Linefollower i Transporter

Autorzy: Wojciech Styczeń, Jakub Sadowski

Budowa Robota

Projekt wykonany został z klocków Lego Mindstorms. Sterownikiem użytym w robocie jest kostka ev3, programowana przy użyciu języka Python oraz biblioteki *ev3dev*.

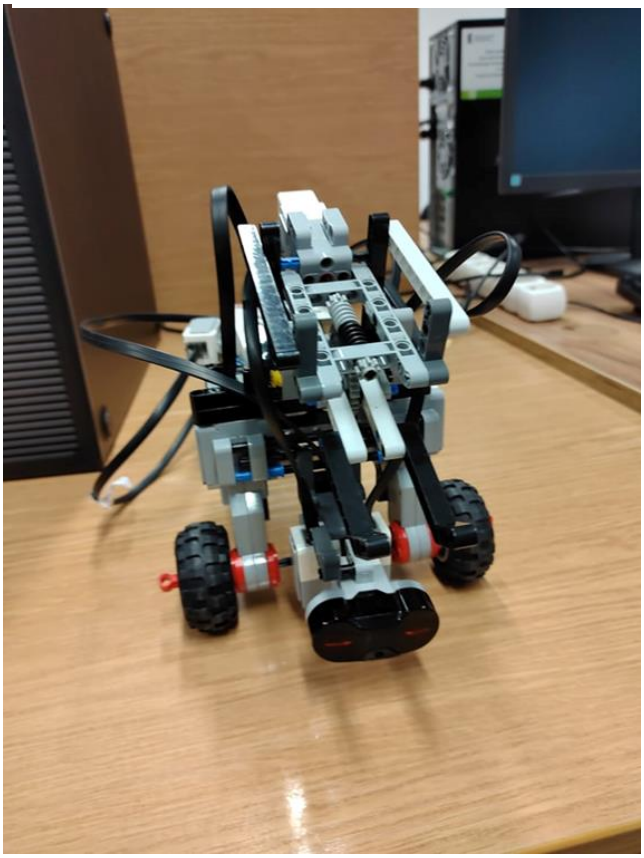
Układ napędowy wykonany został z dwóch dużych serwomotorów. Każdy z nich napędza po jednym kole znajdującym się na przedniej osi. Na tylnej osi znajduje się koło podporowe będące metalową kulką.

Z przodu robota nisko nad ziemią umieszczone zostały dwa czujniki kolorów które podczas realizacji linefolowera działają w trybie sprawdzania refleksu świetlnego, a podczas realizacji transportera dodatkowo przy sprawdzaniu koloru przełączają się w tryb RGB.

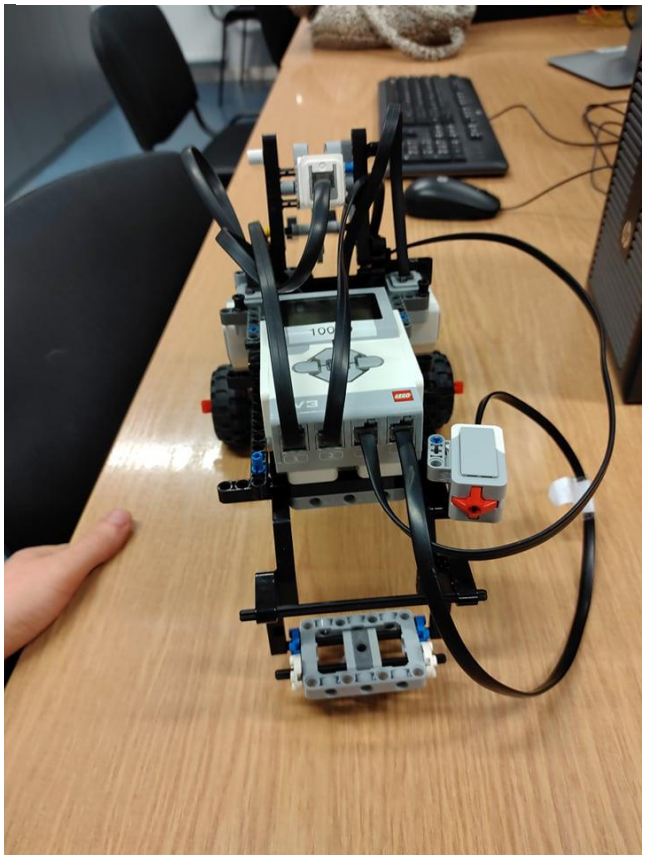
Powyżej czujników koloru zamontowany został czujnik podczerwieni służący do wykrycia klocka, który ma zostać podniesiony podczas realizacji zadania transporter. Również z przodu zamontowany został podnośnik do klocka. Do jego napędu został wykorzystany średniej wielkości serwomotor oraz przekładnia ślimakowa. Dzięki takiemu zastosowaniu po podniesieniu klocka można wyłączyć serwomotor, a klocek zostanie w tej samej pozycji. Oznacza to, że taka konstrukcja oszczędza energię oraz nie powoduje zbytniego obciążenia serwomotoru. Konstrukcja ta została przetestowana i działa poprawnie.

Z prawej strony robota, patrząc z tyłu, znajduje się przycisk. Służy on do włączania oraz wyłączania działania robota.

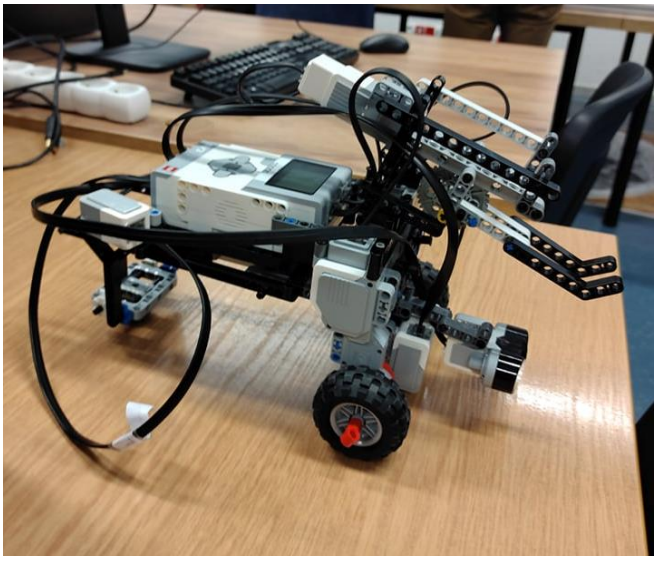
Przód robota



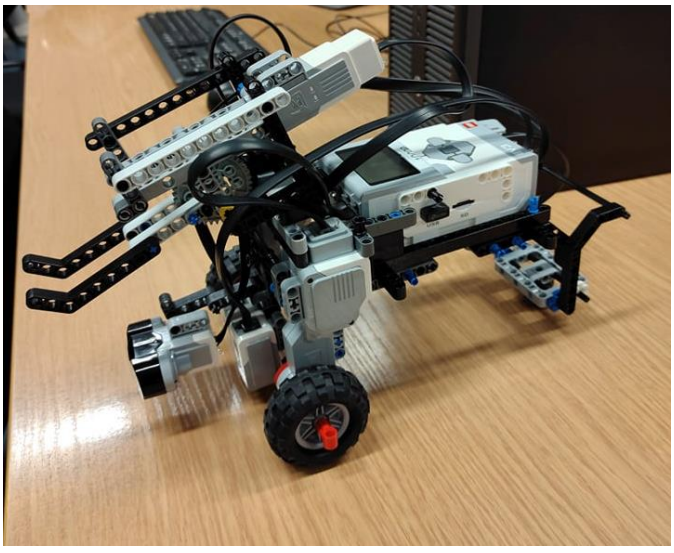
Tył robota



Prawy bok



Lewy bok



Linefollower

Podczas realizacji linefollowera czujniki kolorów ustawione są na tryb pomiaru odbicia refleksu. Działają one w taki sposób że zwracają wartość z zakresu 0-100 gdzie 0 nie ma refleksu (cały kolor czarny) i 100 jest maksymalny refleks (cały kolor biały). Ponieważ działamy na czujnikach rzeczywistych wartości graniczne są nieosiągalne i będą bliżej wartości środkowych. Czujniki ustawione są w taki sposób, że obejmują połowę koloru białego i połowę czarnego.

W celu sprawdzenia czy jedziemy zgodnie z czarną linią obliczamy różnicę lewego czujnika i prawego. Czujniki nie pokazują takich samych wartości, a ponadto dla małych różnic robot nie powinien zmieniać toru ruchu dlatego też sprawdziliśmy, że dla różnicy w zakresie <-10;10> serwomotory powinny poruszać się ze stałą prędkością. Powoduje to, że robot na prostej drodze potrafi jechać prosto bez wykonywania zbędnych ruchów.

Dla różnicy w przedziale (10;20) i (-20;-10) układ wykonuje niewielkie skręty korygujące. Dla wartości z zakresu (-50;-20> i <20;50) wykonywane są skręty mocniejsze. Ponadto w przypadku różnicy większej lub równej 50 bądź też mniejszej lub równej -50 lub wykrycia tylko przez jeden czujnik całkowitego koloru czarnego (dla lewego czujnika wartość mniejsza od 6 a dla prawego mniejsza od 8) robot wykonuje obrót w miejscu. Krok ten powoduje, że nawet dla ostrych zakrętów układ skręca poprawnie. Te wartości zostały dobrane na podstawie testów.

Trudności podczas tworzenia tego zadania napotkaliśmy przy testowaniu ruchu. Pominięcie któregoś z etapu ruchu miało negatywny efekt. Brak dodania członu odpowiedzialnego za jazdę prosto skutkowało mniej płynną jazdą. Nie dodanie rotacji w miejscu doprowadzało na mocniejszych skrętach do wyjeżdżania poza linię, w szczególności jest ona konieczna do pokonywania zakrętów prostokątnych.

Transporter

W projekcie Transporter do poruszania się po torze korzystamy z tego samego kodu co w linefollowerze, z pewnymi dodatkami.

Dla obrotu w miejscu sprawdzamy dodatkowo czy nie jest to zakręt koloru niebieskiego, ponieważ kolor niebieski daje zbliżone odczyty z czujników w trybie refleksu. W ten sposób na każdym pojawiającym się ostrym skręcie sprawdzamy czy ścieżka w prawo bądź też w lewo ma kolor niebieski i jeżeli posiada taki kolor to wykonujemy w miejscu skręt o kąt 90° .

Przy skrętach czerwonych korzystamy z tego że odczytywany refleks jest mniejszy od standardowego. Wtedy też zmieniamy tryb czujników na tryb rgb i sprawdzamy czy zatrzymaliśmy się na skręcie czerwonym. Jeśli jest to rzeczywiście skręt czerwony to podobnie wykonujemy obrót o 90° w daną stronę i podjeżdżamy do przodu żeby z powrotem znaleźć się w pełni na torze.

Ta część programu została przez nas sprawdzona i działa w większości przypadków poprawnie. Również przetestowany został podnośnik, który realizuje swoje zadanie. Także działa rozpoznanie pola niebieskiego oraz obrót i wyjazd z niego. Te części działały przy osobnym testowaniu, ale nie udało nam się przetestować manewru odebrania klocka w całości.

Dalsza część kodu nie została przez nas w pełni przetestowana i zapewne działa błędnie z powodu braku dobrania parametrów – prędkości, czasu, wartości wykrywanych przez czujniki. itd. Z tego powodu mogą wystąpić błędy typu: zatrzymanie się w złym miejscu – za daleko lub za blisko klocka, problemy z powrotem na tor przy wyjeździe i temu podobne. Dalej opisujemy zachowanie robota takie jak było w naszym ‘zamyśle’, ale zapewne w praktyce pojawią się tego typu problemy.

Jeśli robot znajdzie się całkowicie na polu czerwonym podjeżdża on do przodu aż sensor wykryje przed nim klocek - w takim przypadku zatrzymuje on się i podnosi go. Następnie wykonuje on obrót o 180° i jedzie prosto aż czujniki przestaną wykrywać jedynie kolor czerwony. Analogicznie jest w przypadku pola całkowicie niebieskiego, z tym robot podjeżdża prosto na określoną odległość i odstawia klocek w tym miejscu, bez użycia czujnika odległości.

Dodatkowo spróbowaliśmy dopisać również kod odpowiedzialny za powrót na trasę. Działa on w taki sposób, że przy kolorowych skrętach (czerwonych lub niebieskich) program zapamiętuje, w którą stronę robot skręcił i po opuszczeniu klocka o specyficznym kolorze i wykryciu skrzyżowania wykonuje on skręt w stronę, w którą miał jechać.

Napotkane problemy:

Największym problemem przy programowaniu transportera było zaprojektowanie skrętu na tor czerwony bądź niebieski – w szczególności sprawdzenie czy pojawia się odpowiedni kolor, dla którego mamy skręcić, tak aby kod dotyczęcy line follower'a nadal funkcjonował poprawnie. Wymagało to wypróbowania paru różnych pomysłów.

Od strony mechanicznej problemem okazał się również podnośnik, który początkowo został przez nas źle zaprojektowany - podnoszenie wymagało zadania zbyt dużej prędkości średniemu serwomotorowi. Konieczna okazała się przebudowa robota, aby podnośnik działał w pełni poprawnie.

Kod:

Kod dla transportera realizujący również zadanie podążania za linią:

```
#!/usr/bin/env python3
import ev3dev.ev3 as ev3
from time import sleep
from ev3dev2.sensor.lego import ColorSensor

ts = ev3.TouchSensor('in4')
ma = ev3.MediumMotor('outB')
ml = ev3.LargeMotor('outA')
mp = ev3.LargeMotor('outD')
cl = ev3.ColorSensor('in1')
cp = ev3.ColorSensor('in2')
cl.mode = 'COL-REFLECT'
cp.mode = 'COL-REFLECT'
inf = ev3.InfraredSensor('in3')
inf.mode = 'IR-PROX'
turnLeft=0 # 0 - no turns were made yet | 1 - the first turn was made | 2 -
ready to get back on lane
turnRight=0 # 0 - no turns were made yet | 1 - the first turn was made | 2 -
ready to get back on lane

def set_motor_speed(left_spd, right_spd, seconds=1):
    ml.run_timed(time_sp=seconds*1000, speed_sp=left_spd)
    mp.run_timed(time_sp=seconds*1000, speed_sp=right_spd)

def set_forward_speed(speed, seconds=1):
    set_motor_speed(speed, speed, seconds*1000)

def stop():
    set_forward_speed(0)
```

```
def change_to_rgb():
    cl.mode = 'RGB-RAW'
    cp.mode = 'RGB-RAW'
    print("CHANGED MODE TO RGB")
    sleep(1) #TIME TO CHANGE SENSOR MODE

def print_rgb_state():
    print("CL RGB:", str(cl.value(0)) + ", " + str(cl.value(1)) + ", "  +
str(cl.value(2)))
    print("CP RGB:", str(cp.value(0)) + ", " + str(cp.value(1)) + ", "  +
str(cp.value(2)))

def change_to_reflect():
    cl.mode = 'COL-REFLECT'
    cp.mode = 'COL-REFLECT'
    print("CHANGED MODE TO REFLECT")
    sleep(1/2) #TIME TO CHANGE SENSOR MODE

def print_reflect_state():
    print("CL REFLECT:", cl.value())
    print("CP REFLECT:", cp.value())

def move_forward(speed, seconds):
    set_forward_speed(speed, seconds)
    sleep(seconds)

def move(left_spd, right_spd, seconds):
    set_motor_speed(left_spd, right_spd, seconds)
    sleep(seconds)

def rotate(dir):
    if dir in ("left", 'LEFT', 'l', 'L'):
        move(100, -85, 2)
    elif dir in ("right", 'RIGHT', 'r', 'R'):
        move(-85, 100, 2)
    elif dir in ("back", 'BACK', '180'):
        move(-96, 96, 5)
    else: raise ValueError

def pick_up():
    ma.run_timed(time_sp=5000, speed_sp=-200)
    sleep(5)

def put_down():
    ma.run_timed(time_sp=5000, speed_sp=200)
    sleep(5)

clicked = False
```

```
print("READY")
while True:
    # left light flashes red when button is pressed
    ev3.Leds.set_color(ev3.Leds.LEFT, (ev3.Leds.GREEN,
ev3.Leds.RED)[ts.value()])
    # turn the robot on when the button is pressed
    if (ts.value() and not clicked):
        print("START")
        clicked = True
        sleep(2) # so that a press only registers once
    if (ts.value() and clicked): # turn off
        print("STOP")
        stop()
        clicked = False
        sleep(2) # so that a press only registers once
    if clicked:
        change_to_reflect() # make sure the mode is set to reflect

        err = cl.value() - cp.value() # calculate err - the diff between
values shown by left and right sensor(in reflect mode)

        # LINE FOLLOWER
        # adjust direction based on the calculated value of err
        if -10 <= err <= 10: #straight
            set_motor_speed(-100, -100)
        elif 10 < err <= 20: # light left
            set_motor_speed(-120, -30)
        elif 20 < err <= 50: # hard left
            set_motor_speed(-150, -5)
        elif -20 <= err < -10: # light right
            set_motor_speed(-30, -120)
        elif -50 <= err < -20: # hard right
            set_motor_speed(-5, -150)

        # ROTATE LEFT - hard turn left (and BLUE turn left)
        if err >= 50 or (cl.value() < 6 and cp.value() > 8):
            print("ENTERED ROTATE LEFT")
            stop()
            change_to_rgb() # change to rgb to check if there is it's a blue
turn
            print_rgb_state()
            if(cl.value(0) < 30 and 50 < cl.value(1) < 70 and cl.value(2) >
70):# blue to the left
                print("BLUE - LEFT")
                rotate('left')
                move_forward(-200, 0.33) # move forward a bit afterwards
                turnLeft=1
            else:# just a hard turn
                move(70, -70, 0.66) # slight rotation to get back on track
```

```
        # ROTATE RIGHT - hard turn right (and BLUE turn right)
        if err <= -50 or (cl.value() > 8 and cp.value() < 6):
            print("ENTERED ROTATE RIGHT")
            stop()
            change_to_rgb() # change to rgb to check if there is it's a blue
turn
            print_rgb_state()
            if(cp.value(0) < 30 and 70 < cp.value(1) and cp.value(2) > 70):#
blue to the right
                print("BLUE - RIGHT")
                rotate('right')
                move_forward(-200, 0.33) # move forward a bit afterwards
                turnRight=1
            else:# just a hard turn
                move(-70, 70, 0.66) # slight rotation to get back on track
            change_to_reflect()

        # FULL BLUE
        if cl.value() in (4, 5, 6) and cp.value() in (4, 5, 6) and
(turnLeft==1 or turnRight==1) :

            print("FULL BLUE")
            move_forward(-200, 2)

            # PUT DOWN THE OBJECT
            put_down()

            # ROTATE 180
            rotate('180')
            # MOVE FORWARD
            while cl.value() in (4, 5, 6) and cp.value() in (4, 5, 6):
                move_forward(-100, 0.1) # while in full blue move forward
            move_forward(-100, 0.3) # move forward a bit more to get fully on
track
            turnLeft=turnLeft+1
            turnRight=turnRight+1
            print("LEAVE FULL BLUE")

        #COLORS - ONLY BLUE(^) AND RED(v)

        #RED
        if ((28 <= cl.value() <= 32 and 28 <= cp.value() <= 32) or (25 <=
cl.value() <= 30 and 27 <= cp.value() <= 33) and not turnRight and not
turnLeft):#RIGHT OR LEFT
            print("ENTERED RED:")
            print_reflect_state()
            stop()
            change_to_rgb()
```

```
            print_rgb_state()
            if(cl.value(0) > 100 and cl.value(1) < 35 and cl.value(2) < 20):
                #TURN LEFT IN PLACE
                rotate('left')
                #MOVE FORWARD A BIT
                move_forward(-200, 0.33)
                turnLeft=1
            elif (cp.value(0) > 100 and cp.value(1) > 30 and cp.value(1) < 60
and cp.value(2) < 30):
                #TURN RIGHT IN PLACE
                rotate('right')
                #MOVE FORWARD A BIT
                move_forward(-200, 0.33)
                turnRight=1

# NEW PART - UNTESTED

            # ADJUST FACING DIRECTION - TRY TO TURN PARALLEL TO THE TRACK
            err_red = cl.value(0) - cp.value(0)# sensors operating in rgb mode
            while abs(err_red) >= 5:
                if err_red > 0:# rotate right
                    move(-100, 100, 0.1)
                elif err_red < 0:# rotate left
                    move(100, -100, 0.1)
                err_red = cl.value(0) - cp.value(0)
                move_forward(-100, 2) # move forward so as to leave the red
part of the track - sleep time NEEDS ADJUSTMENT
            change_to_reflect()

        # FULL RED
        if ((turnLeft == 1 or turnRight == 1) and cl.value() >= 30 and
cp.value() >= 30): #check if in full red
            stop()
            change_to_rgb()
            if (cl.value(0) > 100 and cl.value(1) < 35 and cl.value(2) < 20
and cp.value(0) > 100 and cp.value(1) > 30 and cp.value(1) < 60 and
cp.value(2) < 30):# CHECK IF IN FULL RED
                print("ENTERED FULL RED")
                while inf.value() > 1: # move forward till infrared detects an
object in front
                    move_forward(-100, 0.1)
                # the robot is still a bit from the object -> move a bit
further
                move_forward(-100, 0.2) # !!! needs further adjustment
                stop() # STOP
                pick_up() # LIFT THE OBJECT
                # ROTATE 180
                rotate('180')
                # MOVE BACK ON TRACK
```

```
                while (cl.value(0) > 100 and cl.value(1) < 35 and cl.value(2)
< 20 and cp.value(0) > 100 and cp.value(1) > 30 and cp.value(1) < 60 and
cp.value(2) < 30):# WHILE IF IN FULL RED
                    move_forward(-100, 0.1)
                move_forward(-100, 0.3) # move a bit further to fully get back
on track
                turnLeft=turnLeft+1
                turnRight=turnRight+1
                print("LEAVING FULL RED")

        # Get back on lane right
        if turnRight==2 and cl.value()<=16 and cp.value()<=16:
            rotate('right')
            move_forward(-200, 0.33)
            turnLeft=0
            turnRight=0

        # Get back on lane left
        if turnLeft==2 and cl.value()<=16 and cp.value()<=16:
            rotate('left')
            move_forward(-200, 0.33)
            turnLeft=0
            turnRight=0
# END OF NEW PART
```