Dokumentacja projektu – Wstęp do Robotyki Laboratorium

Filip Szczygielski 318405

Mikołaj Wewiór 318407

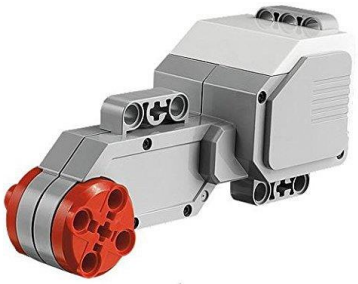
LINEFOLLOWER

Roboty typu „linefollower” mają za zadanie przejechanie wyznaczonej trasy podążając za linią namalowaną na ziemi. Wykorzystuje do tego czujniki koloru (w naszym przypadku natężenia światła), aby śledzić linię i odpowiednio przyspieszać silniki w celu podążania za trasą.

W naszym projekcie wykorzystaliśmy dwa duże silniki (tak naprawdę to serwomechanizmy, ale stosujemy tutaj nazwę silników dla wygody) i dwa czujniki światła (koloru).

Specyfikacja dużych silników:

- Prędkość obrotowa: 160-170 obr/min
- Moment obrotowy: 20 Ncm
- Moment utyku: 40 Ncm



Specyfikacja sensorów koloru:

- Częstotliwość próbkowania: 1 kHz
- Czujnik działa w jednym z trzech możliwych trybów: wykrywanie kolorów, natężenie światła odbitego, natężenie światła otoczenia

W programie kierującym robotem wykorzystaliśmy algorytm PID, który korzysta ze wzoru:

$$u(t) = k_p e(t) + k_i \int e(t)dt + k_d \frac{de(t)}{dt}$$

Pierwsza część równania – proporcjonalna - odpowiada za wyliczanie aktualnego błędu, część całkująca odpowiada za analizowanie błędów z przeszłości, a część różniczkowa odpowiada za przewidywanie błędów w przyszłości. Całe równanie zwraca uchyb, który należy skorygować.

W naszym przykładzie funkcje e(t) uzyskaliśmy odejmując od siebie natężenia światła odbitego z dwóch sensorów (od wartości lewego sensora odejmujemy wartość z prawego), które są ustawione po obu stornach linii. Współczynniki k uzyskaliśmy metodą prób i błędów.

```
while temp is True:

    if touch.is_pressed:
        temp = stopping()
    else:
        left_value = l_color.reflected_light_intensity + 5
        right_value = r_color.reflected_light_intensity
        error = left_value - right_value
        if error < 5:
            integ = 0
        else:
            integ += (last_error + error)

        deriv = (error - last_error)
        correction = (error * Kp) + (integ * Ki) + (deriv * Kd)

        velo_minus = constrain(set_speed - correction, min_speed, max_speed)
        velo_plus = constrain(set_speed + correction, min_speed, max_speed)

        if(correction < -5):
            move.on(-0.4*velo_minus, velo_minus)
        elif(correction > 5):
            move.on(velo_plus, -0.4*velo_plus)
        else:
            move.on(set_speed, set_speed)

        last_error = error
```

W naszym algorytmie całkę (`integ`) wyliczamy wykorzystując uproszczoną metodę trapezów – dodajemy do siebie aktualną wartość błędu do poprzedniej sumy błędów (jeżeli błąd jest mniejszy od zadanej wartości, zerujemy wartość członu całkującego). Według teorii powinniśmy podzielić ją przez dwa oraz wymnożyć przez pewną wartość dt. Jednak te współczynniki uwzględniamy już w k_i.

Z kolei różniczkę (`deriv`) wyliczamy jako iloczyn różnicowy aktualnego i ostatniego błędu. Tutaj podobnie jak w poprzednim przypadku różnicę pomiędzy momentami próbkowania uwzględniamy już we współczynniku k_d.

Po wyliczeniu uchybu (`correction`) następnym krokiem jest odpowiednie przyspieszanie silników. W przypadku dodatniego uchybu przyspieszamy lewy silnik, a prawy uruchamiamy do tyłu, w przypadku ujemnego uchybu silniki przyspieszamy na odwrót (tzn. z ujemnym znakiem).

Ostatecznie nasze współczynniki przyjmują następujące wartości:

```
Kp = 0.22
Ki = 0.06
Kd = 1.3
```

Funkcja `constrain()` pobiera trzy wartości i zwraca tą, która nie jest ani najmniejsza ani największa. Działa ona jako zabezpieczenie w naszym programie przeciw nie pożądanym wartościom.

```
def constrain(val, min_val, max_val):
    return min(max_val, max(min_val, val))
```

Funkcja `stopping()` służy do zatrzymania pojazdu za pomocą dołączonego do niego przycisku i zagrania krótkiej melodii po fakcie.

```
def stopping():
    l_motor.off()
    r_motor.off()
    move.off()
    sound.play_song((
        ('E5', 'e'), ('E5', 'e'), ('C5', 'e')
    ))
    return False
```

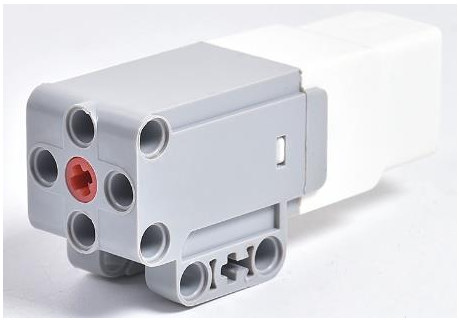
TRANSPORTER

Robot transporter składa się z dwóch modułów – części jeżdżącej oraz podnośnika. Pierwsza z nich oparta jest na algorytmie wykorzystanym w linefollowerze z poprzedniego zadania, jednak z małą korektą. Mianowicie w trakcie jazdy sensory na zmianę próbkują natężenie światła i interpretują kolor nawierzchni. Z kolei część podnośnika wykorzystuje serwomechanizm, którego zadaniem jest podniesienie obiektu i jego chwilowe zablokowanie w trakcie transportu. Dodatkowo korzystamy z czujnika podczerwieni, który służy do wykrycia odległości od obiektu do przechwycenia i aktywacji serwomechanizmu.

Poza elementami, które wykorzystaliśmy w linefollowerze do transportera użyliśmy dodatkowo średni silnik do podnoszenia obiektu i czujnik podczerwieni do badania odległości od obiektu.

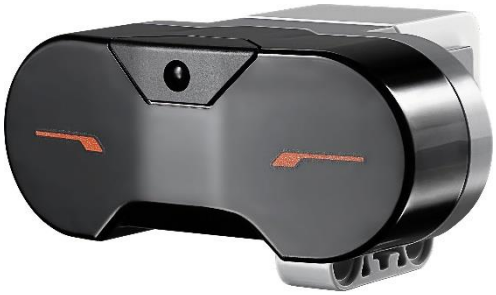
Specyfikacja średniego silnika (dalej nazywany serwomechanizmem):

- Prędkość obrotowa: 240-250 obr/min
- Moment obrotowy: 8 Ncm
- Moment utyku: 12 Ncm



Specyfikacja czujnika podczerwieni:

- Czujnik może pracować w dwóch trybach: Proximity Mode i Beacon Mode
- Przy Proximity Mode czujnik wykrywa obiekt z odległości do 70cm



Działanie transportera można przedstawić jako automat stanów:

1) Linefollowing oraz oczekiwanie na wykrycie czerwonego wjazdu
2) Wjazd po koszyczek na czerwone pole oraz przechwycenie go
3) Wyjazd z czerwonego pola po zabraniu koszyczka
4) Linefollowing oraz oczekiwanie na wykrycie niebieskiego wjazdu
5) Wjazd na niebieskie pole oraz odstawienie koszyka

Algorytm odpowiadający za sterowanie pojazdem w 1. stanie praktycznie się nie zmienił względem „czystego” linefollowera. Jedynym dodatkiem jest sprawdzenie czy pod czujnikami nie występuje kolor czerwony. Dodatkowo potrzebne były zmiany współczynników odpowiadających za sterowanie robotem:

```
if r_color.color == 5:
    package = entry('R')
    package = pack_on(temp, package, 0, 0, 0, min_speed, max_speed)

if l_color.color == 5:
    package = entry('L')
    package = pack_on(temp, package, 0, 0, 0, min_speed, max_speed)
```

```
Kp = 0.33
Ki = 0.06
Kd = 0.40

max_speed = 100
min_speed = 0
set_speed = 18
```

Gdy robot wykryje czerwone pole rozpoczyna zakodowaną sekwencję, w której wykonuje obrót w odpowiednią stronę a następnie cofa aż do momentu wykrycia koszyka:

```
def entry(side):
    if side == 'R':
        L = 1
        R = -1
    if side == 'L':
        L = -1
        R = 1
    move.off()
    speed = 10
    move.on_for_seconds(speed, speed, 1)
    move.on_for_degrees(-10*L, -10*R, 348)
    temp = True
    temp2 = True
    while temp:
        move.on(-1*speed, -1*speed)
        if dist():
            move.on_for_degrees(-0.5*speed, -0.5*speed, 90)
            holder()
            temp = False
    while temp2:
        move.on(speed, speed)
        if l_color.reflected_light_intensity < 20 and r_color.reflected_light_intensity < 20:
            move.on_for_seconds(speed, speed, 1)
            move.on_for_degrees(10*L, 10*R, 348)
            temp2 = False
```

Wykrycie koszyka wykorzystuje czujnik podczerwieni, który przekazuje informacje jak daleko znajduje się obiekt. Zawarte zostało to w funkcji `dist()`.

```
def dist():
    if infra.proximity == 1:
        move.off()
        l_motor.off()
        r_motor.off()
        return True
```

Następnie robot podjeżdża jeszcze, aby na pewno być przed przedmiotem i korzystając z funkcji `holder()` podnosi koszyk z wykorzystaniem serwomechanizmu

```
def holder(sign=0):
    if sign < 0:
        sign = -1
    else:
        sign = 1
    sleep(1)
    servo.on_for_degrees(20, sign*180)
    sleep(1)
    move.on(set_speed, set_speed)
    return False
```

W momencie przechwycenia koszyka robot jedzie do przodu aż wykryje oboma czujnikami, że wrócił na tor. Wtedy rozpoczyna się funkcja `pack_on()`, czyli stan 4. Ta funkcja to znów ten sam algorytm linefollowingu co wcześniej jednak tym razem zamiast wykrywania koloru czerwonego, badamy czy pod czujnikami nie ma koloru niebieskiego.

Jest to jednak większy problem niż w przypadku wykrycia czerwonego pola. Mianowicie przy próbach implementacji tej części, okazało się, że czujniki mają trudność z wykryciem niebieskiego koloru. Często mylą go z barwą czarną oraz białą, albo nawet zieloną. Zdarzało się tak, że robot wykrywał niebieski kolor w trakcie pokonywania zwykłych (czarno-białych) odcinków trasy testowej, albo nie wykrywał go gdy przejeżdżał przez właściwy element. Dodatkowym utrudnieniem były różne stany czujników światła, zdarzało się, że dla bardzo podobnych warunków czujnik lewy i prawy pokazywały różne odczyty. Z tego powodu zastosowaliśmy podwójne sprawdzenie czy wykryty kolor na pewno jest niebieskim. W tym celu skorzystaliśmy z możliwości sprawdzenia wartości pojedynczych barw z modułu RGB oraz wykorzystania modułu rozpoznawającego kolory nad którym się znajdują czujniki.

To skutkowało większymi nakładami czasowymi na sprawdzanie wszystkich potrzebnych warunków, co z kolei rzutowało na zmniejszony okres próbkowania. Dlatego było trzeba ponownie ustawić nowe parametry do algorytmu PID.

```
if (60 < r_color.rgb[2] and r_color.rgb[2] < 160) and (60 < r_color.rgb[1] and r_color.rgb[1] < 90):
    move.off()
    sleep(1)
    move.off()
    if (r_color.color == 2 or r_color.color == 3) and r_color.color != 1 and r_color.color != 6:
        package = entry_with_package('R')
        temp = True
        package = 0
        servo.off()
    else:
        pack_on(temp, package, 0, 0, 0, min_speed, max_speed)

if (25 < l_color.rgb[2] and l_color.rgb[2] < 130) and (20 < l_color.rgb[1] and l_color.rgb[1] < 50):
    move.off()
    sleep(1)
    move.off()
    if (l_color.color == 2 or l_color.color == 3) and l_color != 1 and l_color != 6:
        package = entry_with_package('L')
        temp = True
        package = 0
    else:
        pack_on(temp, package, 0, 0, 0, min_speed, max_speed)
```

```
Kp_p = 0.33
Ki_p = 0.05
Kd_p = 0.4
package_speed = 12
```

Gdy po podwójnym sprawdzeniu barwy program uznał, że jest to barwa niebieska robot wchodził w stan odstawienia paczki.

```
def entry_with_package(side):
    if side == 'R':
        L = 1
        R = -1
    if side == 'L':
        L = -1
        R = 1
    move.off()
    speed = 10
    move.on_for_seconds(speed, speed, 1)
    move.on_for_degrees(-10*L, -10*R, 348)
    move.on_for_seconds(-1*speed, -1*speed, 5)
    move.on_for_degrees(-0.5*speed, -0.5*speed, 90)
    holder(-1)
    servo.off()
    return 0
```

W tym momencie program kończy się, można jednak dodać drobną aktualizację, aby powrócić do fragmentu linefollowingu.