Sprawozdanie nr 1. „Robot śledzący linię”

---

1. Teoria

Pierwsze kroki w tym temacie zacząłem od zrozumienia jak robot Lego Mindstorm ma rozpoznawać linię. Robot będzie ją rozpoznawać na zasadzie badania położenia równowagi pomiędzy linią a tłem. W praktyce oznacza to, że musimy zbadać ilość odbitego światła od linii oraz tła po której będzie poruszać się robot. Wykonamy to za pomocą sensora kolorów, który działa w kilku trybach. Jednak ja będę korzystać tylko z 2. Jednym z nich będzie tryb koloru, który będzie rozpoznawać kolor po którym porusza się robot. Natomiast 2 to będzie tryb światła odbitego, który zbada ilość odbitego światła od podłoża, który dociera do sensora. Pozwoli mi to na badanie różnic pomiędzy podłożami (linia po której ma się poruszać robot oraz tło).
Następnie odpowiedni algorytm wyliczy jak robot ma się poruszać i czy nie zboczył z linii korygując prędkość kół w zależności jak daleko odsunął się od miejsca równowagi pomiędzy odbijanym światłem.

Kolejny krok z którym musiałem się zmierzyć to dobranie odpowiedniego algorytmu do wyliczania prędkości kół aby pojazd poruszał się po wyznaczonej linii. Miałem do dyspozycji 3 algorytmy:

- Regulator dwustawny
- Regulator proporcjonalny
- Regulator PID

Każdy z tych regulatorów wykazuje się innymi właściwościami, które zrozumiałem i opisałem w późniejszym podpunkcie „3. Spostrzeżenia”.
Pierwszy regulator jest prostym algorytmem opartym na jednym warunku. Sygnał sterujący w takim regulatorze przyjmuje tylko dwie wartości. Nie dysponujemy uchybem e=0.

Drugi regulator jest bardziej złożonym algorytmem, który reaguje na bieżąco na sygnał podany na wejściu. Wytwarza on proporcjonalny sygnał sterujący.

Ostatni regulator jest najbardziej skomplikowanym algorytmem opartym na pętli sprzężenia zwrotnego. Oblicza wartość uchybu jako różnicę pomiędzy pożądaną wartością zadaną i zmierzoną wartością zmiennej sygnału i działa w taki sposób, by zredukować uchyb.

2. Pierwsze kroki

Początkowo szybko zrozumiałem jak robot ma działać z kwestii algorytmu. Funkcja, którą stworzyłem do kalibracji sensora kolorów towarzyszyła mi do końca wszystkich następnych programów i nie zmieniałem jej w dużym stopniu. Był to prosty program wyliczający średnią arytmetyczną dwóch punktów.

```
def calibrate():
    print("Na bialy")
    while True:
        if btn.any():
            white = cl.value()
            break

    print("Na czarny")
    while True:
        if btn.any():
            black = cl.value()
            break
    return (white + black) / 2
```

Następnym elementem, który musiałem przemyśleć był dobór regulatora. Na początek wykonałem regulator dwustawny, który był prostym algorytmem sterowania prędkością osi kół. Program działał dopóki nie natrafił na czerwony pasek (wartość dla koloru czerwonego w sensorze kolorów to 5) i poruszał pojazdem z różną prędkością kół.

```
def two_state():
    while cl.value() != 5:
        cl.mode = 'COL-REFLECT'
        e = calibrate() - cl.value()
        if e < 0:
            left.run_forever(speed_sp=200)
            right.run_forever(speed_sp=100)
        else:
            left.run_forever(speed_sp=100)
            right.run_forever(speed_sp=200)
        cl.mode = 'COL-COLOR'
    left.stop()
    right.stop()
```

Jednak ten regulator nie spełniał moich oczekiwań, wnioski na ten temat są opisane w rozdziale „3. Spostrzeżenia”. Przeszedłem więc do kolejnego regulatora, który wydawał się na pierwszy rzut oka przeznaczony do takich zadań jak podążanie za linią. Regulator proporcjonalny zacząłem od tych samych podstaw co regulator dwustanowy. Program miał działać dopóki nie natrafi na czerwoną linię oraz w zależności od stanów wejściowych zmieniać prędkość kół. W moim przypadku rozwiązałem to poprzez różnicę kalibracji i wartości odbitego światła. Robot w zależności od wartości uchybu skręcał aby podążać za granicą linii. Musiałem również zastosować mnożnik uchybu aby algorytm szybciej reagował na zadany stan.

```
def proportional():
    kp = 5
    while cl.value() != 5:
        cl.mode = 'COL-REFLECT'
        e = calibrate() - cl.value()
        left.run_forever(speed_sp=100 - (kp * e))
        right.run_forever(speed_sp=100 + (kp * e))
        cl.mode = 'COL-COLOR'
    left.stop()
    right.stop()
```

Ostatnim programem, który wykonałem był program z regulatorem PID. Był to najcięższy program, który sprawił mi wiele trudności. Regulator PID opiera się na skomplikowanym wzorze składający się z członu proporcjonalnego, członu całkującego oraz członu różniczkującego. Jednak podejście nie zmieniło się i początek programu zacząłem tak jak zwykle aby następnie przejść do wyliczenia alfy.

```
def PID_regulator():
    kp = 4
    suma = 0
    old_e = 0
    ki = 0
    kd = 0
    while cl.value() != 5:
        cl.mode = 'COL-REFLECT'
        e = calibrate() - cl.value()
        suma += e
        alfa = kp * e + ki * suma + kd * old_e
        old_e = e
        left.run_forever(speed_sp=100 - alfa)
        right.run_forever(speed_sp=100 + alfa)
        cl.mode = 'COL-COLOR'
    left.stop()
    right.stop()
```

3. Spostrzeżenia

Pierwsze kroki w tym temacie nie były łatwe. Zrozumienie co to jest uchyb, jak działa rozpoznawanie linii oraz w jaki sposób robot ma się poruszać po wyznaczonej linii były ciężkimi tematami. Jednak po uczestnictwie w laboratoriach doszedłem do kilku wniosków.

Regulator dwustawny jest bardzo dobrym regulatorem na sam początek drogi poznawania w jaki sposób robot ma się poruszać. Jednak ten regulator nie jest najlepszym regulatorem do modelu robotów, które muszą się poruszać. Sposób działania tego regulatora nie pozwala na powolne zwiększanie prędkości pojazdu. Maszyna z takim regulatorem jest bardzo niestabilna i średnio radzi sobie z zakrętami. Wynika to z prostego faktu, że regulator ten zadaje stałą prędkość graniczną. Albo jedno koło będzie kręcić się z maksymalną prędkością albo drugie koło. Natomiast ten regulator będzie świetnie się sprawdzać wszędzie tam gdzie są potrzebne działania jednostkowe, tzn. termostaty, presostaty lub higrostaty. Oczywiście robot z takim regulatorem może jeździć aczkolwiek do większego pojazdu typu Tesla, który ma się samobieżnie poruszać nie jest on odpowiedni gdyż działanie silnika, hamulców i innym części w aucie bardzo szybko by się zniszczyła, a komfort jazdy byłby znikomy.

Regulator proporcjonalny jest zdecydowanie lepszym rozwiązaniem w pojazdach poruszanych się bez ingerencji ludzkiej. Sposób działania tego regulatora jest proporcjonalna zmiana prędkości na zmianę drogi jaką robot musi pokonać. Pojazd, który zbacza z linii zaczyna proporcjonalnie zwiększać prędkość jednego z kół aby pozostać na linii, aczkolwiek nigdy nie będzie w stanie dokładnie jechać po linii. Regulator ten nie jest zbieżny i jak zbliża się do granicy linii to zmniejsza swoją prędkość przez

co pojazd nigdy nie będzie dokładnie na granicy. Robot co chwila reaguje aby poprawić swój błąd i trafić na granicę, lecz nie jest w stanie tego zrobić. Przez co ten regulator również jest dyskwalifikowany w tym zadaniu.

Regulator PID to zdecydowanie najlepsze rozwiązanie do tego typu problemu. Zastosowanie całki oraz różniczki pozwala pozbyć się braku zbieżności, szumów oraz oscylacji. Jednak PID ma jedną wadę, nie gwarantuje stabilności. Pojazd łatwo może się zgubić i nie trafić na trasę, dlatego trzeba umiejętnie dobrać wartości kp,ki,kd aby regulator dobrze działał w danym środowisku. Początkowo mój program nie miał dostosowanych tych wartości, lecz po kilku próbach doszedłem do wartości, które wydaje mi się najlepsze.

4. Poprawiony model

Koniec końców każdy z modeli został poprawiony i czasy w jakich dotarły do mety to:

- Regulator dwustanowy: 37 sekund
- Regulator proporcjonalny: 29 sekund
- Regulator PID: 27 sekund

Oraz ostateczny kod całego programu, gdzie można wybierać program z poziomu systemu wygląda tak :

```python
#!/usr/bin/env python3

from ev3dev.ev3 import *

cl = ColorSensor()
left = LargeMotor(OUTPUT_B)
right = LargeMotor(OUTPUT_C)
btn = Button()


def calibrate():
    print("Na bialy")
    while True:
        if btn.any():
            white = cl.value()
            break

    print("Na czarny")
    while True:
        if btn.any():
            black = cl.value()
            break
    return (white + black) / 2


def two_state():
    while cl.value() != 5:
        cl.mode = 'COL-REFLECT'
        e = calibrate() - cl.value()
        if e < 0:
            left.run_forever(speed_sp=390)
            right.run_forever(speed_sp=150)
        else:
            left.run_forever(speed_sp=150)
            right.run_forever(speed_sp=390)
        cl.mode = 'COL-COLOR'
    left.stop()
    right.stop()
```

```python
def proportional():
    kp = 10
    while cl.value() != 5:
        cl.mode = 'COL-REFLECT'
        e = calibrate() - cl.value()
        left.run_forever(speed_sp=325 - (kp * e))
        right.run_forever(speed_sp=325 + (kp * e))
        cl.mode = 'COL-COLOR'
    left.stop()
    right.stop()


def PID_regulator():
    kp = 15
    suma = 0
    old_e = 0
    ki = 0.2
    kd = 0.5
    while cl.value() != 5:
        cl.mode = 'COL-REFLECT'
        e = calibrate() - cl.value()
        suma += e
        alfa = kp * e + ki * suma + kd * old_e
        old_e = e
        left.run_forever(speed_sp=500 - alfa)
        right.run_forever(speed_sp=500 + alfa)
        cl.mode = 'COL-COLOR'
    left.stop()
    right.stop()


print("Który program chcesz wybrać? ")
print("1. Regulator dwustawny")
print("2. Regulator proporcjonalny")
print("3. Regulator PID")
choose = input()
fun = [two_state(), proportional(), PID_regulator()]
while True:
    if btn.any():
        break
fun[choose]
```

5. Źródła
    - http://mumin.pl/Robotyka/LEGO/Linefollower.html
    - http://marcin.kielczewski.pracownik.put.poznan.pl/ZSP06.pdf
    - http://www.gdzie.pl/mczakan/15_regulatory.pdf
    - Wikipedia
    - Wykład