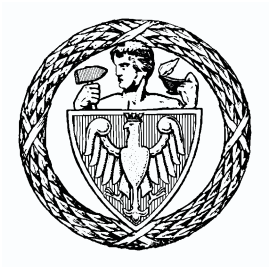
Opracowanie i implementacja zdecentralizowanej struktury sterowania zespołem heterogenicznch robotów mobilnych

Wojciech Szynkiewicz, Andrzej Rydzewski,
Marek Majchrowski, Piotr Trojanek, Cezary Zieliński

Sprawozdanie z wykonania 1-ego etapu prac w ramach grantu MEiN
nr 3T11C038 29

Raport Instytutu Automatyki i Informatyki Stosowanej nr 06-06
Warszawa, Czerwiec 2006



Politechnika Warszawska
Instytut Automatyki i
Informatyki Stosowanej
ul. Nowowiejska 15/19
00–665 Warszawa
tel/fax (48) 022 – 825 37 19

Spis treści

Rozdział 1

Ogólna struktura układu sterowania zespołem heterogenicznych robotów mobilnych

1.1 Systemy wielorobotowe

Zagadnienia dotyczące systemów wielorobotowych zwanych też wieloagentowymi (tutaj zwanymi agentami upostaciowionymi) jest obecnie w robotyce jednym z intensywnie rozwijanych obszarów badawczych [13, 17, 21, 39, 55]. Szczególnie intensywne badania są prowadzone w obszarze robotyki mobilnej [9, 10, 11, 20, 22, 24, 25, 32, 35, 49]. Jeśli wiele robotów porusza się w tej samej przestrzeni roboczej ich ruchy muszą być koordynowane (harmonizowane, jeśli nie ma nadrzędnego koordynatora) w celu uniknięcia wzajemnych kolizji i zakleszczeń. Złożoność problemu planowania działań i sterowania robotami zależy od bardzo wielu czynników m.in. liczby i typu robotów, stopnia ich współpracy, rodzaju środowiska (statyczne/dynamiczne, znane/nieznane) itd. [2, 10, 17].

Zastosowanie systemu wielorobotowego rozumianego zazwyczaj jak zespół wielu współpracujących robotów stwarza możliwości wykonania zadań, które są niewykonalne dla pojedynczego robota [10, 17]. Jednym z częściej podawanych takich przykładów jest zadanie wymagające obecności robota w dwóch oddalonych miejscach w tej samej chwili czasu [18]. W grupach składających się z różnych robotów można wykorzystać możliwości bardziej zaawansowanych robotów np. wyposażonych w precyzyjne czujniki np. dalmierze laserowe do wspomagania prostszych robotów przy samo-lokalizacji i nawigacji w naturalnym środowisku [9, 14, 28, 42, 51]. Wykorzystanie zespołu robotów do wykonania różnorodnych zadań powoduje nie tylko ilościowy wzrost wydajności, lecz również zamianę jakościową – polegającą na istotnym poszerzeniu klasy potencjalnych zastosowań. Odnosi się to w szczególności do wszelkiego rodzaju robotów mobilnych (w tym manipulatorów mobilnych, pojazdów autonomicznych, maszyn kroczących) wykorzystywanych w zadaniach inspekcyjnych, transportowych, usługowych, itp. Jest to szczególnie istotne w systemach heterogenicznych, gdzie roboty różnią się między sobą nie tylko budową, ale także funkcjami, które mogą wykonać [35]. Wśród podstawowych powodów tworzenia systemów wielorobotowych wymienia się [2, 10, 18, 21, 39, 29] :

- możliwość wykonania zadań zbyt trudnych (lub wręcz niewykonalnych) dla pojedynczego robota oraz znaczący wzrost wydajności;
- wykorzystanie różnych funkcjonalności (umiejętności) robotów wynikających z róż-

nej budowy i wyposażenia (efektory, czujniki);

- zbudowanie i wykorzystanie kilku prostych robotów może być łatwiejsze niż stosowanie jednego uniwersalnego, ale za to bardzo skomplikowanego robota do realizacji różnorodnych zadań;

- zwiększenie odporności i niezawodności (roboty mogą nawzajem siebie zastępować, jeśli jeden lub więcej z nich ulegnie awarii);

W literaturze podaje się kilka podstawowych grup zadań (zadań testowych) dla współpracujących robotów mobilnych [10, 17, 21]:

- *Kierowanie ruchem*: Wiele robotów porusza się we wspólnym środowisku i unika wzajemnych kolizji. Planowanie ruchu może być rozpatrywane jako problem *konfliktu dostępu do zasobu*, który może być rozwiązywany przez ustalenie zasad ruchu (kodeks), priorytetów, struktur komunikacyjnych [22].

- *„Pchanie pudła"/kooperatywna manipulacja oraz transportowanie obiektów*: Zadanie polega na manipulowaniu dużymi obiektami wspólnie przez kilka robotów np. pchaniu dużego pudła [11,41,58]. Pchanie pudła zazwyczaj odbywa się na płaszczyźnie, podczas gdy transport obiektów wymaga podnoszenia i przenoszenia przedmiotów zespołowo przez kilka robotów co istotnie zwiększa trudność zadania. Wspólne transportowanie przedmiotów jest przykładem zadania testowego dla takich zagadnień jak koordynacja ruchów oraz utrzymanie ścisłego szyku (formacji) robotów [27, 31, 34, 50, 56].

- *Przeszukiwanie i pokrycie obszaru*: Grupa robotów poszukuje w środowisku określonych obiektów. Można wymienić wiele przykładów tego typu zadań m.in. sprzątanie, poszukiwanie min, zbieranie owoców/warzyw [35]. Może być ono wykonywane przez każdego robota niezależnie albo roboty współpracują przy jego wykonaniu (jest to sprawdzian, czy ewentualna współpraca robotów może przyspieszyć realizację zadania). Pokrycie obszaru wymaga efektywnych sposobów sprawdzenia całego obszaru przez grupę robotów [1, 13]. Typowymi przykładami zadania pokrycia są malowanie dużych powierzchni, koszenie trawników, odśnieżanie, rozminowanie terenu, itp.

- *Obserwacja wielu celów*: Zadanie to wymaga działania grupy robotów w celu wykrycia i śledzenia wielu poruszających się obiektów (celów) [8, 15]. Typowymi aplikacjami są tutaj systemy bezpieczeństwa, w których wymagany jest nadzór i śledzenie ruchomych obiektów. Poza grupą robotów mobilnych, w tego typu zastosowaniach wykorzystuje się sieci czujników. Zazwyczaj takie czujniki charakteryzują się małymi możliwościami obliczeniowymi i niewielkim zasięgiem i ich współdziałanie z ruchomymi robotami może istotnie poszerzać możliwości całego systemu [14, 15].

- *Piłka nożna robotów*: Jest przykładem bardzo wymagającego zadania testowego dla drużyny robotów, gdzie jest wymagane planowanie i sterowanie ruchem robotów w czasie rzeczywistym, w środowisku zawierającym wiele ruchomych przeszkód (innych zawodników). Rozwiązania wymagają takie problemy podział zadań i planowanie działań grupowych, takich jak strategia gry zespołu, możliwość pełnienia różnych ról przez tego samego zawodnika [5, 55].

Stosunkowo nowym, ale potencjalnie bardzo obiecującym, obszarem zastosowań zespołów robotów (w szczególności robotów autonomicznych) są szeroko rozumiane zadania usługowe. Dla szerokiej klasy zadań usługowych zastosowanie kilku robotów współpracujących może przyspieszyć wykonanie lub być wręcz niezbędne do realizacji zadania [60].

Roboty usługowe zazwyczaj działają w niestrukturalnym środowisku dynamicznym, gdzie oprócz robotów mogą być inne ruchome obiekty np. ludzie, których trajektorie ruchu są zazwyczaj nieznane i nieprzewidywalne. W wielu zadaniach usługowych konieczny jest bezpośredni kontakt robotów z człowiekiem. W takiej sytuacji planowanie działań/ruchu robota czy też zespołu robotów jest bardzo złożonym problemem, którego rozwiązanie wymaga stosowania zaawansowanych metod i algorytmów.

Generalnie, planowanie działań i sterowanie zespołem robotów jest znacznie bardziej złożone niż planowanie ruchu jednego robota i tylko nieliczne metody opracowane dla pojedynczego robota można uogólnić na przypadek wielorobotowy. Przykładowo, dla grupy ściśle współpracujących robotów (wymagana jest ścisła koordynacja działań/ruchów robotów) już znalezienie dopuszczalnych trajektorii ruchu zespołu robotów jest bardzo trudnym zadaniem.

Klasyczne podejście z planowaniem trajektorii ruchu i późniejszym jej odtwarzaniem jest dla robotów autonomicznych działających w nieznanym otoczeniu całkowicie nieprzydatne. Stosowane są za to z pewnym powodzeniem metody mające inspiracje biologiczne (sterowanie behawioralne, reaktywne) oraz w zakresie uczenia metody z obszaru sztucznej inteligencji – np. systemy klasyfikujące, uczenie ze wzmacnianiem, Q-uczenie, itp. Metody te są również stosowane do sterowania grupami robotów. Inspiracją dla tych podejść są przykłady zaczerpnięte z przyrody takie jak stada ptaków, ławice ryb czy też mrowiska lub ule. Doświadczenia eksperymentalne pokazują, że wiele prostych robotów o niskim poziomie „inteligencji maszynowej" może ze sobą współpracować w celu realizacji nawet złożonego zadania [2, 4, 31, 35].

Przy planowaniu działań robotów mobilnych kluczowym zagadnieniem staje się wykorzystanie pochodzącej z różnorodnych czujników informacji o środowisku, która zwykle jest cząstkowa, obarczona niepewnością, a czasami błędna [8, 9, 14, 43, 54]. W zależności od rodzaju jak i zastosowań robota może on być wyposażony w czujniki dotykowe, zbliżeniowe (indukcyjne i na podczerwień), sił i momentów, ultradźwiękowe, wizyjne, globalnego położenia (GPS, DGPS, kompasy, żyroskopy). Efektywne wykorzystanie informacji z wielu różnorodnych czujników do planowania ruchu i sterowania robotów w czasie rzeczywistym jest otwartym problemem badawczym [17, 21]. Rozwiązania wymagają m.in. takie problemy jak planowanie ruchu robotów działających we wspólnej przestrzeni roboczej, komunikacja między robotami, harmonizacja działań wielu robotów wykonujących wspólne zadanie i nadzór na systemem [9, 14, 17, 36]. Jest to szczególnie złożony problem, gdy nie istnieje jednostka nadrzędna – koordynator [22, 27, 41, 46].

W literaturze podawane są różne definicje systemów wielorobotowych [10, 18, 39]. O *systemie wielorobotowym* możemy już mówić, jeśli mamy do czynienia z co najmniej dwoma robotami i występują interakcje między nimi wynikające zazwyczaj z działania we wspólnej przestrzeni roboczej. Bardzo często definiując system wielorobotowy przyjmuje się, że jest możliwa (przynajmniej potencjalnie) współpraca robotów. O tym czy roboty będą w stanie wykonać wspólnie choćby najprostszą czynność decydują możliwości ich układów sterowania [2, 4, 22, 60].

1.2 Architektury sterowania robotów mobilnych

Można wyróżnić dwie podstawowe cechy, które charakteryzują każdy system wielorobotowy. Są to *struktura organizacyjna* i *sposób komunikacji* [10, 17]. Pierwsza określa strukturę organizowania współpracy robotów, druga zaś sposób wymiany informacji między robotami. Ze względu na strukturę można wyróżnić dwa główne podejścia do problemu planowania ruchu oraz sterowania zespołem robotów:

- *Systemy scentralizowane o strukturze globalnej*, w których zadania do wykonania oraz sposób współpracy jest narzucony robotom przez jednostkę nadrzędną (koordynatora) lub operatora [20, 58]. Mamy tutaj zatem strukturę hierarchiczną.
- *Systemy rozproszone o strukturze lokalnej*, w których nie ma żadnej jednostki centralnej, zaś możliwości współpracy wynikają z reguł wbudowanych poszczególnym robotom [40, 46, 49].

Struktura scentralizowana nie zawsze jest odpowiednim rozwiązaniem. Wyznaczenie optymalnych planów jest zazwyczaj bardzo kosztowne ze względu na nakład obliczeń. Ogranicza to liczebność zespołu, dla którego jednostka nadrzędna jest w stanie obliczyć taki plan w wymaganym czasie. Warunkiem koniecznym poprawnego działania całości jest zapewnienie szybkich i pojemnych łączy komunikacyjnych między koordynatorem a pozostałymi robotami. W przypadku utraty łączności lub awarii koordynatora cały system przestaje działać. Dla robotów o dużym stopniu autonomiczności, które działają w mało znanym lub nieznanym otoczeniu zazwyczaj lepszym rozwiązaniem jest podejście zdecentralizowane. Nie ma tutaj jednostki nadrzędnej, a poszczególne roboty komunikują się bezpośrednio między sobą, jeśli jest to konieczne. Każdy robot działa w dużym stopniu niezależnie i planuje swe działania korzystając z informacji o lokalnym otoczeniu z własnych czujników. Umożliwia to szybką reakcję na zmiany w otoczeniu i zmniejsza wymagania komunikacyjne z innymi robotami. Uszkodzenie pojedynczego robota nie powoduje awarii całego systemu. System jako całość jest bardziej odporny i znacznie bardziej elastyczny ze względu na modyfikacje struktury. Podstawową wadą podejścia całkowicie zdecentralizowanego jest to, iż planowanie na podstawie lokalnej informacji może prowadzić do całkowicie nieoptymalnych rozwiązań. W rezultacie zadanie może być wykonywane przez zespół robotów w sposób wysoce nieefektywny lub w skrajnym przypadku może okazać się niewykonalne.

Do implementacji struktury zdecentralizowanej najczęściej stosuje się sterowanie behawioralne [2, 4]. Sterowanie jest podzielone na zbiór celowych działań – *zachowań*, które mogą wykonywać się równolegle [2, 59]. Korzystając z selektywnej informacji uzyskiwanej z czujników każde takie *zachowanie* generuje sterowanie dla robota zgodnie z lokalną funkcją celu. Zachowania z różnymi i prawdopodobnie niezgodnymi funkcjami celu mogą powodować konfliktowe działania (akcje), które pozornie są nie do pogodzenia. Konieczne jest więc stworzenie efektywnych mechanizmów koordynacji zachowań. Wykorzystuje się tutaj dwie klasy mechanizmów koordynacji podejścia bazujące na arbitrażu oraz łączeniu (fuzji) zleceń od różnych zachowań tworzących sterowanie, które jest wynikiem uzgodnienia np. przez „głosowanie". Wybór odpowiedniej struktury sterowania nadrzędnego dla systemu wielorobotowego zależy od kilku czynników takich jak: zadanie jakie roboty mają do wykonania, od rodzaju robotów i stopnia ich autonomiczności oraz środowiska

w którym one działają. W wielu praktycznych rozwiązaniach proponuje się struktury hybrydowe, które próbują łączyć pozytywne cechy obu powyższych podejść [2]. Wyróżnia się dwa podstawowe sposoby komunikacji:

- *jawna* – gdy poszczególne jednostki przesyłają do siebie informacje o swoim stanie,
- *niejawna* – nie ma żadnej wymiany bezpośredniej wymiany informacji, wiedza o innych robotach jest uzyskiwana na podstawie obserwacji otoczenia [31].

W przypadku jawnej komunikacji rozważane są takie problemy jak: rodzaj i struktura komunikacji, zasięg, przepustowość łączy, itp. [18, 17, 45]. Obecnie duże zainteresowanie jest związane z tzw. sieciami robotów i czujników, gdzie komunikacja jest jednym z podstawowych problemów badawczych [1, 14].

Jednym z istotnych problemów badawczych jest zapewnienie komunikacji pomiędzy agentami wyposażonymi w bezprzewodowe urządzenia sieciowe, bez wykorzystywania zewnętrznej infrastruktury sieciowej (połączenie agentów siecią ad-hoc). Dostępne są dwa rodzaje urządzeń do komunikacji bezprzewodowej: wykorzystujące technologię Bluetooth oraz Wi-Fi. Poszczególni agenci poruszając się w środowisku wykonują zadania, komunikując się ze sobą droga radiową. Rozważony zostanie zarówno wariant wykorzystujący zewnętrzną infrastrukturę sieciową, jak i wariant bez takiej struktury, w związku z tym komunikacja pomiędzy węzłami, które nie znajdują się w bezpośrednim zasięgu będzie odbywać się za pośrednictwem innych węzłów. Mobilność agentów implikuje dynamicznie zmieniającą się topologię sieci i konieczność stosowania specjalizowanych mechanizmów komunikacji. Charakterystyka urządzeń nadawczo-odbiorczych zainstalowanych na agentach wymusza zastosowanie algorytmów zarządzania topologią z rodziny tzw. *neighbor-based*.

Innym intensywnie rozważanym problemem jest wykorzystanie Internetu do komunikacji między robotami oraz zdalnego sterowania zespołem robotów [20, 38].

Innym ważnym zagadnieniem dotyczącym systemów wielorobotowych jest problem przydziału zadań poszczególnym jednostkom [24, 25, 40, 60]. Wyróżnia się dwa podstawowe podejścia do problemu przydziału zadań w systemach rozproszonych bez koordynatora. Pierwsze z nich wykorzystuje prawa rynkowe i polega na negocjacjach – roboty negocjują ze sobą i podejmują „kontrakty" na wykonanie określonych zadań [60]. Drugie polega na modelowaniu zdolności robotów – roboty wykorzystują modele opisujące możliwości wykonania zadań przez inne roboty do określenia swoich „motywacji" do wykonania konkretnych zadań [25].

1.3 Funkcje sterownika robota mobilnego

Podstawowym założeniem przy projektowaniu układów sterowania tych robotów było przyjęcie modułowej struktury zarówno części sprzętowej jak i programowej sterowników, dzięki temu, w zależności od przyszłych potrzeb, ich rozbudowa nie będzie nastręczała większych problemów. Moduły składające się na system powinny mieć dobrze zdefiniowane interfejsy, metody komunikacji i ograniczenia jakim podlegają. Architektura sprzętowa i programowa sterownika ma hierarchiczną strukturę warstwową. Warstwa wykonawcza

realizuje bezpośrednią obsługę sprzętu, w tym sterowania silnikami oraz obsługę czujników. Warstwa decyzyjna realizuje zadania wyższego poziomu m.in. planowania działań, nawigacji robotem oraz komunikacji między robotami. Założono, że układ sterowania powinien umożliwiać autonomiczne działanie robota.

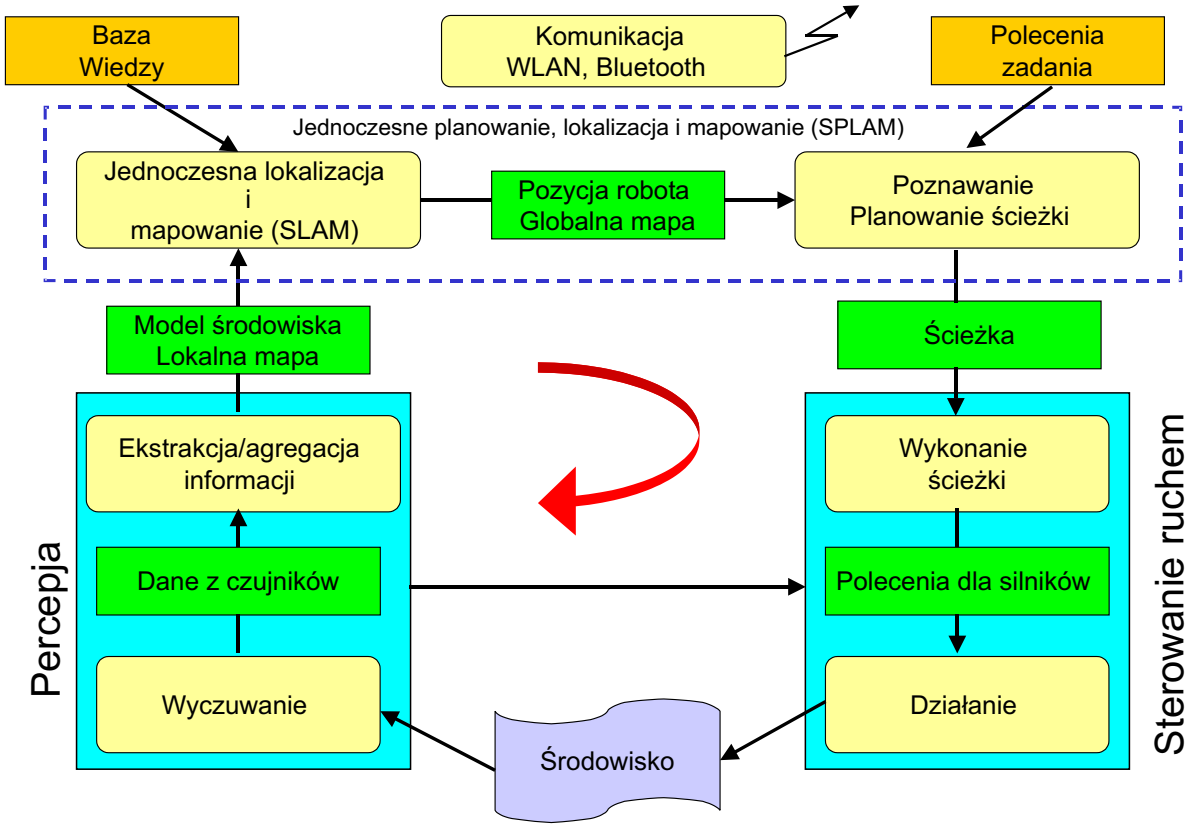
Przy sterowaniu robotami mobilnymi kluczowym zagadnieniem staje się wykorzystanie pochodzącej z różnorodnych czujników informacji o środowisku, która zwykle jest cząstkowa, obarczona niepewnością, a czasami błędna. W zależności od rodzaju jak i zastosowań robota może on być wyposażony w czujniki dotykowe, zbliżeniowe (np. na podczerwień), ultradźwiękowe, wizyjne, globalnego położenia (GPS, DGPS, kompasy, żyroskopy). Efektywne wykorzystanie informacji z wielu różnorodnych czujników do planowania ruchu i sterowania robotów w czasie rzeczywistym jest otwartym problemem badawczym. Do podstawowych funkcji sterownika zaliczamy realizację następujących zadań:

- lokomocję – sterowanie napędami (serwomechanizmy osi);
- przeliczniki kinematyki (proste i odwrotne zadanie kinematyki),
- podstawową obsługę czujników pomiarowych (inicjalizacja, odczyt i ewentualnie wstępne przetwarzanie danych pomiarowych);
- wykrywanie i unikanie kolizji (przy wykorzystaniu danych z czujników dotykowych, zbliżeniowych, itp.);
- samo-lokalizację – określenie pozycji robota (różne metody wykorzystujące dane z czujników odometrycznych, dalmierza laserowego, kamera wizyjnej);
- planowanie i generacja trajektorii ruchu (w tym rozwiązywanie zadań kinematyki);
- budowę modelu otoczenia robota (mapy);
- komunikację wewnętrzną (transmisja równoległa i/lub szeregowa synchroniczna przez specjalizowane łącza), zewnętrzną (radiowa sieć Ethernet).

Struktury danych dla robotów mobilnych obejmują:

- stan robota (pozycja, prędkość, typ, identyfikator, itp.)
- opis ruchu robota (ścieżka ruchu)
- stan środowiska (reprezentacja mapy otoczenia)
- dane z pomiarami czujników (odczyty z skanera laserowego, obrazy w postaci bitmapy, itp.)

Ogólną strukturę funkcjonalną oprogramowania autonomicznego robota mobilnego pokazano na rys.1.1



Rysunek 1.1: Struktura funkcjonalna oprogramowania sterownika robota

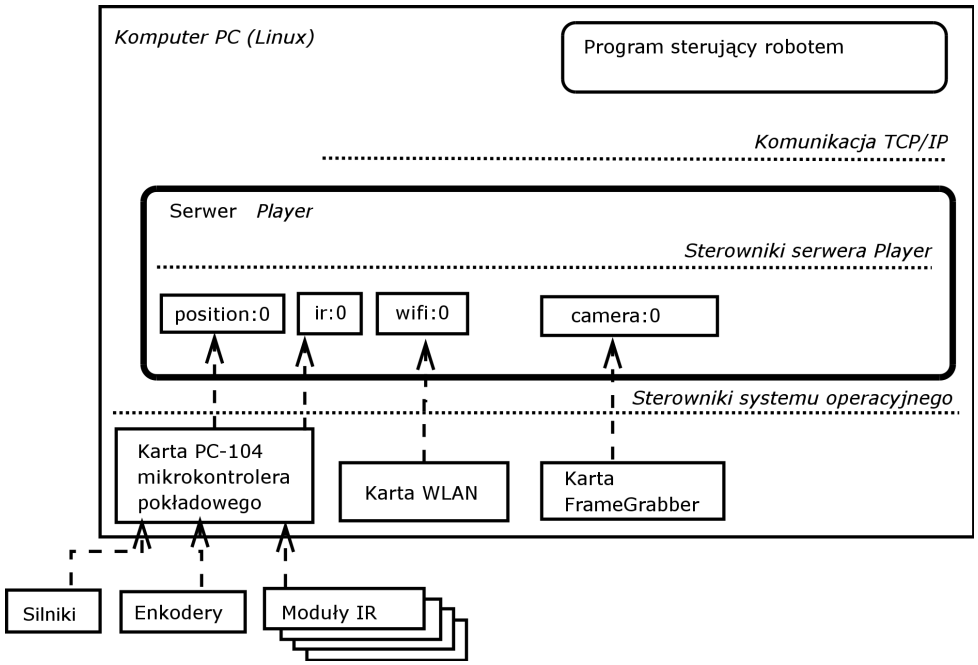
1.3.1 Opis sterownika robota mobilnego

Do sterowania robotem wykorzystana została struktura *Player/Stage*. Ogólny schemat struktury sterowania z wykorzystaniem *Playera* przedstawia rysunek 1.2. W jego centrum znajduje się wielowątkowy proces *Playera*. Na dole rysunku znajdują się rzeczywiste urządzenia, natomiast na samej górze znajdują się klienci.

Każdy klient połączony jest z serwerem *Playera* poprzez gniazda TCP. W przypadku gdy klient jest uruchomiony na tej samej maszynie co *Player*, wtedy połączenie jest po interfejsie lokalnym (*loopback*). W przeciwnym przypadku jest to fizyczne połączenie przez sieć.

Z drugiej strony *Player* łączy się z urządzeniami, poprzez swoje sterowniki, zazwyczaj poprzez łącze RS232, choć w przypadku niektórych sterowników (jak *festival*) połączenie to jest nawiązywane także poprzez sieć TCP. W przypadku sterownika robota mobilnego *Electron* połączenie to jest nawiązywane z modułem jądra *Linuxa*, który dalej obsługuje już dane żądanie.

Wewnątrz *Playera* różne wątki komunikują się wykorzystując globalną przestrzeń adresową. Każde urządzenie jest powiązane z buforem komend i buforem danych. Dostęp do tych buforów wykorzystuje mechanizm wzajemnego wykluczania. W ten sposób zapewniony jest asynchroniczny kanał komunikacyjny pomiędzy wątkami sterowników urządzeń a wątkami serwera odpowiedzialnymi za odbieranie oraz udostępnianie danych programom klienckim.



Rysunek 1.2: Struktura oprogramownia robota

Integracja robota z tym oprogramowaniem wymagała stworzenia sterowników (*drivers*) implementujących interfejsy już wcześniej zdefiniowane w systemie – zapewniające komunikację z układem napędu (sterowanie prędkościowe i pozycyjne) i czujnikami odległości, sterowanie serwomechanizmami modelarskimi oraz monitorowanie stanu naładowania akumulatorów w trakcie pracy robota.

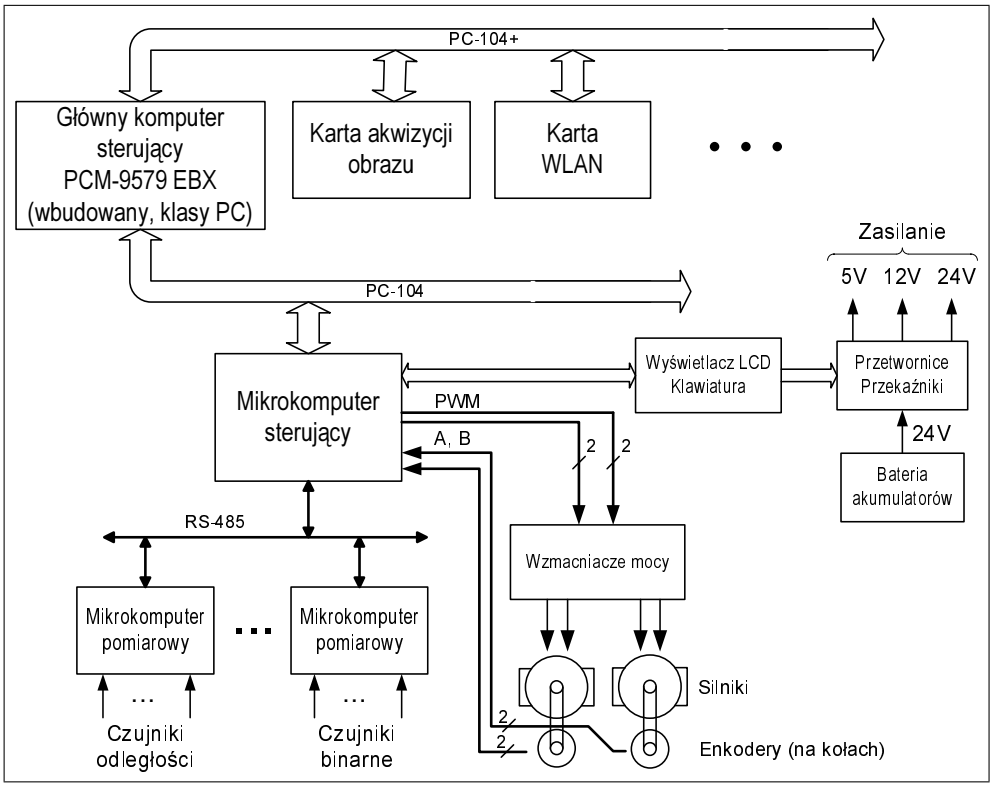
Rozdział 2

Adaptacja warstwy sprzętowej sterowników robotów do zadań wymagających współpracy robotów

Robot mobilny Elektron R1 jest urządzeniem autonomicznym, zdolnym do samodzielnego poruszania się w nieznanym terenie i do komunikowania się z innymi robotami lub komputerem zewnętrznym. W związku z tym jest on wyposażony w komputer sterujący o odpowiednio dużej mocy obliczeniowej, zdolny do realizacji zadań związanych z odometrią, rozpoznawaniem terenu, nawigacją itp. Robot jest przeznaczony do realizacji prac naukowo–badawczych oraz do dydaktyki. Jest wyposażany w różnego rodzaju czujniki i urządzenia do nawigacji (np. dalmierz laserowy, kamera), przy czym wyposażenie poszczególnych egzemplarzy robota jest różne. Dlatego układ sterowania robota ma strukturę otwartą, jest łatwy do rozbudowy i modyfikacji w celu przystosowania go do danej konfiguracji wyposażenia robota.

Schemat blokowy układu sterowania jest pokazany na rys. 2.1. Podstawowym elementem układu jest główny komputer pokładowy, który zarządza praca całego systemu, a także realizuje wszystkie funkcje sterowania wysokiego poziomu. Wykorzystano tu jednopłytkowy komputer PCM–9579 EBX firmy Advantech. Jest to komputer z procesorem Celeron lub Pentium, przeznaczony do wbudowania (embedded), wyposażony we wszystkie typowe elementy komputera klasy PC (grafika, dźwięk, Ethernet, sterowniki napędów dyskowych, portu równoległy i szeregowy RS–232, magistrala PCI), a także w elementy dodatkowe, typowe dla komputerów wbudowanych (magistrale PC–104 i PC–104+, złącze RS–485, złącze pamięci compact flash pracującej w trybie IDE). Magistrale PC–104 i PC–104+ (standard dla komputerów wbudowanych) umożliwiają łatwą rozbudowę i modyfikację układu sterowania, przez dokładanie kart rozszerzeń – zarówno typowych, fabrycznych, jak i prototypowych, własnych. Komputer pracuje pod kontrolą systemu operacyjnego Linux w dystrybucji Gentoo. W konfiguracji bazowej robota, do magistrali PC–104+ jest przyłączona bezprzewodowa karta sieciowa WIB230 firmy Elcard Oy. Jest to karta zgodna z standardem Ethernet 802.11b i 802.11g. Umożliwia ona transmisję danych z szybkością od 6 do 54 Mb/s (z automatycznym wykrywaniem szybkości). Umożliwia ona dostęp do sieci LAN za pośrednictwem punktów dostępowych (access point), a także na tworzenie sieci „ad–hoc"). Karta jest wykorzystywana do szybkiej komunikacji robota z otoczeniem – komputerami zewnętrznymi, a także innymi robotami.

W układzie sterowania robota wyposażonego w kamerę wysokiej jakości, do magistrali PC–104+ jest przyłączona karta akwizycji obrazy, a w układach innych robotów mogą być wykorzystywane inne specjalizowane karty zgodnie z potrzebami. Robot jest napę-

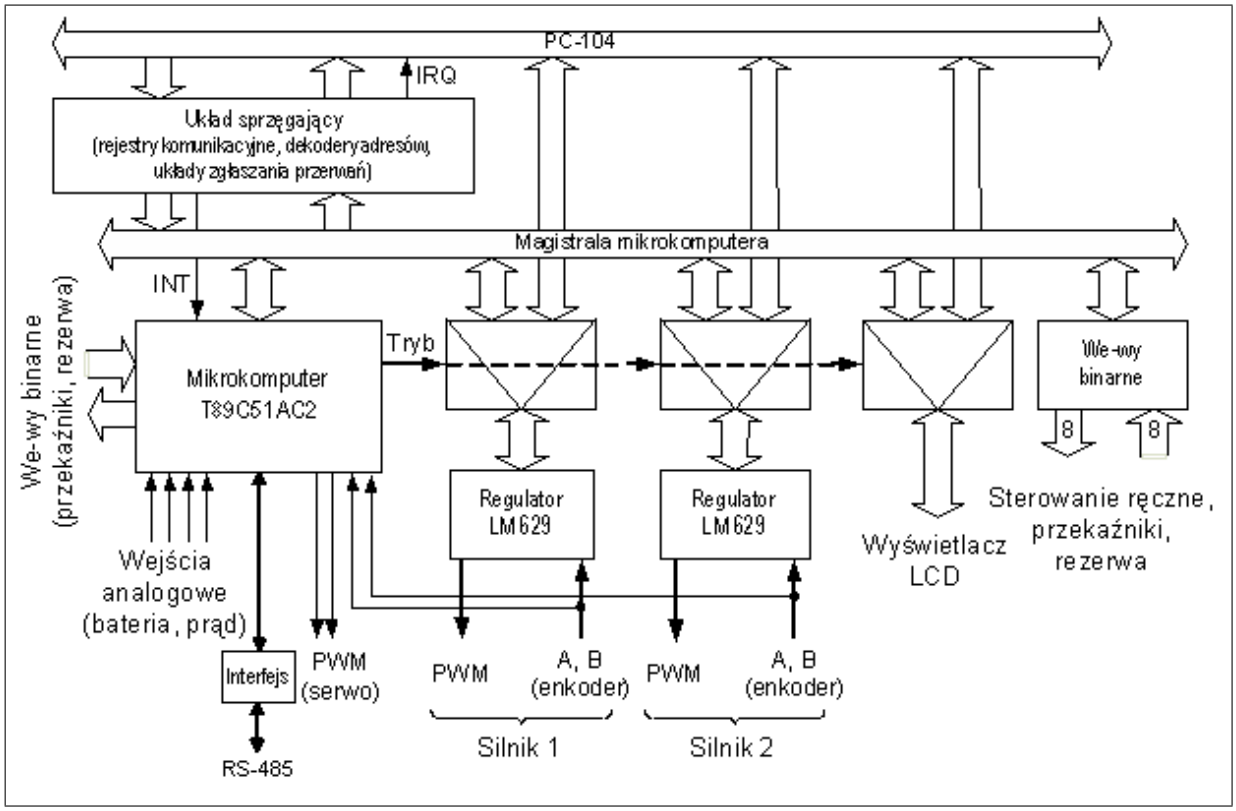


Rysunek 2.1: Struktura układu sterowania

dzany dwoma silnikami prądu stałego (o napięciu nominalnym 24 V i prądzie max. ok. 6 A) z przekładnią. W wersji podstawowej jest on wyposażony w 18 czujników odległości (prod. Sharp – działających z wykorzystaniem promieniowania podczerwonego): 5 dalekiego zasięgu (10 – 120 cm) i 13 krótkiego zasięgu (1 – 30 cm). Dodatkowe wyposażenie różnych egzemplarzy robota ma być różne, zależne od potrzeb. Bezpośrednie sterowanie tych wszystkich elementów wykonawczych robota (silniki, czujniki itp.), a także wykrywanie i obsługa sytuacji awaryjnych jest realizowana przez specjalizowany mikrokomputer sterujący, który komunikuje się z głównym komputerem pokładowym przez magistralę PC–104. Wszystkie czujniki są bezpośrednio obsługiwane przez specjalizowane mikrokomputery pomiarowe, przyłączone do szeregowej magistrali zbudowanej z wykorzystaniem standardu RS–485.

Schemat blokowy mikrokomputera sterującego jest pokazany na rys. 2.2. Centralnym jego elementem jest mikrokomputer jednoukładowy firmy Atmel T89C51AC2. Jest to 8–bitowy mikrokomputer z popularnej rodziny MCS–51, wykonany w statycznej technologii CMOS, dzięki czemu charakteryzuje się niskim zużyciem energii. Dzięki wewnętrznym blokom funkcjonalnym umożliwia on programową (bez dodatkowych elementów sprzętowych) realizację niektórych funkcji, np. pomiar napięć, wytwarzanie sygnałów PWM, obsługa komunikacji szeregowej. Natomiast do zadań sterowania silników napędowych są wykorzystane scalone regulatory PID – układy LM629 firmy National Semiconductor. Są to specjalizowane układy przeznaczone do sterowania silnikiem, które mogą realizować kilka typowych zadań sterowania: regulacja prędkości, realizacja zadanej trajektorii oraz regulacja położenia, wykonując cyfrowy algorytm regulacji PID. Układ mierzy pozycję i prędkość wykorzystując bezpośrednio sygnał z enkodera i wytwarza sterowanie w postaci sygnału PWM. Zadanie regulacji (pozycja zadana, prędkość zadana, przyspieszenie)

oraz parametry regulatora (wzmocnienia wszystkich członów, częstotliwość próbkowania, ograniczenie wartości członu całkującego) mogą być przekazywane programowo. Pozycja i prędkość (aktualne i zadane) mogą być w każdej chwili odczytane z układu. Wartości zadane pozycji i prędkości są zapisywane do układu (w jednostkach regulatora), a pozycja również odczytywana z 32-bitową rozdzielczością. Natomiast wartość prędkości jest odczytywana tylko z rozdzielczością 16-bitową (bardziej znaczące bity). Przy wartościach prędkości występujących w robocie taka rozdzielczość jest zdecydowanie za mała. Z tego powodu prędkość aktualna będzie dodatkowo mierzona przez mikrokomputer, przy wykorzystaniu jogo układów czasowo–licznikowych. W tym celu do mikrokomputera są doprowadzone sygnały z enkodera.

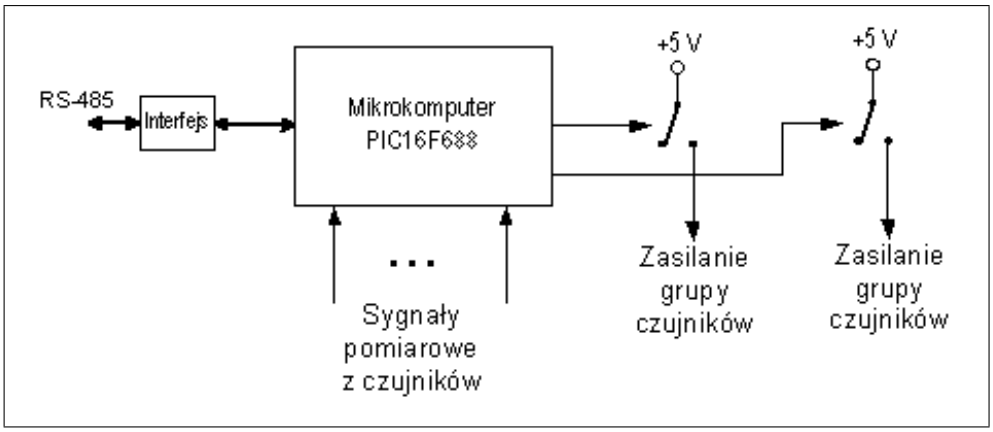


Rysunek 2.2: Schemat blokowy mikrokomputera sterującego

Komunikacja mikrokomputera z głównym komputerem pokładowym odbywa się za pomocą bloku sprzęgającego, przyłączonego z jednej strony do magistrali PC–104 komputera, a z drugiej strony do magistrali mikrokomputera. Układ sprzęgający zawiera m.in. dwa 16–bitowe rejestry poleceń, do zapisu przez komputer główny i do odczytu przez mikrokomputer, dwa (16–bitowy i 8–bitowy) rejestry stanu, do odczytu przez komputer główny i do zapisu przez mikrokomputer oraz układ sterowania przerwaniami do komputera głównego oraz do mikrokomputera z rejestrami stanu przerwań. Przesyłanie danych między komputerem głównym a mikrokomputerem jest sterowane przerwaniami. Zapisanie danych do jednego z rejestrów poleceń powoduje automatyczne ustawienie przerzutnika zgłoszenia przerwania do mikrokomputera – co jest sygnałem pojawienia się nowego polecenia. Przerzutnik ten jest automatycznie zerowany po odczytaniu przez mikrokomputer wszystkich bajtów polecenia (przy odczycie ostatniego bajtu). Przerzutnik jest jednym z elementów rejestru stanu przerwań i jego stan może być programowo sprawdzany przez

komputer główny. W ten sposób komputer główny może sprawdzić przed wysłaniem kolejnego polecenia, czy poprzednio wysłane zostało odebrane przez mikrokomputer. Analogicznie, wpisanie danych do jednego z rejestrów stanu powoduje automatyczne ustawienie przerzutnika zgłoszenia przerwania do komputera głównego, który to przerzutnik jest automatycznie zerowany po odczytaniu tego rejestru przez komputer główny, a stan tego przerzutnika (informujący o odczytaniu danych) może być sprawdzany przez mikrokomputer.

Jednym z podstawowych zadań mikrokomputera sterującego jest zbieranie danych o stanie czujników od mikrokomputerów pomiarowych, przyłączonych do magistrali szeregowej, zbudowanej na bazie standardu RS-485. Mikrokomputer sterujący będzie pełnił w tej strukturze funkcję układu typu master, układy pomiarowe będą typu slave, a protokół komunikacyjny będzie opierał się na zasadzie cyklicznego odpytywania mikrokomputerów sterujących przez mikrokomputer sterujący.



Rysunek 2.3: Schemat blokowy mikrokomputera pomiarowego

Zadaniem mikrokomputera pomiarowego jest obsługa czujników odległości firmy Sharp. Sygnałem wyjściowym z czujnika jest napięcie o wartości zależnej od odległości przedmiotu od czujnika, przy czym zależność ta jest nieliniowa. Schemat blokowy mikrokomputera pomiarowego jest pokazany na rys. 2.3. Jest to bardzo prosty układ, zbudowany na bazie mikrokomputera jednoukładowego PIC16F688 firmy Microchip. Mikrokomputer ten jest wykonany w statycznej technologii CMOS, w 14–końcówkowej obudowie, a zawiera m.in.: 8–kanałowy, 10–bitowy przetwornik analogowo-cyfrowy oraz układ transmisji szeregowej UART, Mikrokomputer może obsłużyć osiem czujników. Sygnały pomiarowe są przyłączone bezpośrednio do wejść analogowych mikrokomputera. Dodatkowo jest możliwość włączania i wyłączania napięcia zasilania dwóch grup czujników za pomocą klucza elektronicznego.

Rozdział 3

Opracowanie i implementacja algorytmu planowania ścieżek ruchu dla zespołu robotów mobilnych

Ogólnie, problem planowania ruchu to odpowiedź na pytanie: *„Dokąd idę i jak mam się tam dostać?”*. Natomiast dokładniej ujmując, polega na poszukiwaniu krzywej geometrycznej przejścia pomiędzy początkowym i zadanym końcowym stanem (pozycją) robota tak, aby nie występowały kolizje z przeszkodami oraz planowany ruch nie powodował naruszenia ograniczeń kinematycznych i dynamicznych robota (ścieżka była możliwa do wykonania).

Obecnie typowe roboty przemysłowe wykonują swoje zadania wchodząc tylko w z góry zaplanowane interakcje z otoczeniem, ponieważ środowisko, w którym pracują jest **statyczne** i **strukturalne** (tzn. uporządkowane). Natomiast w przypadku robotów mobilnych do interakcji z otoczeniem, może dojść w sposób przypadkowy, niezaplanowany. Wynika to z charakteru środowiska, w którym roboty się poruszają. W przeciwieństwie do robotów przemysłowych, ich środowisko jest zazwyczaj **dynamiczne** i często jest nieuporządkowane bądź częściowo uporządkowane.

Planowanie ruchu jest zdolnością do decydowania jakie działanie należy podjąć w konkretnej sytuacji w celu osiągnięcia konfiguracji końcowej. Jest to także rozstrzygnięcie wielu decyzji, począwszy od: *którą ścieżkę wybrać* a skończywszy na: *których informacji o środowisku użyć w danej chwili*.

Planowanie w sytuacji, gdy posiadamy pełną informacji na temat aktualnego stanu środowiska zostało już szeroko zbadane. Jednakże w przypadku robotów mobilnych wiedza na temat otoczenia jest zazwyczaj częściowa i niepewna. W ten sposób problem podejmowania poprawnych decyzji staje się znacznie trudniejszy i wymaga realizacji równolegle wielu zadań (jedne w celu *planowania* ruchu, inne w celu *przetrwania robota – omijania przeszkód*).

Aby planować ścieżki zespołu robotów, musimy najpierw zapewnić, że żaden z robotów w trakcie ich realizacji nie ulegnie kolizji z obiektami w otaczającym go środowisku. Dlatego też główny nacisk położono na algorytmy, które lokalnie modyfikują ścieżki ruchu w celu uniknięcia kolizji z przeszkodami/obiektami w środowisku, w którym robot się porusza.

Wykrywanie lokalnych przeszkód i unikanie z nimi kolizji jest jedną z podstawowych funkcji sterowników robotów mobilnych działających w nieznanym lub częściowo nieznanym bądź dynamicznym otoczeniu. W algorytmach unikania kolizji wykorzystuje się dane pomiarowe z czujników, a także docelową pozycję robota oraz jego aktualną pozycję

względem docelowej, do lokalnej modyfikacji trajektorii ruchu robota. Wymaga się, aby algorytmy były efektywne obliczeniowo, a generowany ruch był możliwie gładki oraz by robot podążał w kierunku celu przy uwzględnieniu fizycznych ograniczeń robota [2, 47].

3.1 Metody i algorytmy unikania kolizji

Większość znanych z literatury algorytmów unikania kolizji można zaliczyć do jednej z dwóch podstawowych grup: wyznaczania kierunku ruchu [6, 30, 52, 53] oraz wyznaczania prędkości liniowej i kątowej robota w wyniku przeszukiwania odpowiednio dobranej przestrzeni prędkości [7, 23, 37, 48].

Jednym z wcześniejszych algorytmów, zaliczanych do pierwszej grupy, jest podejście wykorzystujące sztuczne pola potencjałowe [30]. Robot, traktowany jako cząsteczka, porusza się w polu, zaś jego ruch jest wypadkową działających na niego sił pochodzących od przeszkód (siły odpychające) oraz od celu (siła przyciągająca). W swej pierwotnej wersji metoda pól potecjałowych miała szereg wad, wśród których najpoważniejszą było występowanie minimów lokalnych pola, czyli punktów różnych od docelowego, w których siła wypadkowa jest równa zeru [6, 19].

Wśród metod wyznaczania kierunku największą popularność zdobyły kolejne wersje algorytmów z grupy tzw. histogramów pola wektorowego *Vector Field Histogram (VFH)* [6], *VFH+* [52], oraz *VHF** [53].

Zbliżone do *VFH* podejście, zaprezentowane w pracy [33], wykorzystuje diagramy bliskości (*Nearness Diagram (ND)*). Dzięki uwzględnieniu geometrycznych, kinematycznych i dynamicznych ograniczeń dla robota, algorytm ten daje lepsze rezultaty niż *VFH+*, szczególnie w środowiskach z dużą liczbą przeszkód i wąskimi przejściami.

Techniki wykorzystujące pojęcie okna dynamicznego (*Dynamic Window*) są przykładem podejścia polegającego na przeszukiwaniu pewnego zbioru prędkości (*okna*) w celu znalezienia sterowań (prędkości) dopuszczalnych przy uwzględnieniu ograniczeń wynikających z kinematyki robota i uproszczonego modelu dynamiki [7, 23, 37].

Przestrzeń prędkości jest zbiorem wszystkich par (ω, v), składających z prędkości kątowych ω i liniowych v. W podejściu tym zakłada się, że robot może poruszać się tylko po łukach okręgów reprezentowanych przez parę (ω, v).

Do grupy podejść bazujących na wyznaczaniu prędkości zalicza się także metodę tzw. krzywizn i prędkości (*Curvature Velocity Method (CVM)*) [48]. Zakłada się tutaj, podobnie jak w technikach okna dynamicznego, że robot porusza się tylko po łukach okręgów o krzywiznach $c = \frac{\omega}{v}$. W podejściu tym możliwe jest uwzględnienie ograniczeń na dopuszczalne wartości prędkości i przyspieszenia.

Algorytm przedstawiony w niniejszym rozdziale zalicza się do tej grupy i stanowi rozwinięcie idei zaproponowanej w [48]. W niniejszej pracy starano się wyeliminować niedociągnięcia i błędy oryginalnej wersji tego algorytmu.

3.2 Algorytm krzywizn i prędkości *CVM*

Algorytm ten jest lokalną metodą planowania ruchu bazującą na wyznaczaniu prędkości liniowej i kątowej robota.

Problem omijania przeszkód jest opisywany jako zadanie optymalizacji z ograniczeniami w przestrzeni prędkości robota.

Przestrzeń prędkości jest zbiorem dopuszczalnych prędkości. W zadaniu tym przyjmujemy sterowanie za pomocą niezależnych prędkości: liniowej v oraz kątowej ω.

Wybór takich sterowań jest naturalny dla robotów o napędzie synchronicznym. Dla naszego robota o napędzie różnicowym bardziej oczywiste wydaje się sterowanie za pomocą niezależnych prędkości lewego v_l oraz prawego v_r koła. Przejście pomiędzy tymi prędkościami wyraża się prostymi zależnościami: $\omega = \frac{v_l - v_r}{d}$, gdzie d to rozstaw kół robota, oraz $v = \frac{v_l + v_r}{2}$. Ponieważ przeliczenie takie jest dokonywane w sterowniku robota, więc w poniższych rozważaniach będziemy operować na prędkościach (ω, v).

Pożądane prędkości (ω, v) są uzyskiwane w wyniku maksymalizacji określonej funkcji celu przy spełnieniu ograniczeń.

Do zalet tak sformułowanego zadania należą między innymi:

- łatwość dodawania ograniczeń wynikających z dynamiki robota,
- niezależne sterowanie prędkością oraz zmianą orientacji robota przez zadawanie prędkości liniowej i kątowej (sterowanie prędkością kątową wpływa tylko i wyłącznie na zmianę orientacji, natomiast prędkość liniowa wpływa jedynie na prędkość robota),
- możliwość wprowadzania dodatkowych czynników decydujących o wyborze prędkości przez modyfikację funkcji celu.

W rzeczywistości, z dobrym przybliżeniem, można założyć, że robot, w każdej chwili czasu, porusza się wzdłuż łuku pewnego okręgu, którego krzywizna wyraża się wzorem:

$$c = \frac{\omega}{v}. \tag{3.1}$$

Fizyczna konstrukcja robota narzuca dwa typy ograniczeń:

- na maksymalną oraz minimalną prędkość kątową oraz liniową robota:

$$\omega_{min} \leqslant \omega \leqslant \omega_{max}, \tag{3.2}$$

$$v_{min} \leqslant v \leqslant v_{max}; \tag{3.3}$$

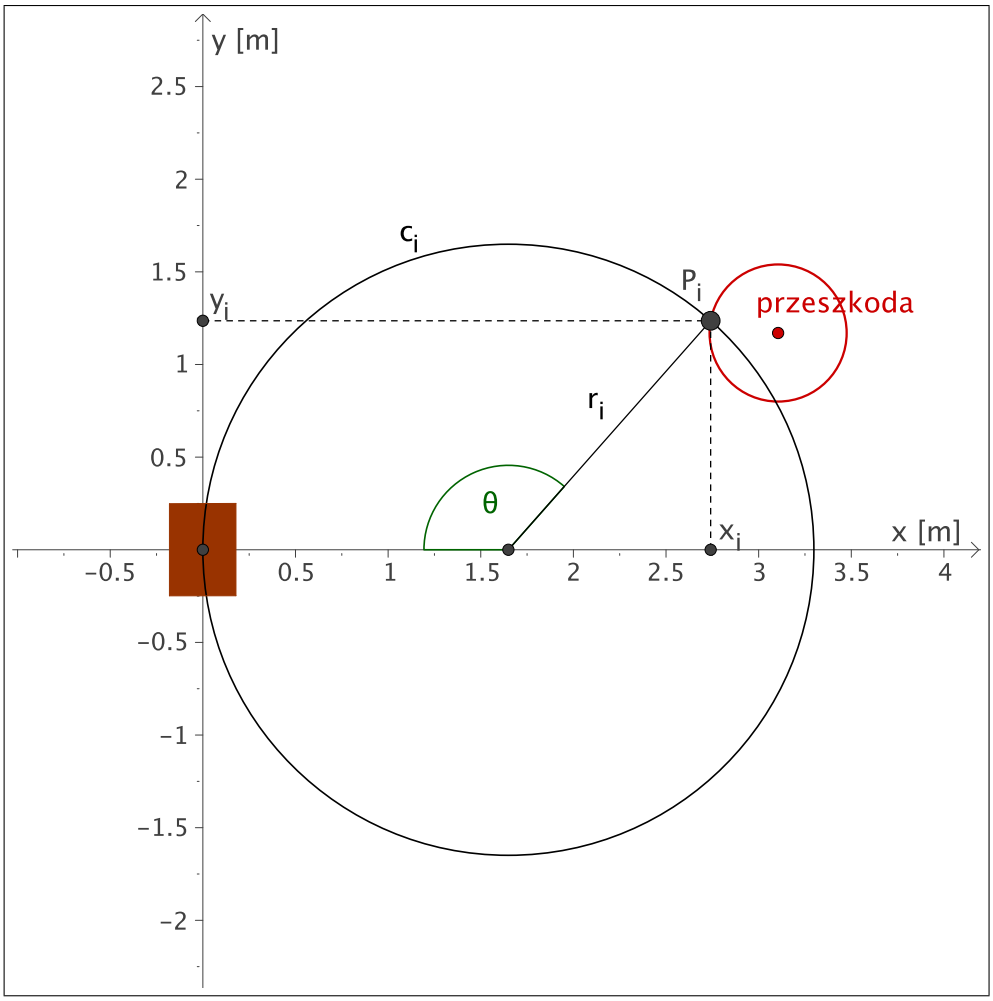
- na przyspieszenie kątowe oraz liniowe. Dla prędkości (ω_n, v_n) w chwili n oraz okresu T_{alg}, będącego czasem jednego wykonania algorytmu, prawdziwe są warunki:

$$\omega_n - \epsilon_{max} \cdot T_{alg} \leqslant \omega_{n+1} \leqslant \omega_n + \epsilon_{max} \cdot T_{alg}, \tag{3.4}$$

$$v_{n+1} \leqslant v_n + a_{max} \cdot T_{alg}, \tag{3.5}$$

  gdzie ϵ_{max} oraz a_{max} to odpowiednio maksymalne przyspieszenie kątowe oraz maksymalne przyspieszenie liniowe robota.

Najważniejszym źródłem ograniczeń na dopuszczalne prędkości $(\omega,\ v)$ są przeszkody znajdujące się w środowisku otaczającym robota. Ponieważ w naturalny sposób przeszkody są opisywane we współrzędnych kartezjańskich, należy dokonać transformacji ich opisu do przestrzeni prędkości robota. Poniżej opisano jak jest to realizowane.



Rysunek 3.1: Wyznaczanie odległości od przeszkody p dla okręgu o krzywiźnie c_i

Dla wszystkich krzywizn c należy obliczyć odległość $d_c((0,\ 0),\ p_i)$ rozumianą jako długość drogi, jaką musi przebyć robot, startując z punktu $(0,\ 0)$ w układzie lokalnym robota, wzdłuż okręgu o krzywiźnie c, zanim zderzy się z przeszkodą p w punkcie p_i. Następnie definiujemy funkcję odległości robota od przeszkody jako (zapis ten nie jest ścisły z matematycznego punktu widzenia):

$$d_v(\omega,\ v,\ p) = \begin{cases} d_c\left((0,\ 0),\ p_i\right) & dla \quad v \neq 0 \\ \infty & dla \quad v = 0 \end{cases}, \text{ gdzie } c = \frac{\omega}{v} \tag{3.6}$$

Dla danego zbioru przeszkód P funkcja ta ma postać:

$$D(\omega,\ v,\ P) = \inf_{p \in P}\ d_v(\omega,\ v,\ p). \tag{3.7}$$

Ponieważ zasięg czujników jest ograniczony należy ograniczyć zakres możliwych wartości funkcji D do pewnej ustalonej wartości L (w naszym przypadku przyjęliśmy $L = 1,1\ m$

dla czujników o zasięgu $1,5\ m$):

$$D_{ogr}(\omega,\ v,\ P) = \min\ (L,\ D(\omega,\ v,\ p)). \tag{3.8}$$

Obliczenie odległości $d_c\left((0,\ 0),\ p_i\right)$ od przeszkody p, o dowolnym kształcie, może być w ogólnym przypadku skomplikowane. Dlatego też, w naszych rozważaniach, będziemy przybliżać przeszkody okręgami opisanymi na nich. Dla robota o orientacji zgodnej z osią OY, dla danego okręgu, o krzywiźnie c_i, który przecina przeszkodę w punkcie $P_i(x_i,\ y_i)$ otrzymujemy (rys. 3.1):

$$\theta = \begin{cases} atan2(y_i,\ x_i - \frac{1}{c_i}) & dla \quad c_i < 0 \\ \pi - atan2(y_i,\ x_i - \frac{1}{c_i}) & dla \quad c_i > 0 \end{cases} \tag{3.9}$$

$$d_{c_i}\left((0,\ 0),\ P_i\right) = \begin{cases} y_i & dla \quad c_i = 0 \\ \left|\frac{1}{c_i}\right| \cdot \theta & dla \quad c_i \neq 0 \end{cases} \tag{3.10}$$

Mając dane już ograniczenia wynikające z otoczenia robota, jak i jego możliwości, wartości prędkości otrzymujemy poszukując maksimum funkcji celu. Funkcję celu dobieramy tak, aby spełnić następujące wymagania:

- robot powinien dążyć do osiągnięcia swojej maksymalnej prędkości liniowej,
- robot powinien poruszać się po krzywych niekolidujących z przeszkodami,
- robot powinien zawsze kierować się na cel.

Powyższe wymagania można zapisać w postaci liniowej funkcji celu:

$$\mathcal{F}(\omega,\ v) = \alpha_1 \cdot \mathcal{V}(v) + \alpha_2 \cdot \mathcal{D}(\omega,\ v) + \alpha_3 \cdot \mathcal{G}(\omega), \tag{3.11}$$

gdzie:

$$\mathcal{V}(v) = \frac{v}{v_{max}} \tag{3.12}$$

$$\mathcal{D}(\omega, v) = \frac{D_{ogr}(\omega,\ v,\ P)}{L} \tag{3.13}$$

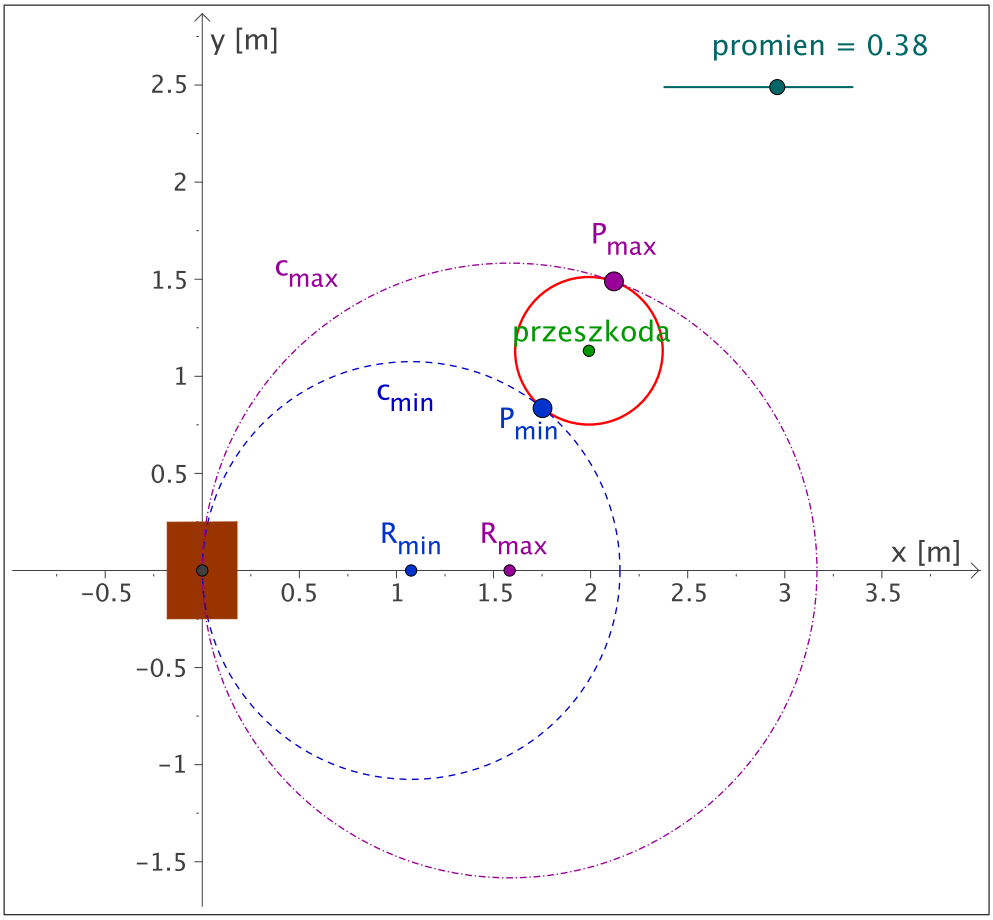
$$\mathcal{G}(\omega) = 1 - \frac{|\theta_{cel} - \omega \cdot T_{alg}|}{\pi} \tag{3.14}$$

Wartości poszczególnych składników $\mathcal{V}$, $\mathcal{D}$, $\mathcal{G}$ funkcji $\mathcal{F}$ (3.11) są znormalizowane. Składowa $\mathcal{V}$ odpowiada za poruszanie się z maksymalną możliwą prędkością liniową, składowa $\mathcal{D}$ jest odpowiedzialna ze podążanie wzdłuż bezkolizyjnych krzywych, natomiast składowa $\mathcal{G}$ preferuje podążanie za celem. W ostatnim członie θ_{cel} jest orientacją do celu w układzie współrzędnych robota.

Wartości współczynników α_i są wagami powyższych składników. Od doboru tych współczynników zależy zachowanie robota, tzn. jak będzie skręcał, jak wcześnie będzie reagował na przeszkody, itd.

3.3 Szczegółowy opis algorytmu

Pomimo uproszczenia, jakim jest przyjęcie, że przeszkody są reprezentowane przez okręgi, obliczenie funkcji D_{ogr} (3.8) dla dużej liczby przeszkód, jest czasochłonne. Zauważmy, że dla danej przeszkody p odległość $d_c\left((0,\ 0),\ p_i\right)$ jest nieskończona na zewnątrz krzywych stycznych do danej przeszkody. Stąd, wystarczy, że będziemy rozpatrywać jedynie krzywe leżące pomiędzy c_{min} a c_{max} (rys. 3.2).



Rysunek 3.2: Krzywe styczne do przeszkody p w punktach P_{min} i P_{max}

Aby wyznaczyć wartości c_{min}, c_{max}, $P_{min}(x_{min},\ y_{min})$ oraz $P_{max}(x_{max},\ y_{max})$ dla przeszkody p o środku w punkcie $P(x_0,\ y_0)$ $((x_0,\ y_0) \neq (0,\ 0))$ i promieniu r_0 należy znaleźć takie r, aby poniższy układ równań, przedstawiający dwa okręgi, miał dokładnie jedno rozwiązanie:

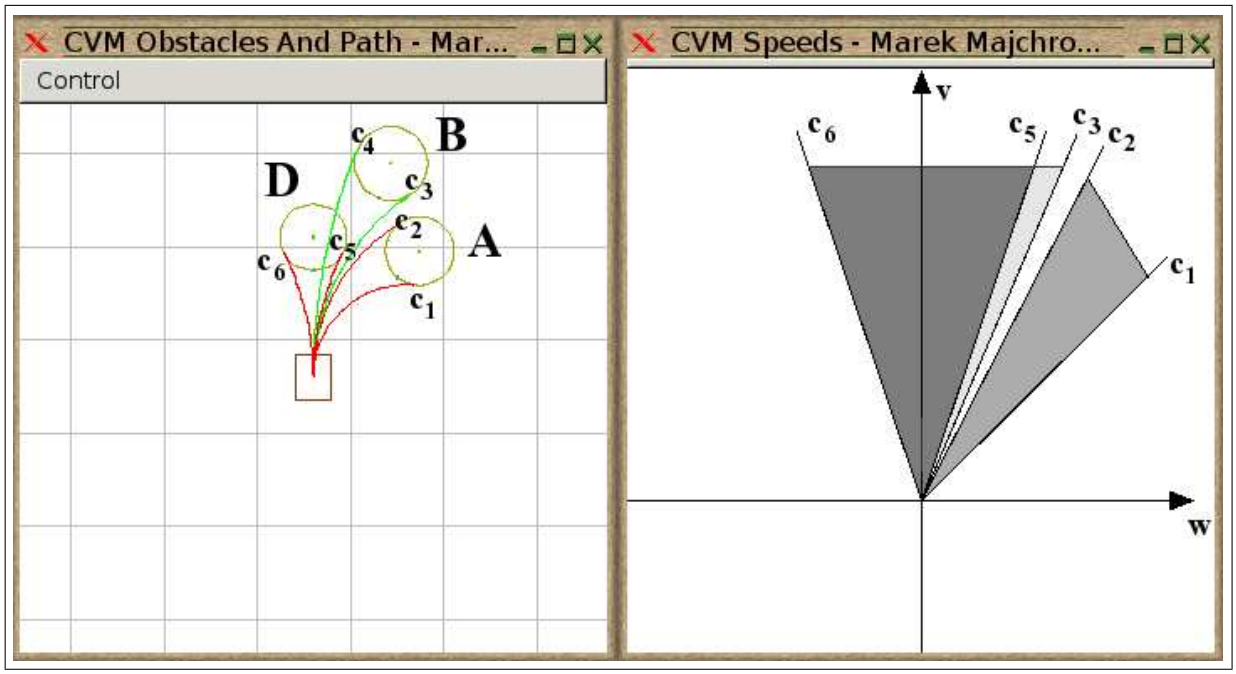
$$\begin{cases} (x - x_0)^2 + (y - y_0)^2 &= r_0^2 \\ (x - r)^2 + y^2 &= r^2 \end{cases} \tag{3.15}$$

Powyższy układ równań należy sprowadzić do równania kwadratowego jednej zmiennej i wyznaczyć wyróżnik tego równania. O tym, względem której zmiennej będziemy rozwiązywać to równanie kwadratowe, będzie decydować położenie przeszkody w układzie współrzędnych, czego powodem jest minimalizacja błędów numerycznych. Wyliczenia znajdują się w 3.3.1 oraz 3.3.2.

Mając dane krzywe styczne do przeszkody, przybliżamy funkcję d_v (3.6) funkcją stałą, dla wszystkich c z zakresu $c \in [c_{min},\ c_{max}]$:

$$d_v(\omega,\ v,\ p) = \begin{cases} \min\left(d_{c_{min}}\left((0,\ 0),\ P_{min}\right), d_{c_{max}}\left((0,\ 0),\ P_{max}\right)\right) & dla \quad c_{min} \leqslant c \leqslant c_{max} \\ \infty & w\ p.\ p. \end{cases} \tag{3.16}$$

W ten sposób, dla zbioru przeszkód, otrzymujemy zbiór przedziałów, każdy o stałej odległości. Każdy z tych przedziałów można reprezentować jako strukturę postaci: $([c_1,\ c_2], d_{1,\ 2})$, gdzie c_1 i c_2 to krzywizny okręgów wyznaczających ten przedział ($c_1 \leqslant c_2$), a $d_{1,\ 2}$ jest odległością do przeszkody wewnątrz tego przedziału. W dalszych rozważaniach *przedział* będzie rozumiany jako jego reprezentacja w postaci struktury typu: $([c_1,\ c_2], d_{1,\ 2})$



Rysunek 3.3: Przeszkody i graficzna interpretacji listy przedziałów, które z nich powstały

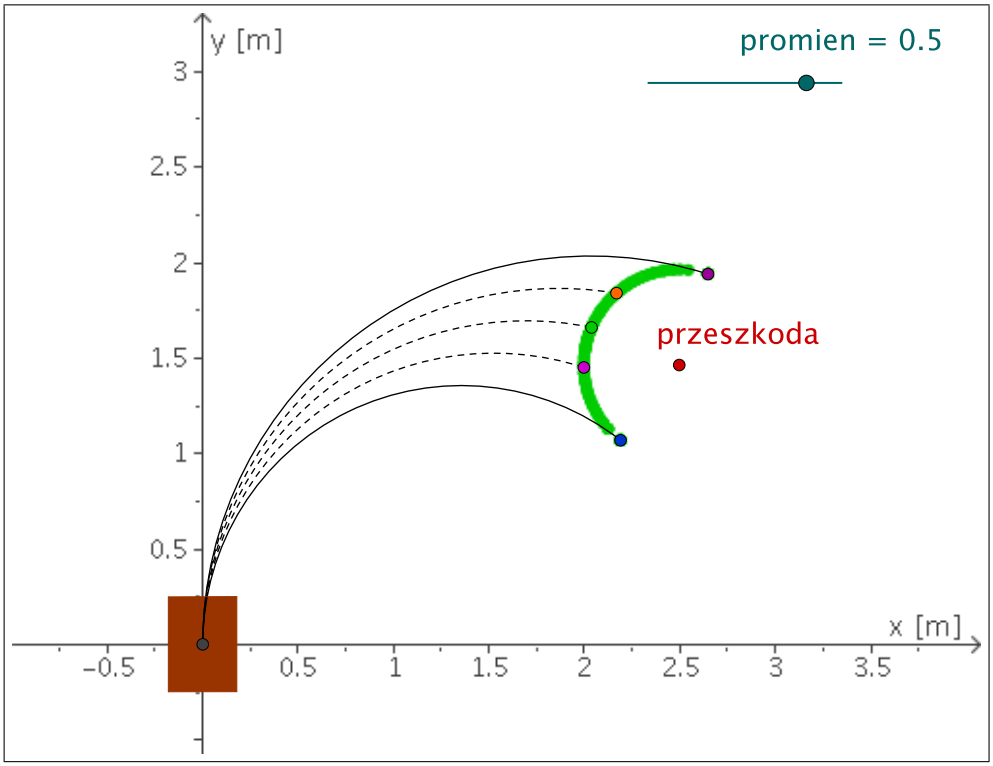
Zaczynając od listy przedziałów zawierającej tylko przedział $([-\infty,\ \infty], L)$, dla każdej przeszkody wyznaczamy przedział $([c_{min},\ c_{max}], d)$ i dodajemy ten przedział do listy według zasady: dla każdego przedziału $([c_1,\ c_2], d_{1,\ 2})$ z listy: *sprawdź w jakich zależnościach jest ten przedział z nowo dodawanym.* Jeśli:

- są to przedziały rozłączne – nic nie rób,
- przedział z listy zawiera się w nowo dodawanym przedziale ($c_1 \leqslant c_{min}$ i $c_2 \leqslant c_{max}$) – ustaw odległość $d_{1,\ 2} = \min(d, d_{1,\ 2})$,
- przedział z listy zawiera nowo dodawany przedział ($c_1 \geqslant c_{min}$ i $c_2 \geqslant c_{max}$) – jeśli $d < d_{1,\ 2}$, to podziel istniejący przedział na trzy: $([c_1,\ c_{min}], d_{1,\ 2})$, $([c_{min},\ c_{max}], d)$ oraz $([c_{max},\ c_2], d_{1,\ 2})$; w przeciwnym przypadku nic nie rób,
- przedziały na siebie zachodzą – jeśli $d < d_{1,\ 2}$, to podziel istniejący przedział na dwa, a wynik uzależnij od tego, które brzegi przedziałów na siebie zachodziły, tzn.

dla $c_{max} > c_2$ podziel na przedziały $([c_1,\ c_{min}], d_{1,\ 2})$ i $([c_{min},\ c_2], d)$, w przeciwnym przypadku na przedziały $([c_1,\ c_{max}], d)$ i $([c_{max},\ c_2], d_{1,\ 2})$; jeśli $d \geqslant d_{1,\ 2}$, to nic nie rób.

Następnie należy połączyć sąsiadujące przedziały, które mają tę samą odległość. W ten sposób otrzymujemy listę rozłącznych przedziałów w przestrzeni prędkości. Geometrycznie, każdy przedział w takiej liście definiuje parę prostych w przestrzeni prędkości (rys. 3.3).

Przybliżenie zbioru długości krzywych z zakresu $[c_{min},\ c_{max}]$ pojedynczą wartością nie daje dobrych wyników (rys. 3.4). W pewnych sytuacjach przybliżenie takie jest zbyt re-

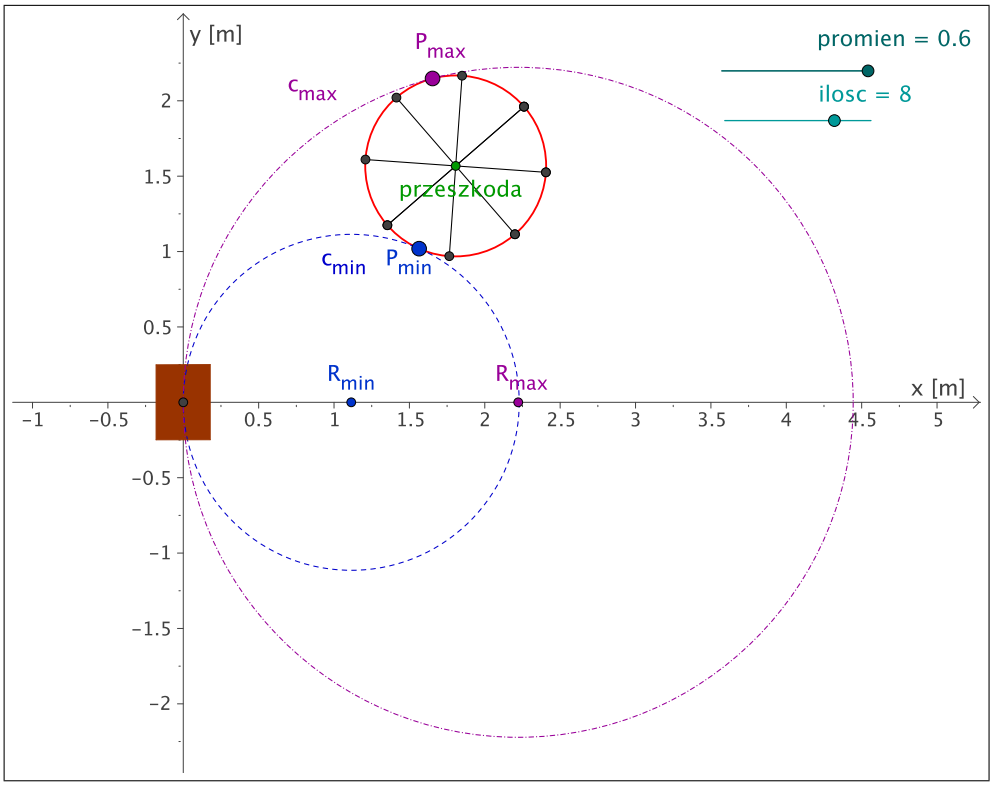


Rysunek 3.4: Zbiór krzywych z zakresu $[c_{min},\ c_{max}]$ o różnych długościach o danej przeszkody p.

strykcyjne, a w innych jest zbyt słabe i może powodować problemy (sytuacja, w której rzeczywista odległość od przeszkody jest mniejsza niż ta, wyliczona z definicji (3.16)).

Jednym z rozwiązań tego problemu jest podzielenie przedziału $[c_{min},\ c_{max}]$ na podprzedziały, zastosowanie równania (3.16) dla tych podprzedziałów, a następnie dodanie ich do listy zawierającej wszystkie takie przedziały w sposób omówiony wyżej.

Rozwiązanie, które przyjęto, polega na wyznaczeniu punktu, na okręgu opisującym przeszkodę, do którego odległość od robota jest najmniejsza, w metryce euklidesowej. Następnie okrąg ten jest dzielony na k segmentów, które wyznaczają nowe punkty, (rys. 3.5) począwszy od wyznaczonego wcześniej punktu. Dla każdych dwóch sąsiadujących punktów leżących pomiędzy punktami, wyznaczonymi przez krzywe styczne do przeszkody, tworzony jest przedział $([c_1,\ c_2], d_{1,\ 2})$ i dodawany do listy w sposób omówiony wyżej. c_1 i c_2 to krzywizny okręgów przechodzących przez te dwa punkty, a jako odległość $d_{1,\ 2}$



Rysunek 3.5: Sposób podziału okręgu opisującego przeszkodę na $k = 8$ części

brana jest mniejsza z odległości liczonych jako droga z równania 3.10, wzdłuż krzywych c_1 i c_2. Dla przykładu z rysunku 3.5 dodane zostaną cztery przedziały, zamiast jednego.

3.3.1 Równanie kwadratowe względem zmiennej x, czyli $|y_0| > |x_0|$

Wyróżnik równania kwadratowego, które powstało po sprowadzeniu układu równań 3.15 do równania kwadratowego zmiennej x, dany jest wzorem:

$$\Delta = y_0^2\left(-4x_0^2+4r_0^2\right)r^2 + y_0^2\left(y_0^2 4x_0 - x_0^2 - 4x_0 - r_0^2 4x_0\right)r + \\ + y_0^2\left(x_0^2\left(2r_0^2-2y_0^2-x_0^2\right) - y_0^4 + r_0^2\left(2y_0^2-r_0^2\right)\right)$$

Rozwiązując równanie kwadratowe $\Delta = 0$ względem zmiennej r otrzymujemy:

$$r_{max} = \frac{x_0^2+y_0^2-r_0^2}{2x_0-2r_0} \qquad r_{min} = \frac{x_0^2+y_0^2-r_0^2}{2x_0+2r_0} \tag{3.17}$$

$$c_{min} = \frac{1}{r_{max}} = \frac{2x_0-2r_0}{x_0^2+y_0^2-r_0^2} \qquad c_{max} = \frac{1}{r_{min}} = \frac{2x_0+2r_0}{x_0^2+y_0^2-r_0^2} \tag{3.18}$$

Punkty styczności okręgów $P_{min}(x_{min},\ y_{min})$ oraz $P_{max}(x_{max},\ y_{max})$ (podstawiając za r odpowiednio r_{min} oraz r_{max}):

$$\begin{cases} x &= \frac{x_0^3+rr_0^2-rx_0^2+ry_0^2-r_0^2x_0+x_0y_0^2}{-4rx_0+2r^2+2x_0^2+2y_0^2} \\ y &= -\frac{-2rx+2xx_0+r_0^2-x_0^2-y_0^2}{2y_0} \end{cases} \qquad \text{dla } y_0 \neq 0 \tag{3.19}$$

Punkty przecięcia okręgów dla $\Delta > 0$ oraz danego r można wyznaczyć ze wzorów:

$$\begin{cases} x_1 &= \frac{x_0^3+rr_0^2-rx_0^2+ry_0^2-r_0^2x_0+x_0y_0^2-\sqrt{\Delta}}{-4rx_0+2r^2+2x_0^2+2y_0^2} \\ y_1 &= -\frac{-2rx_1+2x_1x_0+r_0^2-x_0^2-y_0^2}{2y_0} \end{cases} \quad \text{dla } y_0 \neq 0 \tag{3.20}$$

$$\begin{cases} x_2 &= \frac{x_0^3+rr_0^2-rx_0^2+ry_0^2-r_0^2x_0+x_0y_0^2+\sqrt{\Delta}}{-4rx_0+2r^2+2x_0^2+2y_0^2} \\ y_2 &= -\frac{-2rx_2+2x_2x_0+r_0^2-x_0^2-y_0^2}{2y_0} \end{cases} \quad \text{dla } y_0 \neq 0 \tag{3.21}$$

3.3.2 Równanie kwadratowe względem zmiennej y, czyli $|y_0| \leqslant |x_0|$

Wyróżnik równania kwadratowego, które powstało po sprowadzeniu układu równań 3.15 do równania kwadratowego zmiennej y, dany jest wzorem:

$$\begin{aligned} \Delta = &-x_0^6 + 6rx_0^5 + 2rr_0^4x_0 + 2rx_0y_0^4 - r^2r_0^4 + 4r^4r_0^2 - 13r^2x_0^4 + 12r^3x_0^3 \\ &- 4r^4x_0^2 - r^2y_0^4 + 2r_0^2x_0^4 - r_0^4x_0^2 - x_0^2y_0^4 - 2x_0^4y_0^2 - 8rr_0^2x_0^3 \\ &- 12r^3r_0^2x_0 + 8rx_0^3y_0^2 + 4r^3x_0y_0^2 - 4rr_0^2x_0y_0^2 + 14r^2r_0^2x_0^2 \\ &+ 2r^2r_0^2y_0^2 - 10r^2x_0^2y_0^2 + 2r_0^2x_0^2y_0^2 \end{aligned}$$

Rozwiązując równanie kwadratowe $\Delta = 0$ względem zmiennej r otrzymujemy:

$$r_{max} = \frac{x_0^2+y_0^2-r_0^2}{2x_0-2r_0} \qquad r_{min} = \frac{x_0^2+y_0^2-r_0^2}{2x_0+2r_0} \tag{3.22}$$

$$c_{min} = \frac{1}{r_{max}} = \frac{2x_0-2r_0}{x_0^2+y_0^2-r_0^2} \qquad c_{max} = \frac{1}{r_{min}} = \frac{2x_0+2r_0}{x_0^2+y_0^2-r_0^2} \tag{3.23}$$

Punkty styczności okręgów $P_{min}(x_{min},\ y_{min})$ oraz $P_{max}(x_{max},\ y_{max})$ (podstawiając za r odpowiednio r_{min} oraz r_{max}):

$$\begin{cases} x &= \frac{2yy_0+r_0^2-x_0^2-y_0^2}{2r-2x_0} \\ y &= \frac{-2rx_0y_0+y_0^3+2r^2y_0-r_0^2y_0+x_0^2y_0}{-4rx_0+2r^2+2x_0^2+2y_0^2} \end{cases} \quad \text{dla } x_0 \neq r \tag{3.24}$$

Punkty przecięcia okręgów dla $\Delta > 0$ oraz danego r można wyznaczyć ze wzorów:

$$\begin{cases} x_1 &= \frac{2y_1y_0+r_0^2-x_0^2-y_0^2}{2r-2x_0} \\ y_1 &= \frac{-2rx_0y_0+y_0^3+2r^2y_0-r_0^2y_0+x_0^2y_0-\sqrt{\Delta}}{-4rx_0+2r^2+2x_0^2+2y_0^2} \end{cases} \quad \text{dla } x_0 \neq r \tag{3.25}$$

$$\begin{cases} x_2 &= \frac{2y_2y_0+r_0^2-x_0^2-y_0^2}{2r-2x_0} \\ y_2 &= \frac{-2rx_0y_0+y_0^3+2r^2y_0-r_0^2y_0+x_0^2y_0+\sqrt{\Delta}}{-4rx_0+2r^2+2x_0^2+2y_0^2} \end{cases} \quad \text{dla } x_0 \neq r \tag{3.26}$$

- W przypadku, gdy $x_0 = r$ i $y_0 \neq 0$ równanie kwadratowe należy rozwiązywać względem zmiennej x.
- W przypadku, gdy $x_0 = r$ i $y_0 = 0$ układ równań ma rozwiązanie tylko dla $r = r_0$. Wtedy należy przyjąć rozwiązania postaci:

$$\begin{cases} x_1 &= 0 \\ y_1 &= 0 \end{cases} \qquad \begin{cases} x_2 &= 2r_0 \\ y_2 &= 2r_0 \end{cases}$$

- Przypadku, gdy $y_0 = 0$ i $x_0 = 0$ w praktyce nie trzeba rozpatrywać, gdyż środek przeszkody nie może znajdować się w początku układu związanego z robotem.

3.4 Szczegółowy opis działania programu

Program implementujący algorytm *CVM* został napisany z wykorzystaniem struktury *Player/Stage* i działa w następujący sposób:

1. Pobiera dane o aktualnym położeniu robota oraz bieżące odczyty z czujników (10 razy na sekundę).

2. Na podstawie informacji z odometrii, o przemieszczeniu, przelicza położenie celu względem lokalnego układu współrzędnych. Jeśli w swojej pamięci posiada informacje o przeszkodach, także przelicza położenie tych przeszkód. Dodatkowo, usuwa te przeszkody, po przeliczeniu ich położenia, których współrzędna $y < 0$ (nie chcemy pamiętać o przeszkodach, które minęliśmy).

3. Dla każdego odczytu z czujników odległości wykonywane są następujące czynności:

   (a) Jeśli odczyt odległości jest większy niż pewna ustalona wartość (L w algorytmie), to pomija dany odczyt.

   (b) Sprawdzane jest w pamięci, czy nie ma już informacji o przeszkodzie znajdującej się w podobnym miejscu. Warunkiem, żeby przeszkody zostały uznane za identyczne, jest to, aby ich środki znajdowały się w kwadracie o boku *pos_epsilon*. W implementacji przyjąłem $pos\_epsilon = 0.05$ $[m]$. Jeśli okaże się, że istnieje już taka przeszkoda, pomija dany odczyt.

   (c) Wyznacza promień okręgu opisującego daną przeszkodę. Promień zależy od odległości do przeszkody i wyraża się wzorem:

   $$promien = obstacle\_radius + increase\_radius\_factor \cdot \cdot (odleglosc - obstacle\_radius)$$

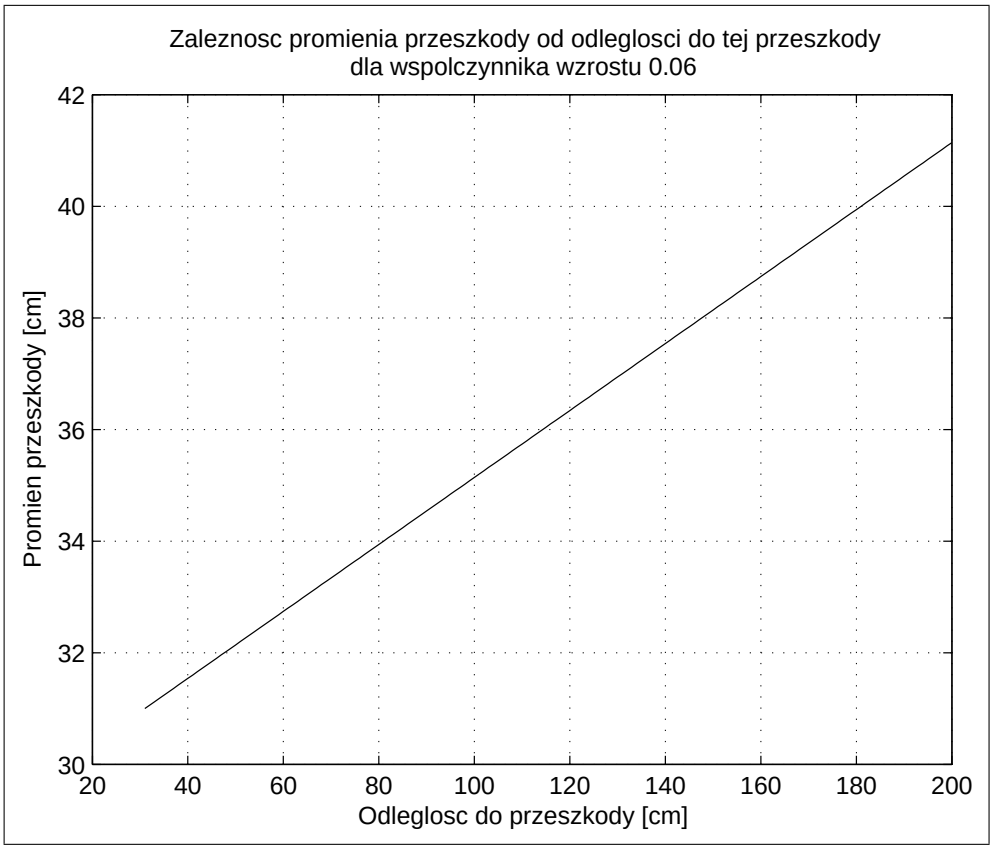
   gdzie:

$odleglosc$	to odległość robota od przeszkody,
$obstacle\_radius$	to promień wyjściowy przeszkody,
$increase\_radius\_factor$	to współczynnik opisujący, jak bardzo należy zwiększyć promień przeszkody w zależności od odległości do niej.

   W implementacji przyjąłem $increase\_radius\_factor = 0.06$ (rys 3.6), a wartość $obstacle\_radius$ ustaliłem na poziomie 0.31 $[m]$, gdyż tyle wynosi połowa przekątnej robota.

   (d) Przeszkoda dodawana jest do pamięci i zwiększany jest licznik przeszkód.

4. Sprawdza odległość do celu. Jeśli jest ona mniejsza niż założona dokładność $dist\_accuracy$, zatrzymuje robota. W przeciwnym przypadku wyznacza nowe prędkości sterujące. W implementacji przyjąłem $dist\_accuracy = 0.06$ $[m]$.



Rysunek 3.6: Zależność promienia przeszkody od odległości do tej przeszkody dla $increase\_radius\_factor = 0.06$

5. Zapamiętuje nowo wyznaczone prędkości robota.

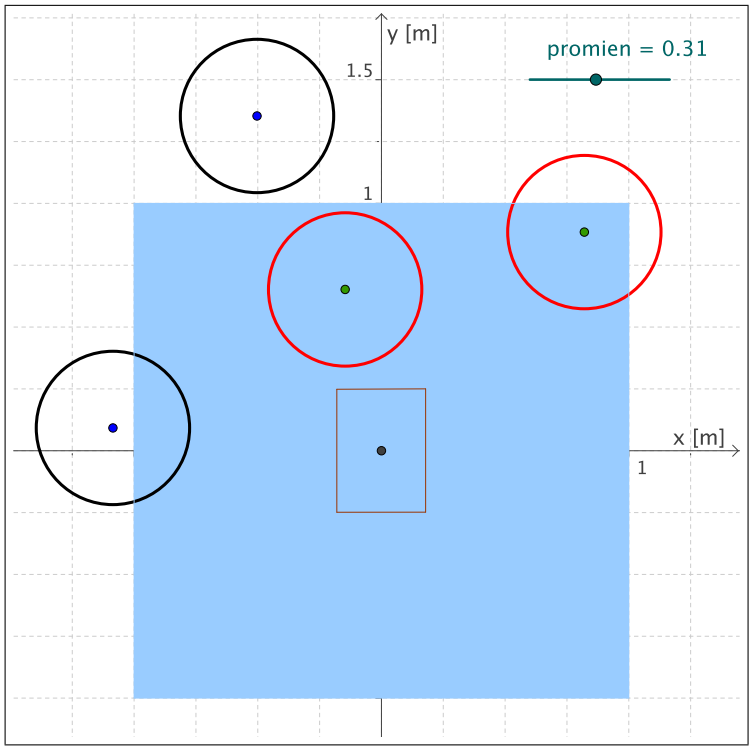
6. Jeśli opcja *remember_obstacles* jest włączona, to z pamięci są usuwane tylko przeszkody *zbędne*. W przeciwnym przypadku usuwane są wszystkie przeszkody. Przeszkodę uważa się za *zbędną*, gdy jej środek znajduje się poza prostokątem, o wymiarach $xbound \times ybound$, którego środkiem symetrii jest punkt reprezentujący robota (rys. 3.7). W implementacji przyjąłem $xbound = ybound = 1$ $[m]$.

Informacje o przeszkodach program może czerpać z trzech rodzajów czujników: podczerwieni, sonarowych oraz dalmierza laserowego (o wyborze źródła należy zadecydować w momencie kompilacji programu). Pamięć o przeszkodach została wprowadzona do programu z powodu rozmieszczenia czujników podczerwieni w naszym robocie. Bez tej pamięci, robot widząc nawet wcześniej czarną przeszkodę na rysunku 3.8, wjechałby w nią w późniejszym czasie, ponieważ jest ona niewidoczna dla jego czujników.

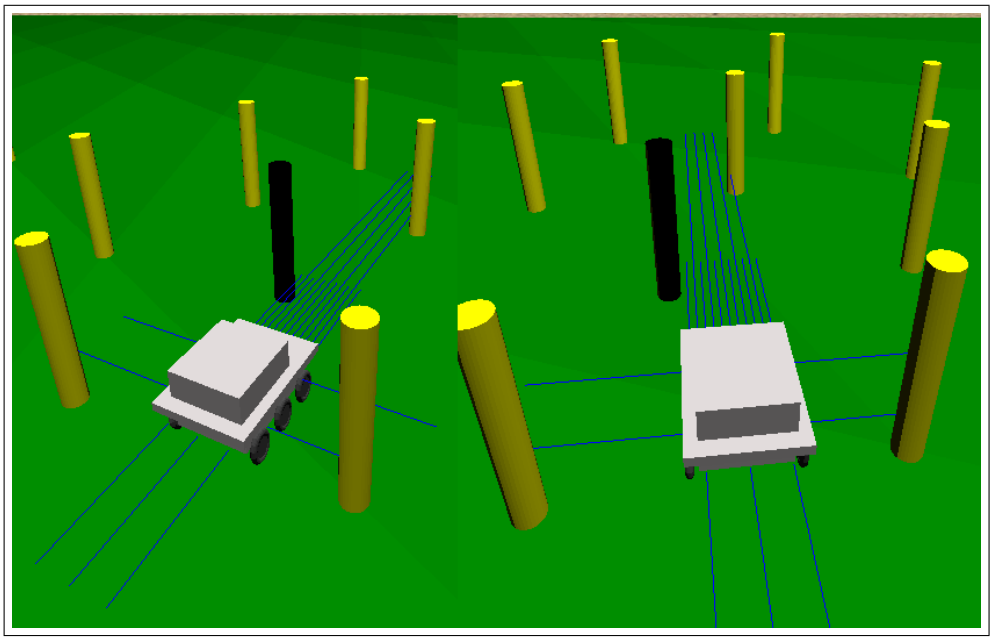
Informacja o położeniu robota może pochodzić z odometrii robota, bądź z globalnej wiedzy o jego położeniu, jak to ma miejsce w symulatorze (o wyborze źródła należy zadecydować w momencie kompilacji programu).

Wyznaczanie nowych prędkości sterujących odbywa się w następujący sposób:

1. Na podstawie bieżącej informacji o prędkościach, wyznaczane są, z równań 3.2, 3.3, 3.4 oraz 3.5, ograniczenia na dopuszczalne prędkości, które będą brane pod uwagę przy wyznaczaniu nowego sterowania.



Rysunek 3.7: Czarne okręgi oznaczają przeszkody zbędne, czerwone – przeszkody, które zostaną zapamiętane



Rysunek 3.8: Wizualizacja wiązki IR czujników podczerwieni w robocie *Protonek*

2. Tworzona jest lista przedziałów, która początkowo zawiera tylko jeden przedział: $([-\infty,\ \infty], L)$.

3. Dla każdej zapamiętanej przeszkody wykonywane są następujące czynności:

   (a) Ze wzoru 3.18 wyznaczane są styczne do przeszkody (c_{min} oraz c_{max}), natomiast

ze wzorów 3.19 oraz 3.24 wyznaczane są punkty styczności (P_{min} oraz P_{max}).

(b) Ze wzoru 3.10 liczona jest odległość (d_{min} oraz d_{max}) do punktów styczności wzdłuż wyznaczonych krzywych.

(c) Wyznaczane są segmenty okręgu opisującego przeszkodę, do których należą punkty styczności, oraz punkty pośrednie pomiędzy tymi segmentami.

(d) Wszystkie przedziały $([c_1,\ c_2], d)$ wynikające z nowo otrzymanych punktów są dodawane do listy przedziałów w sposób opisany wyżej.

4. Otrzymana w ten sposób lista przedziałów poddawana jest kompresji. Łączone są sąsiadujące przedziały, między którymi odległość jest nie większa niż

$$\max(dist \cdot merge\_interval\_factor,\ min\_merge\_dist)$$

gdzie:

$dist$	to odległość ostatniego przedziału, który nie został połączony z żadnym innym,
$merge\_interval\_factor$	to współczynnik odpowiadający za łączenie przedziałów, przy maksymalnej odległości od przeszkody, łączone są przedziały, między którymi różnica odległości wynosi: $merge\_interval\_factor$
min_merge_dist	przedziały, między którymi odległość jest mniejsza niż $merge\_interval\_factor$ zawsze będą łączone.

W implementacji przyjąłem $merge\_interval\_factor = 0.05\ [m]$, a $min\_merge\_dist = 0.05\ [m]$.

5. Dla każdego przedziału sprawdzane jest, czy jego odległość jest większa niż sąsiednich przedziałów. Jeśli tak jest, to krzywa ograniczająca ten przedział jest zmniejszana (jeśli odległość dla przedziału po prawej jest mniejsza) lub zwiększana (jeśli odległość dla przedziału po lewej jest mniejsza) o pewną wartość, która wynika z powiększenia promienia okręgu, do którego ta krzywa jest styczna, o wartość $safety\_margin$. W implementacji przyjąłem $safety\_margin = 0.05\ [m]$.

6. Dla każdego przedziału wyznaczane jest najlepsze sterowanie uwzględniając ograniczenia wynikające z kinematyki i dynamiki robota oraz samego przedziału (każdy przedział wprowadza liniowe ograniczenia w przestrzeni prędkości).

7. Z tak wyznaczonych sterowań wybierane jest to, dla którego funkcja celu 3.11 osiąga maksymalną wartość.

3.5 Wyniki eksperymentalne

Wyniki zostały przygotowane z wykorzystaniem trójwymiarowego symulatora *Gazebo*, modelującego zachowanie i oddziaływania między obiektami.

Czas obliczeń algorytmu wynosi szacunkowo 500 μsec dla średnio 30 przeszkód, na komputerze klasy *Pentium III 900 MHz*.

Przedstawiony algorytm unikania kolizji zależy od wielu parametrów, w szczególności od wartości współczynników α. Po przeprowadzeniu szeregu doświadczeń w środowisku symulacyjnym, a następnie ich weryfikacji w rzeczywistości, dobrano następujące wartości parametrów:

$$\alpha_1 = 0,3 \quad \alpha_2 = 0,6 \quad \alpha_3 = 0,1$$

W obu poniższych eksperymentach roboty rozpoczynały ruch w tej samej chwili.

3.5.1 Środowisko pomiarowe 1

W eksperymencie tym każdy z robotów miał przejechać 4 m przed siebie. Roboty do pomiaru odległości wykorzystywały jedynie czujniki *IR*. Początkowa odległość między robotami wynosiła 5 m, roboty skierowane były do siebie przodem. Na rysunku 3.9 pokazany jest stan początkowy robotów (rys. z lewej), oraz końcowy (rys. z prawej).

Rysunek 3.10 przedstawia wykres położenia robotów w trakcie wymijania się nawzajem (rys. 3.9). Roboty osiągnął cel z uchybem mniejszym niż 6 cm. Na rysunku 3.11 znajdują się wykresy prędkości sterujących, natomiast na rysunku 3.12 – wykresy orientacji robotów w trakcie jazdy.

3.5.2 Środowisko pomiarowe 2

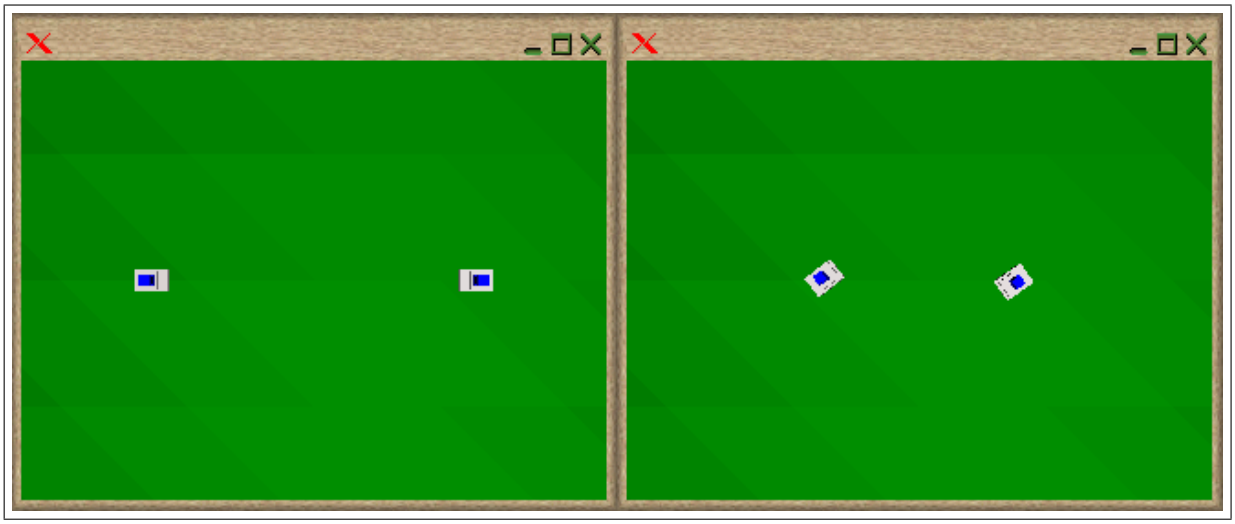
W eksperymencie tym każdy z robotów miał przejechać 4 m przed siebie. Roboty do pomiaru odległości wykorzystywały dalmierz laserowy *SICK LMS 200*. Robot stojący w linii z obiema okrągłymi przeszkodami, o średnicy 10 cm, stał odległy o 2 m od pierwszej z nich. Odległość między przeszkodami wynosiła 1 m. Drugi z robotów odległy był od przeszkody o $2,5$ m. Na rysunku 3.13 pokazany jest stan początkowy robotów (rys. z lewej), oraz końcowy (rys. z prawej).

Rysunek 3.10 przedstawia wykres położenia robotów w trakcie wymijania się nawzajem (rys. 3.9). Roboty osiągnął cel z uchybem mniejszym niż 6 cm. Na rysunku 3.11 znajdują się wykresy prędkości sterujących, natomiast na rysunku 3.12 – wykresy orientacji robotów w trakcie jazdy.

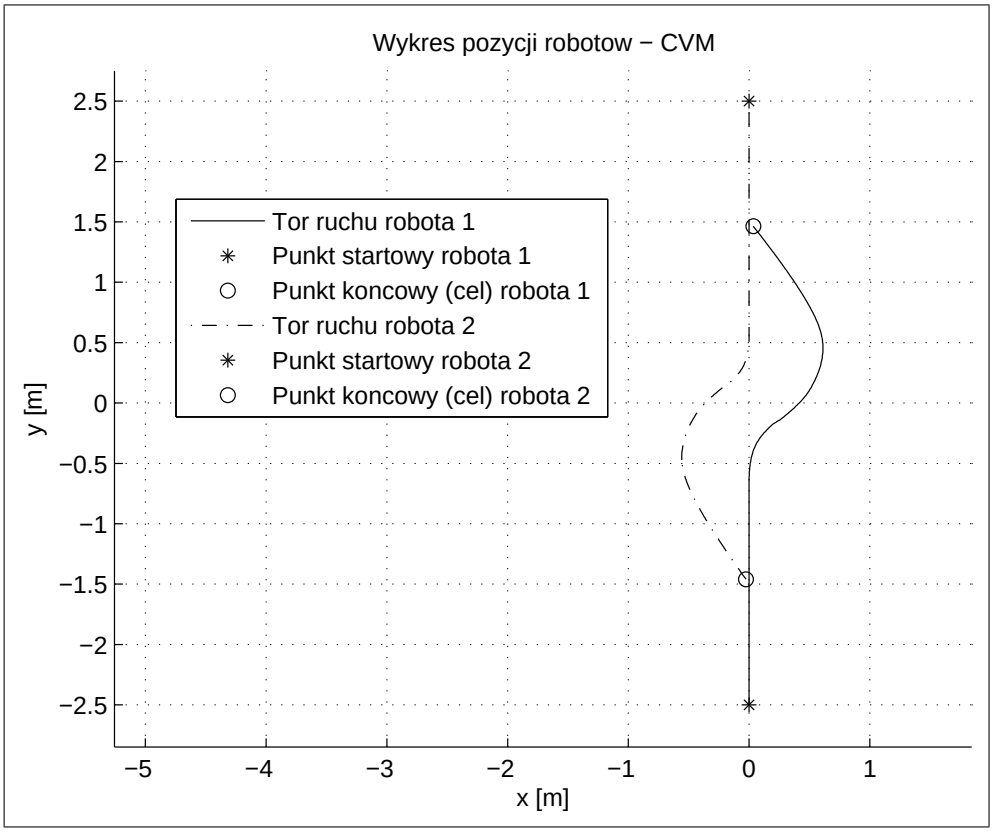
3.6 Wnioski

Wykresy położenia dla zaprezentowanych wyników są gładkie, co jest istotnym faktem. Dodatkowo w większości testów, roboty nie zaklinowały się nigdzie oraz nie zbliżyły się do przeszkód w stopniu, w którym musiałyby się zatrzymać i obrócić w miejscu.

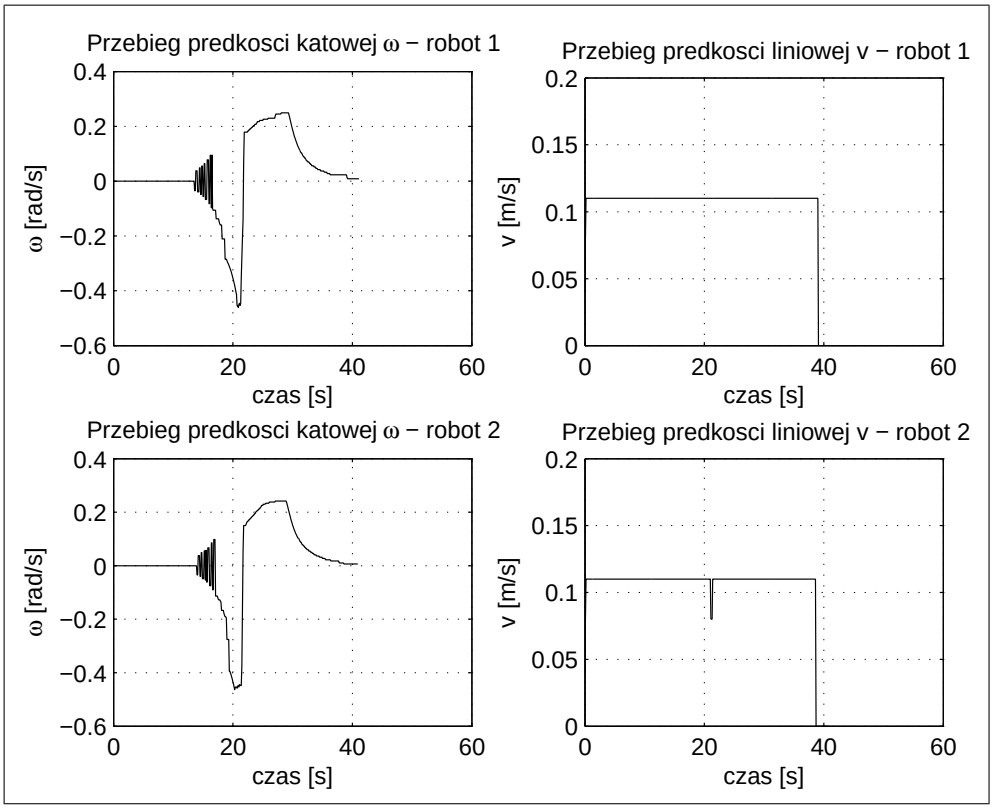
Uzyskane przebiegi prędkości sterujących nie są gładkie, co niekorzystnie wpływa na elementy wykonawcze (silniki w robocie). Jest to spowodowane sposobem doboru prędkości. Algorytm wykorzystuje optymalizację pewnej funkcji kosztu, od której właśnie zależy charakter sterowania. W celu poprawy efektywności obliczeniowej zastosowane zostały funkcje liniowe, które jak widać powodują oscylacje prędkości kątowej.



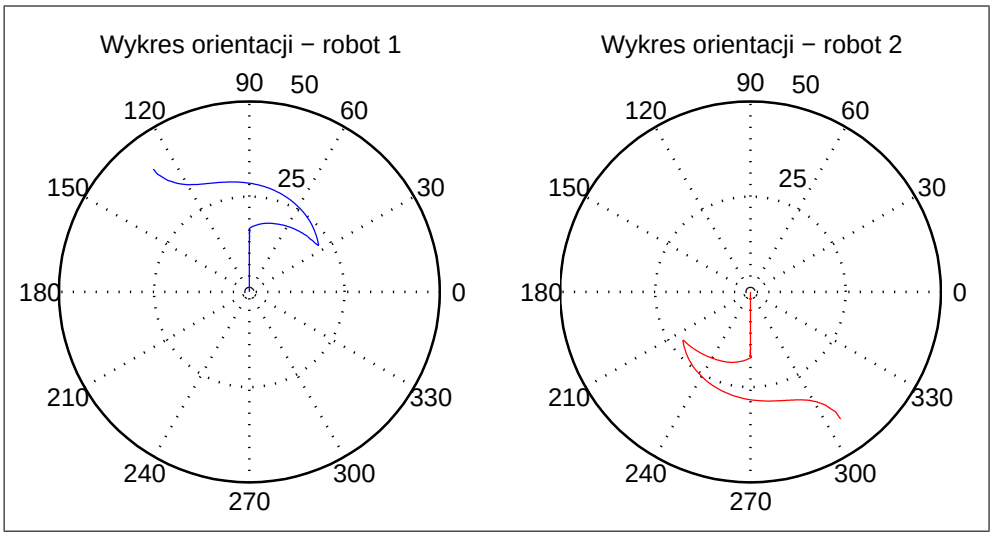
Rysunek 3.9: Wymijanie się robotów nawzajem



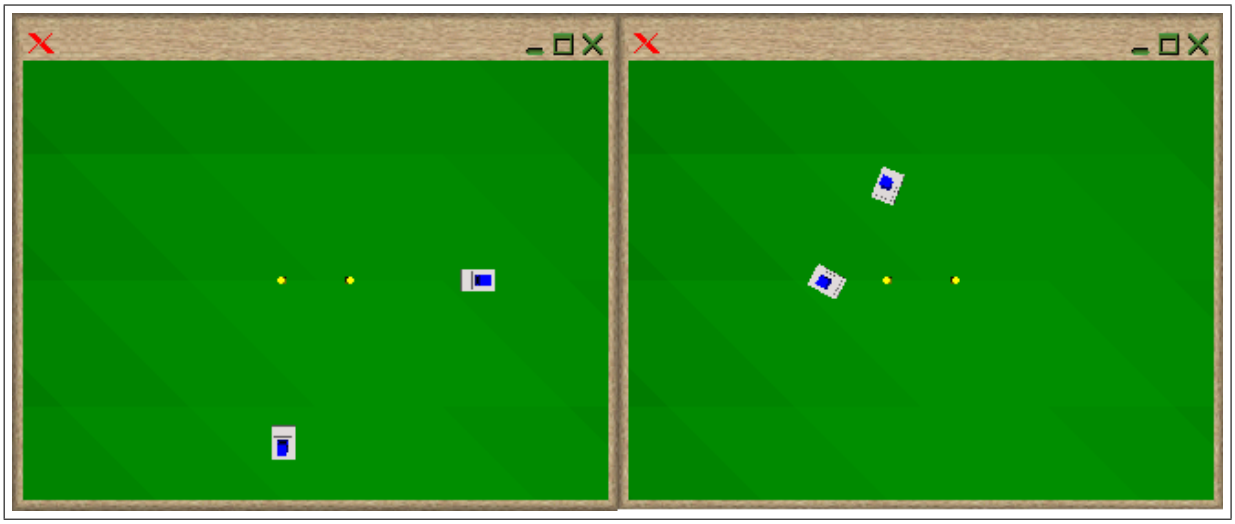
Rysunek 3.10: Wykres położenia robotów w trakcie wymijania się nawzajem



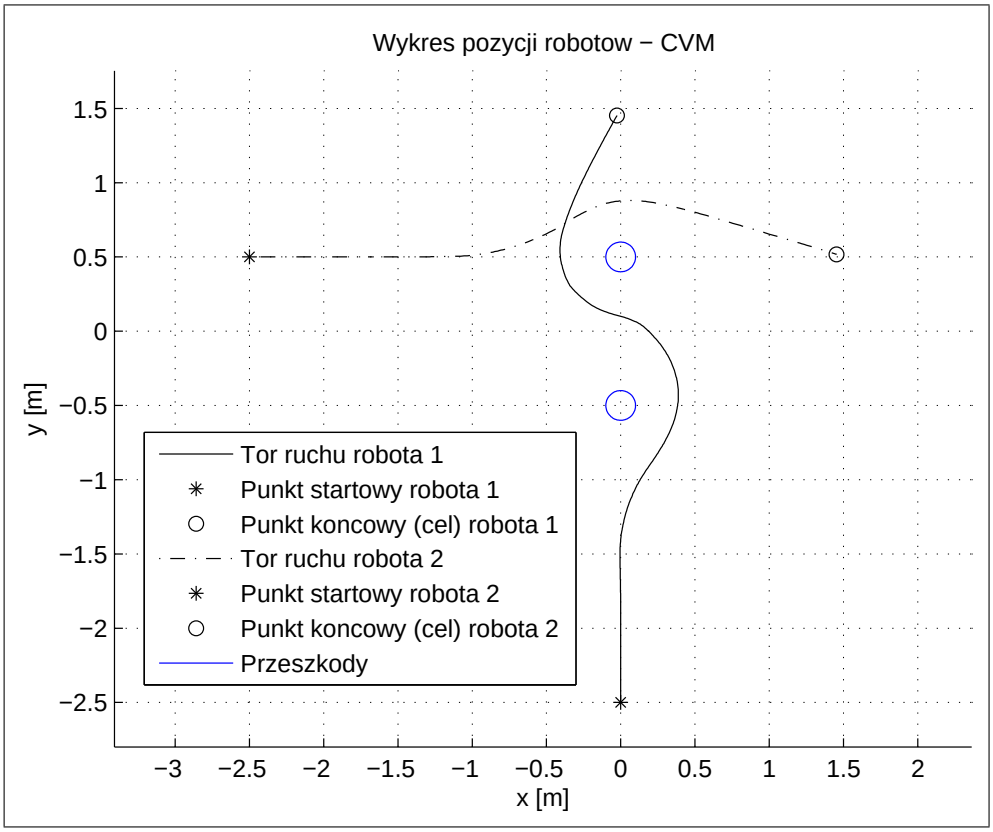
Rysunek 3.11: Przebiegi prędkości robotów w trakcie wymijania się nawzajem



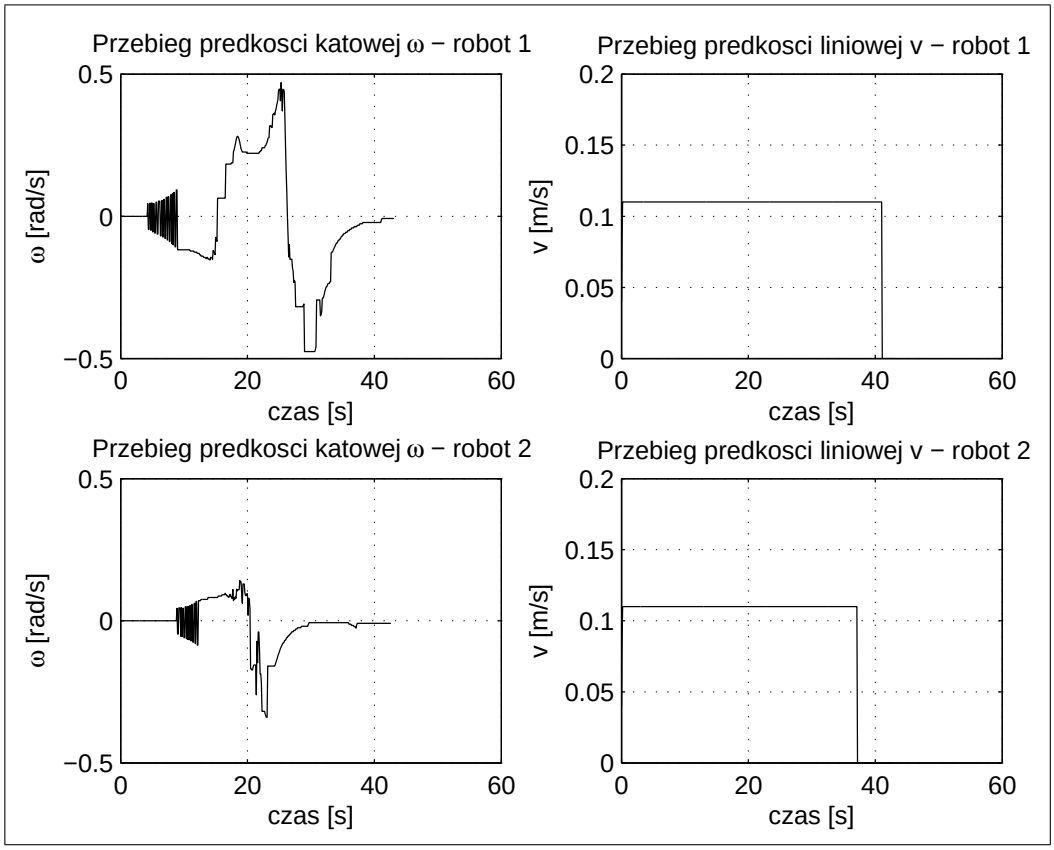
Rysunek 3.12: Wykres orientacji robotów w trakcie wymijania się nawzajem



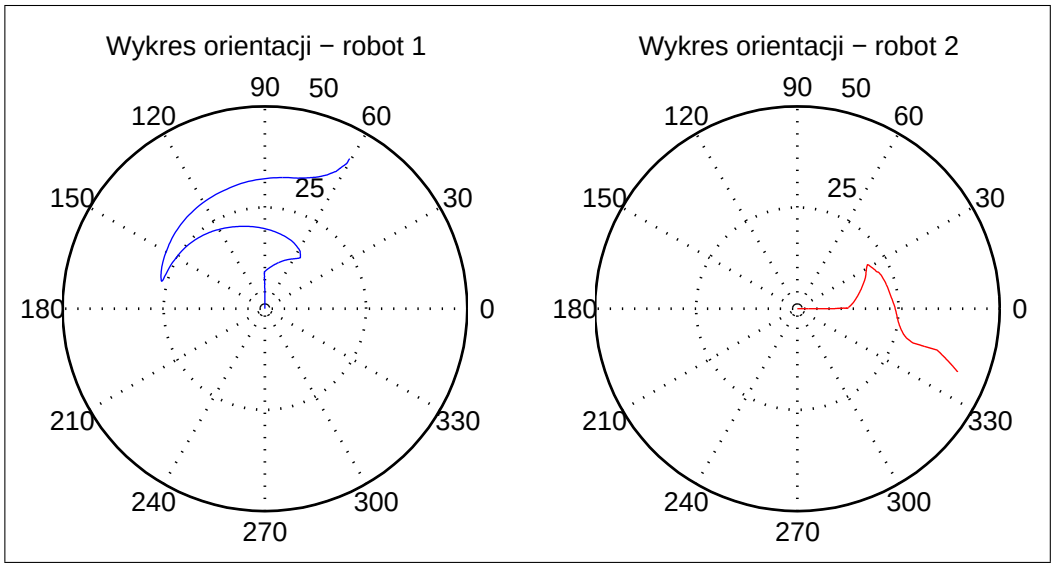
Rysunek 3.13: Omijanie przeszkód przez roboty



Rysunek 3.14: Wykres położenia robotów w trakcie omijania przeszkód



Rysunek 3.15: Przebiegi prędkości robotów w trakcie omijania przeszkód



Rysunek 3.16: Wykres orientacji robotów w trakcie omijania przeszkód

Rozdział 4

Algorytm współpracy zespołu robotów w zadaniu wspólnego przemieszczania obiektu

Sterowanie działaniem robota mobilnego jest procesem znacznie odbiegającym od tradycyjnego przetwarzania komputerowego. Odmienność ta wynika przede wszystkim z faktu, że nie jest realizowane jedno powtarzalne zadanie (do czego początkowo były przewidziane komputery). Zadanie lub inaczej cel działań może nie być ściśle określony, a nawet może zmieniać się podczas pracy robota. Także otoczenie robota mobilnego, w odróżnieniu od manipulatorów przemysłowych często nie jest znane, a za to zmienne oraz trudne do zamodelowania.

W ramach prac zostało uruchomione środowisko programowania heterogenicznych robotów mobilnych. Opracowane i zaimplementowane zostały przykładowe zachowania robotów, których działanie zostało zweryfikowane przez realizacje jednego z typowych zadań testowych dla zespołu robotów mobilnych, jakim jest kooperacyjne transportowanie obiektu.

4.1 Koncepcja sterowania behawioralnego

Jednym z pierwszych podejść do sterowania działaniem robota mobilnego było kontrolowanie zachowań jako *reakcja* na bodźce [3] – sterowanie reaktywne. Ta koncepcja bierze swoje podstawy w obserwacjach zachowań prostych organizmów żywych i jest odwzorowaniem działania instynktu w środowisku naturalnym. Cechuje je przede wszystkim prostota oraz wynikająca z niej skuteczność. Zachowania instynktowne, takie jak unikanie zagrożenia – czyli kolizji, przeszkód lub czynników zagrażających funkcjonowaniu robota (wysoka temperatura, niebezpieczne substancje chemiczne) sprawdzają się doskonale jako metoda przetrwania w nieprzyjaznym środowisku. Korzystając z takiego podejścia do sterowania robotami trudno jednak uzyskać realizację bardziej złożonych zadań, które wymagają przynajmniej częściowego budowania modelu środowiska.

Jednym ze sposobów rozwiązania tego problemu jest koncepcja sterowania behawioralnego, które skrótowo można określić jako „myśl w sposób w jaki działasz“. Tutaj „myślenie” traktowane jest jako podejście do sterowania i programowania robota – jego działanie nie jest opisywane jako reakcje na bodźce (czyli konkretne zdarzenia), ale jako zachowania wynikające z obserwacji środowiska. W takim podejściu kluczowe staje się nie opracowanie systemu reakcji, a dekompozycja zadania na zachowania służące do jego realizacji.

4.2 Sterowanie behawioralne zespołem robotów heterogenicznych

W przypadku sterowania zespołem robotów z wykorzystaniem zachowań możliwe są dwa podejścia. W pierwszym każdy robot może realizować wiele zachowań – wtedy są one dynamicznie (tzn. w trakcie realizacji zadania) przydzielane do poszczególnych robotów przez koordynatora. To podejście najlepiej stosować, gdy wszystkie roboty są homogeniczne, czyli dysponują takimi samymi możliwościami, zestawem sensorów i efektorów. W tym przypadku proces przydziału ról może uwzględniać zmieniające się predyspozycje robotów w trakcie pracy – takie jak choćby stan naładowania akumulatorów, czy awaria jednego z podzespołów. Drugie podejście polega na statycznym przydziale zachowań dla robotów, które może zostać zrealizowane na etapie określania konfiguracji zdania. W tym przypadku określenie ról bazować będzie na informacji o możliwościach każdego robota. Dla przykładu – kryterium przydziału może być moc obliczeniowa, pozycja początkowa, czy dostępność określonego typu czujnika.

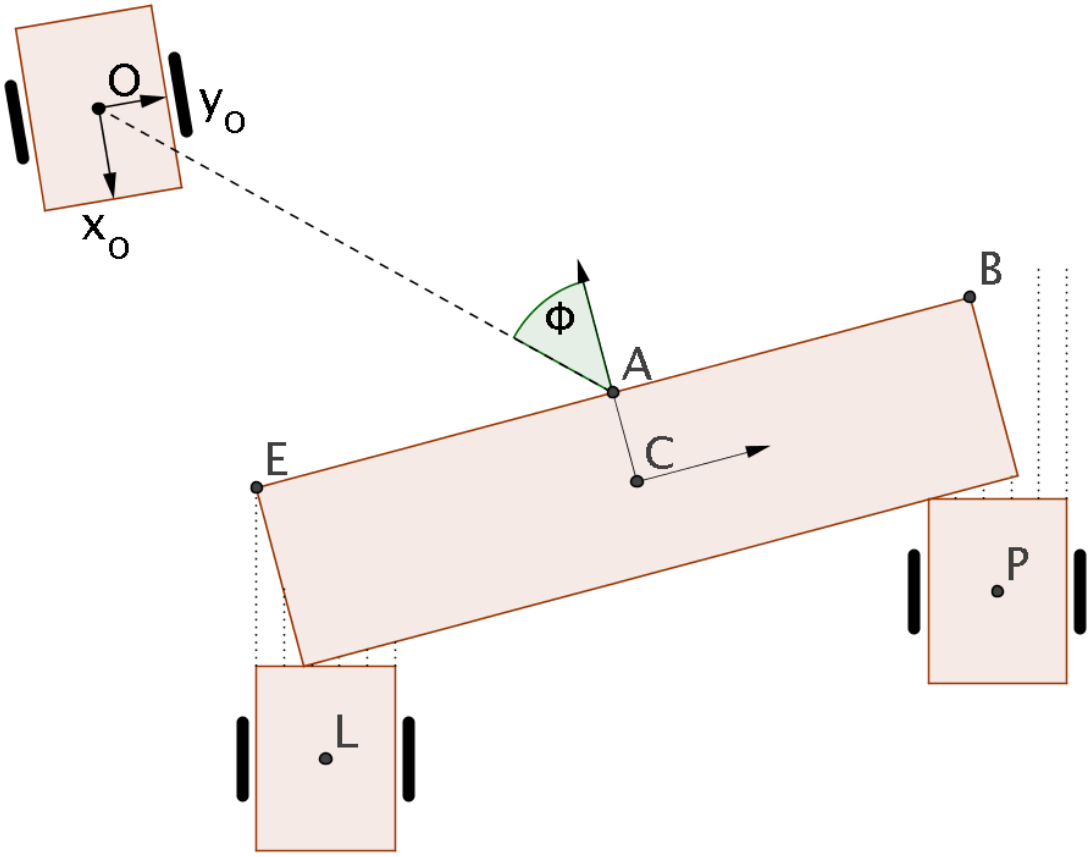
4.3 Przykład zadania wielorobotowego

Jako przykład zadania wykorzystującego sterowanie zespołem heterogenicznych robotów mobilnych zostało wybrane zadanie pchania pudła, gdzie sukces możliwy jest jedynie przy ścisłej współpracy wszystkich elementów systemu robotycznego. Zadanie to jest szeroko znane w literaturze, jednak w większości przypadków było realizowane w sposób, który nie wymagał ciągłej i nieprzerwanej kooperacji.

Kluczowe w trakcie transportu jest to, że obiekt jest pchany bądź ciągnięty przez robota w pożądanym kierunku, który jednocześnie nie może stracić kontaktu z obiektem. Omawiany efekt można uzyskać dzięki m.in. otoczeniu obiektu przez grupę robotów tak, że nie jest możliwe jego wysunięcie się w niepożądanym kierunku [44, 57, 12]. Inne sposoby to np. sztywne uchwycenie obiektu przez roboty, czy oplecenie go liną [16].

Metoda, którą zdecydowano się stosować jest rozwinięciem podejścia określanego jako *pusher–watcher* – gdzie zadanie jest realizowane przez grupę robotów z podziałem na role [26]. Jeden robot, wyposażony w dalmierz laserowy odpowiada za określenie pozycji i orientacji obiektu i na tej podstawie koordynowane jest działanie pozostałych robotów „pchaczy", których dodatkowym zadaniem jest utrzymywanie kontaktu z transportowanym przedmiotem. Wybór tej metody podyktowany był w dużej mierze możliwościami (tj. zestawem czujników) robotów *ELEKTRON*, gdyż wymaga ona i jednocześnie weryfikuje działanie wielu podzespołów:

- dalmierza laserowego, używanego do określenia położenia obiektu,
- czujników zbliżeniowych IR zamontowanych w zderzaku robotów „pchaczy", używanych do utrzymania styczności z transportowanym obiektem,
- układu jezdnego robotów,
- modułu komunikacji WLAN,



Rysunek 4.1: Schemat zadania pchania pudła

- systemu oprogramowania, który musi pozwalać na spójne sterowanie działaniem wielu robotów.

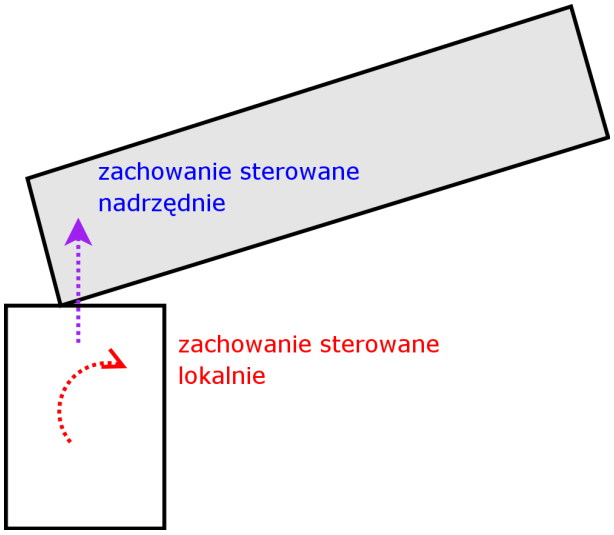
Jednocześnie jest to zadanie, gdzie wyraźnie widać konieczność przydziału różnych ról dla poszczególnych robotów. Dodatkowo, ponieważ określenie globalnej pozycji pudła jest możliwe jedynie przy użyciu dalmierza laserowego (nie można do tego wykorzystać czujników krótkiego zasięgu), zrealizowanie zadania jest możliwe tylko przy jednym, określonym przydziale ról.

4.4 Opis zadania

Rozważane jest następujące zadanie: robot O wyposażony w dalmierz laserowy określa odległość (OA) i orientację (ϕ) obiektu. Roboty pchające (L i P) wyposażone są w m.in. 6 czujników odległości dających odczyt w zakresie ok. $5 \ldots 50cm$ (rys. 4.1).

W trakcie realizacji zadania robot O (prowadzący) jest sterowany zdalnie przez operatora, natomiast roboty pchające realizują dwa zachowania:

- utrzymanie położenia przy krawędzi pudła,
- pchanie z zadaną prędkością.



Rysunek 4.2: Zachowania sterujące ruchem robota pchającego

Pierwsze z nich jest realizowane wyłącznie na bazie informacji z własnych czujników odległości robota. Z powodu bardzo dużych szumów pomiarowych oraz różnic charaterystyk pomiędzy egzemplarzami czujników nie jest w praktyce możliwe dokładne wyznaczanie położenia robota względem krawędzi pudła, gdyż byłoby ono obarczone dużym błędem i ta infomacja byłaby bezużyteczna. Prostą i skuteczną metodą, która została zastosowana jest zliczanie ilości kolejnych odczytów poniżej ustalonego progu wartości (przyjęte zostało 20cm) zaczynając od krawędzi znajdującej się bardziej „wewnątrz" pudła. Na tej podstawie wyznaczana jest składowa prędkości obrotowej ruchu robota ω. Jest to zachowanie, które lokalnie generuje składową sterowania ruchem robota w oparciu wyłącznie o informacje z czujników danego robota (rys. 4.2).

Realizacja drugiego zachowania odbywa się przez taki dobór względnych prędkości w ruchu postępowym w_L i w_P, aby $\phi \to 0$, przy jednoczesnym zapewnieniu, aby przynajmniej jeden robot poruszał się z maksymalną prędkością liniową w_{max}. Oznacza to tyle, że aby ustawić pudło w pożądanym kierunku jeden robot musi jechać szybciej, a drugi wolniej. Jest to zachowanie, które generuje składową sterowania ruchem robota w opraciu wyłącznie o informację o globalnym położeniu pudła, do której robot pchający nie ma bezpośredniego dostępu (rys. 4.2).

Zasadniczą różnicą w przyjętej metodzie w porównaniu z innymi pracami jest to, że regulacja orientacji pudła odbywa się w sposób ciągły, bez wyróżniania fazy obrotu do zadanego kąta i dalej pchania na wprost. Dzięki temu możliwe jest płynne przesuwanie obiektu po linii wyznaczanej ruchem robota wodzącego.

Rola robota prowadzącego polega na analizie danych z dalmierza laserowego i na ich podstawie określenia położenia i orientacji pudła. Mając do dyspozycji te informacje do robotów pchających przesyłana jest zadana składowa prędkości w ruchu postępowym.

Rozdział 5

Opracowanie i implementacja algorytmów sterowania behawioralnego zespołem heterogenicznych robotów mobilnych

5.1 Implementacja

W celu realizacji zadania wielorobotowego został stworzony program klienta *Player*, który bezpośrednio łączy się z serwerami działającymi na trzech robotach. Do komunikacji wykorzystywana jest biblioteka C++ *playerclient*. Do wyznaczania położenia i orientacji pudła wykorzystywany jest sterownik *laserfeaturex* bazujący na danych z dalmierza laserowego. Wynikiem jego działania jest tablica z lokalizacją we współrzędnych układu związanego z dalmierzem laserowym spójnych obiektów tworzących odcinki prostych. Układ ten pokrywa się z układem związanym ze środkiem robota wodzącego O.

Na podstawie współrzędnych początku i końca odcinka wyznaczany jest kąt ϕ oraz odległość OA do pudła. Przyjmując współrzędne punktów B i E w układzie $(\overrightarrow{x_o}, \overrightarrow{y_o})$ jako:

$$B = (x_B, y_B)\,, \qquad E = (x_E, y_E)$$

długość krawędzi obiektu można wyrazić jako:

$$BE = \sqrt{(x_E - x_B)^2 + (y_E - y_B)^2}$$

Na podstawie długości odcinka BE dokonywane jest podstawowe stwierdzenie, czy znaleziony obiekt opisuje krawędź pchanego pudła o długości 180cm:

$$\begin{cases} \text{dla} & |BE - 180cm| \leqslant 20cm & \Rightarrow \text{pudło} \\ \text{dla} & |BE - 180cm| > 20cm & \Rightarrow \text{inny obiekt} \end{cases}$$

Współrzędne środka krawędzi pudła A wyrażają się wzorem:

$$A = (x_A, y_A) = \left(\frac{x_B + x_E}{2}, \frac{y_B + y_E}{2}\right)$$

zaś odległość środka robota O do środka krawędzi pudła:

$$OA = \sqrt{(x_A)^2 + (y_A)^2}$$

Na podstawie twierdzenia o iloczynie skalarnym wektorów $\overrightarrow{OA}$ i $\overrightarrow{BE}$:

$$\cos \angle OAE = \frac{(x_B - x_E) \cdot x_A + (y_B - y_E) \cdot y_A}{OA \cdot BE}$$

i stąd kąt ϕ wyrażony w stopniach:

$$\phi = \arccos\left(\cos \angle OAE\right) \frac{180^\circ}{\pi} - 90^\circ$$

Zadanie można traktować jako układ regulacji, gdzie uchybem jest kąt ϕ, natomiast od wyjścia regulatora u zależy różnica prędkości robotów pchających. Początkowo planowane było zastosowanie prostego regulatora proporcjonalnego, jednak w wyniku eksperymentów okazało się konieczne wprowadzenie także członu całkującego, którego zadaniem jest likwidacja uchybu statycznego powstającego w wyniku różnic w tarciu o podłoże lewego i prawgo robota pchającego.

Zastosowany układ regulacji w idealnym przypadku jest układem ciągłym, jednak dla celów implementacji musiał zostać zrealizowany jako układ dyskretny. Czas próbkowania T_p wynosi 100ms (czyli częstotliwość 10Hz), co odpowiada domyślnej wartości w oprogramowaniu *Player/Stage*, jednak może być łatwo modyfikowane. Przy poruszaniu się robota z maksymalną prędkością w czasie 100ms pokonuje on jedynie ok. 2.5cm, zatem taki czas próbkowania wydaje się wystarczający.

Dla celów zapisu formalnego algorytmu wprowadźmy oznaczenia:

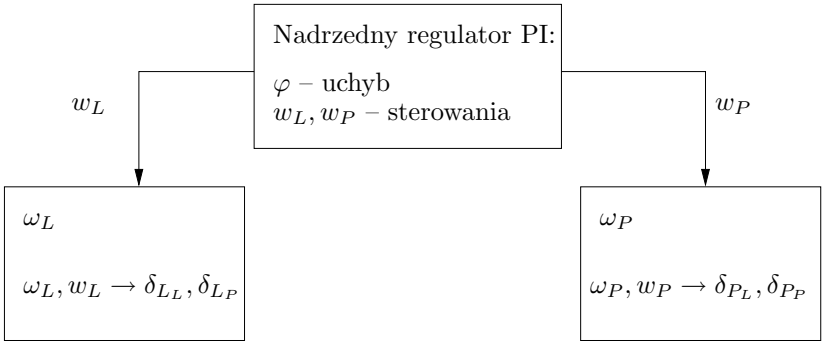
- ω_L, ω_P – prędkość obrotowa odpowiednio lewego i prawego robota pchającego,
- w_L, w_P – prędkość liniowa odpowiednio lewego i prawego robota pchającego,

Prawo sterowania można zapisać w następujący sposób (indeks górny ^{i} oznacza i-tą iterację algorytmu dyskretnego):

$$\begin{cases} u^i = u^{i-1} + K\left(1 + \frac{T_p}{2T_i}\right)\frac{\phi}{\pi/2} + K\left(\frac{T_p}{2T_i} - 1\right)\frac{\phi}{\pi/2} & \\ \begin{cases} w_L^i = w_{max}\left(1 - u^i\right) \\ w_P^i = w_{max} \end{cases} & \text{dla } \phi > 0 \\ \begin{cases} w_L^i = w_{max} \\ w_P^i = w_{max} \end{cases} & \text{dla } \phi = 0 \\ \begin{cases} w_L^i = w_{max} \\ w_P^i = w_{max}\left(1 - u^i\right) \end{cases} & \text{dla } \phi < 0 \end{cases}$$

Prędkości obrotowe ω_L i ω_P są tak ustalane, aby orientować robota do położenia, w którym krawędź pudła będzie znajdowała się pośrodku przedniego zderzaka robota. Jako kryterium przyjmowana jest tutaj liczba kolejnych odczytów z czujników począwszy od znajdującego się bliżej środka pudła, które nie przekraczają ustalonego progu 20cm. Wedle tego kryterium prędkość obrotowa robota jest ustalana na wartość z tablicy eksperymentalnie dobranych wartości. W momencie kiedy odległość $OA < 1$m ruch robotów jest wstrzymywany.

Warto podreślić warstwową strukturę algorytmu – realizacja polega na dekompozycji działań robota przy wykorzystaniu składowych prędkości jego ruchu (rys. 5.1):



Rysunek 5.1: Warstwowa dekompozycja sterowania i zachowań robotów

- obrotową, za które odpowiadają jedynie informacje „lokalne" dla robota,
- liniową, za którą odpowiada informacja o „globalnym" położeniu i orientacji przepychanego pudła.

Konieczność eksperymentalnego doboru współczynników związanych z regulacją prędkości wynika z faktu, że podczas pchania tracona jest przyczepność kół robota do podłoża, zatem informacja z enkoderów umieszczonych na osiach kół staje się w praktyce mało użyteczna. Z tego samego powodu opieranie zadania na bardziej rozbudowanym modelu byłoby niecelowe.

Komunikacja programu sterującego z robotami realizowana jest z użyciem klasy *PlayerMultiClient* biblioteki C++ z pakietu *Player/Stage*, która jest konieczna do zapewnienia działania pętli głównej programu jako przesłanie sterowań w reakcji na przetworzenie danych z czujników wszystkich robotów (tab. 5.1).

5.2 Realizacja zadania na symulatorze

Zadanie zostało przygotowane z wykorzystaniem trójwymiarowego symulatora *Gazebo*, modelującego zachowanie i oddziaływania między obiektami. Zrzut ekranu przedstawia przebieg symulacji – widoczne są okna wizualizacji (lewa górna część), aplikacji rejestrującej ruch robotów i pudła (lewa dolna część), programu *playerv* do ręcznego sterowania robota wodzącego O (prawa górna część) oraz aplikacji sterującej ruchem robotów pchających z zaznaczonym żółtym kolorem odcinkiem reprezentującym położenie pudła (prawy dolny róg) (rys. 5.2).

5.3 Eksperymenty

W trakcie eksperymentów robot wodzący był nieruchomy, zaś zadaniem robotów pchających była współpraca mająca na celu minimalizowanie kąta ϕ pomiędzy prostopadłą do pudła poprowadzoną z jego środka – a półprostą poprowadzoną z tego punktu w kierunku środka robota wodzącego (rys. 4.1). Pchane pudło miało wymiary ok. $180 \times 20 \times 20$cm i wagę kilkunastu kilogramów (rys. 5.9, 5.10).

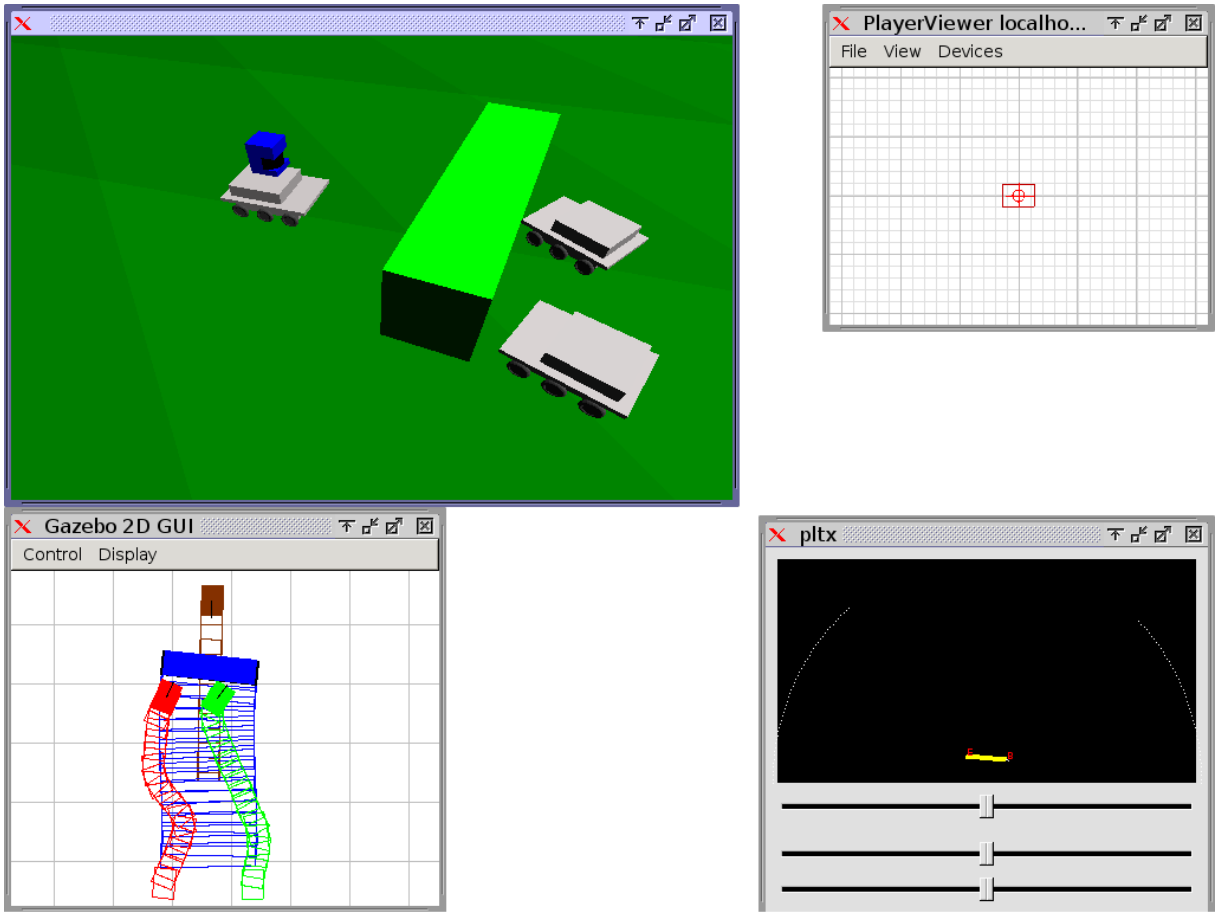
```
#include <playerclient.h>

// inicjalizacja połączenia z robotami
PlayerClient watcher("pcm1",6665,PLAYER_TRANSPORT_TCP);
PlayerClient pusher1("pcm3",6665,PLAYER_TRANSPORT_TCP);
PlayerClient pusher2("pcm4",6665,PLAYER_TRANSPORT_TCP);

// tworzenie obiektu do zarządzania komunikacją
PlayerMultiClient mc;
mc.AddClient(&watcher);
mc.AddClient(&pusher1);
mc.AddClient(&pusher2);

while(true) {
  // odczytanie stanu wszystkich robotów
  do {
    mc.Read();
  } while(!watcher.fresh || !pusher1.fresh || !pusher2.fresh);
  watcher.fresh = pusher1.fresh = pusher2.fresh = false;

  // przetworzenie danych z czujników
  // i~przesłanie sterowań do robotów
  // ...
}
```

Tabela 5.1: Schemat programu sterującego wieloma robotami

Poniżej zamieszczono uzyskane trajektorie transportowanego obiektu (z zaznaczaniem położenia co 10 sekund) oraz wykresy odczytów z czujników odległości robotów pchających dla dwóch przebiegów na rzeczywistych robotach oraz jednego przebiegu na symulatorze (rys. 5.4, 5.6, 5.8). Pudło było początkowo łagodnym łukiem nakierowywane na robota wodzącego, a następnie pchane w kierunku celu po praktycznie prostym odcinku. Ostatni fragment (odległość poniżej 1m) to pchanie przez jednego robota mające na celu ustawienie pudła możliwie równolegle do robota wodzącego tak, aby znajdowało się w korzystnej pozycji przed rozpoczęciem kolejnego fragmentu ścieżki.

Małe wartości odczytów z czujników IR odpowiadają czujnikom zasłoniętym przez pudło, natomiast duże to odczyty z czujników, przed którymi znajdowała się wolna przestrzeń. Odczyty z czujników znajdujących się od strony środka pudła nie przekraczają kilku centymetrów, natomiast odczyty z czujników znajdujących się po zewnętrznej stronie nie są mniejsze niż kilkadziesiąt centymetrów – znacznie powyżej progu ustalonego na 20cm. Krawędź pudła przez większość czasu znajdowała się przed trzecim czujnikiem (licząc od wewnętrznej strony) umieszczonym w przednim zderzaku robota. Wartości odczytów dla tego czujnika oscylują pomiędzy bardzo małymi i bardzo dużymi – co odzwierciedla naprzemienne zasłanianie i odsłanianie czujnika przez pudło.

W dalszej kolejności umieszczono wykresy kąta ϕ traktowanego w tym przypadku jako uchyb regulacji (rys. 5.3, 5.5, 5.7). W celu likwidacji stałej różnicy kąta ϕ konieczne



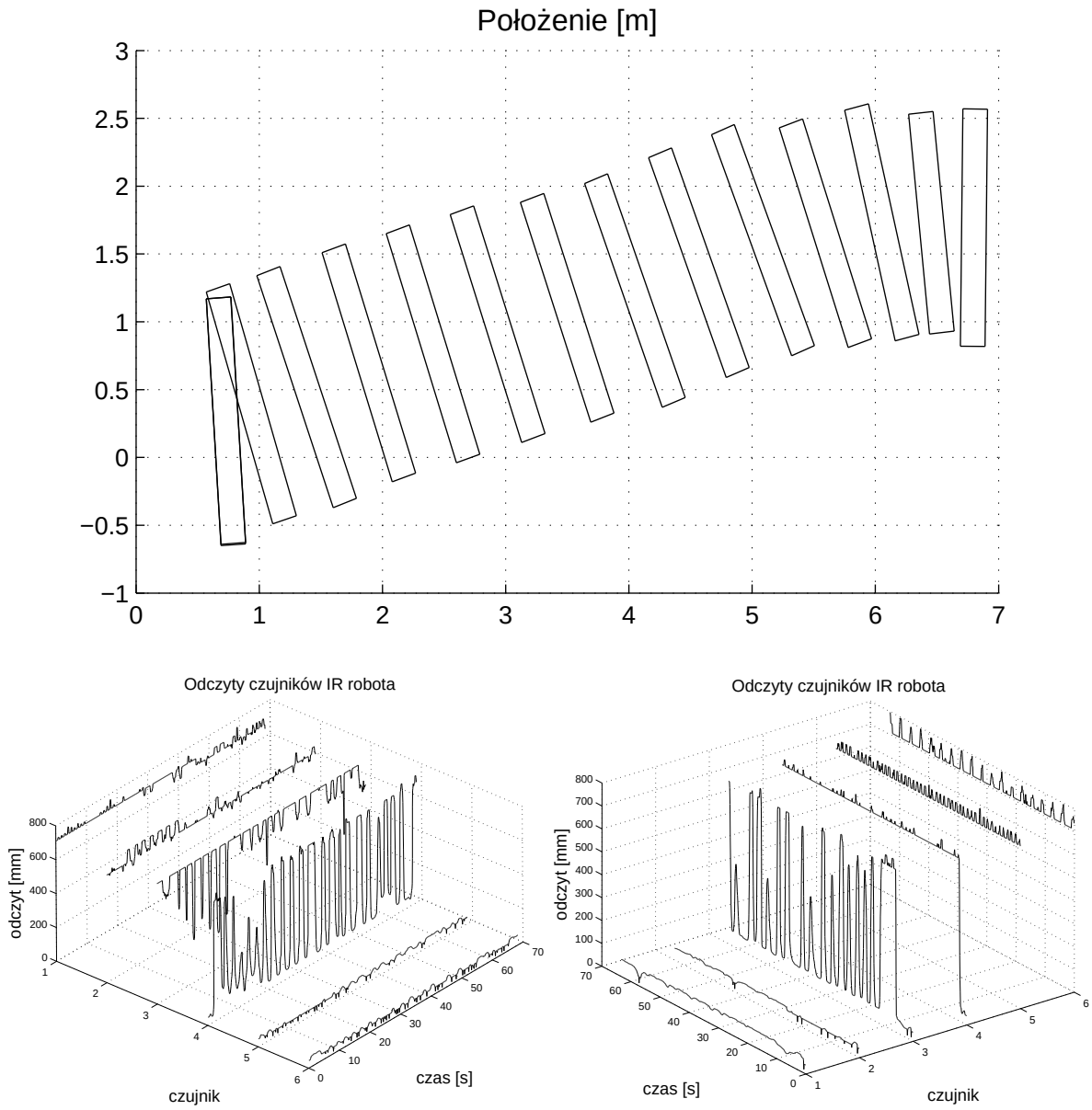
Rysunek 5.2: Realizacja pchania pudła na symulatorze

okazało się wprowadzenie do równań stanowiących w istocie regulator proporcjonalny członu całkującego – czego konsekwencją jest także widoczne przeregulowanie przebiegu na wykresach.

Wykresy prędkości obrotowej robota (ω_L i ω_R) przedstawiają proces regulacji orientacji robota tak, aby krawędź pudła znajdowała się na środku jego zderzaka. Wartości prędkości oscylują pomiędzy ± 0.2 [rad/sek], co odpowiada skrętom w prawo i lewo korygującym ustawienie robota względem krawędzi pudła. Kierunek obracania był zmieniany co około $0.7s$.

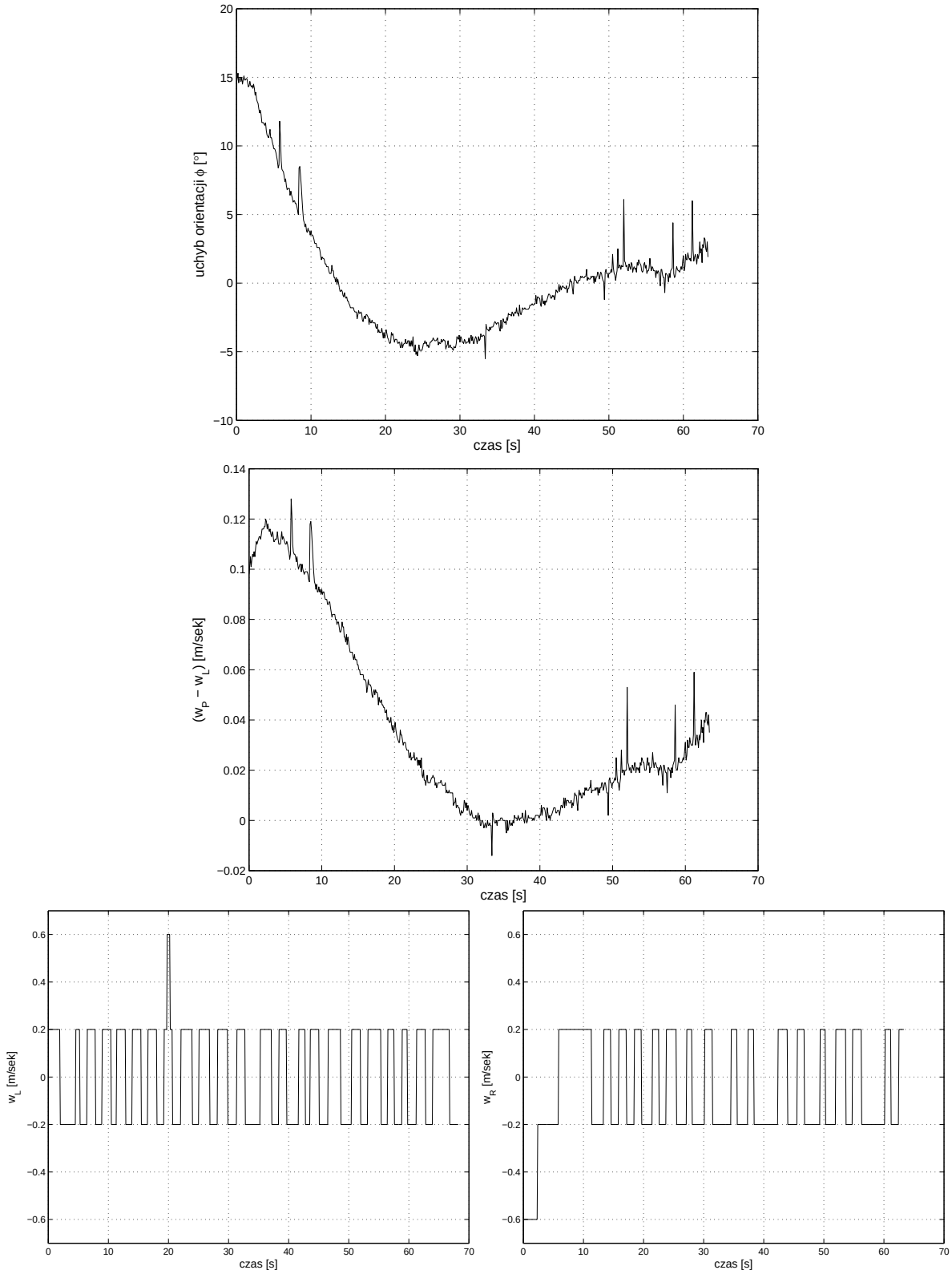
Testowana była komunikacją między robotami bazująca na protokole TCP oraz UDP, jednak nie zostały zaobserwowane żadne różnice związane z wydajnością czy niezawodnością. Komunikacja oparta na protokole TCP w sieciach WLAN może generować duże opóźnienia związane z koniecznością retransmisji utraconych pakietów. W niektórych przypadkach może okazać się korzystne zrezygnowanie z niezawodności transmisji dla niedopuszczenia możliwości powstania dużych opóźnień. W czasie realizacji tego zadania wymiana danych dla wszystkich robotów nie przekraczała łącznie 10kB/sek.

W trakcie realizacji zadania przez rzeczywiste roboty prędkości w^i oraz ω^i uzyskiwane na podstawie odometrii z enkoderów umieszczonych na osiach kół obarczone były dużym błędem – co wynikało z faktu poślizgu robota w trakcie pchania ciężkiego pudła. Zada-

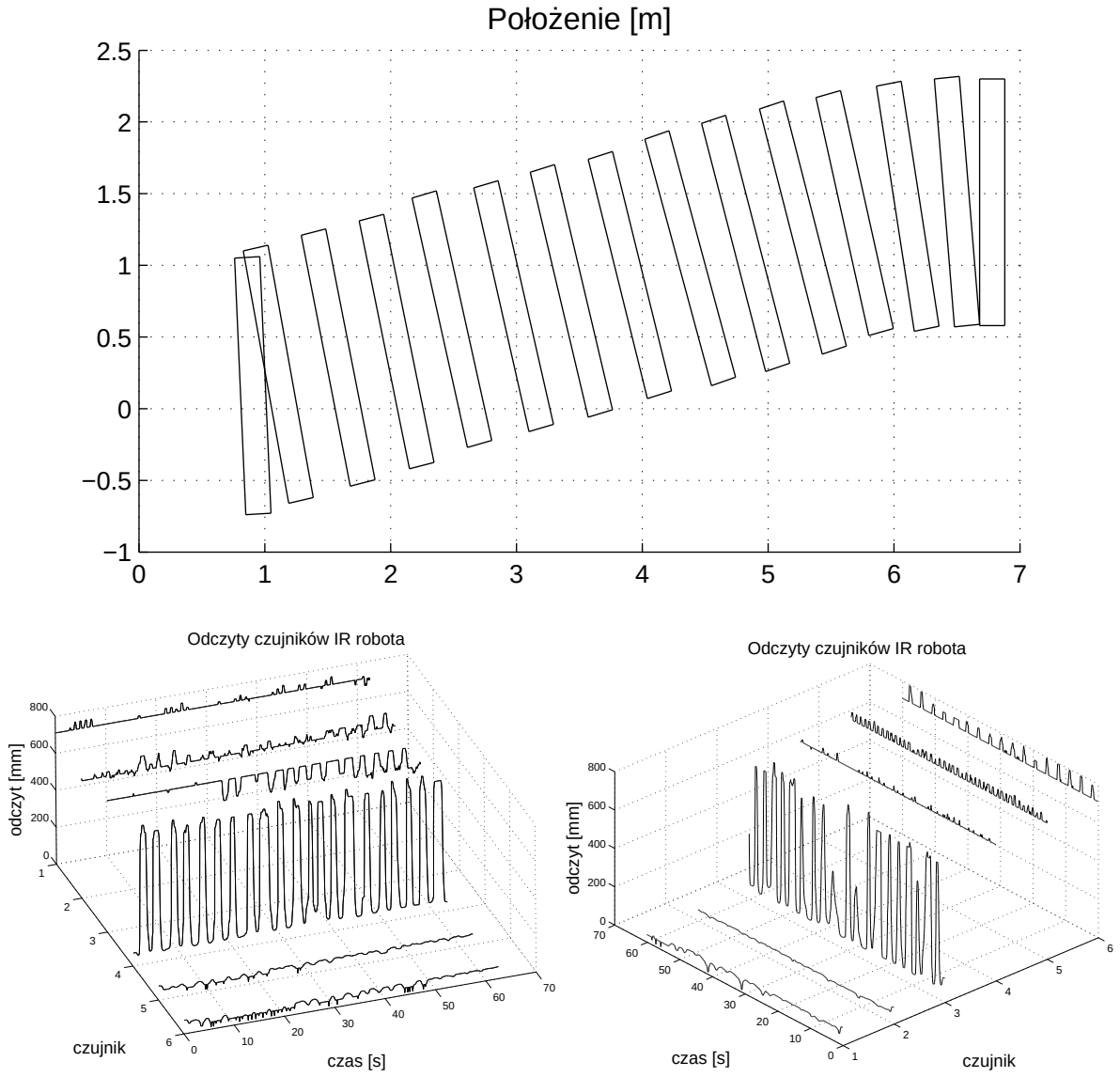


Rysunek 5.3: Wykresy położenia pudła oraz odczytów czujników IR robotów (przebieg 1)

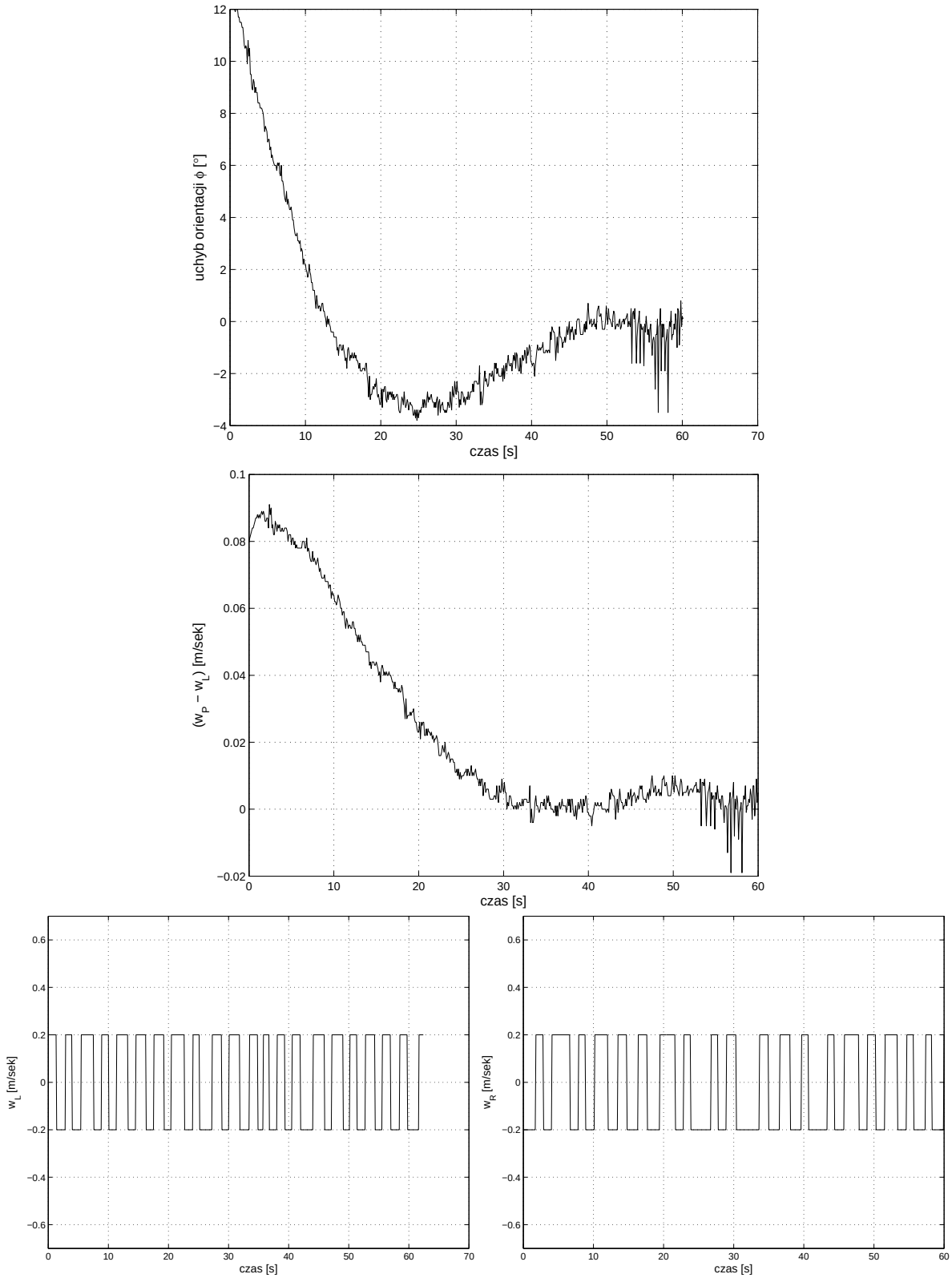
nie było wykonywane przez roboty praktycznie niezawodnie. Problemy, które powodowały brak sukcesu w realizacji zadania były spowodowane głównie resetowaniem się układu mikrokontrolera na karcie PC/104 sterującego silnikami oraz akwizycją danych z czujników IR. Błędy w lokalizacji pudła były także powodowane nieidealnie poziomym ustawieniem skanera laserowego na obrotnicy.



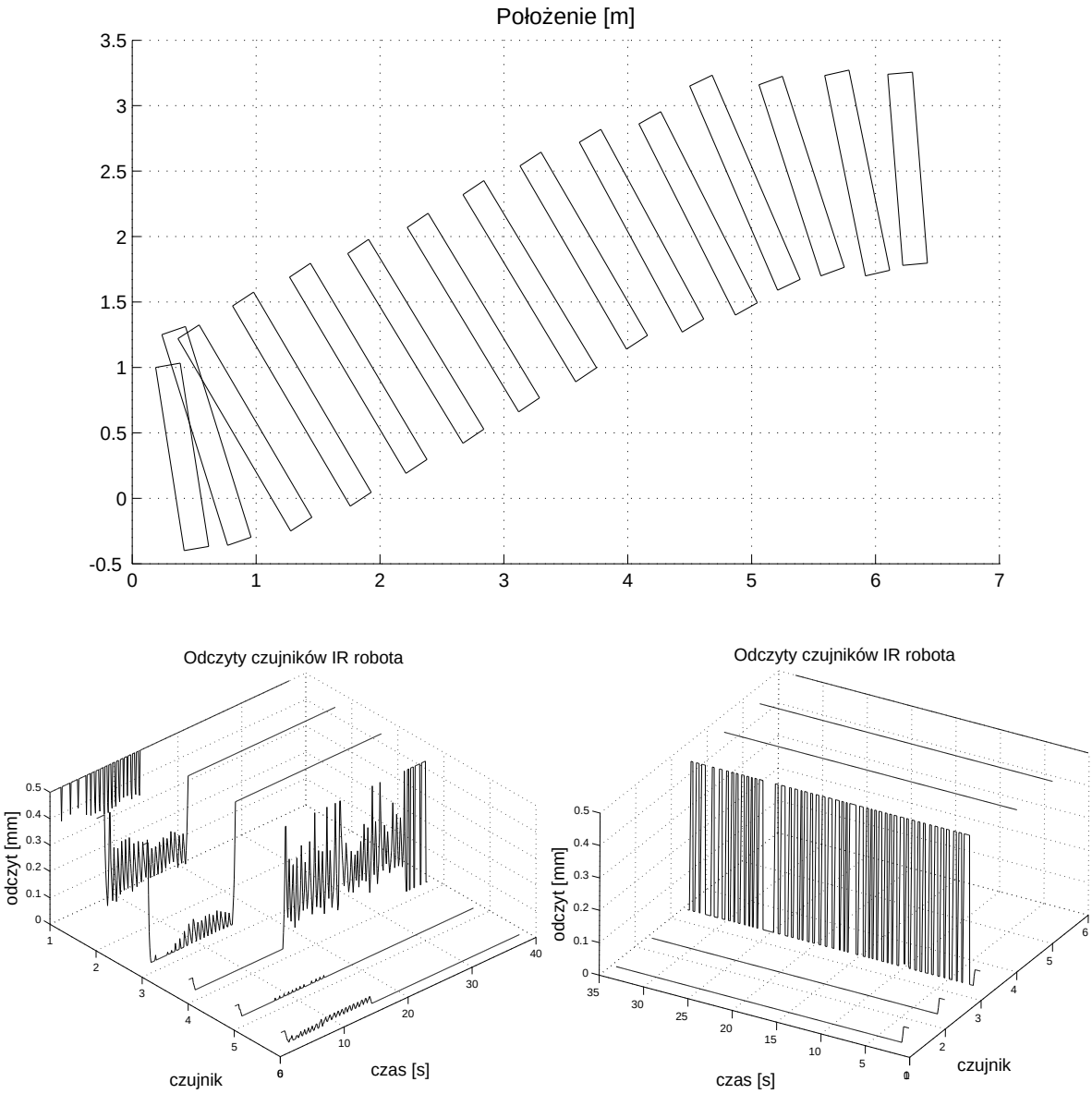
Rysunek 5.4: Wykres uchybu regulacji, różnicy zadawanych prędkości liniowych oraz prędkości obrotowych lewego i prawego robota pchającego (przebieg 1)



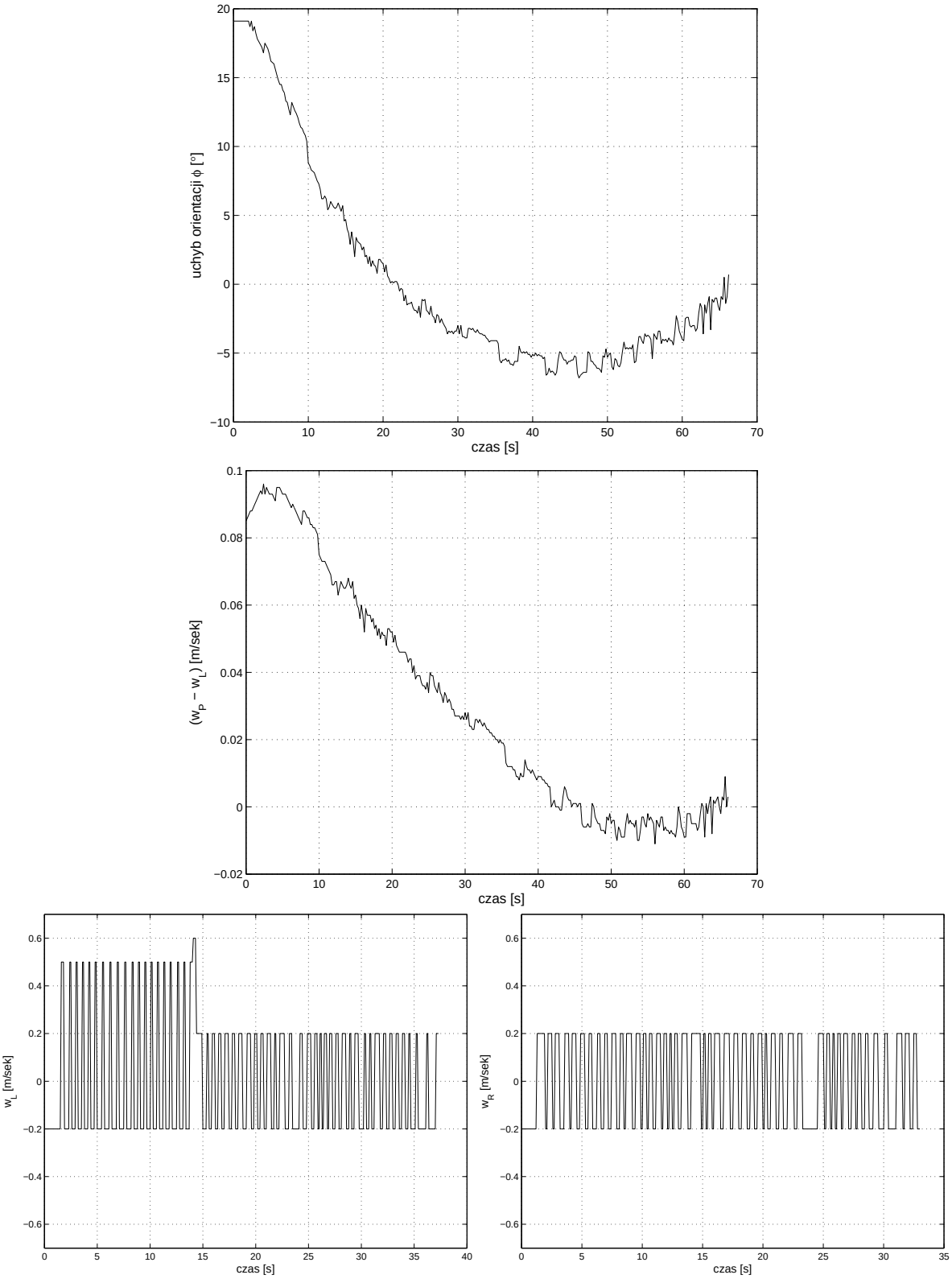
Rysunek 5.5: Wykresy położenia pudła oraz odczytów czujników IR robotów (przebieg 2)



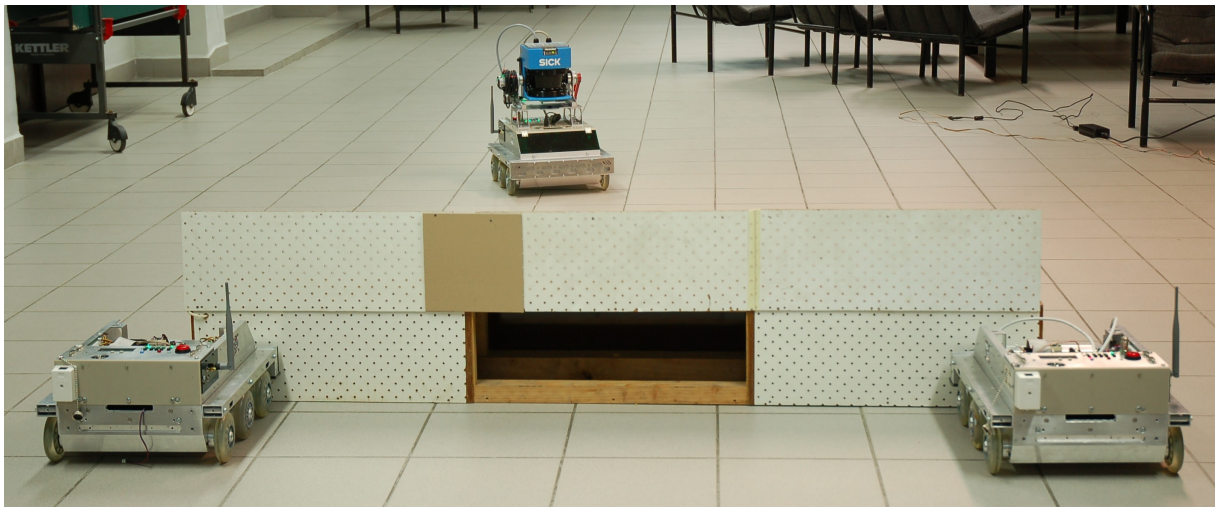
Rysunek 5.6: Wykres uchybu regulacji, różnicy zadawanych prędkości liniowych oraz prędkości obrotowych lewego i prawego robota pchającego (przebieg 2)



Rysunek 5.7: Wykresy położenia pudła oraz odczytów czujników IR robotów w trakcie pchania pudła (przebieg na podstawie symulacji)



Rysunek 5.8: Wykres uchybu regulacji, różnicy zadawanych prędkości liniowych oraz prędkości obrotowych lewego i prawego robota pchającego (przebieg na podstawie symulacji)



Rysunek 5.9: Realizacja pchania pudła – widok od strony robotów pchających

Rysunek 5.10: Realizacja pchania pudła – widok od strony robota wodzącego

Bibliografia

[1] E. U. Acar and H. Choset. Sensor-based coverage of unknown environments: Incremental construction of morse decompositions. *International Journal Robotics Research*, 21(4):345–366, 2002.

[2] R. C. Arkin. *Behavior-Based Robotics*. MIT Press, Cambridge, Mass., 1998.

[3] Ronald C. Arkin. *Behavior-Based Robotics*. MIT Press, Cambridge, Mass., 1998.

[4] T. Balch and A.C. Arkin. Behavior-based formation control for multirobot teams. *IEEE Transactions on Robotics and Automation*, 14(6):926–939, December 1998.

[5] M. Beetz, T. Schmitt, R. Hanek, S. Buck, F. Stulp, D. Schröter, and B. Radig. The agilo robot soccer teamexperience-based learning and probabilistic reasoning in autonomous robot control. *Autonomous Robots*, 17(1):55–77, January 2004.

[6] J. Borenstein and Y. Koren. The vector field histogram – fast obstacle avoidance for mobile robots. *IEEE Trans. on Robotics and Automation*, 7(3):278–288, 1991.

[7] O. Brock and O. Khatib. High-speed navigation using the global dynamic window approach. In *Proc. IEEE Int. Conf. Robotics and Automation*, pages 341–346, 2002.

[8] R. R. Brooks, P. Ramanathan, and A. Sayeed. Distributed target classification and tracking in sensor networks. *Proc. IEEE*, 91:11631171, August 2003.

[9] W. Burgard, M. Moors, C. Stachniss, and F.E Schneider. Coordinated multi-robot exploration. *IEEE Transactions on Robotics*, 21(3):376–386, June 2005.

[10] U.Y. Cao, A.S. Fukunaga, and A.B. Kahng. Cooperative mobile robotics: Antecedents and directions. *Autonomous Robots*, 4(1):7–27, Jan.-Feb. 1997.

[11] L. Chaimowicz, V. Kumar, and M. F. M. Campos. A paradigm for dynamic coordination of multiple robots. *Autonomous Robots*, 17(1):7–21, July 2004.

[12] Luiz Chaimowicza, Vijay Kumarb, and Mario Camposa. A framework for coordinating multiple robots in cooperative manipulation tasks. In *Proceedings of SPIE 4571 – Sensor Fusion and Decentralized Control*, pages 331–336, November, 2001.

[13] H. Choset. Coverage for robotics – a survey on recent results. In *Annals of Mathematics and Artificial Intelligence*, volume 31, pages 113–126. Norwell, MA: Kluwer, 2001.

[14] P. Corke, R. Peterson, and D. Rus. Localization and navigation assisted by networked cooperating sensors and robots. *International Journal Robotics Research*, 24(9):771–786, 2005.

[15] J. Cortés, S. Martínez, T. Karatas, and F. Bullo. Coverage control for mobile sensing networks. *IEEE Transactions on Robotics and Automation*, 20(2):243–255, April 2004.

[16] Bruce Donald, Larry Gariepy, and Daniela Rus. Experiments in constrained prehensile manipulation: Distributed manipulation with ropes. In *Proceedings of the 1999 International Symposium on Experimental Robotics*, 1999.

[17] G. Dudek, M. Jenkin, and E. Milios. A taxonomy of multirobot systems. In *Robot teams: From diversity to polymorphism, T. Balch and L. E. Parker, Eds.* Wellesley, MA: AK Peters, 2002.

[18] G. Dudek, M. Jenkin, E. Milios, and D. Wilkes. A taxonomy for multi-agent robotics. *Autonomous Robots*, 3(4):375–397, December 1996.

[19] I. Dulęba. *Metody i algorytmy planowania ruchu robotów mobilnych i manipulacyjnych.* Akademicka Oficyna Wydawnicza EXIT, Warszawa, 2001.

[20] I. H. Elhajj, A. Goradia, N. Xi, Ch. M. Kit, Y. H. Liu, and T. Fukuda. Design and analysis of internet-based tele-coordinated multi-robot systems. *Autonomous Robots*, 15(3):237–254, November 2003.

[21] A. Farinelli, L. Iocchi, and D. Nardi. Multirobot systems: a classification focused on coordination. *IEEE Transactions on Systems, Man and Cybernetics, Part B*, 34(5):2015–2028, October 2004.

[22] J. T. Feddema, Ch. Lewis, and D. A. Schoenwald. Decentralized control of cooperative robotic vehicles: Theory and application. *IEEE Transactions on Robotics and Automation*, 18(5):852–864, October 2002.

[23] D. Fox, W. Burgard, and S. Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics and Automation Magazine*, 4(1):23–33, 1997.

[24] Ch. H. Fua and S. S. Ge. Cobos: Cooperative backoff adaptive scheme for multirobot task allocation. *IEEE Transactions on Robotics*, 21(6):1168–1178, December 2005.

[25] B. P. Gerkey and M. J. Matarić. A formal analysis and taxonomy of task allocation in multi-robot systems. *International Journal Robotics Research*, 23(4), September.

[26] Brian P. Gerkey and Maja J. Mataric. Pusher-watcher: An approach to fault-tolerant tightly-coupled robot coordination. In *Proceeding of the IEEE International Conference on Robotics and Automation (ICRA 2002)*, pages 464–469, 2002.

[27] T. L. Huntsberger, A. Trebi-Ollennu, H. Aghazarian, P. S. Schenker, P. Pirjanian, and H. D. Nayar. Distributed control of multi-robot systems engaged in tightly coupled tasks. *Autonomous Robots*, 17(1):79–92, July 2004.

[28] David Jung and Alexander Zelinsky. An architecture for distributed cooperative planning in a behaviour-based multi-robot system. *Robotics and Autonomous Systems*, 26(2-3):149–174, 1999.

[29] W. Kasprzak, W. Szynkiewicz, and M. Karolczak. Global colour image features for discrete self-localization of an indoor vehicle. In *Computer Analysis of Images and Patterns, Proc. 11th Int. Conf., CAIP 2005. Lecture Notes in Computer Science 3691*, pages 620–627. Springer Verlag, Berlin, Heidelberg, September 2005.

[30] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *The International Journal Robotics Research*, 5(1):90–98, 1986.

[31] C. R. Kube and E. Bonabeau. Cooperative transport by ants and robots. *Robotics Autonomous Systems*, 30(1–2):85–101, 2000.

[32] J.A. Marshall, T. Fung, M.E. Broucke, G. DEleuterio, and B.A. Francis. Experiments in multirobot coordination. *Robotics and Autonomous Systems*, 54(3):265–275, March 2006.

[33] J. Minguez and L. Montano. Nearness diagram (nd) navigation: Collision avoidance in troublesome scenarios. *IEEE Transactions on Robotics and Automation*, 20(1):45–59, Feb. 2004.

[34] N. Miyata, J. Ota, T. Arai, and H. Asama. Cooperative transport by multiple mobile robots in unknown static environments associated with real-time task assignment. *IEEE Transactions on Robotics and Automation*, 18(5):769–780, October 2002.

[35] R. R. Murphy, Ch. L. Lisetti, R. Tardif, L. Irish, and A. Gage. Emotion-based control of cooperating heterogeneous mobile robots. *IEEE Transactions on Robotics and Automation*, 18(5):744–757, October 2002.

[36] A. Nakamura, J. Ota, and T. Arai. Human-supervised multiple mobile robot system. *IEEE Transactions on Robotics and Automation*, 18(5):728–743, October 2002.

[37] P. Ögren and N.E. Leonard. A convergent dynamic window approach to obstacle avoidance. *IEEE Transactions on Robotics*, 21(2):188–195, April 2005.

[38] S. X. Yang P. X. Liu, M. Q.-H. Meng. Data communications for internet robots. *Autonomous Robots*, 15(3):213–223, November 2003.

[39] L. E. Parker. Current state of the art in multi-robot teams. In *Distributed Autonomous Robotic Systems*, pages 2–12. New York: Springer-Verlag, 2000.

[40] L.E. Parker. Alliance: An architecture for fault tolerant multirobot cooperation. *IEEE Transactions on Robotics and Automation*, 14(2):220–240, April 1998.

[41] G. A. S. Pereira, M. F. M. Campos, and V. Kumar. Decentralized algorithms for multi-robot manipulation via caging. *International Journal Robotics Research*, 23(7–8), JulyAugust.

[42] S. I. Roumeliotis and G. A. Bekey. Distributed multirobot localization. *IEEE Transactions on Robotics and Automation*, 18(5):781–795, October 2002.

[43] S. I. Roumeliotis and I. M. Rekleitis. Propagation of uncertainty in cooperative multi-robot localization: Analysis and experimental results. *Autonomous Robots*, 17(1):41–54, July 2004.

[44] Daniela Rus, Bruce Donald, and Jim Jennings. Moving furniture with teams of autonomous robots. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robot Systems (IROS1995)*, pages 235–242, 1995.

[45] P. E. Rybski, S. A. Stoeter, M. Gini, D. F. Hougen, and N. P. Papanikolopoulos. Performance of a distributed robotic system using shared communications channels. *IEEE Transactions on Robotics and Automation*, 18(5):713–727, October 2002.

[46] W.-M. Shen, P. Will, A. Galstyan, and Ch.-M. Chuong. Hormone-inspired self-organization and distributed control of robotic swarms. *Autonomous Robots*, 17(1):93–105, July 2004.

[47] R. Siegwart and I.R. Nourbakhsh. *Introduction to Autonomous Mobile Robots*. MIT Press, Cambridge, Mass., London. England, 2004.

[48] R. Simmons. The curvature-velocity method for local obstacle avoidance. In *Proc. IEEE Int. Conf. Robotics and Automation*, pages 3375–3382, Minneapolis, MN, 1996.

[49] W. M. Spears, D. F. Spears, J. C. Hamann, and R. Heil. Distributed, physics-based control of swarms of vehicles. *Autonomous Robots*, 17(2–3):137–162, September 2004.

[50] T. G. Sugar and V. Kumar. Control of cooperating mobile manipulators. *IEEE Transactions on Robotics and Automation*, 18(1):94–103, February 2002.

[51] S. Thrun. A probabilistic on-line mapping algorithm for teams of mobile robots. *International Journal Robotics Research*, 20(5):335–363, May 2001.

[52] I. Ulrich and J. Borenstein. Vfh+: reliable obstacle avoidance for fast mobile robots. In *Proc. IEEE Int. Conf. Robotics and Automation*, pages 1572–1577, 1998.

[53] I. Ulrich and J. Borenstein. Vfh*: local obstacle avoidance with lookahead verification. In *Proc. IEEE Int. Conf. Robotics and Automation*, pages 2505–2511, San Francisco, CA, 2000.

[54] R. T. Vaughan, K. Støy, G. S. Sukhatme, and M. J. Matarić. Lost: Localization-space trails for robot teams. *IEEE Transactions on Robotics and Automation*, 18(5):796–812, October 2002.

[55] M. Veloso and P. Stone. A survey of multiagent and multirobot systems. In *Robot teams: From diversity to polymorphism, T. Balch and L. E. Parker, Eds.* Wellesley, MA: AK Peters, 2002.

[56] Z. Wang, E. Nakano, and T. Takahashi. Solving function distribution and behavior design problem for cooperative object handling by multiple mobile robots. *IEEE Transactions on Systems, Man and Cybernetics-Part A: Systems and Humans*, 33(5):537–549, September 2003.

[57] Z-D. Wang, Y. Takano, Y. Hirata, and K. Kosuge. A pushing leader based decentralized control method for cooperative object transportation. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robot Systems (IROS2004)*, 2004.

[58] A. Yamashita, T. Arai, J. Ota, and H. Asama. Motion planning of multiple mobile robots for cooperative manipulation and transportation. *IEEE Transactions on Robotics and Automation*, 19(2):223–237, April 2003.

[59] C. Zieliński. Formal approach to the design of robot programming frameworks: the behavioural control case. *Bulletin of the Polish Academy of Sciences – Technical Sciences*, 53(1):57–67, March 2005.

[60] R. Zlot and A. Stentz. Market-based multirobot coordination for complex tasks. *International Journal Robotics Research*, 25(1):73–102, January 2006.