Akademia Górniczo-Hutnicza
im. Stanisława Staszica w Krakowie

Wydział Inżynierii Mechanicznej i Robotyki

PRACA INŻYNIERSKA

Temat: Budowa i oprogramowanie robota typu line follower z detekcją przeszkód

Autor:	Bartosz Wojdon
Kierunek studiów:	Inżynieria mechatroniczna
Opiekun pracy:	dr inż. Mariusz Gibiec

Kraków, rok Akadem. 2024/2025

Spis treści

1.Wstęp

Wprowadzenie do technologii robotów mobilnych

Robotyka mobilna to interdyscyplinarna dziedzina inżynierii, która łączy elementy mechaniki, elektroniki, informatyki oraz sztucznej inteligencji, umożliwiając projektowanie i budowanie robotów zdolnych do autonomicznego poruszania się w zmieniających się, złożonych środowiskach. Współczesne roboty mobilne charakteryzują się dużą wszechstronnością oraz elastycznością. Znajdują szerokie zastosowanie w różnych branżach, takich jak przemysł[1] , medycyna[2] , rolnictwo[3] czy eksploracja kosmosu .

Główne cele robotyki mobilnej obejmują zwiększenie automatyzacji procesów, zapewnienie wysokiej efektywności oraz podniesienie poziomu bezpieczeństwa w wykonywaniu różnorodnych zadań. Szczególnie istotnym zagadnieniem jest zdolność robotów do autonomicznej nawigacji oraz omijania przeszkód. W tym celu wykorzystywane są zaawansowane technologie, takie jak algorytmy sterowania oraz systemy sensoryczne, które umożliwiają robotom samodzielne podejmowanie decyzji na podstawie danych pochodzących z otoczenia. Na przykład w pracy "Autonomous Mobile Robots in Industrial Applications" autorzy podkreślają, że nowoczesne roboty muszą wykorzystywać zaawansowane technologie, takie jak LIDAR, systemy wizyjne czy algorytmy sztucznej inteligencji, aby dostosować swoje trajektorie do zmieniających się warunków środowiskowych[4].

Jednym z najczęściej wykorzystywanych typów robotów mobilnych są roboty typu Line Follower. Ich podstawowym zadaniem jest podążanie za określoną linią, która jest zazwyczaj wyznaczona na podłożu, przy użyciu różnych technologii sensorycznych i algorytmów sterowania. Roboty te stały się popularne nie tylko w zastosowaniach edukacyjnych, ale również w badaniach naukowych, zastosowaniach przemysłowych, systemach magazynowych oraz transportu bliskiego, które skupiają się na rozwoju algorytmów sterowania oraz integracji systemów sensorycznych. W literaturze dostępne są liczne badania dotyczące tej tematyki, w tym prace poświęcone wykorzystywaniu czujników optycznych do śledzenia linii oraz implementacji zaawansowanych algorytmów nawigacyjnych. Zgodnie z opisem zawartym w pracy [5], autonomiczne łaziki, które potrafią nawigować w nieznanych środowiskach, mają kluczowe znaczenie w kontekście rozwoju technologii robotów mobilnych. Jednym z najistotniejszych wyzwań w nawigacji takich pojazdów jest efektywne znajdowanie ścieżki i omijanie przeszkód. Tradycyjnie, sterowanie tego typu robotami opiera się na dynamicznych paradygmatach sterowania, które umożliwiają podejmowanie lokalnych decyzji oparte na analizie otoczenia robota w czasie rzeczywistym.

Przykładem zastosowania takich technologii jest projektowanie robotów, które mogą uczyć się w procesie "wzmacniania" (Reinforcement Learning, RL). Uczenie przez wzmacnianie pozwala robotowi na adaptację do zmieniającego się otoczenia i na naukę poprzez interakcje z tym otoczeniem. W ramach tego podejścia, robot może samodzielnie decydować o najlepszych działaniach w celu osiągnięcia celu (np. dotarcie do określonego miejsca) lub unikania przeszkód, co zwiększa efektywność i bezpieczeństwo jego działania w zmiennych warunkach.

Na przykład, w pracy wyżej [5] opisano autonomicznego łazika, który nie tylko śledzi określoną linię, ale także posiada zdolność omijania przeszkód. Dzięki zastosowaniu technologii takich jak czujniki podczerwieni (IR), robot jest w stanie wykrywać zarówno ścieżkę, jak i przeszkody. Sterowanie robotem odbywa się za pomocą platformy Arduino Uno, co zapewnia prostotę i elastyczność w realizacji algorytmów.

W artykule przedstawiono również inne podejścia i badania, które koncentrują się na optymalizacji algorytmów sterowania i integracji systemów sensorycznych w robotach mobilnych. Wiele z tych badań stosuje zaawansowane techniki uczenia maszynowego, takie jak sieci neuronowe, uczenie przez wzmacnianie, czy rozmyte systemy sterowania, w celu osiągnięcia wyższej autonomii i precyzyjnej nawigacji w różnych warunkach.

1.1 Cel pracy

Głównym celem tej pracy jest stworzenie robota line follower z funkcją detekcji przeszkód. Robot ten, korzystać będzie z czujnika podczerwieni dzięki któremu, będzie mógł precyzyjnie śledzić linię oraz czujnika ultradźwiękowego aby reagować na przeszkody na swojej drodze, co symuluje podstawowe wyzwania związane z nawigacją autonomicznych robotów mobilnych.

1.2 Zakres i metodologia pracy

Zakres niniejszej pracy obejmuje etapy, które prowadzą od projektowania robota aż do jego pełnej realizacji i testowania. Metodologia projektu obejmuje kolejne kroki realizacyjne, które zostaną opisane poniżej.

1. **Analiza i dobór komponentów** – Przeprowadzenie analizy wymagań funkcjonalnych robota oraz dostępnych technologii. W ramach tego etapu nastąpi wybór komponentów elektronicznych i mechanicznych, takich jak mikrokontroler, czujniki, napędy oraz elementy konstrukcyjne.
2. **Projektowanie mechaniczne i dobór materiałów** – Przygotowanie konstrukcji mechanicznej robota przy użyciu oprogramowania CAD. W tej części zostaną również dobrane odpowiednie materiały do wydruku 3D, takie jak PLA lub ABS, które zapewnią trwałość i wytrzymałość konstrukcji.
3. **Wydruk 3D i montaż** – Wykonanie wydruku 3D poszczególnych elementów konstrukcyjnych robota oraz ich złożenie.
4. **Implementacja algorytmu sterowania** – Stworzenie i zaimplementowanie algorytmu, który będzie umożliwiał robotowi śledzenie linii i unikanie przeszkód. Algorytm zostanie wdrożony na mikrokontrolerze i będzie opierał się na danych z czujników.
5. **Integracja systemu** – Połączenie wszystkich elementów (hardware i software) w spójny system. Integracja obejmie montaż czujników, okablowanie oraz synchronizację pracy poszczególnych komponentów, takich jak mikrokontroler i serwomechanizmy.
6. **Testowanie i kalibracja** – Przeprowadzenie serii testów w różnych warunkach, aby zweryfikować poprawność działania robota. Testy będą obejmować sprawdzenie robota na wyznaczonej trasie, dostosowanie parametrów algorytmu oraz kalibrację czujników.
7. **Dokumentacja i analiza wyników** – Sporządzenie dokumentacji technicznej, opisującej proces projektowania i wyniki testów.

2.Przegląd istniejących rozwiązań

Sensoryka w Robotach Line Follower

W literaturze naukowej istnieje szerokie spektrum badań poświęconych robotom typu Line Follower. Jednym z kluczowych elementów tych robotów jest zastosowanie czujników optycznych, które umożliwiają precyzyjną detekcję linii na podłożu. Czujniki te działają na zasadzie wykrywania różnic w odbiciu światła, co pozwala na rozróżnienie między jasnymi a ciemnymi obszarami. W zależności od konstrukcji robotów, najczęściej wykorzystywane są fotorezystory, fotodiody oraz czujniki odbicia podczerwieni. Dzięki tym technologiom roboty są w stanie śledzić linię nawet w przypadku zakrętów lub na skomplikowanych trasach.

W pracy [6] zaprezentowano konstrukcję robota Line Follower, w której kluczowym elementem sensorycznym był zestaw czujników optycznych. W tym projekcie zastosowano czujniki odbicia podczerwieni, składające się z pary nadajnika i odbiornika podczerwieni. Dzięki temu czujniki te wykazują wysoką skuteczność w wykrywaniu powierzchni o różnych właściwościach odbicia światła, co jest niezbędne na trasach z czarnymi liniami na białym tle lub odwrotnie.

W zaprojektowanym robocie wykorzystano czujniki CNY70 – optyczne czujniki z wyjściem tranzystorowym. Działają one na zasadzie detekcji światła odbitego od powierzchni. Dzięki zastosowaniu takich czujników możliwe było precyzyjne monitorowanie linii, przy minimalnej odległości czujników od powierzchni gruntu, wynoszącej około 3 mm. W projekcie użyto ośmiu takich czujników, rozmieszczonych w odpowiednich odstępach, co pozwalało na precyzyjniejsze wykrywanie zmian w kierunku i reakcji robota na zakręty.

Sygnały analogowe z czujników były następnie konwertowane na wartości cyfrowe przy pomocy układu przetwornika analogowo-cyfrowego (ADC), który przekształcał je na cyfrowe wartości 0 i 1. Tak przetworzone dane były wykorzystywane przez mikrokontroler do podejmowania decyzji dotyczących ruchu robota, co pozwalało na precyzyjne sterowanie jego ruchem wzdłuż linii.

W pracy omówiono również szczegóły związane z układami elektronicznymi, w tym rolę przetworników ADC oraz wykorzystanie układów scalonych, takich jak LM324, do konwersji sygnałów z czujników. Tego typu rozwiązanie umożliwiało łatwą implementację systemu detekcji linii w robocie.

Cały system sensoryczny stanowił część większego układu robota, który potrafił autonomicznie śledzić linię, reagować na zakręty i utrzymywać stabilność na trasie. Takie roboty znajdują szerokie zastosowanie w edukacji, konkursach robotycznych oraz w różnych

aplikacjach przemysłowych i użytkowych, takich jak systemy transportowe czy urządzenia wspomagające osoby niewidome.

Dzięki swojej prostocie, efektywności oraz niskiemu kosztowi implementacji, czujniki optyczne stały się powszechnie stosowane w robotach Line Follower. Te zalety sprawiają, że są one szczególnie atrakcyjne w kontekście edukacyjnym, umożliwiając studentom naukę podstaw robotyki i programowania. Jednym z popularnych projektów edukacyjnych jest budowa robotów, które poruszają się po trasach o różnych rodzajach linii, np. prostych, zakrzywionych, a także złożonych trasach w kształcie liter czy symboli.

Algorytmy PID w Robotach Line Follower

Algorytm PID (Proporcjonalno- Całkująco- Różniczkujący) jest jednym z najczęściej stosowanych w robotach typu Line Follower, gdzie jego głównym zadaniem jest minimalizacja błędu między aktualną a oczekiwaną pozycją robota na śledzonej linii. Składa się on z trzech członów: proporcjonalnego, całkującego i różniczkującego, które współpracują, by zapewnić jak najbardziej precyzyjne sterowanie ruchem robota. Każdy z członów algorytmu ma swoje zadanie: człon proporcjonalny odpowiada za zmniejszenie błędu w danym momencie, całkujący koryguje błędy sumujące się w czasie, a różniczkujący pomaga przewidywać przyszłe zmiany błędu, co zwiększa stabilność robota.

W pracy [7] przedstawiono implementację i strojenie regulatora PID do sterowania pozycją robota mobilnego z napędem różnicowym, zaprojektowanego z myślą o uczestnictwie w europejskich amatorskich zawodach robotycznych. Robot został opracowany w ramach projektu na Wydziale Inżynierii Uniwersytetu w Sybii, w Katedrze Maszyn i Urządzeń Przemysłowych. Jego głównym celem było precyzyjne sterowanie jego ruchem na podstawie danych z odometrii oraz wykorzystanie regulatora PID do kontroli silników bezszczotkowych DC.

Podstawą nawigacji robota było zastosowanie systemu odometrii, który pozwalał określić aktualną pozycję robota w przestrzeni roboczej na podstawie impulsów z dwóch inkrementalnych enkoderów obrotowych. Robot poruszał się w systemie napędu różnicowego, gdzie dwa koła napędzane były przez oddzielne silniki, a ruch robota realizowany był poprzez odpowiednią synchronizację obrotów tych kół. Dzięki temu możliwe było precyzyjne kontrolowanie pozycji i orientacji robota na torze.

Strojenie regulatora PID zaczęto od implementacji w środowisku MATLAB, gdzie początkowo uzyskano zadowalające rezultaty, jednak czas ustalania wynosił 4 sekundy, co nie było optymalne. W celu poprawy wyników, zastosowano metodę Zieglera-Nicholsa, która

pozwoliła na precyzyjniejsze dobranie parametrów regulatora. Proces strojenia polegał na stopniowym zwiększaniu wartości współczynnika proporcjonalnego (KP) aż do momentu, w którym robot zaczął wykonywać oscylacje wokół pożądanej pozycji. Po wyznaczeniu wartości krytycznej KP, wykorzystano dane o okresie oscylacji i wartości Ku do dalszego określenia optymalnych parametrów PID.

Ostateczne ustawienia regulatora PID okazały się bardzo skuteczne, przy czym zdecydowano się na eliminację składnika całkującego (KI) w celu uproszczenia sterowania. Do uzyskania optymalnej reakcji systemu, zastosowano tylko dwa parametry PID: KP i KD, gdzie dla ruchu translacyjnego użyto wartości KP=1.2 oraz KD=0.15, a dla orientacji robota KP=4 oraz KD=0.32. Dzięki tym ustawieniom, robot osiągnął bardzo dobre wyniki: czas ustalania wyniósł 3 sekundy, a przesunięcie w końcowej pozycji zredukowano do zaledwie 3 mm, co pozwoliło na precyzyjne dotarcie do wyznaczonego punktu.

Podsumowując, w pracy zaprezentowano skuteczne zastosowanie regulatora PID do sterowania ruchem robota mobilnego z napędem różnicowym, które zostało zoptymalizowane metodą Zieglera-Nicholsa. Dzięki temu osiągnięto wysoką precyzję w kontroli pozycji i orientacji robota, co stanowiło kluczowy element jego skutecznego działania w zawodach robotycznych.

Wykorzystanie Algorytmów i Detekcji Przeszkód

Roboty Line Follower, poza podstawową funkcją śledzenia linii, mogą być również wyposażone w mechanizmy wykrywania i unikania przeszkód, co znacząco zwiększa ich autonomię. Do realizacji tego celu szeroko wykorzystuje się czujniki ultradźwiękowe, takie jak HC-SR04. Czujnik ten działa na zasadzie emisji fal dźwiękowych, które odbijając się od przeszkód, wracają do czujnika. Na podstawie czasu pomiędzy wysłaniem sygnału a odbiorem echa możliwe jest precyzyjne obliczenie odległości do przeszkody. Dzięki temu roboty mogą w czasie rzeczywistym modyfikować trasę, aby unikać kolizji, np. zmieniając trajektorię lub wykonując manewr omijania. W omawianej pracy [8] czujnik HC-SR04 został zintegrowany z mikrokontrolerem Arduino Uno oraz sterownikiem silników. System obejmuje także serwomechanizmy do precyzyjnego ustawiania czujnika w różnych kierunkach, co umożliwia lepsze monitorowanie otoczenia.

Eksperymentalne badania wykazały, że robot wyposażony w ten czujnik skutecznie unika przeszkód, niezależnie od ich pozycji czy warunków oświetleniowych. System ten jest tanim i wydajnym rozwiązaniem w projektach robotycznych.

Łączenie czujników ultradźwiękowych z zaawansowanymi algorytmami unikania przeszkód znacząco zwiększa możliwości robotów mobilnych, szczególnie w zakresie autonomicznej nawigacji. W literaturze [9] przykład takiego rozwiązania stanowi integracja czujników ultradźwiękowych SRF05 z algorytmami sieci neuronowych, która umożliwia precyzyjne rozpoznawanie przeszkód i skuteczne planowanie trasy w dynamicznie zmieniającym się otoczeniu. Dane z czujników ultradźwiękowych są przetwarzane w czasie rzeczywistym, a systemy oparte na uczeniu maszynowym analizują je, ucząc się optymalnych reakcji na pojawiające się przeszkody.

Wyniki badań wskazują, że zastosowanie takiego podejścia pozwala na zwiększenie efektywności robotów mobilnych, szczególnie w kontekście nawigacji w środowiskach o dużym stopniu złożoności. Integracja czujników z zaawansowanymi algorytmami umożliwia nie tylko skuteczne unikanie przeszkód, ale również rozpoznawanie obiektów o nieregularnych kształtach czy zmiennych właściwościach powierzchni. Przykładowe zastosowania obejmują roboty pracujące w rolnictwie, gdzie manewrują pomiędzy roślinami, roboty przemysłowe poruszające się w złożonych środowiskach produkcyjnych, a także autonomiczne systemy transportowe w przestrzeniach miejskich.

Dzięki tej integracji możliwe jest zwiększenie precyzji działania systemów sterowania, co w konsekwencji pozwala na osiągnięcie wyższego poziomu niezależności i bezpieczeństwa operacyjnego. Zastosowanie takich technologii w robotyce otwiera nowe perspektywy w zakresie ich wykorzystania zarówno w przemyśle, jak i w codziennym życiu.

Planowanie trajektorii jest kluczowym elementem autonomicznego działania robotów mobilnych, szczególnie w dynamicznych środowiskach. W tym kontekście stosuje się szereg zaawansowanych algorytmów opisanych w artykule [10], które pozwalają na optymalne poruszanie się robota, unikanie przeszkód i osiąganie wyznaczonego celu. Do najczęściej wykorzystywanych należą.

Algorytm Dijkstry [11] jest popularnym narzędziem do znajdowania najkrótszej ścieżki w grafach skierowanych. Działa poprzez iteracyjne obliczanie najkrótszych odległości od węzła początkowego do wszystkich innych węzłów grafu. Jest szczególnie skuteczny w dyskretnych, statycznych przestrzeniach roboczych, choć jego efektywność spada w przypadku dużych odległości między punktami.

PRM [12] jest metodą probabilistyczną, która tworzy grafy przedstawiające połączenia między wolnymi przestrzeniami robota. Używa grafów widoczności lub Woronoja, aby znaleźć optymalne ścieżki w przestrzeni roboczej. Jest to skuteczna metoda w dużych, statycznych środowiskach, szczególnie przy wielu zapytaniach o ścieżki.

Dekompozycja komórek [13] dzieli przestrzeń roboczą na prostokątne obszary, co pozwala na uproszczenie procesu poszukiwania ścieżek. Tworzy wykres dostępności, który łączy komórki, umożliwiając znalezienie najkrótszej trasy. Jest stosunkowo łatwa do implementacji, ale mniej efektywna w bardziej złożonych przestrzeniach.

APF [14] wykorzystuje pole sił, które przyciąga robota do celu i odpycha go od przeszkód. Choć skuteczne w prostych przestrzeniach, może prowadzić do problemów, gdy robot utknie w minimach lokalnych, co można rozwiązać przez modyfikację sił przyciągających.

Heurystyka [15] pozwala na szybsze znajdowanie rozwiązań, szczególnie w dynamicznych i złożonych środowiskach.

FL [16] wykorzystuje nieprecyzyjne reguły do podejmowania decyzji w warunkach niepewności. Stosowana w omijaniu przeszkód oraz planowaniu trajektorii w nieznanych przestrzeniach roboczych, jest często łączona z innymi algorytmami, np. sieciami neuronowymi.

Sieci neuronowe [17] to systemy, które uczą się i adaptują do zmieniających się warunków. Wykorzystywane do optymalizacji trajektorii robotów, są szczególnie skuteczne w dynamicznych środowiskach.

PSO to algorytm inspirowany zachowaniami społecznymi zwierząt, [18] który znajduje optymalne rozwiązania, symulując ruch cząstek w przestrzeni problemu. Stosowany do planowania trajektorii w trudnych środowiskach, umożliwia adaptację do zmieniających się warunków.

GA to technika [19] optymalizacji inspirowana procesami selekcji naturalnej. Działa poprzez tworzenie populacji rozwiązań i stosowanie operacji takich jak krzyżowanie, mutacja i selekcja. Jest skuteczna w planowaniu trajektorii w złożonych przestrzeniach roboczych.

Wybrane problemy badawcze i podejścia

Wyzwania związane z zapewnieniem efektywnej nawigacji robotów mobilnych w złożonych i dynamicznych środowiskach są jednym z najważniejszych zagadnień współczesnej robotyki. W pracy przedstawiono szereg nowoczesnych metod planowania trajektorii, które łączą algorytmy heurystyczne z technologiami przetwarzania danych pochodzących z różnych sensorów [7]. Użycie danych z takich urządzeń jak czujniki optyczne, ultradźwiękowe czy systemy wizyjne pozwala na dokładne mapowanie otoczenia i precyzyjne planowanie ruchu, umożliwiając robotowi unikanie przeszkód w czasie rzeczywistym [8].

W robotach typu Line Follower z funkcją detekcji przeszkód powszechnie wykorzystywane są czujniki ultradźwiękowe, takie jak HC-SR04, które umożliwiają wykrywanie przeszkód w

odległości do kilkudziesięciu centymetrów. W pracy "Efficient Obstacle Avoidance Using Ultrasonic Sensors for Line Following Robots" omówiono implementację algorytmów unikania przeszkód opartego na danych pochodzących z tych czujników oraz ocenę ich efektywności w różnych warunkach środowiskowych, takich jak zmiany w oświetleniu czy zakłócenia elektromagnetyczne [9].

Kierunki rozwoju i cel pracy

Rozwój robotyki mobilnej w przyszłości będzie koncentrował się na integracji najnowszych osiągnięć z dziedziny sztucznej inteligencji, algorytmów uczenia maszynowego oraz zaawansowanych technik predykcyjnych. Te innowacje pozwolą na stworzenie systemów, w których roboty będą w stanie nie tylko reagować na zmienne warunki otoczenia w czasie rzeczywistym, ale również przewidywać przyszłe zdarzenia oraz podejmować autonomiczne decyzje. Dzięki temu możliwe będzie zwiększenie efektywności ich działania w dynamicznych środowiskach, takich jak magazyny, hale produkcyjne, a nawet przestrzenie publiczne.

W szczególności, roboty typu Line Follower, które obecnie służą głównie do prostych zadań nawigacyjnych, mogą stać się kluczowym elementem w bardziej zaawansowanych systemach logistycznych i produkcyjnych. Wyposażone w zaawansowane techniki fuzji danych z różnych czujników — takich jak kamery, lidary, radary oraz czujniki inercyjne — będą mogły zapewnić jeszcze wyższą precyzję nawigacji. Połączenie danych z wielu źródeł pozwoli na skuteczniejsze omijanie przeszkód, łatwiejsze dostosowanie do zmian w otoczeniu oraz usprawnienie interakcji z innymi robotami i ludźmi.

Dodatkowo, rozwój oprogramowania zarządzającego flotą robotów [20] w sieciach współpracujących maszyn (Multi-Robot Systems, MRS) będzie stanowił kluczowy kierunek badań. Systemy te będą wykorzystywały algorytmy optymalizacji oraz metody komunikacji w czasie rzeczywistym, aby umożliwić efektywną koordynację zadań i wymianę informacji między robotami. Takie podejście znajdzie zastosowanie w aplikacjach przemysłowych, logistyce miejskiej oraz w obszarach wymagających wysokiej niezawodności, takich jak akcje ratunkowe czy eksploracja kosmiczna.

Analiza istniejących rozwiązań i projektów dotyczących robotów typu line follower oraz technologii detekcji przeszkód dostarcza cennych informacji na temat metod realizacji i zastosowanych technologii. Oto kilka przykładów.

Projekty edukacyjne: Wiele projektów typu line follower jest realizowanych w celach edukacyjnych, aby nauczyć studentów podstaw robotyki i programowania. Przykłady takich projektów obejmują roboty oparte na platformach Arduino oraz Raspberry Pi, które śledzą linię za pomocą czujników optycznych i unikają przeszkód za pomocą czujników ultradźwiękowych. Analiza tych projektów pozwala na zrozumienie, jak wykorzystać dostępne narzędzia i technologie do osiągnięcia pożądanych funkcji. Kursy robotyki, takie jak ten realizowany na ITESM w Monterrey [21], kładą nacisk na nauczanie interdyscyplinarne poprzez praktyczne projekty, w których studenci budują działające prototypy robotów. Dzięki podejściu opartemu na współpracy, uczestnicy kursu integrują wiedzę z różnych dziedzin, takich jak elektronika, mechanika, programowanie, sztuczna inteligencja, teoria sterowania czy projektowanie układów cyfrowych i analogowych.

W trakcie kursu studenci pracują w zespołach wielodyscyplinarnych, co sprzyja rozwijaniu umiejętności komunikacji, współpracy i zarządzania projektami. Każdy zespół ma za zadanie zaprojektować, zbudować i zaprezentować działający prototyp, który rozwiązuje realny problem społeczny lub przemysłowy. Proces ten obejmuje wszystkie etapy projektowania, od opracowania koncepcji, przez analizę wykonalności, aż po implementację i testowanie.

Jednym z unikalnych aspektów tego kursu jest nacisk na kreatywność i zrównoważony rozwój. Studenci są zachęcani do wykorzystywania materiałów z recyklingu, takich jak części ze starych komputerów czy zabawki zdalnie sterowane, co nie tylko zmniejsza koszty, ale także rozwija ich zdolności do twórczego rozwiązywania problemów.

Dzięki temu podejściu uczestnicy nie tylko zdobywają praktyczne doświadczenie w projektowaniu robotów, ale także uczą się, jak przełożyć teoretyczne koncepcje na rzeczywiste, działające rozwiązania. Kurs przygotowuje ich do wyzwań, które napotkają w przemyśle, ucząc ich elastyczności, samodzielności i odpowiedzialności za własne projekty.

Zastosowania przemysłowe: W przemyśle roboty mobilne, takie jak systemy typu line follower, znajdują zastosowanie nie tylko w codziennym transporcie materiałów, ale również w zadaniach związanych z inspekcją, monitorowaniem oraz zarządzaniem ryzykiem w trudnych i niebezpiecznych środowiskach. Jednym z przykładów zastosowania robotów w przemyśle jest projekt ARK [22] (Autonomous Robot for a Known Environment), który został opracowany w celu autonomicznych inspekcji i weryfikacji istniejących systemów monitorujących w środowiskach przemysłowych, takich jak zakłady produkcyjne, laboratoria inżynieryjne oraz obszary narażone na niebezpieczeństwo.

W ramach tego projektu, robot ARK działał w rozległej przestrzeni laboratorium AECL CANDU w Mississauga, w Kanadzie, które zajmuje około 50 000 stóp kwadratowych. W tym środowisku występują liczne wyzwania dla mobilnych robotów, takie jak duże otwarte przestrzenie, zmieniające się warunki oświetleniowe, obecność ludzi i maszyn poruszających się po terenie, a także występowanie przeszkód na podłodze, takich jak rury czy olejowe i wodne wycieki. Celem robota ARK było autonomiczne poruszanie się w tym złożonym środowisku w celu wykonywania inspekcji i weryfikacji funkcji czujników, które monitorowały różne parametry przemysłowe, takie jak np. wycieki czy uszkodzenia systemów.

ARK korzystał z zaawansowanych technologii nawigacyjnych, w tym systemu wizualnego, który wykorzystywał naturalne punkty orientacyjne w otoczeniu (np. znaki alfanumeryczne, drzwi, słupy) do określania swojej lokalizacji w globalnym układzie współrzędnych. Dzięki temu robot mógł samodzielnie poruszać się po zakładzie, bez potrzeby modyfikacji środowiska (np. instalacji specjalnych znaczników czy beaconów), co jest typowym wyzwaniem w wielu zastosowaniach przemysłowych. Ponadto, robot musiał być wyposażony w zaawansowane czujniki (takie jak kamery, sonary czy czujniki podczerwieni), które umożliwiały wykrywanie przeszkód i reagowanie na nie, co było kluczowe w środowisku, w którym pracowały również inne maszyny i ludzie.

W kontekście przemysłowym projekt ARK stanowi doskonały przykład zastosowania robotów mobilnych w zadaniach inspekcyjnych, które przyczyniają się do zwiększenia bezpieczeństwa, redukcji ryzyka oraz optymalizacji operacji. Roboty te mogą zmniejszyć narażenie pracowników na szkodliwe czynniki, takie jak promieniowanie w obszarach reaktorów jądrowych, oraz przyczynić się do poprawy efektywności procesów produkcyjnych poprzez automatyzację działań kontrolnych i diagnostycznych. Dzięki projektom takim jak ARK, możliwe jest tworzenie bardziej zaawansowanych i elastycznych systemów robotycznych, które mogą działać w trudnych, dynamicznych środowiskach przemysłowych.

Projekty badawcze: W literaturze naukowej istnieje wiele projektów badawczych, które koncentrują się na wykorzystaniu robotów mobilnych do detekcji przeszkód oraz na optymalizacji ich nawigacji przy użyciu zaawansowanych algorytmów sterowania. Przykładem takiego projektu jest badanie [23] nad zastosowaniem algorytmów optymalizacji, takich jak Algorytm Kolonii Sztucznych Pszczół (ABC) oraz Algorytm Genetyczny (GA), do dostosowywania parametrów sterowania PID w robotach mobilnych. W pracy badawczej omówiono modyfikację parametrów sterowników PID przy użyciu tych algorytmów, aby poprawić zdolność robota do śledzenia określonych trajektorii, takich jak okrężna czy falowa.

Badanie koncentruje się na dwóch PID: jednym odpowiedzialnym za kontrolę prędkości, a drugim za kontrolę azymutu robota. Zastosowanie algorytmów optymalizujących, takich jak ABC i GA, pozwala na dostosowanie parametrów sterownika PID, co umożliwia robotowi autonomiczne poruszanie się w nieznanych środowiskach, takich jak strefy katastrof czy obszary przemysłowe, gdzie nawigacja jest utrudniona. Testy wykazały, że algorytm ABC skutkuje lepszymi wynikami w porównaniu do GA, oferując wyższą skuteczność w optymalizacji sterowania.

Przykład ten ilustruje najnowsze podejścia do sterowania robotami mobilnymi, w tym zastosowanie sztucznej inteligencji i algorytmów optymalizacyjnych do poprawy precyzyjności i efektywności nawigacji. W kontekście przemysłowym, podobne algorytmy mogą zostać wykorzystane do poprawy funkcji robotów mobilnych, takich jak wykrywanie przeszkód, unikanie kolizji, oraz wykonywanie zadań w niebezpiecznych środowiskach, co ma ogromne znaczenie w branży, w której bezpieczeństwo i efektywność są kluczowe. Analiza tego typu badań pozwala na zrozumienie aktualnych trendów w robotyce mobilnej i integrację nowoczesnych technologii w projektach przemysłowych.

Zawody robotyczne: Zawody robotyczne, takie jak RoboCup, Line Follower Competition czy FIRST Robotics Competition, odgrywają kluczową rolę w rozwoju technologii robotycznych. W takich zawodach drużyny rywalizują w różnych kategoriach, testując i rozwijając swoje rozwiązania w zakresie robotyki mobilnej, detekcji przeszkód, nawigacji i sterowania. Analiza projektów startujących w tych zawodach może dostarczyć cennych inspiracji i praktycznych wskazówek dotyczących implementacji i optymalizacji systemów robotów.

W artykule [24] opisano doświadczenia uczestników międzynarodowych zawodów robotycznych, takich jak RoboCup i Festival International des Sciences et Technologies (FIST), które odbywają się od 1995 roku. Zawody te stawiają przed zespołami wyzwania w zakresie technologii robotycznych, takich jak percepcja, sterowanie ruchem czy współpraca robotów w dynamicznych warunkach. Przykładem może być udział drużyn z Portugalii, takich jak Instituto Superior Técnico (IST), Universidade de Aveiro (UA) i Universidade do Minho (UM), które przez lata osiągnęły sukcesy, rozwijając innowacyjne rozwiązania, jak np. systemy wizyjne do śledzenia toru i unikania przeszkód.

Zawody robotyczne są nie tylko okazją do rywalizacji, ale także do zdobywania wiedzy praktycznej, którą uczestnicy mogą zastosować w innych projektach badawczych. Analiza rozwiązań wykorzystywanych w tych zawodach, takich jak algorytmy detekcji przeszkód,

sterowanie robotami czy integracja różnych technologii, może znacząco wpłynąć na rozwój nowych systemów robotycznych. Udział w takich wydarzeniach sprzyja również rozwojowi współpracy interdyscyplinarnej oraz motywacji do dalszego badania i udoskonalania technologii robotycznych.

Projekty open-source: W społeczności open-source dostępnych jest wiele projektów dotyczących robotów typu line follower oraz technologii detekcji przeszkód. Przykłady takich projektów można znaleźć na platformach takich jak GitHub, które udostępniają kody źródłowe, schematy oraz dokumentację. To umożliwia innym inżynierom i entuzjastom robotyki korzystanie z istniejących rozwiązań i ich modyfikowanie. Jednym z popularniejszych przykładów są roboty podążające za linią, które zazwyczaj śledzą białą lub czarną linię. Te roboty wykorzystywane są w wielu aplikacjach, od edukacyjnych po przemysłowe, gdzie wymagane jest automatyczne podążanie trasą.

W tym kontekście warto wspomnieć o projekcie [25], który opisuje autonomicznego robota, który nie tylko podąża za linią, ale również unika przeszkód na swojej drodze. Robot porusza się po wyznaczonej ścieżce, a w przypadku napotkania przeszkody, zatrzymuje się i zmienia trasę, po czym wraca na pierwotną ścieżkę. Co ważne, jeśli robot zboczy z toru, automatycznie nawiązuje połączenie z zapisanym numerem alarmowym za pomocą modułu GSM [26], informując o awarii. Projekt ten wykorzystuje szereg czujników, które współpracują z kontrolerami, aby zapewnić precyzyjne śledzenie linii oraz skuteczną detekcję przeszkód.

Tego typu roboty, zwłaszcza w erze automatyzacji, zyskują na znaczeniu. Znajdują zastosowanie w wielu dziedzinach, takich jak transport wewnętrzny na halach produkcyjnych, na lotniskach do przewozu bagażu, czy w biurach do transportu dokumentów. Projekt opisany w artykule bazuje na czujnikach podczerwieni do śledzenia linii oraz czujnikach ultradźwiękowych do detekcji przeszkód. Dzięki tej kombinacji, robot jest w stanie nie tylko precyzyjnie poruszać się po zadanej trasie, ale także reagować na przeszkody, co jest kluczowe w przypadku zastosowań wymagających niezawodności i bezpieczeństwa.

Dodatkowo, w projekcie przeprowadzono eksperymenty, które badały czas potrzebny robotowi na pokonanie przeszkód o różnej długości oraz przy różnych prędkościach. System oparty na Arduino umożliwia skuteczne zarządzanie danymi z czujników, co pozwala na podejmowanie odpowiednich decyzji dotyczących kierunku jazdy robota. Gdy robot nie jest w stanie wrócić na zdefiniowaną ścieżkę, moduł GSM zapewnia połączenie alarmowe, co może być przydatne w sytuacjach awaryjnych.

Takie projekty pokazują, jak roboty podążające za linią, wyposażone w technologie detekcji przeszkód, mogą być wykorzystywane w szerokim zakresie zastosowań, jednocześnie będąc doskonałym przykładem zastosowania technologii open-source w rozwoju robotyki.

Komercyjne roboty mobilne: Komercyjne roboty mobilne, takie jak roboty sprzątające (np. Roomba), stanowią istotny obszar analizy w kontekście rozwoju robotyki. Badania te oferują wgląd w technologie, rozwiązania inżynierskie oraz strategie komercjalizacji, które mogą być zastosowane w innych dziedzinach robotyki. W ostatnich latach rośnie znaczenie robotów usługowych, szczególnie tych, które wspierają opiekę zdrowotną, logistykę, mieszkalnictwo oraz działania ratunkowe.

Przykładem jest badanie robota Roomba [27], popularnego robota czyszczącego podłogi, którego system mobilności opiera się na napędzanych kołach i zaawansowanych czujnikach, umożliwiających detekcję przeszkód, wykrywanie brudu oraz dostosowywanie się do różnych powierzchni. Mimo że Roomba jest szeroko badana pod kątem mechanicznego designu i autonomii, istnieje niewiele badań na temat optymalizacji przestrzeni, w której roboty działają, aby maksymalizować ich efektywność. W artykule zaproponowano cztery zasady projektowania przestrzeni inkluzywnych dla robotów: obserwowalność, dostępność, aktywność i bezpieczeństwo, które mają na celu ułatwienie robotom funkcjonowania w ludzkich środowiskach.

Badania te podkreślają potrzebę integracji projektowania robotów i przestrzeni, w których te roboty operują, aby umożliwić ich efektywną i bezpieczną interakcję z ludźmi. Projektowanie takich przestrzeni obejmuje uwzględnienie aspektów, takich jak dostępność, obserwowalność, aktywność oraz bezpieczeństwo, które mają kluczowe znaczenie dla zapewnienia optymalnego działania robotów w środowiskach współdzielonych z ludźmi.

Z powyższego przeglądu wynika, że jednym z głównych wyzwań w rozwoju robotów mobilnych, w tym robotów typu line follower, jest nie tylko ich techniczne doskonalenie, ale również integracja w środowiskach, w których będą operować. Ważnym aspektem jest również edukacja w zakresie projektowania i programowania takich systemów, co pozwoli na dalsze rozwijanie kompetencji w tej dziedzinie oraz zwiększenie świadomości o możliwościach i ograniczeniach tych technologii.

W ramach niniejszej pracy szczególny nacisk zostanie położony na projekt edukacyjny, który ma na celu pogłębienie wiedzy na temat programowania robotów mobilnych. Realizacja projektu ma służyć zarówno jako narzędzie dydaktyczne, jak i jako przykład implementacji nowoczesnych technologii w robotyce mobilnej. Opracowane rozwiązanie będzie uwzględniać

praktyczne aspekty związane z programowaniem oraz interakcją z otoczeniem, co przyczyni się do lepszego zrozumienia wyzwań i możliwości w tej dziedzinie.

3.Konstrukcja robota

Celem tego rozdziału jest przedstawienie procesu projektowania oraz konstrukcji robota typu Line Follower. W pierwszej części omówiona zostanie podstawa konstrukcyjna robota, a następnie szczegółowe elementy mechaniczne, takie jak silniki, czujniki oraz pozostałe komponenty. Na końcu zaprezentowane zostanie złożenie całej konstrukcji robota w programie CAD.

3.1 Podstawa konstrukcji robota

Podstawa robota jest kluczowym elementem, na którym opiera się cała konstrukcja urządzenia, w tym wszystkie pozostałe komponenty, takie jak napęd, czujniki i elektronika. Jej główną funkcją jest zapewnienie stabilności robota oraz umożliwienie skutecznego montażu pozostałych części. Konstrukcja podstawy została zaprojektowana z myślą o mobilności, trwałości oraz łatwości w produkcji i montażu.

Podstawa, wraz z pokrywą robota, została w pełni zaprojektowana w programie SolidWorks, co pozwoliło na precyzyjne dopasowanie wszystkich elementów oraz optymalizację konstrukcji pod kątem wydajności i funkcjonalności. Za pomocą narzędzi CAD, takich jak wyciąganie, zaokrąglanie czy cięcie, stworzono kształt, który zapewnia stabilność robota przy minimalnej wadze. Wybrana konstrukcja pozwala na łatwe umiejscowienie elementów takich jak silniki, czujniki, akumulatory oraz mikrokontroler, jednocześnie zapewniając odpowiednią przestronność i łatwy dostęp do podzespołów.

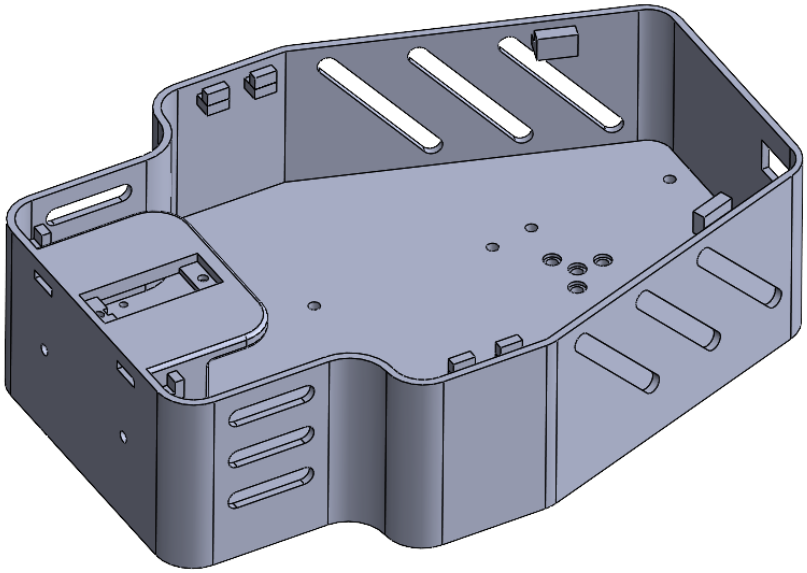
Do produkcji podstawy wykorzystano materiał PLA, który charakteryzuje się wysoką jakością druku 3D, łatwością obróbki oraz wystarczającą wytrzymałością mechaniczną, odpowiednią dla robota mobilnego przeznaczonego do śledzenia linii. PLA jest materiałem ekologicznym, bezpiecznym w obróbce, a jego właściwości termoplastyczne sprawiają, że idealnie nadaje się do druku części o złożonej geometrii, takich jak podstawa robota.

Wymiary podstawy zostały dobrane tak, aby zapewnić odpowiednią przestronność, umożliwiając montaż wszystkich wymaganych komponentów, a także zapewnić odpowiednią równowagę robota. Cała konstrukcja została zaprojektowana w sposób, który umożliwia łatwy montaż elementów napędowych i czujników, a także umożliwia swobodny dostęp do wnętrza robota w przypadku konieczności konserwacji lub modyfikacji.

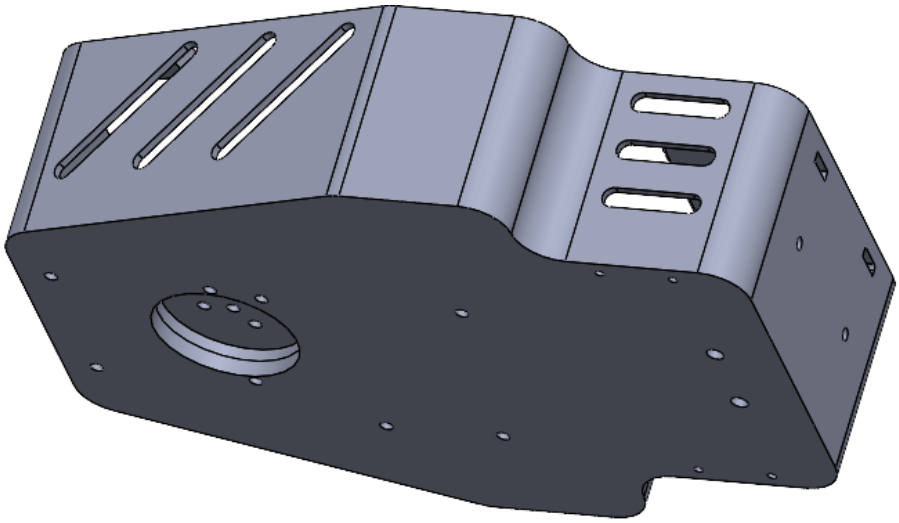
Przyjęta konstrukcja podstawy odpowiada na kluczowe wymagania projektu robota typu Line Follower. Przede wszystkim jest to konstrukcja stabilna i lekka, co umożliwia robota o

odpowiedniej mobilności, zdolnego do precyzyjnego śledzenia linii. Dodatkowo, projekt podstawy uwzględnia możliwość łatwej rozbudowy robota w przyszłości, np. o dodatkowe czujniki, system wizyjny lub zmiany w układzie napędowym.

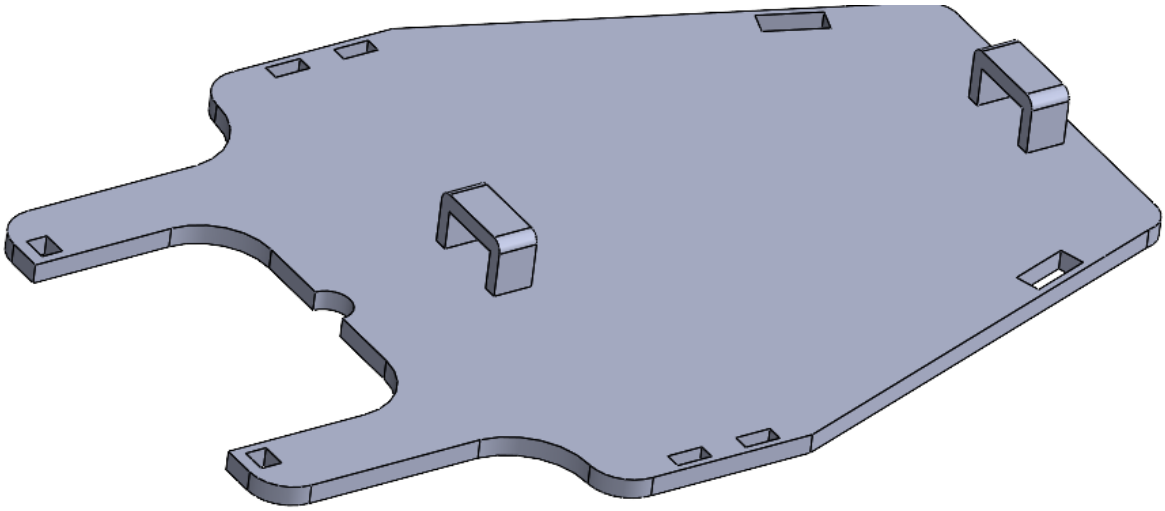
Dzięki zastosowaniu programów CAD oraz materiału PLA, cała konstrukcja jest efektywna kosztowo oraz łatwa w wydruku.



Rysunek 1.Podstawa robota



Rysunek 2.Podstawa robota



Rysunek 3.Pokrywa podstawy

3.2 Elementy mechaniczne robota

W tym podrozdziale zostaną omówione elementy mechaniczne robota, w tym systemy napędowe oraz mechanizmy odpowiedzialne za realizację ruchu.

3.2.1 Napęd

Rysunek 4.Silnik N20-BT38 micro 250:1 120RPM - 9V

źródło: https://botland.com.pl/silniki-micro-n20-seria-mp-medium-power/12599-silnik-n20-bt38-micro-2501-120rpm-9v-5904422306687.html

Do napędu robota wykorzystano dwa silniki N20-BT38 micro 250:1, które charakteryzują się niewielkimi rozmiarami, ale dużą mocą, co czyni je idealnym wyborem do projektów

robotów mobilnych, takich jak Line Follower. Silniki z serii N20 są często wykorzystywane w robotyce amatorskiej, modelarstwie i innych zastosowaniach wymagających precyzyjnego napędu w małych konstrukcjach. Dzięki niewielkim rozmiarom i dużemu momentowi obrotowemu, silniki te zapewniają wystarczającą moc do poruszania robotem na różnych powierzchniach, jednocześnie nie obciążając nadmiernie układu zasilania.

Specyfikacja silników N20-BT38:

- Typ silnika: Silnik DC z przekładnią
- Napięcie zasilania: 3 V – 9 V (optymalne 6 V)
- Prędkość bez obciążenia (9 V): 120 RPM
- Moment obrotowy (9 V): 4,3 kg·cm (0,422 Nm)
- Przełożenie: 250:1
- Prąd bez obciążenia (9 V): 60 mA

Silniki [Rysunek 4] zostały zamocowane na dedykowanych uchwytach Pololu. Mocowania te są specjalnie zaprojektowane do silników z serii N20, co zapewnia stabilne i precyzyjne umiejscowienie silników w podstawie robota. Zestaw mocowań zawiera dwie sztuki uchwytów oraz niezbędne śruby do montażu, co umożliwia łatwą instalację silników i ich szybką wymianę w razie potrzeby.

Rysunek 5.Mocowania do micro silników Pololu

źródło: https://botland.com.pl/mocowania-silnikow/36-mocowania-do-micro-silnikow-pololu-czarne-2szt-pololu-989-5904422300289.html

Silniki są zamocowane w podstawie robota Rysunek 2, a wały silników zostały połączone z kołami o średnicy 60 mm, które zapewniają odpowiednią stabilność robota na różnych powierzchniach.

3.2.2 Koła

Rysunek 6.Koła 32x7mm-Pololu1087

źródło: https://botland.com.pl/kola-z-oponami/12-kola-32x7mm-czarne-pololu-1087-5903351247924.html

W projekcie robota zastosowano dwa koła 32x7 mm Pololu 1087. Są to koła zaprojektowane specjalnie do współpracy z micro silnikami Pololu, które są powszechnie wykorzystywane w robotyce mobilnej. Koła te charakteryzują się wysoką przyczepnością dzięki gumowym oponom, które posiadają bieżnik, co zapewnia lepszą trakcję na różnych powierzchniach i minimalizuje wpływ zanieczyszczeń podłoża na tarcie.

Specyfikacja kół Pololu 1087:

- Średnica koła: 32 mm
- Szerokość opony: 6,5 mm
- Średnica otworu montażowego: 3 mm (w kształcie litery D)
- Masa: 3,2 g

Koła posiadają otwór montażowy w kształcie litery D o średnicy 3 mm, co pozwala na łatwe zamocowanie ich na wałach silników Pololu, zapewniając pewne i stabilne połączenie. Dzięki temu rozwiązaniu, koła te są łatwe w montażu i demontażu, co może być istotnym atutem w przypadku ewentualnej wymiany lub konserwacji.

Dodatkowo, bieżnik na gumowej oponie pomaga zminimalizować poślizg i zapewnia lepszą przyczepność robota do podłoża, co jest niezbędne do precyzyjnego śledzenia linii oraz stabilności w trakcie ruchu

3.2.3 Kulka podporowa

Rysunek 7.Ball Caster 1" plastikowy - Pololu 2691

źródło: https://botland.com.pl/kulki-podporowe/4650-ball-caster-1-plastikowy-pololu-2691-5904422364472.html?cd=21802601646&ad=&kd=&gad_source=1&gclid=Cj0KCQiAkJO8BhCGARIsAMkswyhzcJFXvsV88LS-MQMLs0KopEb42LRrJdQTQgXJ7DExczrt6mkm-_QaApBlEALw_wcB

W projekcie robota zastosowano kulę podporową umieszczoną w podstawie robota [Rysunek 2] (Ball Caster) Pololu 2691 o średnicy 1 cala jako element stabilizujący przednią część konstrukcji. Jest to komponent wykonany z lekkiego plastiku, który pełni kluczową rolę w zapewnieniu stabilności i płynności ruchu robota. Umożliwia on łatwe manewrowanie oraz minimalizuje tarcie na powierzchniach, co wspomaga precyzyjną pracę robota typu Line Follower.

Specyfikacja Ball Caster Pololu 2691:

- Średnica kulki: 1 cal (ok. 25,4 mm)
- Materiał kulki: lekki plastik

Ball Caster składa się z kulki umieszczonej w specjalnej czaszy, która zabezpiecza kulkę przed wypadnięciem i pozwala na niemal pełne jej odkrycie. Dzięki temu konstrukcja umożliwia płynne przetaczanie się kulki, zapewniając minimalny opór i wysoką zwrotność na różnych powierzchniach. Kulka podporowa w tej formie jest idealnym rozwiązaniem do dwukołowych robotów, które nie posiadają aktywnego systemu balansowania, ponieważ efektywnie wspomaga podparcie i stabilność urządzenia.

3.2.4 Serwonapęd

Rysunek 8.Serwo TowerPro SG-90 - micro - 180°

źródło: https://botland.com.pl/serwa-typu-micro/13128-serwo-sg-90-micro-180-5904422350338.html?cd=21802601646&ad=&kd=&gad_source=1&gclid=Cj0KCQiAkJO8BhCGARIsAMkswyiosDt-r__cS_EEVUtFzlGIK99tv7wIeFjsKKVywSKhhmU70QfhKOEaAnWOEALw_wcB

Dodatkowo w projekcie zastosowane zostało serwo SG-90 firmy TowerPro, które odpowiada za obracanie czujnika ultradźwiękowego. Serwo to jest lekkie, małe i energooszczędne, co czyni je doskonałym rozwiązaniem do zastosowań w robotyce, gdzie przestrzeń i masa są ograniczone.

Specyfikacja serwa SG-90:

- Moment obrotowy: 1,8 kg·cm (0,18 Nm)
- Prędkość: 0,1 s/60°
- Zakres obrotu: 180°
- Masa: 9 g

Serwo zostało zastosowane w projekcie do precyzyjnego obracania czujnika ultradźwiękowego w celu mierzenia odległości od wykrytych przeszkód. Dzięki swoim małym wymiarom i dużej precyzji, serwo SG-90 zapewnia idealne warunki do szybkiego i dokładnego ustawiania czujnika, co jest kluczowe w przypadku detekcji przeszkód i odpowiedniej reakcji robota.

3.2.5 Uchwyt montażowy do czujnika odległości HC-SR04

Rysunek 9.Uchwyt montażowy do czujnika odległości HC-SR04

źródło: https://botland.com.pl/ultradzwiekowe-czujniki-odleglosci/4822-uchwyt-montazowy-do-czujnika-odleglosci-hc-sr04-5904422308339.html?cd=20567593583&ad=&kd=&gad_source=1&gclid=Cj0KCQiAkJO8BhCGARIsAMkswyjJTTRqIaFSCnqkgy70WsxJnP1I1cJN5JpTPz-1LW6O75QNsnch65MaAhwMEALw_wcB

Uchwyt montażowy do czujnika odległości HC-SR04, wykonany z przezroczystego akrylu o grubości 2,6 mm. Uchwyt ten służy do stabilnego mocowania ultradźwiękowego czujnika odległości HC-SR04, który został zamontowany na serwomechanizmie SG90.

Specyfikacja uchwytu montażowego:

- Materiał wykonania: przezroczysty akryl
- Grubość: 2,6 mm
- Wymiary: 51 x 33 x 15 mm
- Otwory montażowe: dwa otwory o średnicy 3,5 mm, rozstawione co 13 mm

3.3 System sterowania i czujniki

Centralnym elementem systemu sterowania robota, będącym jego „mózgiem”, jest jednostka sterująca, która koordynuje pracę wszystkich podzespołów. Kluczową rolę odgrywają również czujniki, umożliwiające pozyskiwanie informacji o otoczeniu i zapewniające precyzyjne wykonywanie zadań.

3.3.1 Arduino Uno Rev3

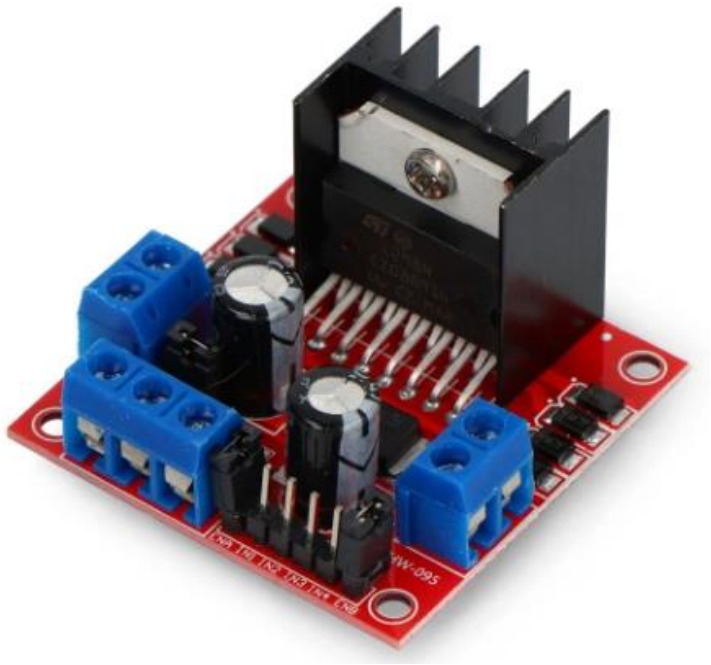
Rysunek 10.Arduino Uno Rev3

źródło: https://botland.com.pl/arduino-seria-podstawowa-oryginalne-plytki/1060-arduino-uno-rev3-a000066-7630049200050.html?cd=18298825138&ad=&kd=&gad_source=1&gclid=Cj0KCQiAkJO8BhCGARIsAMkswyhgIwzi1G8NyNeN7oESJ9vDZFI8XcP9oLCv6mVEyTWE2xB3jqE06gwaAog_EALw_wcB

W projekcie do sterowania komponentami użyto popularnej płytki Arduino Uno Rev3, opartej na mikrokontrolerze ATmega328. Arduino Uno oferuje 14 cyfrowych wejść/wyjść, z czego 6 może działać jako wyjścia PWM, co jest przydatne przy sterowaniu silnikami oraz jasnością diod LED. Dodatkowo płytka posiada 6 wejść analogowych, co umożliwia odczytywanie sygnałów z czujników analogowych.

Dzięki wszechstronnym możliwościom, Arduino Uno Rev3 jest idealnym wyborem dla amatorskich projektów robotycznych, zapewniając intuicyjną kontrolę nad komponentami, co znacznie upraszcza programowanie i uruchamianie robota. W [Tabela 1] pokazano realizację sterowania.

3.3.2 L298N - dwukanałowy sterownik silników



Rysunek 11.L298N - dwukanałowy sterownik silników

źródło: https://botland.com.pl/sterowniki-silnikow-moduly/3164-l298n-dwukanalowy-sterownik-silnikow-modul-12v-2a-5904422359317.html

W projekcie wykorzystano moduł dwukanałowego sterownika silników L298N, który umożliwia sterowanie kierunkiem oraz prędkością dwóch silników prądu stałego. Moduł jest zasilany napięciem do 12 V, a jego maksymalny prąd na kanał wynosi 2 A. Wbudowany regulator napięcia 5 V zasila część logiczną sterownika, co upraszcza integrację z mikrokontrolerami, takimi jak Arduino.

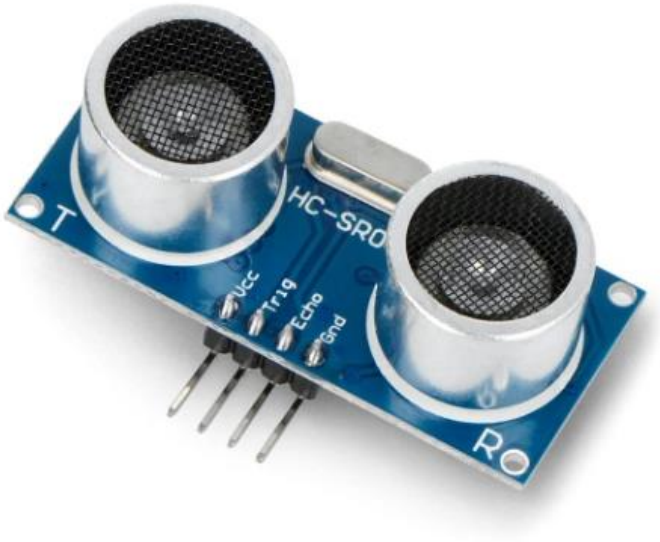
Dzięki zastosowaniu złącz śrubowych typu ARK, zasilanie oraz przewody silników można łatwo podłączyć, bez konieczności lutowania. Sygnały sterujące wyprowadzone są na popularne złącza goldpin, co ułatwia podłączenie do płytek sterujących za pomocą przewodów połączeniowych.

Sterownik L298N umożliwia pełną kontrolę nad kierunkiem obrotów oraz prędkością silników za pomocą sygnałów PWM. To rozwiązanie jest łatwe do wdrożenia i dobrze udokumentowane, co pozwala na intuicyjne podłączenie i uruchomienie w projekcie.

IN1/IN3	IN2/IN4	Wyjścia silników
Stan wysoki	Stan niski	Silnik obraca się z maksymalną prędkością zadaną poprzez pwm (różną od 0) zgodnie ze wskazówkami zegara.
Stan niski	Stan wysoki	Silnik obraca się z maksymalną prędkością zadaną poprzez pwm (różną od 0) przeciwnie do ruchu wskazówek zegara.
Stan niski	Stan niski	Przy podaniu stanu wysokiego na wejście PWM - szybkie hamowanie silników (fast stop).
Stan wysoki	Stan wysoki	Przy podaniu stanu wysokiego na wejście PWM - szybkie hamowanie silników (fast stop).
Stan wysoki	Stan wysoki	Przy podaniu stanu niskiego na wejście PWM - swobodne hamowanie (soft stop).

Tabela 1.Tabela opisująca sterowanie realizowane za pomocą sterownika

3.3.3 Ultradźwiękowy czujnik odległości HC-SR04



Rysunek 12.Ultradźwiękowy czujnik odległości HC-SR04

źródło: https://botland.com.pl/ultradzwiekowe-czujniki-odleglosci/1420-ultradzwiekowy-czujnik-odleglosci-hc-sr04-2-200cm-justpi-5903351241366.html?cd=18298825138&ad=&kd=&gad_source=1&gclid=Cj0KCQiAkJO8BhCGARIsAMkswyiaCswEwmmXv26aeXfUe342_gx8hOZ3ooQY_8NZ_TyCzMp_oahWtv4aAo0AEALw_wcB

W projekcie zastosowano ultradźwiękowy czujnik odległości HC-SR04, działający w zakresie 2–200 cm i zasilany napięciem 5 V. Czujnik wysyła sygnał ultradźwiękowy o częstotliwości 40 kHz, a jego czas powrotu jest proporcjonalny do odległości od obiektu. Kompatybilny z płytkami Arduino i Raspberry Pi, HC-SR04 jest powszechnie stosowany w projektach robotycznych, w tym w zdalnie sterowanych pojazdach, dronach czy systemach parkowania.

Aby wykonać pomiar, na pin TRIG należy podać impuls o wartości 5 V przez 10 µs, co aktywuje sygnał ultradźwiękowy. Czas powrotu sygnału pozwala obliczyć odległość za pomocą wzoru:

$$distance = \frac{high\ level\ time \times velocity\ of\ sound}{2} = \frac{high\ lovel\ time \times 340[\frac{m}{s}]}{2} \quad (1)$$

gdzie:

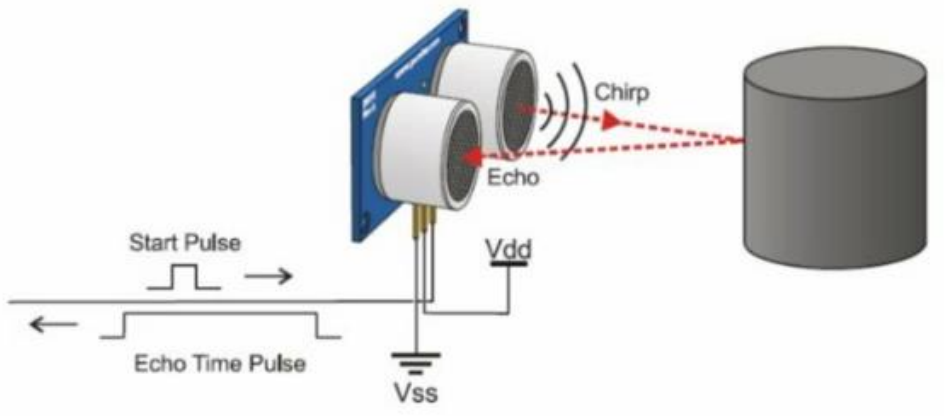
- distance – zmierzona odległość,
- high level time – czas trwania stanu wysokiego w mikrosekundach (µs),
- velocity of sound – prędkość dźwięku w powietrzu (340 m/s).

W uproszczeniu, wynik w centymetrach można obliczyć za pomocą równania:

$$distance[cm] = \frac{high\ level\ time\ [\mu s] \times 34}{1000 \times 2} \quad (2)$$

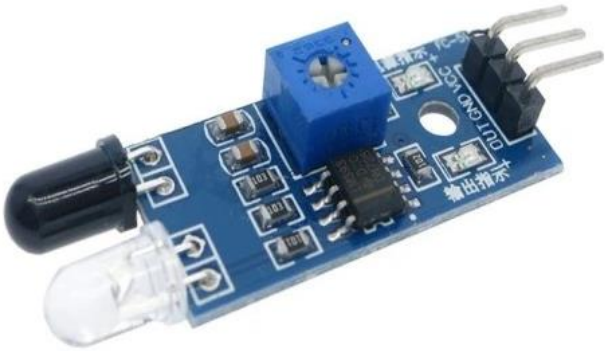
Specyfikacja HC-SR04:

- Zakres pomiarowy: 2–200 cm
- Napięcie zasilania: 5 V
- Pobór prądu: 15 mA
- Częstotliwość pracy: 40 kHz



Rysunek 13.Działanie czujnika ultradźwiękowego

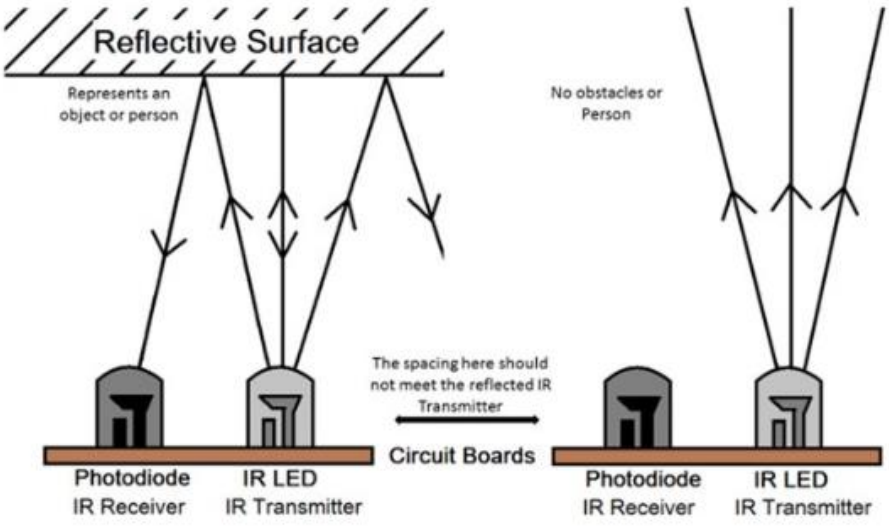
3.3.4 Czujnik odbiciowy optyczny TCRT5000 LM393 IR



Rysunek 14.Czujnik odbiciowy optyczny TCRT5000 LM393 IR

źródło: https://www.elektroweb.pl/pl/czujniki-odleglosci/584-modul-wykrywania-przeszkod-odbiciowy-optyczny-tcrt5000-lm393-ir.html?srsltid=AfmBOorOR4ZGvwVmoS4YZaI3kpi1Su0nUQi8PSW6adYX4ox-qJSbtT2x

W projekcie zastosowano odbiciowy czujnik optyczny TCRT5000 z komparatorem LM393. Czujnik ten, zasilany napięciem od 3.3 V do 5 V, wykrywa przeszkody na zasadzie odbicia światła od powierzchni. Dzięki zastosowanemu potencjometrowi użytkownik może dostosować czułość czujnika, aby precyzyjnie reagował na różnice między jasnymi a ciemnymi powierzchniami. Moduł generuje sygnał cyfrowy, którego aktywność można regulować za pomocą potencjometru, co umożliwia rozpoznanie białych i czarnych powierzchni.
Czujniki IR składają się z diody LED emitującej podczerwień (nadajnik) oraz fotodiody (odbiornik), która wykrywa emitowane światło podczerwone. Oporność i napięcie wyjściowe fotodiody zmieniają się proporcjonalnie do ilości odbieranego światła IR.



Rysunek 15.Działanie czujnika odbiciowego

3.3.5 Akumulator 18650 li-ion 2500 mah samsung inr18650-25r 20a

Rysunek 16.Akumulator 18650 li-ion 2500 mah samsung inr18650-25r 20a

Źródło: https://botland.com.pl/akumulatory-li-ion/23832-ogniwo-18650-li-ion-samsung-inr18650-25r-2500mah-20a.html

W projekcie jako źródło zasilania zastosowałem dwa ogniwa litowo-jonowe Samsung INR 18650-25R, połączone szeregowo. Każde z ogniw ma nominalne napięcie 3,7 V i pojemność 2500 mAh (minimalna pojemność to 2450 mAh). Ogniwa charakteryzują się maksymalnym prądem rozładowania do 20 A (ciągłym) oraz 100 A (impulsowym przez 1 sekundę). Standardowy prąd ładowania wynosi 1750 mA, a szybki prąd ładowania to 4000 mA. Maksymalne napięcie ładowania wynosi 4,2 V, a minimalne napięcie odcięcia to 2,5 V. Połączenie ogniw szeregowo zapewnia odpowiednie napięcie zasilania dla mojego systemu.

3.3.6 Przełącznik kołyskowy dwustabilny

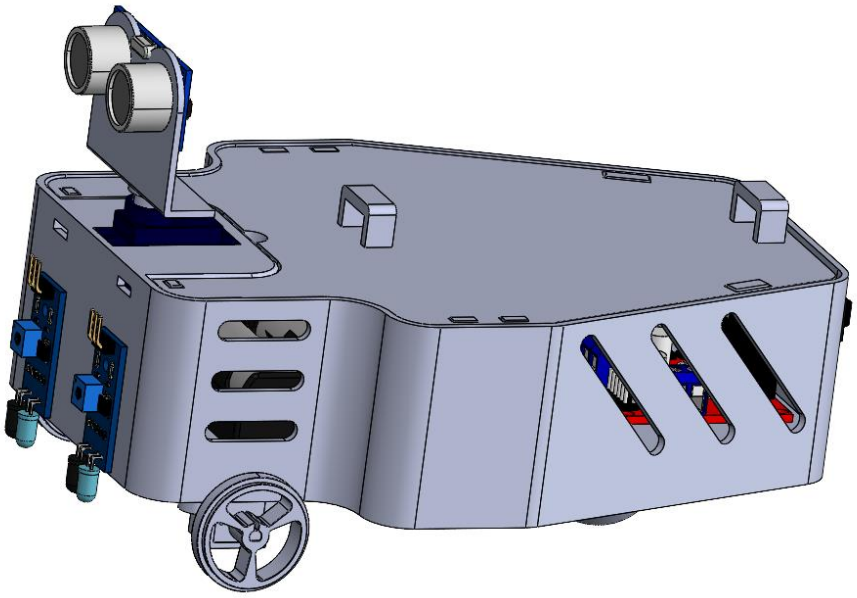
Rysunek 17.Przełącznik kołyskowy dwustabilny

źródło: https://sklep.avt.pl/pl/products/wlacznik-on-off-mrs-101a-c3-6a-250vac-czarny-170036.html?gad_source=1&gclid=Cj0KCQiAkJO8BhCGARIsAMkswyiOenLfowqV4UFpthyBGNyWGKN_wmr8RABc-5jpeOtM0owarkvgXZ8aAjmfEALw_wcB

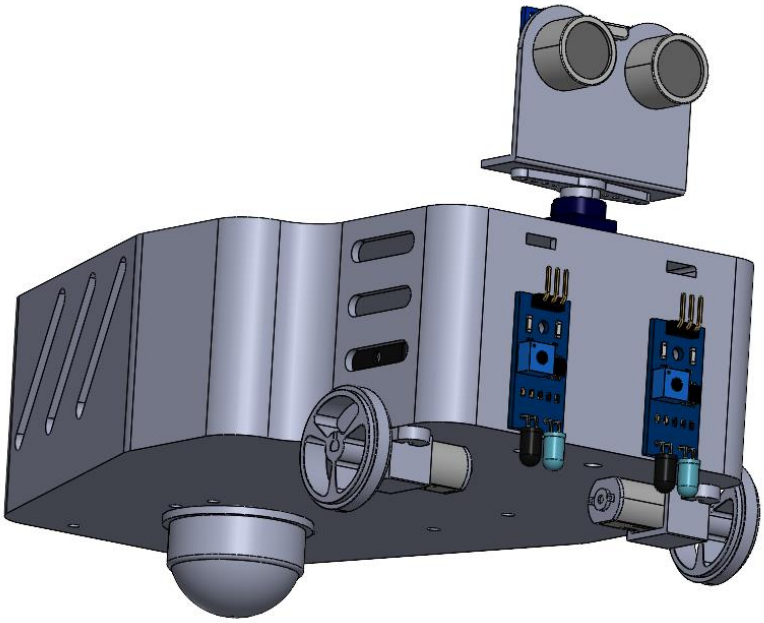
Jako włącznik robota zastosowano przełącznik kołyskowy dwustabilny typu ON-OFF o wymiarach 15x10 mm. Jest to przełącznik bistabilny, który umożliwia przełączenie pomiędzy dwoma stanami: włączonym i wyłączonym. Przełącznik obsługuje maksymalne napięcie 250V oraz maksymalne natężenie prądu 3A

3.4 Zestawienie wszystkich komponentów w CAD

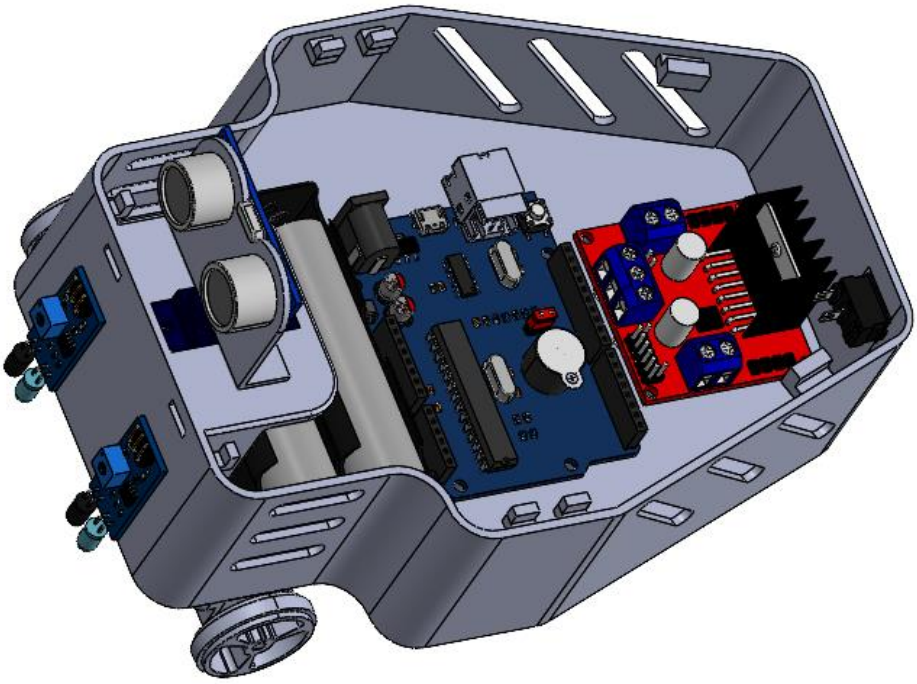
Po zestawieniu wszystkich komponentów w projekcie w programie CAD, całość układa się w kompletny model robota. Każdy element, taki jak czujniki, silniki, płytki elektroniczne, zasilanie czy mechaniczne podzespoły, został umieszczony w odpowiednich miejscach zgodnie z wcześniejszymi założeniami konstrukcyjnymi. Dzięki użyciu odpowiednich narzędzi w programie CAD, możliwe było precyzyjne rozmieszczenie komponentów, zachowanie wymiarów oraz dopasowanie do dostępnej przestrzeni. Model CAD pozwala na wizualizację całego układu, umożliwiając dalszą weryfikację poprawności rozmieszczenia elementów oraz ich wzajemnej kompatybilności. Tak przygotowany model stanowi bazę do dalszych prac, w tym przygotowania do montażu i testów fizycznych robota.



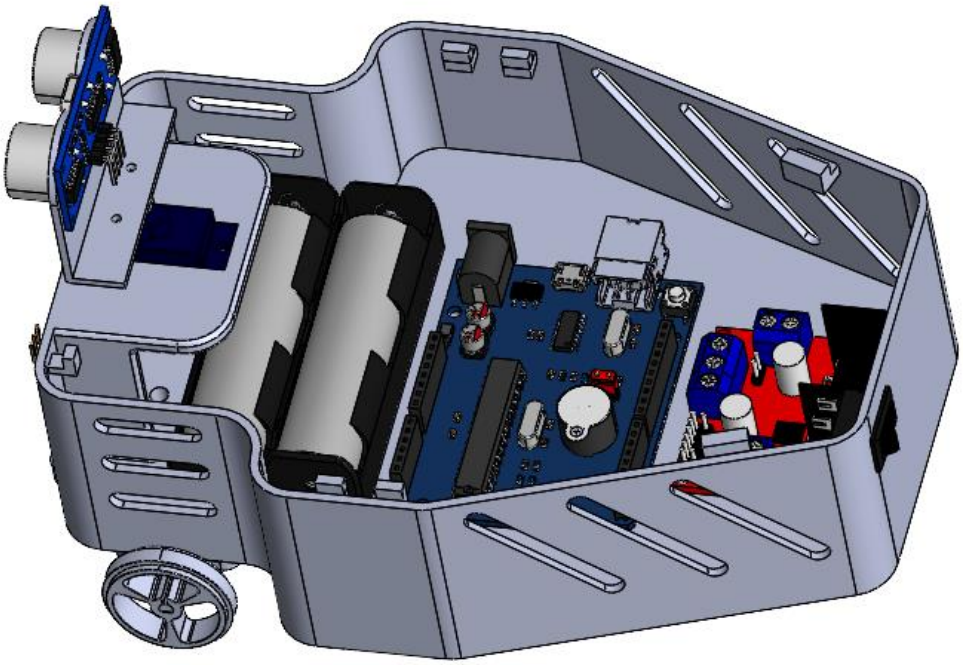
Rysunek 18.Model całości robota



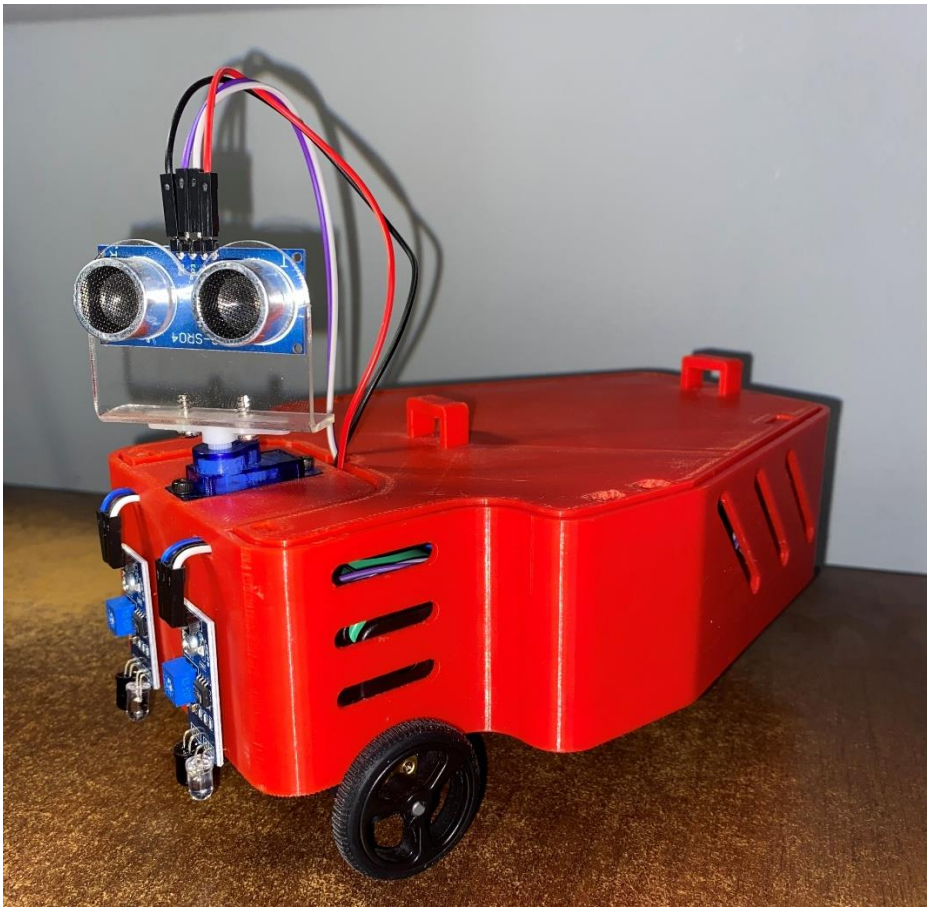
Rysunek 19.Model całości robota



Rysunek 20.Model całości robota



Rysunek 21.Model całości robota

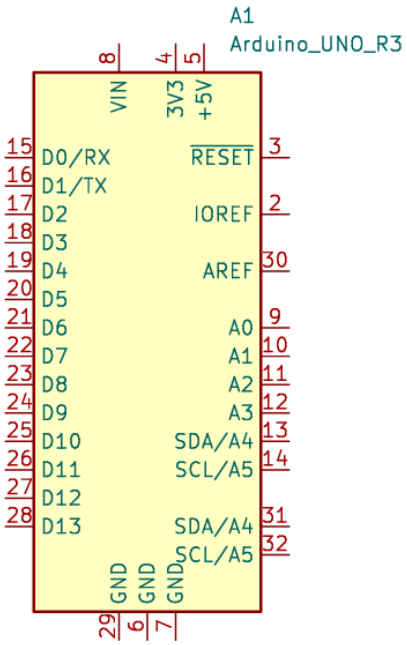


Rysunek 22.Gotowy robot

4.Opis układów elektrycznych i połączeń komponentów

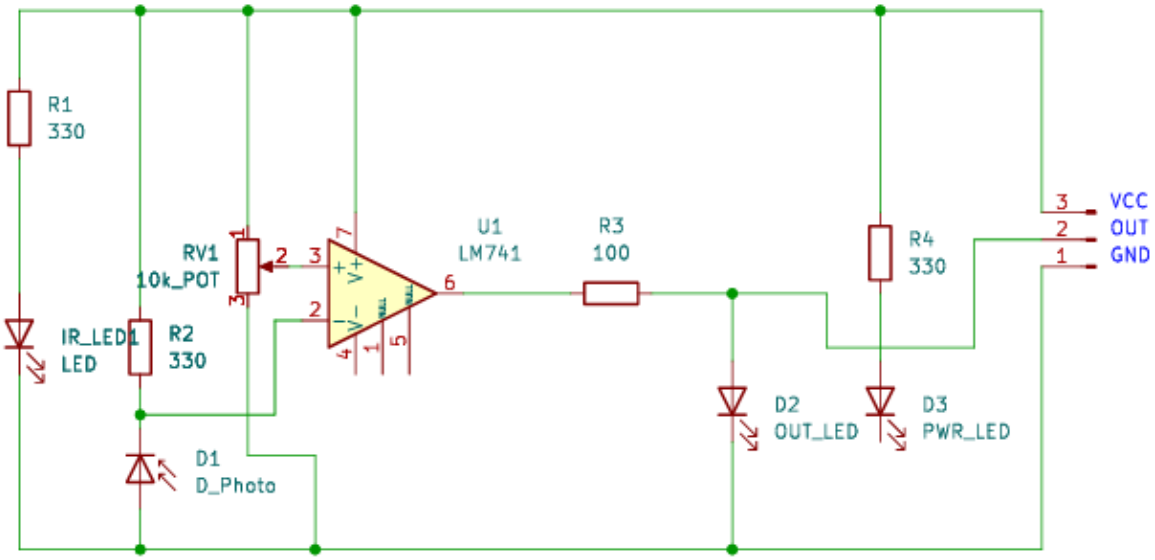
W tym rozdziale zaprezentowane zostały schematy elektryczne poszczególnych komponentów oraz całego robota, które stanowią kluczowy element projektu. Do stworzenia tych schematów wykorzystano oprogramowanie KiCad, pozwalające na profesjonalne projektowanie układów elektronicznych w sposób intuicyjny i efektywny.

Schematy te odzwierciedlają strukturę elektryczną robota, począwszy od modułu zasilania, poprzez układ sterowania, aż po systemy czujników i napędu.



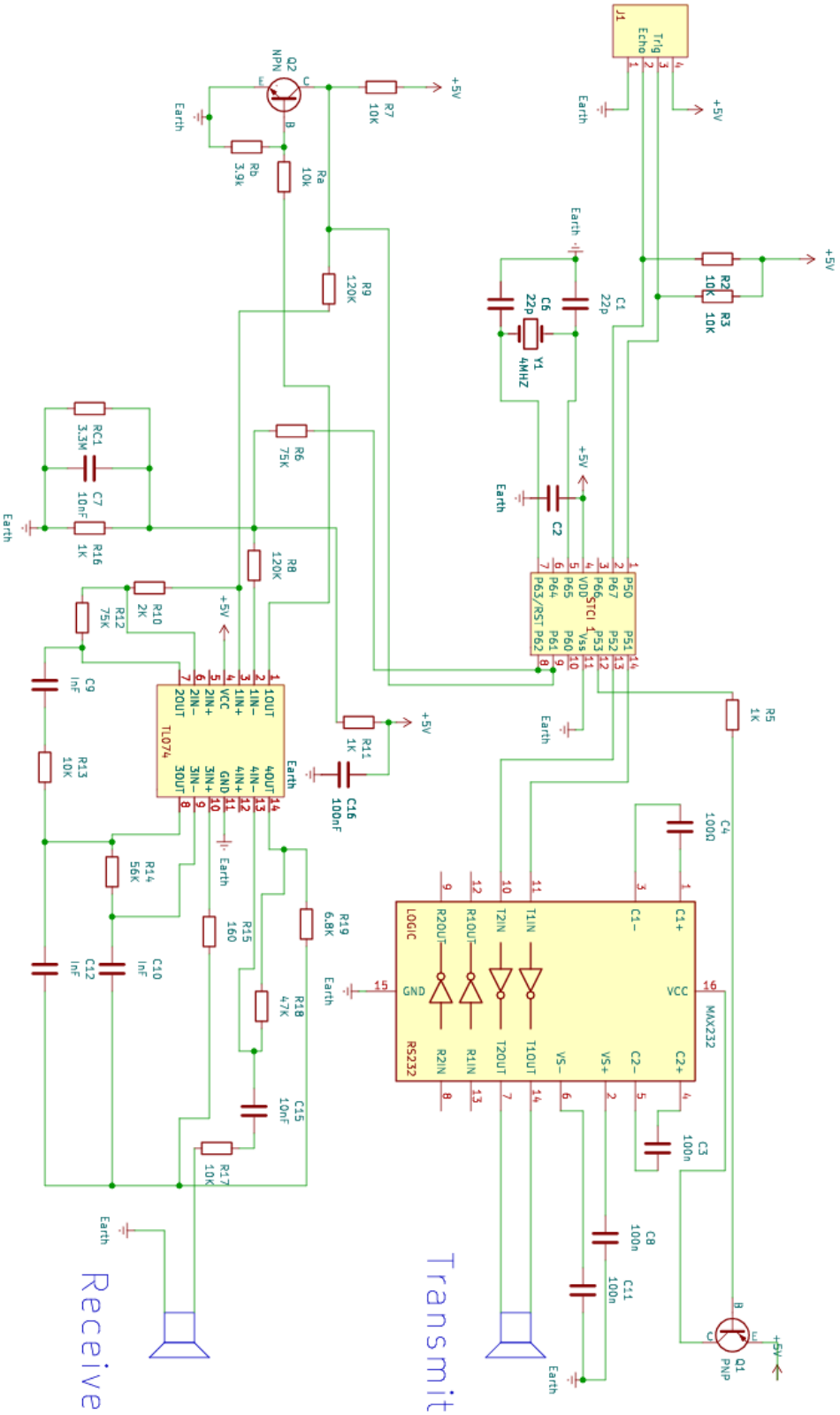
Rysunek 23.Arduino Uno Rev3

Arduino Uno Rev3 [Rysunek 23] stanowi centralny element układu sterującego robota. Mikrokontroler ten pełni funkcję "mózgu" robota, przetwarzając dane wejściowe z różnych czujników i podejmując decyzje dotyczące sterowania napędem. Posiada zestaw wejść i wyjść cyfrowych oraz analogowych, które umożliwiają łatwe połączenie z innymi komponentami. W schemacie widać, jak każdy pin mikrokontrolera jest połączony z odpowiednim czujnikiem, sterownikiem silników lub innym elementem robota. Dzięki temu Arduino Uno zapewnia komunikację pomiędzy poszczególnymi modułami oraz kontroluje ich synchronizację w czasie rzeczywistym.



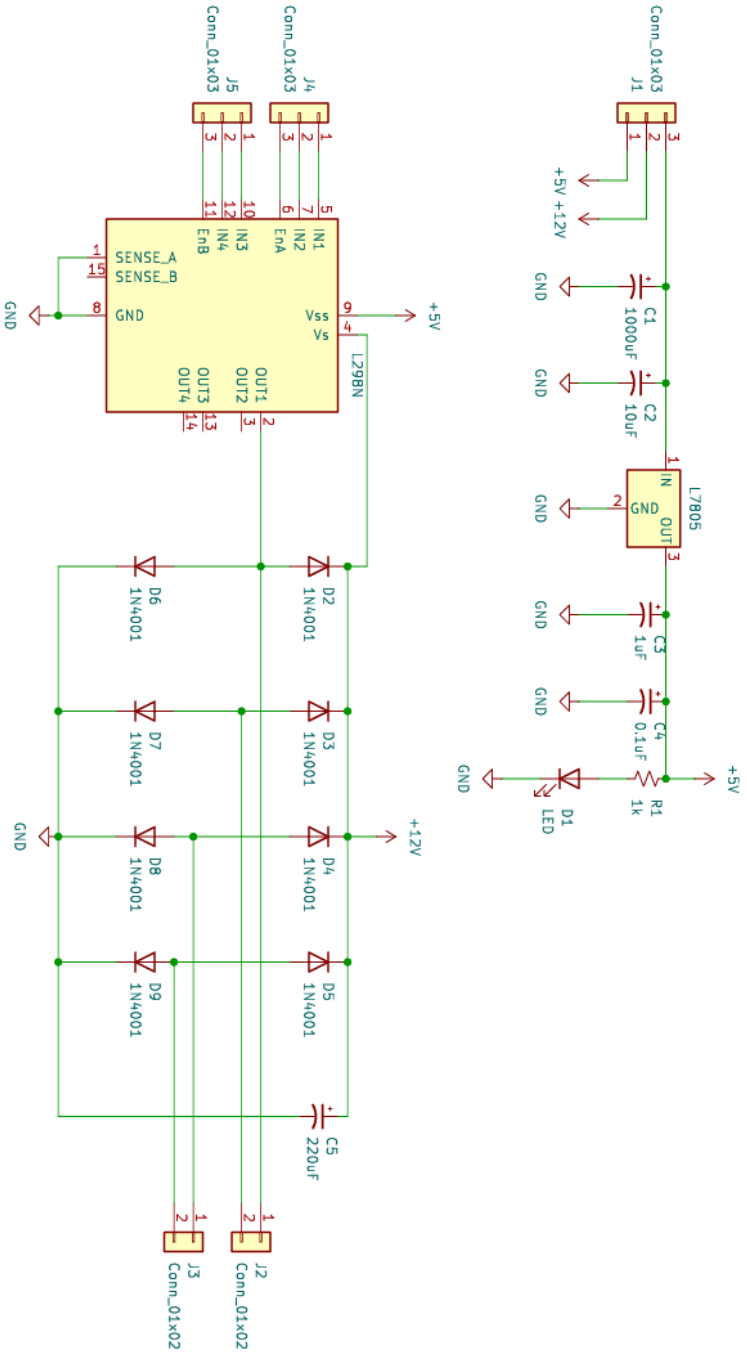
Rysunek 24.Czujnik podczerwieni

Czujnik podczerwieni [Rysunek 24] pełni kluczową rolę w systemie detekcji linii oraz przeszkód. W układzie elektrycznym widoczny jest sposób podłączenia czujnika do mikrokontrolera, który pozwala na monitorowanie zmian w odbitym sygnale podczerwonym. Czujnik ten wykrywa zmiany w odbiciu światła, co pozwala robotowi rozpoznać linię, po której ma się poruszać, a także omijać przeszkody. Na schemacie znajdują się elementy pomocnicze, takie jak rezystory, które regulują natężenie prądu, zapewniając stabilność sygnału oraz dokładność pomiarów.



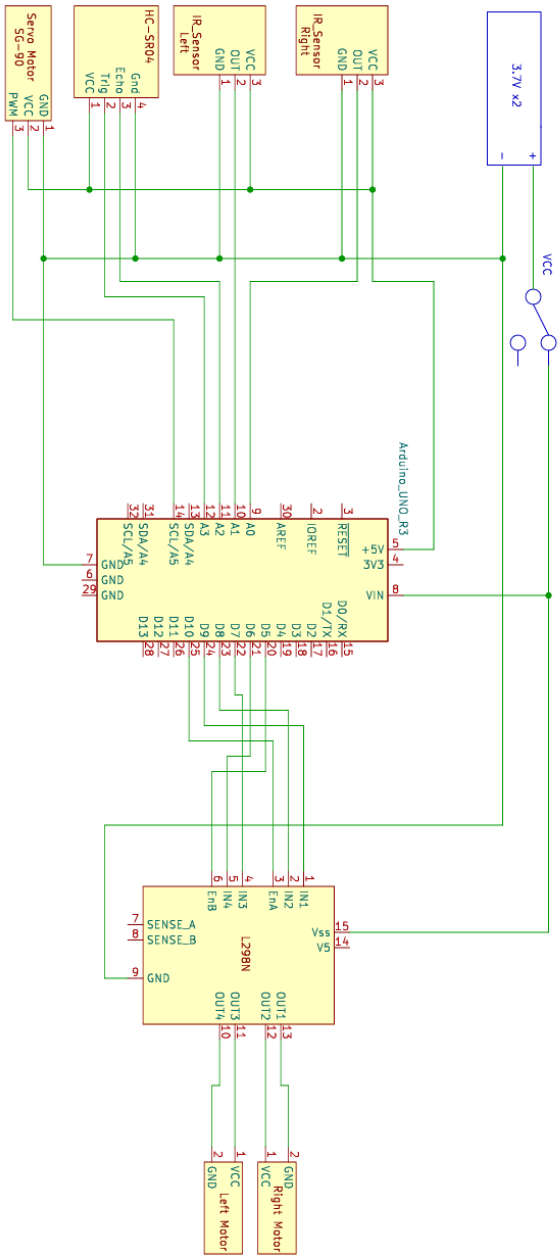
Rysunek 25.Czujnik ultradźwiękowy HC-SR04

Czujnik ultradźwiękowy HC-SR04 [Rysunek 25] odpowiada za pomiar odległości do przeszkód, co jest istotnym elementem systemu omijania przeszkód. W schemacie widać połączenia pinu trigger (wysyłanie sygnału ultradźwiękowego) i echo (odbiór sygnału powrotnego) z odpowiednimi pinami mikrokontrolera. Dzięki temu czujnik umożliwia precyzyjne określenie odległości, co pozwala robotowi wykrywać przeszkody w czasie rzeczywistym i podejmować odpowiednie manewry. Zastosowanie kondensatorów i rezystorów w układzie umożliwia stabilizację sygnałów i poprawia dokładność pomiarów.



Rysunek 26.L298N - dwukanałowy sterownik silników L298N

Sterownik silników L298N [Rysunek 26] jest odpowiedzialny za kontrolowanie napędu robota. Umożliwia on niezależną kontrolę prędkości oraz kierunku obrotów dwóch silników DC. W schemacie widać połączenia L298N z mikrokontrolerem, które pozwalają na wysyłanie sygnałów sterujących do silników. Dzięki temu sterownik umożliwia precyzyjne manewrowanie robotem, zarówno w przód, jak i w tył, a także pozwala na skręcanie w lewo lub w prawo. W układzie uwzględniono także elementy zabezpieczające, takie jak diody flyback, które chronią układ przed przepięciami generowanymi przez silniki podczas ich pracy.



Rysunek 27.Schemat elektryczny robota

5.Oprogramowanie sterujące robota:

5.1 Opis ogólny programu

Oprogramowanie sterujące robota zostało opracowane w środowisku programistycznym Arduino IDE , które umożliwia pisanie kodu, jego kompilację oraz wgrywanie na mikrokontroler. Program zrealizowany w języku C++ odpowiada za integrację komponentów sprzętowych, takich jak czujniki ultradźwiękowe, serwomechanizmy oraz silniki napędowe, zapewniając ich synchronizację w celu realizacji założonych funkcji robota. W rozdziale omówione zostaną podstawowe funkcje a całość kodu znajduje się w załącznikach [**9**]

Cel działania programu

- Śledzenie linii za pomocą czujników podczerwieni.
- Omijanie przeszkód wykrytych przez czujniki ultradźwiękowe.
- Zapewnienie stabilnego sterowania serwomechanizmem oraz silnikami napędowymi.

5.2 Struktura programu

Kod sterujący został podzielony na trzy główne sekcje

- Inicjalizacja systemu (funkcja setup())
  Odpowiada za konfigurację pinów, ustawienie trybu pracy czujników praz wstępną kalibrację serwomechanizmu
- Główna pętla programu (funkcja loop())
  Realizuje podstawową logike sterowania, w tym odczyt danych z czujników, podejmowanie decyzji na podstawie odległości od przeszkód oraz sterowanie ruchem robota
- Funkcje wspierające
  Obejmuje implementację szczegółowych operacji, takich jak sterowanie serwomechanizmem (moveServo()), pomiar odległości (readUltrasonic()), manewrowanie robotem (turnLeft(), turnRight(), moveForward(), itd.) oraz analizę danych wejściowych.

5.3 Opis kluczowych funkcji

Robot mobilny działa dzięki starannie zaprogramowanemu systemowi opartemu na mikrokontrolerze. Kluczowe funkcje wchodzące w skład programu sterującego umożliwiają analizę otoczenia, podejmowanie decyzji oraz realizację określonych manewrów. W tym podrozdziale zostaną szczegółowo opisane najważniejsze elementy kodu, które decydują o działaniu robota.

5.3.1 Funkcja setup()

Funkcja ta jest wykonywana jednokrotnie po uruchomieniu systemu. Odpowiada za

- Inicjalizację komunikacji szeregowej
- Konfigurację trybu pracy pinów mikrokontrolera
- Kalibrację serwomechanizmu w celu ustalenia zakresu ruchu

5.3.2 Funkcja loop()

Funkcja loop() jest wykonywana cyklicznie i zawiera główną logikę programu

1. Odczytuje odległość od przeszkód za pomocą funkcji readUltrasonic()
2. Reaguje na sygnały z czujników podczerwieni (IR_LEFT i IR_RIGHT)
3. Wykonuje odpowiednie manewry w zależności od sytuacji
    - Skręt w prawo (turnRight())
    - Skręt w lewo (turnLeft())
    - Analizę boków w celu omijania przeszkód (checkSides())

5.3.3 Funkcja readUltrasonic()

Odpowiada za pomiar odległości przy użyciu czujnika ultradźwiękowego. Funkcja wysyła impuls na pin ULTRASONIC_TRIGGER, mierzy czas powrotu sygnału na pinie ULTRASONIC_ECHO i przelicza go na odległość.

5.3.4 Funkcja checkSides()

Analizuje odległość z lewej i prawej strony robota, aby podjąć decyzję o kierunku ruchu w przypadku przeszkody przed nim.

6.Badania eksperymentalne

Celem przeprowadzonych badań jest określenie wpływu różnych parametrów sterowania robotem na jego skuteczność w omijaniu przeszkód. W szczególności skupiono się na.

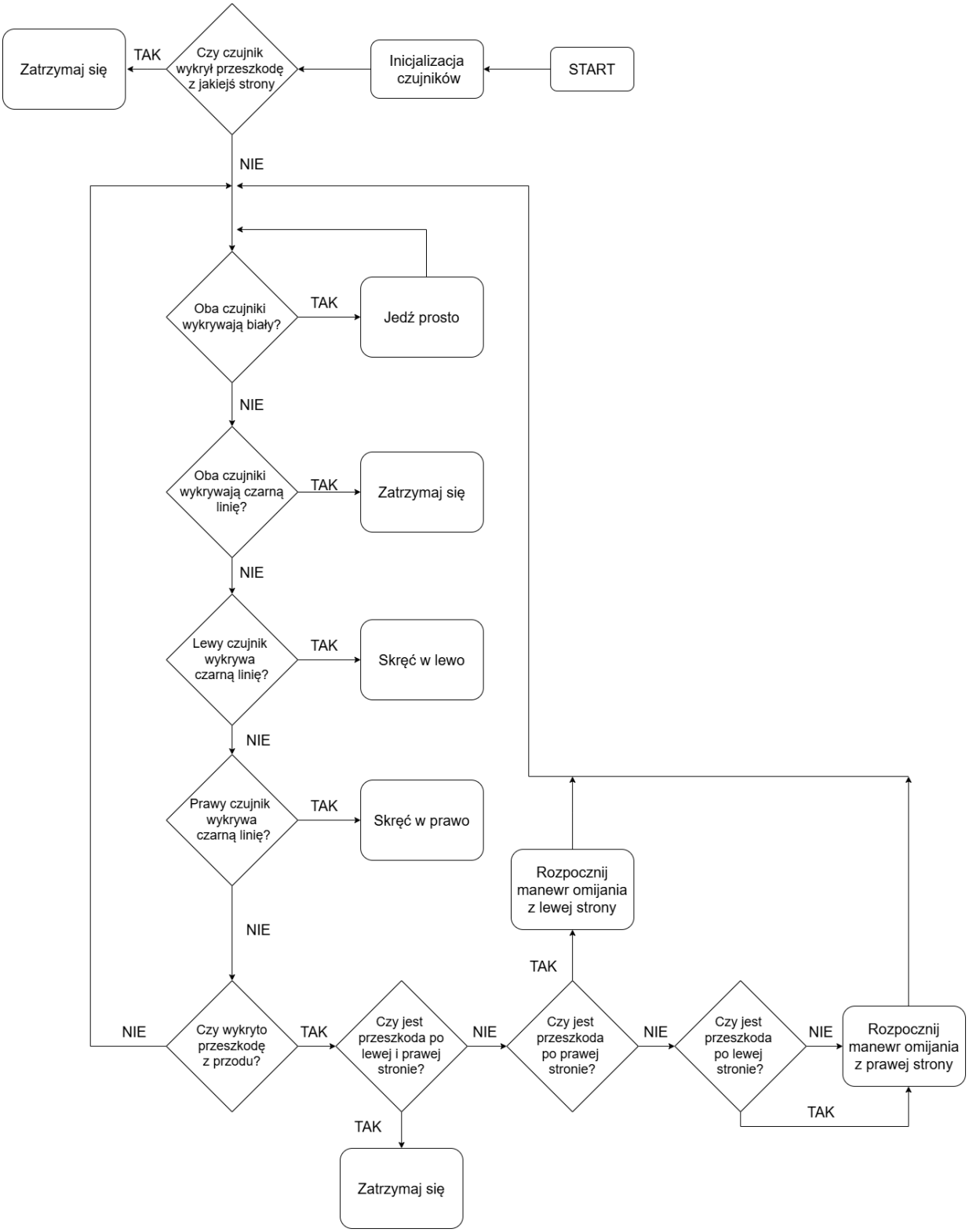
- wartości PWM (prędkości silników)
- minimalnym kącie skrętu potrzebnym do ominięcia przeszkody
- czasie wykonania manewru (delay)
- oraz czasie ignorowania czujnika po wykonaniu manewru

Badania te mają na celu optymalizację zachowań robota, aby był w stanie efektywnie omijać przeszkody w różnych warunkach.

6.1 Parametry Badawcze

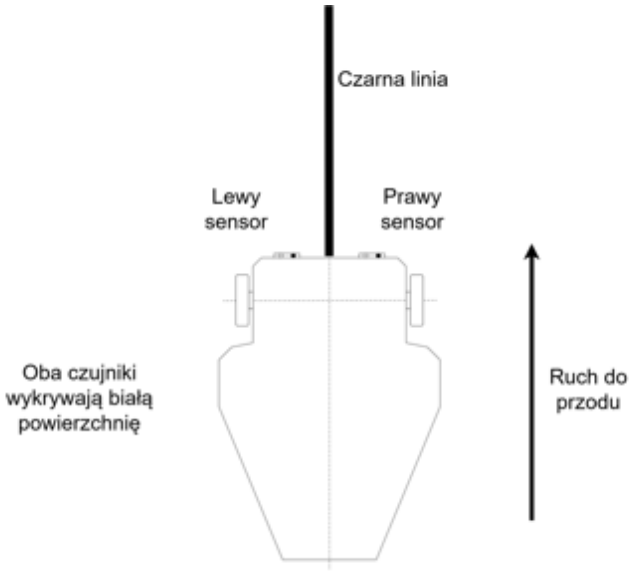
- **PWM (wartość sygnału sterującego silniki)**: Wartość PWM określa prędkość obrotową kół robota. Im większe PWM, tym szybciej robot się porusza, co wpływa na dynamikę manewru.
- **Minimalny kąt skrętu**: To kąt, o który robot musi się obrócić, aby ominąć przeszkodę. Jest to wartość, którą obliczamy na podstawie szerokości przeszkody i odległości detekcji.
- **Wartość opóźnienia (delay)**: Czas, w którym robot wykonuje manewr skrętu, jest obliczany na podstawie kąta skrętu oraz prędkości kół (zależnej od PWM).
- **Czas ignorowania czujnika**: Po wykonaniu manewru, czujnik ultradźwiękowy może chwilowo ignorować nowe przeszkody, aby uniknąć fałszywych odczytów podczas skrętu.

6.2 Schemat blokowy algorytmu



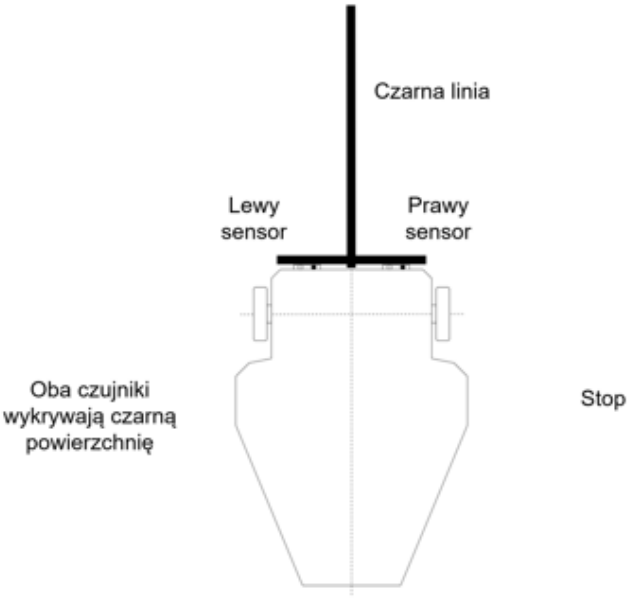
Rysunek 28. Schemat blokowy algorytmu

6.3 Sposób reakcji robota na linię

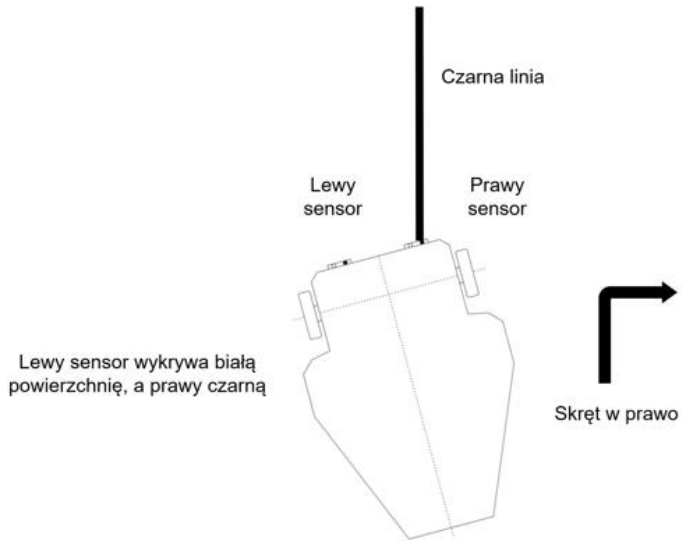


Rysunek 29. Reakcja robota na brak wykrycia linii

Gdy oba czujniki znajdują się nad białą powierzchnią, robot porusza się do przodu. [Rysunek 29] Natomiast gdy oba czujniki znajdują się nad czarną powierzchnią, robot zatrzymuje się. [Rysunek 30] Kluczową rolę odgrywa tutaj pozycja czujników, która decyduje o tym, czy robot będzie kontynuował ruch, czy się zatrzyma.

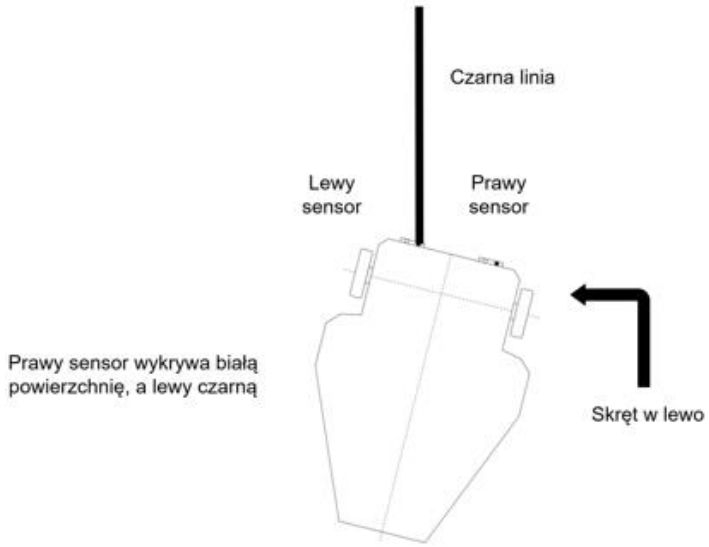


Rysunek 30.Wykrycie linii przez oba czujniki



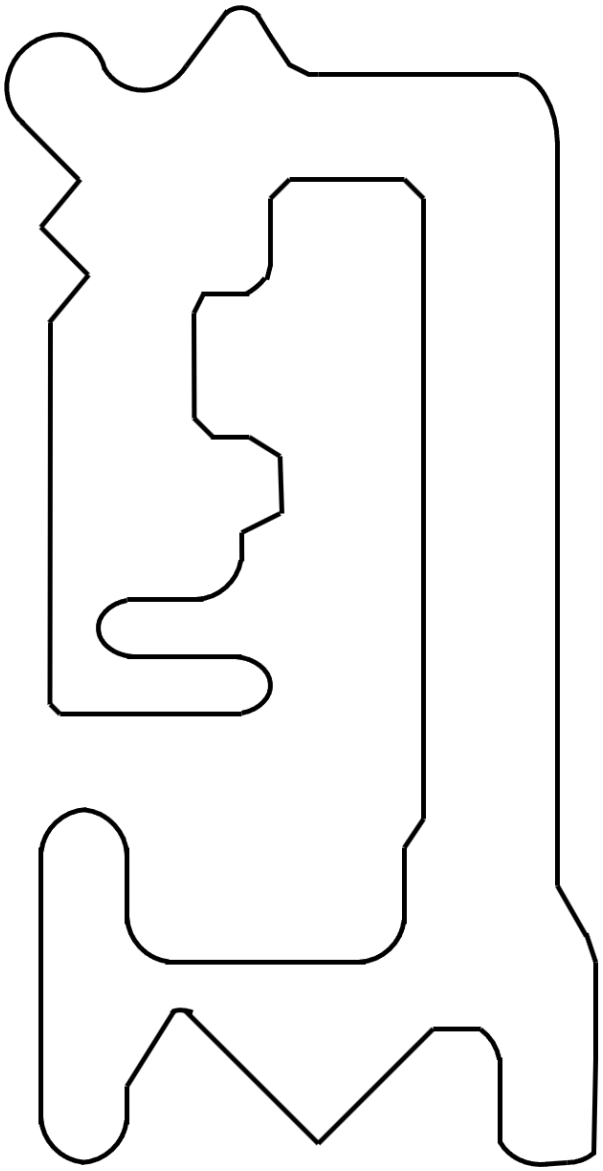
Rysunek 31.Wykrycie linii przez prawy czujnik

W sytuacji, gdy lewy czujnik wykryje czarną linię [Rysunek 32], a prawy pozostaje nad białą powierzchnią, robot powinien skręcić w lewo. Analogicznie, gdy prawy czujnik wykryje czarną linię [Rysunek 31], a lewy znajduje się nad białą powierzchnią, robot powinien skręcić w prawo. Niezależnie od sytuacji, w momencie wykrycia czarnej linii robot musi zatrzymać ruch.



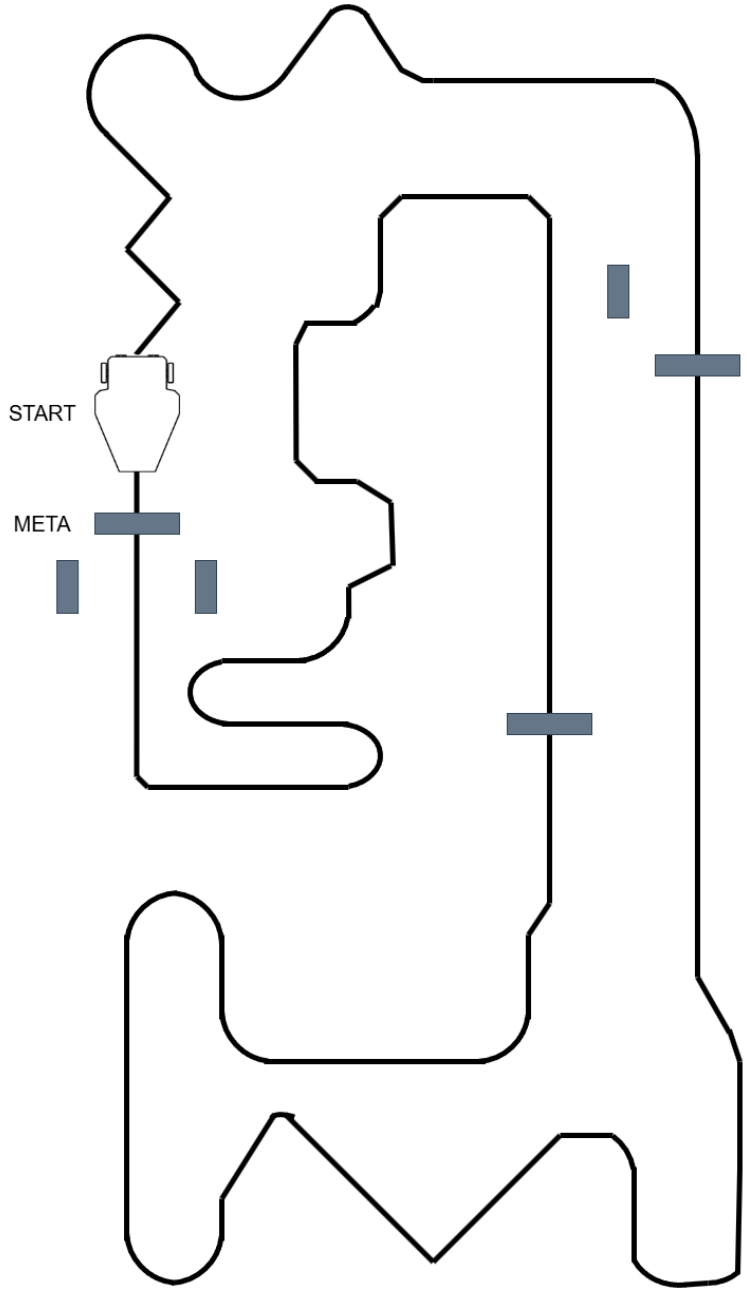
Rysunek 32.Wykrycie linii przez lewy czujnik

6.4 Analizowana trasa robota wraz z przeszkodami



Rysunek 33.Trasa robota

Zaprojektowana trasa [Rysunek 33] dla robota typu line follower zawiera różnorodne zakręty – zarówno o małym promieniu, jak i łagodniejsze – w celu przetestowania funkcjonalności systemu wykrywania linii. W rzeczywistości trasa została wykonana z trzech białych płyt o wymiarach 1200x600 mm, połączonych ze sobą. Samą linię trasy wykonano z czarnej taśmy o szerokości około 17 mm. Projekt trasy został przemyślany w taki sposób, aby szerokość linii mieściła się w zakresie odległości pomiędzy dwoma czujnikami podczerwieni zamontowanymi w robocie, co umożliwia precyzyjne śledzenie toru.



Rysunek 34.Trasa robota z przeszkodami

Trasa [Rysunek 34] została wzbogacona o przeszkody rozmieszczone w strategicznych miejscach. Ich zadaniem jest przetestowanie systemu wykrywania oraz omijania obiektów przez robota. Przeszkody wprowadzono w formie prostokątnych elementów, które symulują różne przeszkody napotykane na trasie. Robot musi nie tylko podążać za linią, ale także odpowiednio reagować na przeszkody, co sprawdza jego zdolność do ich wykrywania i skutecznego omijania, bez opuszczania wyznaczonej trasy.

6.5 Metodologia badania

Krok 1: Obliczenie Prędkości Obrotowej (RPM) na Podstawie Wartości PWM

Pierwszym krokiem w badaniu jest obliczenie prędkości obrotowej silnika (RPM) na podstawie wartości sygnału PWM, którą ustalamy dla silników robota. Wartość PWM określa poziom wypełnienia sygnału sterującego pracą silników, przy czym maksymalne wypełnienie (255) odpowiada maksymalnej prędkości obrotowej silników, czyli ich pełnej wydajności.

W celu obliczenia prędkości obrotowej stosujemy następujący wzór:

$$RPM = \frac{PWM}{MaxPWM} \times MaxRPM \tag{3}$$

Gdzie:

- PWM – wartość sygnału PWM, którą ustalamy (zakres od 0 do 255)
- MaxPWM – maksymalna wartość PWM, czyli 255
- MaxRPM – maksymalna prędkość obrotowa silników przy pełnym wypełnieniu PWM (w RPM)

Wyjaśnienie wzoru:

- RPM (Revolutions Per Minute) to liczba obrotów koła na minutę, która jest wprost proporcjonalna do wartości PWM. Im wyższa wartość PWM, tym większa prędkość obrotowa silnika.
- Przy pełnym wypełnieniu sygnału PWM, czyli gdy PWM = 255, silnik osiąga maksymalną prędkość obrotową, która wynosi MaxRPM. Z kolei gdy wartość PWM jest mniejsza, RPM zmniejsza się proporcjonalnie do zmiany wartości PWM.

Zakładając, że:

- MaxPWM = 255 (maksymalna wartość PWM)
- MaxRPM = 120 [RPM] (maksymalna prędkość obrotowa silnika)
- PWM = 170 (wartość PWM używana do regulacji prędkości)

Obliczamy prędkość obrotową (3):

$$RPM = \frac{170}{255} \times 120 = 80RPM \quad (4)$$

Dla wartości PWM równej 170, prędkość obrotowa silnika wynosi 80 RPM, co stanowi około 67% maksymalnej prędkości obrotowej. Tę wartość RPM będziemy wykorzystywać w dalszych krokach obliczeniowych, szczególnie przy wyznaczaniu parametrów skrętu i opóźnień czasowych.

Krok 2: Obliczenie optymalnego kąta skrętu robota

Kolejnym etapem badania, po obliczeniu prędkości obrotowej silników (RPM), jest wyznaczenie minimalnego kąta skrętu robota. Minimalny kąt skrętu to taki, który pozwala robotowi bezpiecznie ominąć przeszkodę wykrytą przez czujniki, zachowując określoną odległość bezpieczeństwa od przeszkody.

Aby obliczyć minimalny kąt skrętu, musimy uwzględnić kilka parametrów:

- Szerokość robota,
- Szerokość przeszkody,
- Odległość od przeszkody, jaką czujnik ultradźwiękowy wykrywa przed manewrem,
- Dodatkowa odległość bezpieczeństwa, jaką chcemy zachować (np. 2-3 cm od krawędzi robota).

Optymalny kąt skrętu można obliczyć na podstawie poniższych zależności geometrycznych i kinematycznych, wynikających z zachowania robota podczas manewru skrętu. Obliczamy kąt, przy którym robot obróci się wystarczająco, aby jego boki ominęły przeszkodę, biorąc pod uwagę szerokość przeszkody i jej położenie względem robota.

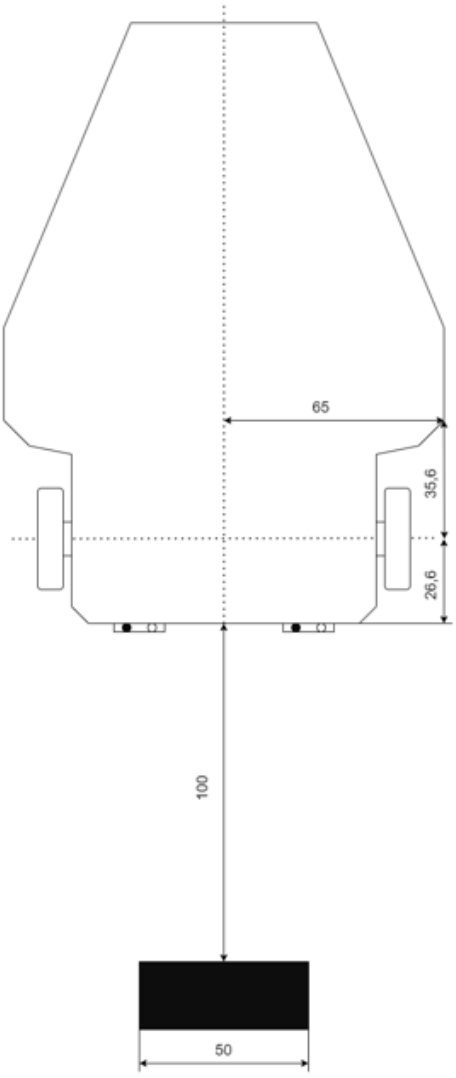
Zakładamy:

- Szerokość przeszkody (umieszczona centralnie),
- Odległość wykrycia przeszkody
- Bezpieczny dystans (2-3cm)

Z tych parametrów możemy wyznaczyć minimalny kąt skrętu, który pozwoli na bezpieczne ominięcie przeszkody. W tym przypadku, stosując relacje geometryczne, możemy obliczyć kąt

skrętu θ, który robot musi wykonać, aby przeszkoda znajdowała się co najmniej 3 cm od jego krawędzi.

Kąt ten będzie uzależniony od odległości czujnika, szerokości przeszkody oraz szerokości i długości robota.

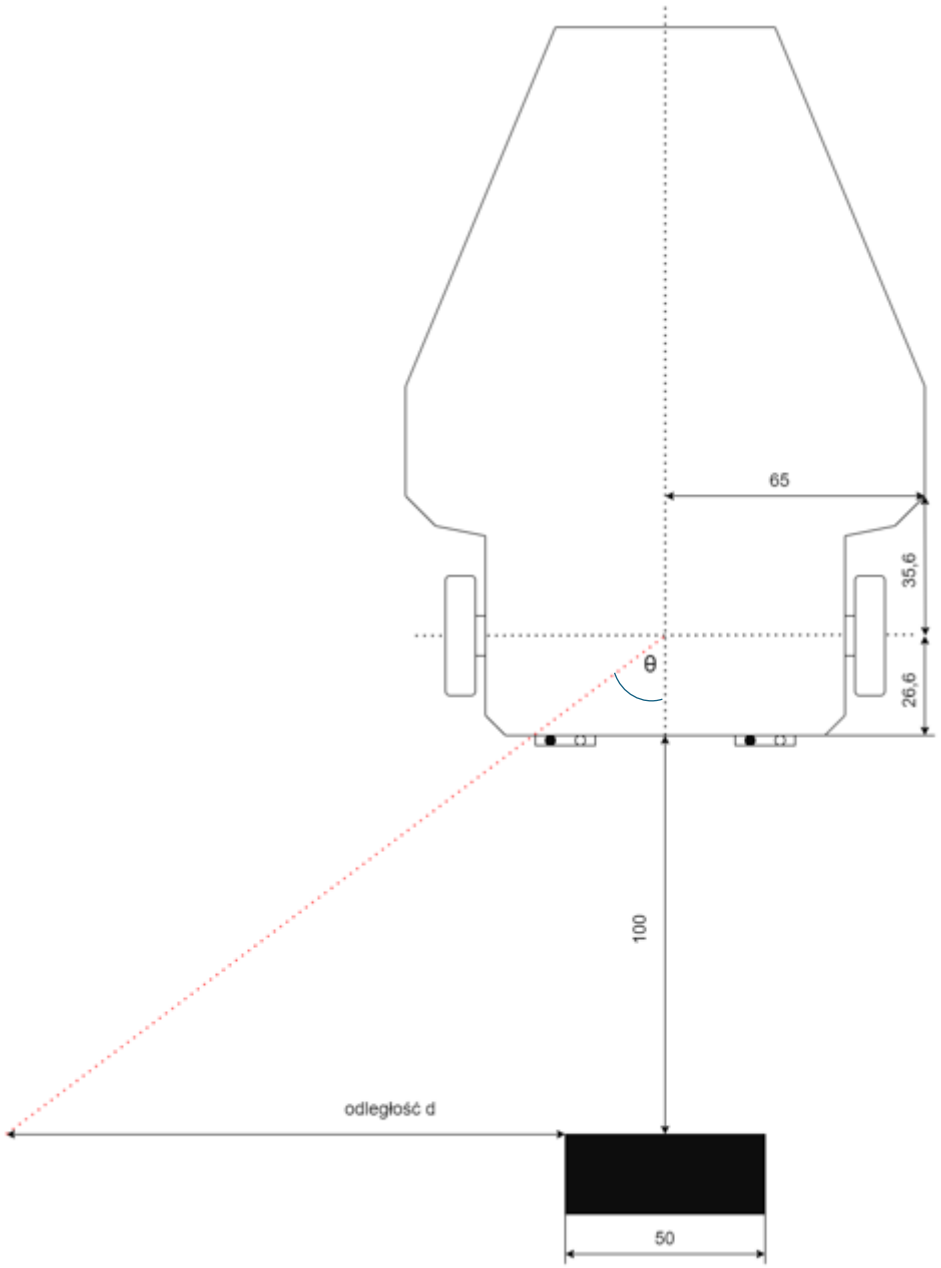


Rysunek 35.Wykrycie przeszkody na trasie

Robot, podczas manewru omijania przeszkody, wykorzystuje napęd różnicowy [28].Oznacza to, że jedno koło obraca się w przeciwnym kierunku na drugie, co sprawia, że środek osi robota podczas skrętu pozostaje w tym samym miejscu. Po uwzględnieniu kluczowych wymiarów robota oraz odległości od przeszkody i jej szerokości, możemy przystąpić do przeprowadzenia odpowiednich obliczeń mających na celu optymalizację manewru.

Odległość *d,* dobierana eksperymentalnie, służy do wyznaczenia punktu, z którego zostanie poprowadzona linia aż do środka osi robota. Celem tego zabiegu jest określenie optymalnego kąta skrętu θ, który zagwarantuje zachowanie bezpiecznej odległości najbardziej wysuniętego

boku robota od przeszkody. Dzięki temu manewr ominięcia przeszkody będzie realizowany w sposób bezpieczny i precyzyjny.



Rysunek 36.Detekcja przeszkody z kątem obrotu

Znając założoną wartość d, szerokość przeszkody oraz odległość od niej, jesteśmy w stanie obliczyć kąt θ obrotu robota. Te parametry pozwalają precyzyjnie określić, pod jakim kątem robot powinien się obrócić, aby bezpiecznie ominąć przeszkodę, zachowując wymaganą odległość od jej krawędzi.

- Założona odległość d – 140 [mm]
- Szerokość przeszkody – 50 [mm]
- Odległość od przeszkody – 100 mm]

Pierwszym krokiem w naszym procesie będzie obliczenie tangensa kąta obrotu robota:

$$tg\theta = \frac{d + \frac{1}{2} szerokości\ przeszkody}{odległość\ od\ przeszkody + 26.6} \tag{5}$$

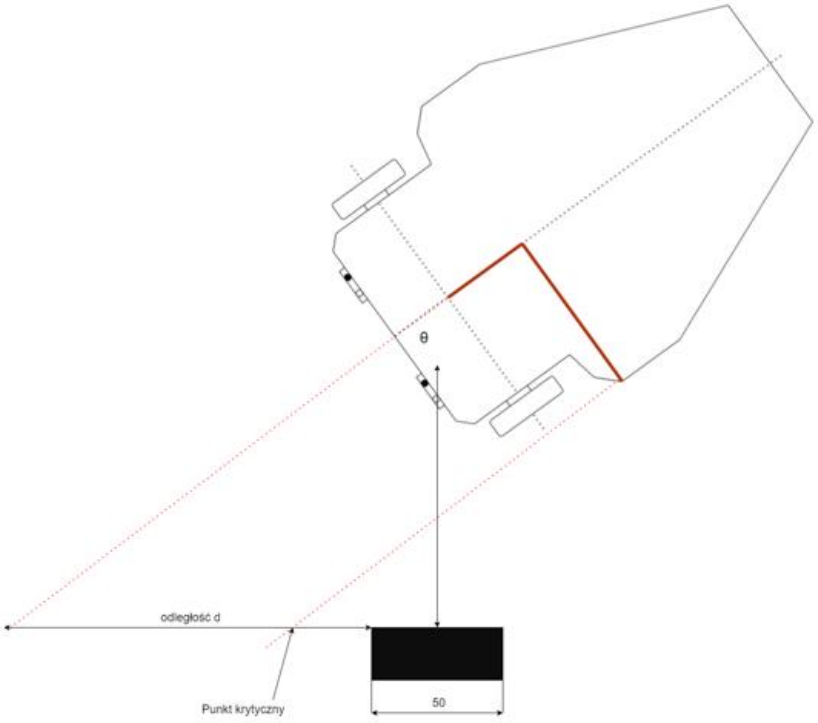
W obliczeniach bierzemy pod uwagę połowę szerokości przeszkody, aby uzupełnić odległość do środka osi robota, oraz długość 26.6[mm] aby uwzględnić dystans od początku robota aż do osi kół. Dzięki temu jesteśmy w stanie obliczyć tangens kąta.

$$tg\theta = \frac{140 + 25}{126.6} \tag{6}$$

$$tg\theta = 1{,}3 \tag{7}$$

$$\theta \approx 52{,}5° \tag{8}$$

Teraz, mając wszystkie dane, możemy przeprowadzić odpowiednie odczyty.
Jesteśmy w stanie poprowadzić linię od najbardziej wysuniętego punktu robota równoległą do kierunku osi robota. Dzięki temu możemy odczytać odległość punktu krytycznego od przeszkody w płaszczyźnie dwuwymiarowej, uwzględniając kąt θ oraz odległość *d*.



Rysunek 37.Robot po zakończeniu manewru obrotu

W ten sposób możemy dokładnie określić, jak daleko punkt krytyczny znajduje się od obiektu Po zmierzeniu tego dystansu na osi, możemy odczytać, że dystans, który tworzy się od poprowadzenia linii równoległej do linii oznaczającej początek przeszkody, wynosi 3,5 cm, co zgadza się z założeniami bezpieczeństwa.

Ten pomiar daje nam dokładną odległość, w jakiej punkt krytyczny znajduje się od przeszkody, co jest kluczowe do dalszych analiz lub działań związanych z planowaniem ruchu robota.

Krok 3: Obliczenie czasu skrętu robota dla wyliczonego kąta

1. Obliczenie prędkości liniowej V

Prędkość liniową V dla jednego koła obliczamy według wzoru

$$V = \frac{\pi \times D \times RPM}{60} \tag{9}$$

Gdzie:

- π stała matematyczna, równa 3.14159
- D to średnica koła (w metrach)
- RPM to prędkość obrotowa koła w obrotach na minutę
- 60 to liczba sekund w minucie, ponieważ chcemy uzyskać prędkość w metrach na sekundę.

Dla przykładowych parametrów

- D=0.033 [m] (średnica koła)
- RPM=80 [RPM] (prędkość obrotowa)

Podstawiamy do wzoru (9)

$$V = \frac{\pi \times 0.033 \times 80}{60} \approx 0.1382 \left[\frac{m}{s}\right] \tag{10}$$

2. Obliczenie przebytej odległości d przez oba koła

Odległość d przebyta przez oba koła w trakcie skrętu jest wyrażona wzorem

$$d = \frac{\theta \times L}{2} \tag{11}$$

Gdzie:

- θ to zadany kąt skrętu w radianach
- L to rozstaw osi robota (odległość między kołami, w metrach),
- 2 w mianowniku uwzględnia, że oba koła współpracują, dzieląc pracę po równo.

Przy obliczeniu kąta w radianach używamy konwersji

$$\theta_{rad} = \theta_{deg} \times \frac{\pi}{180} \tag{12}$$

Dla zadanych wartości

- θ_{deg}= 52.5 [°] (kąt skrętu)
- L=0.106 [m] (rozstaw osi)

Konwertujemy kąt na radiany korzystając ze wzrou (12)

$$\theta_{rad} = 52.5 \times \frac{\pi}{180} \approx 0.916\, rad \tag{13}$$

Następnie obliczamy d korzystając ze wzrou (11)

$$d = \frac{0.916 \times 0.106}{2} \approx 0.0485\, m \tag{14}$$

3. Obliczenie czasu skrętu t

$$t = \frac{\mathrm{d}}{V} \tag{15}$$

Czas skrętu t jest obliczany na podstawie przebytej odległości *d* i prędkości liniowej V
Dla zadanych wartości

- d=0.0485 [m]
- V=0.1382 [m/s]

$$t = \frac{0.0485}{0.1392} \approx 0.35\ sekund \tag{16}$$

4. Konwersja czasu na milisekundy

Czas skrętu w milisekundach

$$t_{ms} = t \times 1000 \tag{17}$$

$$t_{ms} = 0.35 \times 1000 \approx 350\ ms \tag{18}$$

Dla zadanego kąta 52.5° , średnicy kół 0.033 m, rozstawu osi 0.106 m i prędkości obrotowej 80 RPM, obliczony czas skrętu wynosi około 350 ms, który będziemy wstawiać do algorytmu.

Krok 4: Obliczenie dystansu oraz czasu na pokonanie drogi od jednego punktu do drugiego

Rysunek 38.Robot po obrocie – parametry do obliczenia drogi do kolejnej pozycji

Po wyliczeniu kąta i obróceniu robota o ten kąt, możemy przypisać układ współrzędnych do obróconego robota, co pozwoli w prosty sposób zmierzyć odległości *a* i *b*. Dzięki temu precyzyjnie określimy relatywną pozycję robota względem przeszkody. Następnie, korzystając z twierdzenia Pitagorasa, obliczymy odległość *c*, która dokładnie wskaże, do którego punktu robot będzie musiał dojechać względem badanego punktu.

Obliczanie długości c

- a=130 [mm]
- b=98 [mm]

$$c = \sqrt{a^2 + b^2} \tag{19}$$

$$c = \sqrt{130^2 + 98^2} \approx 162.8 \tag{20}$$

Znając wartość obliczonej drogi *c*, jesteśmy w stanie wyznaczyć czas potrzebny na jej pokonanie, biorąc pod uwagę wcześniej obliczoną wartość obrotów na minutę (RPM) kół robota.

1. Obliczenie czasu *t* potrzebnego na pokonanie drogi c

$$t = \frac{c}{V} \tag{21}$$

- c=0.1628 [m]
- V=0.138 [m/s]

$$t = \frac{0.1628}{0.138} \approx 1.18\ sekundy \tag{22}$$

2. Konwersja czasu na milisekundy

$$t_{ms} = t \times 1000 \tag{23}$$

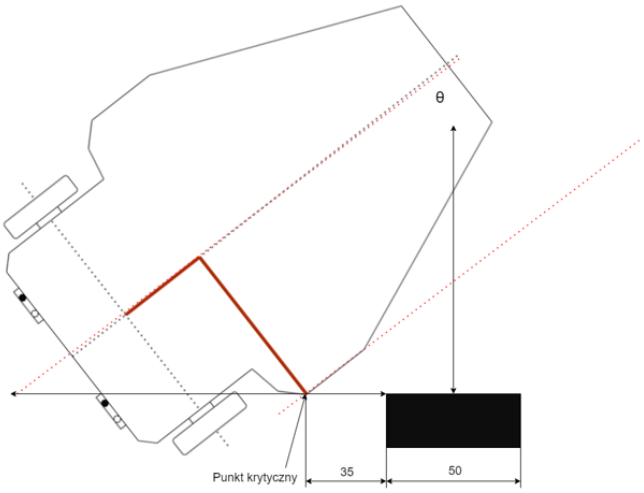
- t=1.18 [s]

$$t_{ms} = 1.18 \times 1000 \approx 1180ms \tag{24}$$

Czas potrzebny do pokonania odległości 0.1628m , przy prędkości obrotowej kół 80 RPPM i średnicy kół 0.033m , wynosi około 1180 ms.

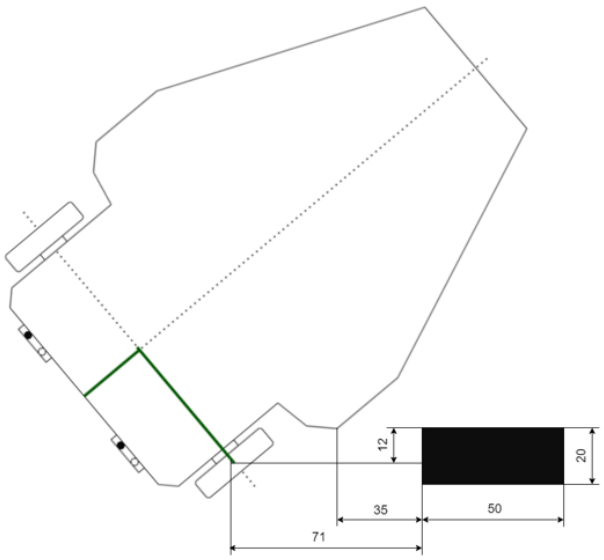
Krok 5: Obliczenie czasu skrętu oraz kąta robota do ustabilizowania się na równi z przeszkodą

W tej części pracy skoncentrujemy się na obliczeniu czasu skrętu robota, który ma na celu ustabilizowanie osi kół równolegle do osi przeszkody. W odróżnieniu od pierwszego manewru omijania, w którym zastosowaliśmy napęd różnicowy – polegający na obracaniu jednego koła w jedną stronę, a drugiego w przeciwną – w tym przypadku algorytm będzie wykorzystywał technikę skręcania, w której jedno koło obraca się, podczas gdy drugie pozostaje w miejscu.



Rysunek 39.Pozycja robota po zakończeniu manewru jazdy na wprost

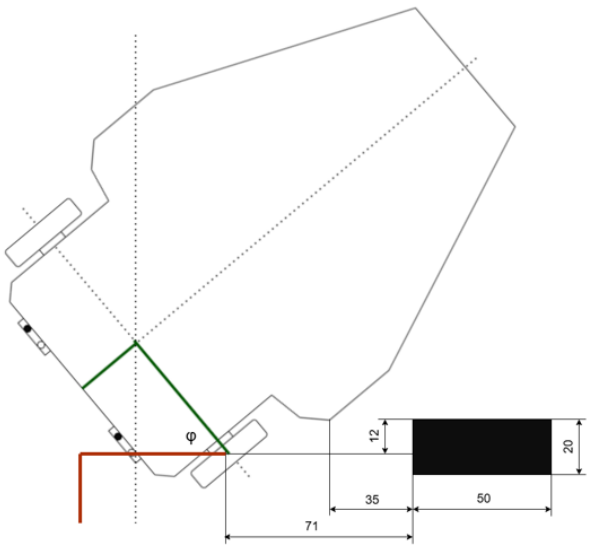
Pierwszym krokiem, który podejmiemy, będzie określenie pozycji robota względem przeszkody, a dokładniej określenie pozycji środka koła bliższego przeszkodzie. W tym celu zrzutujemy wymiary robota na układ współrzędnych, co pozwoli na precyzyjne ustalenie odległości między środkiem tego koła a przeszkodą.



Rysunek 40.Pozycja robota i odległość od środka koła do linii wzdłuż przeszkody

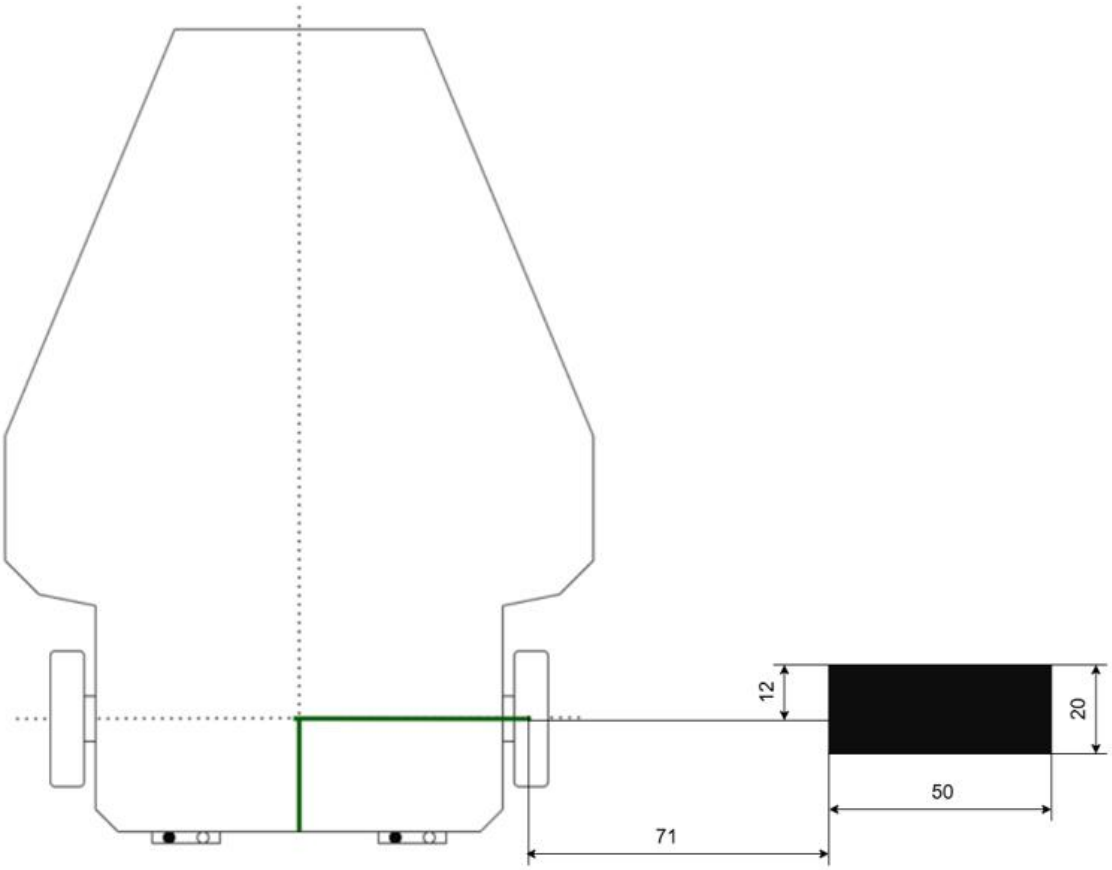
Mając te dane, znamy dokładną odległość środka bliższego koła robota od przeszkody, a także możemy określić, jaką część przeszkody została teoretycznie przejechana. Dzięki temu jesteśmy w stanie obliczyć orientację robota w przestrzeni po wykonaniu manewru skrętu. Zastosowanie tych informacji pozwala również na dokładne oszacowanie czasu potrzebnego na wykonanie skrętu, co jest kluczowe dla dalszej kontroli ruchu robota i jego interakcji z otoczeniem.

Kąt skrętu musi być taki sam, jaki wyliczyliśmy na początku (8) podczas pierwszego manewru, aby zagwarantować, że oś robota będzie równoległa do osi przeszkody.



Rysunek 41.Pozycja robota względem przeszkody z wyznaczonym kątem skrętu

Zakładając, że robot wykonuje manewr, w którym jedno koło się obraca, a drugie pozostaje nieruchome, oraz znając wyliczony kąt obrotu, możemy zrzutować połowę osi robota oraz odległości od jej środka do przodu robota. Rzutowany kąt przesuwamy wzdłuż okręgu, którego promieniem jest odległość od środka osi robota do punktu styku koła nieruchomego z podłożem. W ten sposób można precyzyjnie określić trajektorię ruchu robota i jego pozycję po zakończeniu manewru skrętu.



Rysunek 42.Pozycja robota po wykonaniu skrętu o kąt 52.5°

W ten sposób zrzutowaliśmy i określiliśmy orientację robota w przestrzeni. Teraz możemy obliczyć czas, jaki będzie potrzebny, aby robot wykonał ten manewr. Ustalając prędkość obrotową robota oraz odległość, jaką musi pokonać podczas skrętu, możemy dokładnie oszacować czas niezbędny do zakończenia manewru.

1. Obliczenie kąta skrętu φ w radianach

Wzór na przeliczenie kąta z stopni na radiany

$$\varphi = \frac{\pi \times \theta}{180} \tag{25}$$

Podstawiając wartość kąta θ = 52.5°

$$\varphi = \frac{\pi \times 52.5}{180} = 0.916rad \tag{26}$$

2. Odległość przebyta przez koło d (w metrach)

Wzór na odległość przebytą przez koło, aby osiągnąć zadany kąt

$$d = \varphi \times \mathrm{L} \tag{27}$$

Gdzie:

L– rozstaw osi (odległość między kołami) [m]

φ-kąt skrętu [rad]

$$d = 0.916 \times 0.106 = 0.097\mathrm{m} \tag{28}$$

3. Czas potrzebny na pokonanie odległości d (w sekundach)

$$t = \frac{d}{V} \tag{29}$$

$$t = \frac{0.097}{0.138} \approx 0.702s \tag{30}$$

4. Czas w milisekundach

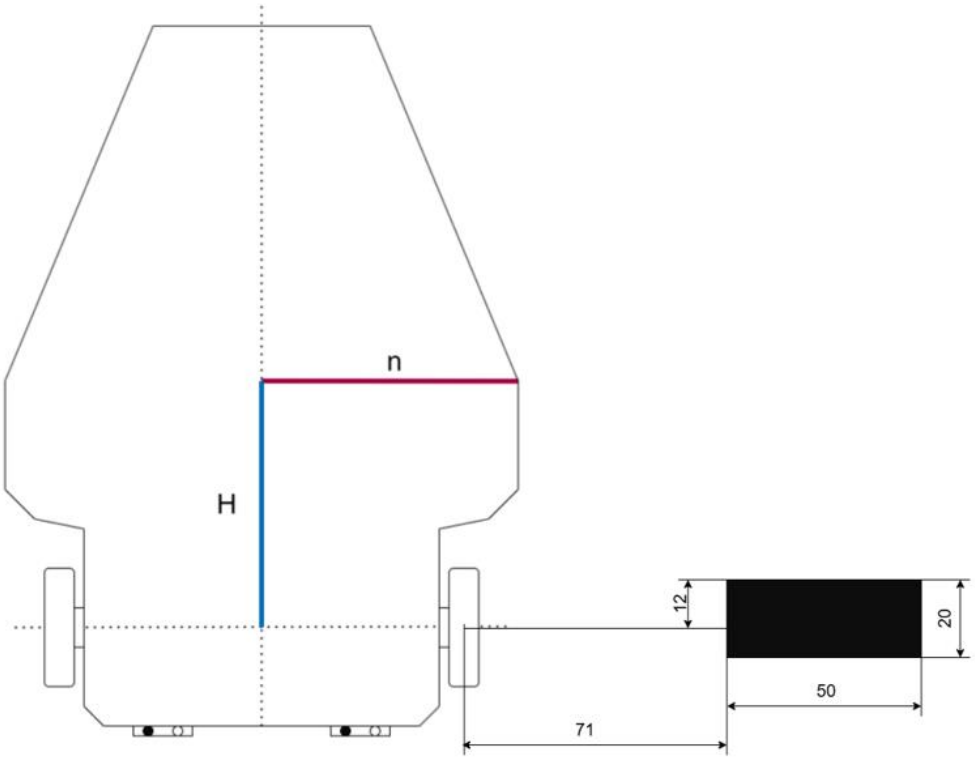
$$t_{ms} = t \times 1000 \tag{31}$$

$$t_{ms} = 0.702 \times 1000 = 702ms \tag{32}$$

Czas potrzebny na skręt o 52.5° wynosi 702ms

Krok 6: Obliczenie czasu i drogi przebytej przez robota w linii prostej

W tym etapie obliczymy drogę oraz czas niezbędny do pokonania dystansu, który pozwoli robotowi całkowicie ominąć przeszkodę najdalej wysuniętym punktem – jego boczną ścianą. Skupienie się na tym aspekcie umożliwi robotowi wykonanie kolejnego manewru skrętu, po którym będzie mógł wznowić ruch prostoliniowy w kierunku wyznaczonej ścieżki.

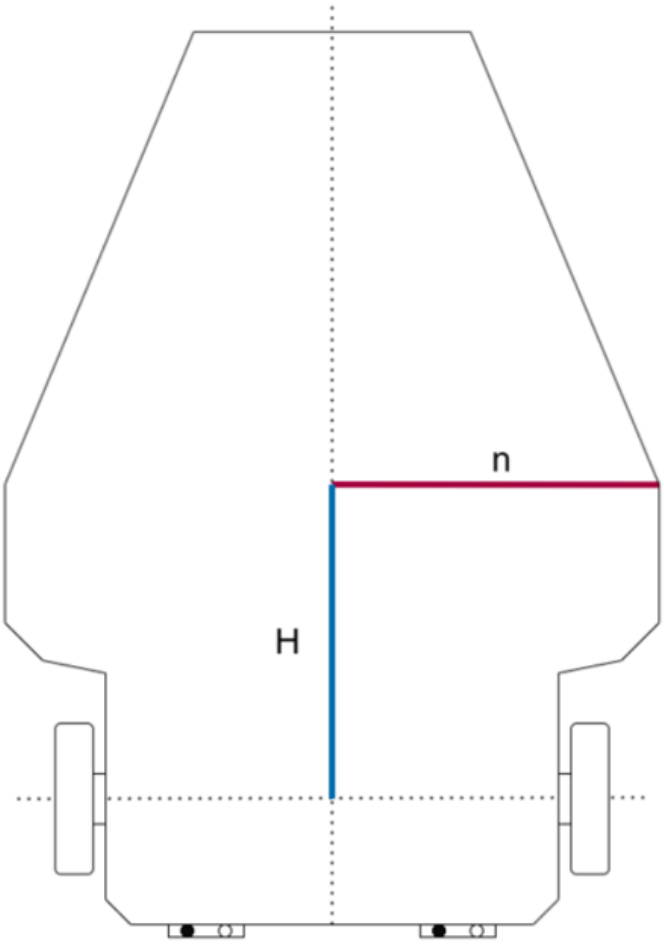


Rysunek 43.Pozycja robota przed manewrem jazdy na wprost w celu ominięcia przeszkody

Znając odległość H od osi do najbardziej wysuniętego punktu szerokości robota oraz pozostałą część nieominiętej przeszkody, możemy określić odległość, jaką robot musi pokonać, aby zagwarantować całkowite ominięcie przeszkody swoimi potencjalnie najdalej oddalonymi punktami.

- H=63.4 [mm]
- Nieprzejechana część przeszkody=8 [mm]

$$63.4 + 8 = 71.4mm \tag{33}$$



[redacted]

Rysunek 44.Pozycja robota po objechaniu przeszkody – gotowość do kolejnych manewrów

Widzimy, że robot przemieścił się o obliczony dystans 71,44 mm [Rysunek 44]. Teraz możemy przystąpić do obliczenia czasu potrzebnego na pokonanie tej odległości.

1. Czas potrzebny na pokonanie dystansu 71.44 (w sekundach)

$$t = \frac{obliczony\ dystans[mm]}{prędkość\ liniowa[\frac{m}{s}]} \tag{34}$$

$$t = \frac{0.07133}{0.138} = 0.518s \tag{35}$$

2. Konwersja czasu na milisekundy

$$t_{ms} = t \times 1000 \tag{36}$$

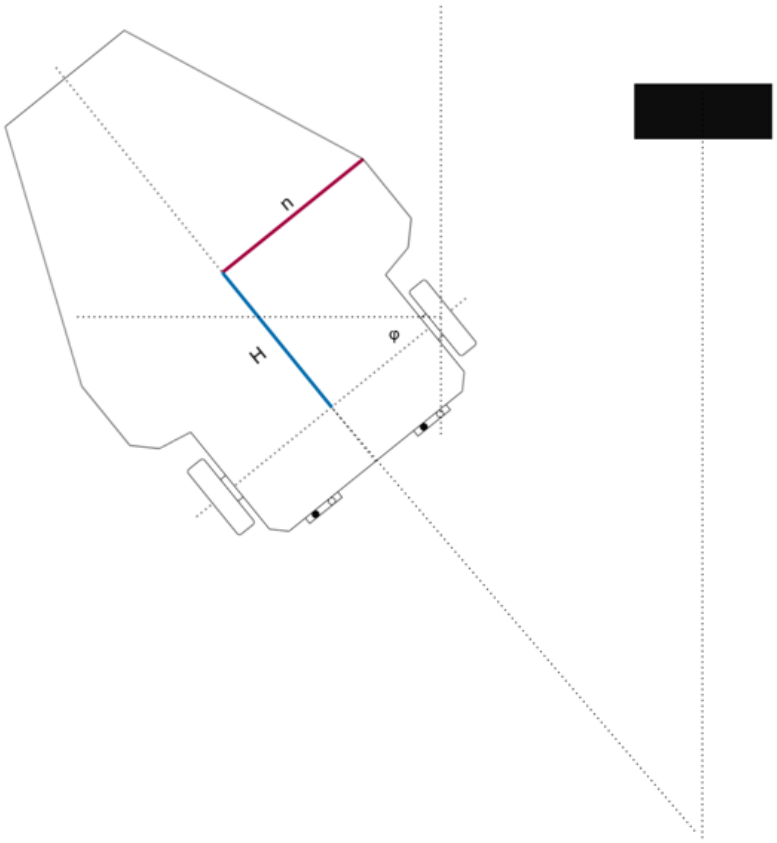
$$t_{ms} = 0.518 \times 1000 = 518ms \tag{37}$$

Ostatecznie czas potrzebny na przebycie dystansu 71,44 mm przez robota wynosi 518 ms.

Krok 7: Wyznaczenie kąta i czasu skrętu

Teraz, gdy najdalszy punkt robota znajduje się na linii końca przeszkody, możemy przystąpić do kolejnego manewru skrętu. Jego celem jest ustawienie robota pod odpowiednim kątem względem jego planowanej trasy, co umożliwi wznowienie ruchu do przodu aż do momentu powrotu na wyznaczony tor. Dla uproszczenia obliczeń przyjmujemy, że robot rozpocznie manewr ominięcia przeszkody pod takim samym kątem jaki został ustalony w

Krok 5 , stosując tę samą metodę, w której jedno koło obraca się, a drugie pozostaje w miejscu.



Rysunek 45.Pozycja robota po wykonaniu manewru skrętu

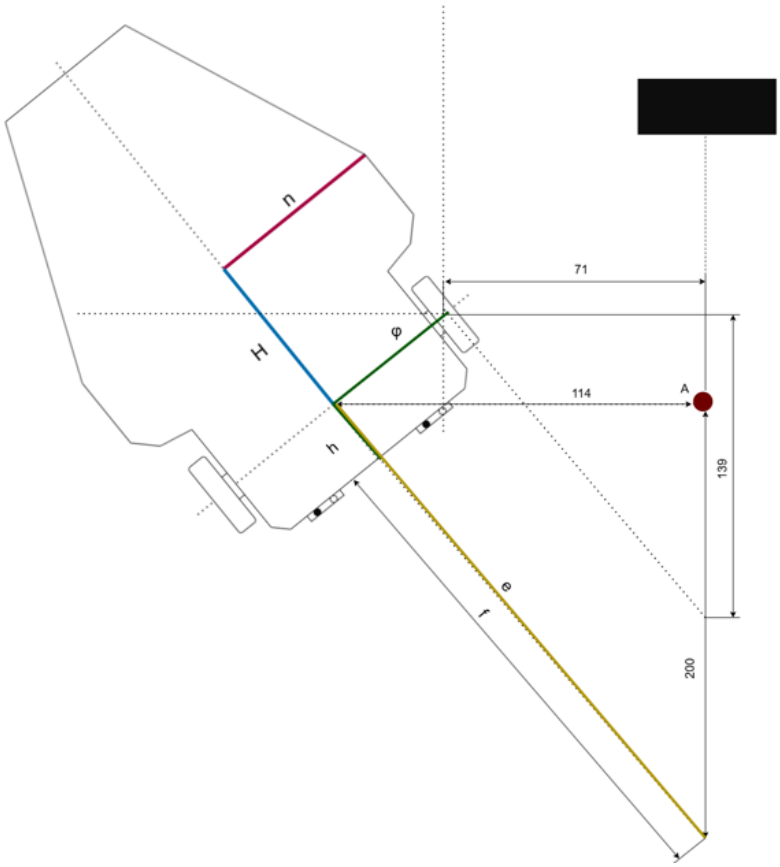
Czas potrzebny na skręt o 52.5° wynosi 702ms

Krok 8: Wyznaczenie drogi i czasu ignorowania czujnika

Znając kąt obrotu robota, możemy określić jego położenie w przestrzeni.

Dalsza część algorytmu kończy się wykonywaniem manewrów skrętu, po których robot wykrywa biały kolor za pomocą obu czujników i kontynuuje jazdę na prostym odcinku. W tym celu wyliczyliśmy kąt skrętu. Jednak po wykonaniu ostatniego manewru skrętu implementujemy funkcję ignorowania jednego z czujników podczerwieni. Dzięki temu robot, wykrywając linię pierwszym czujnikiem ustawionym bliżej linii, nie rozpocznie ponownie algorytmu wykrywania linii, lecz przejedzie pierwszym z czujników po czarnym odcinku, ignorując go. Funkcja ignorowania zostanie wyłączona po pewnym czasie.

Aby to zrealizować, potrzebujemy obliczyć drogę i czas, w jakim robot przejedzie do punktu przecięcia swojego przodu z linią trasy. Następnie od tej wartości odejmujemy odległość h, co pozwoli nam zapewnić, że aktywność czujnika rozpocznie się w momencie, gdy oba czujniki będą znajdować się między linią a robotem



Rysunek 46.Pozycja robota po wykonaniu skrętu – obliczenia wymiarowe dla ignorowania czujnika i jazdy na wprost

Po nałożeniu istotnych wymiarów: odległości od środka osi robota do punktu A oraz odległości od punktu A do miejsca, w którym linia poprowadzona wzdłuż kierunku robota przetnie się z trasą robota, możemy przystąpić do obliczeń

Możemy skorzystać z twierdzenia Pitagorasa, aby wyznaczyć dokładną długość e. Od tej wartości odejmujemy następnie długość h, co pozwoli nam upewnić się, że punkt znajdujący się dokładnie między dwoma czujnikami robota będzie precyzyjnie na środku trasy.

$$e = \sqrt{114^2 + 200^2} = 230.2 \tag{38}$$

$$f = e - h \tag{39}$$

$$f = 230.2 - 26.6 = 203.6 \tag{40}$$

Droga która musi zostać przejechana w czasie ignorowania czujnika, wynosi 203,6. Teraz możemy przejść do obliczenia czasu potrzebnego na przebycie tego dystansu.

1. Czas potrzebny na pokonanie dystansu 203.6 (w sekundach)

$$t = \frac{f[mm]}{prędkość\ liniowa[\frac{m}{s}]} \tag{41}$$

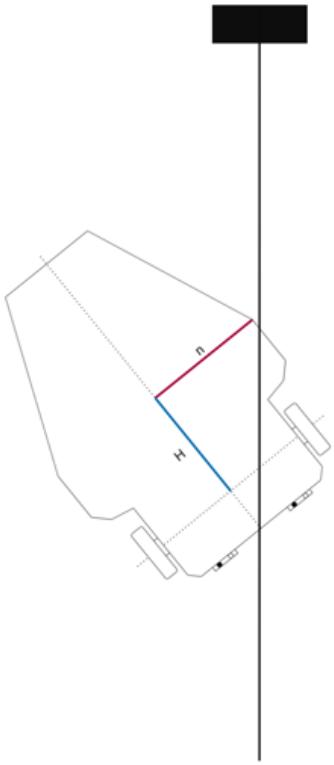
$$t = \frac{0.2036}{0.138} = 1.473s \tag{42}$$

2. Konwersja czasu na milisekundy

$$t_{ms} = t \times 1000 \tag{43}$$

$$t_{ms} = 1.473 \times 1000 = 1473ms \tag{44}$$

Ostatecznie czas potrzebny na przebycie dystansu f, w którym czujnik będzie ignorowany wynosi 1473 ms.



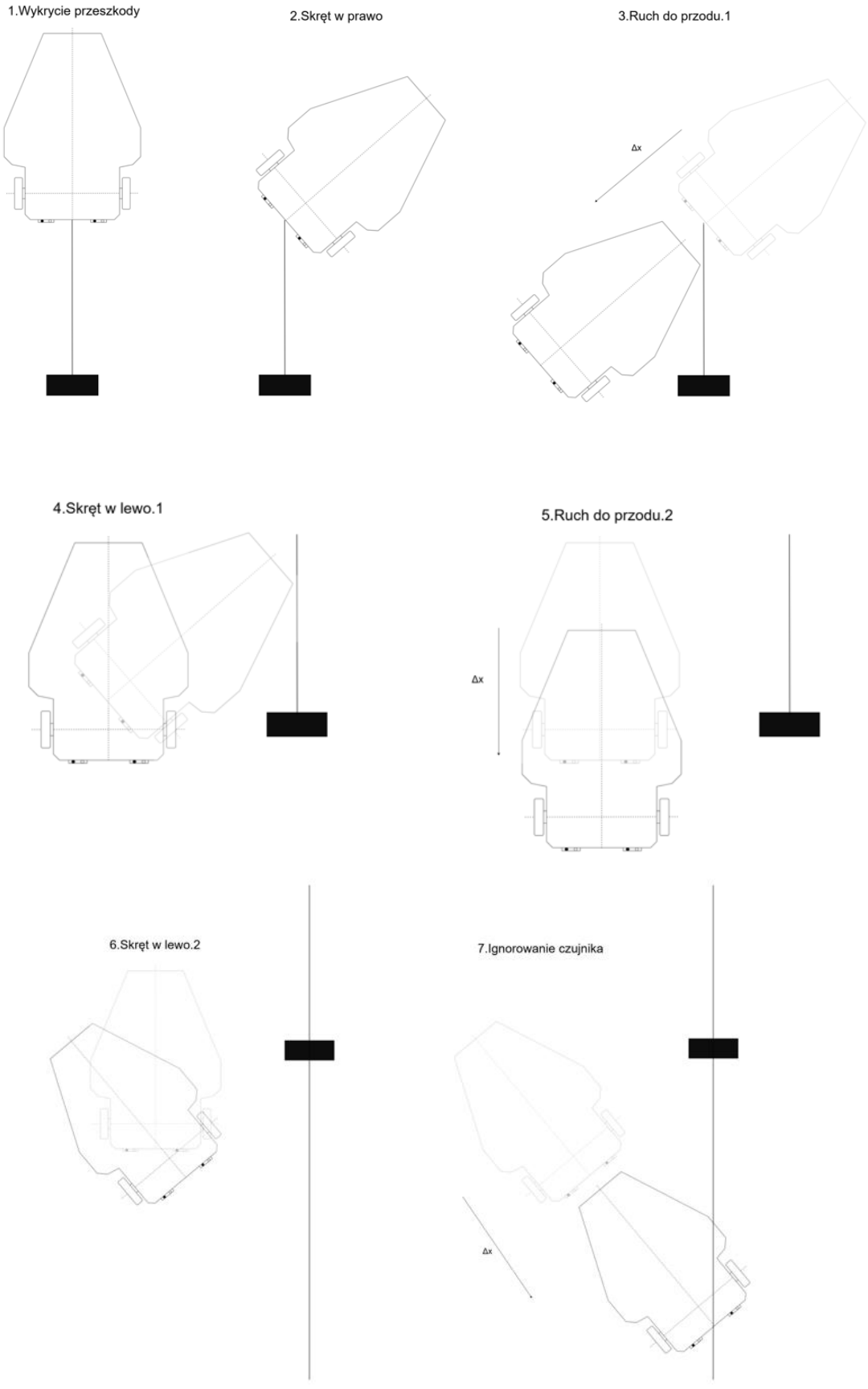
Rysunek 47.Robot po zakończeniu manewrów omijania i ignorowaniu czujnika – gotowy do dalszego wykrywania linii

Widzimy, że robot przemieścił się o wyliczony dystans, przejeżdżając lewym czujnikiem przez linię, którą tymczasowo ignorował przez wyliczony czas. Po upływie tego czasu algorytm powraca do normalnego działania, a robot ponownie podejmuje śledzenie linii.

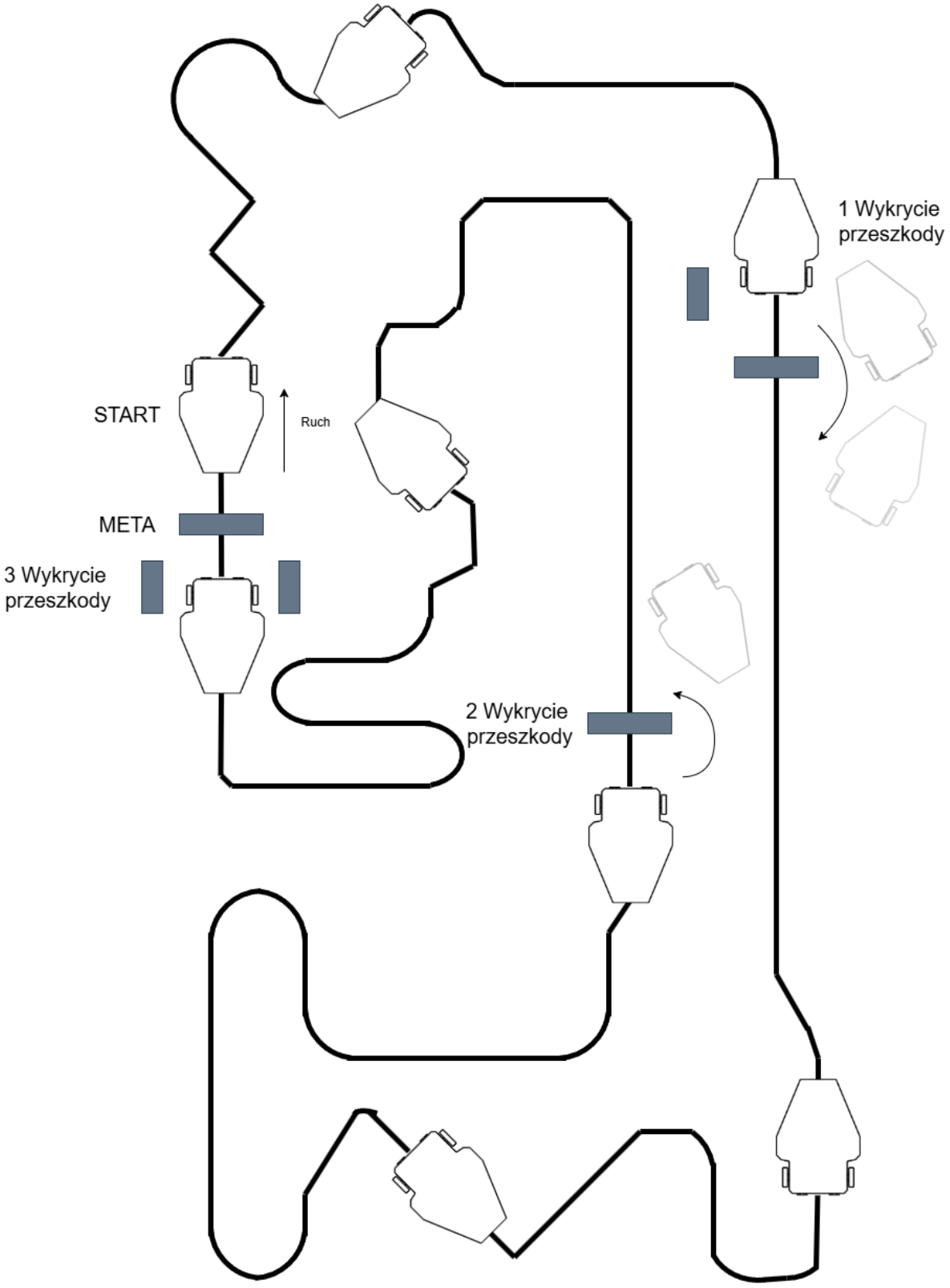
6.6 Zmienne parametry w badaniach

W przeprowadzonych badaniach brano pod uwagę zmienne parametry, które miały wpływ na wydajność manewrów robota. Parametry te obejmowały szerokość przeszkody, odległość detekcji oraz szerokość sygnału PWM. Na podstawie tych zmiennych obliczono czas wykonania poszczególnych manewrów robota, takich jak skręt w lewo, skręt w prawo oraz ruch do przodu czy ignorowanie jednego z czujników. Poniższa tabela przedstawia wartości parametrów i odpowiadające im wyniki pomiarów dla różnych kombinacji zmiennych.

6.6.1 Instrukcja wykonywania manewrów z tabelą pomiarów



Rysunek 48.Instrukcja wykonywania manewrów



Rysunek 49.Przejazd robota po trasie wraz z detekcją przeszkód

Rozpoczynamy ruch robota. Robot na samym starcie nie wykrył w pobliżu żadnej przeszkody, więc zaczyna poruszać się zgodnie z algorytmem śledzenia linii. Ruch jest realizowany aż do momentu, gdy robot wykryje pierwszą przeszkodę oznaczoną jako '1 Wykrycie przeszkody'. W tej sytuacji robot identyfikuje przeszkodę tylko po prawej stronie, co powoduje uruchomienie sekwencji wykonywania manewru omijania przeszkody z lewej strony. Po powrocie na swoją wyznaczoną ścieżkę robot wraca do algorytmu śledzenia linii i kontynuuje poruszanie się po trasie.

Kolejnym punktem jest '2 Wykrycie przeszkody', gdzie robot nie napotyka przeszkód ani po lewej, ani po prawej stronie swojej ścieżki. W takiej sytuacji robot zgodnie z logiką programu zaczyna realizować manewr omijania przeszkody z prawej strony. Po wykonaniu zaprogramowanych ruchów robot wraca na trasę i kontynuuje jazdę zgodnie z algorytmem śledzenia linii.

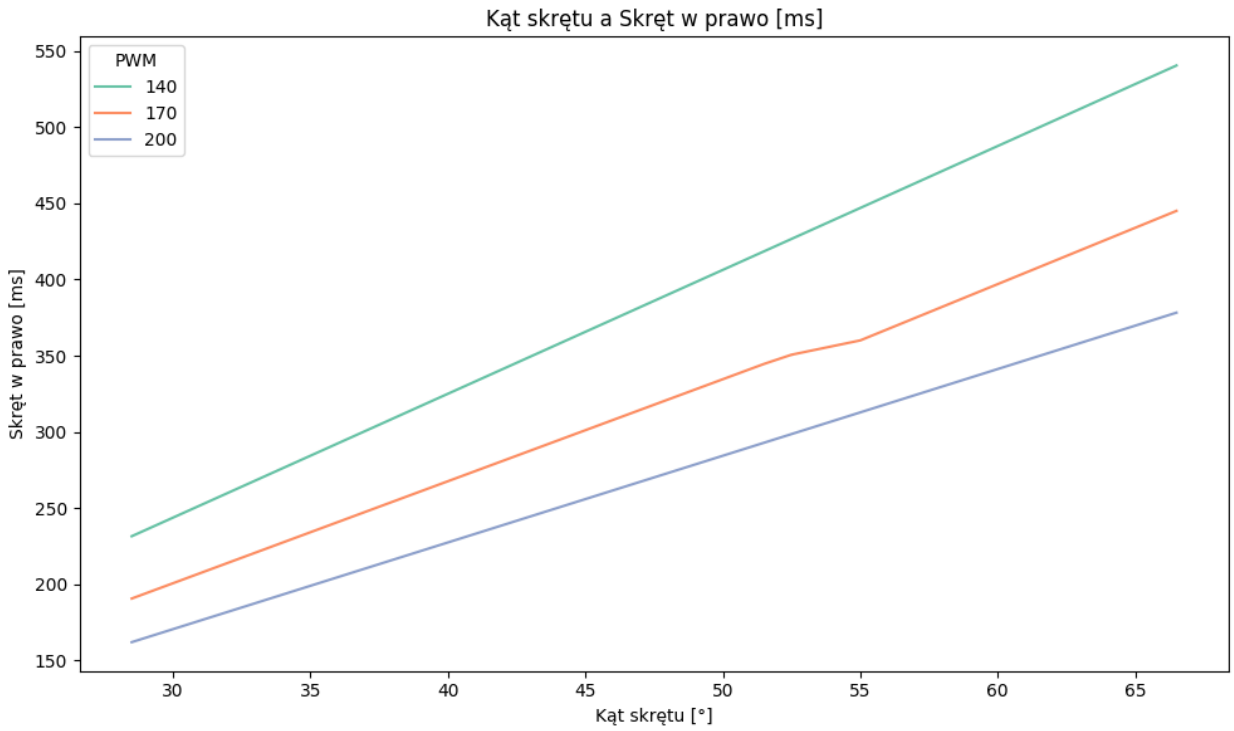
W punkcie oznaczonym jako '3 Wykrycie przeszkody' robot napotyka przeszkodę zarówno po lewej, jak i po prawej stronie, co uniemożliwia mu jej ominięcie. W tej sytuacji robot zatrzymuje się.

Odległość detekcji [cm]	Szerokość przeszkody [cm]	PWM	Kąt skrętu [˚]	Skręt w prawo	Ruch do przodu 1 [ms]	Skręt w lewo 1 [ms]	Ruch do przodu 2 [ms]	Skręt w lewo 2 [ms]	Czas ignorowania czujnika [ms]	Dobierana długość d	Odległość do boku [cm]
10	5	140	52.5	426.63	1430.17	853.25	627.24	853.25	1788.59	14	3.5
15	5	140	38.5	412.86	1961.92	609.46	574.53	609.46	1646.28	12	4
20	5	140	28.5	231.6	2312.26	1080.78	556.96	1080.78	2138.23	10	3
10	10	140	55	446.94	1558.17	837	706.3	837	1196.5	18	4
15	10	140	52.5	426.63	1917.2	658.22	556.96	658.22	1748.8	15	5
20	10	140	37.5	304.73	2684.47	597.24	609.67	597.24	1994.51	12	3.5
10	15	140	66.5	540.39	1619.84	437.98	732.66	437.98	1306.48	22	4
15	15	140	51.5	418.5	1961.92	324.22	688.73	324.22	1706.37	15	4
20	15	140	40.5	329.11	2299.43	625.68	627.24	625.68	1917.38	12	3
10	5	170	52.5	350	1180	702	518	702	1473	14	3.5
15	5	170	38.5	257.64	1615.64	515.28	437.12	515.28	1355.71	12	4
20	5	170	28.5	190.72	1094.14	381.44	458.66	381.44	1760.84	10	3
10	10	170	55	360.06	1283.15	736.11	581.64	736.11	985.31	18	4
15	10	170	52.5	351.33	1578.82	702.65	458.66	702.65	1440.14	15	5
20	10	170	37.5	250.95	2210.66	501.89	502.06	501.89	1642.48	12	3.5
10	15	170	66.5	445.01	1333.94	890.03	603.34	890.03	1075.89	22	4
15	15	170	51.5	344.63	1615.64	689.72	567.17	689.72	1405.12	15	4
20	15	170	40.5	271.02	1893.05	542.05	516.53	542.05	1578.96	12	3
10	5	200	52.5	298.62	1001.06	597.24	439.04	597.24	1251.94	14	3.5
15	5	200	38.5	218.99	1372.26	437.98	402.15	437.98	1152.33	12	4
20	5	200	28.5	162.11	1618.48	324.22	389.85	324.22	1496.67	10	3
10	10	200	55	312.84	1090.65	625.68	494.38	625.68	837.5	18	4
15	10	200	52.5	298.62	1341.96	597.24	389.85	597.24	1224.08	15	5
20	10	200	37.5	213.3	1879.02	426.6	426.74	426.6	1396.07	12	3.5
10	15	200	66.5	378.25	1133.82	756.5	518.83	756.5	914.48	22	4
15	15	200	51.5	292.93	1373.26	585.86	482.08	585.86	1194.38	15	4
20	15	200	40.5	230.36	1609.5	460.73	439.04	460.73	1342.08	12	3

Tabela 2.Badane wartości

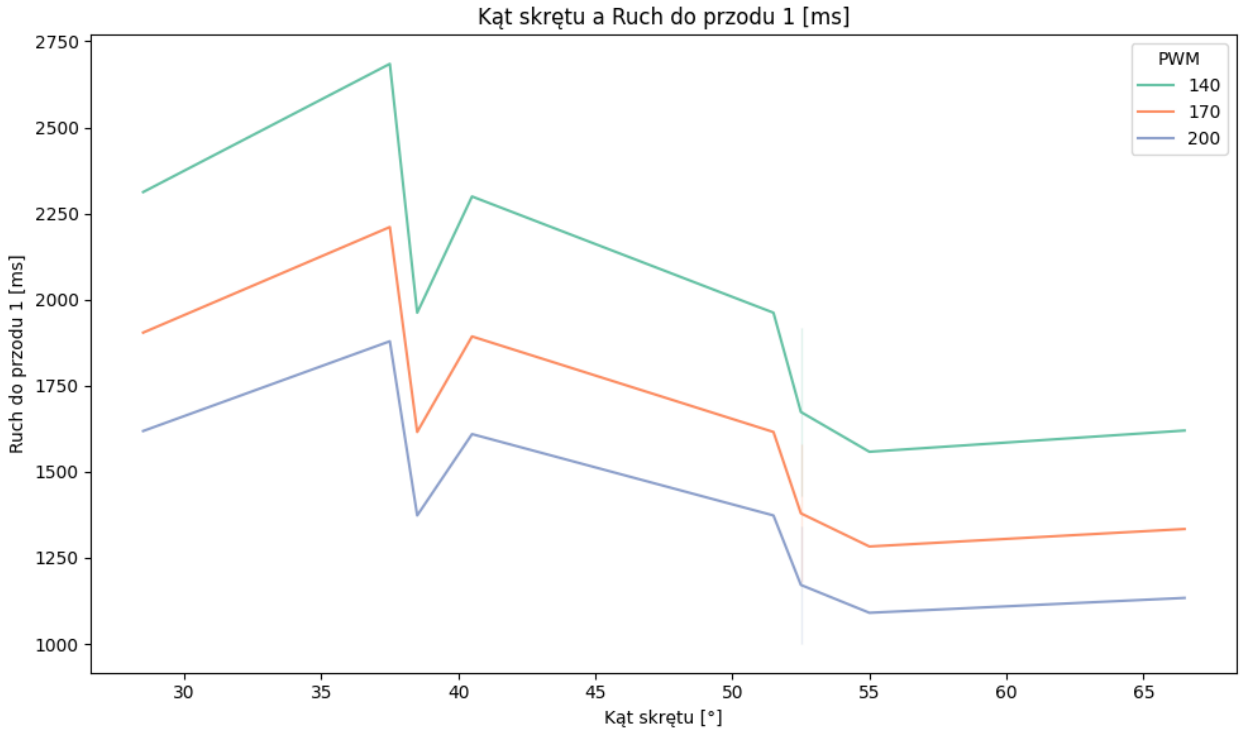
6.7 Wykresy zależności parametrów w badaniach

Analizując tabelę, możemy zaobserwować, że wyższe wartości PWM, takie jak 200, sprzyjają bardziej dynamicznemu i agresywnemu działaniu, co jest szczególnie przydatne w sytuacjach wymagających szybkiego unikania przeszkód. Wiąże się to jednak z wyższym zużyciem energii. Z kolei niższe wartości PWM, takie jak 140, pozwalają na bardziej precyzyjne i oszczędne sterowanie, co sprawdza się w środowiskach, gdzie kluczowa jest dokładność. Aby lepiej zobrazować te zależności, zostaną przedstawione wykresy, które pomogą wizualnie potwierdzić te obserwacje.



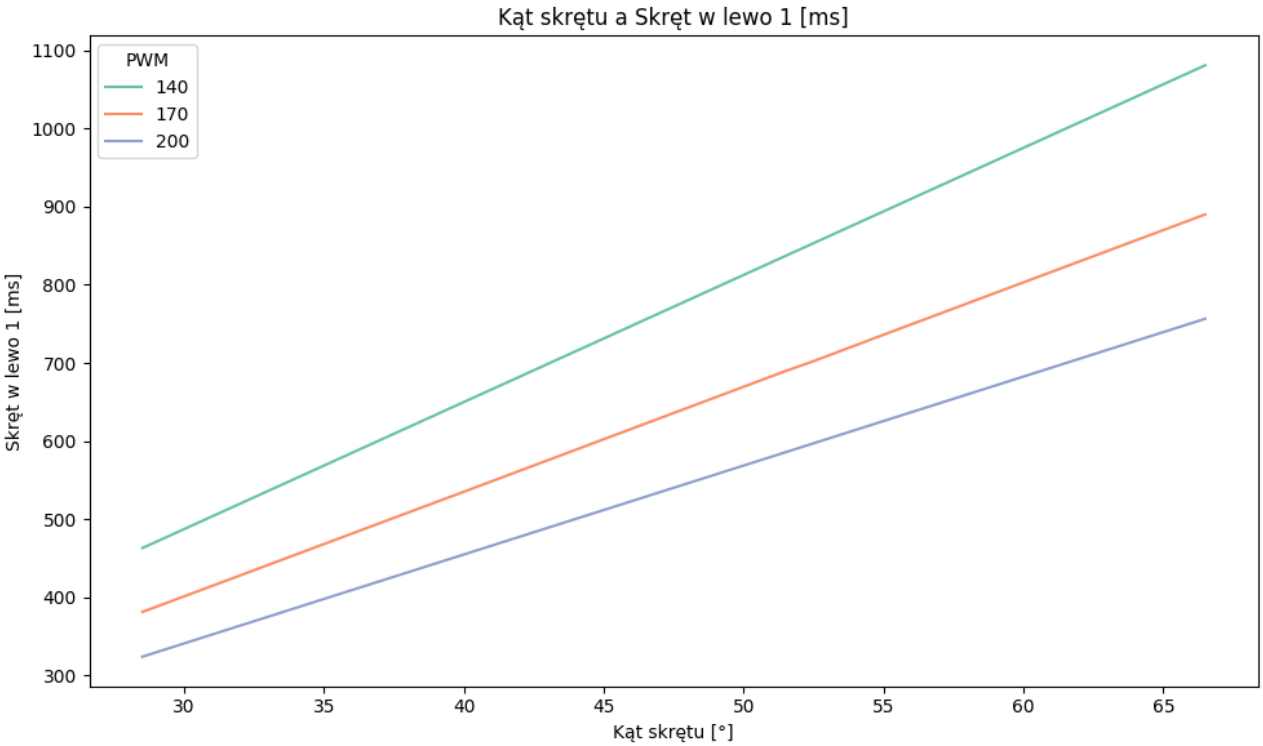
Rysunek 50.Kąt skrętu a skręt w prawo

Wykres,[Rysunek 50] przedstawiający zależność między kątem skrętu a czasem skrętu w prawo , ujawnia, że wraz ze wzrostem kąta skrętu czas, przez który robot skręca w prawo, również wzrasta. Zauważalna jest także zależność tej wielkości od wartości PWM. Wyższe wartości PWM prowadzą do wydłużenia czasu skrętu, co może sugerować, że przy większej mocy silników robot skręca bardziej zdecydowanie i dłużej.



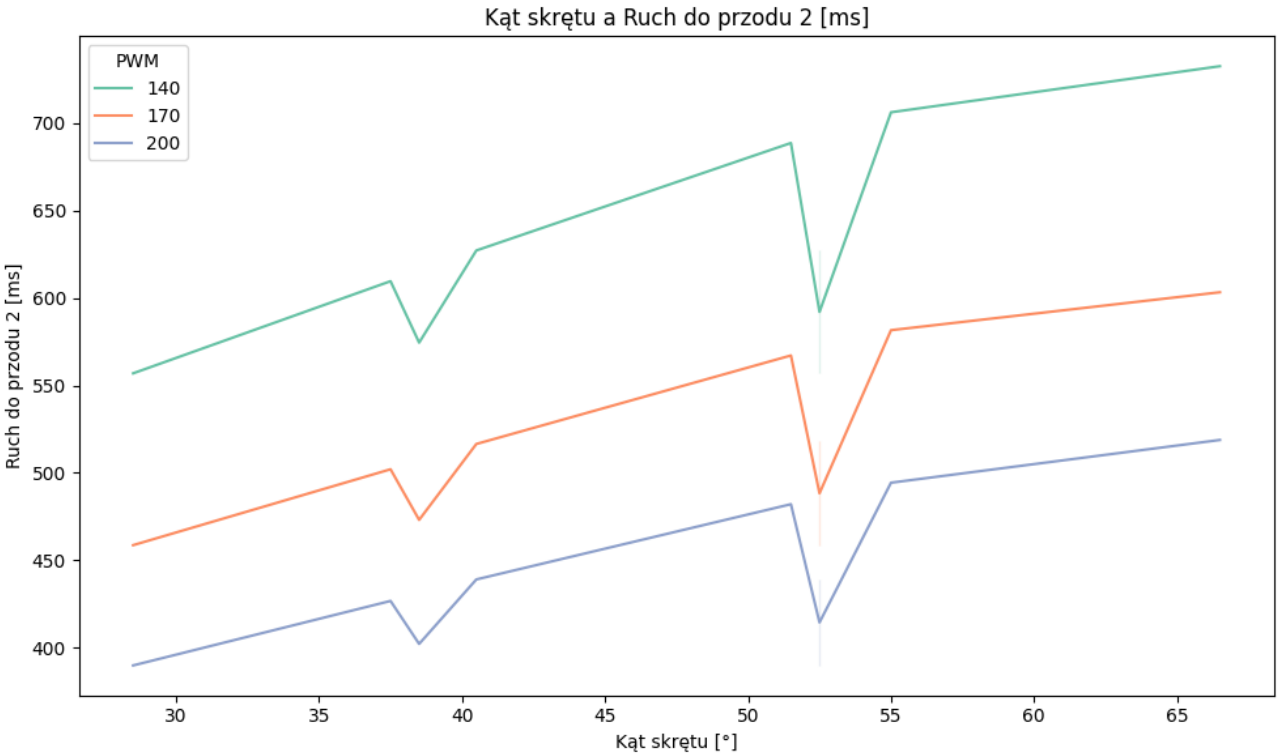
Rysunek 51.Kąt skrętu a Ruch do przodu 1

Podobnie, na wykresie przedstawiającym zależność między kątem skrętu a ruchem do przodu1 [Rysunek 51], widać, że czas ruchu do przodu pozostaje stosunkowo stały niezależnie od kąta skrętu. Może to wskazywać na to, że czas poruszania się do przodu nie zależy bezpośrednio od kąta, co może być efektem zaprogramowanej trajektorii robota, która zapewnia niezależność ruchu do przodu od skrętów.



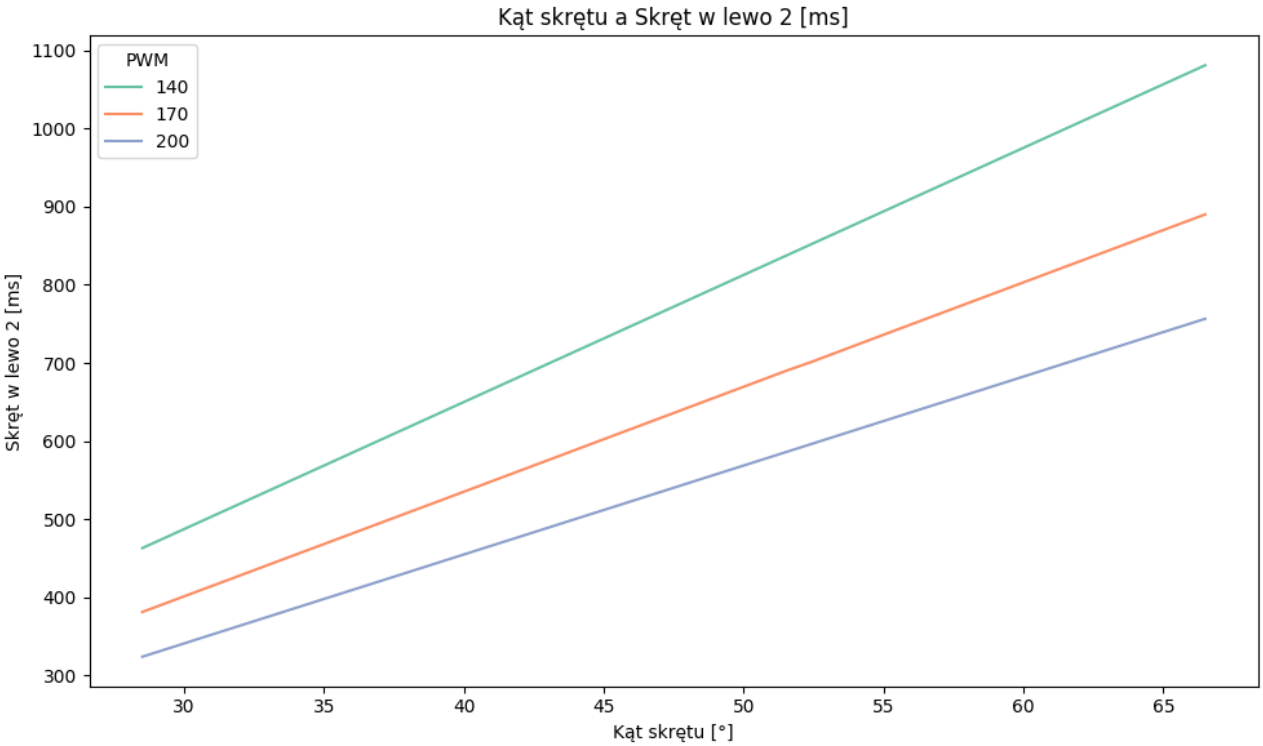
Rysunek 52.Kąt skrętu a Skręt w lewo1

Wykres ilustrujący zależność między kątem skrętu a czasem (skrętu w lewo 1) [Rysunek 52] prezentuje bardzo podobną zależność jak w przypadku skrętu w prawo. Zwiększenie kąta skrętu prowadzi do wydłużenia czasu skrętu w lewo, a wyższe wartości PWM także skutkują dłuższym czasem skrętu. Ta zależność jest logiczna, ponieważ większa moc silników powinna umożliwiać dłuższe i bardziej intensywne skręty.



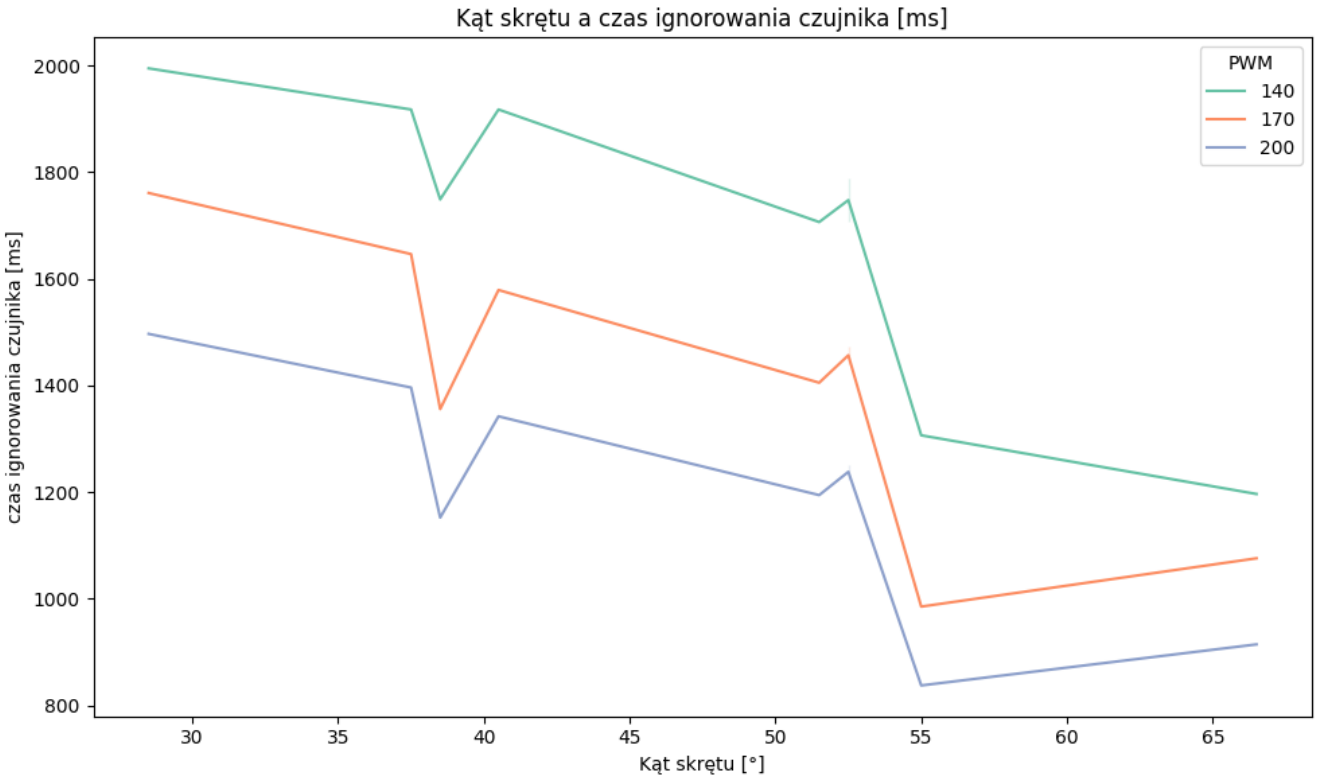
Rysunek 53.Kąt skrętu a Ruch do przodu 2

W przypadku wykresu pokazującego zależność między kątem skrętu a czasem poruszania się do przodu w innym scenariuszu (Ruch do przodu2) [Rysunek 53], również zauważamy, że czas ruchu do przodu nie jest znacząco związany z kątem skrętu. Może to sugerować, że robot w różnych konfiguracjach zachowuje podobne trajektorie poruszania się, niezależnie od zmieniających się kątów skrętu.



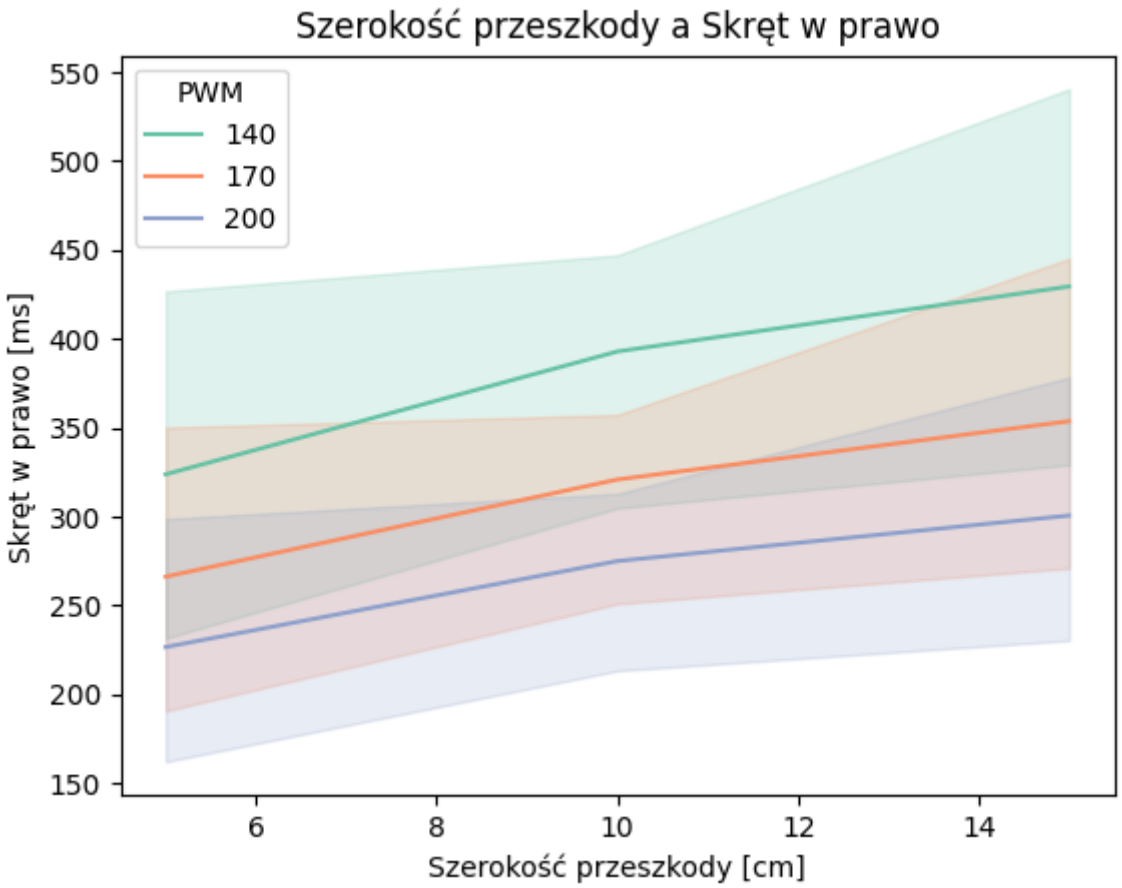
Rysunek 54.Kąt skrętu a Skręt w lewo 2

Kolejny wykres, przedstawiający zależność między kątem skrętu a czasem skrętu w lewo w drugim przypadku [Rysunek 54], wykazuje analogiczny wzrost czasu skrętu w lewo przy wzroście kąta skrętu. Dodatkowo, zmiany w czasie skrętu są bardziej widoczne przy wyższych wartościach PWM, co sugeruje, że robot reaguje dynamiczniej na zmieniające się ustawienia silników.



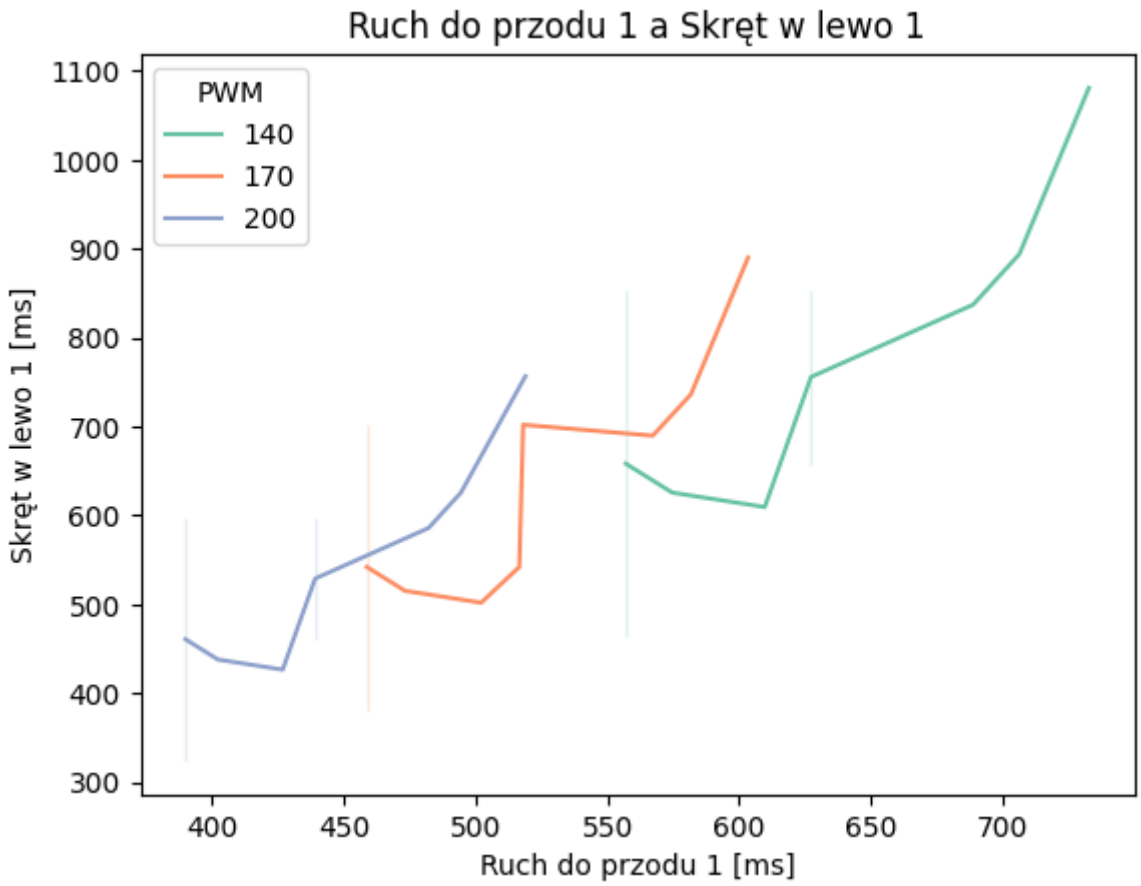
Rysunek 55.Kąt skrętu a czas ignorowania czujnika

Wykres [Rysunek 55], dotyczący zależności między kątem skrętu a czasem ignorowania czujnika, pokazuje bardziej zróżnicowaną reakcję robota. W miarę wzrostu kąta skrętu, czas ignorowania przeszkód (tzw. ignoreDuration) wykazuje nieliniową zależność. Możliwe, że robot dostosowuje swoje reakcje w zależności od zmieniającego się kąta, co jest szczególnie istotne w kontekście analizy unikania przeszkód i ich rozpoznawania przez czujniki.



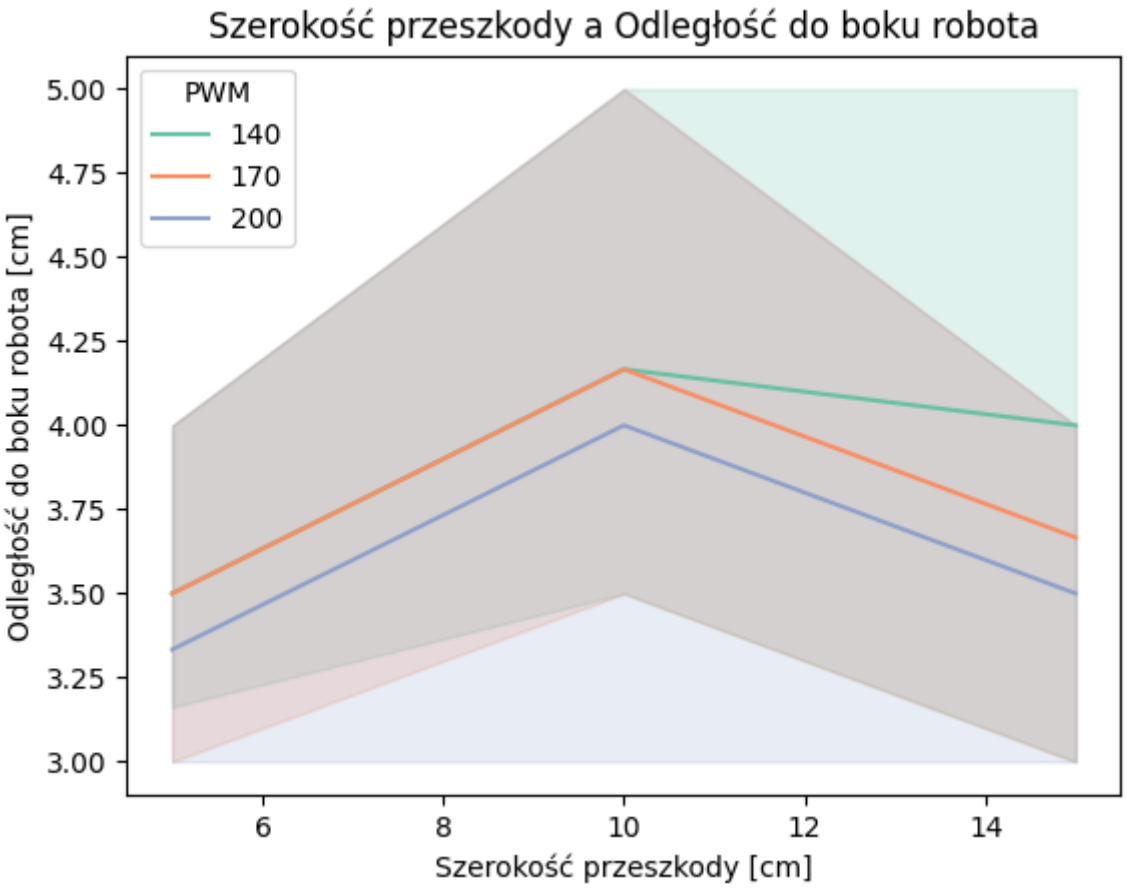
Rysunek 56.Szerokość przeszkody a Skręt w prawo

Kolejny wykres [Rysunek 56] ilustruje zależność między szerokością przeszkody a czasem skrętu w prawo. Na osi X znajduje się szerokość przeszkody, a na osi Y czas skrętu w prawo. Z wykresu wynika, że zwiększenie szerokości przeszkody prowadzi do wydłużenia czasu skrętu w prawo, a wyższe wartości PWM powodują dodatkowe zmiany w tym czasie.



Rysunek 57.Ruch do przodu 1 a Skręt w lewo 1

Wykres [Rysunek 57] przedstawia zależność między czasem poruszania się do przodu w innym scenariuszu (Ruch do przodu 1) a czasem skrętu w lewo (Skręt w lewo 1). Oś X przedstawia czas ruchu do przodu, a oś Y czas skrętu w lewo. Wartości PWM różnicują kolory linii, co pozwala na obserwację, jak zmiany w czasie ruchu do przodu wpływają na czas skrętu w lewo. Z analizy wykresu można zauważyć, że zwiększenie czasu poruszania się do przodu prowadzi do wydłużenia czasu skrętu w lewo, a wartości PWM modyfikują tę zależność



Rysunek 58.Szerokość przeszkody a Odległość do boku robota

Ostatni wykres [Rysunek 58] przedstawia zależność między szerokością przeszkody a odległością robota od boku. Oś X pokazuje szerokość przeszkody, a oś Y odległość robota od boku. Z wykresu wynika, że zmieniająca się szerokość przeszkody wpływa na to, jak robot utrzymuje odległość od boku, z wyraźnym wpływem wartości PWM na tę zależność.

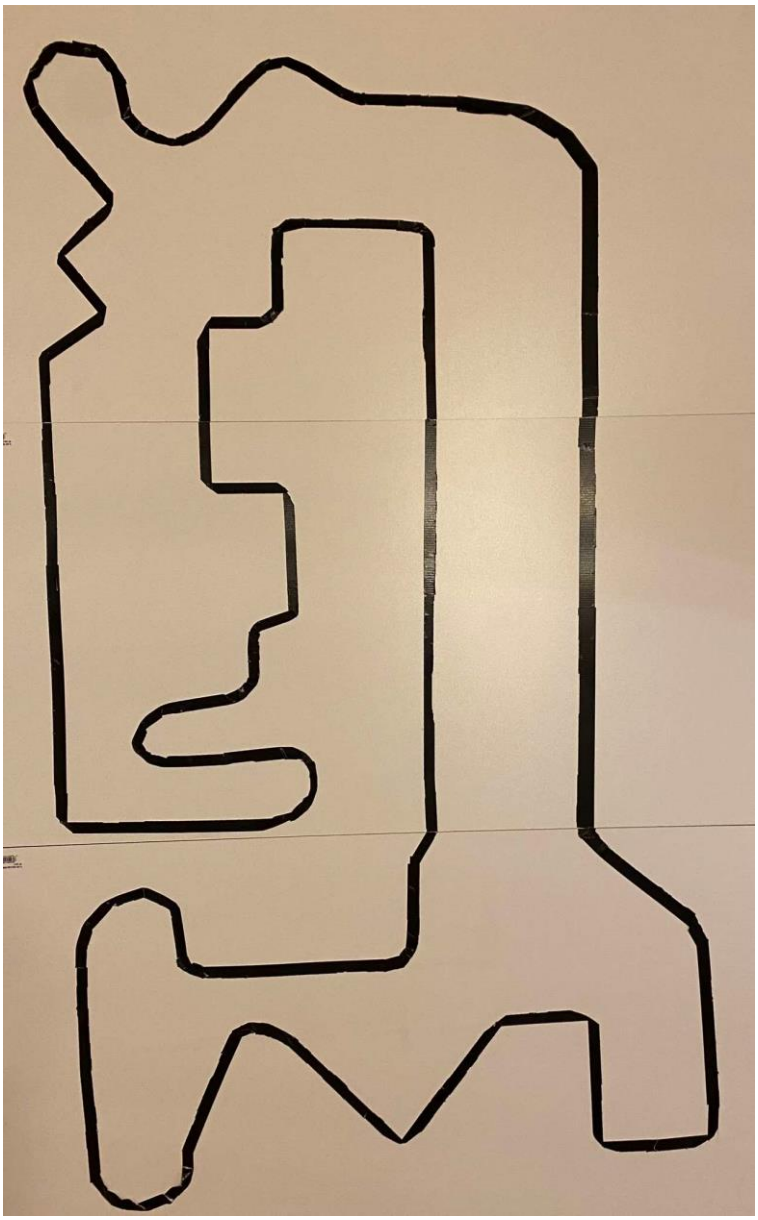
Podsumowując, wykresy te ukazują, jak różne parametry ruchu robota, takie jak czas skrętu czy czas poruszania się do przodu, zmieniają się w zależności od kąta skrętu i wartości PWM. Zwiększenie wartości PWM prowadzi do bardziej intensywnych i dłuższych skrętów, a czas ruchu do przodu wykazuje niezależność od kąta. Takie zależności są istotne w kontekście optymalizacji trajektorii robota oraz jego reakcji na zmieniające się warunki otoczenia, zwłaszcza w kontekście unikania przeszkód i dostosowywania parametrów ruchu w zależności od szerokości przeszkody oraz innych zmiennych.

6.7.1 Analiza wybranego przypadku przejazdu

Kolejnym badaniem, które zostanie przeprowadzone, będzie analiza dokładności i powtarzalności robota, którego zadaniem jest wykrywanie przeszkód w odległości 10 cm, przy założeniu, że szerokość przeszkody również wynosi 10 cm. W kontekście tego zadania, kluczowym aspektem jest osiągnięcie kompromisu między dokładnością przejazdu a szybkością. Po przeanalizowaniu wyników, zdecydowałem się na ustawienie PWM na poziomie 170, co zapewnia odpowiednią prędkość obrotową rzędu 80 RPM. Wartość ta pozwala na zbalansowanie precyzji i dynamiki ruchu robota, umożliwiając mu wykrywanie przeszkód w wymaganym zasięgu, a jednocześnie zapewniając odpowiednią szybkość przejazdu. Wartości PWM inne niż 170, takie jak 200, mogłyby prowadzić do zbyt dużej intensywności ruchu, co utrudniałoby utrzymanie odpowiedniej precyzji, natomiast wartości niższe, jak 140, skutkowałyby spowolnieniem robota, co mogłoby negatywnie wpłynąć na czas reakcji w sytuacjach wymagających szybkiego unikania przeszkód. Wybór PWM=170 zapewnia optymalną równowagę pomiędzy dokładnością, powtarzalnością a szybkością robota.

Badanie dokładności i powtarzalności robota zaczniemy od pomiaru długości trasy, którą robot pokona w trakcie zadania. Następnie, wykorzystując wcześniej obliczoną prędkość robota V, obliczymy teoretyczny czas przejazdu, uwzględniając również czas wykonywania manewrów. Czas ten posłuży nam do porównania z rzeczywistym czasem przejazdu robota. W celu uzyskania rzetelnych wyników badania, przeprowadzimy 10 niezależnych przypadków przejazdu, które pozwolą na ocenę, jak rzeczywiste czasy różnią się od obliczonych. Na podstawie tych danych będziemy w stanie oszacować zarówno dokładność, jak i powtarzalność robota w wykrywaniu przeszkód i poruszaniu się po zadanej trasie.

W rzeczywistości trasa, którą robot pokonał podczas eksperymentów, została wykonana z trzech białych płyt HDF o wymiarach 3×1200×600 mm. Na tych płytach wyznaczono linię, za którą robot miał podążać, przy użyciu czarnej taśmy. Tak przygotowana trasa została zaprojektowana w sposób umożliwiający robotowi wykrywanie przeszkód oraz wykonywanie manewrów w warunkach zgodnych z założeniami eksperymentalnymi. Długość całkowita trasy wynosi ok. 923cm, a jej charakterystyka została zaplanowana tak, aby zaprezentować różnorodność przejazdu.



Rysunek 59.Rzeczywista trasa robota

Z prostej zależności kinematycznej możemy wydedukować całkowity czas potrzebny na przejazd trasy, znając prędkość V robota oraz długość drogi d. Zależność ta wyraża się wzorem

$$t = \frac{\mathrm{d}}{V} \qquad (45)$$

Czas skrętu t jest obliczany na podstawie przebytej odległości *d* i prędkości liniowej V

Dla zadanych wartości

- d=9.23 [m]
- V=0.1382 [m/s]

$$t = \frac{9.23}{0.1392} \approx 66.3\ sekund \tag{46}$$

Z obliczeń wynika, że robot powinien przejechać całą trasę w czasie około 66,3 sekund. Jednak należy zaznaczyć, że jest to czas teoretyczny, obliczany przy założeniu stałej prędkości oraz braku przeszkód, które mogłyby wpłynąć na rzeczywisty czas przejazdu. Rzeczywisty czas może się różnić, gdyż w trakcie przejazdu mogą wystąpić nieprzewidziane zdarzenia, takie jak poślizgnięcie się kół robota, zmiany w oporze ruchu, a także błędy w detekcji przeszkód.

Analizując tor z przeszkodami, przedstawiony na [Rysunek 49], musimy uwzględnić pewne zmiany w obliczeniach. W naszym przypadku detekcja przeszkód odbywa się 2 razy w odległości 10 cm, dlatego musimy odjąć dwa odcinki trasy o łącznej długości około 20 cm. Pierwsze 10 cm trasy to odcinek, na którym robot wykrywa przeszkodę, a kolejne 10 cm stanowi część trasy znajdującą się za przeszkodą. W wyniku tego, całkowita długość trasy, którą robot rzeczywiście pokonuje, zostaje zmniejszona o 40 cm.

Kolejnym czynnikiem, który wpływa na czas przejazdu, są manewry unikania przeszkód. Czas tych manewrów został uwzględniony w [Tabela 2]. Wartość ta odzwierciedla czas, który robot potrzebuje na wykonanie odpowiednich ruchów, takich jak zmiana kierunku jazdy. Dodatkowo, czas wykrywania przeszkody przez czujnik ultradźwiękowy oraz czas reakcji serwomechanizmu, który musi obrócić się w obu kierunkach, również wchodzi w skład całkowitego czasu przejazdu. Wartości te muszą zostać dodane do obliczeń, aby uzyskać rzeczywisty czas potrzebny na pokonanie trasy.

Po odjęciu 40 cm (w wyniku uwzględnienia obszarów detekcji przeszkód), długość trasy skraca się do 883 cm. Na podstawie tej nowej długości trasy możemy ponownie obliczyć teoretyczny czas przejazdu robota, korzystając ze wzoru (45)

$$t_{trasa} = \frac{8.83}{0.1392} \approx 63.4\ sekund \tag{46}$$

Mając obliczony teoretyczny czas przejazdu t_{trasa}, możemy uzupełnić go o brakujące elementy, które mają wpływ na rzeczywisty czas przejazdu robota. W tym celu należy dodać

Czas dwóch manewrów omijania przeszkód – robot wykonuje dwa pełne manewry unikania przeszkód (po jednym dla każdej przeszkody na trasie). Czas pojedynczego manewru to $t_{omijanie}$, który został uwzględniony w [Tabela 2] z wynikami badań.

Czas obrotu serwomechanizmu – serwomechanizm obraca się w obu kierunkach przy każdej przeszkodzie, aby określić, z której strony możliwe jest wykonanie manewru. Łącznie wymagane są dwa takie obroty, a czas jednego obrotu to $t_{obrót}$

Całkowity czas przejazdu można wyrazić jako

$$T_{całkowity} = t_{trasa} + 2 \times t_{omijanie} + 2 \times t_{obrót} \tag{47}$$

Gdzie

- t_{trasa} – czas przejazdu bez przeszkód
- $t_{omijanie}$ – czas potrzebny na wykonanie jednego manewru ominięcia przeszkody (suma wartości czasów: Skręt w prawo, Ruch do przodu 1, Skręt w lewo 1, Ruch do przodu 2, Skręt w lewo 2, Czas ignorowania czujnika)
- $t_{obrót}$ – czas obrotu serwomechanizmu

Wartości $t_{omijanie}$ jesteśmy w stanie odczytać z [Tabela 2] dla danych wartości

- PWM = 170
- Odległość detekcji = 10 [cm]
- Szerokość przeszkody = 10 [cm]

$$t_{omijanie} = 360.06 + 1283.15 + 736.11 + 581.64 + 736.1\ + 985.31\ \text{[ms]} \tag{48}$$

$$t_{omijanie} = 4682.38\ [ms] \tag{49}$$

Przekształcając na sekundy

$$t_{omijanie} = 4682.38\ [ms] \times 0{,}001 \tag{50}$$

Ostatecznie otrzymujemy

$$t_{omijanie} \approx 4.68\ sekundy \tag{51}$$

- Czas obrotu serwomechanizmu $t_{obrót}$ wynosi 4 sekundy

Reasumując

- t_{trasa} = 63.4 [s]
- $t_{omijanie}$ = 4.68 [s]
- $t_{obrót}$ = 4 [s]

Mając wszystkie niezbędne wartości czasów, możemy podstawić je do wzoru na całkowity czas przejazdu robota, wzór (47)

$$T_{całkowity} = 63.4 + 2 \times 4.68 + 2 \times 4\ [s] \tag{52}$$

$$T_{całkowity} = 80.76\ [s] \tag{53}$$

Całkowity teoretyczny czas na przejechanie całej trasy wraz z detekcją przeszkód wynosi 80.76 sekund

Po teoretycznym obliczeniu czasu przejazdu robota przystąpimy do części doświadczalnej, w której zostanie przeprowadzonych 10 praktycznych przejazdów robota. Na podstawie uzyskanych wyników będziemy mogli ocenić dokładność oraz powtarzalność robota w rzeczywistych warunkach. W szczególności zwrócimy uwagę na odchylenia od teoretycznie obliczonego czasu, analizując wpływ czynników takich jak dynamika ruchu, precyzja detekcji przeszkód oraz wykonanie manewrów omijania.

Numer przejazdu	Teoretyczny czas przejazdu [s]	Czas przejazdu w praktyce [s]
1	80.76	87.10
2	80.76	87.96
3	80.76	88.45
4	80.76	89.34
5	80.76	87.56
6	80.76	88.12
7	80.76	88.89
8	80.76	89.21
9	80.76	87.74
10	80.76	90.13

Tabela 3.Analiza różnic między teoretycznym a praktycznym czasem przejazdu robota

6.7.2 Analiza różnic między teoretycznym a praktycznym czasem przejazdu

Cel obliczeń

1. Wyznaczenie różnicy (błędu absolutnego) między czasem teoretycznym a praktycznym
2. Obliczenie błędu względnego w postaci procentowej
3. Obliczenie różnicy oraz średniego błędu procentowego
4. Wyznaczenie odchylenia standardowego praktycznych czasów w celu oceny powtarzalności

1.Błąd absolutny

$$Różnica(błąd\ absolutny) = |T_{teoretyczny} - T_{praktyczny}| \quad (54)$$

Próba	Praktyczny czas($T_{praktyczny}$)	Różnica(s)
1	87.10	6.34
2	87.96	7.2
3	88.45	7.69
4	89.34	8.58
5	87.56	6.8
6	88.12	7.36
7	88.89	8.13
8	89.21	8.45
9	87.74	6.98
10	90.13	9.37

Tabela 4.Porównanie różnic czasów praktycznych a teoretycznych

2.Błąd procentowy

$$Błąd\ procentowy = \frac{|T_{teoretyczny} - T_{praktyczny}|}{T_{teoretyczny}} \times 100\% \quad (55)$$

Próba	$T_{praktyczny}$[s]	Błąd procentowy[%]
1	87.10	$\frac{6.34}{80.76} \times 100 = 7.85\ \%$
2	87.96	$\frac{7.2}{80.76} \times 100 = 8.91\ \%$
3	88.45	$\frac{7.69}{80.76} \times 100 = 9.52\ \%$
4	89.34	$\frac{8.58}{80.76} \times 100 = 10.62\ \%$
5	87.56	$\frac{6.8}{80.76} \times 100 = 8.42\ \%$
6	88.12	$\frac{7.36}{80.76} \times 100 = 9.11\ \%$
7	88.89	$\frac{8.13}{80.76} \times 100 = 10.06\ \%$
8	89.21	$\frac{8.45}{80.76} \times 100 = 10.46\ \%$
9	87.74	$\frac{6.98}{80.76} \times 100 = 8.64\ \%$
10	90.13	$\frac{9.37}{80.76} \times 100 = 11.60\ \%$

Tabela 5.Porównanie błędów procentowych

3.Średnia różnica i średni błąd procentowy

Średnia różnica (błąd absolutny)

$$Średnia\ różnica = \frac{\Sigma|T_{teoretyczny} - T_{praktyczny}|}{n} \tag{56}$$

$$Średnia\ różnica = \frac{6.34 + 7.2 + \cdots + 6.98 + 9.37}{10} = 7.69\ [s] \tag{57}$$

Średni błąd procentowy

$$Średni\ błąd\ procentowy = \frac{\Sigma\ Błąd\ procentowy}{n} \tag{58}$$

$$Średni\ błąd\ procentowy = \frac{7.85 + \cdots + 8.64}{10} = 9.59\ [\%] \tag{59}$$

4.Odchylenie standardowe czasów praktycznych

$$\sigma = \sqrt{\frac{\Sigma(T_{praktyczny} - \overline{T_{praktyczny}})^2}{n}} \tag{60}$$

Średnia czasu praktycznego

$$\overline{T_{praktyczny}} = \frac{\Sigma\ T_{praktyczny}}{n} \tag{61}$$

$$\overline{T_{praktyczny}} = \frac{87.10 + 87{,}96 + \cdots + 90.13}{10} = 88.45\ [s] \tag{62}$$

Odchylenie standardowe liczone ze wzoru (60)

$$\sigma = \sqrt{\frac{(87.10 - 88.45)^2 + \cdots + (90.13 - 88.45)^2}{10}} = 0.88\ [s] \tag{63}$$

Podsumowanie wyników

- Średnia różnica (błąd absolutny) 7.69 [s]
- Średni błąd procentowy 9.59 [%]
- Odchylenie standardowe praktycznych czasów 0.88 [s]

6.7.3 Wnioski z przeprowadzonych badań

Podczas przeprowadzania badań zaobserwowano, że czasy przejazdu robota zwiększały się wraz z każdą kolejną próbą. W szczególności zaobserwowane zmiany czasów przejazdu wskazywały na wydłużenie czasu o około 1 z każdą kolejną próbą. Analiza sytuacji doprowadziła do wniosku, że problem mógł wynikać z niedostatecznego dostarczania mocy przez akumulatory.

W robocie mobilnym zasilanie akumulatorowe odpowiada za działanie wszystkich kluczowych podzespołów, w tym mikrokontrolera (Arduino), sterownia silników, czujników oraz samych silników. Gdy poziom naładowania akumulatorów spada, zmniejsza się wydajność dostarczania energii, co wpływa przede wszystkim na napęd silników, a w konsekwencji na prędkość robota.

Po piątej próbie akumulatory zostały doładowane, co skutkowało znacznym zmniejszeniem czasu przejazdu robota w kolejnych próbach, przywracając go do poziomu zbliżonego do wyników początkowych. Tym samym potwierdzono hipotezę, że regularne ładowanie akumulatorów jest kluczowe dla zachowania powtarzalności wyników.

Dla zapewnienia zbliżonych warunków testowych w przyszłych eksperymentach konieczne jest przeprowadzanie ładowania akumulatorów co kilka prób, aby unikać spadków mocy zasilania.

Podsumowanie wyników badań

Wyniki obliczeń pozwoliły na określenie jakości i stabilności ruchu robota

- Średnia różnica (błąd absolutny) 7.69s
  Oznacza to, że robot średnio przejeżdża trasę o 7.69s dłużej niż wynosi zakładany teoretyczny czas
- Średni błąd procentowy 9.59%
  Robot w każdej próbie odchylał się od teoretycznego czasu średnio o 9.59%
- Odchylenie standardowe praktycznych czasów 0.88s
  Wskazuje to na dobrą powtarzalność wyników po ujednoliceniu warunków testowych (ładowanie akumulatorów).

Regularne ładowanie akumulatorów jest kluczowe dla utrzymania stałej wydajności robota mobilnego. Eksperymenty wykazały, że brak odpowiedniego poziomu naładowania prowadzi do zmniejszenia prędkości silników, a tym samym wydłużenia czasu przejazdu. Po zastosowaniu tej procedury, wyniki stały się bardziej stabilne i powtarzalne, co zostało potwierdzone małym odchyleniem standardowym praktycznych czasów. Dodatkowo stwierdzono, że obniżenie poziomu naładowania akumulatorów skutkuje także pogorszeniem precyzji manewrów omijania przeszkód. W takiej sytuacji robot może nie wykonać planowanego obrotu o odpowiedni kąt, co prowadzi do ryzyka kolizji z przeszkodą. Problem ten jest szczególnie istotny w przypadku braku systemu wizyjnego, który mógłby dodatkowo wspierać proces nawigacji.

Podsumowując, regularne ładowanie akumulatorów w robocie mobilnym jest niezbędne dla uzyskania stabilnych wyników i minimalizacji odchyleń od założonych parametrów. Wyniki eksperymentów potwierdziły również, że po zastosowaniu tej procedury robot osiąga wysoką powtarzalność czasów przejazdu, co zostało potwierdzone niską wartością odchylenia standardowego. Równocześnie należy zwrócić uwagę na wpływ poziomu naładowania akumulatorów na skuteczność wykonywania manewrów, co może bezpośrednio wpływać na zdolność robota do omijania przeszkód i bezpiecznego poruszania się po trasie.

7.Wnioski oraz przyszłe plany

7.1 Wnioski

W ramach pracy inżynierskiej zrealizowano projekt robota typu line follower z funkcją detekcji przeszkód. Celem pracy było skonstruowanie w pełni funkcjonalnego urządzenia, które jest w stanie podążać za linią oraz omijać napotkane przeszkody. Efektem końcowym jest działający robot, spełniający wszystkie założenia projektu, co potwierdziło skuteczność zastosowanych rozwiązań technicznych i algorytmicznych.

W trakcie pracy przeprowadzono szereg badań i analiz, które miały na celu optymalizację działania robota. Badania obejmowały

- Analizę zależności czasów przejazdu robota od różnych wartości PWM
- Dogłębną analizę wybranego przypadku, która umożliwiła określenie najbardziej efektywnej wartości PWM oraz parametrów dotyczących szerokości przeszkód i ich detekcji.

Dzięki przeprowadzonym badaniom i analizom, wypracowano rozwiązania, które zapewniły optymalne działanie robota. Kryterium optymalności w tym przypadku odnosiło się do kilku kluczowych aspektów funkcjonowania robota:

- Płynność poruszania się po wyznaczonej trasie: Robot miał podążać za linią bez niepotrzebnych opóźnień czy gwałtownych ruchów. Optymalizacja parametrów PWM umożliwiła dobranie wartości, które zapewniły płynne sterowanie silnikami i odpowiednią prędkość ruchu, dostosowaną do możliwości czujników oraz dokładności detekcji przeszkód.
- Skuteczność omijania przeszkód: Działanie robota było oceniane pod kątem poprawności wykrywania przeszkód w różnych warunkach oraz efektywności podejmowanych manewrów. Optymalne rozwiązanie to takie, które minimalizowało czas potrzebny na omijanie przeszkody, jednocześnie zapewniając bezpieczeństwo ruchu.

Kryteria te były ustalane na podstawie założeń projektowych oraz wniosków z badań, dzięki czemu robot spełniał swoje zadania w sposób wydajny.

Podczas realizacji projektu zdobyłem cenne doświadczenie oraz pogłębiłem wiedzę z zakresu elektroniki, programowania oraz konstruowania systemów mechatronicznych. Projekt wymagał zastosowania wiedzy teoretycznej zdobytej podczas studiów, w tym szczególnie z przedmiotów takich jak podstawy automatyki, robotyka oraz elektronika analogowa i cyfrowa. Umiejętność projektowania układów elektronicznych, implementacji algorytmów sterowania oraz integracji różnych komponentów hardware'owych i software'owych okazały się kluczowe dla sukcesu przedsięwzięcia.

W trakcie realizacji projektu zmierzyłem się z wieloma wyzwaniami, takimi jak dostosowanie odpowiednich czujników, optymalizacja algorytmów sterowania, projektowanie oraz druk 3D, który nie okazał się w rzeczywistości najprostszą sprawą. Czasami niedokładnie zaprojektowane otwory w konstrukcji uniemożliwiały wmontowanie niektórych elementów, co zmusiło mnie do przerobienia projektu i ponownego wydruku, tym razem z lekko poszerzonymi wymiarami, aby dało się wpasować elementy. W trakcie realizacji projektu zostały spalone dwa czujniki z powodu niewłaściwie dobranego napięcia zasilania, co uświadomiło mi, jak ważne jest dokładne czytanie dokumentacji oraz ostrożność przy łączeniu ich. Niezbędna okazała się także nabyta umiejętność lutowania, którą rozwinąłem między innymi lutując przewody do silników.

Efekt końcowy, w postaci w pełni funkcjonalnego robota, jest zgodny z założonymi celami i spełnia moje oczekiwania. Jestem zadowolony z osiągniętych rezultatów oraz działania zbudowanego urządzenia.

7.2 Przyszłe plany rozwoju projektu

Mimo że cel pracy został zrealizowany, projekt posiada duży potencjał do dalszego rozwoju. Szczególną uwagę należy skupić na kierunkach, które mogą znacząco podnieść funkcjonalność i efektywność robota. Jednym z kluczowych obszarów rozwoju jest integracja systemu wizyjnego. Dodanie kamery oraz implementacja algorytmów analizy obrazu umożliwiłoby precyzyjne wykrywanie przeszkód, monitorowanie otoczenia w czasie rzeczywistym oraz identyfikację obiektów na trasie. Wdrożenie takiego systemu wymagałoby wyboru odpowiedniego sprzętu, na przykład kamery kompatybilnej z mikrokontrolerem lub dodatkową platformą obliczeniową, taką jak Raspberry Pi, oraz zaimplementowania algorytmów, np. detekcji krawędzi lub klasyfikacji obiektów przy użyciu bibliotek takich jak OpenCV.

Kolejnym ważnym krokiem w rozwoju projektu może być wyposażenie robota w czujnik LIDAR. Takie urządzenie pozwoliłoby na dokładne mapowanie otoczenia oraz lepsze wykrywanie przeszkód w trudnych warunkach, gdzie tradycyjne czujniki ultradźwiękowe lub podczerwieni mogą zawodzić. Implementacja LIDAR-u wymagałaby integracji jego danych z istniejącymi algorytmami sterowania oraz kalibracji w celu zapewnienia dokładności w różnorodnych scenariuszach.

Istotnym aspektem rozwoju są również zaawansowane algorytmy sterowania. Wdrożenie regulatora PID z adaptacyjnymi parametrami mogłoby znacząco poprawić reakcję robota na zmienne warunki na trasie. Alternatywnie, zastosowanie metod opartych na sztucznej inteligencji, takich jak algorytmy genetyczne, pozwoliłoby na dynamiczną optymalizację ścieżki oraz bardziej efektywne omijanie przeszkód.

Dalszym krokiem w rozwoju projektu mogłoby być rozszerzenie funkcji robota o możliwość autonomicznej nawigacji. Implementacja algorytmów planowania trasy, takich jak A* lub Dijkstra, umożliwiłaby robotowi samodzielne wyznaczanie optymalnej ścieżki do celu. Taka funkcja, w połączeniu z bardziej zaawansowanymi czujnikami, zwiększyłaby funkcjonalność robota i poszerzyłaby zakres jego potencjalnych zastosowań.

Podsumowując, największy nacisk powinien być położony na integrację systemu wizyjnego oraz zastosowanie czujnika LIDAR, gdyż te technologie najbardziej zwiększą precyzję i możliwości robota. Jednocześnie rozwijanie algorytmów sterowania i nawigacji pozwoli w pełni wykorzystać potencjał nowego sprzętu, co umożliwi stworzenie bardziej zaawansowanego i niezawodnego urządzenia.

8.Bibliografia

[1] M. Čech *et al.*, "AUTONOMOUS MOBILE ROBOT TECHNOLOGY FOR SUPPLYING ASSEMBLY LINES IN THE AUTOMOTIVE INDUSTRY," *Acta logistica*, vol. 7, no. 2, pp. 103–109, Jun. 2020, doi: 10.22306/al.v7i2.164.

[2] P. Dario, E. Guglielmelli, and B. Allotta, "Robotics in medicine," in *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'94)*, IEEE, pp. 739–752. doi: 10.1109/IROS.1994.407554.

[3] X. Gao *et al.*, "Review of Wheeled Mobile Robots' Navigation Problems and Application Prospects in Agriculture," *IEEE Access*, vol. 6, pp. 49248–49268, 2018, doi: 10.1109/ACCESS.2018.2868848.

[4] D. Xie, Y. Xu, and R. Wang, "Obstacle detection and tracking method for autonomous vehicle based on three-dimensional LiDAR," *Int J Adv Robot Syst*, vol. 16, no. 2, Mar. 2019, doi: 10.1177/1729881419831587.

[5] N. Saqib and M. M. Yousuf, "Design and Implementation of Shortest Path Line Follower Autonomous Rover Using Decision Making Algorithms," in *2021 Asian Conference on Innovation in Technology (ASIANCON)*, IEEE, Aug. 2021, pp. 1–6. doi: 10.1109/ASIANCON51346.2021.9544672.

[6] M. Pakdaman and M. M. Sanaatiyan, "Design and Implementation of Line Follower Robot," in *2009 Second International Conference on Computer and Electrical Engineering*, IEEE, 2009, pp. 585–590. doi: 10.1109/ICCEE.2009.43.

[7] A. Bârsan, "Position Control of a Mobile Robot through PID Controller," *Acta Universitatis Cibiniensis. Technical Series*, vol. 71, no. 1, pp. 14–20, Dec. 2019, doi: 10.2478/aucts-2019-0004.

[8] J. Azeta, C. Bolu, D. Hinvi, and A. A. Abioye, "Obstacle detection using ultrasonic sensor for a mobile robot," *IOP Conf Ser Mater Sci Eng*, vol. 707, no. 1, p. 012012, Nov. 2019, doi: 10.1088/1757-899X/707/1/012012.

[9] M. C. De Simone, Z. B. Rivera, and D. Guida, "Obstacle Avoidance System for Unmanned Ground Vehicles by Using Ultrasonic Sensors," *Machines*, vol. 6, no. 2, p. 18, Apr. 2018, doi: 10.3390/machines6020018.

[10] A. N. A. Rafai, N. Adzhar, and N. I. Jaini, "A Review on Path Planning and Obstacle Avoidance Algorithms for Autonomous Mobile Robots," *Journal of Robotics*, vol. 2022, pp. 1–14, Dec. 2022, doi: 10.1155/2022/2538220.

[11] M. I. A. R. E. P. A. M. Sodik Kirono, “ Kirono S. , Arifianto M. , Putra R. i Musoleh A. , Modelowanie oparte na grafach i algorytm Dijkstry do wyszukiwania tras pojazdów na autostradach , International Journal of Mechanical Engineering & Technology . (2018) 9 , nr 8, 12731280.,” *Stikom Poltek Cirebon, Cirebon, West Java 45153, Indonesia* .

[12] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, “Probabilistic roadmaps for path planning in high-dimensional configuration spaces,” *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996, doi: 10.1109/70.508439.

[13] P. N. SJ Russell, “Sztuczna inteligencja: nowoczesne podejście,” *1995 , Prentice-Hall International, Hoboken, New Jersey, Stany Zjednoczone* ..

[14] O. Khatib, “Real-time obstacle avoidance for manipulators and mobile robots,” in *Proceedings. 1985 IEEE International Conference on Robotics and Automation*, Institute of Electrical and Electronics Engineers, pp. 500–505. doi: 10.1109/ROBOT.1985.1087247.

[15] D. Rachmawati and L. Gustin, “Analysis of Dijkstra’s Algorithm and A* Algorithm in Shortest Path Problem,” *J Phys Conf Ser*, vol. 1566, no. 1, p. 012061, Jun. 2020, doi: 10.1088/1742-6596/1566/1/012061.

[16] L. A. Zadeh, “Fuzzy sets,” *Information and Control*, vol. 8, no. 3, pp. 338–353, Jun. 1965, doi: 10.1016/S0019-9958(65)90241-X.

[17] M. Zacksenhouse, R. J. P. deFigueiredo, and D. H. Johnson, “A neural network architecture for cue-based motion planning,” in *Proceedings of the 27th IEEE Conference on Decision and Control*, IEEE, pp. 324–327. doi: 10.1109/CDC.1988.194321.

[18] J. Kennedy and R. Eberhart, “Particle swarm optimization,” in *Proceedings of ICNN’95 - International Conference on Neural Networks*, IEEE, pp. 1942–1948. doi: 10.1109/ICNN.1995.488968.

[19] J. Chen, H. Zhu, L. Zhang, and Y. Sun, “Research on fuzzy control of path tracking for underwater vehicle based on genetic algorithm optimization,” *Ocean Engineering*, vol. 156, pp. 217–223, May 2018, doi: 10.1016/j.oceaneng.2018.03.010.

[20] A. Gautam and S. Mohan, “A review of research in multi-robot systems,” in *2012 IEEE 7th International Conference on Industrial and Information Systems (ICIIS)*, IEEE, Aug. 2012, pp. 1–5. doi: 10.1109/ICIInfS.2012.6304778.

[21] J. M. Mirats Tur and C. F. Pfeiffer, “Mobile robot design in education,” *IEEE Robot Autom Mag*, vol. 13, no. 1, pp. 69–75, Mar. 2006, doi: 10.1109/MRA.2006.1598055.

[22] S. B. Nickerson, “An autonomous mobile robot for known industrial environments,” *Ontario Hydro Technologies, Toronto, Canada, August 28, 1997*.

[23] R. S. , A. A. A. and A. K. A. Ali, “Design an optimal PID controller using artificial bee colony and genetic algorithm for autonomous mobile robot.,” *International Journal of Computer Applications 100.16 (2014): 8-16*.

[24] L. et al Almeida, “Mobile robot competitions: fostering advances in research, development and education in robotics,” *CONTROLO, Conf. on Automatic Control. Vol. 1. No. 1. Portugal: University of Minho, 2000*.

[25] P. Valsalan and P. Surendran, “Implementation of an Emergency Indicating Line Follower and Obstacle Avoiding Robot,” in *2019 16th International Multi-Conference on Systems, Signals & Devices (SSD)*, IEEE, Mar. 2019, pp. 479–482. doi: 10.1109/SSD.2019.8893241.

[26] S. et al Khan, “Waypoint navigation system implementation via a mobile robot using global positioning system (GPS) and global system for mobile communications (GSM) modems,” *International Journal of Computational Engineering Research (IJCER) 3.7 (2013): 1-7*.

[27] M. R. Elara, N. Rojas, and A. Chua, “Design principles for robot inclusive spaces: A case study with Roomba,” in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, May 2014, pp. 5593–5599. doi: 10.1109/ICRA.2014.6907681.

[28] A. Stefek, T. Van Pham, V. Krivanek, and K. L. Pham, “Energy Comparison of Controllers Used for a Differential Drive Wheeled Mobile Robot,” *IEEE Access*, vol. 8, pp. 170915–170927, 2020, doi: 10.1109/ACCESS.2020.3023345.

Zasada działania czujnika odbiciowego [Rysunek 15]

https://www.researchgate.net/publication/350192921_Design_And_Construction_Of_The_Circuits_For_An_Iot-Based_Stand-Alone_Solar_Powered_Street_Light_With_Vandalisation_Monitoring_And_Tracking_Mechanism/figures?lo=1

Zasada działania czujnika ultradźwiękowego [Rysunek 13]

https://www.researchgate.net/publication/352467742_Internet_of_Things_Assisted_Monitoring_based_on_a_Ultrasound-based_Gesture_Recognition_Contactless_System/figures?lo=1

Komponenty mechaniczne, sterujące oraz czujniki

https://botland.com.pl/?cd=1654592848&ad=67274996601&kd=botland&gad_source=1&gclid=CjwKCAiApY-7BhBjEiwAQMrrEecBUp2ckdGe_uzPQTEP9pu9KkP4To408O8BfA8qgujKJnGMen4ybxoCdboQAvD_BwE

Platrofma Arduino

https://www.arduino.cc/

Oprogramowanie Visual Studio

https://code.visualstudio.com/

Oprogramowanie SolidWorks

https://dps-software.pl/projektowanie/solidworks-3d-cad/?utm_source=google&utm_medium=cpc&utm_campaign=Solidworks_Search&gad_source=1&gclid=CjwKCAiApY-7BhBjEiwAQMrrERfBkZFUvT6PZhPXBtViC8-U_zD1tcxVP9y1RRhge1Oy4z6kawRjZRoChN4QAvD_BwE

Drawio.io

https://app.diagrams.net/

Komponenty elektryczne oraz ich rysunki:

https://botland.com.pl/koszyk

Schematy elektryczne:

https://www.kicad.org/

9.Załączniki

Kod programu sterującego

Definicje pinów oraz zmiennych globalnych

```
#define IR_LEFT A1              // Czujnik IR po lewej stronie
#define IR_RIGHT A0             // Czujnik IR po prawej stronie
#define ULTRASONIC_ECHO A2      // Pin Echo dla czujnika ultradźwiękowego
#define ULTRASONIC_TRIGGER A3   // Pin Trigger dla czujnika ultradźwiękowego
#define SERVO_PIN A5            // Pin sterujący serwomechanizmem
#define MOTOR_A_EN 10           // Pin Enable1 dla sterownika L298
#define MOTOR_A_IN1 9           // Pin IN1 dla silnika A
#define MOTOR_A_IN2 8           // Pin IN2 dla silnika A
#define MOTOR_B_IN1 7           // Pin IN1 dla silnika B
#define MOTOR_B_IN2 6           // Pin IN2 dla silnika B
#define MOTOR_B_EN 5            // Pin Enable2 dla sterownika L298

int thresholdDistance = 10;     // Minimalna odległość do przeszkody w cm
int leftDistance, rightDistance, frontDistance; // Odległości od przeszkód

bool ignoreRightOnce = false;   // Flaga ignorowania przeszkód z prawej strony
bool ignoreLeftOnce = false;    // Flaga ignorowania przeszkód z lewej strony
unsigned long ignoreStartTime;  // Czas rozpoczęcia ignorowania przeszkody
const unsigned long ignoreDuration = 2000; // Czas ignorowania przeszkody (ms)
```

Definicje pinów określają, do jakich portów mikrokontrolera są podłączone elementy takie jak czujniki, silniki czy serwomechanizm. Zmienne globalne przechowują dane związane z przeszkodami oraz logiką ich omijania, w tym thresholdDistance, która ustala minimalną bezpieczną odległość od przeszkody, oraz flagi ignoreRightOnce i ignoreLeftOnce, które są używane do czasowego ignorowania przeszkód w określonych sytuacjach.

Inicjalizacja w funkcji setup()

```
void setup() {
  Serial.begin(9600);         // Inicjalizacja portu szeregowego
```

```
    pinMode(IR_LEFT, INPUT);     // Czujnik IR po lewej stronie jako wejście
    pinMode(IR_RIGHT, INPUT);    // Czujnik IR po prawej stronie jako wejście
    pinMode(ULTRASONIC_ECHO, INPUT); // Czujnik ultradźwiękowy - Echo
    pinMode(ULTRASONIC_TRIGGER, OUTPUT); // Czujnik ultradźwiękowy - Trigger
    pinMode(SERVO_PIN, OUTPUT);  // Serwomechanizm jako wyjście
    pinMode(MOTOR_B_EN, OUTPUT); // Sterownik silnika B - Enable
    pinMode(MOTOR_A_IN1, OUTPUT);// Sterownik silnika A - IN1
    pinMode(MOTOR_A_IN2, OUTPUT);// Sterownik silnika A - IN2
    pinMode(MOTOR_B_IN1, OUTPUT);// Sterownik silnika B - IN1
    pinMode(MOTOR_B_IN2, OUTPUT);// Sterownik silnika B - IN2
    pinMode(MOTOR_A_EN, OUTPUT); // Sterownik silnika A - Enable

    analogWrite(MOTOR_B_EN, 170); // Ustawienie początkowej prędkości silnika B
    analogWrite(MOTOR_A_EN, 170); // Ustawienie początkowej prędkości silnika A

    for (int angle = 70; angle <= 145; angle += 5) {
      moveServo(SERVO_PIN, angle); // Kalibracja serwa w zakresie 70°-145°
    }
    for (int angle = 145; angle >= 0; angle -= 5) {
      moveServo(SERVO_PIN, angle); // Kalibracja serwa w zakresie 145°-0°
    }
    for (int angle = 0; angle <= 70; angle += 5) {
      moveServo(SERVO_PIN, angle); // Kalibracja serwa w zakresie 0°-70°
    }

    frontDistance = readUltrasonic(); // Pierwszy odczyt odległości z przodu
    delay(500);                       // Odstęp czasowy dla stabilizacji czujników
}
```

Funkcja setup() wykonuje wszystkie konieczne ustawienia początkowe, takie jak inicjalizacja pinów jako wejścia lub wyjścia w zależności od funkcji urządzenia, kalibracja serwomechanizmu w celu przygotowania go do pracy w pełnym zakresie kątów, ustawienie domyślnej prędkości dla silników za pomocą sygnałów PWM, oraz kalibracja i opóźnienia na końcu funkcji, które zapewniają stabilne działanie czujników i serwomechanizmu.

Główna pętla programu loop()

```
void loop() {
   frontDistance = readUltrasonic(); // Odczyt odległości z przodu
   Serial.print("Front Distance="); Serial.println(frontDistance); // Wyświetlenie odległości w konsoli

   if (ignoreLeftOnce && (millis() - ignoreStartTime > ignoreDuration)) {
      ignoreLeftOnce = false; // Wyłączenie ignorowania lewej strony
   }
   if (ignoreRightOnce && (millis() - ignoreStartTime > ignoreDuration)) {
      ignoreRightOnce = false; // Wyłączenie ignorowania prawej strony
   }

   if (!ignoreRightOnce && (digitalRead(IR_RIGHT) == 1) && (digitalRead(IR_LEFT) == 0)) {
      turnRight(); // Skręt w prawo, gdy prawy czujnik wykrywa linię
       } else if (!ignoreLeftOnce && (digitalRead(IR_RIGHT) == 0) && (digitalRead(IR_LEFT) == 1)) {
      turnLeft();  // Skręt w lewo, gdy lewy czujnik wykrywa linię
   } else if ((digitalRead(IR_RIGHT) == 1) && (digitalRead(IR_LEFT) == 1)) {
      stopMovement(); // Zatrzymanie, gdy oba czujniki wykrywają linię
   } else {
      if (frontDistance > thresholdDistance) {
         moveForward(); // Ruch do przodu przy braku przeszkód
      } else {
         checkSides(); // Analiza boków przy przeszkodzie
      }
   }
   delay(10); // Krótka przerwa
}
```

Główna pętla programu stale monitoruje odczyty czujników, sterując robotem poprzez obsługę flag ignorowania (ignoreRightOnce, ignoreLeftOnce), podejmowanie decyzji o ruchu w zależności od odczytów czujników IR (np. skręcanie lub zatrzymanie), analizowanie otoczenia przy wykryciu przeszkody z przodu (checkSides) oraz zastosowanie opóźnienia delay(10) w celu zapobiegania przeciążeniu mikrokontrolera.

Funkcje wspierające

Funkcje wspierające stanowią kluczowy element oprogramowania robota, umożliwiając mu wykonywanie podstawowych operacji ruchowych oraz interakcję z otoczeniem. Dzięki tym funkcjom, robot jest w stanie podejmować decyzje o kierunku ruchu, unikać przeszkód i precyzyjnie sterować swoimi komponentami, takimi jak silniki czy serwomechanizmy.

moveServo()

```
void moveServo(int pin, int angle) {
  int pwm = (angle * 11) + 500; // Obliczenie sygnału PWM dla danego kąta
  digitalWrite(pin, HIGH);      // Ustawienie pinu w stan wysoki
  delayMicroseconds(pwm);       // Wysłanie impulsu PWM
  digitalWrite(pin, LOW);       // Ustawienie pinu w stan niski
  delay(50);                    // Krótka przerwa między sygnałami
}
```

Funkcja przekształca kąt w impuls PWM, który ustawia serwomechanizm w odpowiedniej pozycji, przy czym czas trwania impulsu obliczany na podstawie kąta w stopniach jest realizowany przez digitalWrite zmieniający stan pinu i delayMicroseconds utrzymujący sygnał przez odpowiedni czas.

readUltrasonic()

```
long readUltrasonic() {
  digitalWrite(ULTRASONIC_TRIGGER, LOW); // Trigger na niski stan
  delayMicroseconds(2);                  // Krótka przerwa
  digitalWrite(ULTRASONIC_TRIGGER, HIGH);// Trigger na wysoki stan
  delayMicroseconds(10);                 // Impuls trwający 10 µs
  long duration = pulseIn(ULTRASONIC_ECHO, HIGH); // Czas trwania Echo
  return duration / 29 / 2;              // Przeliczenie na odległość w cm
```

```
}
```

Funkcja wysyła impuls na pin Trigger, mierzy czas odbicia sygnału na pin Echo, a następnie przelicza wynik na odległość w centymetrach, uwzględniając prędkość dźwięku.

checkSides()

```
void checkSides() {
  moveServo(SERVO_PIN, 145);         // Przesunięcie serwa w lewo
  delay(500);                        // Czekanie na stabilizację
  leftDistance = readUltrasonic();   // Odczyt odległości z lewej strony

  moveServo(SERVO_PIN, 0);           // Przesunięcie serwa w prawo
  delay(500);                        // Czekanie na stabilizację
  rightDistance = readUltrasonic();  // Odczyt odległości z prawej strony

  moveServo(SERVO_PIN, 70);          // Ustawienie serwa na środek
  delay(500);                        // Stabilizacja

  if (leftDistance > thresholdDistance) {
    ignoreLeftOnce = true;           // Ignorowanie przeszkód z lewej strony
    ignoreStartTime = millis();      // Ustawienie czasu rozpoczęcia ignorowania
    turnLeft();                      // Wykonanie skrętu w lewo
  } else if (rightDistance > thresholdDistance) {
    ignoreRightOnce = true;          // Ignorowanie przeszkód z prawej strony
    ignoreStartTime = millis();      // Ustawienie czasu rozpoczęcia ignorowania
    turnRight();                     // Wykonanie skrętu w prawo
  } else {
    stopMovement();                  // Zatrzymanie robota, jeśli obie strony są zajęte
  }
}
```

Funkcja steruje serwem w celu sprawdzenia odległości przeszkód po bokach: na lewej stronie ustawia serwo na 145° i wykonuje odczyt za pomocą readUltrasonic(), a na prawej stronie ustawia serwo na 0° i wykonuje ponowny odczyt. Na podstawie wyników decyduje, w

którą stronę robot ma się poruszać (skręt w lewo lub prawo) lub zatrzymuje robot w przypadku braku bezpiecznych opcji.

checkSides()

```
void checkSides() {  // Funkcja sprawdzająca odległości z lewej i prawej strony robota
   stopMovement();   // Zatrzymanie ruchu robota przed wykonaniem pomiarów
   delay(100);       // Opóźnienie 100 ms, aby upewnić się, że robot się zatrzymał

   // Obrót serwomechanizmu od 70° do 145° co 5°
   for (int angle = 70; angle <= 145; angle += 5) {
      moveServo(SERVO_PIN, angle);  // Ruch serwomechanizmu do danego kąta
   }
   delay(300);  // Odczekanie 300 ms, aby dać czas serwomechanizmowi na wykonanie pełnego obrotu
   leftDistance = readUltrasonic();    // Odczyt odległości za pomocą czujnika ultradźwiękowego
   Serial.print("Left Distance=");   // Wydrukowanie wyniku na monitorze szeregowym
   Serial.println(leftDistance);     // Wydrukowanie odległości z lewej strony
   delay(100);  // Opóźnienie 100 ms przed kolejnym pomiarem

   // Obrót serwomechanizmu od 145° do 0° co 5°
   for (int angle = 145; angle >= 0; angle -= 5) {
      moveServo(SERVO_PIN, angle);  // Ruch serwomechanizmu do danego kąta
   }
   delay(500);  // Odczekanie 500 ms, aby dać czas serwomechanizmowi na wykonanie pełnego obrotu
   rightDistance = readUltrasonic();    // Odczyt odległości za pomocą czujnika ultradźwiękowego
   Serial.print("Right Distance=");   // Wydrukowanie wyniku na monitorze szeregowym
   Serial.println(rightDistance);     // Wydrukowanie odległości z prawej strony
   delay(100);  // Opóźnienie 100 ms przed kolejnym pomiarem

   // Obrót serwomechanizmu z powrotem od 0° do 70° co 5°
   for (int angle = 0; angle <= 70; angle += 5) {
```

```
            moveServo(SERVO_PIN, angle);  // Ruch serwomechanizmu do danego kąta
        }
        delay(300);  // Odczekanie 300 ms przed wykonaniem kolejnych działań

        // Warunek sprawdzający odległości
        if (leftDistance < thresholdDistance && rightDistance < thresholdDistance) {
            stopMovement();  // Jeśli obie odległości są mniejsze niż próg, zatrzymujemy ruch robota
        } else if (leftDistance > thresholdDistance && rightDistance > thresholdDistance) {
            turnRight();    // Jeśli obie odległości są większe niż próg, robot skręca w prawo
            delay(360);     // Odczekanie 360 ms na wykonanie skrętu
            moveForward();  // Robot rusza do przodu
            delay(1283);    // Odczekanie 1283 ms na kontynuację ruchu do przodu
            turnLeftOnly(); // Skręt tylko w lewo
            delay(736);     // Odczekanie 736 ms na wykonanie skrętu
            moveForward();  // Robot rusza do przodu
            delay(581);     // Odczekanie 581 ms na kontynuację ruchu do przodu
            turnLeftOnly(); // Skręt tylko w lewo
            delay(736);     // Odczekanie 736 ms na wykonanie skrętu

            ignoreLeftOnce = true;  // Ustawienie flagi, aby zignorować przyszłe pomiary dla lewej strony
            ignoreStartTime = millis();  // Zapisanie czasu rozpoczęcia ignorowania
        } else {
            evaluateDistances();  // Jeśli tylko jedna z odległości jest większa niż próg, wywołaj funkcję oceny odległości
        }
    }
```

Funkcja stopMovement() zatrzymuje ruch robota, umożliwiając pomiar odległości. Servo movement odpowiada za obrót serwomechanizmu w określonym zakresie (od 70° do 145° i z powrotem) w celu sprawdzenia odległości za pomocą czujnika ultradźwiękowego w różnych pozycjach. Ultrasonic distance measurement wykonuje pomiar odległości z lewej i prawej strony robota. Ocena odległości: jeżeli oba pomiary są mniejsze niż wartość progowa, robot

zatrzymuje się; jeśli oba są większe, robot skręca w prawo, wykonuje kilka ruchów, a potem ustawia flagę do ignorowania lewej strony. Inicjowanie ignorowania: jeśli robot ma ignorować pewne zachowanie w przyszłości, ustawia odpowiednią flagę i zapisuje czas.

evaluateDistances ()

```
void evaluateDistances() {  // Funkcja oceniająca, która strona ma większą odległość
  if (rightDistance > leftDistance) {  // Jeśli odległość po prawej stronie jest większa niż po lewej
    turnRight();    // Skręt w prawo
    delay(360);     // Odczekanie 360 ms na wykonanie skrętu
    moveForward();  // Robot rusza do przodu
    delay(1283);    // Odczekanie 1283 ms na kontynuację ruchu do przodu
    turnLeftOnly(); // Skręt tylko w lewo
    delay(736);     // Odczekanie 736 ms na wykonanie skrętu
    moveForward();  // Robot rusza do przodu
    delay(581);     // Odczekanie 581 ms na kontynuację ruchu do przodu
    turnLeftOnly(); // Skręt tylko w lewo
    delay(736);     // Odczekanie 736 ms na wykonanie skrętu

    ignoreLeftOnce = true;  // Ustawienie flagi, aby zignorować przyszłe pomiary dla lewej strony
    ignoreStartTime = millis();  // Zapisanie czasu rozpoczęcia ignorowania
  } else {  // Jeśli odległość po lewej stronie jest większa niż po prawej
    turnLeft();    // Skręt w lewo
    delay(360);    // Odczekanie 360 ms na wykonanie skrętu
    moveForward(); // Robot rusza do przodu
    delay(1283);   // Odczekanie 1283 ms na kontynuację ruchu do przodu
    turnRightOnly(); // Skręt tylko w prawo
    delay(736);     // Odczekanie 736 ms na wykonanie skrętu
    moveForward();  // Robot rusza do przodu
    delay(581);     // Odczekanie 581 ms na kontynuację ruchu do przodu
    turnRightOnly(); // Skręt tylko w prawo
    delay(736);     // Odczekanie 736 ms na wykonanie skrętu
```

```
            ignoreRightOnce = true;  // Ustawienie flagi, aby zignorować przyszłe pomiary dla prawej strony
            ignoreStartTime = millis();  // Zapisanie czasu rozpoczęcia ignorowania
        }
    }
```

Funkcja podejmuje decyzję o kierunku na podstawie porównania odległości wykrytych przez czujniki: jeśli prawy czujnik wykrywa większą odległość, robot skręca w prawo, a jeśli lewy czujnik wykrywa większą odległość, robot skręca w lewo. Wartości ignoreLeftOnce lub ignoreRightOnce są zwiększane, aby ustawić flagę ignorowania dla odpowiedniej strony, co zapobiega powtarzaniu niepotrzebnych ruchów.

moveForward ()

```
void moveForward() {
  digitalWrite(MOTOR_A_IN1, LOW);
  digitalWrite(MOTOR_A_IN2, HIGH);
  digitalWrite(MOTOR_B_IN1, HIGH);
  digitalWrite(MOTOR_B_IN2, LOW);
}
```

Funkcja steruje silnikami, aby robot poruszał się do przodu, ustawiając odpowiednie piny w stan wysoki lub niski, co powoduje obrót kół w odpowiednim kierunku.

moveBackward ()

```
void moveBackward() {
  digitalWrite(MOTOR_A_IN1, HIGH);
  digitalWrite(MOTOR_A_IN2, LOW);
  digitalWrite(MOTOR_B_IN1, LOW);
  digitalWrite(MOTOR_B_IN2, HIGH);
}
```

Funkcja steruje silnikami, aby robot poruszał się do tyłu. Zmiana stanu pinów powoduje, że koła obracają się w przeciwnym kierunku.

turnRight ()

```
void turnRight() {
   digitalWrite(MOTOR_A_IN1, HIGH);
   digitalWrite(MOTOR_A_IN2, LOW);
   digitalWrite(MOTOR_B_IN1, HIGH);
   digitalWrite(MOTOR_B_IN2, LOW);
}
```

Funkcja umożliwia robotowi skręt w prawo poprzez obrót odpowiednich kół.

turnRightOnly ()

```
void turnRightOnly() {
   digitalWrite(MOTOR_A_IN1, LOW);
   digitalWrite(MOTOR_A_IN2, LOW);
   digitalWrite(MOTOR_B_IN1, HIGH);
   digitalWrite(MOTOR_B_IN2, LOW);
}
```

Skręt tylko jednego koła (lewego) polega na włączeniu ruchu tylko lewego koła, co powoduje skręt robota w prawo.

turnLeft ()

```
void turnLeft() {
   digitalWrite(MOTOR_A_IN1, LOW);
   digitalWrite(MOTOR_A_IN2, HIGH);
   digitalWrite(MOTOR_B_IN1, LOW);
   digitalWrite(MOTOR_B_IN2, HIGH);
}
```

Funkcja umożliwia robotowi skręt w lewo poprzez obrót odpowiednich kół.

turnLeftOnly ()

```
void turnLeftOnly() {
   digitalWrite(MOTOR_A_IN1, LOW);
   digitalWrite(MOTOR_A_IN2, HIGH);
```

```
    digitalWrite(MOTOR_B_IN1, LOW);
    digitalWrite(MOTOR_B_IN2, LOW);
}
```

Skręt tylko jednego koła (prawego) polega na włączeniu ruchu tylko prawego koła, co powoduje skręt robota w lewo.

stopMovement ()

```
void stopMovement() {
    if (ignoreLeftOnce || ignoreRightOnce) {
        return;
    }

    digitalWrite(MOTOR_A_IN1, LOW);
    digitalWrite(MOTOR_A_IN2, LOW);
    digitalWrite(MOTOR_B_IN1, LOW);
    digitalWrite(MOTOR_B_IN2, LOW);
}
```

Funkcja zatrzymuje robota, wyłączając silniki. Jeśli robot ma zignorować zatrzymanie (w oparciu o wcześniejsze flagi ignoreLeftOnce lub ignoreRightOnce), funkcja nie wykonuje żadnej operacji.