12.04.2023

Bartosz Latosek

Hubert Soroka

WR - Laboratorium

1. Zadanie

Celem laboratorium było stworzenie robota zdolnego do poruszania się i wykonywania akcji w zadanej przestrzeni. Środowisko robota stanowiła trasa reprezentowana przez czarną linię, po której robot musi się poruszać. Trasa oprócz czarnej linii posiada skrzyżowania reprezentowane przez odnogę w jednym z czterech podstawowych kolorów. Jest to element środowiska wykorzystywany dopiero w drugiej części laboratorium, służący do zdefiniowania miejsc załadunku i rozładunku.

Do budowy robota zostały nam przydzielone części interaktywne – takie jak czujnik odległości, czujniki światła, motory i główny klocek sterujący robotem – ev3. Oprócz tego mieliśmy dostęp do statycznych klocków LEGO stanowiących resztę ciała robota.

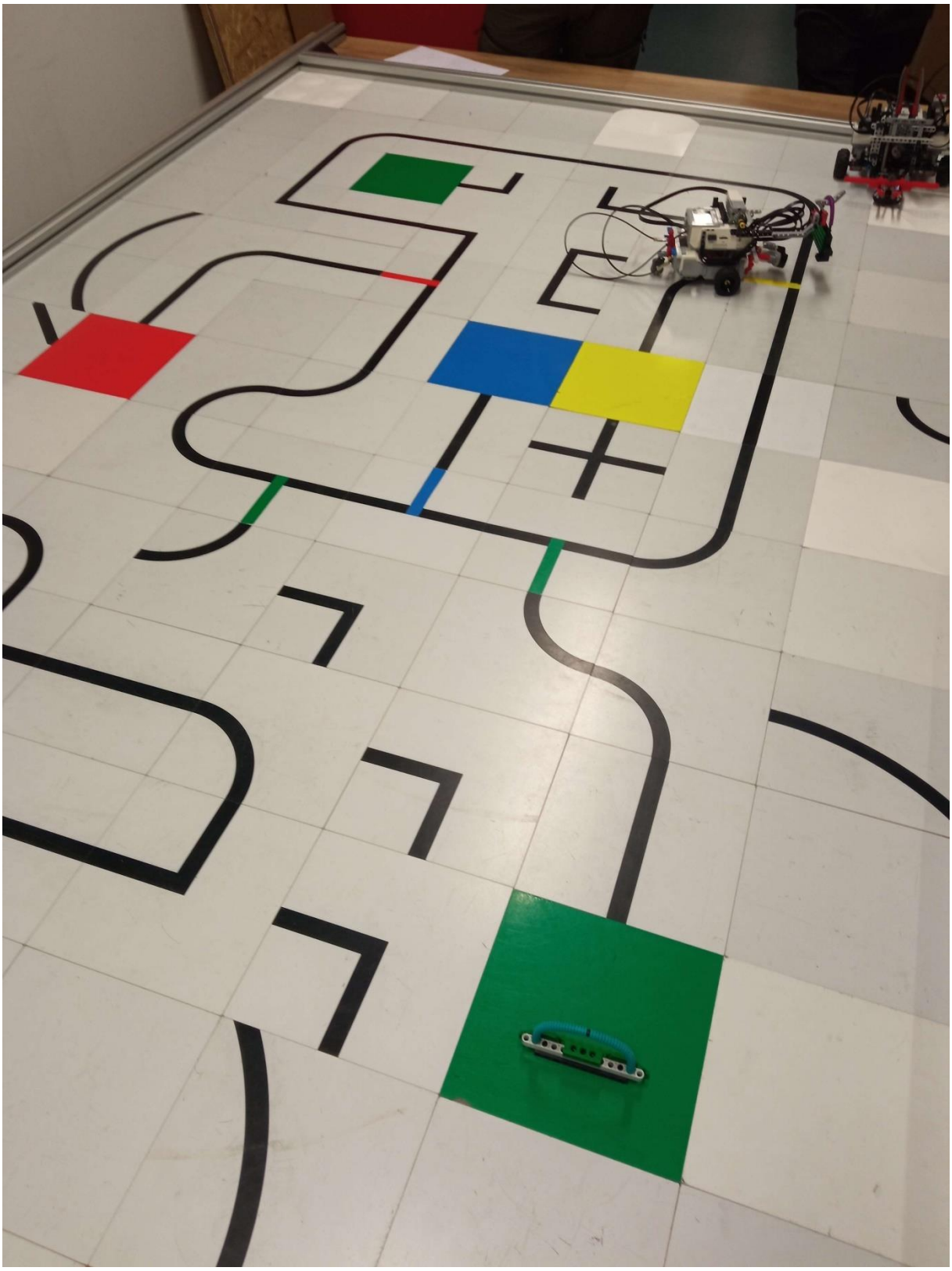
Naszym zadaniem było zbudowanie i zaprogramowanie robota ev3, który potrafi:

a. Podążać po czarnej linii na białym tle po krętym torze bez wypadania z niego. Zadanie to wymagało stworzenia interakcji czujników światła oraz motorów odpowiedzialnych za ruch robota. Duże wyzwanie stanowiło dobranie i kalibracja parametrów wejściowych takich jak prędkość jazdy prosto i prędkość skrętu robota, szerokość okna pamięci itd. Dodatkowo w tej fazie zadania robot powinien ignorować kolorowe skrzyżowania wykorzystywane w drugiej części zadania.
b. Przenieść ładunek z pola o podanym kolorze do pola oznaczonego innym kolorem, ignorując pozostałe kolory i podążając za linią jak w punkcie a) pomiędzy tymi obszarami. Strefom rozładunku i załadunku odpowiadały parametry reprezentujące postrzegane przez robota kolory.

Na laboratoriach 1-4 zajmowaliśmy się pierwszym zadaniem, a zajęciach 4-5 drugim.

Plansza, na której wykonaliśmy zadanie drugie:

Przyjęliśmy konfigurację początkową - strefa załadunku: pole zielone, strefa rozładunku: pole czerwone.



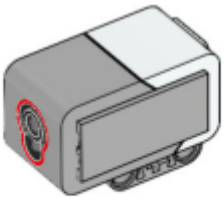
2. Użyte elementy

Robota skonstruowaliśmy z użyciem poniższych elementów z serii ev3:

a. Jednostka centralna

1x
EV3 Brick
6009996

b. 2 sensory optyczne

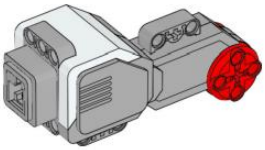


1x
Color Sensor
6008919

c. Sensor ultradźwiękowy

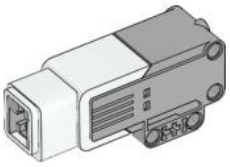
1x
Ultrasonic Sensor
6008924

d. 2 motory napędowe



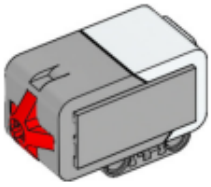
2x
Large Motor
6009430

e. Serwomotor



1x
Medium Motor
6008577

f. Przycisk



2x
Touch Sensor
6008472

Elementy połączyliśmy z użyciem klocków lego oraz przewodów łączących jednostkę centralną z pozostałymi komponentami. Dużym wyzwaniem okazało się być zbudowanie stabilnej ramy robota. Gdy w końcu po wielu próbach udało nam się uzyskać dostatecznie sztywną ramę, nastąpił zauważalny wzrost wydajności robota.

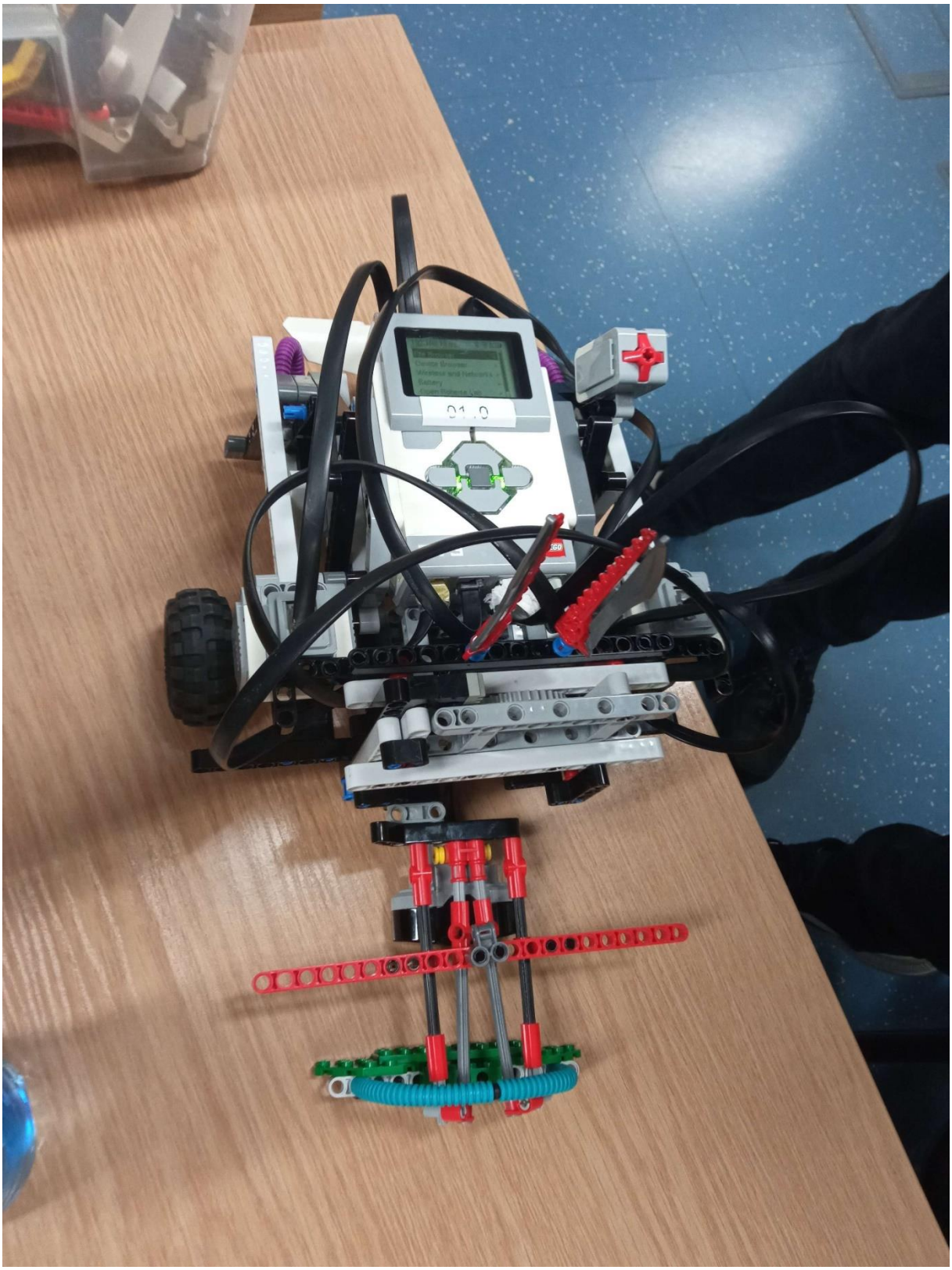
3. Konstrukcja robota

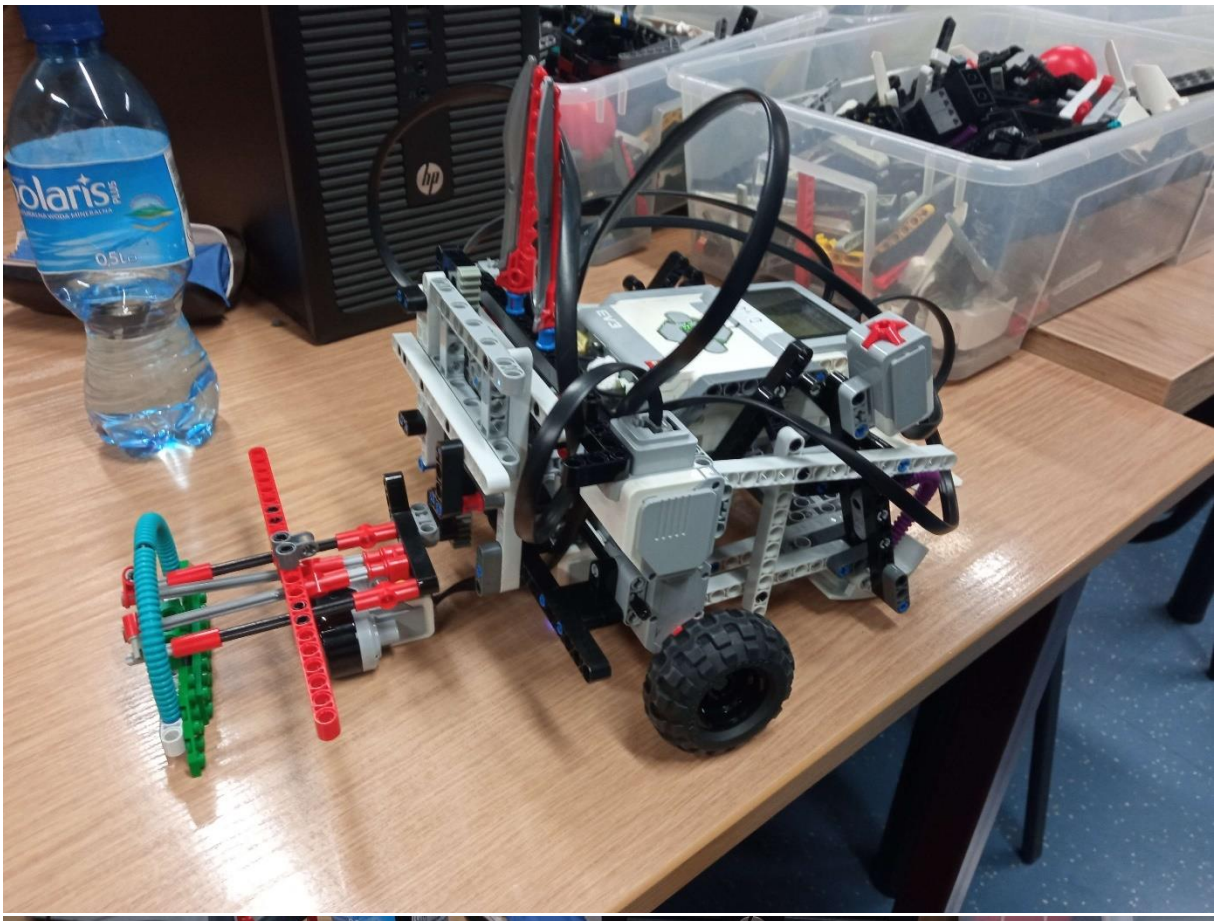
W miarę realizowania laboratoriów, wielokrotnie przebudowywaliśmy robota, częściowo ze względu na poprawienie osiągów a częściowo ze względu na zwiększone wymagania w drugim zadaniu.

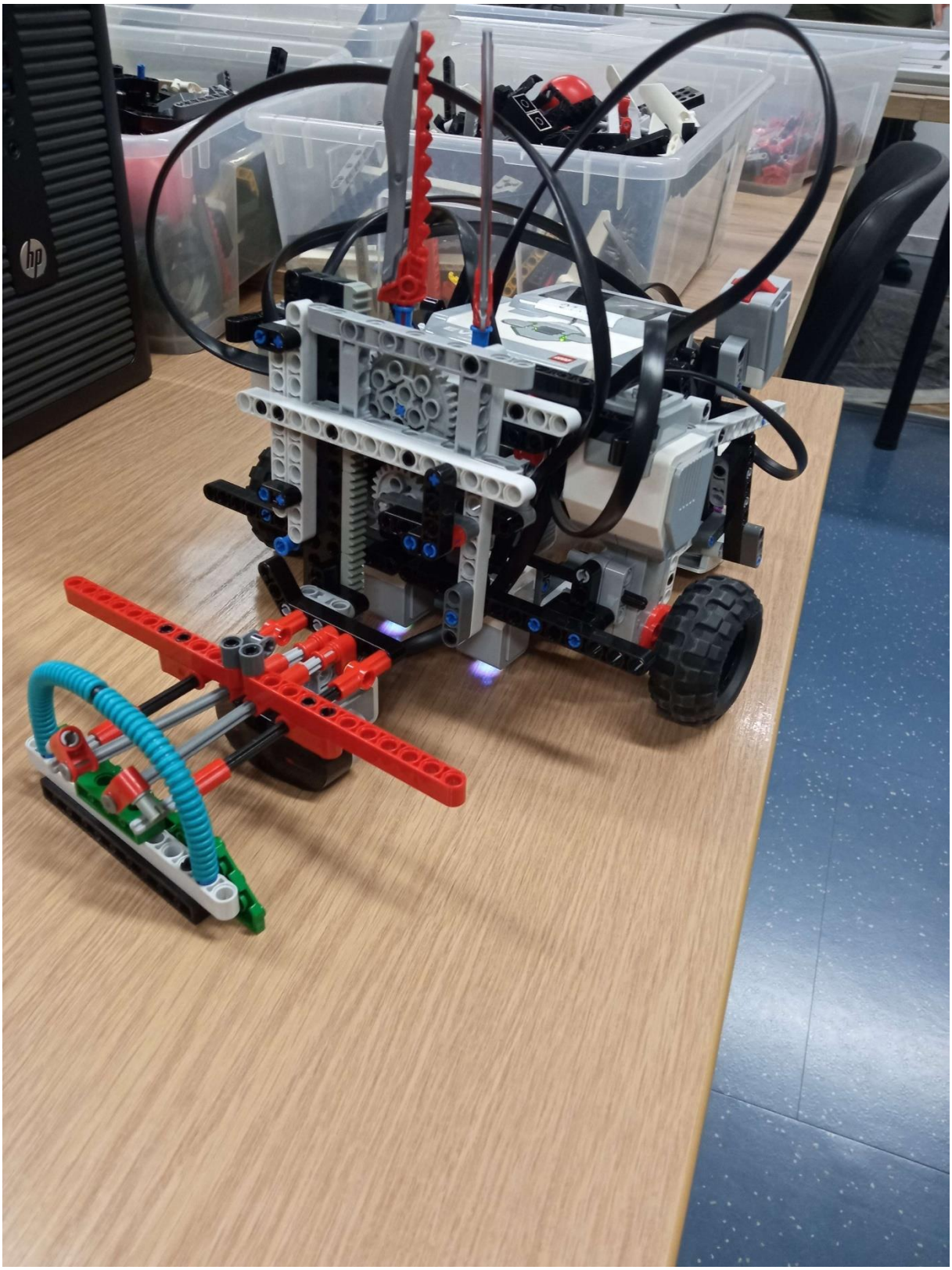
Postawiliśmy na jedną oś z szeroko rozstawionymi kołami, która zapewniła nam dobrą stabilność i skrętność. Bardzo dużo czasu poświęciliśmy na usztywnienie konstrukcji, aby zapobiec nadmiernemu przechylaniu się spowodowanemu przez duże rozmiary i masę robota. Aby zmniejszyć tarcie i zwiększyć skrętność, zrezygnowaliśmy z drugiej osi, zastępując ją niewielkimi płozami.

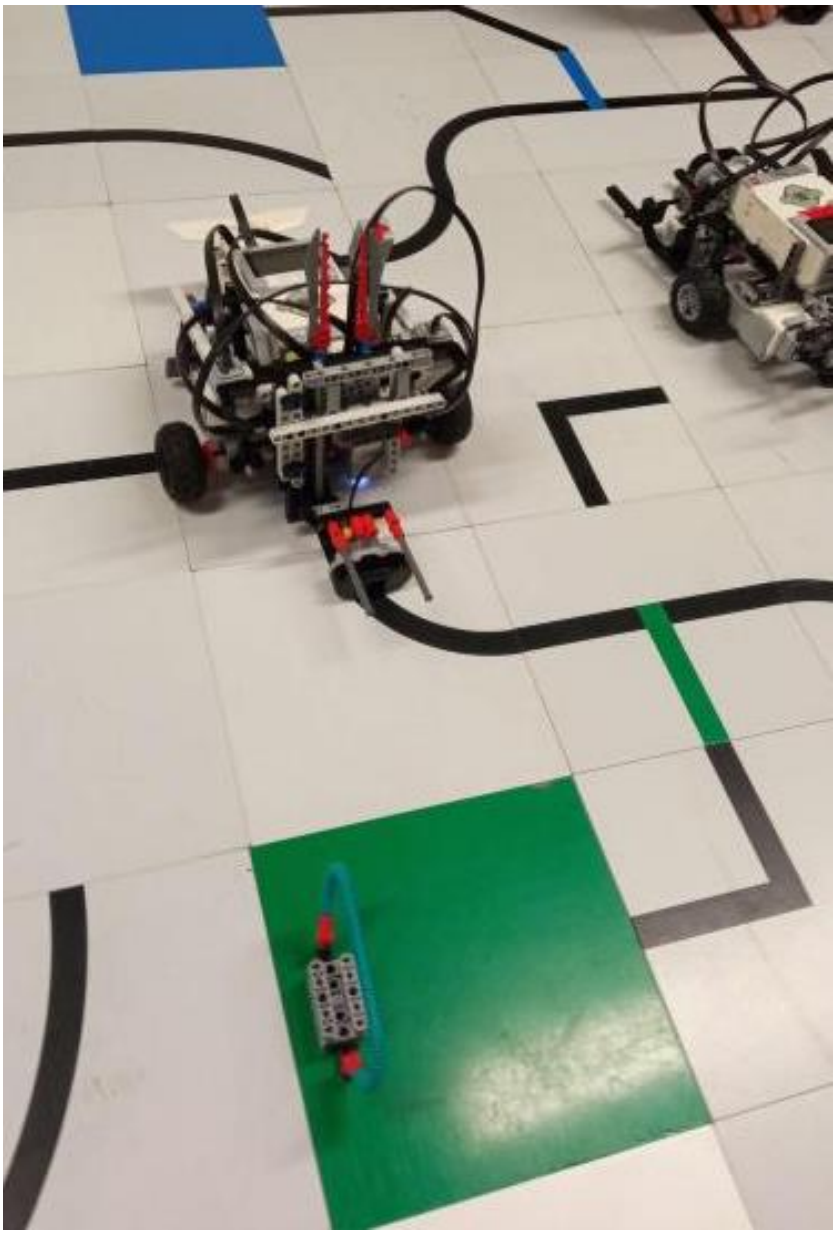
Podnoszenie ładunku zostało zrealizowane przez system podobny do podnośnika wózka widłowego. Serwomotor zamontowany został centralnie i bardzo nisko. Połączyliśmy go zębatkami z podnoszoną platformą z widłami z przodu pojazdu. Z przodu wideł zamontowaliśmy czujnik ultradźwiękowy, który podnosił się razem z przodem. Czujnik zamontowany tuż pod mechanizmem podnoszenia nie miał większych problemów z wykrywaniem ładunku.

Dodaliśmy też zabezpieczenia do wideł, w postaci małych elementów na końcu oraz dużej belki w połowie długości, co sprawiło, że ładunek nie spadał podczas transportu.




polaris
0.5L





4. Pomysł na algorytm

Gdy zapoznaliśmy się z interfejsami dostępnym elementów i zbudowaliśmy wczesny prototyp robota, stanęliśmy przed kwestią wymyślenia algorytmu jazdy. Chcieliśmy stworzyć jak najprostszy algorytm, ponieważ zależało nam bardziej na niezawodności i skupieniu na skonstruowaniu robota niż na zaawansowanym programowaniu w niewydajnym do takich celów języku, jakim jest python.

Istotą algorytmu jest pobranie odczytów z czujników oraz wybranie pożądanego kierunku ruchu na ich podstawie. Całość powtarzana w pętli aż do wyłączenia robota przyciskiem. Ruch w danym kierunku będzie realizowany przez włączenie silników z określoną mocą. W przypadku skręcania, oba silniki otrzymują różne parametry, co umożliwia precyzyjne parametryzowanie skręcania robota pomiędzy uruchomieniami programu.

Od początku chcieliśmy wydzielić stałe wartości z kodu jako parametry wejściowe, aby nie musieć przesyłać kodu z komputera po każdej drobnej kalibracji.

W miarę realizowania laboratorium, zastanawialiśmy się nad użyciem stosowanego przez inne zespoły regulatora PID lub podobnego, ale byłoby to bardziej zawodne, wymagałoby to bardzo dużo kalibracji i testowania i wcale nie musiałoby gwarantować szybszego działania na bardzo krętych torach, ponieważ z takimi nasz robot radził sobie bardzo dobrze.

Dodatkowo, bardzo łatwo było nam napisać algorytm do drugiego zadania, ponieważ wystarczyło wydzielić warunki stopu i skrętu do parametrów funkcji ciągłej jazdy, co umożliwiło podążanie za kolorami podanymi w argumentach wywołania, a nie ustalonymi z góry.

5. Algorytm i kalibracja

a. Podążanie za linią

Algorytm podążania za czarną linią jest bardzo prosty i opiera się na prostym trybie odczytu kolorów przez czujnik optyczny, tj. sensor zwraca liczbę od 0 do 7 odpowiadającą wykrytemu kolorowi. Najlepiej zobrazuje go sam kod wykonywanego na robocie programu:

```
def go():
    while not switch.is_pressed:
        left = left_sensor.color
        right = right_sensor.color

        if left == ColorSensor.COLOR_NOCOLOR or right == ColorSensor.COLOR_NOCOLOR:
            engines.off()

        elif left == ColorSensor.COLOR_WHITE and right != ColorSensor.COLOR_WHITE : # right
            engines.on(SpeedPercent(forward_turn_speed_percent), SpeedPercent(-backward_turn_speed_percent))

        elif left != ColorSensor.COLOR_WHITE  and right == ColorSensor.COLOR_WHITE : # left
            engines.on(SpeedPercent(-backward_turn_speed_percent), SpeedPercent(forward_turn_speed_percent))

        elif left != ColorSensor.COLOR_BLACK and right != ColorSensor.COLOR_BLACK: # straight
            engines.on(SpeedPercent(base_speed_percent), SpeedPercent(base_speed_percent))

        elif left == right: # same
            engines.on(SpeedPercent(base_speed_percent), SpeedPercent(base_speed_percent))

    engines.off()
```

Poruszamy się w pętli dopóki nie zostanie naciśnięty przycisk, ponieważ w pierwszym zadaniu nie ma zdefiniowanego końca działania robota. Jeśli któryś czujnik nie jest w stanie rozpoznać koloru, przerywamy działanie. Dzięki temu po podniesieniu robota, szybko przestaje działać. Następnie sprawdzamy, czy nie powinniśmy skręcić, czyli czy jeden czujnik nie napotkał koloru linii, podczas gdy drugi jest nad białą planszą. Jeśli nie ma powodu do skrętu, sprawdzamy, czy oba czujniki widzą to samo lub czy oba nie widzą linii. W takich przypadkach jedziemy naprzód.

Jazda naprzód przy takich samych odczytach na obu sensorach umożliwiała bezproblemowe radzenie sobie ze skrzyżowaniami.

Prędkości ustawiane przy jeździe naprzód lub skrętach są parametrami wywołania programu, co umożliwia zmianę działania robota bez wgrywania ponownie kodu.

Jeśli wartości forward_turn_percent_speed i backward_turn_percent_speed były jednakowe, robot skręcał w miejscu, co nie sprawdzało się najlepiej przy zakrętach w kształcie kąta prostego, dlatego z reguły forward_turn_percent_speed był trochę większy.

Parametr base_speed_percent ustawaliśmy w okolicach 15-25, ponieważ większe wartości sprawiały, że przy ostrych zakrętach robot czasami wypadał z toru.

W każdej pętli zapisujemy kolory z czujników tylko raz. Podczas prac nad robotem czytaliśmy te wartości zawsze, gdy były potrzebne, co sprawiło że kod był bardzo nieoptymalny i spowalniał reakcje robota. Sięganie po odczyty z czujników koloru tylko na początku pętli znacznie usprawniło działanie robota, szybciej reagował na zakręty.

Podobnie wcześniej przy sprawdzaniu warunków skrętu, porównywaliśmy jeden czujnik do koloru białego, a drugi do czarnego, co wydaje się intuicyjne w przypadku monochromatycznej planszy. Niestety, światło, którym sam czujnik oświetlał planszę, miało kolor niebieskozielony, co sprawiało, że gdy czujnik znajdował się na krawędzi kolorów białego i czarnego, odczytywał kolor niebieski, którym on sam świecił. Przejście na porównania z zaprzeczeniem (np. dla skrętu w lewo, sprawdzamy, czy lewy sensor odczytuje kolor inny niż biały (pokrywa to warunek odczytanego koloru niebieskiego przy krawędzi bieli i czerni) a prawy odczytuje kolor biały) rozwiązało problem dla planszy czarno-białej.

b. Przenoszenie ładunku

Zadanie drugie bazuje na podążaniu za linią po torze, jak w zadaniu pierwszym. Różnica polega na konieczności wykrywania kolorów innych niż czerń i biel podczas jazdy po linii. Sprawdzenia w głównej pętli programu zmieniliśmy na bardziej precyzyjne (tzn. robot podąża za linią o kolorze określoną przez stan algorytmu zadania). Z racji na wykrywanie koloru niebieskiego emitowanego przez sensory, gdy sensor był nad krawędzią czarnej linii, musieliśmy zmniejszyć prędkość jazdy robota. Powolniejszy robot radził sobie z ostrymi zakrętami, nawet jeśli sensory były mniej dokładne niż w pierwszym zadaniu.

Kolor A oznacza kolor z którego robot miał zabrać ładunek, kolor B to kolor, na którym robot powinien zostawić ładunek. Te wartości są parametrami wywołania programu, należy podać numer, któremu odpowiada kolor dla sensora optycznego ev3.

Kroki algorytmu transportera (każdy odpowiada jednemu stanowi algorytmu w kodzie):

i. Znajdź kolor A i obróć się o 90 stopni w jego stronę
ii. Podążaj linią w kolorze A lub czarną dopóki ładunek nie będzie w zasięgu
iii. Podnieś ładunek, obróć się o 180 stopni i zjedź z pola koloru A
iv. Podążaj linią koloru A lub czarną dopóki nie znajdziesz skrzyżowania i obróć się o 90 stopni, w tym samym kierunku co poprzednio
v. Znajdź kolor B i obróć się o 90 stopni w jego stronę
vi. Podążaj linią koloru B lub czarną dopóki oba czujniki nie wykryją koloru B
vii. Obniż ładunek

Skręty o 90 i 180 stopni są zrealizowane przez ustawienie mocy na silnikach i uśpienie programu na pewien czas. Dla 30% szybkości przy obrocie czas skrętu o 90 stopni wynosił około 2s. Dokładną wartość ustaliliśmy wielokrotnie testując obrót na małym wycinku toru. W takich sytuacjach bardzo opłaciło się przyjmowanie takich wartości jako parametrów wywołania programu, co bardzo przyspieszyło testowanie.

Podnoszenie i opuszczanie widłaka zrealizowaliśmy jako włączanie serwomotoru z określoną mocą z naprzemiennie dodatnim lub ujemnym znakiem na stałą liczbę obrotów. Doświadczalnie sprawdziliśmy, że 2 obrotu silnika wystarczają do podnoszenia i opuszczenia ładunku.

6. Wnioski

Zestaw LEGO mindstorm ev3 w połączeniu z wysokopoziomowym językiem programowania, do którego jesteśmy przyzwyczajeni stanowi bardzo potężne narzędzie. Interfejs jest intuicyjny, a całą trudność w

zbudowaniu sprawnej maszyny stanowi dobre zrozumienie pożądanego działania robota i jego fizyczna implementacja. Jest to zestaw dający naprawdę wiele możliwości.

Jeżeli chodzi o zadania realizowane w zakresie laboratorium – stworzenie robota podążającego po wyznaczonej ścieżce nie jest trudne. Stworzenie robota DOBRZE podążającego po ścieżce stanowi duże wyzwanie. Łatwo jest zbudować solidną bazę robota i przykładowo działający program, ale doprecyzowanie go stanowi wyzwanie i wymaga specjalistycznej wiedzy podpartej metodą prób i błędów. Laboratoria stanowiły świetne wprowadzenie w robotykę i dały szersze pojęcie o tej dziedzinie nauki.

7. Repozytorium

https://gitlab-stud.elka.pw.edu.pl/blatosek/WR_Lab