

POLITECHNIKA WROCŁAWSKA
WYDZIAŁ ELEKTRONIKI

KIERUNEK: Automatyka i Robotyka (AIR)
SPECJALNOŚĆ: Technologie informacyjne w systemach automatyki (ART)

**PRACA DYPLOMOWA
INŻYNIERSKA**

Robot mobilny rozpoznający znaki drogowe -
rozpoznawanie znaków metodami uczenia
maszynowego

Mobile robot following road signs - sign
recognition with machine learning methods

AUTOR:
Mateusz Król

PROWADZĄCY PRACĘ:
Dr inż. Piotr Ciskowski

OCENA PRACY:

Serdeczne podziękowania kieruję do mojego promotora, którego pomoc i wsparcie było kluczowym elementem podczas powstawania mojej pracy.

Spis treści

1 Wstęp	3
1.1 Cel i założenia projektu	4
2 Projekt robota	5
2.1 Komponenty	5
2.1.1 Podwozie	6
2.1.2 Czujniki	6
2.2 Sterowanie robotem	14
2.2.1 Wykrywanie linii	14
2.2.2 Unikanie przeszkód	16
2.2.3 Awaryjny STOP	18
2.3 Testy silników oraz czujników	18
3 Algorytmy rozpoznawania	21
3.1 HOG	21
3.2 SVM	24
4 Aplikacja	27
4.1 Narzędzia	27
4.1.1 Raspbian	27
4.1.2 OpenCV	28
4.1.3 Numpy	28
4.1.4 Matplotlib	28
4.2 Wymagania aplikacji	29
4.3 Budowa aplikacji	29
4.3.1 Baza zdjęć znaków drogowych i ich preprocessing	30
4.3.2 Uczenie maszyn wektorów nośnych	32
4.3.3 Testy maszyn wektorów nośnych	33
4.4 Połączenie aplikacji z robotem	34
4.5 Obsługa aplikacji	35
5 Wnioski	37
A Pełne podłączenie komponentów robota	39
B Adaptacje robota do znaków	41
C Raspberry Pi 3 Model B	43
Bibliografia	43

Rozdział 1

Wstęp

W dzisiejszych czasach można zaobserwować wzrost zainteresowania autonomicznymi pojazdami. Niezbędnymi elementami każdego takiego pojazdu od strony programistycznej są: umiejętność adaptacji do warunków panujących na drodze oraz przestrzeganie przepisów drogowych. W dalszym ciągu najczęstszym sposobem informowania kierowcy o zagrożeniach panujących na drodze są znaki drogowe. Kluczowe więc, aby autonomiczne pojazdy były w stanie dostosować swoje zachowanie do przekazu zawartego w tychże znakach.

W niniejszej pracy podjęto się zbudowania robota mobilnego, który ma za zadanie symulować auto autonomiczne. Robot ma zaimplementowany system poruszania się po torze jazdy jednocześnie dostosowując się do modeli znaków drogowych występujących na jego drodze.

Robot do poruszania się używa pary sensorów umożliwiających mu wykrycie czarnej linii, wzdłuż której ma się poruszać. Posiada dodatkowo parę sensorów zbliżeniowych, które mają za zadanie uchronić robota od uderzenia w przeszkodę. Dodatkowy czujnik z tyłu pojazdu ma dać robotowi możliwość ucieczki przed nagłym uderzeniem z tyłu. Dodatkowym zabezpieczeniem przed "ucieczką poza tor" robota jest czujka dźwięku. Jeżeli wykryje ona klaśniecie dłońmi robot natychmiast się zatrzymuje i czeka na komendę startu.

Robot sterowany jest za pomocą płytki komputerowej Raspberry Pi 3 o 4 rdzeniowym procesorze z taktowaniem 1.2 GHz oraz 1 GB RAM. Na płytce został wgrany system operacyjny Linux Raspbian, który jest oparty na Debianie. Elementem umożliwiającym wizję robota jest kamera internetowa podłączona do płytki za pomocą portu USB. Robot zasilany jest z 2 źródeł; zasilanie silników oraz zasilanie płytki Raspberry.

Do predykcji znaków użyto metod uczenia maszynowego z wykorzystaniem Maszyny Wektorów Nośnych. W efekcie powstała aplikacja, która na wejście sczytuje obraz z kamery robota a następnie poddaje go obróbce i klasyfikacji. W końcowym etapie aplikacja adoptuje zachowanie robota do znaku drogowego. Opis reakcji robota na poszczególne znaki zawarto w dodatku B.

W drugim rozdziale opisano budowę robota. Zawarto w nim opisy poszczególnych czujników oraz ich połączeń z platformą oraz w jaki sposób przetestować mobilność robota. W 3 rozdziale pojawia się opis algorytmów umożliwiających przetwarzanie i rozpoznanie znaków drogowych. W 4 rozdziale zaprezentowano finalny opis aplikacji wraz z jej wymaganiami oraz obsługą.

1.1 Cel i założenia projektu

- Budowa robota mobilnego poruszającego się wzdłuż lini czarnej albo białej.
- Zaopatrzenie robota w czujniki pozwalające unikać kolizji.
- Zaimplementowanie systemu awaryjnego hamowania robota.
- Napisanie skryptu sterującego silnikami i czujnikami robota.
- Zebranie odpowiedniej bazy zdjęć oraz ich odpowiednia obróbka w celu nauki algorytmu rozpoznawania i klasyfikacji znaków drogowych.
- Implementacja algorytmu służącego do rozpoznawania znaków przy pomocy maszyny wektorów nośnych.
- Napisanie skryptu adoptującego zachowanie robota do napotkanego znaku drogowego.

Rozdział 2

Projekt robota

Robot został zaprojektowany do poruszania się po płaskich powierzchniach. Do poruszania się robot wykorzystuje dwa koła z silnikami DC oraz jedno koło podporowe. Do sterowania silnikami robota wykorzystano moduł TB6612FNG, który pozwala na oddzielne sterowanie 2 silnikami wraz z możliwością poruszania ich w 2 różnych kierunkach. Do budowy podwozia wykorzystano zakupiony zestaw montażowy platformy robota mobilnego. Zasilanie robota podzielone jest na 2 niezależne źródła:

- zasilanie silników - bateria AA 4x1.5V,
 - zasilanie płytki komputerowej wraz z czujnikami - powerbank z wyjściem 5V DC
- Całość wraz z zasilaniem oraz pozostałymi komponentami ma masę około 1.5kg. Budowa robota rozpoczęła się już w czasie Projektu Zespołowego. W efekcie przed realizacją projektu inżynierskiego robot był już skrócony, ale nie był wyposażony w żadne sensory ani programy sterujące. Z części programistycznej podczas Projektu Zespołowego badano różne algorytmy przetwarzania obrazu (SURF, HOG, HSV) oraz różne algorytmy klasyfikacji obrazów (sieci neuronowe, Haar Cascades, SVM). Dzięki wcześniejszym badaniom możliwy był wybór algorytmów najlepszych z dotychczas testowanych.

2.1 Komponenty

W niniejszym podrozdziale zostaną jedynie wymienione poszczególne komponenty robota z podziałem na podwozie oraz nadwozie. Wszystkie elementy zostaną omówione /;w 2 kolejnych podrozdziałach. Przykładowy schemat połączeń został zawarty w Dodatku A.

Podwozie:

- rama z plexi x1szt
- silnik DC z przekładnią x2szt
- plastikowe koło z gumową oponą x2szt
- tarcza enkodera x2szt
- koło podporowe x1szt
- pojemnik na baterie x1szt

Nadwozie:

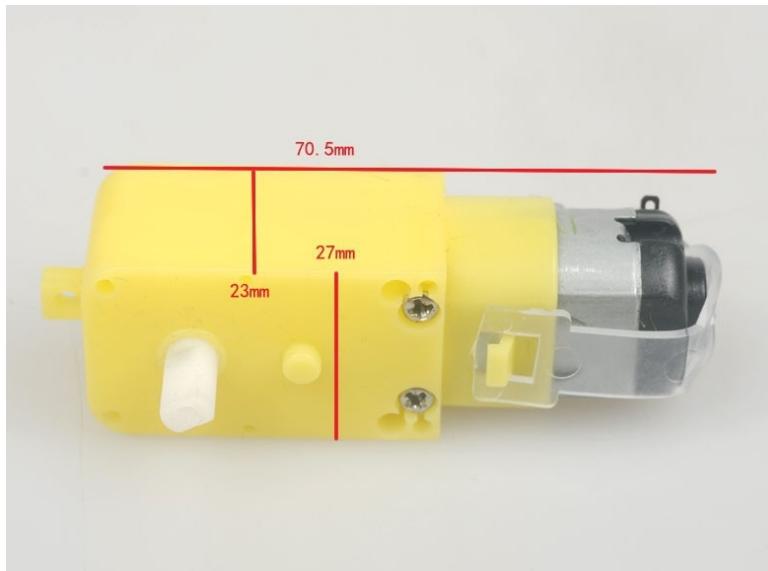
- płytki komputerowa Raspberry Pi 3 model B x1szt
- kamera internetowa LifeCam HD-5000 x1szt
- powerbank PLATINET (15000 mAh) x1szt
- ultradźwiękowy czujnik odległości HC-SR04 x2szt
- czujnik przerwania wiązki IR x2szt
- czujnik dźwięku (cyfrowy) x1szt

-sterownik silników DC TB6612FNG

2.1.1 Podwozie

Całość konstrukcji robota opiera się na ramie z plexi. Zdecydowano się na zakup gotowego elementu, ponieważ producent zadbał o odpowiednie nawiercenia ułatwiające montaż silników oraz kół. Dodatkowo w ramie znajduje się wiele otworów, co pozwala na swobodne dokładanie aparatury pomiarowej.

Częściami napędowymi robota są 2 silniki prądu stałego 6V z przekładnią 1:48. Dane techniczne zawarto w tabeli 2.1



Rysunek 2.1 Silnik prądu stałego 6V w obudowie

Tabela 2.1 Zbiór najważniejszych parametrów silnika

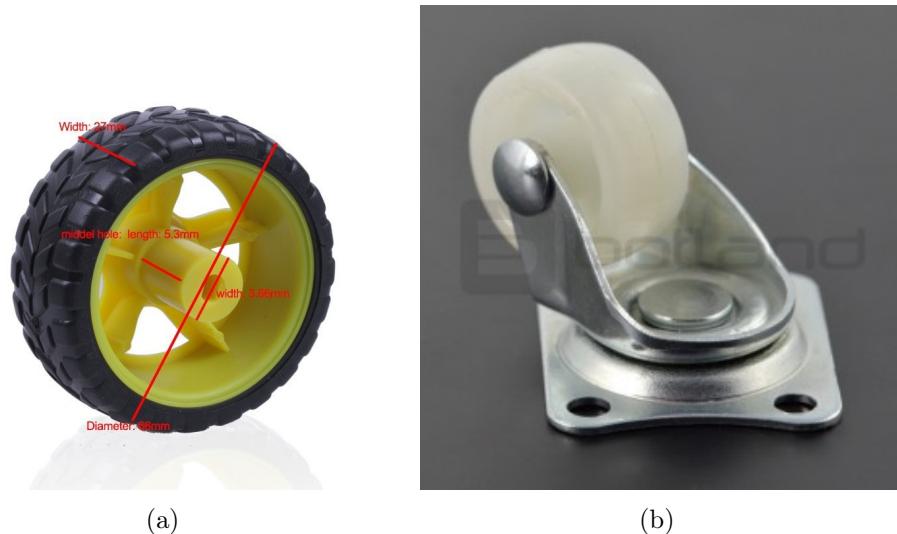
Znamionowe napięcie zasilania	6V
Prędkość obrotowa osi motoreduktora	100RPM
Pobór prądu przy zasilaniu 6V	do 350mA
Pobór prądu bez obciążenia	70mA
Siła ciągu na przekładni (6V)	5.5kg/cm

Do poruszania się po płaskim podłożu robot wykorzystuje dwa koła o średnicy 65mm z gumową oponą, których prędkością i kierunkiem obracania steruje się za pomocą silników prądu stałego 6V. Dodatkowo robot posiada jedno koło podporowe z przodu.

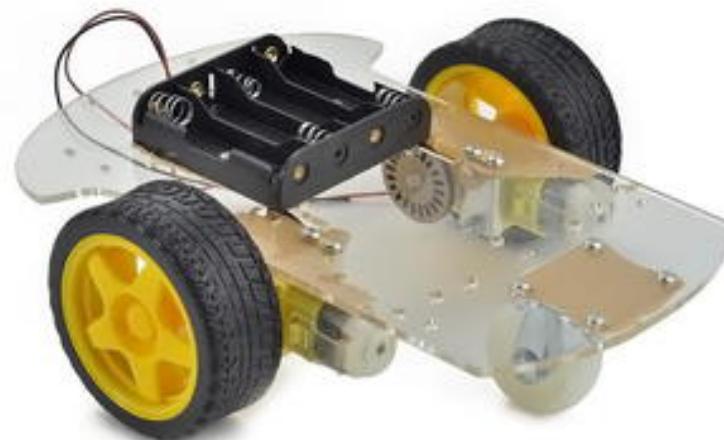
Elementem zasilającym silniki są 4 baterie AA osadzone w koszyku na obudowie. Całość podwozia zaprezentowano na rysku 2.3.

2.1.2 Czujniki

Podstawowym zadaniem robota jest poruszanie się wzduż czarnej linii, aby zrealizować to zadanie robot został wyposażony w dwa cyfrowe czujniki wykrywające przerwanie wiązki (w tym przypadku wjechanie na czarną linię). W celu uchronienia robota od czołowych



Rysunek 2.2 Koła pojazdu a)napędowe b)podporpwe



Rysunek 2.3 Podwozie robota mobilnego

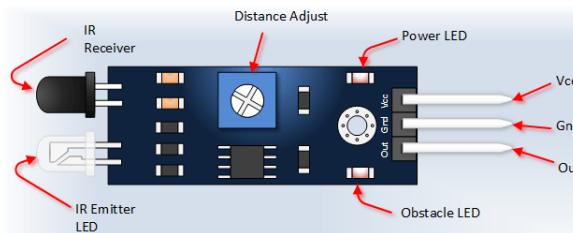
oraz tylnych kolizji został on wyposażony w dwa cyfrowe ultradźwiękowe czujniki odległości. Jeden z takich czujników znajduje się na czołku robota, drugi zaś jest zamontowany na tylu robota. Dodatkowo w celu uniknięcia spadku ze schodów, czy wjazdu na nieporządaną powierzchnię zamontowano cyfrowy czujnik dźwięku, który po wyłapaniu głośnego dźwięku na przykład klaśnięcia dłońmi natychmiast wyśle sygnał do płytki, która wymusi awaryjne i natychmiastowe zatrzymanie robota. Specyfikacje poszczególnych czujników zebrane w tabelach 2.2-2.4.

Moduł pozwalający na wykrywanie linii jest zbudowany z kilku elementów, z pośród których najważniejsze są: transmitter podczerwieni (IR Emitter), odbiornik podczerwieni

Tabela 2.2 Charakterystyka modułu z czujnikiem IR

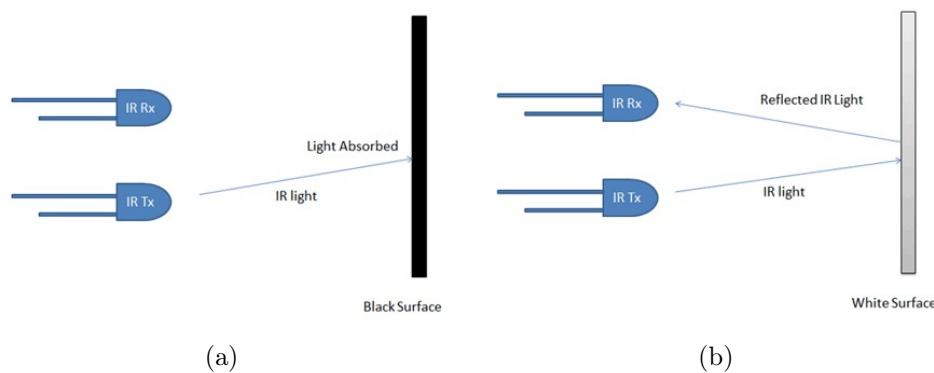
Rodzaj sygnału wjściowego	cyfrowy
Znamionowe napięcie pracy	3.3V - 5V DC
Zasięg działania	2 - 60 cm
Kąt pomiaru	35 stopni
Czas reakcji	do 0.5s
Układ porównujący	LM393
Brak przeszkody	Pin OUT-HIGH
Wykrycie przeszkody	Pin OUT-LOW

(IR Receiver), komparator napięcia (LM393) potencjometr ułatwiający podawanie napięcia do komparatora, co pozwala na regulację czułości sensora (Distance Adjust) oraz 3 pinów: VCC 5V, GND, OUT. Dodatkowo na płytce modułu można znaleźć 2 diody czerwone, jedna z nich informuje nas o podanym zasilaniu (Power LED), a druga wykryciu przeszkody (Obstacle LED) (w tym przypadku natrafienie na czarną linię). Wszystkie wymienione elementy pokazano na rysunku 2.4.



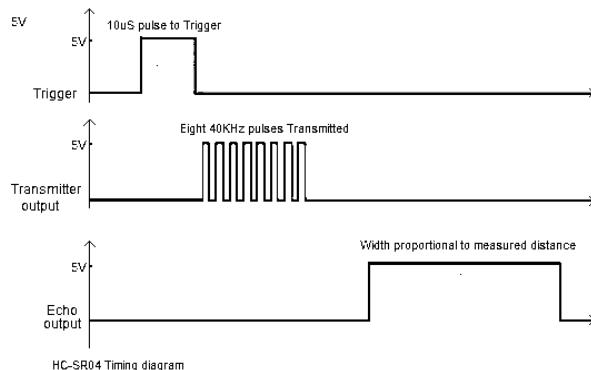
Rysunek 2.4 Moduł do wykrywania linii

Działanie czujnika opiera się na ciągłym wysyłaniu promieniowania podczerwonego przez diodę transmitującą (IR Tx) oraz jego odbioru po odbiciu od powierzchni przez diodę odbierającą (IR Rx). Wraz ze spadkiem sygnału wchodzącego do diody odbierającej następuje spadek napięcia podawanego do komparatora. Następnie sygnał z diody IR Rx wędruje do komparatora, który porównuje go z sygnałem z potencjometru i decyduje o wykryciu lub niewykryciu przeszkody. Gdy napięcie Badanego sygnału jest wyższe niż te podane z potencjometru na wyjściu pojawi się napięcie 5V informujące o braku przeszkody. Gdy sygnał z diody IR Rx jest mniejszy niż ten z potencjometru na wyjściu modułu pojawi się stan niski, jest to informacja o napotkaniu przeszkody.



Rysunek 2.5 Zachowanie transmitera i odbiornika podczerwonego w przypadku a) wykrycia b) niewykrycia przeszkody

Kolejnym elementem są cyfrowe ultradźwiękowe czujniki odległości, które wykorzystywane są w celu wykrycia przeszkody znajdującej się naprzeciw czujnika w zasięgu nie mniejszym niż 2 cm i nie większym niż 4m. Moduł zawiera ultradźwiękowy transmitter sygnału, odbiornik sygnału odbitego oraz układ kontrolny. Jego działanie polega na podaniu na pin wejściowy (TRIG) sygnału 5V przez co najmniej 10uS. Następnie moduł automatycznie wysyła 8 sygnałów o częstotliwości 40HZ i sprawdza czy odbiornik zarejestrował sygnał powracający po odbiciu od przeszkody. Jeżeli sygnał zostanie odebrany to na wyjściu modułu na pinie (ECHO) pojawi się napięcie które trafia następnie do płytki głównej. Czas trwania stanu wysokiego na pinie ECHO jest czasem potrzebnym fali dźwiękowej na dotarcie i powrót od przeszkody. Zobrazowano to na rysunku 2.6 Poniżej w tabeli 2.3 zebrano specyfikację techniczną modułu HC-SR04.

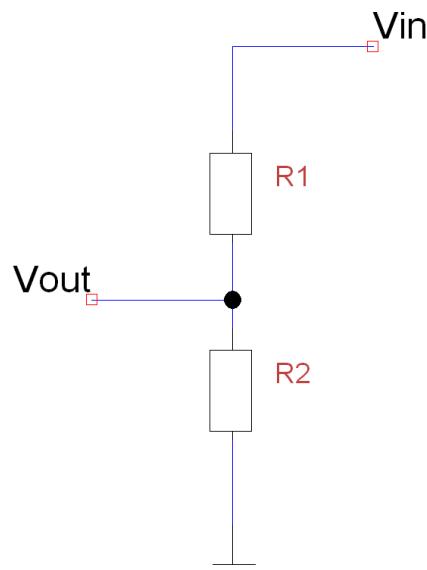


Rysunek 2.6 Napięcia na wejściu (Trigger), wyjściu transmittera oraz na wyjściu modułu - na pinie ECHO

Zgodnie z dokumentacją techniczną urządzenia nie zalecane jest bezpośrednie łączenie pinu ECHO z płytą Raspberry. Napięcie na wyjściu powinno przejść przez dzielnik napięcia, aby w ostatczności sygnał wędrujący do płytki głównej wynosił około 3.3V. Dzielnik napięcia pokazano na rysunku 2.7

Tabela 2.3 Charakterystyka ultradźwiękowego czujnika odległości HC-SR04

Rodzaj sygnału wyjściowego	cyfrowy
Znamionowe napięcie pracy	5V DC
Zasięg działania	2cm - 400cm
Kąt pomiaru	15 stopni
Częstotliwość pracy	40Hz
Znamieniowy prąd	15mA
Czas impulsu wyzwalanego	10us
Dokładność pomiaru	3mm



Rysunek 2.7 Dzielnik napięcia

W pierwszym etapie zapisujemy równość na V_{out}

$$V_{out} = V_{in} * \frac{R2}{R1 + R2}$$

W kolejnym etapie wyliczamy $R2$ przyjmując $R1=1.5\text{k}\Omega$

$$\frac{V_{out}}{V_{in}} = \frac{R2}{R1 + R2}$$

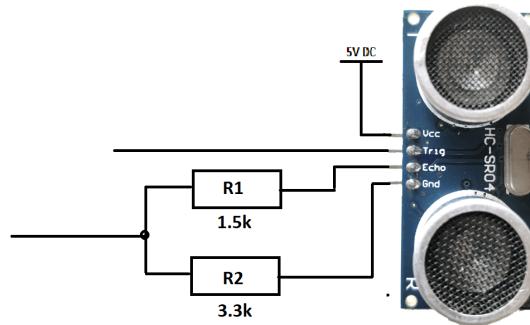
$$\frac{3.3V}{5V} = \frac{R2}{1500\Omega + R2}$$

$$990 + 0.66R2 = R2$$

$$990 = 0.34R2$$

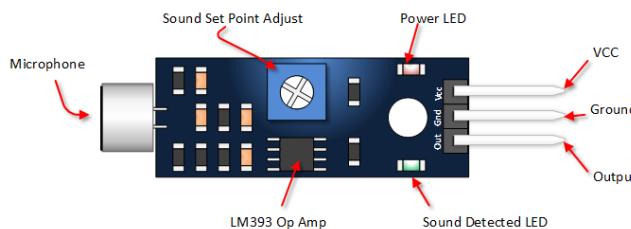
$$2911.76\Omega = R2$$

W ostateczności przyjmujemy rezystor $R2$ o rezystancji równej $3.3\text{k}\Omega$ Rysunek poniżej obrazuje połączenie modułu z dzielnikiem napięcia



Rysunek 2.8 Połączenie HC-SR04 z dzielnikiem

Do realizacji zadania implementacji hamowania awaryjnego niezbędne było użycie modułu z czujką dźwiękową. Głównymi komponentami modułu są: mikrofon, komparator napięcia (LM393), potencjometr, dioda sygnalizująca zasilanie, dioda informująca o wykryciu dźwięku oraz 3 piny: VCC 5V, GND, OUT. Wymienione powyżej elementy zaznaczono na rysunku 2.9



Rysunek 2.9 Moduł do wykrywania linii

Specyfikacja techniczna urządzenia znajduje się w tabeli 2.4

Tabela 2.4 Charakterystyka czujki dźwięku

Rodzaj sygnału wyjściowego	cyfrowy
Znamionowe napięcie pracy	3.3V-5V DC
Czułość	48-66dB
Impedancja	2.2Ω
Częstotliwość pracy	50Hz-20kHz
Komparator napięcia	LM393

Czujka ma za zadanie wyłapać krótki sygnał dźwiękowy wydawany przy kłaśnięciu dłońmi. Po wyzwoleniu sygnału fala dźwiękowa trafia na membranę mikrofonu wprawiając ją w drgania. Następnie dzięki elektromagnesowi wzywala się prąd o napięciu zależnym od natężenia dźwięku. Sygnał następnie wędruje do komparatora napięcia (LM393), który porównuje go z napięciem podawanym z potencjometru. Gdy napięcie sygnału przychodzącego z mikrofonu jest wyższe niż to otrzymane z potencjometru wysyła on sygnał na wyjście modułu. W tym przypadku sygnał na pinie OUT poinformuje nas o wykryciu dźwięku.

Elementem pozwalającym na sterowanie silników jest moduł TB6612FNG. Pozwala on na sterowanie napięciem podawanym na silniki oraz pozwala na obroty silników w dwóch kierunkach. Dane techniczne modułu znajdują się w tabeli 2.5

Tabela 2.5 Charakterystyka modułu TB6612FNG

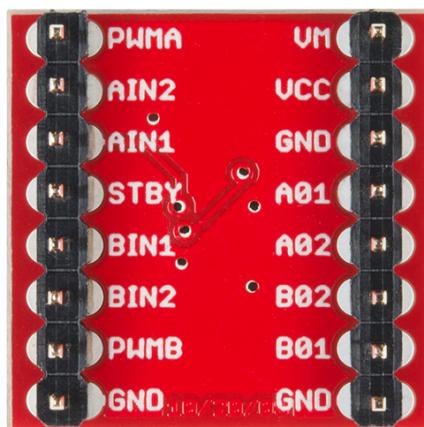
Rodzaj sygnałów wyjściowych	cyfrowy/analogowy
Znamionowe napięcie pracy VM	do 15V
Znamionowe napięcie pracy VCC	2.7-5.5V
Prąd na wyjściu	1.2A (średnio) 3.2A (szczyt)

W tabeli 2.6 zebrano opis pinów znajdujących się w module wraz z ich podziałem na wejścia i wyjścia:

Tabela 2.6 Charakterystyka modułu TB6612FNG

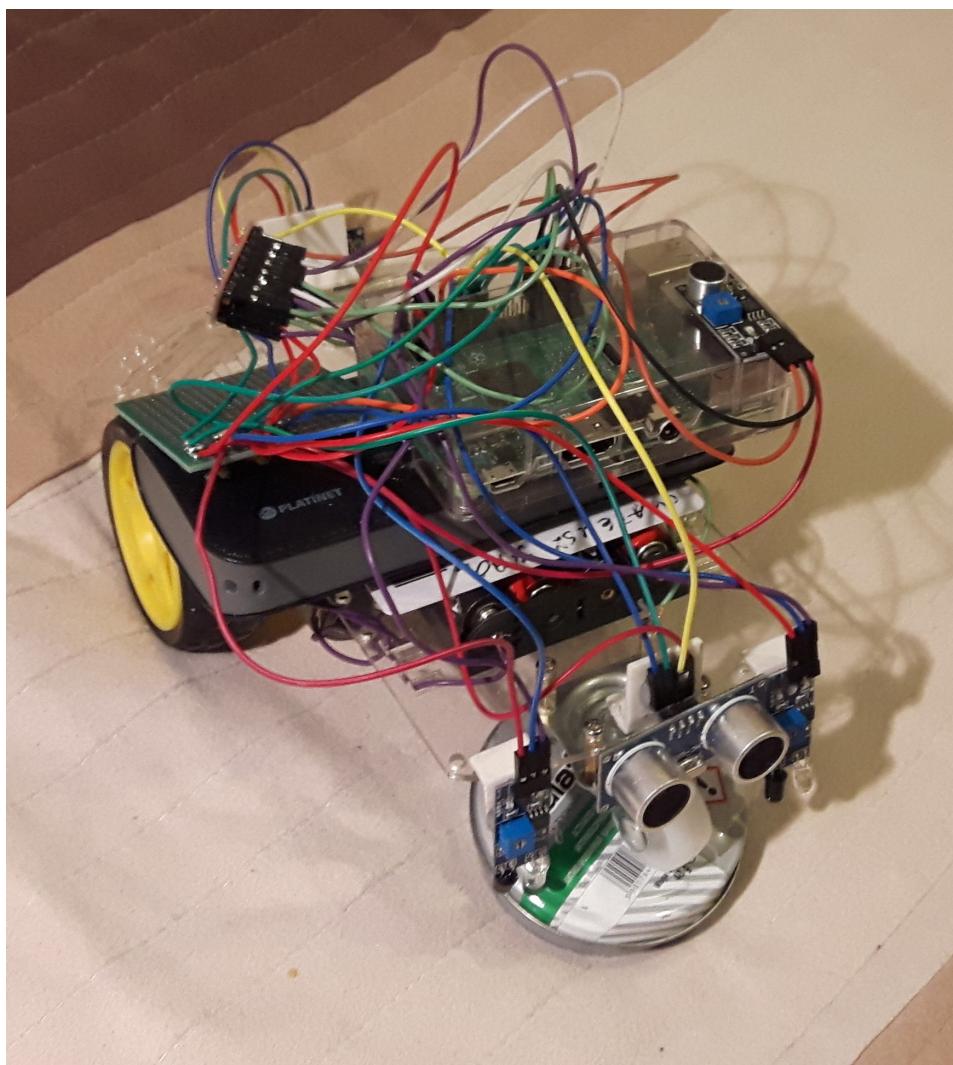
PIN	funkcja	Zasilanie Wejście Wyjście	Uwagi
VM	Zasilanie silników	Zasilanie	między 2.2V a 13.5V DC
VCC	Zasilanie części logicznej	Zasilanie	między 2.7V a 5.5 V DC
GND	Masa	Zasilanie	wspólne dla zasilania silników oraz kontrolera
STBY	Tryb czuwania	Wejście	Pozwala na przesyłanie zasilania silników i kontrolera
AIN1/BIN1	wejście 1 dla kanałów A/B	Wejście	Wybór kierunku obrotu silników
AIN2/BIN2	wejście 2 dla kanałów A/B	Wejście	Wybór kierunku obrotu silników
PWMA/PWMB	Wejście PWM dla kanałów A/B	Wejście	Pozwala na kontrolę prędkości silników
AO1/BO1	Wyjście dla kanałów A/B	Wyjście	Podanie zasilania na silniki
AO2/BO2	Wyjście dla kanałów A/B	Wyjście	Podanie zasilania na silniki

Na rysunku 2.10 pokazano moduł TB6612FNG wraz z podłączonymi silnikami.



Rysunek 2.10 Moduł do sterowania silnikami z podłączonymi silnikami

Całe sterowanie robota odbywa się w płytce Rasberry Pi 3 model B. Płytkę składa się z 4 rdzeniowego procesora o taktowaniu 1.2GHz. Posiada pamięć ram 1 Gb oraz kartę microSD, na której znajduje się system operacyjny Linux Raspbian oraz programy niezbędne do sterowania robotem. Płytkę jest zasilana napięciem 5V DC z powerbanku za pomocą kabla USB. Płytkę posiada dodatkowo 40 pinowe gniazdo GPIO, które pozwala na sterowanie sygnałami wejściowymi oraz wyjściowymi przychodzących albo wychodzących z poszczególnych komponentów robota. Płytkę posiada wiele interfejsów, spośród których użyto Wideo (wyjście HDMI) w celu podłączenia płytki z monitorem w początkowych fazach programowania płytki, WiFi do bezprzewodowej komunikacji oraz portu USB do podłączenia kamerki internetowej LifeCam HD-5000. Pełna specyfikacja techniczna płytki Raspberry Pi 3 model B znajduje się w dodatku C. Poniżej na zdjęciu 2.11 pokazano ostateczny wygląd robota po podłączeniu wszystkich składowych komponentów.



Rysunek 2.11 Robot z połączonymi komponentami

2.2 Sterowanie robotem

Sterowanie robotem można podzielić na 2 zasadnicze zadania. Pierwsze z nich to utrzymanie robota w torze jazdy, realizuje to przy pomocy czujników omawianych w rozdziale 2.1 "Komponenty" oraz odpowiedniemu programowi, który przetwarza sygnały przychodzące do płytki i steruje odpowiednio silnikami. Kolejnym etapem jest adaptacja zachowania robota do napotykanych znaków drogowych, poszczególne zachowania przedstawiono w dodatku B.

Robot od rozpoczęcia działania głównego programu porusza się do przodu badając swoje otoczenie z przodu i z tyłu w celu wykrycia ewentualnych przeszkód. Algorytm wykrycia przeszkód omówiono w rozdziale 2.2.2 "Unikanie przeszkód". Dodatkowo cały czas sprawdza czy nie natrafił na czarną linię, znajdującą się pomiędzy czujnikami podczerwieni. Gdy takie zdarzenie nastąpi, robot koryguje tor jazdy tak, aby od tej linii się odsunąć. Pełny algorytm opisano w rozdziale 2.2.1 "Wykrywanie linii".

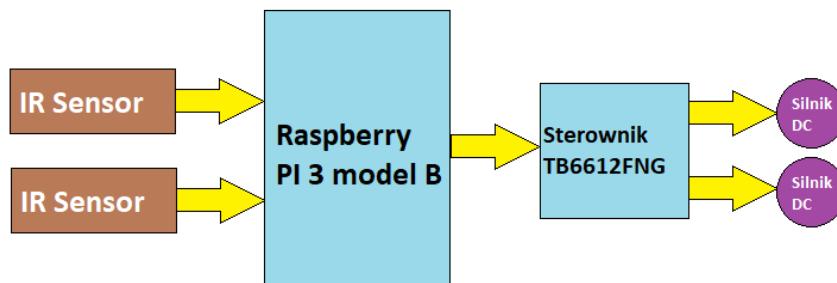
Do komunikacji z robotem używana jest sieć Wifi, jednak program skonstruowano tak, aby umożliwił dalszą pracę robota, nawet jeśli znajdzie się poza zasięgiem sieci bezprzewodowej. Urządzenie jest wtedy niesterowalne, więc niezbędne było wprowadzenia hamulca awaryjnego. Realizacja tego zadania omówiona jest w rozdziale 2.2.3 "Awaryjny STOP".

2.2.1 Wykrywanie linii

Niezbędnym wyposażeniem do wykrywania linii są dwa czujniki wykrywające przerwanie wiązki podczerwonej. Budowę takich czujników oraz zasadę ich działania omówiono w rozdziale 2.1.2 "Czujniki".

[6]

Czujniki zamontowano z przodu robota w odstępie około 5cm. Jeżeli robot zaczyna wyjeżdżać poza tor jeden z sensorów natrafia na czarną linię. Następuje wtedy przerwanie wiązki i na pinie OUT pojawi się stan wysoki. Sygnał ten wędruje do płytki Raspberry Pi i program sprawdza, czy sygnał przyszedł z lewego, czy z prawego sensora i odpowiednio koryguje tor jazdy używając silników sterowanych za pomocą sterownika TB6612FNG. Ogólny schemat blokowy przedstawiono na rysunku 2.12



Rysunek 2.12 Schemat blokowy - wykrywanie linii

Robot w czasie jazdy na wprost może natrafić tylko na 4 scenariusze. Omówione je poniżej oraz zobrazowano na rysunku 2.13

1. Żaden z sensorów nie wykrył linii

Teoretycznie linia powinna znajdować się pomiędzy sensorami. Robot ma się poruszać do przodu podając równą moc na oba silniki. W przypadku pesymistycznym, gdy robot "zgubi" linię i wyjedzie poza tor będzie poruszał się do przodu aż do natrafienia na przeszkodę albo do awaryjnego zatrzymania przez operatora. Aby zapobiec "wypadaniu" robota poza linię można zmniejszyć prędkość, lub zadbać o zwiększenie szerokości czarnej linii.

2. Lewy sensor natrafił na czarną linię

Gdy lewy sensor natrafi na czarną linię program musi szybko zmniejszyć prędkość lewego silnika albo go zatrzymać jednocześnie utrzymując prędkość silnika prawego albo zwiększając jego prędkość.

3. Prawy sensor natrafił na czarną linię

Gdy prawy sensor natrafi na czarną linię program musi szybko zmniejszyć prędkość prawego silnika albo go zatrzymać jednocześnie utrzymując prędkość silnika lewego albo zwiększając jego prędkość.

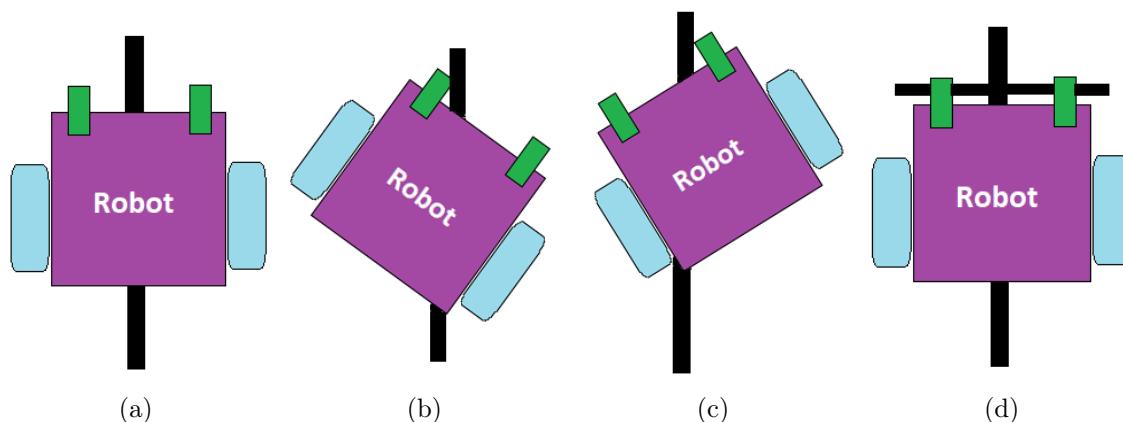
4. Obydwa sensory natrafiły na czarną linię.

Przypadek taki ma miejsce w dwóch sytuacjach:

-robot natrafił na koniec toru. W tym momencie robot zatrzyma się i będzie czekał na dalsze komendy operatora.

-skrzyżowanie dróg. Robot musi być poinformowany o takim zdarzeniu przy pomocy odpowiedniego znaku drogowego, pokazującemu układ skrzyżowania. Następnie wybierze jedną z możliwych dróg poruszania się np. jeżeli ma możliwość skrętu w lewo to na krótką chwilę zignoruje odczyty prawego sensora co poskutkuje skrętem robota w lewo.[5]

Jako że masa robota wynosi około 1.5kg zaleca się aby linia sygnalizująca koniec toru albo skrzyżowanie była przynajmniej dwa razy szersza niż ta, po której porusza się robot, ponieważ bezwładność robota wydłuża drogę hamowania co może poskutkować tym, że robot "przegapi" linię i wypadnie z toru.



Rysunek 2.13 Adaptacja robota do napotkania na czarną linię a) Jazda na wprost b) prawa kontra c) lewa kontra d) STOP

2.2.2 Unikanie przeszkód

Niezbędnym elementem do wykrywania przeszkód jest ultradźwiękowy czujnik odległości HC-SR04. Budowę tego modułu oraz jego parametry techniczne omówiono w rozdziale 2.1.2 "Czujniki".

Zamontowano po jednym czujniku z przodu i z tyłu robota. Umożliwiło to wykrywanie zderzeń czołowych oraz nieoczekiwanych uderzeń w tył pojazdu. Najczęściej czujniki te znajdą swoje zastosowanie w momencie gdy robot wypadł z toru jazdy i porusza się "bezwiednie" po pomieszczeniu w poszukiwaniu czarnej linii, wzduż której ma się poruszać. Ma to zapobiec uderzeniom czołowym w ścianę pomieszczenia, co w konsekwencji mogłoby naruszyć konstrukcję robota.

W czasie jazdy co 0.5s podawane jest na pin TRIG czujnika HC-SR04 napięcie 5V, co skutkuje wyzwoleniem 8 impulsów o częstotliwości 40 kHz, które po odbiciu od przeszkody wracają do odbiornika, co sygnalizowane jest pojawieniem się stanu wysokiego na pinie ECHO. Stan wysoki utrzyma się na pinie Echo od momentu wyzwolenia ostatniego impulsu z transmittera aż do otrzymania pierwszego impulsu po odbiciu od przeszkody. Raspberry musi w tym czasie wykonać pomiar czasu trwania stanu wysokiego na pinie ECHO. Pozwoli to na przeliczenie odległości w jakiej znajduje się przeszkoda. Następnie znając czas stanu wysokiego pinu ECHO i przyjmując prędkość dźwięku 340 m/s możemy ustalić w jakiej odległości znajduje się przeszkoda. [4]

$$D = \frac{\Delta t}{2} * c$$

gdzie:

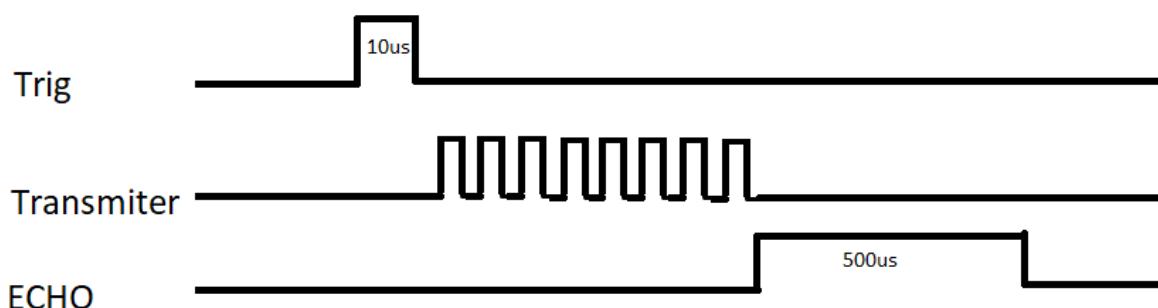
D - zmierzony dystans

Δt - czas trwania stanu wysokiego na pinie ECHO

c - prędkość dźwięku ≈ 340 m/s

Moduł dba o to, aby wyłączyć stan wysoki pinu ECHO po upływie 38mS, co pozwala na zakończenie pomiaru gdy przeszkoda nie została wykryta.

Przykładowy pomiar przedstawiono na rysunku 2.14 wraz z obliczeniami poniżej. Uwaga skala czasu nie została zachowana.



Rysunek 2.14 Badanie odległości umiejscowienia przeszkody

$$D = \frac{0.0005s}{2} * 340 \frac{m}{s}$$

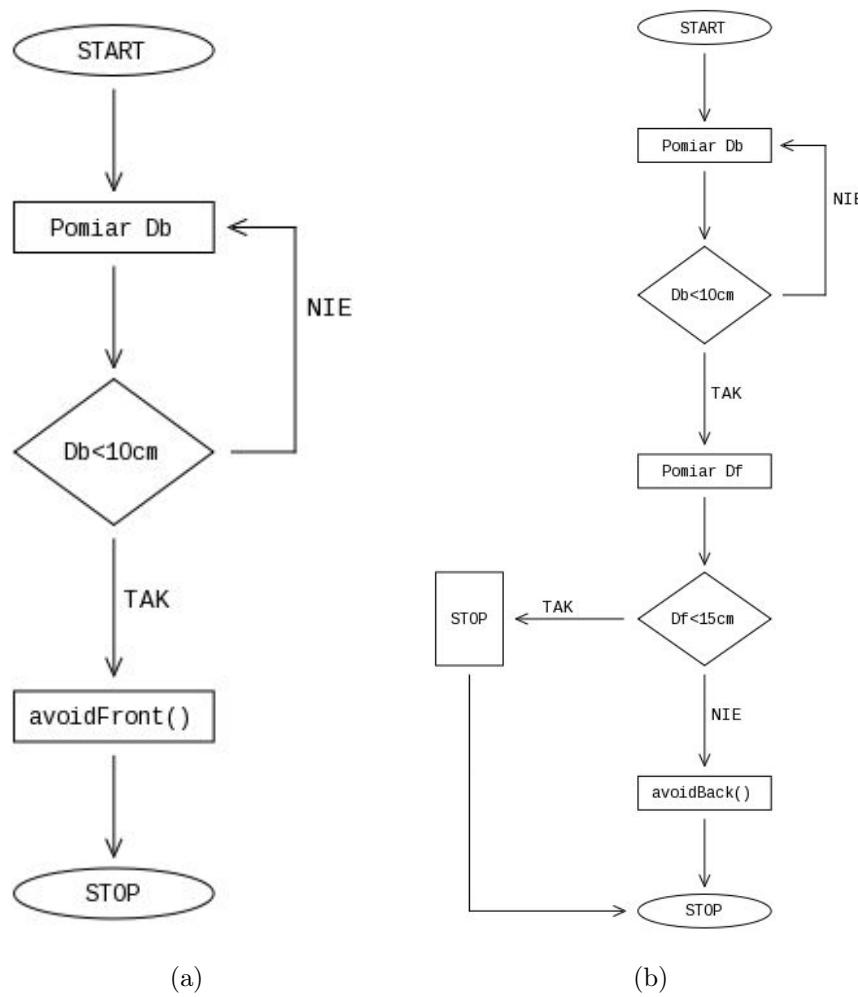
$$D = 0.085m$$

Ostatecznie możemy określić, że przeszkoda znajduje się w odległości 8.5cm od czujnika. Producent deklaruje dokładność czujnika na poziomie 0.3cm.

Robot posiada zaimplementowane 2 algorytmy do wykrywania i omijania przeszkód:

1) Robot trafia na przeszkodę znajdującą się przed nim. Raspberry dokonało pomiaru stanu wysokiego pinu ECHO i obliczenia wskazały, że przeszkoda znajduje się w odległości Df. Algorytm zakłada, że jeśli wyliczona odległość Df jest mniejsza niż 10cm należy uruchomić funkcję `avoidFront()`, która ma natychmiast zatrzymać silniki, następnie odczekać 0.5s (dodatkowy czas potrzebny na wytracenie prędkości spowodowanej bezwładnością pojazdu) i następnie wykonać manewr jazdy do tyłu przez 1s i w ostatniej fazie skręcić robota w lewo. Po tym manewrze robot znów będzie poruszał się do przodu.

2) Robot wykrył zbliżającą się przeszkodę z tyłu pojazdu. Raspberry dokonało pomiaru stanu wysokiego na pinie ECHO czujnika znajdującego się z tyłu pojazdu i po wykonaniu obliczeń okazuje się, że przeszkoda znajduje się w odległości Db od robota. Algorytm zakłada, że jeśli odległość Db jest mniejsza niż 10cm robot ma natychmiast dokonać pomiaru odległości z przodu Df i jeśli jest ona większa niż 15cm ma uruchomić funkcję `avoidBack()`, która każe robotowi poruszać się z prędkością maksymalną do przodu w celu ucieczki przed przeszkodą, jednocześnie cały czas sprawdzając, czy Df nie jest mniejszy niż 15cm. Jeśli pomiar Db wyniesie mniej niż 10cm i odległość Df będzie również mniejsza niż 15cm robot zatrzyma się. Obydwa algorytmy pokazano na schemacie 2.15 blokowym zaś funkcję `avoidFront()` oraz `avoidBack()` omówiono dokładniej w rozdziale 2.3 "Testy silników oraz czujników".



Rysunek 2.15 Zachowanie robota po wykryciu przeszkody a) z przodu b) z tyłu

2.2.3 Awaryjny STOP

Do zatrzymania robota można posłużyć się czujką dźwiękową, której budowę oraz parametry omówiono w rozdziale 2.1.2 "Czujniki". Moduł zamontowano na obudowie, z mikrofonem skierowanym do góry. Ustawienie takie ma za zadanie uniknięcia szumów silników w czasie jazdy robota. Czujkę należy wsterować za pomocą potencjometru, do takiego poziomu, aż zgaśnie dioda sygnalizująca wykrycie sygnału. Zaleca się, aby dać pewien margines zapasu, aby uniewrażliwić mikrofon na szумy otoczenia.

Algorytm ciągle sprawdza, czy na pinie OUT czujki nie pojawiło się napięcie wysokie. Napięcie takie docelowo ma być spowodowane przy głośnym klaścięciu dłońmi. Po wykryciu dźwięku płytki Raspberry natychmiast wysyła sygnał awaryjnego zatrzymania robota

2.3 Testy silników oraz czujników

Przed uruchomieniem głównego opragromowania robota zaleca się odzielne testy silników oraz czujników robota. W tym celu napisano 2 programy, które możemy uruchomic bezpośrednio z poziomu płytki Raspberry PI 3 model B:

`motortest.py`

Ten skrypt ma zdefiniowane poruszanie się robota. Ma na celu sprawdzenie poprawnego podłączenia silników ze sterownikiem TB6612FNG oraz z płytą Raspberry Pi.

Program uruchamiamy w oknie terminala w systemie Linux Raspbian komendą:

`python motortest.py`

Program ma zdefiniowane 9 funkcji poruszania się robota i są to:

Tabela 2.7 Funkcje sterowania silnikami

Nazwa funkcji	Zadanie
<code>forwardDrive()</code>	Jazda na wprost
<code>reverseDrive()</code>	Jazda robota w tył
<code>spinLeft()</code>	Obrót robota "w miejscu" w lewo
<code>SpinRight()</code>	Obrót robota "w miejscu" w prawo
<code>forwardTurnLeft()</code>	Pozwala na skręt robota w lewo w przód
<code>forwardTurnRight()</code>	Pozwala na skręt robota w prawo w przód
<code>reverseTurnLeft()</code>	Pozwala na skręt robota w plewo w tył
<code>reverseTurnRight()</code>	Pozwala na skręt robota w prawo w tył
<code>allStop()</code>	Całkowite zatrzymanie silników

Dodatkowo zdefiniowano funkcję `main()`, która sprawdza każdą funkcjonalność z tabeli 2.7 przez 1s. Jeśli widzimy, że wszystkie ruchy zostały wykonane poprawnie możemy uznać, że silniki są połączone poprawnie ze sterownikiem, a ten poprawnie współpracuje z płytą Raspberry. W kolejnym teście sprawdzimy poprawność działania oraz połączeń czujników.

W celu przetestowania poprawności działania czujników napisano program, który możemy uruchomić bezpośrednio z poziomu płytki wpisując w terminalu komendę

```
python newlinefollower.py
```

Skrypt ten oprócz funkcjonalności ze skryptu motortest.py ma dodatkowo zaimplementowane jazdę z unikaniem czarnej linii, wykrywanie oraz omijanie przeszkód.

Dotatkowe funkcje zawarto w tabeli 2.8

Tabela 2.8 Funkcje sterowania silnikami

Nazwa funkcji	Zadanie
frontSensor()	Zwraca dystans przeszkody w cm
backSensor()	Zwraca dystans przeszkody w cm
avoidFront()	realizuje omijanie przeszkody z przed pojazdu
avoidBack(distance frontside)	Realizuje omijanie przeszkody z tyłu pojazdu

Poniżej przedstawiono fragment kodu z funkcji main() programu. Widzimy, że program sprawdza odczyty lewego i prawego sensora przerwania wiązki i adoptuje się do ich odczytów, jednocześnie badając odległości robota od przeszkód napotykanych z przodu oraz z tyłu.

```
distance_frontside=frontSensor()
distance_backside=backSensor()
print(distance_frontside)
if distance_frontside<15:
    avoidFront()
if distance_backside<10:
    avoidBack(distance_frontside)
elif(IO.input(LEFT_IR)==True and IO.input(RIGHT_IR)==True): #both white move
    allStop()
    print("stop")
elif(IO.input(LEFT_IR)==False and IO.input(RIGHT_IR)==True): #turn right
    forwardTurnRight()
    print("right turn")
elif(IO.input(LEFT_IR)==True and IO.input(RIGHT_IR)==False): #turn left
    forwardTurnLeft()
    print("left turn")
else:
    forwardDrive()
    print("forward move")
```


Rozdział 3

Algorytmy rozpoznawania

Kolejnym założeniem projektu było zaimplementowanie algorytmów pozwalających na wykrywanie i klasyfikację modeli znaków drogowych w czasie rzeczywistym. W tym celu wykorzystano SVM (Maszynę wektorów nośnych/podporowych). Realizuje ona zadanie klasyfikacji obiektów do danej klasy. Możliwe jest to dzięki wcześniejszej nauce algorytmu. Aby nauka algorytmu klasyfikującego miała lepsze rezultaty obrazy wcześniej poddano przetwarzaniu. Polegało ono na odpowiednim przycinaniu obrazu, jego skalowaniu oraz przedstawienia go w postaci Histogramu Gradientów Zorientowanych (HOG). W tym rozdziale skupimy się na opisie teoretycznym przetwarzania obrazu do postaci histogramu gradientów zorientowanych oraz teoretycznym opisie algorytmu klasyfikującego obrazy do poszczególnych klas.

3.1 HOG

HOG (ang. Histogram of Oriented Gradients) jest jednym z wielu deskryptorów cech zawartych w obrazie. Ma on za zadanie "uprościć" obraz akcentując jedynie jego istotne cechy, pozbywając się jednocześnie tych, które okazują się być zbedne. W przypadku detekcji znaków drogowych niezbędne jest uwydatnienie z obrazu takich cech jak krawędzie czy kąty. Pozwoli nam to z dużym prawdopodobieństwem zklasyfikować dany obiekt jako znak drogowy albo jako tło, które żadnego znaku drogowego nie zawiera. Do tego zadania doskonale sprawdza się przedstawianie obrazu w postaci HOG-u, ponieważ jest on doskonałym algorytmem do wykrywania krawędzi. Algorytm można podzielić na kilka kroków.

1. Wstępne przetwarzanie.

Na tym etapie istotne jest odpowiednie przygotowanie obrazu do dalszej analizy. Niedozwolone jest, aby obrazy były tych samych wymiarów. Wymaga to od nas odpowiedniego przycinania obrazów oraz ich przeskalowywanie, tak aby spełnić następny warunek, jakim jest to, że wszystkie obrazy muszą być takich samych wymiarów. Zalecane jest również, aby rozmiary obrazu zawarte były w balansie (szerokość, wysokość) 1:2, czyli np. rozmiar obrazu wynosi 64 x 128 [px]. Ostatnim krokiem wstępnego przetwarzania jest przejście do skali czarno-białej.

2. Obliczenie gradientów obrazu.

Aby obliczyć deskryptor HOG niezbędne jest wyliczenie horyzontalnych oraz wertykalnych gradientów. W tym celu obraz jest filtrowany używając masek $\begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$ poziomo oraz $\begin{bmatrix} -1 & 0 & 1 \end{bmatrix}^T$ pionowo. Pozwoli nam to na wykrycie krawędzi odpowiednio w poziomie i w pionie. Aby policzyć długość gradientu oraz jego orientację niezbędne jest wykorzystanie tw.

Pitagorasa i tak:

$$g = \sqrt{g_x^2 + g_y^2}$$

$$\theta = \arctan \frac{g_y}{g_x}$$

gdzie g to długość gradientu a θ jego kąt kierunku.

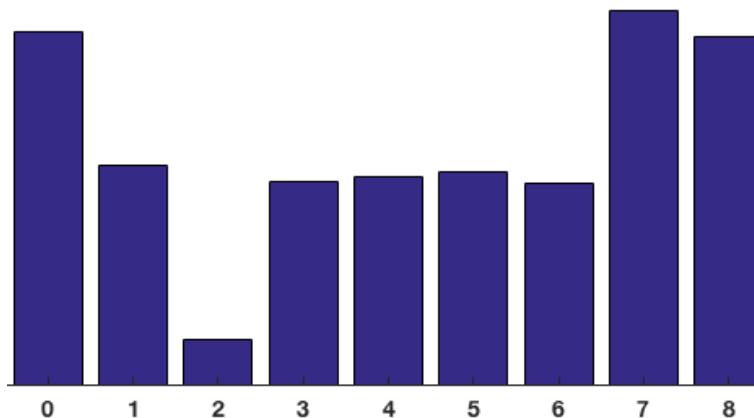
3. Policzenie HOGu w komórkach 8x8 [px]

Na tym etapie skupimy się na policzeniu HOGu dla każdej z komórek 8x8 [px]. Zabieg podziału obrazu na komórki stosuje się w celu zminimalizowania wpływu szumów na obraz. Histogram komórek 8x8 [px] jest dużo mniej podatny na szумy, niż indywidualne gradienty. Teraz tworzy się histogram gradientów zorientowanych. Przy pomocy poszczególnych wartości z macierzy nasycenia gradientu i macierzy kierunku gradientu. Przykładowe macierze dla komórki 8x8[px] pokazano poniżej

$$\begin{bmatrix} 80 & 36 & 5 & 10 & 0 & 64 & 90 & 73 \\ 37 & 9 & 9 & 179 & 89 & 27 & 169 & 166 \\ 87 & 136 & 173 & 39 & 102 & 163 & 152 & 176 \\ 76 & 13 & 1 & 168 & 159 & 22 & 125 & 143 \\ 120 & 70 & 14 & 150 & 145 & 144 & 145 & 145 \\ 58 & 86 & 119 & 98 & 100 & 101 & 133 & 113 \\ 30 & 65 & 157 & 75 & 78 & 165 & 145 & 124 \\ 11 & 170 & 91 & 4 & 110 & 17 & 133 & 110 \end{bmatrix} \cdot \begin{bmatrix} 2 & 3 & 4 & 4 & 3 & 4 & 2 & 2 \\ 5 & 11 & 17 & 13 & 7 & 9 & 3 & 4 \\ 11 & 21 & 23 & 27 & 22 & 17 & 4 & 6 \\ 23 & 99 & 165 & 135 & 85 & 32 & 26 & 2 \\ 91 & 155 & 133 & 136 & 144 & 152 & 57 & 28 \\ 98 & 196 & 76 & 38 & 26 & 60 & 170 & 51 \\ 165 & 60 & 60 & 27 & 77 & 85 & 43 & 136 \\ 71 & 13 & 34 & 23 & 108 & 27 & 48 & 110 \end{bmatrix}$$

Można zauważyć, że w macierzy kierunku wyraźnie zarysuje się krawędź zawarta w obrazie, ponieważ widać nagłe skoki wartości w poszczególnych elementach macierzy. Można dostrzec również, że znaleziona krawędź jest zaokrąglona. Macierz reprezentuje jedną z ramek z rysunku 3.2, który pokazuje nam odpowiedź algorytmu dla znaku STOP. Ramka pochodzi z górnej krawędzi litery 'O'

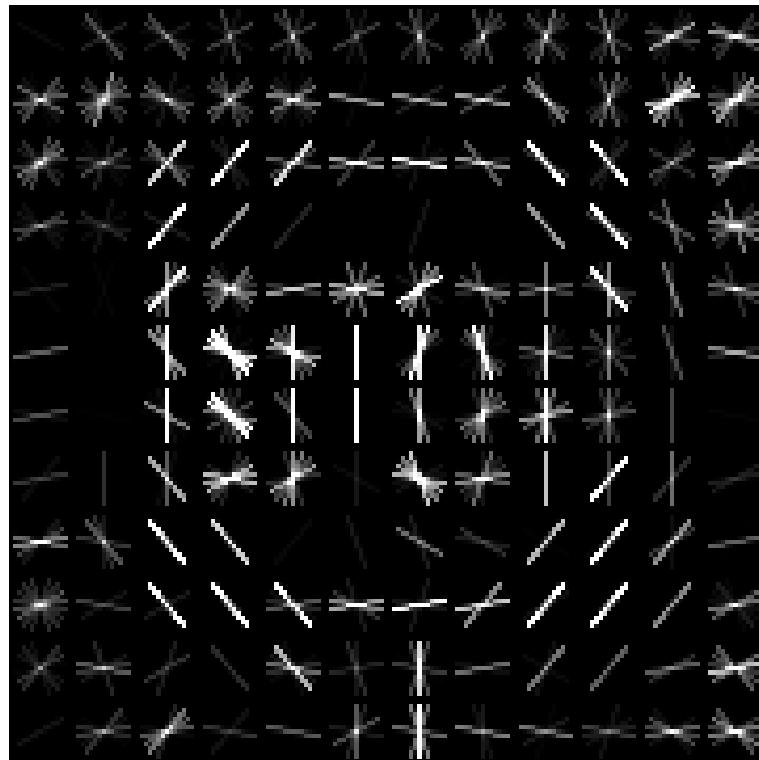
Znając wartości macierzy nasycenia oraz macierzy kierunku możemy stworzyć finalną wersję histogramu wektorów zorientowanych dla danej ramki. Wykonujemy to dzieląc oś x histogramu na odcinki po 20 i następnie wypełniamy histogram biorąc wartość nasycenia gradientu i wstawiając ją w polu odpowiadającym kątowi gradientu. Jeżeli wartość znajduje się w wartościach różnych niż wielokrotności 20 to odpowiednio dzielimy wagę. W ostateczności dla macierzy przedstawionych powyżej otrzymamy histogram gradientów zorientowanych jak ten na rysunku 3.1



Rysunek 3.1 HOG dla fragmentu 8x8[px]

4. Normalizacja bloków

Celem tego etapu jest uniewrażliwienie obrazu na zmiany nasycenia kolorów. Zaleca się normalizację obszarów większych niż pojedyncza ramka. W tym algorytmie zastosowano blok o rozmiarze 16x16 [px]. Zawiera on w sobie 4 histogramy gradientów zorientowanych. Można je przedstawić w postaci wektora o wymiarze 36x1 i znormalizować go używając normę L2 wektora. Następnie przesuwamy nasz blok o 8 [px] i powtarzamy operację aż do unormowania wszystkich bloków. Wizualizację efektu działania algorytmu w krokach 1-4 pokazuje rysunek 3.2



Rysunek 3.2 Prezentacja efektu algorytmu HOG na przykładzie znaku drogowego

5. Obliczenie ostatecznego wektora histogramu gradientów zorientowanych

Na początku policzymy ile bloków 16x16[px] znajduje się na naszym obrazie:

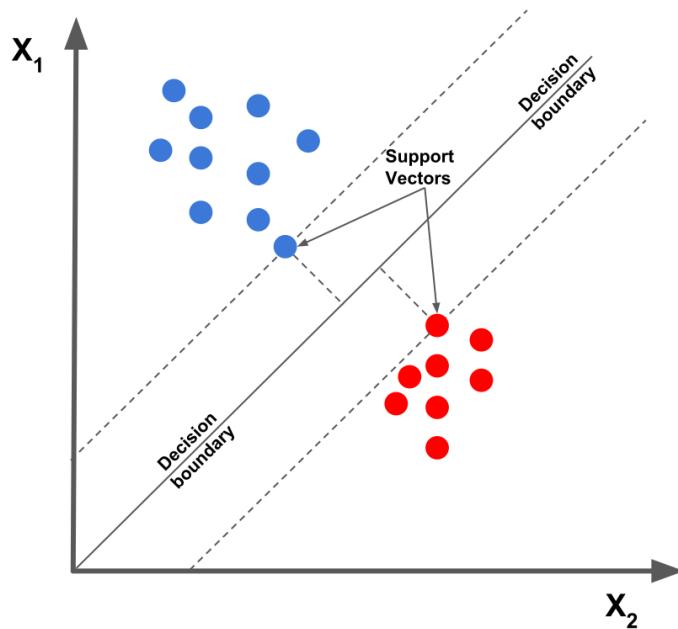
$$7 \times 15 = 105$$

Wiedząc, że każdy z bloków jest reprezentowany wektorem 36x1 możemy w ostateczności policzyć wymiar naszego wektora

$$105 \times 36 = 3780$$

3.2 SVM

SVM (ang. Support Vector Machine) jest jednym z wielu algorytmów używanych do uczenia maszynowego. Rozwiązuje on problem klasyfikacji binarnej. Ma on za zadanie odseparować obiekty z dwóch różnych klas za pomocą hiperpłaszczyzny (Decision boundary) z możliwie największym marginesem (Support Vector). Przykładowy podział obiektów na 2 różne klasy na podstawie 2 cech (x_1 i x_2) pokazano na rysunku 3.3. W tym projekcie użyto SVMu do klasyfikacji znaków drogowych więc możemy potraktować rysunek 3.3 jak podział znaków ze względu na 2 cechy np. x_1 - procentowa wartość koloru czerwonego w zdjęciu, x_2 procentowy udział koloru niebieskiego w obrazie. W ten sposób możemy rozróżnić znak informacyjny od znaku STOP. SVM jest algorymem wymagającym podania ciągu uczącego, czyli nasze obiekty w fazie testów muszą mieć przypisaną przynależność do poszczególnych klas.



Rysunek 3.3 Podział obiektów do 2 klas na podstawie cech x_1 i x_2

Na rysunku 3.2 widzimy, że hiperpłaszczyzna rozdzielająca dwie grupy obiektów znajduje się idealnie pośrodku. Środek ten został wyznaczony przy pomocy wektorów nośnych (Support Vectors), które zakładają największą możliwą odległość od obiektów. Optymalizacja długości wektorów nośnych realizowana jest przez maksymalizację marginesu geometrycznego:

$$\min_{x_i \in T} y_i \frac{x^T x_i + b}{\|w\|}$$

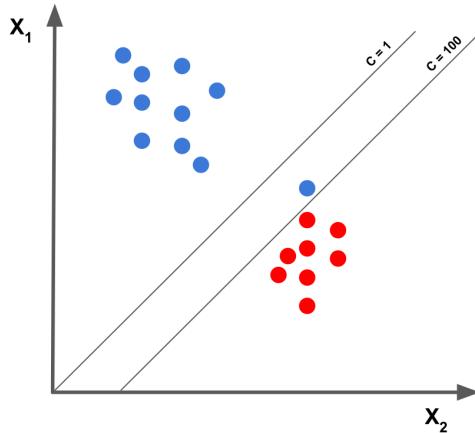
gdzie:

x_i to numer próbki

y_i przyjmuje wartości (-1 albo 1) w zależności od przynależności do klasy

w oraz b są parametrami odpowiedzialnymi za ustalenie nachylenia prostej

Problem pojawia się, kiedy obiekt z klasą x_1 znajdzie się bliżej obiektów klasy x_2 niż swojej klasy x_1 . Pokazano taką sytuację na rysunku 3.4. Problem taki rozwiązujemy za pomocą parametru C, który mając niską wartość wyznaczy margines decyzyjny większy, ale może zdarzyć się, że jakiś obiekt zostanie źle sklasyfikowany, można powiedzieć, że większą wagę przykłada się do środka ciężkości zbioru obiektów, niż tylko do skrajnego obiektu danej klasy ($C=1$), zwiększając parametr C zmniejszamy ryzyko błędnej klasyfikacji.



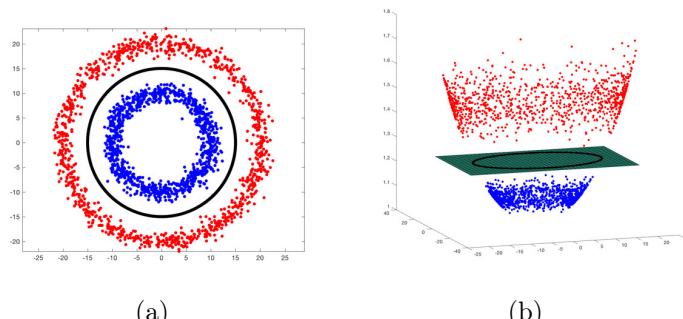
Rysunek 3.4 Regulacja parametru c w klasyfikacji

Nie ma jasno określonej reguły pomagającej wyznaczyć wartości parametru C w sposób najefektywniejszy. Nie zaleca się jednak dobierania dużych wartości dla parametru C jak pokazano na rysunku 3.4, ponieważ obiekt ten mógł zostać przypisany do złej klasy już w fazie uczenia, a my wymuszamy jego sklasyfikowanie do błędnej klasy i w ten sposób "przesuwamy" nasz margines decyzyjny, co w rezultacie może poskutkować złą klasyfikacją obiektów w fazie testów algorytmu.

Często okazuje się, że obiekty nie są separowalne liniowo w przestrzeni 2D. Niezbędne jest wtedy dodanie trzeciego wymiaru i mając szczęście obiekty staną się możliwe do separacji liniowej [3]. Zabieg ten nazywany jest "Kernel Trick" i dodaje on trzeci wymiar (z) do zbioru w sposób następujący.

$$z = e^{-y(x^2+y^2)}$$

Separacje obiektów z zastosowaniem "Kernel Trick" pokazano na rysunku 3.5.



Rysunek 3.5 a) obiekty nieseparowalne prostą b) obiekty odseparowane płaszczyzną po zastosowaniu "Kernel Trick"

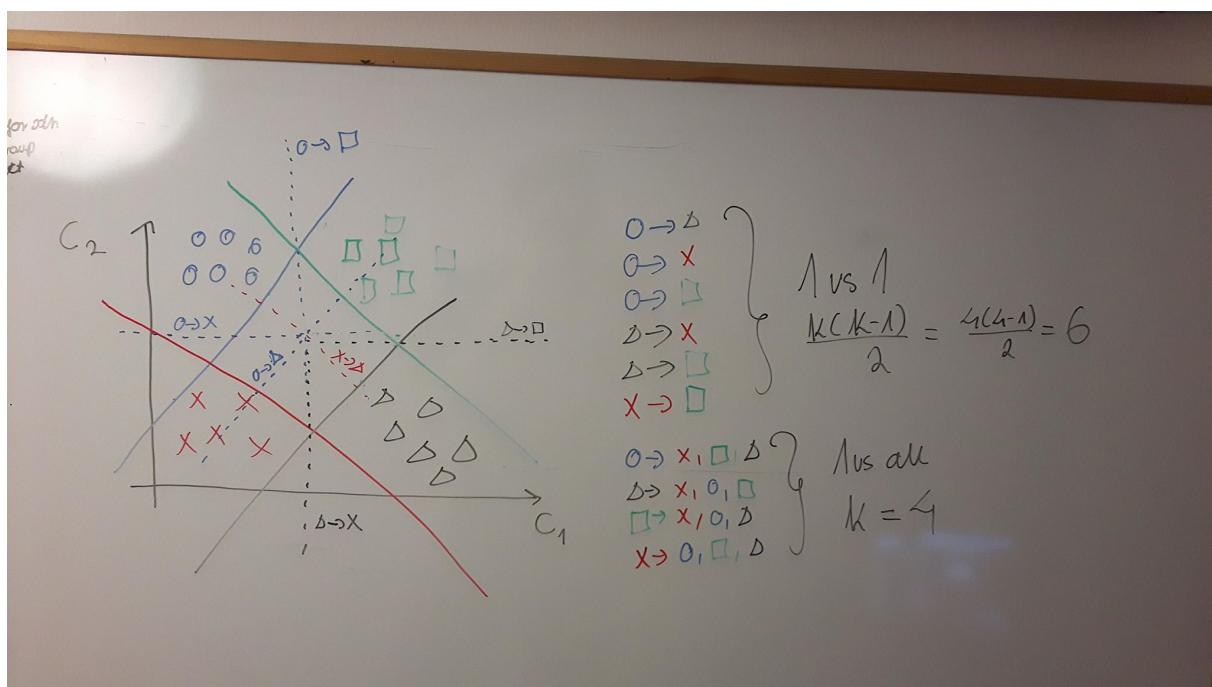
Jak wspomniano na początku bieżącego rozdziału SVM jest separatorem binarnym, czyli pozwala na przypisywanie obiektów jedynie do dwóch klas. Jednak założenia projektu wymagały klasyfikacji wielu znaków drogowych. Zadanie więc wymagało wytrenowania wielu maszyn wektorów nośnych. Możliwe jest to na wiele sposobów, spośród których dwa wydają się najrozsądzniejsze:

one-vs-all - taktyka ta zakłada wyuczenie klasyfikatora, który będzie traktował próbki jednej klasy jako pozytywne (+) a próbki wszystkich pozostałych klas będzie traktował negatywnie (-). W tej taktyce mając k klas musimy wytrenować $k(k-1)/2$ maszyn wektorów nośnych

one-vs-one - taktyka ta zakłada wyuczenie klasyfikatora na parę klas. Traktuje on próbki jednej klasy jako pozytywne (+), zaś próbki innej klasy jako negatywne (-). W tym algorytmie posiadając k klas musimy wytrenować $k(k-1)/2$ maszyn wektorów nośnych.

Pomimo iż pierwsza taktyka wydaje się lepsza, bo wymaga wytrenowania mniejszej liczby maszyn wektorów nośnych, okazuje się, że w przypadku klasyfikacji znaków drogowych zastosowanie taktyki one-vs-all jest mniej efektywne czasowo oraz ma mniejszą skuteczność niż zastosowanie taktyki one-vs-one.

Na rysunku 3.6 pokazano przykładową separację 4 klas na podstawie dwóch cech oraz zliczono liczbę maszyn wektorów nośnych dla obu taktyk nauki. Linią ciągłą zaznaczono SVMy wyuczone za pomocą taktyki one-vs-all, zaś liniami przerywanymi zaznaczono SVMy wyuczone przy pomocy taktyki one-vs-one.[1]



Rysunek 3.6 prezentacja separacji klas dla taktyki one-vs-one oraz one-vs-all wraz ze zliczoną liczką k maszyn wektorów nośnych

Rozdział 4

Aplikacja

Elementem spinającym robota mobilnego z jego sterownikiem oraz wyuczonym algorytmem rozpoznawania i klasyfikowania znaków drogowych jest aplikacja komputerowa pisana w języku python. Dzięki wieloplatformowości Linuxa możliwe było pisanie i testowanie aplikacji zarówno na płytce Raspberry jak i na komputerze personalnym, który posiadał większą moc obliczeniową. Aplikacja w obecnej wersji umożliwia start robota oraz podgląd widoku robota wraz z zaznaczonymi znakami drogowymi. W poniższym rozdziale opisano niezbędne narzędzia do działania aplikacji, połączenia jej z robotem oraz jej obsługę. W kolejnym rozdziale zaprezentowano testy aplikacji.

4.1 Narzędzia

Aplikacja do uruchomienia się wymaga jedynie systemu Linux Raspbian z językiem Python przynajmniej w wersji 3.6, jednak do jej poprawnego działania niezbędne jest zaopatrzenie się w odpowiednie biblioteki.

OpenCv wersja 2

Numpy przynajmniej w wersji 1.14

Matplotlib przynajmniej w wersji 2

4.1.1 Raspbian

Raspbian jest systemem operacyjnym opartym na Debianie dla Raspberry Pi. Istnieje kilka wersji Raspbian, w tym Raspbian Stretch i Raspbian Jessie. Od 2015 roku został oficjalnie udostępniony przez Fundację Raspberry Pi jako podstawowy system operacyjny dla rodziny komputerów jednoplątkowych Raspberry Pi. Raspbian został stworzony przez Mike'a Thompsona i Petera Greena jako niezależny projekt. Początkowa komplikacja została ukończona w czerwcu 2012 r. System operacyjny jest nadal aktywnie rozwijany. Raspbian jest wysoce zoptymalizowany pod kątem wydajnych procesorów ARM Raspberry Pi.

W naszym projekcie używany jest system Raspbian Stretch aktualnie w wersji 4.14. Zdecydowano się na ten system, ponieważ praca w nim jest bardzo podobna do pracy na systemie Ubuntu, co pozwoliło na jednoczesne rozwijanie aplikacji zarówno na płytce Raspberry jak i na komputerze osobistym.

4.1.2 OpenCV

OpenCV (Open Source Computer Vision Library) wydawana na licencji BSD. Jest darmową biblioteką zarówno do celów edukacyjnych i komercyjnych. Ma wbudowane interfejsy zarówno dla C++ jak i pythona oraz Javy i może być używana w systemach: Windows, Linux, Mac OS, iOS oraz Android. Twórcom przyświeca idea zmniejszania mocy obliczeniowej popularnych algorytmów oraz ich implementacji w aplikacjach czasu rzeczywistego. Napisana jest w językach C i C++ i może korzystać z wielu rdzeni procesora. [2]

Szacuje się, że OpenCV pobrano 14 milionów razy od momentu wypuszczenia jej pierwszej wersji w roku 2008. OpenCv oferuje szeroki i nieustannie rozrastający się wachlarz możliwości. Znalazła swoje zastosowanie w sztuce, przetwarzaniu obrazów, uczeniu maszynowym, czy w zawansowanej robotyce.

W tym projekcie OpenCv używane jest do przetwarzania obrazu oraz do implementacji maszyn wektorowych nośnych.

4.1.3 Numpy

Numpy jest biblioteką dla języka programowania Python, dodająca obsługę dużych, wielowymiarowych tablic i macierzy, wraz z dużą kolekcją wysokiej jakości funkcji matematycznych do pracy na tych tablicach. Przodek NumPy, Numeric, został pierwotnie stworzony przez Jima Hugunina z udziałem kilku innych programistów. W 2005 roku Travis Oliphant stworzył NumPy, które jest oprogramowaniem typu open-source i ma wielu współtwórców.

W naszym projekcie Numpy niezbędne jest do "zasobżernych" obliczeń na macierzach reprezentujących obrazy.

4.1.4 Matplotlib

Matplotlib to biblioteka "kreślarska" dla języka programowania Python i jego numerycznego rozszerzenia matematycznego NumPy. Zapewnia obiektowy interfejs API do osadzania przebiegów funkcji w aplikacjach przy użyciu uniwersalnych zestawów narzędzi GUI, takich jak Tkinter, wxPython, Qt lub GTK +.

Matplotlib został pierwotnie napisany przez Johna D. Huntera, ma aktywną społeczność programistyczną i jest dystrybuowany na licencji typu BSD.

W tym projekcie biblioteka okazała się nieocenioną pomocą w czasie testów aplikacji, kiedy potrzebne było wizualizowanie zebranych danych.

4.2 Wymagania aplikacji

Aplikacja do swojego działania potrzebuje zarówno elementów sprzętowych jak i programistycznych.

Niezbędne jest zaopatrzenie się w komponenty składowe robota opisane w rozdziale 2.1 "Komponenty" oraz kluczowe jest ich poprawne połączenie. Aplikacja zadziała poprawnie jeśli dokonamy połączeń zgodnie ze schematem zawartym w dodatku A "Pełne połączenie komponentów robota".

Elementami oprogramowania niezbędnymi do zadziałania aplikacji zostały zawarte w rozdziale 4.1 "Narzędzia".

4.3 Budowa aplikacji

Finalna aplikacja daje możliwość pracy w dwóch trybach:

nauki - aplikacja trenuje maszyny wektorów nośnych na podstawie odpowiednio przygotowanej bazy zdjęć znaków drogowych następnie testuje je oraz prezentuje nam ich skuteczność.

testu - w tym trybie aplikacja pozwala na korzystanie z kamery i na testowanie maszyn wektorów nośnych w czasie rzeczywistym. Jest to finalny etap projektu. W trybie tym niewskazane jest korzystanie z jakichkolwiek innych aplikacji, aby w pełni przeznaczyć moc obliczeniową na cele przetwarzania i klasyfikacje znaków drogowych

Aplikacja składa się z kilku modułów:

1. main - pozwala na spieczenie modułów aplikacji w sprawnie działającą całość. Pozwala też, na wybór jednego z dwóch trybów pracy aplikacji (nauka/test)
2. dataset - skrypt pozwalający na parsowanie zdjęć znaków drogowych do głównego programu. Jego główna funkcja nosi nazwę load data() i dzieli ona zestaw znaków drogowych zawarty w bazie na paczki do nauki i do testów, jednocześnie wczytując z odpowiedniego pliku z bazy odpowiednie wartości ROI i etykiet dla danego zdjęcia. W pliku tym zaimplementowano również odpowiednie obrabianie obrazu.
3. classifier - skrypt służący do implementacji maszyn wektorów nośnych. Następnie trenuje maszyny wektorów nośnych wykorzystując odpowiednio obrobione zdjęcia. Nauka może zostać wykonana przy pomocy jednego z algorytmów opisanych w rozdziale 3.2 "SVM". W ostatnim etapie testuje działanie wyuczonych maszyn wektorów nośnych i wizualizuje nam wyniki działania.
4. newlinefollower - skrypt niezbędny do sterowania robotem oraz poprawnej obsługi czujników. Odpowiada za utrzymanie robota w torze jazdy oraz za omijanie napotykanych przeszkód. Skrypt ten dokładniej omówiono w rozdziale 2.3 "Testy silników oraz czujników"
5. adapt - skrypt, w którym zdefiniowano odpowiednie reakcje robota do napotkanego znaku drogowego. Reakcje te zostały zebrane w dodatku B "Adaptacje robota do znaków"
6. gui - skrypt, w którym zdefiniowano wygląd interfejsu graficznego.

4.3.1 Baza zdjęć znaków drogowych i ich preprocessing

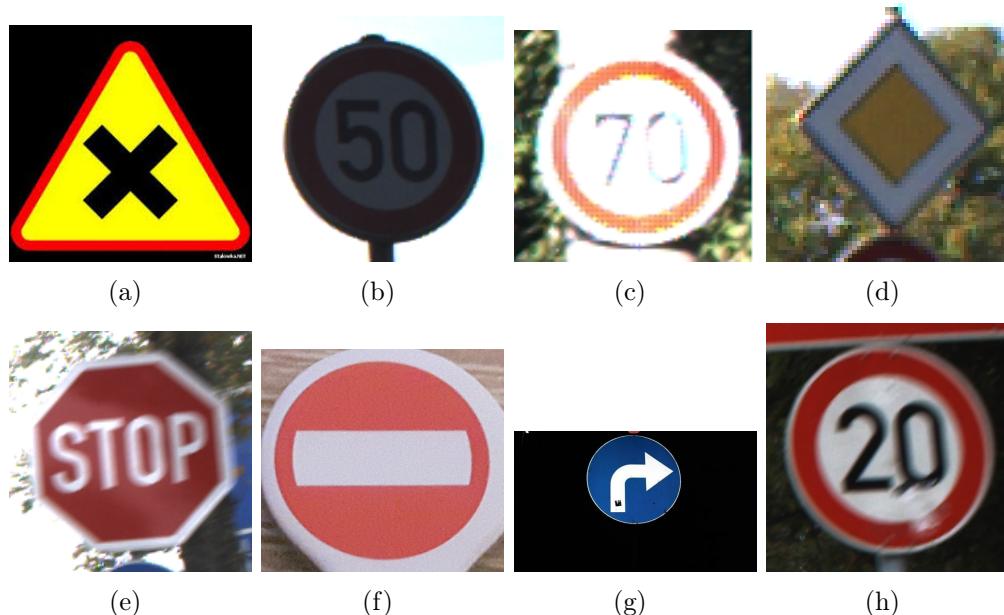
Kluczowym elementem każdego algorytmu wykorzystywanego do klasyfikacji obiektów do odpowiedniej z klas jest racjonalnie duża baza przykładów.

Na cele tego projektu skorzystano z bazy znaków drogowych ogólnodostępnej o nazwie "German Traffic Signs Recognition Benchmark", którą uzupełniono dodatkowymi przykładami zdjęć własnych oraz znalezionych w internecie. Dodatkowym zabiegiem zwiększającym liczbę zdjęć było edytowanie zdjęć pod kątem rozmycia, kontrastu oraz orientacji znaku. Do tego celu posłużyły gotowe komendy z OpenCv:

```
opencv_createsamples -img stop.jpg -bg bg.txt -info info/info.lst
-pngoutput info -maxxangle 0.5 -maxyangle 0.5 -maxzangle 0.5 -num 150
```

Komenda ta tworzy 150 dodatkowych obrazów umieszczających obraz stop.jpg na obrazach zawartych w folderze bg sterując jednocześnie nachyleniem obrazu stop.jpg na osiach x,y,z jednocześnie zapisując informacje o położeniu oryginalnego obrazu w pliku info.txt.

Ostatecznie baza została podzielona na 15 klas znaków drogowych. W każdym pliku znajduje się liczba zdjęć nie mniejsza niż 5000. Przykładowe zdjęcia z bazy pokazano na rysunku 4.1.



Rysunek 4.1 Przykłady znaków drogowych z bazy

Jako, że robot nie będzie miał do czynienia ze znakami rzeczywistych rozmiarów a jedynie z ich modelami do bazy dodano zdjęcia makiet znaków, które robot ma rozpoznać. Przykład takiego znaku umieszczono na rysunku 4.1 (f). Szczęśliwie modele znaków miały dokoła białą obramówkę, co pozwalało na łatwiejsze znalezienie znaku w obrazie.

Każda z klas zawierała plik info w formacie csv, w którym zawarto poszczególne informacje. W tabeli 4.1 pokazano przykładowy rekord z pliku info.csv

Tabela 4.1 Wyciąg z pliku info.csv

Filename	Width	Height	Roi.X1	Roi.Y1	Roi.X2	Roi.Y2	classId
00001.ppm	26	26	6	6	21	21	6
01004.ppm	33	33	12	14	25	28	10
05007.ppm	64	64	38	30	59	54	9
07009.ppm	32	32	4	4	30	30	11



Rysunek 4.2 Przykład obrazu z zaznaczonym obszarem zainteresowania

Mając już dość rozbudowaną bazę zdjęć znaków, która została już podzielona na poszczególne klasy oraz każde zdjęcie posiada zaznaczony obszar zainteresowania można przystąpić do wczesnej obróbki zdjęć, aby umożliwić na ich przykładach naukę maszynom wektorów nośnych.

Preprocessing jest wykonywany również za pomocą OpenCv oraz numpy. W celu przeskalowania zdjęć do odpowiedniego rozmiaru użyto komendy:

```
picture = [cv2.resize(picture, (32,32))]
```

Aby przejść do skali czarno-białej posłużyono się poleceniem:

```
Picture=[cv2.cvtColor(picture, cv2.COLOR_BGR2GRAY)]
```

Aby znormalizować jasności pikseli posłużyono się komendą:

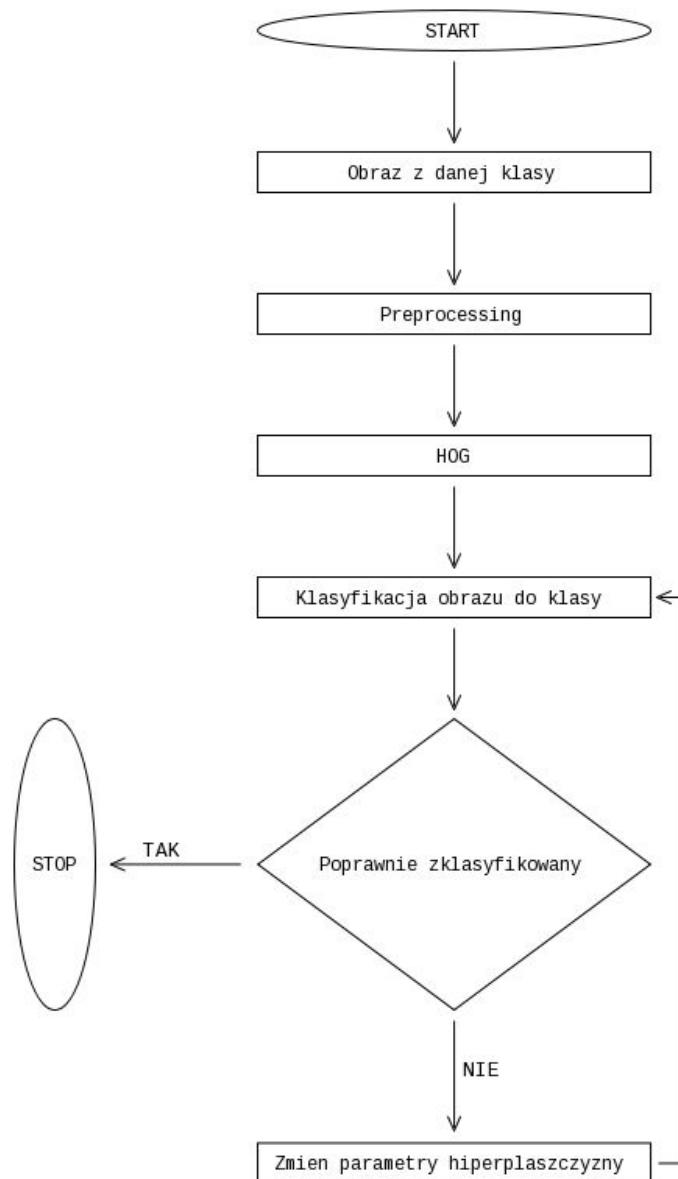
```
Picture=np.array(Picture).astype(np.float32)/255
```

4.3.2 Uczenie maszyn wektorów nośnych

Mając odpowiednio przygotowaną bazę zdjęć możemy przystąpić do implementacji algorytmu klasyfikacji znaków drogowych. W tym celu korzystamy z techniki nauki nazywanej nauką z nauczycielem. Polega ona na podawaniu na wejściu obrazów przypisanych do znanej klasy. Następnie algorytm stara się je klasyfikować do odpowiednich klas zmieniając współczynniki hiperpłaszczyzny odzielającej dane klasy (rozdział 3.2 "SVM").

Aby praca szła szybciej po procesie preprocessingu obraz podawany jest na wejście algorytmu w postaci Histogramu Gradientów Zorientowanych (rozdział 3.1 "HOG"). Zaleca się taki zabieg przy klasyfikowaniu znaków drogowych, ponieważ HOG bardzo skutecznie uwydatnia krawędzie na obrazie, co w przypadku znaków drogowych jest bardzo dużym ułatwieniem w ich identyfikacji a następnie klasyfikacji.

Na rysunku 4.3 pokazano bardzo uproszczony schemat blokowy nauki algorytmu.



Rysunek 4.3 Uproszczony schemat blokowy wędrówki obrazu w procesie klasyfikacji

4.3.3 Testy maszyn wektorów nośnych

Testowanie maszyny jest procesem podobnym do nauki, tyle, że nie mamy sprzężenia zwrotnego, a jedynie zbieramy informacje o poprawnej bądź niepoprawnej klasyfikacji danego obiektu do danej klasy. Testy wykonuje się na obrazach, które nie posłużyły wcześniej do nauki algorytmu. Dalej Algorytm będzie zapamiętywał swoje poprawne klasyfikacje oraz te błędne. Do testów wydzielono specjalnie co piąty z obrazów znajdujących się w bazie.

Znając wyniki testów algorytmu możemy stworzyć macierz Pomyłek (ang. Confusion Matrix). Macierz ta jest macierzą 2D o wymiarach kxk, gdzie k to liczba klas. Wiersze macierzy odnoszą się do przewidywanych klas, zaś kolumny odnoszą się do faktycznych klas danych obiektów. Element [w,k] macierzy zawiera liczbę próbek które przewidywano, że posiadają klasę w, ale w rzeczywistości miały klasę k. Celem dobrego klasyfikatora jest sprawienie, że macierz pomyłek będzie diagonalna, co poświadczyc, że przewidywana klasa pokrywa się z rzeczywistą klasą dla danego obiektu. Poniżej przedstawiono macierz pomyłek przy nauce i testach taktyką one-vs-one po podaniu obrazu w postaci Histogramu gradientów zorientowanych.

$$\begin{bmatrix} 505 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 325 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 253 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 318 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 545 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 441 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 345 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 654 \end{bmatrix}$$

Macierz Pomyłek pozwoli nam na policzenie kolejnych współczynników niezbędnych do stwierdzenia, czy nasz algorytm jest dobrym klasyfikatorem.

1. Dokładność (ang. Accuracy). Miara ta liczy jedynie liczbę próbek testowych rozpoznanych poprawnie i zwraca wynik dzielenia ich przec całkowitą liczbę próbek testowych. W ten sposób korzystając z macierzy Pomyłek tworzymy wektor dokładności, który w tym przypadku wynosi

$$\begin{bmatrix} [0.998 \ 0.993 \ 0.992 \ 0.997 \ 0.996 \ 0.995 \ 0.997 \ 0.994] \end{bmatrix}$$

Często wektor ten się uśrednia. W tym przypadku możemy powiedzieć, że precyzja algorytmu wyniosła 0.9953, czyli 99.53%. Jest to doskonały wynik.

2. Precyzja (ang. Precision) to proporcja przypadków faktycznie pozytywnych wśród wszystkich zaklasyfikowanych przez klasyfikator jako pozytywne. Wektor taki prezentuje się następująco.

$$\begin{bmatrix} [0.998 \ 0.993 \ 0.992 \ 0.997 \ 0.996 \ 0.995 \ 0.997 \ 0.994] \end{bmatrix}$$

Możemy więc mówić o precyzji algorytmu na poziomie 0.9953, czyli 99.53% to również bardzo dobry wynik.

3. Pełność (ang, recall) to proporcja przypadków faktycznie pozytywnych i zaklasyfikowanych przez system jako pozytywne wśród wszystkich faktycznie pozytywnych.

$$\begin{bmatrix} [0.998 \ 0.991 \ 0.992 \ 0.994 \ 0.998 \ 0.993 \ 0.997 \ 0.996] \end{bmatrix}$$

Zatem Pełność tego algorytmu wynosi 0.9949, czyli 99.49% Wynik ten również jest doskonały i gwarantuje poprawną klasyfikację znaku aż w 994 przypadkach na 1000.

Wyniki testów są zdumiewająco dobre, a wynika to z tego, że klasy w macierzy powyżej dobrano tak, aby były łatwo separowalne. Wśród 8 klas testowych znajdowały się znaki:

STOP

Ustęp pierszeństwa

Droga z pierwszeństwem przejazdu

Nakaz skrętu w prawo

ograniczenie prędkości do 50 km/h,

Zakaz wjazdu

Autostrada

Ostry zakręt w lewo

Łatwo zauważać, że znaki te często różnią się kształtem, co w przypadku używania Histogramu Gradientów Zorientowanych dodatkowo jest uwydawniane. Jedynie 3 znaki posiadały kształt tej samej figury geometrycznej, lecz znacznie różniły się wypełnieniem.

Przed zastosowaniem algorytmu HOG, przeprowadzono testy na obrazach czarno białych w taktyce one-vs-all. W tamtych testach współczynniki dokładności, precyzji i pełności wyniosły odpowiednio 89%, 78%, 72%. Wyniki te widzimy nie są aż tak zadawalające jak w przypadku zastosowania HOGu.

Do wizualizacji zebranych współczynników wspomagano się biblioteką matplotlib. W ostateczności testy aplikacji, przy zastosowaniu taktyki one-vs-one oraz poddaniu obrazów wcześniejszej odpowiedniej obróbce i podawwaniu ich na wejście algorytmu w postaci Histogramu Gradientów Zorientowanych utrzymywały się na poziomie wyższym niż 95%. Nie podaje się tutaj dokładnego wyniku, ponieważ ciągle dokładna się do bazy nowe rodzaje znaków drogowych. Obecnie algorytm poprawnie kwalifikuje 15 rodzajów znaków drogowych.

4.4 Połączenie aplikacji z robotem

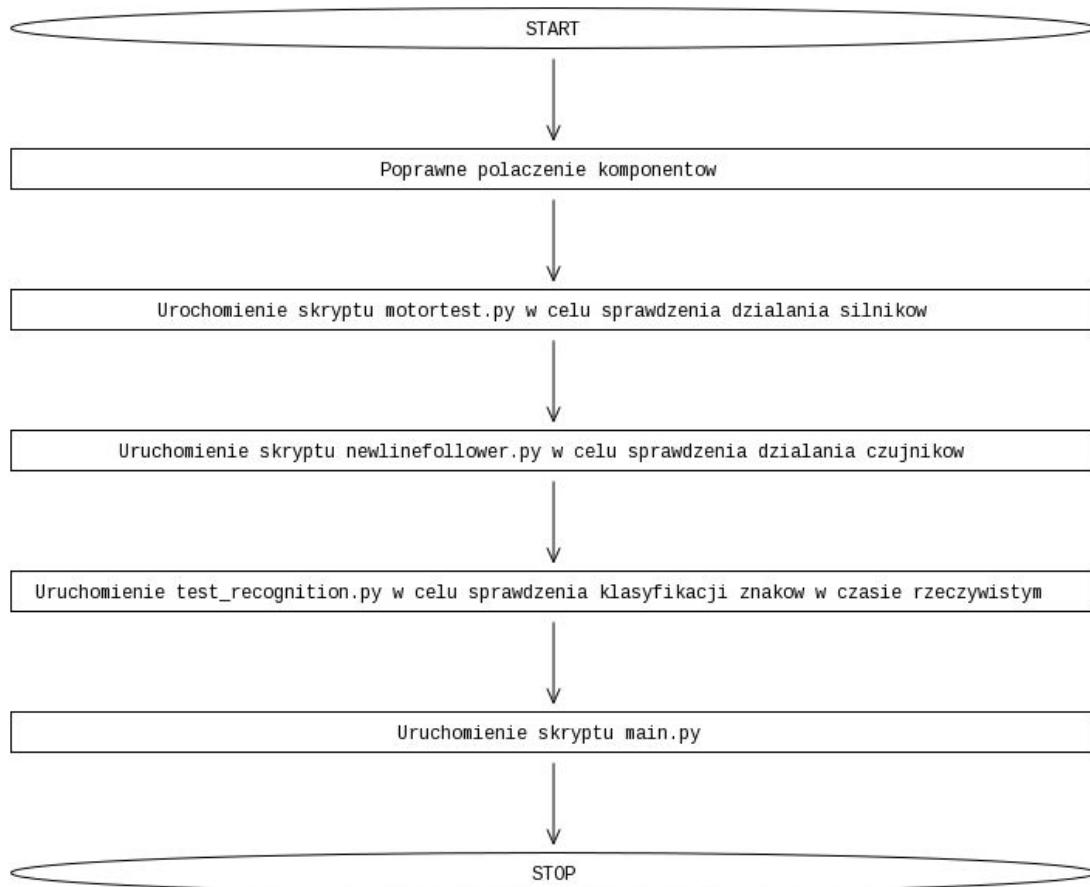
Przeniesienie aplikacji z fazy nauki na komputerze stacjonarnym do fazy testów na Raspberry odbyło się bez problemów, aczkolwiek było procesem czasochłonnym. Wystarczyło jedynie zadbać o doinstalowanie odpowiedniego oprogramowania na płytce (instalacja OpenCv zajęła około 7 godzin).

W obecnej wersji z płytą można połączyć się przez sieć Wi-fi, lecz wcześniej niezbędne jest zalogowanie się do tej sieci na płytce Raspberry Pi 3. Podgląd płytki umożliwia nam program VNC Viewer po zalogowaniu się. Oprócz podglądu program ten daje nam również możliwość sterowania Raspberry, co niweluje ilość potrzebnego sprzętu (myszka, klawiatura).

Aplikacja komunikuje się z robotem dzięki bibliotekom RPi.GPIO. Jest to biblioteka do sterowania pinami Raspberry pi 3 model B. Poszczególne piny płytka wraz z ich funkcjami omówiono w Dodatku C "Raspberry Pi Model B". Informacja co jest podłączone do poszczególnych pinów znajduje się w Dodatku A "Pełne podłączenie komponentów robota"

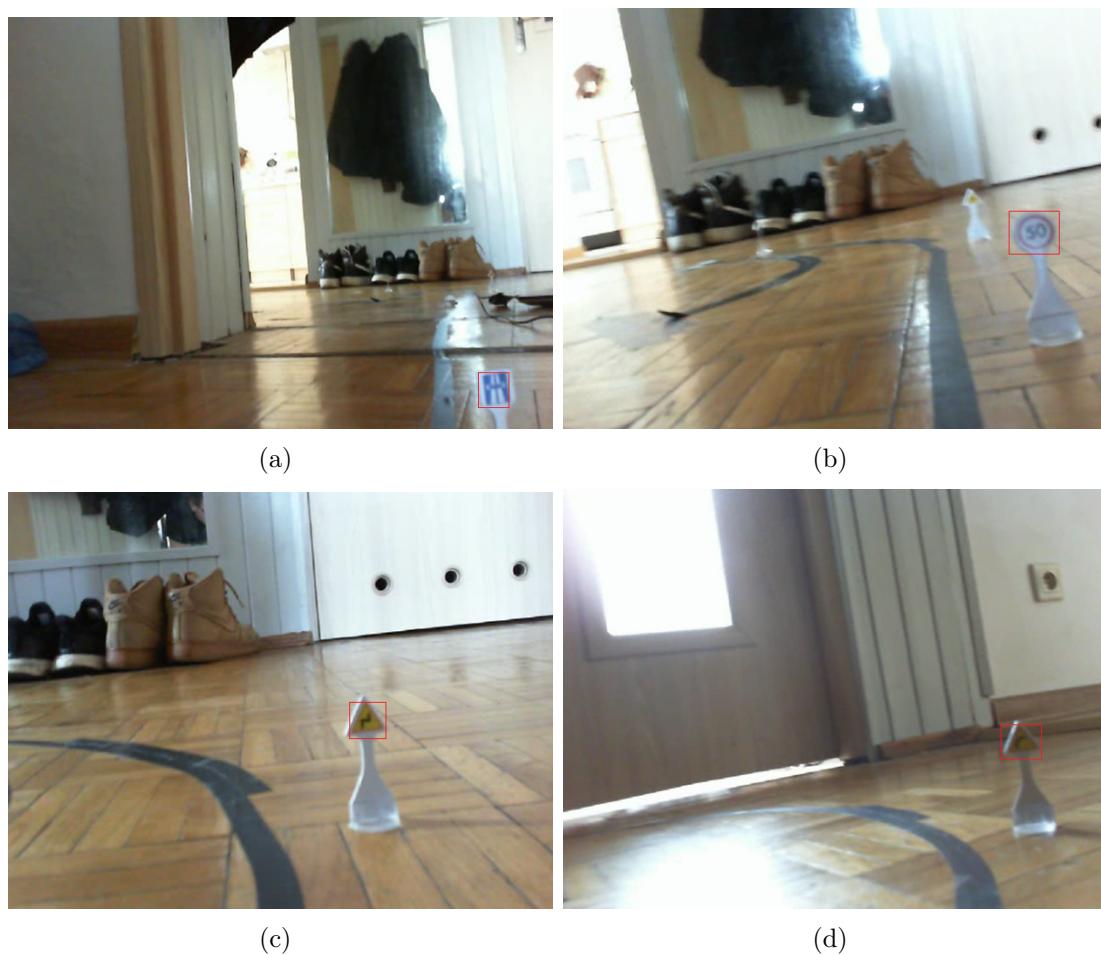
4.5 Obsługa aplikacji

Aplikacja jest bardzo intuicyjna w obsłudze. Wymaga jedynie uruchomienia kodu main.py w terminalu systemu Linux Raspbian, jednak najpierw zaleca się realizację wszystkich kroków z rysunku 4.4



Rysunek 4.4 Schemat blokowy uruchomienia aplikacji

Jeżeli aplikacja main.py zostanie otworzona to naszym oczom ukaże się panel wyboru opcji: nauka, test. Po wejściu w test zostaniemy zapytani o ty czy chcemy wyświetlać obraz z kamery, czy tylko dostawać informację o napotkanym znaku. Niestety zauważymy, że przy wyborze opcji podglądu kamery płynność obrazu jest mocno ograniczona. Średnio wynosi ona 2/3 klatek na sekundę. Dzieje się tak nawet przy zmniejszonej rozdzielczości obrazów płynących z kamery na wejście algorytmu testującego. Dlatego zaleca się zmniejszenie prędkości robota w pliku newlinefollower.py do około 25%. Dodatkowo zalecane jest wybieranie opcji jedynie wyświetlania informacji o napotykanym znaku. Ponadto komunikacja z robotem przez używany do tego program VNCviewer skutkuje tym, że na ekranie odbiornika obrazu ilość klatek na sekundę jeszcze spada. Poniżej, na zdjęciach pokazano kilka ujęć z kamery robota podczas testowej jazdy (rysunek 4.5) oraz zdjęcie z podglądu programu tekstowego (rysunek 4.6).



Rysunek 4.5 Ujęcia z programu głównego z włączoną opcją podglądu z kamery
 a)robot rozpoznał znak informujący o rozpoczęciu autostrady
 b)robot zauważa znak o ograniczeniu prędkości do 50%
 c) robot rozpoznaje znak o informującym o ostrych zakrętach
 d) robot napotyka znak informujący o ostrym zakręcie

A screenshot of a terminal window titled 'pi@raspberrypi: ~ /Desktop'. The window displays a log of the robot's movements:

```

pi@raspberrypi: ~ /Desktop
File Edit Tabs Help
7.34
386.48
STOP
SPINNING
forward move
7.72
325.37
STOP
SPINNING
forward move
7.32
272.17
STOP
SPINNING
forward move
7.4
323.69
STOP
SPINNING
forward move
7.79
325.02
STOP
  
```

Rysunek 4.6 Widok z aplikacji w trybie tekstowym. Robot wykrył przeszkodę i stara się ją ominąć.

Rozdział 5

Wnioski

Udało się spełnić najważniejsze założenia projektowe. Zbudowano robota mobilnego, który jest zdolny do poruszania się po liniach czarnych z dość dobrą dokładnością. Robot ma również możliwość zahamowania awaryjnie przy usłyszeniu klaśnięcia. Robot omija przeszkody zarówno te z przodu jak i te z tyłu. Udało się zaimplementować oraz wyuczyć algorytm klasyfikujący znaki drogowe w czasie rzeczywistym. Udało się również napisać skrypt adoptujący zachowanie robota do napotkanego znaku. Listę zachowań robota w odpowiedzi do napotkanego znaku pokazano w dodatku B "Adaptacje robota do znaku"

Ostateczna wersja aplikacji okazała się działać bardzo wolno. Prędkość przetwarzania obrazu wyniosła zaledwie 3 klatki na sekundę, choć zaglądając do literatury wynik jest całkiem niezły. Na tym polu wciąż pozostają rzeczy do optymalizacji.

Niestety nie rozwiązano problemu podglądu na żywo płytki Raspberry Pi z na przykład komputera personalnego. Początkowe rozwiązanie, czyli instalacja programu VNC Viewer okazała się wprowadzać dodatkowe opóźnienia w przekazie obrazu. Obecnie trwa praca nad implementacją algorytmu pozwalającego na bezpośredni podgląd danego okna otworzonego na Raspberry przez użytkownika komputera stacjonarnego.

Robot niestety wykrywa przeszkody w dość małym polu. Problem ten częściowo rozwiązano. Zamówiono dwa dodatkowe czujniki, które miałyby sprawdzać dodatkowo około 45 stopni pola widzenia z przodu pojazdu. Niestety zamówione czujniki nie zdążyły do trzeć na czas.

Dodatkowo robot omija przeszkody w dość ignorancki sposób. Robot natrafiając na przeszkodę z przodu jedynie hamuje i zwraca po wycofaniu się na pewną odległość. Problem, że omijanie przeszkody realizowane jest przez sterowanie silnikami czasowo, czyli zamiast robot cofać aż do napotkania linii, ten po prostu ma zaprogramowane cofanie przez czas równy jednej sekundzie. Postanowiono wprowadzić kontrolę obrotów silników przy pomocy enkoderów.

Przez dość sporą masę robota zyskał on sporą jak na swoje rozmiary bezwładność, co skutkuje od czasu do czasu wypadaniem na zakrętach albo niewyhamowaniem w linii stop. Problem ten udało się częściowo rozwiązać przez pogrubienie linii na ostrzych zakrętach oraz na skrzyżowaniach jak i na końcach trasy. Pomocny okazał się algorytm wykonywania ostrych skrętów, wtedy jedno z kół (wewnętrzne) zaczynało się kręcić z małą prędkością przeciwnie do kierunku jazdy robota, co umożliwiło robotowi zakręty nawet o 90 stopni. Rozwiązanie to jednak nie odzwierciedla sposobu poruszania się typowego pojazdu autonomicznego.

Brakuje robotowi algorytmu rozpoznającego zgubienie linii. W czasie pracy nad sterowaniem nad robotem próbowało kilku rozwiązań, jednak żadne nie przynosiło owocnych rezultatów. Pracowano nad algorytmem angażującym pracę trzech czujników przerwania

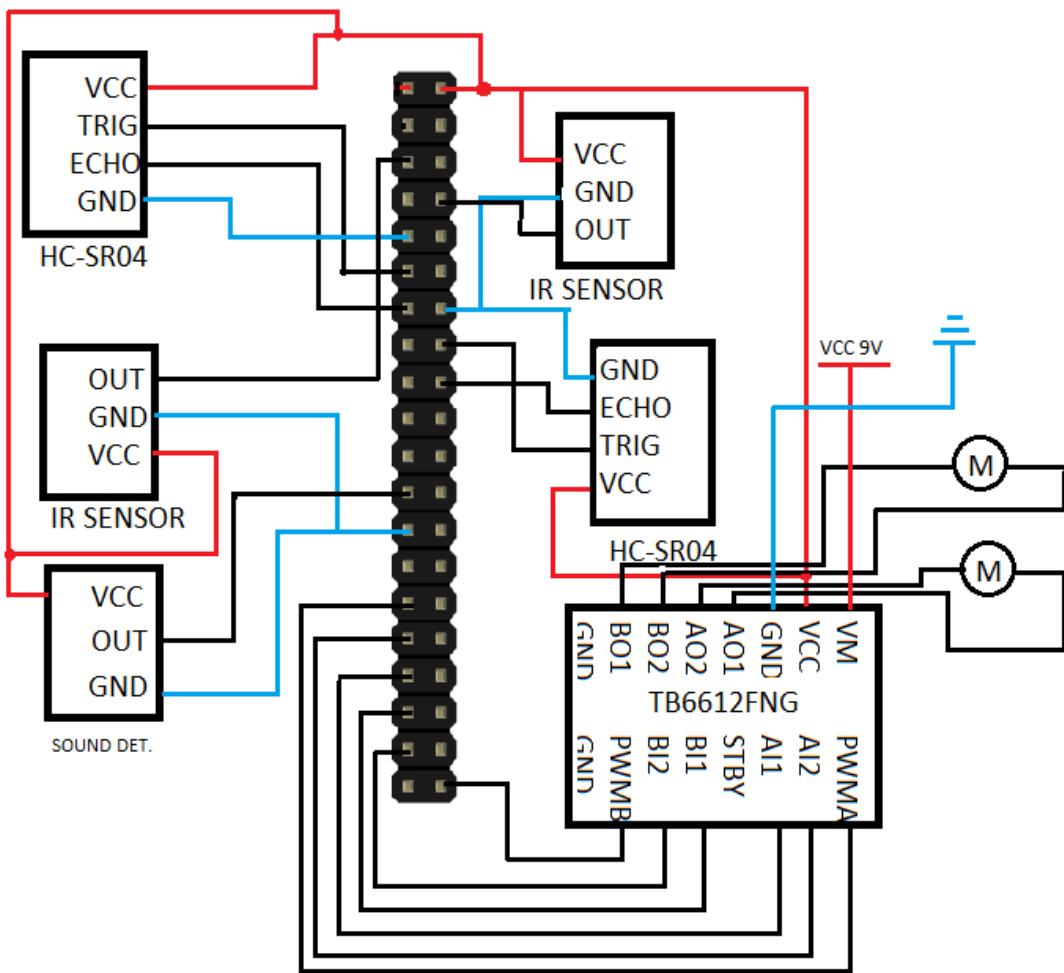
wiązki, jednak wtedy robot musi poruszać się na linii, co oznacza, że linie toru musiałby stać się czterokrotnie grubsze. Obecnie implementowany jest algorytm o charakterze hybrydowym. Algorytm będzie dalej badał dwoma skrajnymi czujnikami wystąpienie najechania na linię czarną, jednak pośrodku znajduje się dodatkowy czujnik, który będzie dość często poruszał się nad linią czarną. Takie rozwiązanie pozwoli wykryć długotrwale zgubienie linii, w przypadku gdy żaden z czujników w dłuższym czasie nie natrafi na żadną czarną linię.

Kolejnym problemem jest rozpoznawanie skrzyżowań. Pojazd miał symulować warunki drogowe, a jak wiadomo nie przed każdym skrzyżowaniem jest znak drogowy informujący o przebiegu dróg na danym skrzyżowaniu. Brakuje robotowi możliwości wychwytywania skrzyżowań równoległych. W czasie prac nad projektem tego problemu nie udało się rozwiązać, zastosowano jedynie algorytm upraszczający, czyli napotykając skrzyżowanie z odpowiednim oznakowaniem robot poruszał się po głównej drodze, albo w przypadku dróg równorzędnych losował skręt w lewo, skręt w prawo albo jazdę na wprost.

Dalszym planem rozwoju projektu jest zbudowanie kolejnego robota lub robotów, możliwe że na innych platformach niż Raspberry Pi 3 Model B, ale z identycznymi funkcjonalnościami. Roboty będą miały poruszać się niezależnie, komunikując się ze sobą tworząc sieć rozproszoną. Będą przekazywały sobie informacje o swoich wspólnych położeniach, aby unikać kolizji, czy korków. Pozytywne byłoby też poszerzenie bazy możliwych znaków do klasyfikacji.

Dodatek A

Pełne podłączenie komponentów robota



Rysunek A.1 Pełne podłączenie komponentów elektronicznych robota

W centralnym punkcie schematu widzimy piny Raspberry pi 3 model B. Poszczególne nazwy pinów zawarto w dodatku C. Poniżej.

Niezbędne jest zagwarantowanie napięcia na pinach VCC wszystkich komponentów, w tym celu zalecane jest wykorzystanie płytki uniwersalnej.

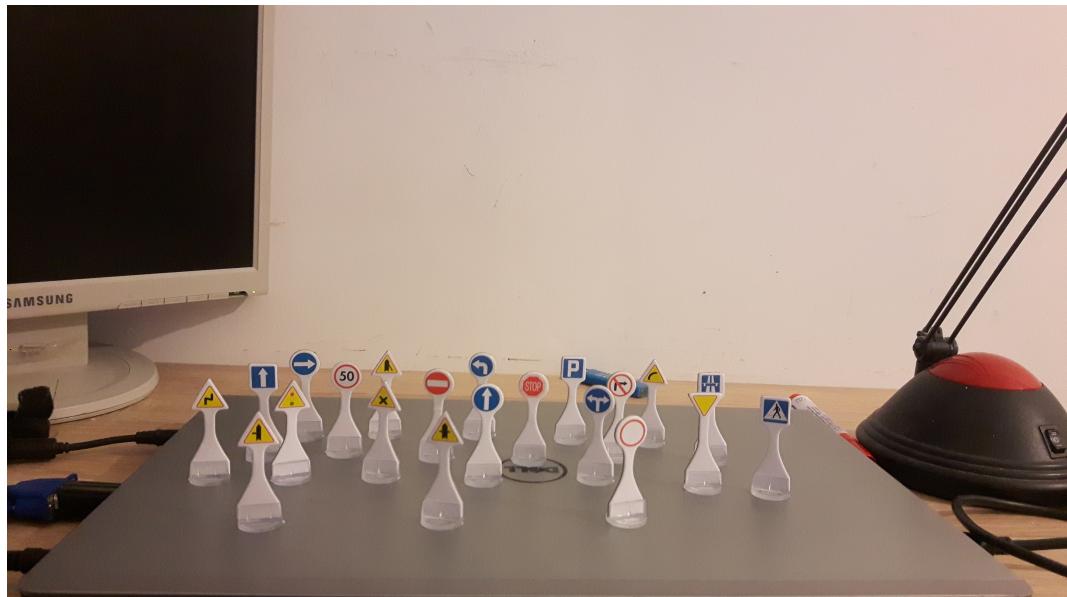
Na schemacie nie zawarto dzielnika napięcia niezbędnego do szalania czujników odległości hc-sr04.

Dodatek B

Adaptacje robota do znaków

W dodatku B zawarto spis wszystkich funkcji przypisanych do danych znaków drogowych. Aktualnie robot poprawnie klasyfikuje 15 klas znaków drogowych. W tabeli B.1 zebrano te funkcje wraz z nazwami znaków, do których są przypisane.

Na obrazku poniżej pokazano zdjęcie modeli znaków drogowych użytych do nawigacji robota. Każdy znak występuje tylko raz, więc niezbędne jest przemieszczanie znaków po torze jazdy, aby uatrakcyjnić przejazd robota.



Rysunek B.1 Modele znaków drogowych stawiane przy torze jazdy

Tabela B.1 Wyciąg z pliku info.csv

STOP	Robot ma zatrzymać się na określony czas (domyślnie 2 sekundy)
Ustęp pierwszeństwa	Robot podjeżdżając do skrzyżowania będzie czekać na wykrycie linii i następnie zatrzyma się na określony czas (domyślnie 2 sekundy)
Droga z pierwszeństwem przejazdu	Robot zwolni do pewnej wartości (domyślnie 50% nominalnej wartości na określony czas (domyślnie 1 sekunda)
Nakaz skrętu w prawo	Robot czeka na natrafienie na czarną linię oznaczającą skrzyżowanie się dróg i następnie zignoruje odczyty z lewego sensora, co umożliwia wykonanie skrętu w prawo.
Nakaz skrętu w lewo	Robot czeka na natrafienie na czarną linię oznaczającą skrzyżowanie się dróg i następnie zignoruje odczyty z prawego sensora, co umożliwia wykonanie skrętu w lewo.
Nakaz jazdy na wprost	Robot czeka na natrafienie na czarną linię oznaczającą skrzyżowanie się dróg i następnie zignoruje odczyty z lewego oraz z prawego sensora na krótką chwilę (domyślnie 3 sekundy), co umożliwia przejazd przez skrzyżowanie na wprost.
Ograniczenie prędkości do 50 km/h	Robot zwolni do 50% swojej nominalnej wartości prędkości do czasu natrafienia na inny znak
Ograniczenie prędkości do 70 km/h	Robot zwolni do 50% swojej nominalnej wartości prędkości do czasu natrafienia na inny znak
Zakaz wjazdu	Widząc ten znak robot czeka na natrafienie na czarną linię symbolizującą natrafienie na skrzyżowanie i następnie pokieruje się w lewo (wykorzysta funkcję znaku nakazu skrętu w lewo)
Zakaz skrętu w lewo	Widząc ten znak robot czeka na natrafienie na czarną linię symbolizującą natrafienie na skrzyżowanie i następnie pokieruje się naw prost (wykorzysta funkcję znaku nakazu jazdy na wprost)
Zakaz skrętu w prawo	Widząc ten znak robot czeka na natrafienie na czarną linię symbolizującą natrafienie na skrzyżowanie i następnie pokieruje się naw prost (wykorzysta funkcję znaku nakazu jazdy na wprost)
Początek autostrady	Robot rozpędzi się do 100% swojej maksymalnej wartości i utrzyma tę prędkość aż do natrafienia na inny znak drogowy
Ostry zakręt w prawo	Robot zredukuje swoją nominalną prędkość do pewnej wartości (domyślnie 30%) i utrzyma ją przez określony czas albo do czasu natrafienia na inny znak drogowy.
Ostre zakręty	Robot zredukuje swoją nominalną prędkość do pewnej wartości (domyślnie 10%) i utrzyma ją przez określony czas albo do czasu natrafienia na inny znak drogowy. Zaleca się ustawianie tego znaku przed zakrętami równymi albo większymi niż 90 stopni
Skrzyżowanie równożądne	Robot podjeżdżając do skrzyżowania będzie czekać na wykrycie linii i następnie zatrzyma się na określony czas (domyślnie 2 sekundy) i następnie wylosuje jedną z 3 możliwości (nakaz jazdy na wprost, nakaz skrętu w lewo, nakaz skrętu w prawo)

Dodatek C

Raspberry Pi 3 Model B

Raspberry Pi 3 GPIO Header			
Pin#	NAME	NAME	Pin#
01	3.3v DC Power	DC Power 5v	02
03	GPIO02 (SDA1 , I ² C)	DC Power 5v	04
05	GPIO03 (SCL1 , I ² C)	Ground	06
07	GPIO04 (GPIO_GCLK)	(TXD0) GPIO14	08
09	Ground	(RXD0) GPIO15	10
11	GPIO17 (GPIO_GEN0)	(GPIO_GEN1) GPIO18	12
13	GPIO27 (GPIO_GEN2)	Ground	14
15	GPIO22 (GPIO_GEN3)	(GPIO_GEN4) GPIO23	16
17	3.3v DC Power	(GPIO_GEN5) GPIO24	18
19	GPIO10 (SPI_MOSI)	Ground	20
21	GPIO09 (SPI_MISO)	(GPIO_GEN6) GPIO25	22
23	GPIO11 (SPI_CLK)	(SPI_CE0_N) GPIO08	24
25	Ground	(SPI_CE1_N) GPIO07	26
27	ID_SD (I ² C ID EEPROM)	(I ² C ID EEPROM) ID_SC	28
29	GPIO05	Ground	30
31	GPIO06	GPIO12	32
33	GPIO13	Ground	34
35	GPIO19	GPIO16	36
37	GPIO26	GPIO20	38
39	Ground	GPIO21	40

Rev. 2
29/02/2016

www.element14.com/RaspberryPi

Rysunek C.1 Podpisane piny płytka Raspberry Pi 3

Źródło: www.element14.com

Bibliografia

- [1] F. Chollet. *Deep Learning with Python*. Manning Publications Company, 2017.
- [2] J. H. et al. *OpenCV Blueprints*. PACT, 2015.
- [3] V. Z. et al. *Python Deep Learning*. PACKT, 2017.
- [4] A. N. Matthew Lamons, Rahul Kumar. *Python Deep Learning Projects*. PACKT, 2018.
- [5] S. Monk. *Raspberry Pi Cookbook, 2nd Edition*. O'Reilly Media, 2016.
- [6] S. Monk. *Electronics Cookbook*. O'Reilly Media, 2017.

Spis rysunków

2.1	Silnik prądu stałego 6V w obudowie	6
2.2	Koła pojazdu a)napędowe b)podporpwe	7
2.3	Podwozie robota mobilnego	7
2.4	Moduł do wykrywania linii	8
2.5	Zachowanie transmitema i odbiornika podczerwonego w przypadku a) wykrycia b) niewykrycia przeszkody	9
2.6	Napięcia na wejściu (Trigger), wyjściu transmitemera oraz na wyjściu modułu - na pinie ECHO	9
2.7	Dzielnik napięcia	10
2.8	Połączenie HC-SR04 z dzielnikiem	11
2.9	Moduł do wykrywania linii	11
2.10	Moduł do sterowania silnikami z podłączonymi silnikami	12
2.11	Robot z połączonymi komponentami	13
2.12	Schemat blokowy - wykrywanie linii	14
2.13	Adaptacja robota do napotkania na czarną linię a) Jazda na wprost b) prawa kontra c)lewa kontra d) STOP	15
2.14	Badanie odległości umiejscowienia przeszkody	16
2.15	Zachowanie robota po wykryciu przeszkody a) z przodu b) z tyłu	17
3.1	HOG dla fragmentu 8x8[px]	22
3.2	Prezentacja efektu algorytmu HOG na przykładzie znaku drogowego	23
3.3	Podział obiektów do 2 klas na podstawie cech x1 i x2	24
3.4	Regulacja parametru c w klasyfikacji	25
3.5	a) obiekty nieseparowalne prostą b) obiekty odseparowane płaszczyzną po zastosowaniu "Kernel Trick"	25
3.6	prezentacja separacji klas dla taktyki one-vs-one oraz one-vs-all wraz ze zliczoną liczką k maszyn wektorów nośnych	26
4.1	Przykłady znaków drogowych z bazy	30
4.2	Przykład obrazu z zaznaczonym obszarem zainteresowania	31
4.3	Uproszczony schemat blokowy wędrówki obrazu w procesie klasyfikacji	32
4.4	Schemat blokowy uruchomienia aplikacji	35
4.5	Ujęcia z programu głównego z włączoną opcją podglądu z kamery a)robot rozpoznał znak informujący o rozpoczęciu autostrady b)robot zauważa znak o ograniczeniu prędkości do 50% c) robot rozpoznaje znak o informującym o ostrych zakrętach d) robot napotyka znak informujący o ostrym zakręcie	36
4.6	Widok z aplikacji w trybie tekstowym. Robot wykrył przeszkodę i stara się ją ominąć.	36

A.1 Pełne podłączenie komponentów elektronicznych robota	39
B.1 Modele znaków drogowych stawiane przy torze jazdy	41
C.1 Podpisane piny płytka Raspberry Pi 3	43

Spis tabel

2.1	Zbiór najważniejszych parametrów silnika	6
2.2	Charakterystyka modułu z czujnikiem IR	8
2.3	Charakterystyka ultradźwiękowego czujnika odległości HC-SR04	10
2.4	Charakterystyka czujki dźwięku	11
2.5	Charakterystyka modułu TB6612FNG	12
2.6	Charakterystyka modułu TB6612FNG	12
2.7	Funkcje sterowania silnikami	18
2.8	Funkcje sterowania silnikami	19
4.1	Wyciąg z pliku info.csv	31
B.1	Wyciąg z pliku info.csv	42