



Politechnika
Śląska

PROJEKT INŻYNIERSKI

Budowa mapy otoczenia z wykorzystaniem robota mobilnego

Krzysztof GRĄDEK

Nr albumu: 300362

Kierunek: Automatyka i Robotyka

Specjalność: Technologie Informacyjne

PROWADZĄCY PRACĘ

dr inż. Krzysztof Jaskot

KATEDRA Katedra Automatyki i Robotyki

Wydział Automatyki, Elektroniki i Informatyki

Gliwice 2024

Tytuł pracy

Budowa mapy otoczenia z wykorzystaniem robota mobilnego

Streszczenie

Projekt koncentruje się na implementacji systemu autonomicznej nawigacji robota mobilnego, z naciskiem na dwa kluczowe aspekty: tworzenie mapy otoczenia oraz realizację precyzyjnej nawigacji z punktu do punktu w zmapowanej przestrzeni wykorzystując rozwiązanie typu SLAM. Rozwiązanie opiera się na dwóch współpracujących ze sobą mikrokontrolerach - Raspberry Pi 4, który odpowiada za obsługę czujnika RPLidar A1 oraz wykonywanie algorytmów mapowania i nawigacji, oraz Arduino Nano zarządzającym silnikami z enkoderami, zapewniającymi precyzyjne sterowanie ruchem robota. Wykonana konfiguracja zrealizowana została w językach C++ oraz Python, z wykorzystaniem narzędzi z ekosystemu ROS 2 (ang. "Robot Operating System 2"), takich jak Nav2 (ang. "Navigation 2"), SLAM Toolbox (ang. "Simultaneous Localization and Mapping Toolbox") oraz ROS2 Control (ang. "Robot Operating System 2 Control").

Słowa kluczowe

Mapowanie, robot mobilny, lokalizacja, SLAM, ROS

Thesis title

Construction of an Environment Map Using a Mobile Robot

Abstract

The project focuses on implementing an autonomous mobile robot navigation system, emphasizing two key aspects: environment mapping and precise point-to-point navigation in the mapped space using SLAM type solution. The solution is based on two cooperating microcontrollers - Raspberry Pi 4, which handles the RPLidar A1 sensor and executes mapping and navigation algorithms, and Arduino Nano managing motors with encoders, providing precise robot motion control. The implemented configuration was realized using C++ and Python languages, utilizing tools from the ROS 2 (Robot Operating System 2) ecosystem, such as Nav2 (Navigation 2), SLAM Toolbox (Simultaneous Localization and Mapping Toolbox) and ROS2 Control (Robot Operating System 2 Control).

Key words

Mapping, mobile robot, localization, SLAM, ROS

Spis treści

1 Wstęp	1
1.1 Wprowadzenie w problem	1
1.2 Osadzenie problemu w dziedzinie	2
1.3 Cel pracy	2
1.4 Zakres pracy	2
1.5 Struktura pracy	3
1.6 Wkład własny autora	3
2 Analiza tematu	5
2.1 Osadzenie tematu w kontekście aktualnego stanu wiedzy	5
2.2 Szczegółowe sformułowanie problemu	5
2.3 Studia literaturowe	6
2.4 ROS 2	6
2.5 SLAM Toolbox	7
2.5.1 Navigation2 (Nav2) i lokalizacja	8
2.5.2 ROS2 Control i sterowanie napędami	8
2.6 Wybór rozwiązań	8
3 Wymagania i narzędzia	9
3.1 Wymagania funkcjonalne	9
3.2 Przypadki użycia	10
3.3 Specyfikacja komponentów	11
3.3.1 Jednostki sterujące	11
3.3.2 Napęd	12
3.3.3 Zasilanie	13
3.4 Budowa robota i sposób połączenia silników z kontrolerem	15
3.5 Metodyka i etapy realizacji	18
3.5.1 Etap 1: Przygotowanie platformy sprzętowej	18
3.5.2 Etap 2: Implementacja sterowania napędem	18
3.5.3 Etap 3: Integracja sensorów	18
3.5.4 Etap 4: Implementacja oprogramowania	18

4 Specyfikacja użytkowa	19
4.0.1 Wymagania sprzętowe i programowe	19
4.0.2 Metoda instalacji oprogramowania i konfiguracji sprzętu	20
4.0.3 Sposób aktywacji i korzystania z robota z przykadem działania	21
4.1 Administracja systemem	28
4.2 Kwestie bezpieczeństwa	28
4.3 Scenariusze korzystania z systemu	29
5 Specyfikacja techniczna	31
5.1 Idea robota autonomicznego mapującego w czasie rzeczywistym	31
5.2 Architektura systemu	32
5.3 Struktura systemu i objaśnienie działania algorytmów	33
5.3.1 Skrypt Arduino	33
5.3.2 Skrypt do obsługi LiDAR-a	33
5.3.3 Skrypt do obsługi silników i tworzenia modelu robota	33
5.3.4 Skrypt do tworzenia mapy	33
5.3.5 Skrypty do nawigacji i lokalizacji	33
5.3.6 Diagramy UML prezentujące działanie konkretnych programów . .	34
6 Weryfikacja i walidacja	35
6.1 Model V	35
6.2 Organizacja eksperymentów	36
6.3 Przypadki testowe	36
6.4 Wykryte i usunięte błędy	36
6.5 Wyniki badań eksperymentalnych	37
7 Podsumowanie i wnioski	39
7.1 Uzyskane wyniki	39
7.2 Kierunki dalszych prac	39
Bibliografia	41
Spis skrótów i symboli	45
Źródła	47
Lista dodatkowych plików, uzupełniających tekst pracy	49
Spis rysункów	51
Spis tabel	53

Rozdział 1

Wstęp

Poniższy projekt obejmuje implementację systemu autonomicznej nawigacji robota mobilnego, z naciskiem na dwa kluczowe aspekty: tworzenie mapy otoczenia oraz realizację precyzyjnej nawigacji z punktu do punktu w zmapowanej przestrzeni. Tego typu zadania są kluczowe w dziedzinie robotyki mobilnej, umożliwiając robotom samodzielne poruszanie się w nowych nieznanych przestrzeniach. W tym rozdziale przedstawiono cel pracy, jej zakres oraz strukturę.

1.1 Wprowadzenie w problem

Jednoczesna lokalizacja i mapowanie - SLAM (ang. "Simultaneous localization and mapping") to proces, w którym robot konstruuje mapę nieznanego środowiska podczas jednoczesnej lokalizacji w tym środowisku i śledzenia swojej trajektorii poruszania się [1]. Jest to jedno z kluczowych zagadnień w robotyce mobilnej, umożliwiając robotom samodzielne poruszanie się w przestrzeni. W praktyce SLAM jest realizowany za pomocą zestawu sensorów, takich jak skanery laserowe, kamery RGB-D, czy IMU (ang. "Inertial Measurement Unit"), oraz algorytmów, które przetwarzają dane z tych sensorów w celu budowy mapy i lokalizacji robota.

1.2 Osadzenie problemu w dziedzinie

Ten projekt zalicza się do dziedziny robotyki mobilnej i systemów autonomicznych. Roboty mobilne są szeroko wykorzystywane w przemyśle, logistyce, czy badaniach naukowych. Realizacja systemu SLAM i autonomicznej nawigacji wymaga integracji wielu zaawansowanych technologii. Główne wyzwania techniczne obejmują wykorzystanie i synchronizację:

- Sensorów w tym LiDAR (ang. "Light Detection and Ranging")
- Algorytmów SLAM
- Platform programistycznych dla robotów jak ROS 2 (ang. "Robot Operating System 2")
- Systemów nawigacji jak Nav2
- Systemów sterowania jak ROS2 Control
- Systemów wizualizacji i analizy danych
- Systemów komunikacji i zarządzania danymi

1.3 Cel pracy

Głavnym celem pracy jest zaprojektowanie i implementacja systemu mapowania otoczenia z wykorzystaniem robota mobilnego. W ramach realizacji tego zadania przewidziano budowę platformy mobilnej, implementację systemu sterowania robotem oraz integrację niezbędnych czujników i urządzeń. Kluczowym elementem jest realizacja algorytmów SLAM, które umożliwiają jednoczesną lokalizację robota i tworzenie mapy otoczenia. Dodatkowo, system ma zapewniać możliwość nawigacji z punktu do punktu oraz sterowania manualnego.

1.4 Zakres pracy

Realizacja projektu obejmuje szereg wzajemnie powiązanych zadań. Na pierwszy etap składa się dogłębna analiza istniejących rozwiązań w dziedzinie mapowania i nawigacji robotów mobilnych, która stanowi podstawę do dalszych prac. Na tej bazie opracowany jest projekt systemu sterowania robotem, a następnie przeprowadzona jego implementacja. Kolejnym krokiem jest integracja komponentów sprzętowych i programowych w spójny system. Szczególną uwagę poświęcono implementacji algorytmów SLAM i nawigacji, które stanowią rdzeń funkcjonalności robota. Całość prac kończy seria testów i walidacja stworzonego rozwiązania.

1.5 Struktura pracy

Praca składa się z sześciu następujących rozdziałów:

- Rozdział pierwszy zawiera wstęp, w którym przedstawiono cel pracy, jej zakres oraz strukturę.
- Rozdział drugi zawiera analizę tematu, osadzenie go w kontekście aktualnego stanu wiedzy, analizę literatury, stan aktualny dziedziny oraz uzasadnienie wyboru rozwiązania.
- Rozdział trzeci zawiera wymagania i narzędzia, w którym opisano wymagania funkcjonalne, przypadki użycia, specyfikację komponentów, sposób połączenia sterowania i metodykę wraz z etapami realizacji projektu.
- Rozdział czwarty zawiera specyfikację użytkową, w którym przedstawiono wymagania użytkownika oraz specyfikację funkcjonalną.
- Rozdział piąty zawiera specyfikację techniczną, w którym przedstawiono podstawowe pojęcia i definicje z dziedziny robotyki mobilnej, jak i implementację zastosowanego rozwiązania.
- Rozdział szósty zawiera weryfikacje i walidacje, w którym przedstawiono testy i wyniki działania systemu.
- Rozdział siódmy zawiera podsumowanie, w którym przedstawiono wnioski z pracy oraz możliwości dalszego rozwoju projektu.

1.6 Wkład własny autora

W ramach pracy autor samodzielnie:

- Zaprojektował i zbudował platformę mobilną
- Zaimplementował sterowniki urządzeń
- Zintegrował komponenty sprzętowe i programowe
- Zaimplementował i dostosował algorytmy SLAM
- Przeprowadził testy i optymalizację systemu

Rozdział 2

Analiza tematu

W niniejszym rozdziale przedstawiono analizę problemu jednoczesnej lokalizacji i mapowania (SLAM) oraz autonomicznej nawigacji robotów mobilnych. Omówiono aktualny stan wiedzy w tej dziedzinie, sformułowano szczegółowo problem badawczy oraz dokonano przeglądu dostępnych rozwiązań i algorytmów. Na podstawie tej analizy wybrano optymalne narzędzia i metody do realizacji projektu.

2.1 Osadzenie tematu w kontekście aktualnego stanu wiedzy

Problem jednoczesnej lokalizacji, mapowania oraz autonomicznej nawigacji robotów mobilnych stanowi jeden z kluczowych obszarów badań w dziedzinie robotyki. W ostatnich latach obserwuje się znaczący postęp w tej dziedzinie, głównie dzięki rozwojowi wydajnych algorytmów optymalizacji, poprawie jakości i dostępności czujników jak LiDAR, wzrostowi mocy obliczeniowej komputerów oraz powstaniu zaawansowanych platform programistycznych jak ROS 2.

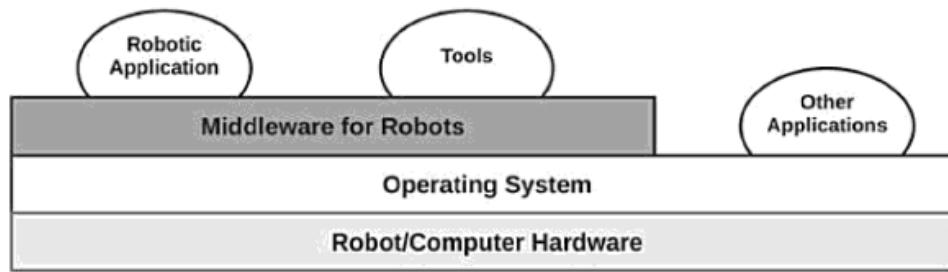
2.2 Szczegółowe sformułowanie problemu

Problem postawiony w niniejszej pracy obejmuje dwa główne aspekty. Pierwszy z nich to mapowanie otoczenia, które wymaga efektywnej akwizycji danych z czujników, przetwarzania chmur punktów pobranych z czujnika, estymacji pozycji robota oraz łączenia kolejnych skanów w spójną mapę. Drugi aspekt to autonomiczna nawigacja, gdzie system musi zapewniać precyzyjną lokalizację w znanej mapie, planowanie ścieżki z uwzględnieniem przeszkód oraz dokładną kontrolę ruchu robota.

2.3 Studia literaturowe

2.4 ROS 2

ROS2 (ang. "Robotic operation system 2") to platforma programistyczna dla robotów będąca oprogramowaniem pośrednim (ang. "middleware"), czyli warstwą programową pomiędzy systemem operacyjnym, a aplikacjami użytkownika do wykonywania oprogramowania aplikacji w domenie robota [10]. Na poniższej ilustracji przedstawiono położenie takiego pośrednika.

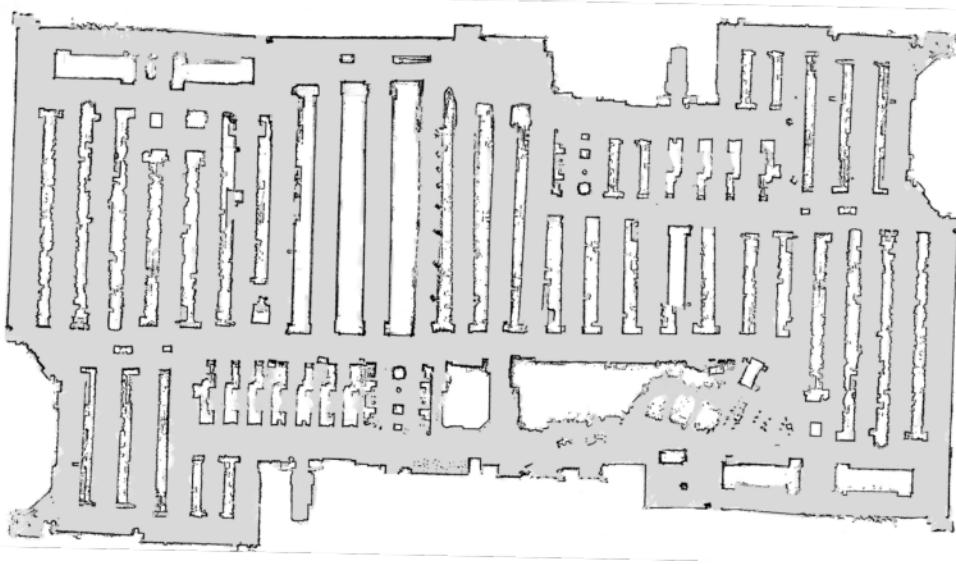


Rysunek 2.1: Reprezentacja pośrednika w systemie robota [10]

Zasadniczo ROS 2 to otwartoźródłowe oprogramowanie bazujące na usłudze dystrybucji danych - DDS (ang. "Data Distribution Service"), które dostarcza ustandaryzowane narzędzia do organizacji kodu aplikacji w modularne pakiety, zapewniania wspólnego wykonania kodu na wiele dostępnych plików wykonywalnych, oraz komunikację między tymi modułami podczas równoległego uruchomienia w systemie robota.[6]

2.5 SLAM Toolbox

SLAM Toolbox to zestaw narzędzi i rozwiązań do tworzenia map 2D w czasie rzeczywistym, stworzone przez Steve'a Mecenski. Stosowane algorytmy w przeszłości to np. GMapping, Karto, Cartographer oraz Hector, jednakże prawie żaden z nich nie potrafił tworzyć map w czasie rzeczywistym, jedynie Cartographer stworzony przez Google miał takie możliwości, lecz przestał być on wspierany. [7] Zastosowanie SLAM Toolbox pozwala na tworzenie map w czasie rzeczywistym obszarów, do nawet $24\ 000\ m^2$ przez niewykwalifikowanych użytkowników. Przykład działania SLAM Toolbox przedstawiono na rysunku poniżej.



Rysunek 2.2: Mapa sklepu utworzona za pomocą SLAM Toolbox [7]

SLAM Toolbox oferuje 3 główne tryby pracy:

- **Mapowanie synchroniczne** - Ten tryb pozwala na mapowanie i lokalizację w przestrzeni zachowując dane poprzednich pomiarów. Pozwala to na większą dokładność mapy kosztem szybkości i odporności na przerwania.
- **Mapowanie asynchroniczne** - W tym trybie mapowanie i lokalizacja odbywają się wyłącznie na podstawie aktualnych pomiarów gdy ostatnie pomiary zakończą się i zostanie spełnione kryterium dokładności. Pozwala to na szybsze i mniej podatne na przerwania mapowanie kosztem jakości.
- **Lokalizacja** - W tym trybie robot dopasowuje aktualne pomiary do istniejącej mapy w celu określenia swojej pozycji. System tworzy tymczasowe punkty odniesienia z nowych pomiarów, które są używane do precyzyjnej lokalizacji. Po określonym czasie te tymczasowe punkty są usuwane, przywracając oryginalną mapę. Tryb ten może również działać bez wcześniejszej mapy, wykorzystując tylko lokalne pomiary do nawigacji.

2.5.1 Navigation2 (Nav2) i lokalizacja

Nav2 jest to pakiet narzędzi do nawigacji robotów mobilnych w ROS 2. Zawiera on zestaw algorytmów do tworzenia modeli środowiska z sensorów, dynamicznego planowania ścieżki, obliczania prędkości silników i omijania przeszkód. Pakiet wykorzystuje drzewa zachowań (ang. "Behavior Trees") do definiowania zachowań robota, przez implementację wielu niezależnych zadań. Niektóre z nich odpowiadają za np. obliczanie trasy do celu, inne za naprawę błędów, a jeszcze inne za omijanie przeszkód. Dzięki komunikacji między tymi zadaniami, robot jest w stanie wykonywać skomplikowane zadania nawigacyjne. [8]

Do lokalizacji robota na mapie można wykorzystać wcześniej opisany SLAM Toolbox, jednak w pakuie Nav2 dostępny jest również pakiet AMCL (ang. "Adaptive Monte Carlo Localization"), który pozwala na lokalizację robota na mapie z wykorzystaniem filtra cząsteczkowego. Algorytm ten polega na generowaniu losowych próbek (cząsteczek) reprezentujących możliwe położenia robota, a następnie porównywaniu ich z pomiarami z czujników. Cząsteczki, które najlepiej pasują do pomiarów są wybierane, a reszta jest odrzucana. W ten sposób algorytm estymuje pozycję robota na mapie. [3]

2.5.2 ROS2 Control i sterowanie napędami

ROS2 Control to platforma do sterowania, zarządzania i komunikacji pomiędzy urządzeniami w robotach z oprogramowaniem. [2]. Dzięki takiemu rozwiązaniu można w łatwy sposób zarządzać silnikami, enkoderami, czy innymi urządzeniami w robocie. Dzięki temu można wykorzystać np. pakiet diffdrive_arduino, do sterowania robotem z napędem różnicowym za pomocą Arduino. Pakiet ten pozwala również na sterowanie prędkością silników, odczyt enkoderów, obliczanie odometrii oraz transformację między układem odometrii a układem bazowym. [9]

2.6 Wybór rozwiązań

Na podstawie analizy dostępnych narzędzi, w projekcie zdecydowano się na wykorzystanie SLAM Toolbox do mapowania, AMCL do lokalizacji podczas nawigacji, Nav2 do planowania ścieżki i kontroli ruchu oraz ROS2 Control z DiffDrive Arduino do sterowania napędami. Wybór ten podyktowany jest stabilnością rozwiązań, oraz dobrą integracją komponentów w ekosystemie ROS 2 oraz aktywnym wsparciem społeczności i dostępnością dokumentacji.

Rozdział 3

Wymagania i narzędzia

W niniejszym rozdziale przedstawiono wymagania funkcjonalne systemu, przypadki użycia w formie diagramu UML, szczegółową specyfikację wykorzystanych komponentów sprzętowych oraz metodykę i etapy realizacji projektu.

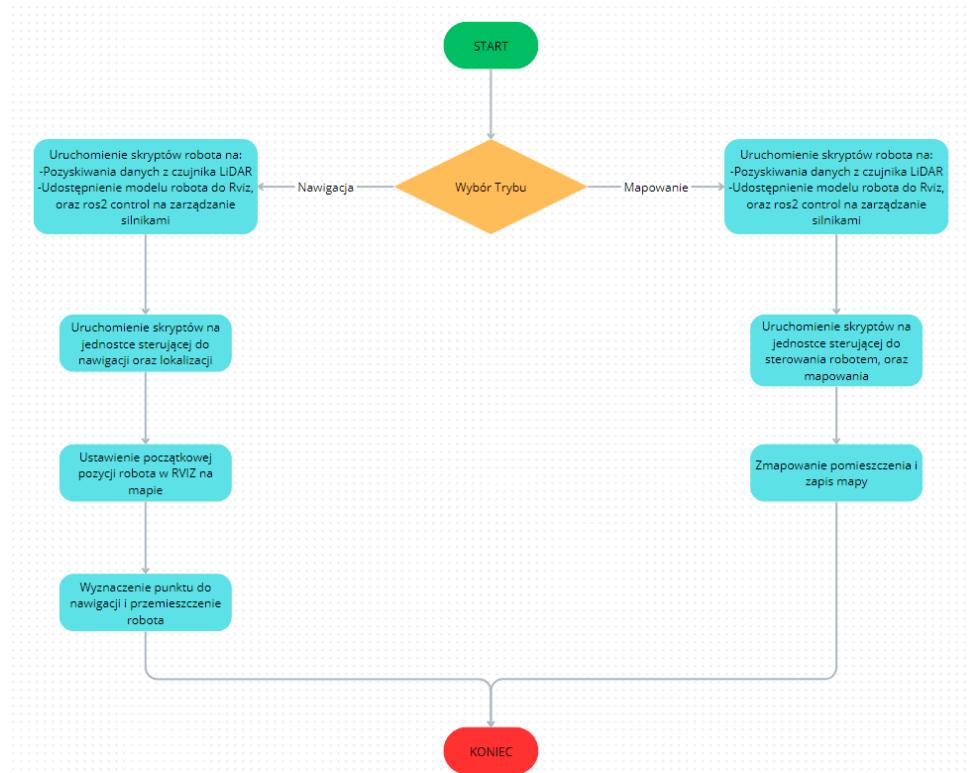
3.1 Wymagania funkcjonalne

System powinien realizować następujące funkcje:

- Zdalne sterowanie robotem poprzez klawiaturę (teleop_twist_keyboard [5]) w celu eksploracji i mapowania otoczenia
- Tworzenie i zapisywanie mapy otoczenia w czasie rzeczywistym
- Lokalizacja robota na zapisanej mapie z wykorzystaniem algorytmu AMCL
- Autonomiczna nawigacja do wyznaczonych punktów na mapie z omijaniem przeszkód

3.2 Przypadki użycia

Na rysunku 3.1 przedstawiono diagram przypadków użycia systemu. System umożliwia użytkownikowi zdalne sterowanie robotem, tworzenie mapy otoczenia, lokalizację robota na mapie oraz autonomiczną nawigację do wyznaczonych punktów.



Rysunek 3.1: Diagram przypadków użycia systemu

3.3 Specyfikacja komponentów

3.3.1 Jednostki sterujące

- Raspberry Pi 4 - główny komputer zarządzający robotem:
 - System operacyjny Ubuntu 22.04
 - ROS 2 Humble
 - Komunikacja przez SSH z jednostką sterującą zachowaniem robota



Rysunek 3.2: Raspberry Pi 4 model B

- Arduino Nano - sterownik silników:
 - Obsługa enkoderów
 - Komunikacja szeregową z Raspberry Pi



Rysunek 3.3: Arduino Nano

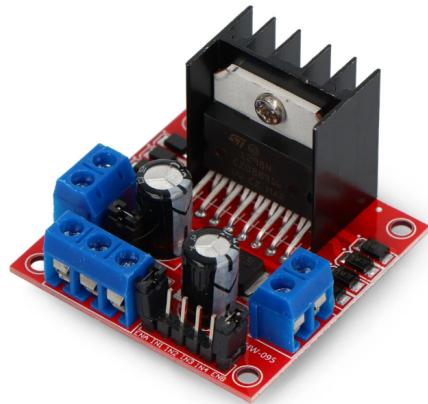
3.3.2 Napęd

- 2x silnik DC 12V 240RPM z metalową przekładnią
- Wbudowane enkodery magnetyczne Halla



Rysunek 3.4: Silnik DC 12V 240RPM typu L z przekładnią metalową - magnetyczny enkoder Halla - Waveshare 22346

- Sterownik L298N - dwukanałowy mostek H



Rysunek 3.5: L298N - dwukanałowy sterownik silników - moduł 12V/2A

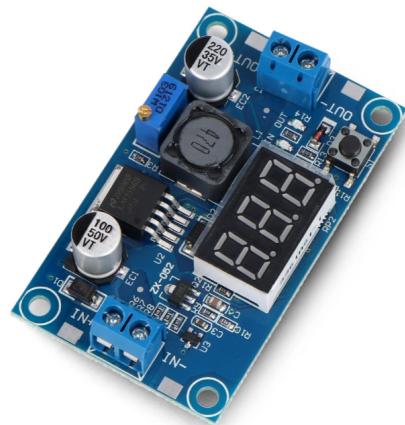
3.3.3 Zasilanie

- 6x akumulatory Li-ion 18650:
 - 4 ogniska (2S2P) dla silników
 - 2 ogniska (2S) dla elektroniki



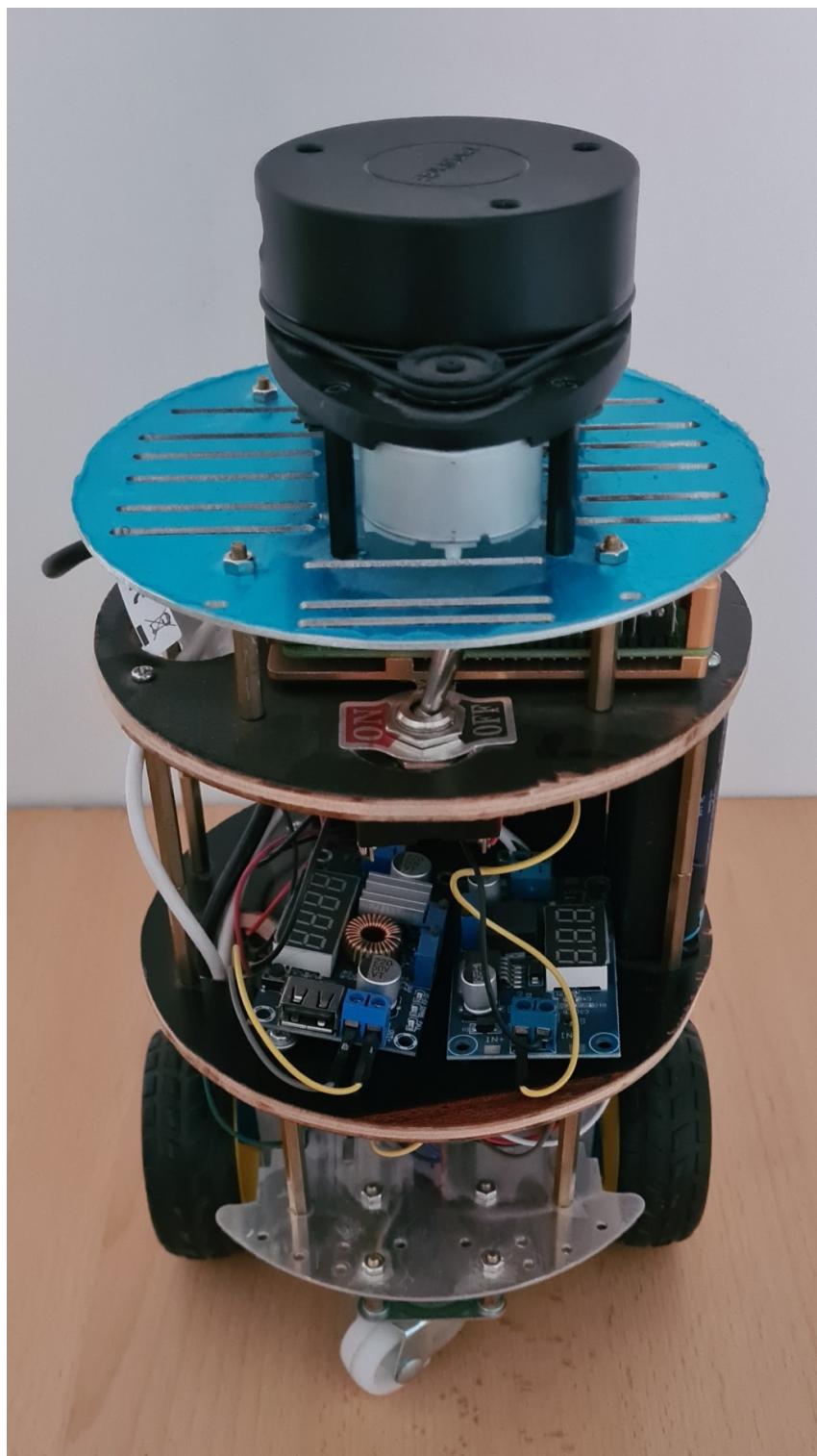
Rysunek 3.6: Ogniska 18650 Li-Ion XTAR - 2600mAh

- 2x przetwornica step-down:
 - 12V dla silników
 - 5V dla Raspberry Pi

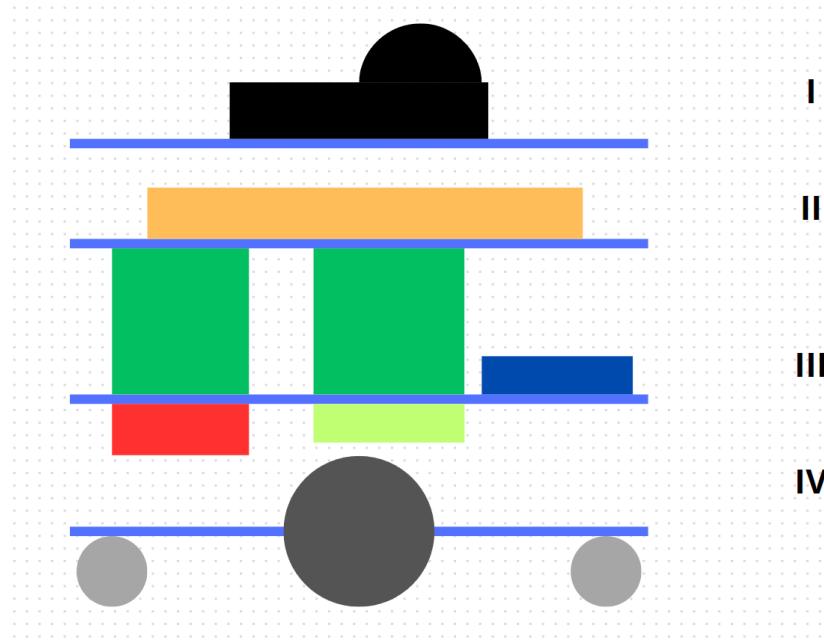


Rysunek 3.7: Przetwornica step-down LM2596 3,2V-35V 3A z wyświetlaczem

3.4 Budowa robota i sposób połączenia silników z kontrolerem



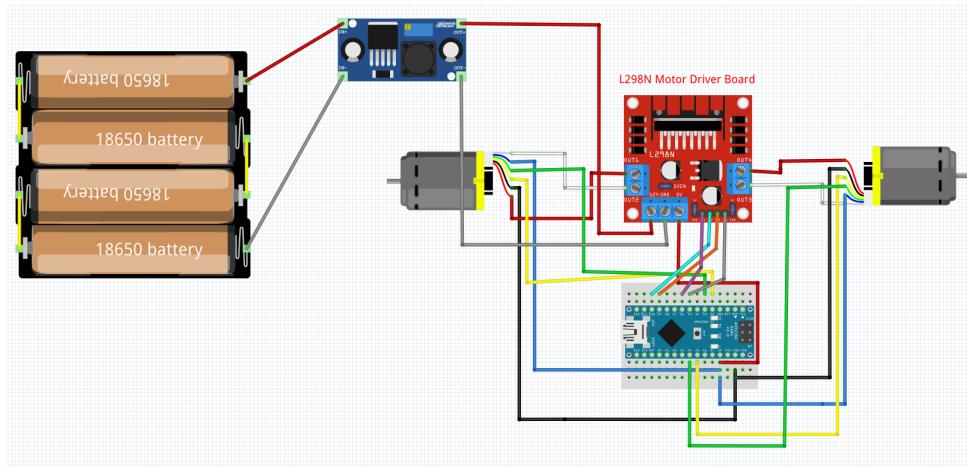
Rysunek 3.8: Zdjęcie przedstawiające zbudowanego robota



Rysunek 3.9: Uproszczony schemat przedstawiający budowę robota

Na schemacie z rysunku 3.9 przedstawiono konkretnie poziomy robota, takie jak:

- Poziom I - LiDAR RPLidar A1 (Czarny element)
- Poziom II - Raspberry Pi 4 (Pomarańczowy element)
- Poziom III - Akumulatory (Zielone elementy) i przetwornice (Niebieski element)
- Poziom IV - Arduino Nano (Żółty element) i sterownik L298N (Czerwony element) z silnikami (Szare elementy)



Rysunek 3.10: Schemat połączenia silników z Arduino i L298N

Na zaprezentowanym schemacie (rys. 3.10) przedstawiono sposób połączenia silników z Arduino i sterownikiem L298N. Silniki DC z enkoderami są zasilane z akumulatorów Li-ion, a sterowane przez Arduino Nano za pomocą sterownika L298N. Enkodery są podłączone do Arduino, które odczytuje impulsy i oblicza prędkość i położenie robota. Komunikacja między Arduino a Raspberry Pi odbywa się przez port szeregowy, co umożliwia przesyłanie danych o prędkości i położeniu robota.

3.5 Metodyka i etapy realizacji

3.5.1 Etap 1: Przygotowanie platformy sprzętowej

- Instalacja systemu Ubuntu 22.04 na Raspberry Pi
- Konfiguracja połączenia SSH
- Instalacja ROS 2 Humble

3.5.2 Etap 2: Implementacja sterowania napędem

- Podłączenie silników do sterownika L298N
- Programowanie Arduino - obsługa silników i enkoderów
- Implementacja komunikacji szeregowej z Raspberry Pi

3.5.3 Etap 3: Integracja sensorów

- Montaż i konfiguracja LiDAR-a
- Kalibracja czujników
- Opracowanie układu mechanicznego i obudowy

3.5.4 Etap 4: Implementacja oprogramowania

- Konfiguracja pakietów ROS 2:
 - SLAM Toolbox do mapowania
 - Nav2 do nawigacji z AMCL do lokalizacji
 - ROS2 Control do sterowania napędem
- Integracja i testy systemu

Rozdział 4

Specyfikacja użytkowa

W tym rozdziale przedstawiono wymagania użytkownika oraz specyfikację funkcjonalną systemu. Opisano kategorie użytkowników, sposób obsługi, administrację systemem, kwestie bezpieczeństwa oraz przykłady działania systemu.

4.0.1 Wymagania sprzętowe i programowe

Projekt ten stworzony był z myślą o następujących wymaganiach dla robota:

Sprzętowe:

- Poruszać się w przestrzeni za pomocą silników, czyli np. przemieszczanie się do przodu, do tyłu, skręcanie w lewo i w prawo po korytarzach, czy w pomieszczeniach w równym podłożem.
- Skanować pomieszczenia za pomocą LiDAR-a, czyli zbieranie danych o otoczeniu wokół robota.

Programowe:

- Tworzyć mapę otoczenia, czyli zapisywanie danych z LiDAR-a w formie mapy 2D w czasie rzeczywistym i wizualizację tych danych w programie Rviz.
- Lokalizować się na mapie, czyli określanie pozycji robota na zapisanej mapie.
- Nawigować do wyznaczonych punktów, czyli planowanie trasy do punktów na mapie i omijanie przeszkód.

4.0.2 Metoda instalacji oprogramowania i konfiguracji sprzętu

- Instalacja systemu Ubuntu 22.04 na Raspberry Pi
- Konfiguracja połączenia SSH
- Instalacja ROS 2 Humble na Raspberry Pi i komputerze sterującym
- Podłączenie silników do sterownika L298N
- Programowanie Arduino - obsługa silników i enkoderów
- Implementacja komunikacji szeregowej z Raspberry Pi
- Montaż i konfiguracja LiDAR-a
- Kalibracja czujników
- Opracowanie układu mechanicznego i obudowy
- Konfiguracja pakietów ROS 2:
 - SLAM Toolbox do mapowania
 - Nav2 do nawigacji z AMCL do lokalizacji
 - ROS2 Control do sterowania napędem

4.0.3 Sposób aktywacji i korzystania z robota z przykadem działania

W tej sekcji wyjaśnione zostaną poszczególne etapy jakie należy podjąć do poprawnego uruchomienia i korzystania z robota.

Na samym początku należy uruchomić Raspberry Pi, przez włączenie zasilania, oraz załączenie silników.

Następnie należy przygotować terminale na jednostce sterującej (dwa terminale mają być połączone przez protokół ssh z Raspberry Pi, a 5 terminali ma być przygotowanych na samej jednostce sterującej do uruchomienia późniejszych skryptów ROS). Przygotowanie tych terminali polega na odpowiednim:

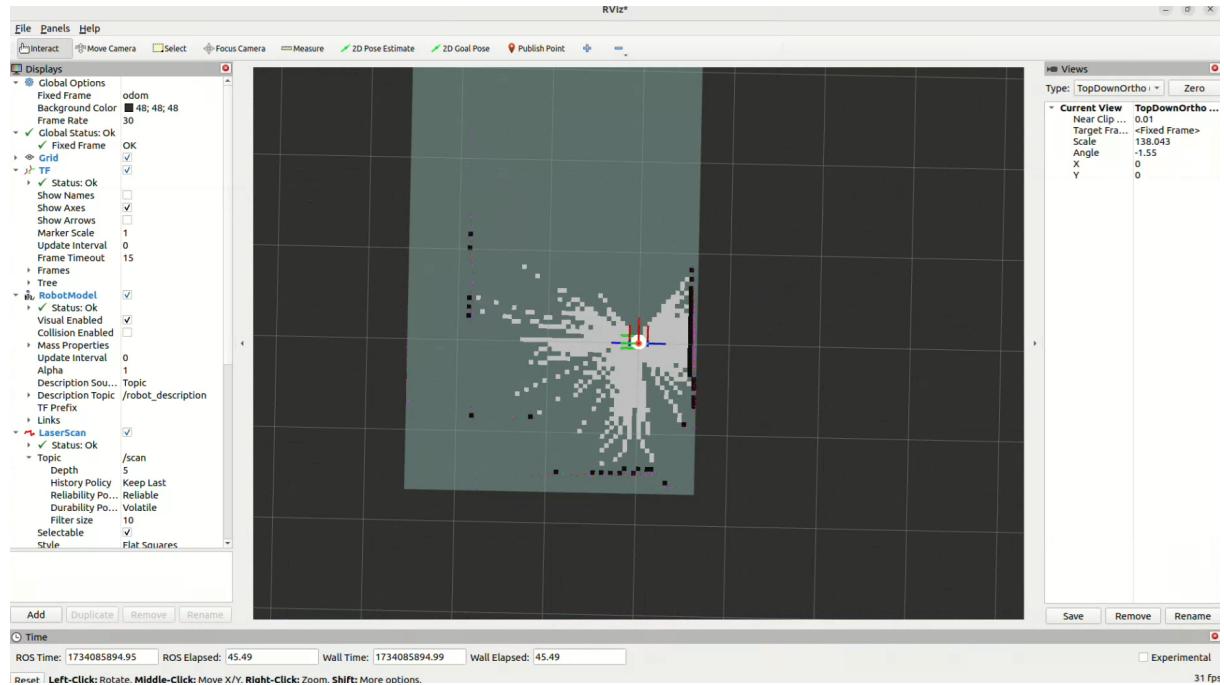
Dla jednostki sterującej: wejściu do katalogu roboczego, uruchomienie komend source /opt/ros/humble/setup.bash, oraz source /install/setup.bash.

Dla Raspberry Pi: wejściu do katalogu roboczego, uruchomienie komend source /opt/ros/humble/setup.bash, oraz source /install/setup.bash.

Następnie, należy uruchomić 2 skrypty przez ssh na Raspberry Pi, które odpowiadają za uruchomienie odpowiednich węzłów ROS. Pierwszy skrypt odpowiada za uruchomienie węzła odpowiedzialnego za sterowanie silnikami, a drugi za uruchomienie węzła odpowiedzialnego za odczyt danych z LiDAR-a. Odpowiednie komendy to: ros2 launch robot_slam model_controll.launch.py oraz ros2 launch robot_slam rplidar.launch.py.

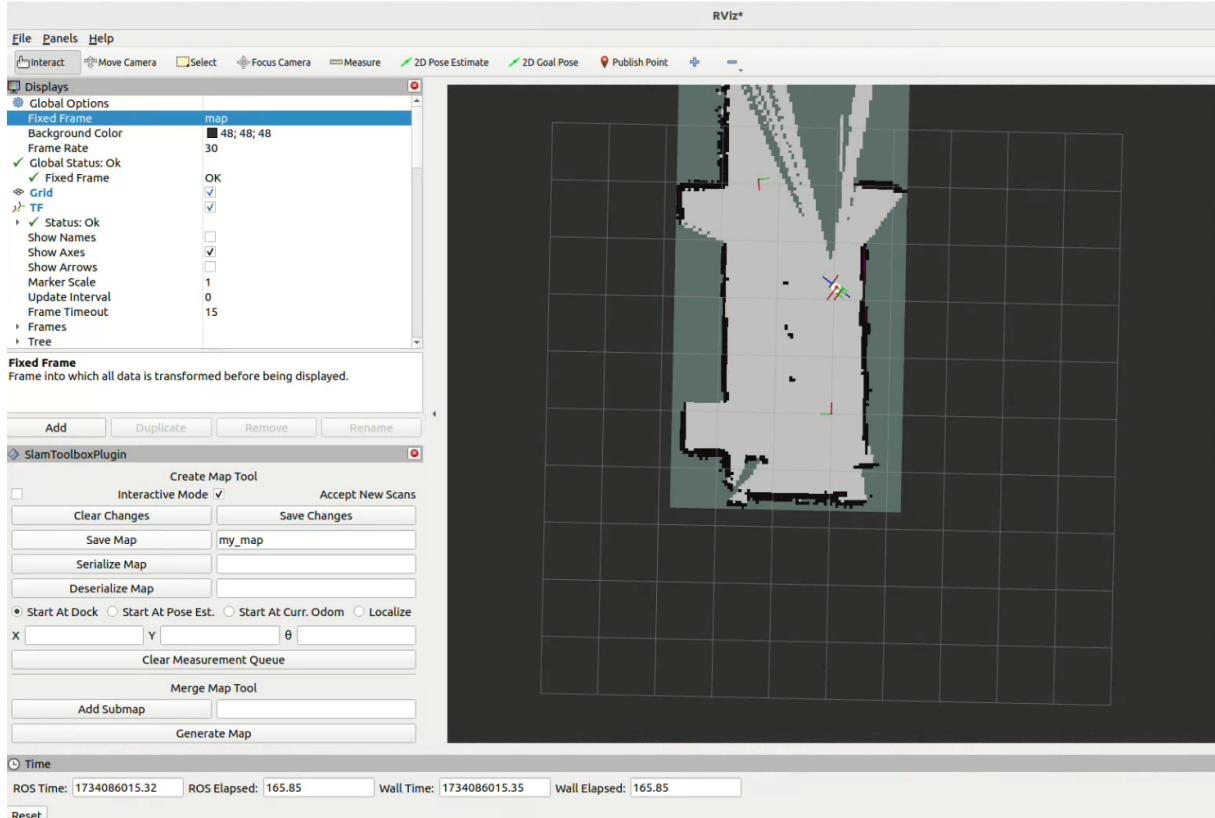
Po uruchomieniu tych skryptów, należy uruchomić skrypt odpowiedzialny za sterowanie robotem za pomocą klawiatury. Komenda to: ros2 launch robot_teleop.launch.py. W tym momencie należy przejść do terminala na jednostce sterującej, i za pomocą klawiatury sterować robotem. Należy również uruchomić program Rviz, który pozwala na wizualizację mapy i położenia robota. Komenda to: rviz2.

Kolejnym krokiem jest uruchomienie skryptu odpowiedzialnego za mapowanie otoczenia. Komenda to: ros2 launch robot_slam slam.launch.py. W tym momencie gdy robot przemieszcza się przez komendy z klawiatury, pomieszczenie zostaje zmapowane.



Rysunek 4.1: Wizualizacja w Rviz po uruchomieniu skryptów.

Gdy wystarczająco dużo pomieszczenia zostanie zmapowane, należy zapisać mapę. Można to wykonać przez komendę ros2 run nav2_map_server map_saver -f map, lub przez dodanie panelu do Rviz z zestawu narzędzi SLAM Toolbox i zapisanie mapy z poziomu tego panelu.



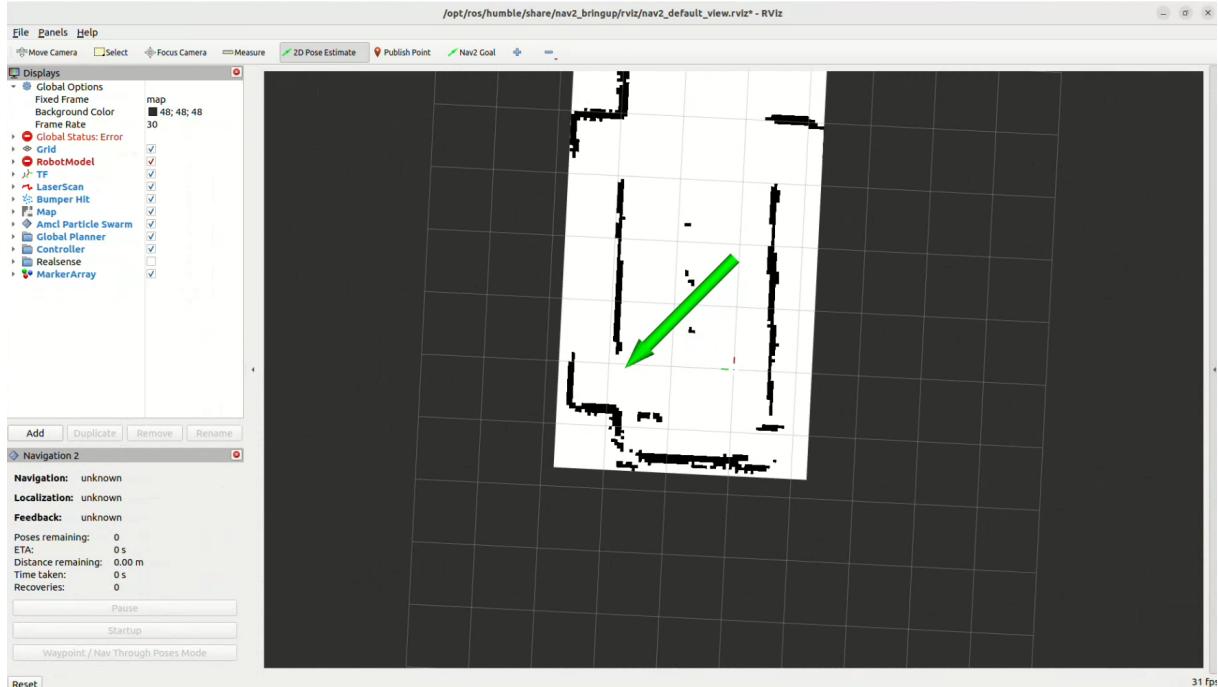
Rysunek 4.2: Wizualizacja zmapowanego pomieszczenia i zapisu mapy.

Z taką zapisaną mapą można zamknąć skrypty odpowiedzialne za sterowanie i mapowanie, jak i Rviz.

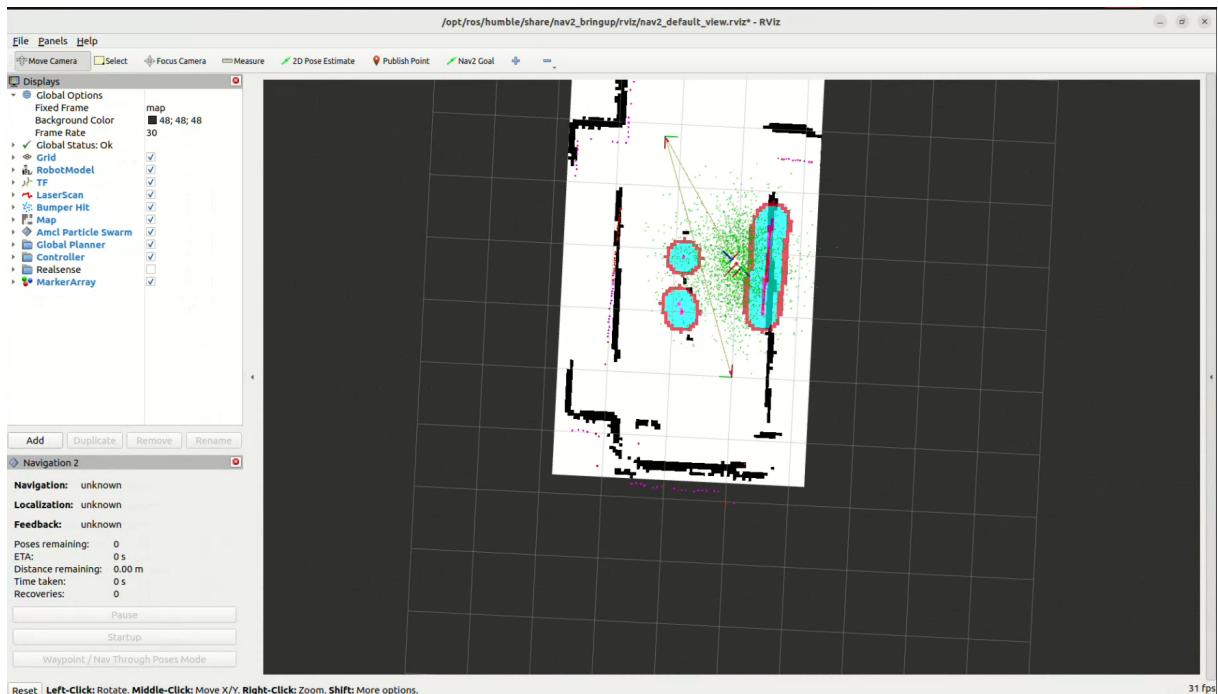
Następnie, należy uruchomić skrypt odpowiedzialny za lokalizację robota na mapie, oraz skrypt odpowiedzialny za nawigację robota.

Komendy to: `ros2 launch robot_slam localization_launch.py`,
`ros2 launch robot_slam navigation_launch.py`. W wyniku uruchomienia tych skryptów powinien pojawić się nowy ekran Rviz, na którym będzie widoczna wcześniej zapisana mapa.

Należy na niej umieścić robota przez wybranie z górnego zestawu narzędzi 2D Pose Estimate, a następnie kliknięcie na mapie w miejscu gdzie znajduje się robot. W tym momencie algorytm AMCL tworzy chmurę przewidywanych punktów w których może znajdować się robot.

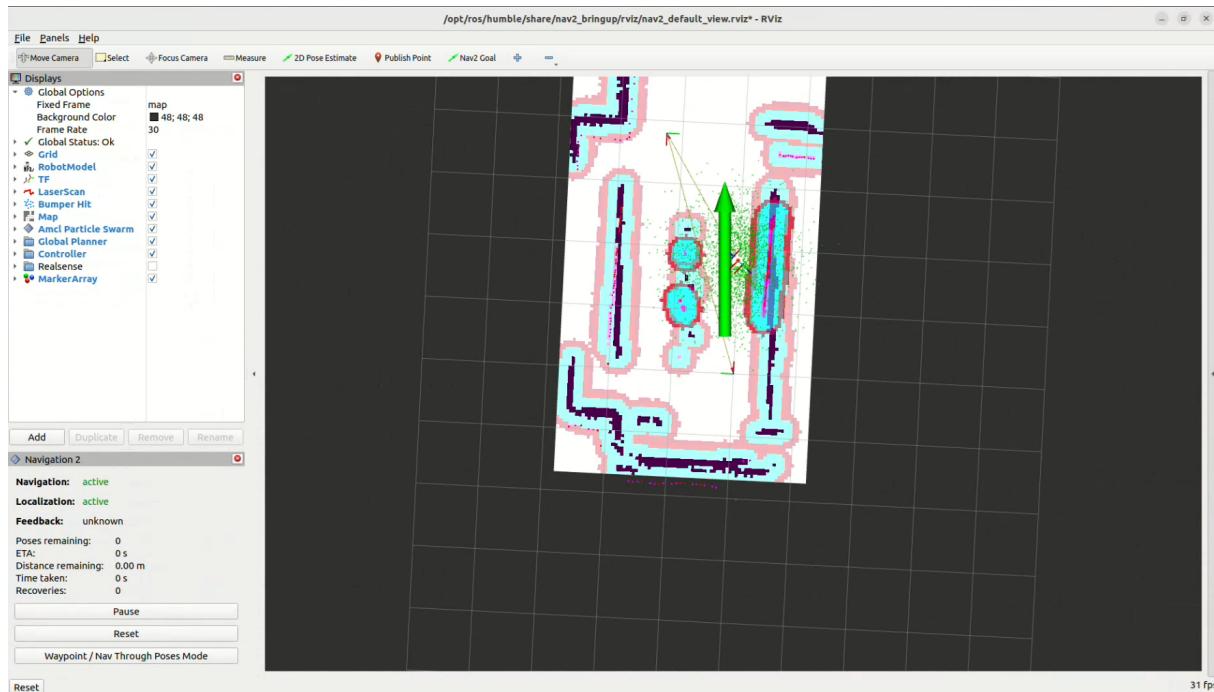


Rysunek 4.3: Wizualizacja po uruchomieniu skryptów do lokalizacji i nawigacji.



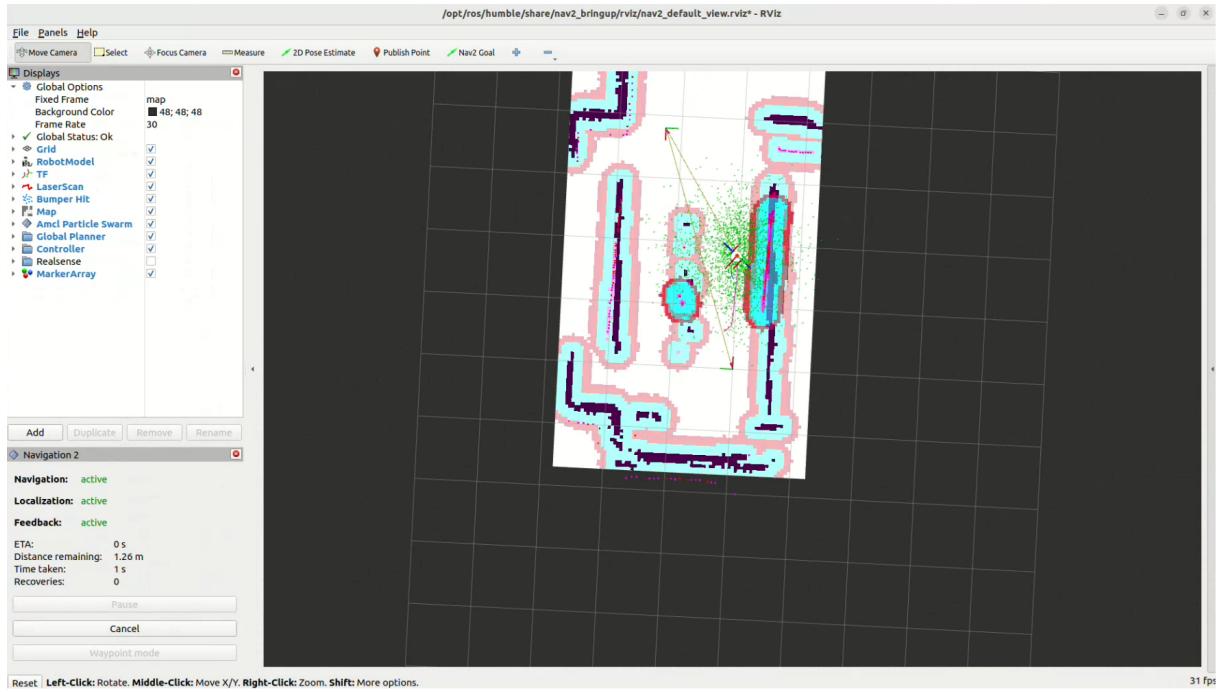
Rysunek 4.4: Wizualizacja rosproszonych punktów lokalizacji robota.

Następnie należy wybrać cel do którego robot ma się przemieścić, przez wybranie z górnego zestawu narzędzi 2D Goal Pose, a następnie kliknięcie na mapie w miejscu gdzie ma się znaleźć cel.

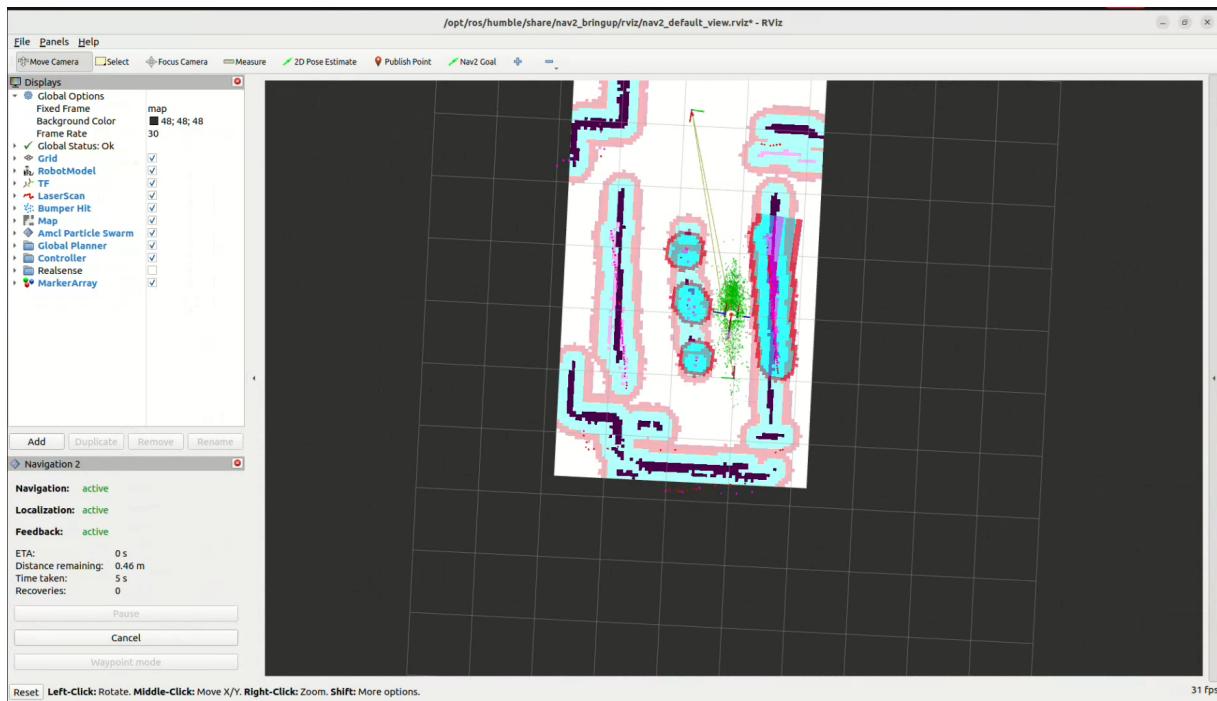


Rysunek 4.5: Wizualizacja po wybraniu celu do którego robot ma się przemieścić.

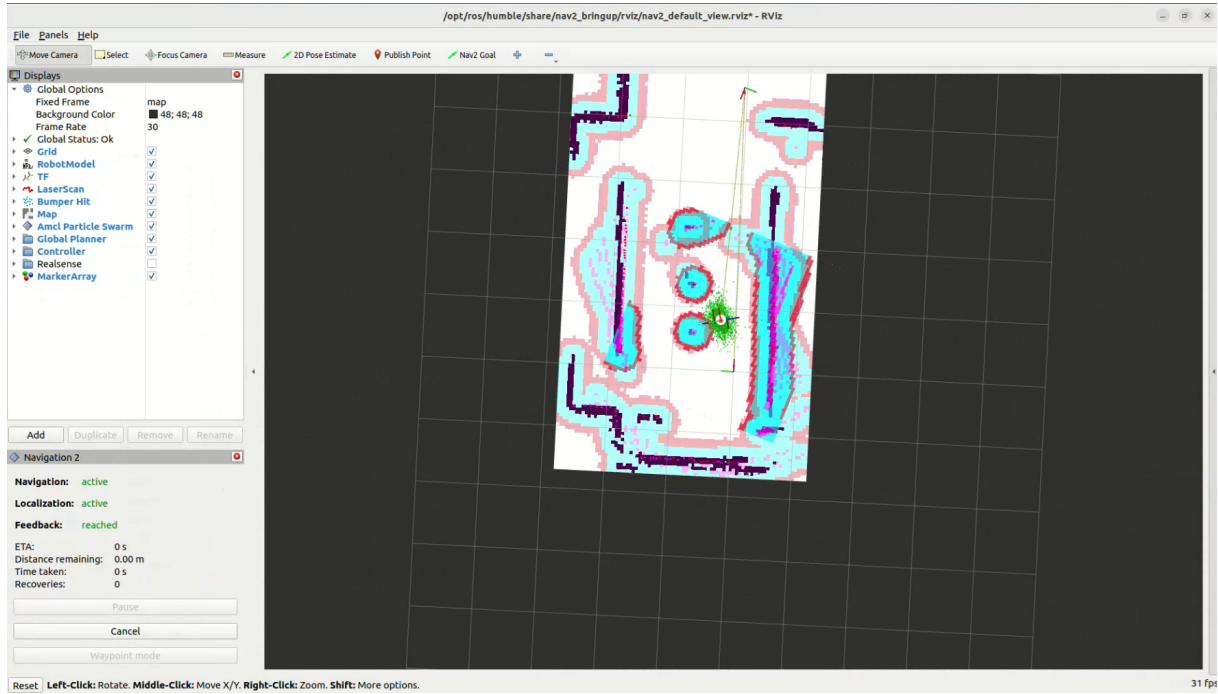
Robot powinien samodzielnie przemieścić się do tego celu, omijając przeszkody.



Rysunek 4.6: Wizualizacja wyznaczenia trasy.



Rysunek 4.7: Wizualizacja nawigacji robota i zmniejszania się chmury przewidywanej lokalizacji robota.



Rysunek 4.8: Wizualizacja po dotarciu robota do celu.

4.1 Administracja systemem

System nie wymaga specjalnej administracji, jednakże w przypadku problemów z działaniem, można skorzystać z narzędzi diagnostycznych dostępnych w ROS 2, jak i z dokumentacji dostępnej na stronie internetowej ROS 2. W celu modyfikacji np. prędkością poruszania robota, przy korzystaniu z klawiatury, można zmienić prędkość przez naciśnięcie q i w. Do modyfikacji prędkości podczas nawigacji, można zmienić prędkość w pliku konfiguracyjnym Nav2.

4.2 Kwestie bezpieczeństwa

Robot opracowany został z myślą o omijaniu przeszkód, nawet tych których nie było na wcześniej stworzonej mapie pozwalając na bezpieczne poruszanie się w przestrzeni. Należy jednak pamiętać że robot ten nie został przygotowany do pracy w środowiskach bez równego podłoża, dlatego należy unikać przemieszczania się po nierównym terenie co może prowadzić do przewrócenia się robota.

4.3 Scenariusze korzystania z systemu

Robot ten pozwala na mapowanie różnego rodzaju pomieszczeń, oraz na nawigację do wyznaczonych punktów, co pozwala na zastosowanie go w różnego rodzaju zastosowaniach, jak np. inspekcja pomieszczeń. Testy przeprowadzane były w małych pomieszczeniach nie przekraczających 40 m^2 , jednakże robot ten jest w stanie zmapować pomieszczenia nawet do $24\,000 \text{ m}^2$ co wynika z dokumentacji Nav2 [8].

Rozdział 5

Specyfikacja techniczna

Rozdział ten skupia się na aspektach technicznych projektu. Opisano w nim architekturę systemu, struktury danych, komponenty, moduły, biblioteki, algorytmy, wzorce projektowe, diagramy UML oraz szczegóły implementacji wybranych fragmentów.

5.1 Idea robota autonomicznego mapującego w czasie rzeczywistym

Robot autonomiczny mapujący w czasie rzeczywistym to robot mobilny, który samodzielnie przemieszcza się w przestrzeni, zbierając dane o otoczeniu i tworząc mapę otoczenia w czasie rzeczywistym. Robot ten wykorzystuje różnego rodzaju sensory, jak np. LiDAR, kamery, czy enkodery, do zbierania danych o otoczeniu. Dane te są przetwarzane przez algorytmy SLAM (Simultaneous Localization and Mapping), które pozwalają na określenie pozycji robota na mapie oraz na tworzenie mapy otoczenia. Robot ten może być wykorzystywany w różnego rodzaju zastosowaniach, jak np. inspekcja pomieszczeń, czy przemieszczanie się w trudno dostępnych miejscach.

5.2 Architektura systemu

Sposób działania systemu oparty jest na współdziałających ze sobą skryptach, których sposób połączenia zaprezentowano na rysunku 5.1.



Rysunek 5.1: Wizualizacja połączenia skryptów w systemie

5.3 Struktura systemu i objaśnienie działania algorytmów

W tej sekcji opisano poszczególne komponenty robota z rysunku 5.1

5.3.1 Skrypt Arduino

Skrypt Arduino jest odpowiedzialny za sterowanie silnikami i enkoderami. Skrypt ten odczytuje dane z enkoderów, oblicza prędkość i położenie robota, oraz steruje silnikami na podstawie danych o prędkościach otrzymanych od Raspberry Pi. Skrypt ten komunikuje się z Raspberry Pi przez port szeregowy, co umożliwia przesyłanie danych o prędkości i położeniu robota. Skrypt ten jest zmodyfikowaną wersją skryptu ros_arduino_bridge[4], który został dostosowany do potrzeb tego projektu.

5.3.2 Skrypt do obsługi LiDAR-a

Skrypt do obsługi LiDAR-a jest odpowiedzialny za odczyt danych z LiDAR-a. Skrypt ten odczytuje dane z LiDAR-a, przetwarza je i przesyła do Raspberry Pi. Skrypt ten jest zmodyfikowaną wersją skryptu rplidar_ros[bib:rplidarros], który został dostosowany do potrzeb tego projektu.

5.3.3 Skrypt do obsługi silników i tworzenia modelu robota

Skrypt do obsługi silników i tworzenia modelu robota jest odpowiedzialny za sterowanie silnikami i tworzenie modelu robota. Skrypt ten odczytuje dane o prędkościach robota z klawiatury, przetwarza je i przesyła do Arduino. Skrypt ten jest odpowiedzialny również za udostępnienie modelu robota z pliku rsp.launch.py, plik ten tworzy węzeł z pliku konfiguracyjnego model_main.xacro, gdzie znajduje się model robota w formacie URDF.

5.3.4 Skrypt do tworzenia mapy

Skrypt do tworzenia mapy jest odpowiedzialny za tworzenie mapy otoczenia. Skrypt ten odczytuje dane z LiDAR-a, przetwarza je i tworzy mapę otoczenia. Skrypt ten jest odpowiedzialny również za udostępnienie mapy z pliku map_server.launch.py, plik ten tworzy węzeł z pliku konfiguracyjnego map.yaml, gdzie znajduje się zapisana mapa otoczenia.

5.3.5 Skrypty do nawigacji i lokalizacji

Skrypty do nawigacji i lokalizacji są odpowiedzialne za lokalizację robota na mapie oraz za nawigację robota. Skrypty te odczytują dane z mapy i z LiDAR-a, przetwarzają je

i określają pozycję robota na mapie oraz planują trasę do wyznaczonego punktu. Skrypty te są odpowiedzialne również za sterowanie robotem, tak aby omijał przeszkody na swojej drodze.

5.3.6 Diagramy UML prezentujące działanie konkretnych programów

Krótką wstawka kodu w linii tekstu jest możliwa, np. `int a;` (biblioteka `listings`). Dłuższe fragmenty lepiej jest umieszczać jako rysunek, np. kod na rys. ??, a naprawdę długie fragmenty – w załączniku.

Rozdział 6

Weryfikacja i walidacja

W ramach pracy zastosowano model V do testowania systemu. Model V jest popularnym modelem w inżynierii oprogramowania, który przedstawia proces tworzenia oprogramowania w formie litery V. Lewa strona litery V reprezentuje fazy definiowania wymagań i projektowania systemu, natomiast prawa strona litery V reprezentuje fazy testowania i walidacji systemu.

6.1 Model V

Model V składa się z następujących etapów:

- **Definiowanie wymagań:** Określenie wymagań funkcjonalnych i niefunkcjonalnych systemu.
- **Projektowanie systemu:** Opracowanie architektury systemu oraz szczegółowego projektu poszczególnych komponentów.
- **Implementacja:** Kodowanie poszczególnych modułów systemu.
- **Testowanie jednostkowe:** Testowanie poszczególnych modułów w celu weryfikacji ich poprawności.
- **Integracja i testowanie integracyjne:** Łączenie modułów i testowanie ich współprądzania.
- **Testowanie systemowe:** Testowanie całego systemu w celu weryfikacji spełnienia wymagań.
- **Walidacja:** Sprawdzenie, czy system spełnia oczekiwania użytkownika i działa zgodnie z założeniami.

6.2 Organizacja eksperymentów

Eksperymenty przeprowadzono zgodnie z model V, realizując następujące kroki:

- **Testowanie jednostkowe:** Każdy moduł systemu, taki jak sterowanie silnikami, odczyt danych z LiDAR-a, tworzenie mapy, lokalizacja i nawigacja, został przetestowany indywidualnie. Testy jednostkowe obejmowały sprawdzenie poprawności działania poszczególnych funkcji i metod.
- **Testowanie integracyjne:** Po zakończeniu testów jednostkowych, moduły zostały zintegrowane i przetestowane pod kątem poprawności współdziałania. Testy integracyjne obejmowały sprawdzenie komunikacji między modułami oraz poprawności przesyłania danych.
- **Testowanie systemowe:** Cały system został przetestowany w warunkach rzeczywistych. Testy systemowe obejmowały sprawdzenie poprawności działania systemu w różnych scenariuszach, takich jak zdalne sterowanie robotem, tworzenie mapy otoczenia, lokalizacja robota na mapie oraz autonomiczna nawigacja do wyznaczonych punktów.
- **Walidacja:** System został zweryfikowany pod kątem spełnienia wymagań użytkownika. Walidacja obejmowała sprawdzenie, czy system działa zgodnie z założeniami i spełnia oczekiwania użytkownika.

6.3 Przypadki testowe

Przypadki testowe obejmowały następujące scenariusze:

- **Testowanie zdalnego sterowania:** Sprawdzenie, czy robot reaguje poprawnie na polecenia z klawiatury.
- **Testowanie tworzenia mapy:** Sprawdzenie, czy system poprawnie tworzy mapę otoczenia na podstawie danych z LiDAR-a.
- **Testowanie lokalizacji:** Sprawdzenie, czy system poprawnie lokalizuje robota na zapisanej mapie.
- **Testowanie nawigacji:** Sprawdzenie, czy system poprawnie planuje trasę i nawigację robota do wyznaczonych punktów, omijając przeszkody.

6.4 Wykryte i usunięte błędy

Podczas testowania systemu wykryto i usunięto następujące błędy:

- **Problem z zasilaniem:** Występowały problemy z zasilaniem silników, które zostały rozwiązane przez zastosowanie odpowiednich przetwornic step-down.
- **Problem z kółkami:** Kółka robota miały tendencję do ślizgania się, co zostało rozwiązane przez zastosowanie gumowych opon.
- **Problem z błędna nawigacją:** Robot czasami poruszał się w nieoczekiwany sposób, co zostało rozwiązane przez poprawne podłączenie silników i kalibrację enkoderów.

6.5 Wyniki badań eksperymentalnych

Wyniki testów potwierdziły poprawność działania systemu. Robot poprawnie reagował na polecenia z klawiatury, tworzył mapę otoczenia, lokalizował się na zapisanej mapie oraz nawigował do wyznaczonych punktów, omijając przeszkody. System spełniał wszystkie wymagania funkcjonalne i niefunkcjonalne określone na początku projektu.

Rozdział 7

Podsumowanie i wnioski

7.1 Uzyskane wyniki

W wyniku przeprowadzonych prac udało się zrealizować wszystkie założone cele projektu. System autonomicznej nawigacji robota mobilnego został zaprojektowany, zaimplementowany i przetestowany. Robot poprawnie tworzy mapę otoczenia, lokalizuje się na zapisanej mapie oraz nawigował do wyznaczonych punktów, omijając przeszkody. System spełnia wszystkie wymagania funkcjonalne i niefunkcjonalne określone na początku projektu.

7.2 Kierunki dalszych prac

W przyszłości możliwe jest rozszerzenie funkcjonalności systemu o dodatkowe elementy, takie jak:

- Integracja z dodatkowymi sensorami, takimi jak kamery RGB-D, aby poprawić dokładność mapowania i lokalizacji.
- Implementacja zaawansowanych algorytmów planowania ścieżki, które uwzględniają dynamiczne przeszkody.
- Rozbudowa systemu o funkcje autonomicznego ładowania baterii.
- Zastosowanie algorytmów uczenia maszynowego do optymalizacji nawigacji i mapowania.
- Integracja z systemami IoT (Internet of Things) w celu zdalnego monitorowania i sterowania robotem.

Bibliografia

- [1] Luis Bermudez. *Medium - Overview of SLAM*. 2024. URL: <https://medium.com/machinevision/overview-of-slam-50b7f49903b7> (term. wiz. 17.04.2024).
- [2] Bence Magyar Christoph Fröhlich Denis Stogl i Sai Kishor Kothakota. *ros2 control*. 2024. URL: <https://control.ros.org/humble/index.html> (term. wiz. 01.12.2024).
- [3] Dieter Fox. „KLD-Sampling: Adaptive Particle Filters.” W: sty. 2001, s. 713–720.
- [4] Patrick Goebel. *ROS.org - ros arduino bridge*. 2012. URL: https://wiki.ros.org/ros_arduino_bridge (term. wiz. 25.12.2012).
- [5] Graylin Trevor Jay. *Teleop Twist Keyboard*. 2015. URL: https://wiki.ros.org/teleop_twist_keyboard (term. wiz. 22.01.2015).
- [6] Matti Kortelainen. „A short guide to ROS 2 Humble Hawksbill”. W: *School of Computing, University of Eastern Finland, Kuopio, Finland* (2023), s. 5.
- [7] Steve Macenski i Ivona Jambrecic. „SLAM Toolbox: SLAM for the dynamic world”. W: *Journal of Open Source Software* 6.61 (2021), s. 2783. DOI: [10.21105/joss.02783](https://doi.org/10.21105/joss.02783). URL: <https://doi.org/10.21105/joss.02783>.
- [8] Steve Macenski, Francisco Martín, Ruffin White i Jonatan Ginés Clavero. „The Marathon 2: A Navigation System”. W: *CoRR* abs/2003.00368 (2020). arXiv: 2003.00368. URL: <https://arxiv.org/abs/2003.00368>.
- [9] Josh Newans. *ROS.org - diffdrive arduino*. 2020. URL: https://wiki.ros.org/diffdrive_arduino (term. wiz. 08.12.2020).
- [10] Francisco Martin Rico. *A Concise Introduction to Robot Programming with ROS2*. Broken Sound Parkway NW: Taylor i Francis, 2022. ISBN: 1032264659.

Dodatki

Spis skrótów i symboli

SLAM jednoczesna lokalizacja i mapowanie (ang. *Simultaneous Localization and Mapping*)

LiDAR urządzenie wykonujące wykrywanie światła i określanie odległości (ang. *Light Detection and Ranging*)

IMU inercyjna jednostka pomiarowa (ang. *Inertial Measurement Unit*)

RGB-D kamera rejestrująca obraz RGB oraz informację o głębi (ang. *RGB-Depth*)

Nav2 system nawigacji dla ROS 2 (ang. *Navigation 2*)

ROS system operacyjny dla robotów (ang. *Robot Operating System*)

ROS2 Control system kontroli robotów dla ROS 2 (ang. *Robot Operating System 2 Control*)

SLAM Toolbox zestaw narzędzi do jednoczesnej lokalizacji i mapowania (ang. *Simultaneous Localization and Mapping Toolbox*)

Źródła

Jeżeli w pracy konieczne jest umieszczenie długich fragmentów kodu źródłowego, należy je przenieść w to miejsce.

```
1 if (_nClusters < 1)
2     throw std::string ("unknown number of clusters");
3 if (_nIterations < 1 and _epsilon < 0)
4     throw std::string ("You should set a maximal number of
5         iteration or minimal difference — epsilon .");
6 if (_nIterations > 0 and _epsilon > 0)
7     throw std::string ("Both number of iterations and minimal
8         epsilon set — you should set either number of iterations
9         or minimal epsilon .");
```

Lista dodatkowych plików, uzupełniających tekst pracy

W systemie do pracy dołączono dodatkowe pliki zawierające:

- źródła programu,
- dane testowe,
- film pokazujący działanie opracowanego oprogramowania lub zaprojektowanego i wykonanego urządzenia,
- itp.

Spis rysunków

2.1	Reprezentacja pośrednika w systemie robota [10]	6
2.2	Mapa sklepu utworzona za pomocą SLAM Toolbox [7]	7
3.1	Diagram przypadków użycia systemu	10
3.2	Raspberry Pi 4 model B	11
3.3	Arduino Nano	12
3.4	Silnik DC 12V 240RPM typu L z przekładnią metalową - magnetyczny enkoder Halla - Waveshare 22346	12
3.5	L298N - dwukanałowy sterownik silników - moduł 12V/2A	13
3.6	Ogniwa 18650 Li-Ion XTAR - 2600mAh	13
3.7	Przetwornica step-down LM2596 3,2V-35V 3A z wyświetlaczem	14
3.8	Zdjęcie przedstawiające zbudowanego robota	15
3.9	Uproszczony schemat przedstawiający budowę robota	16
3.10	Schemat połączenia silników z Arduino i L298N	17
4.1	Wizualizacja w Rviz po uruchomieniu skryptów.	22
4.2	Wizualizacja zmapowanego pomieszczenia i zapisu mapy.	23
4.3	Wizualizacja po uruchomieniu skryptów do lokalizacji i nawigacji.	25
4.4	Wizualizacja rosproszonych punktów lokalizacji robota.	25
4.5	Wizualizacja po wybraniu celu do którego robot ma się przemieścić.	26
4.6	Wizualizacja wyznaczenia trasy.	27
4.7	Wizualizacja nawigacji robota i zmniejszania się chmury przewidywanej lokalizacji robota.	27
4.8	Wizualizacja po dotarciu robota do celu.	28
5.1	Wizualizacja połączenia skryptów w systemie	32

Spis tabel