LEGO Mindstorms EV3 w Pythonie

Repozytorium z kodami: https://github.com/begreeen/SerockRobotyczne2019

Wprowadzenie

Na zajęciach korzystamy z zestawów LEGO Mindstorms EV3 na których uruchamiamy alternatywny system operacyjny oparty o dystrybucję Linuksa Debian – ev3dev – https://www.ev3dev.org/. Można go bezpłatnie pobrać ze strony projektu. System zapisany jest na karcie micro SD z której jest uruchamiany przez komputer LEGO Mindstorms EV3. W celu zapewnienia łączności bezprzewodowej do portu USB wpinamy karty WiFi (mogą być dowolne wspierane przez tego Linuksa). W przypadku braku karty można użyć połączenia przez Bluetooth lub USB (w takich przypadkach w komputerze EV3 również będzie wykrywany jako adapter sieciowy).

Przydatne informacje dla języka Python dla EV3 znajdują się na tej stronie:

https://github.com/ev3dev/ev3dev-lang-python

Do pisania programów używamy środowiska Visual Studio Code (również darmowe) z wtyczką ev3dev-browser – instalacja opisana w linku podanym powyżej.

Regulator PID

Na naszych warsztatach doszliśmy do opanowania dwóch z trzech członów **regulatora PID**, czyli regulatora proporcjonalno-całkująco-różniczkującego. Nasz robot – linefollower – miał za zadanie jak najlepiej jeździć po krawędzi czarnej linii. Na początku zaimplementowaliśmy prosty regulator dwustanowy – robot widząc czarną linię skręcał w przeciwną stronę, aż napotkał na białe podłoże. Wtedy sytuacja się odwracała i znów szukał czarnej linii. Prostota algorytmu jest jego niewątpliwą zaletą, jednak roboty jeździły w sposób bardzo nierówny („zygzakowaty"). Pierwszą ideą upłynnienia ruchu było sterowanie mocą skrętu silników w sposób proporcjonalny. Wykorzystaliśmy do tego człon P regulatora – gdy uchyb (czyli różnica pomiędzy aktualną wartością odczytywaną z czujnika światła, a wartością oczekiwaną) był duży – robot skręcał szybko, aby zminimalizować uchyb, czyli trafić na krawędź linii. Gdy robot był blisko krawędzi, czyli uchyb był niewielki – jechał prosto lub dokonywał niewielkich korekt. Kolejny człon regulatora – całkujący – pozwalał na kompensację uchybów, które zakumulowały się w przeszłości (dokonywaliśmy sumowania poprzednich uchybów).

Poglądowo działanie tych członów w odniesieniu do czasu można zinterpretować następująco:

- działanie członu P kompensuje uchyb bieżący,
- człon I kompensuje akumulację uchybów z przeszłości,
- człon D kompensuje przewidywane uchyby w przyszłości.

Inne kursy robotyki

Dobry kurs budowania robotów, oparty na platformie Arduino, która jest bardziej dostępna (również cenowo) niż LEGO można znaleźć na forum Forbot: https://forbot.pl/blog/kurs-budowy-robotow-arduino-wstep-spis-tresci-id18935.

Wstępne programy / ćwiczenia

1. Szkielet programu

- Wszystkie pliki programów powinny mieć rozszerzenie .py – przykładowa nazwa nowego pliku to np. `program.py`
- Do obsługi LEGO Mindstorms będziemy musieli dołączyć niezbędne biblioteki – od tego zacznie się każdy nasz program:

```
#!/usr/bin/env python3
from ev3dev2.motor import *
from ev3dev2.sensor import *
from ev3dev2.sensor.lego import *
from ev3dev2.led import *
from ev3dev2.sound import *

import time

# TUTAJ WPISZ KOD PROGRAMU
```

2. Pierwszy program w Pythonie – jazda robotem do przodu

```
#!/usr/bin/env python3
from ev3dev2.motor import *
from ev3dev2.sensor import *
from ev3dev2.sensor.lego import *
from ev3dev2.led import *
from ev3dev2.sound import *

import time
```

```
# wyświetl tekst na terminalu
print("Zaczynamy zabawe!")

# silniki podłączone do wyjścia A oraz B
motor_tank = MoveTank(OUTPUT_A, OUTPUT_B)

# wlacz silnik A na moc 50%, silnik B na 50%
motor_tank.on(50, 50, liczba_obrotow)

time.sleep(2)

# wylacz silniki
motor_tank.off()

print("Koniec programu!")
```

3. Zakręcamy robotem

```
#!/usr/bin/env python3
from ev3dev2.motor import *
from ev3dev2.sensor import *
from ev3dev2.sensor.lego import *
from ev3dev2.led import *
from ev3dev2.sound import *

import time

# silniki podłączone do wyjścia A oraz B
motor_tank = MoveTank(OUTPUT_A, OUTPUT_B)

# wyświetl tekst na terminalu
print("Zakrecamy w jedna strone!")

# zakrecamy w jedna strone - jeden silnik na 10%, drugi na 90% mocy
motor_tank.on(10, 90)

# poczekaj 2 sekundy, a nastepnie wylacz silniki
time.sleep(2)
motor_tank.off()
```

```
time.sleep(5)

# wyświetl tekst na terminalu
print("Zakrecamy w druga strone!")

# zakrecamy w druga strone - jeden silnik na 90%, drugi na 10% mocy
motor_tank.on(90, 10)
time.sleep(2)
motor_tank.off()

print("Koniec programu!")
```

4. Odczytajmy wartości czujnika światła odbitego

```
#!/usr/bin/env python3
from ev3dev2.motor import *
from ev3dev2.sensor import *
from ev3dev2.sensor.lego import *
from ev3dev2.led import *
from ev3dev2.sound import *

import time

# tworzymy obiekt czujnika swiatla - tutaj podlaczony do wejscia 4
light_sensor = ColorSensor(INPUT_4)

# nieskonczona petla while
while True:
    # odczytaj wartosc i zapisz do zmiennej light_value
    light_value = light_sensor.reflected_light_intensity

    # wyswietl na teminalu zmienna light value
    print(light_value)

    # poczekaj 2 sekundy pomiedz odczytami
    time.sleep(2)
```

- Jak zmieniają się wskazania czujnika, gdy „patrzy” on na białą kartkę, a jak gdy patrzy na ciemne obiekty?

5. Silnik włączany światłem

```
#!/usr/bin/env python3
from ev3dev2.motor import *
from ev3dev2.sensor import *
from ev3dev2.sensor.lego import *
from ev3dev2.led import *
from ev3dev2.sound import *

import time

# tworzymy obiekt czujnika swiatla - tutaj podlaczony do wejscia 4
light_sensor = ColorSensor(INPUT_4)

# silniki podłączone do wyjścia A oraz B
motor_tank = MoveTank(OUTPUT_A, OUTPUT_B)


# nieskonczona petla while
while True:
    # odczytaj wartosc i zapisz do zmiennej light_value
    light_value = light_sensor.reflected_light_intensity

    if light_value > 50:
        # wlacz oba silniki jesli wartosc light_value jest wieksza niz 50
        motor_tank.on(100, 100)
    else:
        # wylacz oba silniki jesli wartosc jest mniejsza lub rowna 50
        motor_tank.off()

    # poczekaj 0.5 sekundy pomiedzy odczytami
    time.sleep(0.5)
```

6. Szybkość obrotowa silników kontrolowana światłem

- W zależności od natężenia światła będziemy kontrolować szybkość silników!

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

```
from ev3dev2.motor import *
from ev3dev2.sensor import *
from ev3dev2.sensor.lego import *
from ev3dev2.led import *
from ev3dev2.sound import *

import time

# tworzymy obiekt czujnika swiatla - tutaj podlaczony do wejscia 4
light_sensor = ColorSensor(INPUT_4)

# silniki podłączone do wyjścia A oraz B
motor_tank = MoveTank(OUTPUT_A, OUTPUT_B)


# nieskonczona petla while
while True:
    # odczytaj wartosc i zapisz do zmiennej light_value
    light_value = light_sensor.reflected_light_intensity

    # jeden silnik w jedną strone - drugi w druga strone!
    motor_tank.on(light_value, -light_value)

    # poczekaj 0.5 sekundy pomiedzy odczytami
    time.sleep(0.5)
```

7. Funkcje dotyczące silników

- wszystkie dostępne funkcje: https://python-ev3dev.readthedocs.io/en/ev3dev-stretch/motors.html

- `on_for_seconds(moc_1, moc_2, czas_s)`

- `on_for_rotations(predkosc_1, predkosc_2, liczba_obrotow)`

8. Funkcje dotyczące sensorów

- wszystkie dostępne funkcje: https://python-ev3dev.readthedocs.io/en/ev3dev-stretch/sensors.html

a. czujnik dotyku (przycisk)

```
inicjalizacja:
touch = TouchSensor(INPUT_1)

sprawdzanie, czy jest wciśnięty – zwraca True albo False
touch.is_pressed
```

b. czujnik czujnik odległości (ultradźwiękowy)

```
inicjalizacja:
distance = UltrasonicSensor(INPUT_1)


odczytanie wartości odległości w cm:
distance.distance_centimeters()
```

c. czujnik żyroskopowy

```
inicjalizacja:

gyro = GyroSensor(INPUT_1)

odczytanie kąta odbrotu:
gyro.angle()

odczytanie prędkości obrotowej:
gyro.rate()
```

9. Syntezator mowy

```
inicjalizacja:
sound = Sound()

synteza mowy w języku polskim:
sound.speak('Krajowy Fundusz na rzecz Dzieci!', espeak_opts='-a 200 -s 130
-vpl')

synteza mowy w języku angielskim:
```

```
sound.speak('Hello!')
```