Politechnika Warszawska

WYDZIAŁ ELEKTRONIKI
I TECHNIK INFORMACYJNYCH

Instytut Automatyki i Informatyki Stosowanej

Praca dyplomowa inżynierska

na kierunku Informatyka
w specjalności Inżynieria Oprogramowania

Aplikacja mobilna umożliwiająca rozgrywkę on-line w gry logiczne na szachownicy

Bartłomiej Krawczyk, Mateusz Brzozowski
Numer albumu 310774, 310678

promotor
{PROMOTOR}

WARSZAWA 2023

Aplikacja mobilna umożliwiająca rozgrywkę on-line w gry logiczne na szachownicy

Streszczenie. TODO

Słowa kluczowe: Mindstorms, Python, PID

A mobile application that allows you to play logic games on-line on a chessboard

Abstract. {ABSTRACT}

Keywords: Mindstorms, Python, PID

Spis treści

1. Wstęp do Robotyki

Studenci:

```
Bartłomiej Krawczyk - 310774
Mateusz Brzozowski - 310608
```

Nazwa robota:

- Parostatek

2. Mechanika

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

2.1. Schemat

Schemat

2.2. 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

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

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

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

2.3. Kod

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 2.1. FastAPI

```
def speak(message: str) -> None:
    sound.speak(message)
    print(message)


def work() -> None:
    while True:
        if button.is_pressed:
            handle_button_pressed()
```

```
        else:
            try:
                iterate()
            except Exception as e:
                print(e)


def handle_button_pressed() -> None:
    stop()
    speak('STOP')
    button.wait_for_released()
    button.wait_for_bump()
    speak('START')


def iterate() -> None:
    pass


def stop() -> None:
    for motor in motors:
        motor.stop()


def main() -> None:
    sound.set_volume(SOUND_VOLUME)
    speak('READY')

    button.wait_for_bump()
    speak('START')

    try:
        work()
    except KeyboardInterrupt as e:
        stop()
        raise e


if __name__ == '__main__':
    main()
```

3. Zawody

3.1. Robot - iteracja I

Rysunek 3.1. Pierwsza iteracja robota - z kołem samonastawnym

Rysunek 3.2. Pierwsza iteracja robota - z kołem samonastawnym

3.2. Robot - iteracja II

Fast Line Follower Fast Line Follower Fast Line Follower Fast Line Follower

Rysunek 3.3. Druga iteracja robota - z kulą zamiast koła wspierającego

Fast Line Follower Fast Line Follower Fast Line Follower Fast Line Follower

Rysunek 3.4. Druga iteracja robota - z kulą zamiast koła wspierającego

3.3. Kod

3.3.1. Kod “Naiwny”

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

Opisać, że “naiwny” nazywamy sterowanie jedynie na podstawie koloru - widzimy biały jedziemy - widzimy czarny cofamy dla odpowiedniej strony robota. Można zwrócić uwagę, że kolor badamy tylko raz na iteracją

Fragment kodu 3.1. FastAPI

```
FORWARD_SPEED = 30
TURN_SPEED = 40

SLEEP_SECONDS = 0.1

def iterate() -> None:
    colors = (
        left_sensor.color,
        right_sensor.color
    )
    if colors[LEFT] == colors[RIGHT]:
        move_tank.on(
            SpeedPercent(FORWARD_SPEED),
            SpeedPercent(FORWARD_SPEED)
        )
    elif colors[LEFT] != ColorSensor.COLOR_BLACK and colors[RIGHT] !=
         ColorSensor.COLOR_BLACK:
        move_tank.on(
            SpeedPercent(FORWARD_SPEED),
            SpeedPercent(FORWARD_SPEED)
        )
    elif colors[LEFT] != ColorSensor.COLOR_WHITE and colors[RIGHT] ==
         ColorSensor.COLOR_WHITE:
        move_tank.on(
            SpeedPercent(-FORWARD_SPEED - TURN_SPEED),
            SpeedPercent(FORWARD_SPEED + TURN_SPEED)
        )
    elif colors[LEFT] == ColorSensor.COLOR_WHITE and colors[RIGHT] !=
         ColorSensor.COLOR_WHITE:
        move_tank.on(
            SpeedPercent(FORWARD_SPEED + TURN_SPEED),
            SpeedPercent(-FORWARD_SPEED - TURN_SPEED)
        )
    elif colors[LEFT] == ColorSensor.COLOR_NOCOLOR or colors[RIGHT] ==
         ColorSensor.COLOR_NOCOLOR:
        move_tank.off()

    sleep(SLEEP_SECONDS)
```

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

3.3.2. Kod bazujący na PID

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

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 3.2. FastAPI

```
MIN_FORWARD_SPEED = 30
MAX_FORWARD_SPEED = 100

FORWARD_SPEED_CORRECTION = (
    (MAX_FORWARD_SPEED - MIN_FORWARD_SPEED) / MAX_FORWARD_SPEED
)

CONSTANT_P = 4.0
CONSTANT_I = 0.01
CONSTANT_D = 4.0

HISTORY_LOSS = 0.5

AMPLIFIER = 0.1

def iterate(integral: float, last_error: int) -> Tuple[float, int]:
    error = left_sensor.reflected_light_intensity - \
        right_sensor.reflected_light_intensity

    integral = HISTORY_LOSS * integral + error
    derivative = error - last_error
    last_error = error

    turn_speed = CONSTANT_P * error + CONSTANT_I * integral + CONSTANT_D *
         derivative

    forward_speed = max(
        MIN_FORWARD_SPEED,
        MAX_FORWARD_SPEED - FORWARD_SPEED_CORRECTION * abs(turn_speed)
    )

    left_motor.on(forward_speed + AMPLIFIER * turn_speed)
    right_motor.on(forward_speed - AMPLIFIER * turn_speed)

    return integral, last_error
```

3.3.2.1. 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

3.4. Dobieranie parametrów

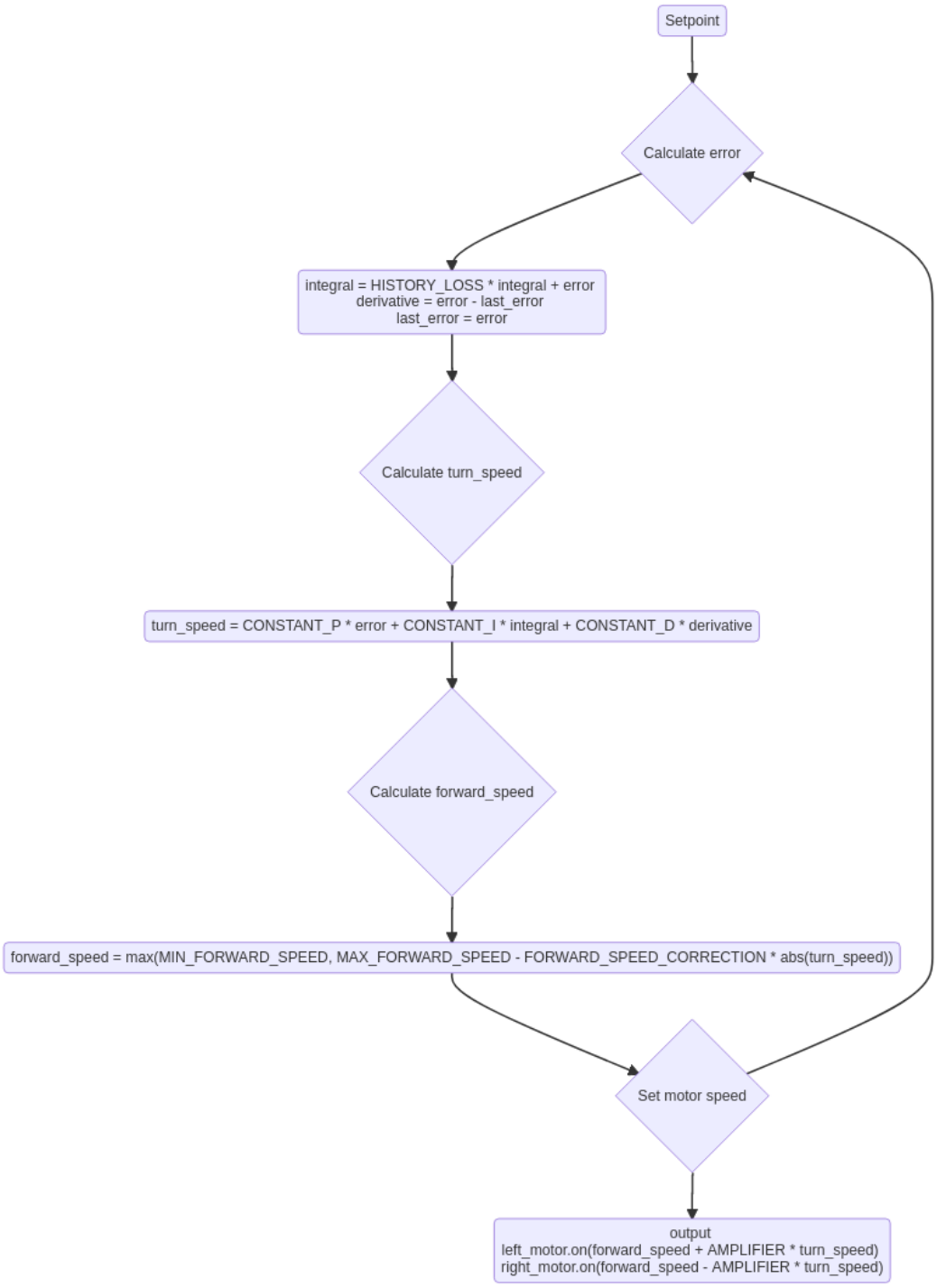
Parametry PID, które zastosowaliśmy w naszym robocie, zostały dobrane metodą inżynierską. Zastosowaliśmy podejście iteracyjne, w którym kolejne wartości parametrów były ustalane na podstawie wyników testów oraz obserwacji zachowania robota.

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. Zwiększaliśmy również parametr I, aby nasz robot na odcinkach prostych mógł osiągnąć jak największą prędkość. W przypadku parametrów D i I staraliśmy się dobrać 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.

3.5. Schemat blokowy PID



3.6. Tor

3.7. Wyniki

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

3.7.1. Wnioski

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

4. Line Follower

4.1. Kod

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

Fragment kodu 4.1. FastAPI

```
MIN_FORWARD_SPEED = 10
MAX_FORWARD_SPEED = 20

FORWARD_SPEED_CORRECTION = (
    (MAX_FORWARD_SPEED - MIN_FORWARD_SPEED) / MAX_FORWARD_SPEED
)

CONSTANT_P = 4.0
CONSTANT_I = 0.01
CONSTANT_D = 4.0

HISTORY_LOSS = 0.5

AMPLIFIER = 0.25
```

4.2. Tor

5. Transporter

W tym etapie przebudowaliśmy trochę nasz robot w taki sposób, aby mógł przewozić on wykrywać i przewozić zbudowany przez nas przedmiot.

Rysunek 5.1. Trzecia iteracja robota - wspierająca detekcję i przenoszenie przedmiotów

5.1. Działający robot

LINK DO NAGRANIA NA YOUTUBE

5.2. Kod

Zdefniowaliśmy stany:

Fragment kodu 5.1. FastAPI

```
################
# #
# STATES #
# #
################

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

Zaktualizowaliśmy główną pętlę:

Fragment kodu 5.2. FastAPI

```
def work() -> None:
    integral = 0.0
    last_error = 0

    state = FOLLOW_LINE_UNTIL_PICK_UP

    while True:
        if button.is_pressed:
            handle_button_pressed()
            integral = 0.0
            last_error = 0
            state = FOLLOW_LINE_UNTIL_PICK_UP
        else:
            state, integral, last_error = iteration(
                state, integral, last_error
            )
```

W pętli na podstawie stanu wołaliśmy odpowiednią funkcję obsługi:

Fragment kodu 5.3. FastAPI

```
ITERATION_FUNCTION = {
    FOLLOW_LINE_UNTIL_PICK_UP: follow_line_until_pick_up,
    FOLLOW_LINE_UNTIL_DETECTED_OBJECT: follow_line_until_detected_object,
    FOLLOW_LINE_UNTIL_TWO_LINES_DETECTED: follow_line_until_two_lines_detected,
    FOLLOW_LINE_UNTIL_DROP_DOWN: follow_line_until_drop_down,
    FOLLOW_LINE_UNTIL_TWO_DROP_DOWN_COLORS_DETECTED:
        follow_line_until_two_drop_down_colors_detected,
}


def iteration(state: int, integral: float, last_error: int) -> Tuple[int, float
    , int]:
    function = ITERATION_FUNCTION.get(state, stop_robot)
    state, integral, last_error = function(state, integral, last_error)
    return state, integral, last_error
```

Dla każdego stanu zdefniowaliśmy obsługę:

- Śledzenie lini 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 lini dopóki nie napotkamy obiektu

Fragment kodu 5.4. FastAPI

```
def follow_line_until_pick_up(state: int, integral: float, last_error: int) ->
    Tuple[int, float, int]:
    colors = detect_colors()
    if colors[LEFT] == COLORS[PICK_UP]:
        turn_left()
        state = FOLLOW_LINE_UNTIL_DETECTED_OBJECT
    elif colors[RIGHT] == COLORS[PICK_UP]:
        turn_right()
        state = FOLLOW_LINE_UNTIL_DETECTED_OBJECT
    else:
        integral, last_error = follow_line(integral, last_error)

    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 5.5. FastAPI

```
def follow_line_until_detected_object(state: int, integral: float, last_error: int) -> Tuple[int, float, int]:
    detected_distance = distance()
    if detected_distance < 2:
        pick_up()
        turn_around()
        state = FOLLOW_LINE_UNTIL_TWO_LINES_DETECTED
    else:
        integral, last_error = follow_line(integral, last_error)
    return state, integral, last_error
```

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

Fragment kodu 5.6. FastAPI

```
def follow_line_until_two_lines_detected(state: int, integral: float, last_error: int) -> Tuple[int, float, int]:
    colors = detect_colors()
    if colors[LEFT] == ColorSensor.COLOR_BLACK and colors[RIGHT] == ColorSensor.COLOR_BLACK:
        turn_right()
        state = FOLLOW_LINE_UNTIL_DROP_DOWN
    else:
        integral, last_error = follow_line(integral, last_error)

    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 5.7. FastAPI

```
def follow_line_until_drop_down(state: int, integral: float, last_error: int) -> Tuple[int, float, int]:
    colors = detect_colors()
    if colors[LEFT] == COLORS[DROP_DOWN]:
        turn_left()
        state = FOLLOW_LINE_UNTIL_TWO_DROP_DOWN_COLORS_DETECTED
    elif colors[RIGHT] == COLORS[DROP_DOWN]:
        turn_right()
        state = FOLLOW_LINE_UNTIL_TWO_DROP_DOWN_COLORS_DETECTED
    else:
        integral, last_error = follow_line(integral, last_error)

    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, puszczamy muzykę i obracamy się w miejscu, po czym zatrzumujemy robota

Fragment kodu 5.8. FastAPI

```
def follow_line_until_two_drop_down_colors_detected(state: int, integral: float, last_error: int) -> Tuple[int, float, int]:
    colors = detect_colors()
    if colors[LEFT] == COLORS[DROP_DOWN] and colors[RIGHT] == COLORS[DROP_DOWN]:
        drop_down()
        sound.play_song(WINNING_SONG)
        turn_around()
        state = STATE_STOP
    else:
        integral, last_error = follow_line(integral, last_error)

    return state, integral, last_error

WINNING_SONG = (
    ('D4', 'e3'), # intro anacrouse
    ('D4', 'e3'),
    ('D4', 'e3'),
    ('G4', 'h'), # meas 1
    ('D5', 'h'),
    ('C5', 'e3'), # meas 2
    ('B4', 'e3'),
    ('A4', 'e3'),
    ('G5', 'h'),
    ('D5', 'q'),
    ('C5', 'e3'), # meas 3
    ('B4', 'e3'),
    ('A4', 'e3'),
    ('G5', 'h'),
    ('D5', 'q'),
    ('C5', 'e3'), # meas 4
    ('B4', 'e3'),
    ('C5', 'e3'),
    ('A4', 'h.'),
)
```

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

Fragment kodu 5.9. FastAPI

```
def stop_robot(state: int, integral: float, last_error: int) -> Tuple[int, float, int]:
    handle_button_pressed()
    state = FOLLOW_LINE_UNTIL_PICK_UP
```

```
    integral = 0.0
    last_error = 0
    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 5.10. FastAPI

```
def detect_colors() -> Tuple[int, int]:
    ensure_mode(ColorSensor.MODE_COL_COLOR)
    return (
        left_sensor.color,
        right_sensor.color
    )

def ensure_mode(color: str) -> None:
    left_sensor.mode = color
    right_sensor.mode = color
    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 lini

Fragment kodu 5.11. FastAPI

```
ROTATIONS_PER_FULL_ROTATION = 3.15

def turn(full_roations: float, speed: int) -> None:
    rotations = ROTATIONS_PER_FULL_ROTATION * full_roations
    forward_for_rotations(0.1)
    left_motor.on_for_rotations(
        speed,
        rotations,
        block=False
    )
    right_motor.on_for_rotations(
        - speed,
        rotations,
    )
    forward_for_rotations(0.3)


```

```
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)
```

5.3. Tor

Trasa dla transportera

Spis rysunków

Spis tabel

Fragmenty kodu

Wykaz symboli i skrótów

PID – Proportional-Integral-Derivative Controller