POLITECHNIKA WROCŁAWSKA

ROZPRAWA DOKTORSKA

Lokalizacja i mapowanie w systemach robotów mobilnych

Michał Drwięga

PROMOTOR:
dr hab. inż. Elżbieta Roszkowska

PROMOTOR POMOCNICZY:
dr inż. Janusz Jakubiak

WROCŁAW 2021

Streszczenie

Kluczową cechą autonomicznych robotów mobilnych jest umiejętność tworzenia map nieznanego środowiska i ustalania swojego położenia na tych mapach. Jednoczesne lokalizowanie robota i mapowanie otoczenia (SLAM) to jeden z fundamentalnych problemów robotyki mobilnej, ponieważ nawet tak podstawowe działania, jak planowanie ruchu, czy śledzenie trajektorii, wymagają aktualnej wiedzy o stanie robota i jego otoczeniu. W przypadku wielu robotów, dane pomiarowe z ich sensorów wykorzystuje się w taki sposób, aby utworzyć spójną, globalną mapę środowiska i zlokalizować na niej poszczególne roboty. Istnieją dwa wiodące podejścia do wielorobotowego mapowania. Pierwsze polega na wymianie danych z czujników w trakcie procesu mapowania, natomiast drugie opiera się na niezależnym mapowaniu otoczenia przez poszczególne roboty i przesyłaniu gotowych map w celu ich połączenia za pomocą dedykowanych metod. Niniejsza praca skupia się na drugim podejściu, czyli na niezależnym mapowaniu środowiska przez poszczególne roboty oraz efektywnych metodach integracji map.

W pracy przedstawiono nową, opracowaną i zaimplementowaną metodę integracji map 3D w postaci drzew ósemkowych (octomap). Metoda ta działa dla przypadku, gdy nie są znane początkowe pozycje robotów i roboty nie spotykają się w trakcie eksploracji. Rozwiązanie opiera się na wykrywaniu i dopasowywaniu do siebie nakładających się części map, co umożliwia określenie transformacji między mapami cząstkowymi. Proces integracji map składa się z kilku etapów. W pierwszym, przeprowadzana jest ekstrakcja modeli, które następnie są dopasowywane do drugiej mapy, w ramach operacji globalnego dopasowania. Operacja ta opiera się na wyszukaniu punktów charakterystycznych, ich deskrypcji i wyszukiwaniu podobieństw w dwóch zbiorach deskryptorów odpowiadających dwóm mapom. Wykorzystano w tym celu metodę probabilistyczną (SAC) oraz metodę grupowania na podstawie zgodności geometrycznej (GCC). Kolejnym etapem jest lokalne dopasowanie opierające się na metodach iteracyjnego najbliższego punktu (ICP, OICP), a także metodzie transformacji rozkładów normalnych (NDT). Ostatnim etapem jest transformacja map do wspólnego układu współrzędnych i integracja danych. Metoda integracji map została zaimplementowana i udostępniona jako otwarte oprogramowanie. W pracy przedstawiono wyniki przeprowadzonych badań obejmujących eksperyment z kołowymi robotami mobilnymi oraz wykorzystanie ogólnodostępnych zbiorów map. Wyniki badań potwierdziły, że wymiana informacji między robotami w systemach wielorobotowych umożliwia zwiększenie efektywności metod SLAM.

W pracy zaprezentowano również wielorobotowy serwer map 3D wraz z wynikami eksperymentów. Przedstawiono metodę kalibracji położenia czujnika głębi, a także system automatycznego doboru parametrów metod SLAM. Ponadto zaprezentowano wyniki przeprowadzonych badań eksperymentalnych dotyczących metod lokalizacji oraz metod SLAM, z wykorzystaniem referencyjnego, optycznego systemu śledzenia ruchu.

Abstract

The key feature of autonomous mobile robots is the ability to create maps of unknown environments and estimate pose on these maps. The simultaneous localization and mapping (SLAM) is a fundamental problem of mobile robotics because robots need to know their state and the state of their surroundings, even for basic operations like motion planning or trajectory following. In the case of multiple mobile robot systems, the data from robots sensors are processed in such a way as to create a consistent, global model of the environment and to localize robots on it. There are two main approaches to multi-robot mapping. The first one is based on the exchange of raw data from sensors between robots. The second approach is based on the independent mapping of the environment by multiple robots. Such maps are exchanged between robots and integrated with a dedicated method into one model. This dissertation focuses on the second approach, what means the independent mapping of the environment by each robot and methods of the maps integration.

This paper introduces a new, developed 3D maps integration method that works with octree based maps (octomaps). The approach solves the case when the initial poses of robots are unknown and meetings between robots do not occur during the exploration process. The method detects and matches overlapping regions on maps, and based on that it estimates the transformation between partial maps. The process of maps integration consists of a few steps. The first one is related to the model extraction from one map that is matched to the second map in the global alignment process. This operation is based on the detection of key points, description of features and finding similarities in two sets of descriptors. For this purpose, there were used two methods: probabilistic Sample Consensus (SAC) method and geometry consistency clustering (GCC) approach. As a next step, the local correction is applied. The local alignment process is based on the iterative closest point (ICP, OICP) algorithms and the normal distributions transform (NDT) based method. Finally, one of the maps is transformed to the coordinate system of the second map and the data from both maps are integrated into a single model. The developed solution has been implemented and released as an open source software. In addition, the paper presents results from multiple experiments with wheeled mobile robots and publicly available datasets with octomaps. The presented results of experiments confirmed that data exchange between robots in a multi-robot system allows increasing the efficiency of the simultaneous localization and mapping (SLAM) process.

Moreover, the dissertation presents a multi-robot 3D maps server with experiments results. It contains the developed position calibration method dedicated to a depth sensor and the system for automatic SLAM methods tuning. This paper also presents results of localization and SLAM methods experiments based on the data from a motion capture system used as a ground truth.

Spis treści

Rozdział 1

Wstęp

1.1 Wprowadzenie

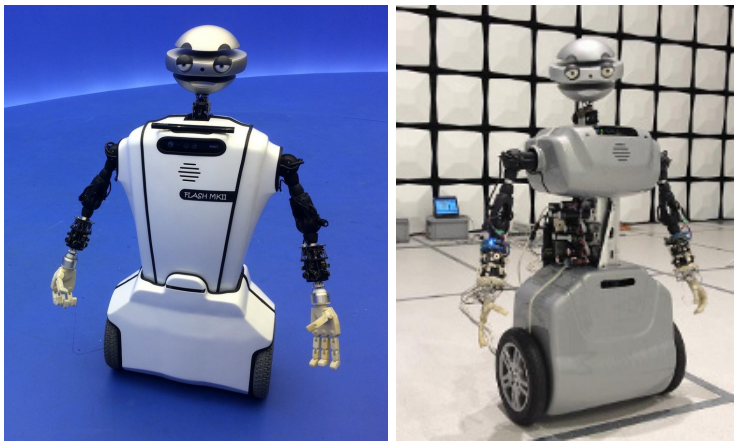
Kluczową cechą autonomicznych robotów mobilnych jest umiejętność tworzenia map nieznanego środowiska i ustalenia swojego położenia na tych mapach. Proces polegający na lokalizowaniu robota w nieznanym otoczeniu, przy równoległym budowaniu mapy, określa się mianem jednoczesnej lokalizacji i mapowania (SLAM, ang. *Simultaneous Localization and Mapping*). Złożoność tego problemu jest związana z wzajemną zależnością zadań realizowanych w tym samym czasie.

Problem lokalizacji i mapowania w przypadku pojedynczego robota jest problemem złożonym, a wykorzystanie grup robotów mimo nowych możliwości, wprowadza również dodatkowy stopień skomplikowania. W przypadku wielu robotów, należy wykorzystać dostępne dane pomiarowe z sensorów robotów, w taki sposób, aby utworzyć spójną, globalną mapę środowiska i zlokalizować na niej poszczególne roboty. Każdy z robotów może być jednak wyposażony w różne sensory, co umożliwia uzyskanie wyższej jakości mapy, ale wymaga dodatkowych metod przetwarzania danych. Pojawiają się również kwestie związane z zapewnieniem odpowiedniej komunikacji między robotami, czyli kanału komunikacyjnego o wystarczającej przepustowości oraz niezawodności, a także kwestie dotyczące koordynacji działań robotów [49, 256].

Wykorzystanie grup robotów mobilnych, zamiast pojedynczych robotów niesie za sobą wiele korzyści [116, 256]. Systemy takie mogą być bardziej efektywne dzięki współbieżnemu wykonywaniu zadań, ale także bardziej odporne na uszkodzenia, ponieważ przy defekcie pojedynczego robota jego funkcje mogą zostać przejęte przez innego robota należącego do grupy. Te czynniki wpływają na coraz większą popularność systemów robotów w znanych aplikacjach, jak i na powstawanie nowych zastosowań. Szybki rozwój tej dziedziny jest stymulowany głównie przez aplikacje przemysłowe, jak transport w halach produkcyjnych, czy magazynowych [7, 43], ale również zadania związane z monitorowaniem i ochroną terenu lub budynków.

W gospodarstwach domowych roboty wykorzystywane są do automatyzacji wielu powtarzalnych zadań, jak sprzątanie, mycie okien lub koszenie trawnika. Stosunkowo szybko rozwijającą się dziedziną są też roboty społeczne (rys. 1.1), których zadaniem jest pomoc w obowiązkach i nawiązywanie interakcji z ludźmi, w szczególności z dziećmi i osobami starszymi. Roboty asystujące człowiekowi pełnią również rolę przewodników w muzeach lub na wystawach, a także różnego rodzaju funkcje edukacyjne.

Coraz częściej powstają też projekty dotyczące kooperacji w kontekście robotów medycznych [217], a konkretniej zdalnej diagnostyki [196] i zdalnej lub nadzorowanej chirurgii. Roboty mobilne mogą być też pomocne w rehabilitacji osób po różnego ro-



Rysunek 1.1 Robot społeczny FLASH opracowany w ramach projektu LIREC [123, 148]

dzaju wypadkach, a także mogą zwiększać autonomię i poprawiać jakość życia osób niepełnosprawnych.

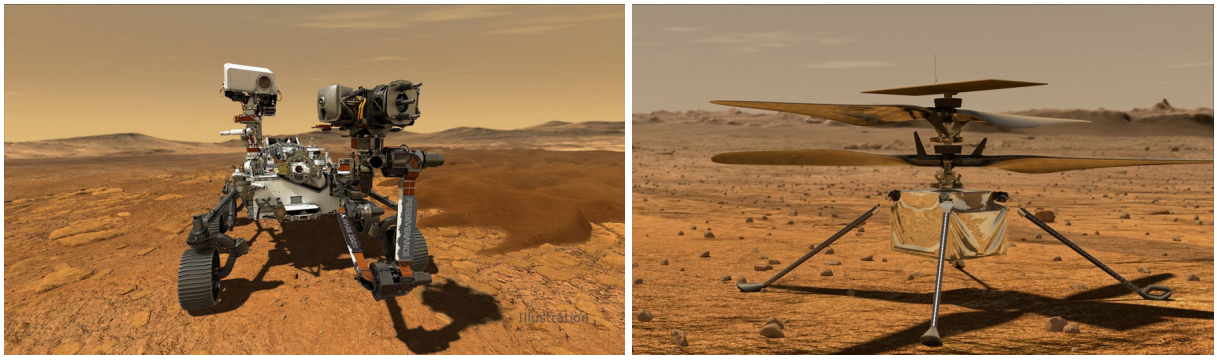
Kolejnym dużym odbiorcą technologii robotycznych związanych z lokalizacją, mapowaniem oraz kooperacją jest przemysł motoryzacyjny, wykorzystujący je na potrzeby pojazdów autonomicznych (rys. 1.2). Działanie takich pojazdów może zostać w znaczny

Rysunek 1.2 Autonomiczny samochód wyposażony w zestaw sensorów oraz system Autoware Auto [71] realizujący zadanie autonomicznej nawigacji. System został zaimplementowany w środowisku ROS 2 [201], który docelowo będzie się komunikował z innymi pojazdami oraz elementami infrastruktury

sposób usprawnione przez wykorzystanie wymiany informacji między poszczególnymi pojazdami, a także między pojazdami i infrastrukturą [185]. Przewiduje się, że wykorzystanie samochodów autonomicznych na szeroką skalę, znacząco wpłynie na bezpieczeństwo transportu, redukując tym samym diametralnie liczbę ofiar wypadków. Obecnie, prawdopodobieństwo poniesienia śmierci w wyniku wypadku drogowego przypadające na jedną godzinę jazdy samochodem prowadzonym przez człowieka szacuje się na 10^{-6}. Celem stawianym przed pojazdami autonomicznymi jest zredukowanie tego prawdopodobieństwa do wartości 10^{-9} na godzinę jazdy w trybie w pełni autonomicznym [223].

Do mniej rozpowszechnionych zastosowań należy bezzałogowe ratownictwo, czy eksploracja kopalni [68]. Grupy robotów mogą być również wykorzystywane do patrolowania granic, lasów i wielu innych obszarów lub obiektów. Innym, dość niszowym zastosowaniem robotów mobilnych jest eksploracja stanowisk archeologicznych, czego przykładem może być europejski projekt Rovina [221].

Kolejnym obszarem zastosowań jest eksploracja planet oraz innych obiektów w przestrzeni kosmicznej. Przykładem może być misja Mars 2020 [176] (rys. 1.3) przeprowadzona przez NASA, w której oprócz łazika kołowego Perseverance, wykorzystano również helikopter Ingenuity. Jednym z celów pojazdu latającego było zbadanie okolicy w celu poprawy planowania trasy dla łazika kołowego jak i przyszłych misji.



Rysunek 1.3 Łazik marsjański Perseverance i helikopter Ingenuity [176] realizujące misję Mars 2020, której jednym z celów jest poszukiwanie życia pozaziemskiego

1.2 Cele pracy

Wykorzystanie systemów robotów mobilnych niesie ze sobą wiele nowych możliwości, jednak wprowadza też szereg problemów badawczych wymagających rozwiązania. Do takich problemów należy koordynacja robotów, dekompozycja zadań na mniej skomplikowane podzadania oraz ich alokacja. Kolejnym wyzwaniem jest odpowiednia wymiana informacji między robotami, co można określić mianem wspólnej percepcji. Dodatkowo pojawia się aspekt związany z radzeniem sobie z dynamiką środowiska oraz defektami występującymi w grupie robotów.

Niniejsza praca skupia się na problemie wymiany informacji między robotami na potrzeby efektywnego mapowania i lokalizacji. Jednym z jej celów było wykazanie następująco sformułowanej tezy.

Teza pracy: *Można zwiększyć efektywność metod rozwiązujących problem SLAM w systemach wielorobotowych przez wymianę informacji między robotami.*

Zagadnienie to wydaje się o tyle ważne, że wszystkie działania związane z koordynacją robotów wymagają informacji o ich stanie. Znajomość stanu robotów wiąże się z poziomem wiedzy robotów o otoczeniu. Aby zdobyć taką wiedzę roboty budują modele środowiska i próbują jak najdokładniej określić swoje położenie w tych modelach. Estymacja modeli i pozycji polega zwykle na fuzji danych z czujników, które obarczone są różnego rodzaju błędami pomiarowymi, dlatego ważnym zagadnieniem wydaje się wykorzystanie dodatkowych informacji, aby zmniejszyć niepewność uzyskiwanych rezultatów.

W trakcie realizacji celu pracy znaczące okazało się poszukiwanie odpowiedzi na kilka pytań pomocniczych:

- W jaki sposób przeprowadzić fuzję danych z poszczególnych czujników robota, aby stworzyć model otoczenia i zlokalizować w nim robota?

- W jaki sposób reprezentować środowisko i niepewności obserwacji?
- Jakie informacje warto przesyłać między robotami i jak przeprowadzić fuzję tych danych?
- W jaki sposób wykorzystać dane z innych robotów, aby poprawić efektywność mapowania?

Zarówno odpowiedzi na powyższe pytania, jak i dowody na poparcie tezy przedstawiono w dalszej części rozprawy.

1.3 Zakres pracy oraz wkład

Zakres pracy obejmuje przegląd literatury związanej z sensorami, sposobami reprezentacji sceny, metodami lokalizacji i mapowania, a także systemami wielorobotowymi. Ponadto opracowano rozwiązania wybranych problemów związanych z lokalizacją i mapowaniem oraz przeprowadzono liczne eksperymenty, między innymi:

- Opracowano i zaimplementowano metodę integracji map 3D z wielu robotów dla przypadku, gdy nie są znane początkowe pozycje robotów i roboty nie spotykają się w trakcie eksploracji. Metoda działa w oparciu o wspólne części map, detekcję i deskrypcję cech oraz ich dopasowanie. Rozwiązanie przewidziane zostało dla map w postaci drzew ósemkowych (ang. *octomap*) i może być wykorzystywane w heterogenicznych grupach robotów, ponieważ niepewność pomiarowa poszczególnych sensorów uwzględniana jest w wartościach prawdopodobieństw zajętości poszczególnych węzłów map. Ponadto przeprowadzono liczne badania eksperymentalne, których wyniki zaprezentowano w niniejszej rozprawie. Opracowanie zamieszczono w rozdziale 8. Na podstawie zrealizowanych badań powstały prace [53–55]. Zaimplementowane rozwiązanie zostało udostępnione jako oprogramowanie o otwartych źródłach [1].
- Opracowano serwera map 3D i przeprowadzono badania eksperymentalne, które umieszczono w rozdziale 9. W rezultacie przeprowadzonych prac powstała publikacja [56].
- Przeprowadzono badania eksperymentalne porównujące działanie wybranych metod SLAM z wykorzystaniem optycznego systemu śledzenia ruchu. Rezultaty eksperymentów umieszczono w rozdziale 5.
- Przeprowadzono eksperymenty porównujące działanie wybranych metod lokalizacji z wykorzystaniem optycznego systemu śledzenia ruchu obiektów. Uzyskane rezultaty zamieszczono w rozdziale 4. Na podstawie przeprowadzonych badań eksperymentalnych powstała praca [51] oraz częściowo praca [108].
- Opracowano i Zaimplementowano system umożliwiający automatyczny dobór parametrów metod SLAM w celu zwiększenia ich efektywności (rozdział 5).
- Zaprojektowano oraz zaimplementowano systemy wielu robotów mobilnych, wykorzystujące zarówno rzeczywiste roboty kołowe jak i symulowane roboty. Szczegółowy opis wykonanych systemów umieszczono w rozdziale 6.

- Opracowano i zaimplementowano sposób kalibracji położenia czujnika głębi oraz wykorzystania danych na potrzeby lokalizacji, mapowania i detekcji przeszkód (rozdział 2). Ponadto zaprezentowano opracowany detektor przeszkód wklęsłych wraz z rezultatami eksperymentów. Wyniki zostały wykorzystane w publikacjach [52, 57], a stworzone oprogramowanie udostępnione publicznie [46].

1.4 Struktura pracy

Praca rozpoczyna się od wprowadzenia do problemu lokalizacji i mapowania w systemach robotów mobilnych, następnie przedstawia urządzenia pomiarowe, metody reprezentacji środowiska oraz metody lokalizacji robotów, metody SLAM, kończąc na metodach lokalizacji i mapowania w systemach wielorobotowych, metodach dystrybucji i łączenia map, a także przedstawieniu zrealizowanych systemów wielorobotowych.

Rozdział 2 zawiera klasyfikację oraz charakterystyki metod pomiarowych sensorów wykorzystywanych do mapowania i lokalizacji przez roboty mobilne. Skupiono się na najczęściej używanych sensorach, czyli sensorach optycznych, ultradźwiękowych, systemach odometrycznych oraz czujnikach inercyjnych. Przedstawiono również opracowaną metodę kalibracji pozycji czujnika głębi w odniesieniu do płaszczyzny podłoża.

W rozdziale 3 przedstawiono klasyfikację metod reprezentacji środowiska, w którym poruszają się roboty. Omówiono metody metryczne, relacyjne oraz hybrydowe. Ponadto przeanalizowano korzyści związane z różnymi sposobami reprezentacji, a także porównano zalety i wady reprezentacji w dwóch i trzech wymiarach.

Rozdział 4 zawiera wprowadzenie do metod i systemów lokalizacji robotów mobilnych, przede wszystkim w ujęciu probabilistycznym. Metody lokalizacji zostały sklasyfikowane, a wybrane szczegółowo omówione. W rozdziale tym zawarto również opis badań eksperymentalnych dotyczących wybranych metod lokalizacji robotów kołowych.

W rozdziale 5 zostało przedstawione podejście opierające się na jednoczesnym lokalizowaniu i mapowaniu (SLAM) wraz z omówieniem kluczowych metod i analizą problemu zamykania pętli. W rozdziale tym umieszczono również opis przeprowadzonych eksperymentów dotyczących wybranych metod SLAM robotów kołowych.

W rozdziale 6 omówiono aspekty związane z systemami wielu robotów mobilnych. Przedyskutowano korzyści i wyzwania związane z takimi systemami w odniesieniu do pojedynczych robotów. Opisano również podstawowe sposoby wymiany danych, a także przedstawiono wprowadzenie do robotyki w chmurze.

Rozdział 7 dotyczy metod lokalizacji i mapowania dla wielu robotów. Przedstawiono w nim klasyfikację metod SLAM dla wielu robotów i omówiono wybrane metody.

W rozdziale 8 zaprezentowano opracowaną metodę łączenia map 3D wykorzystującą wspólne obszary na mapach. Metoda opiera się na na dekompozycji modelu, ekstrakcji i deskrypcji cech oraz dopasowaniu ich do cech drugiej mapy w celu określenia transformacji między mapami.

Rozdział 9 przedstawia opracowany serwer map 3D w systemach wielu robotów mobilnych oraz wyniki przeprowadzonych eksperymentów.

W rozdziale 10 zawarto opis zrealizowanego systemu wielu robotów, zarówno symulacyjnego jak i wykorzystującego kołowe roboty mobilne Turtlebot. Przedstawiono również architekturę systemu oraz ogólny opis środowiska badań eksperymentalnych.

Rozdział 11 stanowi podsumowanie wykonanej pracy oraz zawiera sugestie dotyczące dalszego rozwoju przedstawionych rozwiązań.

Rozdział 2

Systemy sensoryczne w lokalizacji i mapowaniu

W niniejszym rozdziale przedstawiono klasyfikację i charakterystyki metod pomiarowych sensorów wykorzystywanych do mapowania i lokalizacji, przez roboty mobilne. Zaprezentowano także opracowaną metodę kalibracji pozycji czujnika głębi w odniesieniu do płaszczyzny podłoża.

2.1 Wprowadzenie

Percepcja jest jednym z podstawowych zadań realizowanych przez autonomiczne systemy robotyczne. Posiadana wiedza o środowisku, w którym poruszają się roboty jest ściśle powiązana ze stopniem ich autonomii. Dzieje się tak, ponieważ planowanie działań wymaga informacji o warunkach, w jakich te działania będą realizowane. Roboty mogą posiadać wiedzę o otoczeniu a priori, a nawet wiedza ta może być przekazywana z zewnętrznych systemów. Wymagania odnośnie percepcji, zależą przede wszystkim od tego jak bardzo wymagające jest środowisko, w którym poruszają się roboty. Pomijając jednak dość hermetyczne systemy robotyczne (spotykane przede wszystkim w zakładach przemysłowych), dynamika zmian środowiska spowoduje, że bez lokalnych systemów percepcji, roboty nie będą mogły działać w pełni autonomicznie. Z tego powodu roboty mobilne wyposaża się w wiele różnych sensorów umożliwiających zbieranie informacji o środowisku w czasie rzeczywistym lub z nieznacznymi opóźnieniami. Oprócz wykrywania przeszkód statycznych i dynamicznych na potrzeby unikania kolizji, systemy sensoryczne wykorzystuje się też do budowy map środowiska oraz lokalizowania na nich robotów.

Aby zwiększyć niezawodność wykrywania przeszkód, czy działań związanych z lokalizacją, wykorzystuje się systemy sensoryczne składające się z sensorów o różnych zasadach działania. Umożliwia to minimalizowanie wpływu wad poszczególnych typów czujników. Sensory wykorzystywane w robotyce można sklasyfikować ze względu na sposób realizacji pomiarów. Wyróżnia się następujące klasy czujników:

- optyczne,
- ultradźwiękowe,
- radiowe,
- inercyjne,

- magnetyczne,
- oraz taktylne.

Czujniki optyczne wykorzystują do pomiaru promieniowanie elektromagnetyczne, w zakresie widzialnym i podczerwieni. W tej grupie wyróżnia się zarówno czujniki pasywne, jak kamery, które jedynie rejestrują światło z otoczenia oraz czujniki aktywne, które emitują promieniowanie, a następnie rejestrują wiązkę odbitą od obiektów otoczenia. Sensory optyczne są też wykorzystywane jako elementy enkoderów wykorzystywanych do pomiaru parametrów ruchu obrotowego.

Kolejną klasą sensorów, w które wyposaża się roboty mobilne są sensory ultradźwiękowe. Czujniki te do pomiaru odległości wykorzystują fale dźwiękowe, które znajdują się poza zakresem częstotliwości słyszalnych dla człowieka. Sensory ultradźwiękowe umożliwiają detekcję przeszkód wykonanych chociażby ze szkła, przez co wykorzystywane są w systemach bezpieczeństwa robotów. Ich wadą jest natomiast niewielka rozdzielczość kątowa.

Czujniki radiowe działają w oparciu o generowanie fal radiowych i rejestrowanie fal odbitych od przeszkód otoczenia. Podstawowym systemem radiowym wykorzystywanym do detekcji przeszkód i lokalizacji jest Radar (ang. *Radio Detection and Ranging*), stosowany między innymi w pojazdach autonomicznych. Wadą rozwiązań radiowych jest niewielka rozdzielczość kątowa, w porównaniu do czujników optycznych.

Czujniki inercyjne wykorzystywane są do pomiaru prędkości i przyspieszeń, zarówno liniowych jak i kątowych. Działanie sensorów opiera się na bezwładności, czyli własności fizycznej ciał posiadających niezerową masę. Wykorzystywane są przede wszystkim do lokalnej lokalizacji robotów.

Sensory magnetyczne działają w oparciu o detekcję pola magnetycznego. Najpopularniejsze są dwa typy czujników magnetycznych. Pierwszy z nich to kompasy, służące do detekcji pola magnetycznego Ziemi, co pozwala stosować je w lokalizacji globalnej. Drugi typ to enkodery, czyli sensory używane do pomiaru ruchu obrotowego kół, wałów silników lub ruchu liniowego elementów.

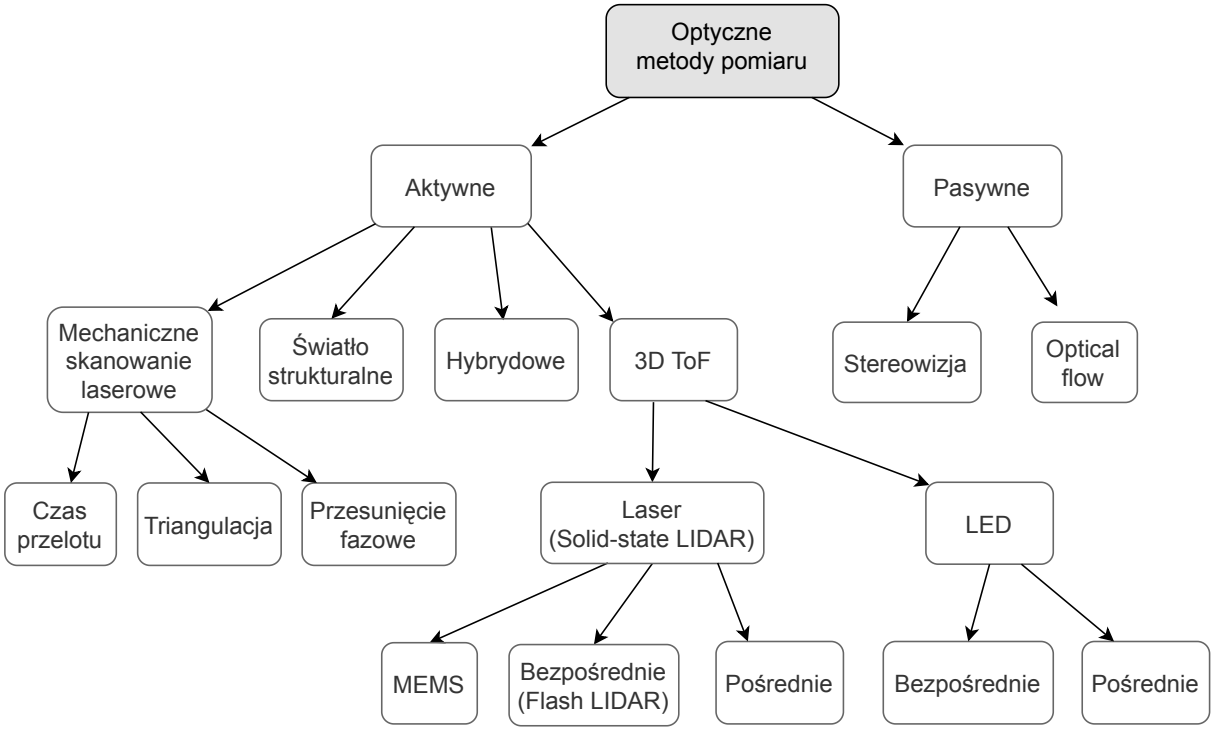
Czujniki taktylne umożliwiają wykrywanie bliskiej odległości robota do elementów otoczenia (przeszkód) lub wręcz kontaktu fizycznego z obiektami. Najczęściej mają postać zderzaków oraz matryc elementów rezystancyjnych lub pojemnościowych, które reagują na dotyk [64]. W roli czujników taktylnych wykorzystuje się też różnego rodzaju bariery optyczne, a także czujniki zbliżeniowe (pojemnościowe lub optyczne) [226]. Sensory taktylne stosowane są głównie w najniższych warstwach systemów bezpieczeństwa robotów, czyli ich zadaniem jest wykrycie kontaktu, gdy zawiodą pozostałe sensory.

W dalszej części przedstawiono charakterystykę wybranych typów sensorów.

2.2 Sensory optyczne

Czujniki optyczne są szeroko rozpowszechnione i stosowane w wielu aplikacjach. Wpływa na to ich precyzja działania, szybkość realizacji pomiarów oraz znaczny zasięg. Czujniki optyczne wykorzystują promieniowanie elektromagnetyczne do określania własności mierzonych obiektów lub odległości do nich. Wykorzystuje się zarówno promieniowanie w zakresie widzialnym, ale też w podczerwieni. Na rys. 2.1 przedstawiono klasyfikację czujników optycznych 2D oraz 3D. Jeden z podziałów dotyczy tego, czy sensory emitują energię do otoczenia i mierzą sygnały powracające (aktywne), czy wyłącznie pobierają energię z otoczenia (pasywne). Kolejne podziały dotyczą metod

realizacji pomiarów. Należy zaznaczyć, że istnieją urządzenia wykorzystujące mechaniczną metodę skanowania laserowego, realizujące pomiar zarówno w 2D jak i w 3D. Pozostałe metody, czyli wykorzystanie światła strukturalnego, sensory ToF oraz stereowizja, przeważnie realizują pomiar w trzech wymiarach. Znacznym zainteresowaniem



Rysunek 2.1 Klasyfikacja optycznych metod pomiaru odległości

cieszą się też czujniki RGB-D (ang. *Red Green Blue - Depth*). Szereg prób wykorzystania tego typu sensorów na potrzeby lokalizacji, mapowania oraz nawigacji opisano w pracach [42, 60, 179, 220, 233, 267]. W pracy [120] zaprezentowano metodę konwersji mapy głębi z sensora RGB-D do postaci 2D, zachowując przy tym informacje istotne z perspektywy systemu nawigacji robota.

Metody laserowego pomiaru odległości, określane także jako LIDAR (ang. *Light Detection And Ranging*), do pomiaru odległości wykorzystują laser (ang. *Light Amplification by Stimulated Emission of Radiation*), czyli urządzenie emitujące światło w oparciu o zjawisko emisji wymuszonej. Określanie odległości wykorzystuje detekcję wiązki światła odbitej od obiektów. Sensory wykorzystujące laser należą do grupy sensorów aktywnych, ponieważ emitują energię do otoczenia i mierzą wiązkę powracającą.

Podstawowy podział lidarów wyróżnia dwie grupy:

- sensory realizujące skanowanie mechaniczne, z wykorzystaniem odpowiednich układów optycznych,
- oraz sensory 3D ToF (ang. *Time of Flight*), określane też mianem stałych lidarów (ang. *Solid-state LIDAR*), które nie posiadają ruchomych elementów, przynajmniej w skali makro [126].

2.2.1 Mechaniczne skanery laserowe

Sensory realizujące mechaniczne skanowanie laserowe, do pomiaru wykorzystują odpowiednie układy optyczne. W przypadku skanerów laserowych 2D (rys. 2.2), reali-

zacja pojedynczego pomiaru urządzenia (skanu) składa się kilku elementów. Pierwszym z nich jest odpowiednia, punktowa metoda pomiaru odległości, czyli sposób samego pomiaru odległości do wybranego obiektu. Drugim aspektem jest metoda wykonania skanu, czyli realizacji wielu punktowych pomiarów w celu odwzorowania drugiego wymiaru otoczenia.

Rysunek 2.2 Skanery laserowe 2D: Hokuyo URG-04LX-UG01 oraz SICK S3000 [225]

Mechaniczne skanery laserowe 2D wykorzystywane są również do skanowania w trzech wymiarach [136]. Takie skanowanie realizuje się przez umieszczenie układu pomiarowego skanera na ruchomych podstawkach, umożliwiając tym samym jego ruch w dodatkowej płaszczyźnie.

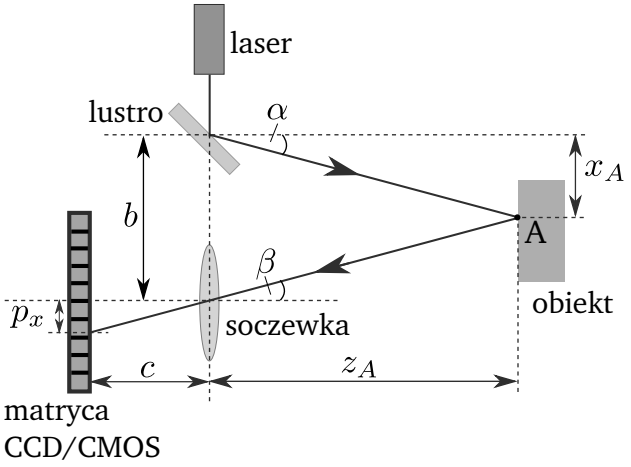
W skanerach laserowych wykorzystywane są następujące metody punktowego pomiaru odległości:

- triangulacja,
- określanie czasu przelotu ToF,
- określanie przesunięcia fazy PS (ang. *Phase Shift*).

Triangulacja

Triangulacja wykorzystuje pośredni pomiar odległości realizowany przez określanie kąta powracającej do czujnika wiązki lasera. Układ pomiarowy (rys. 2.3) składa się ze źródła lasera, lustra kierującego wiązkę, soczewki odbierającej wiązkę odbitą oraz zestawu elementów światłoczułych, na przykład matrycy CCD (ang. *Charge-Coupled Device*) lub CMOS (ang. *Complementary Metal-Oxide-Semiconductor*). Zasada działania opiera się na emitowaniu przez diodę laserową wiązki światła, która trafia do układu optycznego z lustrem, gdzie jest kierowana w stronę obiektu. Odbita od obiektu wiązka światła wraca do urządzenia i trafia do soczewki odbiornika. Soczewka odbiornika kieruje ją na matrycę elementów światłoczułych, co w rezultacie pozwala określić miejsce, w które trafiła wiązka lasera. W oparciu o p_x, czyli odległość punktu w który trafia wiązka od osi przechodzącej przez środek matrycy i soczewki oraz odległość c między matrycą, a soczewką, można obliczyć wartość

$$\tan \beta = \frac{p_x}{c}. \tag{2.1}$$



Rysunek 2.3 Pomiar odległości za pomocą triangulacji [24]

Zakładając, że kąt α, a także odległość bazowa b (rys. 2.3) są znane, odległość do obiektu wyraża się zależnością

$$z_A = \frac{b}{\tan\alpha + \tan\beta}, \tag{2.2}$$

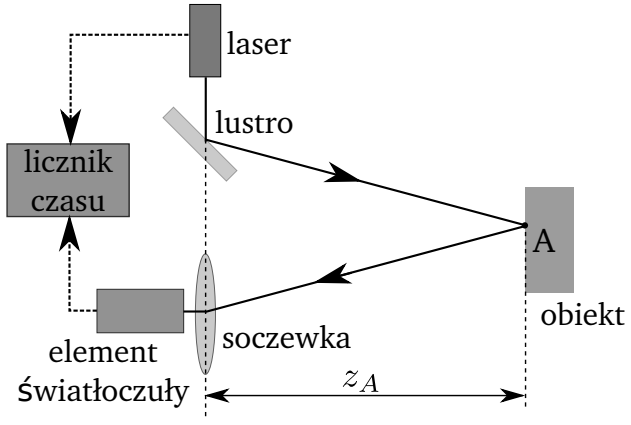
a położenie punktu w osi x określa zależność

$$x_A = \frac{b}{1 + \frac{\tan\beta}{\tan\alpha}}. \tag{2.3}$$

Triangulacja jest precyzyjną metodą [191, 229], ale charakteryzuje się mniejszym zasięgiem od pozostałych, przedstawionych metod pomiaru odległości. Dodatkowo, dokładność pomiaru zależy od barwy i struktury obiektu, którego dotyczy pomiar.

Określanie czasu przelotu

Metoda pomiaru czasu przelotu określana też jako metoda impulsowa, pozwala uzyskać dużą szybkość działania oraz znaczny zasięg pomiarów [12, 77] (rys. 2.4). Dzia-



Rysunek 2.4 Pomiar odległości z wykorzystaniem metody czasu przelotu

łanie polega na wygenerowanie impulsu światła z użyciem lasera oraz precyzyjnym

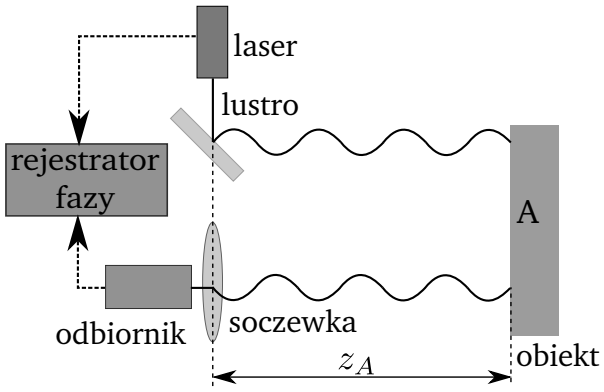
pomiarze czasu powrotu impulsu. Światło po odbiciu się od obiektu wraca do sensora, przechodzi przez soczewkę odbierającą i ostatecznie trafia do elementu światłoczułego, który wykrywa nadejście impulsu i zatrzymuje zliczanie czasu. Zmierzona wartość czasu Δt, przy znanej prędkości światła c, pozwala obliczyć drogę przebytą przez impuls

$$z_A = \frac{c\Delta t}{2}. \tag{2.4}$$

Dokładność metody zależy między innymi od współczynnika odbicia oraz nierówności powierzchni obiektu, do którego mierzona jest odległość.

Określanie przesunięcia fazy

Pośredni pomiar odległości opiera się na określeniu różnicy faz między sygnałem nadawanym i odbieranym. Na rys. 2.5 przedstawiono układ pomiarowy, składający się z rejestratora fazy, który mierzy przesunięcie faz sygnałów. Urządzenie laserowe gene-



Rysunek 2.5 Metoda pomiaru odległości na podstawie przesunięcia fazowego [77]

ruje sygnał sinusoidalny o długości fali λ, a informacja jest przenoszona z wykorzystaniem modulacji amplitudowej (ang. *Amplitude Modulation*) [229]. W oparciu o pomiar przesunięcia faz $\Delta\varphi$ między sygnałem nadawanym i odbieranym, odległość określa się następująco

$$z_A = \frac{\lambda}{2}\left(\frac{\Delta\varphi}{2\pi} + n\right), \tag{2.5}$$

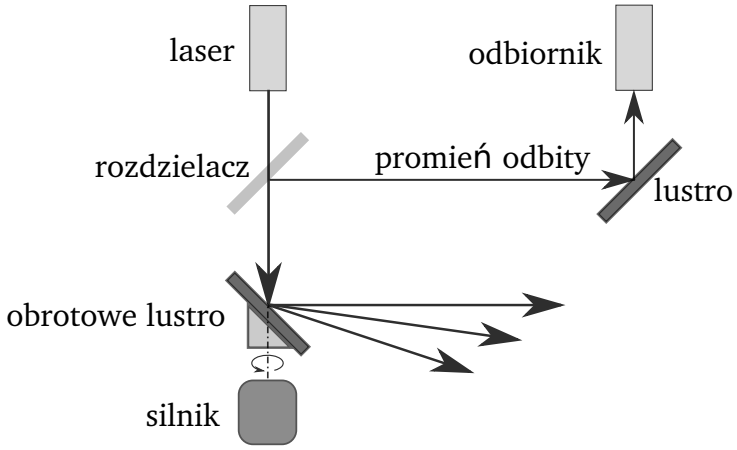
gdzie n określa wielokrotność długości fali λ między czujnikiem i obiektem mierzonym.

Metoda pomiaru odległości na podstawie przesunięcia fazowego posiada mniejszy zasięg w porównaniu do metody pomiaru czasu przelotu [191].

Techniki skanowania 2D z wykorzystaniem systemów optycznych

Omówione metody pomiaru umożliwiają wykonanie jednorazowego pomiaru odległości do określonego punktu. W celu określenia odległości do wszystkich obiektów znajdujących się w polu widzenia skanera FOV (ang. *field of view*), wykorzystywane są dodatkowe systemy optyczne. Najczęściej stosowane są następujące systemy [159, 191, 229]:

- płaskie, obrotowe lustra (rys. 2.6),
- pryzmatyczne, obrotowe lustra,
- obrotowe, piramidalne lustra,
- oscylujące lustra.



Rysunek 2.6 Układ optyczny z obrotowym lustrem [191]. Obracający się element optyczny odpowiada za kierowanie wiązki lasera w stronę badanych obiektów

2.2.2 Sensory 3D ToF

Sensory 3D ToF realizują pomiar odległości na podstawie pomiaru czasu przelotu światła [146, 178, 198] (rys. 2.7). Wyemitowane światło odbija się od otoczenia, a następnie rejestrowane jest przez odbiornik. Na podstawie zmierzonego czasu Δt_k oraz znanej prędkości światła c, dla każdego piksela k oblicza się odległości od czujnika do punktów otoczenia

$$z_k = \frac{c\Delta t_k}{2}. \tag{2.6}$$

Stałe lidary 3D ToF

Lidary 3D ToF (ang. *Solid-state*) stanowią grupę sensorów wykorzystujących laser do pomiaru odległości, ale w odróżnieniu od mechanicznych skanerów laserowych, nie posiadają ruchomych elementów w skali makro (przykładowy lidar przedstawiono na rys. 2.7). Czujniki te mierzą odległość na podstawie czasu przelotu światła, przy czym wyróżnia się kilka technik pomiarowych [126].

Jednym z rozwiązań jest wykorzystanie miniaturowych układów optycznych z ruchomymi lustrami wytworzonych w technologii MEMS (ang. *Micro-electromechanical Systems*) [186]. Działanie tej grupy sensorów jest najbardziej zbliżone do skanerów mechanicznych, jednak w tym przypadku możliwa zmiana orientacji pojedynczego lustra jest znacznie mniejsza, przez co wykorzystuje się kaskadowe układy optyczne.

Kolejną grupą urządzeń są lidary błyskowe (ang. *flash lidars*) [96]. Działanie sprowadza się do wyemitowania impulsu świetlnego, który oświetla otoczenie, a następnie

do odebrania światła odbitego przez macierz elementów światłoczułych. Na podstawie zmierzonych czasów określane są odległości dla poszczególnych punktów matrycy. W tym przypadku pomiar realizowany jest jednorazowo, czyli na podstawie jednego zarejestrowanego obrazu.

Wykorzystuje się również techniki pomiaru odległości przez pomiar przesunięcia fazowego między wiązką wyemitowaną i powracającą do odbiornika. Dużą zaletą lidarów solid-state jest znaczna miniaturyzacja i niezawodność, przez co coraz częściej znajdują zastosowanie w wielu aplikacjach, między innymi w autonomicznych samochodach.

Sensory LED 3D ToF

Zasada działania kamer 3D ToF jest analogiczna do działania lidarów solid-state, z tą różnicą, że w kamerach 3D ToF do emitowania światła wykorzystuje się inne źródła niż laser, przeważnie diody LED.

Czujniki ToF realizują skanowanie całego pola widzenia w pojedynczym pomiarze, co umożliwia wykonywanie pomiarów ze stosunkowo dużą częstotliwością. Jedną z zalet sensorów tego typu jest brak elementów ruchomych. Znaczna miniaturyzacja sensorów ToF sprawia, że są one wykorzystywane nie tylko w robotyce, ale są także montowane w smartfonach na potrzeby systemów detekcji i rozpoznawania twarzy.

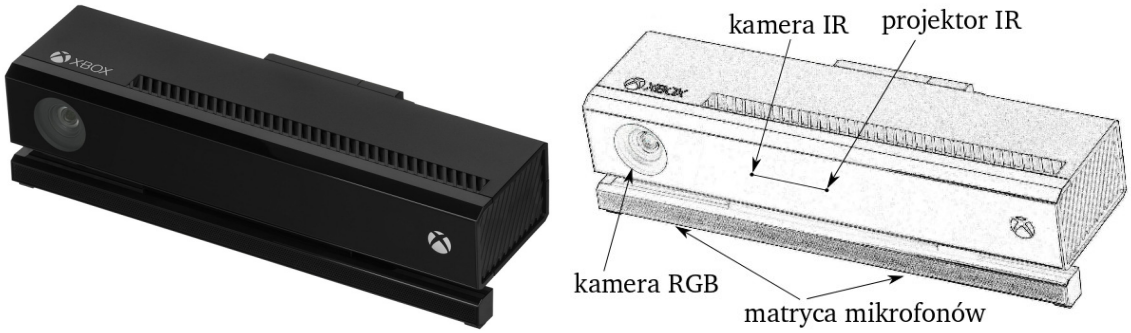
W sensorach tego typu przeważnie stosuje się jedną z dwóch metod pomiaru czasu przelotu światła: bezpośrednią lub pośrednią. Pierwszy sposób korzysta z bezpośredniego pomiaru czasu, w którym impulsy światła docierają do powierzchni badanego obiektu i powracają do odbiornika. Sensory korzystające z tej metody pomiarowej, zwykle wyposażane są w precyzyjne układy odmierzające czas. W metodzie pośredniej, pomiar odległości realizowany jest przez określenie przesunięcia fazowego między odpowiednio zmodulowaną, emitowaną i powracającą wiązką. Generowane sygnały mają częstotliwości rzędu dziesiątek MHz [178]. Pomiar przesunięcia fazowego między sygnałami, przy znanej długości fali umożliwia określenie odległości do obiektów otoczenia.

Sensor Microsoft Kinect v2

Jednym z przykładów czujników korzystających z metody ToF jest druga wersja sensora Microsoft Kinect, w którym zastosowano pośredni pomiar czasu przelotu [66, 139].

Rysunek 2.7 Kamera 3D ToF MESA Imaging SR4000 oraz lidar 3D Velodyne Puck (VLP-16) [249]

Rys. 2.8 przedstawia komponenty czujnika, natomiast w tabeli 2.1 znajdują się parametry urządzenia. Czujnik ten posiada kilka ulepszeń w porównaniu do pierwszej wersji,



Rysunek 2.8 Czujnik Microsoft Kinect v2

między innymi zapewnia większy zasięg, posiada zwiększone pole widzenia oraz wyższe rozdzielczości kamer. Przez zmianę metody pomiarowej osiągnięto mniejszy wpływ warunków oświetleniowych na otrzymywane dane [66].

2.2.3 Skanery wykorzystujące światło strukturalne

Technika pomiaru za pomocą światła strukturalnego (ang. *structured light*) bazuje na wykrywaniu zaburzeń wzoru świetlnego wyświetlanego na badanej powierzchni oraz triangulacji [70, 219]. Do projekcji wzoru na powierzchni badanego obiektu wykorzystuje się promienniki podczerwieni. Obserwacja wzoru realizowana jest przez kamery podczerwieni. Za pomocą dedykowanych algorytmów, systemy przetwarzania analizują obserwowane zniekształcenia wyświetlanego wzoru oraz określają geometrię badanego obiektu.

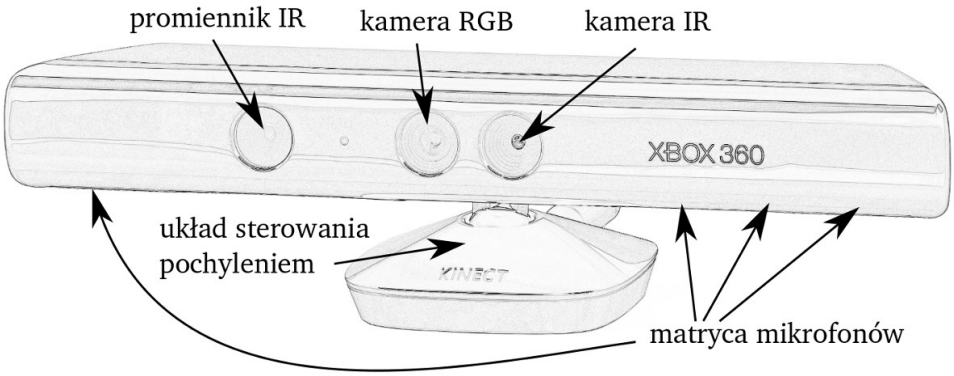
Wzory mogą mieć postać rastrową [219], a do ich generowania wykorzystywane są przeważnie prążki, kody Gray'a, kody binarne, sekwencje De Bruijn, przesunięcia fazowe oraz metody hybrydowe.

Zaletą metody światła strukturalnego jest szybkość wykonywania skanu, ze względu na uzyskiwanie informacji o całej powierzchni będącej w polu widzenia czujnika na podstawie pojedynczej klatki z kamery. Dodatkowo, sensory wykorzystujące światło strukturalne nie posiadają ruchomych elementów. Wadą tej metody pomiarowej jest jednak wrażliwość na oświetlenie zewnętrzne.

Sensor Kinect

Przykładem realizacji pomiaru odległości techniką światła strukturalnego jest sensor Microsoft Kinect (rys. 2.9). Pierwotnym przeznaczeniem tego urządzenia była funkcja kontrolera gier, który nie wymagałby bezpośredniego kontaktu z użytkownikiem. Jednak ze względu na znaczne możliwości i dużą dostępność, zyskał popularność w zastosowaniach robotycznych. Na rys. 2.10 przedstawiono komponenty wchodzące w skład urządzenia, czyli: kamera RGB, kamera IR (podczerwieni), promiennik IR, układ sterujący pochyleniem sensora, matryca mikrofonów oraz procesor sygnałowy. Parametry sensora umieszczono w tabeli 2.1, gdzie znajduje się porównanie kilku sensorów optycznych.

Rysunek 2.9 Sensory: Microsoft Kinect oraz Occipital Structure



Rysunek 2.10 Komponenty wchodzące w skład sensora Kinect

2.2.4 Systemy stereowizyjne

Stereowizja (ang. *stereo vision*) to pasywna metoda optyczna umożliwiająca obliczenie współrzędnych punktów sceny z wykorzystaniem obrazów uzyskiwanych z przynajmniej dwóch kamer wizyjnych [64]. Idea pomiarowa stereowizji inspirowana jest układami wzrokowymi występującymi powszechnie w przyrodzie, w tym u ludzi. W metodzie tej wykorzystuje się własność, że obserwacja dowolnej struktury przestrzennej z różnych położeń na płaszczyźnie prostopadłej do osi optycznych kamer pozwala uzyskać obrazy poprzecznie przesunięte. Stereowizyjny system pomiarowy (rys. 2.11), czyli układ kamer cechuje się równoległością osi optycznych, a także takimi samymi współrzędnymi osi Z ognisk kamer [214].

Aby obliczyć współrzędne punktów sceny należy dopasować do siebie przesunięte obrazy, czyli znaleźć pary odpowiadających sobie punktów na dwóch obrazach. Realizuje się to przez wykorzystanie algorytmów ekstrakcji i dopasowywania cech obrazów. Dopasowanie punktów obrazów pozwala wyznaczyć różnice współrzędnych poszczególnych punktów, co w rezultacie umożliwia uzyskanie obrazu dysparycji (ang. *disparity*). Do otrzymania mapy głębi wykorzystuje się triangulację i oblicza odległości z dla każdego punktu obrazu [214]

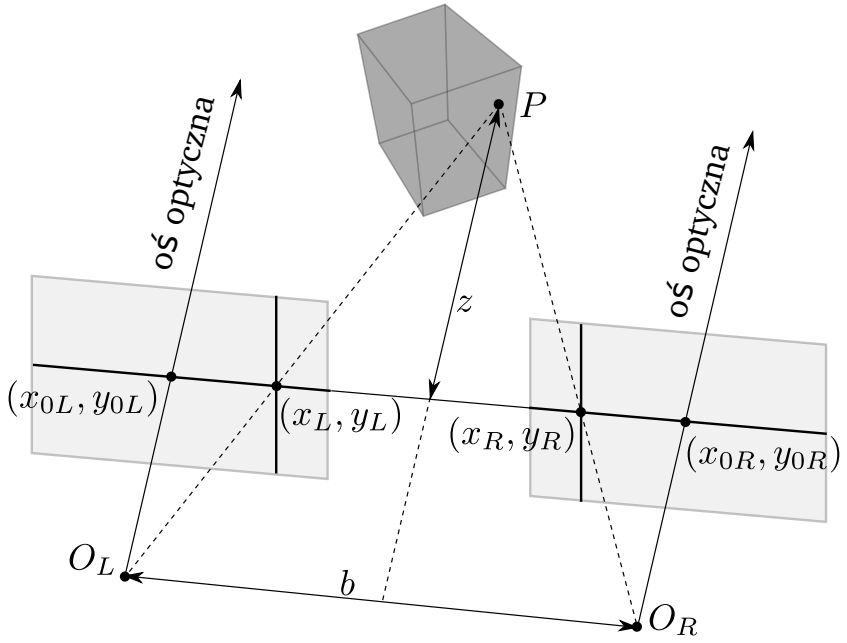
$$z = \frac{b \cdot f_p}{d_{RL}}, \tag{2.7}$$

gdzie:

b – określa bazę, czyli odległość między osiami optycznymi kamer,

f_p – to ogniskowa kamer,

d_{RL} – odwzorowania geometryczne punktów przestrzeni trójwymiarowej na płaszczyzny przetworników obrazowych.



Rysunek 2.11 Stereowizyjny układ pomiarowy wraz z zależnościami między punktami sceny i punktami na obrazach kamer

2.2.5 Sensory hybrydowe

Sensor Intel RealSense D435 (rys. 2.12) wykorzystuje hybrydową metodę pomiaru. Pomiar opiera się zarówno na stereowizji jak i świetle strukturalnym. Urządzenie wypo-

Rysunek 2.12 Sensor Intel RealSense D435 [105]

sażone zostało w dwie kamery IR oraz projektor IR. Projektor generuje statyczny wzór, który następnie rejestrowany jest przez dwie kamery. Dane przesyłane są do dedykowanego procesora sygnałowego, który oblicza odległości dla każdego piksela, bazując na przesunięciu między odpowiadającymi sobie elementami wzorów na dwóch obrazach.

Porównanie wybranych czujników optycznych

W tabeli 2.1 umieszczono zestawienie parametrów wybranych czujników optycznych używających różnych metod pomiarowych: Microsoft Kinect (rys. 2.9), Microsoft

Kinect v2 (rys. 2.8), Occipital Structure (rys. 2.9), Intel RealSense D435 (rys. 2.12) oraz skaner laserowy Hokuyo URG.

Tabela. 2.1 Porównanie parametrów dwóch wersji sensora Kinect, sensora Occipital Structure, Intel RealSense D435 oraz skanera laserowego Hokuyo URG-04LX-UG01 [66, 105, 118, 139, 267]

	Microsoft Kinect	**Microsoft Kinect v2**	**Occipital Structure**	**Intel RealSense D435**	**Hokuyo URG 04LX-UG01**
Technika pomiaru	światło strukturalne	czas przelotu	światło strukturalne	hybrydowa	przesunięcie fazy
Zasięg [m]	0,8 – 4,0	0,5 – 4,5	0,4 – 3,5	0,2 – 10	0,02 – 4,0
Dokładność	$\pm$6 mm (do 1m) $\pm$130 mm (do 4m)	——	——	$\leq$ 2% (do 2m)	$\pm$30 mm (do 1m) $\pm$3% (do 4m)
Poziomy kąt widzenia [^{o}]	57	70	58	87	240
Pionowy kąt widzenia [^{o}]	43	60	45	58	——
Rozdzielczość kątowa [^{o}]	$\approx$ 0,18	$\approx$ 0,14	——	——	0,36
Częstotliwość [Hz]	30	30	30/60	90	10
Rozdzielczość mapy głębi	320$\times$240	512$\times$424	640$\times$480	1280$\times$720	——
Rozdzielczość obrazu RGB	640$\times$480	1920$\times$1080	——	1920$\times$1080	——
Waga [g]	550	970	——	72	160
Rozmiary [mm]	65$\times$290$\times$70	66$\times$249$\times$97	28$\times$119$\times$29	90$\times$25$\times$25	50$\times$50$\times$70

2.3 Sensory ultradźwiękowe

Czujniki ultradźwiękowe, określane też mianem sonarów (ang. *Sound Navigation and Ranging*) do pomiaru wykorzystują ultradźwięki. Ultradźwięki to fale mechaniczne rozchodzące się w elastycznym ośrodku, o częstotliwościach powyżej 20 kHz, czyli powyżej progu słyszalności człowieka. Prędkość fali dźwiękowej w danym ośrodku

jest stała. W przypadku powietrza, prędkość ta zależy głównie od warunków atmosferycznych, w szczególności od temperatury i dla wysokości 0 m n.p.m. jest wyrażona wzorem [91, 183]

$$v_d = v_0\sqrt{1 + \frac{T_C}{273}}, \tag{2.8}$$

gdzie $v_0 = 331$ m/s oznacza prędkość rozchodzenia się fali dźwiękowej w powietrzu przy temperaturze $0°\mathrm{C}$, a T_C oznacza temperaturę powietrza w stopniach Celsjusza. Dla temperatury równej $20°\mathrm{C}$ prędkość dźwięku wynosi około 343 m/s.

Pomiar odległości korzysta ze znanej prędkości v_d fali dźwiękowej w danym ośrodku. Układ pomiarowy wykorzystuje nadajnik oraz odbiornik ultradźwięków. Nadajnik generuje fale ultradźwiękowe, które po odbiciu się od otoczenia wracają do czujnika i są wykrywane przez układ odbiornika. W oparciu o czas Δt, zmierzony od emisji sygnału do momentu odbioru jego echa, obliczana jest odległość do obiektu

$$s = \frac{v_d \Delta t}{2}. \tag{2.9}$$

Jednym ze sposobów określania czasu powrotu echa jest detekcja progowa [134]. Pomiar bazujący na prędkości przelotu fal ultradźwiękowych jest jednak obarczony pewnymi błędami. Do źródeł niedokładności można zaliczyć [135, 152]:

- zjawisko martwej strefy, uniemożliwiające pomiar odległości do obiektów znajdujących się zbyt blisko sensora. Jest to związane z czasem powrotu fali mechanicznej, który jest mniejszy niż czas potrzebny do wygaszenia drgań membrany układu nadawczego,
- wiązka sonaru przypomina kształtem stożek, a ze względu na szerokość tego stożka, nie można precyzyjnie określić położenia obiektu, co jest przyczyną niewielkiej rozdzielczości kątowej sensorów ultradźwiękowych,
- w przypadku zastosowania wielu czujników, mogą się one wzajemnie zakłócać, ponieważ sygnał odbity (echo) z poszczególnych sensorów może trafiać do innych czujników i w rezultacie powodować niepoprawny odczyt,
- występowanie odbić zakłócających pomiar, w szczególności przy próbie wykrywania kątów,
- trudności w wykrywaniu echa dla obiektów o niewielkiej szerokości. Dzieje się tak, ponieważ moc fali dźwiękowej odbitej od otoczenia (tła) może być znacznie większa niż moc sygnału odbitego od konkretnego obiektu,

Any zminimalizować wpływ źródeł niedokładności i zwiększyć możliwość detekcji przeszkód za pomocą sensorów ultradźwiękowych, stosuje się systemy sonarowe składające się z wielu odbiorników i nadajników. Jedno z takich rozwiązań można znaleźć w pracy [135].

Inną odmianą czujników ultradźwiękowych są radary dopplerowskie, umożliwiające pomiar prędkości względem innych obiektów (stałych lub ruchomych). Zasada pomiaru takich urządzeń wykorzystuje efekt Dopplera, czyli różnicę między częstotliwością fali generowanej i rejestrowanej przez poruszającego się obserwatora.

2.4 Detektory pola magnetycznego

Detektory pola magnetycznego (magnetometry) można podzielić na dwie kategorie: skalarne i wektorowe. Sensory należące do pierwszej grupy umożliwiają pomiar jedynie natężenia pola magnetycznego, podczas gdy czujniki z kolejnej grupy pozwalają określić również kierunek i zwrot pola. Spotyka się wiele odmian czujników magnetycznych, między innymi wibracyjne, wykorzystujące efekt Halla lub magnetorezystancję, a także sensory transduktorowe i protonowe [144]. Stosowane są także magnetometry anizotropowe (AMR, ang. *Anisotropic magnetoresistance*), działające w oparciu o anizotropową magnetorezystancję [162], czyli zjawisko zmiany rezystancji spowodowane zmianą orientacji pola magnetycznego względem przepływającego prądu. W czujnikach tych przeważnie wykorzystuje się stop niklu i żelaza, w którym występuje zjawisko anizotropowej magnetorezystancji. Magnetometry anizotropowe wykorzystuje się między innymi do pomiaru wektora pola grawitacyjnego Ziemi. Czujniki tego typu nazywane są też cyfrowymi kompasami. W lokalizacji, magnetometrów używa się w celu poprawy działania akcelerometru i żyroskopu przy wyznaczaniu orientacji obiektu.

2.5 Sensory inercyjne

Sensory inercyjne do pomiaru prędkości lub przyspieszeń wykorzystują zjawisko bezwładności. Sensory tego typu są szeroko stosowane w robotyce mobilnej, w szczególności na potrzeby lokalizacji i mapowania. Jednak ze względu na znaczną miniaturyzację, obecnie używane są niemal w każdej gałęzi przemysłu. Wykorzystywane są w motoryzacji, w systemach poprawiających bezpieczeństwo, znajdują zastosowanie także w układach stabilizacji obrazu kamer, w smartfonach na potrzeby lokalizacji, oraz w różnego rodzaju kontrolerach.

Sensory inercyjne wykonywane są najczęściej w technologii MEMS, która umożliwia integrację elementów elektronicznych i mechanicznych w mikro skali. Technologia ta niesie ze sobą wiele zalet, przede wszystkim niewielkie rozmiary urządzeń, ale też małą bezwładność elementów systemów. Powoduje to, urządzenia są bardziej odporne na uszkodzenia spowodowane przeciążeniami, a także mniej podatne na zmiany temperatur. Do sensorów inercyjnych można zaliczyć akcelerometry, które określają przyspieszenia liniowe oraz żyroskopy służące do pomiaru prędkości kątowych.

Akcelerometry

Akcelerometr mierzy przyspieszenie poruszającego się obiektu, czyli szybkość zmian jego prędkości. Akcelerometry wykonane w technologii MEMS projektowane są przeważnie w taki sposób, że umieszcza się element o określonej masie wewnątrz ramki. Element z ramką łączy się sprężynami w taki sposób, aby mógł on wykonywać ruch w danej płaszczyźnie. Kiedy akcelerometr przyśpiesza, wtedy na masę umieszczoną w ramce działają siły bezwładności (inercji). Do określenia wartości przyspieszenia mierzy się wychylenie masy zawieszonej na sprężynach, lub siły oddziaływania z sąsiednimi strukturami. Akcelerometry można podzielić na kilka typów, w zależności od wykorzystanej metody pomiarowej.

Akcelerometry pojemnościowe buduje się w taki sposób, aby masa zawieszona na sprężynach tworzyła jedną okładkę kondensatora, natomiast nieruchome elementy drugą okładkę. Zmiana położenia zawieszonej masy na którą działają siły bezwładności i

jednocześnie okładki kondensatora powoduje zmianę pojemności układu. Na podstawie zmiany pojemności określa się wychylenie elementu zawieszonego na sprężynach. Wynika to z zależności między pojemnością kondensatora C i odległością d między jego okładkami

$$C = \frac{S\varepsilon_0\varepsilon_r}{d}, \tag{2.10}$$

gdzie:

- S określa pole powierzchni okładek kondensatora,
- ε_0 to przenikalność elektryczna próżni,
- oraz ε_r określa przenikalność elektryczną izolatora rozdzielającego okładki utworzonego kondensatora.

Akcelerometry pojemnościowe są dość precyzyjnymi urządzeniami pomiarowymi, charakteryzującymi się niskim stosunkiem szumów do wartości sygnału wyjściowego oraz niewielką wrażliwością na zmiany temperatury. Są jednak wrażliwe na zakłócenia elektromagnetyczne, które mogą wpływać na pomiar pojemności.

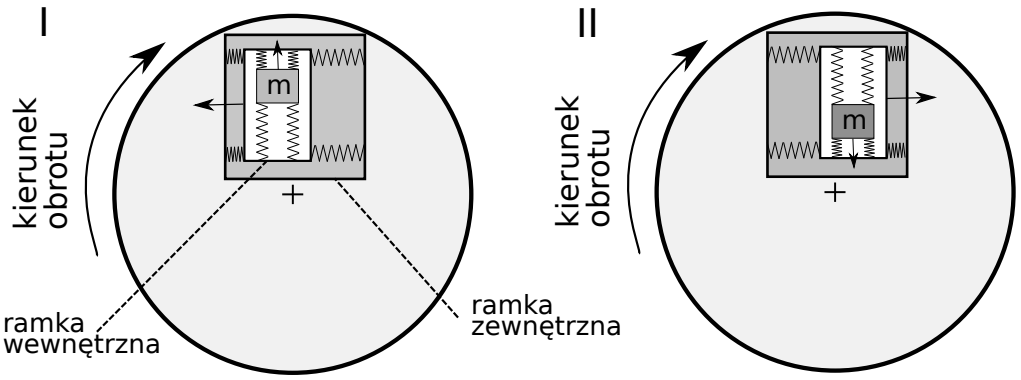
Kolejnym typem akcelerometrów są akcelerometry piezorezystancyjne. Akcelerometry piezorezystancyjne działają w oparciu o efekt piezorezystancyjny [149], czyli zjawisko polegające na zmianie rezystancji elementu wykonanego z odpowiedniego materiału pod wpływem ciśnienia działającego na ten element. Sensory tego typu również posiadają element o określonej masie, na który działają siły inercji. Element jest połączony z ramką za pomocą belek wykonanych z materiałów piezorezystancyjnych. W momencie, gdy na akcelerometr działa przyśpieszenie, wtedy belki pomiarowe są ściskane i rozciągane, a przez pomiar zmian rezystancji elementów, określa się wartość przyspieszenia. Akcelerometry piezorezystancyjne są wrażliwe na zmiany temperatury, oraz są mniej czułe w porównaniu do sensorów pojemnościowych.

Żyroskopy

Żyroskopy [64, 224] umożliwiają pomiar położenia kątowego lub prędkości kątowej. Prędkościowe żyroskopy wibracyjne wykorzystują efekt Coriolisa [83]. Czujnik jest zbudowany w ten sposób, że umieszcza się ramkę z rezonującym elementem o określonej masie na obrotowej tarczy (rys. 2.13). Element, zamontowany jest na sprężynach, tak aby mógł poruszać się tylko w jednym kierunku. Kiedy tarcza obraca się z daną prędkością, wibrująca masa oddziałuje na ramkę i tarczę siłą, skierowaną prostopadle do kierunku wibracji. Aby umożliwić pomiar siły, masę wraz z ramką umieszcza się na sprężynach w kolejnej ramce, w taki sposób aby wewnętrzna ramka mogła poruszać się jedynie prostopadle do ruchu wibracyjnego. W celu określenia prędkości kątowej żyroskopu w danej osi, mierzy się wychylenie wewnętrznej ramki względem zewnętrznej. Do pomiaru wychylenia wykorzystuje się analogiczne metody jak w przypadku akcelerometrów, czyli metodę pojemnościową lub piezorezystancyjną.

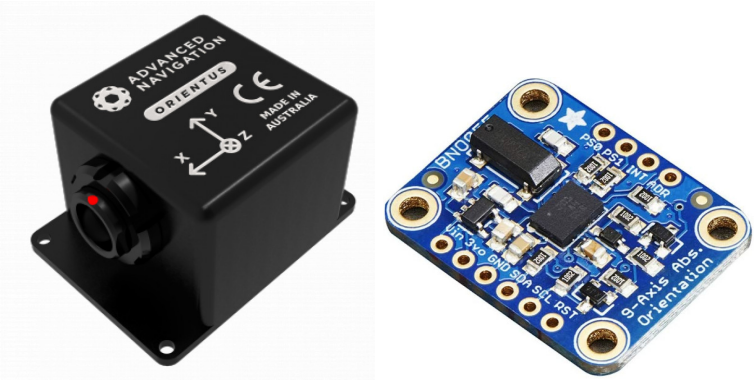
IMU

Sensory inercyjne, czyli akcelerometr i żyroskop umieszcza się również w jednym module (rys. 2.14), w ramach systemu pomiarowego, nazywanego inercyjną jednostką pomiarową IMU, (ang. *Inertial Measurement Unit*). Celem takiej jednostki jest zapewnienie informacji o charakterystyce ruchu i estymacja orientacji obiektu do którego



Rysunek 2.13 Zasada działania żyroskopu wibracyjnego. Rysunek przedstawia tarczę żyroskopu wraz z ramkami i rezonującą masą [83]. Od kierunku poruszania się rezonującej masy, zależy kierunek wychylenia ramki wewnętrznej względem zewnętrznej. W przypadku *I*, masa porusza się na zewnątrz tarczy, co przy ustalonym kierunku obrotu tarczy powoduje, że wewnętrzna ramka odchyla się w lewo (efekt Coriolisa). Analogicznie w przypadku *II*, ramka odchyla się w drugą stronę

została zamontowana. System ten może być wyposażony w odpowiedni algorytm realizujący filtrację i fuzję danych z poszczególnych sensorów. Przykładem może być filtr Kalmana [104] lub filtr Madgwicka [154]. Często w ramach IMU integruje się również



Rysunek 2.14 Przykładowe urządzenia pomiarowe IMU: Advanced Navigation Orientus [5] oraz Bosch BNO055

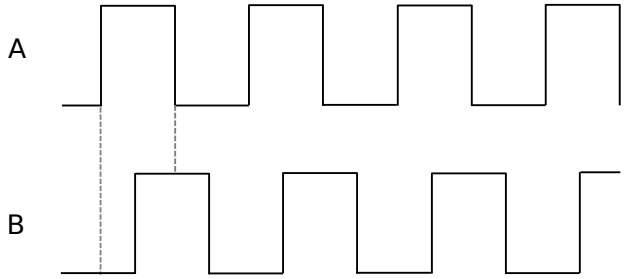
magnetometr, aby umożliwić estymację orientacji obiektu względem pola magnetycznego Ziemi. Pozwala to wyeliminować dryf orientacji, ale wymaga odpowiednich metod kalibracji.

2.6 Enkodery

Enkodery, czyli przetworniki obrotowo-impulsowe są podstawą działania wielu metod lokalizacji i mapowania robotów kołowych. W zależności od tego, jakich informacji dostarczają enkodery, można je podzielić na dwie grupy: inkrementalne i absolutne.

Enkodery inkrementalne mierzą kąt obrotu badanego obiektu. Powszechnie stosowanym rozwiązaniem w enkoderach inkrementalnych jest zwracanie uzyskiwanych wartości w postaci dwóch, wygenerowanych sygnałów prostokątnych, przesuniętych

względem siebie w fazie o $\pi/2$. Sygnały te nazywa się też sygnałami kwadraturowymi (rys. 2.15). Wartość kąta odpowiadająca jednemu impulsowi sygnału zdefiniowana



Rysunek 2.15 Sygnały kwadraturowe generowane przez enkodery inkrementalne. Na podstawie przesunięcia fazowego między sygnałami, można określić kierunek obrotu.

jest przez rozdzielczość enkodera. Dla przykładu, enkoder o rozdzielczości równej 8 bitów generuje $2^8 = 256$ impulsów w trakcie jednego, pełnego obrotu. Enkodery inkrementalne najczęściej wykonywane są jako czujniki optyczne lub magnetyczne. W wielu zastosowaniach ograniczeniem tej grupy enkoderów jest brak możliwości określenia aktualnej pozycji kątowej elementu, którego obroty są mierzone.

Enkodery absolutne pozwalają uzyskać informację o bieżącym położeniu kątowym badanego elementu. W porównaniu do enkoderów inkrementalnych cechują się mniejszą prędkością działania (umożliwiają pomiary przy mniejszej prędkości obrotowej) oraz mniejszą rozdzielczością.

Enkodery optyczne

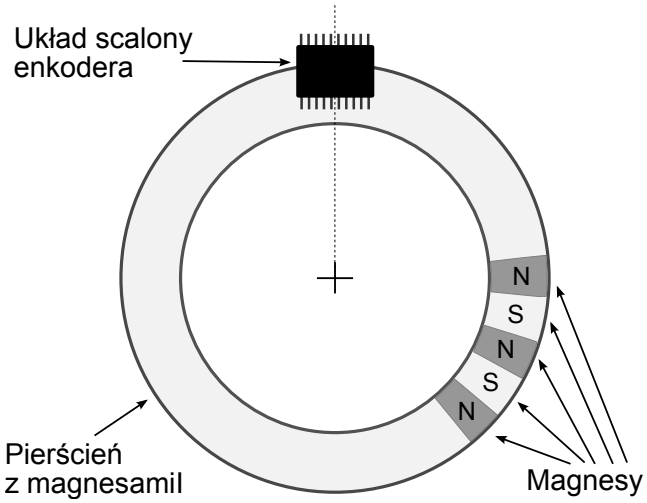
Inkrementalne enkodery optyczne wykonuje się w postaci tarczy z otworami na brzegu, którą umieszcza się na badanym, obracającym się elemencie. Po jednej stronie tarczy umieszcza się fotoelement, a z drugiej emiter w postaci diody IR. W momencie, gdy tarcza się obraca, element światłoczuły wykrywa zmiany natężenia światła, które występują, gdyż emiter jest naprzemiennie odkrywany i zakrywany. Pozwala to wygenerować sygnał prostokątny, którego częstotliwość jest proporcjonalna do prędkości obrotowej tarczy. Liczba otworów umieszczonych na tarczy determinuje rozdzielczość enkodera optycznego.

W celu uzyskania informacji o kierunku obrotu tarczy, umieszcza się emiter po jednej stronie tarczy i dwa elementy światłoczułe po drugiej stronie Fotoelementy są od siebie oddalone, co w rezultacie powoduje, że generowane sygnały są względem siebie przesunięte w fazie i tworzą sygnał kwadraturowy (rys. 2.15).

Enkodery magnetyczne

Enkodery magnetyczne zbudowane są w oparciu o czujniki magnetorezystancyjne lub czujniki Halla [95]. Dostępne są zarówno w wersji inkrementalnej jak i absolutnej. Pod względem budowy, enkodery magnetyczne można podzielić na dwie grupy.

W pierwszej grupie, enkodery składają się z sensora i pierścienia wyposażonego w magnesy o naprzemiennych biegunach. Pierścień montuje się na badanym, obrotowym obiekcie, a sensor umieszcza się w płaszczyźnie równoległej do płaszczyzny pierścienia, oraz w ustalonej odległości (rys. 2.16), aby zapewnić poprawne działanie czujników magnetorezystancyjnych. Czujnik wykrywa różnice w biegunowości kolejnych



Rysunek 2.16 Enkoder magnetyczny składający się z tarczy z magnesami oraz układu scalonego

pól magnetycznych i rejestruje zmianę położenia kątowego. Rozdzielczość enkodera jest zależna od liczby detektorów pola magnetycznego umieszczonych w układzie sensora, ale także od liczby magnesów.

W drugiej grupie można umieścić układy enkoderów składające się z okrągłego magnesu, podzielonego wzdłuż średnicy na dwie części o odmiennym biegunie magnetycznym. Układ czujnika umieszcza się w płaszczyźnie równoległej do płaszczyzny magnesu zamontowanego do obracającego się obiektu. W przypadku takich czujników, układ scalony składa się z matrycy czujników wykrywających zmiany pola magnetycznego, co pozwala rejestrować obrót magnesu, przy czym rozdzielczość jest zależna od liczby czujników umieszczonych w matrycy.

2.7 Przetwarzanie danych z czujników głębi

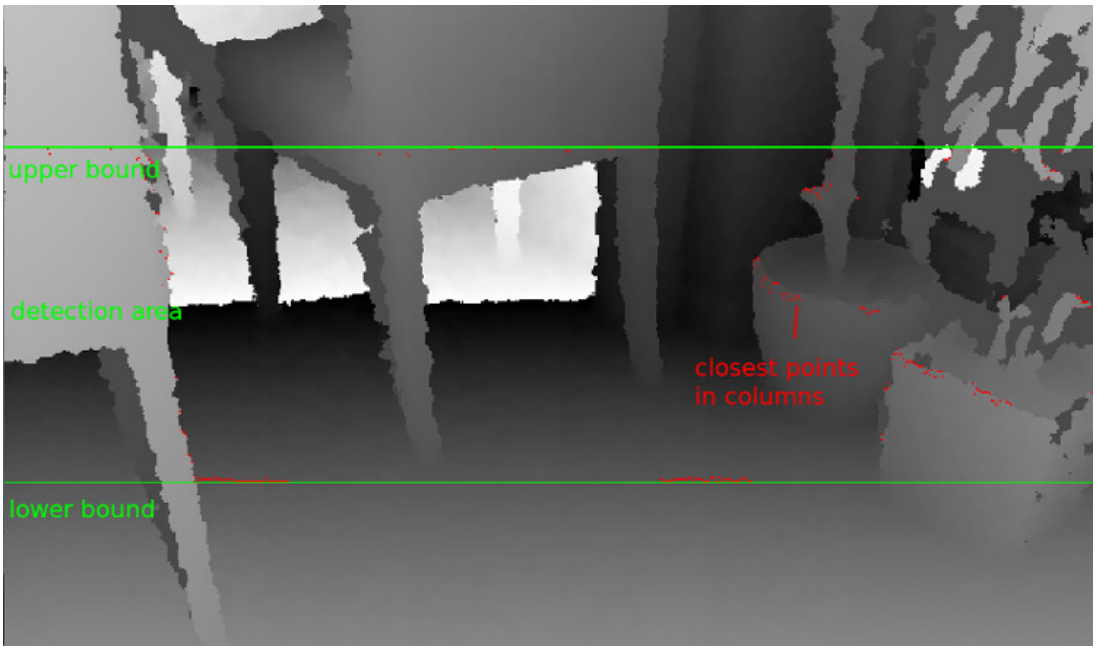
Sensory głębi, niezależnie od wykorzystanej techniki pomiarowej, pozwalają zmierzyć odległości do obiektów znajdujących się w ich polu widzenia. W niektórych zastosowaniach, na potrzeby lokalizacji i mapowania, dane z czujników 3D są konwertowane do postaci 2D, głównie w celu optymalizacji lub kompatybilności z metodami SLAM 2D.

Wymagania stawiane metodzie konwersji mapy głębi do postaci 2D dotyczą przede wszystkim możliwości kompensacji kąta pochylenia wzdłużnego sensora głębi, a także usuwania punktów podłoża, co ma duże znaczenie w przypadkach, kiedy sensor umieszczony jest blisko podłoża.

Dostępnych jest kilka rozwiązań problemu konwersji danych. Jednym z nich jest pakiet w środowisku ROS (*depthimage_to_laserscan*) [199]. Jednak to rozwiązanie nie umożliwia usuwania podłoża ani kompensacji kąta pochylenia czujnika głębi. Kolejną możliwością jest konwersja mapy głębi do chmury punktów i filtrowanie takiego zbioru. Podejście oparte na chmurze punktów wymaga jednak dodatkowego przetwarzania, co zwiększa złożoność obliczeniową rozwiązania.

W związku z tym, opracowano rozwiązanie umożliwiające konwersję danych do postaci 2D, przy jednoczesnym usuwaniu podłoża i kompensacji kąta pochylenia czujnika.

Przykład działania zaprezentowano na rys. 2.17. Rozwiązanie przedstawiono wcześniej w pracy [52].



Rysunek 2.17 Przykład działania stworzonego oprogramowania do konwersji mapy głębi do postaci 2D. Dwie zielone linie wyznaczają granicę strefy z której punkty uwzględniane są przy wyznaczaniu skanu. Na czerwono oznaczono najbliższe punkty w poszczególnych kolumnach obrazu, mieszczące się w strefie detekcji

Kompensacja pochylenia wzdłużnego sensora

Z powodu ograniczonych kątów widzenia czujników głębi, okazuje się korzystne umieszczanie czujników możliwie wysoko, ale w pozycji pochylonej do podłoża (rozdział 2.7). Wymaga to uwzględnienia pochylenia sensora podczas obliczania odległości do przeszkód na podstawie mapy głębi.

W celu uzyskania pełnego odczytu w danej płaszczyźnie, dla każdego pomiaru oblicza się odległości do obiektu z_i oraz przesunięcie x_i względem środka optycznego sensora S. Metoda konwersji bazuje na założeniu, że dla systemu detekcji przeszkód najważniejsze są informacje o najbliżej położonych przeszkodach. Z tego powodu, przy projekcji obrazu głębi na płaszczyznę, wyszukuje się najmniejsze wartości w poszczególnych kolumnach obrazu głębi

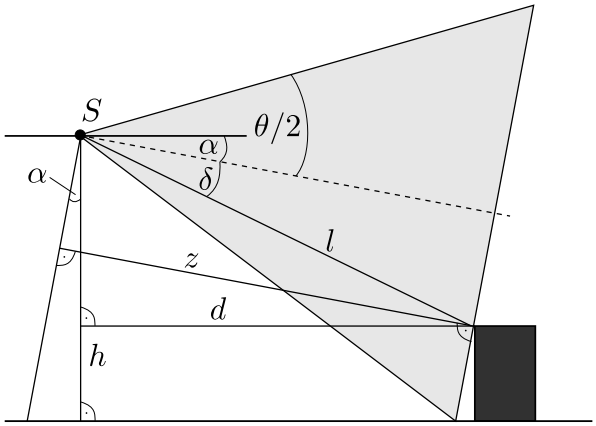
$$z_i = min(z_{0,i}, z_{1,i}, \ldots, z_{n,i}), \tag{2.11}$$

gdzie $z_{i,j}$ oznacza odległość dla i-tej kolumny i j-tego rzędu obrazu. Zakłada się, że pochylenie boczne nie występuje, co ma uzasadnienie, gdy roboty poruszają się w pomieszczeniach. Przesunięcie względem środka optycznego sensora

$$x_i = (i - c_x) z_i \frac{1}{f_x}, \tag{2.12}$$

wykorzystano model kamery, gdzie (c_x, c_y) oznacza współrzędne środka obrazu, natomiast f_x określa długość ogniskowej w kierunku poziomym. Zakłada się również, że kąt

pochylenia wzdłużnego sensora α oraz wysokość sensora od podłoża (środka optycznego kamery) h są znane. Na rys. 2.18 przedstawiono w przekroju poprzecznym zależności geometryczne między odczytami sensora oraz sceny. Kąt między osią optyczną



Rysunek 2.18 Kompensacja pochylenia czujnika - zależności geometryczne w przekroju poprzecznym sceny

i promieniem padającym na matrycę kamery zależy od numeru rzędu obrazu j_{min}, w którym została wykryta przeszkoda

$$\delta = \theta \frac{j_{min} - c_y - \frac{1}{2}}{n-1}, \tag{2.13}$$

gdzie n oznacza liczbę wszystkich rzędów obrazu, a θ określa pionowy zakres kątów widzenia sensora. Biorąc pod uwagę kąt pochylenia wzdłużnego czujnika α, odległość do przeszkody można określić zależnością

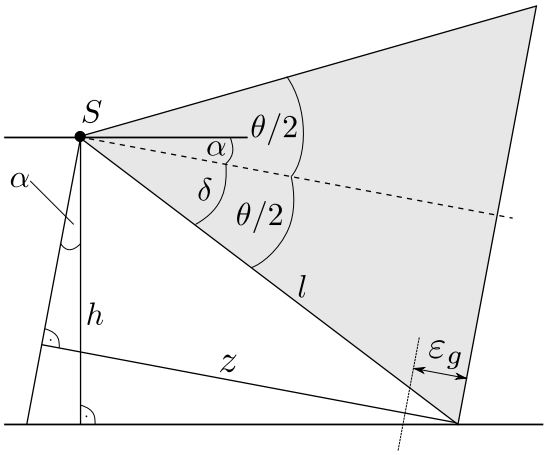
$$d = l\sin(\frac{\pi}{2} - \alpha - \delta) = z \cdot \frac{\sin(\frac{\pi}{2} - \alpha - \delta)}{\sin(\frac{\pi}{2} - \delta)}. \tag{2.14}$$

Usuwanie podłoża z danych pomiarowych

Zaprezentowana metoda służy do usuwania podłoża z wynikowych danych. Rozwiązanie bazuje na określeniu wartości progowych odległości dla każdego rzędu obrazu. Jeżeli odległości są powyżej progu, punkty z tego rzędu zaliczane są do podłoża i w konsekwencji nie są uwzględniane przy wyznaczaniu minimalnych odległości w poszczególnych kolumnach. Opracowana metoda zakłada, że kąt pochylenia sensora α oraz wysokość względem podłoża h są znane.

Na rys. 2.19 przedstawiono zależności w przekroju poprzecznym sceny, dla przypadku, gdy $\delta = \theta/2$. Z powodu niedokładności wartości kąta pochylenia sensora, jego wysokości względem podłoża oraz błędy pomiarów, zmierzone wartości zaliczane są do podłoża z określoną tolerancją ε_g. Zbiór odrzucanych pomiarów jest następujący

$$P_g = \left\{ (x,z) \mid z \geq h\frac{\sin\left(\frac{\pi}{2} - \delta\right)}{\cos\left(\frac{\pi}{2} - \delta - \alpha\right)} - \varepsilon_g \right\}. \tag{2.15}$$



Rysunek 2.19 Usuwanie podłoża z danych pomiarowych - zależności geometryczne

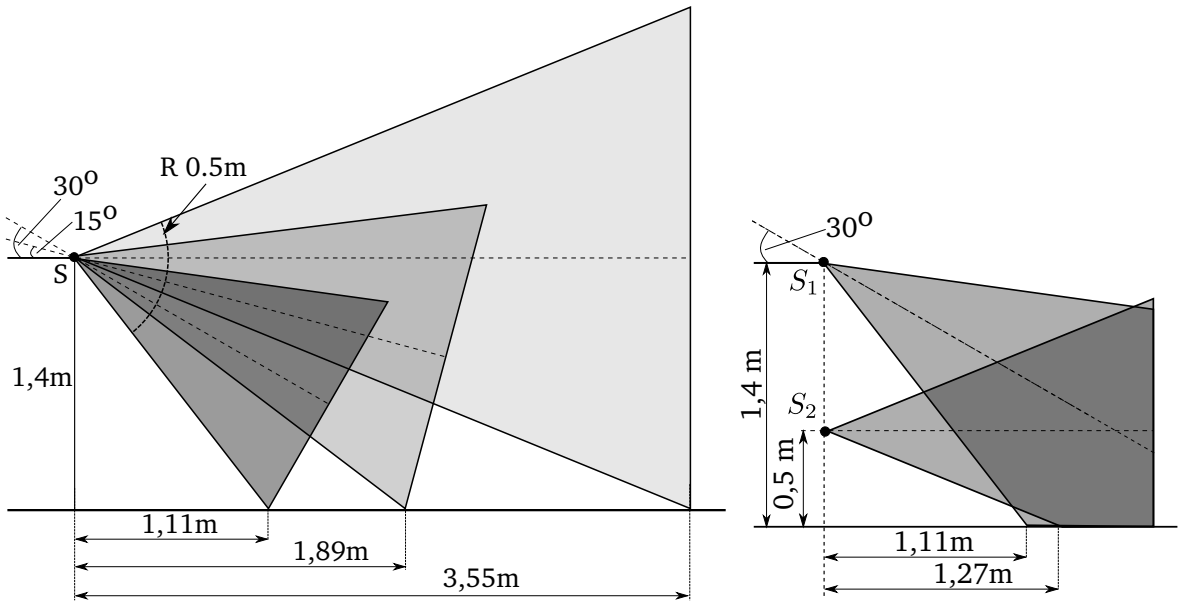
Wpływ położenia i orientacji sensora na otrzymywane dane

Wpływ położenia i orientacji sensora głębi na uzyskiwane informacje o otoczeniu jest bardzo duży. Wiąże się to z kątami widzenia czujnika, które w przypadku sensora Kinect, wynoszą 43° wertykalnie oraz 57° horyzontalnie. Jeżeli czujnik zostanie umieszczony za nisko, a kąt pochylenia do podłoża będzie duży, lub kiedy sensor skierowany zostanie w górę, to użyteczność otrzymywanych z niego informacji będzie niewielka.

Martwa strefa występuje między innymi poniżej pola widzenia czujnika (rys. 2.20). Zakładając, że $h_1 \leq h$, rozmiar strefy d_m można wyrazić zależnością

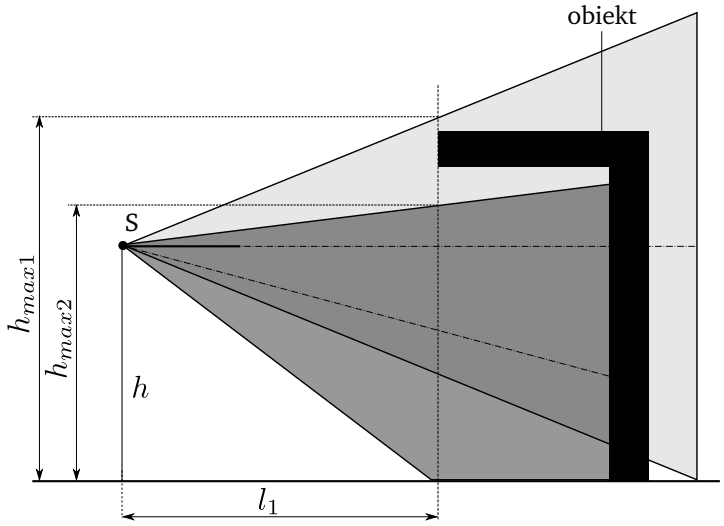
$$d_m = (h - h_1) \cdot \tan\left(\frac{\pi}{2} - \frac{\theta}{2} - \alpha\right) \tag{2.16}$$

gdzie h_1 oznacza wysokość od podłoża, na której długość strefy jest szukana.



Rysunek 2.20 Długość dolnej martwej strefy w zależności od wartości kąta pochylenia wzdłużnego sensora oraz wysokości montażu

Kolejnym parametrem jest rozmiar martwej strefy powyżej obszaru widzenia sensora (rys. 2.21). Jest to maksymalna wysokość do jakiej sensor wykrywa obiekty w zależności



Rysunek 2.21 Rozmiar górnej martwej strefy, w zależności od pozycji umieszczenia czujnika na robocie. W przedstawionym przypadku, obiekt nie zostaje częściowo wykryty, ponieważ znajduje się w górnej martwej strefie sensora

od odległości obiektu. Ma to znaczenie w przypadku niejednolitych przeszkód, posiadających wolną przestrzeń, takich jak krzesła, biurka czy stoły. W celu wyznaczenia tego parametru wykorzystano zależność

$$h_{max} = h + sgn\left(\frac{\theta}{2} - \alpha\right) \cdot l_1 \cdot \tan\left(\left|\frac{\theta}{2} - \alpha\right|\right), \tag{2.17}$$

gdzie l_1 oznacza odległość od obiektu do płaszczyzny pionowej przechodzącej przez środek optyczny sensora.

2.8 Estymacja kąta pochylenia i wysokości sensora głębi

Biblioteka PCL (ang. *Point Cloud Library*) [188] zawiera rozwiązanie do estymacji parametrów podłoża. Rozwiązanie to wykorzystuje punkty wybrane przez użytkownika na obrazie pochodzącym z sensora RGB-D. Jednak w celu wyeliminowania ręcznego określania punktów podłoża powstała metoda, która pozwala zautomatyzować proces estymacji parametrów.

Opracowana metoda umożliwia estymację kąta pochylenia wzdłużnego α oraz wysokości h sensora głębi w odniesieniu do podłoża. Metoda została opisana i umieszczona w publikacjach [52,57]. Rozwiązanie bazuje na algorytmie RANSAC (ang. *Random Sample Consensus*) [69], który estymuje parametry modelu płaszczyzny podłoża na podstawie otrzymanych punktów otoczenia [31, 258].

Do określenia parametrów płaszczyzny podłoża wykonuje się wstępną selekcję punktów należących z dużym prawdopodobieństwem do podłoża. Sposób wybierania punktów podłoża zakłada, że największe prawdopodobieństwo tego, że punkty należą do podłoża jest w dolnej części mapy głębi (poniżej ustalonych wartości progowych). Z tego powodu ustala się zgrubne przedziały wartości wysokości $[h_{min}, h_{max}]$ oraz kąta pochylenia $[\alpha_{min}, \alpha_{max}]$, a następnie określa progowe wartości odległości.

Wybór punktów podłoża

Selekcja punktów, które potencjalnie należą do podłoża wykonywana jest automatycznie. W celu zwiększenia prawdopodobieństwa przynależności punktów do podłoża wykorzystuje się orientacyjne wartości kąta pochylenia α_{est} oraz wysokości h_{est}. Dla tych wartości parametrów, w każdej kolumnie oblicza się maksymalny wiersz, poniżej którego punkty zaliczane są do podłoża. Punkty znajdujące się powyżej wartości progowej są odrzucane. Pozostałe punkty, w przypadku których istnieje podejrzenie, że należą do podłoża, sprawdzane są pod kątem zawierania się w granicy błędu ε_f. Wykorzystuje się w tym celu zależność

$$d_{th} = h_{est} \cdot \frac{\sin\left(\frac{\pi}{2} - \delta\right)}{\cos\left(\frac{\pi}{2} - \delta - \alpha_{est}\right)}, \tag{2.18}$$

przy czym δ określona jest równaniem (2.13). Zbiór punktów wykorzystywany w algorytmie RANSAC do estymacji parametrów modelu płaszczyzny jest następujący

$$P = \left\{ (x, y, z) \mid z \in \left[h_{min} \frac{\sin\left(\frac{\pi}{2} - \delta\right)}{\cos\left(\frac{\pi}{2} - \delta - \alpha_{max}\right)},\ h_{max} \frac{\sin\left(\frac{\pi}{2} - \delta\right)}{\cos\left(\frac{\pi}{2} - \delta - \alpha_{min}\right)} \right] \right\}. \tag{2.19}$$

Współrzędne punktu w przestrzeni na podstawie odległości $z_{i,j}$ punktu (i, j) mapy głębi o rozmiarach $m \times n$ wyznacza się następująco [121]

$$(x_{i,j}, y_{i,j}) = \left((i - c_x)\frac{1}{f_x} \cdot z_{i,j},\ (j - c_y)\frac{1}{f_y} \cdot z_{i,j} \right). \tag{2.20}$$

Algorytm RANSAC

Wspomniany już wcześniej algorytm RANSAC [47, 69] jest probabilistyczną metodą estymującą parametry danego modelu matematycznego, na podstawie zbioru danych. Zakłada się, że zbiór danych zawiera próbki pasujące do modelu (ang. *inliers*) oraz próbki odbiegające od modelu (ang. *outliers*). Prawdopodobieństwo otrzymania poprawnego rezultatu zwiększa się wraz z rosnącą liczbą wykonywanych iteracji. Kroki algorytmu są następujące:

1. Wylosowanie minimalnej liczby próbek, umożliwiającej obliczenie parametrów modelu. W przypadku płaszczyzny wymagane są trzy punkty.
2. Stworzenie odpowiedniego modelu, polegające na obliczeniu jego parametrów.
3. Porównanie próbek ze zbioru do obliczonego modelu oraz określenie ile próbek do niego pasuje, z określoną tolerancją ε.
4. Jeżeli liczba próbek pasujących do modelu przekroczy ustalony próg τ, przyjmuje się, że model jest poprawny i realizuje się kolejny krok metody. W przeciwnym przypadku, punkty 1 – 4 są powtarzane maksymalnie N razy, a najlepszy z dotychczasowych wyników jest zapisywany.
5. Estymacja parametrów modelu na podstawie wszystkich poprawnych próbek.

Estymacja parametrów płaszczyzny podłoża

Założono, że podłoże jest płaskie i może być modelowane w postaci płaszczyzny o równaniu ogólnym

$$Ax + By + Cz + D = 0. \tag{2.21}$$

Kąt pochylenia sensora α wyznaczono jako kąt między wektorami $\vec{n} = [A, B, C]$ i $\vec{m} = [A, B, 0]$

$$\alpha = \arccos\left(\frac{\vec{n} \circ \vec{m}}{|\vec{n}| \cdot |\vec{m}|}\right). \tag{2.22}$$

Wysokość położenia środka optycznego sensora głębi określa zależność

$$h = \frac{|D|}{\sqrt{A^2 + B^2 + C^2}}. \tag{2.23}$$

Estymowane wartości parametrów mogą być wykorzystane do kompensacji kąta pochylenia sensora oraz usuwania podłoża z pomiarów.

2.8.1 Analiza wyników badań eksperymentalnych

Przeprowadzono badania eksperymentalne metody kalibracji wysokości i pochylenia sensora głębi. Metoda wykrywa podłoże na mapie głębi i estymuje parametry płaszczyzny, co umożliwia obliczenie zarówno wysokości jak i kąta pochylenia sensora względem podłoża. Do badań wykorzystano sensor Microsoft Kinect, który w trakcie wykonywania pomiarów umieszczany był na różnych wysokościach. Pochyleniem sensora Kinect sterowano z wykorzystaniem oprogramowania *kinect_aux*. Pakiet ten wykorzystano również do odczytu kąta pochylenia obliczonego na podstawie pomiarów akcelerometru będącego częścią urządzenia.

Przeprowadzone badania eksperymentalne pozwoliły potwierdzić poprawność działania metody kalibracji wysokości i pochylenia sensora Kinect. W trakcie testów zaobserwowano, że dokładność parametrów jest zależna od rzeczywistej wysokości na jakiej znajduje się sensor. Jest to konsekwencją tego, że wyżej umieszczony czujnik mierzy większe odległości do podłoża, zgodnie z zależnością (2.18). Przy wyżej umieszczonym czujniku błąd podawanych parametrów w większym stopniu wpływa na różnicę między rzeczywistą odległością do podłoża, a wartością przybliżoną. W takich przypadkach umożliwienie działania metody wymaga zwiększenia tolerancji δ odpowiadającej za kwalifikowanie punktów do podłoża. Mniejsza dokładność metody w przypadku, gdy czujnik głębi umieszczony jest w większej odległości od podłoża jest też związana z nieliniową charakterystyką sensora głębi, powodującą malejącą z odległością rozdzielczość pomiarów.

Do określenia dokładności uzyskiwanej wysokości sensora, jako wartość referencyjną przyjęto rezultaty pomiaru wysokości dalmierzem laserowym o dokładności $\pm 1,5$ mm, względem podłoża. Maksymalny błąd wyznaczania wysokości środka optycznego czujnika głębi wynosił $1,3$ cm. Błąd ten ma charakter systematyczny i przypuszcza się, że jednym z jego źródeł jest niedokładność wyznaczania wysokości środka optycznego czujnika głębi. W celu wyznaczenia błędu pochylenia, jako dane referencyjne wykorzystano pomiary uzyskiwane z akcelerometru umieszczonego w czujniku i oszacowano błąd w granicach $4,7^\circ$.

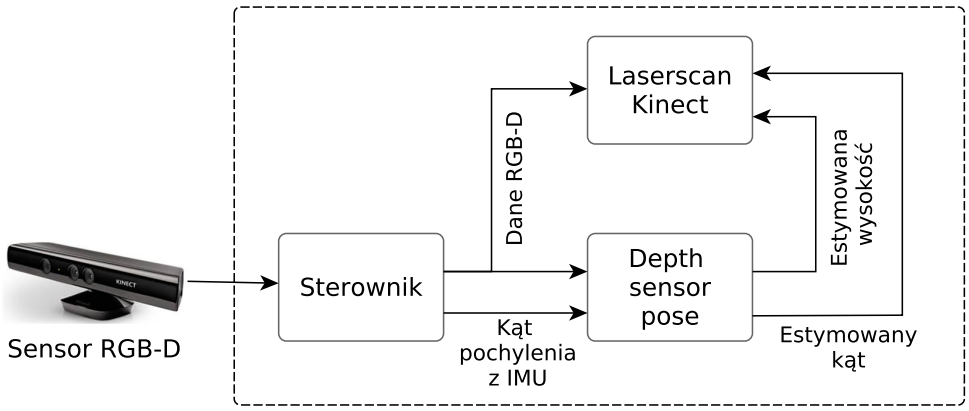
W celu zmniejszenia wpływu błędów systematycznych na wyniki metody kalibracji zastosowano korektę, a wyniki umieszczono w tabeli 2.2. Przez h oraz α oznaczono odpowiednio estymowaną wysokość środka optycznego oraz estymowany kąt pochylenia sensora. Opis metody kalibracji oraz wyniki badań zostały wcześniej opublikowane w pracy [52].

Tabela. 2.2 Korekcja błędów systematycznych

	przed korekcją		**po korekcji**	
	h **[cm]**	α **[°]**	h **[cm]**	α **[°]**
średni błąd	0,54	4,02	0,00	0,00
odchylenie standardowe	0,39	0,42	0,39	0,42

Wybranie odpowiedniego położenia i orientacji czujnika zależy między innymi od geometrii przeszkód i ich odległości od sensora. W przypadku obiektów o znacznej wysokości, korzystnie jest umieścić czujnik wysoko, zapewniając taki kąt pochylenia, aby górna martwa strefa zaczynała się powyżej wysokości robota. Wadą takiego podejścia jest znaczący dolny obszar, będący poza zasięgiem widzenia czujnika. Wykrywanie niewielkich przeszkód, można zrealizować na dwa sposoby. Pierwszy polega na umieszczeniu sensora nisko, z nieznacznym pochyleniem, aby uzyskać duży zasięg wykrywania przeszkód, ograniczony wyłącznie zasięgiem czujnika. Drugie rozwiązanie bazuje na na umieszczeniu czujnika wysoko, ale z dużym pochyleniem. Przy odpowiedniej konfiguracji martwa strefa dla obiektów o niewielkich rozmiarach może zostać niemal całkowicie zredukowana, ale wiąże się to z ograniczeniem zasięgu wykrywania przeszkód.

Na rys. 2.22 przedstawiono strukturę połączeń omawianych narzędzi wykorzystujących dane z czujnika Microsoft Kinect. W tym przypadku przybliżony kąt pochyle-



Rysunek 2.22 Architektura rozwiązania – połączenia między węzłami systemu

nia czujnika otrzymywany jest bezpośrednio z urządzenia wyposażonego w akcelerometr. Metody zostały opracowane w celu nawigacji kołowymi robotami mobilnymi w pomieszczeniach. Węzeł (*laserscan_kinect*) umożliwia konwersję obrazu głębi do dwuwymiarowej postaci, przy jednoczesnej kompensacji pochylenia sensora i usuwaniu podłoża. Drugi pakiet (*depth_sensor_pose*) umożliwia estymację kąta pochylenia oraz wysokości sensora głębi. Parametry wyznaczane są korzystając z wykrywanej na obrazie głębi płaszczyzny podłoża. Przedstawione rozwiązania zostały zaimplementowane i umieszczone w stworzonym pakiecie środowiska ROS *depth_nav_tools* udostępnionym jako oprogramowanie *open-source* [46].

Rozdział 3

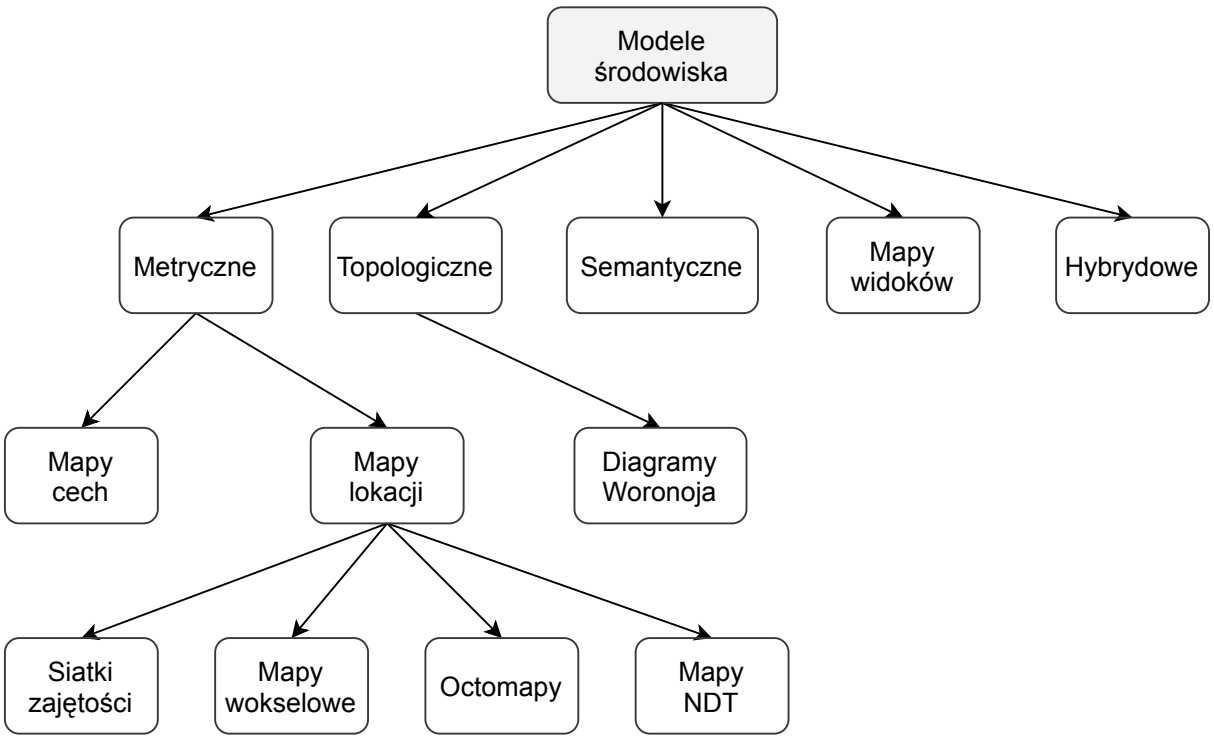
Metody reprezentacji środowiska

Niniejszy rozdział zawiera przegląd sposobów reprezentacji środowiska, w którym poruszają się roboty. Przeanalizowano korzyści związane z wybranymi metodami reprezentacji, a także porównano reprezentację w dwóch i trzech wymiarach.

3.1 Wprowadzenie

Proces mapowania polega na modelowaniu środowiska, w celu umożliwienia robotom planowania swoich działań i poruszania się w otoczeniu. Mapa jest określoną reprezentacją środowiska, odzwierciedlającą jego charakterystyczne punkty, cechy lub inne własności. W literaturze jak i w rozwiązaniach praktycznych spotykanych jest wiele różnych reprezentacji (rys. 3.1). Jeden z podziałów metod reprezentacji dzieli je na rozwiązania metryczne oraz topologiczne [35, 239, 259].

Mapy metryczne opisują geometryczne własności środowiska, przy czym wyróżnia



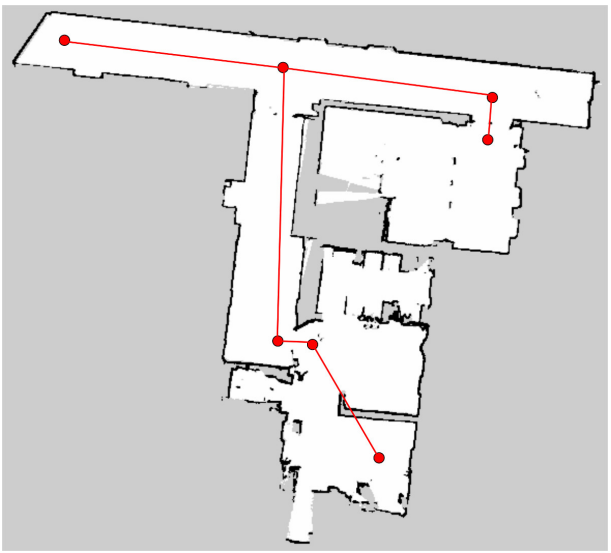
Rysunek 3.1 Klasyfikacja sposobów reprezentacji środowiska

się dwa główne typy map metrycznych: mapy lokacji oraz mapy cech. W przypadku tych pierwszych środowisko modelowane jest przez opis jego poszczególnych fragmentów, a każdy fragment ma przyporządkowane miejsce w przestrzeni. Mapy cech natomiast przechowują jedynie zbiór wybranych cech środowiska, zorientowanych w przestrzeni.

Modele topologiczne opierają się własnościach topologicznych środowiska, czyli relacjach między wybranymi cechami, czy obszarami środowiska. Zwykle realizowane są w postaci grafów odzwierciedlających wspomniane zależności.

Wykorzystuje się również reprezentacje, które trudno zaklasyfikować do jednej z wymienionych grup. Są to między innymi mapy semantyczne, czy mapy widoków (ang. *appearance maps*). Te ostatnie zbudowane są na bazie ważonego grafu i wielu obrazów środowiska utworzonych z wykorzystaniem różnych perspektyw.

W praktyce, przeważnie wykorzystuje się reprezentacje hybrydowe będące kombinacją wymienionych wcześniej rozwiązań [244]. Mapy hybrydowe (rys. 3.2) są szczególnie przydatne podczas poruszania się w skomplikowanych środowiskach, gdzie część topologiczną, czyli tę na wyższym poziomie abstrakcji można używać do planowania działań robotów, a mapy metryczne stosować chociażby do lokalizacji. Przykładem mo-



Rysunek 3.2 Przykład mapy hybrydowej, składającej się z warstwy metrycznej oraz topologicznej (w postaci grafu)

że być robot, który porusza się po wielopiętrowym budynku. Oprócz potrzeby lokalizacji i planowania ruchu na poszczególnych piętrach musi on posiadać również wyższą warstwę abstrakcji, aby przemieszczać się windą, czy być w stanie przejeżdżać przez drzwi. Do planowania takich działań z powodzeniem wykorzystuje się warstwy topologiczne.

Reprezentacja 2D vs 3D

Mapy 3D stają się coraz bardziej powszechne, mimo tego, że ich przetwarzanie jest znacznie mniej efektywne obliczeniowo, a przechowywanie wymaga większej ilości pamięci. Jest to związane z ciągłym rozwojem i większą dostępnością jednostek obliczeniowych zdolnych do przetwarzania takich ilości danych, ale także z pojawiającymi się coraz bardziej wymagającymi zastosowaniami. Przykładem może być poruszanie się robotów latających, które mogą planować ruch efektywniej, wykorzystując mapy 3D.

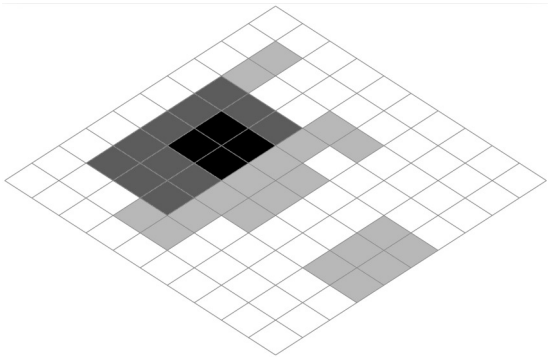
Trójwymiarowe mapy posiadają pewne zalety względem map 2D - lepiej sprawdzają się w przypadku nieregularnego terenu, w budynkach wielopoziomowych, czy kiedy w

środowisku, w którym porusza się robot znajduje się wiele przeszkód o nieregularnych kształtach. Dodatkowo, mapy 3D lepiej sprawdzają się w grupach robotów heterogenicznych, składających się z robotów jeżdżących, kroczących, czy latających. Natomiast, nie zawsze muszą to być pełne mapy 3D. Przykładowo, w przypadku poruszania się robotów w budynkach wielopiętrowych dobrze sprawdzają się mapy składające się z wielu warstw map 2D i odpowiedniego systemu przełączania się między mapami w zależności od wysokości na jakiej znajduje się robot.

3.2 Reprezentacje metryczne

3.2.1 Siatki zajętości 2D

Popularną, metryczną reprezentacją 2D, wykorzystywaną przede wszystkim w algorytmach lokalizacji i SLAM są siatki zajętości (rys. 3.3, 3.4). Siatki zajętości zwane również mapami gridowymi lub rastrowymi zdefiniowane są jako regularne siatki o określonym rozmiarze, gdzie każda komórka przechowuje wartość prawdopodobieństwa jej zajętości [35, 170]. Wartość prawdopodobieństwa zajętości określa jakie jest prawdopodobieństwo wystąpienia przeszkody w danym wycinku przestrzeni.

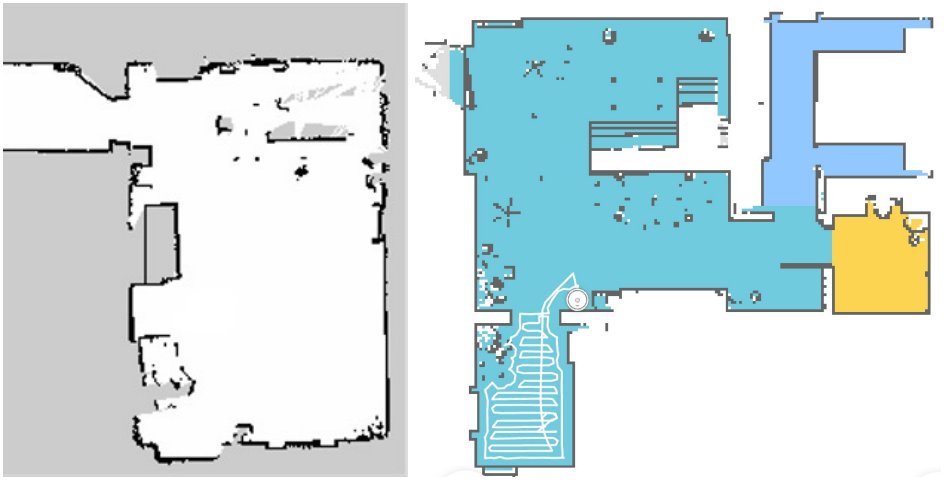


Rysunek 3.3 Siatka zajętości 2D. Ciemniejszy kolor oznacza większe prawdopodobieństwo zajętości danej komórki.

Każda komórka reprezentuje wycinek środowiska, a modelowanie zajętości opiera się na zmiennej losowej. Powodem szerokiego wykorzystania map tego typu jest ich uniwersalność osiągana przez brak definicji konkretnych cech i obiektów środowiska. Mapy gridowe umożliwiają modelowanie dynamicznych środowisk, w których poruszają się inne roboty lub ludzie. Istnieje też możliwość oznaczania niezbadanych dotąd obszarów przez określanie prawdopodobieństwa zajętości równego $p(n) = 0,5$. Mapy gridowe posiadają jednak znaczne wymagania pamięciowe z powodu wykorzystania stałej struktury. Mapę można zdefiniować jako zbiór komórek $m = \{m_1, m_2 \dots m_N\}$. Zakładając, że $z_{1:t}$ określa odczyt z sensorów w pozycji $x_{1:t}$ robota oraz, że komórki są niezależne, prawdopodobieństwo a posteriori mapy określa się zależnością

$$p(m \mid x_{1:t,z_{1:t}}) = \prod_{n=1}^{N} p(m_n \mid x_{1:t}, z_{1:t}). \tag{3.1}$$

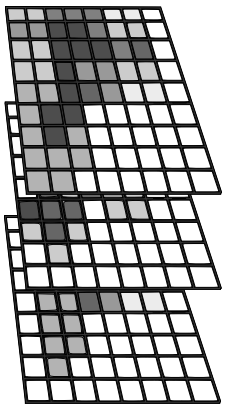
Złożoność pamięciowa map gridowych wynosi $\mathcal{O}(n)$, a czas dostępu do poszczególnych elementów jest stały $\mathcal{O}(1)$.



Rysunek 3.4 Przykładowe mapy oparte na siatce zajętości 2D: utworzona przez algorytm gmapping (po lewej) oraz mapa wykorzystywana przez autonomiczny odkurzacz (po prawej)

3.2.2 Mapy warstwowe 2D

W pracy [151] przedstawiono mapy warstwowe, czyli rozwiązanie składające się z wielu siatek zajętości (rys. 3.5). Pozwala to uzyskać nowe możliwości, szczególnie przy planowaniu działań robotów. Przykładowo, na warstwie proksemicznej umieszcza się informacje o osobach wykrytych w otoczeniu robota, co pozwala planować na tej podstawie zachowania społeczne. Takim zachowaniem może być omijanie osób z zachowaniem większego odstępu niż w przypadku pozostałych przeszkód, co zwiększa komfort osób poruszających się w środowisku wraz z robotami. Kolejnym zastosowaniem są specjalne warstwy bezpieczeństwa, na których umieszcza się strefy bezpieczeństwa, w których ruch robota jest niedozwolony. Strefy bezpieczeństwa mogą być zarówno dodawane do mapy ręcznie przez operatora, ale także wykrywane przez robota i dodawane automatycznie.



Rysunek 3.5 Mapy warstwowe 2D

Mapy warstwowe 2D znajdują zastosowanie przede wszystkim jako lokalne mapy kosztów. W takim przypadku, dodatkowe informacje umieszcza się tylko dla najbliższego otoczenia robota, na podstawie odczytów z sensorów. Na potrzeby map lokalnych wykorzystuje się miedzy innymi dane z następujących sensorów: lidary, sonary, kamery 3D, czy czujniki RGB-D.

3.2.3 Mapy wysokości (2.5D)

Mapy wysokości (ang. *elevation maps*) nazywane też mapami 2.5D, uzyskuje się przez rzutowanie danych 3D na płaszczyznę, a właściwie na siatkę 2D. Wtedy każda komórka siatki o współrzędnych (x_i, y_i) zawiera informację o wysokości przeszkody h w danym wycinku obszaru [35, 175]. Mapy tego typu są wykorzystywane, gdy roboty poruszają się na zewnątrz, ponieważ pozwalają przechować dodatkową informację o wysokości przeszkód, względem siatek zajętości, natomiast nakład związany z przetwarzaniem, czy przechowywaniem danych jest niewielki. Przykładowo, mapy 2.5D zostały wykorzystane podczas misji marsjańskich przeprowadzonych przez NASA [157].

Wada takiego podejścia wiąże się z brakiem pełnej reprezentacji trzeciego wymiaru środowiska, przez co mogą mapy te mogą być stosowane na potrzeby planowania ruchu, ale nie na potrzeby lokalizacji. Pewne rozwiązanie tego problemu zaprezentowano w pracy [213]. Autorzy reprezentowali trzeci wymiar środowiska w postaci list zajętych przedziałów przyporządkowanych do każdej komórki mapy.

3.2.4 Mapy wielopoziomowe

Mapy wielopoziomowe składają się z wielu warstw siatek zajętości 2D. Jednak w przeciwieństwie do map warstwowych, każda warstwa modeluje środowisko na pewnej wysokości. Mapy tego typu umożliwiają poruszanie się po wielopoziomowych budynkach. Jednym z rozwiązań jest wykorzystanie dodatkowego systemu pozwalającego przełączać mapy w zależności od tego na którym piętrze znajduje się robot.

Z punktu widzenia implementacji, mapy wielopoziomowe nie różnią się znacząco od przedstawionych wcześniej map warstwowych 2D. Różnica polega na typie przechowywanych informacji. Mapy warstwowe przechowują różne informacje o tym samym poziomie, natomiast mapy wielopoziomowe przechowują prawdopodobieństwo zajętości dla płaszczyzn równoległych do podłoża i znajdujących się w ustalonej odległości od niego.

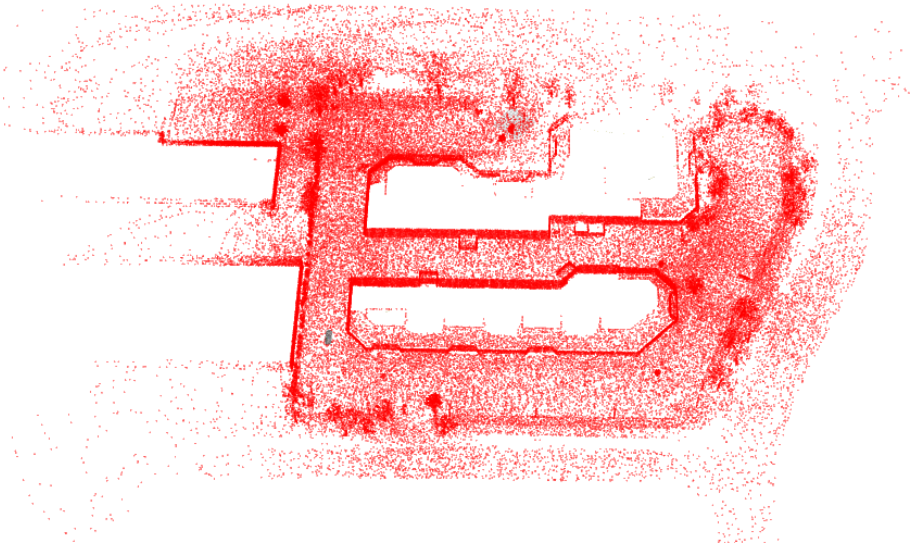
3.2.5 Chmury punktów

W modelowaniu świata w trzech wymiarach wykorzystywane są również zbiory punktów nazywane też chmurami punktów (rys. 3.6). Są to nieuporządkowane kolekcje punktów określone w pewnym układzie współrzędnych, które można zdefiniować jako zbiór n punktów

$$P = \{p_1, p_2, \ldots, p_i, \ldots, p_n\}, \tag{3.2}$$

gdzie $p_i = (x_i, y_i, z_i) \in \mathbb{R}^3$. Dotyczy to jedynie podstawowej wersji chmur punktów, natomiast modyfikacje mogą zawierać dodatkowe informacje, na przykład w postaci koloru punktów.

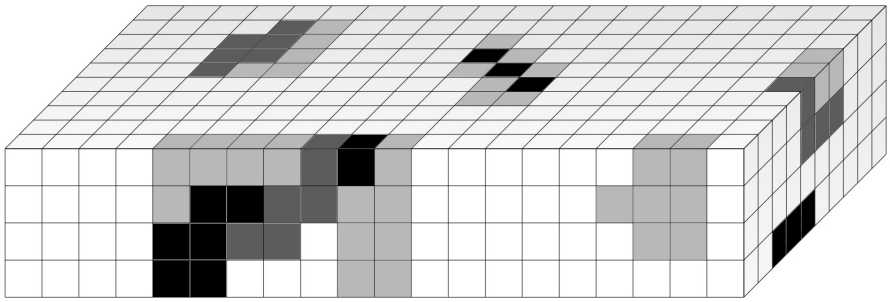
Reprezentacja ta dobrze sprawdza się przy modelowaniu powierzchni obiektów, a także pozwala na łatwą transformację zbiorów. Złożoność pamięciowa chmur punktów wynosi $\mathcal{O}(n)$. Wadą jest natomiast kosztowny czasowo dostęp do poszczególnych elementów. Losowy dostęp wymaga przeszukania całego zbioru, dlatego jego złożoność to $\mathcal{O}(n)$. W praktyce, jeżeli jest potrzeba przeszukiwania zbioru zwykle tworzy się drzewo KD na podstawie chmury punktów, w którym dostęp do elementu w średnim przypadku wynosi $\mathcal{O}(\log n)$.



Rysunek 3.6 Przykład mapy w postaci chmury punktów pochodzącej z systemu Autoware Auto [71]

3.2.6 Siatki zajętości 3D

Woksel jest trójwymiarowym odpowiednikiem piksela, czyli ma formę sześcianu o określonych rozmiarach. Siatki zajętości 3D (rys. 3.7), zwane też mapami wokselowymi składają się z uporządkowanego zbioru wokseli [204], przy czym, analogicznie do siatek 2D, każdy woksel przechowuje informację o zajętości odpowiadającego mu wycinka przestrzeni [35]. Analogicznie jak w przypadku siatek 2D, mapę można zdefiniować



Rysunek 3.7 Przykładowa siatka zajętości 3D (ciemniejszy kolor oznacza większe prawdopodobieństwo zajętości danej komórki)

jako zbiór N wokseli $m = \{m_1, m_2 \dots m_N\}$. Zakładając, że $z_{1:t}$ definiuje odczyt z sensorów w pozycji $x_{1:t}$ robota oraz, że woksele są niezależne, funkcję prawdopodobieństwa a posteriori mapy określa się następująco

$$p(m \mid x_{1:t,z_{1:t}}) = \prod_{n=1}^{N} p(m_n \mid x_{1:t}, z_{1:t}). \tag{3.3}$$

Obszar o rozmiarach $x \times y \times z$ może być reprezentowany przez

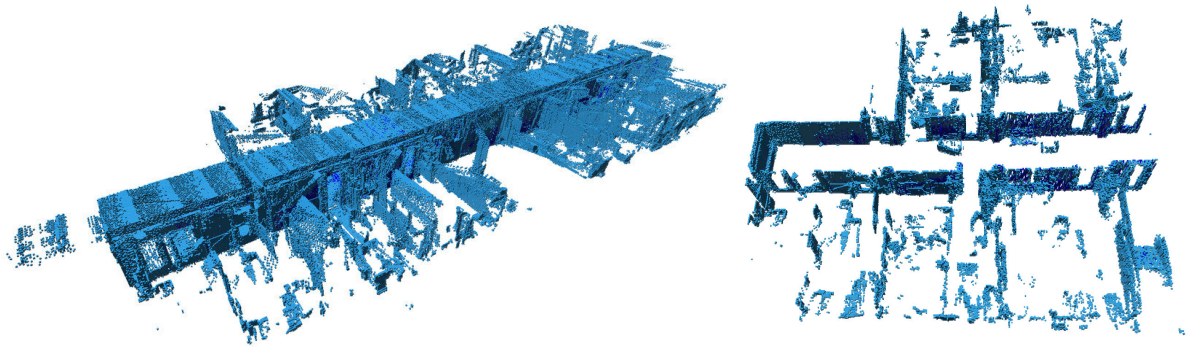
$$N = \frac{x \cdot y \cdot z}{r^3} \tag{3.4}$$

wokseli, gdzie r oznacza rozdzielczość mapy (rozmiar wokseli).

Zaletą takiej reprezentacji jest stały czas dostępu $\mathcal{O}(1)$, natomiast wadą zużycie pamięci, ponieważ wszystkie elementy są inicjalizowane od razu. Dlatego też siatki zajętości 3D znajdują przede wszystkim zastosowanie w mapach lokalnych, których układ współrzędnych powiązany jest z robotem i porusza się razem z nim. Pozwala to zachować niewielkie rozmiary mapy, na przykład na potrzeby unikania kolizji, lub lokalnego planowania ruchu. Siatki 3D znacznie rzadziej są wykorzystywane do reprezentacji dużych obszarów.

3.2.7 Octomapy

Popularną reprezentacją otoczenia robota w trzech wymiarach są octomapy (rys. 3.8), czyli struktury oparte na drzewach ósemkowych, przedstawionych po raz pierwszy w pracy [163], a następnie rozwijanych w ramach prac [98, 99, 187, 254, 263].



Rysunek 3.8 Octomapy wygenerowane na podstawie zbioru danych [97]

Kluczowym pomysłem, na którym opierają się octomapy jest rekursywny podział przestrzeni na osiem równych, sześciennych części (rys. 3.9). Każdy węzeł drzewa reprezentuje wycinek przestrzeni na zadanym poziomie szczegółowości i określa prawdopodobieństwo zajętości tego wycinka. Znaczy to tyle, że im głębiej w drzewie znajduje się węzeł, tym mniejszy wycinek przestrzeni reprezentuje, a tym samym wyższy jest poziom szczegółowości.

Złożoność czasowa losowego dostępu do elementu w przypadku drzewa o głębokości d, zawierającego n węzłów, wynosi $\mathcal{O}(d) = \mathcal{O}(\log n)$. Jednak w praktyce, octomapy implementowane są w oparciu o drzewa posiadające stałą głębokość. W takim przypadku można założyć, że czas dostępu do poszczególnych węzłów drzewa jest również stały $\mathcal{O}(1)$.

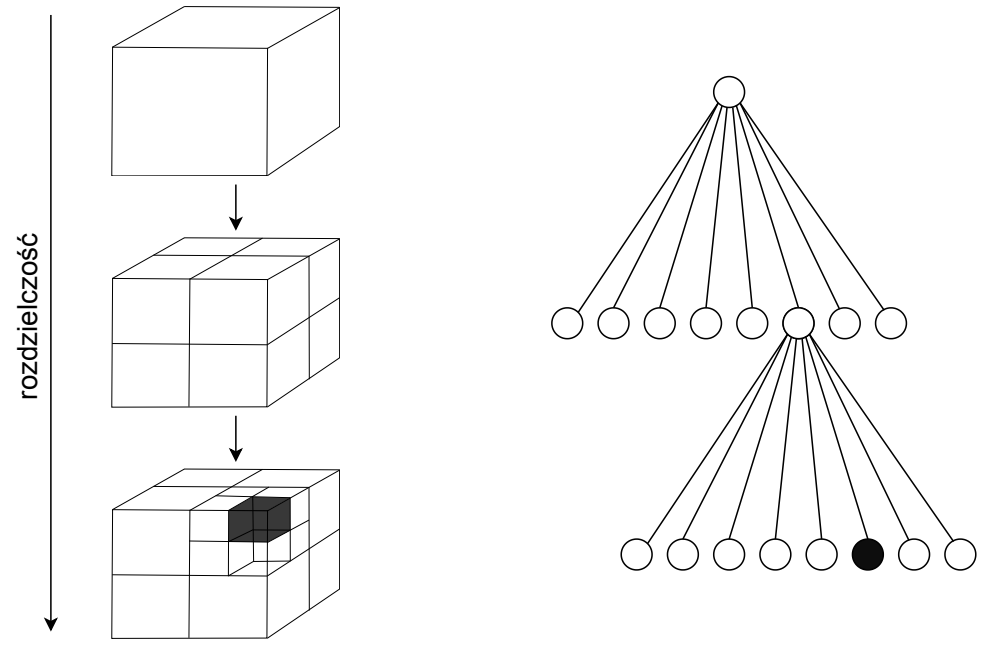
Drzewo o głębokości d może posiadać maksymalnie

$$N_L = 8^d \tag{3.5}$$

węzłów końcowych, nazywanych liśćmi, a maksymalna liczba wszystkich węzłów drzewa wynosi

$$N = \sum_{i=0}^{d} = \frac{8^{d+1} - 1}{7}. \tag{3.6}$$

Jedną z kluczowych cech reprezentacji opartej na drzewach ósemkowych jest możliwość optymalizacji wykorzystania pamięci. W zależności od wybranego rozwiązania,



Rysunek 3.9 Rekursywny podział przestrzeni na 8 równych części, na którym opierają się octomapy

wycinki przestrzeni skojarzone z węzłami drzewa mogą być dzielone tak długo, jak elementy powstałe z podziału różnią się od siebie. Jeżeli otrzymane bloki miałyby być takie same, nie ma potrzeby wykonywania podziału, ponieważ nie otrzymujemy nowej informacji. W tym przypadku dokonuje się odcięcia gałęzi drzewa, a węzeł reprezentuje spójną część przestrzeni. Takie rozwiązanie wymaga zdefiniowania progów wartości prawdopodobieństwa powyżej którego węzeł uważany jest za zajęty.

Drugie podejście opiera się na rozwijaniu gałęzi drzewa, które posiadają węzły zajęte lub wolne do maksymalnej głębokości. W tym przypadku optymalizacja polega na odcinaniu gałęzi, które reprezentują nieodkrytą część środowiska.

Octomapy pozwalają uniknąć jednej z większych wad stałych struktur do modelowania środowiska, czyli potrzeby ich inicjalizacji. W przypadku octomap inicjalizację danego obszaru można opóźnić do momentu, gdy robot uzyska dane o tym obszarze, czyli wykona odpowiedni pomiar. Modyfikacje octomap wykorzystują dodatkowe informacje w węzłach, jak na przykład kolor, lub etykiety.

Octomapę definiuje się jako zbiór N węzłów, które reprezentują fragmenty przestrzeni i zawierają wartości prawdopodobieństw zajętości

$$m = \{n_1, n_2 \dots n_N\}. \tag{3.7}$$

Wartości prawdopodobieństw zajętości węzłów są aktualizowane na podstawie zależności [99, 187]

$$p(n \mid z_{1:t}) = \left[1 + \frac{1 - p(n \mid z_t)}{p(n \mid z_t)} \frac{1 - p(n \mid z_{1:t-1})}{p(n \mid z_{1:t-1})} \frac{p(n)}{1 - p(n)}\right]^{-1}, \tag{3.8}$$

gdzie:

- z_t – określa obecny pomiar,
- $p(n)$ – prawdopodobieństwo a priori,

- $p(n \mid z_{1:t-1})$ – poprzednia estymacja prawdopodobieństwa zajętości,
- $p(n \mid z_t)$ – prawdopodobieństwo zajętości przy pomiarze z_t obliczonym na podstawie modelu sensora.

Zakładając, że prawdopodobieństwo a priori $p(n) = 0.5$ oraz, że

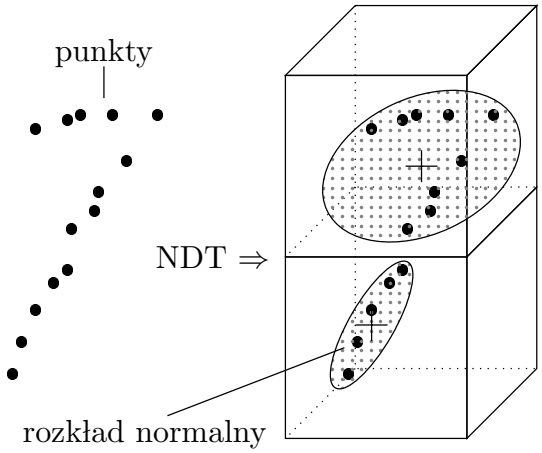
$$L(n) = \log \frac{p(n)}{1 - p(n)}, \tag{3.9}$$

aktualizacji wartości węzła można dokonać na podstawie notacji logarytmicznej

$$L(n \mid z_{1:t}) = L(n \mid z_{1:t-1}) + L(n \mid z_t). \tag{3.10}$$

3.2.8 Mapy NDT

Mapy NDT opierają się na transformacji rozkładów normalnych (NDT, ang. *Normal Distributions Transform*) zaprezentowanej w pracach [22, 235]. Idea NDT jest związana z reprezentacją sceny robotycznej w postaci zbioru lokalnych rozkładów normalnych (rys. 3.10).



Rysunek 3.10 Transformacja punktów do rozkładów normalnych w przypadku trójwymiarowych map NDT

Proces tworzenia map jest następujący, przy czym przyjęto, że mapa tworzona jest na podstawie danych w postaci zbioru punktów $X = \{x_1, x_2, \ldots, x_i, \ldots, x_n\}$, gdzie $x_i \in \mathbb{R}^3$, a n określa liczbę punktów. Podobnie jak w przypadku map zajętości, przestrzeń dzieli się na komórki (lub woksele w przypadku 3D) o ustalonym rozmiarze. Dla każdej komórki oblicza się wartość średnią (rys. 3.11)

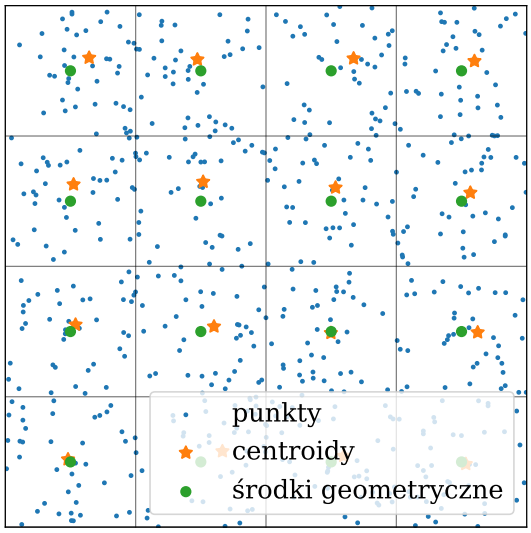
$$q = \frac{1}{n} \sum_i x_i \tag{3.11}$$

oraz macierz kowariancji

$$\Sigma = \frac{1}{n} \sum_i (x_i - q)(x_i - q)^T. \tag{3.12}$$

Prawdopodobieństwo otrzymania z pomiaru próbki w punkcie x należącym do danej komórki modelowane jest przez rozkład normalny $N(q, \Sigma)$ i wynosi

$$p(x) \sim \exp\left(-\frac{(x-q)^T\Sigma^{-1}(x-q)}{2}\right). \tag{3.13}$$



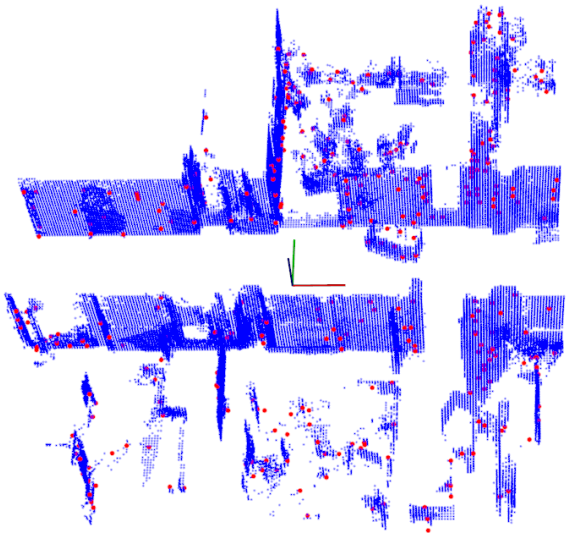
Rysunek 3.11 Przykład podziału przestrzeni na komórki i obliczenia centroidów

Jak już wspomniano, podobnie jak w przypadku siatek zajętości, przestrzeń dzielona jest na komórki o ustalonym rozmiarze. Różnicą jest to, że dla siatek zajętości określa się prawdopodobieństwo zajętości danej komórki, a w przypadku map NDT określa się prawdopodobieństwo otrzymania w pomiarze punktu dla każdej pozycji wewnątrz komórki.

Istnieje zarówno wersja 2D jak i 3D map NDT, a ich budowa jest analogiczna. Mapy NDT 3D wykorzystywane są z powodzeniem w lokalizacji i metodach SLAM. W pracy [215] przedstawiono rozwiązanie wykorzystujące hybrydowe mapy 3D (NDT-OM, ang. *Normal Distribution Transform - Occupancy Map*), wykorzystujące zarówno NDT jak i siatki zajętości.

3.2.9 Mapy cech

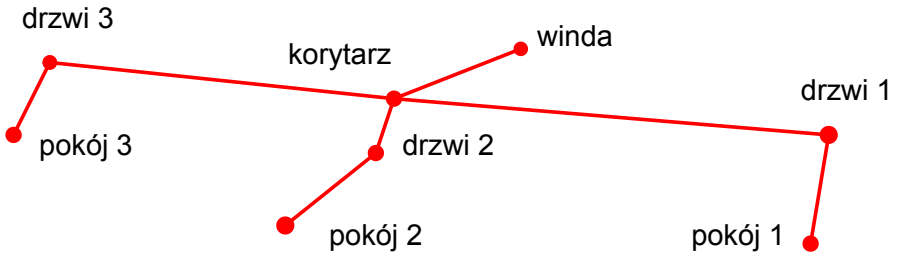
Mapy cech, inaczej zwane też mapami znaczników opierają się na zbiorze cech wyodrębnionych ze środowiska (rys. 3.12). Z każdą cechą skojarzona jest jej pozycja oraz uniwersalny identyfikator. Wykorzystywane są różnego rodzaju detektory i deskryptory cech. W przypadku reprezentacji 2D do często stosowanych detektorów należą: metoda SIFT (ang. *Scale-invariant feature transform*) [232], metoda SURF (ang. *Speeded up robust features*), ale też ORB (ang. *Oriented FAST and Rotated BRIEF*) [207] czy detektory Harrisa [89]. Natomiast detekcja cech w danych 3D, na przykład z sensora RGB-D, opiera się na metodach FPFH [209, 211], SHOT [218], a także na metodzie ISS [265].



Rysunek 3.12 Mapa składająca się z opisanych cech punktowych (deskryptorów) w punktach kluczowych (czerwony)

3.3 Reprezentacje topologiczne

W odróżnieniu od reprezentacji metrycznych, które skupiają się na geometrycznych własnościach środowiska, modelowanie topologiczne opiera się na relacjach między miejscami i obiektami w środowisku (rys. 3.13). Z tego powodu jest to reprezentacja



Rysunek 3.13 Przykładowa mapa topologiczna jednej kondygnacji budynku

łatwa w odbiorze dla ludzi.

Mapa topologiczna może być zdefiniowana w formie grafu

$$G = (V, E), \tag{3.14}$$

którego zbiór węzłów V określa poszczególne miejsca, natomiast krawędzie E odzwierciedlają relacje między nimi.

Ponieważ mapy topologiczne są reprezentacją o wyższym poziomie abstrakcji niż mapy metryczne, to sprawdzają się w wysokopoziomowym planowaniu działań robotów. Jednym z rozwiązań topologicznych są diagramy Voronoi'a [14]. Metoda ta opiera się na minimalnych, bezpiecznych odległościach od przeszkód.

3.4 Mapy semantyczne

Mapy semantyczne stanowią abstrakcyjną reprezentację środowiska. Zawierają one informacje o relacjach i funkcjonalnościach obiektów w środowisku [259]. Ich konstrukcja jest zbliżona do konstrukcji map topologicznych, z tą różnicą, że mapy semantyczne posiadają bardziej rozbudowane informacje odnośnie obiektów będących częścią środowiska, w którym porusza się robot.

Przykładowo, mapa semantyczna budynku może składać się z informacji o obiektach takich jak drzwi, meble, czy ludzie. Każdy obiekt jest zidentyfikowany i opisany. Mapy tego typu dobrze sprawdzają się przy wysokopoziomowym planowaniu działań robotów.

3.5 Mapy widoków

Tworzenie map widoków (ang. *appearance maps*) opiera się na danych z systemów wizyjnych. Mapy reprezentowane są przez graf

$$G = (V, E), \tag{3.15}$$

zawierający obrazy będące widokami na te same obszary środowiska z różnych perspektyw [16, 26, 74] (rys. 3.14). Wierzchołki grafu V zawierają obrazy wykonane przy

Rysunek 3.14 Przykład mapy widoków - krawędzie łączą podobne obrazy [63]

ustalonej pozycji sensora, natomiast krawędzie E łączą ze sobą obrazy o odpowiednim stopniu podobieństwa, przy czym stopień podobieństwa wyrażony jest jako waga krawędzi. Przykładowo, krawędź $e_{ij} \in E$ o wysokiej wartości wagi w_{ij} oznacza, że obrazy i oraz j są do siebie podobne i z dużym prawdopodobieństwem zostały wykonane w zbliżonym obszarze środowiska. Wadą tego rozwiązania są duże wymagania pamięciowe, aby przechować wszystkie obrazy.

3.6 Reprezentacje hybrydowe

W praktyce najwięcej korzyści pojawia się przy stosowaniu rozwiązań opierających się na łączeniu wcześniej przedstawionych metod reprezentacji. Jednym z pomysłów

jest wykorzystanie reprezentacji metryczno-topologicznej [244]. W pracy [132] przedstawiono reprezentację środowiska w postaci grafu pozycji

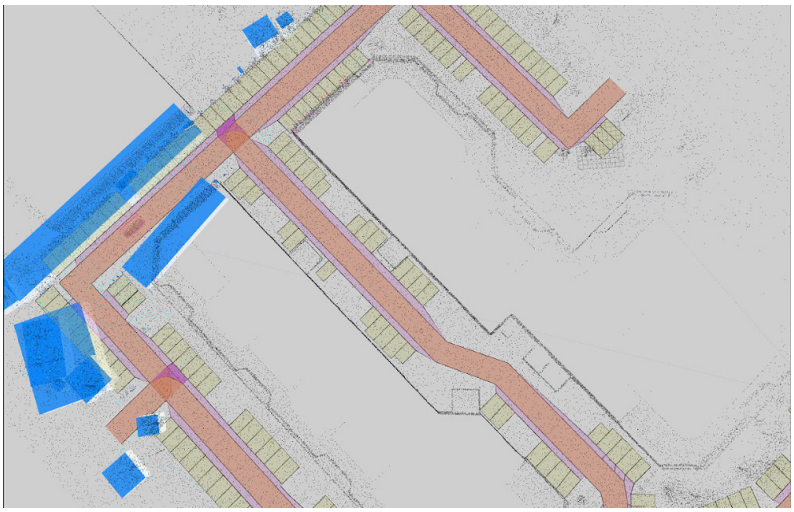
$$G = (N, E, S, C, P), \tag{3.16}$$

gdzie N określa zbiór węzłów, E to zbiór krawędzi definiujących ograniczenia między pozycjami węzłów, a dodatkowo z każdym węzłem $n \in N$ skojarzony jest zbiór danych pomiarowych z sensorów S_n w układzie współrzędnych danego węzła. Poza tym, C określa zbiór ograniczeń reprezentujących transformacje między dwoma układami odniesienia, a P jest zbiorem pozycji skojarzonych z poszczególnymi węzłami, globalnie zoptymalizowanych. Autorzy wykorzystują także drugi graf, w skład którego wchodzi między innymi siatka zajętości. Takie podejście pozwala zarówno na lokalizację robota jak i planowanie jego działań na wyższym poziomie, korzystając z grafu.

Rozwijane są też inne, mniej popularne rozwiązania. W pracy [45] przedstawiono reprezentację 3D SkiMap, która wykorzystuje zarówno strukturę drzewiastą jak i listy z przeskokami. Próby połączenia siatek 3D z octomapami zostały przedstawione w pracy [138] w postaci nowej struktury Jittree.

3.7 Mapy HD

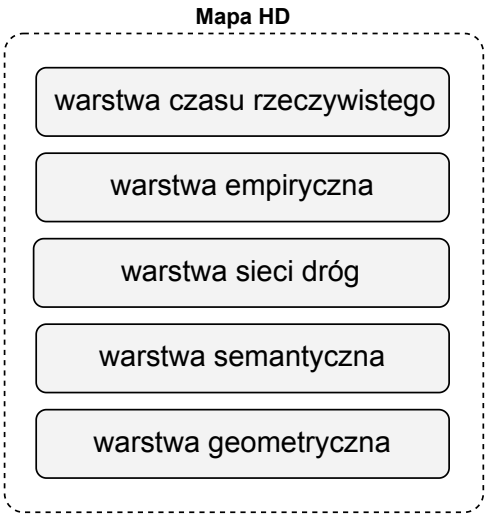
Kompleksowym podejściem jeżeli chodzi o reprezentację środowiska są mapy HD (ang. *High-Definition*) (rys. 3.15), rozwijane głównie na potrzeby przemysłu motoryzacyjnego, a konkretniej na potrzeby samochodów autonomicznych [8,113,185]. Koncep-



Rysunek 3.15 Wizualizacja mapy HD pochodzącej z systemu Autoware Auto [71], na której widać drogi, po których mogą poruszać się pojazdy, a także miejsca postojowe

cja map HD opiera się na wykorzystaniu niezależnych warstw o różnym przeznaczeniu (rys. 3.16) oraz odrębnej budowie. Zawartość i przeznaczenie poszczególnych warstw map są następujące.

Warstwa czasu rzeczywistego zawiera informacje o obecnym stanie otoczenia autonomicznego pojazdu. Jest tworzona na podstawie bieżących odczytów z sensorów oraz danych przesyłanych z innych pojazdów będących w otoczeniu, a także z obiektów infrastruktury. Informacje zawarte w tej warstwie powinny umożliwić autonomicznym pojazdom reagowanie na dynamiczne otoczenie.



Rysunek 3.16 Warstwy wchodzące w skład mapy HD [8]

Warstwa empiryczna przechowuje informacje z poprzednich przejazdów przez dany obszar otoczenia, czyli realizuje pewnego rodzaju bazę wiedzy doświadczalnej. Przykładem informacji zawartych w tej warstwie może być natężenie ruchu, czy zwiększone zagrożenie wypadkami. Zakłada się, że baza wiedzy nie musi pochodzić z danego pojazdu, ale może być gromadzona globalnie i przesyłana do poszczególnych pojazdów, co w rezultacie pozwoli na wykorzystywanie doświadczeń z innych pojazdów autonomicznych.

Warstwa sieci dróg definiuje sieć dróg, czyli określa w jaki sposób poszczególne drogi łączą się ze sobą. Warstwa ta może przyjmować formę grafu. Graf połączeń dróg może być wykorzystany między innymi do planowania ruchu pojazdów.

Warstwa semantyczna nadaje znaczenie obiektom geometrycznym przez przypisanie do nich etykiet. Przykładem mogą być linie jezdni, przejścia dla pieszych, czy znaki drogowe. Zdefiniowanie odpowiednich klas znaków oraz ich znaczenia, a następnie przypisanie tych klas do wykrytych obiektów może pozwolić na implementację odpowiednich zasad poruszania się pojazdów.

Warstwa geometryczna przechowuje pozycje fizycznych obiektów środowiska w postaci punktów, linii, czy wielokątów.

Zakłada się, że dokładność map HD powinna być na poziomie jednego centymetra. Jednym z pomysłów jest też zapewnienie automatycznego generowania wybranych warstw na podstawie obrazów rejestrowanych przez satelity. Rozwiązanie to wydaje się perspektywiczne, z powodu zaangażowania wielu korporacji w ich rozwój, ale także ze względu na otwarty format. Do bardziej znanych implementacji map HD należą te rozwijane przez firmy Lyft, TomTom, Nvidia, czy Here (HD Live Map). W kontekście map przeznaczonych dla pojazdów autonomicznych, warto wspomnieć również o otwartym projekcie OSM (ang. *Open Street Map*), skupiającym się na tworzeniu map, w tym sieci dróg, głównie przez wielomilionową społeczność użytkowników. Mapy OSM są z powodzeniem stosowane jako elementy map HD.

Rozdział 4

Lokalizacja robotów mobilnych

Niniejszy rozdział zawiera przegląd metod i systemów lokalizacji robotów mobilnych. W rozdziale przedstawiono również wyniki przeprowadzonych badań eksperymentalnych dotyczących wybranych metod lokalizacji robotów kołowych.

4.1 Wprowadzenie

Poprawna lokalizacja robotów, czyli zapewnienie im wiedzy, gdzie aktualnie znajdują się w środowisku, jest podstawą większości zadań stawianych autonomicznym robotom mobilnym. Znajomość własnej pozycji względem modelu środowiska pozwala na planowanie ruchu do miejsc docelowych oraz realizację tego ruchu.

Metody lokalizacji można podzielić na dwie grupy: lokalne (względne) i globalne (bezwzględne). W przypadku metod lokalnych, największym problemem jest narastająca z czasem wartość błędu. Z drugiej strony, metody globalne, nie zawsze pozwalają uzyskać zadowalającą dokładność i jednocześnie zapewnić odpowiednio niskie opóźnienia, czyli wysoką częstotliwość aktualizacji pozycji. Przy niskiej częstotliwości aktualizacji, wzrasta błąd między estymowaną i rzeczywistą pozycją. Utrudnia lub uniemożliwia to poprawną realizację wielu zadań, jak na przykład śledzenie trajektorii. W praktyce, wady poszczególnych rozwiązań kompensuje się jednoczesne wykorzystanie zarówno metod lokalnych jak i globalnych [239, 246].

W ogólności, lokalizowanie robota polega na estymacji jego wektora stanu względem wybranego układu odniesienia. Wektor stanu robota w chwili t zawiera położenie oraz orientację robota

$$x_t = [x, y, z, \varphi, \psi, \theta]^T, \quad x_t \in \mathbb{R}^6. \tag{4.1}$$

W przypadku robotów poruszających się po płaszczyźnie, wektor stanu redukuje się do dwóch współrzędnych i jednego kąta orientacji

$$x_t = [x\ y\ \theta]^T, \quad x_t \in \mathbb{R}^3. \tag{4.2}$$

Lokalizacja lokalna

Lokalizacja lokalna (względna) opiera się na pomiarach przemieszczenia i zmian orientacji oraz aktualizowaniu wartości położenia i orientacji robota. Zazwyczaj realizowane jest to z wysoką częstotliwością, aby zapewnić niewielkie opóźnienia. W metodach lokalnych, nazywanych też metodami całkowania ścieżki (ang. *Dead reckoning*) [27]

zakłada się znajomość początkowej konfiguracji robota. Wykorzystuje się następujące techniki lokalizacji lokalnej:

- odometria na podstawie obrotów kół (w przypadku robotów kołowych),
- lokalizacja inercyjna, czyli określanie przemieszczeń i zmian orientacji w oparciu o odczyty z IMU [104, 154, 195].
- odometria wizyjna, lub inne odmiany odometrii działające w oparciu o dane z czujników ultradźwiękowych, skanerów laserowych, czy czujników głębi,
- oraz metody hybrydowe, opierające się na filtracji i fuzji danych z różnych czujników. Powszechnie stosowanym rozwiązaniem jest wykorzystanie filtrów Kalmana do fuzji danych odometrycznych z danymi z IMU.

Lokalizacja globalna

Lokalizacja globalna pozwala określić położenie i orientację robota względem ustalonego, globalnego układu odniesienia. W przypadku metod bezwzględnych zwykle jeden pomiar wystarcza do zlokalizowania robota. Wykorzystywane są następujące techniki lokalizacji globalnej:

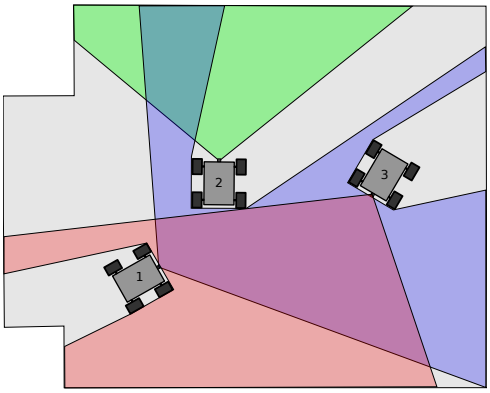
- dopasowanie odczytów z czujników do modelu otoczenia (mapy), metodę określa się też mianem dopasowania skanu (ang. *scan matching*),
- detekcja cech charakterystycznych otoczenia (naturalnych znaczników),
- detekcja sztucznych znaczników,
- metody bazujące na aktywnych nadajnikach, przykładowo systemy pozycjonowania satelitarnego, z których najbardziej rozpowszechniony jest system GPS.

Środowisko statyczne lub dynamiczne

Środowiska, w których poruszają się roboty mogą być statyczne lub dynamiczne. W środowiskach statycznych stan zależy wyłącznie od pozycji robota, czyli zakłada się, że robot jest jedynym poruszającym się obiektem. Założenie to może być spełnione jedynie w dość hermetycznych środowiskach przemysłowych. W praktyce, niemal zawsze mamy do czynienia ze środowiskami dynamicznymi, w których nie tylko robot zmienia swoje położenie, ale także inne obiekty oraz ludzie.

Od tego, czy środowisko jest statyczne, czy dynamiczne, zależy trudność zadania lokalizacji [241]. Z oczywistych względów, lokalizacja robota w środowisku dynamicznym jest większym wyzwaniem. Istnieje kilka powszechnie stosowanych rozwiązań problemu dynamiki otoczenia [239]. Jedno z nich opiera się na próbie modelowania wszystkich obiektów dynamicznych i dołączeniu ich do wektora stanu. Jednak takie podejście nie zawsze może zostać wykorzystane, a szczególną trudność sprawia modelowanie ludzi poruszających się w otoczeniu robotów.

Drugie podejście opiera się na wykrywaniu dynamicznych obiektów i odfiltrowaniu ich z odczytów czujników. Przykładowo, jeżeli jesteśmy w stanie wykryć człowieka w danych ze skanera laserowego, możemy wyciąć odpowiednią część odczytu. W przypadku systemu wielu robotów (rys. 4.1), istnieje możliwość wykrywania pozostałych robotów lub przesyłania między robotami informacji o położeniu w celu usunięcia zakłóceń z danych pomiarowych.



Rysunek 4.1 Pozostałe roboty widoczne w danych pomiarowych

4.2 Robotyka probabilistyczna

4.2.1 Podstawy robotyki probabilistycznej

Robotyka probabilistyczna opiera się na założeniu, że zarówno pozycja robota, jego ruch jak i obserwacje mogą być modelowane z wykorzystaniem zmiennych losowych. Za podstawę robotyki probabilistycznej oraz wnioskowania bayesowskiego uważa się wzór Bayesa (4.3) [40].

Twierdzenie 1 *Zakładając, że x, y oznaczają pewne zdarzenia oraz $p(y) > 0$, a także $p(y \mid x)$ oznacza prawdopodobieństwo warunkowe, czyli prawdopodobieństwo zajścia zdarzenia y pod warunkiem zajścia zdarzenia x, to prawdopodobieństwo przeciwne, czyli zajścia zdarzenia x pod warunkiem zdarzenia y definiuje się następująco*

$$p(x \mid y) = \frac{p(y \mid x)p(x)}{p(y)}. \tag{4.3}$$

Prawdopodobieństwo $p(x)$ określa się też mianem prawdopodobieństwo a priori, natomiast $p(x \mid y)$ nazywa się prawdopodobieństwem a posteriori. Wykorzystane w dalszej części aksjomaty prawdopodobieństwa umieszczono w dodatku B.

Na potrzeby dalszych rozważań związanych z lokalizacją wymagane jest wprowadzenie podstawowych pojęć związanych z modelowaniem robota mobilnego i sceny. Niech x_t oznacza stan robota w chwili t, a historia stanów będzie zdefiniowana następująco

$$x_{1:t} = \{x_1, x_2, \ldots, x_t\}. \tag{4.4}$$

Robot wykonuje pewne obserwacje otoczenia z_t i gromadzi wiedzę w postaci serii pomiarów

$$z_{1:t} = \{z_1, z_2, \ldots, z_t\}. \tag{4.5}$$

Ruch robota realizowany jest w oparciu o wartości sterowań u_t w okresie $(t-1, t]$, przy czym seria sterowań ma postać

$$u_{1:t} = \{u_1, u_2, \ldots, u_t\}. \tag{4.6}$$

Przez

$$p(x_t \mid x_{t-1}, u_t) \tag{4.7}$$

definiuje się probabilistyczny model ruchu robota, czyli generator stanów ($x \leftarrow f(x,u)$). Probabilistyczny model obserwacji, czyli funkcja prawdopodobieństwa pomiarów pod warunkiem stanu robota ma postać

$$p(z_t \mid x_t), \tag{4.8}$$

a początkowa wiedza odnośnie stanu robota określana jest przez funkcję gęstości prawdopodobieństwa $p(x_0)$.

Proces Markowa

Proces stochastyczny spełniający założenia Markowa, jest to proces dla którego warunkowe rozkłady prawdopodobieństwa przyszłych stanów procesu zależą wyłącznie od bieżącego stanu, a nie stanów poprzednich. W robotyce mobilnej nie jest to do końca prawda i założenia te w wielu miejscach są naruszane, co wpływa jakość modelu [241]. Do takich czynników można zaliczyć:

- występowanie obiektów dynamicznych w otoczeniu, które nie są zawarte w wektorze stanu x_t, na przykład inne roboty, poruszający się ludzie, lub przedmioty zmieniające położenie,
- błędy aproksymacji reprezentacji funkcji przekonań za pomocą rastrów lub rozkładami Gaussa,
- niedokładności probabilistycznych modeli sensorów $p(z_t \mid x_t)$,
- niedokładności probabilistycznej funkcji przejścia między stanami $p(x_t \mid x_{t-1}, u_t)$.

Podsumowując, stosując proces Markowa, zakłada się, że świat jest statyczny, zakłócenia mają charakter niezależny, a modele nie posiadają błędów aproksymacji.

Ukryty model Markowa

Ukryte modele Markowa (ang. *Hidden Markov Model, HMM*) pozwalają modelować procesy stochastyczne. Modelowany system składa się z procesu Markowa, którego stany są niewidoczne dla obserwatora, natomiast widoczne jest wyjście systemu w postaci obserwacji. Obserwacje mają postać funkcji losowych zależnych od stanu systemu [94]. W ukrytym modelu Markowa zakłada się, że:

- obserwowana sekwencja zdarzeń jest uporządkowana,
- obserwacje w sekwencji spełniają założenia Markowa, czyli dowolna obserwacja nie zależy od poprzednich obserwacji.

Cechy ukrytego modelu Markowa:

- HMM modeluje proces stochastyczny, którego własności nie są znane,
- HMM jest automatem skończonym (ang. *finite state machine, FSM*), składającym się ze skończonej liczby stanów i przejść między nimi,
- architektura HMM jest określona przez topologię FSM,
- każdemu stanowi w HMM przypisuje się prawdopodobieństwo emisji wartości zmiennych losowych ze skończonego zbioru i prawdopodobieństwo przejść między tymi stanami.

Filtr Bayesa

W systemie, który spełnia założenia Markowa, każdy nowy stan x_t jest generowany stochastycznie ze stanu poprzedniego x_{t-1}. Dlatego probabilistyczne prawo przejścia, opisujące zmianę stanu może być zdefiniowane przez [241]

$$p(x_t \mid x_{0:t-1}, z_{1:t-1}, u_{1:t-1}). \tag{4.9}$$

Na potrzeby dalszych rozważań, zakłada się, że robot najpierw wykonuje sterowanie u_t a następnie pomiar z_t. Dla procesu Markowa, jeżeli stan x_t jest kompletny, można pominąć poprzednie sterowania oraz obserwację, a także wszystkie stany oprócz ostatniego

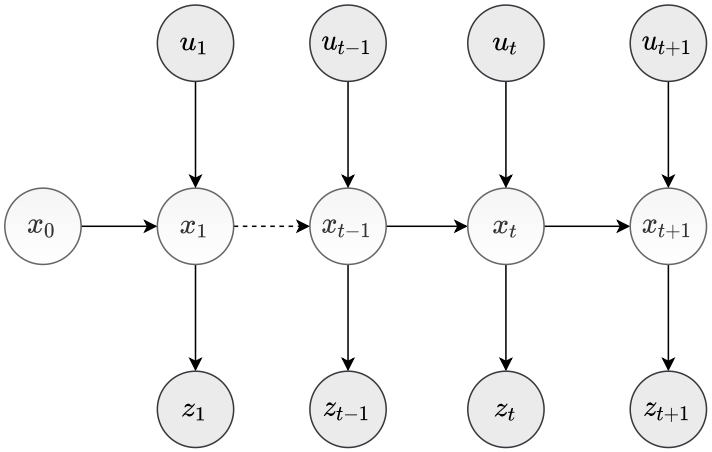
$$p(x_t \mid x_{0:t-1}, z_{1:t-1}, u_{1:t-1}) = p(x_t \mid x_{t-1}, u_t). \tag{4.10}$$

Przedstawiony model przejścia określa prawdopodobieństwo wystąpienia stanu x_t pod warunkiem wystąpienia stanu x_{t-1} w chwili poprzedniej i pod warunkiem zastosowania sterowania u_t.

Zakładając niezależność zmiennych warunkowych oraz kompletność stanu x_t, można również zredukować model pomiaru

$$p(z_t \mid x_{0:t}, z_1 : t-1, u_{1:t}) = p(z_t \mid x_t). \tag{4.11}$$

Na rys. 4.2 przedstawiono dynamiczną sieć Bayesa, która określa zmianę stanów w systemie, a także relacje między poszczególnymi zmiennymi losowymi.



Rysunek 4.2 Dynamiczna sieć Bayesa, określająca zmianę stanów

Wewnętrzną wiedzę robota odnośnie swojego stanu, ale również odnośnie stanu środowiska można wyrazić w postaci przekonań (ang. *belief*) [241]

$$bel(x_t) = p(x_t \mid z_{1:t}, u_{1:t}). \tag{4.12}$$

Przekonania reprezentowane są przez dystrybucje prawdopodobieństwa warunkowego i przypisują prawdopodobieństwo każdej możliwej hipotezie dotyczącej stanu prawdziwego robota i otoczenia. Innymi słowy, przekonanie wyraża rozkład prawdopodobieństwa a posteriori odnośnie stanu robota i środowiska.

Filtr Bayesa pozwala obliczyć rekursywnie przekonanie $bel(x_t)$ w chwili t w oparciu o przekonanie $bel(x_{t-1})$ w chwili $t-1$ oraz na podstawie sterowania u_t oraz obserwacji z_t w chwili t. Działanie opiera się na dwóch operacjach. Pierwszą z nich jest predykcja w oparciu o funkcję zmiany stanu (4.10), natomiast drugą korekcja bazująca na wykonanym pomiarze i przyjętym modelu (4.11). Algorytm można opisać następująco:

- dla każdego x_t :

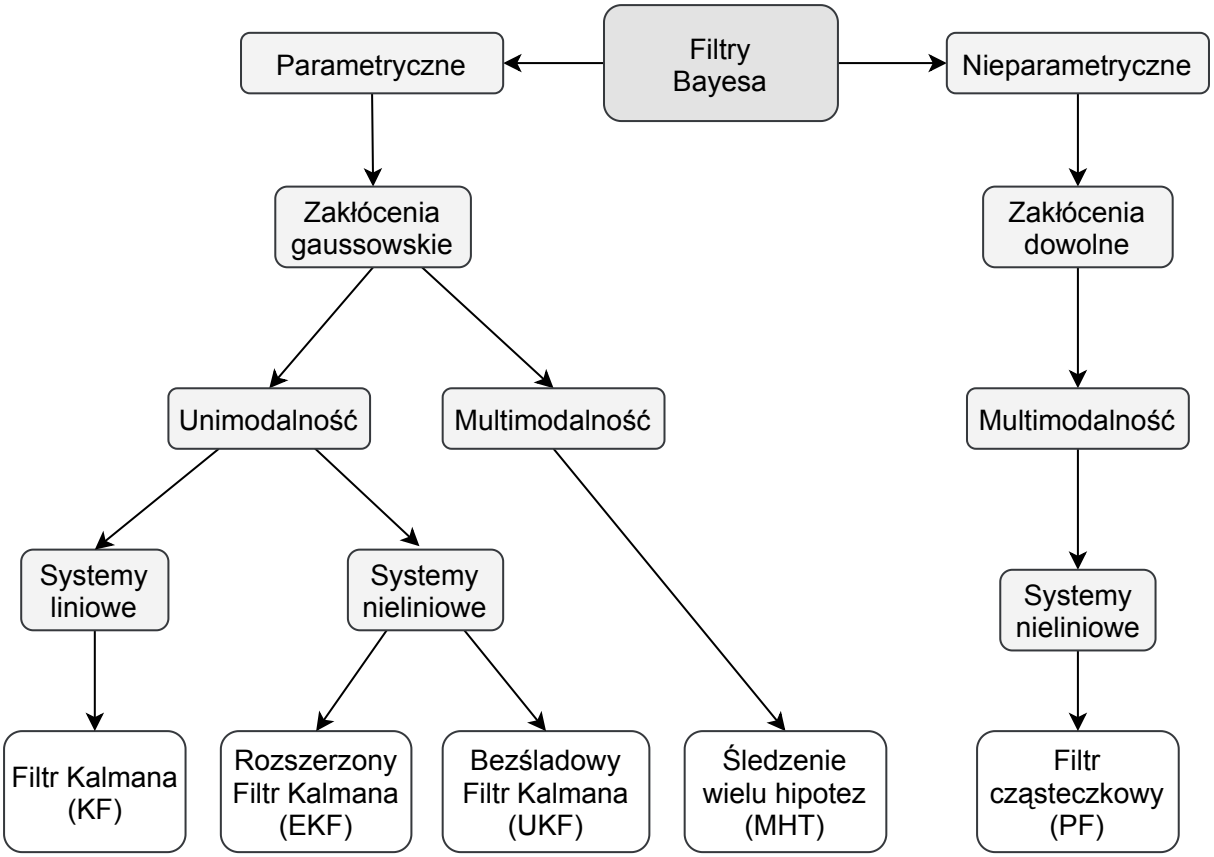
  predykcja: $\overline{bel}(x_t) = \int p(x_t \mid u_t, x_{t-1}) bel(x_{t-1}) dx$,
  korekcja: $bel(x_t) = \eta p(z_t \mid x_t) \overline{bel}(x_t)$,

przy czym η oznacza współczynnik normalizujący.

W przypadku robotów mobilnych nie wszystkie założenia procesu Markowa są spełnione, ale w praktyce filtry bayesowskie są stosunkowo odporne na błędy modeli.

Klasyfikacja metod opartych na filtrach Bayesa

Istnieje wiele odmian filtrów Bayesa (rys. 4.3). Jeden z podziałów dotyczy typu zakłóceń występujących w systemie. Wyróżnia się wtedy filtry gaussowskie przeznaczone dla systemów, w których zakłócenia mają rozkład normalny, oraz filtry dla systemów o dowolnych zakłóceniach. Kolejny podział dotyczy tego ile hipotez może być śledzonych. Filtry zakładające unimodalność systemu umożliwiają śledzenie tylko jednej hipotezy, natomiast w przypadku założenia multimodalności nie ma wspomnianego ograniczenia i śledzonych może być wiele hipotez odnośnie stanu prawdziwego. Ostatni podział dotyczy liniowości lub nieliniowości systemu.



Rysunek 4.3 Klasyfikacja filtrów Bayesa

Filtry gaussowskie

Filtry gaussowskie są to filtry Bayesa przeznaczone dla przestrzeni ciągłych. Opierają się na założeniu, że funkcja przekonań może być reprezentowana przez wielowymiaro-

we rozkłady normalne [40]

$$f_{\mu,\Sigma}(x) = \frac{1}{\sqrt{(2\pi)^n \left|\Sigma\right|}} \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)\right), \tag{4.13}$$

gdzie μ oznacza wartość średnią, a Σ kwadratową macierz kowariancji.

Podstawowe filtry gaussowskie zakładają unimodalność. Konsekwencją unimodalności rozkładu normalnego jest możliwość śledzenia tylko jednej hipotezy, czyli w szczególności umożliwiają estymację stanu jeżeli zmienna losowa jest w pobliżu stanu rzeczywistego. Z tego powodu podstawowe filtry gaussowskie rzadko są stosowane w globalnej lokalizacji robota, gdzie szczególnie w przypadku symetrycznych środowisk istnieje często wiele równie prawdopodobnych hipotez odnośnie pozycji robota. Natomiast istnieją też rozwiązania pozwalające na śledzenie wielu hipotez jednocześnie (MHT, ang. *Multi-Hypotheses Tracking*).

4.2.2 Filtr Kalmana

Filtr Kalmana (KF, ang. *Kalman Filter*) został opracowany w 1960 roku przez Rudolpha Kalmana [119]. Od tego czasu algorytm ten wraz z licznymi modyfikacjami, zyskał miano jednego z najpopularniejszych algorytmów wykorzystywanych do estymacji stanu systemów [33, 67, 80, 84, 161, 253].

Filtr Kalmana jest optymalnym estymatorem stanu $x_t \in \mathbb{R}^n$ systemów liniowych, których zakłócenia mają rozkład gaussowski. Zakłada się więc liniowość funkcji ewolucji stanów $p(x_t \mid u_t, x_{t-1})$, przez co wektor stanu przyjmuje postać

$$x_t = A_t x_{t-1} + B_t u_t + w_t, \tag{4.14}$$

gdzie A_t jest macierzą przejścia o rozmiarze $n \times n$ określającą zmianę stanu x_t od chwili $t-1$ do t, w przypadku braku sterowań oraz szumu. B_t to macierz wejścia o rozmiarze $n \times l$ określająca zmianę sterowań $u_t \in \mathbb{R}^l$ od chwili $t-1$ do t. Zmienna losowa w_t wyraża szum przetwarzania. Dodatkowo zakłada się liniowość funkcji prawdopodobieństwa obserwacji

$$z_t = H_t x_t + v_t, \tag{4.15}$$

gdzie H_t to macierz wyjścia o rozmiarze $m \times n$ określająca pomiary $z_t \in \mathbb{R}^m$, a v_t reprezentuje szum pomiarowy. Zakłada się, że zmienne losowe w_t i v_t są niezależne i mają rozkłady gaussowskie

$$p(w_t) \sim N(0, Q_t), \tag{4.16}$$
$$q(v_t) \sim N(0, R_t), \tag{4.17}$$

gdzie Q_t i R_t to odpowiednio macierze kowariancji szumu przetwarzania i pomiarowego.

Algorytm można podzielić na dwie fazy: predykcję i korekcję [253]. W przypadku KF dla systemów dyskretnych (ang. *Discrete Kalman filter, DKF*), równania predykcji, czyli aktualizacji czasu, przyjmują postać

$$\hat{x}_t^- = A_t \hat{x}_{t-1} + B_t u_t, \tag{4.18}$$
$$\hat{P}_t^- = A_t P_{t-1} A_t^T + Q_t. \tag{4.19}$$

Natomiast korekcja, czyli aktualizacja pomiarów, jest realizowana w oparciu o równania

$$K_t = \hat{P}_t^- H_t^T (H_t \hat{P}_t^- H_t^T + R_t)^{-1}, \tag{4.20}$$

$$\hat{x}_t = \hat{x}_t^- + K_t(z_t - H_t \hat{x}_t^-), \tag{4.21}$$

$$P_t = (I - K_t H_t)\hat{P}_t^-, \tag{4.22}$$

gdzie K_t określa się mianem wzmocnienia Kalmana.

4.2.3 Rozszerzony filtr Kalmana (EKF)

Rozszerzony filtr Kalmana (ang. *Extended Kalman Filter, EKF*) [33, 84, 169, 253] rozwiązuje problem estymacji stanu $x_t \in \mathbb{R}^n$ układu nieliniowego przez linearyzację w pobliżu punktu pracy systemu. Definiuje się następujące, nieliniowe funkcje stanu i pomiarów,

$$x_t = g(u_t, x_{t-1}) + w_t, \tag{4.23}$$

$$z_t = h(x_t) + v_t, \tag{4.24}$$

przy czym $z_t \in \mathbb{R}^m$, a zmienne losowe w_k oraz v_k podobnie jak w przypadku podstawowej wersji filtra Kalmana, określają szum przetwarzania (4.16) i szum pomiarowy (4.17). Linearyzacji dokonuje się przez rozwinięcie nieliniowych funkcji przejścia systemu oraz pomiaru $g(u_t, x_{t-1})$ w szereg Taylora

$$\begin{aligned} g(u_t, x_{t-1}) &\approx g(u_t, \mu_{t-1}) + g'(u_t, \mu_{t-1})(x_{t-1} - \mu_{t-1}) \\ &= g(u_t, \mu_{t-1}) + A_t(x_{t-1} - \mu_{t-1}), \end{aligned} \tag{4.25}$$

$$g'(u_t, x_{t-1}) = \frac{\partial g(u_t, x_{t-1})}{\partial x_{t-1}}. \tag{4.26}$$

Analogicznie rozwijana jest funkcja związana z pomiarami

$$h(x_t) \approx h(\overline{\mu}_t) + h'(\overline{\mu}_t)(x_t - \overline{\mu}_t), \tag{4.27}$$

$$h'(x_t) = \frac{\partial h(x_t)}{\partial x_t} = h(\overline{\mu}_t) + H_t(x_t - \overline{\mu}_t). \tag{4.28}$$

Przy czym macierze A_t i H_t to odpowiednio, jakobiany funkcji g i h określone dla x_t, natomiast W_t i V_t to również jakobiany funkcji g i h, ale określone dla w_t oraz v_t .

Podobnie jak w podstawowej wersji, algorytm można podzielić na dwie fazy: predykcję i korekcję [253]. W przypadku EKF dla systemów dyskretnych, równania predykcji, czyli aktualizacji czasu, przyjmują postać

$$\hat{x}_t^- = g(\hat{x}_{t-1}^-, u_{t-1}), \tag{4.29}$$

$$\hat{P}_t^- = A_t P_{t-1} A_t^T + W_t Q_t W_t^T. \tag{4.30}$$

Natomiast korekcja, czyli aktualizacja pomiarów, jest realizowana w oparciu o równania

$$K_t = \hat{P}_t^- H_t^T (H_t \hat{P}_t^- H_t^T + V_t R_t V_t^T)^{-1}, \tag{4.31}$$

$$\hat{x}_t = \hat{x}_t^- + K_t(z_t - h(\hat{x}_t^-)), \tag{4.32}$$

$$P_t = (I - K_t H_t)\hat{P}_t^-, \tag{4.33}$$

gdzie K_t to wzmocnienie Kalmana. Ze względu na linearyzację przeprowadzoną z wykorzystaniem szeregu Taylora, występują znaczne wartości błędów przy układach o dużych nieliniowościach.

EKF znajduje zastosowanie szczególnie w lokalizacji lokalnej, gdzie może być wykorzystywany do fuzji danych z różnych sensorów, jak system odometryczny, czy IMU.

4.2.4 Bezśladowy filtr Kalmana (UKF)

Modyfikacją algorytmu EKF jest zaproponowany w pracy [115] bezśladowy filtr Kalmana (ang. *Unscented Kalman Filter*) wykorzystujący przekształcenie bezśladowe (ang. *unscented*). Podstawowym przeznaczeniem algorytmu EKF jest estymacja stanu systemów o niewielkich nieliniowościach, natomiast znacznie gorzej radzi on sobie w mocno nieliniowych systemach, ze względu na zastosowaną linearyzację. Algorytm UKF zakłada estymowanie rozkładu prawdopodobieństwa zamiast linearyzacji funkcji nieliniowej. Estymowany rozkład nadal określa gaussowska zmienna losowa reprezentowana przez wyznaczone punkty sigma [50]. Punkty te poddawane są transformacjom nieliniowym opartym na modelach dynamiki i obserwacji systemu.

Bezśladowa transformacja pozwala obliczyć statystykę zmiennej losowej poddawanej nieliniowej transformacji [250]. Zakładając, że zmienna losowa x poddawana jest transformacji $y = g(x)$, oraz, że x posiada wartość średnią $\overline{x}$ i kowariancję Q_x. Obliczenie statystyki wymaga utworzenia macierzy $\mathcal{X}$ składającej się z $2n+1$ wektorów sigma wraz z odpowiadającymi im wagami W_i:

$$\begin{aligned}
\mathcal{X}_0 &= \overline{x} \\
\mathcal{X}_i &= \overline{x} + \left(\sqrt{(n+\lambda)Q_x}\right)_i \qquad i = 1, \ldots, n \\
\mathcal{X}_i &= \overline{x} - \left(\sqrt{(n+\lambda)Q_x}\right)_{i-n} \qquad i = n+1, \ldots, 2n \\
W_0^m &= \frac{\lambda}{(n+\lambda)} \\
W_0^c &= \frac{\lambda}{(n+\lambda) + (1-\alpha^2+\beta)} \\
W_i^m &= W_i^c = \frac{1}{2(n+\lambda)} \qquad i = 1, \ldots, 2n
\end{aligned} \tag{4.34}$$

gdzie

$$\lambda = \alpha^2(n+\kappa) - n \tag{4.35}$$

oznacza współczynnik skalujący [250]. Parametr α określa jak bardzo punkty sigma są rozproszone względem $\overline{x}$, natomiast κ i β to kolejne współczynniki skalujące. Wektory sigma przekazywane są przez funkcję nieliniową

$$\mathcal{Y}_i = g((X)_i), \tag{4.36}$$

natomiast wartość średnia i kowariancja są estymowane za pomocą wzorów:

$$\overline{y} \approx \sum_{i=0}^{2n} W_i^m \mathcal{Y}, \tag{4.37}$$

$$Q_y \approx \sum_{i=0}^{2n} W_i^c (\mathcal{Y}_i - \overline{y})(\mathcal{Y}_i - \overline{y})^T. \tag{4.38}$$

4.2.5 Filtry nieparametryczne - filtr cząsteczkowy

Kolejną pochodną filtrów Bayesa są filtry nieparametryczne. Rozwój i wykorzystanie filtrów nieparametrycznych w lokalizacji robotów mobilnych był podyktowany kilkoma kwestiami, a raczej ograniczeniami przedstawionych do tej pory filtrów.

Rozpoczynając od charakterystyki problemu lokalizacji robota w naturalnym środowisku, przedstawione zostaną ograniczenia obecnych rozwiązań. Założenia problemu lokalizacji:

- nieliniowość systemu, ponieważ robot podlega różnego rodzaju poślizgom, a także może zderzać się z innymi obiektami,
- nieliniowość pomiarów, gdyż charakterystyki wielu czujników są nieliniowe,
- szum nie zawsze ma rozkład normalny,
- wielowymiarowość, ponieważ lokalizacja wymaga śledzenia wielu zmiennych, takich jak współrzędne położenia, orientacji, ale także prędkość,
- multimodalność, szczególnie w przypadku lokalizacji globalnej. Przy ograniczonej ilości informacji, wiele hipotez odnośnie pozycji robota, może być równie prawdopodobnych.

Wracając do ograniczeń dotychczas przedstawionych rozwiązań, podstawowy dyskretny filtr Bayesa posiada ograniczenie w postaci jednowymiarowości zmiennej stanu. Filtr Kalmana pozwala uzyskać optymalną estymatę jedynie dla liniowych, unimodalnych systemów z szumem o rozkładzie normalnym. Pozostałe metody, czyli EKF i UKF mimo radzenia sobie z nieliniowością systemu, nadal ograniczone są unimodalnością.

Filtr cząsteczkowy

Filtr cząsteczkowy (PF, ang. *Particle Filter*) [34, 133, 227, 241] należy do grupy filtrów nieparametrycznych. Pozwala na estymację zmiennych stanu systemu nieliniowego, wielowymiarowego, a ponadto nie ma założenia o normalnym rozkładzie zakłóceń, ponieważ rozkład jest aproksymowany przez skończoną liczbę parametrów. Dodatkowo filtr pozwala śledzić wiele hipotez, czyli spełnia założenie multimodalności.

Idea działania filtra cząsteczkowego opiera się na reprezentacji rozkładu prawdopodobieństwa zmiennej stanu przez zbiór losowych próbek (cząsteczek) wylosowanych z tego rozkładu [241]. Stan robota w chwili t reprezentowany jest przez zbiór N cząsteczek

$$\mathcal{X}_t = \left\{x_t^1, x_t^2, \ldots, x_t^N\right\}, \tag{4.39}$$

oraz przez zbiór odpowiadających im wag

$$\mathcal{W}_t = \left\{w_t^1, w_t^2, \ldots, w_t^N\right\}. \tag{4.40}$$

Każda cząsteczka x_t^n jest hipotezą odnośnie stanu systemu w chwili t. Do każdej cząsteczki x_t^n przypisana jest odpowiadająca jej waga w_t^n, która określa prawdopodobieństwo prawdziwości hipotezy. Schemat postępowania w każdej iteracji filtru cząsteczkowego składa się z kilku operacji:

predykcja - w kroku tym na podstawie stanu poprzedniego x_{t-1}, wartości sterowań u_t oraz modelu przejścia $p(x_t \mid u_t, x_{t-1}^i)$, wyznaczane są pozycje każdej cząsteczki należącej do zbioru $\mathcal{X}_t$,

korekcja polega na obliczeniu współczynników wagowych w_t^i w oparciu o dane pomiarowe z_t oraz model pomiaru $p(z_t \mid x_t)$,

ponowny wybór cząsteczek (ang. *resampling*) polega na wylosowaniu z aktualnego zbioru cząsteczek z prawdopodobieństwem określonym przez przypisane im wagi. Krok ten wykonuje się aby zwiększyć efektywność filtra przy stałej liczbie cząsteczek, czyli aby zapewnić pozostanie cząsteczek w odpowiedniej części przestrzeni stanów. Bez ponownego wyboru cząsteczek istnieje możliwość, że filtr przestanie śledzić poprawne hipotezy.

Lokalizacja Monte-Carlo

Jedną z metod lokalizacji opierających swoje działanie na filtrze cząsteczkowym jest algorytm lokalizacji Monte-Carlo (MCL, ang. *Monte-Carlo Localization*) [72, 189, 255]. Metoda ta wykorzystuje znaną mapę otoczenia m w celu dopasowania do niej pomiarów z czujników odległości, na przykład LIDARów. Dodatkowo opiera swoje działanie na danych z systemu lokalizacji lokalnej, na przykład z systemu odometrycznego. Metoda MCL pozwala na reprezentację multimodalnych rozkładów prawdopodobieństwa, co powoduje, że z powodzeniem może być stosowana w globalnej lokalizacji robotów mobilnych.

W przypadku lokalizacji Monte-Carlo, każda cząsteczka x_t^n będąca hipotezą odnośnie stanu systemu w chwili t, ma postać pozycji robota, czyli jego położenia i orientacji. Dokładne kroki metody zostały przedstawione w postaci algorytmu 1.

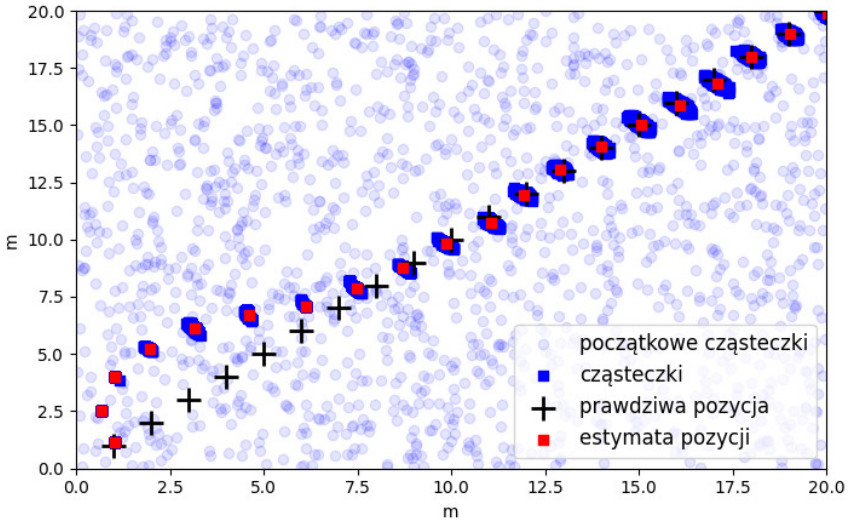
Algorytm 1: Lokalizacja Monte-Carlo bazująca na filtrze cząsteczkowym

Data: $\mathcal{X}_{t-1}, u_t, z_t$
Result: $\mathcal{X}_t$

$\overline{\mathcal{X}}_t = \mathcal{X}_t = \emptyset$
for *i = 1 to N* **do**
  $x_t^i \sim p(x_t \mid u_t, x_{t-1}^i)$ // Model ruchu
  $w_t^i = p(z_t \mid x_t^i)$ // Model obserwacji
  $\overline{\mathcal{X}}_t = \overline{\mathcal{X}}_t + [x_t^i, w_t^i]$
end
for *i = 1 to N* **do**
  wylosuj x_t^i z prawdopodobieństwem $\propto w_t^i$
  dodaj x_t^i do $\mathcal{X}_t$
end
return $\mathcal{X}_t$

Na rys. 4.4 umieszczono rezultat symulacji działania algorytmu lokalizacji Monte-Carlo. Symulowany robot poruszał się po linii prostej, a filtr wykorzystywał $N = 1500$ cząsteczek, początkowo rozmieszczonych losowo. Można zauważyć, że na początku działania metody, pozycja robota nie jest znana, więc wszystkie stany są równie prawdopodobne. Natomiast wraz z działaniem algorytmu, estymowany stan robota jest coraz bliższy rzeczywistego stanu.

Zwiększając liczbę cząsteczek N można uzyskać większą dokładności aproksymacji. Jednak wraz ze wzrostem liczby rośnie złożoność obliczeniowa algorytmu, dlatego wybór odpowiedniej liczby cząsteczek jest ważnym krokiem procedury strojenia algorytmu. Wersja adaptacyjna algorytmu (AMCL, ang. *Adaptive Monte-Carlo Localiza-*



Rysunek 4.4 Przykład symulacyjny działania metody Monte-Carlo

tion) [255] nie wykorzystuje stałej liczby cząsteczek, ale dobiera na bieżąco potrzebną w danym momencie liczbę cząsteczek, w oparciu o odpowiednio zdefiniowane miary.

4.3 Metody odometryczne

4.3.1 Odometria

Standardowa odometria określa pozycję robota na podstawie obrotów jego kół. Zwykle pomiar wykonywany jest przez czujniki położenia osi silników lub czujniki umieszczone bezpośrednio na kołach. Wykorzystuje się różnego rodzaju enkodery inkrementalne, między innymi optyczne i magnetyczne. Informacja z takich czujników oraz znajomość średnicy i konfiguracji kół, pozwala określać względną zmianę pozycji robota [27, 28].

Kinematyka monocykla

Rozważmy nieholonomicznego, dwukołowego robota mobilnego z układem jezdnym typu różnicowego (ang. *differential-drive*), nazywanego też monocyklem (ang. *unicycle*), poruszającego się po płaszczyźnie (rys. 4.5). W takim układzie, dwa niezależnie sterowane koła ustawione są prostopadle do osi symetrii robota. Wektor stanu $q = [x\ y\ \theta]^T \in \mathbb{R}^3$ określa pozycję robota, przy czym x, y określają położenie, a θ orientację robota. Kinematyka monocykla ma postać

$$\dot{q} = G(q)u = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} \nu \\ \omega \end{bmatrix}, \tag{4.41}$$

gdzie ν to prędkość liniowa, a ω prędkość kątowa robota. W oparciu o prędkość liniową i prędkość kątową robota można wyznaczyć prędkości kątowe kół robota: prawego v_r

oraz lewego v_l, za pomocą zależności

$$\begin{bmatrix} v_r \\ v_l \end{bmatrix} = \frac{1}{2r} \begin{bmatrix} 2 & b \\ 2 & -b \end{bmatrix} \cdot \begin{bmatrix} \nu \\ \omega \end{bmatrix}, \tag{4.42}$$

gdzie r to promień koła, a b oznacza odległość między kołami robota.

Model dyskretny aktualizacji pozycji

Niech $q_n = [x_n, y_n, \theta_n]^T \in \mathbb{R}^3$ oznacza wektor stanu w kroku n, q_{n-1} wektor stanu w kroku poprzednim, a q_0 wartości początkowe. Zakładając, że robot porusza się bez poślizgów, czyli prędkości chwilowe kół w punktach ich styku z podłożem są równe zeru, pozycję robota można wyznaczyć w oparciu o model dyskretny (rys. 4.5) [228]:

$$\begin{bmatrix} x_n \\ y_n \\ \theta_n \end{bmatrix} = \begin{bmatrix} x_{n-1} \\ y_{n-1} \\ \theta_{n-1} \end{bmatrix} + \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta\theta \end{bmatrix}, \tag{4.43}$$

gdzie:

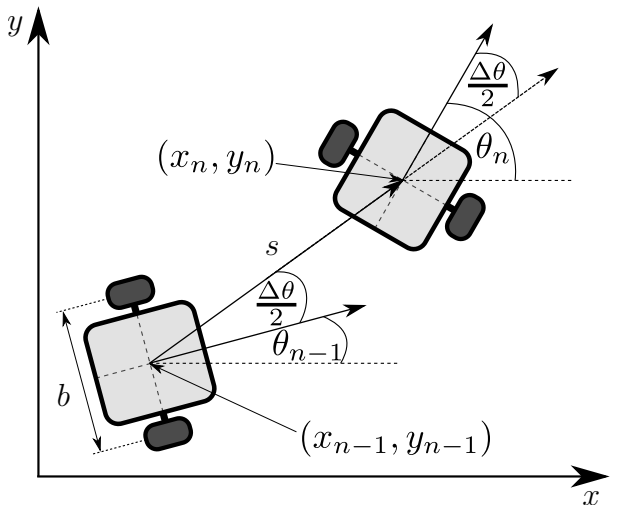
$$s = \frac{s_R + s_L}{2}, \tag{4.44}$$

$$\Delta\theta = \frac{s_R - s_L}{b}, \tag{4.45}$$

$$\Delta x = s \cdot \cos\left(\theta + \frac{\Delta\theta}{2}\right), \tag{4.46}$$

$$\Delta y = s \cdot \sin\left(\theta + \frac{\Delta\theta}{2}\right), \tag{4.47}$$

a s_L i s_R oznaczają odpowiednio: przemieszczenie koła lewego i prawego w danym kroku. W przypadku robotów czterokołowych, jednym ze stosowanych rozwiązań jest uśrednianie wartości dla każdej pary kół.



Rysunek 4.5 Wyznaczanie odometrii dla kołowego robota mobilnego z napędem różnicowym

Błędy wyznaczania pozycji

Błędy wyznaczania pozycji występujące w odometrii dzieli się na dwie kategorie [30]: systematyczne, czyli wynikające w głównej mierze z przyjętych uproszczeń w metodzie obliczeniowej oraz losowe. Do przyczyn powstawania błędów systematycznych można zaliczyć:

- niedokładność określania położenia kół robota, związaną ze skończoną rozdzielczością pomiarową sensorów,
- niedokładność określania średnicy kół. Dodatkowo z powodu obecności na kole elementów amortyzujących i zwiększających przyczepność, na przykład w postaci opon, średnica kół może być zależna od aktualnego nacisku koła na podłoże,
- niedokładność wyznaczania rozstawu kół L,
- brak idealnej zbieżności kół robota, czyli asymetria montażu.

Do kolejnej grupy błędów zalicza się zjawiska o charakterze losowym, między innymi:

- poślizgi kół, zależne od rodzaju układu napędowego oraz zmiennego rodzaju podłoża. Wpływ na nie mogą mieć również przyspieszenia robota, a także kolizje z obiektami otoczenia.
- nierówności podłoża po którym przemieszcza się robot,
- w rzeczywistości, styk kół z podłożem nie jest punktowy.

Błędy systematyczne wynikają w większości z niedokładności wykonania konstrukcji robota, w szczególności układu jezdnego. Odpowiednia kalibracja pozwala zniwelować wpływ takich błędów na dokładność wyznaczania pozycji. W pracach [29, 32] przedstawiono metodę kalibracji odometrii dla robotów z dwoma kołami napędowymi. Natomiast jeżeli chodzi o eliminowanie wpływu błędów losowych, w szczególności związanych z poślizgami kół, jedną z metod jest wykorzystanie fuzji danych [236, 268] z dodatkowych sensorów, na przykład inercyjnych. Pozwala to ograniczyć wpływ większych poślizgów powodowanych między innymi najechaniem na nierówność podłoża lub kolizją z przeszkodami.

4.3.2 Odometria wizyjna

Zasada działania odometrii wizyjnej (VO, ang. *Visual Odometry*) opiera się na wykrywaniu i estymowaniu ruchu robota na podstawie danych z czujników wizyjnych. Otrzymywane przemieszczenia są względne, a pozycja robota może być rozpatrywana w lokalnym układzie odniesienia. Metoda ta pozwala lokalizować robota z dużą dokładnością i w sposób odporny na czynniki zewnętrzne.

Zestawienie wizyjnych metod odometrycznych zostało umieszczone w pracy [9]. Metody można podzielić na podstawie typu wykorzystywanych sensorów, lub systemów sensorów. Rozwiązania opierają się na:

- systemach monowizyjnych [111, 150],
- systemach stereowizyjnych [15, 112, 160, 171],

- systemach kamer wielokierunkowych [248],
- kamerach RGB-D [65, 125].

Zaletą systemów monowizyjnych jest niewielki koszt i łatwość implementacji rozwiązań, między innymi z powodu niewymagającej kalibracji sensora. Inną zaletą jest niewielki rozmiar i waga kamer, co nie jest pomijalne w przypadku niewielkich robotów.

Od strony przetwarzania danych, metody wizyjnej odometrii można podzielić ze względu na przyjęty sposób estymacji przemieszczenia.

Metody oparte na detekcji wykrywają cechy takie jak linie, narożniki, czy krzywe na kolejnych obrazach [19, 112]. Następnie, cechy z dwóch kolejnych obrazów są do siebie dopasowywane, co pozwala uzyskać wektor par cech, gdzie każda para składa się z cechy z poprzedniego i bieżącego obrazu. Na podstawie wektora par oblicza się przemieszczenie robota. Odometria wizyjna oparta na detekcji cech została zaimplementowana między innymi w systemie wykorzystywanym podczas misji eksploracji Marsa [156].

Metody oparte na analizie wyglądu obrazów analizują zmiany wyglądu zarejestrowanych obrazów, przez monitorowanie zmian jasności pikseli. Ruch kamery jest estymowany z użyciem metody optycznego przepływu OF (ang. *Optical Flow*). Algorytm ten wykorzystuje wartości intensywności sąsiednich pikseli w celu obliczenia przesunięcia szablonów jasności z jednego obrazu na drugi.

Optyczne czujniki przemieszczenia

Jednym z przykładów wykorzystania odometrii wizyjnej są rozwiązania bazujące na zintegrowanych, optycznych czujnikach przemieszczenia, stosowanych między innymi w myszach komputerowych. Podobne rozwiązania można z powodzeniem wykorzystywać w robotyce mobilnej.

Zasada działania optycznych czujników przemieszczenia opiera się na technologii nawigacji optycznej (ang. *Optical Navigation Technology*) [184], czyli dedykowany układ obliczeniowy określa przemieszczenie podłoża bazując na zmieniającym się obrazie.

Zastosowanie takiego czujnika w lokalizacji robota mobilnego pozwala rozwiązać problem poślizgów kół oraz innych błędów występujących w odometrii. Dodatkowo na podstawie parametrów sensorów [184], można zauważyć, że umożliwiają one odświeżanie pomiarów z częstotliwością dochodzącą do $6kHz$, przy rozdzielczości równej 400 CPI (ang. *Counts Per Inch*), co oznacza liczbę punktów na cal.

Rozwiązania tego typu mają jednak pewne ograniczenia. Poprawna praca sensora wymaga umieszczenia go w ustalonej, stałej odległości od badanej powierzchni, co może być trudne do zrealizowania w przypadku niektórych konstrukcji robotów. Dodatkowo stała odległość czujnika od powierzchni, znacząco ogranicza jego zastosowanie tylko do przypadków z równym podłożem. Kolejnym ograniczeniem mogą być maksymalne wartości prędkości i przyspieszeń badanego obiektu.

4.3.3 Odometria inercyjna

Względna metoda lokalizacji inercyjnej wykorzystuje czujniki inercyjne do pomiaru zmiany pozycji robota, takie jak akcelerometry, żyroskopy [137, 186, 264], oraz opcjonalnie magnetometry, zwykle umieszczone w jednym module IMU. W metodzie tej położenie oblicza się przez dwukrotne całkowanie przyspieszenia zmierzonego akcelerome-

trem, natomiast orientację robota wyznacza się korzystając z wartości przemieszczeń, oraz całkując dane pomiarowe z żyroskopu prędkościowego.

Z powodu dużego dryfu żyroskopu, a także zakłóceń w pomiarach akcelerometru, niezwykle trudno jest stosować wyłącznie metodę inercyjną do lokalizacji robotów w przestrzeni. Jednak przez połączenie jej z innymi technikami pozwala uzyskiwać zadowalające rezultaty.

Jednym z rozwiązań jest wykorzystanie fuzji danych z IMU oraz standardowej odometrii [264]. Ma to na celu zmniejszenie wartości błędów wyznaczania orientacji robota, co jest problematyczne w przypadku wykorzystywania jedynie informacji o obrotach kół. Szczególne znaczenie ma to w konstrukcjach robotów, w których zmiana kierunku ruchu jest następstwem poślizgów kół, jak na przykład w robotach czterokołowych. W takich konstrukcjach IMU może zostać wykorzystane do wykrywania poślizgów kół.

4.3.4 Odomteria ultradźwiękowa

Czujniki ultradźwiękowe (sonary) mogą z powodzeniem być stosowane do lokalizacji robotów mobilnych. Wykorzystuje się w tym celu triangulację i oblicza zmianę położenia między wartościami uzyskiwanymi z par czujników. Natomiast stosując więcej czujników, w formie matryc można uzyskać wysoką dokładność lokalizacji względem otoczenia [134].

Wadą sonarów jest natomiast wrażliwość na materiał, z którego wykonana została przeszkoda oraz kierunek przeszkody. Dodatkowym problemem są zakłócenia powstające od innych sonarów, jeżeli czujniki działają na tej samej częstotliwości, a także problem z rejestracją wielokrotnie odbijanych fal dźwiękowych.

4.3.5 Odometria w oparciu o skaner laserowy

Skanery laserowe mogą być wykorzystywane do lokalizacji robotów mobilnych na wiele sposobów. Jednym z nich jest estymowanie przemieszczenia robota na podstawie zmian odległości do przeszkód [106, 147].

Skanery laserowe mają w tym zastosowaniu kilka zalet w porównaniu do sonarów. Przede wszystkim ich kąt widzenia jest większy, ale też cechują się większą dokładnością pomiarów. Do minusów można natomiast zaliczyć wysoki koszt takich rozwiązań oraz większą złożoność obliczeniową algorytmów, które estymują ruch dopasowując do siebie kolejne skany.

4.4 Lokalizacja w oparciu o znaczniki

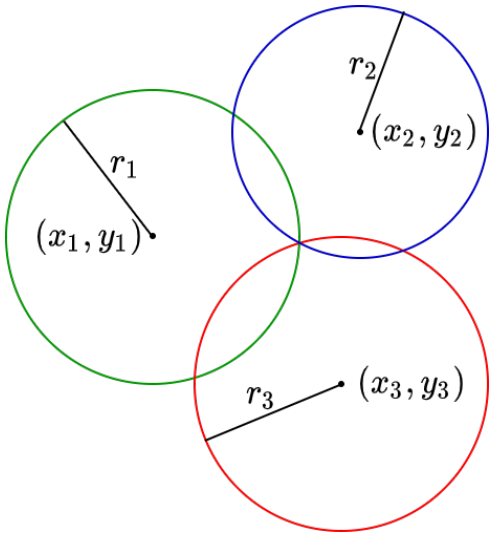
Metody lokalizacji w oparciu o znaczniki dzieli się na dwie grupy: wykorzystujące znaczniki naturalne [62], oraz takie, które korzystają ze sztucznych znaczników [192]. Zaletą pierwszego rozwiązania jest brak modyfikacji środowiska, a dodatkowo działania robota nie muszą się ograniczać do obszaru, gdzie umieszczone zostały specjalnie przygotowane znaczniki. Jednak wykrywanie naturalnych znaczników jest trudniejszym zadaniem.

W przypadku drugiej grupy metod, na obszarze po którym będzie poruszał się robot, umieszcza się zestaw znaczników: aktywnych lub pasywnych. Zadaniem robota jest detekcja takich znaczników bądź komunikacja z nimi. Główna wada takiego rozwiązania to potrzeba instalacji dedykowanych systemów znaczników. Natomiast uzyskuje się

przez to dużą dokładność i niezawodność systemów lokalizacji. Z powodzeniem rozwiązania tego typu stosowane są w halach produkcyjnych, halach magazynowych, czy kopalniach.

Sztuczne znaczniki mogą mieć formę zarówno wizualną, na przykład kody QR (ang. *Quick Response*), jak i formę nadajników radiowych. Istnieją dwie podstawowe techniki wyznaczania pozycji robota na podstawie znaczników:

- trilateracja - pozycja określana jest na podstawie odległości do znaczników, których położenia są znane (rys. 4.6). Technika ta jest stosowana w przypadku nadajników radiowych, gdzie mierzony jest czas przelotu fali (ToF) radiowej do poszczególnych nadajników,
- triangulacja - pozycja jest wyznaczana przez kąty względem znanych znaczników.



Rysunek 4.6 Przykład trilateracji 2D - położenie obiektu w miejscu przecięcia trzech okręgów

4.4.1 Globalne systemy nawigacji satelitarnej

Globalne systemy nawigacji satelitarnej (GNSS, ang. *Global Navigation Satellite Systems*), pokrywające swoim zasięgiem całą Ziemię, zdobyły ogromną popularność. Do obecnych i przyszłych realizacji takich systemów można zaliczyć:

- GPS (ang. *Global Positioning System*, USA),
- GLONASS (ang. *Global Navigation Satellite System*, Rosja),
- GALILEO (ang. *European Global Satellite Navigation System*, Europa),
- Beidou/Compass (Chiny).

Systemy nawigacji satelitarnej z dużą precyzją dostarczają informacji o aktualnym czasie i absolutnej lokalizacji [3]. W dalszej części skupiono się na systemie GPS, aczkolwiek zasady działania poszczególnych systemów są zbliżone. Aby zapewnić pokrycie całej Ziemi, system GPS wykorzystuje 24 satelity orbitujące wokół planety. Satelity

transmitują zakodowane sygnały radiowe (RF, ang. *Radio Frequency*). Do określenia lokalizacji, wystarczy, że odbiornik GPS użytkownika jest w zasięgu 4 satelitów. Odbiornik GPS oblicza swoje położenie w oparciu o metodę trilateracji. Odległości do satelitów obliczane są na podstawie czasu przelotu fal radiowych od nadajników. Każdy satelita wyposażony jest w transmiter, odbiornik radiowy oraz zegar atomowy.

W robotyce mobilnej, odbiorniki systemów globalnej nawigacji wykorzystywane są przede wszystkim, gdy roboty poruszają się na zewnątrz [79, 102].

4.5 Badania eksperymentalne wybranych metod lokalizacji

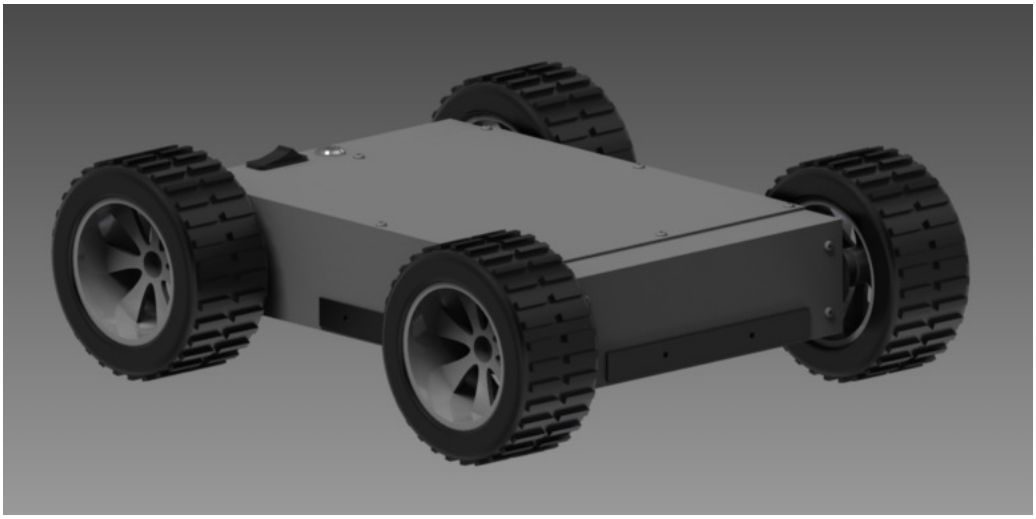
Celem przedstawionych badań eksperymentalnych było zweryfikowanie i porównanie wybranych metod lokalizacji kołowych robotów mobilnych. Przedstawione wyniki zostały wcześniej opublikowane w pracy [51].

4.5.1 Opis stanowiska badawczego

Na potrzeby badań eksperymentalnych wykorzystano dwa roboty mobilne: robot czterokołowy [51] i prototypowy robot ReMeDi [10, 11, 107, 108, 196]. Roboty te wyposażone były w szereg sensorów, między innymi enkodery, IMU, lidary, czy sensory RGB-D.

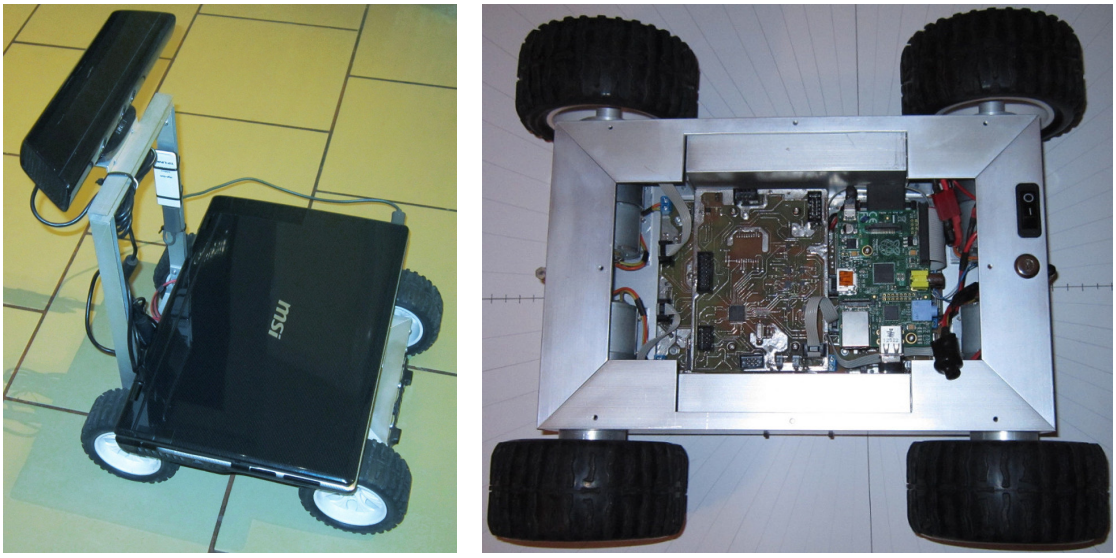
Robot czterokołowy

Robot mobilny posiada cztery koła z niezależnymi napędami (rys. 4.7 i 4.8). Jego



Rysunek 4.7 Wizualizacja podstawowej wersji czterokołowego robota mobilnego wykorzystywanego w badaniach

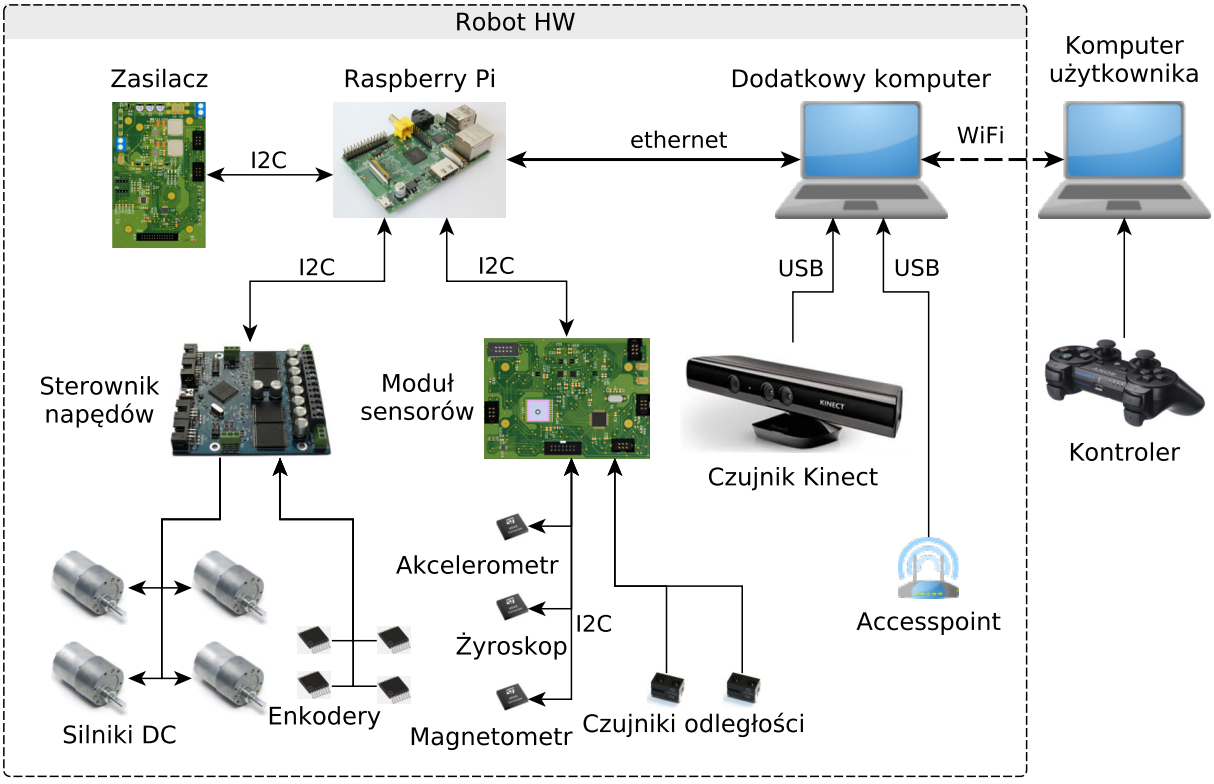
szerokość i długość wynoszą około 35 cm, a wysokość wraz z zamontowanym czujnikiem Kinect to około 50 cm. Podstawowa wersja robota (rys. 4.7) wyposażona została w jednostkę obliczeniową Raspberry Pi [194], opartą na procesorze *ARM* (pojedynczy rdzeń, 700 MHz). We wnętrzu robota umieszczono cztery silniki prądu stałego (DC, ang. *Direct Current*) oraz szereg sensorów. Dodatkowo na potrzeby testów robot został



(a) z czujnikiem RGB-D (b) baza robota

Rysunek 4.8 Czterokołowy robot mobilny wykonany na potrzeby pracy [51]

rozbudowany o drugi komputer (*Intel Core i3*, 4GB RAM), oraz czujnik Microsoft Kinect. Architektura sprzętowa (rys. 4.9) robota składa się z także z modułu sensorów i dedykowanego sterownika silników.



Rysunek 4.9 Architektura sprzętowa czterokołowego robota mobilnego

Robot ReMeDi

Robot mobilny ReMeDi (rys. 4.10) został zaprojektowany i wykonany przez firmę Accrea Engineering w ramach międzynarodowego projektu ReMeDi (ang. *Remote Medical Diagnostician*) [196] finansowanego przez UE. Prototypowy robot wyposażony jest w

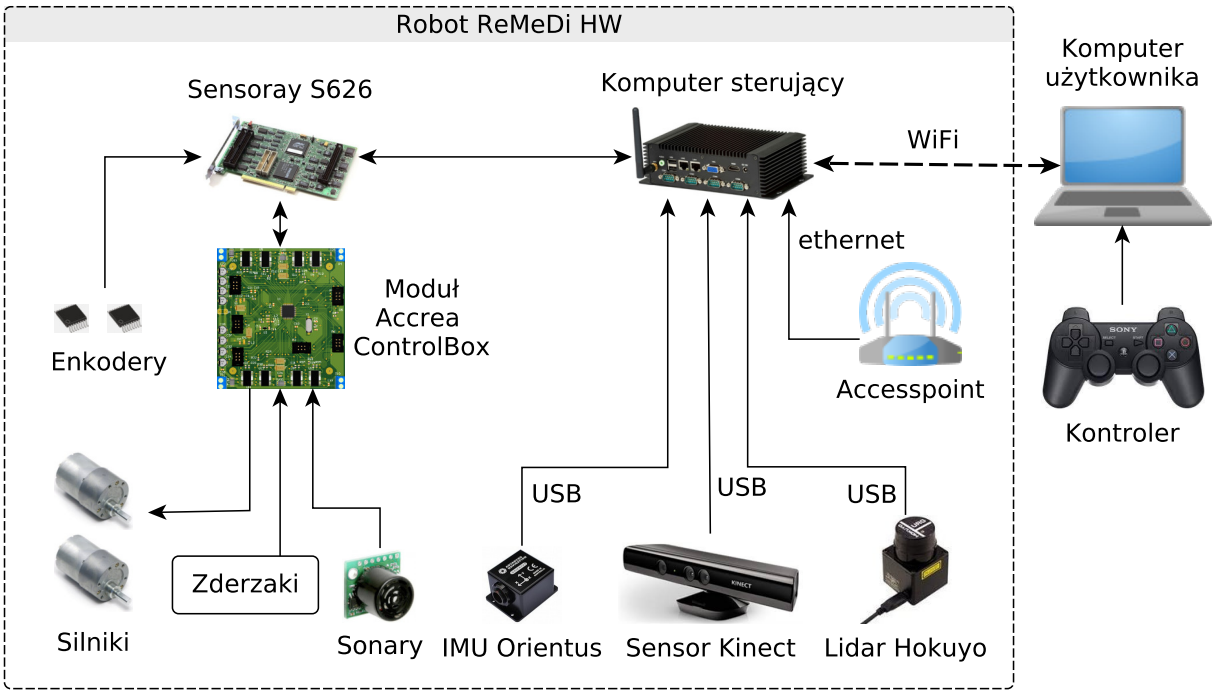
Rysunek 4.10 Prototypowa wersja robota mobilnego ReMeDi [196]

dwa koła napędowe (układ różnicowy) oraz dwa koła samonastawne. Wymiary podstawy w kształcie sześciokąta to około $0,7 \times 0,7$ m, a wysokość wynosi $1,5$ m. Robot został wyposażony w szereg czujników, między innymi lidar Hokuyo, sonary, sensor Kinect, IMU oraz zderzaki. Za sterowanie napędami odpowiada moduł *ControlBox* wykonany przez firmę Accrea, który komunikuje się z komputerem sterującym za pośrednictwem karty *Sensoray S626*. Pełna architektura sprzętowa została przedstawiona na rys. 4.11.

System śledzenia ruchu

Badania eksperymentalne dotyczące metod lokalizacji robotów przeprowadzono w laboratorium robotyki mobilnej (rys. 4.12), wyposażonym w optyczny system śledzenia ruchu (ang. *Motion Capture System*) OptiTrack. W skład systemu wchodziło sześć kamer wizyjnych OptiTrack Prime 17W (rys. 4.13(a)), których kąty widzenia wynoszą $70°$, a rozdzielczości to $1,7$ MPx. Na potrzeby systemu śledzenia ruchu, na robotach umieszczono odbijające światło znaczniki (rys. 4.13).

Przedstawiony system śledzenia ruchu umożliwia wysyłanie przez sieć pozycji śledzonych obiektów. Aby odebrać dane z poziomu środowiska ROS, wykorzystano moduł *mocap_optitrack* [201].



Rysunek 4.11 Architektura sprzętowa robota mobilnego ReMeDi

Rysunek 4.12 Pomieszczenie w którym realizowano eksperymenty wraz metalową konstrukcja, do której zamontowano kamery systemu śledzenia ruch OptiTrack

4.5.2 Charakterystyka badań eksperymentalnych

Badania metod lokalizacji przeprowadzono w dwóch etapach. W pierwszym etapie zebrano dane z przejazdów robotów i zapisano w postaci plików *rosbag*. W drugim etapie, realizowanym offline, odtworzono działanie systemu na podstawie zebranych danych i zbadano jakość estymacji pozycji przez poszczególne algorytmy.

Pierwszy etap składał się z przejazdów robotami w obszarze działania optycznego systemu śledzenia ruchu OptiTrack. W trakcie tych przejazdów, roboty były sterowane manualnie. System sterowania robotów zaimplementowano w środowisku ROS. Do

(a) kamera OptiTrack Prime 17W

(b) odblaskowy znacznik

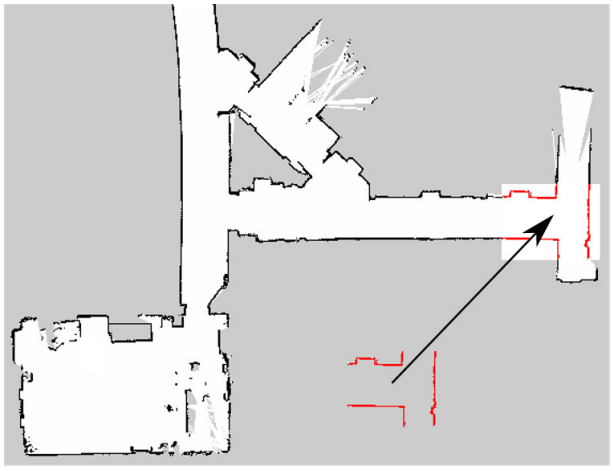
(c) robot z zamontowanymi znacznikami

Rysunek 4.13 Elementy używanego systemu śledzenia ruchu OptiTrack

zbierania danych wykorzystano mechanizm *rosbag* [201], umożliwiający zapisywanie i przechowywanie wybranych danych z działającego systemu w formie plików *rosbag*. Z każdego przejazdu zapisano następujące dane:

- transformacje lokalnych układów współrzędnych (ramek) związanych z robotem (*/tf* oraz */tf_static*),
- zadane prędkości robota (*/cmd_vel*),
- odometria zawierająca aktualne prędkości robota (*/odom*),
- dane pomiarowe z IMU (*/imu*),
- dane pomiarowe z lidara (*/scan*),
- odczyty z czujnika RGB-D przekształcone do postaci 2D (*/scan2*),
- oraz dane z systemu śledzenia ruchu (*/motion_capture_gt*).

Jakość estymacji orientacji i położenia porównywano dla wybranych metod, z wykorzystaniem zebranych wcześniej danych. Badano między innymi metodę filtracji Madgwicka MF (ang. *Madgwick Filter*) [154], który umożliwia estymację orientacji na podstawie pomiarów z akcelerometru i żyroskopu. Sprawdzano również algorytmem UKF [115], który realizuje fuzję danych z systemu odometrii (Odom) oraz danych z IMU. Przeprowadzono również eksperymenty z adaptacyjnym algorytmem lokalizacji Monte-Carlo (AMCL) [242] (rys. 4.14). Metoda AMCL wymaga do działania danych odometrycznych (Odom), odczytów z lidara lub sensora Kinect oraz utworzonej wcześniej statycznej mapy otoczenia. Działanie tego algorytmu badane zarówno z odczytami z lidara Hokuyo (AMCL) oraz sensora Kinect (AMCL2). W trakcie eksperymentów obserwowano również wpływ prędkości liniowych (lin) i kątowych (ang) na błędy poszczególnych metod.

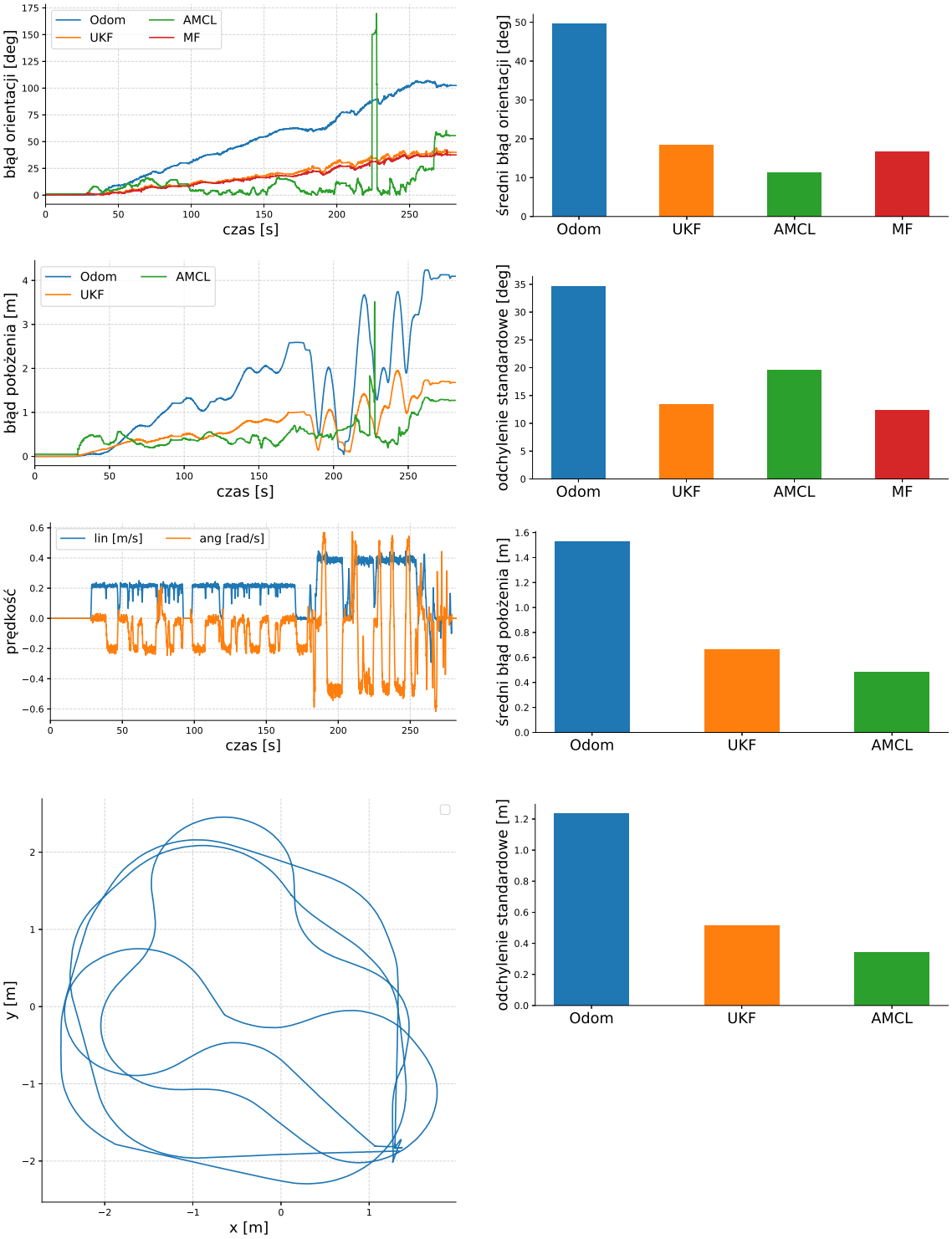


Rysunek 4.14 Przykład dopasowania skanu do mapy 2D algorytmem AMCL

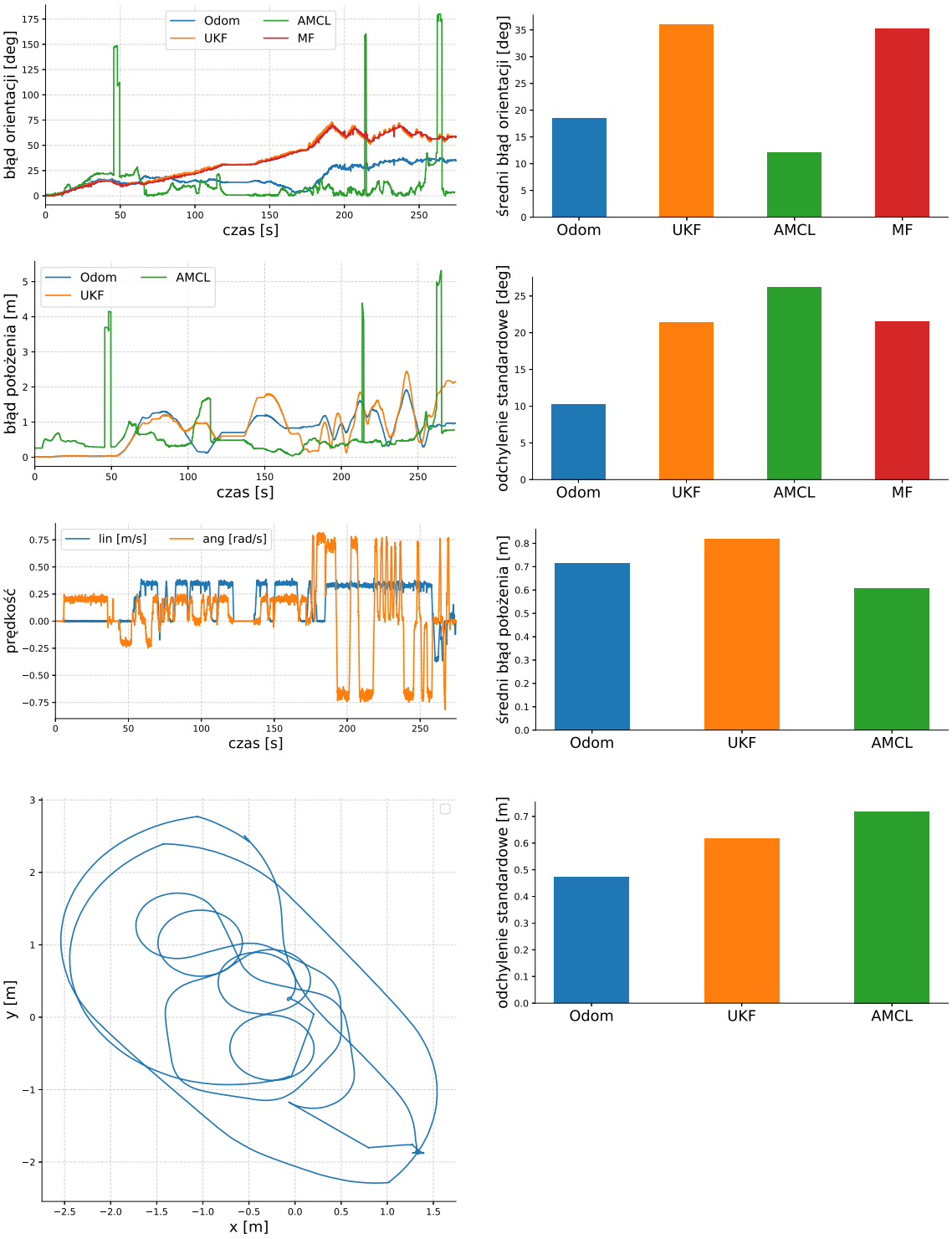
4.5.3 Wyniki badań eksperymentalnych jakości estymacji pozycji

Rezultaty eksperymentów z dwóch przejazdów robota czterokołowego przedstawiono na rys. 4.15 oraz 4.16. Dwa zarejestrowane przejazdy robota ReMedi przedstawiono na 4.17 oraz 4.18.

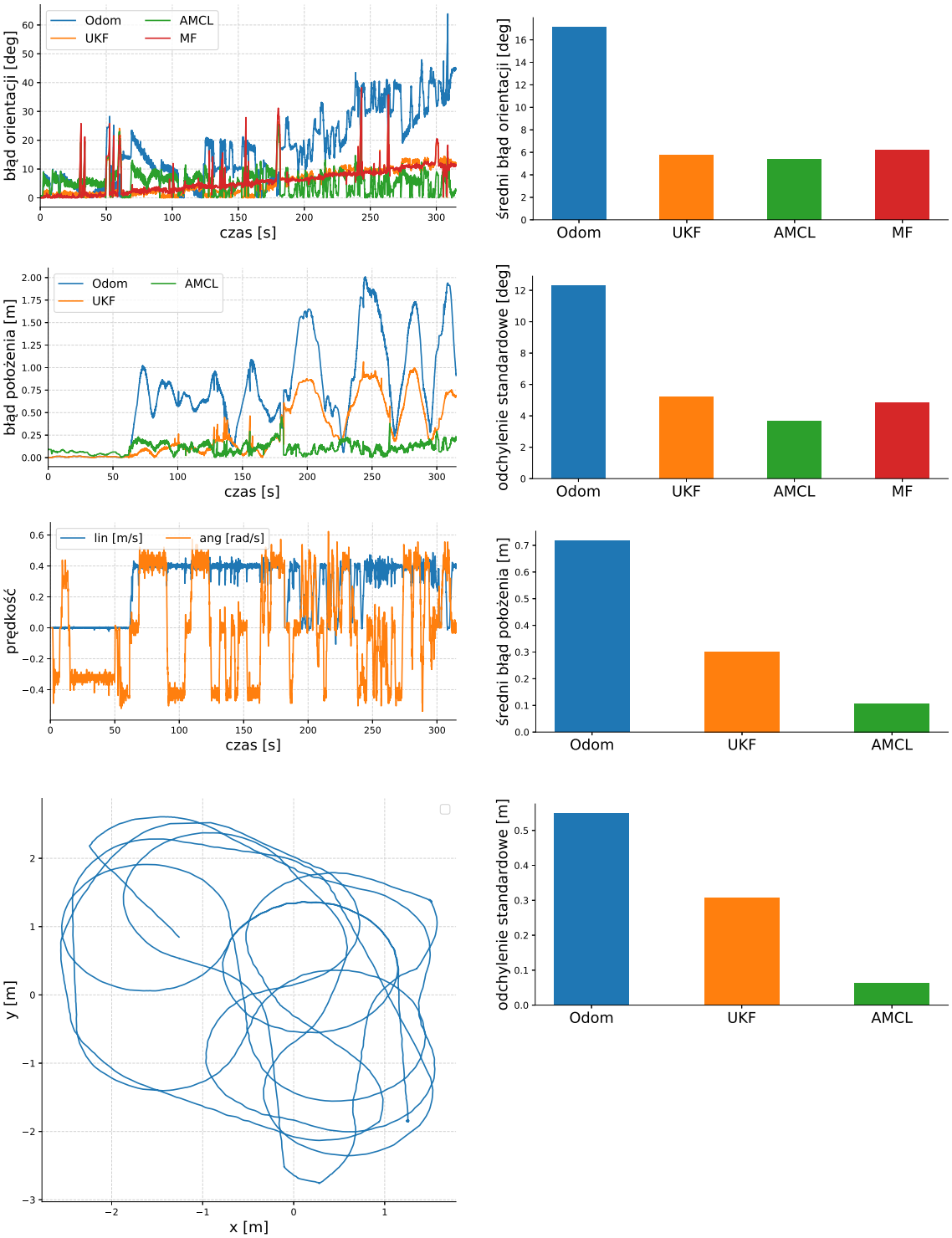
Kolejne rezultaty zaprezentowano na rys. 4.19 i dotyczą one porównania wielkości błędów pozycji dwóch robotów o odmiennej konstrukcji, które poruszały się po podobnych trajektoriach. W każdym z przypadków okrążenia były powtarzane wielokrotnie. Na rys. 4.20 oraz 4.21 umieszczono zbiorcze, uśrednione wartości błędów obliczone na podstawie kilkunastu przejazdów, odpowiednio dla robota czterokołowego oraz dwukołowego ReMeDi. Na rys. 4.22 oraz 4.23 zamieszczono porównanie działania algorytmu AMCL wykorzystującego odczyty z lidara (oznaczenie AMCL) oraz z sensora Kinect (oznaczenie AMCL2). Przejazdy wykonywano robotem mobilnym ReMeDi wyposażonym w obydwa sensory. W obydwu przypadkach wykorzystano tę samą mapę w postaci siatki zajętości 2D, utworzoną algorytmem gmapping.



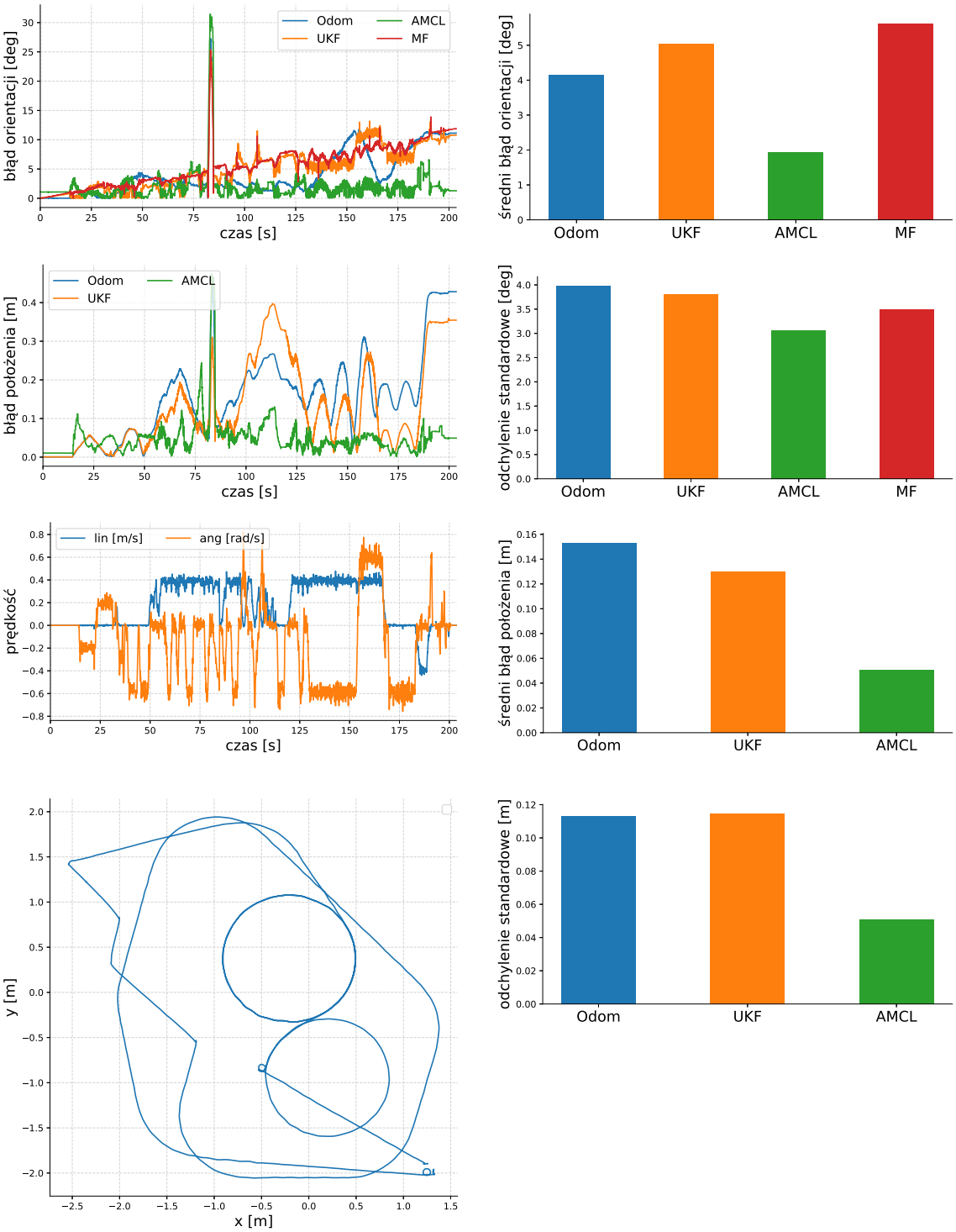
Rysunek 4.15 Porównanie jakości estymacji trajektorii przez metody lokalizacji na podstawie przejazdu robota czterokołowego w eksperymencie nr I



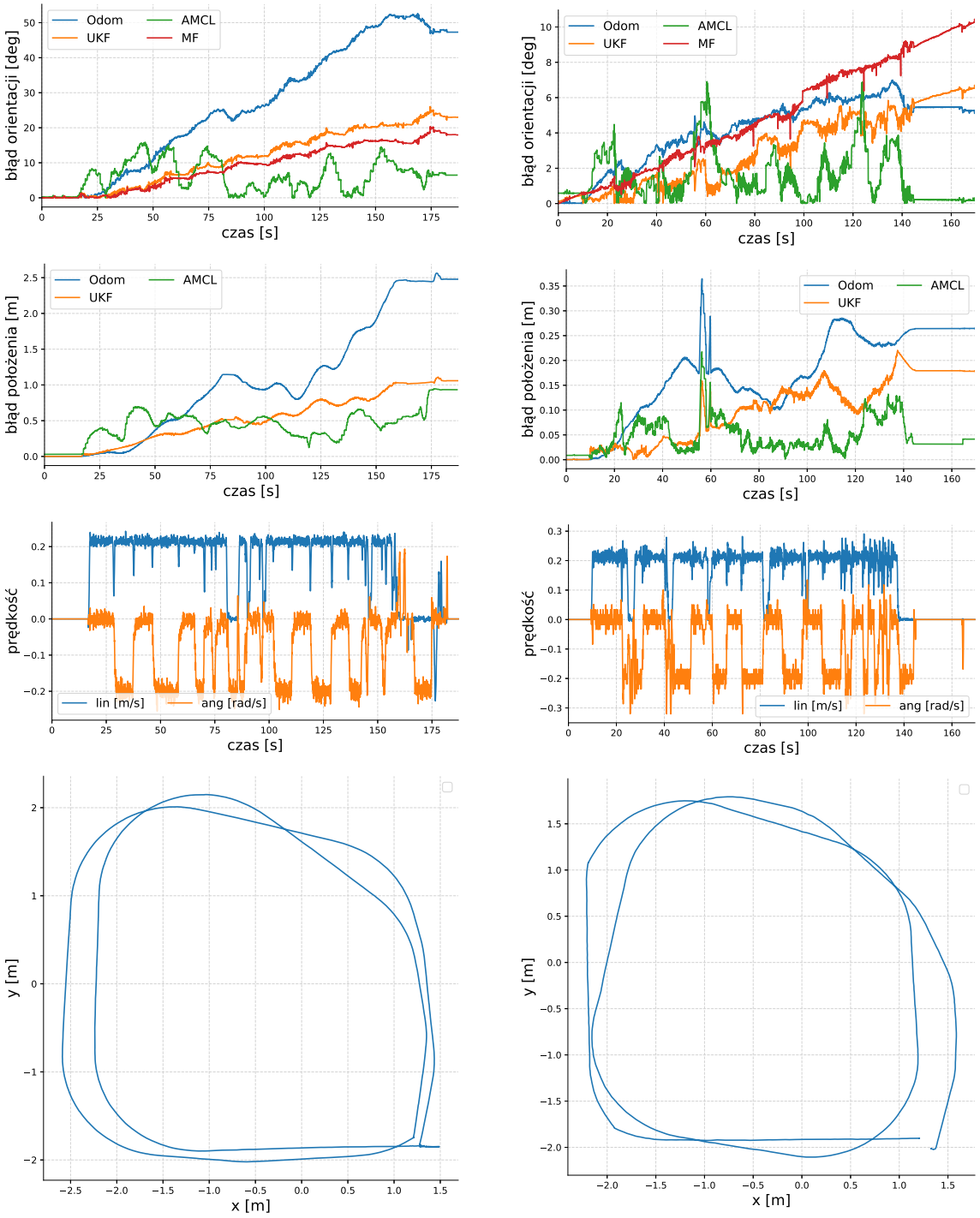
Rysunek 4.16 Porównanie jakości estymacji trajektorii przez metody lokalizacji na podstawie przejazdu robota czterokołowego w eksperymencie nr II



Rysunek 4.17 Porównanie jakości estymacji trajektorii przez metody lokalizacji na podstawie przejazdu robota ReMeDi w eksperymencie nr III



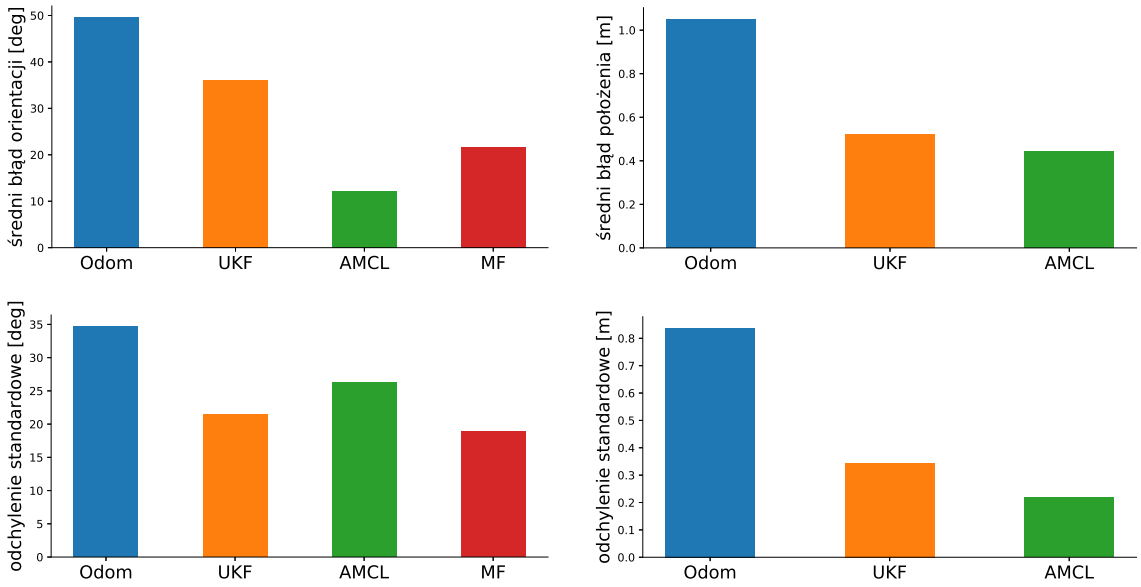
Rysunek 4.18 Porównanie jakości estymacji trajektorii przez metody lokalizacji na podstawie przejazdu robota ReMeDi w eksperymencie nr IV



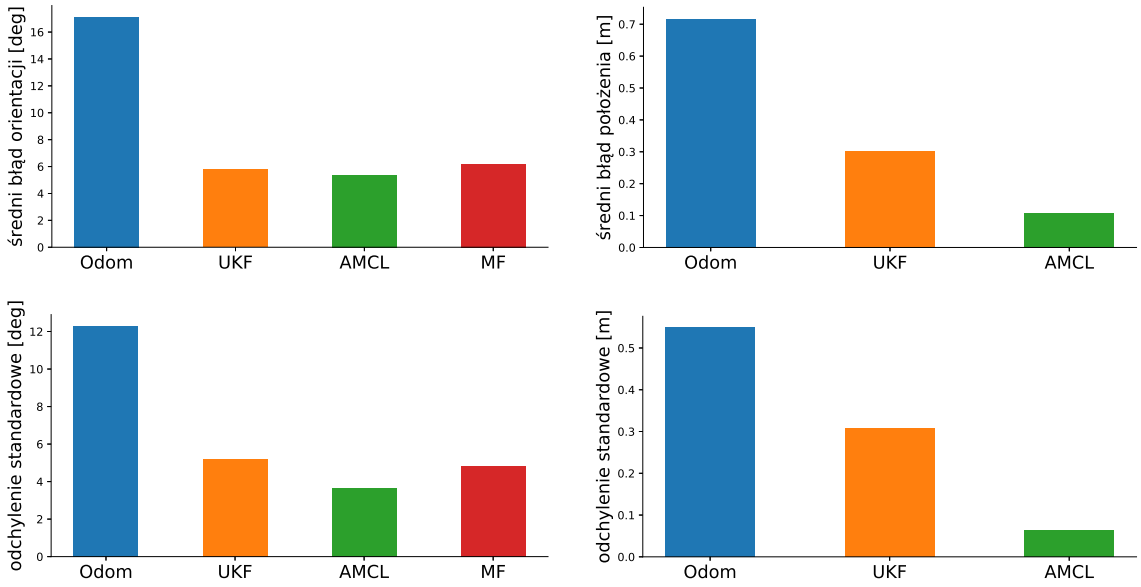
(a) robot czterokołowy

(b) robot ReMeDi

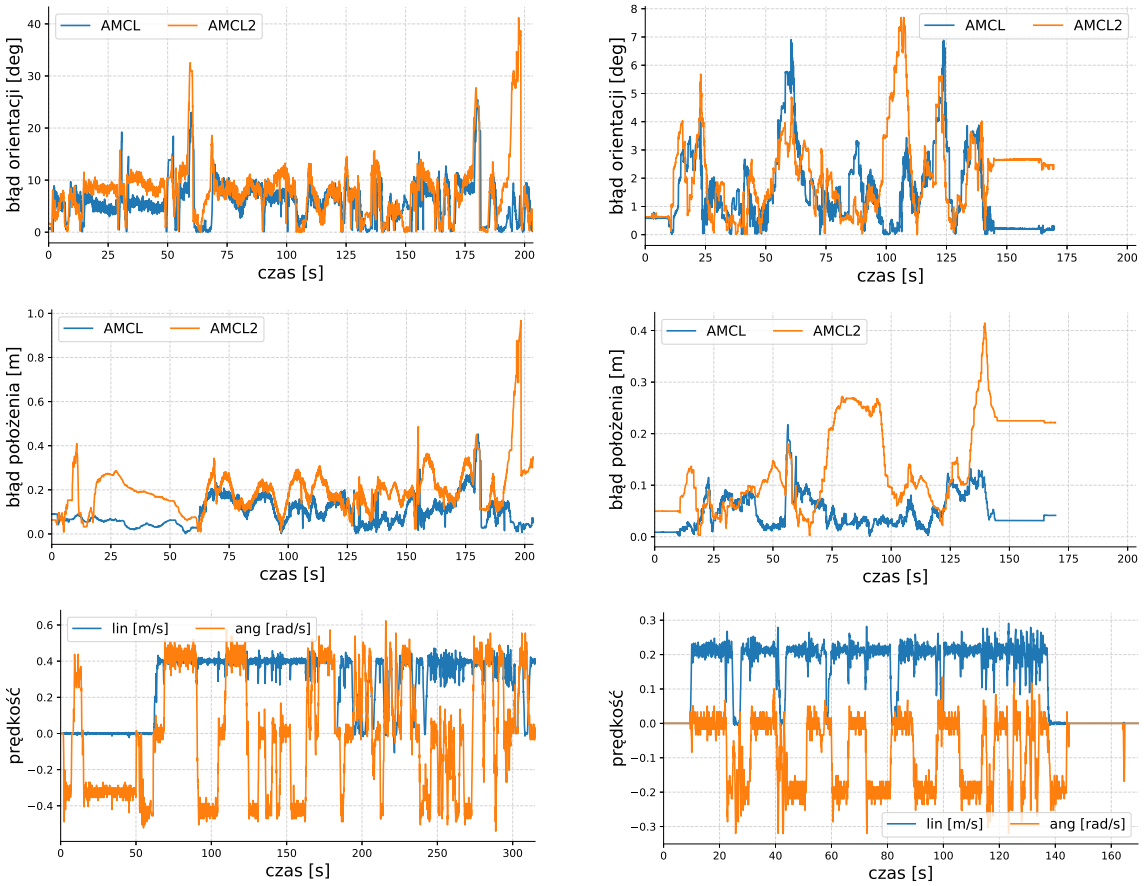
Rysunek 4.19 Porównanie błędów lokalizacji robotów o różnych konstrukcjach mechanicznych



Rysunek 4.20 Zbiorcze porównanie jakości estymacji pozycji przez metody lokalizacji na podstawie wielu przejazdów robota czterokołowego



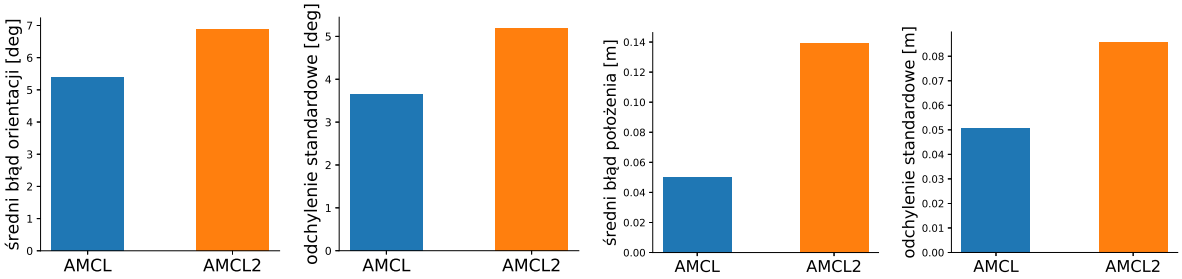
Rysunek 4.21 Zbiorcze porównanie jakości estymacji pozycji przez metody lokalizacji na podstawie wielu przejazdów robota ReMeDi



(a) przejazd I

(b) przejazd II

Rysunek 4.22 Wyniki badań eksperymentalnych algorytmu lokalizacji z danymi z lidara (AMCL) oraz sensora Kinect (AMCL2), z robotem ReMeDi



Rysunek 4.23 Porównanie średnich wartości błędów wyznaczania pozycji, obliczone na podstawie dwóch przejazdów robota ReMeDi

4.6 Podsumowanie

W niniejszym rozdziale omówiono szereg wybranych metod i systemów lokalizacji robotów mobilnych. Ponadto przedstawiono wyniki badań jakości estymacji pozycji za pomocą wybranych metod lokalizacji lokalnej i globalnej, z wykorzystaniem dwóch kołowych robotów mobilnych.

Przeprowadzone badania eksperymentalne, pozwoliły stwierdzić, że dla większości przypadków globalna metoda lokalizacji AMCL estymuje pozycję robota z najwyższą jakością, czyli z najmniejszymi wartościami błędów. Jest to w pełni zgodne z intuicją, ponieważ algorytm AMCL był jedynym globalnym algorytmem w grupie badanych metod. Zaletą tego algorytmu jest brak narastania błędów, ponieważ z określoną częstotliwością realizowana jest korekcja pozycji robota na podstawie dopasowania odczytu lidara do znanej, statycznej mapy środowiska. W przypadku pozostałych algorytmów lokalizacji, takich jak: odometria, bezśladowy filtr Kalmana (UKF), oraz filtr Madgwicka estymujący orientację, przez brak globalnego odniesienia błędy ciągle narastają.

Na wykresach błędów orientacji i położenia zaobserwowano też pewną wadę metody AMCL. Uwidacznia się ona w formie nagłego wzrostu błędów, zarówno orientacji jak i położenia. Jest to związane z błędnym dopasowaniem odczytu z lidara lub sensora RGB-D do oddalonego od robota obszaru na mapie. Zjawisko to może się nasilać wraz ze zwiększaniem wartości parametrów algorytmu AMCL ustalających poziom niepewności danych z systemu odometrycznego. Kluczem do wyeliminowania takiego zjawiska jest zapewnienie odpowiedniej jakości danych odometrycznych i określenie odpowiednio niskiego poziomu niepewności związanego z tym źródłem informacji. Niestety, nawet po takich modyfikacjach, sytuacja może się powtarzać w przypadku poślizgów kół. Pewnym rozwiązaniem jest wykorzystanie dodatkowo danych z IMU, aby wykrywać takie sytuacje i minimalizować ich wpływ na działanie algorytmu.

Z wykorzystaniem robota ReMeDi, wyposażonego zarówno w lidar Hokuyo oraz czujnik Kinect, badano także działanie metody AMCL używającej odczyty z pierwszego lub drugiego sensora. W oparciu o uzyskane wyniki (rys. 4.22), stwierdzono, że zarówno w przypadku orientacji jak i położenia robota, rezultaty AMCL działającego na danych z lidara charakteryzują się niższym poziomem błędów. Przy czym różnica jest znacznie większa w przypadku błędów położenia niż błędów orientacji. Źródeł zwiększonych błędów lokalizacji używającej czujnika Microsoft Kinect może być kilka. Jednym z nich jest mniejszy zasięg tego sensora oraz mniejszy zakres horyzontalnych kątów widzenia, który dla sensora Kinect wynosi $57°$, a w przypadku lidara Hokuyo został ograniczony przez konstrukcję mechaniczną robota i wynosił $180°$. Nie bez znaczenia pozostaje też dokładność sensora, która w przypadku lidara jest znacząco większa.

Rozdział 5

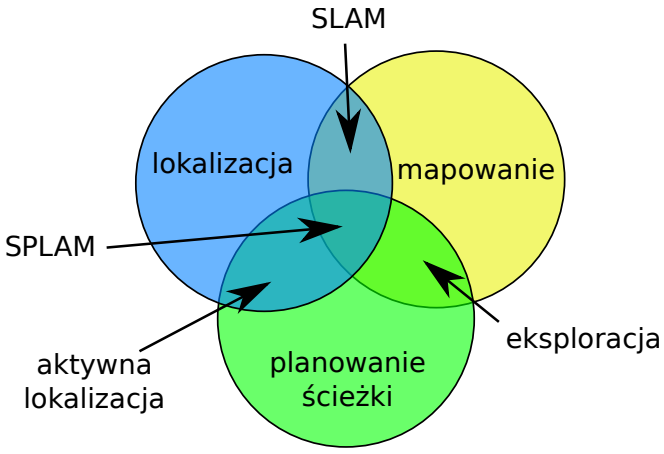
Problem SLAM

Niniejszy rozdział zawiera przegląd metod SLAM dla robotów mobilnych. Ponadto w rozdziale zaprezentowano wyniki przeprowadzonych badań eksperymentalnych dotyczących wybranych metod SLAM.

5.1 Wprowadzenie

Jednym z podstawowych zadań robotów mobilnych jest nawigacja. Jest to pojęcie dość ogólne, określające w zasadzie szereg zadań, których realizacja umożliwia robotom mobilnym autonomiczne poruszanie się w środowisku. Podążając za klasyfikacją wprowadzoną w [158] do podstawowych zadań systemu nawigacji należą (rys. 5.1):

- lokalizacja, czyli określanie pozycji robota w środowisku,
- mapowanie, na które składa się tworzenie i przechowywanie map środowiska,
- planowanie ścieżki, co oznacza wyznaczanie ścieżki, która powinna być śledzona przez robota, aby dotrzeć do punktu docelowego, unikając przy tym kolizji z przeszkodami,



Rysunek 5.1 Umiejscowienie SLAM w rozwiązaniach związanych z nawigacją robotów

- planowanie i śledzenie trajektorii, co dotyczy planowania trajektorii w krótkim horyzoncie czasowym, zgodnej z globalną ścieżką, a także generowanie sterowań zapewniających śledzenie trajektorii.

W zależności od tego, które z zadań systemu nawigacji są realizowane, rozpatruje się takie podejścia jak:

- aktywna lokalizacja,
- eksploracja,
- jednoczesna lokalizacja i mapowanie (SLAM, ang. *Simultaneous Localization And Mapping*),
- oraz jednoczesne planowanie, lokalizowanie i mapowanie (SPLAM, ang. *Simultaneous Planning, Localization And Mapping*) [37].

Zagadnienie SLAM dotyczy jednoczesnego tworzenia mapy i lokalizowania na niej robota, przy czym jest to jeden z fundamentalnych problemów robotyki mobilnej. Trudność problemu SLAM wynika z założenia, że początkowe położenie robota jest nieznane, a podczas tworzenia mapy nieznanego otoczenia wymagane jest jednoczesne estymowanie pozycji w tym otoczeniu. Problem ten jest znacznie trudniejszy do rozwiązania niż lokalizacja robota na znanej mapie lub mapowanie, gdy znana jest trajektoria robota. Algorytm SLAM zajmuje się trzema kwestiami:

- sensorami,
- przetwarzaniem danych,
- reprezentacją środowiska w postaci mapy.

Jednym z celów niniejszego rozdziału jest kategoryzacja i przedstawienie metod SLAM, jako wstęp do metod SLAM dla wielu robotów.

Probabilistyczna definicja problemu

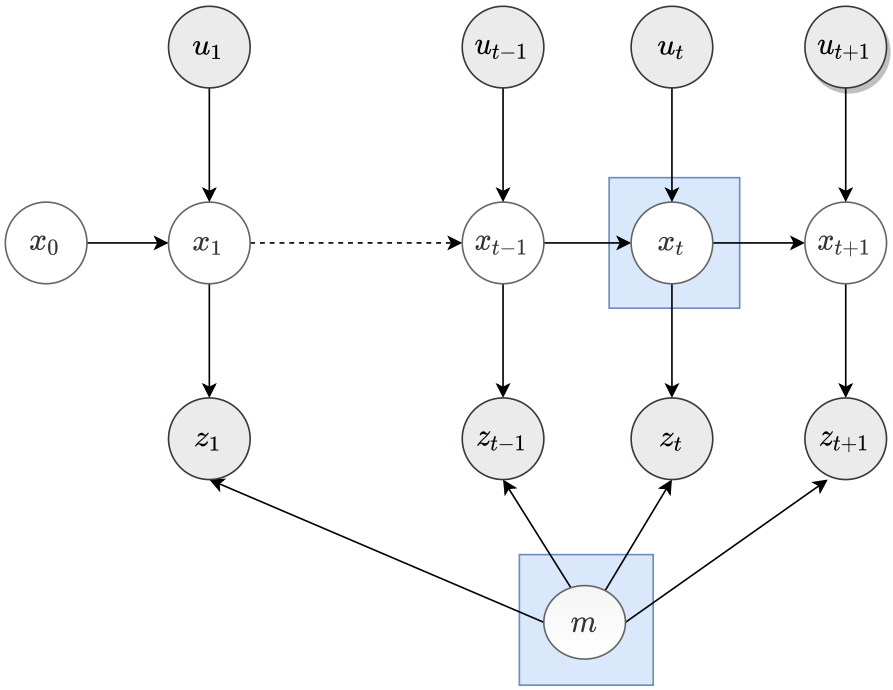
Niech $z_{1:t} = \{z_1, z_2, \ldots, z_t\}$ określa zbiór danych pomiarowych do chwili t, natomiast $u_{1:t} = \{u_1, u_2, \ldots, u_t\}$ zbiór sterowań. Problem estymacji stanu robota x_t na mapie m można zdefiniować jako poszukiwanie funkcji rozkładu prawdopodobieństwa [59, 241]

$$p(x_t, m \mid z_{1:t}, u_{1:t}). \tag{5.1}$$

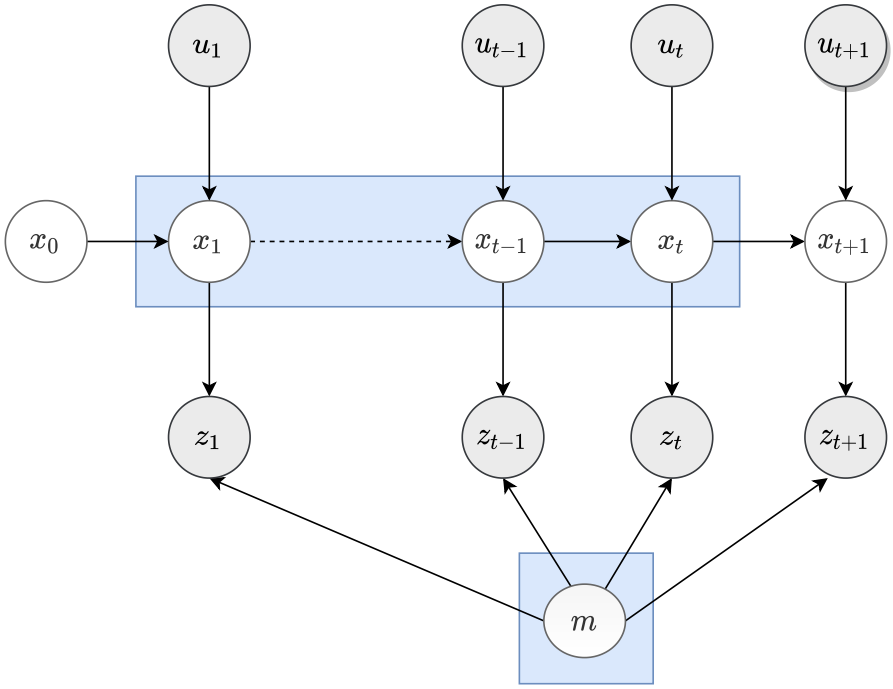
Przedstawiony problem jest określany mianem problemu SLAM online (rys. 5.2), ponieważ dotyczy estymacji stanu robota x_t jedynie w bieżącej chwili t, natomiast pomija to co działo się z robotem wcześniej, czyli jego trajektorię.

Kolejną odmianą problemu SLAM jest pełny problem SLAM (ang. *full SLAM*) przedstawiony na rys. 5.3 w postaci sieci bayesowskiej. W tym przypadku, rozwiązanie składa się z estymowanej mapy oraz pełnej trajektorii robota $x_{1:t} = \{x_1, x_2, \ldots, x_t\}$. Funkcja rozkładu prawdopodobieństwa została określona następująco

$$p(x_{1:t}, m \mid z_{1:t}, u_{1:t}). \tag{5.2}$$



Rysunek 5.2 Model sieci bayesowskiej dla problemu SLAM online



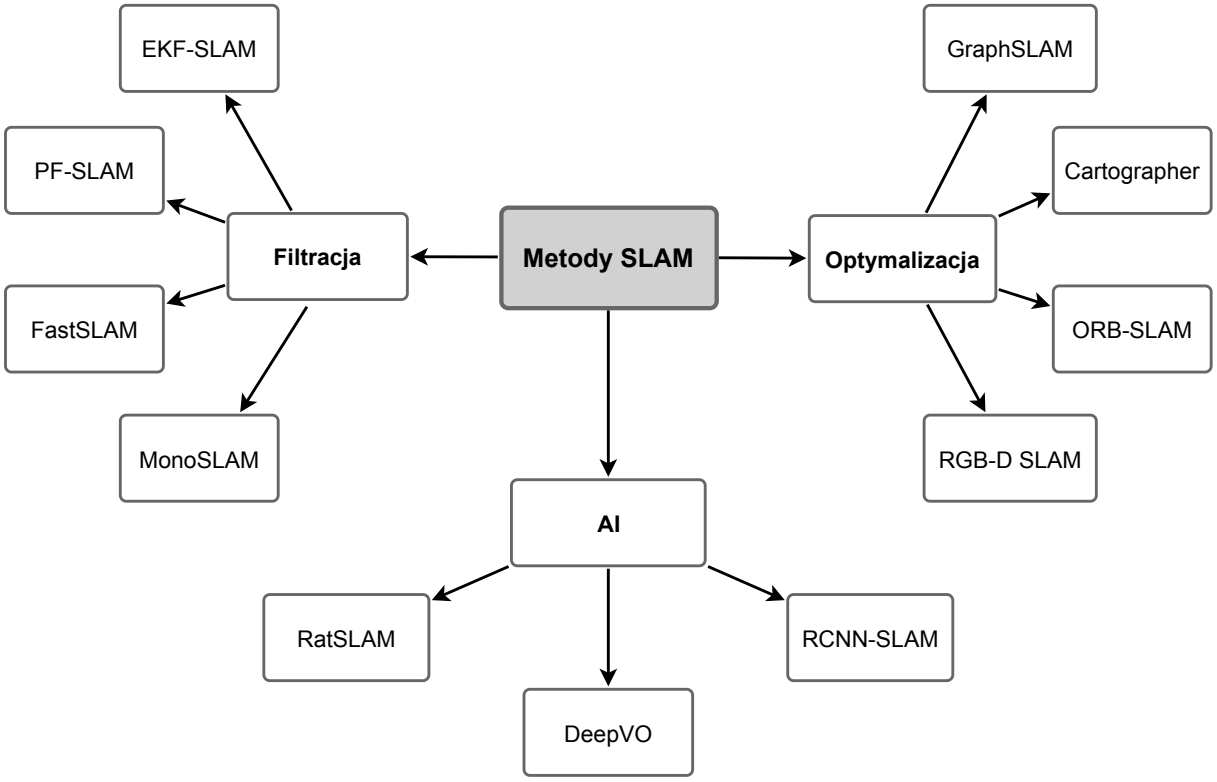
Rysunek 5.3 Model sieci bayesowskiej dla problemu full SLAM

5.2 Metody SLAM

5.2.1 Klasyfikacja metod SLAM

Biorąc pod uwagę sposób przetwarzania danych, metody SLAM można podzielić na trzy kategorie (rys. 5.4):

metody oparte na filtracji bayesowskiej - do tej grupy można zaliczyć metodę EKF-



Rysunek 5.4 Klasyfikacja metod SLAM

SLAM wykorzystującą rozszerzony filtr Kalmana, metodę EIF-SLAM (rozszerzony filtr informacyjny), czy też metodę PF-SLAM opierającą się na filtrze cząsteczkowym. A także metoda FastSLAM również oparta na filtrze cząsteczkowym, w której zredukowano liczbę cząsteczek. Popularną metodą opartą na tym samym filtrze jest też algorytm gmapping [85].

metody oparte na optymalizacji estymują całą trajektorię robota przez optymalizację procesu, przy różnego rodzaju ograniczeniach, związanych na przykład z obserwacjami. Wykorzystuje się kilka podejść opierających się na optymalizacji. Jedno z nich polega na budowie grafu pozycji (ang. *pose graph*) i jego optymalizacji. W grafie pozycji, pozycje robota reprezentowane są w postaci węzłów, a krawędzie modelują przemieszczenia i ograniczenia związane z obserwacjami. Do tej grupy należą takie metody jak GraphSLAM, Cartographer, RGB-D SLAM [60], czy metody grafowe zawarte w SLAM Toolbox [153].

Kolejne podejście to dopasowanie wiązki (BA, ang. *Bundle Adjustment*), które opiera się na jednoczesnej optymalizacji zbioru pozycji kamery (sensora) i widzianych punktów (obserwacji). Jest to podstawowa metoda do rozwiązywania problemu rekonstrukcji trójwymiarowej struktury środowiska na podstawie zbioru obrazów 2D (SFM, ang. *structure-from-motion*) [260].

Kolejną techniką opartą na optymalizacji jest dopasowywanie submap. W tej metodzie niewielkie lokalne mapy są łączone z innymi w celu utworzenia jednej globalnej mapy. Metoda łączenia submap jest podobna do metod łączenia map w systemach wielorobotowych.

metody bazujące na sztucznej inteligencji (AI, ang. *Artificial Intelligence*) można podzielić na dwie grupy, w zależności od tego, czy wykorzystują uczenie nadzorowa-

ne, czy nienadzorowane. Do pierwszej grupy należy metoda DeepVO przedstawiona w pracy [168]. Metoda ta wykorzystuje dane z pojedynczej kamery i opiera się na sieciach AlexNet oraz końcowej warstwie konwolucyjnej. W pracy [252] również przedstawiono rozwiązanie dla pojedynczej kamery wizyjnej, oparte na głębokim uczeniu z rekurencyjnymi, konwolucyjnymi sieciami neuronowymi (RCNN, ang. *Recurrent Convolutional Neural Networks*).

Do grupy metod opartych na AI można też zaliczyć metodę RatSLAM [17,166,167, 172] przedstawioną przez Milforda w roku 2004. Metoda opiera się na sieciach neuronowych wykorzystywanych do modelowania mózgu gryzoni, a konkretniej hipokampu. Sieć neuronowa została wykorzystana do fuzji danych z kamery i systemu odometrycznego. Rozwinięciem metody RatSLAM jest metoda VitaSLAM [231], która oprócz danych wizyjnych wykorzystuje również informacje taktylne, które w przypadku gryzoni zbierane są z otoczenia za pomocą wąsów.

Metody SLAM można też podzielić ze względu na typ wykorzystywanych informacji. Przede wszystkim stosowane są kamery wizyjne oraz różnego rodzaju czujniki odległości, jak skanery laserowe, kamery ToF, czy systemy stereowizyjne. Na potrzeby SLAM wykorzystuje się też dane z systemów odometrycznych i jednostek inercyjnych (IMU). W przypadku, gdy roboty poruszają się na zewnątrz, używane są dodatkowo systemy globalnej nawigacji jak GPS.

Znaczącą grupę metod SLAM stanowią metody wizyjne, czyli opierające działanie na danych z systemów kamer. Wyróżnia się dwa podejścia w takich metodach: detekcja i dopasowanie cech lub bezpośrednie wykorzystanie obrazu.

Metody oparte na detekcji cech używają różnego rodzaju detektorów do wykrycia cech i śledzenia ich między kolejnymi ramkami danych. Jednym z pierwszych algorytmów wykorzystujących detekcję i dopasowanie cech, dostosowanych do działania z pojedynczą kamerą był algorytm MonoSLAM [44] opracowany w roku 2007. Algorytm ten wykorzystuje filtrację za pomocą filtra EKF. Filtr EKF używany jest do estymacji zarówno ruchu kamery, jak i struktury środowiska. Pozycja kamery jak i zestaw wykrywanych cech stanowią wektor stanu w filtrze EKF, przy czym w trakcie działania dodawane są nowe cechy. Problemem tego rozwiązania jest skalowalność, wynikająca z dużego rozmiaru wektora stanu.

Rozwiązanie tego problemu pojawiło się w 2007 roku, w metodzie PTAM [76, 128], gdzie rozdzielono mapowanie i śledzenie na dwa odrębne zadania. Dodatkowo G. Klein i D. Murray po raz pierwszy w wizyjnej metodzie SLAM, wykorzystali metodę dopasowania wiązki (BA) do optymalizacji. Następnie BA zostało wykorzystane w metodzie ORB-SLAM [173].

Odmiennym podejściem są metody bezpośrednie, w których pomija się detekcję cech i bezpośrednio wykorzystuje dane z kamer, a dokładniej z poszczególnych pikseli, w celu optymalizacji problemu estymacji trajektorii kamery. Określane jest to również mianem minimalizacji błędu fotometrycznego. Metoda ta pozwala wyeliminować koszt ekstrakcji cech, ale zwiększa rozmiar problemu optymalizacyjnego [58]. Do metod bezpośrednich można zaliczyć metodę LSD-SLAM (ang. *Large-Scale Direct Monocular SLAM*) [61].

W dalszej części przedstawiono opis wybranych metod SLAM.

5.2.2 EKF-SLAM

Jednym ze sposobów rozwiązania problemu SLAM jest zastosowanie rozszerzonego filtru Kalmana (EKF), analogicznie jak przedstawiono to dla problemu lokalizacji. W

metodzie EKF-SLAM, mapa jest wektorem zawierającym stany znaczników i modelowana jest gaussowską zmienną losową. Mapa utrzymywana jest w procesie predykcji (ruch sensorów) i korekcji (obserwacja wcześniej utworzonych znaczników).

Wektor stanu

$$x = \begin{bmatrix} \mathcal{R} \\ m \end{bmatrix} \tag{5.3}$$

obejmuje zarówno stan robota $\mathcal{R}$, w postaci położenia i orientacji, ale też stan otoczenia w postaci mapy będącej zbiorem n znaczników

$$m = [\mathcal{L}_1, \mathcal{L}_2, \ldots, \mathcal{L}_n]^T. \tag{5.4}$$

Wektor stanu x modelowany jest przez zmienną losową o rozkładzie Gaussa, o wartości średniej $\overline{x}$ oraz macierzy kowariancji

$$P = \begin{bmatrix} P_{\mathcal{R}\mathcal{R}} & P_{\mathcal{R}m} \\ P_{m\mathcal{R}} & P_{mm} \end{bmatrix}. \tag{5.5}$$

Podobnie jak w przypadku lokalizacji, działanie metody opiera się na dwóch operacjach: predykcji i korekcji. Krok predykcji wygląda następująco:

$$\overline{x} = f(\overline{x}, u, n), \tag{5.6}$$
$$P = F_x P F_x^T + F_n N F_n^T. \tag{5.7}$$

przy czym F_x oraz F_n to jakobiany, a N macierz kowariancji.

Krok korekcji:

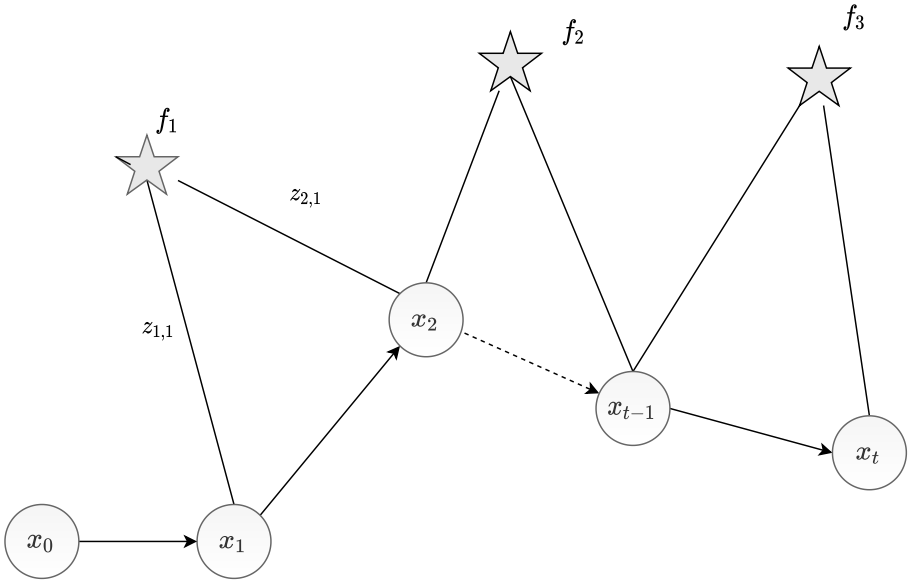
$$\overline{z} = y - h(\overline{x}), \tag{5.8}$$
$$Z = H_x P H_x^T + R, \tag{5.9}$$
$$K = P H_x^T Z^{-1}, \tag{5.10}$$
$$\overline{x} = \overline{x} + K\overline{z}, \tag{5.11}$$
$$P = P - KZK^T, \tag{5.12}$$

gdzie $H_x = \dfrac{\partial h(\overline{x})}{\partial x}$ jest jakobianem, R określa macierz kowariancji pomiaru, a przez K oznacza się wzmocnienie Kalmana.

5.2.3 GraphSLAM

Metoda pozwala rozwiązać problem full SLAM. Przechowuje w tym celu pełną trajektorię, czyli historię wszystkich położeń robota. Zasada działania metody polega na budowie grafu pozycji i poszukiwaniu takiej konfiguracji węzłów, która minimalizuje funkcję celu przy zadanych ograniczeniach. Kluczowe cechy:

- wykorzystuje graf pozycji $G = (E, V)$, gdzie E to krawędzie, a V wierzchołki, do reprezentacji problemu (rys. 5.5),
- każdy węzeł grafu x_i odpowiada pozycji robota podczas mapowania,
- każda krawędź grafu między dwoma węzłami odpowiada relacji przestrzennej między nimi,



Rysunek 5.5 Graf zawierający pozycje robota, cechy środowiska i relacje między nimi

- obserwacja poprzednio widzianych cech środowiska pozwala wygenerować ograniczenia między węzłami,
- błąd jest definiowany jako różnica między przewidywanym, a aktualnym pomiarem odległości do opisanego punktu środowiska f_i

$$e_i(x) = z_i - f_i(x), \tag{5.13}$$

- zakłada się, że błąd pomiaru ma zerową wartość średnią i rozkład gaussowski,
- kwadrat błędu pomiaru zależy tylko od stanu

$$e_i(x) = e_i(x)^T \Omega_i e_i(x), \tag{5.14}$$

gdzie Ω_i jest macierzą informacji.

Minimalizacja funkcji celu polega na znalezieniu takiej wartości x, która minimalizuje błąd wszystkich pomiarów

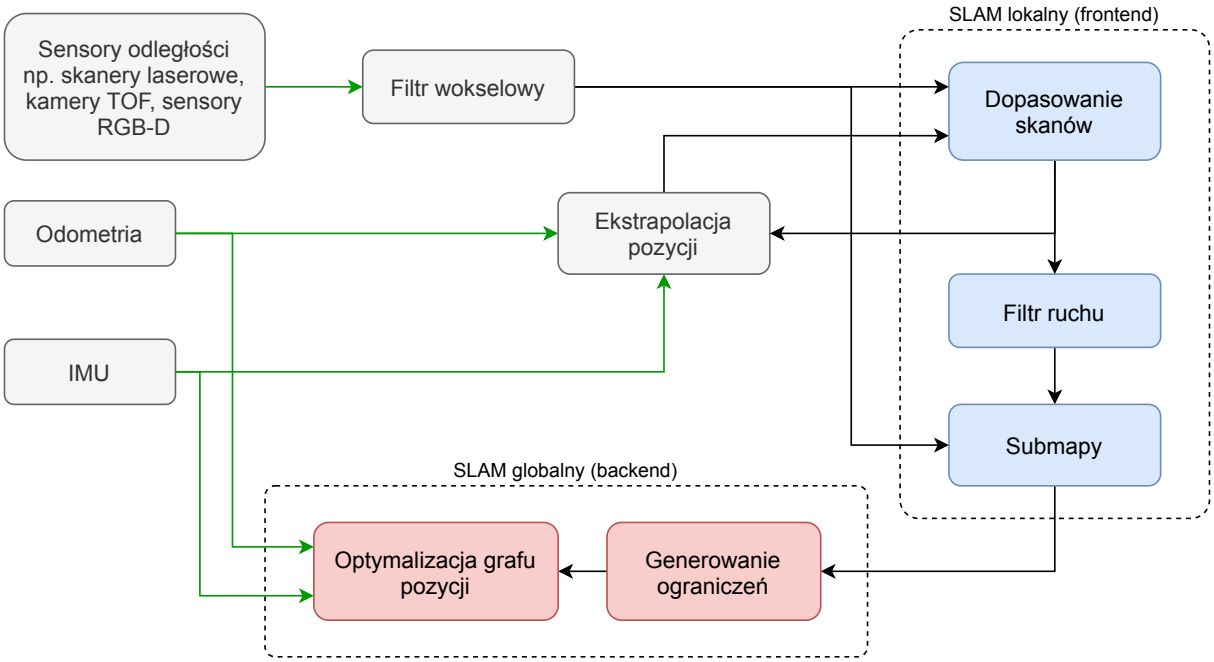
$$x^* = \arg\min_x F(x) = \arg\min_x \sum_i e_i(x)^T \Omega_i e_i(x). \tag{5.15}$$

5.2.4 Cartographer

Cartographer [81, 92, 177] jest to metoda SLAM działająca zarówna w 2D jak i 3D. Rozwijana jest jako oprogramowanie open-source, głównie przez Google. Rozwiązanie to opiera się na podejściu grafowym, jednak wprowadza kilka modyfikacji. Metoda Cartographer zyskała znaczną popularność, ze względu na swoją efektywność, a także dostępność szeregu narzędzi służących do dobierania wartości parametrów, a także uruchamiania algorytmu offline na podstawie zebranych danych. Projekt Cartographer został również zintegrowany ze środowiskiem ROS.

Architektura rozwiązania została przedstawiona na rys. 5.6. W metodzie można wyróżnić dwie części:

- lokalną, zwaną też frontendem,
- oraz globalną, określaną mianem backendu.



Rysunek 5.6 Architektura metody Cartographer

Zadaniem części lokalnej jest tworzenie spójnych submap (wycinków map), natomiast zadaniem części globalnej jest znalezienie takich transformacji między poszczególnymi submapami, aby były spójne globalnie. W założeniu, że poszczególne submapy są spójne lokalnie, wykorzystuje się fakt, że w krótkim horyzoncie czasowym dryf pozycji robota jest stosunkowo niewielki.

Lokalny SLAM

Jak już wspomniano, część lokalna, określana też mianem generatora lokalnych trajektorii, odpowiada za tworzenie kolejnych submap. W pierwszej kolejności dopasowywane są do siebie kolejne skany, czyli odfiltrowane odczyty z sensorów odległości. Metoda dopasowywania skanów (ang. *scan matcher*) [180] działa w czasie rzeczywistym dzięki wykorzystaniu obliczeń równoległych. Jako rozwiązanie początkowe przyjmowana jest pozycja ekstrapolowana na podstawie danych odometrycznych oraz danych z IMU.

Dopasowanie skanów do siebie pozwala określić przemieszczenie robota w czasie, który upłynął między zebraniem pierwszego i drugiego skanu. Obliczone wartości przemieszczenia robota przekazywane są do filtra ruchu, który określa, czy należy wykonać aktualizację submap. Jeżeli przemieszczenie przekracza pewną wartość progową, wtedy następuje aktualizacja aktualnej submapy, przez dodanie do niej skanu, w miejscu określonym na podstawie wspomnianego przemieszczenia robota. Przyjmuje się, że submapa jest skończona w momencie, gdy zostanie do niej dodana ustalona liczba skanów. Submapy dodawane są do grafu, którego węzły określają również kolejne pozycje robota.

Globalny SLAM

Część globalna, czyli backend optymalizuje przemieszczenia między submapami, aby uzyskać spójną globalnie mapę. Optymalizacja odpowiada też za rozwiązanie problemu domkniętej pętli. Generalnie, działanie części globalnej jest zbliżone do metody GraphSLAM, czyli polega na optymalizacji grafu pozycji. Do grafu dodawane są też ograniczenia wykorzystywane w procesie optymalizacyjnym. Ograniczenia dzielą się na dwie grupy: lokalne i globalne. Ograniczenia należące do pierwszej grupy, generowane są między węzłami, które znajdują się blisko siebie, więc mają charakter lokalny.

W drugiej grupie ograniczenia mają charakter globalny i są dodawane w momencie pojawienia się nowej submapy. Odpowiedni algorytm przeszukuje wszystkie dotychczasowe submapy i w przypadku znalezienia podobieństwa, generuje ograniczenia. Globalne ograniczenia umożliwiają rozwiązywanie problemu zamykania pętli.

Problem optymalizacji można zdefiniować następująco [92]

$$x^* = \arg\min_x F(x) = \arg\min_{\Xi^m, \Xi^s} \frac{1}{2} \sum_{ij} \rho \left(E^2(\xi_i^m, \xi_j^s, \Sigma_{ij}, \xi_{ij}) \right), \tag{5.16}$$

gdzie ρ definiuje funkcję strat, $\Xi^s = \{\xi_i^m\}, i \in [1, m]$ określa pozycje submap, a $\Xi^s = \{\xi_j^s\}, j \in [1, n]$ to zbiór pozycji poszczególnych skanów. Ograniczenie ξ_{ij} dla submapy i oraz skanu j określa gdzie w układzie lokalnym submapy został dopasowany dany skan. Pozostałe ograniczenia wyrażone są w formie macierzy kowariancji Σ_{ij}. Dodatkowo E można obliczyć następująco

$$E^2(\xi_i^m, \xi_j^s, \Sigma_{ij}, \xi_{ij}) = e(\xi_i^m, \xi_j^s, \xi_{ij})^T \Sigma_{ij}^{-1} e(\xi_i^m, \xi_j^s, \xi_{ij}), \tag{5.17}$$

$$e(\xi_i^m, \xi_j^s, \xi_{ij}) = \xi_{ij} - \begin{bmatrix} R_i^{-1}(T_i - T_j) \\ \xi_i^m - \xi_j^s \end{bmatrix}, \tag{5.18}$$

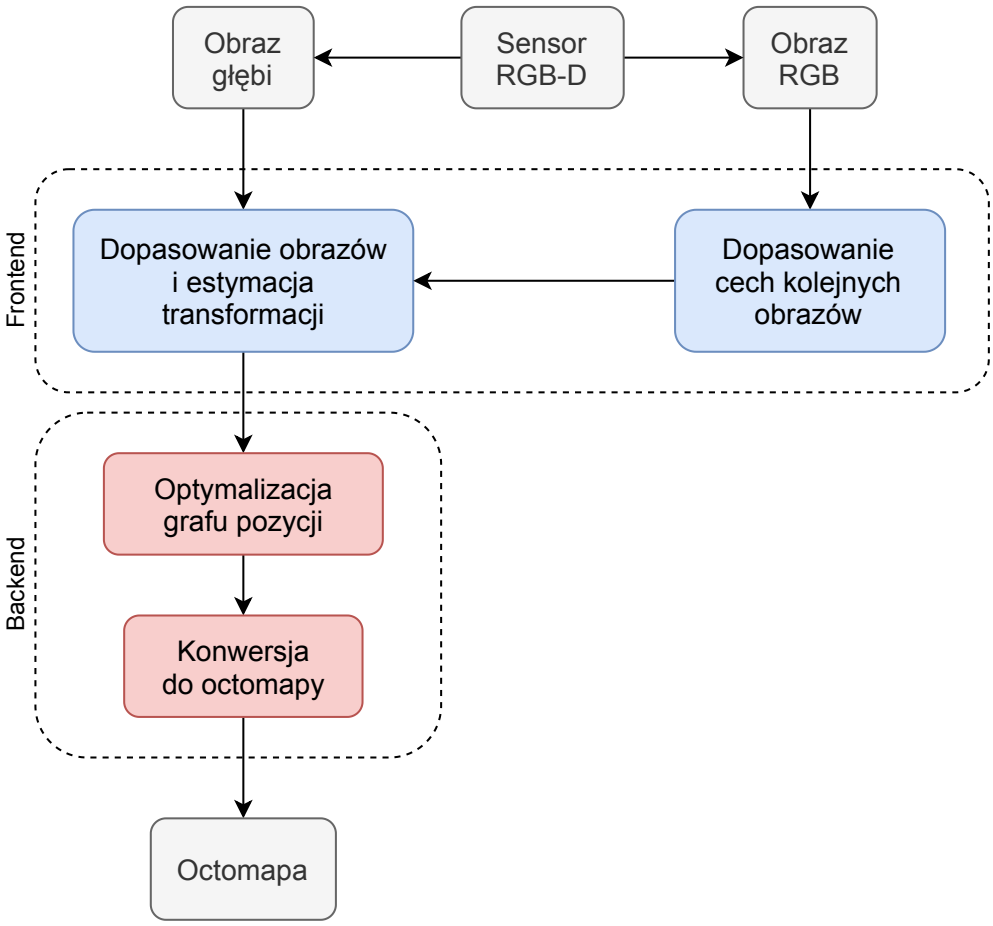
gdzie R_i określa macierz rotacji, a T_i oraz T_j wektory translacji. Do rozwiązania powyższego problemu wykorzystywane jest narzędzie o otwartych źródłach Ceres [6] przeznaczone do optymalizacji nieliniowych.

5.2.5 RGBD-SLAM

Metoda RGB-D SLAM [60] opiera się na detekcji i dopasowywaniu cech na podstawie danych z czujników RGB-D. Umożliwia tworzenie map i lokalizację w trzech wymiarach. Rozwiązanie składa się z dwóch części (rys. 5.7): frontendu i backendu. Frontend odpowiada za estymację przemieszczenia między dwoma kolejnymi obrazami z czujnika w oparciu o dopasowanie cech. Backend odpowiada natomiast za tworzenie i optymalizację grafu pozycji z uwzględnieniem ograniczeń związanych z występowaniem pętli w środowisku.

Estymacja transformacji między parami kolejnych obrazów RGB

W tym celu na obrazach RGB wykrywane są cechy, które następnie są opisywane z użyciem deskryptorów SURF, SIFT lub ORB [207]. Operacje powtarzane są na dwóch kolejnych obrazach, co umożliwia określenie transformacji między nimi. Tak wykryte cechy są rzutowane na obraz 3D otrzymany na podstawie pomiaru głębi. Następnie



Rysunek 5.7 Schemat blokowy metody RGB-D SLAM

cechy są do siebie dopasowywane z wykorzystaniem algorytmu RANSAC. Pozwala to przyspieszyć obliczenia, ale też zmniejszyć wpływ niejednoznaczności deskrypcji cech na proces estymacji transformacji między kolejnymi obrazami.

Graf pozycji

Graf pozycji $G = (E, V)$ składa się ze zbioru wierzchołków V i zbioru krawędzi E. Wierzchołki grafu to pozycje sensora w momencie wykonywania kolejnych obrazów, natomiast transformacje między parami obrazów tworzą krawędzie grafu. Ponieważ graf nie zawsze jest spójny, chociażby przez błędy dopasowań cech, przeprowadzana jest jego optymalizacja. Proces ten polega na minimalizacji nieliniowej funkcji błędu

$$F(x) = \sum_{i,j \in C} e(x_i, x_j, z_{ij})^T \Omega_{ij} e(x_i, x_j, z_{ij}), \tag{5.19}$$

przy czym $x = (x_1^T, \ldots, x_n^T)^T$ jest wektorem pozycji, a z_{ij} oraz Ω_{ij} to macierze ograniczeń związanych z pozycjami x_j. Przez $e(x_i, x_j, z_{ij})$ wyraża się wektor błędu, który określa jak bardzo x_i oraz x_j spełniają ograniczenie z_{ij}. Poszukiwane rozwiązanie, które minimalizuje funkcję błędu ma postać

$$x^* = \arg\min_x F(x). \tag{5.20}$$

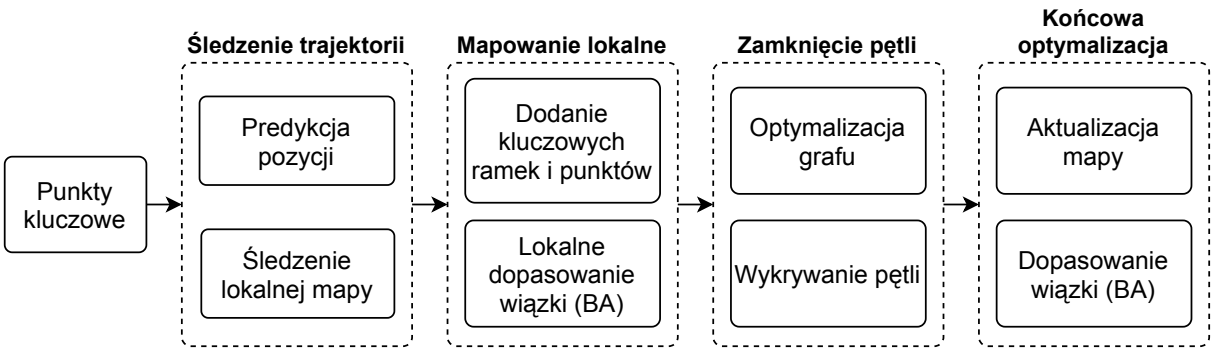
5.2.6 ORB-SLAM

Kolejne rozwiązanie problemu SLAM zostało przedstawione w pracach [173, 174], w postaci metody ORB-SLAM opierającej się na ekstrakcji, deskrypcji i dopasowaniu cech. Nazwa metody pochodzi od deskryptorów, które wykorzystuje. Są to zorientowane, szybkie i obrócone deskryptory BRIEF, czyli ORB (ang. *Oriented FAST and Rotated BRIEF*) [207]. Przy czym, BRIEF (ang. *Binary Robust Independent Elementary Features*) [103] jest rozwijanym wcześniej typem wydajnego deskryptora cech, stanowiącym bazę dla ORB. W celu detekcji cech autorzy wykorzystali detektor FAST (ang. *Features from Accelerated Segment Test*), który umożliwia działanie metody w czasie rzeczywistym. Część optymalizacyjna metody opiera się na algorytmie dopasowania wiązki (BA), przez co nie ma założenia o sekwencyjności rejestrowanych obrazów.

Pierwsza wersja [173] metody ORB-SLAM przeznaczona jest dla systemów monowizyjnych, natomiast druga wykorzystuje dane z systemów stereowizyjnych lub sensorów RGB-D. Znaczna popularność metody wynika z jej skuteczności [58]. Zaletą systemu ORB-SLAM jest brak potrzeby stosowania dodatkowych sensorów, takich jak enkodery, czy IMU, ponieważ przemieszczenie określane jest na podstawie estymowanych transformacji między kolejnymi obrazami.

Na rys. 5.8 przedstawiono komponenty wchodzące w skład systemu ORB-SLAM. System można zdekomponować na cztery główne części:

- śledzenie trajektorii,
- mapowanie lokalne,
- zamykanie pętli,
- końcowe optymalizacje.



Rysunek 5.8 Architektura rozwiązania ORB-SLAM

System realizuje optymalizację na trzech poziomach, z wykorzystaniem metod dopasowania wiązki (BA) [174]:

- optymalizacja pozycji sensora, w komponencie śledzenia trajektorii,
- lokalne BA do optymalizacji punktów i ramek kluczowych (ang. *keyframes*) na potrzeby komponentu mapowania,
- pełne BA do optymalizacji wszystkich ramek ramek kluczowych i punktów.

W pierwszym przypadku metoda BA optymalizuje macierz orientacji sensora $R \in SO(3)$ oraz wektor translacji $t \in \mathbb{R}^3$, przez minimalizację błędu między dopasowywanymi punktami $X^i \in \mathbb{R}^3$ i punktami kluczowymi $x_s^i \in \mathbb{R}^3$. Funkcja optymalizacyjna przyjmuje postać

$$R, t = \arg\min_{R,t} \sum_{i \in \chi} \rho \left(\left\| x_s^i - \pi_s(RX^i + t) \right\|_\Sigma^2 \right), \tag{5.21}$$

gdzie χ określa zbiór wszystkich dopasowań, ρ jest funkcją strat. Funkcję projekcji π_s dla przypadku stereowizyjnego definiuje się następująco:

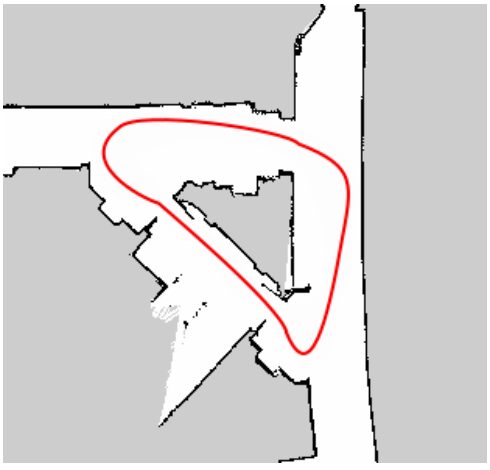
$$\pi_s = \begin{bmatrix} f_x \dfrac{X}{Z} + c_x \\ f_y \dfrac{Y}{Z} + c_y \\ f_x \dfrac{X - b}{Z} + c_x \end{bmatrix}, \tag{5.22}$$

gdzie f_x oraz f_y to długości ogniskowej, c_x i c_y określają środek optyczny sensora, a b bazę, czyli odległość między środkami kamer w układzie stereowizyjnym.

5.2.7 Problem zamykania pętli

Podczas budowania map, poruszający się robot podlega wpływowi wielu czynników. Może się zderzyć z dowolnym obiektem otoczenia, może wpaść w poślizg lub jego pozycja może zostać zmieniona przez siły zewnętrzne. Przykładem może być przestawienie przez człowieka autonomicznego odkurzacza w trakcie pracy. Dlatego też wyznaczanie pozycji robota jedynie w oparciu o przemieszczenia w kolejnych ramkach czasowych powoduje, że z czasem ten błąd narasta. Nawet jeżeli robot korzysta z dodatkowych systemów, jak określanie przemieszczenia na podstawie danych z IMU, ale nie posiada zewnętrznego systemu odniesienia, błąd określania pozycji będzie narastał.

Istnieją metody pozwalające na korekcję narastających błędów, przy założeniu, że pewne obiekty otoczenia są statyczne, jak na przykład ściany, nieruchome meble, czy budynki lub drzewa, w przypadku poruszania się na zewnątrz. Realizowane jest to w ten sposób, że wykrywa się pętle (rys. 5.9) w trajektorii robota przez detekcję, czy ro-



Rysunek 5.9 Przykład pętli w środowisku

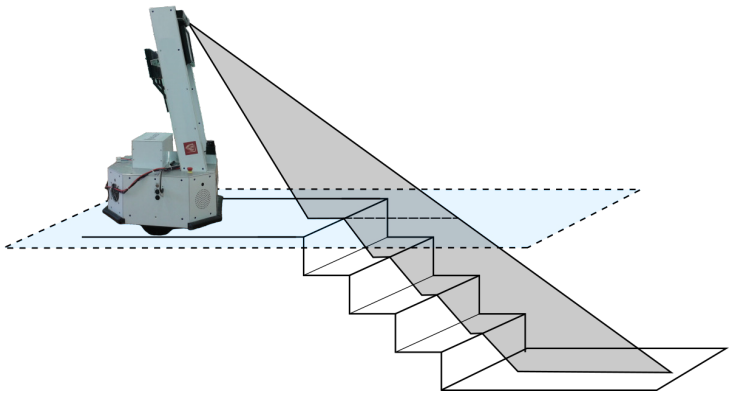
bot powrócił do miejsca, w którym był wcześniej. Problem ten jest też określany mianem detekcji cyklu. Detekcja pętli umożliwia w rezultacie optymalizację mapy, czyli poprawę jej jakości [140, 205]. Zamknięcie pętli jest kluczowym czynnikiem zapewniającym spójność tworzonej mapy. Istnieje kilka możliwości, aby dowiedzieć się, czy robot wrócił w to samo miejsce:

- porównanie bieżącego obrazu z czujnika z poprzednimi obrazami,
- porównanie bieżącego obrazu z mapą,
- porównanie aktualnej pozycji na mapie z innymi obszarami mapy.

5.3 Mapowanie z przeszkodami wklęsłymi

5.3.1 Detektor przeszkód wklęsłych

Na potrzeby tworzenia map wewnątrz budynków opracowano i zaimplementowano detektor przeszkód wklęsłych. Detektor umożliwia wykrywanie przeszkód wklęsłych, takich jak: schody w dół, uskoki, oraz ubytki podłoża (5.10).



Rysunek 5.10 Wykrywanie schodów przez robota wyposażonego w sensor głębi

Detektor wymaga wartości wysokości h środka optycznego sensora głębi oraz jego kąta pochylenia α w odniesieniu do podłoża. Wykrywanie przeszkód opiera się na wyszukiwaniu punktów obrazu głębi, dla których odległość z jest większa od wartości progowej, określającej odległość do płaszczyzny podłoża, z pewną tolerancją ε_g. Wstępny zbiór punktów przeszkód wklęsłych zdefiniowano następująco

$$P_c = \left\{ (x,z) \mid z \geq h\frac{\sin\left(\frac{\pi}{2}-\delta\right)}{\cos\left(\frac{\pi}{2}-\delta-\alpha\right)} + \varepsilon_g \right\}, \tag{5.23}$$

gdzie

$$\delta = \theta\frac{j - c_y - \frac{1}{2}}{n-1}, \tag{5.24}$$

a θ oznacza pionowy kąt widzenia czujnika, c_y określa współrzędną osi y środka obrazu głębi, n liczbę rzędów obrazu, a j rząd obrazu dla aktualnie przetwarzanego punktu.

Ze względu na występowanie zakłóceń, a także w celu zmniejszenia złożoności obliczeniowej algorytmu, obraz głębi dzielony jest na kwadratowe bloki b_{ij} o rozmiarach

b_s. W każdym z bloków obliczana jest liczba punktów P_c należących do przeszkód wklęsłych, gdzie i, j określają numer bloku

$$n_{ij} = \#\{P_c \mid i, j \in b_{ij}\}. \tag{5.25}$$

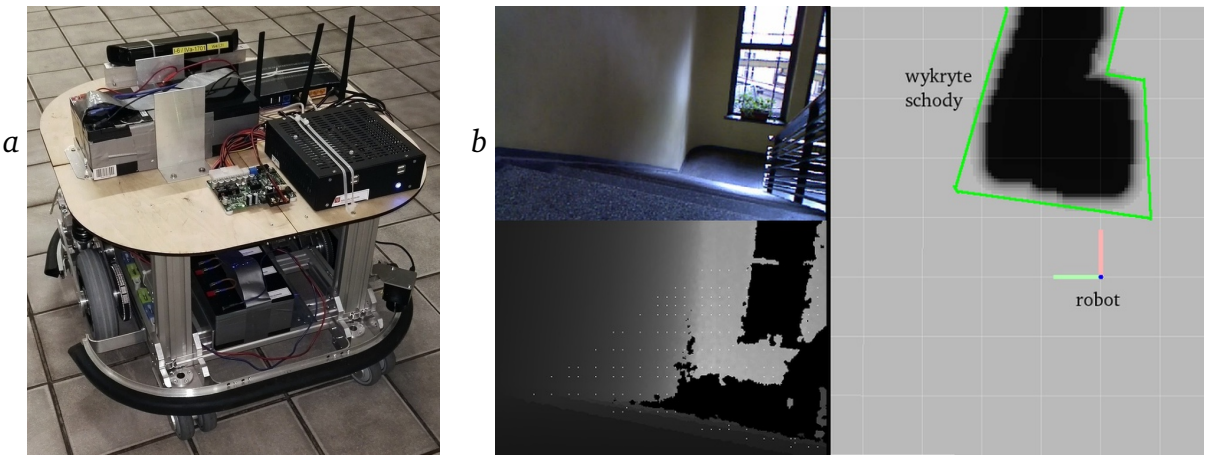
Jeżeli w danym bloku liczba punktów przeszkód przekroczy wartość progową b_{th}, wtedy wszystkie punkty należące do bloku uznawane są jako przeszkody. Ostatecznie, zbiór punktów przeszkód wklęsłych określa się jako

$$P_{obs} = \{b_{ij} \mid n_{ij} \geq n_{th}\}. \tag{5.26}$$

5.3.2 Eksperymenty

Przeprowadzono eksperymenty związane z detektorem przeszkód wklęsłych. W tym celu wykorzystano prototypową wersję robota kołowego ReMeDi [196] wyposażonego w sensor RGB-D (rys. 5.11a).

Na rys. 5.11b przedstawiono rezultaty eksperymentów, w postaci dwóch obrazów: RGB oraz głębi. Na obrazie głębi znajdują się naniesione przeszkody (białe piksele) oraz



Rysunek 5.11 Prototypowa wersja robota ReMeDi (a) oraz przykład detekcji schodów w postaci obrazu z kamery RGB robota, obrazu głębi i mapy kosztów z naniesioną przeszkodą (b)

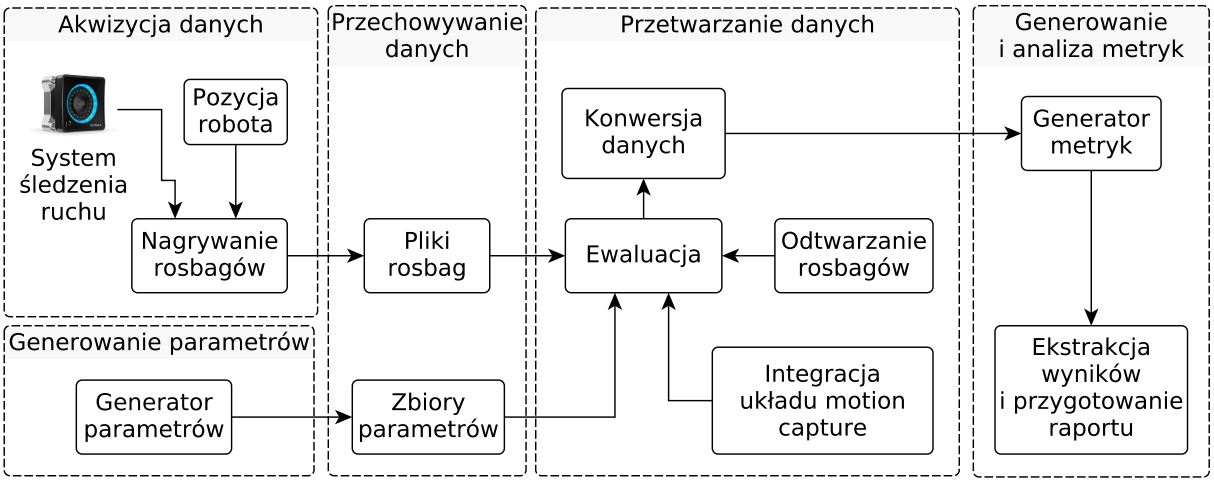
wykryte schody umieszczone na mapie kosztów. W trakcie testów czujnik Kinect zamontowany był na robocie na wysokości około $0,5$ m i pochylony do poziomu o około $15°$. Przy tak umieszczonym czujniku głębi można było z powodzeniem wykrywać przeszkody wklęsłe. Przedstawione wyniki zostały wcześniej opublikowane w pracy [52].

5.4 Automatyczny dobór parametrów metod SLAM

Na potrzeby doboru parametrów algorytmów SLAM opracowano system umożliwiający przeszukiwanie przestrzeni wartości parametrów rozbudowanych metod SLAM i porównywanie efektywności tych metod. Powodem opracowania takiego systemu były trudności w manualnym strojeniu metod SLAM, których efektywność zależy od wielu, często kilkudziesięciu parametrów.

Architektura systemu została przestawiona na rys. 5.12, a działanie opiera się na mechanizmach dostępnych w oprogramowaniu ROS.

Proces doboru parametrów składa się z kilku kroków:



Rysunek 5.12 Dobór parametrów metod SLAM

- stworzenia bazy danych,
- określenia zakresu parametrów,
- ewaluacji,
- oraz wygenerowania metryk i selekcji rozwiązań.

Tworzenie bazy danych

Najbardziej czasochłonnym zadaniem jest stworzenie bazy danych. Wymagany jest też do tego zewnętrzny system lokalizacji absolutnej, który będzie pełnił rolę referencyjną. W pracy wykorzystywano wizyjny system śledzenia ruchu (ang. *motion capture*), ale może to być też system radiowy lub inny pozwalający estymować pozycję robota w zewnętrznym układzie odniesienia. Tworzenie bazy danych polega na realizacji wielokrotnych przejazdów robota, w zasięgu systemu referencyjnego. Zbierane dane zapisywane są w postaci jednego z mechanizmów środowiska ROS - plików *rosbag*. Z każdego przejazdu powstaje *rosbag*, zawierający dane z czujników robota (kamer, czujników odległości, systemu odometrycznego, IMU i innych), transformacje między układami współrzędnych związanych z poszczególnymi elementami robota, a także informacje o pozycji w danej chwili z systemu referencyjnego.

Ważne jest aby przejazdy były możliwie zróżnicowane, tak aby zminimalizować ryzyko strojenia metody jedynie w niewielkim zakresie jej działania. Opisany system został wykorzystany na potrzeby strojenia kilku algorytmów SLAM, w tym: gmapping, hector i cartographer.

Określanie zakresu parametrów

Działanie metody opiera się na sekwencyjnym przeszukiwaniu przestrzeni wartości parametrów metod SLAM, dlatego też wymaga określenia dolnego i górnego zakresu dla każdego parametru liczbowego, a także kroku. W przypadku parametrów przyjmujących jedynie wybrane wartości, jak typy mechanizmów, można określić zbiór wartości. Po zdefiniowaniu wybranych parametrów odpowiednie oprogramowanie generuje zestawy testowe w formie plików *yaml*, które następnie wczytywane są podczas ewaluacji.

Ewaluacja

Ewaluacja opiera się na narzędziu, które uruchamia metodę SLAM na podstawie danych umieszczonych w zbiorze danych (plik *rosbag*), czyli danych z sensorów i drzewa transformacji. Działanie metody SLAM jest symulowane, a rezultaty w postaci mapy i estymowanej trajektorii robota zapisywane do dalszej analizy. Dodatkowo realizowana jest operacja, która określa transformację między układem odniesienia systemu referencyjnego i systemu robota, aby umożliwić porównywanie rezultatów. Ewaluacji jest poddawany każdy przejazd robota z każdym możliwym zestawem parametrów, więc często ten proces wymaga znacznych zasobów obliczeniowych lub czasu.

Generowanie metryk i selekcja rozwiązań

Ostatnim etapem jest wygenerowanie metryk dla każdej jednostki ewaluacyjnej składającej się z konkretnego przejazdu i zestawu parametrów. Można skorzystać w tym celu z narzędzia evo [86]. Jedna z generowanych metryk ma postać błędu średniokwadratowego między trajektorią referencyjną i trajektorią estymowaną. Następnie wygenerowane metryki są analizowane i wybierane jest rozwiązanie optymalne dla określonego zestawu danych.

5.5 Badania eksperymentalne metod SLAM

5.5.1 Badanie jakości estymacji trajektorii robotów

Przeprowadzono badania jakości estymacji trajektorii robotów przez metody SLAM i metody lokalizacji. Celem badań było porównanie działania różnych metod i określenie, która metoda pozwala uzyskać najlepsze rezultaty.

Na potrzeby testów wykorzystano robota mobilnego ReMeDi [196] wyposażonego w enkodery, skaner laserowy Hokuyo URG-04LX, IMU Orientus oraz czujnik RGB-D Kinect. Wykorzystano również optyczny system śledzenia ruchu, który umożliwił wyznaczenie trajektorii referencyjnej (ang. *ground truth data*). Dokładny opis środowiska badawczego znajduje się w sekcji 4.5.2.

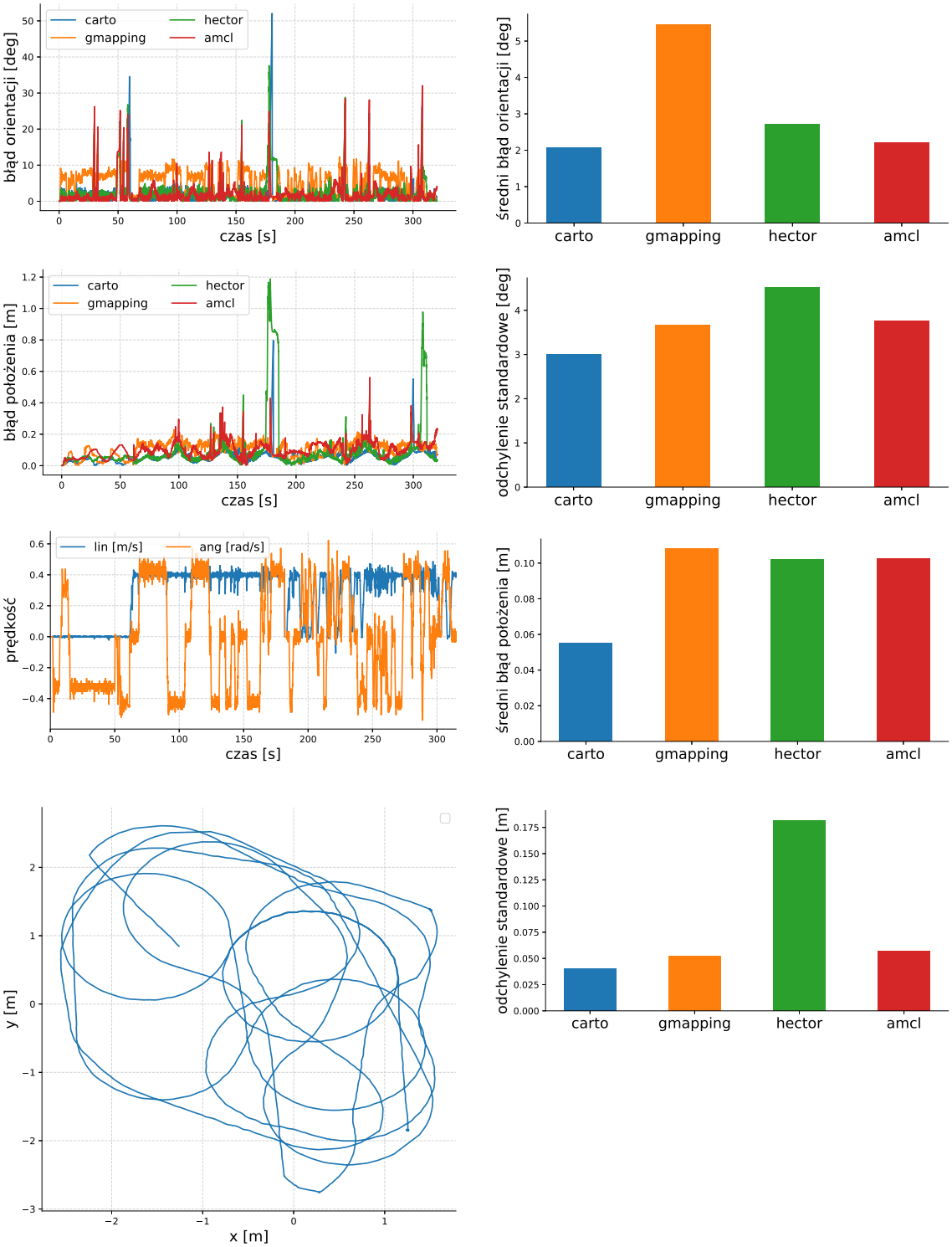
Eksperyment zrealizowano w dwóch etapach. W pierwszym etapie wykonano przejazdy oraz zebrano i zapisano dane pomiarowe. W takcie przejazdów robot poruszał się po względnie płaskiej powierzchni, na której występowały niewielkie nierówności, powodujące poślizgi kół robota, co pozwoliło lepiej odwzorować rzeczywiste warunki pracy robota mobilnego. W kolejnym etapie, na podstawie zapisanych danych uruchamiano metody SLAM i lokalizacji w celu estymacji trajektorii robota oraz stanu środowiska.

Badano metody takie metody SLAM jak *gmapping*, Hektor SLAM (*hektor*) [129] oraz cartographer (*carto*). Dodatkowo w celu porównania wyników, badano algorytm lokalizacji AMCL, wykorzystujący statyczną mapę wykonaną wcześniej metodą *gmapping*. Wszystkie wspomniane metody wykorzystywały dane pomiarowe ze skanera laserowego oraz systemu odometrycznego opartego na enkoderach.

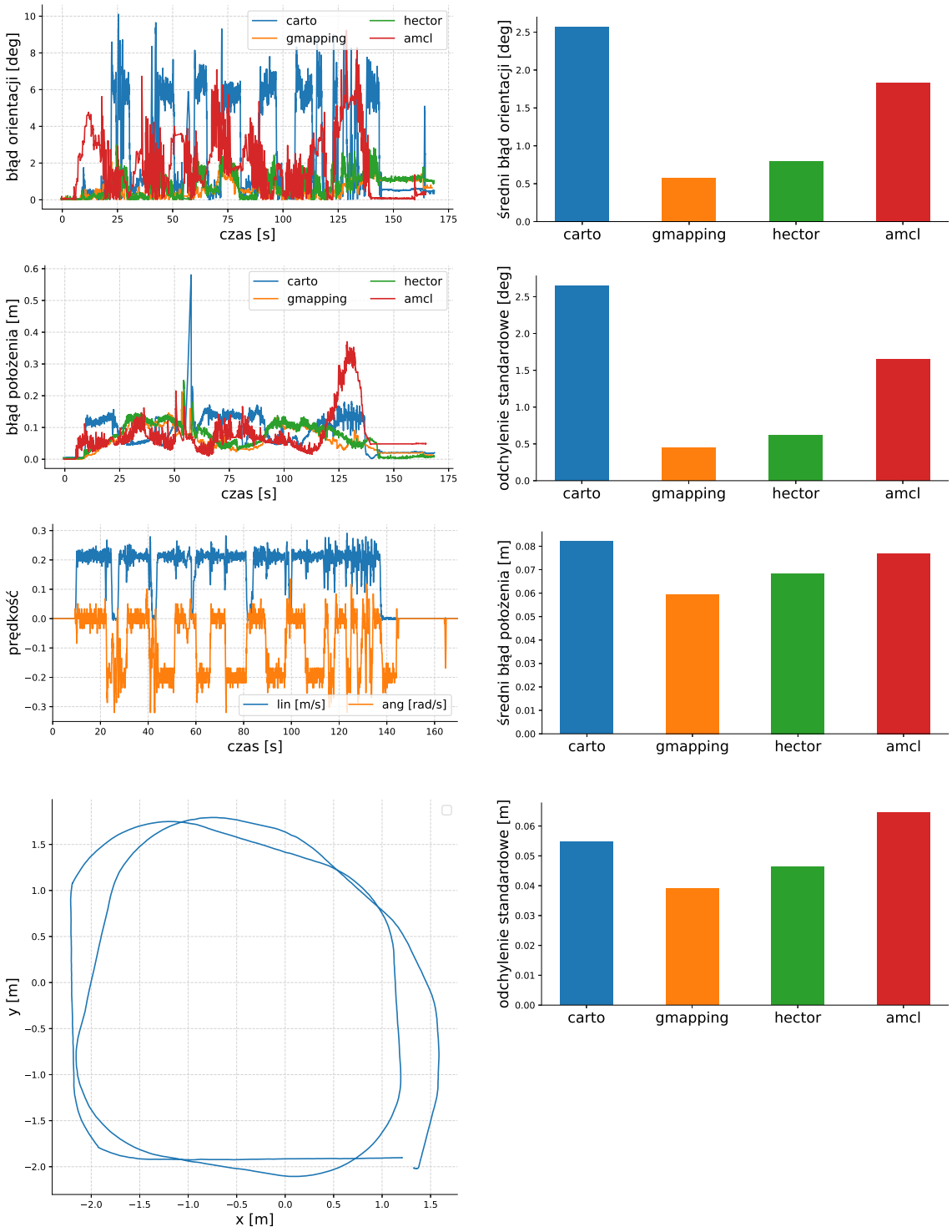
Metody zostały wstępnie dostrojone w sposób manualny, a następnie w celu poprawienia rezultatów wykorzystano system opisany w sekcji 5.4.

Rezultaty

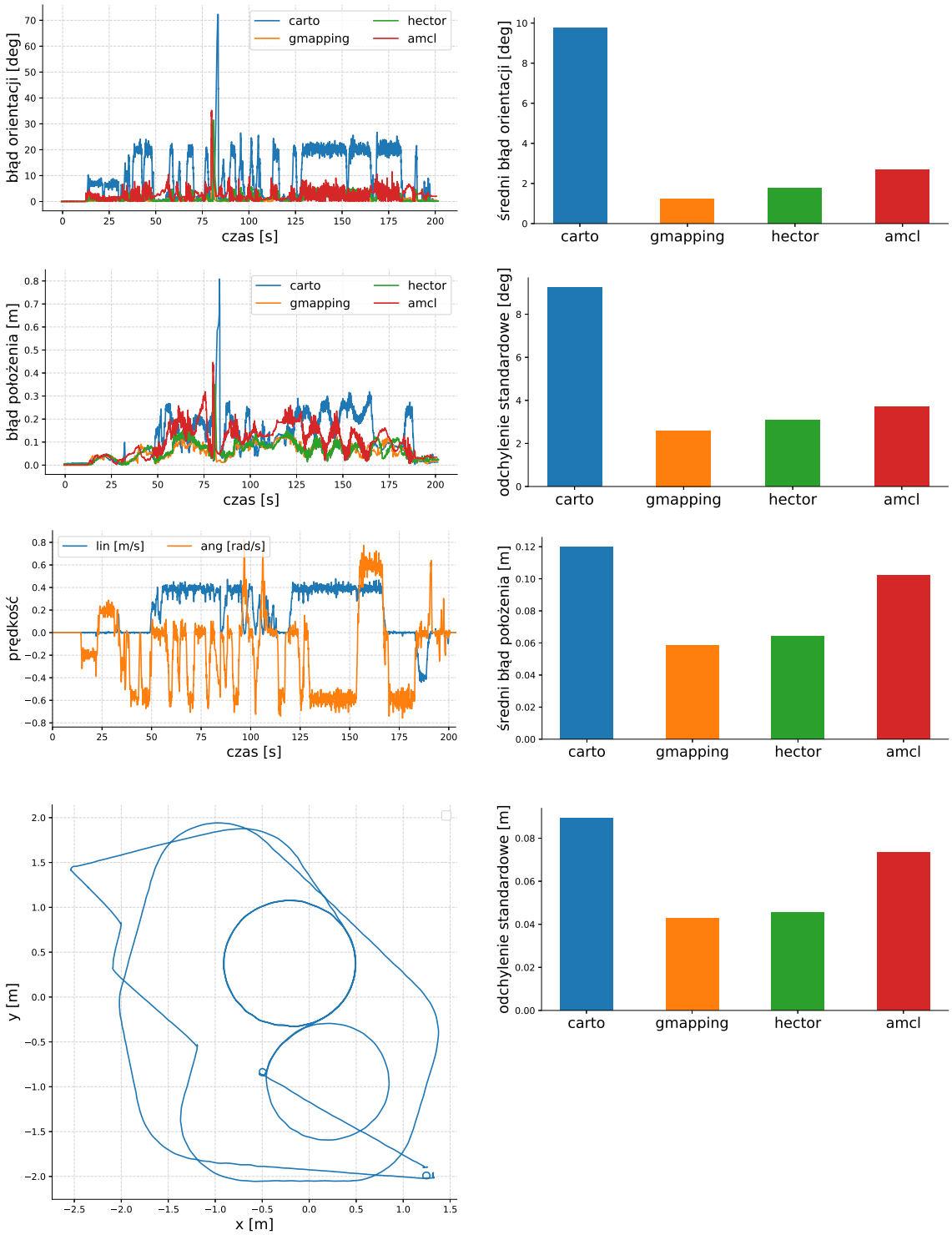
Na rys. 5.13, 5.14, 5.15 przedstawiono rezultaty estymacji trajektorii za pomocą wybranych metod SLAM i metod lokalizacji. Rys. 5.16 przedstawia zbiorcze porównanie średniego błędu estymacji orientacji i położenia, a także odchyleń standardowych. Zbiorcze porównanie obejmuje kilkanaście przejazdów robota w tej samej lokacji.



Rysunek 5.13 Porównanie jakości estymacji trajektorii przez metody SLAM oraz lokalizacji na podstawie przejazdu robota ReMeDi w eksperymencie nr I

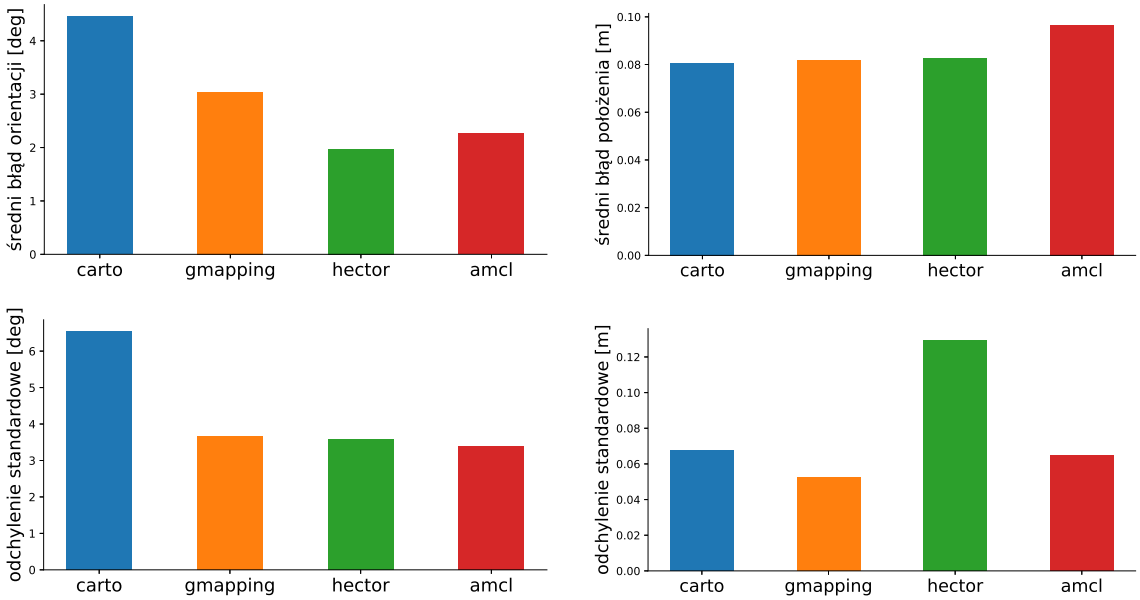


Rysunek 5.14 Porównanie jakości estymacji trajektorii przez metody SLAM oraz lokalizacji na podstawie przejazdu robota ReMeDi w eksperymencie nr II



Rysunek 5.15 Porównanie jakości estymacji trajektorii przez metody SLAM oraz lokalizacji na podstawie przejazdu robota ReMeDi w eksperymencie nr III

W tabelach 5.1 oraz 5.2 umieszczono zbiorcze statystyki dotyczące kolejno, jakości estymacji orientacji oraz jakości estymacji położenia robota.



Rysunek 5.16 Zbiorcze porównanie jakości estymacji trajektorii przez metody SLAM oraz lokalizacji na podstawie wielu przejazdów robota

Tabela. 5.1 Zbiorcze porównanie jakości estymacji orientacji robota

metoda	średni błąd	odchylenie standardowe	mediana	min	max
carto	4.5	6.5	1.6	3.2e-05	7.2e+01
gmapping	3.0	3.7	1.2	2e-05	3e+01
hector	2.0	3.6	1.1	4.4e-05	3.8e+01
amcl	2.3	3.4	1.4	4.8e-07	3.5e+01

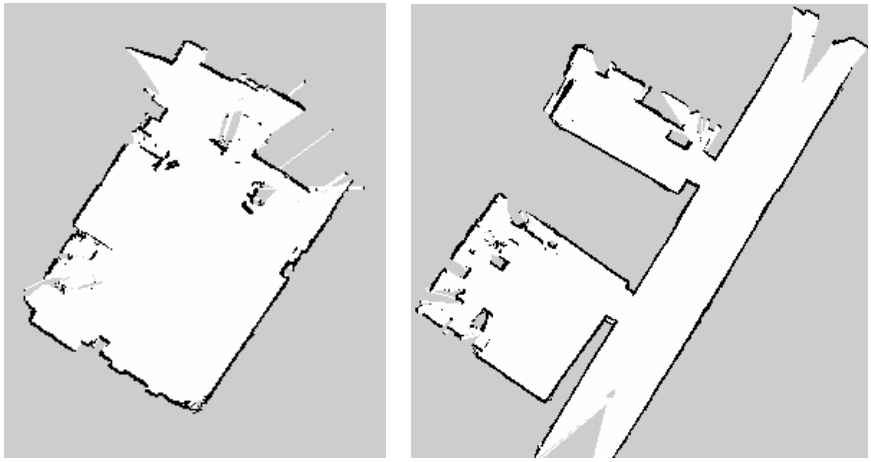
Tabela. 5.2 Zbiorcze porównanie jakości estymacji położenia robota

metoda	średni błąd	odchylenie standardowe	mediana	min	max
carto	0.081	0.068	0.062	0.00013	0.81
gmapping	0.082	0.053	0.08	6.2e-06	0.43
hector	0.083	0.13	0.057	0.0006	1.2
amcl	0.097	0.065	0.084	6.7e-06	0.56

5.5.2 Porównanie map utworzonych metodami SLAM

Przeprowadzono również szereg eksperymentów metod SLAM związanych z tworzeniem map. Wykorzystano kilka robotów oraz różne konfiguracje sensorów. W dalszej części przedstawiono wybrane mapy dwuwymiarowe, które powstały w trakcie eksperymentów. Podczas ich tworzenia wykorzystywano tryb autonomicznej nawigacji, a punkty docelowe zadawane były przez operatora.

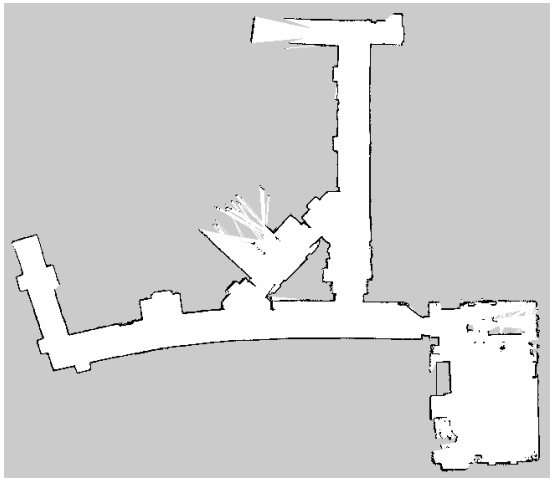
Na rys. 5.17 przedstawiono mapy wykonane robotem ReMeDi [196], wyposażonym w skaner laserowy Hokuyo URG-04LX, z wykorzystaniem algorytmu gmapping. Do utworzenia kolejnej mapy (rys. 5.18) również wykorzystano algorytm gmapping



(a) laboratorium Politechniki Wrocławskiej (b) budynek Uniwersytetu Medycznego w Lublinie

Rysunek 5.17 Mapy 2D wykonane robotem mobilnym ReMeDi i algorytmem gmapping

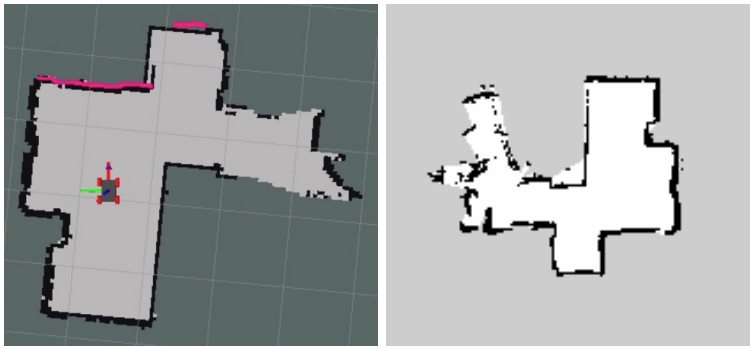
oraz dwukołowego robota mobilnego wyposażonego w dwa skanery laserowe Hokuyo UST-10LX. Mapę przedstawioną na rys. 5.19 wykonano niewielkim robotem czteroko-



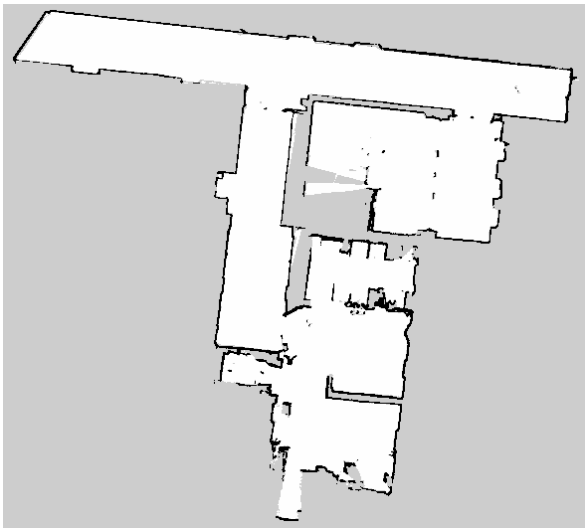
Rysunek 5.18 Mapa piętra wybranego budynku Politechniki Wrocławskiej

łowym [51]. Robot wyposażony był sensor Microsoft Kinect, z którego dane zostały transformowane do postaci dwuwymiarowej, a następnie podane do algorytmu gmapping. Na potrzeby lokalnej lokalizacji wykorzystano enkodery umieszczone na wszystkich kołach, IMU i algorytm UKF realizujący fuzję danych. Wykorzystanie algorytmu UKF pozwoliło znacząco poprawić jakość lokalizacji, ponieważ czterokołowa konstrukcja robota powoduje, że porusza się on z poślizgami.

Na rys. 5.20 zaprezentowano mapę 2D wykonaną algorytmem Cartographer. Do jej utworzenia wykorzystano robota mobilnego Turtlebot wyposażonego w skaner laserowy Hokuyo UST-10LX oraz zestaw enkoderów.



Rysunek 5.19 Mapa pomieszczenia wykonana robotem czterokołowym wyposażonym w sensor Kinect



Rysunek 5.20 Mapa piętra budynku Politechniki Wrocławskiej wykonana robotem Turtlebot wyposażonym w skaner laserowy Hokuyo

5.6 Podsumowanie

W niniejszym rozdziale przedstawiono przegląd metod SLAM dla robotów mobilnych oraz scharakteryzowano wybrane metody. Ponadto w rozdziale zaprezentowano wyniki przeprowadzonych badań eksperymentalnych jakości estymacji trajektorii oraz modelu środowiska.

Na podstawie wyników przeprowadzonych badań eksperymentalnych jakości estymacji trajektorii robota, można stwierdzić, że najmniejszy średni błąd orientacji uzyskano w przypadku metody *hector* SLAM. Zaobserwowano także, że metoda *cartographer* znacznie gorzej radziła sobie z rotacjami robota. Odzwierciedla to znacznie większa wartość średniego błędu orientacji w porównaniu do pozostałych metod, ale też większa wartość odchylenia standardowego oraz maksymalnego błędu wynoszącego chwilowo nawet $72°$. Sytuację tę mogłoby poprawić wykorzystanie danych z odpowiednio skalibrowanego IMU.

Jeżeli chodzi o średni błąd położenia, dla trzech metod SLAM uzyskano zbliżone rezultaty, natomiast wartość o około 20% wyższą zaobserwowano dla metody lokalizacji AMCL. Mimo zbliżonych wartości średnich błędów, zaobserwowano znacznie większą wartość odchylenia standardowego dla błędu estymacji położenia w przypadku metody *hector*, co sugeruje większą niestabilność tej metody przy wyznaczaniu położenia. Rów-

nież maksymalna wartość błędu była największa dla metody *hector* i wynosiła $1,2$ m.

Należy jednak mieć na uwadze, że porównanie zostało przygotowane w oparciu o wiele przejazdów robota w tym samym środowisku, ze względu na dostępność systemu śledzenia ruchu. Aby uzyskać bardziej reprezentatywne wyniki, należałoby powtórzyć badanie również w innych środowiskach.

Przedstawione wyniki eksperymentów obejmują też mapy wykonane różnymi metodami, z wykorzystaniem różnych robotów wyposażonych w odmienne zestawy sensorów. Mapy te z powodzeniem były wykorzystywane jako mapy statyczne dla algorytmów lokalizacji.

Rozdział 6

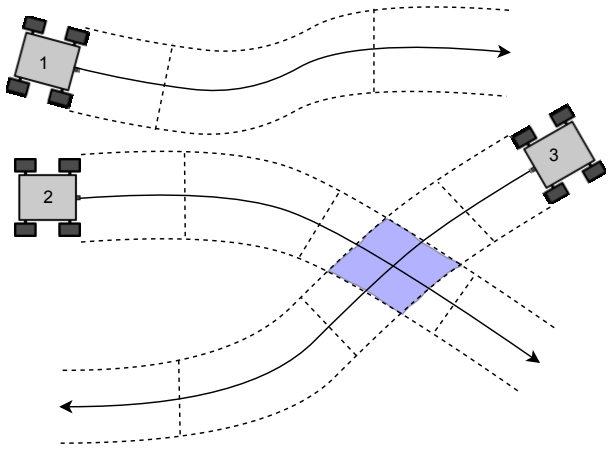
Systemy wielorobotowe

W niniejszym rozdziale omówiono podstawowe aspekty związane z systemami wielu robotów mobilnych.

6.1 Wprowadzenie

Znaczący postęp w dziedzinie robotów mobilnych jest stymulowany przez liczne aplikacje [43, 68, 256]. Okazuje się, że systemy wielorobotowe (*ang. Multi-Robot Systems, MRS*) mają przewagę względem pojedynczych robotów w wielu zastosowaniach, jak na przykład w eksploracji kopalni [68], eksploracji planet, ratownictwie bezzałogowym, czy różnego rodzaju aplikacjach przemysłowych [116,130,256]. W szczególności, systemy wielorobotowe (rys. 6.1) cechują się:

- współbieżnością - co pozwala uzyskać większą wydajność systemu, a miarą wydajności systemu może być przykładowo czas wykonania zadania,
- niezawodnością - są odporne na uszkodzenia w obrębie pojedynczych robotów i w przypadku ich wystąpienia mogą kontynuować realizację zadania, przy założeniu, że funkcje uszkodzonego robota może przejąć inny robot należący do grupy,



Rysunek 6.1 Przykład kolizyjnych ścieżek robotów w systemie wielorobotowym. Jednym z rozwiązań, które pozwala uniknąć kolizji jest podział ścieżek na sektory i odpowiednia koordynacja robotów

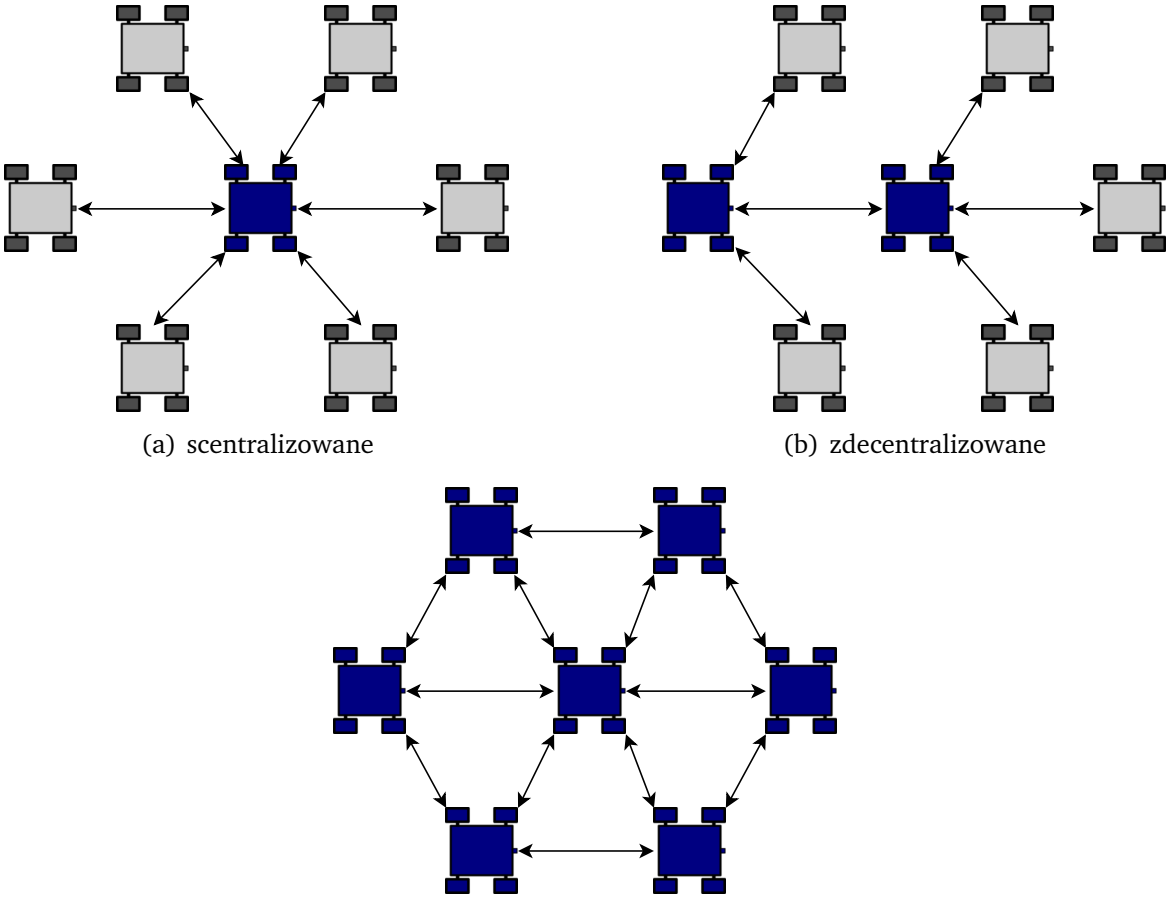
- elastycznością - mogą łatwiej adaptować się do różnych aplikacji. Dotyczy to głównie systemów heterogenicznych, w których każdy robot może posiadać inny zestaw umiejętności.

Systemy wielorobotowe charakteryzuje jednak dodatkowy stopień skomplikowania względem pojedynczych robotów. Przede wszystkim, należy rozwiązać problemy związane z komunikacją, unikaniem kolizji, czy wzajemną interakcją robotów. Dochodzą także kwestie optymalizacyjne, czyli odpowiednia alokacja zadań oraz koordynacja działań poszczególnych robotów [203]. Przegląd metod koordynacji i sterowania grup robotów można znaleźć w pracach [49, 256]

6.2 Klasyfikacja systemów wielorobotowych

Systemy wielorobotowe można podzielić ze względu na to, który robot pełni funkcje nadzorcze. Wyróżnia się systemy (rys. 6.2):

- scentralizowane,
- zdecentralizowane,
- oraz rozproszone.



(a) scentralizowane (b) zdecentralizowane

(c) rozproszone

Rysunek 6.2 Klasyfikacja systemów wielorobotowych

W systemach scentralizowanych, jest tylko jeden nadzorca. Taką rolę może pełnić zarówno wybrany robot jak i serwer. Systemy zdecentralizowane opierają się na wykorzystaniu wielu nadzorców. Każdy z nich zarządza pewną grupą robotów, a dodatkowo zarządcy komunikują się między sobą. W systemach rozproszonych, każdy robot jest niezależny.

6.2.1 Grupy homogeniczne i heterogeniczne

Formalnie, system wielorobotowy można zdefiniować jako grupę robotów współpracujących ze sobą w celu zrealizowania danego zadania.

Grupa robotów, posiadających takie same funkcjonalności, nazywana jest grupą homogeniczną (rys. 6.3), ale spotykane jest też określanie takich grup mianem roju (ang. *swarm robotics*). W grupie homogenicznej każdy z robotów może realizować dokładnie te same funkcje. Zaletą takiego podejścia jest to, że w przypadku awarii dowolnego robota należącego do grupy, można go łatwo zastąpić, przez co systemy tego typu cechują się dużą niezawodnością.

Rysunek 6.3 Przykład roju miniaturowych robotów

Grupę robotów o różnych funkcjonalnościach określa się mianem grupy heterogenicznej. W takim przypadku każdy robot może być wyspecjalizowany do wykonania konkretnego zadania, co całej grupie pozwala wykonywać znacznie bardziej skomplikowane operacje. Jednak systemy heterogeniczne są bardziej skomplikowane od systemów homogenicznych, między innymi ze względu na trudniejsze planowanie zadań i koordynację. Dodatkowo są mniej odporne na uszkodzenia od grup homogenicznych, ponieważ nie zawsze jest możliwość zastąpienia funkcji uszkodzonego robota przez innego robota należącego do grupy.

Do oddzielnej klasy można zaliczyć systemy robotów heterogenicznych, współpracujących z człowiekiem [43]. Samą interakcję człowieka z robotem określa się mianem HRI (*ang. Human-robot interaction*). Jedno z wykorzystywanych podejść polega na tym, że człowiek jest nadzorcą grupy robotów. Komunikuje się on z całą grupą robotów i przydziela jej zadania. Wspomniane rozwiązania są testowane między innymi na potrzeby ratownictwa oraz operacji militarnych.

6.3 Komunikacja między robotami

W systemach wielorobotowych jedną z kluczowych cech jest możliwość komunikowania się robotów. Pojawiają się jednak związane z tym wyzwania. Między innymi, kanały komunikacyjne mogą nie być dostępne przez cały czas, co musi być uwzględnione podczas planowania zadań przez system. Dodatkowo kanały komunikacyjne mają ograniczoną przepustowość. Komunikacja odbywa się zazwyczaj z wykorzystaniem proactive routing lub routingu ad hoc, który potrzebuje sporych zasobów obliczeniowych i pamięciowych, a także generuje znaczne opóźnienia.

6.3.1 Synchronizacja danych

W celu synchronizacji danych z robotów do ramek danych dodaje się informację o czasie ich zebrania. Jednak, aby miało to sens w całym systemie potrzebny jest ten sam zegar. Należy jednak rozważyć dwa poziomy synchronizacji zegarów. Pierwszy poziom dotyczy lokalnych zegarów, w obrębie danego robota, czyli zegarów związanych na przykład z poszczególnymi czujnikami, co pozwala precyzyjnie określić czas zebrania danych pomiarowych. Kolejny poziom określa synchronizację zegarów między poszczególnymi robotami. Jednym z powszechnie stosowanych rozwiązań problemu synchronizacji zegarów jest wykorzystanie narzędzia *chrony*.

6.3.2 Dystrybucja danych

Systemy wielorobotowe mogą być scentralizowane, zdecentralizowane lub rozproszone [216]. W systemach scentralizowanych, obliczenia realizowane są przez wyznaczonego robota lub serwer, pełniący rolę centralnego agenta. Systemy zdecentralizowane opierają się na obliczeniach wykonywanych przez więcej niż jednego agenta. Wiąże się to ze zwiększonymi wymaganiami sprzętowymi dla poszczególnych agentów. Ostatnią grupę systemów stanowią systemy rozproszone, gdzie obliczenia związane z realizacją zadań rozdzielone są na wszystkie roboty. Przykładem może być kooperacyjne mapowanie środowiska, w którym część robotów tworzy swoje mapy, po czym są one optymalizowane przez zewnętrznego agenta.

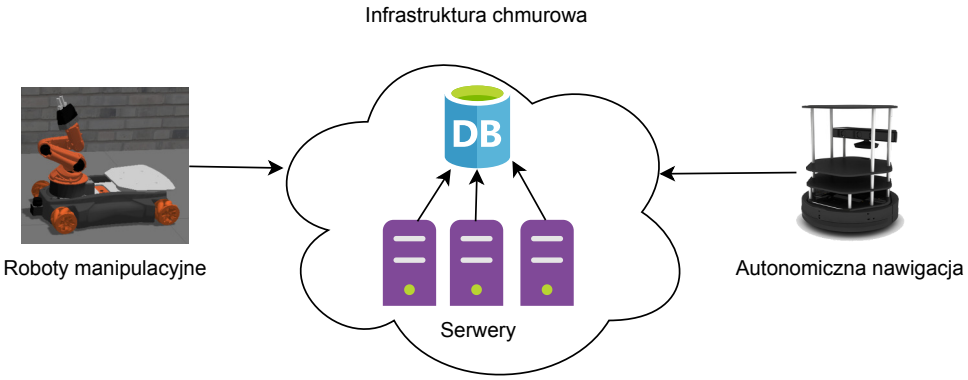
6.3.3 Rodzaj przesyłanych danych

Jeżeli chodzi o lokalizację i mapowanie, istnieją dwa główne podejścia odnośnie rodzaju danych przesyłanych między poszczególnymi robotami. Pierwsze podejście zakłada przesyłanie jedynie odczytów z czujników wraz z czasem ich zebrania. W tym przypadku wadą jest potrzeba niezawodnej komunikacji niemal przez cały czas oraz duża przepustowość łącza komunikacyjnego. Kolejna wada takiego podejścia wiąże się z duplikacją wykonywanych operacji obliczeniowych, ponieważ te same dane są przetwarzane przez wiele robotów.

Drugie podejście opiera się na przesyłaniu przetworzonych danych, takich jak mapy, rozpoznane obiekty, czy inne informacje wyższego poziomu. W tym przypadku wymagania odnośnie przepustowości łącza są mniejsze, natomiast wadą może być potrzeba opracowania bardziej skomplikowanych algorytmów łączących dane tego typu.

6.4 Robotyka w chmurze

Zauważalnym trendem w robotyce jest wykorzystanie technologii chmurowych. Termin robotyka w chmurze (*ang. cloud robotics*) po raz pierwszy pojawił się w roku 2010, w ramach wykładu wygłoszonego przez Jamesa Kuffnera [217]. Od tego czasu termin ten jak i utworzona dziedzina zyskały znacznie na popularności. Obejmuje ona takie zagadnienia jak przechowywanie danych w chmurze, przetwarzanie w chmurze, usługi w chmurze, a także kwestie związane z rozproszoną inteligencją i udostępnianiem informacji [164] (rys. 6.4).



Rysunek 6.4 Przegląd rozwiązań robotyki w chmurze [217]

Infrastruktura chmurowa składa się przede wszystkim z wydajnych serwerów obliczeniowych oraz serwerów baz danych. Celem jest zapewnienie jak najszybszego przetwarzania danych oraz dostępu do przestrzeni dyskowych.

Aby wykorzystać ogromne możliwości, które kryją się za technologiami chmurowymi, należy jednak opracować sposób przeniesienia obecnych aplikacji do chmury. Obecnie technologie chmurowe oferują trzy typy modeli usług:

- infrastruktura jako usługa (*ang. Infrastructure as a Service, IaaS*),
- platforma jako usługa (*ang. Platform as a Service, PaaS*),
- oprogramowanie jako usługa (*ang. Software as a Service, SaaS*),

W pracy [39] pojawił się jednak termin związany z nowym typem modeli usług, czyli robot jako usługa (*ang. Robot as a Service, RaaS*), określający sposób integracji robotów i urządzeń wbudowanych z technologiami chmurowymi. Podejście chmurowe ma wiele zalet, między innymi:

- odciążenie lokalnych jednostek obliczeniowych i przeniesienie bardziej skomplikowanych obliczeń do chmury. Może mieć to szczególne znaczenie w przypadku rozpoznawania obiektów, czy syntezy mowy,
- dostęp do przestrzeni dyskowych, co może umożliwić składowanie chociażby danych z czujników wykorzystywanych na potrzeby SLAM,
- umożliwia dostęp robotom do wielu zasobów globalnych, takich jak mapy do lokalizacji, ale również modele obiektów potrzebne do zadań związanych z manipulacją.

- stwarza możliwość wzajemnego uczenia się robotów, przez wymianę informacji między robotami znajdującymi się w różnych lokalizacjach [124].

Jednak z podejściem takim wiąże się też kilka nowych wyzwań:

- Potrzeba zwiększenia przepustowości i redukcji opóźnień w komunikacji między chmurą i robotami [217]. Pewnym rozwiązaniem tego problemu mogą być rozwijające się szybko sieci komórkowe 5G.

- Należy opracować sposoby działania w przypadku zerwania połączenia z chmurą lub znacznych opóźnień w komunikacji. Nietrudno wyobrazić sobie sytuację, w której połączenie zostaje utracone w trakcie wykonywania kluczowej operacji. Przykładem mogą być samochody autonomiczne korzystające z rozwiązań chmurowych, które muszą być w stanie również działać samodzielnie. Obecnie takie rozwiązania stosowane są w urządzeniach mobilnych na potrzeby rozpoznawania mowy. Działanie polega na jednoczesnym wykonywaniu analizy lokalnie i wysyłaniu danych do chmury. Następnie wybierane jest lepsze rozwiązanie, przy założeniu maksymalnego czasu oczekiwania na rezultat [124].

- Ważną kwestią jest bezpieczeństwo i prywatność. Przykładem jest przesyłanie do chmury wrażliwych danych, chociażby nagrań z kamer utworzonych na potrzeby nawigacji, czy mapowania. Kolejną kwestią jest podatność na zdalne ataki - kontrola nad robotem podłączonym do chmury może zostać przejęta przez nieuprawnione osoby [124].

- Efektywna alokacja zadań również jest ważnym zagadnieniem, w szczególności wymaga podejmowania decyzji czy dane zadanie powinno zostać wykonane lokalnie, czy przesłane do chmury [217].

Obecnie, wiele projektów robotycznych jest związanych z technologiami chmurowymi. Jednym z nich jest projekt RoboEarth [238] wraz z platformą robotyczną Rapyuta. Autorzy określają projekt RoboEarth mianem www dla robotów (ang. *World Wide Web*). Jest to wielkoskalowa sieć oraz baza danych, której celem jest umożliwienie dzielenia się informacjami przez roboty, a także ich wzajemnego uczenia się. System działający w oparciu o bazę wiedzy RoboEarth składa się z trzech warstw: warstwy serwera, warstwy komponentów generycznych, a także warstwy komponentów specyficznych dla robota. Platforma Rapyuta realizuje usługi w modelu "Platforma jako usługa" (Paas), co pozwala w stosunkowo łatwy sposób przenieść operacje obliczeniowe do chmury. Kolejnym projektem wykorzystującym rozwiązania chmurowe jest projekt DAVinCi [13] bazujący na klastrach Hadoop zintegrowanych i wykorzystujący środowisko ROS jako warstwę komunikacyjną.

Rozdział 7

Metody lokalizacji i mapowania dla wielu robotów

Niniejszy rozdział zawiera przegląd metod lokalizacji i mapowania przeznaczonych dla wielu robotów. Przedstawiono klasyfikację takich metod oraz omówiono wybrane algorytmy.

7.1 Wprowadzenie

Systemy wielorobotowe mogą lepiej realizować pewne zadania w porównaniu do pojedynczych robotów, o czym wspomniano już w rozdziale 6. W kontekście lokalizacji i mapowania, warto wyróżnić następujące możliwości wynikające z zastosowania systemu wielu robotów:

- roboty mogą być rozmieszczone w różnych obszarach środowiska i wykonywać pracę równolegle, przez co są w stanie znacznie szybciej zbudować mapę całego otoczenia,
- niezawodność związana z możliwością kontynuacji realizacji zadań, w szczególności zadania eksploracji, w przypadku defektu jednego z robotów,
- możliwość wymiany informacji między robotami pozwala na zmniejszenie niepewności w procesie estymacji lokalizacji i tworzenia mapy,
- poszczególne roboty w grupie mogą być mniej rozbudowane i nadal będą w stanie realizować te same zadania co bardziej rozbudowane pojedyncze roboty.

Dodatkowo, wykorzystanie heterogenicznych grup robotów może wprowadzać kolejne korzyści. Przede wszystkim pozwala uzyskać lepszej jakości mapy kiedy tworzone są one przez różne roboty wyposażone w zestawy sensorów o odmiennych możliwościach. Jako przykład można przytoczyć współpracę robotów latających i jeżdżących. Nie wszystkie elementy środowiska są dostępne dla robotów jeżdżących, a robot latający nie jest w stanie przemieszczać się wszędzie tam gdzie robot jeżdżący, w szczególności w budynkach. Mapy z poszczególnych robotów można połączyć w jeden kompletny model otoczenia. Jednak takie podejście wprowadza też dodatkowy stopień skomplikowania systemu przetwarzania danych. Potrzebny jest algorytm umożliwiający łączenie map różnych typów. W pracy [165] przedstawiono współpracę grupy robotów składającej się z dwóch robotów jeżdżących oraz wielowirnikowca na potrzeby budowy map miejsc dotkniętych kataklizmami.

Jednak wykorzystanie grup robotów do lokalizacji i mapowania wiąże się z wieloma nowymi wyzwaniami. Jednym z nich jest kwestia skalowalności systemów. Skalowalność wiąże się z efektywnym zarządzaniem grupą robotów, czyli między innymi koordynacją działań robotów, ale także redukowaniem ilości przesyłanych między nimi informacji.

Kolejną kwestią są środowiska wielkoskalowe. Zupełnie inaczej wygląda przetwarzanie danych z niewielkich obszarów niż w przypadku dużych otwartych przestrzeni. Problemem może być brak dostatecznie dużej liczby cech takiego środowiska i ograniczony zasięg czujników. Również problematyczna może być kwestia komunikacji między robotami na otwartych przestrzeniach.

Wyzwaniem związanym z wielorobotowym mapowaniem i lokalizacją jest też potrzeba znajomości względnych pozycji robotów. Zazwyczaj w metodach SLAM każdy z robotów buduje lokalną mapę, czy tworzy lokalną trajektorię, w lokalnym układzie współrzędnych. Aby wykorzystać dane z innych robotów należy w jakiś sposób określić względne pozycje między nimi. Jeden ze sposobów polega na wzajemnym wykrywaniu innych robotów w celu określenia wzajemnych pozycji. Można w tym celu wykorzystać czujniki wizyjne. Innym podejściem jest wykorzystanie czujników radiowych i określanie odległości między robotami na podstawie czasu przelotu sygnału. Jednak to rozwiązanie pozwala określić jedynie odległości między robotami, a nie względne transformacje między nimi. Kolejnym sposobem na określenie wzajemnych pozycji jest poszukiwanie i dopasowywanie wspólnych części map do siebie. Rozwiązanie opierające się na tym pomyśle przedstawiono w rozdziale 8.

7.2 Definicja problemu

Zagadnienie SLAM dla pojedynczego robota jest wymagającym problemem, natomiast wykorzystanie wielu robotów wprowadza dodatkowy stopień komplikacji. Zakładając, że grupa składa się z N robotów, problem wielorobotowego SLAM online, czyli MRSLAM (ang. *Multi-robot Simultaneous Localization and Mapping*) można zdefiniować w postaci funkcji prawdopodobieństwa stanów robotów $x_t^1, x_t^2 \ldots, x_t^N$ i mapy otoczenia m pod warunkiem wartości sterowań $u_{1:t}^1, u_{1:t}^2 \ldots, u_{1:t}^N$ i obserwacji $z_{1:t}^1, z_{1:t}^2 \ldots, z_{1:t}^N$ w chwili t

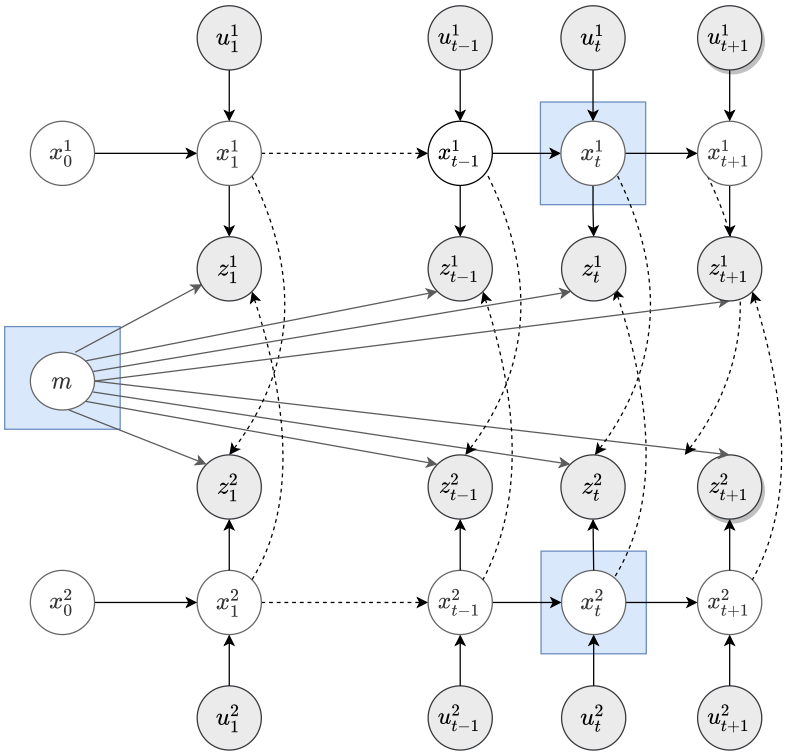
$$p(x_t^1, x_t^2, \ldots, x_t^N, m \mid z_{1:t}^1, z_{1:t}^2, \ldots, z_{1:t}^N, u_{1:t}^1, u_{1:t}^2, \ldots, u_{1:t}^N). \tag{7.1}$$

Na rys. 7.1 przedstawiono sieć bayesowską dla problemu SLAM online, w przypadku dwóch robotów. Pełny problem MRSLAM (ang. *full-MRSLAM*) dla grupy N robotów definiuje się analogicznie (rys. 7.2), przy czym określa się prawdopodobieństwo trajektorii robotów zamiast ostatniej pozycji

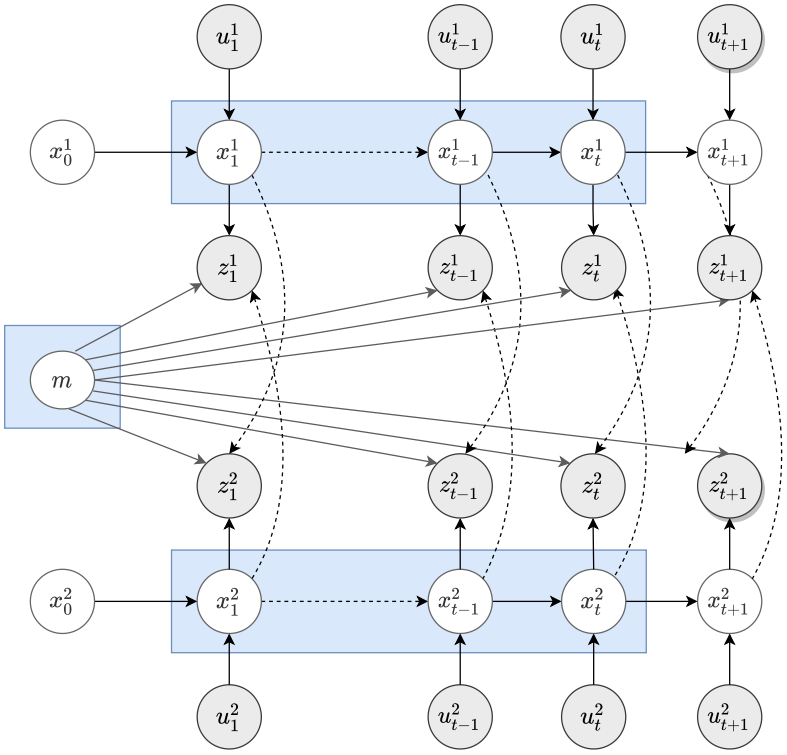
$$p(x_{1:t}^1, x_{1:t}^2, \ldots, x_{1:t}^N, m \mid z_{1:t}^1, z_{1:t}^2, \ldots, z_{1:t}^N, u_{1:t}^1, u_{1:t}^2, \ldots, u_{1:t}^N). \tag{7.2}$$

7.3 Klasyfikacja metod SLAM dla systemów wielorobotowych

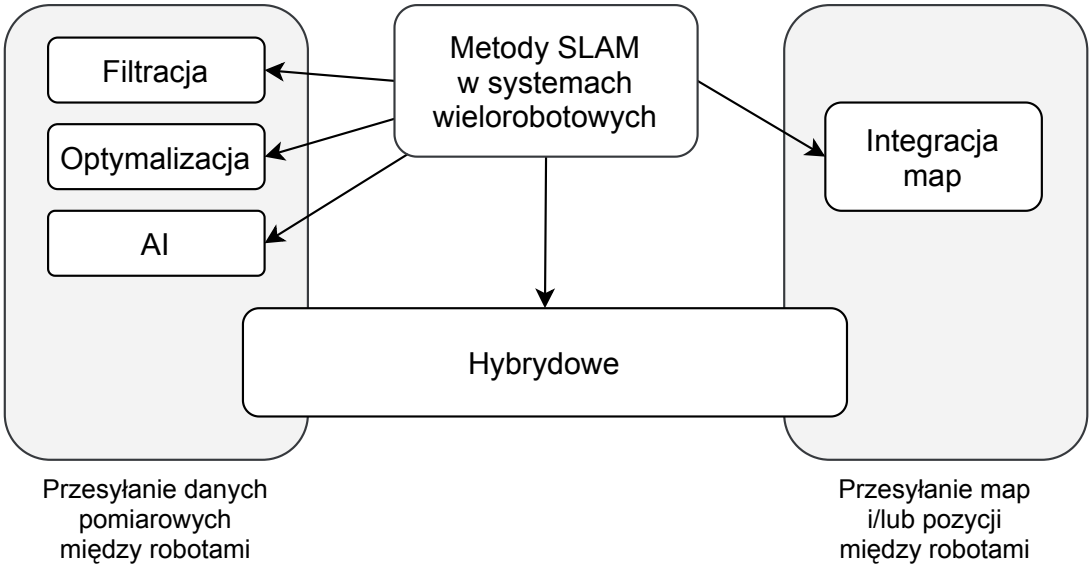
Metody SLAM dla wielu robotów można sklasyfikować, na podstawie typu danych wymienianych między robotami (rys. 7.3). Wyróżnia się trzy główne podejścia:



Rysunek 7.1 Sieć bayesowska dla problemu online MRSLAM, w przypadku dwóch robotów. Problem ten polega na określeniu zmiennych losowych m oraz x_t^1 i x_t^2, przy założeniu znanych x_0^1 oraz x_0^2



Rysunek 7.2 Sieć bayesowska dla problemu full MRSLAM, w przypadku dwóch robotów. Problem ten polega na określeniu zmiennych losowych m oraz $x_{1:t}^1$ i $x_{1:t}^2$, przy założeniu znanych x_0^1 oraz x_0^2



Rysunek 7.3 Ogólna klasyfikacja metod SLAM dla wielu robotów

- przesyłanie nieprzetworzonych danych pomiarowych z czujników,
- przesyłanie przetworzonych danych w postaci map lub pozycji robotów,
- metody hybrydowe opierające się zarówno na danych z czujników jak i mapach lub pozycjach robotów.

W metodach opierających się na przesyłaniu przetworzonych danych, celem jest zwykle połączenie tych map. Do ich połączenia wymagana jest znajomość transformacji między układami współrzędnych map. Realizuje się to w ten sposób, że wyposaża się roboty w dodatkowe sensory do wzajemnego wykrywania się lub bazuje się na wspólnych częściach map. Metody integracji map zostały szerzej opisane w rozdziale 8, natomiast w tym rozdziale skupiono się na pozostałych podejściach.

Metody SLAM wykorzystujące przesyłanie nieprzetworzonych danych z sensorów, można dodatkowo podzielić ze względu na zasadę działania. W tym przypadku, metody SLAM dla wielu robotów, podobnie jak dla pojedynczych robotów, można podzielić na trzy grupy: oparte na filtracji, oparte na optymalizacji oraz wykorzystujące techniki AI. Do metod opartych na filtracji należą wielorobotowe wersje metod EKF-SLAM oraz PF-SLAM, czyli odpowiednio MR EKF-SLAM oraz MR PF-SLAM. Istnieje również odmiana metody bazującej na optymalizacji, czyli GraphSLAM dla systemów wielorobotowych.

Rozwiązania hybrydowe zwykle wykorzystują przesyłanie zarówno map jak i danych z czujników. Przykładem może być przesyłanie map oraz odczytów ze skanera laserowego, który następnie dopasowywany jest do mapy robota, który dane otrzymał i na tej podstawie określana jest transformacja między robotami.

Odrębnym podejściem do mapowania i lokalizacji są kooperacyjne systemy pozycjonowania (CPS, ang. *Cooperative Positioning System*). Wykorzystuje się w nich roboty pełniące rolę ruchomych znaczników i w rezultacie zapewnia dużą dokładność lokalizacji. Pozwala to sprowadzić problem SLAM głównie do problemu mapowania.

Kolejnym podejściem, wykorzystanym między innymi w metodzie Cartographer jest łączenie submap. W tym rozwiązaniu każdy z robotów zapisuje swoją trajektorię i tworzy zestaw submap wraz z ograniczeniami. Następnie submapy i trajektorie są wymieniane między robotami i optymalizowane.

Zamykanie pętli

Roboty w trakcie działania podlegają wpływowi czynników zewnętrznych. Z czasem rośnie wpływ tych czynników na jakość estymowanej pozycji oraz tworzonej mapy. Sposobem na radzenie sobie z tym jest wykrywanie pętli w środowisku, czyli miejsc, które roboty odwiedzają ponownie. W przypadku pojedynczych robotów jest to trudny problem, jednak dla wielu robotów dodatkowo się komplikuje. Zwłaszcza że, powodem przeprowadzenia optymalizacji może być nie tylko ponowne odwiedzenie tego samego miejsca, ale także spotkanie z innymi robotami, czy znalezienie odwiedzonych już miejsc w danych przesyłanych z innych robotów [18].

7.4 Algorytm MR EKF-SLAM

Jedną z metod stosowanych w przypadku systemów wielorobotowych jest MR EKF-SLAM [266]. Metoda rozwiązuje problem SLAM oparty na cechach, wykorzystując algorytm EKF do filtrowania pozycji znaczników i pozycji robotów. Dodatkowo, zakłada się, że roboty wymieniają dane w momencie spotkań i posiadają możliwość określania swoich względnych pozycji, czyli zarówno odległości jak i różnicy orientacji.

Dla uproszczenia rozpatrzono przypadek dwóch robotów mobilnych a i b . W tej sytuacji, wektor stanu przyjmuje postać

$$x_t^{ab} = \begin{bmatrix} x_t^a & x_t^b & x_t^{f_1} & y_t^{f_1} & \dots & x_t^{f_N} & y_t^{f_N} \end{bmatrix}^T, \tag{7.3}$$

gdzie:

- x_t^a oraz x_t^b oznaczają odpowiednio pozycje robota a oraz b,
- N określa liczbę cech środowiska,
- $x_t^{f_k}$ oraz $y_t^{f_k}$ określają współrzędne k-tej cechy dla $k \in [1, N]$.

Dodatkowo, przez Σ_t^{ab} oznacza się macierz kowariancji dla x_t^{ab}.

Działanie metody EKF-SLAM dla systemów MR zostało przedstawione w postaci algorytmu 2. Predykcja i korekcja realizowane są dla każdego robota i są to standardowe

Algorytm 2: Algorytm EKF-SLAM dla wielu robotów [216]

Data: x_{t-1}^{ab} - wektor stanu
Σ_{t-1}^{ab} - macierz kowariancji,
u_t^a, u_t^b - sterowania,
z_t^a, z_t^b - pomiary
Result: x_t^{ab}, Σ_t^{ab}

1 **for** *robot r w {a,b}* **do**
2 $(\overline{x}_t^{ab}, \overline{\Sigma}_t^{ab})$ = predykcja($x_{t-1}^{ab}, \Sigma_{t-1}^{ab}, u_t^r$)
3 $(x_t^{ab}, \Sigma_t^{ab})$ = korekcja($\overline{x}_t^{ab}, \overline{\Sigma}_t^{ab}, z_t^r$)
4 $(x_t^{ab}, \Sigma_t^{ab})$ = dodaj_nowe_cechy($x_t^{ab}, \Sigma_t^{ab}, z_t^r$)
5 **end**
6 $(x_t^{ab}, \Sigma_t^{ab})$ = połącz_zduplikowane_cechy(x_t^{ab}, Σ_t^{ab})

procedury metody EKF. Następnie do wektora stanu dodaje się nowe cechy, otrzymane na podstawie danych z innych robotów. Cechy te są transformowane do układu odniesienia robota do którego zostały przesłane, w oparciu o transformację określoną przez system wyznaczania względnych pozycji. W ostatnim kroku, już po dodaniu wszystkich nowych cech, wyszukuje się podobne cechy i łączy je ze sobą. Do wyszukiwania zduplikowanych cech autorzy [266] wykorzystują algorytm 3 oparty na drzewie KD.

Algorytm 3: Wykrywanie i łączenie zduplikowanych cech [266]

Data: M_1, M_2 - zbiory cech odpowiednio dla robota R_1 i robota R_2,
γ - wartość progowa odległości cech

```
Posortuj cechy l_j ∈ M_2 rosnąco na podstawie ich odległości od robota R_2
Zbuduj drzewo KD składające się z cech w L_i ∈ M_1
for cecha w l_j ∈ M_2 do
    Znajdź najbliższego sąsiada w L_i ∈ M_1
    Oblicz odległość D_ij
    if D_ij ≤ γ then
        Korekcja za pomocą filtru KF
        Przebudowanie drzewa KD
        Usunięcie zduplikowanych cech z M_2
    end
end
```

7.5 Algorytm MR PF-SLAM

Metoda SLAM oparta na filtrze cząsteczkowym dla systemów wielorobotowych została przedstawiona w pracy [101]. Zadanie polega na estymacji prawdopodobieństwa trajektorii dwóch robotów: $x^a_{1:t}$, $x^b_{1:t}$, a także mapy m_t. Zakładając, że $u^a_{1:t}, u^b_{1:t}$ to sterowania, $z^a_{1:t}, z^b_{1:t}$ to obserwacje, a x^a_0, x^b_0 to pozycje początkowe, zależność między funkcjami prawdopodobieństwa jest następująca

$$\begin{aligned} &p(x^a_{1:t}, x^b_{1_t}, m_t \mid z^a_{1:t}, z^b_{1:t}, u^a_{1:t}, u^b_{1:t}, x^a_0, x^b_0) = \\ &\quad p(m_t \mid x^a_{1:t}, x^b_{1_t}, z^a_{1:t}, z^b_{1:t}) \\ &\quad \cdot p(x^a_{1:t} \mid z^a_{1:t}, u^a_{1:t}, x^a_0) \\ &\quad \cdot p(x^b_{1:t} \mid z^b_{1:t}, u^b_{1:t}, x^b_0). \end{aligned} \tag{7.4}$$

Pierwsza część zależności $p(m_t \mid x^a_{1:t}, x^b_{1_t}, z^a_{1:t}, z^b_{1:t})$ określa problem mapowania przez wiele robotów przy znanych pozycjach początkowych. Kolejne części to dystrybucje związane z pozycjami robotów, które są wyznaczane z wykorzystaniem filtra cząsteczkowego.

Jednym z założeń przyjętych do wyznaczenia zależności (7.4) jest niezależność trajektorii oraz obserwacji poszczególnych robotów. W rzeczywistości nie jest to prawda, ponieważ, kiedy roboty są blisko siebie, to wpływają wzajemnie na swoje dane pomiarowe. Jednym z rozwiązań tego problemu jest wykorzystanie metody do wzajemnego wykrywania robotów i usuwania pozostałych robotów z danych pomiarowych. Pozwala to spełnić założenie o niezależności obserwacji i trajektorii robotów.

Zbiór cząsteczek ma postać

$$S_t = \{x_t^{a(i)}, x_t^{b(i)}, m_t^{(i)}, w_t^{(i)}\}, \tag{7.5}$$

gdzie $x_t^{a(i)}$ oraz $x_t^{b(i)}$ oznaczają pozycje robotów w czasie t, $m_t^{(i)}$ to mapa, a $w_t^{(i)}$ określa wagi cząsteczek. Filtr cząsteczkowy przedstawiono w formie algorytmu nr 4, gdzie M określa generator map.

Algorytm 4: Filtr cząsteczkowy dla wielu robotów przy założeniu znanych pozycji początkowych i niezależnych trajektorii oraz obserwacji robotów

Data: S_{t-1} - zbiór cząsteczek reprezentujących rozkład prawdopodobieństwa a posteriori w chwili $t-1$,
u_t^a, u_t^b - sterowania w chwili t,
z_t^a, z_t^b - obserwacje w chwili t

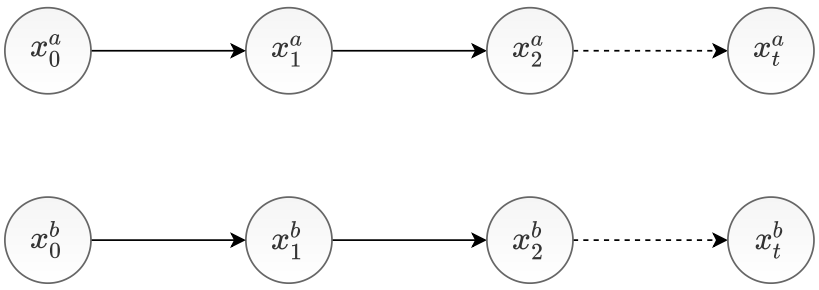
Result: S_t - zbiór cząsteczek reprezentujących rozkład prawdopodobieństwa a posteriori w chwili t

$S_t = \emptyset$
for *i od 1 do* N **do**
  $\{x_{t-1}^{a(i)}, x_{t-1}^{b(i)}, m_{t-1}^{(i)}, w_{t-1}^{(i)}\} = S_{t-1}^{(i)}$
  wybierz próbkę $x_t^{a(i)} \sim p(x_t^{a(i)} \mid x_{t-1}^a, u_t^a)$
  wybierz próbkę $x_t^{b(i)} \sim p(x_t^{b(i)} \mid x_{t-1}^b, u_t^b)$
  $w_t^{(i)} = p(z_t^a \mid x_t^{a(i)}, m_{t-1}^{(i)}) p(z_t^b \mid x_t^{b(i)}, m_{t-1}^{(i)}) w_{t-1}^{(i)}$
  $m_t^{(i)} = M(z_t^a, x_t^{a(i)}, m_{t-1}^{(i)}) + M(z_t^b, x_t^{b(i)}, m_{t-1}^{(i)}) + m_{t-1}^{(i)}$
  $S_t = S_t \cup \{x_t^{a(i)}, x_t^{b(i)}, m_t^{(i)}, w_t^{(i)}\}$
end
Ponowny wybór cząsteczek ze zbioru S_t
return S_t

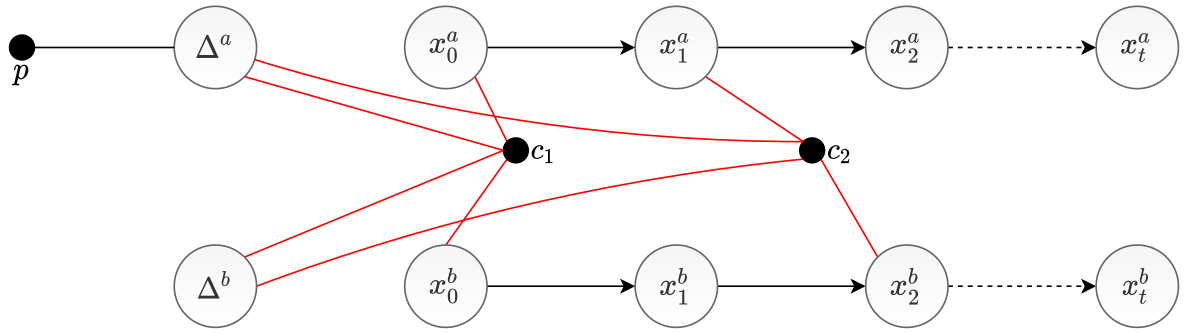
W pracy [101] przedstawiono również przypadek, gdy początkowe pozycje robotów nie są znane.

7.6 Algorytm MR GraphSLAM

W pracy [18] przedstawiono wersję metody GraphSLAM dla wielu robotów, czyli rozwiązania opartego na optymalizacji grafu pozycji. Przedstawiona metoda wykorzystuje bezpośrednie spotkania robotów i wzajemne detekcje. Graf pozycji dla przypadku, gdy roboty się nie spotykają i nie wymieniają informacji przedstawiono na rys. 7.4, a na rys. 7.5 dla przypadku, gdy do spotkań między robotami dochodzi.



Rysunek 7.4 Grafy pozycji dla dwóch robotów, które się nie spotykają



Rysunek 7.5 Graf pozycji dla dwóch robotów, które się spotykają (miejsca spotkań c_1 oraz c_2). Obserwacje między robotami oznaczono kolorem czerwonym, natomiast ograniczenia związanie z ruchem i pomiarami robotów oznaczono kolorem czarnym. Dodatkowo, przez Δ^a oraz Δ^b oznaczono względne pozycje między robotami.

Problem optymalizacyjny w metodzie GraphSLAM ma postać nieliniowego problemu średniokwadratowego. Zakładając, że r określa numer robota, problem definiuje się następująco [18]

$$X^* = \arg\min_X \left\{ \sum_{r=\{a,b\}} \left(\|p^r - x_0^r\|_\Sigma^2 + \sum_{i=1}^{M_r} \left\| f(x_{i-1}^r, u_i^r) - x_i^r \right\|_{\Lambda_i^r}^2 \right) + \sum_{j=1}^{C} \left\| h(x_{i'_j}^{r'_j}, c_j, \Delta^{r_j}, \Delta^{r'_j}) - x_{i_j}^{r_J} \right\|_{\Gamma_j}^2 \right\}, \tag{7.6}$$

gdzie:

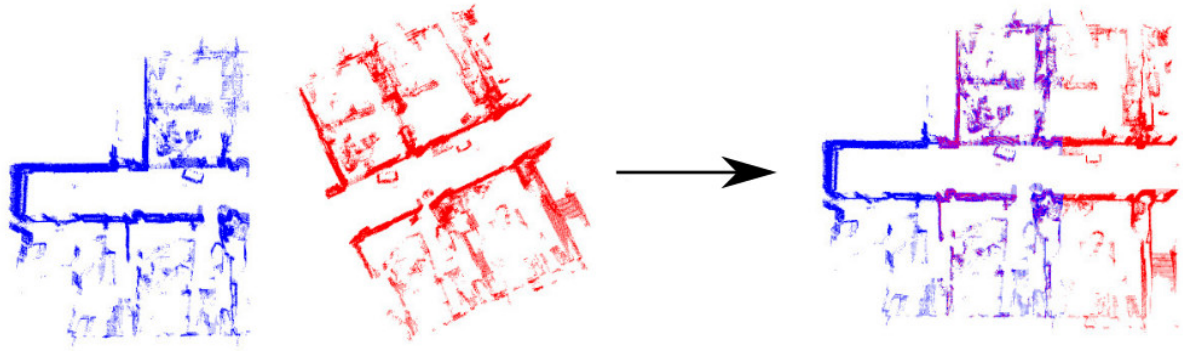
- $f(\cdot)$ określa model ruchu,
- $h(\cdot)$ to model obserwacji,
- M_r liczba pozycji danego robota,
- C liczba spotkań robotów, przy czym zbiór spotkań ma postać $\{c_k\}_{k=1}^C$.

Rozdział 8

Algorytm integracji map z wielu robotów

W niniejszym rozdziale przedstawiono opracowany i zaimplementowany na potrzeby tej pracy algorytm umożliwiający integrację map 3D z wielu robotów mobilnych. Algorytm działa dla przypadku, gdy nie są znane początkowe pozycje robotów i roboty nie spotykają się w trakcie eksploracji. Metoda działa w oparciu o wspólne części map, detekcję i deskrypcję cech oraz ich dopasowanie. Rozwiązanie wykorzystuje reprezentację środowiska w postaci drzew ósemkowych (octomap) i może być używane w heterogenicznych grupach robotów, ponieważ niepewności pomiarowe poszczególnych sensorów uwzględniane są w wartościach prawdopodobieństw zajętości poszczególnych węzłów map. Przykład działania zaprojektowanego algorytmu przedstawiono na rys. 8.1.

Ponadto w rozdziale przedstawiono przegląd dostępnych w literaturze metod integracji map oraz zaprezentowano wyniki przeprowadzonych badań.



Rysunek 8.1 Przykład łączenia dwóch map 3D opracowaną metodą

8.1 Wprowadzenie

Przeważnie podczas mapowania wielorobotowego każdy z robotów tworzy swoją własną mapę, w lokalnym układzie współrzędnych. Jednym z zagadnień związanych z mapowaniem wielorobotowym jest integracja map cząstkowych z poszczególnych robotów, w jedną, spójną, globalną mapę [7]. Jednak utworzenie globalnej mapy wymaga rozwiązania kilku problemów. Przede wszystkim, należy znaleźć transformacje między

poszczególnymi mapami lokalnymi, a następnie połączyć dane z map, tak aby uzyskać spójną reprezentację.

Metody integracji map można podzielić ze względu na moment, w którym uzyskuje się informacje o relacjach między układami odniesienia poszczególnych map [7]. Można wyróżnić trzy przypadki:

- pozycje startowe robotów są znane,
- pozycje startowe robotów są nieznane, ale zakłada się, że roboty spotykają się podczas eksploracji i posiadają możliwość określania swoich względnych pozycji,
- pozycje początkowe robotów są nieznane i roboty nie spotykają się w trakcie eksploracji.

W pierwszym przypadku wykorzystuje się początkowe pozycje robotów i ich systemy lokalizacji, aby w momencie wymiany informacji estymować wzajemne transformacje między mapami. Niestety, ten sposób może być obarczony znacznymi błędami, chociażby ze względu na niedokładności systemów lokalizacji robotów, a w szczególności systemów działających lokalnie.

Kolejnym pomysłem jest wykonywanie fuzji map, jedynie w momencie spotkań robotów, na podstawie wzajemnych detekcji. Ten sposób wymaga jednak dedykowanych systemów sensorów oraz metod służących do detekcji innych robotów. Pewną modyfikacją tej grupy metod jest estymacja odległości jedynie na podstawie pomiaru czasu przelotu sygnału między robotami. Dzięki temu, roboty nie muszą się widzieć, a jedynie być w zasięgu swojej aparatury radiowej.

Ostatni przypadek, dotyczy sytuacji, gdy pozycje początkowe robotów nie są znane i roboty nie spotykają się w trakcie eksploracji. Rozwiązanie tego problemu wymaga bardziej złożonych metod, których działanie opiera się na założeniu, że mapy cząstkowe mogą posiadać wspólne obszary. Zadaniem tych metod jest odnalezienie takich obszarów oraz ich wzajemnych pozycji. Bazując na wspomnianym założeniu, wyróżnia się dwa podejścia.

Pierwsze polega na przesyłaniu między robotami danych pomiarowych z czujników i próbie dopasowania ich do lokalnych map. Przykładem może być przesłanie odczytów ze skanera laserowego do innego robota, który dopasowuje dane do bieżącej mapy za pomocą filtra cząsteczkowego. Jednak w tym przypadku konieczne jest, aby obszar wspólny znajdował się w polu widzenia robota wysyłającego dane do dopasowania, w momencie zbierania tych danych.

Drugie podejście polega na przesyłaniu przetworzonych danych, przykładowo w postaci map. Wspólne obszary wykrywa się z wykorzystaniem metod detekcji i dopasowania cech. Postępowanie jest następujące, na mapach wykrywa się punkty charakterystyczne, opisuje cechy, a następnie dopasowuje do siebie cechy z różnych map, tak aby uzyskać transformację między mapami.

Proces dopasowania map jest znacznie bardziej wymagający niż standardowe mapowanie, wykorzystujące dopasowanie do siebie dwóch kolejnych odczytów z sensorów takich jak skanery laserowe, czy czujniki głębi. Głównym powodem jest występowanie znacznie większych przemieszczeń, czego przyczyną może być rozpoczęcie mapowania przez poszczególne roboty z zupełnie innych lokacji. W takim przypadku, lokalne algorytmy optymalizacji nie pozwalają uzyskać zadowalających rezultatów, a w celu dopasowania map wykorzystuje się metody globalne, odporne na lokalne minima.

8.2 Definicja problemu

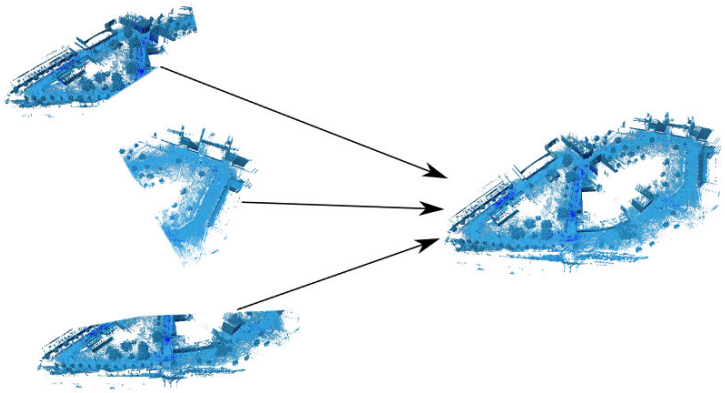
Rozważmy system N robotów $R = \{r_1, r_2, \ldots, r_n, \ldots, r_N\}$ poruszających się w przestrzeni $\mathbb{R}^3$, gdzie każdy robot tworzy lokalną mapę M_n, w lokalnym układzie współrzędnych T_n. Mapa określona jest jako zbiór N_n węzłów $M_n = \{m_1, m_2, \ldots, m_{N_n}\}$. Niech T_1^n oznacza transformację między lokalnymi układami współrzędnych map T_1 oraz T_n, pozwalającą transformować mapę M_n do układu współrzędnych mapy M_1, a

$$M_n' = T_1^n M_n \tag{8.1}$$

oznacza mapę M_n transformowaną do układu odniesienia mapy M_1. Integrację map można zdefiniować jako utworzenie jednego, spójnego modelu środowiska

$$M_{1,\ldots,N} = M_1 \cup M_2' \cup \ldots \cup M_N', \tag{8.2}$$

w oparciu o zbiór N modeli cząstkowych $M = \{M_1, \ldots, M_N\}$ (rys. 8.2).



Rysunek 8.2 Lokalne mapy utworzone przez różne roboty, połączone w jeden spójny model środowiska

8.2.1 Ograniczenie problemu do integracji tylko dwóch map

Na potrzeby dalszych rozważań problem zawężono do jedynie dwóch robotów, czyli dwóch map lokalnych. Zakłada się znajomość transformacji T_1^2 pozwalającej transformować model M_2 do układu współrzędnych modelu M_1. Integrację dwóch map określa się jako kompozycję map cząstkowych w układzie odniesienia pierwszej mapy

$$M_{1,2} = M_1 \cup T_2^1 M_2. \tag{8.3}$$

W przypadku większej liczby map, które będą integrowane, można ten proces przeprowadzić sekwencyjnie. Kolejność integracji poszczególnych par map w przypadku większej ich liczby nie powinna mieć wpływu na proces łączenia danych. Natomiast może mieć wpływ na ustalanie transformacji między nimi, ponieważ nie każda para map musi posiadać części wspólne. Jednym z rozwiązań może być przeszukiwanie dostępnych map w celu określenia, które z nich posiadają części wspólne.

8.3 Przegląd dostępnych rozwiązań

Metody mapowania środowiska są rozwijane od dłuższego czasu, dlatego też powstało kilka prac podsumowujących aktualny stan wiedzy [7, 142, 239, 261]. Jedną z nich jest praca [239], w której dokonano klasyfikacji i opisu wielu metod mapowania. Skupia się ona jednak na metodach przeznaczonych dla pojedynczych robotów.

Praca [7] dotyczy wyłącznie problemu łączenia map z wielu robotów, przez co zawiera rozbudowaną klasyfikację rozwiązań. Podział dokonywany jest względem różnych czynników, między innymi ze względu na wiedzę o układach odniesienia map, typ map, strukturę środowiska, czy moment, w którym podejmowana jest próba integracji map.

Kolejna praca przeglądowa [142] wprowadza podział metod łączenia map na bezpośrednie i pośrednie. W przypadku metod bezpośrednich wykorzystuje się dodatkową informację odnośnie transformacji między układami odniesienia map. Taką informacją mogą być znane początkowe pozycje robotów, ale również źródłem dodatkowej informacji może być detekcja innego robota, w momencie ich spotkania, przez system wizyjny lub inny zestaw sensorów. Natomiast metody pośrednie skupiają się również na znalezieniu transformacji między układami odniesienia map.

Metody bezpośrednie

Jedna z metod bezpośrednich została przedstawiona w pracy [240], gdzie wykorzystano przyrostowe tworzenie map w oparciu o metodę największej wiarygodności. Początkowe pozycje robotów nie są podawane wprost, ale autorzy założyli, że na początku wszystkie roboty są w zasięgu robota pełniącego rolę lidera grupy. Pozwala to na lokalizację innych robotów na głównej mapie i w rezultacie otrzymanie względnych pozycji początkowych.

W pracy [100] również wykorzystano metodę największej wiarygodności w procesie budowania mapy, ale w odróżnieniu od poprzedniej pracy, użyto innego typu reprezentacji środowiska. Opracowano mapy w postaci nakładających się na siebie warstw.

Inne podejście przedstawiono w pracy [200], gdzie wykorzystano filtr Kalmana do tworzenia lokalnych map, a także do ich łączenia w globalną mapę. Nadal jednak zakładano znane pozycje początkowe robotów.

Kolejne rozwiązanie oparte na filtrze Kalmana przedstawiono w [206]. Każdy z robotów tworzył lokalną mapę, a fuzja danych następowała jedynie w momencie ich spotkań. Autorzy wykorzystali także filtr Kalmana w procesie łączenia map.

W pracy [131] opisano rozwiązanie, w którym roboty generowały hipotezy odnośnie transformacji między nimi na podstawie bezpośrednich detekcji, w momencie spotkania. Przy czym dana hipoteza była potwierdzana dopiero, gdy te same roboty spotkały się po raz kolejny. Każde potwierdzenie hipotezy wiązało się z połączeniem map. Autorzy wykorzystali mapy metryczne rozbudowane o dodatkowe cechy, ułatwiające proces integracji.

Praca [266] przedstawia rozwiązanie oparte na danych z systemu wizyjnego. Podobnie jak poprzednio detekcja innego robota powodowała wygenerowanie zgrubnej transformacji. W tym przypadku jednak transformację od razu wykorzystano do przekształcenia jednej z map. Natomiast w celu poprawy dokładności metody wykorzystano dodatkowy krok polegający na poszukiwaniu wspólnych obszarów na wstępnie transformowanych mapach, detekcji znaczników i próbie ich dopasowania.

Innym pomysłem jest wykorzystanie filtrów cząsteczkowych, czego przykład można znaleźć w pracy [145]. Autorzy wykorzystali filtry cząsteczkowe wraz z techniką dopasowywania skanów laserowych, aby utworzyć dokładne mapy lokalne. Następnie w momencie spotkań robotów następowało łączenie map, oparte wyłącznie na dokładnej metodzie detekcji innych robotów.

Kolejną grupą bezpośrednich metod łączenia map są te wykorzystujące położenia znanych obszarów lub znaczników wspólnych na wielu mapach. Zakładano, że obszary lub znaczniki są znane i roboty posiadają dedykowane metody do ich wykrywania. W pracy [247] przedstawiono rozwiązanie, w którym robot za pomocą systemu wizyjnego wykrywa i rozpoznaje wspólne miejsca na kolejno tworzonych mapach. Następnie położenia tych obszarów wykorzystywane są podczas łączenia map. Zaprezentowana metoda dotyczy jednego robota i wielu sesji mapowania, ale rozwiązanie może być także zaadoptowane do systemu wielorobotowego.

Metody pośrednie

Metody pośrednie wykorzystywane są w sytuacji, gdy pozycje początkowe robotów nie są znane i nie zakłada się, że roboty będą się spotykać i wykrywać wzajemnie w trakcie eksploracji. Jest to trudniejszy przypadek od poprzednich i wymaga bardziej złożonych rozwiązań. W takich metodach, fuzja map opiera się na poszukiwaniu i dopasowywaniu do siebie wspólnych obszarów lub znaczników, które nie są znane a priori. Należy też rozpatrywać przypadek, w którym obszar wspólny na mapach nie istnieje lub nie zostaje poprawnie odnaleziony. W takim przypadku nie można połączyć map, ale metody muszą być w stanie wykrywać takie sytuacje, aby zapewnić odpowiedni poziom odporności na błędne integracje.

Jednym z pomysłów jest detekcja cech punktowych na mapach i dopasowanie ich do siebie w taki sposób, aby znaleźć transformację między mapami. W pracy [143] opisano system detekcji wspólnych obszarów na mapach sufitów pomieszczeń, przy czym mapy te były tworzone w oparciu o kamery wizyjne. Przedstawiony algorytm wykrywał nakładające się obszary i przeprowadzał proces dopasowania i estymacji transformacji.

W pracy [93] zaprezentowano rozwiązanie, w którym oprócz mapy metrycznej roboty tworzyły również obrazy środowiska skojarzone z danym obszarem. Podczas integracji lokalnych map robotów porównywano obrazy, aby uniknąć błędnych fuzji map. Ograniczeniem metody okazała się być niewystarczająca różnorodność wykonywanych obrazów, przez co rozwiązanie może być stosowane jedynie w specyficznych przypadkach.

Praca [23] opisuje rozwiązanie, które wykorzystuje metryki podobieństw obrazów w celu określenia, czy dana translacja i rotacja pozwala uzyskać spójną mapę z map lokalnych. W celu przyspieszenia obliczeń autorzy wykorzystują metodę losowego błądzenia do przeszukiwania przestrzeni możliwych transformacji.

Innym pomysłem jest wykorzystanie dopasowania skanów. W pracy [251] przedstawiono rozwiązanie dla map 2D wykorzystujące odczyty ze skanera laserowego. Do ekstrakcji i opisania cech wykorzystano deskryptor SIFT. Następnie dopasowano cechy do siebie z wykorzystaniem algorytmu ICP (ang. *Iterative Closest Point*) [21,87,90,208], który pozwala znaleźć transformację między dwoma zbiorami punktów. Ograniczeniem przedstawionej metody jest sam algorytm ICP, który działa lokalnie i może być stosowany jedynie w przypadku niewielkich przesunięć między zbiorami cech.

Praca [4] przedstawia metodę opierającą się na idei wirtualnego robota i wykorzystującą mapy metryczne, uzupełnione o cechy. W tym przypadku lokalne mapy z

poszczególnych robotów traktowane są jako skany laserowe wirtualnych robotów i są dopasowywane do bieżącej mapy. Metoda wykorzystuje filtr cząsteczkowy do fuzji map, natomiast cechy dołączone do mapy metrycznej używane są do znajdowania transformacji między ich układami odniesienia.

Metodę integracji map 2D, opierającą się na transformacji Hougha zaprezentowano w pracach [36, 141]. Metoda wykrywa krzywe geometryczne i pozwala znaleźć optymalną transformację, przy czym dobrze radzi sobie z lokalnymi minimami.

Odmienne podejście zostało zaprezentowane w pracy [117], gdzie oprócz mapy metrycznej wykorzystano informacje semantyczne w postaci planów kondygnacji. Roboty generują lokalne mapy metryczne wykorzystując SLAM, a następnie poprzez segmentację regionów tworzone są grafy. Lokalizacja robota określana jest na podstawie dopasowania grafu powstałego z mapy rastrowej do grafu związanego z planem kondygnacji. W konsekwencji pozwala to związać układ lokalny mapy robota z globalnym planem budynku.

Praca [78] opiera się na idei abstrakcyjnej reprezentacji map metrycznych 2D na potrzeby poszukiwania dopasowania. W tym celu autorzy dokonują geometrycznej dekompozycji map, czyli segmentacji regionów. Następnie poszukują podobieństw i generują hipotezy.

W pracy [122] przedstawiono metodę łączenia map w postaci struktur hierarchicznych składających się z obrazów środowiska. Wykorzystanie takich map zakłada, że roboty nie używają bezpośrednio danych o swoim położeniu, a jedynie określą dopasowanie do danych obrazów.

Obecnie, coraz częściej wykorzystuje się mapy 3D, między innymi dlatego, że jednostki obliczeniowe zdolne do przetwarzania znacznych ilości danych są coraz łatwiej dostępne. Nie bez znaczenia jest też rozwój metod przetwarzania równoległego, które stosunkowo łatwo można zastosować w pracy z danymi 3D. Mapy 3D mają wiele zalet względem map 2D. Przede wszystkim lepiej sprawdzają się w przypadku nieregularnego terenu, w budynkach wielopoziomowych, czy kiedy w środowisku, w którym porusza się robot znajduje się wiele przeszkód o nieregularnych kształtach [2]. Dodatkową zaletą map 3D, jest możliwość ich zastosowania w grupach robotów heterogenicznych [216], które mogą składać się z robotów jeżdżących, latających, czy kroczących. Jednak większość istniejących prac dotyczy problemu łączenia map 2D lub map 3D ze znanymi początkowymi pozycjami robotów. W pracach [109, 110] zaproponowano wykorzystanie metody ICP w celu znalezienia transformacji między mapami 3D w postaci octomap [99]. Mimo, że mapy przechowywane i integrowane są w postaci octomap, to proces poszukiwania transformacji wymaga konwersji map do zbiorów punktów. Natomiast w pracy [53] przedstawiono rozwinięcie wcześniej wspomnianej metody, działające na octomapach.

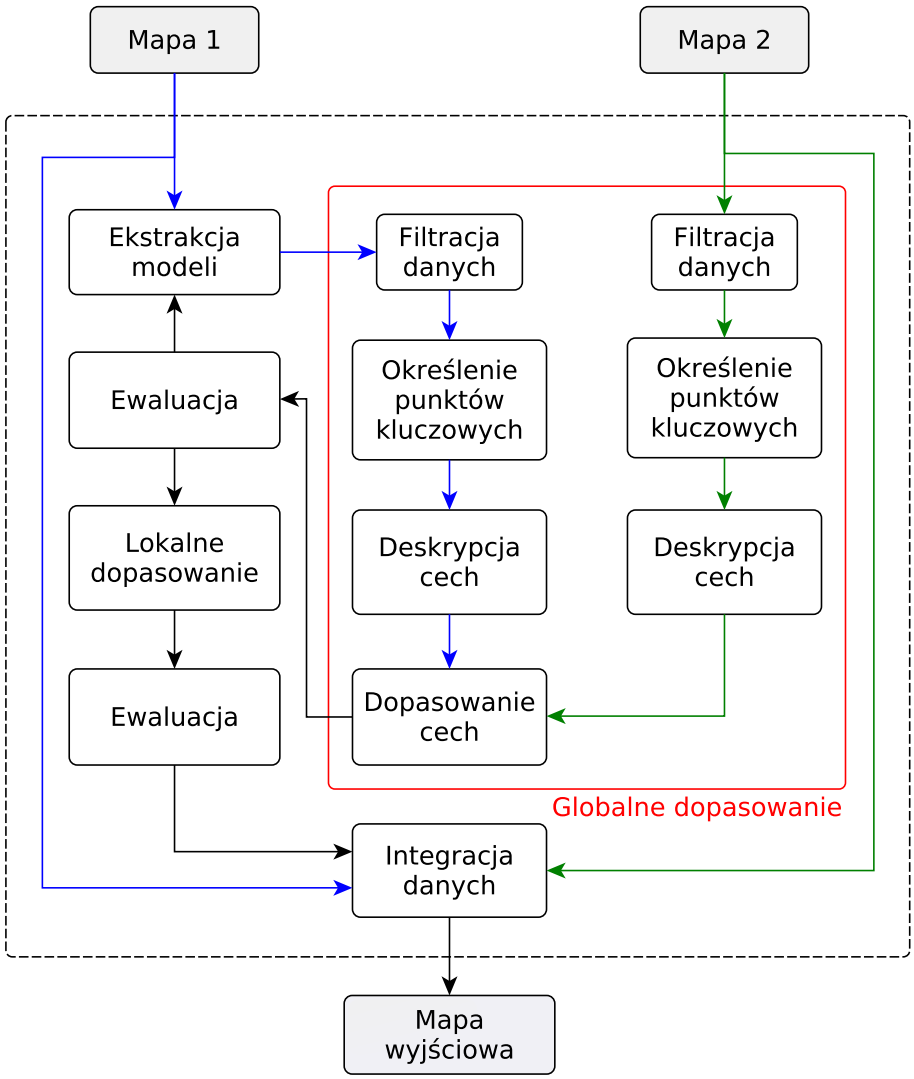
Inne podejście dla map 3D zaprezentowano w pracy [25], gdzie wykorzystuje się rozwiązanie podobne do metod SLAM opartych na grafie. Autorzy utworzyli graf, którego węzły tworzą chmury punktów z lokalnymi układami odniesienia, natomiast krawędzie grafu określają transformacje między poszczególnymi mapami cząstkowymi. Zaletą takiego podejścia jest możliwość wykrywania pętli i ich optymalizacja.

8.4 Metoda integracji map przy nieznanych początkowych pozycjach robotów

Na potrzeby pracy opracowano metodę integracji map 3D. Przedstawiona metoda nie wymaga początkowej informacji o wzajemnych transformacjach między mapami. Zakłada się, że mapy cząstkowe mają takie same skale i zawierają wspólny obszar. W procesie początkowego dopasowania, jedna z map jest dzielona na bloki, które następnie dopasowywane są do drugiej mapy. Kolejny krok polega na wyborze najlepszego rozwiązania, w oparciu o zdefiniowane funkcje oceny jakości dopasowania.

Metoda została zaprojektowana pod kątem działania z mapami ósemkowymi, ale wykorzystuje również chmury punktów na pewnym etapie przetwarzania. Dualna reprezentacja pozwala wykorzystać zalety każdej z reprezentacji i wydajnie wykonać wybrane operacje. Kosztem w tym przypadku jest zwiększone zużycie pamięci.

Proces przetwarzania danych został zaprezentowany na rys. 8.3. Na wejściu algoryt-



Rysunek 8.3 Przetwarzanie danych w opracowanej metodzie integracji map

mu znajdują się dwie mapy, przy czym zakłada się, że posiadają wspólny obszar, który jest konieczny do poprawnego wyznaczenia transformacji między nimi. Jedna z map jest używana jako scena do której dopasowywane będą modele, natomiast z drugiej

mapy te modele są generowane. Zarówno dla sceny jak i modelu kilka kroków przetwarzania jest wspólnych. Należą do nich: filtracja, wykrywanie punktów kluczowych, czy opisywanie cech przez obliczanie deskryptorów. Bazując na deskryptorach i wspomnianym założeniu odnośnie obszaru wspólnego, mapy są do siebie dopasowywane z wykorzystaniem odpowiedniego algorytmu.

Z powodu podziału jednej z map na wiele modeli, rezultatem początkowego dopasowania jest zbiór n hipotez $H = \{h_1, \ldots, h_n\}$. W oparciu o metryki jakości, wybierane jest z nich najlepsze rozwiązanie. Rozwiązaniem jest transformacja pozwalająca transformować jedną mapę do układu współrzędnych drugiej mapy. W kolejnym kroku, rozwiązanie jest poprawiane lokalnymi metodami dopasowania. Ostatnim krokiem, jeżeli chociaż jedna hipoteza zostanie zaakceptowana, jest połączenie danych z poszczególnych map. Rezultatem rozważanej metody jest zintegrowana mapa, będąca spójnym opisem środowiska, w którym poruszają się roboty.

W dalszej części opisano dokładniej działanie metody integracji map. W trakcie opisu korzystano z pojęć powszechnie używanych w dziedzinach związanych z przetwarzaniem wizji komputerowej, w szczególności pojęć modelu i sceny. Model i scena są to dwa zbiory danych, przykładowo w postaci zbiorów punktów, przy czym model jest zbiorem, który jest dopasowywany do drugiego zbioru, czyli sceny.

8.5 Estymacja transformacji między mapami na podstawie dopasowania cech

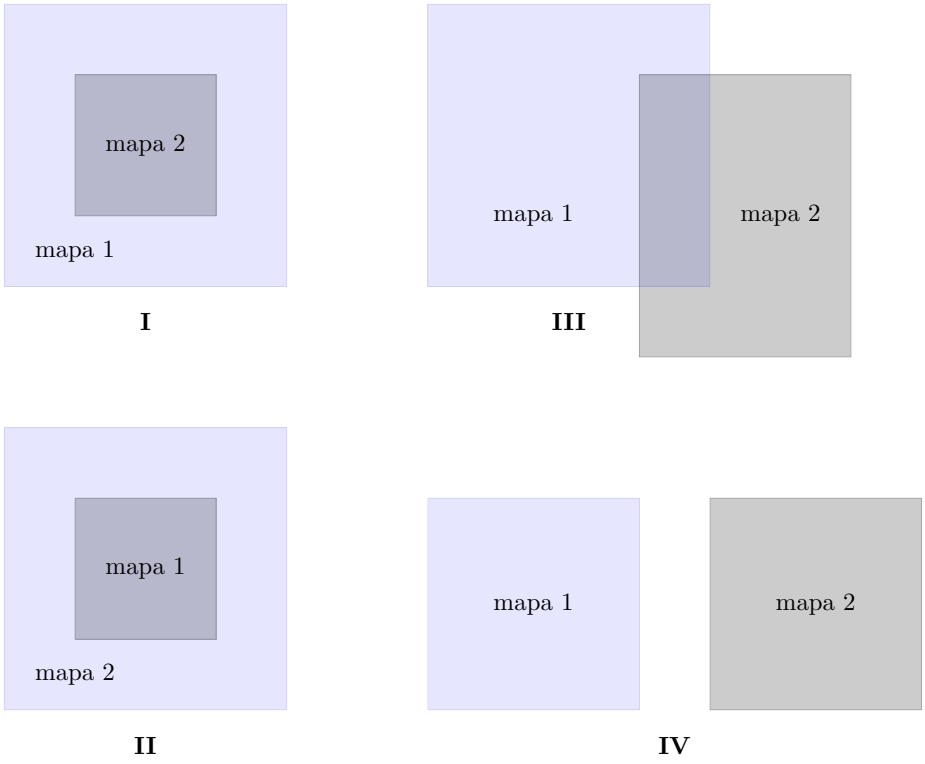
8.5.1 Wybór modelu i sceny

Istotną kwestią jest wybór, która z dwóch map wejściowych będzie sceną, a z której będą pobierane modele. Ze względów optymalizacyjnych, jako scenę wybiera się mniejszą z map. Powodem takiego działania jest fakt, że w przypadku sceny należy znaleźć wszystkie punkty kluczowe i odpowiednio opisać ich sąsiedztwo. Jest to stosunkowo kosztowne obliczeniowo zadanie. W przypadku mapy, z której pobierane są modele, opisywanie cech wykonywane jest jedynie dla aktualnie przetwarzanych modeli. Dlatego znacznie efektywniej jest opisać cechy mniejszej sceny. Pozostaje jednak pytanie, jakie kryteria należy przyjąć podczas porównywania map. Jedną z możliwości jest wykorzystanie rozmiarów geometrycznych map, ale porównanie może też opierać się na liczbie węzłów poszczególnych map. Dobór rozwiązania zależy przede wszystkim od tego jak wiele cech znajduje się na danym obszarze środowiska.

8.5.2 Ekstrakcja modelu

Rozpatrywane są następujące przypadki relacji między dwoma mapami (rys. 8.4):

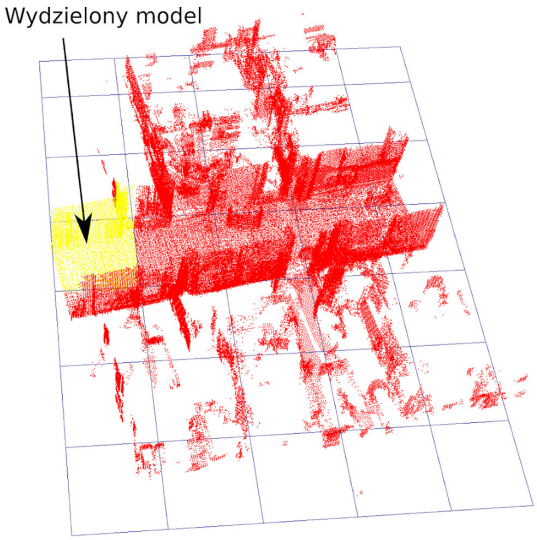
- mapa 2 może zawierać się w mapie 1,
- mapa 1 może zawierać się w mapie 2,
- mapy 1 i 2 mają jedynie wspólny fragment,
- mapy 1 i 2 nie posiadają części wspólnych.



Rysunek 8.4 Przypadki nakładania się map (I-III) lub braku części wspólnych (IV)

Na potrzeby metody integracji, interesujące są jedynie trzy pierwsze przypadki, ponieważ w ostatnim przypadku, czyli gdy mapy nie mają części wspólnej, estymacja transformacji nie jest możliwa.

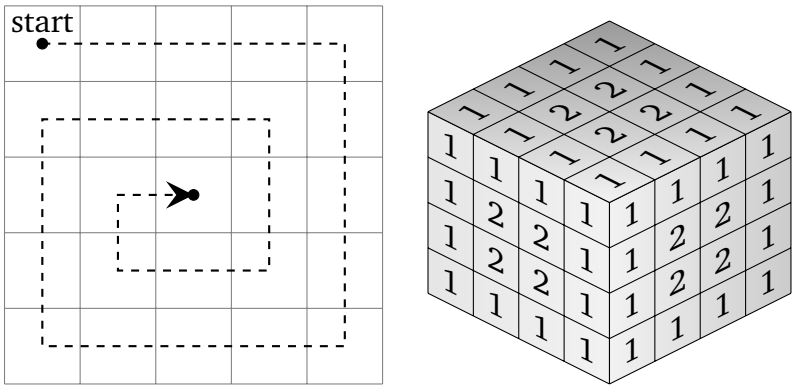
Jedną z pierwszych operacji metody integracji map jest podział pierwszej mapy na bloki, które pełnią rolę modeli (rys. 8.5). Cel takiego podziału mapy jest następujący.



Rysunek 8.5 Przykład ekstrakcji modelu z jednej z map

Zakładając, że roboty poruszają się po podłożu zbliżonym do płaszczyzny, wysokość mapy w osi Z jest znacznie mniejsza niż długość i szerokość (osie X i Y). Dodatkowo

zakłada się, że mapy reprezentują różne obszary otoczenia i mimo istnienia wspólnej części, żadna z nich nie zawiera się w całości w drugiej. W takim przypadku obszar wspólny map musi zawierać granice map. Z tego powodu, przetwarzanie rozpoczynane jest od bloków (modeli) znajdujących się w zewnętrznej części mapy, czyli zawierających granice. Następnie wybierane są bloki znajdujące się coraz bliżej środka, w sposób spiralny (rys. 8.6).



(a) podział 2D (b) podział 3D

Rysunek 8.6 Kolejność sprawdzania wyodrębnionych modeli

W przypadku, gdy mapa ma znaczną wysokość, czyli wykonywana jest na przykład pojazdami latającymi, czy reprezentuje budynek wielopoziomowy, podział na modele może zostać zrealizowany na podstawie siatki 3D (rys. 8.6).

Metoda pozwala stosunkowo szybko znaleźć rozwiązanie, ponieważ nie ma potrzeby przetwarzania wszystkich bloków. Jeżeli model z zewnętrznej części mapy spełnia kryteria dopasowania, przetwarzanie dalszych bloków zostaje wstrzymane. W przypadku, gdy mapa z której podbierany jest model zawiera wiele informacji (cech), czyli jest mapą gęstą (ang. *dense*), wtedy przetwarzanie zatrzymywane jest stosunkowo szybko. Jeżeli natomiast mapa nie posiada wielu cech w pobliżu granic (mapa rzadka, ang. *sparse*), wtedy przetwarzanie jest kontynuowane aż do osiągnięcia środkowego modelu. Podział jednej mapy na bloki pozwala w łatwy sposób na zrównoleglenie obliczeń.

8.5.3 Przygotowanie danych wejściowych

Filtracja danych wejściowych

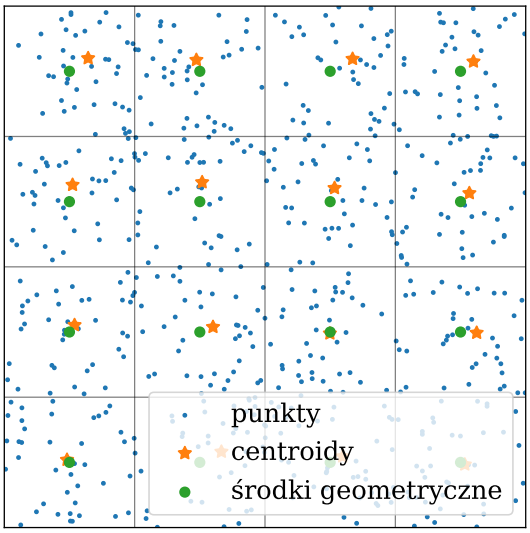
W procesie łączenia map ważne jest odpowiednie przygotowanie danych wejściowych. Krok ten dotyczy dwóch map, zarówno sceny jak i wyodrębnionych modeli. Filtracja realizowana jest w kilku etapach. Pierwszym z nich jest zastosowanie filtru odrzucającego punkty znajdujące się poza użytecznym obszarem, przykładowo mapa jest docinana w osi Z, aby usunąć podłoże i zredukować liczbę punktów na potrzeby dalszych obliczeń.

Filtr wokselowy

W następnym kroku zmniejszana jest rozdzielczość mapy za pomocą filtra wokselowego. Ideą takiego rozwiązania jest podział przestrzeni na woksele o określonym rozmiarze i określenie centroidu, uśredniającego wszystkie inne punkty należące do tego

woksela (rys. 8.7). Zakładając, że w danym wokselu znajduje się n punktów, centroid oblicza się następująco

$$p_c = \frac{1}{n} \sum_{i=0}^{n} p_i. \tag{8.4}$$



Rysunek 8.7 Przykład działania filtra wokselowego w rzucie 2D na woksele

Filtr statystyczny

Kolejnym krokiem jest wykonanie analizy statystycznej punktów w oparciu o ich sąsiedztwo oraz usunięcie tych, które nie spełniają założeń [212]. Jednym z rozwiązań jest wykorzystanie filtru obliczającego dystrybucję odległości do punktów należących do sąsiedztwa. Zakładając, że rozkład odległości powinien mieć postać rozkładu normalnego, odrzucane są punkty nie spełniające tego warunku.

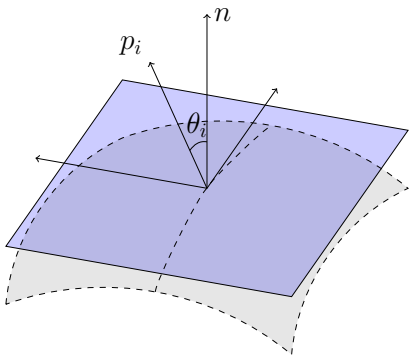
Estymacja wektorów normalnych

Na potrzeby metod opisu cech otoczenia estymowane są wektory normalne w punktach kluczowych (rys. 8.8). Algorytm dla każdego punktu znajduje sąsiednie punkty, zgodnie z przyjętym promieniem poszukiwań. Następnie, estymowana jest dla tych punktów płaszczyzna z wykorzystaniem metody najmniejszych kwadratów [210]. W oparciu o równanie estymowanej płaszczyzny $ax + by + cz + d = 0$, obliczany jest wektor normalny $n = [a, b, c]$.

8.5.4 Detekcja i deskrypcja cech

Wykrywanie punktów charakterystycznych

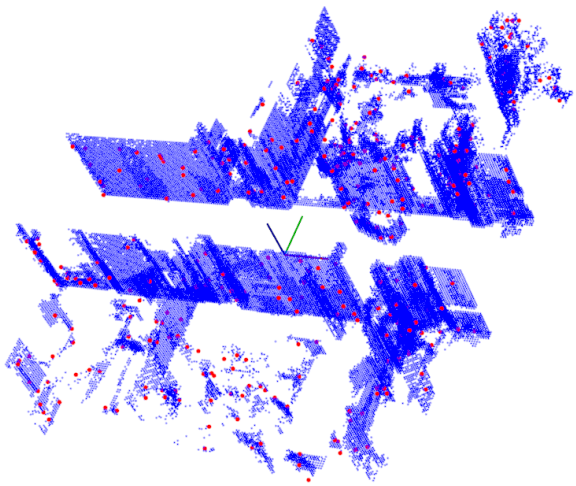
Opisywanie cech środowiska w postaci deskryptorów jest zadaniem złożonym obliczeniowo. Dlatego też, deskryptory są obliczane jedyne w wybranych punktach charakterystycznych (ang. *key points*). Właściwie wybrane punkty kluczowe powinny różnić



Rysunek 8.8 Wektor normalny w danym punkcie oraz kąt θ_i między wektorem normalnym i punktem p_i należącym do sąsiedztwa

się od siebie i jednocześnie być powtarzalne, aby umożliwić porównywanie cech środowiska, biorąc pod uwagę występowanie szumów i kiedy dany wycinek środowiska obserwowany jest z różnych perspektyw. Do popularnych metod wykrywania punktów kluczowych w przestrzeni 3D należą detektory: NARF (ang. *Normal Aligned Radial Feature*) [230], ISS (ang. *Intrinsic Shape Signatures*) [265], SIFT3D (ang. *Scale Invariant Feature Transform 3D*) [88], SUSAN (ang. *Smallest Univalue Segment Assimilating Nucleus*) [88], a także zmodyfikowany detektor Harrisa [89]. Innym, mniej złożonym obliczeniowo rozwiązaniem jest wykorzystanie równomiernego samplowania (*ang. uniform sampling*).

Na potrzeby przedstawianej metody wykorzystano detektor punktów charakterystycznych ISS. Detektor ten opiera się na badaniu wartości własnych macierzy kowariancji obliczonej dla otoczenia punktu kluczowego k. Na tej podstawie określa się istotność danego punktu. Przykład działania detektora ISS w postaci wykrytych punktów przedstawiono na rys. 8.9.



Rysunek 8.9 Punkty charakterystyczne wyznaczone metodą ISS

Zakładając, że zbiór punktów $X = \{x_1, x_2, \ldots, x_N\}$ opisuje otoczenie punktu cha-

rakterystycznego k, macierz kowariancji dla tego otoczenia oblicza się następująco

$$\Sigma_k = \frac{1}{N}\sum_{i=1}^{N}(x_i - p_k)(x_i - p_k)^T, \tag{8.5}$$

gdzie

$$p_k = \frac{1}{N}\sum_{i=1}^{N} x_i \tag{8.6}$$

określa wartość średnią.

Macierz kowariancji Σ_k jest dekomponowana na wartości własne z wykorzystaniem metody EVD (ang. *Eigenvalue Decomposition*). Ustawiając wartości własne w kolejności malejącej

$$\lambda_1 < \lambda_2 < \lambda_3, \tag{8.7}$$

do zbioru punktów charakterystycznych zalicza się jedynie te punkty, których wartości własne spełniają poniższe warunki

$$\frac{\lambda_2}{\lambda_1} \leq \gamma_{12} \wedge \frac{\lambda_3}{\lambda_2} \leq \gamma_{23}, \tag{8.8}$$

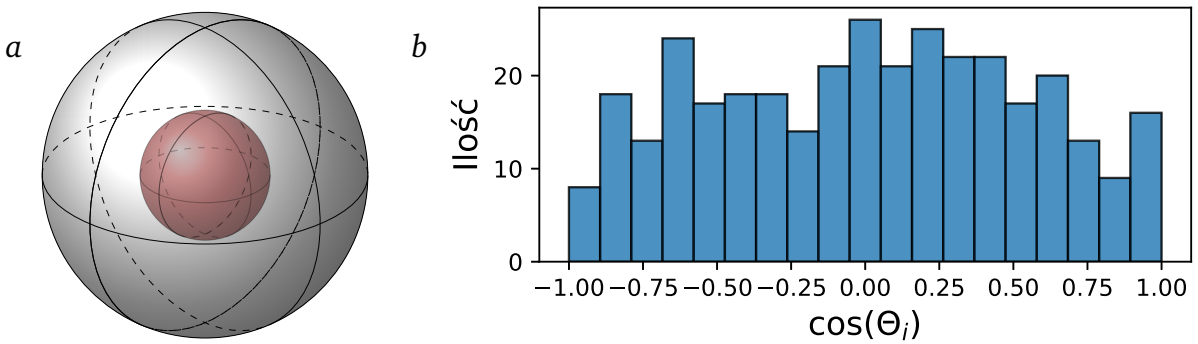
gdzie γ_{12} i γ_{23} to parametry metody. Ma to na celu odrzucenie tych punktów, dla których nie można jednoznacznie wyznaczyć lokalnego układu odniesienia, czyli dla których wartości własne macierzy kowariancji mają zbliżone wartości.

Obliczenie lokalnych deskryptorów cech

W celu odnalezienia podobnych obszarów na mapach, wszystkie wykryte cechy powinny być opisane w możliwie spójny i zwięzły sposób, aby umożliwić ich efektywne porównywanie. Jak już wspomniano, w celu optymalizacji, deskryptory obliczane są jedynie w wyznaczonych wcześniej punktach kluczowych. Lokalne deskryptory opisują jedynie lokalne sąsiedztwo danego punktu, zazwyczaj w postaci sferycznej lub cylindrycznej. Deskryptory globalne natomiast kodują informację zawartą w całym zbiorze punktów 3D. Szeroko stosowaną metodą opisu jest algorytm FPFH (ang. *Fast Point Feature Histograms*), który jest przyśpieszoną wersją algorytmu PFH (ang. *Point Feature Histograms*) [209]. W przypadku algorytmu algorytmu PFH dla chmury n punktów złożoność obliczeniowa wynosi $\mathcal{O}(nk^2)$, gdzie k określa liczbę punktów w sąsiedztwie, natomiast w metodzie FPFH złożoność udało się zredukować do $\mathcal{O}(nk)$.

Inne popularne deskryptory lokalne to SI (ang. *Spin Image*), USC (ang. *Unique Shape Context*), SHOT (ang. *Signature of Histograms of Orientations*), SH (ang. *Spectral Histogram*) [88], oraz globalne GSH (ang. *Global Structure Histograms*), ESF (ang. *Ensemble of Shape Functions*).

Na potrzebny pracy zastosowano lokalny deskryptor SHOT [245] wykorzystujący otoczenie (ang. *support*) sferyczne. W celu obliczenia deskryptora SHOT, określa się lokalne układy odniesienia (LRF, ang. *Local Reference Frame*) dla punktów kluczowych. W tym celu oblicza się macierz kowariancji na podstawie punktów z sąsiedztwa sferycznego i dokonuje jej dekompozycji w oparciu o wartości własne (EVD, ang. *Eigenvalue Decomposition*). W kolejnym kroku sferyczne otoczenie punktu dzieli się na 32 części



Rysunek 8.10 Podział sferycznego sąsiedztwa punktu na 32 części (a) oraz histogram zliczający punkty dla których wartość $\cos\theta_i$ należy do danego przedziału (b)

(rys. 8.10). Dla każdego segmentu metoda oblicza lokalny histogram reprezentujący rozkład $\cos\theta_i$, czyli kąta między wektorem normalnym każdego punktu w sąsiedztwie i wektorem normalnym w punkcie dla którego obliczany jest deskryptor (rys. 8.8). W kolejnym kroku, lokalne histogramy łączone są w wektor, który opisuje dany punkt.

Jedną z zalet deskryptora SHOT jest możliwość wykorzystania informacji o kolorze punktu otrzymanej na przykład z sensora RGB-D. W tym przypadku deskryptor może być określony następująco

$$D(k) = D_s(k) \cup D_t(k), \tag{8.9}$$

gdzie:

- $D_s(k)$ opisuje kształt otoczenia danego punktu,
- $D_t(k)$ opisuje teksturę otoczenia danego punktu.

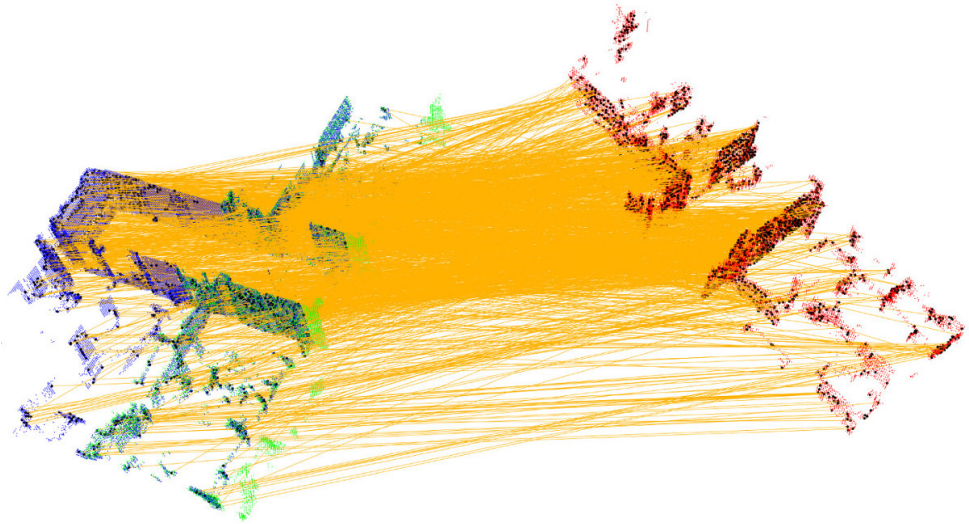
8.5.5 Globalne dopasowanie cech metodą probabilistyczną (SAC)

Dopasowanie cech przeprowadza się w oparciu o obliczone wcześniej deskryptory dla punktów kluczowych. Pierwszy zbiór deskryptorów D_M jest obliczany dla punktów kluczowych z wyodrębnionego modelu $M' = \{m_i \mid m_i \in \mathbb{R}^3, i = 1, \ldots, N_M\}$, a drugi zbiór D_S dla punktów kluczowych sceny $S' = \{s_i \mid s_i \in \mathbb{R}^3, i = 1, \ldots, N_S\}$. Dopasowywanie zbiorów D_M i D_S polega na poszukiwaniu podobieństw deskryptorów. Oznacza to, że dla każdego punktu ze zbioru M' wyszukiwany jest odpowiedni punkt w zbiorze S', ze zbliżonym deskryptorem, co oznacza zwiększone prawdopodobieństwo tego, że obydwa deskryptory opisują ten sam wycinek otoczenia. Po utworzeniu par deskryptorów (rys. 8.11), oblicza się transformację, która minimalizuje odległości między deskryptorami.

Jedną z metod dopasowywania deskryptorów jest metoda SAC-IA (*ang. Sample Consensus Initial Alignment*) [211]. Zasada działania algorytmu jest zbliżona do algorytmu RANSAC (*ang. Random Sample Consensus*) [47] i polega na losowym dopasowaniu 'k'-najbliższych sąsiadów.

Kroki metody są następujące:

- pobranie k punktów ze zbioru M' takich, że ich wzajemne odległości są większe od wartości progowej d_{min} i dodanie ich do zbioru $P = \{p_i \mid p_i \in \mathbb{R}^3, i = 1, \ldots, k\}$,



Rysunek 8.11 Wizualizacja par deskryptorów dopasowanych przez wyszukiwanie najbliższych deskryptorów w drzewie KD

- dla każdego punktu p_i ze zbioru P utworzenie listy punktów z podobnymi deskryptorami w S' i losowo wybierz z nich jeden punkt, który utworzy parę (p_i, q_i),
- dla każdej pary punktów (p_i, q_i) określenie transformacji między punktami i obliczenie metryki błędu.

Następnie, powyższe kroki są powtarzane do momentu spełnienia kryterium końcowego w postaci maksymalnej liczby iteracji lub maksymalnej wartości metryki błędu.

8.5.6 Globalne dopasowanie cech metodą grupowania na podstawie zgodności geometrycznej (GCC)

Metoda GCC (ang. *Geometry Consistency Clustering*) [38], czyli metoda grupowania w oparciu o zgodność geometryczną, wyszukuje grupy odpowiadających sobie punktów, w oparciu o ich położenia. Metoda ta składa się z kilku kroków:

1. Znalezienie listy par odpowiadających sobie punktów,
2. Grupowanie par punktów w oparciu o zgodność geometryczną,
3. Znalezienie największej grupy par punktów,
4. Estymacja transformacji.

W pierwszym kroku, w oparciu o podobieństwa deskryptorów skojarzonych z poszczególnymi punktami znajduje się pary odpowiadających sobie punktów

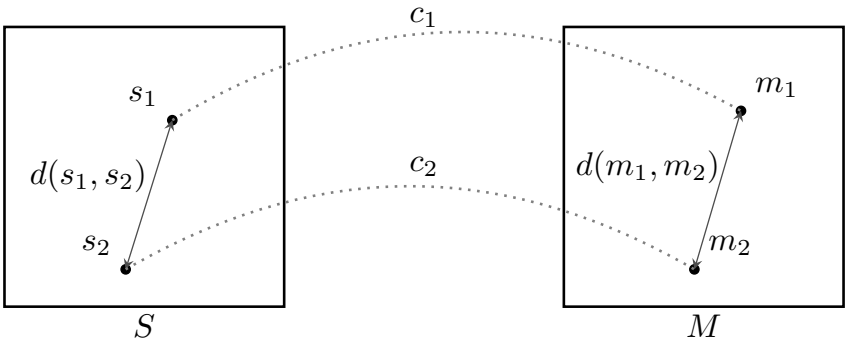
$$c_i = \{s_i, m_i\}, \tag{8.10}$$

gdzie s_i to punkt pierwszej mapy $S = \{s_i \mid s_i \in \mathbb{R}^3, i = 1, \ldots, N_S\}$, a m_i punkt drugiej drugiej mapy $M = \{m_i \mid m_i \in \mathbb{R}^3, i = 1, \ldots, N_M\}$. Zwykle w celu przyśpieszenia poszukiwań buduje się drzewo deskryptorów. Rezultatem poszukiwania podobnych deskryptorów jest utworzenie listy par skojarzonych punktów

$$L = \{c_1, c_2, \ldots, c_N\}, \tag{8.11}$$

gdzie N oznacza liczbę par punktów.

W kolejnym kroku, realizowane jest grupowanie punktów w oparciu o zależności geometryczne między tymi punktami (rys. 8.12). Grupowanie polega na tworzeniu pod-



Rysunek 8.12 Zależności geometryczne między parami punktów z dwóch zbiorów

zbiorów L_k zbioru L takich, że

$$\underset{c_i,c_j \in L_k}{\forall} d(c_i, c_j) \leq t_{gc}, \tag{8.12}$$

gdzie $d(c_i, c_j)$ to miara zgodności dwóch par punktów c_i oraz c_j zdefiniowana jako

$$d(c_i, c_j) = |d(s_i, s_j) - d(m_i, m_j)|, \tag{8.13}$$

a $d(s_i, s_j)$ oraz $d(m_i, m_j)$ to odpowiednio odległości między punktami w zbiorze S i M.

Kolejnym etapem jest znalezienie największego podzbioru wśród wszystkich wygenerowanych podzbiorów. Zakłada się, że istnieje największe prawdopodobieństwo, że największy zbiór reprezentuje poprawne dopasowanie między mapami. W ostatnim kroku, w oparciu o wspomniany, największy podzbiór par punktów, spełniających zależności geometryczne, estymuje się transformację. Można w tym celu użyć między innymi metody najmniejszych kwadratów.

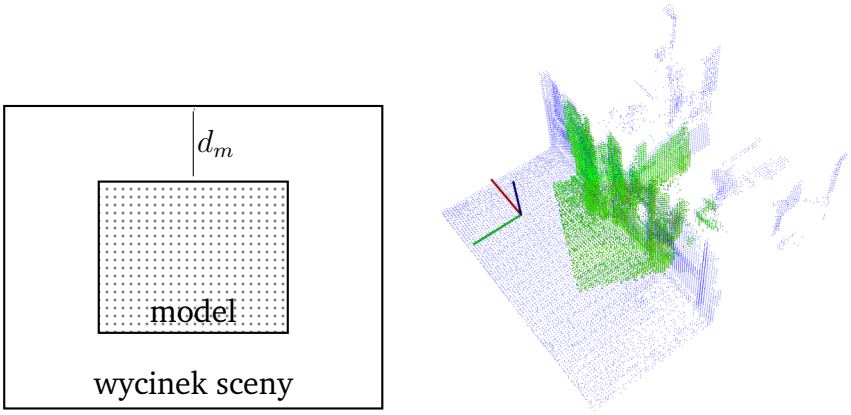
W pracy [257] można znaleźć szczegółową analizę wydajności wybranych metod dopasowania deskryptorów, w tym metody probabilistycznej SAC i metody GCC.

8.5.7 Lokalne dopasowanie algorytmem ICP

W końcowym etapie procesu estymacji transformacji między mapami, obliczana jest lokalna korekta. W tym celu wykorzystywano algorytm ICP lub metodę opartą na NDT.

Na potrzeby estymacji transformacji, scena jest przycinana do rozmiaru modelu powiększonego o pewien promień d_m (rys. 8.13). Następnie, do poprawienia transformacji między sceną $S = \{s_i \mid s_i \in \mathbb{R}^3, i = 1, \ldots, N_S\}$ oraz modelem $M = \{m_i \mid m_i \in \mathbb{R}^3, i = 1, \ldots, N_M\}$ wykorzystuje się algorytm ICP. Rozwiązanie opiera się na dopasowaniu jednej mapy M do drugiej S zwanej sceną, przez wyszukiwanie najbliższych sąsiadów w dwóch zbiorach punktów i minimalizacji odległości między parami punktów. W metodzie ICP po określeniu takiej transformacji, transformuje się model, a następnie powtarza proces od początku. Proces ten jest powtarzany wielokrotnie, do momentu uzyskania zadowalających rezultatów.

Kroki k-tej iteracji algorytmu ICP są następujące:



Rysunek 8.13 Scena przycięta do rozmiarów modelu powiększonego o promień d_m. Po prawej stronie znajduje się model dopasowany do sceny

- $\forall m_i^k \in M$ znalezienie najbliższego punktu (najbliższego sąsiada) $s_i^k \in S$,
- minimalizacja funkcji celu wyrażającej odległości między parami odpowiadających sobie punktów, mającą postać problemu najmniejszych kwadratów

$$E(R,t) = \frac{1}{N_M} \sum_{i=1}^{N_M} \left\| Rm_i + t - s_i^k \right\|^2 . \tag{8.14}$$

  Do minimalizacji funkcji wykorzystuje się metodę SVD,
- transformacja modelu $M^{k+1} = RM^k + t^k$,
- zakończenie działania, jeżeli zostały osiągnięte warunki końcowe. Jednym z warunków może być otrzymanie wartości błędu, czyli średniej odległości między parami punktów, poniżej ustalonej wartości progowej τ.

Rezultat jest uzyskiwany przez wielokrotne powtarzanie powyższych kroków.

8.5.8 Lokalne dopasowanie algorytmem OICP

W przypadku standardowej metody ICP, stosowanej do map posiadających informację o prawdopodobieństwie zajętości, ta informacja jest częściowo tracona. Pewne rozwiązanie tego problemu zostało zaprezentowane w pracy [263], gdzie przedstawiono metodę OICP (ang. *Occupancy Iterative Closest Point*), będącą modyfikacją standardowej metody ICP.

W metodzie OICP, optymalizowana wartość między dwoma punktami

$$d(m_i, s_i) = d_e(m_i, s_i)^2 + \omega d_f(m_i, s_i)^2 \tag{8.15}$$

składa się z dwóch elementów, czyli odległości euklidesowej zdefiniowanej jako

$$d_e(m_i, s_i) = \left\| Rm_i + t - s_i \right\| , \tag{8.16}$$

oraz odległości między cechami

$$d_f(m_i, s_i) = \left\| p_{m_i} - p_{s_i} \right\| , \tag{8.17}$$

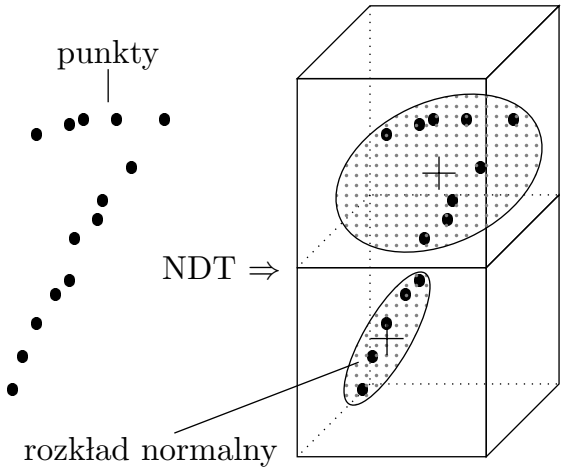
rozumianymi jako prawdopodobieństwa zajętości, a także współczynnika skalującego ω. Analogicznie jak w standardowej metodzie R oraz t oznaczają odpowiednio macierz rotacji i wektor translacji, natomiast p_{m_i} oraz p_{s_i} odpowiednio prawdopodobieństwa zajętości punktów m_i oraz s_i. Ostatecznie, optymalizowana funkcja przyjmuje następującą postać

$$E(R,t) = \frac{1}{N_M}\sum_{i=1}^{N_M} d(m_i, s_i) = \frac{1}{N_M}\sum_{i=1}^{N_M} \|Rm_i + t - s_i\|^2 + \omega \|p_{m_i} - p_{s_i}\|^2 . \qquad (8.18)$$

Pozostałe kroki są dokładnie takie same w przypadku jak w przypadku standardowego algorytmu ICP.

8.5.9 Lokalne dopasowanie algorytmem opartym na NDT

Na potrzeby lokalnej korekcji transformacji między wycinkami map, wykorzystano również metodę dopasowania opartą na reprezentacji NDT [234, 235]. Podejście NDT opiera się na podziale przestrzeni na woksele i reprezentacji danych w wokselach za pomocą rozkładów normalnych (rys. 8.14). Szerzej reprezentację tę omówiono w rozdziale 3.



Rysunek 8.14 Transformacja punktów do rozkładów normalnych w poszczególnych wokselach

Zakładając poszukiwanie transformacji między dwoma mapami w postaci zbiorów punktów: pierwszą (referencyjną) i drugą (wejściową), kroki metody są następujące:

1. Zastosowanie NDT do referencyjnej mapy,
2. Inicjalizacja wektora parametrów,
3. Transformacja punktów zgodnie z wektorem parametrów,
4. Przyporządkowanie do każdego punktu z mapy wejściowej woksela z rozkładem normalnymi,
5. Estymacja wektora parametrów, z wykorzystaniem algorytmu optymalizacji nieliniowej Newtona,

6. Zakończenie działania, jeżeli kryterium zbieżności zostało spełnione,

7. Powrót do kroku nr 3.

W pierwszym kroku, stosuje się operację NDT do mapy referencyjnej, czyli w każdym wokselu estymowane są parametry rozkładu normalnego. Zakładając, że zbiór $X_k = \{x_{k,1}, x_{k,2}, \ldots, x_{k,N_k}\}$, zawierający N_k punktów, można przypisać do woksela k, oblicza się parametry rozkładu normalnego, czyli wartość średnią

$$p_k = \frac{1}{N_k} \sum_{i=1}^{N_k} x_{k,i}, \tag{8.19}$$

oraz macierz kowariancji

$$\Sigma_k = \frac{1}{N_k} \sum_{i=1}^{N_k} (x_{k,i} - p_k)(x_{k,i} - p_k)^T. \tag{8.20}$$

Prawdopodobieństwo znalezienia próbki w punkcie x definiuje się następująco

$$p(x) \sim \exp\left(-\frac{(x - p_k)^T \Sigma_k^{-1} (x - p_k)}{2}\right). \tag{8.21}$$

Wektor poszukiwanych parametrów reprezentujących transformację ma postać

$$q = (t_x, t_y, t_z, \alpha, \beta, \gamma)^T, \tag{8.22}$$

gdzie t_x, t_y, t_z to elementy wektora translacji, a α, β, γ to kąty rotacji. Wektor ten jest inicjalizowany wartościami zerowymi $q = (0, 0, 0, 0, 0, 0)^T$. Następnie, punkty mapy wejściowej są transformowane zgodnie z wektorem q

$$x_i' = T(x_i, q). \tag{8.23}$$

W kolejnym kroku, do każdego punktu z drugiej mapy (wejściowej) przyporządkowuje się odpowiedni woksel. Umożliwia to obliczenie wartości funkcji określającej jakość dopasowania i jej optymalizację

$$E(X, q) = \sum_{i=1}^{N} \exp\left(-\frac{(x_i' - p_i)^T \Sigma_i^{-1} (x_i' - p_i)}{2}\right), \tag{8.24}$$

gdzie N określa liczbę punktów w zbiorze wejściowym. Ponieważ algorytm Newtona umożliwia poszukiwanie jedynie minimum funkcji, optymalizowana funkcja przyjmuje postać

$$f(q) = -E(X, q). \tag{8.25}$$

Po wykonaniu jednego kroku algorytmu Newtona, weryfikowane jest kryterium zbieżności. Jeżeli zostało spełnione, to obecna wartość q definiuje poszukiwaną transformację, a działanie metody zostaje zakończone. W przeciwnym przypadku, następuje powrót do poszukiwania odpowiadających wokseli dla transformowanych punktów z wejściowej mapy. Operacje są powtarzane do momentu spełnienia kryterium zbieżności.

Jedną z zalet wykorzystania metody dopasowania opartej na NDT względem ICP jest mniejsza złożoność obliczeniowa, ponieważ dopasowanie można zrealizować w czasie $\mathcal{O}(n)$, co zostało przedstawione w pracy [155].

8.5.10 Ocena estymowanej transformacji

W warunkach laboratoryjnych, znając położenia startowe robotów można niewielkim kosztem ocenić jakość dopasowania map. Niestety, w rzeczywistych warunkach nie jest to kwestia trywialna. Wiele czynników takich jak zakłócenia czy niejednoznaczność opisu cech, sprawia, że błędne dopasowanie może zostać potraktowane jako poprawne.

W celu poradzenia sobie z tym problemem, wprowadzono kryteria oceny dopasowania wyznaczane dla par odpowiadających sobie punktów (p_i, q_i), znalezionych metodą najbliższych sąsiadów. Punkty tworzące pary należą odpowiednio do zbiorów

$$P = \{p_i \mid p_i \in R^3, i = 1, \ldots, n\}, \tag{8.26}$$

oraz

$$Q = \{q_i \mid q_i \in R^3, i = 1, \ldots, n\}. \tag{8.27}$$

Kryterium I

Podstawowym kryterium oceny jakości dopasowania są współczynniki dopasowania wyrażające średnią odległość między n parami punktów (p_i, q_i). Standardową wersję współczynnika dopasowania f_s definiuje się jako średnią arytmetyczną wszystkich odległości [188]

$$f_s = \frac{1}{n} \sum_{i=1}^{n} \|p_i - q_i\| . \tag{8.28}$$

Kryterium II

Niemniej jednak, podstawowa wersja współczynnika dopasowania nie jest wystarczająco odporna na błędne dopasowania. Przykładowo, może być obliczona dla niewielkiej liczby par punktów, przez co wynik będzie mało wiarygodny. Z tego powodu, na potrzeby niniejszej pracy zaproponowano kilka modyfikacji. Jedną z nich jest przypisanie do każdej pary punktów, binarnych wag $w_{1,i}$ zależących od odległości między tymi punktami

$$w_{1,i} = \begin{cases} 1, & \text{jeżeli } \|p_i - q_i\| \leq d_{th1} \\ 0, & \text{w przeciwnym przypadku} \end{cases} . \tag{8.29}$$

Jeżeli odległość przekracza pewną wartość progową d_{th1}, wtedy waga przyjmuje wartość zerową co oznacza, że taka para jest odrzucana, ponieważ z dużym prawdopodobieństwem została błędnie skojarzona. Dodatkowo oblicza się ilość wszystkich dobrze określonych par punktów

$$n_{w1} = \sum_{i=1}^{n} w_{1,i}. \tag{8.30}$$

Jeżeli liczba dobrze określonych par punktów n_{w1} nie przekracza wartości progowej n_{th}, obliczony współczynnik dopasowania f_{s2} oznaczany jest jako niewiarygodny

$$f_{s1} = \begin{cases} \dfrac{1}{n_{w1}} \displaystyle\sum_{i=1}^{n} \|p_i - q_i\| \, w_{1,i}, & \text{jeżeli } n_{w1} \geq n_{th} \\ +\infty, & \text{w przeciwnym przypadku} \end{cases} . \tag{8.31}$$

Kryterium III

Wprowadzono też kolejny współczynnik dopasowania bazujący na odchyleniu od wartości średniej odległości. Wykorzystuje on również binarne współczynniki wagowe

$$w_{2,i} = \begin{cases} 1, & \text{jeżeli } |\,\|p_i - q_i\| - d_m| \leq d_{th2} \\ 0, & \text{w przeciwnym przypadku} \end{cases}, \tag{8.32}$$

gdzie średnia odległość d_m określona jest jako

$$d_m = \frac{1}{n} \sum_{i=1}^{n} \|p_i - q_i\|\,, \tag{8.33}$$

a d_{th2} określa wartość progową odchylenia od średniej. Zakładając, że

$$n_{w2} = \sum_{i=1}^{n} w_{2,i}, \tag{8.34}$$

określa liczbę par o niezerowych wagach, współczynnik dopasowania można zdefiniować jako

$$f_{s2} = \begin{cases} \dfrac{1}{n_{w2}} \displaystyle\sum_{i=1}^{n} \|p_i - q_i\|\, w_{2,i}, & \text{jeżeli } n_{w2} \geq n_{th} \\ +\infty, & \text{w przeciwnym przypadku} \end{cases}. \tag{8.35}$$

Wnioskowanie kryjące się za tak zdefiniowanym kryterium jest następujące. W przypadku idealnego dopasowania średnia odległość między punktami powinna wynosić zero. Niestety, w przypadku obserwacji otoczenia i tworzenia jego modeli pojawiają się różnego rodzaju zakłócenia, powodujące, że reprezentacja nie odzwierciedla dokładnie rzeczywistych obiektów. Zakładając, że zakłócenia mają rozkład normalny, w przypadku poprawnego dopasowania modelu do sceny, odległości między poszczególnymi parami punktów również powinny mieć rozkład normalny. Z tego względu kryterium oceny powiązano z odchyleniem standardowym, tak aby wyeliminować wpływ na metrykę większości niepoprawnych dopasowań.

8.6 Transformacja octomap

Posiadając macierz transformacji $T' = T_1^2$, określającą estymowaną transformację między dwoma mapami, można sprowadzić te mapy do wspólnego układu współrzędnych. W tym celu jedna z octomap zostanie transformowana do układu drugiej octomapy.

Zakładając, że wejściem algorytmu jest drzewo ósemkowe, a macierz transformacji jest znana, to jeden z algorytmów transformacji (pseudokod 5) można przedstawić w następujących krokach:

- obliczenie odwrotnej transformacji $(T')^{-1}$ oraz wartości granicznych docelowego drzewa ósemkowego (p_{min}, p_{max}),
- bazując na obliczonych granicach, dla każdego węzła szukany jest odpowiadający mu punkt w początkowym drzewie. W przypadku znalezienia, dodawany jest do nowego drzewa.

- wartość prawdopodobieństwa dla nowego węzła obliczana jest metodą interpolacji trójliniowej.

Algorytm 5: Algorytm transformacji probabilistycznego drzewa ósemkowego (*OctreeTransform*)

Data: M – mapa w postaci drzewa ósemkowego,
r – rozdzielczość drzewa,
T – estymowana macierz transformacji
Result: M' – mapa po transformacji

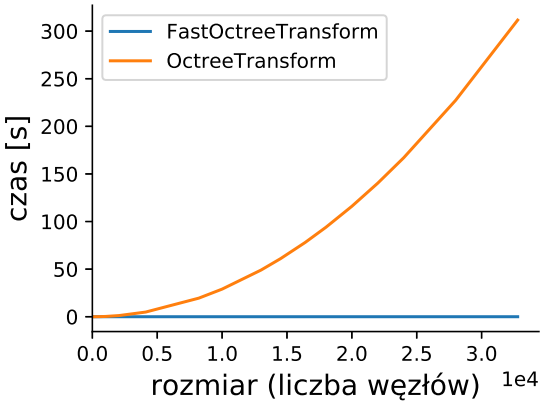
```
1  (p'_min, p'_max) ← zakres(M)
2  p_min ← T · p'_min
3  p_max ← T · p'_max
4  M' ← ∅
5  // Przejście przez wszystkie możliwe liście nowego drzewa ósemkowego
6  for x = p_min,x; x < p_max,x; x += r do
7      for y = p_min,y; y < p_max,y; y += r do
8          for z = p_min,z; z < p_max,z; z += r do
9              p ← (x, y, z)
10             p' ← T^-1 · p // Transformacja węzła do układu pierwotnego drzewa
11             if węzeł p' exists in M then
12                 n ← dodaj węzeł dla punktu (x, y, z) w drzewie M'
13                 c ← oblicz prawdopodobienstwo zajetosci na podstawie: M, r, p'
14                 ustaw pradopodobieństwo zajętości węzła n jako c
15             end
16         end
17     end
18 end
```

Przedstawiony algorytm, dzięki wykorzystaniu interpolacji trójliniowej pozwala uzyskać wysoką dokładność transformacji, bez zbędnej utraty informacji. Niestety, jego wadą jest duża złożoność obliczeniowa, co ma szczególne znaczenie przy transformowaniu większych octomap. Z tego powodu opracowano kolejny algorytm (*FastOctreeTransform*), mniej dokładny, ale działający znacznie szybciej, z powodu mniejszej złożoności obliczeniowej. Kroki są następujące:

- utworzenie nowego drzewa M',
- przejście przez wszystkie liście drzewa wejściowego M, transformowanie ich do nowego układu współrzędnych i dodanie do drzewa M'.

Pozwoliło to uzyskać złożoność obliczeniową na poziomie $\mathcal{O}(n \log n)$, gdzie n określa liczbę węzłów wejściowej mapy. Ponadto dla octomap o stałej głębokości można założyć, że złożoność ta jest liniowa. Należy jednak zauważyć, że w tym przypadku błąd związany z dyskretyzacją przestrzeni nie jest kompensowany.

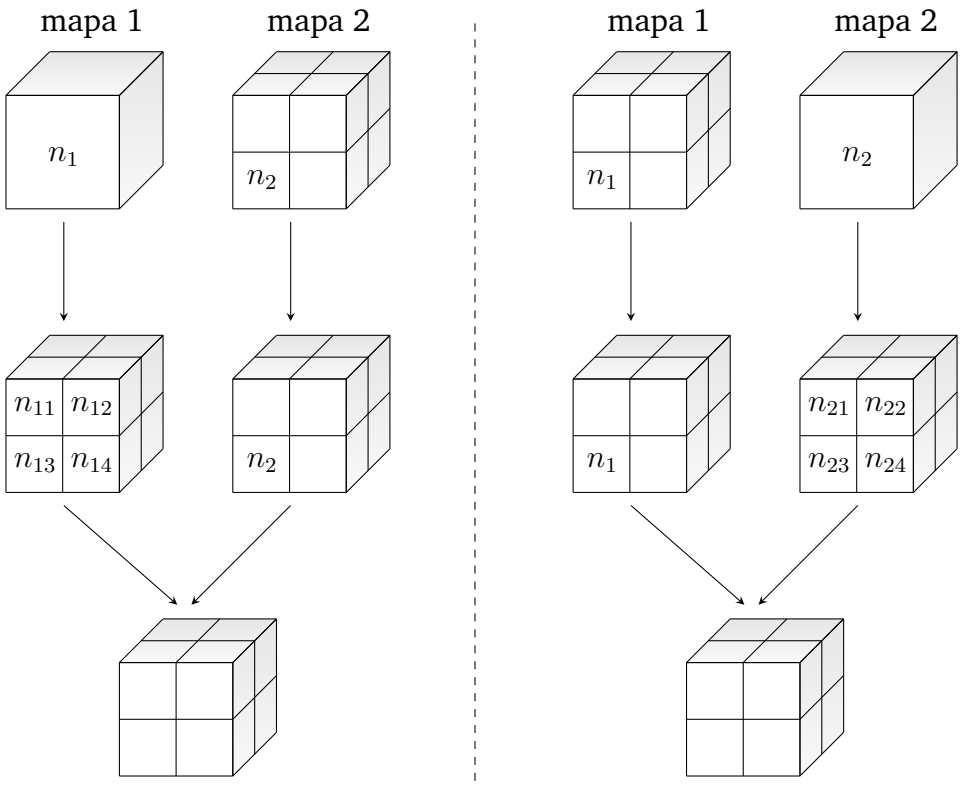
Przeprowadzono również eksperymenty porównujące dwie metody transformacji, a wyniki przedstawiono na rys. 8.15.



Rysunek 8.15 Porównanie dwóch metod transformacji octomap

8.7 Integracja danych

Zakładając, że drzewa znajdują się w tym samym układzie odniesienia, można je połączyć i obliczyć nowe wartości prawdopodobieństw poszczególnych węzłów. Należy rozważyć dwa przypadki podczas łączenia poszczególnych węzłów (rys. 8.16):



Rysunek 8.16 Integracja map z węzłami na różnych głębokościach

- liście drzew są na tej samej głębokości – wtedy wartości węzłów są sumowane.
- liście drzew znajdują się na różnej głębokości – w tym przypadku węzeł, który znajduje się na mniejszej głębokości rozwijany jest do głębokości węzła z drugiego drzewa, a powstałe wartości są sumowane.

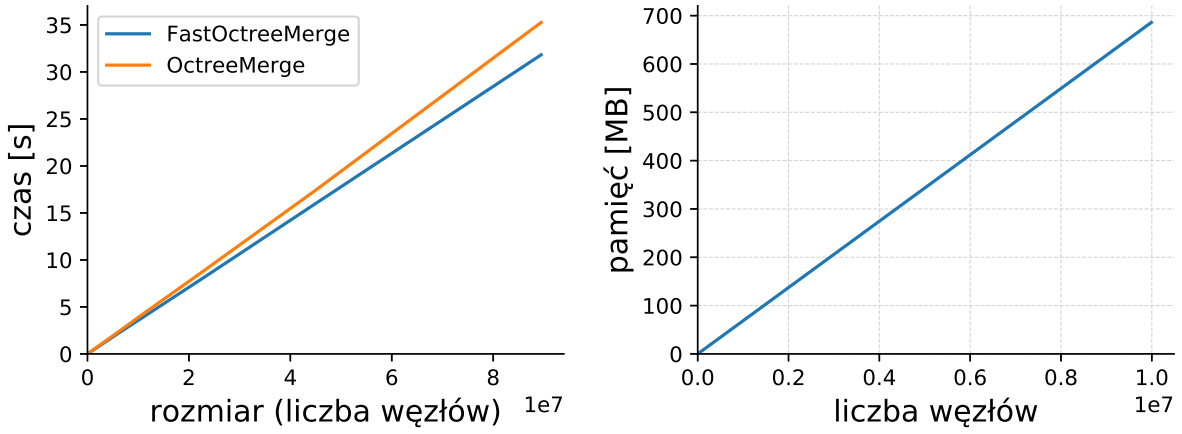
Rozwiązanie zostało przedstawione w formie algorytmu 7. Ponadto opracowano uproszczony algorytm łączenia octomap, opierający się na rozwinięciu drzew do maksymalnej głębokości (algorytm 6).

Algorytm 6: Pseudokod drugiej wersji algorytmu łączenia drzew ósemkowych (*FastOctreeMerge*)

Data: t_1 – pierwsze drzewo,
t_2 – drugie drzewo
Result: t_{out} – drzewo wynikowe

```
1  t_out ← t_1
2  expand(t_out)    // rozwiń węzły drzewa t_out do maksymalnej głębokości
3  expand(t_2)      // rozwiń węzły drzewa t_2 do maksymalnej głębokości
4  // Przejście przez liście drzewa t_2
5  for leaf ∈ leaves(t_2) do
6  |   point ← get_coordinates(leaf)        // określ współrzędne
7  |   occupancy ← get_occupancy(leaf)      // pobierz prawdopodobieństwo
8  |   t_out.add_node(leaf, occupancy)
9  end
10 return t_out
```

Przeprowadzono eksperymenty związane z rozmiarem octomap oraz czasem działania metod łączenia dwóch octomap. Rezultaty przedstawiono na rys. 8.17.



Rysunek 8.17 Porównanie dwóch metod łączenia octomap (po lewej) oraz zależność rozmiaru octomapy od liczby węzłów (po prawej)

Algorytm 7: Pseudokod algorytmu łączenia drzew ósemkowych (*OctreeMerge*)

Data: t_1 – pierwsze drzewo,
t_2 – drugie drzewo
Result: t_{out} – drzewo wynikowe

```
1  t_out ← t_1
2  for leaf_2 ∈ leaves(t_2) do
3      p ← get_coordinates(leaf_2)
4      leaf_1 ← t_out.search(p)
5      if leaf_1 exists in t_1 then
6          d_1 ← get_depth(leaf_1)
7          d_2 ← get_depth(leaf_2)
8          o_2 ← get_occupancy(leaf_2)
9          // Węzły na tym samym poziomie
10         if d_2 = d_1 then
11             t_out.update_occupancy(leaf_1, o_2)
12         end
13         // Węzeł jest głębiej w drzewie t_2 niż w drzewie t_1
14         if d_2 > d_1 then
15             for i ← 0 to d_2 - d_1 do
16                 t_out.expand_node(leaf_1)
17                 leaf_1 = t_out.search(p)
18             end
19             t_out.update_occupancy(leaf_1, o_2)
20         end
21         // Węzeł jest głębiej w t_1 niż w t_2
22         if d_2 < d_1 then
23             for i ← 0 to d_1 - d_2 do
24                 n ← t_out.search_on_depth(p, d_2 + i)
25                 n.set_occupancy(o_2)
26                 expand_node_empty_childs(n, t_out)
27             end
28             n ← t_out.search_on_depth(p, d_1)
29             t_out.update_occupancy(leaf_1, o_2)
30         end
31     else
32         t_out.add_node(p, o_2)
33     end
34 end
35 return t_out
```

8.8 Charakterystyka badań eksperymentalnych

Badania eksperymentalne metody integracji map 3D zostały przeprowadzone z użyciem danych z publicznie dostępnych baz [97], danych otrzymanych z symulacji Gazebo, a także z użyciem robotów kołowych Turtlebot.

Mapy były generowane i łączone offline na podstawie zebranych wcześniej danych, na komputerze wyposażonym w sześciordzeniowy procesor i7-8750H i 32 GB pamięci RAM. Oprogramowanie w całości zostało napisane w języku C++, z wykorzystaniem technologii wielowątkowości. Kod źródłowy został udostępniony jako otwarte oprogramowanie (ang. *open-source*) w postaci repozytorium *3d_map_server* [1]. Dokładny opis środowisk badawczych, zarówno symulacyjnego jak i rzeczywistego znajduje się w rozdziale 10.

8.8.1 Część praktyczna eksperymentów z robotami Turtlebot

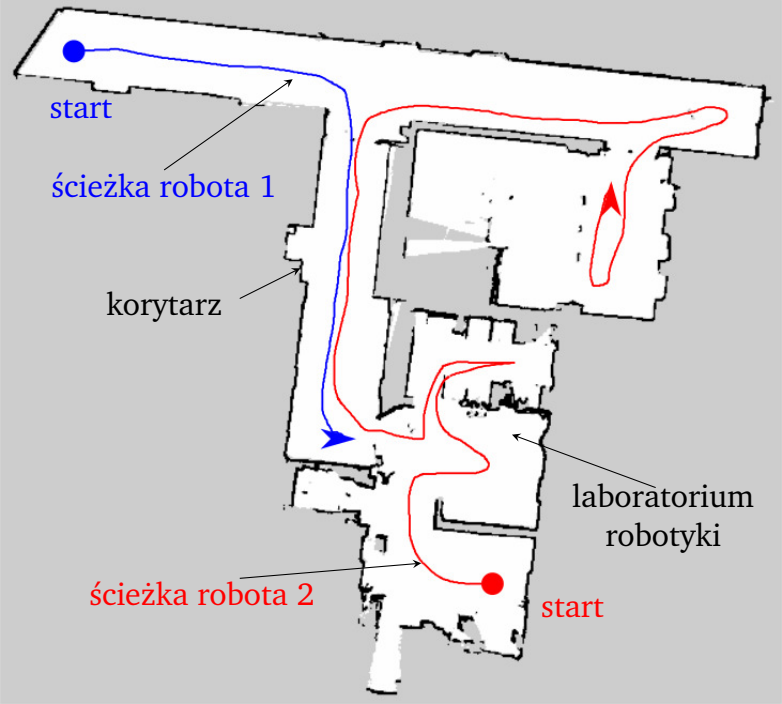
W celu potwierdzenia działania metody integracji map w rzeczywistym środowisku, został przeprowadzony eksperyment w budynku kampusu Politechniki Wrocławskiej z użyciem kołowych robotów mobilnych. Wykorzystano roboty mobilne Turtlebot (rys. 10.9), wyposażone w system odometryczny, skaner laserowy Hokuyo UST-10LX oraz sensor RGB-D Intel RealSense D435. Szczegółowy opis środowiska badawczego i systemu robotów znajduje się w rozdziale 10.

Badania można podzielić na dwa etapy. W pierwszym etapie zebrano dane pomiarowe z rzeczywistych przejazdów robotów, natomiast w drugim etapie uruchamiano oprogramowanie zawierające metodę integracji i analizowano uzyskane dane.

Zbieranie danych pomiarowych

W części praktycznej eksperymentów zrealizowano przejazdy robotów i zebrano dane pomiarowe. Roboty poruszały się w obrębie laboratorium robotyki i korytarza (rys. 8.19), w budynku D-20 kampusu Politechniki Wrocławskiej. Ścieżki po których poruszały się roboty w jednym z przejazdów, przedstawiono na rys. 8.18. Roboty rozpoczynały działanie po dwóch przeciwnych stronach kondygnacji budynku, a trasy przejazdu zaplanowano w taki sposób, aby część odwiedzanego przez roboty środowiska była wspólna (w przedstawionym przypadku jest to część komunikacyjna budynku). Dodatkowe założenia odnośnie eksperymentów:

- roboty posiadały uruchomiony system nawigacji 2D w środowisku ROS (architektura na rys. 8.20), wykorzystujący do lokalizacji metodę *AMCL*, do planowania globalnego algorytm A* oraz do planowania lokalnego metodę dynamicznego okna DWA (ang. *Dynamic Window Approach*),
- roboty poruszały się w trybie półautomatycznym, czyli użytkownik wybierał punkty docelowe, natomiast system nawigacji planował ruch, omijał przeszkody i sterował napędami robota,
- mapa statyczna dla systemu nawigacji została uprzednio wykonana metodą *gmapping* i dostarczona do wszystkich robotów,
- ze względu na wykorzystanie tej samej mapy przez różne roboty, możliwe było określenie rzeczywistych, względnych pozycji robotów.

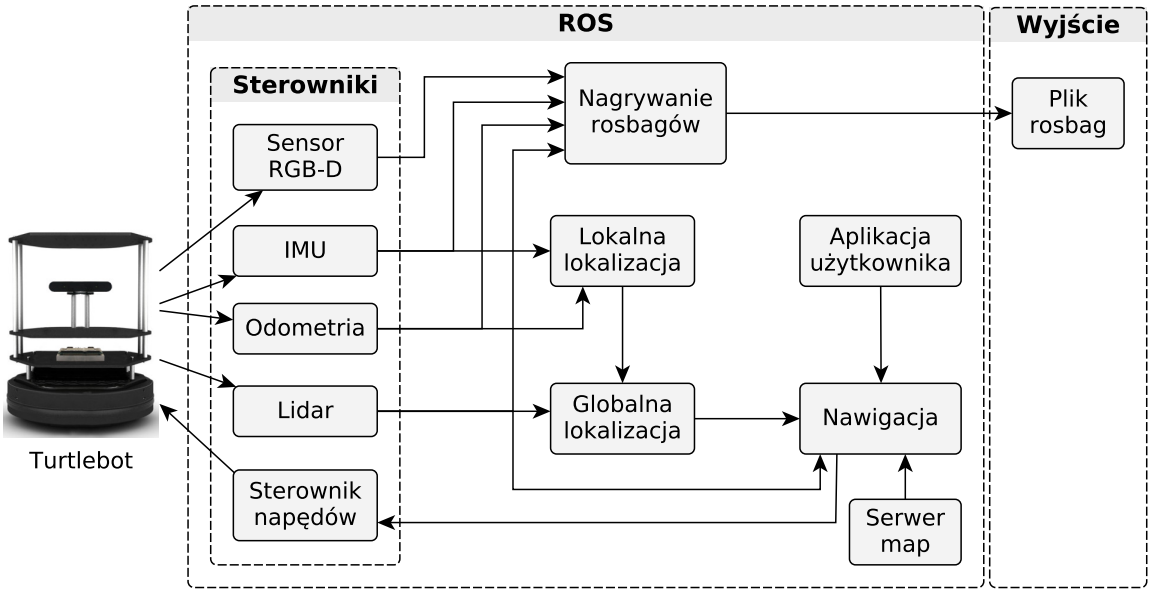


Rysunek 8.18 Ścieżki po których poruszały się roboty Turtlebot w trakcie eksperymentu

Rysunek 8.19 Laboratorium robotyki mobilnej i część budynku Politechniki Wrocławskiej, gdzie realizowano eksperymenty

Dane pomiarowe z robotów zbierane były z wykorzystaniem mechanizmu *rosbag* [201] ze środowiska ROS, umożliwiającego zapis danych do odpowiednich plików, wraz z informacją o dokładnym czasie pomiarów. Zbierano następujące informacje:

- dane z IMU (*/imu_raw*),
- dane odometryczne (*/odom*),
- odczyt ze skanera laserowego (*/laser_scan*),
- odczyt z sensora RGB-D w postaci chmury punktów (*/camera/depth/color/points*, */camera/depth/camera-info*),
- stan drzewa transformacji między układami współrzędnych związanych z robotem (*/tf* oraz */tf_static*),



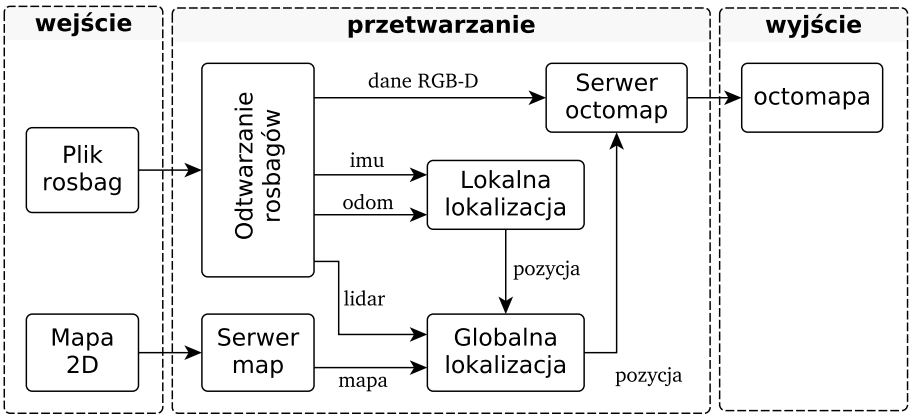
Rysunek 8.20 Architektura systemu wykorzystywanego w etapie przejazdów robotów i zbierania danych

- zadane prędkości robota (*/cmd_vel*),
- globalna ścieżka generowana przez algorytm planowania (*/global_plan*).

8.8.2 Przetwarzanie i analiza zebranych danych

Przetwarzanie danych może być podzielone na kilka kroków:

- odtworzenie systemu tworzenia octomap,
- wczytanie danych zapisanych mechanizmem *rosbag*,
- wygenerowanie octomap,
- integracja octomap,
- oraz ocena jakości rozwiązań.



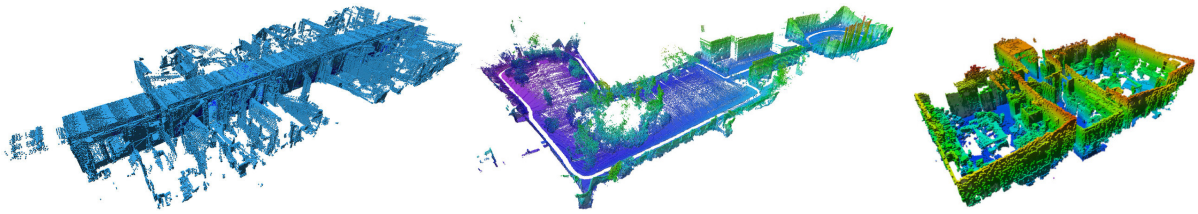
Rysunek 8.21 System używany do tworzenia octomap na podstawie zebranych danych

Konfiguracja systemu umożliwiającego zrealizowanie trzech pierwszych kroków została przedstawiona na rys. 8.21. W trakcie generowania octomap wykorzystano algorytm lokalnej lokalizacji, a także globalnej lokalizacji wykorzystujący statyczną mapę 2D. Estymowana pozycja była wykorzystywana w celu określenia transformacji między kolejnymi ramkami danych z sensora głębi, aby umożliwić zbudowanie octomapy przez dedykowany serwer.

8.8.3 Wykorzystanie ogólnodostępnych zbiorów map

Jak już wspomniano, na potrzeby badań wykorzystano octomapy pochodzące z ogólnodostępnych zbiorów udostępnionych przez Uniwersytet we Fryburgu (rys. 8.22) [97]
.

Warto zaznaczyć, że wspomniane mapy zostały utworzone z użyciem skanera laserowego SICK LMS umieszczonego na dodatkowym module umożliwiającym regulację kąta pochylenia względem płaszczyzny podłoża. Z tego też powodu dokładność map jest większa, w porównaniu do map, które zostały utworzone na podstawie danych zebranych czujnikiem RGB-D, ale nie zawierają dodatkowej informacji, chociażby w postaci koloru węzłów.



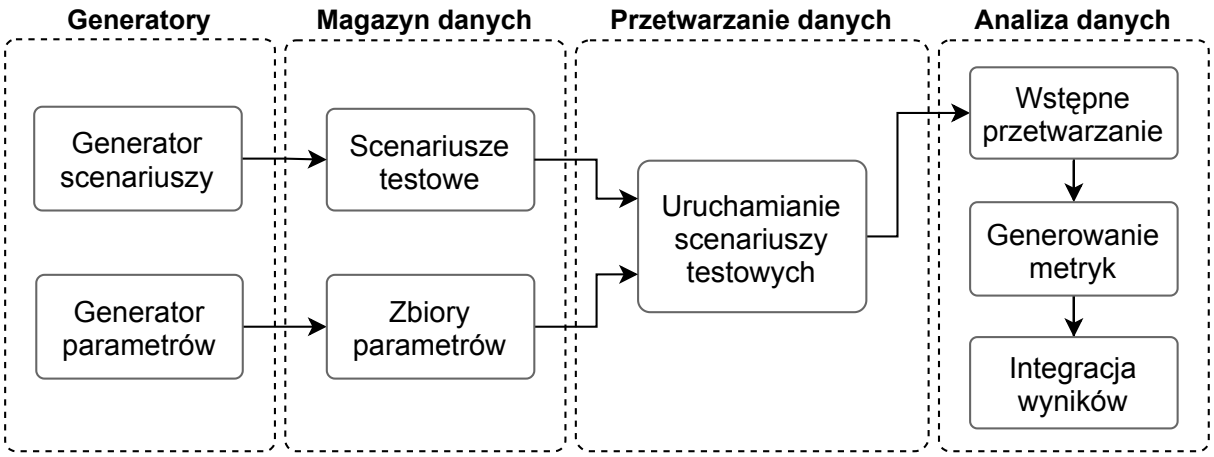
Rysunek 8.22 Wybrane mapy pochodzące ze zbioru [97]

W oparciu o mapy ze zbiorów przygotowano scenariusze testowe. W celu otrzymania dwóch map do scenariusza, przeprowadzono ekstrakcję poszczególnych map cząstkowych z całych map, w taki sposób, aby zachować część wspólną między nimi. W następnym kroku mapy cząstkowe były transformowane, a wykorzystywana transformacja zapisana jako referencyjna. Ostatni krok polegał na dodaniu szumu o rozkładzie normalnym do jednej z mapy. W rezultacie, każdy ze scenariuszy składał się z następujących elementów:

- dwóch map cząstkowych,
- transformacji między nimi,
- obliczonej części wspólnej map, jako wartości procentowej jej objętości do objętości dwóch map cząstkowych.

8.8.4 Automatyczne przetwarzanie scenariuszy testowych

W celu oceny działania metody integracji map w wielu scenariuszach testowych, zaprojektowano i zaimplementowano system (rys. 8.23) do przetwarzania i analizy wcześniej utworzonych octomap. System składa się z kilku części. Pierwsza z nich zawiera generatory: scenariuszy testowych i zbioru parametrów. Każdy scenariusz testowy składa się z dwóch octomap oraz rzeczywistej transformacji między nimi. Scenariusze generowane są na podstawie dostępnego zbioru octomap i zdefiniowanych własności.



Rysunek 8.23 System przetwarzania scenariuszy testowych

Dodatkowo generowane są zbiory parametrów dla metody integracji map, również na podstawie wcześniej zdefiniowanych zakresów danych parametrów.

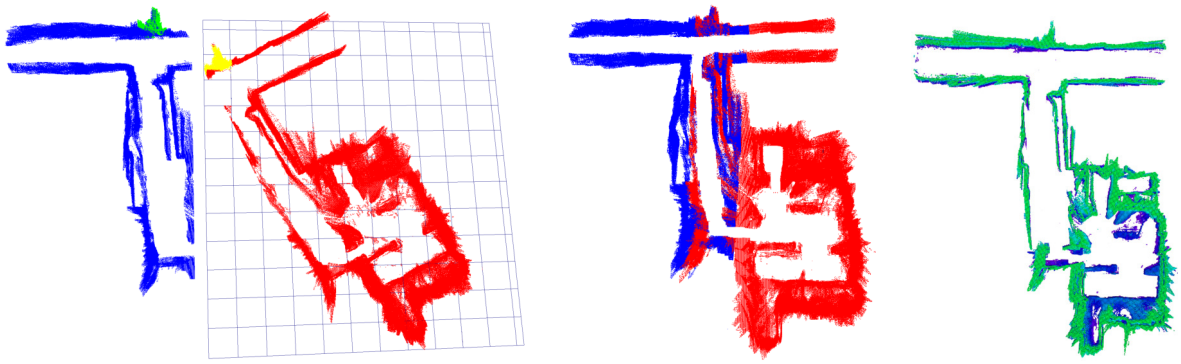
W kolejnej części, dedykowany program wybiera po jednym scenariuszu i jednym zestawie parametrów i uruchamia metodę integracji map. Rezultatem działania programu jest zintegrowana octomapa, ale też szereg parametrów jej działania, jak czasy obliczeń na poszczególnych etapach metody i własności octomap.

Ostatnia warstwa systemu testowego odpowiada za tworzenie metryk i generowanie zestawień wyników. W tej części określane są rozmiary obszaru wspólnego dwóch map oraz oceniana jest poprawność dopasowania.

8.9 Wyniki badań eksperymentalnych

8.9.1 Wyniki eksperymentów z robotami mobilnymi Turtlebot

Rezultaty wykonanych badań eksperymentalnych w budynkach kampusu Politechniki Wrocławskiej, z użyciem robotów Turtlebot zostały umieszczone na rys. 8.24. Dla



(a) Mapy wraz z wydzielonym modelem (żółty) i dopasowanym modelem (zielony)

(b) Zintegrowane mapy 3D

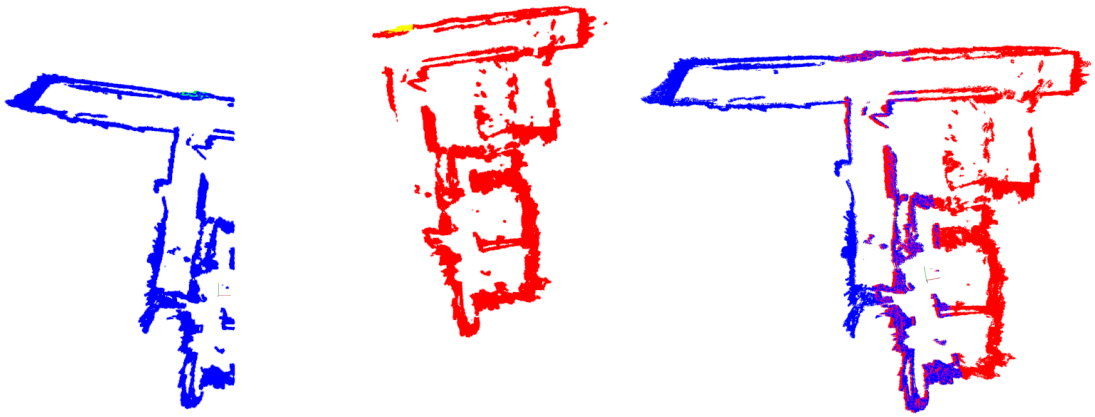
(c) Mapa wykonana przez jednego robota

Rysunek 8.24 Integracja map laboratorium i korytarza w jednym z budynków Politechniki Wrocławskiej

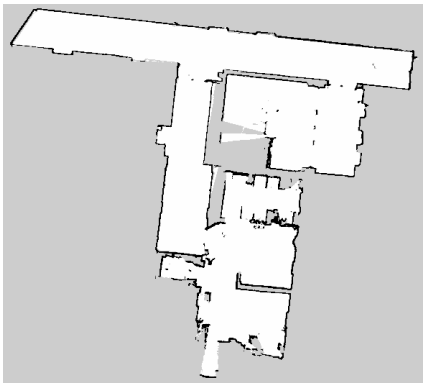
porównania przedstawiono również mapę 3D tej samej kondygnacji budynku wykonaną

przez pojedynczego robota.

Proces integracji map uzyskanych na podstawie danych zebranych podczas kolejnych przejazdów robotów w tym samym budynku, ale na większym obszarze, przedstawiono na rys. 8.25 oraz rys. 8.26.

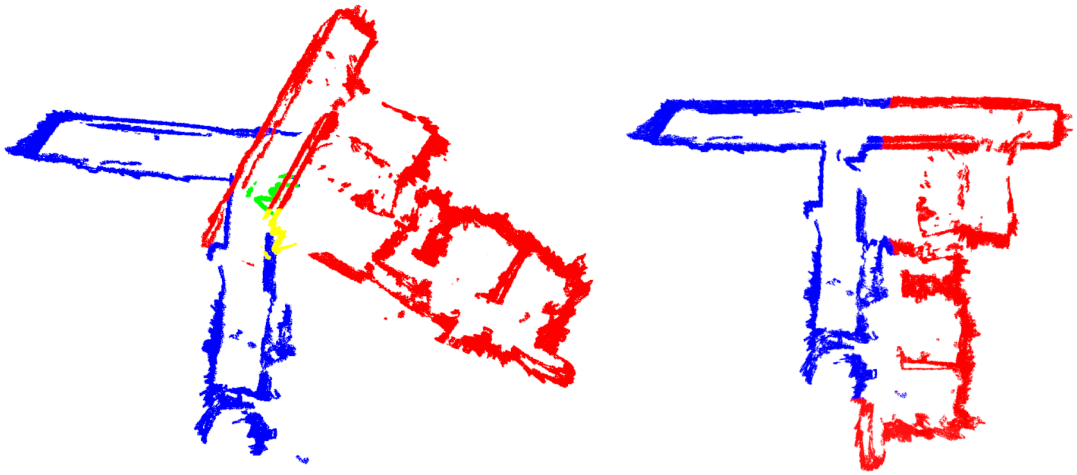


(a) Mapy przed połączeniem (b) Mapa wyjściowa



(c) Mapa 2D

Rysunek 8.25 Kolejny proces integracji map z tej samej lokacji, wraz z mapą 2D

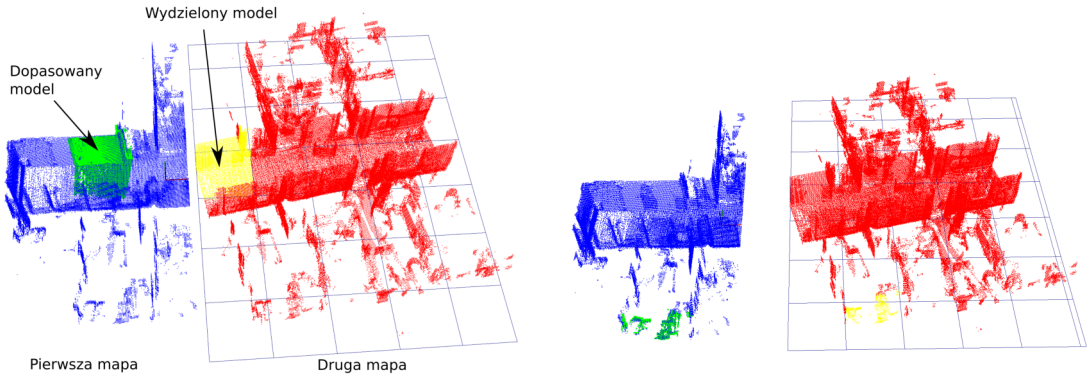


(a) Mapy przed połączeniem (b) Mapa wyjściowa

Rysunek 8.26 Integracja map z kolejnego przejazdu robotów Turtlebot

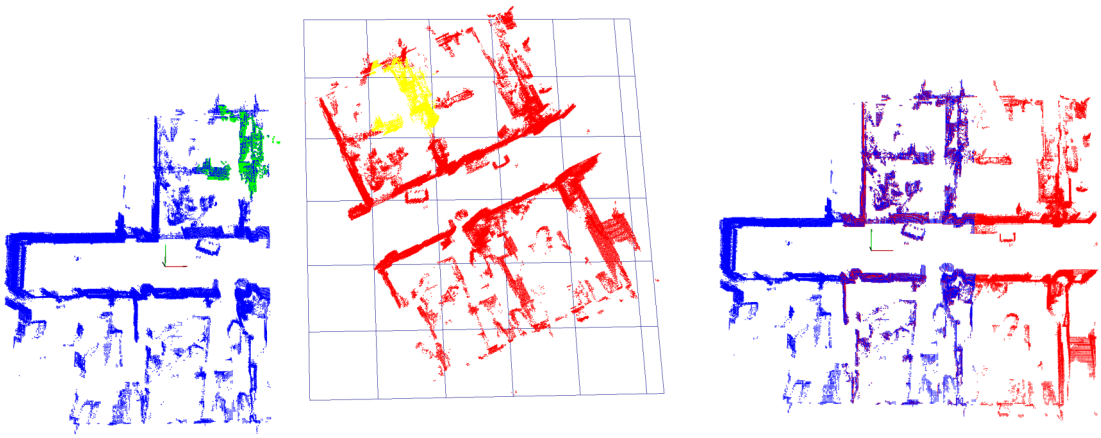
8.9.2 Wyniki eksperymentów z mapami pochodzącymi z ogólnodostępnych zbiorów

Otrzymane rezultaty łączenia map w oparciu o dane z ogólnych zbiorów, zostały umieszczone na rysunkach: 8.27, 8.28 oraz 8.29.



(a) $T_R = (7.5, 0.2, 0.1, 1°, 1°, 5°)$ (b) $T_R = (9, 1.0, 0.1, 1°, 2°, 15°)$

Rysunek 8.27 Dopasowanie modelu i estymacja transformacji między mapami. Pierwsza mapa (niebieski) pełni rolę sceny. Z drugiej mapy (czerwony) został wydzielony model (żółty), który następnie został dopasowany do pierwszej mapy. Dopasowany i transformowany model został oznaczony kolorem zielonym. Transformacja między częściami mapy (T_R) została określona w formacie: $(x, y, z, roll, pitch, yaw)$.

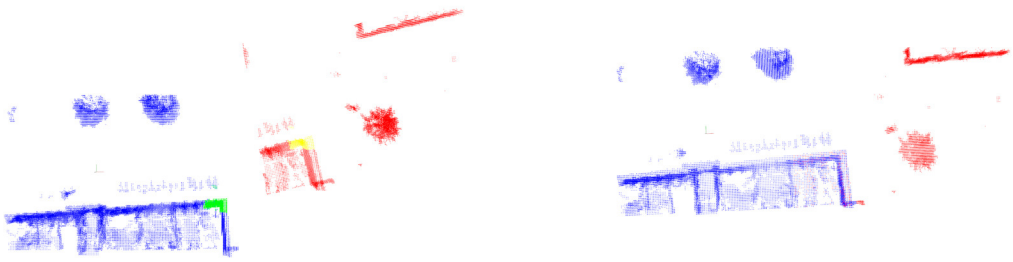


(a) Mapy przed połączeniem$T_R = (10, 1.5, 0.1, 3°, 2°, 25°)$ (b) Mapa wyjściowa

Rysunek 8.28 Kolejny przykład integracji map. Pierwsza część zawiera wyodrębnienie modelu (żółty) i dopasowanie go do sceny (niebieski). W drugiej części została umieszczona zintegrowana mapa.

Pozostałe rezultaty umieszczone zostały w tabeli 8.1, gdzie:

- n_1 oraz n_2 oznaczają rozmiary (liczbę węzłów) dwóch map wejściowych,
- T_R to rzeczywista transformacja między mapami,
- r przybliżony rozmiar części wspólnej map wyrażony jako procent pełnej mapy,



(a) Mapy przed połączeniem (b) Mapa wyjściowa

Rysunek 8.29 Integracja dwóch map zewnętrznych, utworzonych na podstawie zbioru danych New College [97]

- f_s współczynnik dopasowania najlepszego rozwiązania ze wszystkich wydzielonych modeli,
- T_{err} błąd między rzeczywistą i estymowaną transformacją obliczony jako $T_{err} = \|T_{est} \cdot T_R^{-1} - I_4\|_F$,
- t określa czas przetwarzania danych w sekundach.

Tabela. 8.1 Wybrane wyniki estymacji transformacji między mapami

n_1	n_2	T_R (x, y, z, R, P, Y)	r	f_s [m]	T_{err}	t [s]
$51k$	$67k$	$(10, 1.5, 0.1,$ $3°, 2°, 25°)$	12%	0.02	0.6	4.3
		$(10, 1.5, 0.1,$ $3°, 2°, 25°)$	24%	0.002	0.16	2.1
		$(10, 1.5, 0.1,$ $0, 0, 0°)$	36%	0.0006	0.10	1.9
		$(12, 6, 0.5,$ $5°, 5°, 60°)$	24%	0.0009	0.21	2.4

8.9.3 Wyniki badań jakości globalnego dopasowania

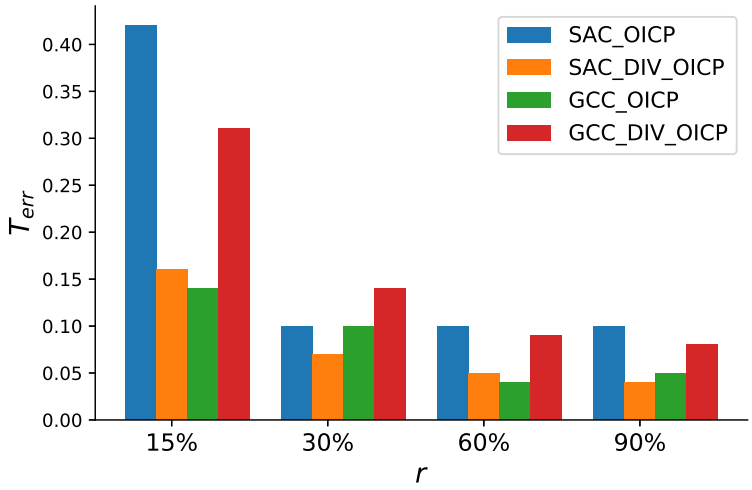
W dalszej części przedstawiono rezultaty badań porównawczych między wariantami opracowanej metody integracji map. Oznaczenia metod są następujące:

- *SAC* - probabilistyczna metoda globalnego dopasowania map,
- *SAC_DIV* - metoda globalnego dopasowania map wykorzystująca technikę ekstrakcji modelu z jednej z map,
- *GCC* - metoda globalnego dopasowania GCC [38],
- *GCC_DIV* - metoda globalnego dopasowania GCC, wykorzystująca technikę ekstrakcji modelu z jednej z map.

Wszystkie warianty wykorzystywały algorytm ISS do ekstrakcji cech oraz algorytm OICP do lokalnej korekcji rozwiązania. Przedstawione wyniki są to wyniki uśrednione dla zbioru 31 przypadków testowych.

Nakładanie się map

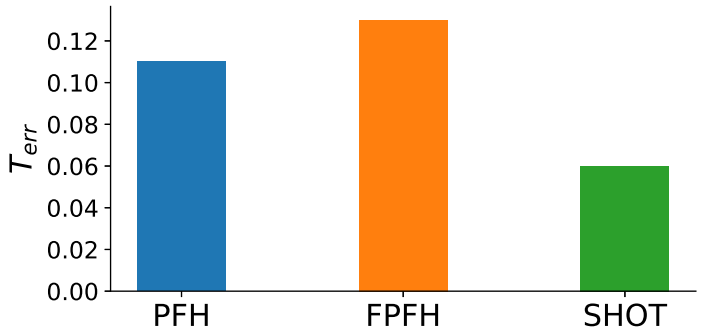
Przeprowadzono eksperymenty, których celem było określenie wpływu rozmiaru wspólnego obszaru dwóch map na wartość błędu dopasowania. Badanie powtórzono dla różnych scenariuszy testowych, o różnym rozmiarze obszarów wspólnych. Rezultaty przedstawiono na rys. 8.30. Rozmiar obszaru wspólnego obliczono jako wartość procentową $r = v_{overlapping}/(v_1 + v_2)$, gdzie $v_{overlapping}$, v_1, v_2 określają objętości prostopadłościanów wyznaczających granice odpowiednio: części wspólnej map, pierwszej i drugiej mapy. Błąd między rzeczywistą i estymowaną transformacją określa T_{err}.



Rysunek 8.30 Błąd dopasowania w zależności od stopnia nakładania się map

Wpływ rodzaju deskryptora na wartość błędu dopasowania

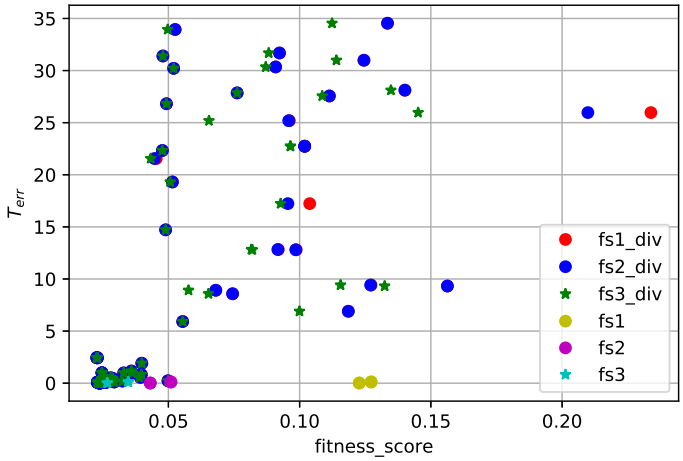
Na rys. 8.31 przedstawiono porównanie średnich wartości błędów dopasowania T_{err}, w zależności od wybranego typu deskryptora. Porównanie przeprowadzono dla metody probabilistycznej SAC z podziałem jednej z map na modele.



Rysunek 8.31 Wartość błędu dopasowania w zależności od typu deskryptora

Ewaluacja kryteriów oceny jakości dopasowania map

W sekcji 8.5.10 omówiono przyjęte kryteria oceny jakości dopasowania map. Na rys. 8.32 przedstawiono zależność wartości współczynników dopasowania od rzeczywistego błędu między poprawną transformacją i estymowaną. Współczynniki $f_{s1} - f_{s3}$ dotyczą

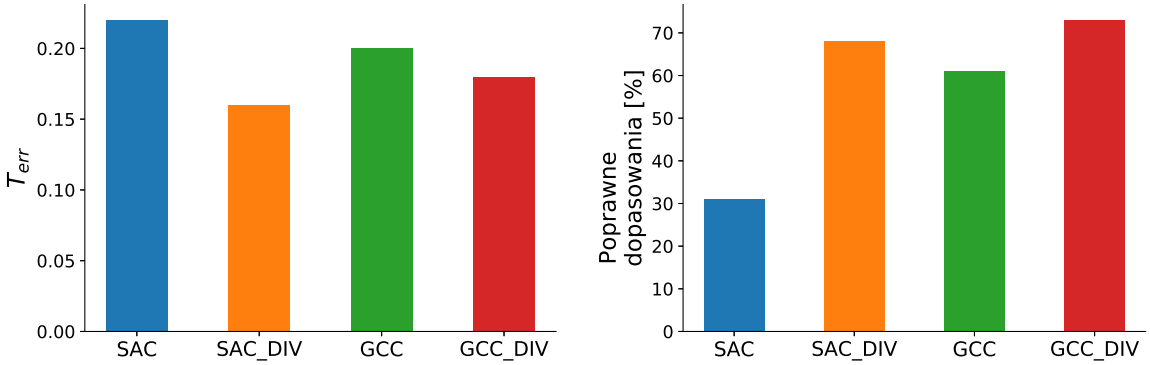


Rysunek 8.32 Porównanie różnych kryteriów oceny jakości dopasowania map

metody integracji bez ekstrakcji modelu, natomiast współczynniki dopasowania $f_{s1\_div} - f_{s3\_div}$ odnoszą się do metody integracji z ekstrakcją modelu.

Procent poprawnych dopasowań oraz średni błąd

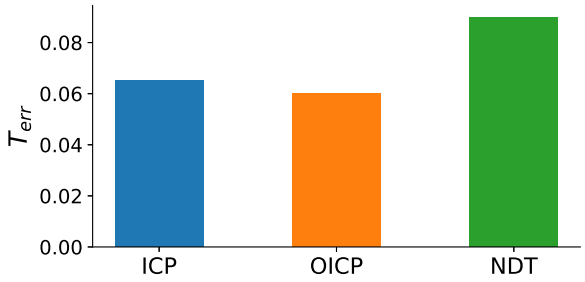
Na rys. 8.33 przedstawiono porównanie średniego błędu T_{err} oraz procentowej wartości poprawnych dopasowań map w zależności od wybranej metody. Jako poprawne dopasowania kwalifikowano takie, które nie przekroczyły ustalonej wartości błędu T_{err}.



Rysunek 8.33 Porównanie średnich wartości błędów oraz wartości procentowej poprawnych dopasowań dla globalnych metod dopasowania

8.9.4 Wyniki badań jakości lokalnego dopasowania

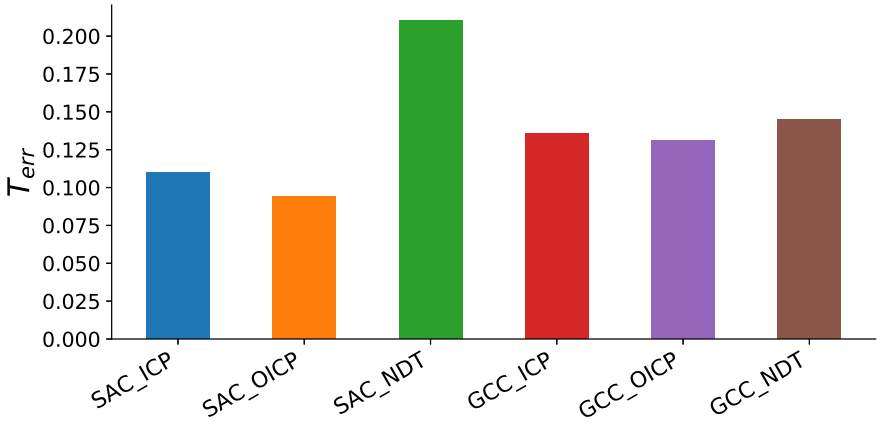
W oparciu o zestaw testowy składający się z 31 przypadków testowych, porównano działanie metod lokalnego dopasowania: ICP, OICP oraz NDT. Na rys. 8.34 przedstawiono porównanie średniego błędu estymowanej transformacji względem referencyjnej. Dla metod lokalnych ustalono wstępny błąd położenia nie przekraczający 15 cm oraz 5° w przypadku każdego z kątów.



Rysunek 8.34 Średni błąd lokalnego dopasowania w zależności od wybranej metody

8.9.5 Ogólne wyniki badań jakości estymacji transformacji

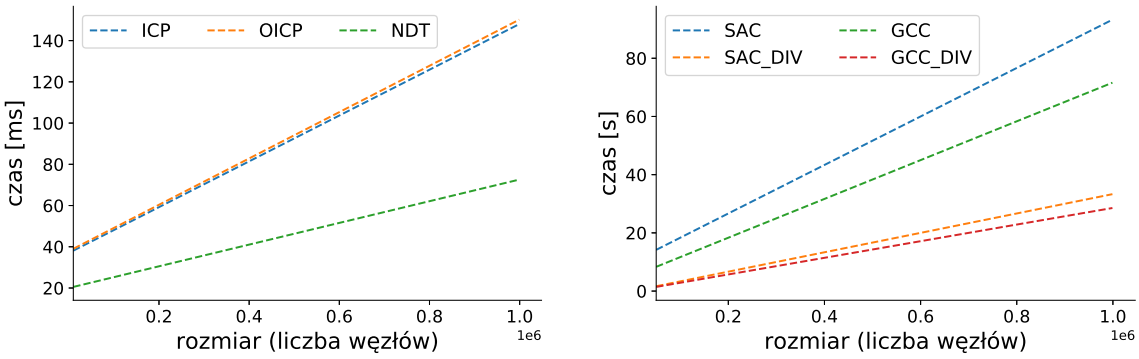
Rys. 8.35 przedstawia porównanie średniego błędu w zależności od wybranej kombinacji metod globalnych i lokalnych.



Rysunek 8.35 Porównanie kombinacji metod globalnego dopasowania z metodami lokalnego dopasowania

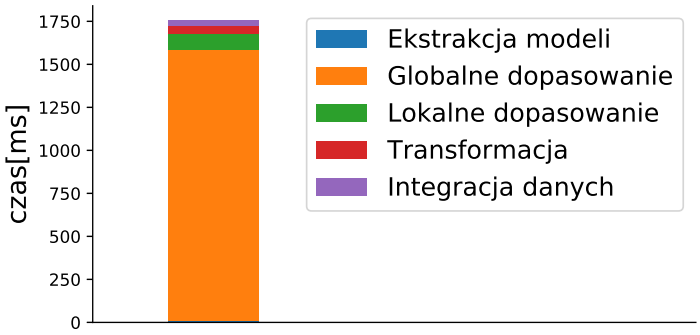
8.9.6 Wyniki badań czasu działania algorytmów

Na rys. 8.36 przedstawiono czasy działania metod lokalnego i globalnego dopasowania, w zależności od średniej liczby węzłów dwóch map. Rys. 8.37 przedstawia czasy trwania poszczególnych etapów procesu integracji map na przykładzie pojedynczego uruchomienia algorytmu. Mierzone były czasy następujących etapów: ekstrakcji modeli, estymacji transformacji między mapami (globalne i lokalne dopasowanie), transformacji jednej z octomap oraz integracji octomap, czyli połączenia drzew ósemkowych.



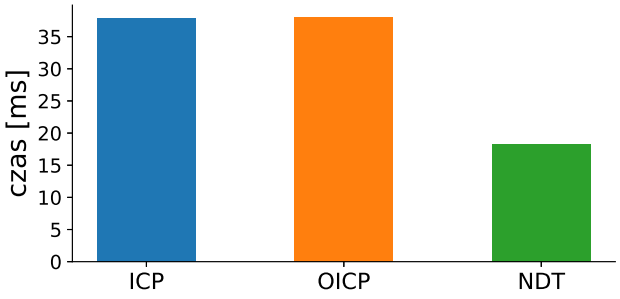
(a) metody lokalne (b) metody globalne

Rysunek 8.36 Porównanie czasu działania metod lokalnego oraz globalnego dopasowania w zależności od rozmiaru map



Rysunek 8.37 Czas trwania poszczególnych etapów procesu integracji. Przedstawiony przypadek dotyczy dwóch map o rozmiarach około $6 \cdot 10^4$ oraz $8 \cdot 10^4$ integrowanych z wykorzystaniem globalnego dopasowania metodą probabilistyczną SAC z podziałem modelu oraz metodą lokalną ICP

Na rys. 8.38 umieszczono średnie czasy działania metod lokalnego dopasowania. Czasy te dotyczą metod w przypadku, gdy model ma stały rozmiar, czyli został wydzielony z jednej z map przedstawioną wcześnie metodą ekstrakcji modeli.



Rysunek 8.38 Średni czas operacji lokalnego dopasowania jako części metody integracji, przy założeniu stałego rozmiaru modelu

8.10 Podsumowanie

W niniejszym rozdziale przedstawiono metodę integracji map 3D przy nieznanych początkowych pozycjach robotów oraz nieznanych transformacjach między mapami. Dodatkowo przy założeniu że roboty nie spotykają się w trakcie eksploracji. Do określenia transformacji wykorzystano wspólne obszary znajdujące się na mapach. Detekcja wspólnych obszarów i estymacja transformacji między nimi składała się z dwóch etapów. W pierwszym etapie realizowane jest dopasowanie początkowe, z użyciem metod ekstrakcji i dopasowywania cech. W kolejnym kroku, rozwiązanie jest poprawiane z użyciem metod lokalnej korekcji, takich jak metoda ICP, OICP oraz NDT. Dalsza integracja map składa się z transformacji jednej mapy do układu odniesienia drugiej mapy, a następnie z połączenia danych w jedną mapę.

Przeprowadzono szereg badań metody integracji map z wielu robotów. Wykorzystano ogólnodostępne zbiory danych z mapami oraz zrealizowano eksperymenty z robotami Turtlebot. W przypadku zbiorów danych, używano już gotowych map w postaci drzew ósemkowych, z których zostały przygotowane scenariusze testowe. Natomiast jeżeli chodzi o eksperymenty z robotami, w trakcie przejazdów zebrano dane, a następnie wygenerowano octomapy offline. W celu sprawdzenia większej liczby scenariuszy testowych, zaprojektowano i zaimplementowano system automatyzujący to zadanie.

Zrealizowano również badania pod kątem tego, jak bardzo mapy muszą się na siebie nakładać, aby można było skutecznie je połączyć (rys. 8.30). Minimalny rozmiar nakładającej się części map jako wartość procentowa rozmiaru map zależy od wielu czynników. Generalnie, środowiska przybierają różną formę, a możliwość dopasowania map do siebie zależy w głównej mierze od ilości cech, które uda się z tego środowiska wyznaczyć. Przeprowadzone badania pokazały, że w większości przypadków, jeżeli mapy nakładają się co najmniej w 15% to jest to wystarczające do poprawnej ich integracji.

Porównano też czasy działania metody integracji w zależności od rozmiaru map. Wyniki badań potwierdziły wstępne szacunki, określające złożoność obliczeniową algorytmu jako liniową $\mathcal{O}(n)$, przy czym n oznacza liczbę węzłów map. Osiągnięcie takiej złożoności obliczeniowej jest możliwe jedynie w przypadku wykorzystania drzewa ósemkowego o ograniczonej głębokości. W przypadku wykorzystanej implementacji octomap, głębokość ta została ograniczona do 16, więc warunek został spełniony. Wspomniana złożoność obliczeniowa metody ma związek ze złożonością operacji wyszukiwania węzłów w mapie. Kiedy głębokość drzewa jest ograniczona, można założyć, że czas dostępu do danego elementu jest stały, czyli złożoność wynosi $\mathcal{O}(1)$. W przypadku ogólnym, gdy głębokość drzewa nie jest ograniczona, czas dostępu do elementu zależy od głębokości drzewa d i wynosi $\mathcal{O}(d) \sim \mathcal{O}(\log n)$. Wtedy też ogólna złożoność metody integracji map wyniosłaby $\mathcal{O}(n \log n)$.

Badania pokazały również, że w zależności od przyjętego wariantu metody, skuteczność dopasowania wspólnych obszarów map i integracji samych map określana jest na około 70% (rys. 8.33).

Opracowana metoda może być wykorzystywana do integracji map w systemach robotów heterogenicznych, tworzonych różnymi czujnikami i metodami, zakładając, że tworzone mapy lokalne mają taką samą skalę. Jest to związane z faktem, że niepewności pomiarowe każdej octomapy lokalnej są uwzględniane są w postaci odpowiednich prawdopodobieństw zajętości, które określane są między innymi w oparciu o probabilistyczne modele sensorów. W momencie integracji map z dwóch różnych robotów, informacje o niepewnościach są propagowane do nowej mapy.

Przedstawiona metoda integracji map ma też pewne słabe strony. Jedną z nich jest

stosunkowo długi czas obliczeń, mimo tego, że szacowana zależność czasu działania od rozmiaru map jest liniowa, co potwierdzono badaniami. Integracja dwóch map o rozmiarach około $20\ m \times 20\ m \times 2\ m$ i rozdzielczości równej 5 cm, w pesymistycznym scenariuszu zajmuje ponad 5 sekund. Najbardziej złożony obliczeniowo jest krok związany z początkowym dopasowaniem map. Ma to szczególne znaczenie podczas pierwszej wymiany map między daną parą robotów, ponieważ wtedy należy znaleźć transformację między mapami, bez posiadania nawet przybliżonej informacji. Przy kolejnych integracjach map między tą samą parą robotów, można tego kosztu uniknąć i skorzystać z wcześniej estymowanych transformacji, jedynie korygując rozwiązanie lokalnie.

Zaobserwowano też, że w niektórych przypadkach trudno jest dopasować wspólne obszary map do siebie, szczególnie jeżeli mapy zawierają płaszczyznę podłoża. W takich sytuacjach potrzebny jest dodatkowy krok przetwarzania wstępnego danych polegający na usunięciu podłoża. Nie zaobserwowano natomiast problemu ze skalą map, głównie dlatego, że korzystano z czujników odległości do ich tworzenia.

Kolejnym problemem, który nie został zaadresowany w niniejszej pracy jest problem zamykania pętli. W obecnej metodzie integracji map zakłada się, że błąd pojedynczej integracji jest na tyle niewielki, że jego wpływ może być pomijany. Jednak nie jest to prawdą i po wielokrotnych łączeniach map wartość błędu może znacząco narastać i mieć znaczący wpływ na jakość map. Rozwiązaniem tego problemu może być dodanie warstwy wyższego poziomu opartej na grafie, która będzie zarządzać mapami cząstkowymi i optymalizować zależności między nimi na podstawie przyjętych ograniczeń. Innym podejściem, wykorzystywanym obecnie jest estymacja transformacji jedynie przy pierwszej wymianie map między robotami i wykorzystanie tej transformacji jako rozwiązania początkowego przy kolejnych integracjach. Jednak wiąże się to z potrzebą przechowywania dodatkowo map cząstkowych, tak, aby zawsze integrować tylko najnowsze mapy.

Dalsze prace związane z integracją map 3D mogą być prowadzone w kilku obszarach. Jednym z nich jest optymalizacja algorytmów, szczególnie, że wiele z nich można stosunkowo łatwo zaimplementować w postaci równoległej, z wykorzystaniem przetwarzania na GPU. Kolejnym kierunkiem badań będzie wykorzystanie metod maszynowego uczenia do wykrywania obszarów wspólnych na mapach, co mogłoby poprawić czas działania metody i jednocześnie zwiększyć skuteczność dopasowania.

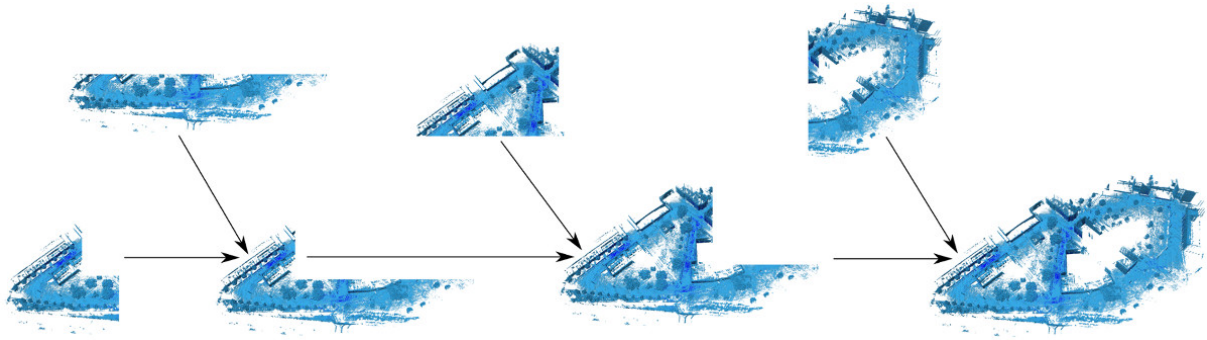
Rozdział 9

Serwer map dla systemów wielorobotowych

Niniejszy rozdział przedstawia opracowany serwer map 3D przeznaczony dla systemów robotów mobilnych, wykorzystujący podejście grafowe i umożliwiający określanie transformacji między mapami cząstkowymi na podstawie wykrywania i dopasowywania wspólnych obszarów. Na rys. 9.1 przedstawiono przykład działania serwera. Ponadto w rozdziale umieszczono wyniki przeprowadzonych eksperymentów dotyczących działania opracowanego serwera map.

9.1 Wprowadzenie

W przypadku rozwiązań wielorobotowych, ważne jest zapewnienie odpowiedniej wymiany informacji między robotami, w szczególności dotyczy do budowy i przetwarzania map. Pozwala to uniknąć wielokrotnego zdobywania tej samej wiedzy przez każdego robota oddzielnie. W literaturze można znaleźć szereg rozwiązań dotyczących tego zagadnienia. Dostępne rozwiązania można podzielić między innymi ze względu na miejsce składowania danych i przeprowadzania operacji obliczeniowych. Wyróżnia się dwa podejścia. Pierwsze jest oparte o przesyłanie danych do chmury obliczeniowej i wykonywanie obliczeń po jej stronie. W drugim podejściu, zakłada się, że roboty indywidualnie przetwarzają oraz przechowują mapy. Zarówno pierwsze jak i drugie rozwiązanie wiąże się z pewnymi wadami. W przypadku wykorzystania chmur, roboty wymagają stabilnego połączenia z serwerami. Dodatkowo, jeżeli zbyt duża odpowiedzialność za operacje zostanie przeniesiona do serwerów, może się okazać że robot nie jest w stanie



Rysunek 9.1 Budowa mapy na podstawie dodawanych kolejno map cząstkowych

funkcjonować bez wspomnianego połączenia. W przypadku indywidualnego przetwarzania, wadą jest z pewnością większe wymaganie na moc obliczeniową poszczególnych robotów. Jednak dzięki temu zyskuje się ich niezależność, nawet w przypadku braku połączenia z innymi robotami lub serwerami. Pewnym sposobem kompensacji przedstawionych wad może być podejście hybrydowe, polegające na wykorzystaniu w większej mierze przetwarzania lokalnego, z wsparciem serwerowym. Przykładowo, każdy z robotów może samodzielnie przechowywać i przetwarzać mapy, a w momencie, gdy istnieje połączenie z serwerem, przesyłać do niego mapy w celu kosztownej obliczeniowo optymalizacji.

Wracając do metod opartych na chmurze, do tej kategorii można zaliczyć rozwiązanie przedstawione w pracy [190]. Metoda wykorzystuje chmurę, która pośredniczy w wymianie danych między robotami. Wymiana danych dotyczy nie tylko map środowiska, ale także abstrakcyjnych opisów zadań, czy modeli obiektów. Autorzy skupili się na systemie scentralizowanym, gdzie operacje w postaci skomplikowanych obliczeń, oraz składowanie informacji realizowane są po stronie serwera.

Kolejne rozwiązanie oparte na technologiach chmur obliczeniowych przedstawiono w pracy [197], w postaci środowiska programistycznego C^2TAM. Środowisko to przeznaczone jest do kooperacyjnego lokalizowania robotów i mapowania otoczenia. Rozwiązanie umożliwia wykorzystanie wizyjnych metod SLAM, przy czym zadania wymagające dużej mocy obliczeniowej lub znacznej przestrzeni dyskowej realizowane są w chmurze. Mowa przede wszystkim o optymalizacji map, ale też ich składowaniu. Natomiast mniej skomplikowane obliczeniowo operacje realizowane są po stronie robotów, na ich lokalnych jednostkach obliczeniowych. Dodatkowo, przedstawione rozwiązanie pozwala na wykrywanie części wspólnych map i ich łączenie po stronie chmury. Ponieważ mapy składowane są w chmurze, do działania robotów wymagane jest stabilne połączenie internetowe.

Jak już wspomniano, wykorzystuje się też rozwiązania nie wykorzystujące chmur obliczeniowych. Przykładowo, w pracy [41] przedstawiono sposób generowania i wymiany map 3D na potrzeby pojazdów autonomicznych. Mapy zostały utworzone z wykorzystaniem odbiorników GPS, skanerów laserowych o wysokiej precyzji oraz metody ICP. Następnie były wysyłane między pojazdami w postaci octomap i integrowane lokalnie. Integracja map była możliwa dzięki danym z systemu GPS, które umożliwiły określenie wzajemnych pozycji poszczególnych pojazdów.

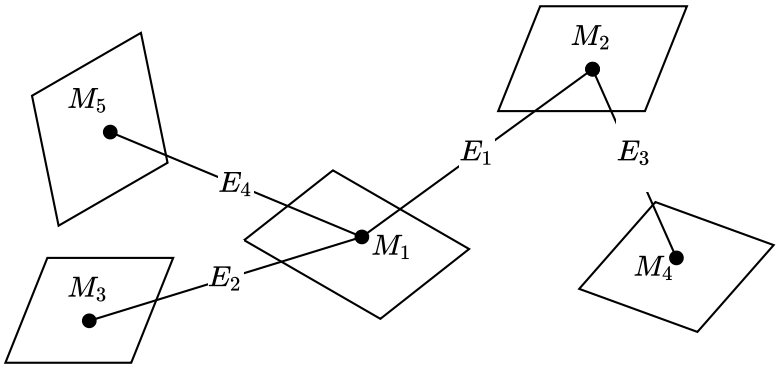
Kolejne rozwiązanie przedstawiono w pracy [262], które skupia się na podejściu topologicznym oraz na przetwarzaniu i przechowywaniu map z wielu robotów. Autorzy przedstawili rozwiązanie pozwalające wykrywać oraz optymalizować niespójności w takich mapach.

Kwestia serwera map została również podjęta w środowisku ROS [201]. W środowisku tym jest dostępny opracowany na potrzeby kołowych robotów mobilnych serwer map 2D w postaci siatek zajętości. W podstawowej wersji nie wspiera on jednak systemów wielorobotowych. Środowisko ROS zawiera także zaimplementowany serwer octomap [99], jednak do poprawnego działania wymaga on znajomości transformacji między kolejnymi ramkami danych, które są dodawane do bieżącej mapy, co zwykle jest realizowane przez wykorzystanie metody lokalnej SLAM. Rozwiązanie to nie zostało też przewidziane dla systemów wielorobotowych.

9.2 Opis problemu

Problem tworzenia map 3D w systemie wielorobotowym może być definiowany jako połączenie różnych reprezentacji tego samego środowiska w jeden model. Rozważany system składa się z N robotów $R_1, R_2, \ldots R_N$ w przestrzeni $\mathbb{R}^3$, gdzie każdy robot w oparciu o obserwacje i estymatę trajektorii, tworzy swoją lokalną mapę M_n w lokalnym układzie współrzędnych T_n. Jako rezultat procesu integracji map można rozumieć utworzenie jednego, spójnego modelu środowiska M w oparciu o k cząstkowych map (submap) $M' = \{M_1, \ldots M_k\}$ (rys. 9.1).

Przyjmując reprezentację map globalnych w postaci grafu $G = (M, E)$, każdy węzeł grafu określa mapę cząstkową M_n wykonaną przez robota R_n i posiadającą zdefiniowany punkt początkowy p_n, będący punktem zaczepienia lokalnego układu współrzędnych (rys. 9.2). Krawędź grafu $e_{ij} \in E$ określa transformację między układami bazowymi dwóch map cząstkowych.



Rysunek 9.2 Podejście grafowe w reprezentacji map z wielu robotów

Problem utworzenia globalnej mapy można zdefiniować jako problem budowy grafu G w oparciu o mapy cząstkowe M_n, zakładając, że mapa lokalna jest mapą niepodzielną i tworzy jeden węzeł grafu.

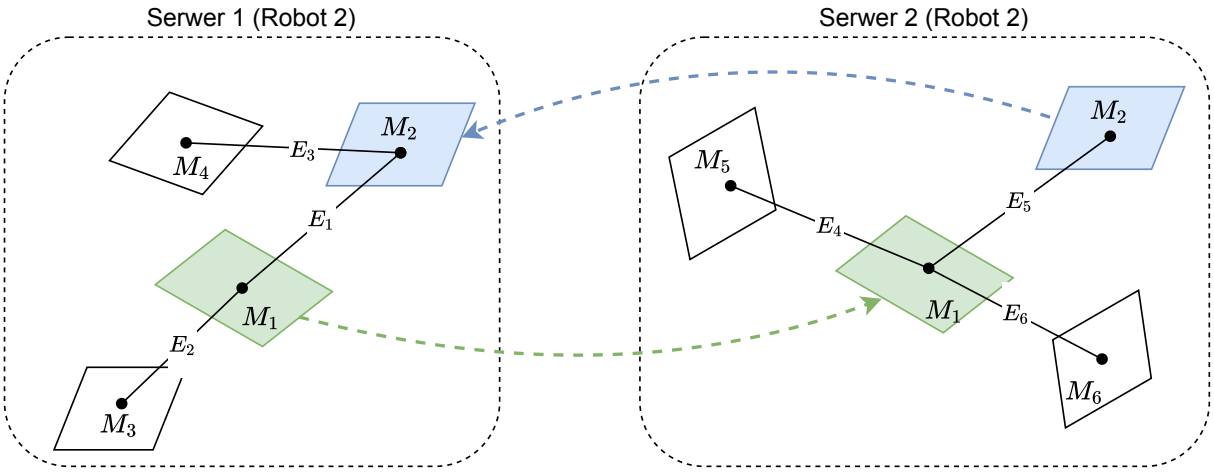
9.3 Serwer map wykorzystujący podejście grafowe

Mimo wielu rozwiązań problemu dystrybucji i przechowywania map w systemach wielorobotowych dostępnych w literaturze, nie znaleziono rozwiązania spełniającego poniższe założenia:

- efektywna obsługa map 3D,
- niezależność od zewnętrznych systemów pozycjonowania, takich jak GPS,
- możliwość określania transformacji między mapami,
- decentralizacja.

Opracowane i zaimplementowane rozwiązanie umożliwia zarządzanie mapami 3D oraz detekcję wspólnych obszarów na mapach cząstkowych i po odpowiednim dostosowaniu może być wykorzystywane w systemach rozproszonych. Przedstawiane podejście zakłada posiadanie przez każdego robota własnej instancji serwera map, jednak dzięki

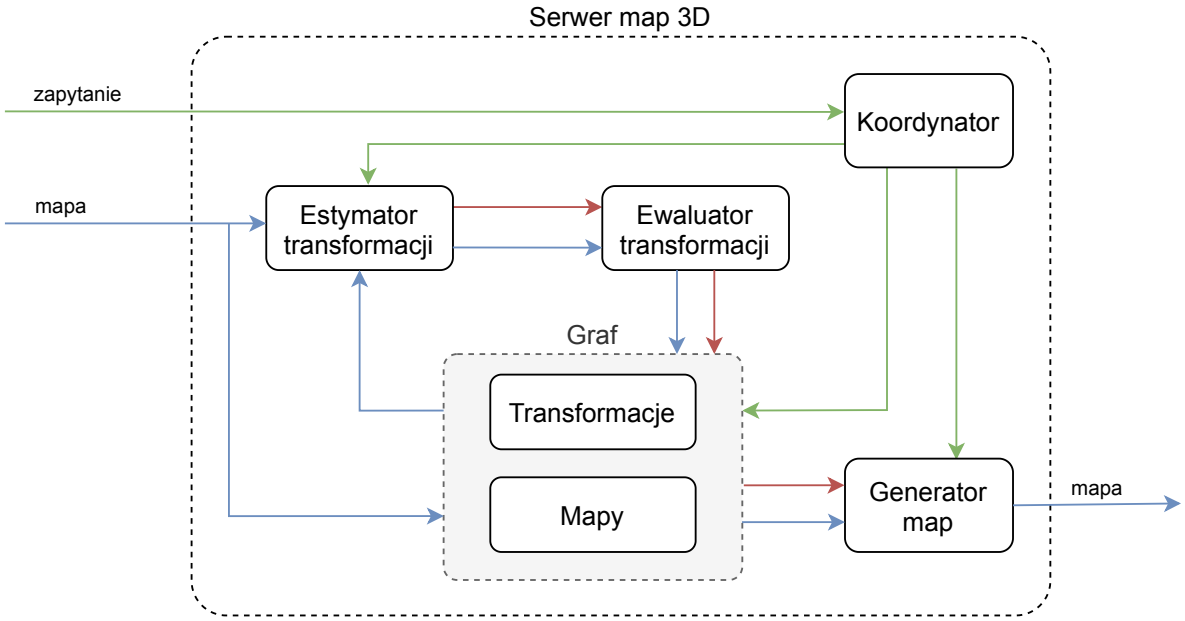
wymianie między robotami danych w postaci map cząstkowych, można zminimalizować ilość potrzebnych operacji. Na rys. 9.3 przedstawiono dwie przykładowe instancje serwera map, które współdzielą te same mapy cząstkowe.



Rysunek 9.3 Dwie instancje serwera map z wspólnymi mapami cząstkowymi

9.3.1 Architektura serwera map

Architektura opracowanego serwera map 3D została przedstawiona na rys. 9.4. Przyjmuje sie, że wejściem serwera są mapy cząstkowe oraz zapytania, natomiast wyj-



Rysunek 9.4 Architektura opracowanego serwera map 3D

ściem odpowiednio przygotowana mapa, będąca odpowiedzią na zapytanie. Serwer map otrzymuje mapy cząstkowe od poszczególnych robotów, a następnie dodaje je do grafu. Wspomniany już graf jest głównym elementem serwera. Jego węzły stanowią mapy cząstkowe z poszczególnych robotów, a krawędzie reprezentują transformacje między tymi mapami.

Kolejnym komponentem systemu jest koordynator, który odbiera zapytania od klientów serwera oraz zarządza pozostałymi komponentami, wysyłając odpowiednie polecenia. Rolą koordynatora jest rozpoczęcie procesu dodawania nowej mapy do grafu, ale także uruchomienie procesu generowania mapy wyjściowej, w odpowiedzi na zapytanie o mapę obszaru.

Estymator transformacji wykorzystywany jest w celu określania transformacji między mapą wejściową, a obecnie przechowywanymi mapami. Wykorzystuje on w tym celu metodę opisaną w rozdziale 8, opierającą działanie na opisie cech lokalnych i ich dopasowywaniu. W trakcie poszukiwania dopasowań, estymator przeszukuje zbiór map dostępnych w grafie i sekwencyjnie próbuje dopasować nową mapę do każdej z istniejących. Jeżeli się to udaje, wtedy transformacja jest przesyłana do ewaluatora, który dokonuje oceny i w przypadku pozytywnej oceny, nowa mapa jest dodawana do grafu wraz z odpowiednimi transformacjami w postaci krawędzi. Jeżeli mapę wejściową uda się dopasować do większej liczby przechowywanych map, wtedy dodawanych jest więcej krawędzi. Z tego powodu, mogą powstawać cykle w grafie.

Przyjmuje się, że robot operuje zazwyczaj na pewnej części mapy, dlatego klienci serwera, jak system planujący lub nawigujący mogą wystosować zapytanie do serwera map, aby pobrać mapę danego obszaru. Komponentem odpowiedzialnym za generowanie map obszarów jest generator map. Mapy są generowane w oparciu o zestaw parametrów podanych w zapytaniu. Proces generowania mapy wyjściowej na podstawie zbioru map cząstkowych został opisany w dalszej części.

Ponadto działanie serwera map wykorzystuje kilka założeń. Przede wszystkim, każdy robot musi posiadać unikalny identyfikator, a także mieć możliwość generowania globalnie unikalnych identyfikatorów dla kolejnych map cząstkowych. W oparciu o te informacje koordynator jest w stanie określić, czy mapy między odpowiednimi robotami były wymieniane już wcześniej.

9.3.2 Obsługa zapytań przez serwer

Rozważa się dwa typy podstawowych operacji wykonywanych przez serwer: operacje wejściowe oraz operacje wyjściowe. Do operacji wejściowych można zaliczyć:

- dodanie nowej mapy, przy znanej pozycji w układzie współrzędnych związanym z serwerem, a dokładniej jedną z przechowywanych przez niego map,
- dodanie nowej mapy, przy nieznanych pozycjach względem map zawartych w serwerze,
- polecenie usunięcia mapy o danym identyfikatorze.

Operacje wyjściowe to:

- zapytanie o mapę obszaru,
- zapytanie o listę przechowywanych map cząstkowych,
- pobranie wybranej mapy cząstkowej.

Istnieją dwie możliwości dodania nowej mapy cząstkowej do serwera. W pierwszej, serwer (komponent estymator transformacji) estymuje transformację między mapami, czyli między obecnie przechowywanymi w grafie mapami i otrzymaną mapą cząstkową.

W drugim przypadku, transformacja jest przesyłana razem z mapą lub przesyłany jest identyfikator robota, z którego mapa została dopasowana wcześniej i wykorzystywana jest wcześniej określona transformacja.

W zależności od tego, czy wymieniano już dane z określonym robotem, serwer wykonuje jedną z dwóch, wspomnianych wcześniej operacji wejściowych:

- przy pierwszej integracji map dla danego robota - brak początkowej transformacji, więc uruchamiana jest pełna procedura,
- kolejna integracja map - jeżeli poprzednia integracja była pomyślna to wykorzystuje się poprzednią estymatę transformacji jako rozwiązanie początkowe.

9.3.3 Realizacja zapytania o dany obszar przez generator map

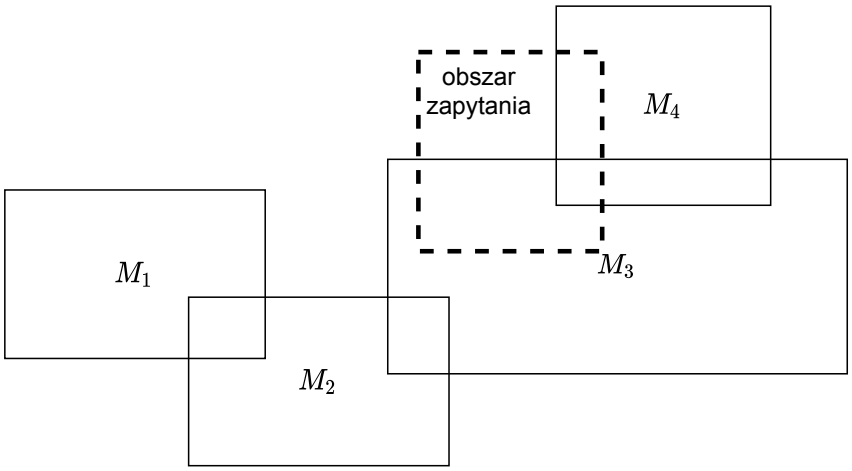
Roboty operują przeważnie w wybranym fragmencie środowiska. W związku z tym, klient serwera może wystosować zapytanie do serwera, aby pobrać mapę danego obszaru. Odpowiedź serwera nie jest jednak natychmiastowa, ponieważ nie przechowuje on połączonych map, a jedynie zbiór map cząstkowych oraz transformacji między nimi. Z tego powodu, w odpowiedzi na każde zapytanie, serwer łączy mapy dla określonego obszaru i wysyła odpowiedź. Zajmuje się tym wspomniany już generator map.

Zapytanie do serwera, którego celem jest uzyskanie mapy określonego obszaru, składa się z pary punktów $\{p_{min}, p_{max}\} \mid p_{min}, p_{max} \in \mathbb{R}^3$, które definiują prostopadłościan określający granice wycinka map. Niech granice mapy cząstkowej M_n skojarzonej z węzłem grafu zdefiniowane będą przez parę punktów $\{p^n_{min}, p^n_{max}\} \mid p^n_{min}, p^n_{max} \in \mathbb{R}^3$. W celu obliczenia wycinka mapy, realizuje się następujące kroki:

1. Porównanie granic map cząstkowych $\{p^n_{min}, p^n_{max}\}$ z parametrami zapytania i ustalenie, czy dana mapa posiada część wspólną obszarem zapytania.
2. Utworzenie zbioru map M', które posiadają część wspólną z obszarem dotyczącym zapytania.
3. Utworzenie drzewa ósemkowego reprezentującego mapę wyjściową t_{out}.
4. Dla każdej mapy ze zbioru M', pobranie wszystkich węzłów znajdujących się w części wspólnej z obszarem zapytania i po transformacji, zgodnie z wartościami krawędzi grafu, dodanie wartości węzłów do drzewa wyjściowego t_{out}.

Na rys. 9.5 przedstawiono przykład wyznaczania części wspólnej między mapami cząstkowymi i obszarem zapytania. W przykładzie tym, obszar zapytania przecina jedynie mapy M_3 oraz M_4 i jedynie te mapy zostaną uwzględnione w odpowiedzi na zapytanie.

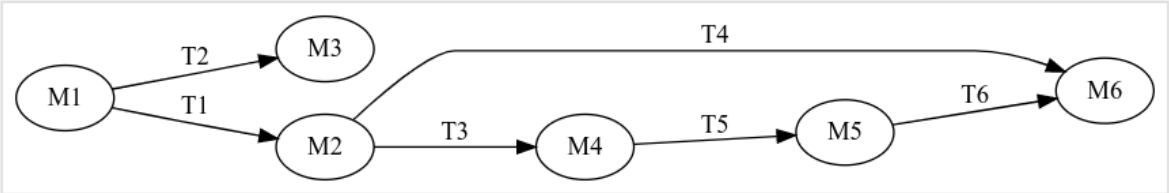
Złożoność obliczeniowa zapytania szacowana jest jako $\mathcal{O}(n \cdot k)$, gdzie n określa średnią liczbę węzłów map cząstkowych, a k liczbę map cząstkowych. Wspomniana złożoność obliczeniowa zapytania jest prawdziwa jedynie w przypadku octomapy o ograniczonej głębokości.



Rysunek 9.5 Przykład określania części wspólnych między mapami i obszarem zapytania, w rzucie 2D

9.3.4 Szczegóły implementacji

Serwer został zaimplementowany w języku `C++` i wykorzystuje opracowany algorytm integracji map, przedstawiony w poprzednim rozdziale 8. Opracowany serwer działa w oparciu o octomapy, czyli struktury oparte na drzewach ósemkowych [99], omówione w rozdziale 3. W celu wizualizacji zależności między mapami znajdującymi się w grafie (rys. 9.6), wykorzystano oprogramowanie *graphviz* [82], pozwalające automatycznie generować aktualny graf w postaci węzłów i krawędzi.



Rysunek 9.6 Przykładowy, automatycznie wygenerowany graf map i transformacji

9.4 Charakterystyka eksperymentów

9.4.1 Eksperyment z robotami Turtlebot

W celu potwierdzenia działania serwera map w rzeczywistym środowisku, przeprowadzony został eksperyment w budynku kampusu Politechniki Wrocławskiej. Na potrzeby eksperymentu wykorzystano roboty mobilne Turtlebot, wyposażone w system odometryczny, skaner laserowy Hokuyo UST-10LX oraz sensor RGB-D Intel RealSense D435. Szczegółowy opis środowiska badawczego i systemu robotów znajduje się w rozdziale 10.

Przebieg eksperymentów można podzielić na dwa etapy. W pierwszym etapie eksperymentów zebrano dane pomiarowe z przejazdów robotów, natomiast w drugim etapie uruchamiano oprogramowanie realizujące funkcję serwera map i analizowano uzyskane dane. Pozostałe elementy badań zorganizowano analogicznie jak w sekcji 8.8.

9.4.2 Wykorzystanie zbiorów map

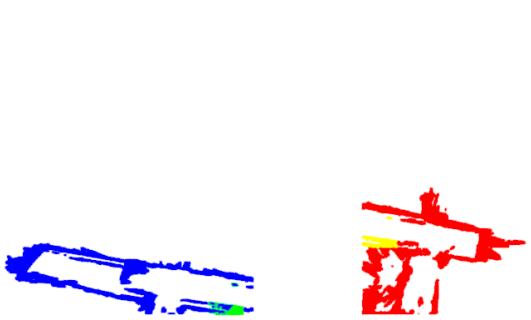
Na potrzeby badań wykorzystano octomapy pochodzące z ogólnodostępnych zbiorów udostępnionych przez Uniwersytet we Fryburgu [97]. Na podstawie pobranych octomap przygotowano scenariusze testowe wykorzystywane na potrzeby badań serwera map. Każdy scenariusz testowy składał się z następujących elementów:

- zbioru N map cząstkowych posiadających między sobą części wspólne,
- oraz zestawu transformacji rzeczywistych $T_R^{i,j}$ między mapami i oraz j.

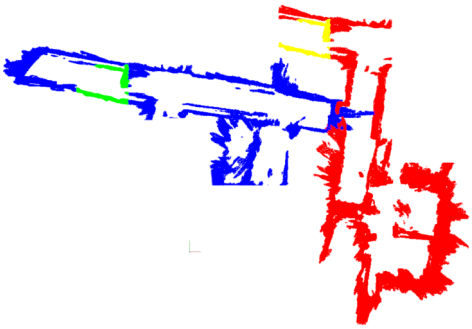
W trakcie eksperymentu, mapy były podawane do serwera w ustalonej kolejności.

9.5 Wyniki eksperymentów

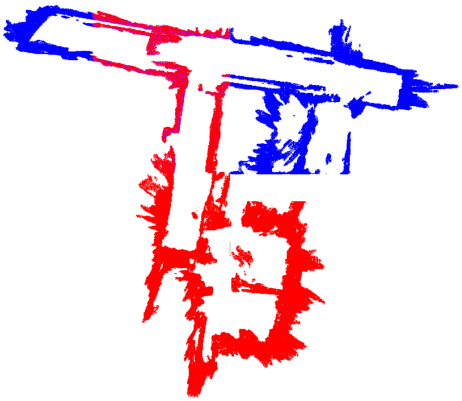
Na rys. 9.7 przedstawiono rezultat działania serwera map do którego były przesyłane kolejne mapy cząstkowe pochodzące z przejazdów robotów Turtlebot. Ostatecznie



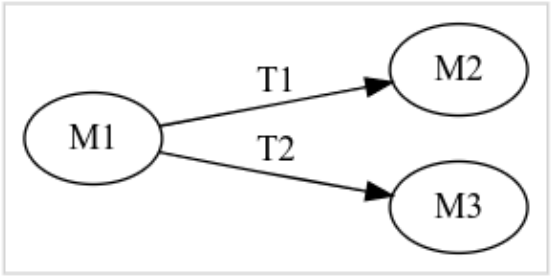
(a) Mapy przed integracją wraz z wyodrębnionym modelem (żółty) i dopasowanym modelem (zielony)



(b) Rezultat poprzedniej operacji (niebieski) i nowa mapa cząstkowa (czerwony)



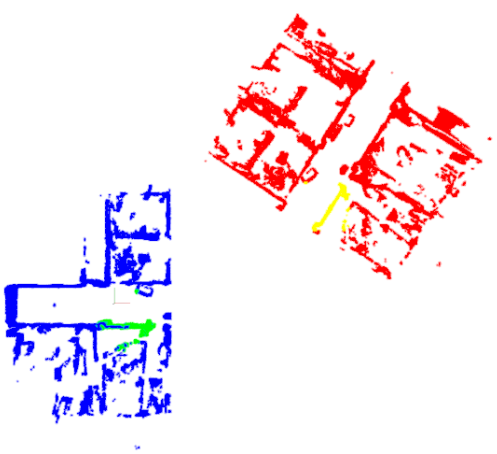
(c) Finalna mapa



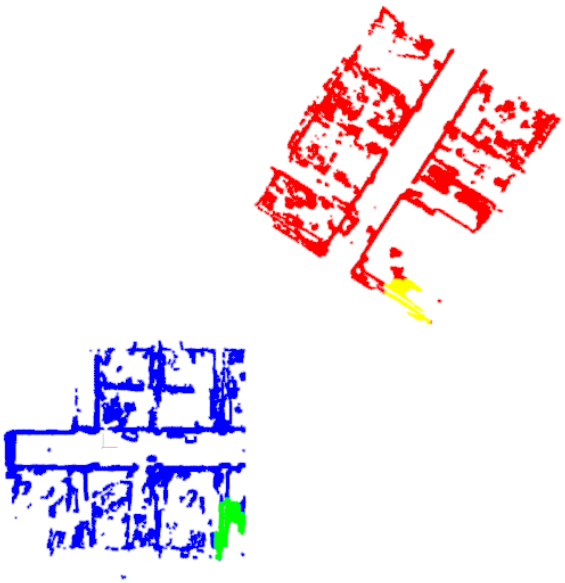
(d) Graf serwera, określający relacje między mapami

Rysunek 9.7 Wynik eksperymentu przeprowadzonego w oparciu o octomapy utworzone w jednym z budynków Politechniki Wrocławskiej, z wykorzystaniem robotów mobilnych Turtlebot. Eksperyment polegał na wielokrotnym przesyłaniu map cząstkowych do opracowanego serwera map

powstała mapa piętra jednego z budynków Politechniki Wrocławskiej. Rys. 9.8 przedstawia rezultaty uzyskane po wielokrotnym łączeniu map cząstkowych pochodzących ze zbiorów danych.



(a) Mapy przed integracją wraz z wyodrębnionym modelem (żółty) i dopasowanym modelem (zielony)



(b) Rezultat poprzedniej operacji (niebieski) oraz nowa mapa cząstkowa (czerwony)

(c) Rezultat (niebieski) i nowa część mapy (czerwony)

(d) Finalna mapa

(e) Graf określający relacje między mapami cząstkowymi, wygenerowany przez *graphviz*

Rysunek 9.8 Integracja kolejno przesyłanych octomap cząstkowych przez opracowany serwer map. Przedstawiony przykład został przygotowany na podstawie map ze zbioru przygotowanego na Uniwersytecie we Fryburgu [97]

9.6 Podsumowanie

W niniejszym rozdziale przedstawiono opracowany serwer octomap oparty na grafie. Węzły tego grafu reprezentują mapy cząstkowe, a krawędzie transformacje między nimi. Serwer pozwala na dodawanie nowych map cząstkowych na dwa sposoby. Pierwszy opiera się na estymacji transformacji w oparciu o poszukiwanie nakładających się fragmentów map i dopasowywanie ich do siebie. Drugi opiera się na podmianie map, dla których ustalono już wcześniej transformacje, co jest szczególnie przydatne w przypadku aktualizacji map z poszczególnych robotów. Serwer umożliwia generowanie map określonych obszarów w odpowiedzi na zapytania przesyłane przez klientów. Mapy obszarów tworzone są w oparciu o integrację danych z poszczególnych map cząstkowych.

Przeprowadzone zostały eksperymenty w oparciu o dane z przejazdów robotów Turtlebot, a także w oparciu o dane z ogólnodostępnych zbiorów [97]. Zaprezentowano wybrane wyniki eksperymentów, w których kolejne mapy cząstkowe były dodawane do serwera. Na podstawie wyników eksperymentów, potwierdzono działanie serwera.

Przedstawiona reprezentacja okazuje się efektywna w systemie wielorobotowym, ponieważ umożliwia wymianę jedynie map cząstkowych z poszczególnych robotów, bez wpływu na pozostałe. Kosztem takiego rozwiązania jest duża złożoność pamięciowa, wynosząca $\mathcal{O}(n \cdot k)$, gdzie n określa średnią liczbę węzłów map, a k liczbę map cząstkowych.

Serwer map może być wykorzystywany również w systemach heterogenicznych, ponieważ nie ma wymagania, aby mapy były tworzone takimi samymi sensorami. Jest to związane z wybraną formą reprezentacji, w postaci drzew ósemkowych zajętości, które przechowują informacje o prawdopodobieństwie zajętości poszczególnych wokseli. Prawdopodobieństwa zajętości wokseli uwzględniają już niepewności pomiarowe sensorów, które zostały wykorzystane podczas ich tworzenia. Ponadto prawdopodobieństwa zajętości modelują w pewnym stopniu dynamikę otoczenia.

Jednak obecne rozwiązanie posiada na tym etapie kilka wad. Jedną z nich jest brak możliwości poprawy dopasowań, ponieważ raz wyznaczona transformacja w procesie globalnego i lokalnego dopasowania tworzy stałą krawędź grafu. Rozwiązaniem tego problemu mogłoby być wykorzystanie algorytmów wyszukiwania cykli i optymalizacji grafu, które optymalizowałyby transformacje między mapami, w sposób zbliżony do tego jak to jest realizowane w metodach SLAM opartych na grafie pozycji, gdzie optymalizowane są transformacje między poszczególnymi pozycjami.

Rozdział 10

Realizacja systemu wielu robotów mobilnych

W niniejszym rozdziale zaprezentowano zaprojektowany i zaimplementowany system wielu robotów mobilnych. System powstał między innymi na potrzeby badań przedstawionych w rozdziałach związanych z integracją map (rozdział 8) oraz dystrybucją map (rozdział 9). Wykonano dwie instancje systemu wielorobotowego:

- symulacyjną, bazującą na robotycznym symulatorze Gazebo [75],
- oraz rzeczywistą, wykorzystującą roboty mobilne Turtlebot (rys. 10.1).

Rysunek 10.1 System wielu robotów mobilnych Turtlebot w laboratorium robotyki mobilnej Politechniki Wrocławskiej

Położono nacisk na to, aby architektura systemów była wspólna w możliwie dużym stopniu. Takie podejście umożliwia nie tylko stosunkowo łatwe przejście między środowiskiem symulacyjnym i laboratoryjnym, ale także między różnymi typami robotów. Opis systemów zorganizowano następująco. W pierwszej kolejności przedstawiono

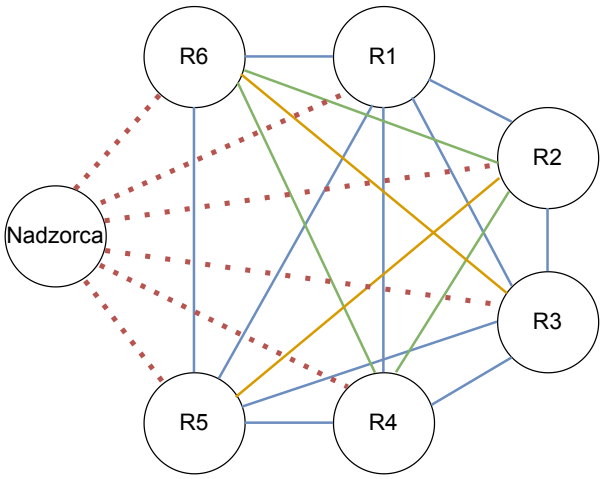
część wspólną architektury, z naciskiem na system sterowania wysokiego poziomu, pomijając część elementów niższych poziomów, oraz elementów związanych z infrastrukturą. Następnie przedstawiono dwie instancje systemu, wraz z elementami charakterystycznymi. W przypadku rzeczywistego systemu będą to kwestie sprzętowe, natomiast w przypadku systemu symulacyjnego rozwiązania do symulacji sensorów i interfejsy aktuatorów.

10.1 System wielorobotowy w środowisku ROS

Zaprojektowany system wielorobotowy został zaimplementowany z wykorzystaniem środowiska ROS [193]. Zaletą takiego podejścia jest dostępność mechanizmów do komunikacji między komponentami tworzonego oprogramowania oraz do zarządzania nimi.

Komponenty systemu sterowania wysokiego poziomu zostały zaimplementowane w postaci węzłów (ang. *node*) środowiska ROS. Węzły komunikują się ze sobą za pomocą kilku mechanizmów, w tym mechanizmu zwanego tematami (ang. *topics*) opierającego się na idei publikacji i subskrypcji (ang. *publish-subscribe*).

Jednym z ograniczeń systemu ROS dla zastosowań w systemach wielorobotowych jest potrzeba utworzenia głównego węzła (ang. *master node*), który nadzoruje całą komunikację w systemie. Niestety, wyklucza się to z ideą systemu rozproszonego, dlatego wykorzystano rozwiązanie *multimaster_fkie* [243], pozwalające na tworzenie odrębnych węzłów głównych dla poszczególnych robotów. Dodatkowy mechanizm wykrywa nowe węzły główne i synchronizuje je ze sobą. Ponadto w celu uporządkowania kanałów komunikacyjnych w systemie, do każdego robota przyporządkowano prefiksy, odpowiednio `/r1`, `/r2`, ... , `/rn`. Pozwala to na realizację bezpośredniej wymiany informacji między poszczególnymi robotami należącymi do systemu, a także między robotami i nadzorcą (rys. 10.2). W drugiej wersji systemu ROS, czyli ROS 2 mechanizmy

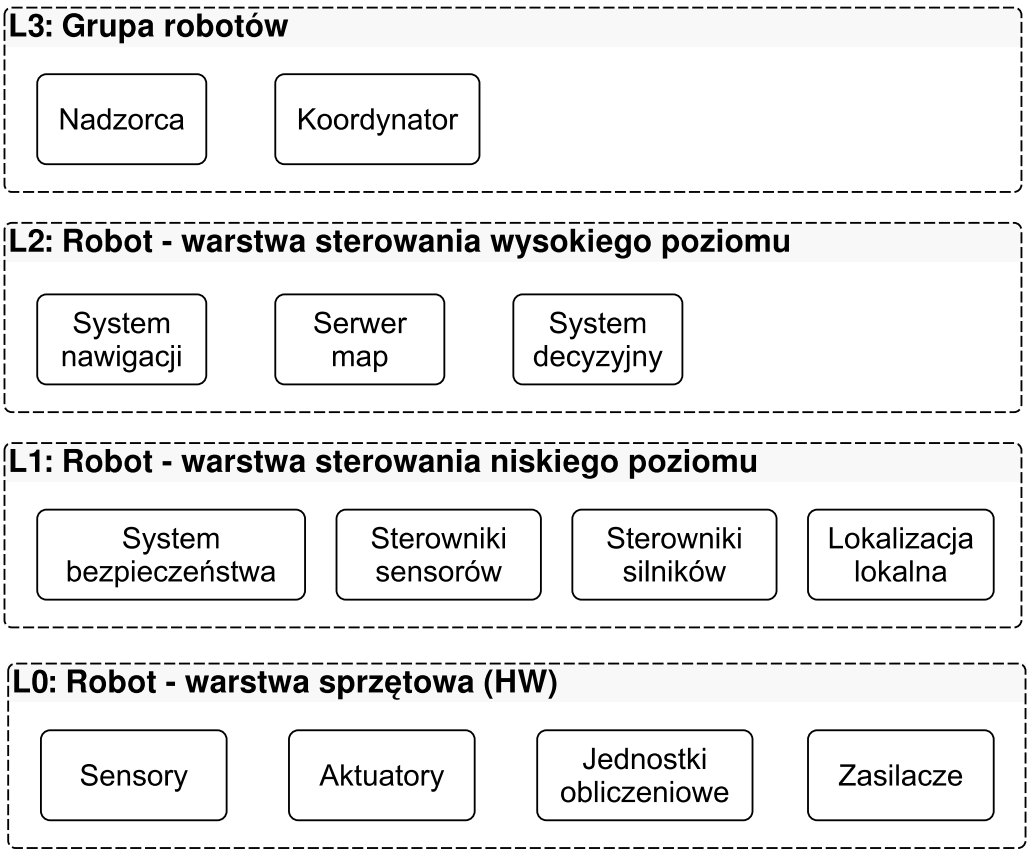


Rysunek 10.2 Bezpośrednia wymiana informacji między robotami i nadzorcą w systemie wielorobotowym

pozwalające na wykorzystanie oprogramowania w systemach wielorobotowych są już wbudowane w system.

10.2 Ogólna architektura systemu

Architektura systemu wielorobotowego została podzielona na warstwy (rys. 10.3). Takie podejście umożliwia hermetyzację logiki poszczególnych poziomów i komunikację



Rysunek 10.3 Ogólna architektura systemu wielorobotowego

między poziomami systemu jedynie za pomocą zdefiniowanych interfejsów. W aktualnej wersji, przewidziane są następujące warstwy systemu:

- warstwa poziomu 3 (L3) związana z nadzorcą i zarządcą grupy robotów, odpowiedzialnym za generowanie zadań,
- warstwa poziomu 2 (L2) obejmująca system wysokiego poziomu pojedynczego robota,
- warstwa poziomu 1 (L1) obejmująca system niskiego poziomu pojedynczego robota,
- warstwa poziomu 0 (L0) zawierająca rozwiązania sprzętowe w obrębie pojedynczego robota.

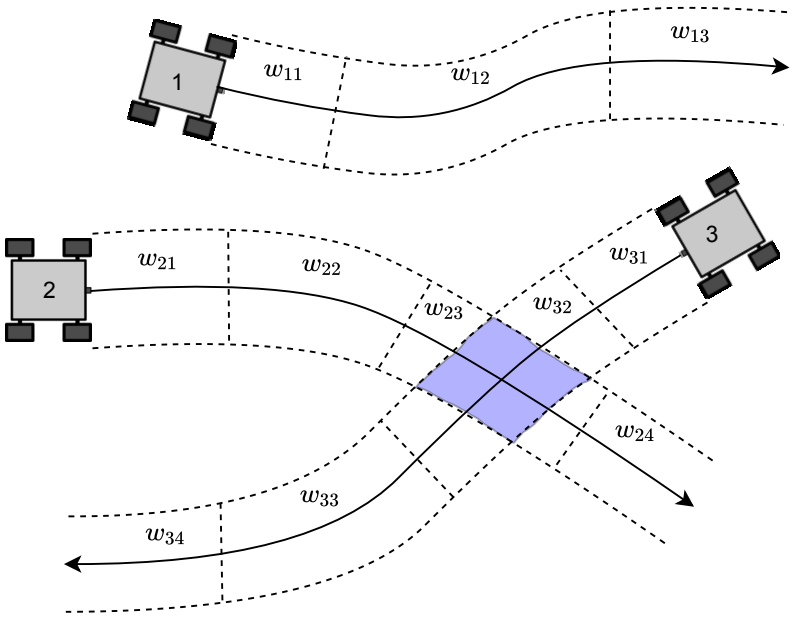
Istnieje również możliwość rozwinięcia systemu o kolejne warstwy wyższego poziomu, realizujące na przykład nadzór nad grupami robotów.

10.3 System na poziomie grupy robotów - warstwa L3

Zadania warstwy systemu operującej na poziomie grupy robotów są następujące: koordynacja działań robotów i nadzór nad grupą.

Koordynacja działań robotów

Koordynacja robotów oznacza pobieranie zadań z zewnątrz w odpowiednio sformalizowanym języku i planowanie działań robotów na poziomie grupy. Jedna z metod koordynacji zakłada rezerwowanie dla robotów obszarów środowiska, w których mogą one swobodnie działać. Może to być podział przestrzeni na komórki, ale też podział ścieżek po których poruszają się roboty na sektory [202]. Przykład podziału kolizyjnych ścieżek na sektory oraz koordynacji robotów przedstawiono na rys. 10.4. Koordynator



Rysunek 10.4 Koordynacja robotów przez podział ścieżek na sektory. W przedstawionym przypadku ścieżki robotów nr 2 i 3 są kolizyjne. Zapobieganie kolizjom polega na zezwoleniu na wjazd do odpowiednich sektorów, które mają części wspólne tylko po jednym robocie. Przykładowo, najpierw zezwolenie otrzymuje robot nr 3 na wjazd do sektora w_{32} i dopiero, gdy robot ten opuści wspomniany sektor, to zezwolenie na wjazd do w_{23} otrzymuje robot nr 2.

posiada kilka możliwości jeżeli chodzi o kontrolowanie poszczególnego robota:

- może wygenerować ścieżkę, która będzie śledzona przez robota,
- może przekazać do robota jedynie odpowiednio zdefiniowany sektor oraz punkt docelowy znajdujący się w tym sektorze,
- a także, może przekazać jedynie punkt docelowy, przy założeniu, że unikanie kolizji zostanie zrealizowane przez lokalny system robota.

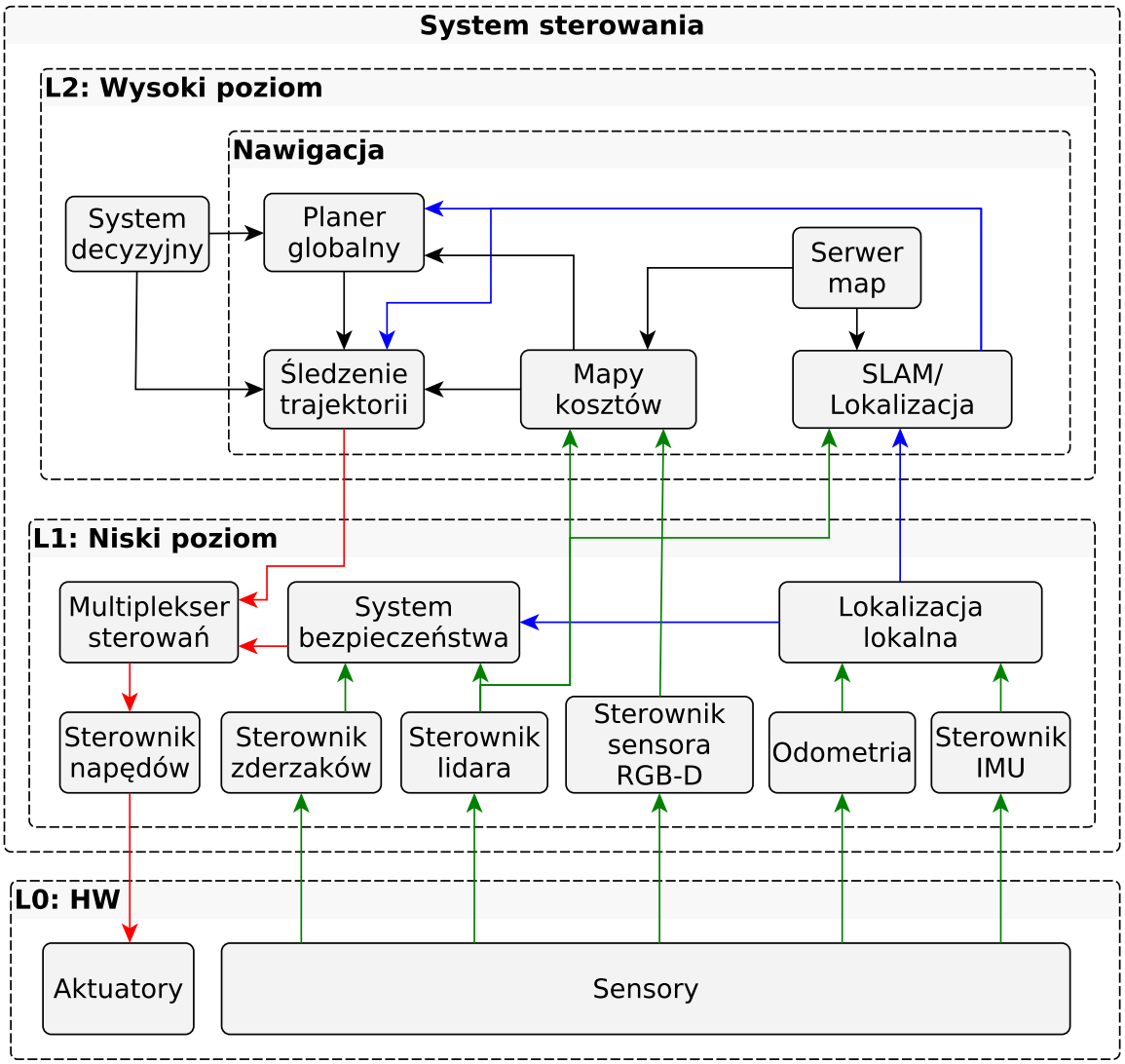
Nadzór nad grupą robotów

Przez nadzór nad grupą robotów należy rozumieć zgłaszanie defektów oraz kolizji poszczególnych robotów. Jedna z form realizacji tego zadania zakłada cykliczne odpytywanie wszystkich agentów o ich bieżący stan (ang. *polling*).

Inną możliwością jest oczekiwanie na statusy od członków grupy i mierzenie czasów od ostatniej aktualizacji dla każdego robota. W przypadku przekroczenia ustalonej, progowej wartości czasu, nadzorca może zgłosić defekt grupy.

10.4 System na poziomie pojedynczego robota - warstwy L2 i L1

System sterowania robota mobilnego ma strukturę wielopoziomową (rys. 10.5). Ar-



Rysunek 10.5 Generyczna architektura systemu sterowania robota mobilnego

chitektura systemu sterowania pojedynczego robota została podzielona na warstwę wysokiego poziomu (L2) oraz warstwę niższego poziomu (L1). Chociaż w przedstawionym przypadku warstwy te nie działają w czasie rzeczywistym, to podział systemu sterowania na warstwy umożliwia implementację części niskopoziomowej jako części systemu czasu rzeczywistego. Dodatkowo na diagramie przedstawiono warstwę sprzętową (L0).

Generyczność architektury wynika z podejścia opartego na warstwach. Celem było umożliwienie działania zarówno z rzeczywistym robotem jak i z symulatorem, a docelowo też z innymi robotami o podobnej budowie. Można to zrealizować implementując odpowiednio warstwę sprzętową (L0). W przypadku rzeczywistego robota będzie ona miała postać sprzętu takiego jak sensory, czy napędy, natomiast w przypadku symulacji warstwa sprzętowa zastąpiona zostanie symulatorami komponentów. W dalszej części przedstawiono szczegóły wspomnianych dwóch implementacji.

Do zadań systemu sterowania należy nawigacja robotem, która składa się z takich elementów jak: percepcja, lokalizacja, planowanie ścieżki, generowanie trajektorii oraz śledzenie trajektorii. W dalszej części przedstawiono charakterystykę warstw systemu sterowania, które realizują wspomniane zadania.

10.4.1 System sterowania wysokiego poziomu (L2)

W skład opracowanego systemu wchodzą następujące komponenty:

- system decyzyjny,
- globalny planer,
- śledzenie trajektorii,
- serwer map,
- mapy kosztów,
- SLAM/lokalizacja globalna.

Każdy z wymienionych komponentów jest odrębnym węzłem systemu, co powoduje, że dowolny komponent może zostać zamieniony na inny, realizujący tę samą funkcję, ale z wykorzystaniem innej klasy algorytmów.

System decyzyjny (ang. *decision system*)

System decyzyjny odpowiada za przekazywanie decyzji z systemów nadrzędnych (przykładowo zajmujących się koordynacją grup robotów) do komponentów planujących ścieżki lub śledzących trajektorie. W zależności od konfiguracji komponent ten może też samodzielnie generować cele w oparciu o zastosowany algorytm (na przykład do monitorowania terenu) lub przekazywać cele z aplikacji użytkownika

Globalny planer (ang. *global planner*)

Moduł realizuje funkcję globalnego planera ścieżki. Globalne planowanie ścieżki można rozważać jako wyznaczenie takiej ścieżki, aby robot mógł się przemieścić z obecnej pozycji do położenia docelowego, unikając przy tym kolizji z przeszkodami. Problem ten przeważnie jest upraszczany przez zastosowanie następujących założeń:

- robot to obiekt punktowy, który porusza się w przestrzeni roboczej,
- własności dynamiczne robota mogą być na tym etapie pominięte,
- środowisko jest statyczne, a jedynym poruszającym się obiektem jest robot,
- ruch ma charakter bezkontaktowy, co pozwala zignorować kwestie związane z interakcją obiektów.

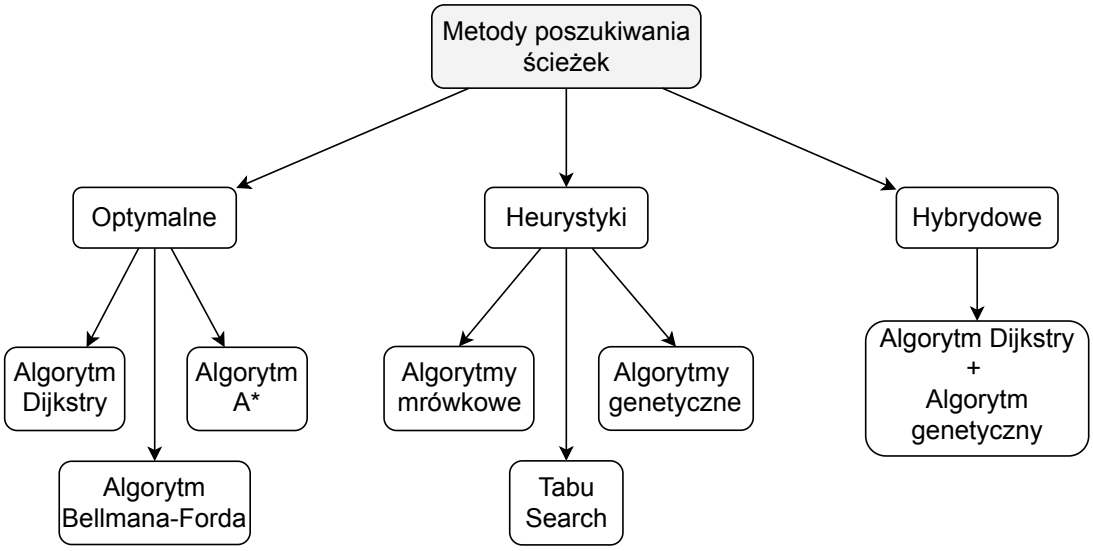
Takie założenia pozwalają rozpatrywać problem planowania ruchu jako problem planowania geometrycznej ścieżki. Pominięte na tym etapie kwestie, takie jak dynamika robota i otoczenia, uwzględniane są na etapie planowania i śledzenia trajektorii. Kolejnym założeniem odnośnie zadania planowania globalnego jest możliwość opisania ścieżki za pomocą grafu. W przypadku map zajętości, każda komórka może być węzłem grafu. Pozwala to, sprowadzić problem planowania do poszukiwania najkrótszej drogi w grafie ważonym $G = (V, E)$, dla którego V określa zbiór wierzchołków, a E to zbiór krawędzi, definiujących przejścia między węzłami. Każda krawędź grafu $e_{i,j} = (v_i, v_j) \in E$

posiada nieujemną wagę $w(v_i, v_j) \geq 0$, odpowiadającą odległości między danymi wierzchołkami.

Przyjmując, że $s \in V$ oznacza wierzchołek początkowy, a $t \in V$ wierzchołek docelowy, problem poszukiwania najkrótszej ścieżki w grafie ważonym definiuje się następująco. Znajdź ścieżkę $p = (s, v_1, v_2, v_3 \ldots v_n, t) \in V \times V \times \ldots \times V$ minimalizującą całkowity koszt przebycia drogi między początkowym i końcowym wierzchołkiem grafu, czyli minimalizującą funkcję

$$f(p) = \sum_{i=1}^{|p|-1} w(p[i], p[i+1]). \tag{10.1}$$

Wyróżnia się trzy klasy metod poszukiwania ścieżek (rys. 10.6): optymalne, heurystyczne oraz hybrydowe. Algorytmy z pierwszej grupy (algorytm A*, algorytm Dijk-



Rysunek 10.6 Klasyfikacja metod poszukiwania ścieżek [222]

stry [48] i inne) pozwalają uzyskać rozwiązania optymalne, jednak są często zbyt złożone obliczeniowo dla większych instancji problemów. Metody heurystyczne przeszukują zwykle niewielką część przestrzeni rozwiązań, przez co działają znacznie szybciej, ale zwracane rezultaty są suboptymalne. W podejściach hybrydowych łączy się rozwiązania z dwóch pierwszych grup.

W opisywanym komponencie realizującym funkcję globalnego planera wykorzystano algorytm A*, pozwalający na znalezienie optymalnej ścieżki. Algorytm A* wykorzystuje funkcję, która estymuje całkowity koszt ścieżki wykorzystującej dany wierzchołek grafu

$$f(n) = g(n) + h(n), \tag{10.2}$$

gdzie:

- $g(n)$ określa dotychczasowy koszt dojścia do wierzchołka n,
- $h(n)$ to funkcja heurystyczna estymująca koszt dotarcia z wierzchołka n do wierzchołka docelowego t.

Aby algorytm A* był kompletny, heurystyka $h(n)$ musi być dopuszczalna, czyli nie może zawyżać wartości wag na ścieżce między dwoma wierzchołkami. Dodatkowo heurystyka musi spełniać warunek spójności, czyli dla węzłów n i m

$$h(n) \leq d(n,m) + h(m), \tag{10.3}$$

gdzie $d(n,m)$ oznacza rzeczywistą odległość między węzłami n i m.

Śledzenie trajektorii (ang. *trajectory follower*)

Komponent ten spełnia kilka funkcji:

- generuje lokalną trajektorię w oparciu o globalną ścieżkę oraz lokalne mapy kosztów,
- śledzi wygenerowaną lokalną trajektorię i generuje komendy sterujące dla napędów, czyli zadane prędkości kół, uwzględniając przy tym kinematykę robota.

Algorytm ten planuje w krótszym horyzoncie czasowym niż algorytm planujący globalną ścieżkę. Z drugiej strony, algorytm ten nie jest odporny na występowanie minimów lokalnych, ponieważ planowanie opiera się jedynie w oparciu o najbliższe otoczenie robota.

Na potrzeby przedstawionego systemu sterowania wykorzystano metodę dynamicznego okna DWA (ang. *Dynamic Window Approach*) [20, 73, 127]. Algorytm ten w oparciu o ścieżkę globalną generuje takie sterowania, które w krótkim horyzoncie czasowym zapewniają bezkolizyjność i zbieżność do globalnej ścieżki. Metoda DWA opiera się na przeszukiwaniu przestrzeni sterowań i wyborze wartości według ustalonych kryteriów. Działanie opiera się na redukcji przestrzeni sterowań do sterowań bezpiecznych i optymalizacji w celu wybrania sterowań optymalnych.

Krok I Poszukiwanie przestrzeni dopuszczalnych sterowań polega na określeniu przestrzeni spełniających poszczególne warunki i znalezieniu części wspólnej:

- Zredukowana przestrzeń sterowań zawiera jedynie trajektorie kołowe opisane w postaci par (v, ω), gdzie v oznacza prędkość liniową, a ω prędkość kątową robota

$$V_1 = \{(v,\omega) \,|v\epsilon\{0, v_{max}\} \wedge \omega\epsilon\{-\omega_{max}, \omega_{max}\}\}, \tag{10.4}$$

  gdzie v_{max} i ω_{max} określają maksymalne prędkości robota.

- Kolejna przestrzeń zawiera sterowania z dopuszczalnymi prędkościami i takie, które zapewniają brak kolizji z obiektami sceny. Odrzucane są wszystkie pary (v, ω) które powodują, że w chwili zakończenia symulacji z przyjętym krokiem, robot znajduje się za blisko przeszkody (odległość poniżej progu). W rezultacie pozostają jedynie sterowania zapewniające bezpieczny ruch [73]

$$V_2 = \left\{(v,\omega) \,\middle|\, v \leq \sqrt{2\dot{v}_{max} \cdot dist(v,\omega)} \wedge \omega \leq \sqrt{2\dot{\omega}_{max} \cdot dist(v,\omega)}\right\}, \tag{10.5}$$

  gdzie $\dot{v_{max}}$ oraz $\dot{\omega_{max}}$to maksymalne przyspieszenia robota, a $dist(v,\omega)$ oznacza odległość do najbliższej przeszkody znajdującej się na wybranej trajektorii.

- Kolejna, zredukowana przestrzeń zawiera jedynie takie sterowania, które mogą zostać zrealizowane uwzględniając dynamikę robota, w szczególności wartości przyspieszeń $\dot{v}$, $\dot{\omega}$. Zbiór zawiera sterowania

$$V_3 = \{(v,\omega)\,|v\epsilon\{v_a - \dot{v}t, v_a + \dot{v}t\} \wedge \omega\epsilon\{\omega_a - \dot{\omega}t, v_a + \dot{\omega}t\}\}, \tag{10.6}$$

  gdzie v_a i ω_a to aktualne prędkości, a t definiuje horyzont czasowy.

- Przestrzeń dopuszczalnych sterowań musi spełniać przedstawione dotychczas warunki

$$V = V_1 \cap V_2 \cap V_3. \tag{10.7}$$

Krok II Kolejnym etapem jest optymalizacja funkcji celu [20]. Niech $goal(\omega)$ określa, czy robot porusza się w stronę celu, a $vel(v)$ oznacza jaką część maksymalnej prędkości liniowej robota stanowi jego aktualna prędkość $vel(v) = v/v_{max}$. Funkcję celu definiuje się jako

$$G(v,\omega) = \alpha \cdot goal(\omega) + \beta \cdot dist(v,\omega) + \gamma \cdot vel(v), \tag{10.8}$$

gdzie α, β, γ to współczynniki wagowe. Wartość sterowania maksymalizująca funkcję celu przesyłana jest do niższego poziomu sterownika napędów robota.

Serwer map (ang. *map server*)

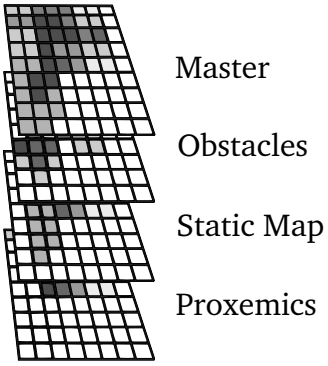
Zadaniem serwera map jest zarządzenie mapami dostępnymi w systemie. Do podstawowych funkcji należy przechowywanie map i udostępnianie ich pozostałym komponentom. W pracy wykorzystano serwer przechowujący octomapy (mapy w formie drzew ósemkowych), ale posiadający także możliwość ich konwersji do postaci siatek zajętości 2D, na potrzeby innych komponentów. Dodatkową funkcjonalnością zastosowanego serwera map jest integrowanie map częściowych z pozostałych robotów. Szczegółowy opis zastosowanego rozwiązania znajduje się w rozdziale 9.

Mapy kosztów (ang. *costmaps*)

Mapy kosztów używane są na potrzeby planowania ruchu robotów i mają przeważnie postać map zajętości. Jedna z możliwych implementacji map kosztów zakłada wykorzystanie map warstwowych 2D, opisanych w sekcji 3.2.2. Mapy kosztów można podzielić na globalne i lokalne. Te pierwsze generowane są dla całego znanego środowiska i używane są na etapie globalnego planowania. Z kolei mapy lokalne, wykorzystywane są do planowania lokalnego, ponieważ zawierają aktualne informacje o najbliższym otoczeniu robota.

Jedna z implementacji map kosztów, zakłada podejście warstwowe (rys. 10.7) [151]. Do najczęściej stosowanych warstw przechowujących informacje o najbliższym otoczeniu robota należą: mapa statyczna, warstwa przeszkód oraz warstwa przeszkód powiększonych o margines bezpieczeństwa.

Warstwa statyczna (ang. static layer) zawiera statyczne informacje o środowisku. Jeżeli używany jest algorytm SLAM warstwa statyczna opiera się na mapie z tego algorytmu. W przeciwnym przypadku, wykorzystywana jest z góry znana, statyczna mapa otoczenia. Jest to podstawowa warstwa, wykorzystywana przez globalny algorytm planujący.



Rysunek 10.7 Dwuwymiarowe, warstwowe mapy kosztów

Warstwa przeszkód (ang. obstacles layer) zawiera przeszkody z najbliższego otoczenia robota, które wykrywane są przez jego czujniki.

Warstwa proksemiczna (ang. proxemics layer) zawiera informację o wykrytych osobach w otoczeniu robota. Informacja ta może zostać wykorzystana do implementacji ruchu robotów, który jest akceptowalny przez ludzi. Przykładem może być poruszanie się w odpowiedniej odległości od ludzi przy ich wymijaniu, tak aby zmniejszyć ich dyskomfort.

SLAM lub globalna lokalizacja (ang. *SLAM/Global localization*)

Najczęściej stosowane jest jedno z dwóch rozwiązań. Pierwsze polega na utworzeniu mapy środowiska z wykorzystaniem metod SLAM i następnie działanie na już gotowej mapie (aktywna lokalizacja). W drugim przypadku, implementuje się podejście SPLAM [37], czyli jednocześnie uruchamiany jest algorytm SLAM oraz algorytm planowania ruchu. Zarówno jedno jak i drugie podejście ma pewne wady. Poruszanie się po utworzonej wcześniej mapie powoduje, że robot nie może opuszczać środowiska, dla którego przewidziana została mapa, ani reagować na zmiany tego otoczenia. Przykładowo, jeżeli zostanie usunięta przeszkoda obecna w trakcie tworzenia mapy, algorytm planujący nadal będzie omijał miejsce, w którym znajdowała się przeszkoda. Z drugiej strony, podejście SLAM jest bardziej złożone obliczeniowo i wymaga większej mocy obliczeniowej. Dokładniej metody lokalizacji oraz SLAM dla pojedynczych robotów omówiono w rozdziałach 4 i 5.

10.4.2 System sterowania niskiego poziomu (L1)

Niższa warstwa systemu sterowania, podobnie jak wyższa, charakteryzuje się modułową architekturą. W jej skład wchodzą następujące komponenty, pełniące rolę węzłów systemu:

- multiplekser komend sterujących,
- system bezpieczeństwa,
- lokalna lokalizacja,
- sterownik napędów,
- odometria,
- sterowniki sensorów.

Multiplekser komend sterujących (ang. *commands mux*)

Zastosowany multiplekser komend sterujących umożliwia zmianę źródła komend dla napędów robota przez przełączanie danych wejściowych na wyjście. Aktualnie możliwe są trzy źródła sterowania:

- system bezpieczeństwa,
- sterowanie manualne,
- oraz system nawigacji robota.

Komponent obsługuje priorytety sterowań, co jest szczególnie istotne w przypadku systemu bezpieczeństwa, który ma wyższy priorytet od pozostałych źródeł. Jeżeli w trakcie działania, system bezpieczeństwa wykryje niebezpieczną sytuację, będzie mógł wtedy zadziałać niezależnie od sterowania manualnego, czy nawigacyjnego.

System bezpieczeństwa (ang. *safety system*)

System bezpieczeństwa odpowiada za wykrywanie i zapobieganie kolizjom na najniższym poziomie systemu sterowania. Jeżeli system nawigacji nie wykryje przeszkody lub nie zdąży zareagować, wtedy rolą systemu bezpieczeństwa jest przejęcie kontroli i zapobiegnięcie kolizji. Z tego też powodu system bezpieczeństwa powinien działać w czasie rzeczywistym. W obecnej implementacji, działanie opiera się na detekcji przeszkód z wykorzystaniem danych z lidara, czujników taktylnych oraz na wykrywaniu anomalii w danych odometrycznych i danych z IMU. System może być też rozbudowany o dodatkowe sensory odległości, na przykład do wykrywania przeszkód wklęsłych, jak schody czy ubytki podłoża.

Lokalna lokalizacja (ang. *local localization*)

W celu poprawy jakości lokalizacji lokalnej opracowano fuzję danych odometrycznych oraz danych z IMU. Wykorzystano w tym celu filtr UKF [114], omówiony dokładniej w rozdziale 4.

Sterownik napędów (ang. *motors driver*)

Zadaniem sterownika napędów jest przesyłanie komend sterujących wygenerowanych przez moduł śledzenia trajektorii lub inne źródło sterowania do sterownika sprzętowego napędów. Natomiast to sterownik sprzętowy zapewnia takie generowanie sygnałów elektrycznych, aby zapewnić odpowiednie wartości prędkości kół. W szczególności regulatory PID (proporcjonalno-całkująco-różniczkujące) zaimplementowano po stronie sterownika sprzętowego.

Odometria (ang. *odometry*)

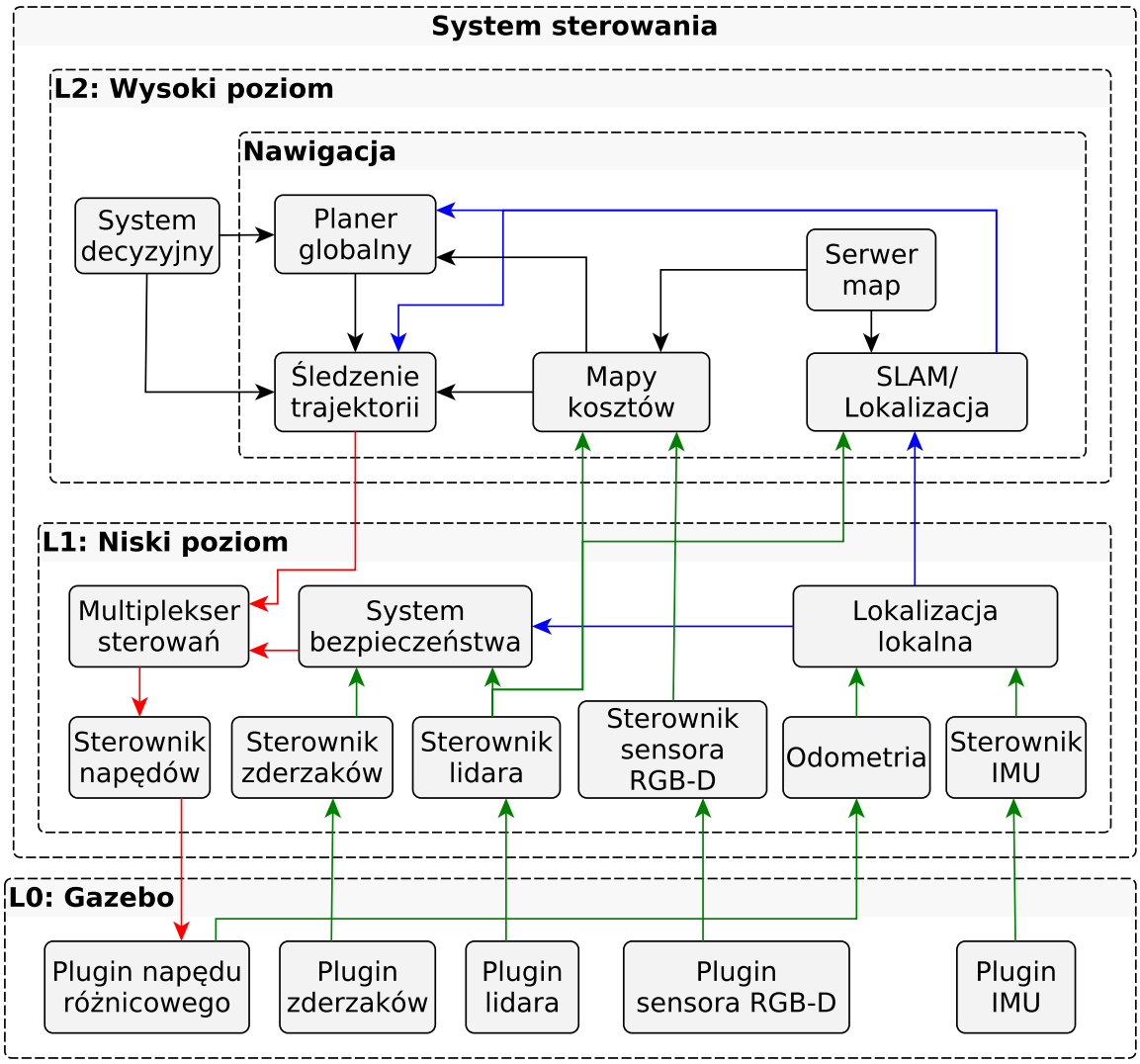
Komponent odpowiada za pobieranie danych odometrycznych ze sterownika sprzętowego i publikowanie ich w systemie sterownia na potrzeby algorytmu lokalnej lokalizacji. Dane odometryczne zawierają względną pozycję oraz prędkości robota. Sposób obliczania odometrii dla robotów z napędem różnicowym przedstawiono w rozdziale 4.

Sterowniki sensorów (ang. *(sensors drivers*)

W tej kategorii mieszczą się różnego rodzaju sterowniki sensorów. Między innymi sterownik lidara, sterownik czujnika RGB-D, oraz IMU.

10.5 Realizacja systemu w środowisku symulacyjnym

Jako środowisko symulacyjne wykorzystano symulator Gazebo [75] oraz system ROS [193]. Każdy z symulowanych robotów był wyposażony w przedstawiony już system sterowania, jedyna różnica polegała na wykorzystaniu wtyczek (ang. *plugins*) symulujących zachowanie sensorów oraz napędów robota (rys. 10.8). Symulowano takie czujniki jak lidar, IMU, sensor RGB-D, a także system odometryczny.



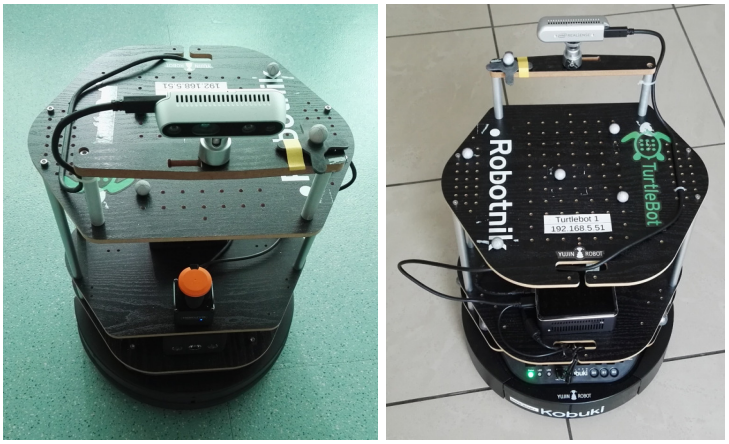
Rysunek 10.8 Architektura systemu sterowania symulowanego robota

10.6 Realizacja rzeczywistego systemu

System składał się z sześciu kołowych robotów mobilnych Turtlebot (rys. 10.9). Turtlebot (wersja 2) [181] jest niewielkim, kołowym robotem mobilnym przeznaczonym

głównie do celów edukacyjnych i badawczych. Jego duża popularność związana jest otwartą architekturą sprzętową, łatwością wprowadzania modyfikacji oraz znacznymi możliwościami integracji z różnego rodzaju oprogramowaniem. Właściwości mechaniczne robota:

- wymiary: 354×354×420 mm [182],
- waga: 6,3 kg,
- maksymalna prędkość liniowa: 0,65 m/s,
- maksymalna prędkość kątowa: $180^\circ/s$,
- nośność: 5 kg,
- prześwit: 15 mm,
- średnica kół: 76 mm.



Rysunek 10.9 Robot mobilny Turtlebot wchodzący w skład systemu wielorobotowego

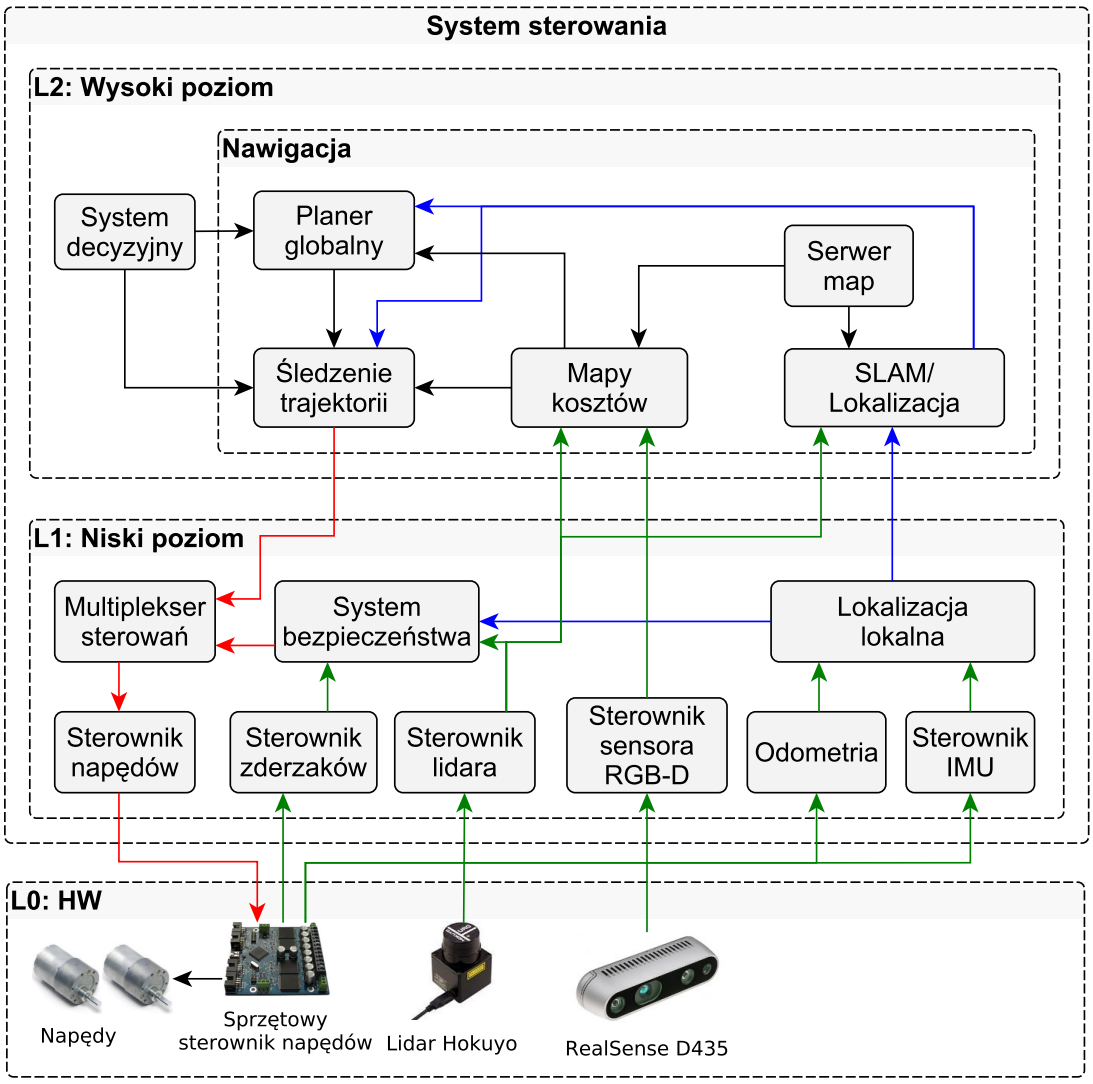
Standardowo, robot Turtlebot jest wyposażony w enkodery, zderzaki, sterowniki napędów oraz podstawowy moduł obliczeniowy. Moduł ten pełni rolę systemu sterowania niskiego poziomu, a do jego zadań należy obliczanie odometrii, przekazywanie sterowań do sterowników napędów, oraz komunikacja z wyższymi warstwami systemu sterowania robota, zwanego też mózgiem robota (ang. *robot brain*).

Na potrzeby laboratorium robotów mobilnych Politechniki Wrocławskiej, robot został wyposażony w dodatkowe systemy pomiarowe. Umieszczono na nim lidar Hokuyo UST-10LX, a także sensor RGB-D Intel RealSense D435. Ponadto robota wyposażono w dodatkowy akumulator oraz komputer Intel NUC i5, który pełni rolę jednostki obliczeniowej systemu sterowania wysokiego poziomu.

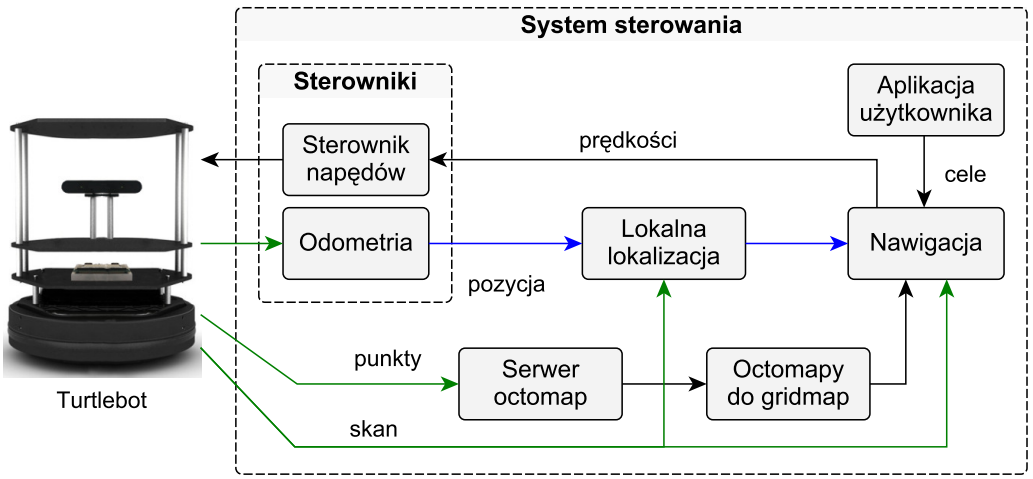
10.6.1 System sterowania robota mobilnego Turtlebot

Na rys. 10.10 przedstawiono architekturę systemu opartego na robocie mobilnym Turtlebot. Część wysokopoziomowa (L2) i niskopoziomowa (L1) różni się tylko w kwestii zastosowanych sterowników do czujników systemu bezpieczeństwa. Pozostałe komponenty nie różnią się od siebie. Natomiast część sprzętowa zawiera wspomniane już czujniki oraz sterownik sprzętowy odpowiadający za zbieranie danych z wybranych elementów robota oraz sterowanie napędami.

Do generowania octomap w trakcie przejazdów robotów, wykorzystywano uproszczony system sterowania (rys. 10.11). W sekcji 8 również zaprezentowano system do tworzenia octomap, składający się z dwóch etapów. Wspomniane rozwiązanie opiera się na zbieraniu danych w trakcie przejazdu i zapisywaniu ich do specjalnych plików *rosbag*. Następnie, pliki te są odtwarzane *offline*, a dane z nich trafiają do działającego systemu, co umożliwia utworzenie map.



Rysunek 10.10 Architektura systemu sterowania robota Turtlebot



Rysunek 10.11 Uproszczony system wykorzystywany do generowania octomap online

Rozdział 11

Podsumowanie

W ramach pracy przedstawiono szereg zagadnień związanych z lokalizacją i mapowaniem zarówno w przypadku pojedynczych robotów, jak i systemów wielu robotów. W rozdziale 2 zawarto przegląd systemów sensorycznych wykorzystywanych na potrzeby lokalizacji i mapowania. Przeprowadzono przegląd sposobów reprezentacji środowiska (rozdział 3) zarówno metrycznych, jak i topologicznych. Omówiono wiele metod lokalizacji, metod SLAM dla pojedynczych robotów, między innymi metody oparte na filtracji oraz optymalizacji. W rozdziale 6 omówiono kwestie związane z systemami wielu robotów mobilnych. Przedyskutowano korzyści i wyzwania związane z takimi systemami, w odniesieniu do pojedynczych robotów. Rozdział 7 dotyczy metod lokalizacji i mapowania dla wielu robotów, które są przeważnie rozwinięciem metod SLAM dla pojedynczych robotów. W rozdziale 8 przedyskutowano kwestie związane z łączeniem map z wielu robotów. Ponadto przedstawiono metodę łączenia map 3D opierającą się na ekstrakcji i dopasowaniu cech. Rozdział 9 zawiera omówienie sposobów dystrybucji map w systemach wielu robotów mobilnych. W rozdziale tym zaprezentowano również opracowane rozwiązanie problemu dystrybucji map w postaci grafowego serwera map. Kolejny rozdział 10 przedstawia zrealizowany system wielu robotów wykorzystujący kołowe roboty Turtlebot.

W pracy przedstawiono również rozwiązania wybranych problemów związanych z lokalizacją i mapowaniem oraz omówiono liczne eksperymenty, w szczególności:

- Opracowano i zaimplementowano metodę integracji map 3D z wielu robotów, przy nieznanych pozycjach początkowych, dla przypadku, gdy roboty nie spotykają się w trakcie eksploracji, ale mapy posiadają pewną część wspólną. Metoda działa w oparciu o ekstrakcję części wspólnych map, detekcję i deskrypcję cech oraz ich dopasowanie. Wyniki licznych eksperymentów zaprezentowano w rozdziale 8.

- Opracowano wielorobotowy serwer map 3D, a także przedstawiono wyniki przeprowadzonych eksperymentów (rozdział 9).

- Przeprowadzono badania eksperymentalne porównujące działanie wybranych metod SLAM, z wykorzystaniem optycznego systemu śledzenia ruchu (rozdział 5).

- Przeprowadzono badania eksperymentalne wybranych metod lokalizacji, z wykorzystaniem systemu do śledzenia obiektów (rozdział 4).

- Zaprojektowano oraz zrealizowano systemy wielu robotów mobilnych zarówno rzeczywiste, jak i symulacyjne. Opis systemów umieszczono w rozdziale 6.

- Zaprojektowano i zaimplementowano system umożliwiający automatyczny dobór parametrów metod SLAM w celu polepszenia ich efektywności (rozdział 5).
- Opracowano i zaimplementowano sposób kalibracji położenia czujnika głębi na potrzeby lokalizacji i mapowania 2D (rozdział 2).

Ponadto w ramach pracy próbowano znaleźć odpowiedź na pytanie w jaki sposób przeprowadzić fuzję danych z poszczególnych czujników robota, aby stworzyć model otoczenia i zlokalizować w nim roboty. W systemach wielorobotowych fuzja danych realizowana jest na wielu poziomach. Jeden z poziomów dotyczy fuzji w obrębie danego robota, kolejny w obrębie danych wymienianych między robotami. Aby odpowiedzieć na przedstawione pytanie, skupmy się jednak na pojedynczym robocie. W celu umożliwienia poprawnego działania funkcji systemu nawigacji, takich jak planowanie ruchu i śledzenie trajektorii, ważne jest, aby tworzona mapa jak najlepiej odwzorowywała otoczenie oraz, aby estymowana pozycja robota była możliwie dokładna, a przy tym aktualna. Z jednej strony, zapewnienie aktualnej pozycji, czyli o niewielkich opóźnieniach, jest możliwe przez zastosowanie względnych metod lokalizacji, czyli śledzących przemieszczenie robota względem poprzednich pozycji. Metody te są zwykle bardzo szybkie, ale nie nadają się do estymacji pozycji w dłuższym horyzoncie czasowym z powodu dryfu. Z tego powodu wykorzystuje się dodatkowe systemy, lokalizujące robota względem globalnego układu odniesienia, mogą to być metody dopasowania odczytów do map lub odbiorniki systemów nawigacji satelitarnej. Otrzymywane informacje o pozycji globalnej są wykorzystywane do korekcji pozycji robota lokalizowanego z dużą częstotliwością również lokalnie. Wykorzystanie dwóch systemów i odpowiedniej fuzji pozwala zrealizować cel w postaci zapewnienia aktualnej i precyzyjnej pozycji robota. Znajomość dokładnej pozycji robota ma ogromne znaczenie w przypadku tworzenia map otoczenia, ponieważ mapowanie w obrębie pojedynczego robota zwykle opiera się na dodawaniu do modelu nowych danych pomiarowych, przy założeniu, że zostały zebrane w danej pozycji robota.

Pojawia się również kwestia odpowiedniej reprezentacji środowiska, tak aby zarówno modelować niepewność obserwacji, ale też umożliwić planowanie działań robota. Jedno z podejść zakłada modelowanie niepewności w postaci prawdopodobieństwa występowania przeszkody w danym fragmencie środowiska. Takie podejście umożliwia tworzenie mapy jednocześnie za pomocą różnych czujników, przy założeniu znajomości modeli probabilistycznych tych czujników. O ile rozwiązanie to dobrze sprawdza się na potrzeby lokalizacji, o tyle nie zawsze jest wystarczające w przypadku planowania działań robotów. Jednym z rozwiązań, jest wykorzystanie map hybrydowych składających się z części metrycznych, przechowujących informacje w ujęciu probabilistycznym oraz części topologicznych wykorzystywanych podczas planowania działań. Cześć topologiczna może zawierać informacje o drzwiach występujących w budynkach, windach, a także innych specyficznych punktach otoczenia.

Próbowano również odpowiedzieć na pytanie przedstawione we wstępie, dotyczące tego jakie informacje warto przesyłać między robotami i jak przeprowadzić fuzję tych danych. Istnieje kilka sposób realizacji fuzji danych, które zależą w głównej mierze od tego, jakie dane są przesyłane między robotami. Jedno z podejść zakłada przesyłanie nieprzetworzonych danych z czujników do innych robotów, tak aby mogły one te dane przetworzyć i wykorzystać. Takie podejście ma pewne wady, między innymi potrzebuje zwiększonej przepustowości łącza komunikacyjnego, ale też powoduje, że każdy robot musi przetworzyć te same dane, więc obliczenia są nadmiarowe. Drugie podejście zakłada realizację fuzji danych w ramach poszczególnych robotów i dzielenie się rezultatami

z innymi robotami. Przykładem może być generowanie map lokalnych przez roboty, a następnie przesyłanie ich do innych robotów. Takie podejście wykorzystano w pracy, opracowując metodę łączenia lokalnych map z poszczególnych robotów, przy braku informacji o ich wzajemnych pozycjach.

W oparciu o przedstawione, opracowane metody, a także wyniki przeprowadzonych badań, można stwierdzić, że rozwiązanie opierające się na wymianie map między robotami pozwala zwiększyć efektywność procesu SLAM, w porównaniu do realizacji tego zadania przez pojedynczego robota.

Odnośnie możliwych kierunków rozwoju pracy, jednym z nich są dalsze modyfikacje metody łączenia map z wielu robotów. Znaczącym ulepszeniem może być wykorzystanie metod AI do wyszukiwania wspólnych obszarów map. Wydaje się to perspektywicznym rozwiązaniem, ponieważ mogłoby znacząco poprawić skuteczność dopasowania i jednocześnie przyspieszyć ten etap integracji map, który jest obecnie najbardziej złożony obliczeniowo.

Bibliografia

[1] 3D Map Server Software. https://github.com/mdrwiega/3d_map_server.

[2] D. W. R. Abdulmajeed, R. Z. Mansoor. Comparison Between 2D and 3D Mapping For Indoor Environments. *Journal of Engineering Research and Technology*, 2:8, 2013.

[3] N. Acosta, J. Toloza. Techniques to improve the GPS precision. *International Journal of Advanced Computer Science and Applications*, 3(8), 2012.

[4] N. Adluru, L. J. Latecki, M. Sobel, R. Lakaemper. Merging maps of multiple robots. *2008 19th International Conference on Pattern Recognition*, wolumen 1, strony 1–4, Tampa, FL, USA, Grudzień 2008. IEEE.

[5] Advanced Navigation. Orientus IMU. https://www.advancednavigation.com/solutions/orientus/.

[6] Agarwal S. and Mierle K. et al. Ceres solver. http://ceres-solver.org.

[7] I. Andersone. The Characteristics of the Map Merging Methods: A Survey. *Scientific Journal of Riga Technical University. Computer Sciences*, 41(1):113–121, 2010.

[8] Apex.AI. Apex.AI. https://www.apex.ai/.

[9] M. O. A. Aqel, M. H. Marhaban, M. I. Saripan, N. B. Ismail. Review of visual odometry: Types, approaches, challenges, and applications. *SpringerPlus*, 5(1), Grudzień 2016.

[10] K. Arent, J. Jakubiak, M. Drwięga, M. Cholewiński, G. Stollnberger, M. Giuliani, M. Tscheligi, D. Szczesniak-Stanczyk, M. Janowski, W. Brzozowski, A. Wysokinski. Control of mobile robot for remote medical examination: Design concepts and users' feedback from experimental studies. *2016 9th International Conference on Human System Interactions (HSI)*, strony 76–82, Portsmouth, United Kingdom, Lipiec 2016. IEEE.

[11] K. Arent, D. Szczęśniak-Stańczyk, M. Cholewiński, Ł. Chojnacki, W. Domski, M. Drwięga, A. Kurnicki, B. Stańczyk, J. Jakubiak, M. Janiak, B. Kreczmer. Zastosowanie robota do telediagnostyki medycznej z perspektywy projektu ReMeDi. *Krajowa Konferencja Robotyki*, strona 12, 2016.

[12] A. E. Arslan, K. Kalkan. Comparison of working efficiency of terrestrial laser scanner in day and night conditions. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XL-7/W2:19–21, 2013.

[13] R. Arumugam, V. R. Enti, Liu Bingbing, Wu Xiaojun, K. Baskaran, Foong Foo Kong, A. S. Kumar, Kang Dee Meng, Goh Wai Kit. DAvinCi: A cloud computing framework for service robots. *2010 IEEE International Conference on Robotics and Automation*, strony 3084–3089, Anchorage, AK, Maj 2010. IEEE.

[14] F. Aurenhammer. Voronoi diagrams—a survey of a fundamental geometric data structure. *ACM Computing Surveys*, 23(3):345–405, Wrzesień 1991.

[15] H. Azartash, N. Banai, T. Q. Nguyen. An integrated stereo visual odometry for robotic navigation. *Robotics and Autonomous Systems*, 62(4):414–421, Kwiecień 2014.

[16] B. Bacca, J. Salvi, X. Cufí. Appearance-based mapping and localization for mobile robots using a feature stability histogram. *Robotics and Autonomous Systems*, 59(10):840–857, Październik 2011.

[17] D. Ball, S. Heath, J. Wiles, G. Wyeth, P. Corke, M. Milford. OpenRatSLAM: An open source brain-based SLAM system. *Autonomous Robots*, 34(3):149–176, Kwiecień 2013.

[18] Been,Kim, M. Kaess, L. Fletcher, J. Leonard, A. Bachrach, N. Roy, S. Teller. Multiple relative pose graphs for robust cooperative mapping. *2010 IEEE International Conference on Robotics and Automation*, strony 3185–3192, Anchorage, AK, Maj 2010. IEEE.

[19] H. E. Benseddik, O. Djekoune, M. Belhocine. SIFT and SURF Performance Evaluation for Mobile Robot-Monocular Visual Odometry. *Journal of Image and Graphics*, strony 70–76, 2014.

[20] H. Berti, A. D. Sappa, O. E. Agamennoni. Improved Dynamic Window Approach by using Lyapunov Stability Criteria. *Latin American Applied Research*, strona 10, 2008.

[21] P. J. Besl, N. D. McKay. A method for registration of 3-D shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–256, 1992.

[22] P. Biber, W. Strasser. The normal distributions transform: A new approach to laser scan matching. *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453)*, wolumen 3, strony 2743–2748, Las Vegas, Nevada, USA, 2003. IEEE.

[23] A. Birk, S. Carpin. Merging Occupancy Grid Maps From Multiple Robots. *Proceedings of the IEEE*, 94(7):1384–1397, Lipiec 2006.

[24] G. Bitelli, A. Simone, F. Girardi, C. Lantieri. Laser scanning on road pavements: A new approach for characterizing surface texture. *Sensors*, 12(7):9110–9128, 2012.

[25] T. M. Bonanni, B. Della Corte, G. Grisetti. 3-D Map Merging on Pose Graphs. *IEEE Robotics and Automation Letters*, 2(2):1031–1038, Kwiecień 2017.

[26] O. Booij, B. Terwijn, Z. Zivkovic, B. Krose. Navigation using an appearance based topological map. *Proceedings 2007 IEEE International Conference on Robotics and Automation*, strony 3927–3932, Rome, Italy, Kwiecień 2007. IEEE.

[27] J. Borenstein, H. R. Everett, L. Feng. *Where Am I Sensors and Methods for Mobile Robot Positioning.Pdf*. 1995.

[28] J. Borenstein, H. R. Everett, L. Feng, D. Wehe. Mobile Robot Positioning & Sensors and Techniques. strona 23, 1997.

[29] J. Borenstein, L. Feng. UMBmark: A benchmark test for measuring odometry errors in mobile robots. W. J. Wolfe, C. H. Kenyon, redaktorzy, *Photonics East '95*, strony 113–124, Philadelphia, PA, Grudzień 1995.

[30] J. Borenstein, Liqiang Feng. Correction of systematic odometry errors in mobile robots. *Proceedings 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems. Human Robot Interaction and Cooperative Robots*, wolumen 3, strony 569–574, Pittsburgh, PA, USA, 1995. IEEE Comput. Soc. Press.

[31] D. Borrmann, i in. The 3D Hough transform for plane detection in point clouds: A review and a new accumulator design. *3D Res.*, strony 32:1–32:13, 2011.

[32] D. M. Bradley. Odometry: Calibration and Error Modeling. strona 4, 1997.

[33] R. G. Brown, P. Y. C. Hwang. *Introduction to Random Signals and Applied Kalman Filtering: With MATLAB Exercises and Solutions; 3rd Ed.* Wiley, New York, NY, 1997.

[34] K. Brzozowska, A. L. Dawidowicz. Metoda filtru cząsteczkowego. strona 39, 2009.

[35] W. Burgard, M. Hebert. World Modeling. *Springer Handbook of Robotics*, strony 853–867. Berlin, 2008.

[36] S. Carpin. Fast and accurate map merging for multi-robot systems. *Autonomous Robots*, 25(3):305–316, 2008.

[37] S. Chakravorty, R. Saha. Simultaneous planning localization and mapping: A hybrid bayesian/ frequentist approach. *American Control Conference, 2008*, strony 1226–1231, June 2008.

[38] H. Chen, B. Bhanu. 3D Free-Form Object Recognition in Range Images Using Local Surface Patches. *th International Conference on Pattern Recognition*, strona 4.

[39] Y. Chen, H. Hu. Internet of intelligent things and robot as a service. *Simulation Modelling Practice and Theory*, 34:159–171, Maj 2013.

[40] Z. Chen. Bayesian filtering: From Kalman filters to particle filters, and beyond. *Statistics*, 182:69, 2003.

[41] L. F. Contreras-Samame, S. Dominguez-Quijada, O. Kermorgant, P. Martinet. Co-Mapping: Multi-robot Sharing and Generation of 3D-Maps applied to rural and urban scenarios. *2018 15th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, strony 789–794, Singapore, Listopad 2018. IEEE.

[42] D. S. O. Correa, D. F. Sciotti, M. G. Prado, D. O. Sales, D. F. Wolf, F. S. Osorio. Mobile Robots Navigation in Indoor Environments Using Kinect Sensor. *2012 Second Brazilian Conference on Critical Embedded Systems*, strony 36–41, Sao Paulo, Campinas, Brazil, Maj 2012. IEEE.

[43] R. N. Darmanin, M. K. Bugeja. A review on multi-robot systems categorised by application domain. *2017 25th Mediterranean Conference on Control and Automation (MED)*, strony 701–706, Valletta, Malta, Lipiec 2017. IEEE.

[44] A. J. Davison, I. D. Reid, N. D. Molton, O. Stasse. MonoSLAM: Real-Time Single Camera SLAM. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(6):1052–1067, Czerwiec 2007.

[45] D. De Gregorio, L. Di Stefano. SkiMap: An Efficient Mapping Framework for Robot Navigation. *arXiv:1704.05832 [cs]*, Kwiecień 2017.

[46] depth_nav_tools. `https://github.com/mdrwiega/depth_nav_tools`.

[47] K. G. Derpanis. Overview of the RANSAC Algorithm. strona 2.

[48] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[49] R. Doriya, S. Mishra, S. Gupta. A brief survey and analysis of multi-robot communication and coordination. *International Conference on Computing, Communication & Automation*, strony 1014–1021, Greater Noida, India, Maj 2015. IEEE.

[50] K. Dróżdż. Zastosowanie bezśladowego filtru Kalmana w sterowaniu adaptacyjnym układu dwumasowego. *Przegląd elektrotechniczny*, 1(4):193–198, Kwiecień 2016.

[51] M. Drwięga. Validation and comparison of navigation systems in wheeled platforms with different drive mechanisms and sensors. 2015.

[52] M. Drwięga. Zestaw narzędzi wspomagających nawigację kołowym robotem mobilnym wyposażonym w sensor głębi. *Krajowa Konferencja Robotyki*, strona 10, 2016.

[53] M. Drwięga. Efficient Integration of Octree Based Maps in Multi-Robot System. *2018 23rd International Conference on Methods & Models in Automation & Robotics (MMAR)*, strony 487–492, Miedzyzdroje, Sierpień 2018. IEEE.

[54] M. Drwięga. Features Matching based Merging of 3D Maps in Multi-Robot Systems. *2019 24th International Conference on Methods and Models in Automation and Robotics (MMAR)*, strony 663–668, Międzyzdroje, Poland, Sierpień 2019. IEEE.

[55] M. Drwięga. 3D Maps Integration Based on Overlapping regions matching. *Journal of Automation, Mobile Robotics & Intelligent Systems*, accepted on 22.11.2021(3), 2021.

[56] M. Drwięga. Incremental 3D Maps Server based on Feature Matching for Multi-robot Systems. *2021 25th International Conference on Methods and Models in Automation and Robotics (MMAR)*, Międzyzdroje, Poland, 2021.

[57] M. Drwięga, J. Jakubiak. A Set of Depth Sensor Processing ROS Tools for Wheeled Mobile Robot Navigation. *Journal of Automation, Mobile Robotics & Intelligent Systems*, 11(2):48–56, Czerwiec 2017.

[58] C. Duan, S. Junginger, J. Huang, K. Jin, K. Thurow. Deep Learning for Visual SLAM in Transportation Robotics: A review. *Transportation Safety and Environment*, 1(3):177–184, Grudzień 2019.

[59] H. Durrant-Whyte, T. Bailey. Simultaneous Localisation and Mapping (SLAM): Part I The Essential Algorithms. strona 9, 2006.

[60] F. Endres, J. Hess, N. Engelhard, J. Sturm, D. Cremers, W. Burgard. An evaluation of the RGB-D SLAM system. *2012 IEEE International Conference on Robotics and Automation*, strony 1691–1696, St Paul, MN, USA, Maj 2012. IEEE.

[61] J. Engel, T. Schöps, D. Cremers. LSD-SLAM: Large-Scale Direct Monocular SLAM. D. Fleet, T. Pajdla, B. Schiele, T. Tuytelaars, redaktorzy, *Computer Vision – ECCV 2014*, wolumen 8690, strony 834–849. Springer International Publishing, Cham, 2014.

[62] N. Engel, S. Hoermann, M. Horn, V. Belagiannis, K. Dietmayer. DeepLocalization: Landmark-based Self-Localization with Deep Neural Networks. *arXiv:1904.09007 [cs, stat]*, Lipiec 2019.

[63] G. Erinc, S. Carpin. Anytime Merging of Appearance Based Maps. strona 7, 2012.

[64] H. R. Everett. *Sensors for Mobile Robots: Theory and Application*. A.K. Peters, Wellesley, Mass, 1995.

[65] J. Fabian, G. M. Clayton. Error Analysis for Visual Odometry on Indoor, Wheeled Mobile Robots With 3-D Sensors. *IEEE/ASME Transactions on Mechatronics*, 19(6):1896–1906, Grudzień 2014.

[66] P. Fankhauser, M. Bloesch, D. Rodriguez, R. Kaestner, M. Hutter, R. Siegwart. Kinect v2 for mobile robot navigation: Evaluation and modeling. *2015 International Conference on Advanced Robotics (ICAR)*, strony 388–394, Istanbul, Turkey, Lipiec 2015. IEEE.

[67] R. Faragher. Understanding the Basis of the Kalman Filter Via a Simple and Intuitive Derivation [Lecture Notes]. *IEEE Signal Processing Magazine*, 29(5):128–132, Wrzesień 2012.

[68] A. Ferrein, I. Scholl, T. Neumann, K. Krückel, S. Schiffer. A System for Continuous Underground Site Mapping and Exploration. M. Reyhanoglu, G. De Cubber, redaktorzy, *Unmanned Robotic Systems and Applications*. IntechOpen, Kwiecień 2020.

[69] M. Fischler, R. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, 1981.

[70] D. Fofi, T. Sliwa, Y. Voisin. A comparative survey on invisible structured light. J. R. Price, F. Meriaudeau, redaktorzy, *Machine Vision Applications in Industrial Inspection XII*, wolumen 5303, strony 90–98. International Society for Optics and Photonics, SPIE, 2004.

[71] T. A. Foundation. Autoware. `https://www.autoware.auto/`.

[72] D. Fox. Adapting the Sample Size in Particle Filters Through KLD-Sampling. *The International Journal of Robotics Research*, 22(12):985–1003, Grudzień 2003.

[73] D. Fox, W. Burgard, S. Thrun. The dynamic window approach to collision avoidance. *Robotics Automation Magazine, IEEE*, 4(1):23–33, Mar 1997.

[74] F. Fraundorfer, C. Engels, D. Nister. Topological mapping, localization and navigation using image collections. *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, strony 3872–3877, San Diego, CA, USA, Październik 2007. IEEE.

[75] Gazebo. Simulation. `http://gazebosim.org/`.

[76] A. Geiger, P. Lenz, R. Urtasun. Are we ready for autonomous driving? The KITTI vision benchmark suite. *2012 IEEE Conference on Computer Vision and Pattern Recognition*, strony 3354–3361, Providence, RI, Czerwiec 2012. IEEE.

[77] K. Gergont, E. Kierys. Wykorzystanie technologii skaningu naziemnego w procesie dokumentowania wypadków drogowych i miejsc przestępstw. Praca magisterska, AGH, 2009.

[78] S. Gholami Shahbandi, M. Magnusson. 2D map alignment with region decomposition. *Autonomous Robots*, 43(5):1117–1136, Czerwiec 2019.

[79] P. Goel, S. I. Roumeliotis, G. S. Sukhatme. Robot Localization Using Relative and Absolute Position Estimates. strona 7.

[80] J. M. Gomez, redaktor. *Kalman Filtering*. Mathematics Research Developments. Nova Science Publishers, Hauppauge, N.Y, 2011.

[81] Google. Cartographer. `https://google-cartographer-ros.readthedocs.io/`.

[82] Graphviz. Visualization Software. `https://graphviz.org/`.

[83] J. Green, D. Krakauer. New iMEMS Angular Rate-Sensing Gyroscope. *Analog Dialogue*, 37, 3 2003.

[84] M. S. Grewal, A. P. Andrews. *Kalman Filtering: Theory and Practice Using MATLAB*. Wiley, Hoboken, N.J, wydanie 3rd ed, 2008.

[85] G. Grisettiyz, C. Stachniss, W. Burgard. Improving Grid-based SLAM with Rao-Blackwellized Particle Filters by Adaptive Proposals and Selective Resampling. *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, wolumen 1, strony 2432–2437, Barcelona, Spain, Kwiecień 2005. IEEE.

[86] M. Grupp. evo: Python package for the evaluation of odometry and slam. `https://github.com/MichaelGrupp/evo`, 2017.

[87] J. Han, P. Yin, Y. He, F. Gu. Enhanced ICP for the Registration of Large-Scale 3D Environment Models: An Experimental Study. *Sensors*, 16(2):228, Luty 2016.

[88] X.-F. Han, J. S. Jin, J. Xie, M.-J. Wang, W. Jiang. A comprehensive review of 3D point cloud descriptors. *arXiv:1802.02297 [cs]*, Luty 2018.

[89] C. Harris, M. Stephens. A Combined Corner and Edge Detector. *Procedings of the Alvey Vision Conference 1988*, strony 23.1–23.6, Manchester, 1988. Alvey Vision Club.

[90] Y. He, B. Liang, J. Yang, S. Li, J. He. An Iterative Closest Points Algorithm for Registration of 3D Laser Scanner Point Clouds with Geometric Features. *Sensors*, 17(8):1862, Sierpień 2017.

[91] S. Hellesund. Measuring the speed of sound in air using a smartphone and a cardboard tube. *Physics Education*, 54(3):035015, Maj 2019.

[92] W. Hess, D. Kohler, H. Rapp, D. Andor. Real-time loop closure in 2D LIDAR SLAM. *2016 IEEE International Conference on Robotics and Automation (ICRA)*, strony 1271–1278, Stockholm, Sweden, Maj 2016. IEEE.

[93] K. L. Ho, P. Newman. Multiple Map Intersection Detection using Visual Appearance. strona 6, 2005.

[94] P. Hofmann, Z. Tashman. Hidden Markov Models and Their Application for Predicting Failure Events. V. V. Krzhizhanovskaya, G. Závodszky, M. H. Lees, J. J. Dongarra, P. M. A. Sloot, S. Brissos, J. Teixeira, redaktorzy, *Computational Science – ICCS 2020*, wolumen 12139, strony 464–477. Springer International Publishing, Cham, 2020.

[95] Honeywell. Hall Effect Sensing and Application. `http://sensing.honeywell.com/index.php?ci_id=47847`.

[96] R. Horaud, M. Hansard, G. Evangelidis, C. Menier. An Overview of Depth Cameras and Range Scanners Based on Time-of-Flight Technologies. *Machine Vision and Applications*, 27(7):1005–1020, Październik 2016.

[97] A. Hornung. OctoMap 3D scan dataset. http://ais.informatik.uni-freiburg.de/projects/datasets/octomap/.

[98] A. Hornung, K. M. Wurm, M. Bennewitz. Humanoid robot localization in complex indoor environments. *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, strony 1690–1695, Taipei, Październik 2010. IEEE.

[99] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, W. Burgard. OctoMap: An efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots*, 34(3):189–206, Kwiecień 2013.

[100] A. Howard. Multi-robot mapping using manifold representations. *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004*, strony 4198–4203 Vol.4, New Orleans, LA, USA, 2004. IEEE.

[101] A. Howard. Multi-robot Simultaneous Localization and Mapping using Particle Filters. strona 8, 2006.

[102] Hu Lin, Gu Zhengqi, Huang Jing, Tang Yali, Huang Wei. Global positioning of off-road vehicles by sensor fusion for precision agriculture. *2008 7th World Congress on Intelligent Control and Automation*, strony 7005–7010, Chongqing, China, 2008. IEEE.

[103] D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M. Y. Vardi, G. Weikum, M. Calonder, V. Lepetit, C. Strecha, P. Fua. BRIEF: Binary Robust Independent Elementary Features. K. Daniilidis, P. Maragos, N. Paragios, redaktorzy, *Computer Vision – ECCV 2010*, wolumen 6314, strony 778–792. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[104] IMU Data Fusing. Complementary, Kalman, and Mahony Filter. `http://www.olliw.eu/2013/imu-data-fusing/`, 2013.

[105] Intel. Intel RealSense Product Family D400 Series, 2020.

[106] M. Jaimez, J. G. Monroy, J. Gonzalez-Jimenez. Planar odometry from a radial laser scanner. A range flow-based approach. *2016 IEEE International Conference on Robotics and Automation (ICRA)*, strony 4479–4485, Stockholm, Maj 2016. IEEE.

[107] J. Jakubiak, M. Drwięga, A. Kurnicki. Development of a mobile platform for a remote medical teleoperation robot. *2016 21st International Conference on Methods and Models in Automation and Robotics (MMAR)*, strony 1137–1142, Miedzyzdroje, Poland, Sierpień 2016. IEEE.

[108] J. Jakubiak, M. Drwięga, B. Stanczyk. Control and perception system for ReMeDi robot mobile platform. *2015 20th International Conference on Methods and Models in Automation and Robotics (MMAR)*, strony 750–755, Miedzyzdroje, Poland, Sierpień 2015. IEEE.

[109] J. Jessup, S. N. Givigi, A. Beaulieu. Merging of octree based 3D occupancy grid maps. *2014 IEEE International Systems Conference Proceedings*, strony 371–377, Ottawa, ON, Canada, Marzec 2014. IEEE.

[110] J. Jessup, S. N. Givigi, A. Beaulieu. Robust and efficient multi-robot 3D mapping with octree based occupancy grids. *2014 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, strony 3996–4001, San Diego, CA, USA, Październik 2014. IEEE.

[111] Y. Jiang, G. Xiong, H. Chen, D.-J. Lee. Incorporating a Wheeled Vehicle Model in a New Monocular Visual Odometry Algorithm for Dynamic Outdoor Environments. *Sensors*, 14(9):16159–16180, Wrzesień 2014.

[112] Y. Jiang, Y. Xu, Y. Liu. Performance evaluation of feature detection and matching in stereo visual odometry. *Neurocomputing*, 120:380–390, Listopad 2013.

[113] K. Jo, C. Kim, M. Sunwoo. Simultaneous Localization and Map Change Update for the High Definition Map-Based Autonomous Driving Car. *Sensors*, 18(9):3145, Wrzesień 2018.

[114] S. Julier, J. Uhlmann. Unscented filtering and nonlinear estimation. *Proceedings of the IEEE*, 92(3):401–422, Mar 2004.

[115] S. J. Julier, J. K. Uhlmann. A New Extension of the Kalman Filter to Nonlinear Systems. *Signal Processing, Sensor Fusion, and Target Recognition VI*, wolumen 3068, strony 182–193. SPIE, wydanie ivan kadar, 1997.

[116] E. Kagan, N. Shvalb, I. Ben-Gal, redaktorzy. *Autonomous Mobile Robots and Multi-Robot Systems: Motion-Planning, Communication, and Swarming*. Wiley, Hoboken, NJ, 2019.

[117] D. Kakuma, S. Tsuichihara, G. A. G. Ricardez, J. Takamatsu, T. Ogasawara. Alignment of Occupancy Grid and Floor Maps Using Graph Matching. *2017 IEEE 11th International Conference on Semantic Computing (ICSC)*, strony 57–60, San Diego, CA, USA, 2017. IEEE.

[118] M. Kalantari, M. Nechifor. Accuracy and utility of the Structure Sensor for collecting 3D indoor information. *Geo-spatial Information Science*, 19(3):202–209, Lipiec 2016.

[119] R. E. Kalman. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 82(1):35, 1960.

[120] K. Kamarudin, S. Mamduh, A. Shakaff, A. Zakaria. Performance Analysis of the Microsoft Kinect Sensor for 2D Simultaneous Localization and Mapping (SLAM) Techniques. *Sensors*, 14(12):23365–23387, Grudzień 2014.

[121] K. Kamarudin, S. M. Mamduh, A. Y. M. Shakaff, S. M. Saad, A. Zakaria, A. H. Abdullah, L. M. Kamarudin. Method to convert Kinect's 3D depth data to a 2D map for indoor SLAM. *2013 IEEE 9th International Colloquium on Signal Processing and Its Applications*, strony 247–251, Kuala Lumpur, Marzec 2013. IEEE.

[122] H. Karaoğuz, H. I. Bozma. Merging of appearance-based place knowledge among multiple robots. *Autonomous Robots*, 44(6):1009–1027, Lipiec 2020.

[123] J. Kędzierski, M. Janiak. Budowa robota społecznego FLASH. *12 Krajowa Konferencja Robotyki*, strona 15, 2012.

[124] B. Kehoe, S. Patil, P. Abbeel, K. Goldberg. A Survey of Research on Cloud Robotics and Automation. *IEEE Transactions on Automation Science and Engineering*, 12(2):398–409, Kwiecień 2015.

[125] C. Kerl, J. Sturm, D. Cremers. Robust odometry estimation for RGB-D cameras. *2013 IEEE International Conference on Robotics and Automation*, strony 3748–3754, Karlsruhe, Germany, Maj 2013. IEEE.

[126] M. Khader, S. Cherian. An Introduction to Automotive LIDAR. strona 7, 2020.

[127] D. Kiss, G. Tevesz. Advanced dynamic window based navigation approach using model predictive control. *2012 17th International Conference on Methods & Models in Automation & Robotics (MMAR)*, strony 148–153, Miedzyzdroje, Poland, Sierpień 2012. IEEE.

[128] G. Klein, D. Murray. Parallel Tracking and Mapping for Small AR Workspaces. *2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality*, strony 1–10, Nara, Japan, Listopad 2007. IEEE.

[129] S. Kohlbrecher, O. von Stryk, J. Meyer, U. Klingauf. A flexible and scalable SLAM system with full 3D motion estimation. *2011 IEEE International Symposium on Safety, Security, and Rescue Robotics*, strony 155–160, Kyoto, Japan, Listopad 2011. IEEE.

[130] P. Kohout, M. De Bortoli, J. Ludwiger, T. Ulz, G. Steinbauer. A multi-robot architecture for the RoboCup Logistics League. *e & i Elektrotechnik und Informationstechnik*, 137(6):291–296, Październik 2020.

[131] K. Konolige, D. Fox, B. Limketkai, J. Ko, B. Stewart. Map merging for distributed robot navigation. *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453)*, wolumen 1, strony 212–217, Las Vegas, Nevada, USA, 2003. IEEE.

[132] K. Konolige, E. Marder-Eppstein, B. Marthi. Navigation in hybrid metric-topological maps. *2011 IEEE International Conference on Robotics and Automation*, strony 3041–3047, Shanghai, China, Maj 2011. IEEE.

[133] P. Kozierski. Wykorzystanie filtru cząsteczkowego w problemie identyfikacji układów automatyki. strona 14, 2012.

[134] B. Kreczmer. Objects Localization and Differentiation Using Ultrasonic Sensors. H. Yussof, redaktor, *Robot Localization and Map Building*. InTech, Marzec 2010.

[135] B. Kreczmer. Estimation of the Azimuth Angle of the Arrival Direction for an Ultrasonic Signal by Using Indirect Determination of the Phase Shift. 44(3):585–601, 2019.

[136] L. Kurnianggoro, V. Hoang, K. Jo. Calibration of a 2d laser scanner system and rotating platform using a point-plane constraint. *Computer Science and Information Systems, Vol. 12, No. 1*, strony 307 – 322, 2015.

[137] H.-J. Kwak, D.-H. Lee, J.-M. Hwang, J.-H. Kim, C.-K. Kim, G.-T. Park. Improvement of the inertial sensor-based localization for mobile robots using multiple estimation windows filter. *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, strony 876–881, Vilamoura-Algarve, Portugal, Październik 2012. IEEE.

[138] M. Labschutz, S. Bruckner, M. E. Groller, M. Hadwiger, P. Rautek. JiTTree: A Just-in-Time Compiled Sparse GPU Volume Data Structure. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):1025–1034, Styczeń 2016.

[139] E. Lachat, H. Macher, M.-A. Mittet, T. Landes, P. Grussenmeyer. First experiences with Kinect v2 sensor for close range 3D modelling. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XL-5/W4:93–100, Luty 2015.

[140] Y. Latif, C. Cadena, J. Neira. Robust Loop Closing Over Time. strona 8, 2017.

[141] H.-C. Lee, B.-H. Lee. Improved Feature Map Merging Using Virtual Supporting Lines for Multi-Robot Systems. *Advanced Robotics*, 25(13-14):1675–1696, 2011.

[142] H.-C. Lee, Seung-Hwan Lee, Tae-Seok Lee, Doo-Jin Kim, B.-H. Lee. A survey of map merging techniques for cooperative-SLAM. *2012 9th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*, wolumen 1, strony 285–287, Daejeon, Korea (South), Listopad 2012. IEEE.

[143] H. S. Lee, K. M. Lee. Multi-robot SLAM Using Ceiling Vision. *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, wolumen 1 serii *1*, strona 6, Październik 2009.

[144] J. Lenz, A. S. Edelstein. Magnetic Sensors and Their Applications. *IEEE Sensors Journal*, 6(3), 6 2006.

[145] Á. León García, R. Barea Navarro, L. M. Bergasa Pascual, E. López Guillén, M. Ocaña Miguel, D. Schleicher Gómez. SLAM and map merging. *Journal of Physical Agents (JoPha)*, 3(1):13–23, 2009.

[146] L. Li. Time-of-flight camera – an introduction. Raport instytutowy, Texas Instruments, 2014.

[147] K. Lingemann, A. Nüchter, J. Hertzberg, H. Surmann. High-speed laser localization for mobile robots. *Robotics and Autonomous Systems*, 51(4):275–296, Czerwiec 2005.

[148] LIREC. EU Project. `https://cordis.europa.eu/project/id/215554`.

[149] C. Liu. *Foundations of MEMS*, rozdzia/l Piezoresistive Sensors. Prentice Hall, 2011.

[150] S. Lovegrove, A. J. Davison, J. Ibanez-Guzman. Accurate visual odometry from a rear parking camera. *2011 IEEE Intelligent Vehicles Symposium (IV)*, strony 788–793, Baden-Baden, Germany, Czerwiec 2011. IEEE.

[151] D. V. Lu, D. Hershberger, W. D. Smart. Layered Costmaps for Context-Sensitive Navigation. *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2014.

[152] Luis Puebla Palma. *Ultrasonic Distance Measurer. Implemented with the MC9RS08KA2*. Freescale Semiconductor, 2 2008.

[153] S. Macenski, I. Jambrecic. SLAM Toolbox: SLAM for the dynamic world. *Journal of Open Source Software*, 6(61):2783, Maj 2021.

[154] S. O. H. Madgwick, A. J. L. Harrison, R. Vaidyanathan. Estimation of IMU and MARG orientation using a gradient descent algorithm. *2011 IEEE International Conference on Rehabilitation Robotics*, strony 1–7, Zurich, Czerwiec 2011. IEEE.

[155] M. Magnusson, A. Nuchter, C. Lorken, A. Lilienthal, J. Hertzberg. Evaluation of 3D registration reliability and speed - A comparison of ICP and NDT. *2009 IEEE International Conference on Robotics and Automation*, strony 3907–3912, Kobe, Maj 2009. IEEE.

[156] M. Maimone, Y. Cheng, L. Matthies. Two years of Visual Odometry on the Mars Exploration Rovers. *Journal of Field Robotics*, 24(3):169–186, Marzec 2007.

[157] M. W. Maimone, P. C. Leger, J. J. Biesiadecki. Overview of the Mars Exploration Rovers' Autonomous Mobility and Vision Capabilities. strona 8, 2007.

[158] A. A. Makarenko, S. B. Williams, F. Bourgault, H. F. Durrant-Whyte. An experiment in integrated exploration. *In Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS*, strony 534–539, 2002.

[159] G. F. Marshall. Handbook of Optical and Laser Scanning. strona 784, 2004.

[160] G. Martinez. Intensity-Difference Based Monocular Visual Odometry for Planetary Rovers. Y. Sun, A. Behal, C.-K. R. Chung, redaktorzy, *New Development in Robot Vision*, wolumen 23, strony 181–198. Springer Berlin Heidelberg, Berlin, Heidelberg, 2015.

[161] S. Mau. What is the Kalman Filter and How can it be used for Data Fusion? *Robotics Math*, strona 10, 2005.

[162] T. R. McGuire, R. I. Potter. Anisotropic Magnetoresistance in Ferromagnetic 3d Alloys. *IEEE Transactions on magnetics*, 11(4), 7 1975.

[163] D. Meagher. Geometric Modeling Using Octree Encoding. *Computer Graphics And Image Processing*, (19):129–147, 1982.

[164] P. Mell, T. Grance. The NIST Definition of Cloud Computing. strona 7, 2011.

[165] N. Michael, S. Shen, K. Mohta, Y. Mulgaonkar, V. Kumar, K. Nagatani, Y. Okada, S. Kiribayashi, K. Otake, K. Yoshida, K. Ohno, E. Takeuchi, S. Tadokoro. Collaborative mapping of an earthquake-damaged building via ground and aerial robots. *Journal of Field Robotics*, 29(5):832–841, Wrzesień 2012.

[166] M. Milford, G. Wyeth. Persistent Navigation and Mapping using a Biologically Inspired SLAM System. *The International Journal of Robotics Research*, 29(9):1131–1153, Sierpień 2010.

[167] M. Milford, G. Wyeth, D. Prasser. RatSLAM: A hippocampal model for simultaneous localization and mapping. *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004*, strony 403–408 Vol.1, New Orleans, LA, USA, 2004. IEEE.

[168] V. Mohanty, S. Agrawal, S. Datta, A. Ghosh, V. D. Sharma, D. Chakravarty. DeepVO: A Deep Learning approach for Monocular Visual Odometry. *arXiv:1611.06069 [cs]*, Listopad 2016.

[169] T. Moore, D. Stouch. A Generalized Extended Kalman Filter Implementation for the Robot Operating System. E. Menegatti, N. Michael, K. Berns, H. Yamaguchi, redaktorzy, *Intelligent Autonomous Systems 13*, wolumen 302, strony 335–348. Springer International Publishing, Cham, 2016.

[170] H. Moravec, A. Elfes. High resolution maps from wide angle sonar. *International Conference on Robotics and Automation*, IEEE, 1985.

[171] T. Mouats, N. Aouf, A. D. Sappa, C. Aguilera, R. Toledo. Multispectral Stereo Odometry. *IEEE Transactions on Intelligent Transportation Systems*, 16(3):1210–1224, Czerwiec 2015.

[172] S. Müller, C. Weber, S. Wermter. RatSLAM on Humanoids - A Bio-Inspired SLAM Model Adapted to a Humanoid Robot. S. Wermter, C. Weber, W. Duch, T. Honkela, P. Koprinkova-Hristova, S. Magg, G. Palm, A. E. P. Villa, redaktorzy, *Artificial Neural Networks and Machine Learning – ICANN 2014*, wolumen 8681, strony 789–796. Springer International Publishing, Cham, 2014.

[173] R. Mur-Artal, J. M. M. Montiel, J. D. Tardos. ORB-SLAM: A Versatile and Accurate Monocular SLAM System. *IEEE Transactions on Robotics*, 31(5):1147–1163, Październik 2015.

[174] R. Mur-Artal, J. D. Tardos. ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo and RGB-D Cameras. *IEEE Transactions on Robotics*, 33(5):1255–1262, Październik 2017.

[175] T. Nam, J. Shim, Y. Cho. A 2.5D Map-Based Mobile Robot Localization via Cooperation of Aerial and Ground Robots. *Sensors*, 17(12):2730, Listopad 2017.

[176] NASA. Mars 2020 Mission. `https://mars.nasa.gov/mars2020/`.

[177] A. Nüchter, M. Bleier, J. Schauer, P. Janotta. Continuous-Time SLAM—Improving Google's Cartographer 3D Mapping. *Latest Developments in Reality-Based 3D Surveying and Modelling*. MDPI, Styczeń 2018.

[178] T. Oggier, B. Büttgen, F. Lustenberger, G. Becker, B. Rüegg, A. Hodac. SwissRanger SR3000 and First Experiences based on Miniaturized 3D-TOF Cameras. strona 12, 2015.

[179] A. Oliver, S. Kang, B. Wünsche, B. MacDonald. Using the kinect as a navigation sensor for mobile robotics. *Proceedings of the 27th Conference on Image and Vision Computing New Zealand*, strony 509–514, 2012.

[180] E. Olson. Real-time correlative scan matching. *2009 IEEE International Conference on Robotics and Automation*, strony 4387–4393, Kobe, Maj 2009. IEEE.

[181] I. Open Source Robotics Foundation. Turtlebot. `https://www.turtlebot.com`.

[182] I. Open Source Robotics Foundation. Turtlebot 2. `https://www.turtlebot.com/turtlebot2/`.

[183] OpenStax. Speed of Sound. Listopad 2020.

[184] E. Ostrowski, J. Kędzierski. Optyczny czujnik przemieszczenia. `http://www.konar.pwr.wroc.pl/pl/czujnik/optyczny-czujnik-przemieszczenia`. KoNaR - Koło Naukowe Robotyków.

[185] D. Pannen, M. Liebner, W. Hempel, W. Burgard. How to Keep HD Maps for Automated Driving Up To Date. *2020 IEEE International Conference on Robotics and Automation (ICRA)*, strony 2288–2294, Paris, France, Maj 2020. IEEE.

[186] P. F. Partnership. An Introduction to MEMS. Raport instytutowy, Loughborough University, 2002.

[187] P. Payeur, P. Herbert, D. Laurendeau, C. Gosselin. Probabilistic octree modeling of a 3D dynamic environment. *Proceedings of International Conference on Robotics and Automation*, 1997.

[188] PCL. Point Cloud Library. `http://pointclouds.org`.

[189] G. Peng, W. Zheng, Z. Lu, J. Liao, L. Hu, G. Zhang, D. He. An Improved AMCL Algorithm Based on Laser Scanning Match in a Complex and Unstructured Environment. *Complexity*, 2018:1–11, Grudzień 2018.

[190] A. Perzylo, B. Schießle, K. Häussermann, O. Zweigle, P. Levi, A. Knoll. Server-Sided Automatic Map Transformation in RoboEarth. P. Levi, O. Zweigle, K. Häußermann, B. Eckstein, redaktorzy, *Autonomous Mobile Systems 2012*, strony 203–216. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[191] N. Pfeifer, C. Briese. Laser scanning – principles and applications. *GeoSiberia 2007 - International Exhibition and Scientific Congress*, Novosibirsk, Russia,, 2007. European Association of Geoscientists & Engineers.

[192] X. Qu, B. Soheilian, N. Paparoditis. Landmark based localization in urban environment. *ISPRS Journal of Photogrammetry and Remote Sensing*, 140:90–103, Czerwiec 2018.

[193] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, A. Ng. ROS: An open-source Robot Operating System. strona 6, 2009.

[194] Raspberry Pi Foundation. Raspberry Pi. Dokumentacja. `http://www.raspberrypi.org`.

[195] M. Reinstein, V. Kubelka, K. Zimmermann. Terrain adaptive odometry for mobile skid-steer robots. *2013 IEEE International Conference on Robotics and Automation*, strony 4706–4711, Karlsruhe, Germany, Maj 2013. IEEE.

[196] ReMeDi. EU Project. `http://www.remedi-project.eu`.

[197] L. Riazuelo, J. Civera, J. Montiel. C2TAM: A Cloud framework for cooperative tracking and mapping. *Robotics and Autonomous Systems*, 62(4):401–413, Kwiecień 2014.

[198] T. Ringbeck, B. Hagebeuker. A 3d time of flight camera of object detection. A. Grün, redaktor, *Optical 3-D measurement techniques VIII : applications in GIS, mapping, manufacturing, quality control, robotics, navigation, mobile mapping, medical imaging, cultural heritage, VR generation and animation; papers presented to the conference organized at ETH Zurich, Switzerland, July 9 - 12, 2007. - Vol. 1*, strony 1 – 16, 2007.

[199] C. Rockey. Pakiet depthimage_to_laserscan. `http://wiki.ros.org/depthimage_to_laserscan`.

[200] D. Rodriguez-Losada, F. Matia, A. Jimenez. Local maps fusion for real time multirobot indoor simultaneous localization and mapping. *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004*, strony 1308–1313 Vol.2, New Orleans, LA, USA, 2004. IEEE.

[201] ROS. Robot Operating System. `http://www.ros.org`.

[202] E. Roszkowska. Supervisory control for multiple mobile robots in 2D space. *Proceedings of the Third International Workshop on Robot Motion and Control*, strony 187–192, 2002.

[203] E. Roszkowska. Hybrid motion control for multiple mobile robot systems. *Archives of Control Sciences*, 28:189–200, 2018.

[204] Y. Roth-Tabak, R. Jain. Building an environment model using depth information. *Computer*, 22(6):85–90, Czerwiec 1989.

[205] N. Rottmann, R. Bruder, A. Schweikard, E. Rueckert. Loop Closure Detection in Closed Environments. *arXiv:1908.04558 [cs]*, Sierpień 2019.

[206] S. Roumeliotis, G. Bekey. Distributed multirobot localization. *IEEE Transactions on Robotics and Automation*, 18(5):781–795, Październik 2002.

[207] E. Rublee, V. Rabaud, K. Konolige, G. Bradski. ORB: An efficient alternative to SIFT or SURF. *2011 International Conference on Computer Vision*, strony 2564–2571, Barcelona, Spain, Listopad 2011. IEEE.

[208] S. Rusinkiewicz, M. Levoy. Efficient variants of the ICP algorithm. *Proceedings Third International Conference on 3-D Digital Imaging and Modeling*, strony 145–152, Quebec City, Que., Canada, 2001. IEEE Comput. Soc.

[209] R. Rusu, N. Blodow, Z. Marton, M. Beetz. Aligning point cloud views using persistent feature histograms. *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, strony 3384–3391, Nice, 2008. IEEE.

[210] R. B. Rusu. Semantic 3D Object Maps for Everyday Manipulation in Human Living Environments. *KI - Künstliche Intelligenz*, 24(4):345–348, Listopad 2010.

[211] R. B. Rusu, N. Blodow, M. Beetz. Fast Point Feature Histograms (FPFH) for 3D registration. *2009 IEEE International Conference on Robotics and Automation*, strony 3212–3217, Kobe, Maj 2009. IEEE.

[212] R. B. Rusu, S. Cousins. 3D is here: Point Cloud Library (PCL). *2011 IEEE International Conference on Robotics and Automation*, strony 1–4, Shanghai, China, Maj 2011. IEEE.

[213] J. Ryde, H. Hu. 3D mapping with multi-resolution occupied voxel lists. *Autonomous Robots*, 28(2):169–185, Luty 2010.

[214] D. Rzeszotarski, P. Strumiłło, P. Pełczyński, B. Więcek, A. Lorenc. System obrazowania stereoskopowego sekwencji scen trójwymiarowych. *Elektronika : prace naukowe*, nr 10:165–184, 2005.

[215] J. Saarinen, H. Andreasson, T. Stoyanov, J. Ala-Luhtala, A. J. Lilienthal. Normal Distributions Transform Occupancy Maps: Application to large-scale online 3D mapping. *2013 IEEE International Conference on Robotics and Automation*, strony 2233–2238, Karlsruhe, Germany, Maj 2013. IEEE.

[216] S. Saeedi, M. Trentini, M. Seto, H. Li. Multiple-Robot Simultaneous Localization and Mapping: A Review: Multiple-Robot Simultaneous Localization and Mapping. *Journal of Field Robotics*, 33(1):3–46, Styczeń 2016.

[217] O. Saha, P. Dasgupta. A Comprehensive Survey of Recent Trends in Cloud Robotics Architectures and Applications. *Robotics*, 7(3):47, Sierpień 2018.

[218] S. Salti, F. Tombari, L. Di Stefano. SHOT: Unique signatures of histograms for surface and texture description. *Computer Vision and Image Understanding*, 125:251–264, Sierpień 2014.

[219] D. Scharstein, R. Szeliski. High-accuracy stereo depth maps using structured light. *2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2003. Proceedings.*, strony I–195–I–202, Madison, WI, USA, 2003. IEEE Comput. Soc.

[220] C. K. Schindhelm. Evaluating SLAM Approaches for Microsoft Kinect. *International Conference on Wireless and Mobile Communications (ICWMC)*, strona 6, 2012.

[221] J. Serafin, M. D. Cicco, T. M. Bonanni, G. Grisetti, L. Iocchi, D. Nardi, C. Stachniss, V. A. Ziparo. Robots for Exploration, Digital Preservation and Visualization of Archeological Sites. strona 8, 2016.

[222] E. A. Shaikh, A. Dhale. AGV Path Planning and Obstacle Avoidance Using Dijkstra's Algorithm. 2(6):7, 2013.

[223] S. Shalev-Shwartz, S. Shammah, A. Shashua. On a Formal Model of Safe and Scalable Self-driving Cars. *arXiv:1708.06374 [cs, stat]*, Październik 2018.

[224] A. Shkel, C. Acar, C. Painter. Two types of micromachined vibratory gyroscopes. *Sensors, 2005 IEEE*, 2005.

[225] Sick. Lidar documentation. `https://sick.com`.

[226] R. Siegwart, I. R. Nourbakhsh. *Introduction to Autonomous Mobile Robots*. Intelligent Robots and Autonomous Agents. MIT Press, Cambridge, Mass, 2004.

[227] B. Siemiątkowska, J. Szklarski, J. Syryczyński, P. Węclewski, M. Gnatowski. Wykorzystanie filtrów cząstkowych w procesie lokalizacji robota mobilnego. *Pomiary Automatyka Robotyka*, 15(171):344–353, Luty 2011.

[228] S. Slusny, R. Neruda, P. Vidnerova. Localization with a low-cost robot. *Proceedings of the Conference on Theory and Practice of Information Technologies*, strona 4, 2009.

[229] P. P. Smith. *Active Sensors for Local Planning in Mobile Robotics*. Number v. 26 serii World Scientific Series in Robotics and Intelligent Systems. World Scientific, River Edge, NJ, 2001.

[230] B. Steder, R. B. Rusu, K. Konolige, W. Burgard. NARF: 3D Range Image Features for Object Recognition. strona 2, 2011.

[231] O. Struckmeier, K. Tiwari, M. Salman, M. J. Pearson, V. Kyrki. ViTa-SLAM: A Bio-inspired Visuo-Tactile SLAM for Navigation while Interacting with Aliased Environments. *arXiv:1906.06422 [cs]*, Sierpień 2019.

[232] R. Szeliski. *Computer Vision: Algorithms and Applications*. Wrzesień 2010.

[233] Y. Takeda, N. Aoyama, T. Tanaami, S. Mizumi, H. Kamata. Study on the Indoor SLAM Using Kinect. J.-H. Kim, K. Lee, S. Tanaka, S.-H. Park, redaktorzy, *Advanced Methods, Techniques, and Applications in Modeling and Simulation*, wolumen 4, strony 217–225. Springer Japan, Tokyo, 2012.

[234] E. Takeuchi, T. Tsubouchi. A 3-D Scan Matching using Improved 3-D Normal Distributions Transform for Mobile Robotic Mapping. *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, strony 3068–3073, Beijing, China, Październik 2006. IEEE.

[235] T. Takubo, T. Kaminade, Y. Mae, K. Ohara, T. Arai. NDT scan matching method for high resolution grid map. *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, strony 1517–1522, St. Louis, MO, USA, Październik 2009. IEEE.

[236] C. Tarin Sauer, H. Brugger, E. Hofer, B. Tibken. Odometry error correction by sensor fusion for autonomous mobile robot navigation. *IMTC 2001. Proceedings of the 18th IEEE Instrumentation and Measurement Technology Conference. Rediscovering Measurement in the Age of Informatics (Cat. No.01CH 37188)*, wolumen 3, strony 1654–1658, Budapest, Hungary, 2001. IEEE.

[237] K. Tchoń, A. Mazur, I. Dulęba, R. Hossa, R. Muszyński. *Manipulatory i Roboty Mobilne: Modele, Planowanie Ruchu, Sterowanie*. Akademicka Oficyna Wydawnicza PLJ.

[238] M. Tenorth, A. Clifford Perzylo, R. Lafrenz, M. Beetz. The RoboEarth language: Representing and exchanging knowledge about actions, objects, and environments. *2012 IEEE International Conference on Robotics and Automation*, strony 1284–1289, St Paul, MN, USA, Maj 2012. IEEE.

[239] S. Thrun. Robotic Mapping: A Survey. *Exploring artificial intelligence in the new millennium*, strony 1–35, 2002.

[240] S. Thrun, W. Burgard, D. Fox. A real-time algorithm for mobile robot mapping with applications to multi-robot and 3D mapping. *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, wolumen 1, strony 321–328, San Francisco, CA, USA, 2000. IEEE.

[241] S. Thrun, W. Burgard, D. Fox. *Probabilistic Robotics*. Intelligent Robotics and Autonomous Agents. MIT Press, Cambridge, Mass, 2005.

[242] S. Thrun, D. Fox, W. Burgard, F. Dellaert. Robust monte carlo localization for mobile robots. *Artificial Intelligence*, 128(1–2):99 – 141, 2001.

[243] A. Tiderko. Multimaster FKIE. `http://wiki.ros.org/multimaster_fkie`.

[244] N. Tomatis, I. Nourbakhsh, R. Siegwart. Combining Topological and Metric: A Natural Integration for Simultaneous Localization and Map Building. strona 8, 2001.

[245] F. Tombari, S. Salti, L. Di Stefano. A combined texture-shape descriptor for enhanced 3D feature matching. *2011 18th IEEE International Conference on Image Processing*, strony 809–812, Brussels, Belgium, Wrzesień 2011. IEEE.

[246] O. Tosun, redaktor. *Mobile Robotics: Solutions and Challenges: Proceedings of the Twelfth International Conference on Climbing and Walking Robots and the Support Technologies for Mobile Machines, Istanbul, Turkey, 9-11 September 2009*. World Scientific, Singapore ; Hackensack, NJ, 2010.

[247] F. Tungadi, W. L. D. Lui, L. Kleeman, R. Jarvis. Robust online map merging system using laser scan matching and omnidirectional vision. *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, wolumen 1, strony 7–14, Taipei, Październik 2010. IEEE.

[248] D. Valiente García, L. Fernández Rojo, A. Gil Aparicio, L. Payá Castelló, O. Reinoso García. Visual Odometry through Appearance- and Feature-Based Method with Omnidirectional Images. *Journal of Robotics*, 2012:1–13, 2012.

[249] Velodyne. Lidar documentation. `https://velodynelidar.com`.

[250] E. Wan, R. Van Der Merwe. The unscented Kalman filter for nonlinear estimation. *Proceedings of the IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium (Cat. No.00EX373)*, strony 153–158, Lake Louise, Alta., Canada, 2000. IEEE.

[251] K. Wang, S. Jia, Y. Li, X. Li, B. Guo. Research on map merging for multi-robotic system based on RTM. *2012 IEEE International Conference on Information and Automation*, wolumen 1, strony 156–161, Shenyang, China, Czerwiec 2012. IEEE.

[252] S. Wang, R. Clark, H. Wen, N. Trigoni. DeepVO: Towards End-to-End Visual Odometry with Deep Recurrent Convolutional Neural Networks. *2017 IEEE International Conference on Robotics and Automation (ICRA)*, strony 2043–2050, Maj 2017.

[253] G. Welch, G. Bishop. An Introduction to the Kalman Filter. strona 16, 2006.

[254] K. M. Wurm, A. Hornung, M. Bennewitz, C. Stachniss, W. Burgard. OctoMap: A Probabilistic, Flexible, and Compact 3D Map Representation for Robotic Systems. *ICRA*, strona 8, 2010.

[255] S. Xu, W. Chou. An Improved Indoor Localization Method for Mobile Robot Based on WiFi Fingerprint and AMCL. *2017 10th International Symposium on Computational Intelligence and Design (ISCID)*, strony 324–329, Hangzhou, Grudzień 2017. IEEE.

[256] Z. Yan, N. Jouandeau, A. A. Cherif. A Survey and Analysis of Multi-Robot Coordination. *International Journal of Advanced Robotic Systems*, 10(12):399, Grudzień 2013.

[257] J. Yang, K. Xian, Y. Xiao, Z. Cao. Performance Evaluation of 3D Correspondence Grouping Algorithms. *2017 International Conference on 3D Vision (3DV)*, strony 467–476, Qingdao, Październik 2017. IEEE.

[258] M. Y. Yang, W. Förstner. Plane detection in point cloud data. *Department of Photogrammetry, University of Bonn*, 2010.

[259] C. Yi. Map Representation for Robots. *The Smart Computing Review*, Luty 2012.

[260] K. Yousif, A. Bab-Hadiashar, R. Hoseinnezhad. An Overview to Visual Odometry and Visual SLAM: Applications to Mobile Robotics. *Intelligent Industrial Systems*, 1(4):289–311, Grudzień 2015.

[261] S. Yu, C. Fu, A. K. Gostar, M. Hu. A Review on Map-Merging Methods for Typical Map Types in Multiple-Ground-Robot SLAM Solutions. *Sensors*, 20(23):6988, Grudzień 2020.

[262] Y. Yue, D. Wang, P. G. C. N. Senarathne, C. Yang. Robust submap-based probabilistic inconsistency detection for multi-robot mapping. *2017 European Conference on Mobile Robots (ECMR)*, strony 1–6, Paris, Wrzesień 2017. IEEE.

[263] Yufeng Yue, D. Wang, P. Senarathne, D. Moratuwage. A hybrid probabilistic and point set registration approach for fusion of 3D occupancy grid maps. *2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, strony 001975–001980, Budapest, Hungary, Październik 2016. IEEE.

[264] R. Zhang, F. Hoflinger, L. Reindl. Inertial Sensor Based Indoor Localization and Monitoring System for Emergency Responders. *IEEE Sensors Journal*, 13(2):838–848, Luty 2013.

[265] Y. Zhong. Intrinsic shape signatures: A shape descriptor for 3D object recognition. *2009 IEEE 12th International Conference on Computer Vision Workshops, ICCV Workshops*, strony 689–696, Kyoto, Japan, 2009. IEEE.

[266] X. Zhou, S. Roumeliotis. Multi-robot SLAM with Unknown Initial Correspondence: The Robot Rendezvous Case. *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, strony 1785–1792, Beijing, China, Październik 2006. IEEE.

[267] S. Zug, F. Penzlin, A. Dietrich, T. T. Nguyen, S. Albert. Are laser scanners replaceable by Kinect sensors in robotic applications? *2012 IEEE International Symposium on Robotic and Sensors Environments Proceedings*, strony 144–149, Magdeburg, Germany, Listopad 2012. IEEE.

[268] I. Zunaidi, N. Kato, Y. Nomura, H. Matsui. Positioning System for 4-Wheel Mobile Robot: Encoder, Gyro and Accelerometer Data Fusion with Error Model Method. strona 14, 1991.

Dodatek A

Akronimy i oznaczenia

3DHV Metoda dopasowania (ang. *3D Hough voting*)

AGV Wózek samojezdny (ang. *Automated Guided Vehicle*)

AI Sztuczna inteligencja (ang. *Artificial Intelligence*)

AMR Magnetorezystancja anizotropowa (ang. *Anisotropic magnetoresistance*)

BA Dopasowanie wiązki (ang. *Boundle Adjustment*)

BRIEF Typ deskryptora (ang. *Binary Robust Independent Elementary Feature*)

CCD Typ matrycy elementów światłoczułych (ang. *Charge Coupled Device*)

CMOS Technologia wytwarzania układów scalonych (ang. *Complementary Metal-Oxide-Semiconductor*)

CPI Liczba pomiarów na cal (ang. *Counts Per Inch*)

CV Wizja komputerowa (ang. *Computer Vision*)

CPS Kooperacyjne systemy pozycjonowania (ang. *Cooperative Positioning System*)

DWA Metoda dynamicznego okna (ang. *Dynamic Window Approach*)

DKF Dyskretny filtr Kalmana (ang. *Discrete Kalman Filter*)

EKF Rozszerzony Filtr Kalmana (ang. *Extended Kalman Filter)*

EVD Dekompozycja ze względu na wartości własne (ang. *Eigenvalue Decomposition)*

FSM Automat skończony (ang. *Finite State Machine*)

FOV Pole widzenia (ang. *Field of View*)

GALILEO Europejski system nawigacji satelitarnej (ang. *European Global Satellite Navigation System*)

GCC Metoda dopasowania (ang. *Geometry Consistency Clustering*)

GPS Globalny system pozycjonowania (ang. *Global Positioning System*)

GNSS Globalny system nawigacji satelitarnej (ang. *Global Navigation Satellite System*)

GUI Graficzny interfejs użytkownika (ang. *Graphical User Interface*)

ICP Iteracyjny najbliższy punkt (ang. *Iterative Closest Point*)

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

INS Inercyjny system nawigacji (ang. *Inertial Navigation System*)

ISS Metoda ekstrakcji punktów kluczowych (ang. *Intrinsic Shape Signatures*)

HD Wysokiej rozdzielczości (ang. *High-Definition*)

HMM Ukryte modele Markowa (ang. *Hidden Markov Models*)

HRI Interakcja człowieka z robotem (ang. *Human-robot interaction*)

KF Filtr Kalmana (ang. *Kalman Filter*)

LIDAR Optyczna technika pomiarowa (ang. *Light Detection and Ranging*)

MRS System wielorobotowy (ang. *Multi-Robot System*)

MCL Lokalizacja Monte-Carlo (ang. *Monte-Carlo Localization*)

MRSLAM SLAM wielorobotowy (ang. *Multi-robot SLAM*)

MHT Śledzenie wielu hipotez (ang. *Multi-Hypotheses Tracking*)

MHT Filtr Madgwicka (ang. *Madgwick Filter*)

NARF Metoda detekcji cech (ang. *Normal Aligned Radial Feature*)

NASA Narodowa Agencja Aeronautyki i Przestrzeni Kosmicznej (ang. *National Aeronautics and Space Act*)

NDT Transformacja rozkładów normalnych (ang. *Normal Distributions Transform*)

NN Najbliższy sąsiad (ang. *Nearest Neighbour*)

OF Przepływ optyczny (ang. *Optical Flow*)

ORB Typ deskryptora (ang. *Oriented FAST and Rotated Brief*)

PCL Biblioteka programistyczna (ang. *Point Cloud Library*)

PF Filtr cząsteczkowy (ang. *Particle Filter*)

QR Kody szybkiego dostępu (ang. *Quick Response*)

RADAR Radiowa technika pomiarowa (ang. *Radio Detection and Ranging*)

RANSAC Probabilistyczna metoda zgodności próbek (ang. *Random Sample Consensus*)

RCNN Rekurencyjne, konwolucyjne sieci neuronowe (ang. *Recurrent Convolutional Neural Networks*)

RF Częstotliwość radiowa (ang. *Radio Frequency*)

ROS Robotyczna platforma programistyczna (ang. *Robot Operating System*)

RPY Zestaw kątów Eulera (ang. *Roll Pitch Yaw*)

SAC-A Metoda dopasowania punktów (ang. *Sample Consensus Alignment*)

SFM Struktura z ruchu (ang. *Structure From Motion*)

SLAM Jednoczena lokalizacja i mapowanie (ang. *Simultaneous Localization And Mapping*)

SIFT Metoda detekcji cech (ang. *Scale-Invariant Feature Transform*)

SONAR Dźwiękowa technika pomiarowa (ang. *Sound Navigation and Ranging*)

SURF Metoda detekcji cech (ang. *Speeded Up Robust Features*)

SHOT Typ deskryptora (ang. *Unique Signatures of Histograms*)

ST Techniki spektralne (ang. *Spectral Technique*)

SVD Rozkład według wartości osobliwych (ang. *Singular Value Decomposition*)

ToF Czas przelotu (ang. *Time of Flight*)

UKF Bezśladowy Filr Kalmana (ang. *Unscented Kalman Filter*)

VO Odometria wizyjna (ang. *Visual Odometry*)

QoS Jakość usług (ang. *Quality of Service*)

Dodatek B

Aksjomaty prawdopodobieństwa

Definicja 1 *Niech Ω określa skończony zbiór zdarzeń elementarnych ω, nazywany przestrzenią zdarzeń elementarnych. Przez 2^Ω oznaczmy rodzinę podzbiorów zbioru Ω. Rodzina $\mathcal{F} \subset 2^\Omega$ jest σ-ciałem podzbiorów Ω jeżeli spełnione są warunki:*

$$\Omega \in \mathcal{F} \tag{B.1}$$

$$\forall A \subset \Omega \;\; A \in \mathcal{F} \implies A' \in \mathcal{F} \tag{B.2}$$

$$\forall A_1, A_2, \ldots \subset \Omega \;\; A_1, A_2, \ldots \in \mathcal{F} \implies \cup_{i=1}^{\infty} A_i \in \mathcal{F}. \tag{B.3}$$

Definicja 2 *Prawdopodobieństwem (miarą probabilistyczną) nazywamy funkcję*

$$p : \mathcal{F} \to \mathbb{R}, \tag{B.4}$$

o wartościach rzeczywistych, określoną na σ-ciele zdarzeń $\mathcal{F} \subset 2^\Omega$, jeżeli

$$p(\Omega) = 1, \tag{B.5}$$

$$\forall A \in \mathcal{F} \;\; p(A) \geq 0, \tag{B.6}$$

oraz spełniona jest własność przeliczalnej addytywności

$$\forall A_1, A_2, \ldots \in \mathcal{F} : A_i \cap A_j = \emptyset \;\; \forall i \neq j \implies p\left(\bigcup_{i=1}^{\infty} A_i\right) = \sum_{i=1}^{\infty} p(A_i). \tag{B.7}$$

Powyższe warunki zostały sformułowane przez A. Kołmogorowa w 1933 roku.

Definicja 3 *Przestrzenią probabilistyczną określa się trójkę*

$$(\Omega, \mathcal{F}, p), \tag{B.8}$$

określoną na σ-ciele podzbiorów zbioru zdarzeń elementarnych Σ, przy założeniu, że $\Omega \neq \emptyset$, a p jest prawdopodobieństwem określonym na $\mathcal{F}$.

Definicja 4 *Zdarzenia $A_1, A_2, \ldots, A_n \subset \Omega$ są parami niezależne, gdy*

$$\forall i, j \in [1, n], i \neq j \;\; p(A_i \cap A_j) = p(A_i) \cdot p(A_j) \tag{B.9}$$

Definicja 5 *Prawdopodobieństwem warunkowym zajścia zdarzenia A pod warunkiem zajścia zdarzenia B, określa się prawdopodobieństwo*

$$p(A \mid B) = \frac{p(A \cup B)}{p(B)}, \tag{B.10}$$

przy założeniu, że $p(B) > 0$. Jeżeli A i B są niezależne, wtedy $p(A \mid B) = p(A)$.

Definicja 6 *Niech $\{B_i \mid i \in I\} \subset \mathcal{F}$ będzie rodziną zdarzeń, które tworzą rozbicie przestrzeni Ω, czyli spełnione są warunki:*

$$\forall i, j \in I, i \neq j \;\; B_i \cap B_j = \emptyset, \tag{B.11}$$

$$\bigcup_{i \in I} B_i = \Omega, \tag{B.12}$$

$$\forall i \in I \;\; p(B_i) > 0. \tag{B.13}$$

Prawdopodobieństwo całkowite dla dowolnego zdarzenia $A \in \mathcal{F}$ określa się jako

$$p(A) = \sum_{i \in I} p(A \mid B_i) P(B_i). \tag{B.14}$$

Definicja 7 *Wzór Bayesa dla dwóch zdarzeń A i B definiuje się następująco*

$$p(A \mid B) = \frac{p(B \mid A) p(A)}{p(B)}, \tag{B.15}$$

dla $p(B) > 0$. Należy zauważyć, że wartość mianownika $p(B)$ nie zależy od zdarzenia A, dlatego też wartość $1/p(B)$ można zastąpić współczynnikiem normalizującym η

$$p(A \mid B) = \eta \, p(B \mid A) p(A). \tag{B.16}$$

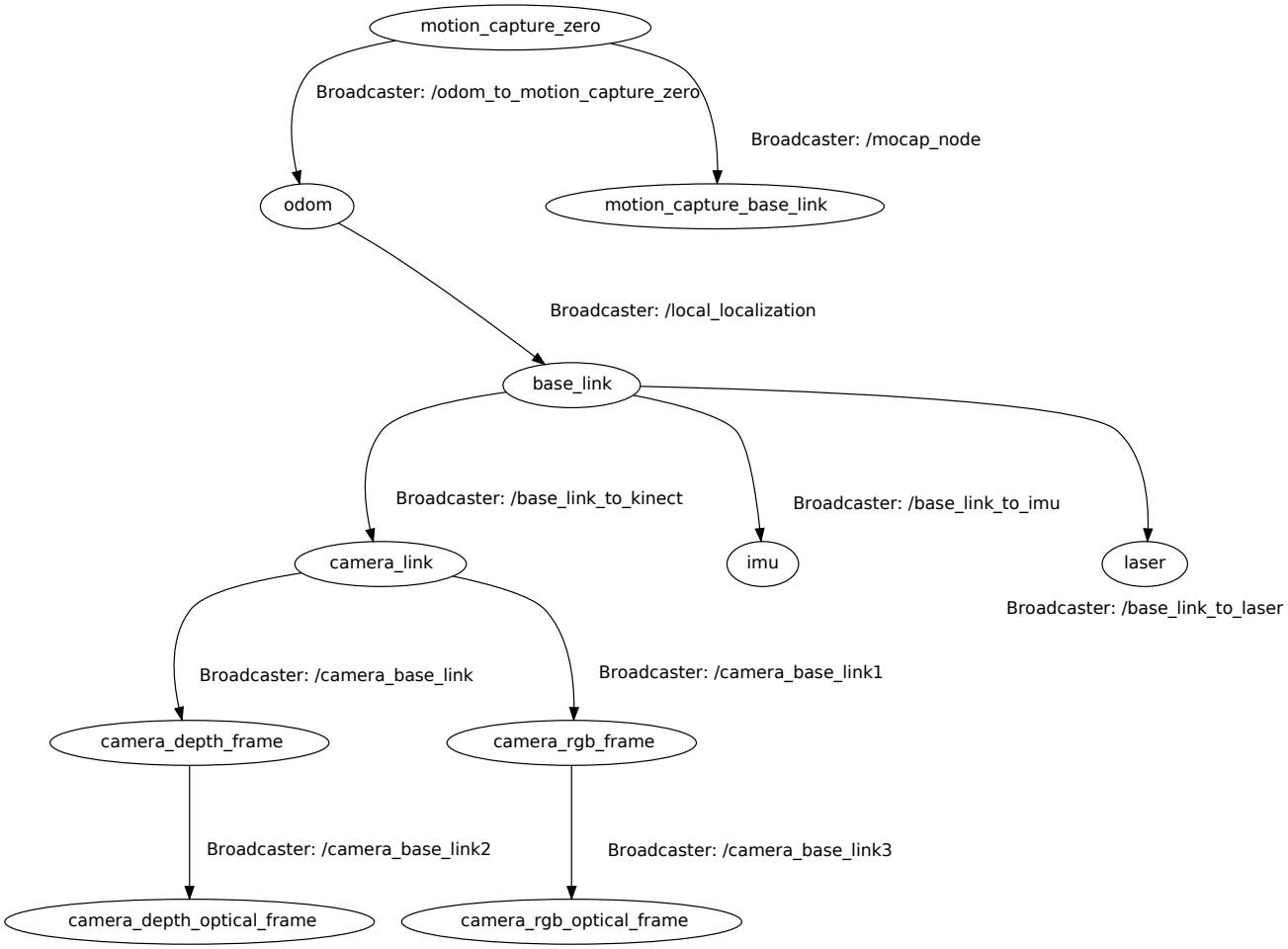
Definicja 8 *Pozostałe własności prawdopodobieństwa:*

- *prawdopodobieństwo zdarzenia niemożliwego*
$$p(\emptyset) = 0, \tag{B.17}$$
- *prawdopodobieństwo zdarzenia przeciwnego*
$$p(A') = 1 - p(A), \tag{B.18}$$
- *monotoniczność*
$$A \subset B \implies p(A) \leq p(B), \tag{B.19}$$
- *prawdopodobieństwo alternatywy zdarzeń*
$$p(A \cup B) = p(A) + p(B) - p(A \cap B), \tag{B.20}$$
-
$$p(A) = p(A \cap B) + p(A \cap B'), \tag{B.21}$$
-
$$p(A \mid B) + p(A' \mid B) = 1. \tag{B.22}$$

W pracy, stosowano również uproszczony zapis alternatywy zdarzeń, w postaci $p(A, B) = p(A \cup B)$.

Dodatek C

Transformacje układów współrzędnych związanych z robotem



Rysunek C.1 Drzewo transformacji robota mobilnego Turtlebot wraz z lokalnymi układami współrzędnych systemu śledzenia ruchu

Dodatek D

Reprezentacje orientacji obiektów

D.1 Macierz rotacji

Macierz rotacji $\mathbf{R}$ to macierz rozmiaru 3×3, należąca do specjalnej grupy obrotów $\mathbf{R} \in \mathbb{SO}(3)$ i spełniająca warunki [237]:

- ortogonalność, czyli $\mathbf{R}\mathbf{R}^{\mathbf{T}} = \mathbf{R}^{\mathbf{T}}\mathbf{R} = \mathbb{I}_3$,
- $\mathbf{R}(\mathbf{v} \times \mathbf{w}) = (\mathbf{R}\mathbf{v}) \times (\mathbf{R}\mathbf{w})$,
- $\det \mathbf{R} = 1$, zakładając prawoskrętność układu.

Macierze rotacji umożliwiają obrót obiektu względem określonej osi układu współrzędnych. Rotacja wokół każdej z osi definiowana jest przez elementarną macierz obrotu. Złożenie obrotów odpowiada mnożeniu macierzy rotacji. Elementarne macierze są następujące:

$$Rot(X,\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta) \end{bmatrix}, \tag{D.1}$$

$$Rot(Y,\theta) = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix}, \tag{D.2}$$

$$Rot(Z,\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{D.3}$$

D.2 Kąty Eulera

Kąty Eulera to minimalna reprezentacja orientacji, składająca się z trzech kątów: ϕ, θ, ψ. Kąty te określają dowolną orientację jako wykonanie kolejnych obrotów względem wybranych trzech osi układu współrzędnych. W zależności od osi wokół których obracany jest obiekt, otrzymuje się inny zestaw kątów i w rezultacie odrębną reprezentację, przy czym możliwych jest 24 kombinacji takich kątów.

Kąty Eulera można podzielić na dwie grupy: kąty klasyczne Eulera oraz kąty Tait-Bryana. Do pierwszej grupy zalicza się zestawy, w których układ obracany jest dwukrotnie względem tej samej osi, czyli:

- $x - y - x$,
- $x - z - x$,
- $y - x - y$,
- $y - z - y$,
- $z - x - z$,
- $z - y - z$.

Natomiast do grupy kątów Tait-Bryana należą zestawy składające się z obrotów względem trzech osi. Należą do niej następujące zestawy kątów:

- $x - y - z$,
- $x - z - y$,
- $y - x - z$,
- $y - z - x$,
- $z - x - y$,
- $z - y - x$.

Popularnym zestawem są kąty RPY (ang. *Roll Pitch Yaw*), które otrzymuje się przez realizację obrotów względem osi: $x - y - z$. W przypadku reprezentacji kątami Eulera pojawia się też kwestia osobliwości, określana także jako *gimbal lock*. Osobliwość ta w pewnych przypadkach jest powodem utraty jednego stopnia swobody układu.

D.3 Kwaterniony

Kwaterniony zdefiniowane jako

$$q = \begin{bmatrix} q_1 & q_2 & q_3 & q_4 \end{bmatrix}^T \in \mathbb{R}^4 = \begin{bmatrix} cos\frac{\theta}{2} & -r_x \\ sin\frac{\theta}{2} & -r_y \\ sin\frac{\theta}{2} & -r_z \\ sin\frac{\theta}{2} & \end{bmatrix} \tag{D.4}$$

są strukturą algebraiczną, która rozszerza ciało liczb zespolonych. Orientację układu współrzędnych A względem układu współrzędnych B można określić jako kąt rotacji θ układu A wokół osi utworzonej przez wektor $\hat{r} = \begin{bmatrix} r_x & r_y & r_z \end{bmatrix}$, zdefiniowany we współrzędnych układu B.

Reprezentacja orientacji za pomocą kwaternionów jest nadmiarowa, ponieważ składa się z czterech parametrów. W przypadku kwaternionów nie występują osobliwości powodujące utratę stopni swobody.

Praca częściowo finansowana z grantu Narodowego Centrum Nauki (NCN) o numerze 2016/23/B/ST7/01441.