Sprawozdanie z projektu Line Follower i Transporter

Sprawozdanie z laboratoriów z przedmiotu „Wstęp do robotyki”

Sekcje omówione w sprawozdaniu:

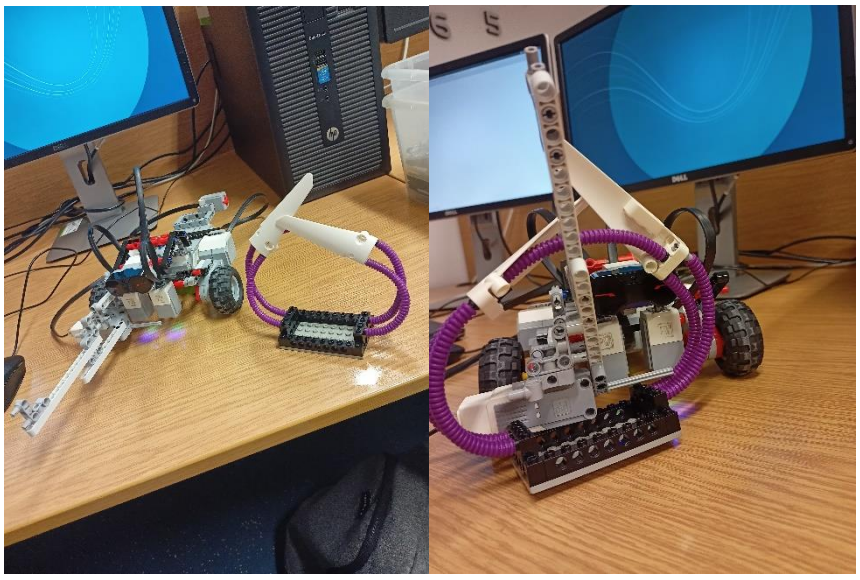
1. Budowa robota i jego wygląd
2. Klasyfikacja robota, czujników oraz silników
3. Pełnione funkcje przez robota
4. Sposób projektowania i oprogramowanie zastosowane w robocie
5. Klasyfikacja działania robota
6. Uruchomienie robota
7. Algorytmy robota oraz ich działanie
8. Wnioski końcowe

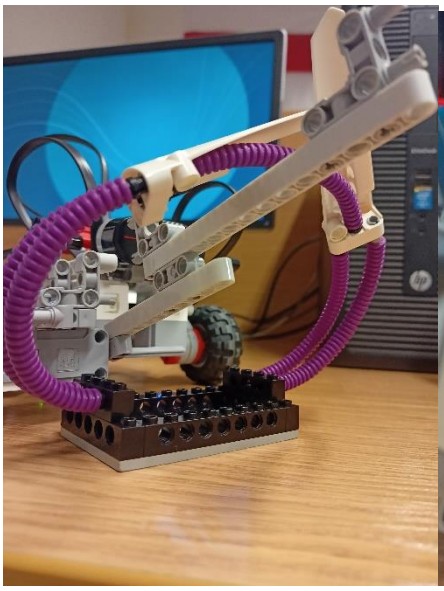
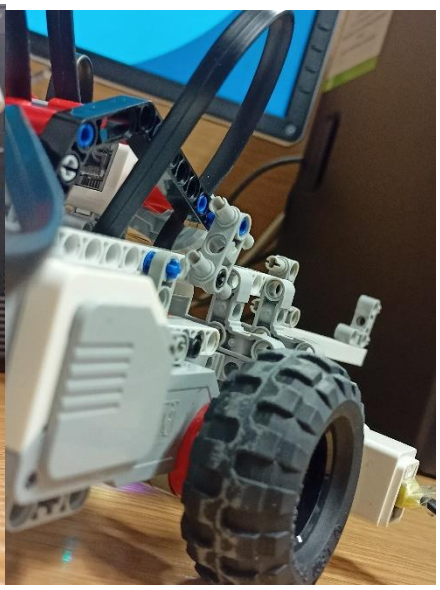
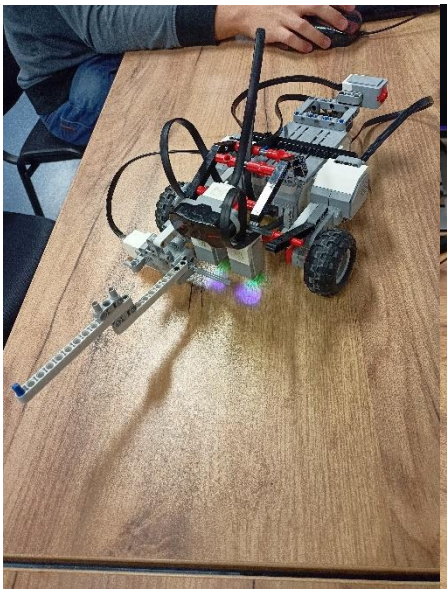
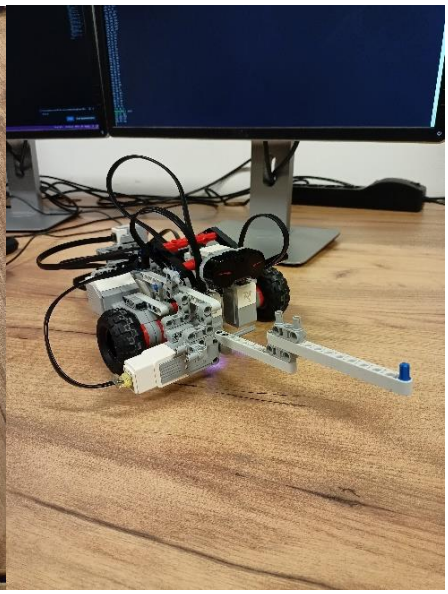
1.Budowa robota i jego wygląd:

Nazwa robota: *SKORPION*

Repozytorium: https://gitlab-stud.elka.pw.edu.pl/mchomans/lego-line-follower.git

Poniższe zdjęcia przedstawiają wygląd robota:



Zbudowany robot składa się z:

- sterownika EV3
- czujnika odległości (czujnika podczerwieni)
- dwóch czujników optycznych umożliwiających wykrywanie koloru za pomocą dwóch trybów działania (tryb wykrywania natężenia światła odbitego, tryb wykrywania RGB; inaczej czujnik kolorów)
- czujnika dotyku
- dwóch dużych silników ruchu postępowego (duże serwomechanizmy)
- silnika ruchu obrotowego (średni serwomechanizm)
- elementów łączących
- elementów pełniących funkcje mechaniczne (podnośnik)
- kabli łączących silniki oraz czujniki ze sterownikiem

2. Klasyfikacja robota, czujników oraz silników:

Zbudowanego robota możemy zaklasyfikować jako:
- robota przemysłowego (Pracuje on w uporządkowanym środowisku, w przestrzeni zamkniętej. Nie przeprowadza interakcji z człowiekiem oraz nie posiada rozbudowanej autonomii)
- robota półautomatycznego (Operator zadaje cel, a robot porusza się do celu i wykonuje go samodzielnie)
- robota mobilnego będącego pojazdem kołowym
- robota, który ma zastosowanie przemysłowe (mające imitować zastosowanie przemysłowe)
- robota nieholonomicznego

Środowisko działania robota: ląd (naziemne).
Mechanizm lokomocji: toczenie się.
Stopień autonomii: półautonomiczny.
Rodzaj autonomii: decyzyjna.
Rodzaj kół: napędzane stałe (zwykłe), bierne sferyczne.
Napęd: różnicowy.
Silniki: elektryczne prądu stałego DC.

Klasyfikacja czujników zawartych w robocie:
- bierne
- analogowe
- stanu zewnętrznego robota
- kontaktowe (czujnik dotyku) oraz bezkontaktowe
- dotykowe, odległości, środowiska

Czujniki i silniki zawarte w robocie:
- czujniki koloru - wykrywają dany kolor w trybie RGB zwracając trójelementową krotkę z wartościami 0 – 1020, jednakże po kalibracji (tak aby po włączeniu wykryły biały kolor) ich wartości mieszczą się w zakresie 0 – 255; czujnik ten posiada także tryb wykrywania intensywności światła odbitego, który zwraca wartości od 0 do 100, im wyższa tym więcej światła odbitego dociera do czujnika, a co za tym idzie wykrywane jest jaśniejszy kolor
- czujnik odległości - wykrywa jak daleko od czujnika jest umieszczony dany obiekt, zwraca wartości 0 – 100; 100, gdy obiekt jest poza zasięgiem czujnika; po przekroczeniu pewnego maksymalnego progu odległości wykrywania, im mniejsza wartość tym obiekt jest bliżej czujnika; 0 oznacza, że obiekt jest najbliżej czujnika jak to tylko możliwe
- czujnik dotyku – tutaj służył jako przycisk do włączania robota; można zaryzykować stwierdzenie, że działa jak dwustanowy przycisk
- dwa duże silniki operujące wartościami 0-1050 - służące do ustalenia prędkości robota i poruszania się; prędkość bazowa robota wynosiła 250, dla każdego silnika była również zmieniana przez regulator PID tak, aby umożliwić skręcanie (wtedy kierunek ruchu kół był wzajemnie przeciwny)
- jeden średni silnik - służył jako silnik umożliwiający obrót chwytaka, w naszym kodzie

przyjmuje dwa parametry: time – przez ile czasu się obraca (w ms), speed – z jaką szybkością (w stopniach na sekundę 0 - 1560) się obraca (jeśli prędkość jest dodatnia, obraca się do góry, jeśli ujemna, obraca się w dół)

3. Pełnione funkcje przez robota:

- line following - podążanie za czarną linią realizowane za pomocą regulatora PID
- transport ładunku z obszaru objętym danym kolorem do obszaru objętego innym kolorem

4. Sposób projektowania i oprogramowanie zastosowane w robocie:

Podejście do tworzenia struktury sterowania: góra-dół (klasyczne).
Struktura sterowania: reaktywna.
Metoda programowania: offline, tekstowo.
Oprogramowanie: EV3 software.
Język programowania: Python.
Środowisko: EV3dev dla dystrybucji Linux Debian.
IP kostki: 192.168.18.83.
Komunikacja: protokół SSH.
Algorytm sterowania dla line-follower: regulator PID połączony z sterowaniem na if-ach.
Pozyskiwanie informacji o obecności linii lub wystąpieniu danego koloru: tryb RGB czujników.

5. Klasyfikacja działania robota:

Źródła niepewności: percepcja, napędy, środowisko.
Sterowanie: reaktywne.
Lokomocja robota:
- **Stabilność**:
  - liczba kontaktów: 3
  - środek ciężkości: w sterowniku
  - stabilność:
  - nachylenie terenu: brak
- **Charakterystyka kontaktu**:
  - kontakt powierzchniowy dla kół przednich
  - kontakt punktowy dla kulki z tyłu robota
  - tarcie dla kół przednich
- **Rodzaj środowiska**:
  - struktura: gładka
  - ośrodek: powietrze

6. Uruchomienie robota

a) uruchomianie transportera

Aby uruchomić algorytm transportera trzeba przesłać dla robota kod, w którym wywołujemy metodę *pid_line_follower.* Po przesłaniu pliku należy uruchomić go z modułu głównego EV3. Potem trzeba postawić na spód jednej z wydrukowanych w drukarce 3D płytek (w celu kalibracji). Po usłyszeniu sygnału dźwiękowego *„NOW" należy* odczekać krótką chwilę (np. 2s) i postawić robota na trasie. Aby wprowadzić robota w ruch należy wcisnąć czujnik dotyku. Po odstawieniu pakunku robot powinien się zatrzymać i zakończyć działanie programu. Uwaga, trzeba pamiętać, aby chwytak był równoległy do podłoża przed rozpoczęciem pracy.

b) uruchomienie line followera w trybie light intensity

Aby uruchomić algorytm transportera trzeba przesłać dla robota kod, w którym wywołujemy metodę *pid_line_follower_li_intens.* Po przesłaniu pliku należy połączyć się z robotem przez SSH i wtedy uruchomić program. W tym trybie robot nie potrzebuje kalibracji więc można od razu postawić go na trasie i czekać aż program się skompiluje. Po starcie robot będzie krążył po torze do momentu przerwania programu przez użytkownika.

7. Algorytmy robota oraz ich działanie:

Stworzyliśmy dwa podobne algorytmy podążające za linią bazującą na różnych trybach pracy czujnika. Pierwszy bazuje na pracy w trybie zwracania intensywności świata odbitego i używamy go w line followerze. Drugi bazuje na pracy w trybie, w którym czujnik zwraca wartości RGB i używamy go w transporterze. Oba algorytmy zamknęliśmy w jednej klasie, której atrybuty to odpowiednie wartości do regulatora PID, domyślna prędkość robota, flagi związane z pracą transportera, oraz motory i sensory robota.

Klasa Line follower posiada kilka atrybutów prywatnych. Te atrybuty to:

- _left_motor – jest to atrybut odpowiadający lewemu motorowi
- _right_motor – jest to atrybut odpowiadający prawemu motorowi
- _lifter– jest to atrybut odpowiadający motorowi podnoszącemu pakunek
- _left_sensor– jest to atrybut odpowiadający prawemu sensorowi
- _right_sensor – jest to atrybut odpowiadający lewemu motorowi
- _button – jest to atrybut odpowiadający czujnikowi dotyku
- _distance_sensor– jest to atrybut odpowiadający czujnikowi odległości (czujnikowi IR)
- _base_speed – jest to atrybut przechowujący wartość podstawowej prędkości robota
- _turn_speed – jest to atrybut przechowujący wartość prędkości na zakrętach w trybie RGB
- _dt – różnica czasu
- _kp – wzmocnienie
- _Ti – czas całkowania
- _Td – czas różniczkowania

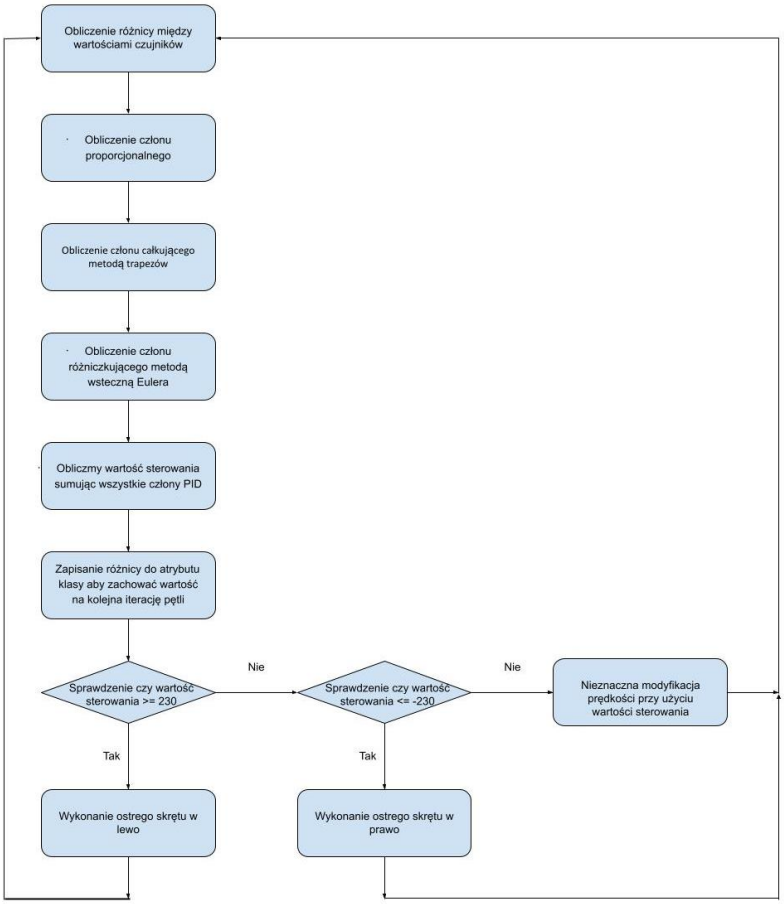
- _integral – atrybut ten przechowuje wartość operacji całkowania
- _prev_diff – atrybut ten przechowuje informacje o poprzedniej wartości różnicy

W trakcie programowania popełniliśmy błąd i przez przypadek sensor prawy nazwaliśmy lewym i na odwrót. Nie zauważyliśmy tego do momentu pisania sprawozdania i nie chcieliśmy niczego zmieniać w kodzie bez możliwości sprawdzenia tego. Nie wpłynęło to na działanie kodu, ale może spowodować małe problemy podczas jego analizowania. Nie zauważyliśmy tego błędu w czasie testów, ponieważ rzadko patrzyliśmy na nazwy w kodzie, a częściej obserwowaliśmy zwracane wartości.

Algorytm, który zastosowaliśmy do jazdy po linii to hybryda między algorytmem PID i algorytmem z użyciem If-ów.

Pierwszy algorytm:

Pierwszy algorytm znajduje się w metodzie „*pid_line_follower_li_intens*”. Główną częścią funkcji jest pętla *while* w której wykonują się wszystkie operacje. Operacje, które wykonują się w każdej iteracji pętli to:



- Obliczenie różnicy między wartościami czujników – w tym trybie wykorzystujemy atrybut *reflected_light_intensity* obu czujników do obliczenia różnicy.
- Obliczenie członu proporcjonalnego (ze wzmocnieniem K podanym w dalszej części).

- Obliczenie członu całkującego metodą trapezów.
- Obliczenie członu różniczkującego metodą wsteczną Eulera.
- Obliczenie wartości sterownia (*ster_val*) sumując wszystkie człony PID (człon całkujący i różniczkujący mierzymy odpowiednio przez czas całkowania i czas różniczkowania)
- Zapisanie wartości różnicy do odpowiedniego atrybutu klasy, który będzie mówił jaka była wartość różnicy w poprzedniej iteracji pętli
- Sprawdzenie czy robot musi wykonać ostry zakręt.
- Jeżeli tak to wykonujemy ostry zakręt w odpowiednią stronę.
- Jeżeli nie musi wykonać ostrego zakrętu, to zmieniamy prędkość w odpowiedni sposób przy pomocy wyliczonej wartości sterownia.

W obliczonej wartości sterownia polegamy na regulatorze PID. Następnie na podstawie tej wartości określamy, czy robot napotkał ostry zakręt i jeżeli tak wtedy wykonujemy ostry zakręt w odpowiednia stronę. Jeżeli wartość sterowania jest większa lub równa 70 to robot wykonuje ostry zakręt w lewo, a jeżeli wartość ta jest mniejsza lub równa -20. Niesymetryczność tego zakresu wynika z tego, że nie kalibrujemy sensorów i używamy bazowych wartości sensora. Zdecydowaliśmy się na taki krok, ponieważ zapewniało to pewną powtarzalność między próbami co pozwoliło na szybsze skalibrowanie ruchu robota i w tym trybie brak kalibracji czujników nie przeszkadzał. Sposób wyznaczenia tych parametrów zostanie wyjaśniony w dalszej części sprawozdania. Jeżeli robot wykryje ostry zakręt i wykona odpowiedni zakręt przejdzie do kolejnej iteracji pętli. Nie użyliśmy „czystego” PID-a, ponieważ dodanie *ifów* pozwoliło nam na mniej dokładne dopasowanie nastaw, co pozwoliło na szybsze wykonanie projektu. Spowodowało to natomiast że na zakrętach w kształcie ćwiartki koła nasz robot musi kilkukrotnie poprawić tor jazdy, aby przejechać dalej.

Prędkość obrotu silników na zakrętach to:

- Dla koła po stronie wewnętrznej - domyślna prędkość powiększona o moduł wartości sterownia i to wszystko przemnożone przez -1
- Dla koła po stronie zewnętrznej – dwukrotność domyślnej prędkości powiększona o moduł wartości sterownia

W kodzie moduły są zakodowane na stałe, ponieważ w danym bloku *if* możemy oczekiwać wartości sterownia zawsze z tym samym znakiem.

Dzięki kręceniu się koła wewnętrznego w przeciwną stronę do kierunku jazdy uzyskujemy szybsze wykonanie zakrętu. Stosunek tych wartości nie jest 1:1, ponieważ jeżeli byłby taki to w wypadku drgań robot nie poruszałby się do przodu, a oscylowałby jedynie wokół jednego punktu. Obie te wartości są co do modułu większe od wartości domyślnej, ponieważ pozwala to na szybsze pokonywanie zakrętów i w zakrętach problem niestabilności jest nieco mniej widoczny.

Prędkość na prostych to:

- Dla koła prawego: domyślna prędkość powiększona o wartość sterownia

- Dla koła lewego: domyślna prędkość pomniejszona o wartość sterownia

Takie wartości umożliwiają nieznaczne poprawy w trajektorii robota na prostych co pozwala uzyskać całkiem dużą stabilność w tych sekcjach toru.

Poszczególne wartości takie jak nastawy PID czy informacje na temat powyżej których wartości sterowania robot skręcał wyznaczyliśmy metodą prób i błędów.

Nastawy PID na początku wybraliśmy dowolne a następnie na podstawie obserwacji i testów z użyciem pojedynczych elementów toru lub stawiając robot na pełny tor zmienialiśmy je do momentu osiągnięcia stabilności na zadawalającym poziomie. W trakcie tych prób musieliśmy również zmieniać szerokości przedziałów, w których robot wykonywał ostre zakręty, aby odpowiednio odpowiedzieć na zmiany wartości sterowania.

Częstotliwość próbkowania sensora koloru to 1 kHz, wybraliśmy różnicę czasu *dt* na nieco większą, ponieważ pętla zdarzeń dodawała opóźnienia oraz na wypadek, jeżeli częstotliwość zmieniałaby się nieco pod wpływem różnych czynników. W naszym zastosowaniu nie potrzebowaliśmy, aby robot tak często sprawdzał kolor. Ta wartość nie mogła również być za duża, aby nie zniekształcać wartości. Ostatecznie ustawiliśmy ją na 0,01.

Ostateczne wartości:

- Wzmocnienie - *_kp = 9*
- Stała czasowa całkowania - *_Ti = 1,5*
- Stała czasowa różniczkowania - *_Td = 0,08*
- Przedział do skrętu w prawo - *_ster_val <= -20*
- Przedział do skrętu w lewo – *_ster_val >= 70*

Kiedy mieliśmy już ustalone nastawy PID zaczęliśmy dopracowywać wartości przy których robot wykonywał ostry zakręt. Braliśmy zakręt 90° i obserwowaliśmy, jak zmieniała się wartość sterowania podczas ręcznego zbliżania robota do zakrętu (używaliśmy funkcji *print* w każdej iteracji pętli). Na tej podstawie określaliśmy powyżej jakiej wartości (co do modułu) robot powinien wykonać zakręt w daną stronę i przechodziliśmy do analogicznych testów dla skrętu w drugą stronę. Następnie wartości, które wybraliśmy przez ręczne przeprowadzenie robota przez zakręt nieznacznie poprawialiśmy na podstawie obserwacji zachowania robota na pełnej planszy. Im wartości przy których robot wykonywał ostry zakręt były większe (co do modułu) tym oscylacje na prostych były mniejsze i rzadsze, ale jeżeli były zbyt duże to robot nie „zauważał" zakrętów. W trakcie testów określiliśmy, że wartości 70 i -20 były wystarczająco dobre.

Domyślna wartość prędkości w ostateczności to 250. W trakcie testów zwiększaliśmy stopniowo prędkość i w tym samy czasie poprawialiśmy nastawy algorytmu. Nie zwiększyliśmy prędkości powyżej tej, ponieważ ta prędkość wydawała się nam wystarczająca oraz po testowym zwiększeniu prędkości 300 stwierdziliśmy, że poprawienie nastaw wymagałoby znacznie więcej casu niż moglibyśmy na to poświęcić.

Ten algorytm służy wyłącznie jako algorytm do podążania za linią i nie ma zaimplementowanych funkcji związanych z transportowaniem pakunku. W trakcie zaliczenia

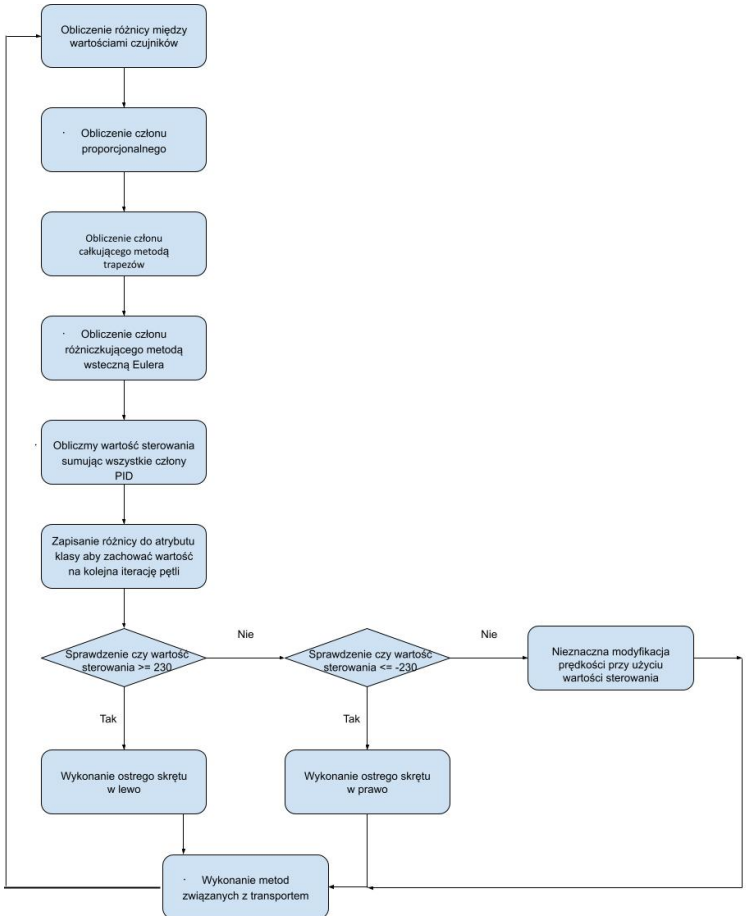
nie był użyty, ale zostawiliśmy go w kodzie i sprawozdaniu, aby pokazać ewolucję naszego algorytmu.

Drugi algorytm:

Drugi algorytm tak naprawdę w identyczny sposób podąża za linią, zmieniły się tylko niektóre wartości, ponieważ w tym przypadku korzystamy z trybu RGB oraz wykonujemy kalibrację.

Po uruchomieniu kodu należy postawić robota na odwrocie jednej z wydrukowanych na drukarce 3D płytek na planszy. Po pojawieniu się sygnału dźwiękowego trzeba odczekać krótką chwilę (1-2s wystarczy), a następnie postawić robota na linii i przycisnąć czujnik dotyku. W czasie tej krótkiej chwili wykonuje się kalibracja czujników koloru, która będzie przyjmować wartość 255 dla koloru białego, czyli zgodnie z konwencją RGB. W ten sposób dostajemy zakres wartości RGB odpowiadające naszemu środowisku i identyczny dla obu czujników. Czujniki kalibrujemy dzięki funkcji: *calibrate_white()* przed wejściem do głównej pętli programu. Ten zabieg pozwala na uzyskanie symetryczności przedziału, w którym robot używa regulatora PID, a poza tym przedziałem wykonuje ostre zakręty. Jak możemy zauważyć w line followerze wykorzystującym czujnik światła odbitego nie dokonywaliśmy kalibracji przez co ten przedział nie był symetryczny.

Po przyciśnięciu czujnika dotyku zaczyna wykonywać się główna pętla zadań. Kolejne operacje wykonywane w każdej iteracji pętli to:



Na schemacie przedstawiającym zachowanie robota w pętli nie przedstawiliśmy schematu blokowego dla poszczególnych funkcji transportera, aby uczynić schemat bardziej czytelny. Każda metoda będzie miała oddzielny schemat blokowy.

- Obliczenie różnicy między wartościami R czujników i przeskalowanie przez 100/255.
- Obliczenie członu proporcjonalnego (ze wzmocnieniem K podanym w dalszej części).
- Obliczenie członu całkującego metodą trapezów.
- Obliczenie członu różniczkującego metodą wsteczną Eulera.
- Obliczenie wartości sterownia (*ster_val*) sumując wszystkie człony PID (człon całkujący i różniczkujący mierzymy odpowiednio przez czas całkowania i czas różniczkowania)
- Zapisanie wartości różnicy do odpowiedniego atrybutu klasy, który będzie mówił jaka była wartość różnicy w poprzedniej iteracji.
- Sprawdzenie czy robot musi wykonać ostry zakręt
- Jeżeli nie musi wykonać ostrego zakrętu, to zmieniamy prędkość w odpowiedni sposób przy pomocy wyliczonej wartości sterownia.
- Wykonanie funkcji związanych z zabieraniem pakunku (wszystkie zostaną wyjaśnione w dalszej części).

Używamy jedynie wartości R (red) przeskalowanych przez 100/255, ponieważ przenieśliśmy kod z metody wykorzystującej częstotliwość światła. Podczas testów zauważyliśmy, że tryb częstotliwości światła wykorzystuje jedynie czerwona diodę i pomyśleliśmy, że wystarczy skopiować kod i odpowiednio przeskalować wartość R (red), aby można było używać tego samego regulatora. Niestety nie mogliśmy użyć tych samych nastaw, co mogło wynikać z faktu, że tryb RGB wprowadza większe opóźnienia od trybu z częstotliwością światła lub po prostu czujnik nie działa w taki sposób jak założyliśmy. Ostatecznie zostaliśmy przy tych założeniach, ponieważ dla kolorów czarnego i białego wszystkie wartości RGB powinny być identyczne więc to bez różnicy czy użyjemy wszystkich czy tylko jednej wartości. Parametry samego algorytmu podążającego za linią mają takie same nazwy i funkcje co w poprzednim algorytmie, ale mają inne wartości. Ostateczne wartości:

- Wzmocnienie - *_Kp = 9*
- Stała czasowa całkowania - *_Ti = 1,2*
- Stała czasowa różniczkowania - *_Td = 0,005*
- Przedział do skrętu w prawo - *_ster_val <= -230*
- Przedział do skrętu w lewo – *_ster_val >= 230*

Prędkość bazowa jest taka sama co w poprzednim algorytmie, ale dodaliśmy prędkość na zakrętach równą 175, ponieważ zauważyliśmy podczas testów, że prędkość na zakrętach jest za duża, a na prostych, za mała. Ta zmiana pomogła nam w stabilności. Musieliśmy dokonać jej prawdopodobnie z powodu zwiększonego opóźnienia czujnika koloru w trybie RGB.

W porównaniu do pierwszego algorytmu musieliśmy również zmienić, jak wyglądają prędkości na prostych i w zakrętach. Prędkość na prostych to:

- Dla koła prawego: domyślna prędkość powiększona o wartość sterownia podzieloną na 2
- Dla koła lewego: domyślna prędkość pomniejszona o wartość sterownia podzieloną na 2

Prędkość obrotu silników na zakrętach to:

- Dla koła po stronie wewnętrznej - domyślna prędkość powiększona o moduł wartości sterownia podzielony na 3,5 i to wszystko przemnożone przez -1
- Dla koła po stronie zewnętrznej – dwukrotność domyślnej prędkości powiększona o moduł wartości sterownia podzielony na 3,5

Wnioski do algorytmów podążania za linią

Zastosowanie hybrydy regulatora PID i sterowania przy pomocy if-ów pozwoliło nam osiągnąć algorytm, który dzięki regulatorowi PID poprawia swoją pozycję na linii w zależności od tego jak jest oddalony od środka i dzięki if-om szybkie pokonywanie zakrętów bez konieczności bardzo dokładnego dobru nastaw PID.

Metody związane z przenoszeniem pakunku:

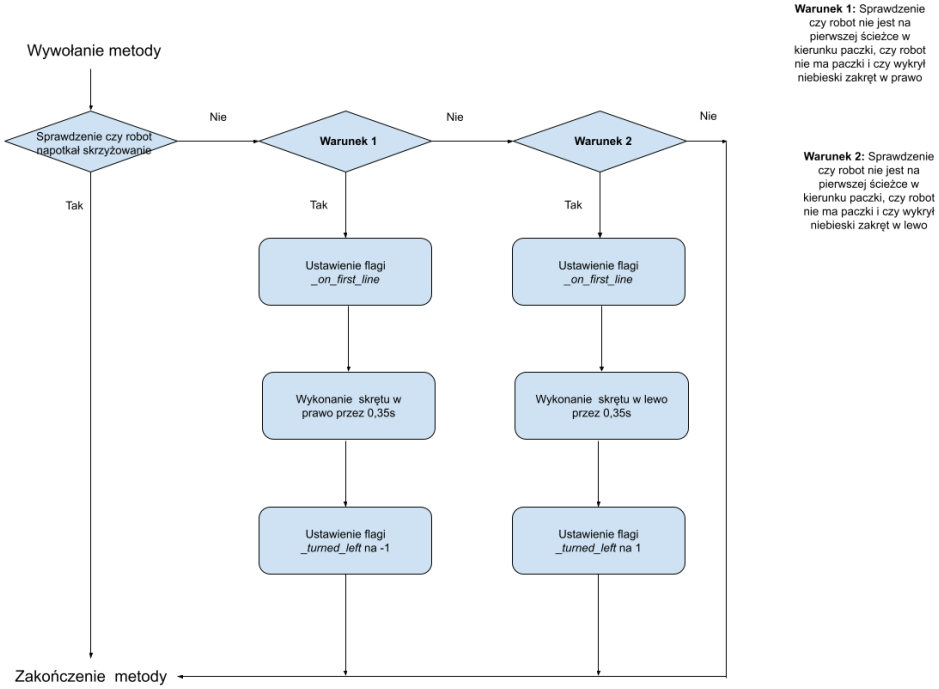
Aby te funkcje działały poprawnie dodaliśmy trzy flagi stanu jako atrybuty klasy w konstruktorze. Te flagi to:

- on_first_line – jest prawdziwa, gdy robot będzie na linii, na którą musi wjechać, aby zebrać paczkę
- on_second_line – jest prawdziwa, gdy robot będzie na linii, na którą musi wjechać, aby odstawić paczkę
- has_package – jest prawdziwa, gdy robot będzie miał paczkę
- _turned_left – jest 1, gdy pierwszy zakręt był w lewo i -1, gdy pierwszy zakręt był w prawo

Wykrywanie pierwszego zjazdu (metoda *sense_first_turn*)

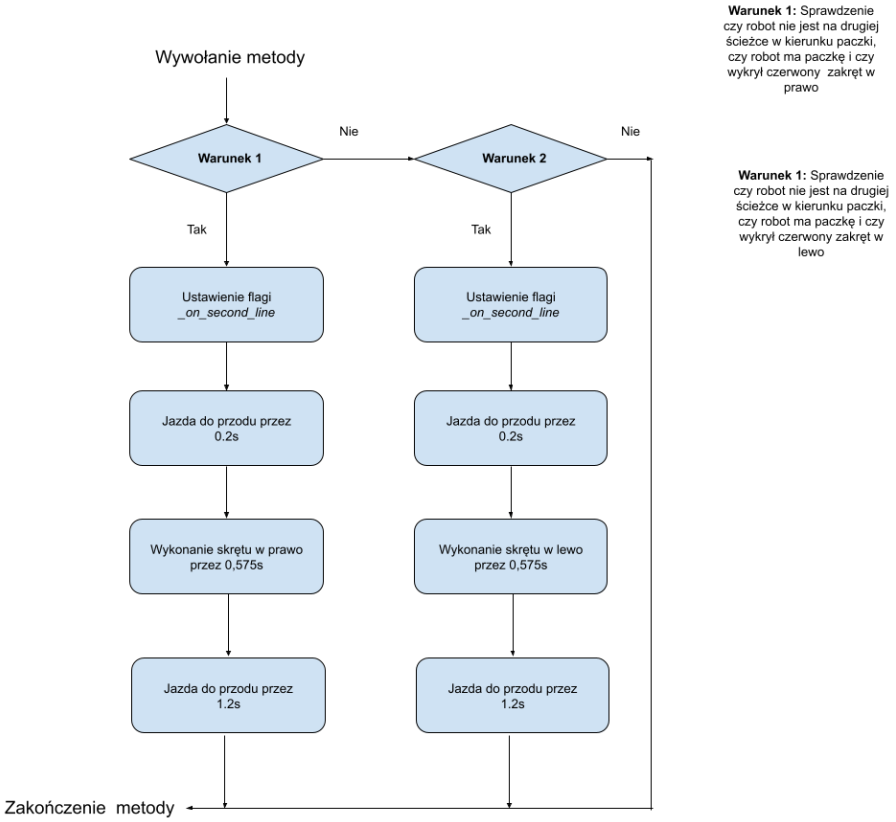
Na początku sprawdzamy czy nie pojawiły się skręty w obie strony i jeżeli tak to metoda jest kończona.

Aby wykryć i wykonać pierwszy zjazd sprawdzamy na początku czy robot nie jest na pierwszej linii (flaga *on_first_line*) oraz nie ma paczki (flaga *has_package*). Następnie sprawdzamy stosunek wartości RGB na jednym z sensorów. W przypadku linii niebieskiej sprawdzamy czy suma R i G (green) jest mniejsza od wartości B (blue) + 10 oraz wartość B jest większa od 110. Zależności te wyznaczyliśmy przez przykładanie niebieskiej płytki do sensora. Jeżeli warunki są spełnione to wykonujemy skręt w stronę sensora, który wykrył taki stosunek z zdwojoną prędkością domyślną dla koła zewnętrznego i zdwojoną prędkością domyślną pomnożoną przez -1 dla koła wewnętrznego, co zapewnia zakręt w miejscu. Zakręt wykonujemy przez 0.35s, wartość tą ustaliliśmy podczas testów patrząc czy robot w przybliżeni wykonał skręt o 90°. W przypadku zakrętów o konkretny kąt prawdopodobnie lepiej by było wykonać zakręt przy pomocy metody *on_for_degrees*, ale znaleźliśmy tą metodę za późno w projekcie i nie było sensu zmieniać całego kodu (ta uwaga jest do wszystkich metod odpowiedzialnych za przenoszenie paczki, ponieważ w nich też używamy motorów włączonych przez dany czas). Pierwszy skręt skalibrowaliśmy dla niebieskiego i przez to tuż po zakręcie linia niebieska przez algorytm podążający z linią jest interpretowana jako czarna (wartości R tych kolorów na planszach są zbliżone). Dzięki temu nie musimy wykonać skrętu o dokładnie 90° i nie musimy ustawiać czasu skrętu co do setnych sekundy. Przez ten fakt pierwszy zakręt w naszym przypadku musi być jednym z „ciemniejszych" kolorów np. niebiskim lub zielonym.



Wykrywanie drugiego zjazdu (metoda *sense_second_turn*)

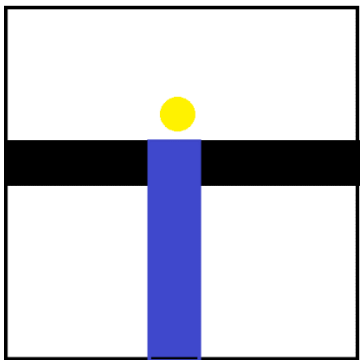
Algorytm ten działa analogicznie do algorytmu wykrywania pierwszego zjazdu, ale pojawiają się nieznaczne różnice. Część kodu odpowiedzialna za skręcanie jest wykonywana tylko wtedy, kiedy robot ma paczkę (jeżeli jest ustawiona flaga *has_package*) oraz robot nie jest na linii po drugim zjeździe (jeżeli nie jest ustawiona flaga *on_second_line)*. Drugi zjazd skalibrowaliśmy na wykrywanie czerwonej linii więc warunek wykonania to wartość R jest większa od sumy G i B pomniejszonej o 10. Dodatkowo przez to, że nasz program interpretuje linię czerwoną jako obszar biały musimy na początku podjechać do przodu przez 0,2s, potem wykonać bardziej dokładny skręt przez 0,575s i podjechać trochę do przodu przez 1,2s, aby znaleźć się ponownie na czarnej linii (za to jest odpowiedzialna druga pętla *while*).



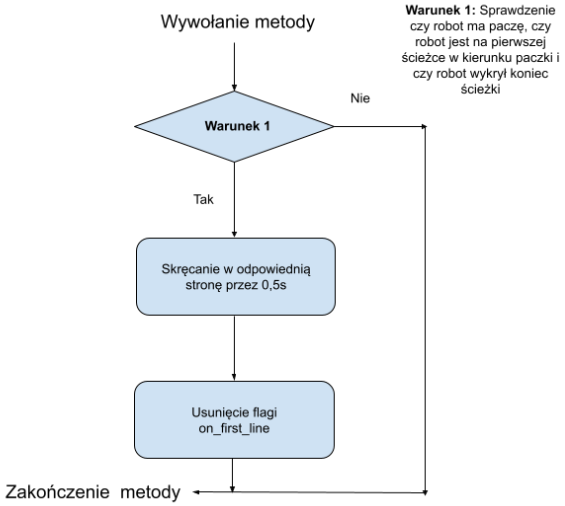
Różnica między czasem skrętu dla zakrętów wynika z faktu żenasz robot interpretuje niebeski jako bliski czarnemu więc w pierwszym przypadku nie potrzebuje aż tak dokładnego zakrętu bo sam się potem poprawi.

Powrót na główną linię (metod *return_to_line*)

Algorytm ten jest wykonywany wtedy i tylko wtedy, gdy robot jest na pierwszej linii i ma paczkę. Sposób działania jest prosty i polega tak naprawdę nie na wykryciu zakrętu powrotnego, ale na wykryciu faktu, że skończyła się ścieżka, z której robot miał zabrać paczkę. Najłatwiej to będzie zobrazować na prostym obrazku. Poniższy obrazek w uproszczeniu przedstawia płytkę z niebieskim zakrętem. Jeżeli czujniki koloru znajdą się w pozycji oznaczonej żółtą kropką, robot wykrywa, że niebieska linia się skończyła. W tym momencie oba czujniki pokazują wartość odpowiadającą kolorowi białemu.

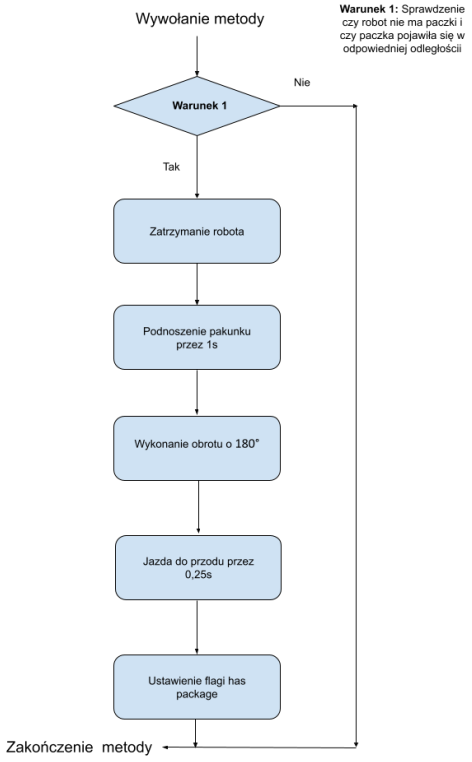


W warunkach *if* sprawdzamy czy suma wszystkich komponentów jest większa od 3*235 (mniejsze od 3*255, aby zapewnić jakiś margines błędu). Jeżeli pojawi się taka sytuacja robot automatycznie skręca w tą samą stronę, w którą skręcił wjeżdżając na niebieską linię. Skręcania w odpowiednią stronę odbywa się przez przemnożenie prędkości skręcania przez flagę _turned_left. W ten sposób otrzymujemy odpowiedni skręt bez użycia *if*-ów. Zakręt musi być wykonywany nieco dłużej, ponieważ koła robota w momencie wykrycia zakrętu są już za zakrętem więc aby powrócić na linię robot musi wykonać zakręt nieco większy od 90°. Zakręt wykonujemy z taką samą prędkością jak dla obu metod służących do wjazdu na linię.



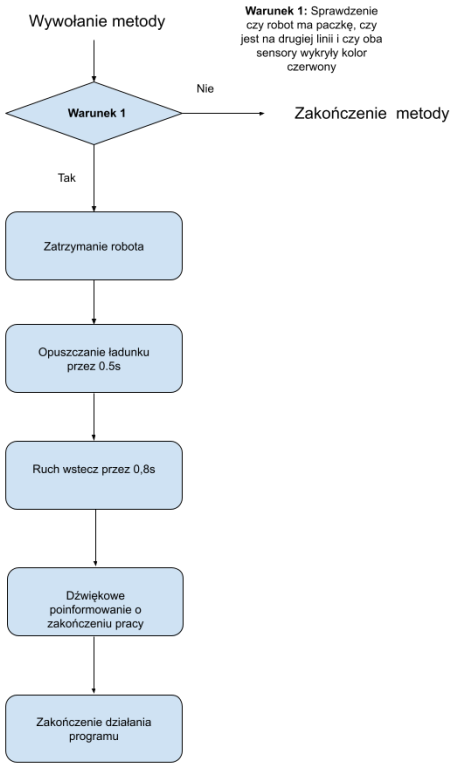
Podnoszenie pakunku (metoda *sense_package*)

Jeżeli robot nie ma pakunku (jeżeli flaga has_package nie jest ustawiona) sprawdzamy czy coś się znajduje w odległości 30% maksymalnego zasięgu czujnika. Wartość tą uzyskaliśmy przez zwracanie wartości *proximity* czujnika i zbliżanie pakunku do miejsca, w którym nasz wysięgnik będzie w stanie go podnieść. Po wykryciu pakunku w zasięgu podnośnika, przez 0,2 sekundy wykonujemy podnoszenie, następnie przez 1,2 s wykonujemy skręt, aby odwrócić się o 180° stopni, a potem jedziemy prosto do linii przez 1s. Na początku nasza czujnik miał problem z wykrywaniem rączek pakunku. Dlatego do pakunku dodaliśmy elementy które pomogły w jego wykrywaniu co pozwoliło na dokładniejsze określenie odległości, w której musimy podnieść pakunek oraz wyeliminowaliśmy przypadki, gdy nasz czujnik wykrywał coś co znajdowało się poza planszą.



Opuszczanie pakunku (metoda *drop_package*)

Metoda wykonywana wtedy i tylko wtedy, gdy robot jest na drugim zjeździe (jeżeli jest ustawiona flaga *on_second_line*) i jeżeli robot ma pakunek (jeżeli jest ustawiona flaga *has_package*). Wtedy robot sprawdza czy na jednym z sensorów pojawiła się wartość koloru taka sama jak dla zakrętu, czyli w naszym wypadku wartość R jest większa od sumy G i B pomniejszonej o 10. Wtedy robot zatrzymuje się opuszcza pakunek przez 0,5s, jedzie do tyłu przez 0,8s i kończy wykonywanie programu (komenda *exit()*). Pierwotnie założyliśmy, że po wykonaniu zadania robot się zatrzymuje, ale można go tak zaprogramować, aby powracał do pracy, wystarczy odpowiednio zaadaptować mechanizmy użyte w innych metodach. Opuszczamy krócej niż podnosimy, ponieważ nie musimy opuścić chwytaka do pozycji równoległej do podłoża, wystarczy, że będzie nieco poniżej rączki pakunku.



Uwagi do metod:

W metodach użyliśmy funkcji motorów *run_timed,* zrobiliśmy to, ponieważ myśleliśmy, że mogą one zastąpić pętle, których w ostateczności użyliśmy. Zostawiliśmy je w kodzie, ponieważ w ostateczności nic nie psuły i nie chcieliśmy marnować czasu na coś co by nie poprawiło działania robota.

8. Wnioski końcowe:

Podczas tych laboratoriów:

- nauczyliśmy się implementować regulator PID na praktycznym przykładzie,
- poznaliśmy zasady działania wykorzystywanych czujników,
- poznaliśmy wpływ środowiska zewnętrznego na zachowanie robota,
- nauczyliśmy się analizować i rozwiązywać problemy na praktycznym przykładzie
- nauczyliśmy się implementować proste programy dla robotów o charakterze przemysłowym

Wnioski ogólne:

Wiele rzeczy mogliśmy zrobić inaczej, np.: użycie metod *run_for_degree* tam, gdzie było trzeba, ale niestety skończył się nam czas. Nauczyliśmy się też, że z obiektem rzeczywistym jest trudniejsza i bardziej nieprzewidywalna niż praca z programem w komputerze. Dobry przekładem tego są kable łączące czujniki i moduł główny przez które pewnie straciliśmy kilka godzin, ponieważ ciągle się rozłączały. Dzięki temu projektowi nauczyliśmy się, że nie zawsze należy ufać w 100% parametrom podanym przez producenta, ponieważ tak jak w przypadku czujników koloru mogą one się nieco różnić od rzeczywistości (chodzi nam o fakt, że prawdopodobnie czujniki miały inne bazowe wartości białego). Na pewno jednym z najważniejszych wniosków jaki możemy wyciągnąć z tego projektu jest fakt, że bardzo ważne jest testowanie zaimplementowanych funkcjonalności.