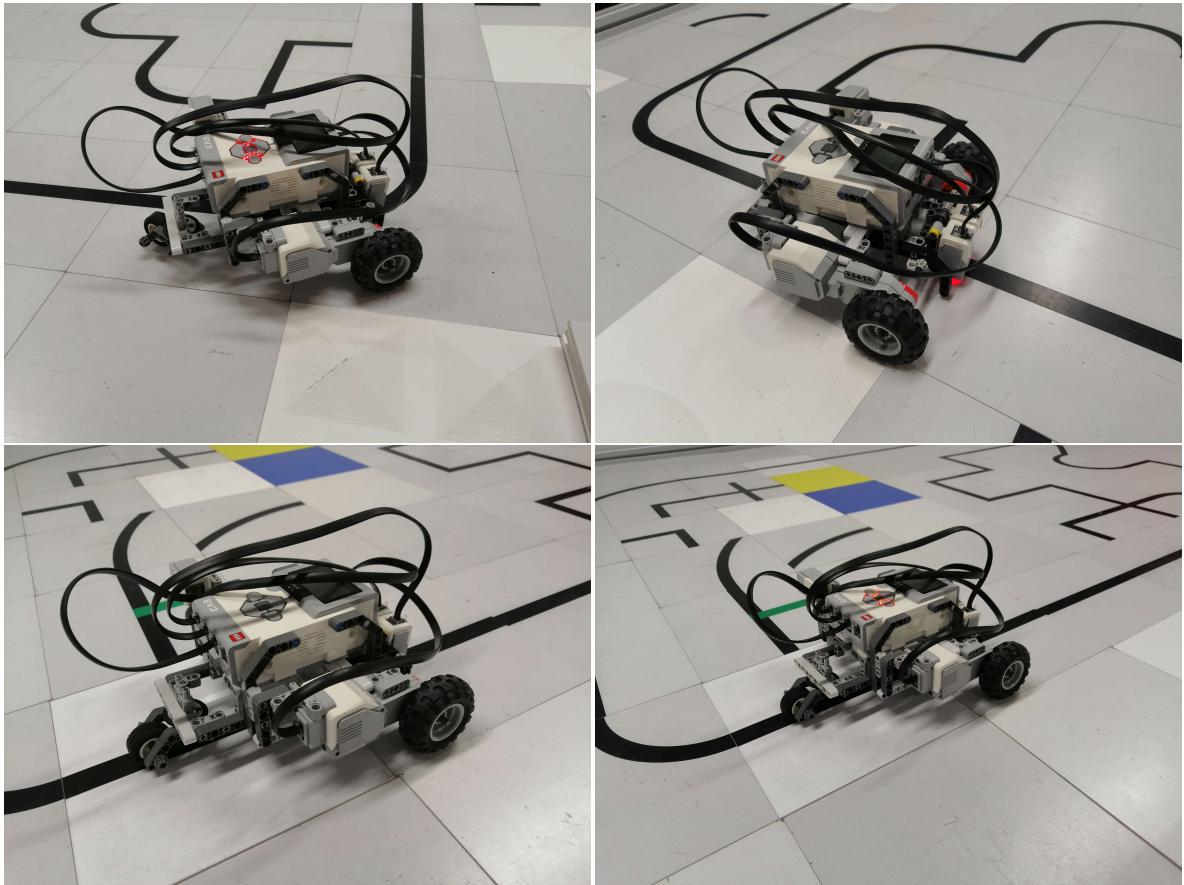
Wykorzystując otrzymane przez prowadzącego laboratoria elementy robota, a także udostępnioną pokaźną ilość klocków lego przystąpiliśmy do budowy robota.

5.1.1. Robot - iteracja I

Budowę naszego robota rozpoczęliśmy od zamontowania kół, dwóch z przodu robota, oraz jednego samonastawnego z tyłu. Posłużyła nam do tego dolna podstawa, zamontowana do robota, zwiększająca jego stabilność, a także umożliwiająca przemontowanie czujników światła, które chcieliśmy aby znalazły się na wysokości przednich kół.



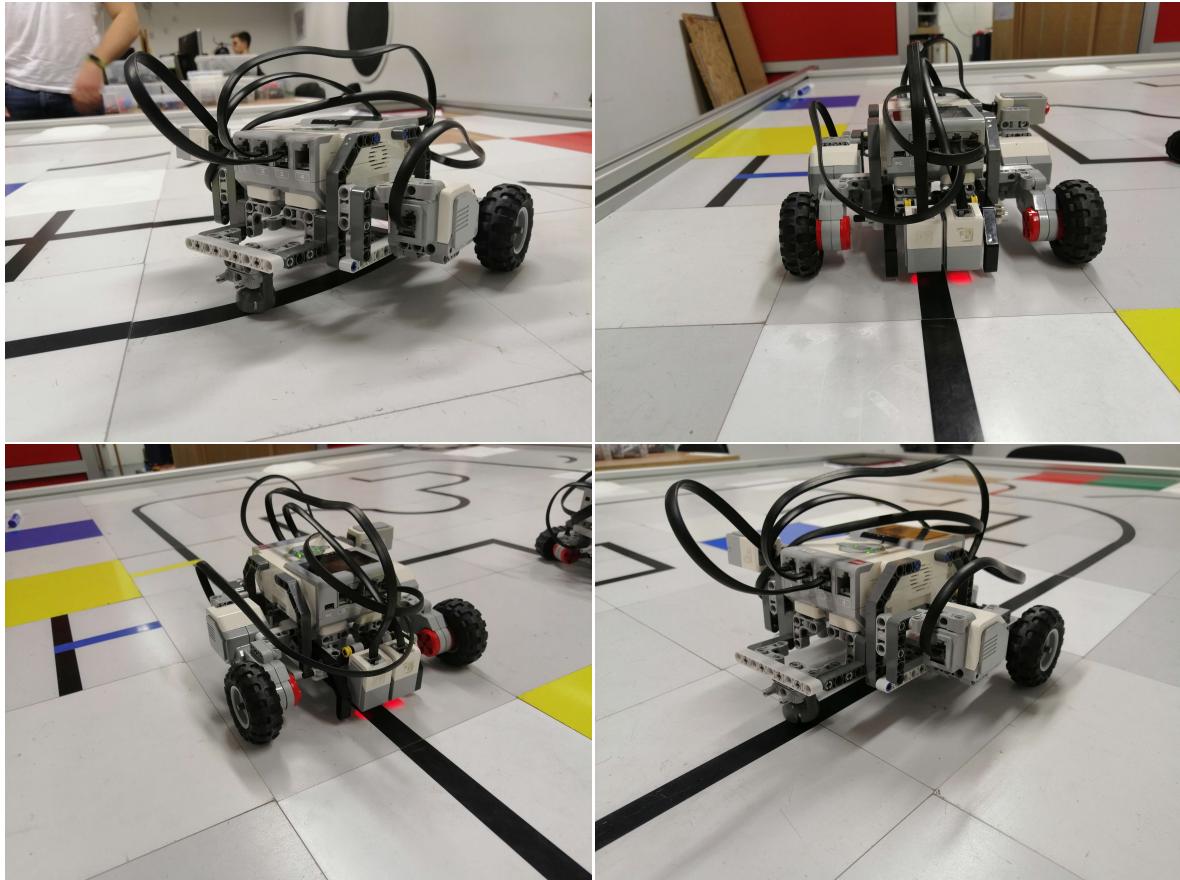
Rysunek 5.1. Pierwsza iteracja robota - z kołem samonastawnym



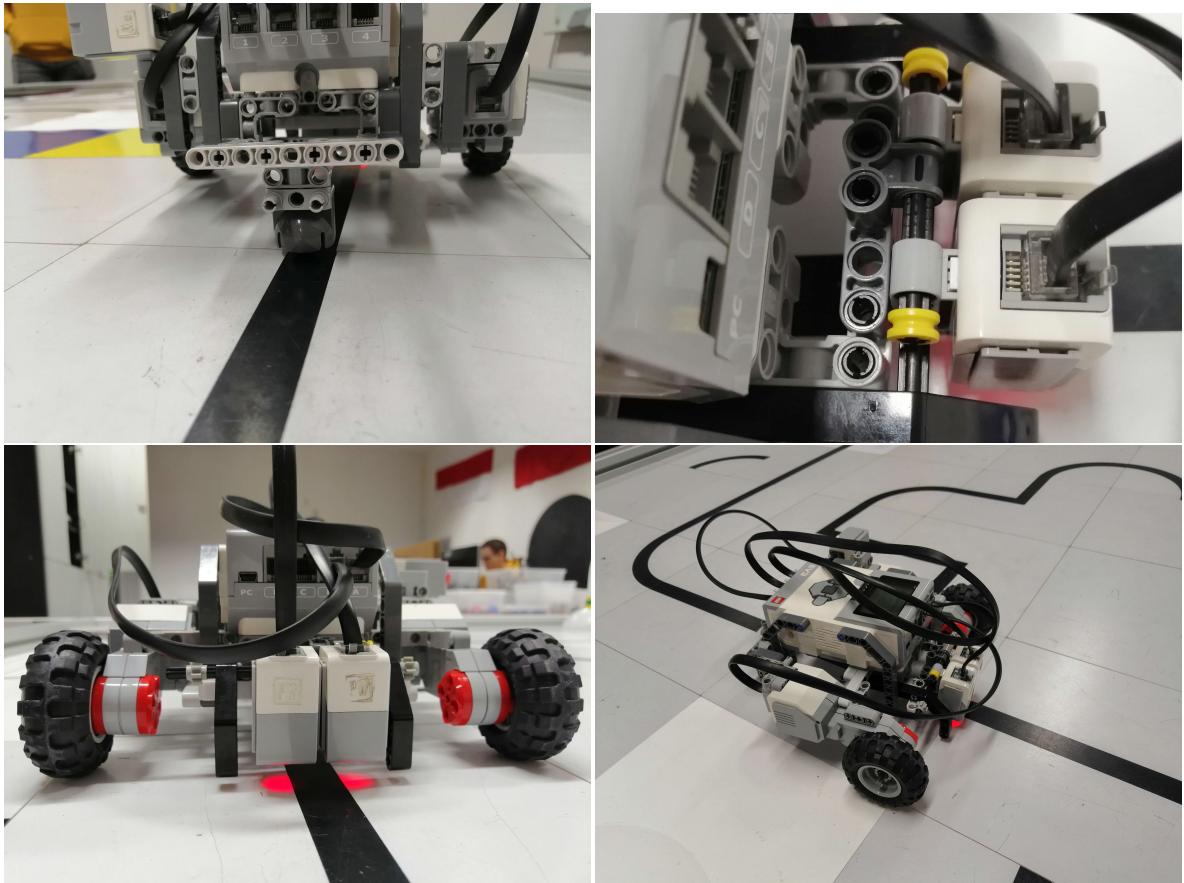
Rysunek 5.2. Pierwsza iteracja robota - z kołem samonastawnym

5.1.2. Robot - iteracja II

W celu uzyskania lepszych czasów w czasie trwania zawodów zamieniliśmy koło samonastawne na kulę, która spowodowała zmniejszenie tarcia o powierzchnię w rezultacie, zwiększyła szybkość poruszania się robota.



Rysunek 5.3. Druga iteracja robota - z kulą zamiast koła wspierającego



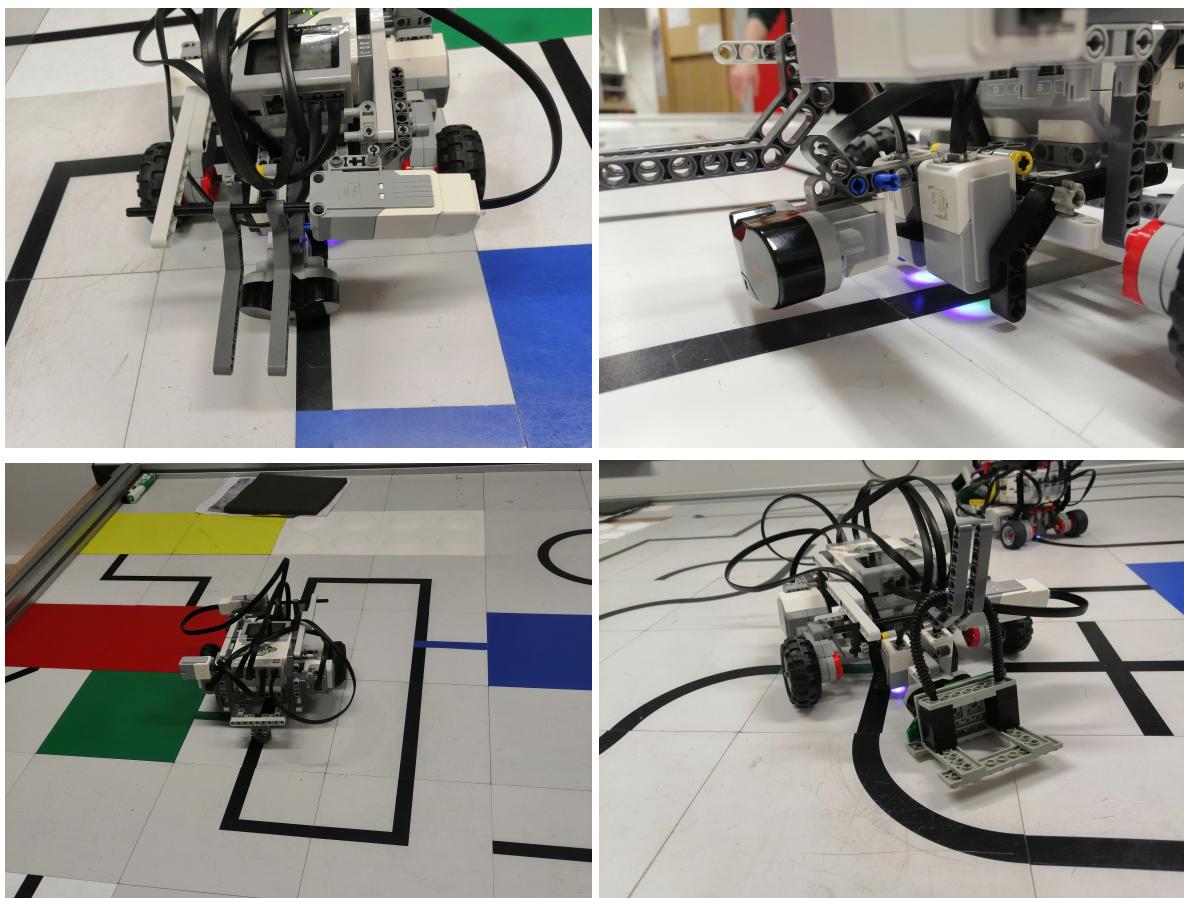
Rysunek 5.4. Druga iteracja robota - z kulą zamiast koła wspierającego

5.2. Line Follower

Do zaliczenia pierwszego zadania, polegającego na śledzeniu linii, wykorzystaliśmy II iterację robota zbudowanego w czasie trwania zawodów.

5.3. Transporter

W tym etapie przebudowaliśmy trochę nasz robot w taki sposób, aby mógł wykrywać i przewozić zbudowany przez nas przedmiot. Dlatego też, czujnik odległości, aby poprawnie rozpoznawał obliczał odległość obiektów znajdujących się przed robotem, musiał być zamontowany przed czujnikami światła, dodatkowo musiał znaleźć się na tyle nisko, aby nie utrudniał pracy wyświetlnika. Tak więc przemontowaliśmy go do dolnej podstawy robota, pomiędzy czujnikami światła, lekko wysuniętym do przodu. Podnośnik zamontowaliśmy u góry, na wyciągniętych ramionach, tak aby znajdował się nad czujnikiem odległości, a jako ramiona do podnoszenia wybraliśmy prostopadle zamontowane zakrzywione klocki, tak aby w łatwy sposób unieść obiekt znajdujący się przed robotem.



Rysunek 5.5. Trzecia iteracja robota - wspierająca detekcję i przenoszenie przedmiotów

5.4. Działający robot

LINK DO NAGRANIA NA YOUTUBE

6. Implementacja

6.1. Kod bazowy

Najpierw napisaliśmy podstawę do rozwijania kolejnych iteracji naszego kodu. Pozwoliło to nam przy kolejnych iteracjach jedynie kopiować podstawę i dowolnie ją modyfikować według potrzeb. Kod podstawy umieściliśmy w pliku Baza

Kod bazowy umożliwia nam: - start programu - zatrzymanie programu z upewnieniem się, że koła przestaną się poruszać - głosowe potwierdzenie stanu (START, READY, STOP) - wymagana jest jedynie implementacja jednej funkcji `iterate()`

Fragment kodu 6.1. Bazowy kod programu

```

1 def speak(message: str) -> None:
2     sound.speak(message)
3     print(message)
4
5
6 def work() -> None:
7     while True:
8         if button.is_pressed:
9             handle_button_pressed()
10        else:
11            try:
12                iterate()
13            except Exception as e:
14                print(e)
15
16
17 def handle_button_pressed() -> None:
18     stop()
19     speak('STOP')
20     button.wait_for_released()
21     button.wait_for_bump()
22     speak('START')
23
24
25 def iterate() -> None:
26     pass
27
28
29 def stop() -> None:
30     for motor in motors:
31         motor.stop()
32
33
34 def main() -> None:
35     sound.set_volume(SOUND_VOLUME)
36     speak('READY')
```

6. Implementacja

```
37
38     button.wait_for_bump()
39     speak('START')
40
41     try:
42         work()
43     except KeyboardInterrupt as e:
44         stop()
45         raise e
46
47
48 if __name__ == '__main__':
49     main()
```

6.2. Kod Line Follower'a

W ramach tego zadania rozbudowaliśmy bazowy kod, o funkcjonalność robota, tak aby ten mógł rozpoznawać linię i podążać za nią. Na zajęcia przygotowaliśmy dwie wersje kodu. Kod ["Naiwny"](#), bezpośrednio reagujący na to co odbierają czujniki oraz Kod bazujący na PID, który ostatecznie został wykorzystany do zaliczenia zadania.

6.2.1. Kod “Naiwny”

Na samym początku założyliśmy naiwny sposób śledzenia lini - kod dostępny jest w pliku Naiwny

Określeniem “naiwny” nazywamy sterowanie jedynie na podstawie koloru - widzimy biały jedziemy - widzimy czarny cofamy dla odpowiedniej strony robota.

Fragment kodu 6.2. Śledzenie lini - kod naiwny

```
1 FORWARD_SPEED = 30
2 TURN_SPEED = 40
3
4 SLEEP_SECONDS = 0.1
5
6 def iterate() -> None:
7     colors = (
8         left_sensor.color,
9         right_sensor.color
10    )
11    if colors[LEFT] == colors[RIGHT]:
12        move_tank.on(
13            SpeedPercent(FORWARD_SPEED),
14            SpeedPercent(FORWARD_SPEED)
15        )
16    elif colors[LEFT] != ColorSensor.COLOR_BLACK and colors[RIGHT] != ColorSensor.COLOR_BLACK:
17        move_tank.on(
18            SpeedPercent(FORWARD_SPEED),
```

```

19         SpeedPercent(FORWARD_SPEED)
20     )
21     elif colors[LEFT] != ColorSensor.COLOR_WHITE and colors[RIGHT] ==
22         ColorSensor.COLOR_WHITE:
23         move_tank.on(
24             SpeedPercent(-FORWARD_SPEED - TURN_SPEED),
25             SpeedPercent(FORWARD_SPEED + TURN_SPEED)
26         )
27     elif colors[LEFT] == ColorSensor.COLOR_WHITE and colors[RIGHT] !=
28         ColorSensor.COLOR_WHITE:
29         move_tank.on(
30             SpeedPercent(FORWARD_SPEED + TURN_SPEED),
31             SpeedPercent(-FORWARD_SPEED - TURN_SPEED)
32         )
33     elif colors[LEFT] == ColorSensor.COLOR_NOCOLOR or colors[RIGHT] ==
34         ColorSensor.COLOR_NOCOLOR:
35         move_tank.off()
36
37     sleep(SLEEP_SECONDS)

```

Przygotowaliśmy kilka iteracji kodu naiwnego, jednak nie sprawdzały się w takim stopniu, jaki chcieliśmy:

- Naiwny
- Naiwny
- Naiwny

6.2.2. Kod bazujący na PID

Przygotowaliśmy także kilka wersji kodu, które działają na bazie PID, opartej o poziom odbitego światła:

- PID
- PID
- PID

Ostatecznie po dopracowaniu kodu wpadliśmy na pomysł, żeby manipulować prędkość na prostych w zależności od wyliczonej prędkości skrętu. Na odcinkach prostych, gdy prędkość skrętu była bliska 0 jechaliśmy z prędkością maksymalną - 100, gdy prędkość skrętu wzrastała odpowiednio zmniejszaliśmy prędkość do przodu, tak do osiągnięcia minimalnej prędkości do przodu.

- Najszybszy

Fragment kodu 6.3. Śledzenie lini - kod bazujący na PID

```

1 MIN_FORWARD_SPEED = 30
2 MAX_FORWARD_SPEED = 100
3
4 FORWARD_SPEED_CORRECTION = (
5     (MAX_FORWARD_SPEED - MIN_FORWARD_SPEED) / MAX_FORWARD_SPEED
6 )

```

6. Implementacja

```
7
8 CONSTANT_P = 4.0
9 CONSTANT_I = 0.01
10 CONSTANT_D = 4.0
11
12 HISTORY_LOSS = 0.5
13
14 AMPLIFIER = 0.1
15
16 def iterate(integral: float, last_error: int) -> Tuple[float, int]:
17     error = left_sensor.reflected_light_intensity - \
18             right_sensor.reflected_light_intensity
19
20     integral = HISTORY_LOSS * integral + error
21     derivative = error - last_error
22     last_error = error
23
24     turn_speed = CONSTANT_P * error + CONSTANT_I * integral + CONSTANT_D * \
25                 derivative
26
27     forward_speed = max(
28         MIN_FORWARD_SPEED,
29         MAX_FORWARD_SPEED - FORWARD_SPEED_CORRECTION * abs(turn_speed)
30     )
31
32     left_motor.on(forward_speed + AMPLIFIER * turn_speed)
33     right_motor.on(forward_speed - AMPLIFIER * turn_speed)
34
35     return integral, last_error
```

6.3. Kod Transportera

W ramach ostatniego zadania rozbudowaliśmy funkcjonalność kodu z poprzedniego zadania Kod bazujący na PID.

Zdefiniowaliśmy stany:

Fragment kodu 6.4. Stany osiągane przez transporter

```
1 FOLLOW_LINE_UNTIL_PICK_UP = 0
2 FOLLOW_LINE_UNTIL_DETECTED_OBJECT = 1
3 FOLLOW_LINE_UNTIL_TWO_LINES_DETECTED = 2
4 FOLLOW_LINE_UNTIL_DROP_DOWN = 3
5 FOLLOW_LINE_UNTIL_TWO_DROP_DOWN_COLORS_DETECTED = 4
6 STATE_STOP = 5
```

Zaktualizowaliśmy główną pętlę:

Fragment kodu 6.5. Zmodyfikowana pętla bazowa programu

```
1 def work() -> None:
2     integral = 0.0
```

```

3     last_error = 0
4
5     state = FOLLOW_LINE_UNTIL_PICK_UP
6
7     while True:
8         if button.is_pressed:
9             handle_button_pressed()
10            integral = 0.0
11            last_error = 0
12            state = FOLLOW_LINE_UNTIL_PICK_UP
13        else:
14            state, integral, last_error = iteration(
15                state, integral, last_error
16            )

```

W pętli na podstawie stanu wołaliśmy odpowiednią funkcję obsługi:

Fragment kodu 6.6. Decyżję, jakiej funkcji obsługi użyć podejmowaliśmy na podstawie stanu

```

1 ITERATION_FUNCTION = {
2     FOLLOW_LINE_UNTIL_PICK_UP: follow_line_until_pick_up,
3     FOLLOW_LINE_UNTIL_DETECTED_OBJECT: follow_line_until_detected_object,
4     FOLLOW_LINE_UNTIL_TWO_LINES_DETECTED: follow_line_until_two_lines_detected,
5     FOLLOW_LINE_UNTIL_DROP_DOWN: follow_line_until_drop_down,
6     FOLLOW_LINE_UNTIL_TWO_DROP_DOWN_COLORS_DETECTED:
9         follow_line_until_two_drop_down_colors_detected,
7 }
8
10 def iteration(state: int, integral: float, last_error: int) -> Tuple[int, float
11     , int]:
11     function = ITERATION_FUNCTION.get(state, stop_robot)
12     state, integral, last_error = function(state, integral, last_error)
13     return state, integral, last_error

```

Dla każdego stanu zdefiniowaliśmy obsługę:

- Śledzenie linii z wykorzystaniem algorytmu korzystającego z PID, dopóki nie napotkamy koloru z którego powinniśmy podnieść przedmiot
- Gdy odpowiednio na lewym lub prawym czujniku wykryjemy dany kolor to zaczynamy obracać się w tym kierunku
- Obrót o 90 stopni jest wyliczony i zawsze wykonywane jest tyle samo obrotów kół - zakładamy brak poślizgu
- Następnie aktualizowany jest stan - średzenie linii dopóki nie napotkamy obiektu

Fragment kodu 6.7. Obsługa śledzenia linii do momentu napotkania koloru z którego należy podnieść przedmiot

```

1 def follow_line_until_pick_up(state: int, integral: float, last_error: int) ->
2     Tuple[int, float, int]:
3     colors = detect_colors()
3     if colors[LEFT] == COLORS[PICK_UP]:

```

6. Implementacja

```
4     turn_left()
5     state = FOLLOW_LINE_UNTIL_DETECTED_OBJECT
6     elif colors[RIGHT] == COLORS[PICK_UP]:
7         turn_right()
8         state = FOLLOW_LINE_UNTIL_DETECTED_OBJECT
9     else:
10        integral, last_error = follow_line(integral, last_error)
11
12    return state, integral, last_error
```

- Śledzenie lini z wykorzystaniem PID dopóki czujnik odległości nie wykryje przedmiotu w odległości 1 od przodu robota
- Gdy odległość jest wystarczająco bliska to podnosimy przedmiot, obracamy się o 180 stopni (wyliczona ilość obrotów kół) i przechodzimy do następnego stanu

Fragment kodu 6.8. Obsługa śledzenia lini do momentu wykrycia przedmiotu przed robotem

```
1 def follow_line_until_detected_object(state: int, integral: float, last_error: float) -> Tuple[int, float, int]:
2     detected_distance = distance()
3     if detected_distance < 2:
4         pick_up()
5         turn_around()
6         state = FOLLOW_LINE_UNTIL_TWO_LINES_DETECTED
7     else:
8         integral, last_error = follow_line(integral, last_error)
9     return state, integral, last_error
```

- Po podniesieniu przedmiotu śledziliśmy linię dopóki nie napotkamy na obu czujnikach koloru - koloru czarnego - oznaczało, że dojechaliśmy do skrzyżowania
- na skrzyżowaniu skręcaliśmy w prawo, a następnie przechodziliśmy do kolejnego stanu

Fragment kodu 6.9. Obsługa śledzenia lini do momentu wykrycia skrzyżowania

```
1 def follow_line_until_two_lines_detected(state: int, integral: float, last_error: float) -> Tuple[int, float, int]:
2     colors = detect_colors()
3     if colors[LEFT] == ColorSensor.COLOR_BLACK and colors[RIGHT] == ColorSensor.COLOR_BLACK:
4         turn_right()
5         state = FOLLOW_LINE_UNTIL_DROP_DOWN
6     else:
7         integral, last_error = follow_line(integral, last_error)
8
9     return state, integral, last_error
```

- Podobnie jak w stanie pierwszym śledziliśmy linię dopóki na jednym z czujników nie wykryjemy koloru na który należy odłożyć przedmiot
- Gdy wykryjemy kolor to skręcamy w odpowiednią stronę i przechodzimy do następnego stanu

Fragment kodu 6.10. Obsługa śledzenia lini do momentu napotkania koloru na który należy odłożyć przedmiot

```

1 def follow_line_until_drop_down(state: int, integral: float, last_error: int)
2     -> Tuple[int, float, int]:
3     colors = detect_colors()
4     if colors[LEFT] == COLORS[DROP_DOWN]:
5         turn_left()
6         state = FOLLOW_LINE_UNTIL_TWO_DROP_DOWN_COLORS_DETECTED
7     elif colors[RIGHT] == COLORS[DROP_DOWN]:
8         turn_right()
9         state = FOLLOW_LINE_UNTIL_TWO_DROP_DOWN_COLORS_DETECTED
10    else:
11        integral, last_error = follow_line(integral, last_error)
12
12    return state, integral, last_error

```

- Na sam koniec pozostało śledzenie lini dopóki nie wykryjemy na obu czujnikach koloru - koloru docelowego
- Gdy wykryjemy ten kolor to odkładamy przedmiot, puszczaamy muzykę i obracamy się w miejscu, po czym zatrzymujemy robota

Fragment kodu 6.11. Obsługa śledzenia lini do momentu wykrycia kwadratu na który należy odłożyć przedmiot

```

1 def follow_line_until_two_drop_down_colors_detected(state: int, integral: float
2     , last_error: int) -> Tuple[int, float, int]:
3     colors = detect_colors()
4     if colors[LEFT] == COLORS[DROP_DOWN] and colors[RIGHT] == COLORS[DROP_DOWN]
5         ]:
6         drop_down()
7         sound.play_song(WINNING_SONG)
8         turn_around()
9         state = STATE_STOP
10    else:
11        integral, last_error = follow_line(integral, last_error)
12
11    return state, integral, last_error
12
13 WINNING_SONG = (
14     ('D4', 'e3'), # intro anacrouse
15     ('D4', 'e3'),
16     ('D4', 'e3'),
17     ('G4', 'h'), # meas 1
18     ('D5', 'h'),
19     ('C5', 'e3'), # meas 2
20     ('B4', 'e3'),
21     ('A4', 'e3'),
22     ('G5', 'h'),
23     ('D5', 'q'),
24     ('C5', 'e3'), # meas 3
25     ('B4', 'e3'),

```

6. Implementacja

```
26     ('A4', 'e3'),
27     ('G5', 'h'),
28     ('D5', 'q'),
29     ('C5', 'e3'), # meas 4
30     ('B4', 'e3'),
31     ('C5', 'e3'),
32     ('A4', 'h.'),
33 )
```

- W ostatnim stanie resetujemy ustawienie robota i oczekujemy na przycisk, aby robot mógł wystartować ponownie

Fragment kodu 6.12. Ostatni stan - zatrzymanie robota i oczekiwanie na wciśnięcie przycisku

```
1 def stop_robot(state: int, integral: float, last_error: int) -> Tuple[int,
2     float, int]:
3     handle_button_pressed()
4     state = FOLLOW_LINE_UNTIL_PICK_UP
5     integral = 0.0
6     last_error = 0
7     return state, integral, last_error
```

Funkcje pomocnicze:

Wykrywanie kolorów - mieliśmy problem z kolorem wykrywanym przez migające na zmianę czujniki - aby ujednolicić pomiary najpierw zmienialiśmy tryb wykrywania czujników, a następnie dopiero wykrywaliśmy kolor

Fragment kodu 6.13. Funkcja pomocnicza - wykrywanie koloru

```
1 def detect_colors() -> Tuple[int, int]:
2     ensure_mode(ColorSensor.MODE_COL_COLOR)
3     return (
4         left_sensor.color,
5         right_sensor.color
6     )
7
8 def ensure_mode(color: str) -> None:
9     left_sensor.mode = color
10    right_sensor.mode = color
11    sleep(TIME_PER_MODE_CHANGE)
```

Stała ilość rotacji w miejscu: - ponieważ czujniki są minimalnie przesunięte do przodu względem osi kół, to przed obrotem jedziemy minimalnie do przodu, aby po obrocie robot skończył z czujnikami wokół lini - podobnie po skończonym obrocie jasność kolorów na które wjeżdżamy nie zawsze pozwalała nam na dobre rozróżnianie tych kolorów od koloru białego za pomocą czujników odbijających światło czerwone - więc rozwiązaliśmy to przejechaniem przez ten kolor po prostej i dopiero gdy dojechaliśmy do koloru czarnego załączało się dalsze śledzenie linii

Fragment kodu 6.14. Funkcje pomocnicze - obrót

```
1 ROTATIONS_PER_FULL_ROTATION = 3.15
2
3 def turn(full_rotations: float, speed: int) -> None:
4     rotations = ROTATIONS_PER_FULL_ROTATION * full_rotations
5     forward_for_rotations(0.1)
6     left_motor.on_for_rotations(
7         speed,
8         rotations,
9         block=False
10    )
11    right_motor.on_for_rotations(
12        - speed,
13        rotations,
14    )
15    forward_for_rotations(0.3)
16
17
18 def forward_for_rotations(rotations: float) -> None:
19     left_motor.on_for_rotations(
20         MIN_FORWARD_SPEED,
21         rotations,
22         block=False
23    )
24     right_motor.on_for_rotations(
25         MIN_FORWARD_SPEED,
26         rotations
27    )
28
29
30 def turn_around() -> None:
31     turn(0.5, MAX_FORWARD_SPEED)
32
33
34 def turn_left() -> None:
35     turn(0.25, -MAX_FORWARD_SPEED)
36
37
38 def turn_right() -> None:
39     turn(0.25, MAX_FORWARD_SPEED)
```

Spis rysunków

1.1	Przykładowa trasa do podążania za wyznaczoną linią	3
1.2	Przykładowa plansza do zadania Transporter	3
2.1	Prosta linia	4
2.2	Ostry zakręt	4
2.3	Zaokrąglony zakręt	4
2.4	Skrzyżowanie	4
2.5	Zakrzywiona linia	4
2.6	Czerwone skrzyżowanie	5
2.7	Zielone skrzyżowanie	5
2.8	Czerwona platforma	5
2.9	Zielona platforma	5
3.1	Otrzymane elementy robota LEGO Mindstorms Ev3	5
4.1	Schemat	7
4.2	Tor na zawody	11
4.3	Tor - podążanie za linią	11
4.4	Trasa dla transportera	11
5.1	Pierwsza iteracja robota - z kołem samonastawnym	12
5.2	Pierwsza iteracja robota - z kołem samonastawnym	13
5.3	Druga iteracja robota - z kulą zamiast koła wspierającego	14
5.4	Druga iteracja robota - z kulą zamiast koła wspierającego	15
5.5	Trzecia iteracja robota - wspierająca detekcję i przenoszenie przedmiotów	16

Spis tabel

3.1	Otrzymane elementy robota - zadanie 1	6
3.2	Otrzymane elementy robota - zadanie 2	6
4.1	Wyniki zawodów	9

Fragmenty kodu

4.1	Zmodyfikowane parametry - zadanie śledzenia lini	9
6.1	Bazowy kod programu	17
6.2	Śledzenie lini - kod naiwny	18
6.3	Śledzenie lini - kod bazujący na PID	19
6.4	Stany osiągane przez transporter	20
6.5	Zmodyfikowana pętla bazowa programu	20
6.6	Decyzję, jakiej funkcji obsługi użyć podejmowaliśmy na podstawie stanu	21

6.7	Obsługa śledzenia linii do momentu napotkania koloru z którego należy podnieść przedmiot	21
6.8	Obsługa śledzenia lini do momentu wykrycia przedmiotu przed robotem	22
6.9	Obsługa śledzenia lini do momentu wykrycia skrzyżowania	22
6.10	Obsługa śledzenia lini do momentu napotkania koloru na który należy odłożyć przedmiot	23
6.11	Obsługa śledzenia lini do momentu wykrycia kwadratu na który należy odłożyć przedmiot	23
6.12	Ostatni stan - zatrzymanie robota i oczekiwanie na wciśnięcie przycisku	24
6.13	Funkcja pomocnicza - wykrywanie koloru	24
6.14	Funkcje pomocnicze - obrót	24