Uniwersytet Warszawski
Wydział Matematyki, Informatyki i Mechaniki

Maciej Łaszcz
Nr albumu: 248350

Symulacja robota eksplorującego nieznane środowisko

Praca magisterska
na kierunku INFORMATYKA

Praca wykonana pod kierunkiem
dr Janiny Mincer-Daszkiewicz

Grudzień 2012

Oświadczenie kierującego pracą

Potwierdzam, że niniejsza praca została przygotowana pod moim kierunkiem i kwalifikuje się do przedstawienia jej w postępowaniu o nadanie tytułu zawodowego.

Data Podpis kierującego pracą

Oświadczenie autora pracy

Świadom odpowiedzialności prawnej oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie i nie zawiera treści uzyskanych w sposób niezgodny z obowiązującymi przepisami.

Oświadczam również, że przedstawiona praca nie była wcześniej przedmiotem procedur związanych z uzyskaniem tytułu zawodowego w wyższej uczelni.

Oświadczam ponadto, że niniejsza wersja pracy jest identyczna z załączoną wersją elektroniczną.

Data Podpis autora pracy

Streszczenie

W pracy opisano sterownik robota, którego zadaniem jest autonomiczna eksploracja i jednoczesne budowanie mapy nieznanego środowiska. Analizowane są środowiska o strukturze biura, czyli zbudowane z rozróżnialnych pomieszczeń i korytarzy, ograniczonych ścianami. Sterownik zaprojektowano z myślą o amatorskich konstrukcjach robotów i zrealizowano w środowisku symulowanym. Jego warstwowa konstrukcja pozwala na uzyskanie satysfakcjonujących wyników przy niskich wymaganiach sprzętowych, odpowiednich do amatorskich rozwiązań. Wnioski z pracy nad sterownikiem oraz powstała implementacja stanowią bazę do realizacji podobnego projektu z użyciem prawdziwych robotów.

Słowa kluczowe

robotyka, SLAM, mapowanie i lokalizacja, filtr cząsteczkowy, mapa topologiczna, diagram Voronoi, siatka zajętości

Dziedzina pracy (kody wg programu Socrates-Erasmus)

11.3 Informatyka

Klasyfikacja tematyczna

I.2.9 Robotics

Tytuł pracy w języku angielskim

Simulation of a robot exploring an unknown environment

Spis treści

1. Wprowadzenie 5
1.1. Zarys problemu 5
1.2. Cel pracy 6
1.3. Struktura pracy 6

2. Mapowanie i lokalizacja w robotyce 7
2.1. Historia 7
2.2. Pozyskiwanie informacji 7
2.2.1. Skanowanie otoczenia 8
2.2.2. Odometria 9
2.3. Przemieszczanie się 10
2.3.1. Napęd kołowy 10
2.3.2. Inne napędy 11
2.4. Reprezentacja otoczenia 11
2.4.1. Punkty orientacyjne 11
2.4.2. Macierz zajętości 12
2.4.3. Reprezentacja topologiczna 13
2.5. Podsumowanie 14

3. Opis projektu 15
3.1. Symulacja 15
3.1.1. Wybór symulatora 15
3.1.2. Model robota 16
3.1.3. Mapowane środowisko 17
3.2. Mapowanie i lokalizacja 17
3.2.1. Mapa lokalnego otoczenia 18
3.2.2. Filtr cząsteczkowy 20
3.2.3. Wnioskowanie z mapy metrycznej 26
3.2.4. Topologiczna reprezentacja mapy 31
3.3. Podsumowanie 31

4. Szczegóły implementacji 33
4.1. Realizacja projektu 33
4.1.1. Interakcja z symulatorem 33
4.1.2. Przepływ danych 33
4.1.3. Konfiguracja i uruchamianie 35
4.2. Wyniki działania 38
4.2.1. Mapowanie metryczne 38

4.2.2. Oznaczanie lokacji 39

5. Podsumowanie 43
5.1. Realizacja celu pracy 43
5.2. Możliwości rozwoju 44
5.2.1. Maper topologiczny 44
5.2.2. Wieloagentowość 44
5.3. Wnioski odnośnie realizacji sprzętowej 44

A. Zawartość płyty CD 47

B. Klatki z dołączonego do pracy filmu 49

Rozdział 1

Wprowadzenie

Robotyka to interdyscyplinarna dziedzina nauki łącząca ze sobą m.in. informatykę, mechanikę, elektronikę, automatykę, a także nauki mniej techniczne jak filozofia czy psychologia. Ta rozległość umożliwia wspólną pracę naukowców z różnych dziedzin, stymulując jej rozwój. Główną przyczyną zainteresowania robotyką jest jednak obszar jej zastosowania. Słowo *robot* pochodzi od czeskiego rzeczownika *robota* oznaczającego, podobnie jak w języku polskim, pracę fizyczną i wskazuje na podstawowy motyw pracy nad robotami: mają one wspomagać lub całkowicie wyręczać człowieka w jego zajęciach.

Roboty realizują to zadanie w różnych formach od kilkudziesięciu lat. Są one używane w pracach niemożliwych do wykonania przez ludzi podczas pożarów, klęsk lub akcji ratunkowych. Przy ich pomocy transportuje się niebezpieczne materiały lub dokonuje się napraw i konserwacji niedostępnych ludziom miejsc w elektrowniach atomowych lub w głębinach pod platformami wiertniczymi. Zadaniem robotów jest też eksploracja przestrzeni kosmicznej oraz podwodnej, ponieważ są one odporniejsze na nieprzyjazne ludziom warunki fizyczne. Ostatnim i być może najbardziej wpływającym na nasze życie zadaniem robotów jest wspomaganie procesów produkcyjnych w fabrykach, gdzie wykonują one czynności z prędkością i precyzją nieosiągalną dla ludzi.

Większość wspomnianych zadań dotyczy robotów albo niesamodzielnych, czyli sterowanych zdalnie przez operatorów, albo wysoce wyspecjalizowanych w jednej czynności i niezdolnych do swobodnego ruchu. Roboty takie zachowują się w sposób przewidywalny ze względu na małą liczbę decyzji jakie muszą podejmować. Analizowane przez nie środowisko jest ograniczone, co także zmniejsza przestrzeń dostępnych im stanów. Roboty zdolne do ruchu, które samodzielnie podejmują decyzje to, pomimo wielu lat badań, wciąż sfera bliższa futurystycznym dziełom literackim i filmowym.

1.1. Zarys problemu

Niniejsza praca dotyczy samodzielnych robotów mobilnych. Podstawowym wymaganiem stawianym przed nimi jest bezpieczna nawigacja oraz orientacja w terenie. W sytuacji, gdy robot nie posiada planu swojego otoczenia musi taki plan lub mapę przygotować samodzielnie, nie gubiąc przy tym swojej pozycji. Problem ten nazywany jest *SLAM* (od ang. *Simultaneous Localization And Mapping*) czyli jednoczesne mapowanie i lokalizacja. Wyobraźmy sobie robota, którego zadaniem jest patrolowanie obszaru biurowego. Nie ma możliwości planowania i realizacji takiego zadania bez posiadania opisu otoczenia. Robot może korzystać z różnych sensorów, takich jak czujniki podczerwieni, sonary, kamery czy kompasy, jednak odczyty z tych urządzeń zawsze obarczone są błędem. Nawet przy założeniu istnienia idealnych

sensorów, ich natura uniemożliwa jednoczesne analizowanie całego otoczenia (np. wiązka sonaru odbija się od ściany i nie daje żadnych informacji o tym, co za tą ścianą się znajduje). Z ostatniego spostrzeżenia wynika, że robot musi wykonać pomiary w różnych miejscach. Zbudowanie przestrzennego opisu otoczenia wymaga poruszania się po nim, a więc robot musi nieustannie śledzić swoją pozycję posiadając tylko częściową mapę. Jednocześnie mapa musi być aktualizowana na podstawie odczytów z sensorów oraz tej wydedukowanej pozycji, co wskazuje na rekurencję z jaką mamy do czynienia w problemie *SLAM*. Błędy w odczytach sensorów mogą się nawarstwiać i deformować zarówno mapę, jak i zapis domniemanej trasy robota, pogłębiając ten błąd w miarę podróży.

1.2. Cel pracy

Celem niniejszej pracy jest zaprojektowanie i zaimplementowanie sterownika robota, zdolnego do samodzielnej eksploracji nieznanej wcześniej przestrzeni biurowej, z wykorzystaniem istniejących algorytmów i rozwiązań problemu *SLAM*. Projekt będzie realizowany w symulatorze, ale powstałe przy pracy wnioski mają wspomóc realizację w fizycznym świecie, na prawdziwych robotach. Motywacją do podjęcia się tego projektu jest brak kompletnych i uniwersalnych rozwiązań praktycznych problemu, które można zastosować w robocie o amatorskiej konstrukcji. Cechą wyróżniającą takiej konstrukcji jest konieczność minimalizowania jej kosztu, a zatem niemożność zastosowania drogich urządzeń o wysokiej precyzji. Jednocześnie utrudnienia związane z pracą ze sprzętem elektronicznym (niemożność łatwego wprowadzania zmian) sugerują przygotowanie projektu przy użyciu symulatora.

1.3. Struktura pracy

Rozdział 2 jest wprowadzeniem do tematyki. Zarysowano w nim realia, które obowiązują w robotyce. Następuje po nim rozdział 3, będący szczegółowym opisem projektu sterownika. Przedstawiono w nim zastosowane w pracy techniki, uzasadniono ich wybór i wyjaśniono pochodzenie. Realizacja oprogramowania zgodnego z tym opisem przedstawiona jest w rozdziale 4. Zawiera on szczegóły techniczne implementacji oraz prezentuje wyniki działania sterownika i jego komponentów. Ostatni rozdział 5 to podsumowanie pracy wraz z wnioskami i rozważaniem na temat przeniesienia symulowanego projektu na fizyczne roboty.

Rozdział 2

Mapowanie i lokalizacja w robotyce

Istnieje wiele rozwiązań algorytmicznych, w dużej mierze opartych na rachunku prawdopodobieństwa, które umożliwiają wyłuskiwanie cennych informacji z obarczonych błędem odczytów sensorycznych. Zanim jednak przedstawię użyte w tej pracy techniki, przydatne będzie zapoznanie się z charakterem informacji dostarczanej przez sensory. Lektura tego rozdziału będzie pomocna podczas analizy projektu, którego założenia opierają się na realiach robotyki.

2.1. Historia

Problem *SLAM* jest z jednej strony doskonale przebadany i posiada wiele kompletnych rozwiązań (z teoretycznego punktu widzenia). Z drugiej jednak strony implementacje tych rozwiązań borykają się z podstawowymi problemami i wymagają wielu założeń ograniczających ich praktyczność. Uwidacznia to trafność tezy (zwanej paradoksem Moraveca) przedstawionej przez Hansa Moraveca w 1988 roku, która w wolnym przekładzie brzmi: stosunkowo proste jest spowodowanie, aby komputer grał w warcaby lub rozwiązywał testy na inteligencję na poziomie dorosłego człowieka, ale trudne lub nawet niemożliwe zaprogramowanie mu umiejętności jednorocznego dziecka w zakresie mobilności oraz percepcji.[1]

Zgodnie z [8] w latach '80-tych stwierdzono, że *SLAM* to jeden z istotniejszych problemów robotyki mobilnej. Wcześniejsze podejścia do problemu uniezależniały od siebie mapowanie i lokalizację zakładając, że pozycja robota jest znana ([21]) albo dana jest informacja o jego otoczeniu (np. podejście topologiczne w [18] czy *scan-matching* w [14]).

Przełomem w badaniach, który nastąpił w okolicy roku 1995, było udowodnienie, że istnieją podejścia do problemu, które przy traktowaniu lokalizacji i mapowania jako jednego złożonego problemu, dają wyniki zbieżne do stanu rzeczywistego. W [7] można odnaleźć listę prac, które istotnie wpłynęły na to odkrycie.

2.2. Pozyskiwanie informacji

Robot może posiadać wiele sensorycznych źródeł informacji. Na potrzeby tej pracy można je podzielić na dwie kategorie: informacje o otoczeniu oraz o pozycji. W pierwszej kategorii umieszczam urządzenia pozwalające na bezkontaktową analizę otoczenia robota. Są to przede wszystkim skanery laserowe oraz sonary ultradźwiękowe. Do drugiej kategorii należą techniki pomiaru przesunięcia oraz pozycji robota.

[1] „it is comparatively easy to make computers exhibit adult level performance on intelligence tests or playing checkers, and difficult or impossible to give them the skills of a one-year-old when it comes to perception and mobility." [20]

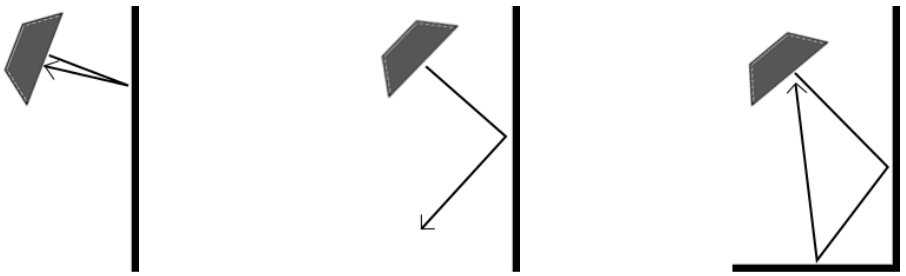
W obu przypadkach koncentruję się tylko na istotnych dla tej pracy aspektach tych urządzeń.

2.2.1. Skanowanie otoczenia

Skaner laserowy

Skaner laserowy (ang. *LIDAR – Light Detection And Ranging*) jest to mechaniczne rozszerzenie dalmierza laserowego, który pozwala na bardzo dokładny pomiar odległości do najbliższego obiektu znajdującego się na linii lasera. Dalmierze mogą być precyzyjnym zastępstwem dla taśm pomiarowych w budownictwie czy topografii, podstawowym narzędziem rozpoznania wojskowego lub źródłem danych dla obliczeń przy namierzaniu broni artyleryjskiej. Najprostsza wersja skanera laserowego jest w rzeczywistości dalmierzem laserowym umieszczonym w układzie optycznym wyposażonym w ruchome lustro, pozwalające na dokładne ustalenie kierunku promienia laserowego. W zależności od szczegółów tego układu, możliwe jest pobranie odczytów odległości od przeszkód punktowych znajdujących się na ustalonej płaszczyźnie (z reguły równoległej do podłoża) lub przeskanowanie punktów całej półpłaszczyzny. Ta trójwymiarowa wersja stosowana jest między innymi w samobieżnych pojazdach firmy *Google* testowanych obecnie na ulicach Nevady. Precyzja i szybkość działania skanerów niesie ze sobą ich wysoki koszt, w porównaniu do pozostałych urządzeń pomiarowych. W czasie pisania tej pracy cena popularnie stosowanych *LIDAR*-ów wynosi od około $500 aż do $5000, czyli więcej o przynajmniej jeden rząd wielkości od kosztu kupna pojedynczego dalmierza ultradźwiękowego dobrej jakości. *LIDAR*-y są też fizycznie większe i delikatniejsze od sonarów co utrudnia stosowanie ich w amatorskich projektach.

Sonar ultradźwiękowy



(a) Odczyt poprawny. Droga wiązki nieznacznie wydłużona.

(b) Zbyt duży kąt padania wiązki. Przeszkoda nie zostaje wykryta.

(c) Wiązka wraca po wielu odbiciach. Przeszkoda wyraźnie oddalona.

Rysunek 2.1: Symboliczne przedstawienie przekłamań wynikających z różnego sposobu odbicia się wiązki od przeszkody. Odczyty sugerują odległość od przeszkody inną niż rzeczywista.

Pojedynczy sonar (dalmierz) ultradźwiękowy to alternatywa dla dalmierza laserowego. Jego działanie jest pozbawione wysokiej precyzji informacji o odległości i azymucie przeszkody charakteryzującej skaner laserowy. Działanie takiego sonaru oparte jest na pomiarze czasu pomiędzy wysłaniem impulsu ultradźwiękowego, a powrotem jego odbicia od napotkanej na drodze przeszkody. Wiązka akustyczna nie jest spójna jak laser i rozchodzi się kuliście od źródła, tak więc skanowany przez sonar obszar kształtem przypomina stożek i dokładna informacja o pozycji obiektu, od którego nastąpiło odbicie, nie jest znana. Druga cecha, działająca na niekorzyść sonaru to jego podatność na błędy w pomiarze odległości spowodowane

niebezpośrednią drogą odbitej fali dźwiękowej (zobrazowane na rysunku 2.1) czy innymi niepożądanymi zjawiskami falowymi. Do tych ostatnich należy też interferencja, która utrudnia stosowanie wielu sonarów w jednym robocie lub współdziałania wielu robotów w obrębie jednego pomieszczenia. Krótka analiza tego problemu opisana jest w sekcji 5.3, a zastosowany w tej pracy model sonaru w sekcji 3.2.1, gdzie opisany jest także pierwszy etap przetwarzania danych.

Pojedynczy dalmierz obejmuje swym obszarem pomiaru zakres otoczenia będący w przybliżeniu wycinkiem kulistym, więc pomijając zjawiska falowe zestaw dalmierzy ustawionych w różnych kierunkach lub dalmierz umieszczony na obrotowej platformie pozwala na efektywne wykrywanie obecności najbliższych przeszkód. Dodatkowo niski koszt i możliwość własnego wykonania takich dalmierzy powoduje, że są one preferowane w rozwiązaniach amatorskich.

Kamery

Należy też wspomnieć o analizie obrazu pochodzącego z kamery. Jest to technika, która przy zastosowaniu zaawansowanych algorytmów pozwala uzyskać trójwymiarowy odczyt. Jest ona efektywniejsza z punktu widzenia kosztu urządzenia od skanera laserowego, ponieważ nie ma dużej zależności od ruchomych i wymagających precyzji elementów mechanicznych. Niestety jej zależność od warunków oświetlenia stanowi problem w środowiskach o małej rozróżnialności wizualnej elementów otoczenia. Dodatkowo przetworzenie obrazu na spójną informację wymaga dużej mocy obliczeniowej.

2.2.2. Odometria

Informacje pochodzące z sensorów zawsze umieszczone są w układzie odniesienia robota. Potrzebna jest więc wiedza o zmianie pozycji robota pomiędzy kolejnymi odczytami albo jej przybliżenie, aby móc uzależnić od siebie odczyty sensoryczne. Przykładową metodą na pozyskanie tego przybliżenia w przypadku działań wielkoskalowych jest skorzystanie z *GPS*, jednak jego precyzja ogranicza zastosowanie w robotyce (w skali rozpatrywanej w tej pracy), a w przestrzeniach zamkniętych dodatkowym utrudnieniem jest słaby sygnał satelitarny.

Najprostszym rozwiązaniem, możliwym do zastosowania w dowolnych robotach, jest nawigacja zliczeniowa (ang. *dead reckoning*), czyli wnioskowanie na temat pozycji tylko na podstawie komend wysyłanych do systemu jezdnego. Dokładność generowanego przez takie rozwiązanie przybliżenia jest zależna od kalibracji, ale także innych czynników, takich jak poziom naładowania baterii, który zmienia się podczas działania programu. Wyeliminowanie tych zmiennych można uzyskać poprzez odczytywanie stanu sprzętu znajdującego się za wykonawczym elementem systemu jezdnego (w tej pracy są to silniki kół). Realizacja tej idei to np. liczniki obrotów kół lub enkodery ich pozycji, czyli urządzenia mierzące prędkość i kierunek ich obrotu. Jest to rozwiązanie dokładniejsze, a kalibracji wymaga tylko poślizg kół, pozwalając już na zadowalająco dokładne wyliczenie przemieszczenia robota.

Oczywiście robot, który posiada więcej niż jedno urządzenie przybliżające jego pozycje będzie mniej podatny na błędy, zwłaszcza gdy urządzenia są od siebie mało zależne. Inną od enkoderów obrotu techniką, która bezpośrednio mierzy przesunięcie robota, jest analiza obrazów wykonywanych przez kamerę tak, jak jest to stosowane w myszkach optycznych, które w bardzo krótkich odstępach czasu wykonują zdjęcia niskiej rozdzielczości (rzędu 32^2 pikseli) i porównując kolejne, wyliczają względne przesunięcie. Czynnikiem wpływającym na dokładność jest tu wzór na powierzchni, nad którą przejeżdża robot oraz jej oświetlenie. Podobne rozwiązanie lecz z użyciem kamery większej rozdzielczości zastosowane jest w łazikach wysyłanych przez *NASA* na Marsa.

Nawet przy zastosowaniu obu przedstawionych tu technik przydatna jest metoda wykrywania sytuacji krytycznej jaką jest kolizja z otoczeniem. Jej pominięcie grozi bardzo dużym przekłamaniem odometrii i zagubieniem się robota. Z pomocą przychodzą czujniki zderzeniowe, które sygnalizują sytuację bezpośredniego kontaktu z przeszkodą, a rolą oprogramowania jest próba wycofania się z tej niepożądanej sytuacji.

2.3. Przemieszczanie się

W poprzedniej sekcji została opisana technologia pobierania informacji o świecie zewnętrznym. Drugim kierunkiem przepływu jest oddziaływanie na otoczenie. Zawiera się w tym zmiana pozycji przez robota, a w niektórych projektach również wpływanie na obiekty otoczenia (np. przestawanie ich). Tylko pierwszy aspekt ma znaczenie dla opisanego w tej pracy projektu, więc to na nim koncentruję się w dalszej części.

Istotną kategorię podziału pomiędzy robotami stanowi środowisko, w którym mają za zadanie pracować. Roboty mobilne dzielą się głównie na konstrukcje latające, pływające, naziemne oraz ich kombinacje. Niniejsza praca dotyczy robotów naziemnych, dlatego pozostałe rodzaje nie są w niej uwzględnione. Dalsza część tej sekcji poświęcona jest na wyjaśnienie możliwych do zastosowania w tej pracy konstrukcji napędowych.

2.3.1. Napęd kołowy

Najliczniejszą rodzinę stanowią roboty kołowe. Ich konstrukcja charakteryzuje się stosunkowo niewielką złożnością oraz dużą niezawodnością. Są popularne ze względu na swoją uniwersalność oraz fakt, że napędy kołowe towarzyszą ludzkości od setek lat, w związku z tym są znane i dopracowane przez pokolenia badaczy. Istniejące rozwiązania to konstrukcje jednokołowe, przez napędy wzorowane na samochodach, aż do mechanizmów wielokierunkowych opartych na kołach szwedzkich.

Pojazdy jedno- i dwukołowe są jednymi z mniej popularnych konstrukcji. Są one trudne w sterowaniu, ponieważ wymagają złożonych systemów aktywnego utrzymywania równowagi i nie radzą sobie z pokonywaniem przeszkód. Praca nad nimi zaowocowała wieloma ciekawymi projektami, jednak nie są one zbyt popularne w robotach o praktycznych zastosowaniach. Roboty dwukołowe można ulepszyć przez dodatkowy punkt podparcia eliminujący potrzebę aktywnego równoważenia. Robot taki przestaje być symetryczny i ruch w stronę podpory jest utrudniony, ale jeśli podpora będzie pasywnym kołem, to taki trzykołowy robot z dwoma napędzanymi kołami charakteryzuje się znacząco lepszymi właściwościami jezdnymi. Jeśli napędzane koła działają niezależnie od siebie, to osiągamy konstrukcję zwaną napędem różnicowym, która przy odpowiednim sterowaniu kołami potrafi jeździć prosto lub wykonywać przejazdy po łukach o dowolnym promieniu, w tym obracać się w miejscu. Wymaga to oczywiście precyzyjnej regulacji prędkości obrotowej kół i stwarza kłopoty konstrukcyjne.

Kolejnym możliwym rozwiązaniem jest napęd czterokołowy. Wykazuje się on większą niż poprzednie stabilnością i przyczepnością. Przy sztywnych osiach kół skręcanie jest mniej efektywne i obarczone większym tarciem. Pomocne jest zastosowanie skrętnej osi, czyli konstrukcji znanej z samochodów osobowych. Wadami takich konstrukcji są ograniczenia promienia skrętu oraz możliwość utraty przyczepności jednego z kół podczas pokonywania nierówności terenu, jeśli nie zastosuje się dostosowanego zawieszenia, które komplikuje jednak konstrukcję pojazdu.

Istnieją też napędy o większej liczbie kół, tak jak warty uwagi *rocker-bogie* stosowany w łazikach *NASA* eksplorujących powierzchnię Marsa. Jest to napęd charakteryzujący się dużą precyzją, posiadający sześć niezależnie napędzanych kół, w tym cztery skrętne, wszystkie

osadzone na dynamicznej, ale sztywnej ramie. Zadaniem tej konstrukcji jest minimalizowanie wstrząsów przy jednoczesnej możliwości pokonywania przeszkód o wysokości nawet dwóch średnic koła. Łazik *curiosity* wyposażony w ten napęd porusza się jednak z maksymalną predkością 10 cm/s przy średnicy kół wynoszącej 50 cm, a komplikacja systemu sterowania wykracza daleko poza zakres większości innych projektów.

Ostatni rodzaj napędu, który wart jest nadmienienia, to omninapęd, czyli konstrukcja oparta na specjalnych kołach umożliwiających ruch prostopadły względem ich osi. Robot o takiej konstrukcji teoretycznie może poruszać się w dowolną stronę nie wykonując przy tym obrotu. Niestety to bardzo uniwersalne rozwiązanie posiada swoje wady. Duże tarcie i zwiększona liczba stanów kół wymaga bardziej przemyślanego podejścia do odometrii niż zwykłe enkodery obrotów kół. Napęd ten wydaje się też mniej efektywny energetycznie i jest praktyczny tylko na płaskich i jednolitych podłożach [22].

2.3.2. Inne napędy

Istnieje oczywiście wiele innych rodzajów konstrukcji, takich jak zastosowane w humanoidalnych robotach kroczących, czy zainspirowanych przez owady o wielu parach odnóży. Znajdują one zastosowanie, jednak sposób ich działania i komplikacja w sposobie kontroli nie są balansowane przez żadne korzyści dla robota w środowisku biurowym.

2.4. Reprezentacja otoczenia

Format danych w jakim zapisywana jest mapa otoczenia jest czynnikiem decydującym w kwestii wyboru algorytmu aktualizującego. Często stosuje się różny sposób reprezentacji wiedzy na różnych stadiach przetwarzania. To zróżnicowanie daje możliwość dokładnego dopasowania struktury danych do przetwarzanej informacji, przy jednoczesnym zbalansowaniu wad poszczególnych formatów.

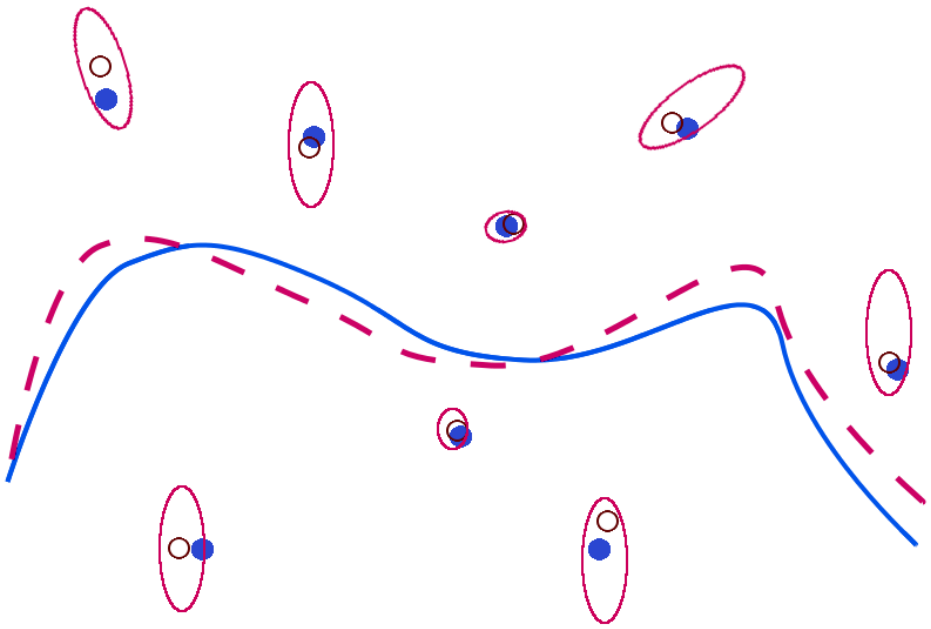
2.4.1. Punkty orientacyjne

Wiedza reprezentowana jest przez zbiór punktów orientacyjnych (ang. *landmarks*) oraz relacji przestrzennej między nimi. Punkty orientacyjne mogą być odwzorowaniem drzew w parku, skał na Marsie ale także ścian i kątów wewnątrz budynku, co trzeba wziąć pod uwagę przy projektowaniu algorytmu.

Zapis przebytej trasy oraz rozmieszczenia punktów orientacyjnych zajmuje pamięć proporcjonalną do różnorodności otoczenia, a nie od jego metrycznego rozmiaru jak to ma miejsce w macierzy zajętości, opisanej w następnej sekcji. Pozwala to na wykonywanie obliczeń na dowolnym poziomie precyzji oraz, jeśli zachodzi taka potrzeba, na swobodne rozwinięcie problemu do przestrzeni trójwymiarowej, bez drastycznego wzrostu rozmiaru struktury danych. Rysunek 2.2 przedstawia ideę mapy opartej na punktach orientacyjnych.

Ta reprezentacja nadaje się i jest stosowana z powodzeniem przy algorytmach opartych na rozszerzonych filtrach Kalmana, technikach, które pozwalają wywnioskować z sekwencji zmiennych losowych opisujących odczyty obarczone błędem przybliżenie lepsze niż same odczyty. Często do implementacji potrzebne jest obliczanie macierzy kowariancji albo Jakobianu i właśnie dzięki redukcji otoczenia do punktów, obliczenie tych kwadratowych obiektów może zużywać realną do uzyskania ilość mocy obliczeniowej.

W zamian za czytelność i wymagania pamięciowe do implementacji tej reprezentacji potrzebna jest większa "wbudowana" wiedza o charakterze mapowanego środowiska niż przy



Rysunek 2.2: Mapa pozycji punktów orientacyjnych wraz z trasą przejazdu. Linia przerywana oraz obszary zawarte w elipsach to przypuszczenia robota w kontraście do linii ciągłej i wypełnionych okręgów będących rzeczywistymi danymi.

macierzy zajętości (sekcja 2.4.2). Jest tak, ponieważ odnalezione i analizowane punkty orientacyjne muszą być niezawodnie rozpoznawalne i odróżnialne od siebie, aby filtr Kalmana mógł być zastosowany, podczas gdy fizyczne sensory najczęściej nie wykrywają punktów orientacyjnych tylko niejednoznaczne przeszkody lub obrazy otoczenia.

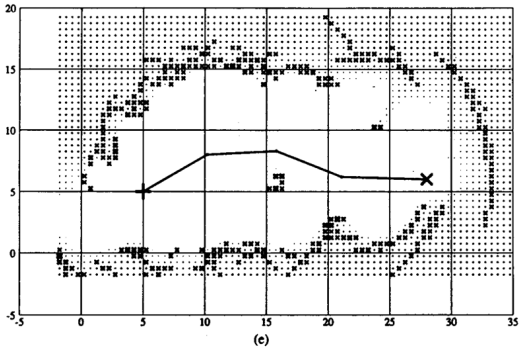
Istnieje wiele rozwiązań tego podproblemu i warto nadmienić takie, które mogłyby znaleźć zastosowanie w mapowaniu przestrzeni biurowej, czyli techniki wykrywania ścian (w pracach [6, 4, 28]) oparte na metrycznych danych oraz zaawansowany algorytm, opierający się na przetwarzaniu sygnałowym (*DSP*) odczytów z pierścienia sonarów, opisany w [9]. Wszystkie borykają się jednak z niejednoznacznością. Opracowano podejścia dopuszczające niepewność w fazie identyfikacji punktów orientacyjnych jednak powodują one istotny wzrost komplikacji i dodają istotny błąd do wynikowej reprezentacji [19].

2.4.2. Macierz zajętości

Macierze zajętości (ang. *occupancy grid*) są zupełnie innym podejściem. Ich użycie nie wymaga pośredniego kroku w analizie danych z sensorów, jakim jest wykrycie oraz identyfikacja punktów charakterystycznych. Pomijany jest w ten sposób jeden z podatnych na błędy etapów i zachowywane jest więcej informacji o otoczeniu, gdyż wszystkie odczyty wliczane są do reprezentacji, a nie tylko posiadające informację o punktach orientacyjnych.

Macierz zajętości reprezentuje otoczenie przez zbiór komórek, tak jak robi się to w grafice rastrowej. Każde z pól macierzy odpowiada niewielkiemu obszarowi otoczenia i zawiera informację o jego zajętości. Dokładność takiej mapy zależna jest od rozdzielczości macierzy, należy zatem zrównoważyć odpowiednio precyzję z rosnącym kwadratowo zużyciem pamięci i wymaganiami obliczeniowymi.

(a) Mapa zbudowana przy pomocy odczytów laserowych. [16]



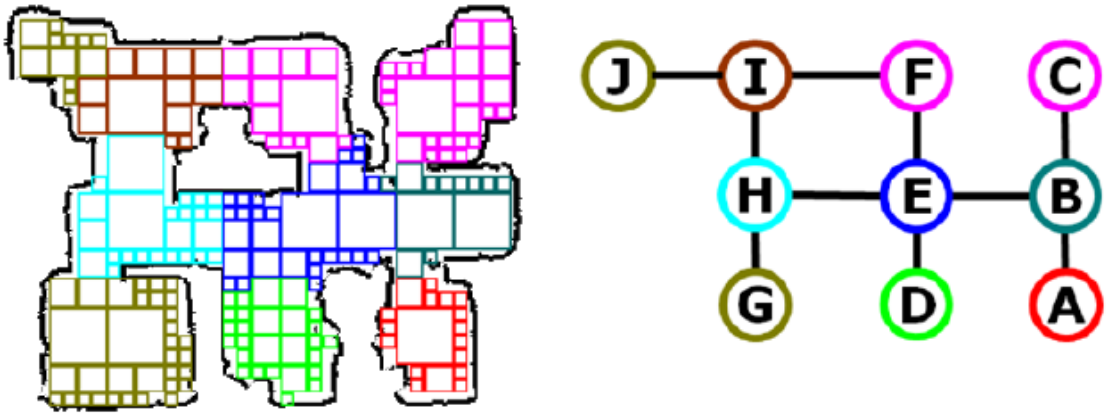
(b) Wynik przejazdu robota wyposażonego w zestaw sonarów. Czarną linią naznaczona jest jego trasa. [21]

Rysunek 2.3: Przykładowe macierze zajętości pochodzące z cytowanych pod nimi prac, zbudowane z rzeczywistych danych. Im ciemniejsza komórka tym większe jest prawdopodobieństwo, że odpowiadający jej obszar jest zajęty.

Każda z komórek takiej macierzy mogłaby być traktowana jako osobny punkt orientacyjny, co bardzo ogranicza możliwość skorzystania z algorytmów opierających wyliczenia pozycji robota na podstawie jego relacji do tych punktów [19]. Rysunek 2.3 przedstawia przykłady dwóch pochodzących z literatury macierzy zajętości przedstawiających wiedzę robota o jego środowisku.

2.4.3. Reprezentacja topologiczna

Reprezentacją najbliższą ludzkiemu odbiorowi rzeczywistości jest reprezentacja topologiczna. Otoczenie dzielone jest na logiczne składowe takie jak pomieszczenia czy korytarze, które traktowane są jako wierzchołki grafu. Taki graf jest bardzo przydatny w systemach patrolujących oraz gdy wymagana jest kolaboracja wielu robotów. Przykład takiej reprezentacji widnieje na rysunku 2.4.



Rysunek 2.4: Topologiczna reprezentacja planu budynku. Wierzchołki grafu odpowiadają odpowiednio pokolorowanym pomieszczeniom. Mapa prezentuje wyniki pracy [5].

Jest to reprezentacja bardzo wysokiego poziomu. Jej budowa wymaga wstępnego przekształcenie odczytów metrycznych podchodzących z sensorów w symbole, które są wierzchołkami grafu. Projekt w niniejszej pracy korzysta z tej reprezentacji jako ostatecznej abstrakcji eksplorowanego środowiska.

2.5. Podsumowanie

Model symulowanego robota, użytego w niniejszej pracy, wyposażony jest w napęd kołowy oraz zestaw sonarów, których użycie zostało umotywowane niskim kosztem i dostępnością. Zastosowana w sterowniku robota reprezentacja otoczenia oparta jest zarówno na reprezentacji metrycznej (czyli macierzach zajętości), jak i elementami reprezentacji topologicznej. Zaznajomienie się z tymi technikami i terminologią jest istotne dla zrozumienia następnego rozdziału zawierającego projekt zrealizowanego w tej pracy sterownika.

Rozdział 3

Opis projektu

Niniejszy rozdział poświęcony jest na opis zrealizowanego projektu. Jest to jeden z dwóch głównym rozdziałów pracy i zawiera szczegółową prezentację warunków symulacji oraz użytych w sterowniku algorytmów i struktur danych. Implementacja projektu przedstawiona jest w rozdziale 4 i do jej zrozumienia wymagane jest wcześniejsze zapoznanie się z opisaną w tym rozdziale strukturą rozwiązania oraz motywacją podjętych decyzji.

3.1. Symulacja

Specyfika pracy z fizycznymi robotami znacząco utrudnia swobodną pracę badawczą. Wiele decyzji projektowych podjętych na początku budowy robota jest nieodwracalnych i ich zmiana wymaga rozpoczynania projektu od nowa. Podobne stwierdzenie prawdziwe jest dla oprogramowania na systemy wbudowane. Algorytmy muszą być optymalizowane niskopoziomowo z myślą o sprzęcie, na którym będą uruchamiane, a to istotnie wydłuża czas tworzenia oprogramowania, zwłaszcza jeśli jego dokładna specyfikacja nie jest znana od samego początku.

Te utrudnienia sugerują skorzystanie z symulatora jako tymczasowej platformy do budowy sterownika dla robota. Rozwiązuje to oba problemy i umożliwia skoncentrowanie się na wysokopoziomowych aspektach projektu. Poprawnie działający w symulowanym świecie sterownik, może potem być podstawą do implementowania jego odpowiednika na prawdziwym robocie.

Podążając za tym rozumowaniem, niniejsze rozwiązanie jest zaprojektowane i zaimplementowane z myślą o symulowanym środowisku.

3.1.1. Wybór symulatora

Do realizacji projektu wykorzystany został symulator *Stage* [26], będący elementem projektu *Player/Stage* [11]. *Player* to środowisko mające na celu oddzielenie fizycznej, sprzętowej realizacji robota od jego oprogramowania. Zaprojektowany w tym celu interfejs umożliwia utworzenie sterownika do robota uniezależniając się od szczegółów technicznych użytego sprzętu. Sterownik (zwany klientem) można uruchomić na komputerze i przy pomocy połączenia *TCP/IP* podłączyć go do serwera uruchomionego na platformie robota. Rozwiązanie wymaga od tej platformy wsparcia dla systemu operacyjnego typu `Linux`, `Solaris` lub `*BSD`, ale w zamian umożliwia przyłączanie klienta do innych serwerów, niekoniecznie uruchomionych na fizycznych robotach.

Protokół zastosowany w projekcie *Player* zaimplementowany został w paru popularnych

symulatorach. Dwa z nich, *Stage* oraz *Gazebo*, są tego samego autorstwa co *Player*. Pierwszy z nich symuluje dwuwymiarowe środowisko i nastawiony jest na wydajną symulację oraz możliwość uruchamiania dużej liczby współdziałających robotów. Drugi powstał później i rozszerzony jest o pełen trzeci wymiar oraz implementację silnika symulującego fizykę ciała sztywnego, co umożliwia testowanie z większą dozą realizmu. Warto wspomnieć o innym symulatorze *USARSim* [1], również wspierającym protokół *Player*. Oparty jest on na silniku *Unreal Tournament* i z powodzeniem stosowany jest w jednej z konkurencji corocznego konkursu *RoboCup* [27], w którym zespoły akademickie rywalizują w budowie robotów mających za zadanie wspierać akcje ratunkowe.

Z trzech wymienionych symulatorów *Stage* wydaje się najbardziej stosowny do projektu w niniejszej pracy. Przemawia za nim jego wydajność, możliwość uruchamiania go bez środowiska graficznego oraz łatwość tworzenia opisów do symulowanych modeli i środowisk.

Symulator *Stage* symuluje wirtualny świat w sposób transparentny dla sterownika robota. Po uruchomieniu symulacji sterownik otrzymuje regularne odczyty z sensorów oraz może wydawać polecenia elementom wykonawczym, takim jak mechanizm napędowy. Szczegóły techniczne interakcji z symulatorem opisane są w sekcji 4.1.1.

3.1.2. Model robota

Użyty w projekcie model robota jest przybliżonym odwzorowaniem docelowej realizacji sprzętowej. Ze względu na dużą warstwowość, dostosowanie do innego modelu robota powinno dotyczyć tylko modułów najbliższych sensorom i silnikom.

Sensory

Robot wyposażony jest w trzy sensory, wykorzystywane bezpośrednio przy mapowaniu otoczenia. Są to dwa ruchome sonary akustyczne umieszczone na serwomechanizmach oraz czujnik zderzeniowy. Dla uproszczenia każdy z dwóch ruchomych sonarów traktowany jest jako grupa nieruchomych sonarów skierowanych w różne strony, reprezentując w ten sposób różne ustawienia serwomechanizmu jednocześnie. Takie podejście pozwala na prostszą implementację omijania przeszkód oraz traktowanie sonarów tylko jako źródła danych.

(a) bez ograniczeń (b) w pomieszczeniu

Rysunek 3.1: Rzut z góry obrazujący zakres sonarów na pierścieniu wokół robota. Zielona wiązka jest skierowana na wprost kierunku jazdy. Długość wiązek obrazuje przekazywaną przez sonar odległość od przeszkody lub, w przypadku jej braku, maksymalny zasięg sonaru. Zakresy sonarów w przedniej części nakładają się na siebie. Obrazek jest wycinkiem wizualizacji prezentowanej przez symulator *Stage*.

Rysunek 3.2: Użyta mapa fragmentu szpitala. Pochodzi ze zbioru map dołączonych do symulatora *Stage*.

Sposób przemieszczania się

Platforma robota oparta jest na dwukołowej konstrukcji z biernym punktem podparcia. Taki napęd różnicowy jest prostszy w sterowaniu od napędu samochodowego, a jego budowa powoduje mniej tarć z podłożem od omninapędu, tak więc kosztem swobody kierunku poruszania się zyskuje się wyższą jakość odczytów odometrii. Środowisko *Stage* implementuje dwie metody przemieszczania się: omninapęd oraz napęd samochodowy. Napęd różnicowy można zasymulować w modelu z omninapędem poprzez korzystanie tylko z możliwości obracania się i poruszania wzdłuż jednej osi.

3.1.3. Mapowane środowisko

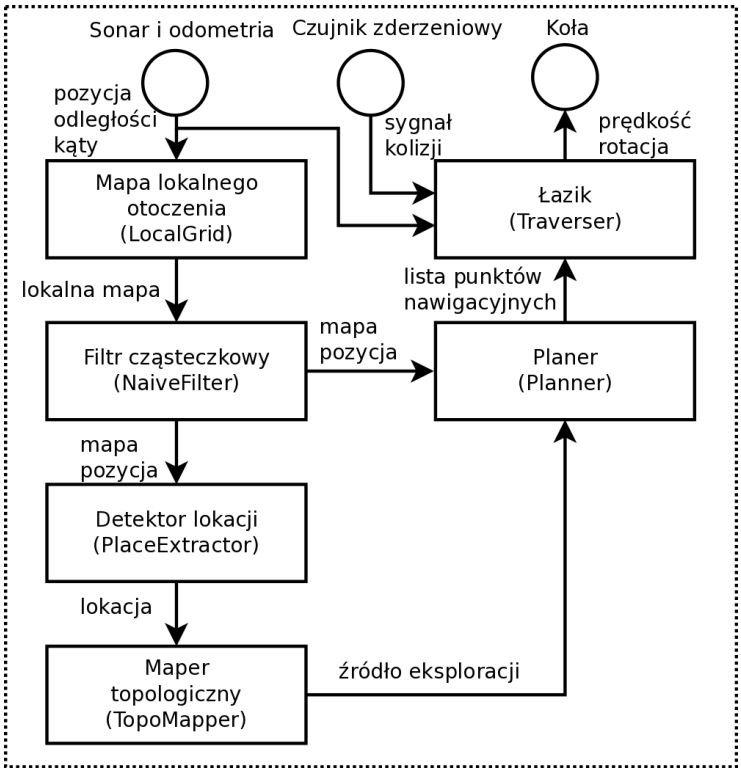
Do testowania implementacji wykorzystana została mapa reprezentująca fragment szpitala, załączona z domyślną instalacją środowiska *Stage*. Jej poziom komplikacji, różnorodność oraz rozmiar pomieszczeń są odpowiednie dla tego projektu.

3.2. Mapowanie i lokalizacja

Projekt przedstawiony w kolejnych sekcjach oparty jest na hybrydowym podejściu do problemu *SLAM*, podobnie do przedstawionego w [3]. Otoczenie odczytane przez sensory przetwarzane jest w kolejnych fazach algorytmu przedstawionych na rysunku 3.3.

Robot korzysta z trzech punktów dostępu do otoczenia oznaczonych kolistymi symbolami. Sonary, odometria oraz czujnik zderzeniowy stanowią niskopoziomowe źródła danych, a system jezdny jest ujściem sterowanym przez łazik (ang. *traverser*).

Lewa strona diagramu reprezentuje warstwy zbudowane z komponentów odpowiedzialnych za budowę kolejnych reprezentacji otoczenia. W miarę przesyłania w dół diagramu, informacje przetwarzane są i łączone w mapę o rosnącym poziomie abstrakcji. Budowa rozpoczyna się w komponencie reprezentującym lokalną mapę otoczenia. Przetwarza on grupy odczytów pochodzące z sonarów i traktując odczyty odometryczne jako informację o położeniu robota pozbawioną błędów buduje reprezentację metryczną o rozmiarze zbliżonym do zasięgu sensorów, umieszczoną w lokalnym układzie odniesienia robota. Takie kolejno budowane



Rysunek 3.3: Kolejne fazy przetwarzania danych

lokalne mapy stanowią strumień wejściowy dla filtra cząsteczkowego. Jest to komponent odpowiedzialny za aktualizowanie globalnej mapy metrycznej uwzględniającej błędy pomiarowe wszystkich odczytów. Globalna mapa metryczna jest bazą do budowy szkieletu rozpoznanego otoczenia używanego przez algorytm oznaczania pomieszczeń oraz planer trasy bezpiecznie oddalonej od wykrytych przeszkód. Szkielet ten stanowi bazę do określania i definiowania pomieszczeń, lub ogólniej lokacji, które odwiedza robot. Opisy odkrywanych na bieżąco pomieszczeń analizowane są przez ostatni komponent: maper topologiczny, którego zadaniem jest zbudować spójną i poprawną mapę topologiczną.

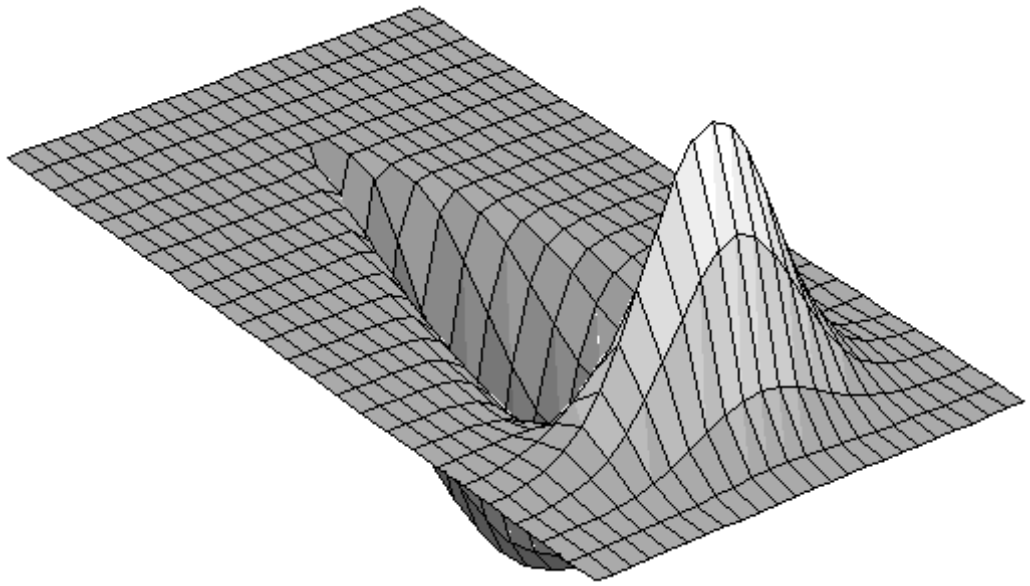
3.2.1. Mapa lokalnego otoczenia

Pierwszą fazą budowy mapy jest interpretacja zarejestrowanych przez sonary odczytów odległości od przeszkód terenowych. Każdy z sonarów raportuje w stałych odstępach czasu wykrytą odległość, tak więc analizie podlega strumień danych $S_i = (a_i, d_i)$, gdzie a_i jest azymutem (stałą pojedynczego sonaru), a d_i zarejestrowaną odległością od przeszkody lub ∞, jeśli takiej nie znaleziono.

W sekcji 2.2.1 wspomniana została istotna negatywna cecha sonarów: występuje w nich podwójna niepewność pomiarowa. Pierwsza z nich to niedokładność odczytu spowodowana zjawiskami falowymi objawiająca się niepoprawnych odczytem odległości od przeszkody. Druga to nieznajomość dokładnej pozycji obiektu, od którego nastąpiło odbicie. Problemy te adresowane są poprzez zastosowanie probabilistycznego modelu sonaru, a następnie scalanie ze sobą odczytów z niego pochodzących.

Model sonaru

Rysunek 3.4 przedstawia wizualizację zastosowanego modelu sonaru, pochodzącego z pionierskiej pracy [21]. Przy znanej odległości oraz azymucie każdy z punktów mieszczących się w zakresie pomiaru sonaru przydzielany jest do jednej z dwóch stref. Punkty o ujemnej wartości na osi pionowej odwzorowują wolny obszar, a dodatnie wartości reprezentują obszar, na którym znajduje się przeszkoda. Prawdopodobieństwo, że punkt jest wolny lub odpowiednio zajęty jest zobrazowane odległością od płaszczyzny.



Rysunek 3.4: Wizualizacja modelu sonaru użytego w projekcie. Pochodzi z [21].

Format danych

Mapa lokalnego otoczenia posiada stały rozmiar, więc wybór formatu nie jest ograniczony wymaganiami pamięciowymi. Jednocześnie jej zadaniem jest przekazanie do następnej fazy (filtr cząsteczkowy) jak najpełniejszej informacji wywnioskowanej ze strumienia danych sensorycznych.

Powyższe założenia sugerują zastosowanie macierzy przechowującej dla każdego pola mapy prawdopodobieństwo, że jest ono zajęte lub wolne. Idea rozdzielenia tych dwóch potencjalnie w pełni od siebie zależnych wartości również zaproponowana została w [21]. Takie podejście pozwala zachować maksymalnie dużą część informacji o wolnym od przeszkód terenie, a pomijając błąd w odczycie odległości, jest to informacja pewna.

Scalanie odczytów sensorów

Dzięki opisanejwłaściwości oraz użyciu tego formatu mapy możliwe jest zwiększenie dokładności. Odpowiednie scalenie wielu odczytów sensorycznych tworzy wyraźniejsze krawędzie zajętego obszaru i dobrze przygotowuje dane do następnej fazy, jaką jest filtr cząsteczkowy (sekcja 3.2.2).

Strategia łączenia wielu odczytów w jeden nazywana jest w literaturze *multiscan* [4] i proponowana jest, gdy źródło danych ma charakter rozległy (ang. *sparse*), jak użyte w projekcie sonary (pojedyncza wiązka pomiarowa bada obszar). Zastosowanie tej strategii ma na celu

oczyszczenie odczytów z szumu oraz zredukowanie liczby aktualizacji przesyłanych do filtra, którego czas działania jest o rząd wielkości większy niż czas budowy mapy lokalnej.

Multiscan można zastosować restrykcyjnie, do łączenia odczytów z różnych sonarów na pierścieniu, ale pochodzących z tego samego cyklu aktualizacji, albo swobodniej, poprzez dołączanie do lokalnej mapy wielu kolejnych cykli pomimo występującego pomiędzy nimi przesunięcia robota.

Swobodniejsze rozwiązanie jest obarczone dodatkowym błędem. Podczas budowania lokalnej mapy nie ma jeszcze możliwości korygowania informacji zwracanych przez odometrię, zatem występujące lokalnie przekłamania będą traktowane jako informacja pewna i zostaną rozpropagowane do następnego etapu algorytmu. Z tego powodu w implementacji odczyty scalane są tak długo, aż robot nie przekroczy z góry ustalonego limitu przebytej odległości oraz kąta obrotu. Wartości te, dostosowane do jakości odczytów odometrycznych, dobrane zostały eksperymentalnie i są również zależne od czasu działania filtra cząsteczkowego.

Na rysunku 3.5 zaprezentowane są scalone mapy lokalne powstałe podczas symulacji.

3.2.2. Filtr cząsteczkowy

W opisanych dotychczas elementach projektu nie podjęto jeszcze problemu lokalizacji. Wstępnie budowana mapa lokalna uwzględnia tylko szum w odczytach sonarów oraz wbudowaną w sonar niepewność pozycji przeszkody. Przed dokładnym przedstawieniem działania filtru cząsteczkowego użytego w projekcie wprowadzę potrzebne formalne pojęcia oraz rozwiązanie dla czystego problemu lokalizacji, który rozwiązywalibyśmy posiadając dokładną mapę otoczenia, a z którego ostatecznie wywodzi się wykorzystany filtr.

Lokalizacja

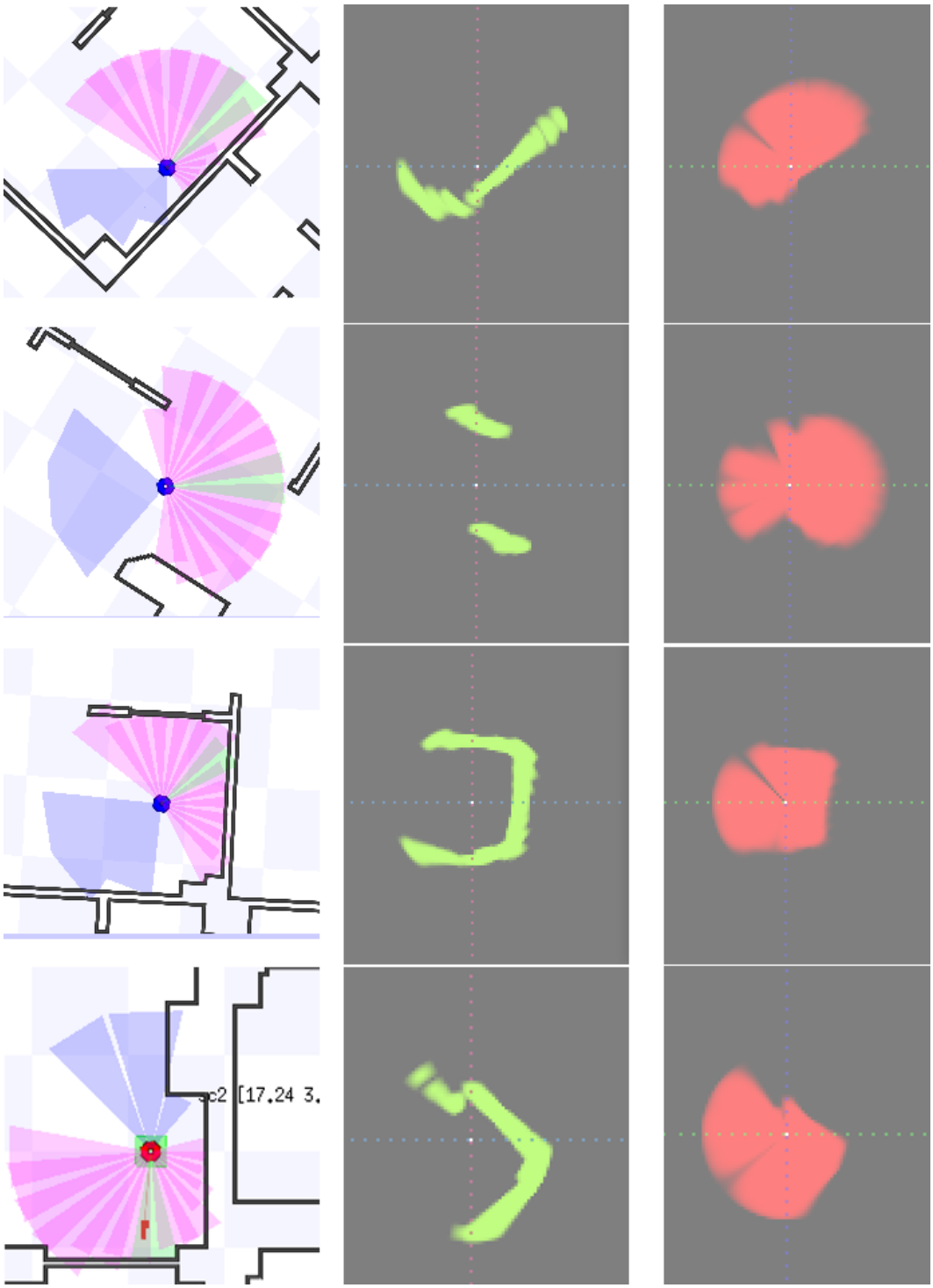
Formalizacja algorytmu użytego na tym etapie projektu wymaga wprowadzenia kilku oznaczeń (są one spójne z przedstawionymi w [23, 12]). Niech z_i oznacza dane o otoczeniu uzyskane z sensorów, w punkcie czasu i, $z_{1:i}$ sekwencję tych danych od początku działania algorytmu do czasu i. Lokalna mapa zbudowana w poprzednim etapie jest pochodną tej informacji i reprezentuje ją w filtrze. Przybliżenia pozycji pochodzące z odczytów odometrii oznaczone będą przez u_i, a rzeczywista pozycja przez x_i, oba indeksowane w taki sam sposób. Teraz, jeśli m będzie poprawną mapą otoczenia robota, to przy pewnych założeniach, prawdziwe jest poniższe równanie opisujące rozkład prawdopodobieństwa w punkcie czasu t:

$$\mathbb{P}(x_{1:t}, m \mid z_{1:t}, u_{1:t}) = \mathbb{P}(m \mid x_{1:t}, z_{1:t})\mathbb{P}(x_{1:t} \mid z_{1:t}, u_{1:t}). \tag{3.1}$$

Wspomniane założenia dotyczą niezależności kolejnych map lokalnych oraz niezależność odometrii od map. Niestety są to założenia, których nie można zapewnić przy formalnym podejściu. Jedno z nich jest nawet świadomie łamane przy scalaniu odczytów w mapie lokalnej[1]. W praktyce jednak problem ten jest pomijany, a ponieważ składowa opisująca zależność dominowana jest przez pozostałe elementy, nie powoduje to zaburzeń wyniku większych niż sama niedokładność użytych modeli.

Powróćmy do naszego obecnego celu, a mianowicie odnalezienie pozycji robota na znanej mapie na podstawie jego odczytów z_i oraz u_i. W tym celu ciąg x_i chcemy przedstawić jako indeksowaną zmienną losową, której rozkład będzie definiowany przez przedstawiony wzór.

[1] Jeśli użyjemy preferowanego rozluźnionego sposobu składania skanów pochodzących z różnych pozycji robota.

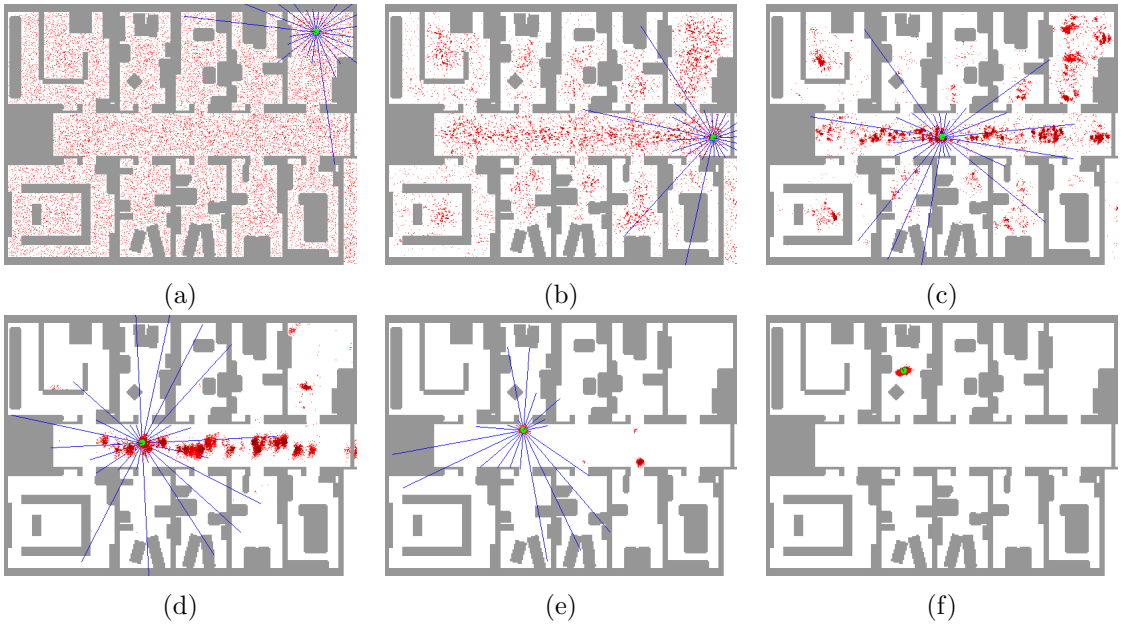


Rysunek 3.5: Przykładowe mapy lokalne zbudowane podczas eksploracji. Od lewej do prawej: otoczenie robota, składowa zajętości oraz jej przeciwność. W górnych trzech klatkach wizualizacja robota sprowadzona jest do tego samego układu współrzędnych co mapa. Dolna klatka pokazuje robota w globalnym układzie odniesienia, co powoduje niezgodność obrotu składowych mapy lokalnej z wizualizacją. Rysunki skompilowane ze zrzutów ekranu z zaimplementowanego projektu oraz symulatora *Stage*.

Docelowy algorytm tej fazy projektu opiera się na lokalizacji Monte-Carlo [10]. Jest to popularna technika pozwalająca na iteracyjne przybliżanie rozkładu zmiennej losowej opisującej pozycję przez ważony zbiór reprezentantów, zwanych cząstkami.

Na początek cząstki wybierane są losowo ze zbioru wszystkich możliwych lokalizacji zgodnie z ustalonym w warunkach początkowych rozkładem. Z reguły jest to jedna znana nam pozycja startowa lub też rozkład równomierny po wszystkich pozycjach, jeśli nie posiadamy żadnej wiedzy. Algorytm przewiduje aktualizowanie zbioru cząstek osobno w odpowiedzi na odometrię oraz odczyty sensorów. Otrzymany z odometrii odczyt u_t powoduje utworzenie nowej generacji cząstek o rozmiarze N_t w następujący sposób: z poprzedniej generacji cząstek losowo wybierana jest dokładnie jedna z szansą wyboru proporcjonalną do jej wagi, a następnie zgodnie z modelem ruchu (druga składowa wzoru 3.1, opisana w swojej części sekcji 3.2.2) losowana jest pozycja nowej cząstki, którą dodajemy do nowego zbioru z wagą $\frac{1}{N_t}$. Druga faza korzysta z informacji o otoczeniu zawartej w odczycie z_t. Jednolite wagi cząstek podlegają aktualizacji na podstawie rozkładu $\mathbb{P}(m|x_{1:t}, z_{1:t})$, czyli mnożone są przez współczynnik zależny od dopasowania odczytu do przechowywanej w cząstce pozycji, a następnie normalizowane, aby suma wszystkich wag była równa 1 lub zawierała się w rozsądnym obliczeniowo przedziale.

W miarę wykonywania kolejnych iteracji, reprezentowany przez cząstki rozkład powinien zbiegać do rzeczywistego rozkładu pozycji robota. Wyraźnie widoczne jest, że liczba cząstek ma istotne znaczenie dla dokładności filtra i jego odporności na nieregularne rozkłady. Na rysunku 3.6 zaprezentowano użycie techniki Monte-Carlo do odnalezienia pozycji, gdy w sytuacji startowej nie posiadamy żadnej wiedzy o swojej lokacji.



Rysunek 3.6: Rysunki przedstawiają kolejne kroki algorytmu lokalizacji rozpoczynanego bez żadnej wiedzy o pozycji startowej. W miarę przemieszczania się robota, rozkład prawdopodobieństwa, reprezentowany przez zbiór cząstek, zagęszcza się wokół rzeczywistej pozycji. Rysunek pochodzi z [10].

Oryginalna implementacja w pracy [10] ustala rozmiar N_t w każdej iteracji na podstawie sumy nieznormalizowanych wag, w drugim etapie aktualizacji. Ta heurystyka wymaga połączenia obu faz aktualizacji: kolejne losowane cząstki po przesunięciu ważone są zgodnie

z odczytem i proces powtarza się dopóki suma wag nie przekroczy pewnej ustalonej z góry stałej. W ten sposób dobre trafienia pochodzące z odometrii zmniejszają N_t.

Rozszerzenie lokalizacji

W opisanej technice tylko x_t traktowane jest jako niewiadoma, pozostałe obiekty są dane, a w problemie *SLAM* również mapa m jest nieznana i chcemy móc ją aproksymować. Okazuje się, że technikę Monte-Carlo można rozszerzyć tak, aby nadawała się do tego zagadnienia. Filtr Monte-Carlo poddany transformacji wykorzystującej twierdzenie Rao-Blackwella (inaczej *RBPF* od ang. *Rao-Blackwellized Particle Filter*) opisany został w [12], a dostosowany do odczytów sonarowych w [23]. Zauważmy, że jeśli m będzie dotychczasową mapą (indukcja może rozpocząć się od pustej mapy), to po każdej iteracji filtra możemy zaktualizować ją tak jak gdyby cząstki przedstawiały rzeczywistą pozycję robota i nasza zależność rekurencyjna pomiędzy m i $x_{1:t}$ zostanie domknięta. W związku z tym, że każda z cząstek opisuje inną pozycję, aproksymacja mapy nie może być globalna i musi zostać dodana do każdej cząstki z osobna, a to znacząco ogranicza rozmiar filtra z punktu widzenia wymagań pamięciowych. Drugim osłabieniem jest dodanie zależności pomiędzy cząstką a wszystkimi jej potomkami, ponieważ cząstka zawiera informację pochodną od całej trajektorii robota $x_{1:t}$, a nie tylko aktualnej pozycji x_t, jak przy lokalizacji.

Realizacja filtra

Zaimplementowanie przedstawionego filtra wymaga dokonania ustaleń w kwestii technicznej. Model ruchu wykorzystany przeze mnie w pracy opisany jest w następnej sekcji. Pozostaje jednak format przechowywanych w cząstce danych. Jako format lokalnej mapy, powstałej ze scalonych odczytów sonarów, wybrałem macierz zajętości, przechowującą dla każdego pola dwie wartości prawdopodobieństwa. Dla aktualizacji w chwili t mapa ta reprezentuje zależności pochodzące z odczytu z_t. Przyjmijmy teraz, że rozpatrujemy tę mapę przechowywaną w konkretnej cząstce filtra na ustalonej pozycji robota. Niech i będzie indeksem pola w mapie cząstki, a $\mathbb{P}(O_i)$ oraz $\mathbb{P}(E_i)$ będą oznaczały prawdopodobieństwo, że to pole jest odpowiednio zajęte oraz wolne. Zakładając, że mapa lokalna zawiera wnioski z odczytu z_t, wartości te to $\mathbb{P}(z_t|O_i)$ oraz $\mathbb{P}(z_t|E_i)$.

Aktualizowanie mapy przechowywanej w cząstce wymaga obliczenia $\mathbb{P}(O_i|z_{1:t})$ na podstawie wartości $\mathbb{P}(O_i|z_{1:t-1})$ przechowywanej w tej mapie oraz wniosków z z_t zawartych w lokalnej mapie. Zgodnie z teorią *Bayes*a wzór na aktualizację (prawdopodobieństwo a posteriori) przedstawia się następująco:

$$\mathbb{P}(O_i|z_{1:t}) = \frac{\mathbb{P}(z_t|O_i)\mathbb{P}(O_i|z_{1:t-1})}{\mathbb{P}(z_t)}$$
$$\text{oraz } \mathbb{P}(E_i|z_{1:t}) = \frac{\mathbb{P}(z_t|E_i)\mathbb{P}(E_i|z_{1:t-1})}{\mathbb{P}(z_t)}.$$

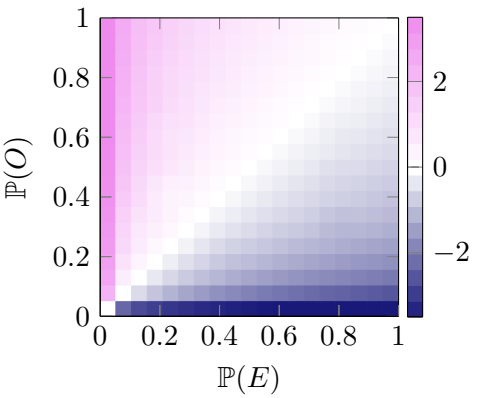
Wzory te jednak są niepraktyczne ze względu na potrzebę obliczenia trudnego do zdefiniowania $\mathbb{P}(z_t)$. Wiemy jednak, że $O_i \equiv \neg E_i$, a więc $\mathbb{P}(O_i)+\mathbb{P}(E_i) = 1$. Korzystając z tej zależności możemy obliczyć szansę (ang. *odds*) na zajście O_i:

$$\frac{\mathbb{P}(O_i|z_{1:t})}{1-\mathbb{P}(O_i|z_{1:t})} = \frac{\mathbb{P}(O_i|z_{1:t})}{\mathbb{P}(E_i|z_{1:t})} = \frac{\mathbb{P}(z_t|O_i)}{\mathbb{P}(z_t|E_i)}\frac{\mathbb{P}(O_i|z_{1:t-1})}{\mathbb{P}(E_i|z_{1:t-1})}.$$

Podczas aktualizacji posiadamy tylko jedną mapę lokalną przykładaną do wielu różnych map w cząstkach, więc jeśli skorzystamy z zapisu tej szansy jako *logit*, to dla każdego pola po

jednokrotnym obliczeniu logarytmu wykonujemy tylko operację dodania wartości w każdej z cząstek:

$$\text{logit}(\mathbb{P}(O_i|z_{1:t})) = \log \frac{z_t|O_i}{z_t|E_i} + \text{logit}(\mathbb{P}(O_i|z_{1:t-1})),$$
$$\text{gdzie } \text{logit}(p) = \log \frac{p}{1-p}.$$



Rysunek 3.7: Wartość *logit* dla nieznormalizowanych wartości mapy lokalnej

Zapis prawdopodobieństwa w postaci *logit* pozwala efektywnie ograniczyć wartości trzymane w mapach do małego przedziału (rysunek 3.7), a samo sprawdzenie znaku wystarczy do interpretacji wartości jako zajętej lub wolnej. Zaletą praktyczną jest też symetryczność, która dobrze wykorzystuje sposób działania liczb zmiennoprzecinkowych i w razie potrzeby ułatwia wygodną zamianę na postać stałoprzecinkową.

Aktualizowanie map wykonujemy po obliczeniu $\mathbb{P}(m|x_{1:t}, z_{1:t})$, czyli współczynnika dopasowania mapy lokalnej do cząstki. Jest to rozkład, którym możemy manipulować w zakresie pozwalającym na strojenie algorytmu: duże rozróżnienie pomiędzy trafieniami sonarów przyspiesza zbieżność, ale obniża podatność na błędy. Formuła na liczenie nieznormalizowanej wartości prawdopodobieństwa w zasięgnięta jest z [23] i dla mapy lokalnej l, mapy cząstki m, stałych $T_e < \frac{1}{2} < T_o$ regulujących udział niepewnych pól oraz współczynnika F ustalającego "agresywność" w rozróżnianiu jakości dopasowań jest równa:

$$w = \exp\left(\frac{1}{F}\sum_{i\in l} match(i)\right)$$
$$\text{gdzie } match(i) = \begin{cases} 1 & \mathbb{P}(O_i \mid m) > T_o \wedge \mathbb{P}(O_i \mid l) > T_o \\ -1 & \mathbb{P}(O_i \mid m) < T_e \wedge \mathbb{P}(O_i \mid l) > T_o \\ 0 & \text{wpp.} \end{cases} . \tag{3.2}$$

Każde z zajętych pól na mapie lokalnej przyrównywane jest do odpowiadającego mu pola na mapie cząstki tak, aby sumarycznie *match* równe było różnicy w liczbie pól zgadzających się i nie zgadzających się. Pola o wartości pomiędzy T_e i T_o nie są brane pod uwagę ze względu na zbyt duży poziom niepewności. Brak symetrii w traktowaniu pól wolnych oraz zajętych pozwala na nadanie tym pierwszym większego znaczenia. W efekcie, pola wolne, posiadające zajętych sąsiadów i oznaczone jako zajęte ze względu na niepewność pomiaru sonarowego, mogą wciąż zostać poprawione.

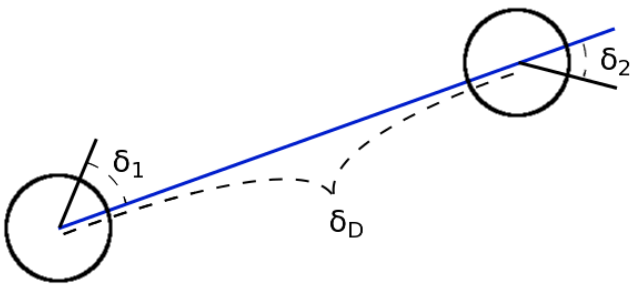
W pracy [12] prezentującej użyty tutaj filtr autorzy proponują, aby robot podczas poruszania się po otoczeniu kontrolował czy znajduje się na znanej mu części mapy, czy też eksploruje nieznane terytorium i w zależności od tego przyjmował inną strategię aktualizacji (jest tam opisana też sytuacja trzecia, gdy robot wraca do znanego miejsca inną drogą, ale w niniejszej pracy problem zamknięcia pętli rozwiązywany jest na poziomie mapowania topologicznego). W tej pracy w znamienitej większości przypadków robot porusza się po znanym obszarze lub na jego granicach. Dzieje się tak, ponieważ mapa przechowywana w cząstkach

ma charakter lokalny z punktu widzenia aktywnie rozpatrywanej lokacji i wszelka eksploracja odbywa się na granicy znanego obszaru (przeszukiwanie wszerz, a nie w głąb). W pozostałych przypadkach mamy do czynienia z zagubieniem się robota (spowodowanym np. niezarejestrowanym przez odometrię zderzeniem z przeszkodą). Jest to sytuacja awaryjna wymagająca powrotu do rozpoznanego obszaru, a wszelka aktualizacja mapy musi zostać wstrzymana. Ten scenariusz obsługiwany jest poprzez stałe monitorowanie średniej wartości wag w pochodzących ze wszystkich cząstek, której obniżenie poniżej z góry ustalonego poziomu oznacza zagubienie i uniemożliwia aktualizowanie mapy.

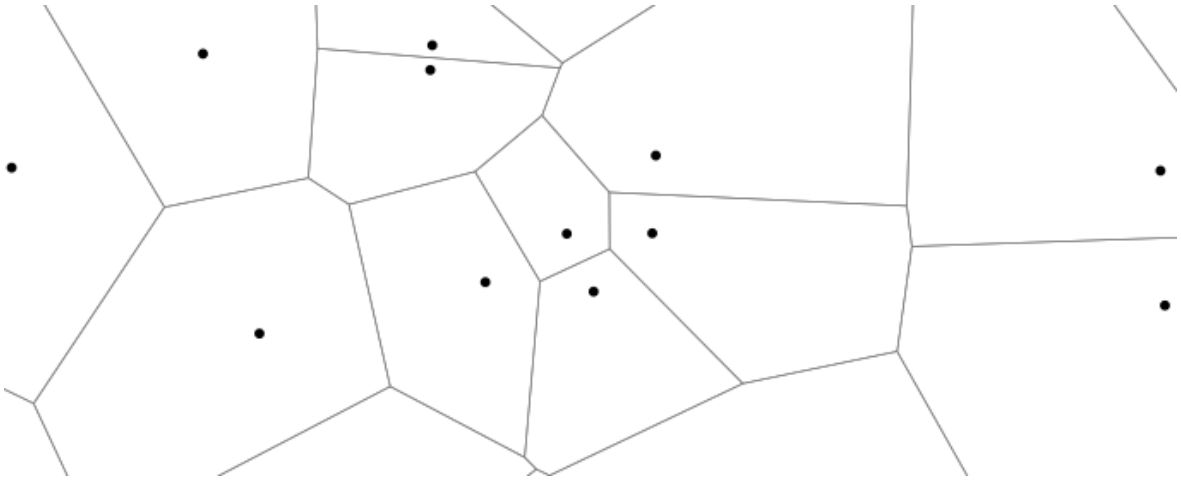
Model ruchu

Model ruchu, przy pomocy którego losowane są nowe pozycje cząstek, oznaczony przez $p(x_{1:t} \mid z_{1:t}, u_{1:t})$ korzysta z informacji zawartej w $z_{1:t}$. Intuicyjnie interpretując chodzi o poprawienie aproksymacji pochodzącej z odometrii przy pomocy pozostałych sensorów. Jedną z technik stosowanych, gdy mamy do czynienia ze skanerami laserowymi jest dopasowywanie odczytów (ang. *scan matching*). Skanery laserowe są sensorami wysokiej precyzji i gęstości, więc dwa kolejne zestawy odczytów można porównywać ze sobą i wnioskować z tego porównania przesunięcie robota. Niestety w przypadku sonarów poziom precyzji i niepewności uniemożliwia tak wiarygodny pomiar przesunięcia przy pomocy prostego porównywania odczytów, a zaawansaowane techniki są wymagające obliczeniowo. W związku z tymi utrudnieniami pomijam tę składową rozkładu i korzystam z "czystego" modelu opartego tylko na odometrii.

Celem jest zatem uzyskanie rozkładu będącego funkcją $u_{1:t}$, w praktyce korzystamy tylko z dwóch kolejnych aproksymacji pozycji u_{t-1} i u_t, gdzie pozycją nazywamy współrzędne oraz kąt odchylenia robota od osi OX. Wybrany przeze mnie model pochodzi z [13]. Podejście polega na rozdzieleniu przemieszczenia robota (w znaczeniu: przesunięcie oraz obrót) na trzy elementy, będące uproszczeniem rzeczywistej trajektorii robota. Rysunek 3.8 prezentuje ten podział. Zakładamy, że robot wykonał obrót w miejscu o kąt δ_1, przesunął się w linii prostej o długości δ_D, a następnie wykonał drugi obrót o kąt δ_2. Wszystkie te operacje obarczamy błędem. Ta strategia jest bliska rzeczywistemu sposobowi poruszania się robota, zaimplementowanego w tej pracy. W modelu brane są pod uwagę trzy rodzaje błędów i wiązane z nimi parametry rozkładu normalnego: błąd zasięgu (k_R) opisuje pomiar przebytego w linii prostej dystansu, błąd obrotu (k_θ) opisuje pomiar obrotu wykonywanego w miejscu, błąd dryfu (k_D) reprezentuje odchylenie od prostej podczas ruchu.



Rysunek 3.8: Oznaczenie symboli użytych w modelu ruchu. Robot przemieszcza się z lewej strony na prawą, a skierowanie jego przodu oznaczone jest przez czarne linie.



Rysunek 3.9: Diagram Voronoi zbudowany na zbiorze punktów. Linie podziału znajdują się maksymalnie daleko od przeszkód i umożliwiają przejazd pomiędzy każdą ich parą.

Przy tych oznaczeniach transformacja odbywa się następująco:

$$\begin{aligned}\delta_1 &\rightarrow \delta_1 + \mathcal{N}(0, k_\theta|\delta_1| + k_D|\delta_D|)\\ \delta_D &\rightarrow \delta_D + \mathcal{N}(0, k_R|\delta_D|)\\ \delta_2 &\rightarrow \delta_2 + \mathcal{N}(0, k_\theta|\delta_2| + k_D|\delta_D|)\end{aligned}$$

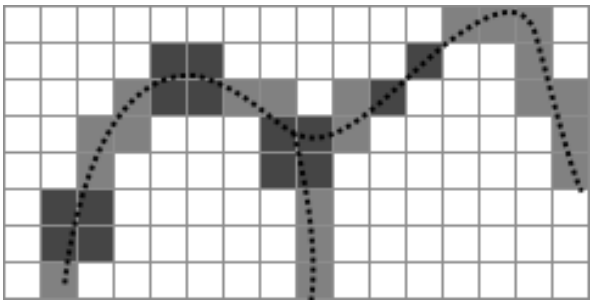
Udoskonalenie modelu ruchu, tak aby brane były pod uwagę więcej niż dwa sąsiednie odczyty odometryczne, jest silnie uzależnione od sprzętowej realizacji robota i ograniczeń ruchowych z nią związanych (ograniczenia przyspieszenia i utrata przyczepności przy skręcaniu).

3.2.3. Wnioskowanie z mapy metrycznej

Mapa metryczna ma cechy utrudniające korzystanie z niej przy topologicznej analizie otoczenia. Te same miejsca zwiedzane różną drogą lub mapowane w innym układzie odniesienia będą generowały inną macierz zajętości. Duże fluktuacje wartości pojedynczych pól przy każdej aktualizacji powodują dodatkową niestabilność przy próbie bezpośredniego wnioskowania. Na rysunku 4.3 zamieszczonym w rozdziale 4, opisującym wyniki pracy, zobrazowane są przekłamania mapy metrycznej spowodowane nawarstwionymi błędami odczytów. Uniknięcie takich przekłamań poprzez zwiększenie dokładności oraz rozmiaru filtra cząsteczkowego wymaga dużej wydajności obliczeniowej, na co nie można sobie pozwolić przy systemie wbudowanym.

Popularnym w literaturze podejściem są techniki oparte na budowie szkieletu aproksymującego diagram Voronoi względem oznaczonych przeszkód. Diagram taki w ogólnym znaczeniu jest podziałem przestrzeni euklidesowej (w tym wypadku 2-wymiarowej) na rozłączne komórki. Podział ten parametryzowany jest podzbiorem punktów tej przestrzeni S, nazywanych centrami albo zalążkami. Każda komórka przypisana jest do jednego z zalążków, tak że do zalążka $s \in S$ przypisane są wszystkie punkty przestrzeni, których najbliższym zalążkiem jest s. Obrazek 3.9 przedstawia taki podział.

Elementem tego pojęcia, które przyciąga uwagę w robotyce jest zbiór punktów leżących na granicach komórek, czyli równoodległych od dwóch najbliższych sobie zalążków. Takie punkty tworzą na płaszczyźnie planarny graf, którego wierzchołkami są punkty leżące na granicy



Rysunek 3.10: Prezentacja problemów występujących przy rastrowaniu ciągłych linii. Ciemne komórki pokazują trywialne rzutowane krzywej (przerywana linia) na siatkę. Podkreślono grupy komórek posiadające nadmiarowe elementy lub tworzące niespójności.

przynajmniej trzech komórek, a krawędzie reprezentowane są przez krzywe łączące te punkty (przynależność do konkretnego zalążka jest już na tym etapie nieistotna i zostaje pominięta). Dla dwóch sąsiadujących w tym grafie punktów, krzywa między nimi reprezentuje drogę robota, która w każdym punkcie jest lokalnie najbardziej odległa od dowolnego z zalążków. Jeśli zbiór zalążków reprezentuje niebezpieczne przeszkody, to zbudowana na bazie takiego grafu trasa jest lokalnie optymalna w sensie maksymalizacji odległości od mijanych przeszkód.

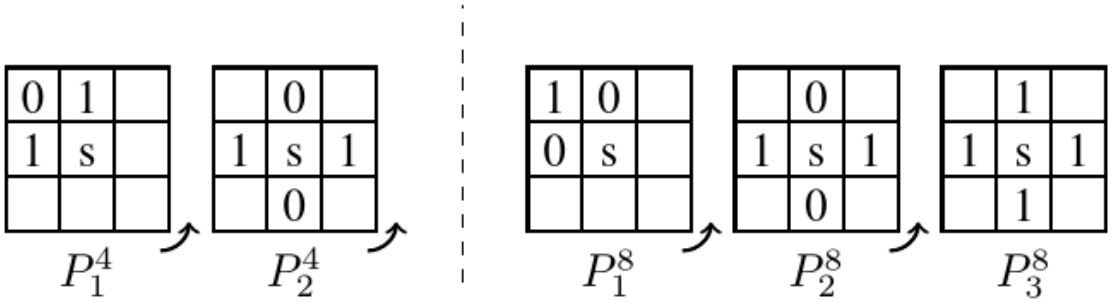
Nawet zakładając, że potrafimy łatwo generować taki diagram, jego zastosowanie "wprost" działa tylko dla przeszkód punktowych, więc nadaje się do wyznaczania trasy w drzewiastym parku lub pomiędzy innymi punktowymi przeszkodami. W przypadku przestrzeni zamkniętych, przeszkody definiowane są jako zajęte obszary ograniczone ścianami i utworzenie odpowiednika zbioru zalążków nie jest jednoznaczne. Drugim utrudnieniem przy budowie szkieletu jest wymuszona dyskretyzacja otoczenia w macierzy zajętości. Krawędzie i wierzchołki grafu zbudowanego na bazie diagramu Voronoi są krzywymi i punktami na ciągłej płaszczyźnie i przy mapowaniu ich na macierz zajętości napotykamy typowy problem *aliasing*u i artefaktów mogących tworzyć przekłamane wierzchołki lub rozspójniać graf (rysunek 3.10).

Budowa diagramu i jego aktualizowanie

Przy analizie problemu budowy szkieletu napotkałem dwie istniejące implementacje o porównywalnym (z punktu widzenia tej pracy) wyniku działania: *EVG-Thin* [29, 2] oraz *DynamicVoronoi* [17]. Po przetworzeniu macierzy zajętości określają dla każdego z pól macierzy binarną przynależność do szkieletu, przy czym generowane szkielety posiadają różne właściwości co opisane jest po przedstawieniu ich działania.

Pierwsze rozwiązanie (ang. *thinning*) przypomina w działaniu grafowy algorytm przeszukiwania wszerz inicjowany ze wszystkich pól należących do przeszkód. Na początku cała macierz oznaczana jest jako przynależąca do szkieletu. Następnie kolejne, synchronicznie oddalające się od przeszkód warstwy tworzą ruchomy front usuwający ze szkieletu napotkane pola. W miejscach, w których spotykają się ze sobą dwa fronty znajdują się punkty równoodległe od dwóch różnych przeszkód zatem są one pozostawiane w szkielecie. Algorytm kończy się, gdy cała macierz zostanie odwiedzona.

Drugie rozwiązanie zostało opublikowane jako dynamiczna alternatywa względem pierwszego i opiera się wprost na sprawdzaniu warunku przynależności do brzegów diagramu Voronoi. Algorytm podobnie do poprzednika realizuje przeszukiwanie wszerz. Dla każdego wolnego pola wyznaczane jest najbliższe zajęte pole, a jego współrzędne zapisywane. Następnie sprawdzany jest warunek, tzn. jeśli dwa sąsiadujące pola są w tej samej odległości od najbliższych sobie pól zajętych, należących do różnych przeszkód, to między nimi przebiega krzywa



Rysunek 3.11: Szablony dla odpowiednio czterech i ośmiu sąsiadów komórki. Są one przykładane w czterech możliwych obrotach, tak że komórka niepewna s znajduje się po środku. Jeśli szablon pasuje, to od s zależy spójność szkieletu i nie może zostać usunięte. Obrazek pochodzi z pracy [17]

diagramu Voronoi. Autorzy algorytmu rozszerzyli jego implementację o możliwość robienia dynamicznych aktualizacji, w większości przypadków, po zmianie wartości pojedynczych komórek macierzy (z zajętej na wolną lub przeciwnie) trzeba przetworzyć tylko lokalne otoczenie zmienionych komórek, aby zaktualizować cały diagram.

Zarówno *EVG-Thin* jak i *DynamicVoronoi* borykają się z opisanym wcześniej problemem dyskretyzacji diagramu Voronoi. Rozwiązaniem jest lokalna analiza na poziomie poszczególnych pól. We wstępnej fazie algorytmów szkielet oznaczany jest w sposób nadmiarowy i przynależą do niego wszystkie pola, sąsiadujące z krzywą diagramu. W następnej fazie (*pruning*) usuwane są wszystkie pola, od których nie zależy spójność szkieletu. Zakłada się przy tym, że pola sąsiadujące to takie, które w metryce miejskiej odległe są o dokładnie 1 (pola wewnątrz macierzy mają czterech sąsiadów). W tym celu algorytm przyrównuje ustalone wcześniej szablony rozmiaru 3×3 (ang. *connectivity patterns*) z testowanym polem po środku. Przypadki, w których pole ma mniej niż trzech sąsiadów, ale takich, którzy nie mają ze sobą wspólnego sąsiada, wymuszają pozostawienie pola w szkielecie. Sytuacje sporne to takie, w których pole ma przynajmniej trzech sąsiadów i przynajmniej dwa elementy po przekątnej. Takie układy pozostawione są do ponownego rozpatrzenia na koniec, a następnie traktowane jako do usunięcia, jeśli układ nie zmienił się. Szablony zobrazowane są na rysunku 3.11. Istnieje też możliwość ustalenia sąsiedztwa pól po przekątnych (pola wewnętrzne mają wtedy ośmiu sąsiadów), jednak w praktyce prowadzi to do tworzenia się większej liczby małych i utrudniających późniejszą obróbkę cykli, więc pozostałem przy przedstawionym wcześniej modelu czterech sąsiadów.

Rozwiązanie oparte na obliczaniu mapy odległości generuje więcej artefaktów w postaci nadmiarowych gałęzi szkieletu, ponieważ jest podatne na drobne zmiany w linii brzegowej pomiędzy terenem wolnym i tym oznaczonym jako przeszkoda. Problem ten rozwiązany jest dzięki opisanej w sekcji 3.2.3 obróbce. Za tym rozwiązaniem przemawia możliwość swobodnego regulowania parametrów ustalających warunek przynależności do szkieletu oraz możliwość dokonywania częściowych aktualizacji bez wyliczania całości szkieletu.

W definicji warunku przynależności do szkieletu padło określenie *przypisywanie pól zajętych do przeszkód*. Autorzy algorytmu proponują zastosowanie struktury danych przedstawionej w [24] do utrzymywania informacji o spójnych składowych jakie budowane są przez pola. Komplikuje to jednak rozwiązanie i w pewnych szczególnych przypadkach wymuszałoby przebudowanie szkieletu poza zakresem lokalnych zmian. Skuteczna okazuje się bardzo prosta heurystyka opierającej się tylko i wyłącznie na wycinaniu ze szkieletu pól, których najbliższe

odpowiadające im zajęte pola nie są od siebie wystarczająco odległe i w ostateczności to podejście okazuje się wystarczające.

Ostatnim niesprecyzowanym elementem jest sposób traktowania pól o niepewnej wartości. W dotychczasowie analizie zakładano, że pola są zajęte albo wolne, jednak wartości niektórych z nich nie są jednoznacznie ustalone. Pola takie występują w dwóch miejscach mapy, jedne z nich zajmują niewyeksplorowany obszar, drugie natomiast znajdują się na granicach przeszkód. Dla zachowania stabilności algorytmu pierwsza grupa traktowana jest jako pola wolne, natomiast druga jako pola zajęte. Dzięki temu budowany diagram zawiera gałęzie wchodzące w teren jeszcze nie poznany, a brzegi przeszkód są ostrzejsze.

Oznaczanie lokacji

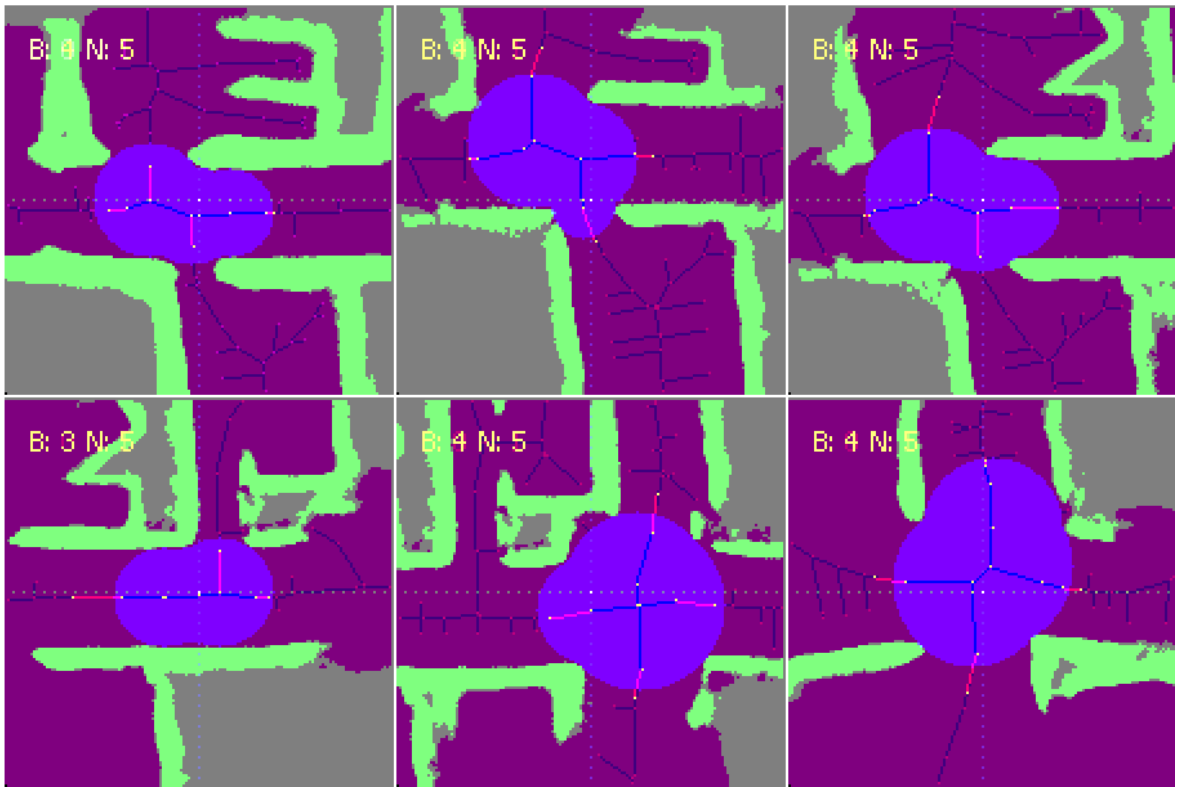
Budowa diagramu jest dopiero wstępem do zamiany formy metrycznej w topologiczną. Kolejnym etapem jest wykorzystanie tego diagramu do podzielenia eksplorowanej przestrzeni na pomieszczenia. Technika, którą zrealizowałem, bazuje na metodzie opisanej w pracy [3].

Wstępnie do lokacji przydzielane są węzły diagramu znajdujące się bliżej robota niż najbliższej sobie przeszkody (odległość od najbliższej przeszkody oraz jej pozycja zapisywane są podczas generowania diagramu). Następnie brane są pod uwagę rozwidlenia szkieletu, czyli węzły o więcej niż dwóch sąsiadach. Węzeł A należy do tej samej lokacji co węzeł B, gdy odległość pomiędzy nimi jest mniejsza niż odległość węzła A od najbliższej przeszkody. Po domknięciu symetrycznym i przechodnim, otrzymujemy relację równoważności definiującą zbiory rozwidleń. Jeśli któreś z rozwidleń znajduje się wewnątrz wstępnie wydzielonej lokacji, to cała klasa abstrakcji również do niej przynależy. Oznaczona lokacja zbudowana jest zatem ze szkieletu rozpiętego na punktach startowych oraz rozwidleniach, a jej węzły krańcowe definiują wyjścia/wejścia i są punktami połączenia między sąsiadującymi miejscami. Rysunek 3.12 przedstawia przykładowo oznaczone lokacje wraz z ich szkieletami.

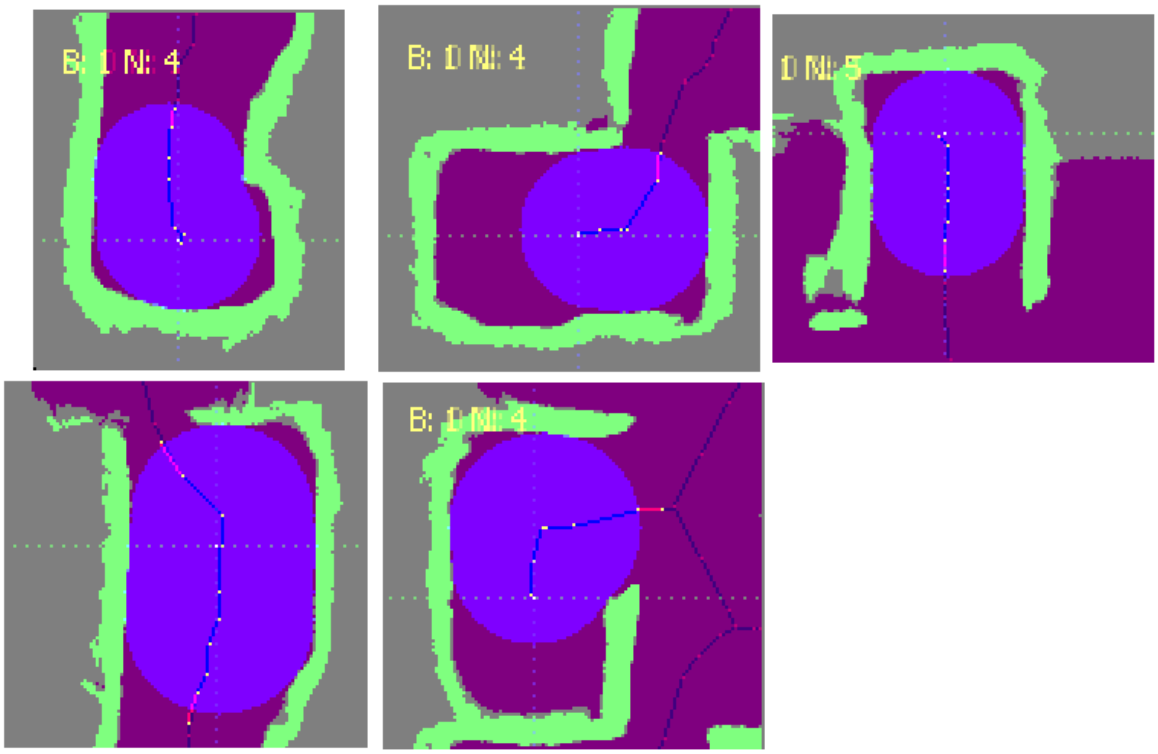
Szczególnym przypadkiem jest sytuacja, w której robot nie znajduje się w pobliżu żadnego rozwidlenia. Napotykamy ją podczas przemierzania korytarzy lub eksplorowania pomieszczeń z jednym wejściem. Rysunek 3.13 obrazuje ten przypadek. Oznaczone na nim lokacje nie zawierają rozwidleń.

W pracy [3] oznaczanie lokacji rozwinięte jest o bardziej sprecyzowane określenie ich granic. Lokacja zdefiniowana jest tam jako zbiór przejść (ang. *gateways*) połączonych ze sobą ścieżkami, a jednym ze sposobów określania tych przejść jest odnajdywanie przewężeń, przez które przebiegają gałęzie diagramu. Technika ta jest jednak niestabilna w przypadku nieostrych krawędzi z jakimi mamy do czynienia w przypadku map budowanych na podstawie sonarów i nie została zastosowana kosztem mniej naturalnie zdefiniowanych lokacji.

Opisana metoda łączenia węzłów wymaga szkieletu pozbawionego zbędnych rozwidleń. W przeciwnym wypadku możliwe jest, że zbyt wiele węzłów zostanie ze sobą połączonych. Zastosowany algorytm *DynamicThinning*, pomimo kalibracji, pozostawia gałęzie nie wnoszące nic do opisu miejsca, które narzucają zastosowanie dodatkowej fazy oczyszczającej diagram. Lokacja określana jest więc przy pomocy diagramu zbudowanego na bazie tylko bezpośredniego otoczenia robota. Globalna mapa metryczna pochodząca z filtra cząsteczkowego analizowana jest przez stałej wielkości nieruchome okno, wyśrodkowane na źródle eksploracji. Oznaczanie lokacji nie jest wykonywane, dopóki wszystkie nieznane obszary wewnątrz tego okna nie zostaną odkryte lub przyłączone do brzegu mapy. Następnie generowany jest szkielet, z którego usuwane są „martwe" gałęzie, czyli takie, które prowadzą do samotnych węzłów wewnątrz wolnego od przeszkód obszaru. Istnienie takich węzłów spowodowane jest zastosowaniem heurystyki ustalającej przynależność poszczególnych zajętych pól do przeszkód. Dzięki analizie w pełni wyeksplorowanego fragmentu mapy, gałęzie szkieletu



Rysunek 3.12: Lokacje oznaczone przez zaimplementowany algorytm. Obszary nie przypisane do miejsca zawierają szkielet nieoczyszczony z „martwych" gałęzi. Skompilowane ze zrzutów ekranu. Górne trzy obrazki reprezentują to samo miejsce, ale podczas innych faz eksploracji.



Rysunek 3.13: Oznaczone przez algorytm ślepe zaułki oraz korytarze na oczyszczonych diagramach. Obrazki wygenerowane podczas eksploracji symulowanego środowiska.

prowadzące poza bieżącą lokację stykają się z krawędzią okna i nie zostają oznaczone jako „martwe". Prezentuje to rysunek 3.14.

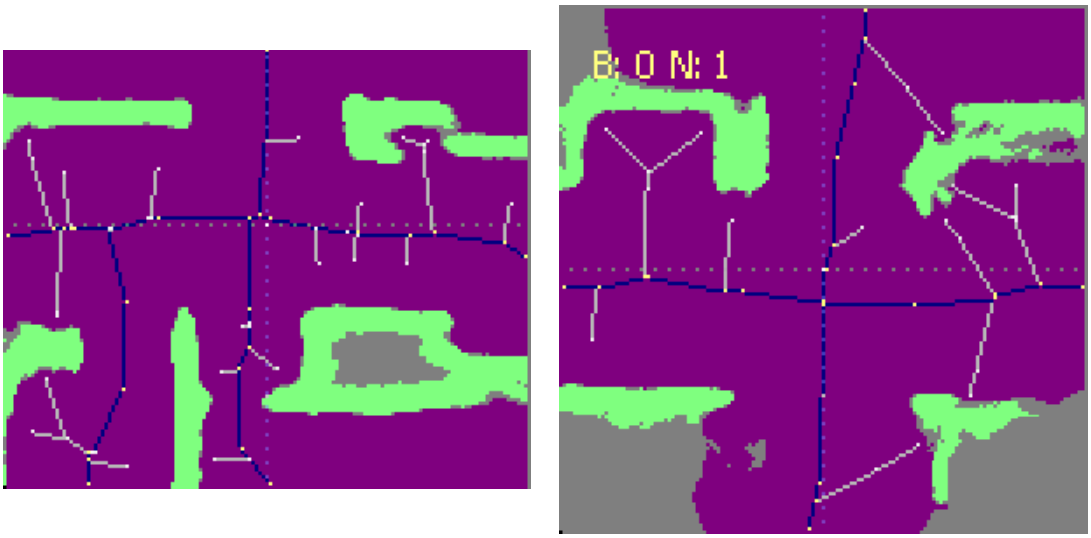
3.2.4. Topologiczna reprezentacja mapy

Ostatnim etapem przetwarzania otoczenia jest budowa mapy topologicznej na podstawie lokacji oznaczonych w poprzedniej fazie. Mapa taka jest grafem, którego wierzchołki odpowiadają odkrytym lokacjom, a krawędzie istniejącym pomiędzy nimi przejściom. Taka reprezentacja pozwala na pominięcie korytarzy posiadających tylko dwa wejścia, ponieważ są one niejawnie przedstawione przez krawędzię pomiędzy lokacjami o większej liczbie połączeń.

Problemem przy budowie takiego grafu jest niejednoznaczność w przypadku pętli (powrotu do lokacji, w której robot już się znajdował), gdyż ponownie odwiedzona lokacja może zostać zapisana jako wcześniej nienapotkana. Rozwiązanie jest prostsze na poziomie topologicznym niż metrycznym, a fakt ten jest motywacją do zastosowania takiego podejścia. Jedna z możliwości to realizacja mapera topologicznego, który przechowuje wszystkie możliwe grafy lokacji, spójne z dotychczasowymi odczytami. W miarę eksploracji powstają nowe możliwości, ale przy pewnych założeniach (np. planarności) przeglądana przestrzeń zawęża się. Szczegółowy opis rozwiązania, zapisującego mapę jako drzewo wszystkich możliwych grafów, przedstawiony jest w pracy [3], z której pochodzi też idea oznaczania lokacji.

3.3. Podsumowanie

W rozdziale przedstawiono projekt wielowarstwowego sterownika, mającego w zamierzeniach autora realizować cel postawiony w pracy. Opisane techniki oraz struktury danych są teore-



Rysunek 3.14: Diagramy wygenerowane przez algorytm *DynamicThinning*, a następnie oczyszczone z „martwych” gałęzi, które nie biorą udziału przy oznaczaniu lokacji.

tycznie wystarczające, aby bazująca na nich implementacja była efektywna i dawała satysfakcjonujące rezultaty.

Realizacja sterownika zaprezentowana jest w rozdziale 4. Opisano w nim implementację powstałą na potrzeby tej pracy oraz zawarto wyniki uruchomienia sterowanego robota w symulatorze *Stage*, co pozwala na ocenę osiągniętego rezultatu.

Rozdział 4

Szczegóły implementacji

Poprzedni rozdział poświęcony jest projektowi implementowanego w ramach niniejszej pracy sterownika. W tym rozdziale przedstawione zostaną najważniejsze aspekty tej implementacji tak, aby opisany w dodatku A kod źródłowy umożliwił jej dokładniejszą analizę oraz uruchomienie. W dalszej części przedstawiono także wyniki działania symulacji.

4.1. Realizacja projektu

Projekt zaimplementowany jest jako biblioteka współdzielona, implementująca protokół symulatora *Stage*. Językiem implementacji jest *C++*, ale zagnieżdżony jest w nim także interpreter języka *Python*, w którym napisany jest zarys mapera topologicznego. Projekt budowany jest przy pomocy programu *cmake*, a podczas pracy nad nim używany był kompilator *clang*, będący częścią platformy *LLVM*. Projekt powinien być także kompatybilny z *gcc*.

4.1.1. Interakcja z symulatorem

Początkowo implementacja realizowana była jako niezależny program linkowany z biblioteką *libplayerc++*, która pozwala na łączenie się z serwerem przez protokół *Player*. Takie rozwiązanie pozwala na podłączanie programu do działającego symulatora i odłączanie go w dowolnym momencie oraz uruchamianie programu klienckiego na innym komputerze. Asynchroniczny charakter połączenia prowadzi jednak do niestabilnej i niedeterministycznej pracy, ponieważ zbyt długi czas poświęcony na obliczenia powoduje przerwanie symulacji lub pominięcie części odczytów sensorycznych. Problem ten można rozwiązać przez wielowątkową, również asynchroniczną, implementację klienta, ale prowadzi to do niepotrzebnej komplikacji niezwiązanej z celem samego projektu. Zamierzeniem początkowym było wykorzystanie symulatora *Stage*, a umożliwia on również inny sposób sterowania modelami robotów: poprzez dynamicznie ładowaną bibliotekę. Funkcje przez nią udostępnianie wywoływane są synchronicznie i podczas ich działania symulowany świat jest wstrzymany. Eliminuje to problem niestabilności połączenia i dodatkowo umożliwia interaktywne zachowanie uruchomionego sterownika.

4.1.2. Przepływ danych

Moduły napisane na potrzebę projektu odwzorowują opisane w poprzednim rozdziale komponenty, a ich nazwy widoczne są na diagramie przedstawionym na początku rozdziału (rysunek 3.3).

Dostęp do sterownika odbywa się poprzez moduł `init`. Udostępnia on funkcję o tej samej nazwie, która wywoływana jest przez symulator jednokrotnie przed rozpoczęciem symula-

cji i odpowiedzialna jest za stworzenie struktur danych oraz rejestrację wywołań zwrotnych (ang. *callbacks*), przy pomocy których zgłaszane są aktualizacje odczytów sensorycznych. Moduł `init` oraz klasa `ModelRangerWrapper`, opakowująca model sonaru udostępniany przez *Stage*, to jedyne elementy projektu zależne od symulatora. Następuje w nich zmiana specyficznych jednostek pomiarowych oraz struktur danych na ich odpowiedniki używane wewnątrz sterownika.

Aktualizacja reprezentacji otoczenia odbywa się synchronicznie poprzez wywoływanie metody aktualizującej w obiekcie klasy `Robot`. Obiekt ten zawiera instancje klas odpowiedzialnych za mapowanie metryczne (`Mapper`), mapowanie topologiczne (`TopoMapper`), planowanie trasy (`Planner`) oraz jej realizację (`Traverser`). Poniżej opisane jest ich działanie oraz zadania, które realizują.

Klasa `Mapper`

Obiekty tej klasy zawierają mapę lokalnego otoczenia (klasa `LocalGrid`) wraz z implementacją filtra cząsteczkowego (klasa `NaiveFilter`). Maper aktualizuje filtr reprezentujący mapę lokalną, z częstotliwością zależną od przebytej drogi oraz liczby zebranych odczytów sensorycznych, sprawując w ten sposób kontrolę nad scalaniem odczytów (sekcja 3.2.1). Mapa lokalna jest czyszczona każdorazowo po dołączeniu jej do mapy reprezentowanej przez filtr.

Element realizowany w `LocalGrid` jest implementacją mapowania ze znaną pozycją. Przechowuje on dwie niezależne płaszczyzny mapy opisane w sekcji 3.2.1. Ich aktualizacja na podstawie odczytów sonarów (opakowanych w obiekty klasy `SonarScan`) korzysta z modelu sonaru zaimplementowanego w klasie `SonarModel`, której obiekty są budowane na podstawie modelu symulowanego robota podczas inicjalizacji sterownika.

Tworzona przez obiekt klasy `Mapper` mapa metryczna oraz aktualna pozycja robota, pochodzące z cząstki filtra o największej wadze, udostępnianie są detektorowi lokacji oraz planerowi.

Klasy `Planner` oraz `Traverser`

Te dwie klasy odpowiedzialne są za poruszanie się robota. Zadaniem planera jest wytyczać bezpieczne trasy prowadzące do niepewnych rejonów mapy lub konkretnego, ustalonego miejsca (zależnie od trybu). Planer kontrolowany jest przez `TopoMapper`, który ustala źródło eksploracji w celu oznaczenia miejsca, a po jego oznaczeniu wymusza powrót do jego centrum. Rysunek 4.1 prezentuje przykładowe trasy wyznaczone przez planer. Trasy generowane są na podstawie przetworzonego diagramu Voronoi, budowanego przez zewnętrzny komponent (zaimplementowany w ramach pracy [17]), a poddanego późniejszej obróbce w obiekcie klasy `DynamicThinning` należącego do projektu.

Efektem działania planera jest ciąg punktów kontrolnych, przez które powinien przejechać robot, aby osiągnąć cel. Poruszanie się po tych punktach jest zadaniem łazika (obiektu klasy `Traverser`), który wykonuje to poprzez bezpośrednie sterowanie obrotem i prędkością robota. Łazik ma też za zadanie unikać przeszkód na drodze robota, które pojawiają się, jeśli trasa zaprojektowana przez `Planner` opiera się na niedokładnej mapie. W przypadku napotkania przeszkody, aktualna trasa jest anulowana i `Traverser` przechodzi w stan oczekiwania na wygenerowanie nowej. Taka sytuacja zdarza się regularnie przy eksploracji nieznanego obszaru, ponieważ generowany szkielet mapy posiada odnogi w obrębie obszaru o niepewnej zajętości i budowane na jego podstawie trasy również się w ten obszar zagłębiają.

Ostatnim zadaniem obiektu klasy `Traverser` jest reagowanie na zderzenia z przeszkodami, a ponieważ zdarzenia te nie są frontalne (inaczej zostałyby wykryte przez sonar), to

rozwiązaniem jest niewielkie wycofanie się robota i dalsze kontynuowanie podróży.

Klasa `TopoMapper` oraz `PlaceExtractor`

W rozdziale 3 przedstawiony został projekt warstwy mapowania topologicznego, opartej na oznaczaniu lokacji, a następnie budowaniu z nich grafu opisującego całe otoczenie. W chwili pisania tej pracy, klasa `TopoMapper` odpowiedzialna za tworzenie tego opisu oraz ustalanie kolejnych źródeł eksploracji nie została jeszcze dopełniona modułem mapującym, którego zadania tymczasowo przypisane są użytkownikowi, uruchamiającemu symulację. Docelowo, część mapująca `TopoMapper` wydzielona jest do modułu mapującego (napisanego w języku *Python*), którego zadaniem jest wcielanie oznaczonych lokacji do mapy topologicznej oraz ustalanie kolejnych punktów eksploracji dla robota. Obiekt klasy `TopoMapper`, po otrzymaniu źródła eksploracji, oznacza znajdującą się w nim lokację. Algorytm zajmujący się oznaczaniem zaimplementowany jest w klasie `PlaceExtractor` i oparty jest na regularnych próbach wygenerowania diagramu wolnego od węzłów zawartych w obszarze o nieznanej zajętości. Bazą do budowy tego diagramu jest okno oznaczania lokacji, czyli fragment mapy metrycznej, będący prostokątnym wycinkiem o środku w źródle eksploracji i stałym rozmiarze. Dopóki takie niejednoznaczne węzły istnieją, dopóty zadanie eksploracji kontrolowane jest przez planer. Kiedy okno oznaczania lokacji jest już wolne od niepewnych węzłów, następuje detekcja lokacji, a robotowi nakazywany jest powrót do jej wnętrza. Po powrocie i ponownym oznaczeniu lokacji (w celu potwierdzenia) jest ona przekazywana do modułu w *Python*ie, który decyduje o nowym źródle eksploracji i proces oznaczania odbywa się od początku.

Na chwilę obecną zadanie wyznaczania lokacji do eksploracji przypisane jest użytkownikowi. Proszony jest on o wskazanie nowego celu na mapie metrycznej, po zakończeniu cyklu oznaczania. Proces tworzenia drzewa grafów (pochodzący z pracy [3]) nie został efektywnie zaimplementowany.

4.1.3. Konfiguracja i uruchamianie

Głównym celem kompilacji jest zbudowanie biblioteki współdzielonej `librobot.so` implementującej protokół komunikacji z symulatorem *Stage*. Oprócz niej budowane są też testy jednostkowe oraz narzędzia pomocne przy pracy nad poszczególnymi komponentami. Są to `batch_mapper`, `batch_planner`, `batch_extractor`. Na podstawie raportów zapisywanych przez sterownik podczas symulacji, odtwarzają one częściowy stan struktur danych i pełnią rolę pomocniczą.

Zależności

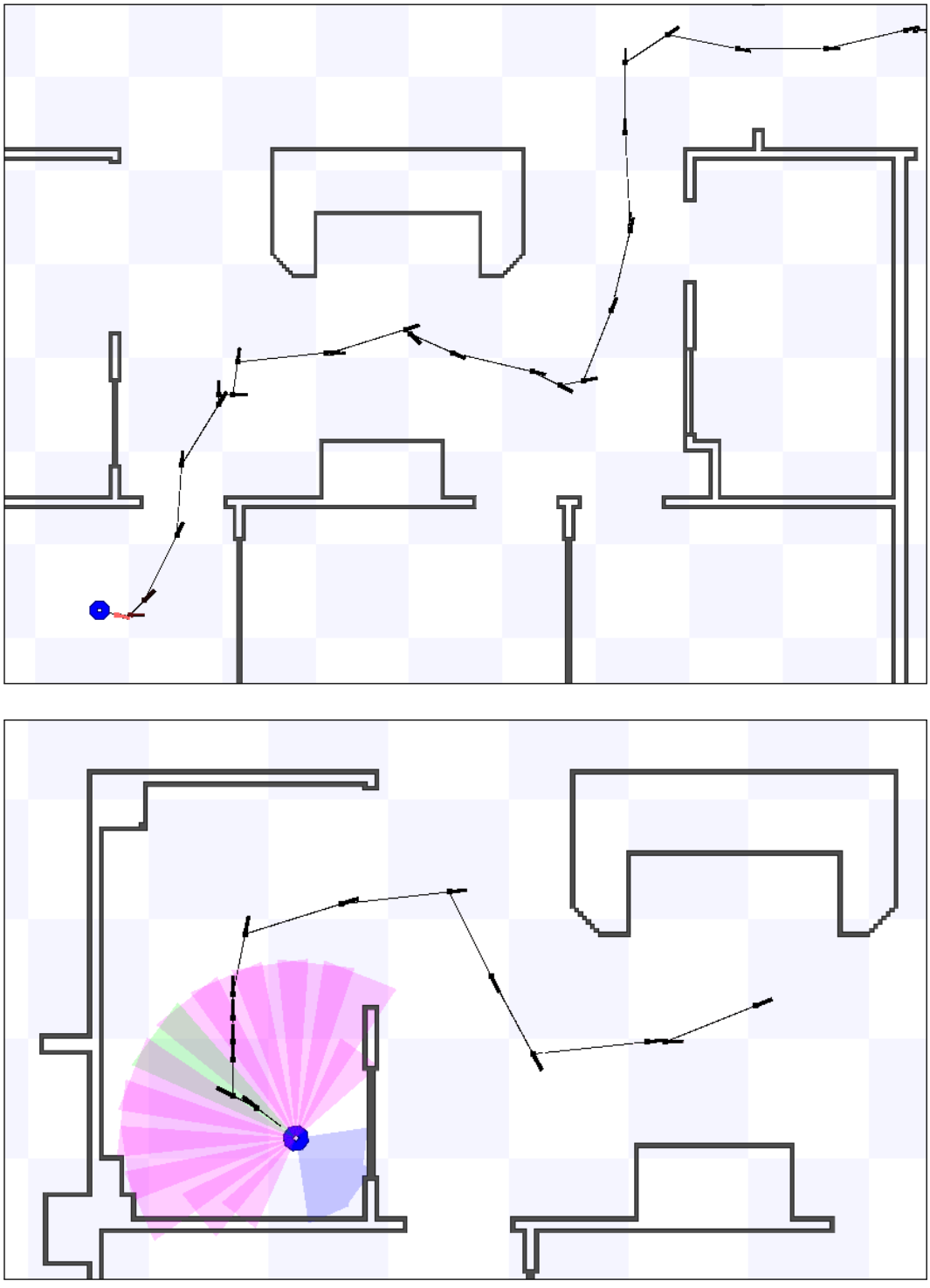
Skompilowanie projektu wymaga kilku zależności:

`boost_program_options` Umożliwia specyfikowanie ustawień konfiguracyjnych.

`boost_python` Biblioteka, przy pomocy której zagnieżdżony jest interpreter języka *Python*.

`boost_serialization` Biblioteka ułatwiająca serializację danych, używana przez pomocnicze narzędzia.

`CImg` Biblioteka dołączona do projektu, zawierająca się w pojedynczym pliku nagłówkowym. Przy jej pomocy odbywa się interakcja z użytkownikiem oraz tworzone i prezentowane są wszystkie pochodzące z pracy sterownika rysunki. Na potrzebę tej prezentacji dodatkowo odbywa się linkowanie z `X11`. Biblioteka udostępniona na licencji `CeCILL-C`.



Rysunek 4.1: Czarnym kolorem oznaczone są zaplanowane przez planer trasy. Punkty kontrolne znajdują się w równomiernej odległości od ścian minimalizując ryzyko najechania na przeszkodę. Zadaniem łazika jest przebyć tę trasę przez ustalanie prędkości obrotu i przemieszczania się. Obrazki pokazują widok symulacji w programie *Stage*.

`google-glog` System logowania, z którego korzystają wszystkie obiekty projektu. Jest on linkowany statycznie. Na potrzeby tej biblioteki dołączone jest też `libunwind`.

`DynamicVoronoi` Biblioteka złożona z kodu napisanego przez autorów [17]. Jest ona udostępniana na licencji `BSD` i została dołączona do projektu.

Oprócz wymienionych bibliotek do kompilacji wymagany jest plik nagłówkowy `stage.hh` opisujący typy danych stosowane przez symulator. Powinien on zostać zainstalowany wraz z symulatorem.

Konfiguracja

Sterownik robota posiada konfigurowalne parametry, które kontrolują działanie poszczególnych komponentów. Konfiguracja przekazywana jest przez zmienne środowiskowe, a skrypt `config.sh` dołączony do pracy zawiera ich wartości ładowane przy uruchamianiu symulacji.

Dostępne są następujące parametry:

drift_error, turn_error, range_error Parametry modelu ruchu używanego przez filtr cząsteczkowy (sekcja 3.2.2).

sonar_model_epsilon Parametr modelu sonaru (sekcja 3.1.2) ustalający szerokość przeszkody jako ułamek zasięgu sonaru.

place_window_size_m Rozmiar okna używanego przez `PlaceExtractor` do oznaczania lokacji (sekcja 3.2.3 oraz opis klasy w tym rozdziale).

filter_particle_count Liczba cząstek używanych w filtrze (N_t z sekcji 3.2.2).

filter_movement_draws Średnia (na jedną cząstkę) liczba przypasowań do mapy globalnej w fazie generowania nowych cząstek.

filter_match_divisor Współczynnik (F ze wzoru 3.2) sterujący wagą przy obliczaniu dopasowania.

pythonpath Ścieżka, która dodawana jest do `PYTHONPATH` zagnieżdżonego interpretera.

show_grid, show_local_grid, planner_show_grid, place_show_grid
Zmienne ustalające, które z diagramów wyświetlać na ekranie. Dotyczą kolejno: mapy globalnej, pary map lokalnych, diagramu planera, map oznaczanych miejsc.

serialize_updates Jeśli ustawione, to do pliku `scans_stream` zapisywanie są zserializowane dane sensoryczne dla `batch_mapper`.

save_voronoi_data_freq Okres zrzucania diagramów Voronoi do plików `voronoiXXXXX`.

interactive_topomapper Ustawienie tej flagi przełącza `TopoMapper` w tryb, w którym po każdym oznaczeniu miejsca użytkownik pytany jest o kolejne źródło eksploracji. Wyboru dokonuje się przez kliknięcie na wyeksplorowane miejsce na mapie globalnej.

disable_topomapper Ustawienie tej flagi wyłącza `TopoMapper`. Robot będzie eksplorować coraz dalsze nieznane rejony i budował mapę metryczną bez oznaczania lokacji.

Odpowiadające tym parametrom zmienne środowiskowe posiadają prefix `ROBOT_` i pisane są wielkimi literami.

Logowanie

Komponenty sterownika generują logi zawierające informacje o ich wewnętrznym stanie. Log zapisywany jest do folderu tymczasowego `/tmp/`. Biblioteka `google-glog` odczytuje konfigurację podsystemu logowania ze zmiennych środowiskowych i pozwala na zmianę tego zachowania. Plik `config.sh` definiuje `GLOG_logtostderr = 1`, aby log widoczny był w terminalu. Dodatkowe zmienne o nazwach `GLOG_v` oraz `GLOG_vmodule` pozwalają na podwyższenie szczegółowości informacji, które są logowane. Zmienne te opisane są w dokumentacji `google-glog`, ale korzystanie z nich potrzebne jest tylko przy diagnozowaniu działania sterownika.

Uruchamianie

Do symulowania robota wymagane jest zainstalowanie programu *Stage* oraz uruchomienie go z odpowiednim plikiem z opisem świata (ang. *world file*). Prezentowane w tej pracy wyniki powstały przy użyciu pliku `basic.world` dołączonego do projektu. Zawiera on opis modelu robota, definicję mapy oraz warunki symulacji: pozycję startową robota i poziom błędu w odczytach odometrii. Po ustawieniu odpowiednich zmiennych środowiskowych (przez uruchomienie skryptu `config.sh` w bieżącej sesji powłoki) możliwe jest uruchomienie symulacji poleceniem `stage worlds/basic.world`. Plik sterownika `librobot.so` musi znajdować się na ścieżce opisywanej przez zmienną środowiskową `STAGEPATH`. W bieżącym katalogu musi też znajdować się folder `dumps` na zrzuty raportów. Symulacja trwa nieprzerwanie i kontrolowana jest z poziomu okna *Stage*, pomijając wybór źródła eksploracji, który odbywa się na oknie głównej mapy. Podczas jej trwania generowane są zrzuty zawierające aktualną mapę oraz strumień odczytów sensorycznych pozwalający programowi `batch_mapper` na zewnętrzne (bez użycia *Stage*) odtworzenie symulacji z innymi ustawieniami parametrów mapera. Przekazanie parametru `-g` uruchamianemu symulatorowi pozwala na szybszą symulację, pozbawioną wizualizacji środowiska wyświetlanej przez *Stage*. Parametr nie wpływa na zrzuty stanu wyświetlane przez sterownik.

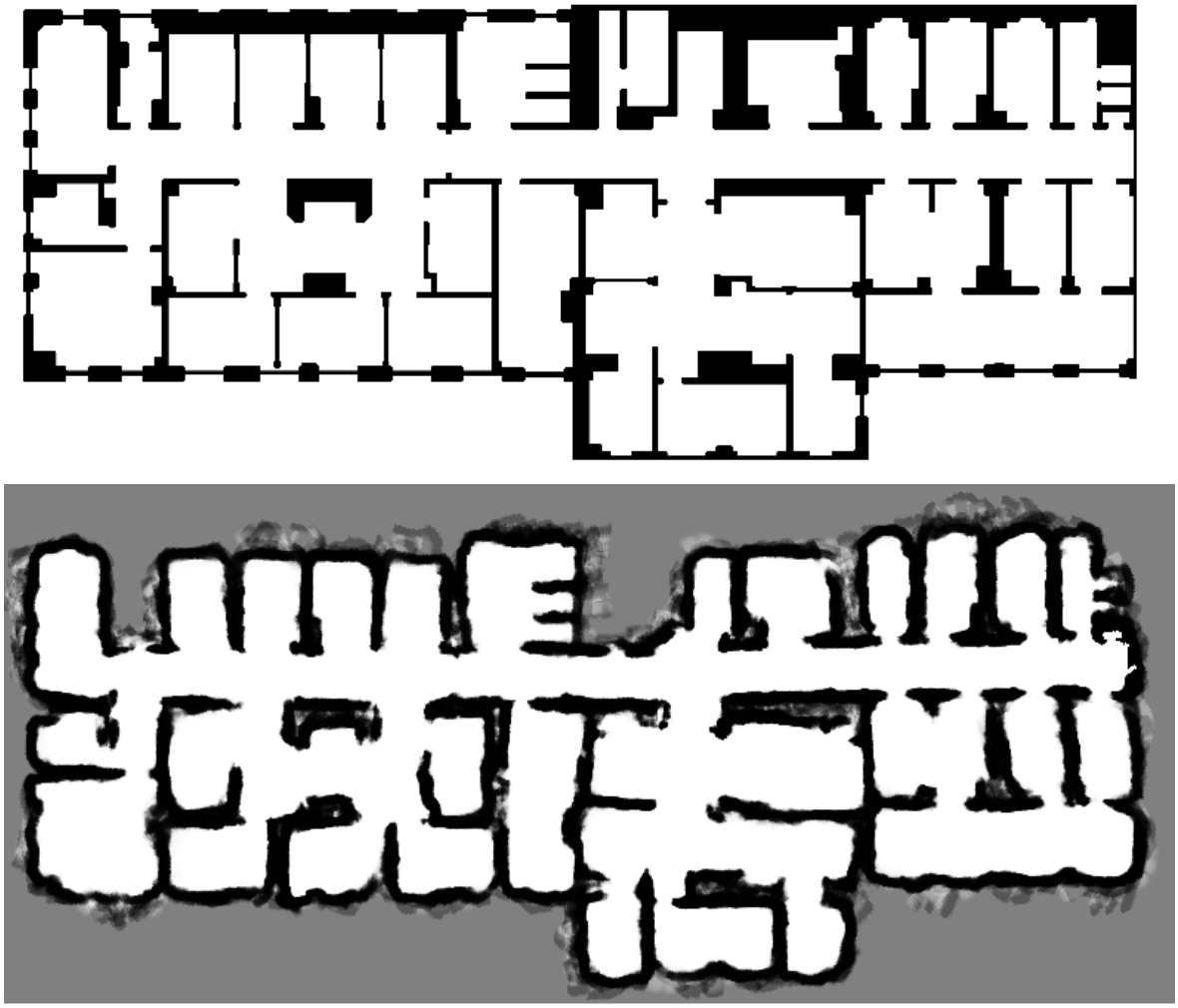
W katalogu `/stage/` na dołączonej płycie znajduje się `Makefile` automatyzujący zadanie zbudowania biblioteki i uruchomienia symulacji. Najistotniejsze cele budowy to `runG` oraz `run`, uruchamiające odpowiednio symulację ze środowiskiem graficznym i bez. Cel `replay` pozwala na uruchomienie narzędzia `batch_mapper` na danych z ostatnio zakończonej symulacji. Przed uruchomieniem programów ładowana jest konfiguracja z pliku `config.sh`.

4.2. Wyniki działania

Efektem niniejszej pracy jest sterownik symulowanego modelu robota, zdolnego do eksplorowania swojego otoczenia. Powstała implementacja pozwala na działanie w dwóch trybach, przedstawionych w kolejnych punktach tej sekcji.

4.2.1. Mapowanie metryczne

Wyłączenie mapera topologicznego (parametr `disable_topomapper`) powoduje, że robot nie będzie oznaczał lokacji, a jedynie eksplorował swoje otoczenie i aktualizował mapę metryczną, przechowywaną w filtrze cząsteczkowym. W tym trybie działania, skuteczność robota jest mocno uzależniona od jakości i precyzji sensorów. Duży odchył od rzeczywistości w odczytach przez nie generowanych utrudnia dokładne zbadanie otoczenia, gdyż nawarstwiający się w trakcie eksploracji błąd uniemożliwia poprawne mapowanie rejonów otoczenia, które odwiedzane są wielokrotnie. Dobrze obrazuje to dokładnie opisany rysunek 4.3b. Osiągnięcie



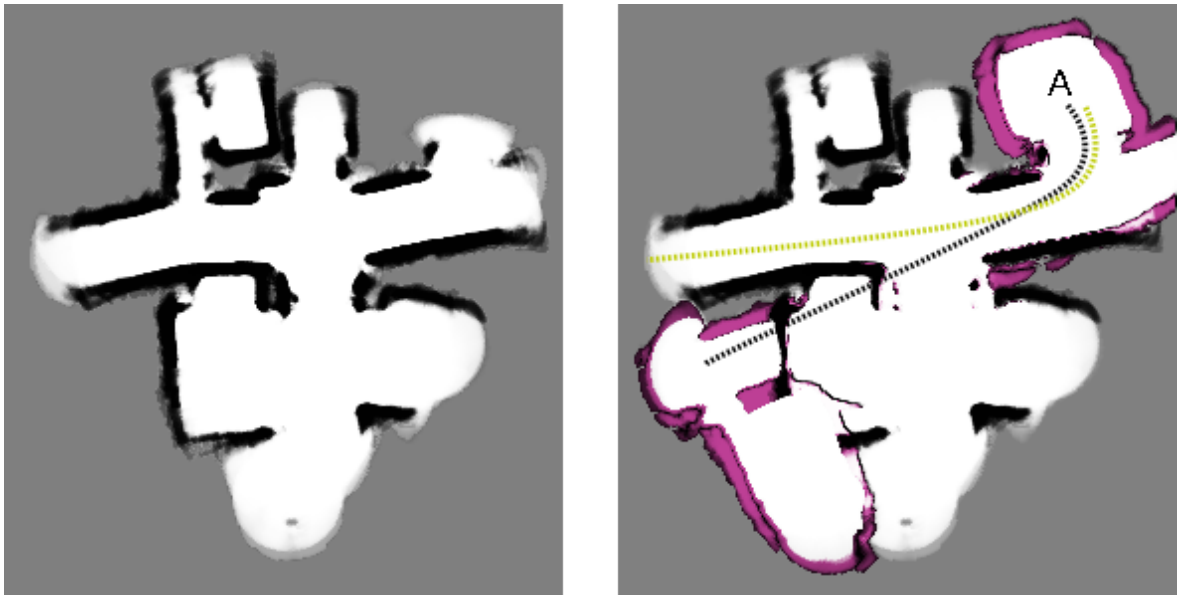
Rysunek 4.2: Prawie pełna mapa wygenerowana przez robota, przy znikomym błędzie odometrii. Nieodwiedzone miejsca zostały pominięte z powodu zbyt wąskich korytarzy. Powyżej dokładny plan użyty w symulacji

poprawności w takiej sytuacji nakłada duże wymagania na filtr cząsteczkowy, których nie można zrealizować bez dostępu do odpowiednio dużych zasobów. Na rysunku 4.2 zaprezentowana jest mapa wygenerowana podczas jednej z sesji symulacji przy błędzie procentowym obrotu oraz dystansu poniżej 0.03.

4.2.2. Oznaczanie lokacji

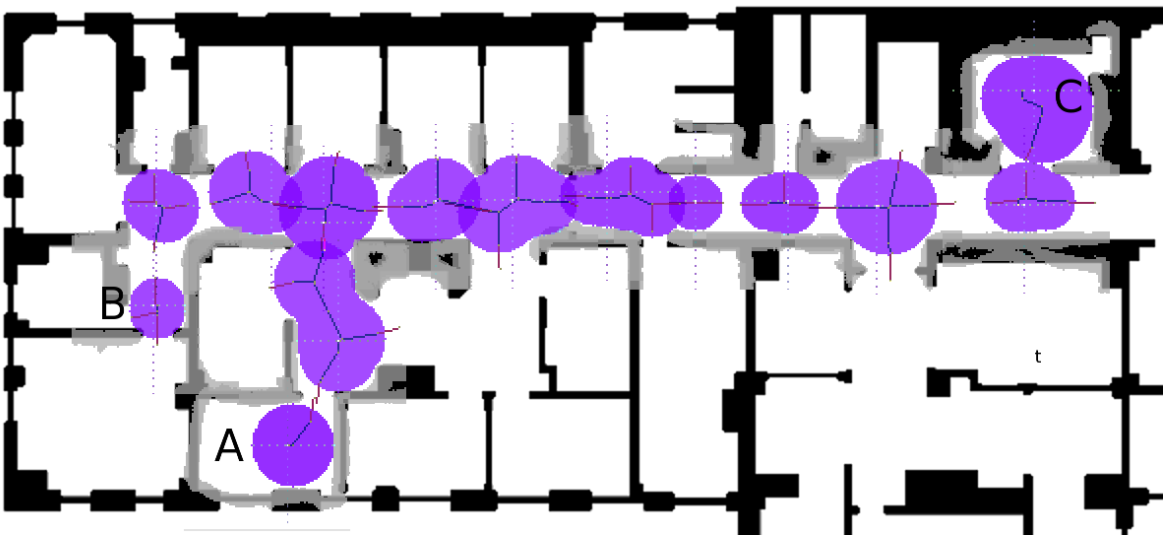
Drugim trybem działania, ważniejszym dla celu pracy, jest częściowe mapowanie topologiczne. Ze względu na nieukończony moduł mapera topologicznego, kolejne odwiedzane miejsca wskazuje użytkownik i nie jest budowany graf. Rysunek 4.4 przedstawia skompilowany wynik symulacji, podczas której robot prowadzony był wzdłuż korytarza i oznaczał wszystkie napotkane na swej drodze lokacje. Wyznaczone w ten sposób miejsca oraz ich szkielety nałożone zostały na plan otoczenia, aby uwidocznić ciągłość procesu mapowania. W dodatku B zamieszczono klatki z filmu dołączonego na płycie. Przedstawia on asystowaną eksplorację środowiska wraz z podglądem stanu komponentów projektu.

(a) Częściowa mapa lewej strony otoczenia wygenerowana na podstawie przekłamanych danych odometrycznych. Pomimo przekłamań oznaczanie lokacji i mapowanie topologiczne byłoby możliwe.



(b) Przykład błędu mapowania całkowicie uniemożliwiającego dalsze działanie. Po zbudowaniu lewej mapy robot udał się do pomieszczenia A, w którym utracił informację o swoim obrocie. Następnie robot przemierzył korytarz po raz drugi powodując fatalne przekłamanie mapy. Ciemnym kolorem oznaczona jest błędna trajektoria, przyjęta przez robota, a jasnym poprawna. Efektem jest złożenie się ze sobą dwóch kopii tej samej mapy, ale w różnych układach odniesienia.

Rysunek 4.3: Przekłamania występujące podczas mapowania metrycznego



Rysunek 4.4: Obrazek przedstawia lokacje oznaczone podczas eksploracji z asystowanym maperem topologicznym. Robot poprowadzony został z punktu A przez punkt B do punktu C, w którym zakończył pracę. Obszary na trasie, które nie przynależą do żadnej lokacji to elementy korytarzy pominiętych dla zachowania czytelności. Obrazek powstał przez nałożenie na wyeksplorowany fragment planu zrzutów ekranów z symulacji.

Rozdział 5

Podsumowanie

5.1. Realizacja celu pracy

Wynikiem niniejszej pracy jest projekt oraz realizacja sterownika dla robota, który przy udziale asystującego operatora, eksploruje symulowane środowisko oraz oznacza napotkane w nim lokacje. Jest to zaprezentowane na załączonym do pracy filmie oraz odtwarzalne na podstawie dołączonego kodu źródłowego.

Głównymi celami stawianymi na początku pracy była samodzielność zaimplementowanego sterownika robota oraz możliwość wykorzystania go do realizacji prawdziwych robotów amatorskiej konstrukcji.

Najistotniejszym wymaganiem stawianym przed sterownikiem jest jego małe zapotrzebowanie na moc obliczeniową oraz rozmiar pamięci operacyjnej. Projekt przedstawiony w tej pracy ma wymagania zależne od konfiguracji parametrów. Poziom dokładności map metrycznych, kontrolowany stałą w kodzie programu, najbardziej wpływa na te wymagania. Jego wartość powinna być minimalna, ale taka, aby generowane mapy zawierały wszystkie elementy otoczenia potrzebne do nawigacji. Pozostałe parametry umożliwiające strojenie to rozmiar okna detektora lokacji oraz liczba cząstek i losowań z modelu ruchu w filtrze cząsteczkowym. Pierwszy z nich jest zależny od maksymalnego rozmiaru lokacji, która może zostać napotkana, więc jest wartością wywodzącą się bezpośrednio z charakteru mapowanego środowiska. Pozostałe dwa parametry, określające dokładność filtra cząsteczkowego, są zależne zarówno od rozmiaru okna, jak i poziomu błędu w odczytach odometrycznych. Jest tak, ponieważ zastosowanie większego okna wymaga od warstwy metrycznej zdolności mapowania bardziej rozległego obszaru, na którą negatywnie wpływa również duży poziom błędu.

Podsumowując, wymagania zasobów zależne są przede wszystkim od rozmiaru pomieszczeń w mapowanym środowisku oraz jakości sonsarów dostępnych robotowi.

Drugim postawionym w pracy wymaganiem jest zdolność sterowanego robota do samodzielnej eksploracji. Projekt uwzględnia te wymagania, jednak ograniczenia czasowe uniemożliwiły mi na ukończenie implementacji, która całkowicie by im sprostała. Brak działającego modułu decyzyjnego i mapującego w komponencie mapera topologicznego uniemożliwia samodzielne działanie robota, ale efekty uzyskane przy manualnym sterowaniu, dają dobre rokowania na skuteczność rozwiązania, po zaimplementowaniu brakującego elementu. Ten optymizm wsparty jest także przez wyniki w pracy [3], z której pochodzi, zastosowane w niniejszej pracy, warstwowe podejście do problemu mapowania oraz technika mapowania topologicznego.

Na podstawie wyników działania robota, którego sterownik zaimplementowany został w niniejszej pracy można wywnioskować, że ostateczny cel jakim jest autonomiczny system

wielu robotów oparty na dostępnym amatorsko sprzęcie jest możliwy do zrealizowania, przy odpowiednio dużym nakładzie pracy. Pomimo ograniczeń sprzętu objawiających się niską mocą obliczeniową oraz dużym błędem pomiarowym, rozwiązanie korzystające z dostępnych technik i struktur danych byłoby kompletne. Teza ta jest prawdziwa przy założeniu małego rozmiaru pomieszczeń, aby błędy w warstwie metrycznej mapera nie prowadziły do fatalnych przekłamań mapy.

5.2. Możliwości rozwoju

5.2.1. Maper topologiczny

Główną drogą rozwoju projektu jest zapewnienie całkowicie niezależnego działania robota. W tym celu musi zostać ukończony maper topologiczny. Istniejąca implementacja jest niezależna od operatora w warstwie mapowania metrycznego, gdzie problem nawarstwiania się błędów odczytu nie jest rozwiązywalny bez zwiększenia wymagań związanych z mocą obliczeniową. Działający maper topologiczny jest obejściem tej sytuacji i to właśnie jest motywacją warstwowej struktury systemu. Dalsza praca nad sterownikiem i zaimplementowanie warstwy topologicznej pozwoli na zmniejszenie rozmiaru globalnej mapy metrycznej tak, aby zawierała ona tylko otoczenie okna służącego do oznaczania lokacji. Zabezpieczy to robota przed sytuacją zaprezentowaną na obrazku 4.3b w poprzednim rozdziale, ponieważ powrót do lokacji wcześniej odwiedzonych zarejestrowany zostanie tylko na warstwie topologicznej. Dodatkowo z poprawnej mapy topologicznej i dokładnej informacji o lokacjach możliwe jest odtworzenie dokładnej mapy metrycznej obejmującej swym zakresem całe otoczenie (za [3]).

5.2.2. Wieloagentowość

Kolejnym etapem rozwoju jest rozszerzenie implementacji na większą liczbę robotów, komunikujących się ze sobą oraz zdolnych do podziału zadań między siebie. Kompaktowy format mapy topologicznej (w porównaniu do mapy metrycznej) umożliwia zaprojektowanie algorytmu, który łączyłby dwie częściowe mapy, pochodzące od różnych robotów, w jedną pełniejszą. Pozwoliłoby to na wydajną eksplorację nawet dużych środowisk.

5.3. Wnioski odnośnie realizacji sprzętowej

Projekt powstały w ramach pisania niniejszej pracy oraz jego późniejsza implementacja stanowią bazę do dalszej pracy nad działaniem robotów w rzeczywistym środowisku. Przeniesienie projektu wymaga niskopoziomowych optymalizacji użytych algorytmów, jednak struktura rozwiązania pozostałaby taka sama.

Rzeczywiste roboty charakteryzują się bardziej skomplikowanym rodzajem błędów pomiarowych niż te zastosowane w symulatorze, co wymagałoby dokładnej kalibracji i, co bardzo prawdopodobne, bardziej szczegółowych modeli ruchu i sonaru.

Innym utrudnieniem jest także ograniczona pojemność źródła zasilania. Symulowany model robota może mapować środowisko bez ograniczeń czasowych, podczas gdy jego fizyczny odpowiednik może wymagać w między czasie doładowywania baterii. Problem ten może zostać rozwiązany poprzez umieszczenie stacji dokującej w punkcie startowym robota. Dzięki mapowaniu, działającym w czasie rzeczywistym, powrót do stacji mógłby odbyć się na podstawie tymczasowej wiedzy o otoczeniu.

Ostatnim istotnym problemem jest możliwość wystąpienia interferencji pomiędzy sonarami pochodzącymi z różnych robotów, jeśli zastosuje się wieloagentową realizację projektu.

Istnieją jednak rozwiązania stosujące kodowanie sygnału dźwiękowego (jedno z nich zaprezentowano w [15]), które wykrywają interferencję i pozwalają na odróżnienie błędnych odczytów od poprawnych. Konsekwencją zastosowania takiego rozwiązania jest mniejsza efektywność sonaru, ponieważ część odczytów musi zostać odrzucona.

Dodatek A

Zawartość płyty CD

Do pracy dołączono płytę CD z kodem źródłowym sterownika, treścią niniejszej pracy oraz filmem obrazującym działanie symulowanego robota. Poszczególne katalogi zawierają:

/src/ – kod źródłowy projektu z plikiem konfiguracyjnym dla programu `cmake` oraz w podkatalogach:

dynamicvoronoi/ – kod algorytmu *DynamicVoronoi* z pracy [17],

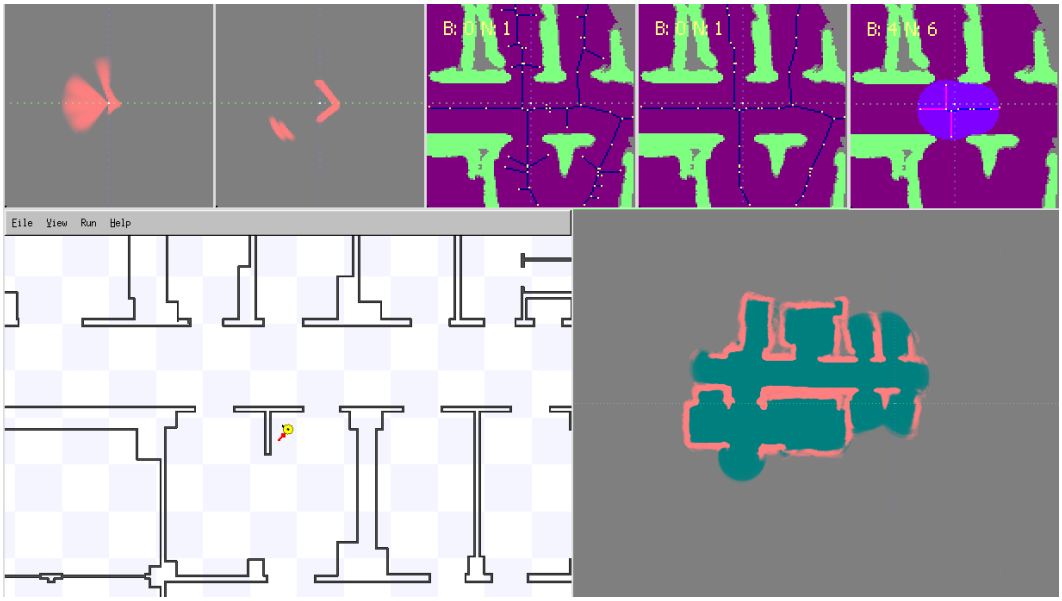
CImg/ – biblioteka `CImg`,

/stage/ – pliki konfiguracyjne, skrypty pomocnicze oraz katalog `worlds/` zawierający opis świata wymagany do uruchomienia *Stage*.

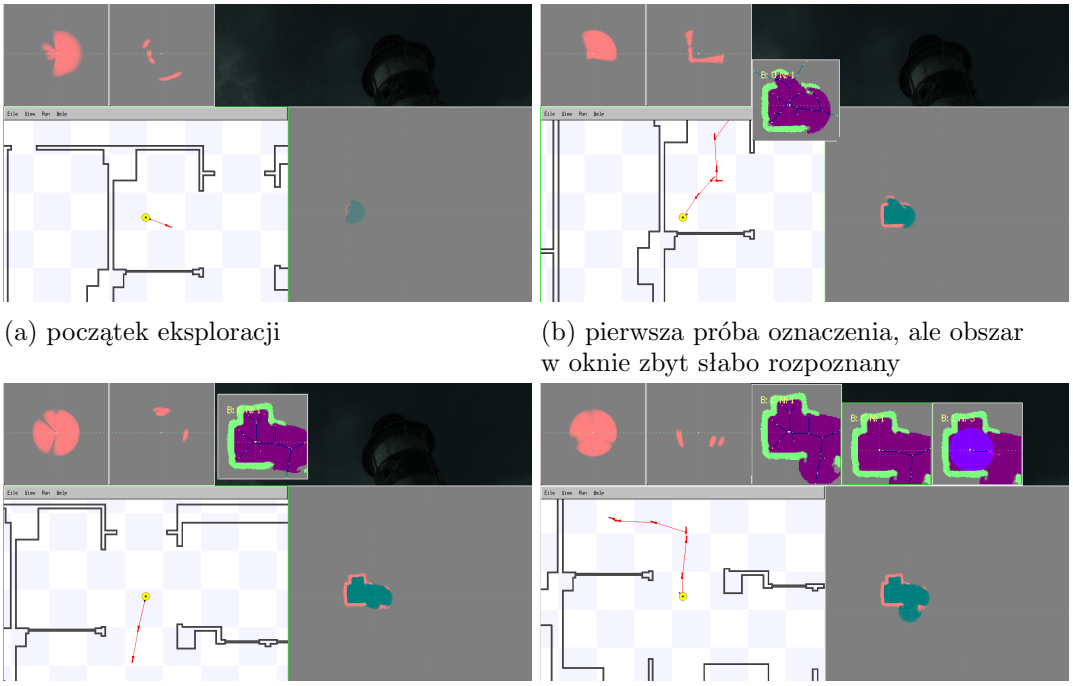
Dodatek B

Klatki z dołączonego do pracy filmu

Dla zaprezentowania efektów pracy skompilowany został film prezentujący eksplorację fragmentu mapy, w trybie z asystującym operatorem. Poniżej zaprezentowane są klatki tego filmu wraz opis wyjaśniający poszczególne jego elementy.



Rysunek B.1: Ramka z filmu dołączonego do pracy. Okna w najwyższym rzędzie kolejno od lewej to: dwie składowe mapy zajętości, nieoczyszczony diagram lokacji, dwa diagramy przedstawiające lokację po jej oznaczeniu. Poniżej okno symulacji *Stage* oraz mapa globalna.



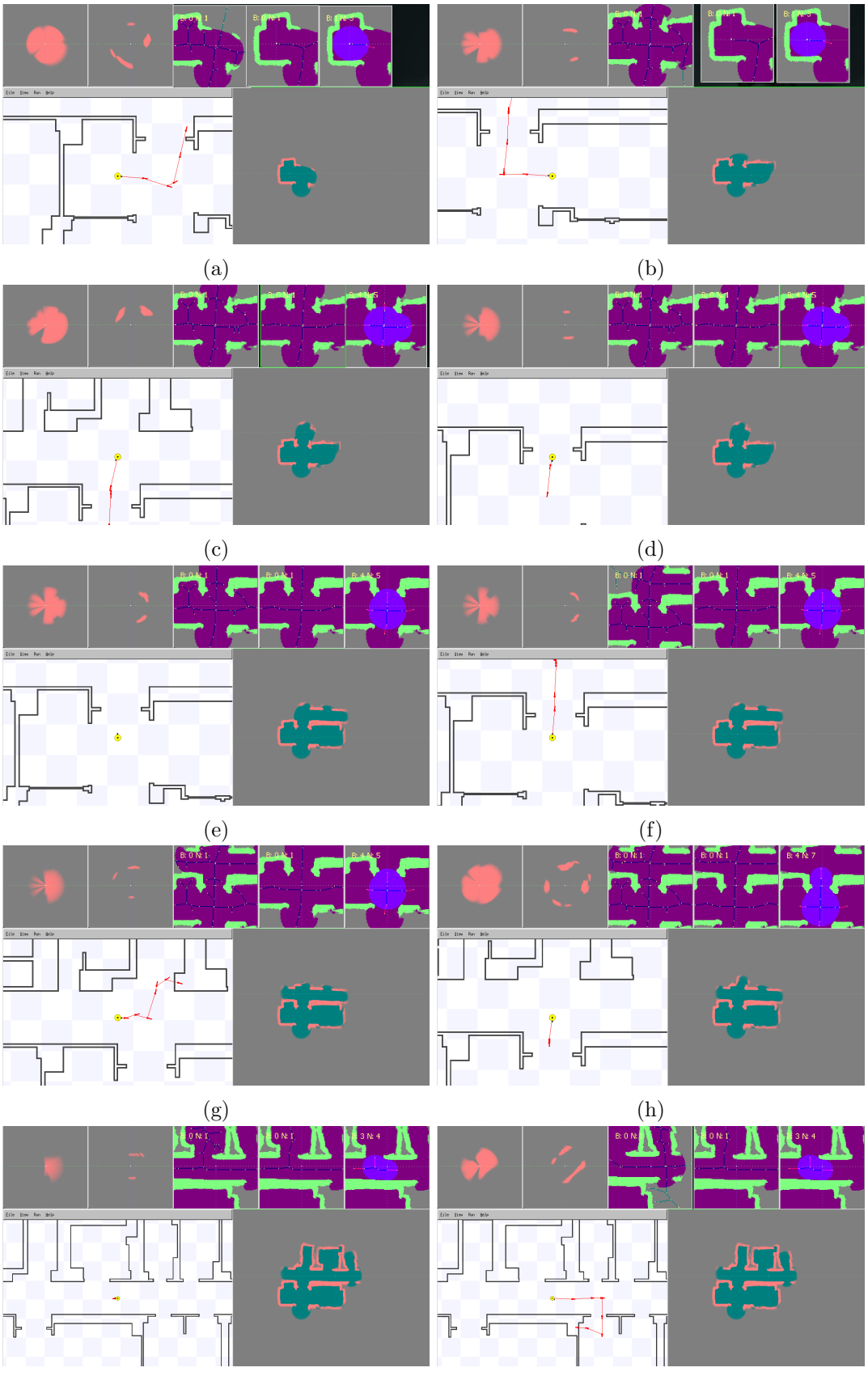
(a) początek eksploracji

(b) pierwsza próba oznaczenia, ale obszar w oknie zbyt słabo rozpoznany

(c) do zatwierdzenia pozostało odkrycie dwóch nieznanych węzłów

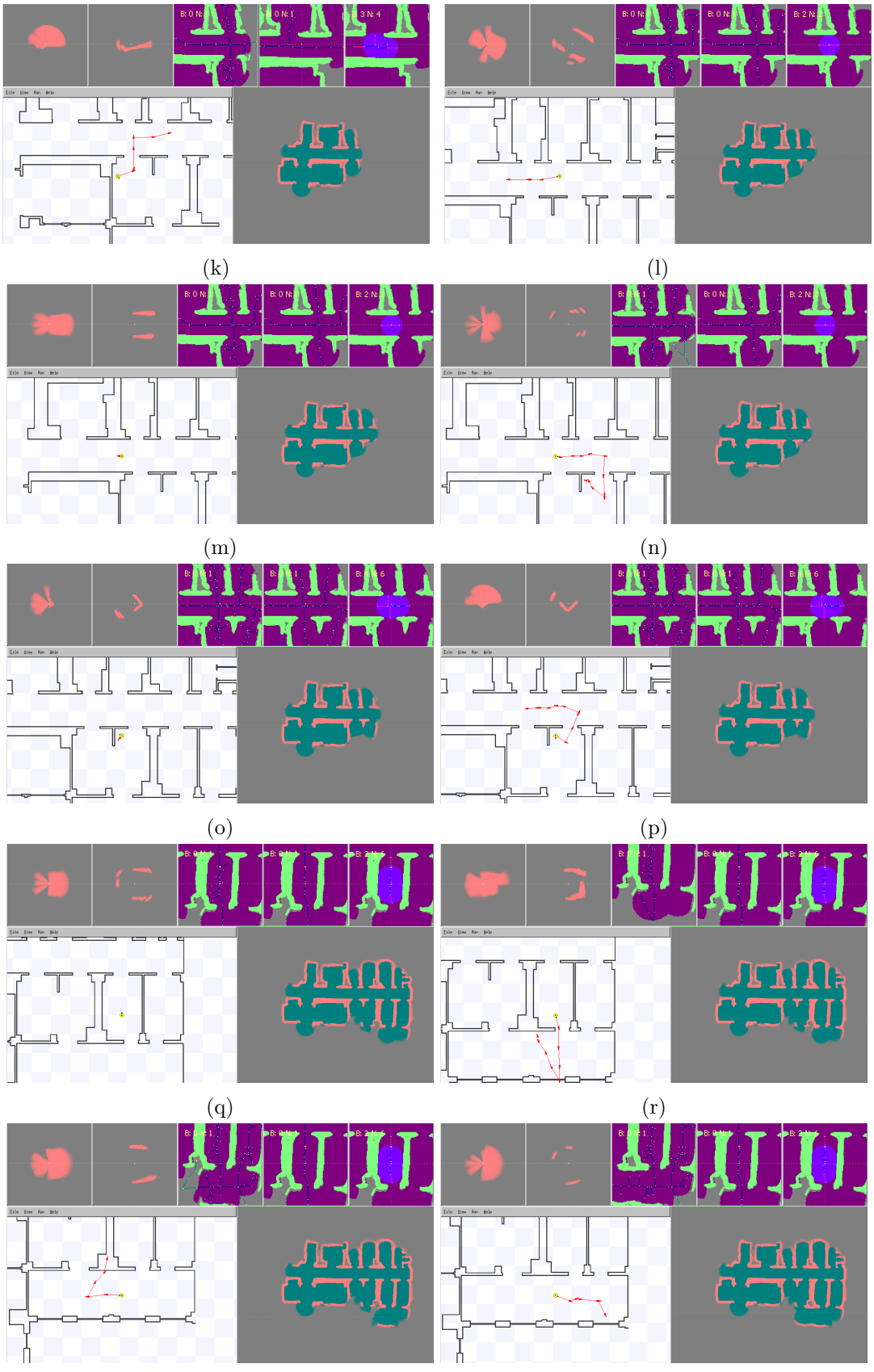
(d) oznaczono lokację, robot powraca do jej środki

Rysunek B.2: Kolejne kroki oznaczania lokacji startowej robota.



(a) (b) (c) (d) (e) (f) (g) (h) (i) (j)

Rysunek B.3: Wybrane klatki z dołączonego do pracy filmu



(k) (l)

(m) (n)

(o) (p)

(q) (r)

(s) (t)

Rysunek B.4: Wybrane klatki z dołączonego do pracy filmu (kontynuacja)

Bibliografia

[1] S. Balakirsky. Usarsim: Providing a framework for multi-robot performance evaluation. In *Proceedings of PerMIS*, pages 98–102, 2006.

[2] P. Beeson, N.K. Jong, and B. Kuipers. Towards autonomous topological place detection using the extended Voronoi graph. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 4373–4379. IEEE, 2005.

[3] P. Beeson, J. Modayil, and B. Kuipers. Factoring the mapping problem: Mobile robot map-building in the Hybrid Spatial Semantic Hierarchy. *The International Journal of Robotics Research*, 29(4):428–459, 2010.

[4] K.R. Beevers and W.H. Huang. SLAM with sparse sensing. In *Proceedings 2006 IEEE International Conference on Robotics and Automation*, pages 2285–2290. IEEE, 2006.

[5] J. Choi, M. Choi, S.Y. Nam, and W.K. Chung. Autonomous topological modeling of a home environment and topological localization using a sonar grid map. *Autonomous Robots*, 30(4):351–368, 2011.

[6] Y.H. Choi, T.K. Lee, and S.Y. Oh. A line feature based SLAM with low grade range sensors using geometric constraints and active exploration for mobile robot. *Autonomous Robots*, 24(1):13–27, 2008.

[7] H. Durrant-Whyte. Localisation, Mapping and the Simultaneous Localisation and Mapping (SLAM) Problem. *SLAM Summer School*, 2002.

[8] H. Durrant-Whyte and T. Bailey. Simultaneous localization and mapping: part I. *Robotics & Automation Magazine, IEEE*, 13(2):99–110, 2006.

[9] S. Fazli and L. Kleeman. Simultaneous landmark classification, localization and map building for an advanced sonar ring. *Robotica*, 25(03):283–296, 2007.

[10] D. Fox, W. Burgard, F. Dellaert, and S. Thrun. Monte Carlo localization: Efficient position estimation for mobile robots. In *Proceedings of the National Conference on Artificial Intelligence*, pages 343–349. JOHN WILEY & SONS LTD, 1999.

[11] B.P. Gerkey, R.T. Vaughan, and A. Howard. The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems. In *Proceedings of the 11th International Conference on Advanced Robotics*, 2003.

[12] G. Grisetti, G.D. Tipaldi, C. Stachniss, W. Burgard, and D. Nardi. Fast and accurate SLAM with Rao-Blackwellized particle filters. *Robotics and Autonomous Systems*, 55(1):30–38, 2007.

[13] A. Grossmann. Probabilistic Robotics on University of Southern California, course slides. http://www-scf.usc.edu/ csci445/, August 2007. CSCI 445.

[14] J.S. Gutmann and C. Schlegel. Amos: Comparison of scan matching approaches for self-localization in indoor environments. In *Proceedings of the First Euromicro Workshop on Advanced Mobile Robot 1996,*, pages 61–67. IEEE, 1996.

[15] L. Kleeman. Fast and accurate sonar trackers using double pulse coding. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS'99*, volume 2, pages 1185–1190. IEEE, 1999.

[16] B. Kuipers. Robotics Course. Lecture 13 Slides. http://www.cs.utexas.edu/ kuipers/slides/, October 2008. CS 344R/393R.

[17] B. Lau, C. Sprunk, and W. Burgard. Improved updating of Euclidean distance maps and Voronoi diagrams. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2010*, pages 281–286. IEEE, 2010.

[18] J.J. Leonard and H.F. Durrant-Whyte. Mobile robot localization by tracking geometric beacons. *IEEE Transactions on Robotics and Automation*, 7(3):376–382, 1991.

[19] A. Milstein. Occupancy grid maps for localization and mapping. *Motion Planning*, 2008. ISBN: 978-953-7619-01-5.

[20] H. Moravec. *Mind Children: The Future of Robot and Human Intelligence.* Harvard University Press, 1988.

[21] H. Moravec and A. Elfes. High resolution maps from wide angle sonar. In *Proceedings of IEEE International Conference on Robotics and Automation*, volume 2, pages 116–121. IEEE, 1985.

[22] P.E. Sandin. *Robot Mechanisms and Mechanical Devices Illustrated.* McGraw-Hill, 2003.

[23] C. Schroeter, H.J. Boehme, and H.M. Gross. Memory-efficient gridmaps in Rao-Blackwellized particle filters for SLAM using sonar range sensors. In *Proceedings of the European Conference on Mobile Robots*, pages 138–143. Citeseer, 2007.

[24] M. Thorup. Near-optimal fully-dynamic graph connectivity. In *Proceedings of the thirty-second annual ACM symposium on Theory of computing, STOC'00*, pages 343–350, 2000.

[25] S. Thrun. Robotic mapping: A survey. In G. Lakemeyer and B. Nebel, editors, *Exploring Artificial Intelligence in the New Millenium.* Morgan Kaufmann, 2002.

[26] R. Vaughan. Massively multi-robot simulation in stage. *Swarm Intelligence*, 2(2):189–208, 2008.

[27] Wikipedia. Robocup rescue simulation — wikipedia, the free encyclopedia, 2011. [Online; accessed 22-December-2012].

[28] Teddy N. Yap, Jr. and Christian R. Shelton. SLAM in large indoor environments with low-cost, noisy, and sparse sonars. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1395–1401, 2009.

[29] T.Y. Zhang and C.Y. Suen. A fast parallel algorithm for thinning digital patterns. *Communications of the ACM*, 27(3):236–239, 1984.