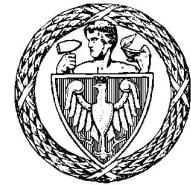


Politechnika Warszawska

WYDZIAŁ ELEKTRONIKI
I TECHNIK INFORMACYJNYCH



Praca dyplomowa inżynierska

na kierunku informatyka
w specjalności Systemy Informacyjno-Decyzyjne

**Unikanie kolizji człowiek-robot
z wykorzystaniem czujnika Kinect**

Piotr Walas

Numer albumu 269361

promotor
dr hab. inż. Wojciech Szynkiewicz

Warszawa, 2018

Streszczenie

Tytuł: Unikanie kolizji człowiek-robot z wykorzystaniem czujnika Kinect

Słowa kluczowe: ROS, Gazebo, symulacja, nawigacja, kolizja, detekcja, człowiek

Celem pracy inżynierskiej było przygotowanie systemu pozwalającego na unikanie kolizji człowiek-robot. W celu realizacji zadania posłużono się symulatorem platformy dookólnej.

W niniejszej pracy omówione zostają problemy specyfiki działania robota w środowisku człowieka, wymagania jakie powinien spełniać budowany system oraz zaprezentowany zostaje jego projekt i implementacja.

Praca wymagała zrealizowania systemu nawigacyjnego, który na podstawie danych z receptorów i odometrii robota, jest w stanie autonomicznie kierować robotem. Ważnym elementem systemu jest zdolność detekcji człowieka. W tym celu wykorzystano skaner laserowy (LIDAR), sensor RGB-D Kinect oraz odpowiednie algorytmy przetwarzania obrazu, dzięki którym możliwe było uwzględnienie strefy osobistej człowieka.

System zbudowano z wykorzystaniem programowej struktury ramowej ROS, bibliotek OpenCV i Dlib oraz symulatora Gazebo.

Abstract

Title: Avoiding human-robot collision using Kinect sensor

Keywords: ROS, Gazebo, simulation, navigation, collision, detection, human-aware

The aim of thesis was to design human-robot collision avoidance system. In order to accomplish this goal, the omnidirectional mobile platform was used.

This paper discusses issues concerning environment that contains people, specification of working in such environment, requirements which need to be met. Afterwards, project and implementation of particular system are presented.

Thesis required implementation of navigation system that allows autonomous navigation, based on received sensor's data and robot's odometry. Crucial part of system is ability to detect human and take human's personal space into consideration. LIDAR sensor, Kinect RGB-D sensor and proper image processing algorithms were utilized for that purpose.

System was implemented based on ROS framework, OpenCV and Dlib libraries and Gazebo simulator.

Spis treści

1 Wstęp	4
1.1 Motywacje	4
1.2 Cel pracy	5
1.3 Założenia	6
2 Nawigacja robotów w obecności ludzi	7
3 Opis bazy mobilnej	12
3.1 Dookólna baza jezdna	12
3.2 Czujniki	14
3.2.1 Kinect	14
3.2.2 LIDAR	16
4 Zastosowane oprogramowanie	18
4.1 Robot Operating System	18
4.2 Gazebo	18
4.3 RVIZ	19
4.4 OpenCV	19
4.5 Dlib	19
4.6 Symulator bazy mobilnej	20
5 Implementacja	22
5.1 Środowisko symulacyjne	22

Rozdział 1

Wstęp

1.1 Motywacje

Roboty usługowe i społeczne działają w bezpośrednim otoczeniu człowieka. Ich zadaniem jest wspomaganie ludzi w wykonywanych czynnościach. Przykładem takich robotów są roboty asystenci, przewodnicy, czy roboty mogące wykonywać pracę fizyczną. Istotną kwestią jest również starzenie się społeczeństwa i brak wystarczającej liczby opiekunów. Rozwiążanie problemu może dostarczyć robotyka poprzez stworzenie robotów pomagających ludziom starszym zarówno w aspekcie czysto fizycznym, jako ich pomoc w codziennych czynnościach, ale również w aspekcie społecznym, w rozumieniu robota-towarzysza.

Niezależnie od zadań jakie postawię przed nimi ich twórcy, roboty takie muszą mieć zdolność autonomicznej nawigacji w środowisku człowieka, w szczególności unikania kolizji z człowiekiem. Jest to zagadnienie nieco szersze niż zagadnienie unikania kolizji z innymi przeszkodami, gdyż należy wziąć pod uwagę kwestie bezpieczeństwa ludzi oraz kwestie społecznej akceptowalności ruchu robota. Ważnym jest, aby robot reagował odpowiednio na położenie ludzi, ich orientacje w przestrzeni, przestrzenne rozmieszczenie grupy ludzi, czy inne aspekty wynikające z uwarunkowań kulturowych. Przykładem niech będzie fakt, iż robot nie powinien zbliżać się do ludzi zbyt blisko, poruszać się tuż za ich plecami, czy wykonywać gwałtownych ruchów w ich pobliżu.

1.2 Cel pracy

Celem pracy jest opracowanie systemu obejmującego nawigację robota mobilnego, uwzględniającą położenie i orientacje ludzi w środowisku pracy.

W skład systemu wchodzi moduł budowy mapy środowiska pracy robota. Reprezentacja środowiska zawiera informację o lokalizacji robota, występujących przeszkodach, w tym o wykrytych osobach. Robot na bazie tych danych powinien generować możliwie optymalną ścieżkę ruchu do punktu zadanego oraz egzekwować ruch robota, poprzez zadawanie odpowiednich prędkości liniowych i kątowych bazy jednej.

Podsystemy detekcji ludzi w środowisku wykrywać będą położenie, orientację oraz prędkość człowieka. Aby zrealizować to zadanie użyte zostaną skanery laserowe LIDAR oraz czujnik Kinect. Dane pochodzące z czujników są wykorzystywane przez algorytmy detekcji odpowiednio nóg, oraz części głowy osoby. Wykrycie nóg pozwoli na określenie położenia oraz prędkości danej osoby, natomiast na podstawie aktualnie wykrytych części głowy można estymować kierunek w jaki zwrócona jest twarz człowieka. W efekcie system aktualizuje mapę środowiska o strefę osobistą człowieka, co z kolei pozwala na odpowiednią nawigację.

Zadania niezbędne zrealizowania systemu przedstawiono poniżej:

- Przystosowanie istniejącego symulatora bazy jezdnej do integracji z pozostałymi elementami systemu
- Stworzenie modelu czujnika Kinect w symulatorze
- Dodanie możliwości obrotu Kinecta w celu śledzenia wykrytego uprzednio człowieka
- Przygotowanie modelu środowiska testowego w symulatorze Gazebo
- Stworzenie modelu człowieka oraz możliwości poruszania nim za pomocą klawiatury
- Wykorzystanie algorytmów lokalizacji robota
- Wykorzystanie map kosztu do budowy modelu środowiska pracy robota
- Opracowanie i implementacja systemu planowania ścieżki ruchu
- Wykorzystanie sterownika platformy mobilnej

- Detekcja nóg człowieka i śledzenie jego położenia oraz prędkości za pomocą skanerów laserowych LIDAR
- Detekcja części głowy człowieka za pomocą czujnika Kinect w celu uzyskania wektora reprezentującego orientację twarzy osoby
- Scalenie list ludzi wykrytych przez czujnik LIDAR i czujnik Kinecta w celu stworzenia wspólnej listy wszystkich wykrytych ludzi
- Implementacja strefy osobistej człowieka jako odpowiedniego kształtu widocznego na mapie kosztu

1.3 Założenia

Fundamentalnym założeniem jest wykorzystanie dookólnej bazy jezdnej, wykorzystującej koła szwedzkie. Baza taka posiada zdolność do ruchu w dowolnym kierunku bez zmiany swojej orientacji. Zwiększa to wybór potencjalnych, dopuszczalnych ścieżek ruchu. Wybór robota ograniczonego więzami nieholonomicznymi mogłoby stanowić istotny problem.

W realizacji zadania zostanie użyty istniejący, odpowiednio przystosowany symulator bazy jezdnej robota Velma. Zastosowanie symulatora w znaczący sposób przyspiesza implementację, a przede wszystkim testowanie rozwiązania, bez potrzeby narażenia rzeczywistego robota na uszkodzenie wynikające z wadliwego działania systemu. Ważne jednak, by system w stosunkowo prosty sposób można było przenieść w przyszłość na rzeczywistą platformę.

Przyjętym założeniem jest określenie orientacji człowieka na podstawie orientacji jego głowy. Zadanie można rozwiązać inaczej, biorąc pod uwagę ustawienie jego tułowia, lecz zastosowana koncepcja jest interesująca z punktu widzenia zbadania sprawności detekcji tak niewielkich obiektów jak profil twarzy człowieka. W celu detekcji wykorzystane będą modele czujników skanera laserowego LIDAR i czujnika Kinect.

System powinien zostać zaimplementowany w sposób umożliwiający jego przeniesienie na platformę rzeczywistą.

Rozdział 2

Nawigacja robotów w obecności ludzi

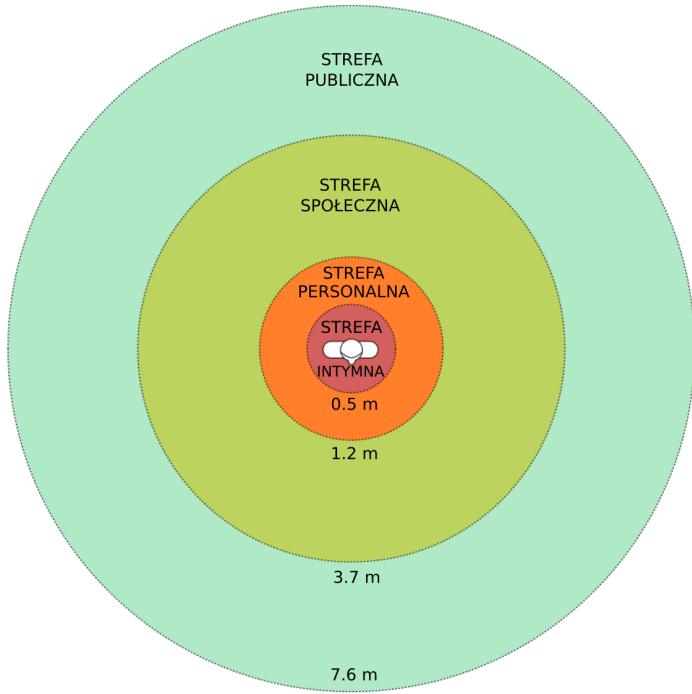
Problem nawigacji robotów w środowisku człowieka, znany w literaturze anglojęzycznej jako *human-aware navigation*, jest zagadnieniem badawczym o rosnącej popularności. W trakcie badań określono następujące problemy odnoszące się do ruchu robota [1]:

- Bezpieczeństwo ludzi i robota
- Komfort ludzi, czyli brak stresu wywołanego nieprawidłowym zachowaniem robota
- Naturalność, a więc podobieństwo zachowania robota do ludzkiego
- Zdolność robota do rozumienia i imitacji konwencji kulturowych.

Zauważono również, że kwestia bezpieczeństwa ruchu nie jest tożsama z komfortem ludzi, gdyż robot może poruszać się bezpiecznie, nie doprowadzać do kolizji z człowiekiem, lecz dla człowieka jego zachowanie może wydawać się niepewne, nie wzbudzać zaufania ludzi do użytej technologii [1].

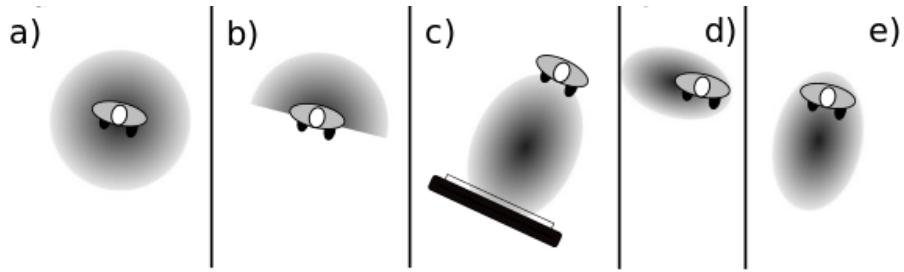
Podstawą do rozważań jest studium E. T. Halla na temat dystansu interpersonalnego [2], które wyróżnia cztery strefy:

- strefa intymna - strefa bliskich relacji, dotyk - do 0.5m
- strefa personalna - interakcje w obrębie przyjaciół i rodziny - od 0.5m do 1.2m
- strefa społeczna - interakcje ze znajomymi - od 1.2m do 3.7m
- - strefa publiczna - publiczne wystąpienia - od 3.7m.



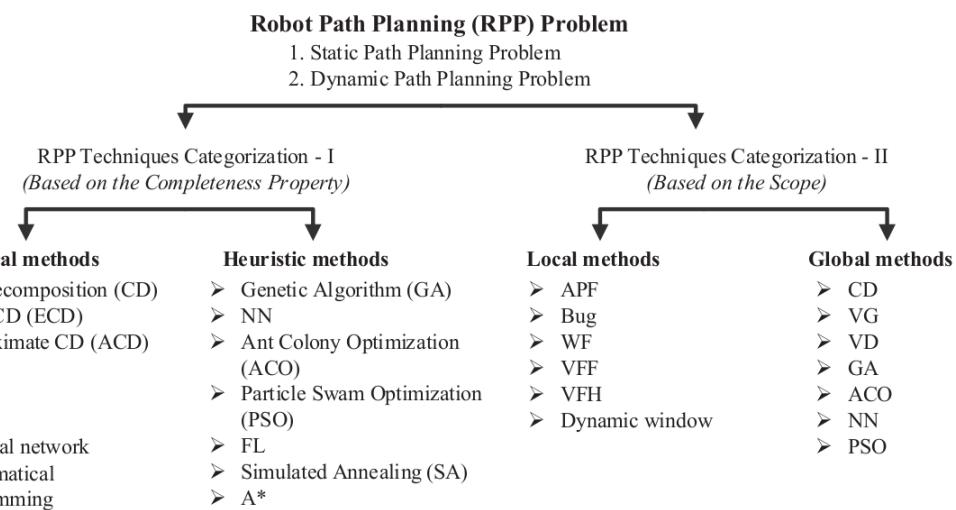
Rysunek 2.1: Przedstawienie stref zaproponowanych przez E.T. Halla w metrach, na bazie rysunku [3]

Badania dotyczące stref społecznych mają istotne znaczenie dla określenia naturalności ruchu robota. Robot pracujący z człowiekiem musi zachować odległość, zależnie od wykonywanych zadań i konstrukcji robota. Strefy personalne implementowane najczęściej za pomocą pól potencjału, bądź funkcji kosztu [1]. Drugie podejście obrazuje rysunek [2.2]. Podobną koncepcję można również wykorzystać w przypadku poruszającego się człowieka, gdzie kształt strefy personalnej, na bazie wektora ruchu, można obliczyć za pomocą funkcji gaussowskich [4] [5]. Część badaczy wysuwa jednak wniosek, że istotniejszym czynnikiem niż zachowanie odległości, jest przestrzeganie zasad społecznych, kulturowych. To ich naruszenie powoduje dyskomfort pracy z robotem [1], [6]. Pożadaną umiejętnością robota jest także zmniejszenie prędkości przy podejściu do człowieka, oraz zachowanie odpowiedniego kierunku skierowania czujników, dla robotów przypominających istoty humanoidalne.



Rysunek 2.2: Przykładowe strefy o zwiększym koszcie ruchu wokół człowieka. Mogą się one różnić w zależności od przyjętych założeń [1]. a) strefa o kształcie symetrycznym, kołowym, b) strefa o kształcie półkola - robot może poruszać się tuż przed twarzą osoby, lecz nie tuż za jej plecami, c), d) strefa eliptyczna skierowana zgodnie z orientacją osoby, d) strefa eliptyczna zwrócona w kierunku prawej strony osoby - robot będzie preferował ruch z jej lewej strony

Kwestia nawigacji i unikania kolizji robot-człowiek nie byłaby możliwa bez rozwiązania problemu planowania ścieżki ruchu. Jedną z możliwych klasyfikacji metod planowania ścieżek ruchu przedstawia rysunek [2.3].



Rysunek 2.3: Taksonomia algorytmów planowania ścieżki dla robotów mobilnych [7]

Algorytmy planowania ścieżki można podzielić na metody statyczne i dynamiczne. Metody statyczne zakładają, informację na temat środowiska znane są a priori, a środowisko jest niezmienne. Metody dynamiczne nie posiadają z góry zdefiniowanego środowiska, bądź jest ono zdefiniowane jedynie częściowo. Algorytmy można również podzielić ze względu na *kompletno* i *zasieg*. Kompletność określa czy w przypadku istnienia ścieżki ruchu, zostanie ona z pewnością znaleziona przez algorytm. Wyróżniamy metody klasyczne i heurystyczne. Celem metod klasycznych jest wyznaczanie najlepszej dopuszczalnej ścieżki ruchu, bądź stwierdzenie, że takowa nie istnieje. Odbywa się to jednak kosztem dużego nakładu obliczeniowego. Algorytmy

heurystyczne nie gwarantują znalezienia optymalnego rozwiązania, ani znalezienia rozwiązania, nawet jeśli takowe istnieje [8]. Zaletą algorytmów heurystycznych jest jednak większa szybkość działania. Ze względu na zasięg, algorytmy można podzielić również na metody globalne i lokalne. Metody globalne umożliwiają znalezienie całej ścieżki ruchu przed jego rozpoczęciem. W metodach globalnych zakłada się dostępność modelu środowiska. Nie są one jednak efektywne w przypadku dynamicznych przeszkód. Do tego celu wykorzystuje się metody lokalne, które operują na danych dostarczonych przez czujniki, takie jak kamery RGB czy skanery laserowe [7], [8], [9]. Metody lokalne planują ścieżkę ruchu inkrementacyjnie, istnieje jednak możliwość niepowodzenia generacji ścieżki. Często więc systemy planowania ścieżki ruchu budowane są z użyciem dwóch planistów, globalnego, który wyznacza ogólną ścieżkę ruchu od robota do zadanego celu i lokalnego, którego zadaniem jest obliczenie aktualnego wycinka ścieżki [7].

Popularnym podejściem stosowanym przy implementacji planisty lokalnego jest Metoda Okna Dynamicznego (Dynamic Window Approach). W metodzie uwzględnia się wszystkie ograniczenia ruchu robota i w każdej iteracji tworzy zadaną liczbę par (v, w) , będących parami prędkościami liniowymi i kątowymi robota. Następnie dla każdej pary badany jest wpływ jej zastosowania na przyszły stan robota - to czy zderzy się on z przeszkodą, czy podąży w stronę celu, czy będzie trzymał się zadanej globalnej ścieżki itp. [10]. Wszystkie możliwe trajektorie są oceniane i wybierana jest najwyższej punktowana. Sposób oceny można dostosować zmieniając wagę kryteriów.

Kompletnym systemem realizującym zadania nawigacji w środowisku człowieka jest system opracowany przez Sisbota i.in. w 2007, opisany w pracy [11]. Praca zakładała stworzenie systemu planowania ruchu uwzględniającego pozycję, posturę (siedząca, stojąca) oraz zależności przestrzennych, takie jak nie przejeżdżanie w niewielkiej odległości od człowieka, tuż za jego plecami, czy też bliskość przeszkód blokujących pole widzenia danej osoby.

Niezależnie od modułu planowania ścieżki ruchu zaimplementowano również system wykrywania człowieka na bazie odczytów czujników laserowych oraz kamery RGB. Dane pozyskane z sensorów laserowych były przetwarzane przez algorytm pozwalający na wykrycie nóg człowieka. Na tej podstawie można było stwierdzić jego położenie. Obraz z kamery RGB służył do detekcji twarzy z bliskiej odległości. Badacze sformułowali dwa kryteria: bezpieczeństwa - zapewniające bezpieczeństwo człowiekowi i robotowi, poprzez zapewnienie odpowiedniego odległości oraz widoczności - stwierdzono, że ludzie czują się lepiej, gdy robot znajduje się

możliwie długo w ich polu widzenia. Koszt łączny bezpieczeństwa i widoczności (2.1), był następnie nanoszony na mapę kosztu, z której korzystał zaimplementowany planista ruchu nazwany HAMP (Human Aware Motion Planer).

$$Cost_{merged}(x, y) = w_1 Cost_{safety}(x, y) + w_2 Cost_{visibility}(x, y) \quad (2.1)$$

gdzie x i y to współrzędne na mapie, a w_1 i w_2 to współczynniki wagowe.

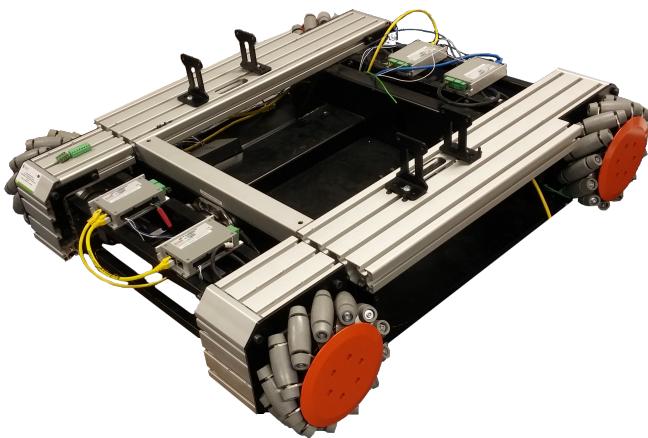
A. Mateus w pracy [4] przedstawia implementację systemu wykorzystującego moduł nawigacyjny (*navigation*) wbudowany w środowisko ROS. Celem systemu było wykrycie człowieka i planowanie ścieżki ruchu uwzględniającej szereg ograniczeń, m.in. odległość od człowieka, jak najniższy koszt zaplanowanej trasy, nawigacja ze stałą prędkością, czy omijanie ludzi z ich lewej strony. System zakładał istnienie globalnego i lokalnego modułu planowania ścieżki. Globalny wykorzystywał algorytm A^* (rozwijany był również algorytm Dijkstry), a lokalny algorytm *Trajectory Rollout*. Model środowiska budowano za pomocą pakietu *costmap_2d*, będącym częścią środowiska ROS. Zawiera on implementacje map kosztów. Wykryte osoby były na nie nanoszone wykorzystując zdefiniowaną przez autora funkcję gaussa. W innej pracy [5] D. Vasquez i.in. proponują system zawierający wykrywanie i śledzenie ludzi, predykcje konfiguracji otoczenia, biorąc pod uwagę pozycję i prędkość ludzi, oraz nawigację robota. System predykcyjny skonstruowany został za pomocą Rosnącego Ukrytego Modelu Markova [12] oraz Filtru Kalmana. Strefa personalna człowieka modelowana była ponownie z wykorzystaniem funkcji gaussa. R. Kirby w pracy [13] przedstawia strukturę strukturę systemu nawigacyjnego akceptowalnego przez ludzi z użyciem metody zoptymalizowanej pod kątem ograniczeń ruchu (*Constraint – Optimizing Method for Person-Acceptable NavigatION –COMPANION*). Praca rozwija temat ograniczeń ruchu robota w środowisku pracy z ludźmi, planowania ścieżki oraz śledzenia (wizyjnego) ludzi, oraz podążania za nimi. Wśród zdefiniowanych ograniczeń ruchu występują: minimalny dystans robota od człowieka, odpowiednia przestrzeń buforowa pomiędzy przeszkodami a robotem, przestrzeganie stref komfortu człowieka, omijanie człowieka z prawej jego strony oraz poruszanie się robota zgodnie z kierunkiem zwrotu jego "twarz". W systemie zaproponowano użycie robota opartego o platformę dookólną.

Rozdział 3

Opis bazy mobilnej

3.1 Dookólna baza jezdna

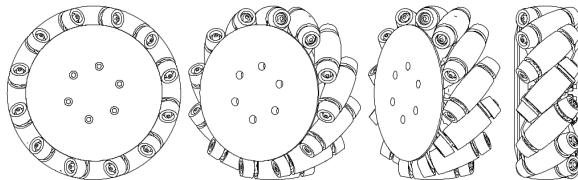
Bazą sprzętową, wykorzystywaną w niniejszej pracy jest dookólna baza mobilna. Jest to baza holonomiczna, która może poruszać się w dowolnym kierunku bez zmiany swojej orientacji oraz zmienić kierunek ruchu w dowolnej chwili. Możliwe jest to dzięki zastosowaniu kół szwedzkich. Koła te umieszczone są w narożnikach prostokątnej platformy i napędzane są niezależnie przez serwomotory. Dzięki zamontowanym enkoderom możliwy jest odczyt z odometrii robota. Platformę przedstawia rysunek [3.1].



Rysunek 3.1: Dookólna baza mobilna [14]

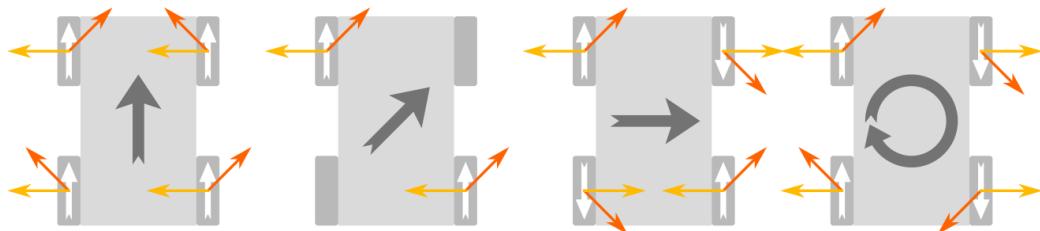
Koła szwedzkie mają specjalne rolki ustawione pod kątem 45° w stosunku do osi obrotu koła. Rolki te są pasywne, podążają jedynie za ruchem koła macierzystego. Ustawienie globalne kół, w znaczeniu kierunku skierowania rolek na platformie odpowiada literze X , patrząc na

platformę z góry. W kole szwedzkim wyróżnia się dwa kierunki ruchu: ruch napędzany w kierunku równoległy do płaszczyzny koła oraz ruch swobodny w kierunku prostopadłym do osi obrotu rolki będącej aktualnie w kontakcie z podłożem. Ustawienie to dostrzec można na rysunku [3.1]. Szkic kół szwedzkich pokazano na rysunku [3.2].



Rysunek 3.2: Koła szwedzkie [14]

Ruch platformy odbywa się poprzez odpowiednie zadawanie prędkości każdego z kół. Konstrukcja koła szwedzkiego powoduje, że nie występuje tarcie w kierunku 45° do osi obrotu koła. Ma to istotne znaczenie, gdyż siła jaka działa na koło jest prostopadła do tego kierunku (moment siły jest równoległy do osi koła). Konstrukcyjnie więc ustawiając koła w literę X i odpowiednio manipulując wartościami prędkości dla każdego z kół, otrzymujemy możliwość ruchu w dowolnym kierunku. Momenty sił, siły, wektory prędkości kół oraz kierunek poruszania się platformy demonstruję rysunek [3.3].

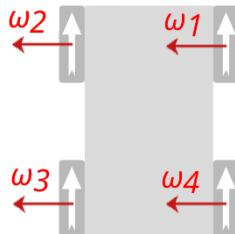


Rysunek 3.3: Ruch platformy mobilnej (widok z góry) - ciemna strzałka symbolizująca kierunek ruchu platformy, biała strzałka symbolizująca ruch koła, żółta strzałka symbolizująca moment siły działający na koło, czerwono-pomarańczowa strzałka symbolizująca kierunek siły działającej na koło [14]

Ruch platformy można opisać równaniem (3.1) kinematyki prostej:

$$\begin{bmatrix} v_x \\ v_y \\ \omega_z \end{bmatrix} = \frac{r}{4} \begin{bmatrix} -1 & 1 & -1 & 1 \\ 1 & 1 & 1 & 1 \\ \frac{2}{a+b} & \frac{-2}{a+b} & \frac{-2}{a+b} & \frac{2}{a+b} \end{bmatrix} \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ \omega_4 \end{bmatrix} \quad (3.1)$$

Zmienne v_x, v_y wykorzystane w równaniu to zadane prędkości liniowe, w_z to prędkość kątowa platformy wokół osi z, r jest promieniem koła, a to rozstaw kół na tej samej osi, b to rozstaw osi. Zmienne w_i to prędkość kątowa koła. Numerację kół przedstawia rysunek [3.4]



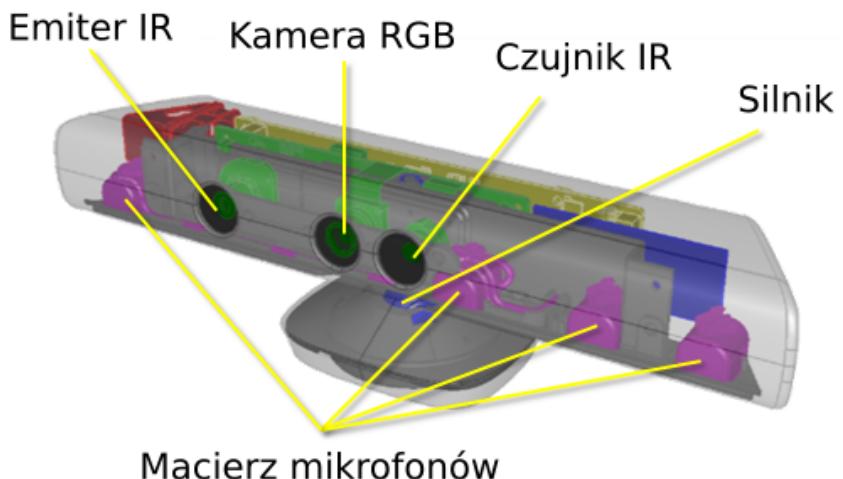
Rysunek 3.4: Numeracja kół platformy dookólnej

Użycie platformy dookólnej ma uzasadnienie w kontekście realizacji ruchu w środowisku pracy z człowiekiem, ponieważ pozwala na efektywne tworzenie ścieżki ruchu, bez narzutu ograniczeń ruchu bazy nieholonicznej. Przy bardzo bliskiej detekcji człowieka, bądź jego względnie dużej prędkości, system nie byłby w stanie stworzyć odpowiedniej trajektorii dla robota, co mogłoby powodować zatrzymanie ruchu do czasu oddalenie się osoby. Ponad to platforma dookólna sprawnie porusza się w wąskich przejściach oraz potrafi obrócić się w miejscu.

3.2 Czujniki

3.2.1 Kinect

Kinect jest czujnikiem wyprodukowanym przez Microsoft dla konsoli Xbox 360 [15]. W oryginalnym zastosowaniu czujnik był wykorzystywany do obsługi konsoli za pomocą gestów i poleceń głosowych oraz wykorzystany był w grach wideo. Jest również często wykorzystywany w robotyce ze względu możliwości i niską cenę. Urządzenie składa się z promiennika podczerwieni oraz dwóch kamer - kamery wizyjnej RGB oraz kamery na podczerwień (IR) zwracającej informację o głębi obrazu. Emiter podczerwieni emittuje wiązkę promieni podczerwonych, a kamera podczerwieni bada odbicia promieni od obiektów. Im obiekt jest bliżej, tym mocniejsze pada na niego światło. W ten sposób otrzymuję się chmurę punktów, reprezentującą rozmieszczenie przestrzenne obiektów i interpolowany obraz głębi [16].



Rysunek 3.5: Budowa Kinecta - emiter podczerwieni, kamera RGB, czujnik głębi, silnik i macierz mikrofonów [16]

Kinect zawiera również macierz mikrofonów, pozwalającą na realizację wykrywania komend użytkownika oraz silnik pozwalający na zmianę kąta pochylenia bazy. Tabela [3.1] przedstawia specyfikacje urządzenia.

Rozdzielcość obrazu RGB	640x480
Rozdzielcość obrazu głębi (interpolowana)	300x200 (640x480)
Zakres pracy	0.4m-6.5m
Pole widzenia	43° pionowo, 57° poziomo
Klatki na sekunde (dla obu kamer)	30
Format audio	16-kHz, 24-bit, modulacja PCM
Charakterystyka audio	Macierz czterech mikrofonów 24-bit, przetwornik A/C, niwelacja echa i szumu

Tablica 3.1: Specyfikacja czujnika Kinect [16]

Posługując się inżynierią wsteczną ustalono, że Kinect jest w stanie dostarczyć obraz RGB o rozdzielcości do 1280x1024, lecz przy zmniejszonej liczbie klatek na sekundę. Wysyła on

także bezpośrednio (zanim zostanie skonwertowany na mapę głębi) obraz z kamery IR w rozdzielczości 640x480, lub 1280x1024 przy zmniejszeniu liczby klatek na sekundę. Rzeczywisty zakres pracy określono na 1.2m-3.5m [17].

Obraz RGB oraz dane o biegłości obrazu posłużą do wykrycia i ustalenia orientacji człowieka w przestrzeni. Baza mobilna sama w sobie nie posiada jednak zamontowanego Kinecta, co będzie poruszane w trakcie implementacji.

3.2.2 LIDAR

Zasada działania skanerów laserowych LIDAR opiera się na wysyłaniu wiązki laserowej i mierzenia czasu jej powrotu po odbiciu od obiektów. Na tej podstawie otrzymuję się informację o rozmieszczeniu obiektów w polu widzenia sensora. Wewnątrz sensora znajduje się ustawione pod kątem 45° w stosunku do pionu lustro. Lustro zamontowane jest na wale silnika wraz z enkoderem. Urządzenie emittuje wiązkę laserową i na podstawie opóźnień w czasie powrotu wiązki do detektora oraz kąta obrotu wału silnika generowane są wyniki pomiarów [18].



Rysunek 3.6: LIDAR LMS100-1000 firmy SICK

Platforma mobilna wyposażona jest w dwa czujniki LIDAR LMS100-1000 firmy SICK (rysunek 3.6). Umiejscowione po jej przeciwnych stronach. Powodem zastosowania dwóch skanerów jest ograniczone pole widzenia pojedynczego czujnika. Dwa skanery pozwalają dookólną obserwację otoczenia. Specyfikację techniczną skanera przedstawia tabela [3.2].

Pole widzenia	270°
Długość wykorzystanej fali lasera	905nm (podczerwień)
Częstotliwość pracy	50Hz
Maksymalna odległość pomiaru	20m
Rozdzielcość pomiaru	0.5°
Systematyczny błąd pomiarowy	0.03m
Przypadkowy błąd pomiaru odległości	0.012m

Tablica 3.2: Specyfikacja skanera laserowego LIDAR LMS100-1000 firmy SICK [14]

Skanery LIDAR w niniejszej pracy są wykorzystywane do tworzenia mapy środowiska, wykrywania przeszkód oraz detekcji człowieka.

Rozdział 4

Zastosowane oprogramowanie

4.1 Robot Operating System

Robot Operating System (ROS) to programowa struktura ramowa, zawierająca zbiór bibliotek realizujących wykorzystywane w robotyce funkcje. ROS określa strukturę i mechanizmy komunikacji międzyprocesowej. Umożliwia stworzenie struktury rozproszonej, pozwalającej na wykonywanie obliczeń na wielu maszynach jednocześnie. Zasada działania opiera się na implementacji pakietów zwanych węzłami, realizujących konkretne zadania. W systemie istnieje jeden węzeł główny, poprzez którego realizowana jest komunikacja między poszczególnymi pakietami. Wymiana informacji polega na publikacji i subskrypcji wiadomości. Wiadomości nadawane są specjalnymi kanałami zwany tematami. Każdy węzeł może nadawać i subskrybować dowolną liczbę tematów. Same wiadomości to zbiór zmiennych, które twórca może stworzyć samodzielnie, bądź posłużyć się gotowymi wzorcami [19]. ROS współdziała z językami C++, Python i Lisp.

ROS dostarcza abstrakcje sensorów, w tym przypadku sensorów LIDAR i Kinect, zawiera implementacje map kosztu i planerów ruchu. Istotnym pakietem jest pakiet *TF*, który pozwala na zarządzanie wieloma systemami koordynatów [20].

W pracy wykorzystano dystrybucję ROS Kinetic z 2016r., w systemie operacyjnym Ubuntu 16.04.4.

4.2 Gazebo

Gazebo jest narzędziem pozwalającym na tworzenie symulacji i ich zobrazowanie w 3D. Umoż-

liwia implementację modeli robotów, środowiska ich pracy oraz innych elementów niezbędnych do przeprowadzenia symulacji. Modele zazwyczaj definiuje się w formacie SDF, bądź korzysta z wbudowanego kreatora. Format SDF opracowany został na bazie formatu XML, specjalnie na potrzeby opisu robotów i czujników [21]. Plik w formacie SDF zawiera informacje o reprezentacji graficznej modelu, modelu jego kolizji, przegubach i innych drugorzędnych parametrach. Zachowanie modelu implementowane jest poprzez specjalny mechanizm wtyczek. Podobnie definiuje się środowisko symulacji. Gazebo zawiera wbudowane cztery popularne biblioteki fizyczne: ODE, Bullet, Simbody, DART, niezależne od implementacji samej warstwy wizualnej.

W bibliotece Gazebo dostępne są gotowe modele, a także opis tworzenia własnych. Oznosi się to również modeli czujników LIDAR i Kinect. Gazebo działać może niezależnie, zazwyczaj jednak wykorzystywane jest jako narzędzie zintegrowane z systemem ROS. Zaletą Gazebo jest bogate API, dzięki któremu, poprzez mechanizm wtyczek, możliwa jest implementacja zaawansowanych funkcji modeli i samej symulacji [22].

4.3 RVIZ

RVIZ to narzędzie wbudowane w środowisko ROS, służące do wizualizacji aktualnego stanu systemu. Program subskrybuje zadane przez użytkownika tematy i wyświetla dane w sposób zależny od rodzaju wiadomości. Umożliwia wyświetlenie pozycji robota, map kosztu, trajektorii ruchu, chmury punktów, tzw. znaczników i innych.

4.4 OpenCV

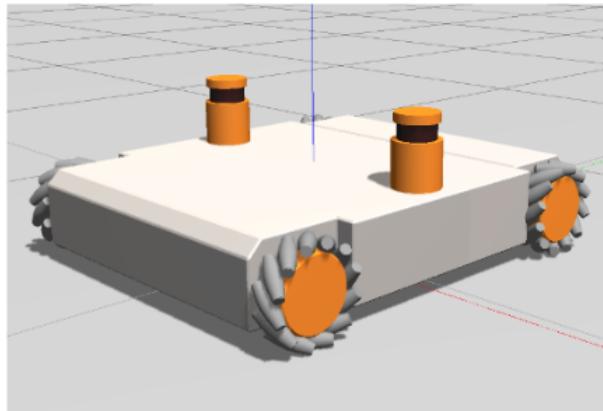
OpenCV to biblioteka służąca do przetwarzania obrazów, zawierająca implementacje ponad stu najczęściej wykorzystywanych algorytmów. Główne funkcje biblioteki dzieli się na następujące moduły: *cv* - funkcje główne, *imgproc* - przetwarzane obrazu, *highgui* - tworzenie interfejsu, *object detect* - detekcja obrazów, *ml* - uczenie maszynowe [23].

4.5 Dlib

Dlib jest biblioteką zawierającą implementację algorytmów uczenia maszynowego. Pozwala m.in. na tworzenie klasyfikatorów i ich wykorzystanie w celu detekcji obiektów [24].

4.6 Symulator bazy mobilnej

Mimo istnienia rzeczywistej bazy dookólnej, w niniejszej pracy podjęto decyzję o wykorzystaniu stworzonego wcześniej przez R. Świątkiewicza symulatora [14]. Symulator ten to dokładne odwzorowanie modelu platformy, zaimplementowany w środowisku symulacyjnym Gazebo [4.1]. Zamodelowano korpus platformy, koła szwedzkie, silniki, enkodery, skandery LIDAR oraz jednostkę inercyjną i zapewniono zbliżone do pracy rzeczywistego robota działanie.



Rysunek 4.1: Model platformy w programie Gazebo [14]

W toku prac powstał model kinematyczny i dynamiczny platformy. Poszczególne elementy zdefiniowano w plikach SDF, a następnie scalono do jednego modelu całej platformy. Zaimplementowano możliwość odczytu z dometrii robota i umożliwiono zadawanie mu prędkości liniowej i kątowej. Możliwe jest to dzięki istnieniu modułów odpowiadających za integrację modelu z systemem ROS. Symulator publikuje wiadomości o odometrii, wynikach pomiarów sensorów LIDAR i jednostki inercyjnej korzystając z infrastruktury środowiska ROS, natomiast specjalny moduł nasłuchuje danych o zadanej prędkościach v_x, v_y, v_{theta} , które są przeliczane na odpowiednie prędkości poszczególnych kół platformy poprzez równanie kinematyki odwrotnej. Równanie kinematyki odwrotnej przedstawia równanie (4.1). Symulator posiada również możliwość ręcznego (za pomocą klawiatury i myszy komputera) zadawania prędkości.

$$\begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ \omega_4 \end{bmatrix} = \frac{1}{r} \begin{bmatrix} 1 & -1 & \frac{a+b}{2} \\ 1 & 1 & -\frac{a+b}{2} \\ 1 & -1 & -\frac{a+b}{2} \\ 1 & 1 & \frac{a+b}{2} \end{bmatrix} \begin{bmatrix} v_y \\ v_x \\ \omega_z \end{bmatrix} \quad (4.1)$$

Równanie kinematyki odwrotnej platformy [14]

Pierwotnie symulator nie przewidywał istnienia sensora Kinect, o czym mowa w implementacji.

Z praktycznego punktu widzenia zastosowanie symulatora pozwala na szybszą implementację systemów mających działać z platformą. W przypadku bezpośredniego działania na robocie, należałoby przede wszystkim zapewnić jego bezpieczeństwo na każdym etapie tworzenia systemu. Sama rekompilacja kodu i ponowne uruchomienie platformy byłoby procesem znacznie bardziej czasochłonnym, gdzie w przypadku symulatora można to robić nieustannie. Zdecydowanie ułatwione jest prototypowanie, w trakcie prac można było przetestować wiele rozwiązań przed wybraniem ostatecznego. W przypadku tworzenia i testów podsystemów detekcji człowieka, ktoś nieustannie musiałby fizycznie znajdować się przy robocie, co jest praktycznie niemożliwe.

Z założenia natomiast system powstały na bazie symulacji nie różni się wiele od tego, który ostatecznie będzie mógł zostać uruchomiony na robocie.

Rozdział 5

Implementacja

W implementacji systemu będącego tematem niniejszej pracy można wyróżnić trzy składowe: **przygotowanie środowiska symulacyjnego**, **implementację modułu nawigacyjnego** oraz **implementację modułu detekcji ludzi**. Każdy z nich jest autonomiczny pod tym względem, że spełnia określone zadania i wymaga określonych danych z pozostałych modułów, ale system nie jest ograniczony do korzystania z konkretnego typu detektora, czy konkretnego modelu robota. Ważne jedynie by dane pochodzące z modułu miały konkretną postać. Jest to istotne ze względu na ewentualny dalszy rozwój projektu.

5.1 Środowisko symulacyjne

Do realizacji niniejszej pracy wykorzystano symulator platformy dookółnej [14]. Model platformy przygotowany został w środowisku symulacyjnym Gazebo i zintegrowany ze środowiskiem ROS. Sterowanie modelem i odbieranie rozsyłanych przez niego danych odbywa się poprzez mechanizm wiadomości środowiska ROS. Wiadomości wykorzystywane przez model platformy prezentuję tabelę [5.1].

Nazwa tematu	Zawartość wiadomości	Opis
<i>omnivelma/pose</i>	<i>geometry_msgs/PoseStamped</i>	pozycja i orientacja platformy wraz z numerem sekwencyjnym oraz nazwą okna (układu współrzędnych według którego podawane są wartości)
<i>omnivelma/twist</i>	<i>geometry_msgs/TwistStamped</i>	prędkość liniowa i kątowa platformy, numer sekwencyjny i nazwa okna
<i>odom</i>	<i>nav_msgs/Odometry</i>	scalone dane z <i>omnivelma/pose</i> i <i>omnivelma/twist</i>
<i>omnivelma/vels</i>	<i>omnivelma_msgs/Vels</i>	dane o prędkości kątowej każdego z kół platformy
<i>omnivelma/kinect_rotation</i>	<i>std_msgs::Float64</i>	absolutny kąt (w radianach) pomiędzy globalną osią x a osobą, w stronę której powinien obrócić się czujnik kinect
<i>monokl_l/scan</i>	<i>sensor_msgs/LaserScan</i>	odczyt pomiarów lewego czujnika laserowego
<i>monokl_r/scan</i>	<i>sensor_msgs/LaserScan</i>	odczyt pomiarów prawa czujnika laserowego

Tablica 5.1: Wiadomości środowiska ROS wykorzystywane przez model platformy

Model platformy przygotowany przez R. Świątkiewicza [14] został zaimplementowany w pliku o formacie SDF, który następnie interpretowany jest przez środowisko symulacyjne Gazebo. Działanie modelu zaimplementowano używając mechanizmu wtyczek, wykorzystującego interfejs Gazebo.

Aby nadać prędkość liniową lub kątową platformie należy nadać wiadomość na temat *omnivelma/vels*. Temat ten operuje na zdefiniowanych przez autora wiadomościach typu

omnivelma_msgs/Vels zawierających cztery liczby zmienoprzecinkowe reprezentujące prędkości kątowe każdego z czterech kół platformy. Aby zadać prędkości w formie v_x , v_y , w_z , który będzie wykorzystywany w systemie, powstał moduł przyjmujący zadane prędkości w formie prędkości liniowych i kątowych na temacie *cmd_vel* zawierającym wiadomości typu *geometry_msgs/Twist*, przeliczający je na prędkość poszczególnych kół (korzystając z równania kinematyki odwrotnej (4.1)), a następnie nadający je na temat *omnivelma/vels* odczytywany przez model platformy. Zadane prędkości utrzymywane są do czasu zadania nowych.

Pierwotny model nie posiadał implementacji czujnika Kinect, dlatego w toku prac dodano jego obsługę w symulacji. Czujnik Kinect nie jest bezpośrednio dostępny jako gotowy model w środowisku Gazebo, jednak producent simulatora [25] prezentuje sposób jego implementacji. Model (jego reprezentacja graficzna oraz własności fizyczne) został zapisany w formacie SDF. W pliku znajdują się również informacje o typie czujnika. Gazebo dostarcza wtyczki *libgazebo_ros_openni_kinect.so*, która imituje działanie rzeczywistego Kinecta, wykorzystującą bibliotekę *OpenNI* [26]. *OpenNI* integruje działanie czujnika Kinect z systemem ROS, rozsyłając dane o obrazie RGB, głębi i chmurę punktów na odpowiednich tematach środowiska ROS. Dzięki *libgazebo_ros_openni_kinect.so* model czujnika Kinect rozsyła pomiary z użyciem takich samych typów danych i na takich samych tematach środowiska ROS co rzeczywiste urządzenie. Poniżej znajduję się najistotniejsza część pliku SDF, ukazująca sposób implementacji, a także możliwe do manipulacji właściwości modelu.

Kod 5.1: Część pliku *kinect.sdf* zawierający implementację modelu czujnika Kinect

```

1 <sensor name="camera" type="depth">
2 <update_rate>20</update_rate>
3 <camera>
4   <horizontal_fov>1.094321441</horizontal_fov>
5   <image>
6     <width>640</width>
7     <height>480</height>
8     <format>R8G8B8</format>
9   </image>
10  <clip>
11    <near>0.4</near>
12    <far>6.5</far>
13  </clip>
14 </camera>
15 <plugin name="camera_controller" filename="libgazebo_ros_openni_kinect.so">
16   <baseline>0.2</baseline>
17   <cameraName>camera</cameraName>
```

```
18 <imageTopicName>rgb/image_raw</imageTopicName>
19 <depthImageTopicName>depth/image_raw</depthImageTopicName>
20 <cameraInfoTopicName>rgb/camera_info</cameraInfoTopicName>
21 <depthImageInfoTopicName>depth/camera_info</depthImageInfoTopicName>
22 <pointCloudTopicName>depth/points</pointCloudTopicName>
23 <frameName>camera_link</frameName>
24 <pointCloudCutoff>0.4</pointCloudCutoff>
25 <hackBaseline>0.07</hackBaseline>
26 <distortionK1>0</distortionK1>
27 <distortionK2>0</distortionK2>
28 <distortionK3>0</distortionK3>
29 <distortionT1>0</distortionT1>
30 <distortionT2>0</distortionT2>
31 <focalLength>0</focalLength>
32 <hackBaseline>0</hackBaseline>
33 </plugin>
34 </sensor>
```

Parametr *sensor* definiuję typ czujnika, *camera* definiuję jaki obraz będzie generował model czujnika, natomiast *plugin* zawiera parametry wtyczki imitującej działanie biblioteki *OpenNI* - nazwy tematów środowiska ROS na których przesyłane są obrazy i chmura punktów, czy poziom zakłóceń.

Bibliografia

- [1] T. Kruse, A. K. Pandey, R. Alami, and A. Kirsch, "Human-aware robot navigation: A survey," *Robotics and Autonomous Systems*, vol. 61, pp. 1726–1743, 2013.
- [2] E. T. Hall, *The Hidden Dimension*. Anchor Books, 1966.
- [3] "E. T. Hall - Proxemics /Personal Space in Different Cultures." <https://laofutze.wordpress.com/category/e-t-hall-2/>.
- [4] A.G. Mateus, "Human-aware navigation in networked robot systems." Instituto Superior Tecnico.
- [5] D. Vasquez, P. Stein, J. Rios-Martinez, A. Escobedo, A. Spalanzani, and C. Laugier, "Human aware navigation for assistive robotics,"
- [6] P. Althaus, H. Ishiguro, T. Miyashita, and H. Christensen, "Navigation for human–robot interaction tasks," *ICRA IEEE*, 2004.
- [7] T. Weerakoon, K. Ishii, and A. A. F. Nassiraei, "An artificial potential field based mobile robot navigation method to prevent from deadlock," *Journal of Artificial Intelligence and Soft Computing Research*, vol. 5, no. 3, pp. 189–203, 2015.
- [8] Y. K. Hwang and N. Ahuja, "Gross motion planning: a survey," *ACM Computing Surveys (CSUR)*, vol. 24, no. 3, pp. 219–291, 1992.
- [9] P. K. Mohanty and D. R. Parhi, "Controlling the motion of an autonomous mobile robot using various techniques: a review," *Journal of Advance Mechanical Engineering*, vol. 1, no. 1, pp. 24–39, 2013.
- [10] O. Brockand and O. Khatib, "High-speed navigation using the global dynamic window approach," *Robotics Laboratory, Department of Computer Science Stanford University, Stanford; International Conference on Robotics and Automation Detroit. Michigan*.

- [11] E. A. Sisbot, L. F. Marin-Urias, R. Alami, and T. Siméon, "A human aware mobile robot motion planner," *IEEE Transactions on Robotics*, vol. 23, no. 5, 2007.
- [12] D. A. V. Govea, T. Fraichard, and C. Laugier, "Growing hidden markov models: An incremental tool for learning and predicting human and vehicle motion," *International Journal of Robotics Research*, vol. 28, 2009.
- [13] R. Kirby, "Social robot navigation," *Theses and Dissertations at Research Showcase, Paper 552*, 2010.
- [14] R. Świątkiewicz, "Symulacja dookólnej bazy mobilnej," 2018. Praca dyplomowa inżynierska, Politechnika Warszawska.
- [15] Microsoft, "Project Natal 101." <https://web.archive.org/web/20120121223600/http://download.microsoft.com/download/A/4/A/A4A457B3-DF5D-4BF2-AD4E-963454BA0BCC/ProjectNatalFactSheetMay09.zip>.
- [16] Microsoft, "Kinect for windows sensor components and specifications." <https://msdn.microsoft.com/en-us/library/jj131033.aspx>.
- [17] OpenKinect, "Openkinect protocol documentation." https://openkinect.org/wiki/Protocol_Documentation#Control_Commands;a=summary.
- [18] H. England, "3d laser scanning for heritage. advice and guidance to users on laser scanning in archaeology and architecture." www.english-heritage.org.uk.
- [19] W. Garage, S. A. I. Laboratory, and O. Robotics, "Ros - robot operating system." <http://www.ros.org>.
- [20] P. Rabiński, "Vision based gesture-driven human-robot interface," 2013. Praca dyplomowa inżynierska, Politechnika Warszawska.
- [21] "Format sdf." <http://sdformat.org/>.
- [22] O. Robotics, "Symulator gazebo." <http://gazebosim.org/>.
- [23] "Opencv library." https://docs.opencv.org/master/d9/df8/tutorial_root.html/.
- [24] D. E. King, "Dlib library." <http://dlib.net/>.

- [25] O. Robotics, "Model czujnika kinect w środowisku gazebo." http://gazebosim.org/tutorials?tut=ros_depth_camera&cat=connect_ros.
- [26] "Openni framework." <http://openni.ru/openni-sdk/index.html>.