Sprawozdanie nr 2. „Napęd różnicowy i odometria”

---

1. Teoria

Kolejny element, który trzeba zrozumieć przy poruszaniu się robota, to jak określa on swoje położenie w środowisku. Jednym z podstawowych elementów obliczania jak daleką drogę przebył pojazd jest optyczny enkoder obrotowy. Enkoder zlicza obroty kół od włączenia silnika. Znając liczbę impulsów od włączenia silnika, liczbę impulsów w bieżącym kroku, liczbę impulsów z poprzedniego kroku oraz promień koła możemy obliczyć drogę jaką przebyła maszyna. Wtedy znając drogę przebytą przez lewe jak i prawe koło wystarczy wyliczyć średnią arytmetyczną obu kół. Jedyny problem w takim modelu jest poślizg kół, który nie jest zliczany.

2. Pierwsze kroki

Sam główny program nie jest trudny. Warto jednak pamiętać, że poślizg kół nie zawsze doprowadzi robota do wyznaczonego miejsca. Warto więc dodać pewną niedokładność do ustalonego wyniku. Ja osobiście ustawiłem drift na 15 aby na pewno robot w początkowej fazie dotarł do miejsca wyznaczonego przeze mnie. Funkcja będzie działać dopóki robot nie dotrze do celu z obsługą +- 15 cm. W swoim programie również zastosowałem odświeżanie ekranu aby widzieć na bieżąco w jakiej pozycji według robota się znajduje. Pierwszą funkcją jest wywołanie pozycji, aby maszyna umiała zorientować się jak daleko przejechała. Następnie jest wyliczany uchyb, który musi skorygować aby dotrzeć do celu. Końcowy program to funkcja jedz, która pozwala robotowi jechać. Program kończy się zatrzymaniem kół.

```
def move_target(self, goal):
    drift = 15
    while not (abs(self.get_x() - goal[0]) < drift and abs(self.get_y() - goal[1]) < drift):
        lcd.clear()
        lcd.draw.text((6, 55), 'x:{},y:{}'.format(int(self.get_x()), int(self.get_y())))
        lcd.update()
        print(int(self.get_x()))
        print(int(self.get_y()))
        self.position()
        e = self.go_to_goal(goal[0], goal[1])
        self.run(e)
    self.stop()
```

Początkowo napisanie klasy Robot było dla mnie priorytetem, gdzie w konstruktorze utworzyłem wszystkie potrzebne zmienne aby w późniejszym czasie móc się do nich odwoływać i nie zapominać potrzebnych danych.

```
class Robot:
    def __init__(self):
        self.motor_right = LargeMotor(OUTPUT_B)
        self.motor_left = LargeMotor(OUTPUT_C)
        self.position_motor_old_right = 0
        self.position_motor_old_left = 0
        self.x = 0
        self.y = 0
        self.fi = 0
        self.R = 2.8
        self.L = 12
```

Następnie utworzyłem funkcję, która będzie wyliczać pozycję robota według wzoru. Ważne jest aby, pobierać ilość impulsów silnika i wyliczenie dla obu kół pozycję z średniej arytmetycznej. Następnie zaktualizować pozycję robota aby kolejne funkcje korzystały już z aktualnej pozycji. Końcowo warto również zapisać ilość impulsów aby w następnym kroku móc znowu wyliczyć długość drogi jaką pokonaliśmy.

```
def position(self):
    n_r = self.motor_right.position
    n_l = self.motor_left.position
    delta_n_r = n_r - self.get_position_motor_old_right()
    delta_n_l = n_l - self.get_position_motor_old_left()

    D_r = 2 * math.pi * self.get_R() * (delta_n_r / 360)
    D_l = 2 * math.pi * self.get_R() * (delta_n_l / 360)

    Dc = (D_r + D_l) / 2

    self.x = self.get_x() + (Dc * math.cos(self.get_fi()))
    self.y = self.get_y() + (Dc * math.sin(self.get_fi()))
    self.fi = self.get_fi() + ((D_r - D_l) / self.L)

    self.position_motor_old_right = n_r
    self.position_motor_old_left = n_l
```

Kolejnym elementem jest wyliczenie uchybu również z wzoru. W moim programie są do tego przeznaczone 2 funkcje. Pierwsza wyliczająca Psi, a druga wyliczająca sam uchyb. Funkcje te potrzebują celu do którego maszyna chce dotrzeć.

```
def angle_to_goal(self, xg, yg):
    psi = math.atan2(yg - self.get_y(), xg - self.get_x())
    return psi

def go_to_goal(self, xg, yg):
    e = self.angle_to_goal(xg, yg) - self.get_fi()
    return e
```

Ostatnimi funkcjami jest jedz oraz stój odpowiedzialne kolejno za jazdę robota i zatrzymanie robota jeśli dotrze do celu. W funkcji jedź zastosowałem regulator proporcjonalny. Nie zależało mi na dokładności poruszającego się robota, lecz na dojechaniu do celu.

```
def run(self, e):
    kp = 68
    self.motor_left.run_forever(speed_sp=400 - (kp * e))
    self.motor_right.run_forever(speed_sp=400 + (kp * e))

def stop(self):
    self.motor_right.stop()
    self.motor_left.stop()
```

3. Spostrzeżenia

Na sam początek robot nie chciał dojechać do celu. Sprawdzając liczbę jaką maszyna miała już przejechane okazało się, że robot jej nie resetuje po starcie programu. Jest to naturalne ponieważ enkoder zlicza liczbę impulsów od startu urządzenia, a nie od wybrania konkretnego programu. Więc pierwszym krokiem było resetowanie liczby impulsów za każdym razem gdy program zostanie uruchomiony.
Kolejnym elementem, który musiałem poprawić było przepięcie silników robota. Okazało się, że robot źle skręcał. Miałem do dyspozycji 2 możliwości. Jedną z możliwości było poprawienie funkcji run. Drugą możliwością było przepięcie fizyczne silników. Wybrałem opcję 2 i pojazd już dobrze skręcał.
Ostatnimi elementami było dopisanie funkcji, która nie pozwoli aby kąty wychodziły po za [-π, π]. Wystarczyło zastosować funkcję `e = math.atan2(math.sin(e), math.cos(e))`

oraz poprawić kp w funkcji run, aby pojazd poruszał się szybciej.

4. Poprawiony model

Ostateczny program działa i przekracza wszelkie punkty, które zostały wyznaczone. Program wygląda następująco:

```python
from ev3dev.ev3 import *
from time import sleep
import math


class Robot:
    def __init__(self):
        self.motor_right = LargeMotor(OUTPUT_B)
        self.motor_left = LargeMotor(OUTPUT_C)
        self.motor_right.position = 0
        self.motor_left.position = 0
        self.position_motor_old_right = 0
        self.position_motor_old_left = 0
        self.x = 0
        self.y = 0
        self.fi = 0
        self.R = 2.8
        self.L = 12

    def position(self):
        n_r = self.motor_right.position
        n_l = self.motor_left.position
        delta_n_r = n_r - self.get_position_motor_old_right()
        delta_n_l = n_l - self.get_position_motor_old_left()

        D_r = 2 * math.pi * self.get_R() * (delta_n_r / 360)
        D_l = 2 * math.pi * self.get_R() * (delta_n_l / 360)

        Dc = (D_r + D_l) / 2

        self.x = self.get_x() + (Dc * math.cos(self.get_fi()))
        self.y = self.get_y() + (Dc * math.sin(self.get_fi()))
        self.fi = self.get_fi() + ((D_r - D_l) / self.L)

        self.position_motor_old_right = n_r
        self.position_motor_old_left = n_l
```

```
def angle_to_goal(self, xg, yg):
    psi = math.atan2(yg - self.get_y(), xg - self.get_x())
    return psi

def go_to_goal(self, xg, yg):
    e = self.angle_to_goal(xg, yg) - self.get_fi()
    e = math.atan2(math.sin(e), math.cos(e))
    return e

def run(self, e):
    kp = 68
    self.motor_left.run_forever(speed_sp=400 - (kp * e))
    self.motor_right.run_forever(speed_sp=400 + (kp * e))

def stop(self):
    self.motor_right.stop()
    self.motor_left.stop()

def move_target(self, goal):
    drift = 3
    while not (abs(self.get_x() - goal[0]) < drift and abs(self.get_y() - goal[1]) < drift):
        lcd.clear()
        lcd.draw.text((6, 55), 'x:{},y:{}'.format(int(self.get_x()), int(self.get_y())))
        lcd.update()
        print(int(self.get_x()))
        print(int(self.get_y()))
        self.position()
        e = self.go_to_goal(goal[0], goal[1])
        self.run(e)
    self.stop()

def get_position_motor_old_right(self):
    return self.position_motor_old_right

def get_position_motor_old_left(self):
    return self.position_motor_old_left

def get_fi(self):
    return self.fi
```

```
    def get_R(self):
        return self.R

    def get_x(self):
        return self.x

    def get_y(self):
        return self.y


wspolrzedne = [[200, 0], [125, 100], [125, 200]]
robot = Robot()
sleep(5)
for i in wspolrzedne:
    robot.move_target(i)
```

5. Źródła

- Wykład
- http://mumin.pl/Robotyka/LEGO/Napedroznicowy.html
- Wikipedia