**POLITECHNIKA WARSZAWSKA**
Wydział Elektroniki i Technik Informacyjnych
Instytut Automatyki i Informatyki Stosowanej
Zespół Sterowania i Programowania Robotów

Piotr Trojanek

# Środowisko programowania robotów mobilnych z wykorzystaniem pakietu Player/Stage

Praca magisterska wykonana pod kierunkiem
dr Wojciecha Szynkiewicza


Warszawa 2006

# Environment for mobile robots programming with the Player/Stage software

**Abstract**

In this thesis the author addresses the problem of building mobile robot control software framework. The thesis is divided into four parts.

First part describes robot control architectures and programming frameworks currently in use. The requirements of such a software in general are discussed.

In the second part of the thesis one of the presented software packages is described in detail. For newly built mobile robots the Player/Stage package was choosen as a complete control framework.

In the third part the details of robot internals are described. The integration process of sensor and effector devices with control software is discussed.

The fourth part describes two simulation packages with their capabilities. The role of simulation is discussed and diffrent robot models are presented.

The last part covers verification of the whole presented environment. As an example application the multi–robot box pushing task is described.

**Streszczenie**

Praca poświęcona jest tworzeniu środowiska do programowania robotów mobilnych.

Pierwsza część opisuje architektury sterowania robotami oraz struktury programowe znajdujące się obecnie w użyciu. Omówione zostały także wymagania, jakie powinno spełniać tego typu oprogramowanie.

W drugiej części pracy został opisany szczegółowo jeden z przedstawionych pakietów oprogramowania. Struktura Player/Stage została wybrana jako oprogramowanie sterujące nowozbudowanych robotów mobilnych.

Część trzecia opisuje szczegółowo budowę posiadanych robotów. Został w niej omówiony proces integracji sensorów i efektorów z oprogramowaniem sterującym.

Czwarta część opisuje dwa pakiety symulacji robotów. Przedstawione zostały ich możliwości oraz zastosowania, jak również zaprezentowane zostały modele różnych robotów.

Ostatnia część pracy poświęcona jest weryfikacji stworzeonego środowiska programowania. Jako przykładowa aplikacja zostało zrealizowane zadanie wielorobotowego transportu pudła.

# Spis treści

# Spis rysunków

# Spis tabel

# Rozdział 1

# Wprowadzenie

## 1.1 Cel i zakres pracy

Głównym celem tej pracy było przygotowanie środowiska do programowania robotów mobilnych na potrzeby prac dydaktycznych oraz naukowo–badawczych prowadzonych w Instytucie Automatyki i Informatyki Stosowanej (IAiIS) Politechniki Warszawskiej. Jako środowisko jest tutaj rozumiana całość oprogramowania robotów – prace obejmowały zarówno tworzenie programów sterowników mikrokontrolerów jak i narzędzi do realizacji zadań przez grupę robotów.

Niezbędnym elementem w prowadzeniu systematycznej pracy nad zagadnieniami robotyki mobilnej jest posiadanie systemu sterowania, którego funkcją jest zapewnienie dostępu do układów sensorów i efektorów robota. Należało dokonać przeglądu istniejących struktur oprogramowania robotów oraz sporządzić charakterystykę wymagań stawianych takiemu środowisku. Struktura taka powinna umożliwiać przejrzystą dekompozycję oprogramowania na moduły o wydzielonej funkcjonalności – takie jak implementacje algorytmów unikania kolizji, budowania map czy planowania ścieżki.

Na podstawie przeprowadzonej analizy dostępnego oprogramowania został wybrany pakiet *Player/Stage*, który w największym stopniu spełniał wyszczególnione w pracy wymagania. Jednocześnie stwarzał on realną szanse na możliwość szybkiego przystosowania do działania z posiadanymi robotami, a przez to przygotowania kompletnego i spójnego systemu do wykorzystania w dalszych pracach.

Zakres pracy obejmował także stworzenie sterowników do układów sensorów i efektorów robota – takich jak silniki układu jezdnego z enkoderami optycznymi, czujniki odległości oraz dalmierz laserowy zamontowany na obrotnicy w celu pozyskiwania map trójwymiarowych. Uruchomione moduły

sprzętowe robota należało połączyć z wybraną wcześniej strukturą oprogramowania oraz przygotować obraz systemu operacyjnego do pracy na komputerze robota z uwzględnieniem specyfiki i wymagań takiej konfiguracji.

Kompletne środowisko programowania powinno zapewniać możliwie zbliżone do rzeczywistości symulowanie działań robota, co znacząco wpływa na przyśpieszenie prac. Roboty IAiIS zostały zamodelowane w dwóch uruchomionych narzędziach symulacyjnych. Prostsze – dwuwymiarowe umożliwia badanie zachowań dużych grup robotów w środowisku płaskim, takim jak piętro budynku oraz bardziej zaawansowane, wymagające większych zasobów obliczeniowych do symulowania środowiska trójwymiarowego z uwzględnieniem oddziaływań między obiektami – takimi jak tarcie czy możliwość manipulacji przedmiotami.

Jako weryfikacja i podsumowanie przeprowadzonych prac zostało zrealizowane zadanie współpracy robotów na przykładzie transportu obiektu. Zadanie to sprawdzało zarówno niezawodność działania większości uruchomionych modułów sprzętowych jak i możliwość wykorzystania przygotowanego środowiska do sterowania zespołem współpracujących robotów. Zaproponowany algorytm realizacji zadania cechuje się dużą odpornością na szumy układu pomiarowego oraz brakiem wrażliwości na zmiany parametrów modelu kinematycznego robota, które spowodowane są znaczącym poślizgiem kół. Duży stopień wymaganej kooperacji w trakcie realizacji zadania wymaga od struktury oprogramowania niezawodności w trakcie działania, gdyż awaria bądź nawet przejściowy brak komunikacji z jednym z robotów decyduje o niepowodzeniu.

# Rozdział 2

# Przegląd istniejących struktur programowania robotów

## 2.1 Struktury sterowania robotami mobilnymi

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.

Sterowanie robotem mobilnym nie jest zadaniem łatwym, musi bowiem uwzględniać wszystkie te cechy charakterystyczne. Najtrudniejszym, ale zarazem najciekawszym elementem jest opisywanie za pomocą modeli zarówno otoczenie robota, interakcje z nim, a także zachowanie. Przedstawię poniżej krótki przegląd podejść do omawianego zagadnienia.

### 2.1.1 Sterowanie reaktywne

Jednym z pierwszych podejść do sterowania działaniem robota mobilnego było kontrolowanie zachowań jako *reakcja* na bodźce [1]. Takie rozwiązanie 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, prze-

szkód lub czynników zagrażających funkcjonowaniu robota (wysoka temperatura, niebezpieczne substancje chemiczne) sprawdza się doskonale jako metoda przetrwania w nieprzyjaznym środowisku.

Taka struktura oprogramowania ma jednak istotne wady w przypadku kiedy stosowana jest do sterowania złożonymi obiektami. Często zachodzi konieczność realizowania przez robota kilku, bardzo słabo powiązanych ze sobą (a czasem sprzecznych) zadań. Mając do dyspozycji moc obliczeniową obecnych komputerów korzystniej wybrać inne, bardziej złożone podejście to sterowania.

### 2.1.2 Struktury warstwowe

Odmienne podejście do organizacji struktury sterowania robotem mobilnym to podzielenie na warstwy realizowanego zadania [2]. W swojej pracy Brooks definiuje wymagania co do układu sterowania. Powinien on mianowicie uwzględniać następujące elementy charakterystyczne dla robota mobilnego: wiele celów działania, wiele czujników, wymaganie odporności oraz rozszerzalności.

Zadania robota mobilnego często mogą być realizowane współbieżnie – tak jak poruszanie się do wyznaczonego punktu oraz zbieranie danych pomiarowych. W sytuacji jednak, kiedy dodatkowo dochodzą wymagania, aby zamierzona trajektoria została osiągnięta przy możliwie najmniejszym wykorzystaniu mocy silników oraz z ominięciem przeszkód z odpowiednio bezpiecznym dystansem należy zdecydować, które cele będą realizowane przed innymi.

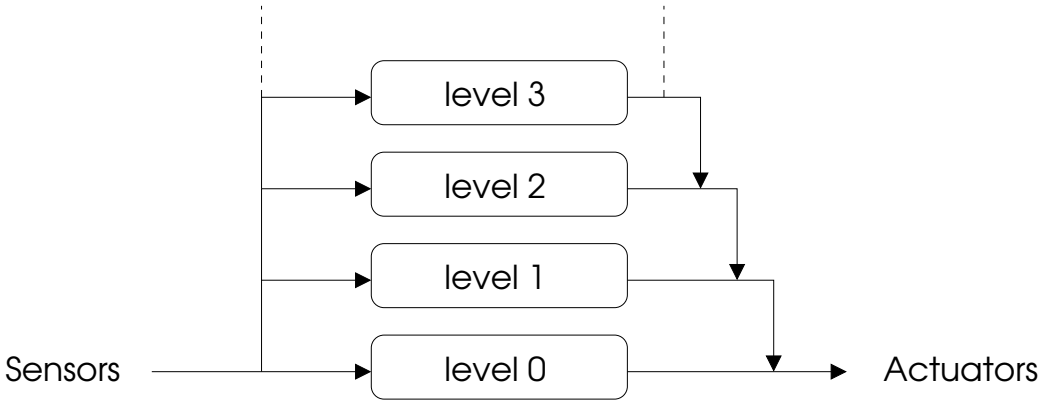
Robot mobilny, wyposażony w zestaw czujników musi podjąć decyzję, które z nich w danej chwili wykorzystywać i do jakich celów. Często zachodzi konieczność uzupełniania danych pomiarowych jednego rodzaju (np. ze skanera laserowego) informacjami z innych sensorów (kamera wizyjna, czujniki zbliżeniowe podczerwieni). Dodatkowo awaria jednego z modułów nie powinna powodować unieruchomienia całego systemu (*odporność*), zaś struktura musi być przystosowana do dodawania nowych elementów (*rozszerzalność*).

W zaproponowanym modelu zachowań wyróżnia się kolejne poziomy kompetencji robota:

0) unikanie zderzeń z przeszkodami,

1) bezkolizyjne, ale też bezcelowe poruszanie się w otoczeniu,

2) eksploracja otoczenia polegająca na osiąganiu punktów charakterystycznych,




Rysunek 2.1: Struktura warstwowa sterowania wg Brooks'a

3) budowanie mapy i planowanie ścieżek,

4) wykrywanie zmian w otoczeniu,

5) postrzeganie obiektów i wykonywanie związanych z nimi zadań,

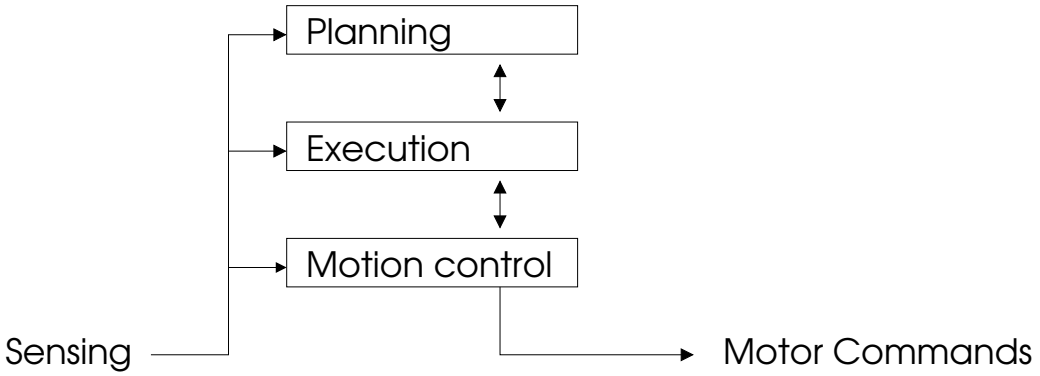
6) formułowanie i wykonywanie zadań, przez które robot oddziałuje na otoczenie,

7) wnioskowanie co do zachowań obiektów w otoczeniu i odpowiednie ich uwzględnianie w planowaniu działań.

Realizacja poszczególnych poziomów zawiera (ang. *subsume*) wszystkie niższe (rysunek 2.1). Zakłada się istnienie mechanizmu głosowania decydującego, która z warstw aktualnie uzyska prawo sterowania robotem, działanie zaś pozostałych będzie tłumione.

Problemem występującym w takim podejściu jest konieczność implementacji tej samej funkcjonalności przez wiele warstw, ponieważ z założenia kolejne poziomy mają wzbogacać możliwości robota, jednak nie bazują na zachowaniach już zaprogramowanych. Początkowo każda warstwa miała być realizowana przez osobny procesor (także dla zwiększenia niezawodności działania), a rozszerzanie polegać miało na dokładaniu kolejnych modułów sprzętowych. Obecnie wygodniej i prościej byłoby realizować taką strukturę jako osobne procesy, działające na jednej maszynie – komputerze pokładowym robota mobilnego. Nie byłaby jednak wykorzystywana możliwość współpracy pomiędzy warstwami.

Obecnie najczęściej stosowana jest modyfikacja oryginalnej struktury zaproponowanej przez Brooks'a, polegająca na tym, że kolejne warstwy są nadbudowywane w oparciu o funkcjonalność oferowaną przez niższe poziomy.




Rysunek 2.2: Budowanie funkcjonalności w warstwowej strukturze sterowania

W praktyce realizowane jest to poprzez wydzielanie obszarów funkcjonalności (np. budowanie mapy, obsługa czujników, sterowanie silnikami) i realizację ich w postaci modułów oprogramowania. Moduł taki za pośrednictwem dobrze określonego interfejsu komunikuje się z niższymi warstwami, sam jednocześnie udostępniając określoną funkcjonalność dla poziomów struktury sterowania znajdujących się powyżej.

W praktyce takie podejście polega na przydzieleniu poszczególnych zadań do różnych procesów oraz zapewnieniu komunikacji pomiędzy nimi z wykorzystaniem mechanizmów udostępnianych przez system operacyjny (*IPC – Inter-Process Communication*). Część funkcjonalności może także być implementowana w postaci bibliotek dynamicznie dołączanych do programów.

Obecnie tworzone i stosowane struktury wydają się odbiegać od zaproponowanych przez Brooks'a. Obiektowe podejście do programowania preferuje tworzenie oprogramowania składającego się z komponentów, które można wykorzystywać w wielu projektach. Twórcy robota *Frontier–I* kładą nacisk na trzy reguły, którymi należy kierować się przy tworzeniu oprogramowania dla robota mobilnego: *otwartość*, *abstrakcja* oraz *modułowość* [14].

Łatwość w dodawaniu nowych elementów zarówno oprogramowania (np. nowe algorytmy planowania ścieżki) jak i sprzętu (np. dodatkowe czujniki) może być z powodzeniem zrealizowana z wykorzystaniem metod programowania współbieżnego (*otwartość*), gdzie za realizację danego fragmentu sterowania odpowiada wydzielony proces. Dzięki tworzeniu interfejsów dostępu do poszczególnych warstw (*abstrakcja*) struktury sterowania (zwłaszcza sprzętu) możliwe staje się oddzielenie korzystania z określonej funkcjonalności od konieczności znajomości szczegółów implementacyjnych innych poziomów. Obudowanie dostępu do urządzeń zewnętrznych polega na dostarczeniu API (*Application Program Interface*), za pomocą którego inne moduły uzy-

skają do nich dostęp. Wykorzystywanie obiektowych języków programowania ogólnego przeznaczenia (takich jak *Java* czy *C++*) wspiera zarówno *modułowość* całej struktury, jak również ułatwia realizację wcześniej postawionych wymagań.

### 2.1.3 Struktury komponentowe

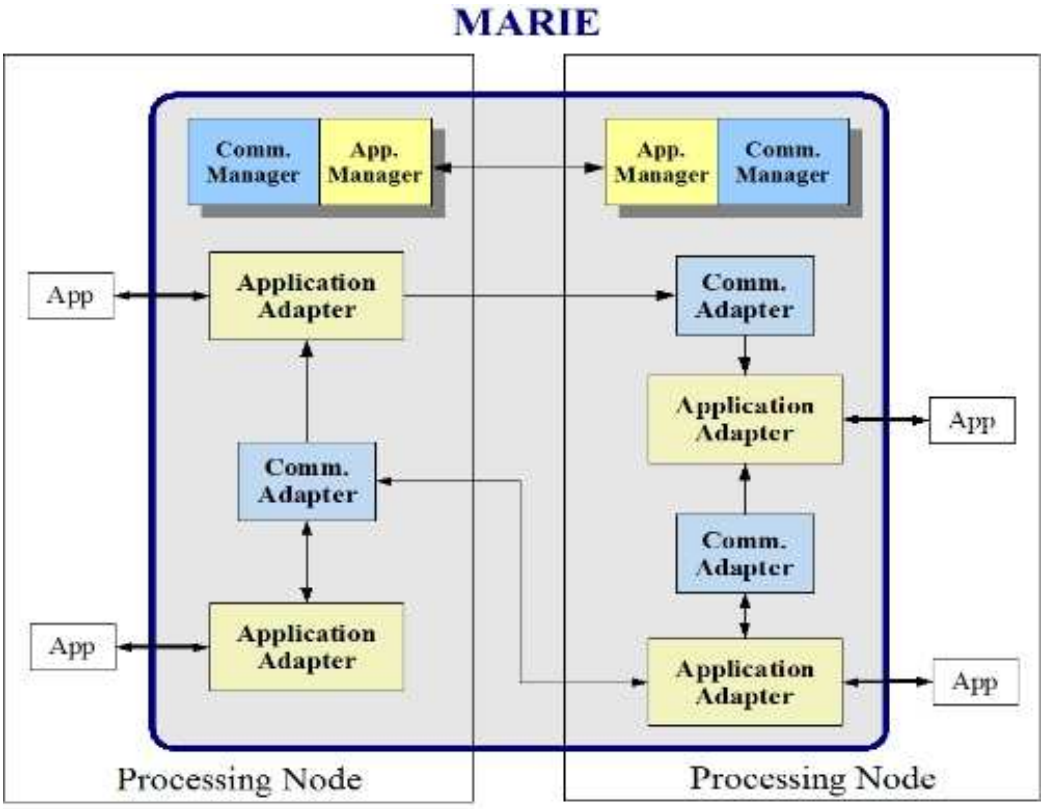
Od kilku lat coraz większą popularnością cieszą się struktury sterowania robotami budowane w oparciu o istniejące komponenty. Wysiłek programistyczny skupia się wtedy na scaleniu wielu dostępnych już elementów, takich jak biblioteki do obliczeń numerycznych, wizualizacji, czy komunikacji międzyprocesowej. W rozwoju takich struktur dużą rolę odgrywają społeczności *Open-Source*, a także możliwość współpracy wielu zespołów za pośrednictwem Internetu.

Przykładem takiej struktury jest projekt *Orocos* [28]. Jedną z cech oprogramowania tworzonego w ramach tego projektu jest uniezależnienie od systemu operacyjnego na jakim wykonywana będzie aplikacja sterująca robotem. Obecnie wspierane są najbardziej popularne rozszerzenia czasu rzeczywistego dla systemu Linux (*RTAI* [18], *RTLinux* [19]) oraz dedykowany dla zastosowań wbudowanych system czasu rzeczywistego *eCos* [8]. Podstawowym zastosowaniem jest sterowanie manipulatorami przemysłowymi, jednak podejście prezentowane przez twórców zgodne jest z wcześniej przedstawionymi założeniami co do projektowania struktur dla robotów mobilnych. Obiektowy język programowania, integracja z bibliotekami obsługującymi wiele dostępnych kart rozszerzeń wejścia–wyjścia to przykłady realizacji *rozszerzalnej* ramy programowej.

Podobne podejście prezentują twórcy projektu *MARIE* [27] – środowiska rozwojowego nastawionego na wykorzystywanie istniejącego już oprogramowania oraz dostępnych metod łączenia komponentów. Całość struktury składa się tu z takich elementów, jak narzędzia do programowania zachowań robota, środowiska symulacyjne (zarówno 2D jak i 3D, wykorzystujące elementy fizyki ciała stałego), biblioteki wspomagające nawigację, planowanie ścieżek oraz budowanie map. Do łączenia tych wszystkich elementów wykorzystywana jest biblioteka komunikacji międzyprocesowej, umożliwiająca synchronizację pracy procesów na wielu maszynach, działających pod kontrolą różnych systemów operacyjnych (rysunek 2.3).

### 2.1.4 Programowe struktury ramowe

W ramach badań naukowych prowadzonych w Instytucie Automatyki i Informatyki Stosowanej Politechniki Warszawskiej została stworzona programowa




Rysunek 2.3: Łączenie istniejących aplikacji w strukturę programową [27]

struktura ramowa *MRROC++* [25]. Sterowanie systemem wielorobotowym musi uwzględniać rozmaitość zarówno sprzętu, jak i realizowanych zadań. Narzędzie, które będzie do tego wykorzystywane musi zatem cechować się dużą elastycznością. Ogólna struktura systemu przedstawiona została poniżej (rys. 2.4).

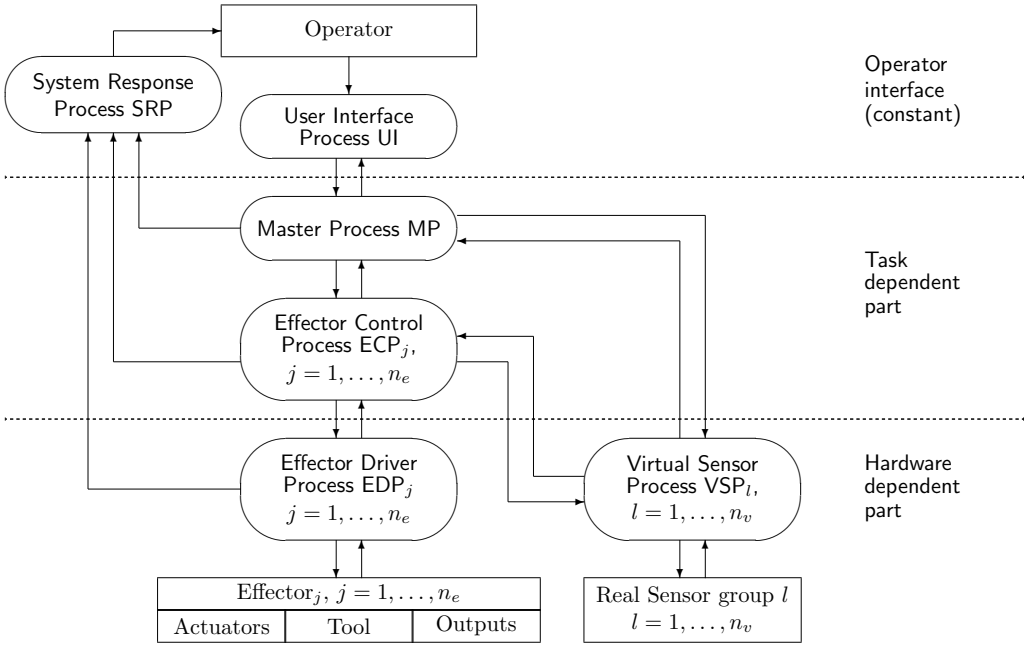
W strukturze *MRROC++* system robota został podzielony na trzy podsystemy:

– **efektory** *e*, urządzenia wywierające wpływ na otoczenie robota,

– **receptory** *r*, rzeczywiste czujniki gromadzące informacje o systemie oraz jego otoczeniu,

– **podsystem sterowania** *c*, wykonujący zadanie poprzez wykorzystywanie danych z receptorów oraz odpowiednie poruszanie efektorami.

Każda część jest realizowana przez osobny proces, działający w systemie operacyjnym czasu rzeczywistego *QNX*. Komunikacja pomiędzy warstwami




Rysunek 2.4: Schemat struktury programowej MRROC++

odbywa się z wykorzystaniem mechanizmu spotkań (*Send/Receive/Reply*). Warstwy wyższe, w których odbywa się planowanie uruchamiane są z odpowiednio niższym priorytetem niż warstwy najniższe, odpowiedzialne za komunikację ze sprzętem.

## 2.2 Porównanie struktur sterowania

Po zapoznaniu się z różnymi podejściami do zagadnienia sterowania robotami przedstawię kryteria, wedle których dokonam ich oceny.

Przede wszystkim rozróżnienia można dokonać pomiędzy systemami działającymi na zasadzie reagowania na bodźce, a tymi, które kładą nacisk na zbieranie i analizowanie danych o środowisku zewnętrznym. Pierwsze z nich są z reguły o wiele prostsze w implementacji, nie występuje tu element współbieżności ani komunikacji z innymi robotami. Przy obecnie dostępnych mocach obliczeniowych, jakimi może dysponować układ sterowania robotem mobilnym aspekt prostoty i niskiego zapotrzebowania na zasoby traci na znaczeniu.

O wiele bardziej atrakcyjne wydają się zatem struktury realizujące warstwowe sterowanie robotem. Programowanie obiektowe promuje wykorzysty-

wanie raz napisanego fragmentu oprogramowania przez wiele modułów.

Na podstawie zaprezentowanych wcześniej przykładowych realizacji stworzyłem zestaw wymagań, jakie według mnie powinna obecnie spełniać struktura sterowania robotem:

- jasne rozdzielenie warstw realizujących poszczególne elementy funkcjonowania w oparciu o API udostępniane przez niższe poziomy struktury,
- wykorzystywanie metod programowania obiektowego w celu stworzenia warstw abstrakcji pomiędzy elementami struktury,
- wykorzystywanie programowania współbieżnego w oparciu o mechanizmy udostępniane przez system operacyjny,
- brak przywiązania do wyłącznie jednej platformy sprzętowej i systemu operacyjnego – realizowane przez użycie mechanizmów komunikacji i synchronizacji pomiędzy różnymi platformami,
- stworzenie środowiska symulacyjnego dla tworzenia i szybszego testowania nowych algorytmów sterowania,
- modularność pozwalająca na łatwą i szybką integrację z nowymi elementami wyposażenia robota oraz nowymi algorytmami sterowania,
- wykorzystanie dostępnego oprogramowania *Open–Source*, jak np. biblioteki numeryczne.

# Rozdział 3

# Oprogramowanie robotów mobilnych

## 3.1 Roboty mobilne ELEKTRON

W posiadaniu Instytutu znajdują się trzy roboty mobilne o identycznej konstrukcji bazowej – napędzane są dwoma niezależnymi silnikami prądu stałego, z których napęd jest przenoszony paskami na zestaw trzech kół po każdej stronie robota. Na środkowych kołach znajdują się enkodery, z których zbierane są odczyty odometrii, a jednocześnie służą one do zamknięcia pętli sterowania silnikami przez regulatory scalone LM629.

Wszystkie trzy egzemplarze robotów są wyposażone w czujniki odległości umieszczone na obwodzie robota, różnią się natomiast sensorami zamontowanymi na górnej płycie konstrukcji – są to skaner laserowy SICK LMS200, zestaw dwóch kamer do realizacji stereowizji oraz kamera dookólna. W chwili obecnej nie jest jeszcze zamontowana kamera dookólna. Dodatkowo zamontowana jest obrotnica umożliwiającej manipulację skanerem laserowym tak, aby uzyskiwać odczyt odległości z trzech wymiarów (rys. 3.1).

Wewnątrz każdego robota znajduje się komputer typu *embedded PC*, zawierający karty rozszerzeń w standardzie PC/104 (odpowiednik magistrali ISA) oraz PC/104+ (odpowiednik magistrali PCI).

Są to odpowiednio:

- karta sterowania układem zasilania, silnikami oraz akwizycji danych z czujników odległości,
- karta komunikacji bezprzewodowej WLAN, pracująca w standardzie 802.11g (PC/104+),


Rysunek 3.1: Robot mobilny z IAiIS

- karty akwizycji obrazu framegrabber (tylko dla egzemplarzy wyposażonych w kamery).

Komputery PC (model Advantech PCM–9579) znajdujące się w każdym robocie wyposażone są m.in. w 4 porty RS232 (jeden z nich może pracować jako RS232/422/485), 4 porty USB 1.1, kartę sieciową, kartę dźwiękową, procesor Intel Pentium III 650/933 MHz, 256MB pamięci RAM oraz kartę CompactFlash o pojemności 512MB.

## 3.2 Oprogramowanie robota

Jedną z podstawowych decyzji, jakie należało podjąć odnośnie oprogramowania robota był wybór systemu operacyjnego. W tym celu należało zebrać wszystkie dopuszczalne możliwości, a następnie w wyniku dyskusji w gronie osób zaangażowanych w prace nad robotami znaleźć najlepsze rozwiązanie.

Rozpatrywane były tutaj dwie propozycje – system operacyjny czasu rzeczywistego QNX oraz system ogólnego zastosowania Linux. Podsumowanie wad i zalet obydwu systemów zostało zamieszczone poniżej (tab. 3.1).

Wybór padł na system LINUX ponieważ jego zalety uznane zostały za na tyle silne i istotne, że będą decydowały o szybkości i efektach pomimo nakładu pracy potrzebnego na poradzenie sobie z problemami wynikającymi z jego wad.

| **QNX** | **Linux** |
|---|---|
| + „hard real–time” | - nie „real–time” |
| + obsługa przerwań w trybie „user--space” | - obsługa przerwań tylko w modułach jądra |
| - słabe wsparcie techniczne | + znakomita baza użytkowników |
| - brak narzędzi (kompilatory, biblioteki) | + bogactwo narzędzi |
| - niepewna przyszłość | + obecny rozwój pozawala nie bać się o przyszłość |
| - brak sterowników | + bogata baza gotowych sterowników urządzeń |

Tabela 3.1: Zalety (+) i wady (-) rozważanych systemów operacyjnych

Po wyborze systemu operacyjnego należało podjąć decyzję co do struktury oprogramowania, która będzie dalej wykorzystywana. Do wyboru były tutaj zasadniczo dwie możliwości:

1. **MRROC++** – jest oprogramowaniem rozwijanym w ramach działalności Zespołu Sterowania i Programowania Robotów IAiIS, silnie związanym z systemami operacyjnymi QNX (początkowo w wersji 4, obecnie w wersji 6); jest to ramowa struktura programowa do sterowania robotami, jednak bardzo silnie ukierunkowana na manipulatory a nie roboty mobilne; cechuje się małym stopniem integracji z istniejącym oprogramowaniem (bibliotekami),

2. **Player/Stage** – oprogramowanie będące serwerem urządzeń dedykowanym do sterowania robotami mobilnymi, tworzone na systemie Linux, ale działające także na innych platformach; jest szeroko stosowana na świecie; szeroko wykorzystuje istniejące biblioteki m.in. do analizy i syntezy mowy, do obliczeń numerycznych oraz rozpoznawania obrazu.

Wybór padł na oprogramowanie *Player/Stage* (*P/S*) przede wszystkim ze względu na przewidywany szybki i dynamiczny rozwój systemu sterowania robotami, którymi dysponujemy. Realna szansa szybkiego uzyskania pełnego środowiska programowania wraz z dołączonymi do *P/S* narzędziami, bibliotekami, wielorobotowym symulatorem *Stage* zadecydowała o właśnie takim wyborze. Integracja robotów z *P/S* wymagała jedynie utworzenia sterownika do komunikacji z układami robota znajdującymi się na karcie PC/104 ([15]).

## 3.3 Pakiet oprogramowania Player/Stage

Pakiet oprogramowania *Player/Stage* składa się z dwóch głównych elementów – serwera urządzeń *Player* oraz modułu symulatora *Stage*.

Proces *Player* zaimplementowany został jako serwer bazujący na protokole TCP, który w jednym połączeniu przesyła do klientów pakiety z aktualnym stanem obsługiwanych urządzeń zwrotnie oczekując na pakiety sterujące i konfiguracyjne (jak np. wartości zadane prędkości i polecenia zmiany trybu pracy czujników). Rozróżnia się tutaj dwa pojęcia:

**driver** – obsługuje jedno lub więcej interfejsów urządzeń, implementowany jest najczęściej w formie biblioteki dzielonej (plik z rozszerzeniem **.so*); urządzeniami mogą być fizyczne elementy robota, jak np. układ jezdny, zestawy czujników, ale również jako „urządzenie" widziane są algorytmy, dla przykładu algorytm lokalizacji, który wyznacza pozycję robota na mapie,

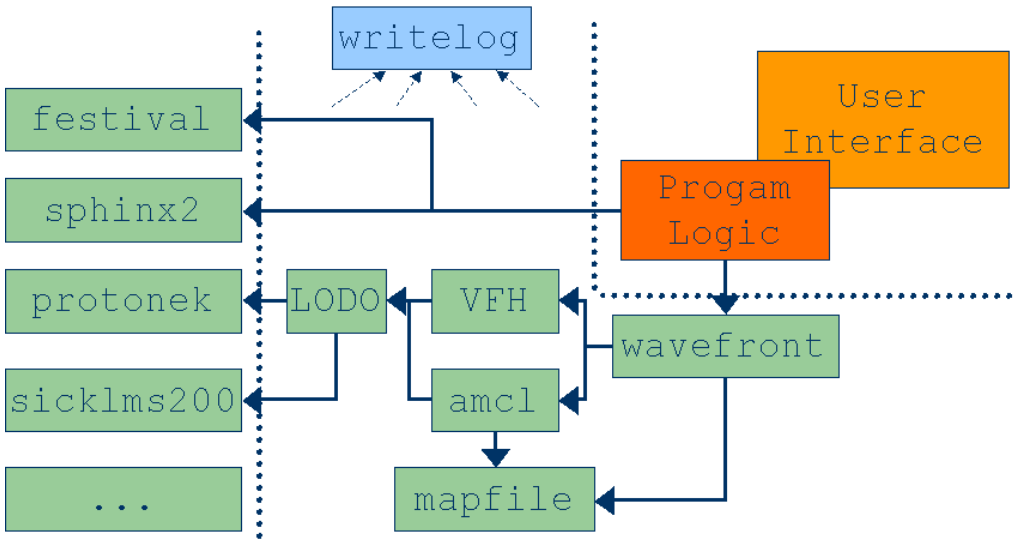
**interface** – definiuje struktury danych oraz metody (jako kody komend sterujących i konfiguracyjnych) związane z pojedynczym urządzeniem.

Każdy *driver* udostępnia (ang. *provides*) określony *interface* oraz może wymagać (ang. *requires*) interfejsów innych urządzeń. Dla przykładu implementacja algorytmu stabilizacji odometrii na podstawie odczytów ze skanera laserowego wymaga do działania właśnie sterowników udostępniających interfejsy lasera i odometrii ([26]). Na podobnej zasadzie algorytm lokalizacji robota wymaga modułu udostępniającego mapę otoczenia, który może opierać się na statycznej jej postaci wczytanej z pliku, jak też może dynamicznie zmieniać jej stan na podstawie odczytów z czujników. Na poniższych rysunkach został przedstawiony schemat zależności pomiędzy sterownikami (rys. 3.2) oraz interfejsami (rys. 3.3) dostępnymi w dystrybucji *Player/Stage*.

Przykładowa konfiguracja do sterowania robotem mobilnym wyposażonym w skaner laserowy modułem algorytmu omijania przeszkód została przedstawiona poniżej. Wykorzystuje ona cztery sterowniki, kolejno są to:

1. **protonek** – sterownik układu jezdnego robota, zapewniający realizację sterowań (przyjmuje wartości zadane składowych prędkości – liniowej i obrotowej) oraz zwrotnie udostępniający nieprzetworzone odczyty z enkoderów umieszczonych na kołach robota,

2. **sicklms200** – sterownik skanera laserowego, udostępnia tablicę odczytów pomiaru odległości do przeszkód znajdujących się przed czołem robota,




Rysunek 3.2: Struktura współdziałania driver'ów Player/Stage

3. **lodo_driver** – implementacja stabilizacji pomiarów odometrii w oparciu o odczyty z dalmierza laserowego,

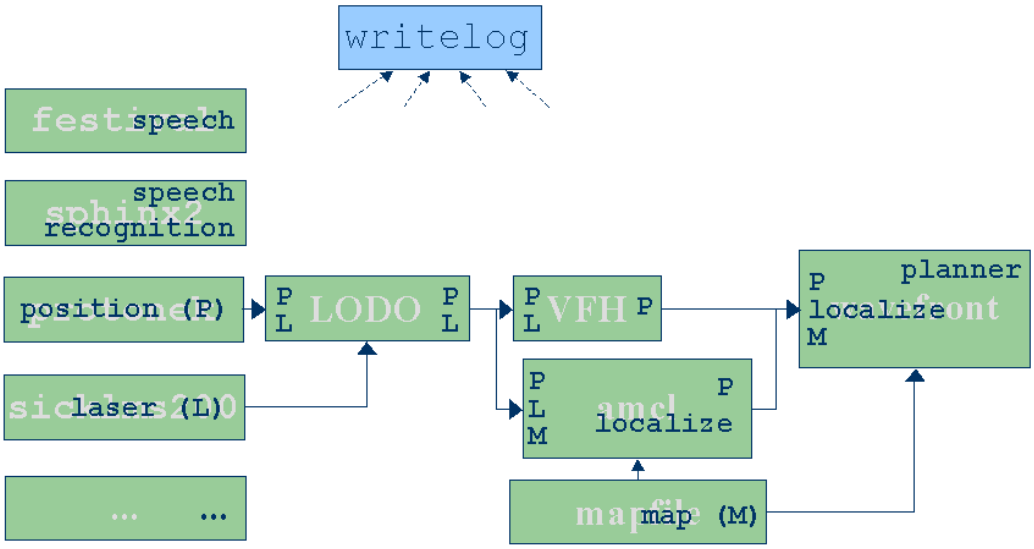
4. **vfh** – implementacja algorytmu *Virtual Histogram Field+* do lokalnej nawigacji z omijaniem przeszkód z wykorzystaniem skanera laserowego.

Wykorzystywane sterowniki oraz zależności pomiędzy nimi definiowane są wraz z parametrami w pliku konfiguracyjnym serwera *Player* (tab. 3.2).

W dystrybucji *Player/Stage* znajdują się gotowe sterowniki zarówno do rzeczywistych urządzeń – ze względu na brak specyficznego sprzętu dla którego były tworzone są dla nas użyteczne tylko jako przykłady, na których można opierać pisanie nowych sterowników. Są w niej zawarte także w postaci sterowników implementacje algorytmów, które dzięki wykorzystaniu dobrze zdefiniowanych interfejsów, są gotowe do wykorzystania niezależnie od kontrolowanego typu robota.

Poniżej została zawarta lista tych implementacji algorytmów, które są gotowe do wykorzystania i przyśpieszenia realizacji prostych zadań przez posiadane w Instytucie roboty mobilne:

**acoustics** – generowanie dźwięków o określonej częstotliwości i czasie trwania; potencjalne zastosowanie to sygnalizacja stanu robota, wykonywanych przez niego zadań i algorytmów;




Rysunek 3.3: Struktura współdziałania interfejsów Player/Stage

**mixer** – kontrola głośności generowanych dźwięków;

**cmvision** – wykrywanie spójnych obszarów barwnych w obrazie z kamery przy wykorzystaniu biblioteki *CMVision* [3];

**simpleshape** – wykrywanie wzorców geometrycznych w obrazie video z kamery z użyciem o biblioteki *OpenCV* [17];

**laserbar** – wykrywa w polu skanowania dalmierza laserowego płaskie i okrągłe obiekty;

**laserbarcode** – identyfikuje specjalnie skonstruowane wzorce z materiału przepuszczającego i odbijającego promień lasera;

**lifomcom** – system komunikacji klientów serwera *Player* wykorzystujący kolejkę komunikatów LIFO;

**wavefront** – algorytm planera globalnego;

**vfh** – algorytm planera lokalnego bazującego na algorytmie VFH+ [23];

**service_adv_mdns** – rozgłaszanie informacji o zasobach każdego z serwerów *Player* w celu realizacji przypisania zadania do robota w zależności od jego wyposażenia;

| | |
|---|---|
| **driver**<br>(<br>**name** "protonek"<br>plugin "protonekdriver.so"<br>**provides** ["position:0"]<br>max_enc_ticks 62000<br>max_speed 0.2178692665<br>wheel_diam 0.10<br>ticks_per_rev 4000<br>max_accel 80<br>robot_length 500<br>robot_width 360<br>robot_axle_length 330<br>) | **driver**<br>(<br>**name** "vfh"<br>**requires** ["position:3" "laser:0"]<br>**provides** ["position:2"]<br>safety_dist 0.40<br>distance_epsilon 0.3<br>angle_epsilon 5<br>max_speed 0.2178692665<br>) |
| **driver**<br>(<br>**name** "sicklms200"<br>**provides** ["laser:0"]<br>port "/dev/ttyS2"<br>rate 38400<br>resolution 100<br>range_res 1<br>) | **driver**<br>(<br>**name** "lodo_driver"<br>plugin "lodo_driver.so"<br>**provides** ["position:3" "laser:3"]<br>**requires** ["position:0" "laser:0"]<br>max_range 8.192<br>) |

Tabela 3.2: Plik konfiguracyjny serwera Player

**festival** – sterownik syntezatora mowy bazujący na oprogramowanie o tej samej nazwie;

**sphinx2** – sterownik analizatora mowy działający w czasie rzeczywistym wykorzystujący biblioteki projektu *Sphinx*;

**iwspy** – odczyt pomiaru natężenia sygnału bezprzewodowych kart Ethernet.

# Rozdział 4

# Oprogramowanie modułów sprzętowych robota

## 4.1 Opis działania układu sterowania

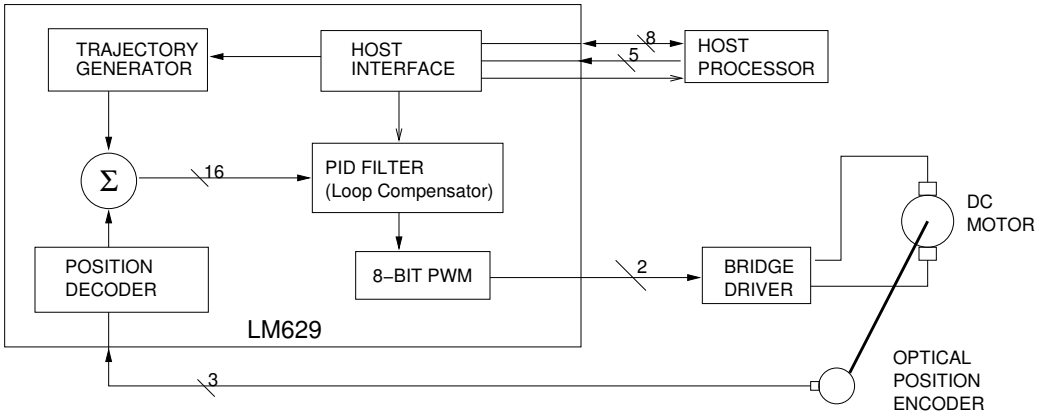
Bazę jezdną robota stanowi 6 kół, połączonych po trzy za pomocą pasków. Każda trójka napędzana jest niezależnie silnikiem prądu stałego z przekładnią. Z dwoma środkowymi kołami sprzęgnięte są enkodery obrotowe o rozdzielczości 4000 działek na obrót.

Bezpośrednie sterowanie silnikami realizowane jest za pomocą scalonych regulatorów LM629 [16], do których podłączone są sygnały z enkoderów, zaś na wyjściu generowany jest sygnał PWM podawany przez wzmacniacze mocy na silniki. Parametry ruchu generowane są przez oprogramowanie wykonujące się na komputerze PC, realizujące określone wcześniej zadanie robota mobilnego.

### 4.1.1 Opis działania regulatora LM629

Układy LM629 odpowiedzialne są za generowanie sygnałów PWM sterujących silnikami. Poniżej znajduje się schemat połączenia układów (rysunek 4.1):

- z enkoderami (3 linie),
- wzmacniaczami mocy (linia wypełnienia – PWM oraz kierunku obrotów),
- układem nadrzędnym (linie danych oraz sygnały przerwań).




Rysunek 4.1: Schemat podłączenia regulatora LM629 [12]

Rolę układu nadrzędnego dla LM629 może pełnić zarówno komputer PC jak i mikrokontroler nadzorujący pracę podstawowych układów robota. W chwili obecnej mikrokontroler pośredniczy w komunikacji komputera z regulatorami zapewniając kontrolę zakresów parametrów regulacji oraz obsługując sytuacje awaryjne.

Układ LM629 implementuje regulator PID realizujący sterowanie zarówno pozycyjne jak i prędkościowe z zadanymi parametrami ruchu, to jest przyśpieszeniem, prędkością i przemieszczeniem. Generowany jest trapezowy przebieg prędkości. Parametry regulatora PID – wzmocnienia poszczególnych członów oraz stałe czasowe mogą być zadawane.

Podstawowy okres próbkowania dla regulatorów LM629 wynika z zastosowanego oscylatora kwarcowego i dla 6MHz wynosi $t_s = 2048/f_{osc} = 341\mu s$. Możliwe jest dobranie stałej czasowej osobno dla członu różniczkującego.

W chwili obecnej dostępne są następujące funkcje regulatora z poziomu komputera PC przez mikrokontroler pośredniczący:

1. wyzerowanie rejestru położenia i przywrócenie początkowych nastaw regulatora,

2. zatrzymanie silników przez podanie zerowego sterowania (wyhamowanie przekładniami),

3. zatrzymanie silników przez łagodne wyhamowanie (z zadanym przyśpieszeniem),

4. załadowanie parametrów regulacji PID – zakresu całkowania, wzmocnień *kp*, *ki*, *kd*, okresu próbkowania dla członu różniczkującego,

5. odczyt aktualnego położenia – ze względu na ograniczony rozmiar rejestru komunikacyjnego pomiędzy mikrokontrolerem a PC (16-bajtów) przesyłana jest różnica pomiędzy aktualnym, a ostatnio odczytanym położeniem,

6. załadowanie wartości przyśpieszenia dla generowanego profilu prędkości,

7. załadowanie wartości prędkości dla sterowania pozycyjnego,

8. załadowanie przesunięcia dla sterowania pozycyjnego.

Wprowadzone zostały ograniczenia na maksymalne wartości prędkości oraz przyśpieszenia tak, że wypełnienie generowanego sygnału PWM nie przekracza 90%.

Należy zaznaczyć, że osiągane prędkości koła są względnie małe – poniżej jedności liczone jako ilość impulsów enkodera na okres próbkowania – $341\mu s$. O ile zadawanie tak małych prędkości jest możliwe, to niemożliwy jest ich odczyt przez układ LM629, który nie raportuje części ułamkowej. Z drugiej strony enkoder generuje impulsy z na tyle dużą częstotliwością, że niepraktyczne jest zliczanie czasu pomiędzy nimi przez mikrokontroler i na tej podstawie wyznaczanie prędkości. Jedyne praktyczne rozwiązanie, to regularny odczyt przez komputer zmiany położenia raportowanej przez układ regulatora, czyli obliczanie prędkości poprzez numeryczne różniczkowanie.

### 4.1.2 Opis karty PC/104 z układami sterującymi

Regulatory LM629 znajdują się na karcie PC/104, zaś dostęp do nich przełączany jest pomiędzy przestrzeń portów ISA (z wykorzystaniem dekodera adresów) a magistralę łączącą je z mikrokontrolerem nadzorującym pracę układów robota.

Układ AT89C51AC2 firmy ATMEL znajdujący się na karcie jest ośmiobitowym procesorem z rodziny ’51 wzbogaconym o szereg układów peryferyjnych, m.in.:

- 32 KB pamięci programu typu FLASH,
- 1 KB dodatkowej pamięci danych (RAM),
- 2 KB pamięci EEPROM,
- 8–kanałowy, 10–bitowy przetwornik analogowo cyfrowy,
- 5 układów czasowo–licznikowych,

- układ transmisji szeregowej (UART),
- 21-bitowy watch-dog timer.

Mikrokontroler dodatkowo ma dostęp do wyświetlacza LCD i przycisków znajdujących się na panelu robota, zezwala na włączenie zasilania dodatkowych elementów wyposażenia (kamera, dalmierz laserowy), obsługuje przywrócenie kontroli po aktywacji stopu awaryjnego. Realizuje ponadto funkcję sprawdzania stanu naładowania akumulatorów (przez wbudowane układy ADC), zapewnia komunikację z mikrokontrolerami pomiarowymi czujników zbliżeniowych podczerwieni (przez transmisję RS485) oraz generowanie sygnałów PWM sterujących serwomechanizmami modelarskimi obracającymi kamerę.

Znajdujący się na karcie dekoder adresów pozwala na komunikację pomiędzy PC a mikrokontrolerem przez:

- dwa 16-bitowe rejestry poleceń, do zapisu przez komputer główny i do odczytu przez mikrokontroler,
- dwa (16-bitowy i 8-bitowy) rejestry stanu, do odczytu przez komputer główny i do zapisu przez mikrokontroler.

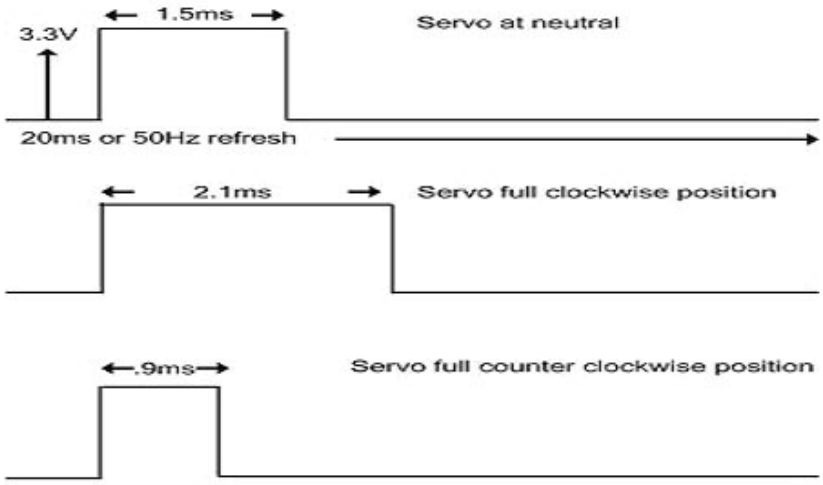
Do generowania sygnału PWM sterującego serwomechanizmem modelarskim wykorzystywane są dwa układy licznikowe mikrokontrolera: *Timer 2* do generowania okresowego przerwania (50Hz) oraz *Timer 0* do odmierzania czasu trwania impulsu. Układ *Timer 1* wykorzystany jest jako generator dla układu transmisji szeregowej. Ze względu na specyfikę sygnału sterującego serwomechanizmami (rysunek 4.2) niemożliwe okazało się wykorzystanie układów czasowo–licznikowych mikrokontrolera.

W celu zapewnienia płynnego sterowania serwomechanizmami możliwe jest podanie zarówno docelowej pozycji, jak i prędkości z jaką będzie ona osiągana. Realizowane jest to przez zwiększanie czasu trwania kolejnych impulsów aż do uzyskania końcowej wartości.

Pomiar napięcia akumulatorów realizowany jest przez uśrednienie kolejnych 130 odczytów. Taki dobór liczby powtórzeń sprawia, że rezulatat otrzymywany jako suma odczytów 8-bitowych składa się na wartość rzędu $130 * 184 \approx 24000$, czyli $10000 \times napięcie\_na\_akumulatorze$.

### 4.1.3 Komunikacja pomiędzy komputerem PC a mikrokontrolerem na karcie PC/104

Przesyłanie danych pomiędzy komputerem PC a mikrokontrolerem na karcie PC/104 odbywa się przez wcześniej opisane rejestry komunikacyjne dostępne




Rysunek 4.2: Charakterystyka sygnału PWM do sterowania serwomechanizmami modelarskimi [11]

dla PC przez w przestrzeni adresowej portów wejścia–wyjścia, a dla mikrokontrolera zmapowane w przestrzeń adresową zewnętrzenej pamięci RAM. Zapis do rejestrów powoduje zgłoszenie przerwania u strony odbierającej.

Wymiana danych inicjowana jest zawsze przez komputer PC – wysyła on parametry funkcji realizowanych następnie przez mikrokontroler bądź odpytuje o informacje. Wymiana realizowana jest w ten sposób, że pojedyncza funkcja realizowana jest w całości przez pojedynczą wymianę informacji. W ten sposób unika się przechowywania informacji o stanie komunikacji i uodparnia wymianę danych na problemy z ewentualną synchronizacją transmisji.

Mikrokontroler w procedurze obsługi przerwania (jako reakcja na zapis danych do rejestrów komunikacyjnych przez PC) wyłącznie kopiuje zawartość rejestrów do zmiennych globalnych. Właściwa realizacja żądania bądź zapytania realizowana jest w pętli programu głównego, dzięki czemu unika się długotrwałego blokowania przerwań np. na czas transferu danych do regulatorów. Po tym jak żądanie PC zostanie pomyślnie zrealizowane jego wynik jest wpisywany do rejestrów komunikacyjnych i zgłaszane jest przerwanie do komputera. W przeciwnym wypadku w rejestrze statusu umieszczony zostaje kod zawierający informację o błędzie.

### 4.1.4 Protokół komunikacji z kartą PC/104 robota mobilnego

Kompletna specyfikacja przedstawiona poniżej zawiera zestaw funkcji obsługiwanych przez mikrokontroler wraz z kodami ich wywołania. Rejestry komunikacyjne są umieszczone w przestrzeni adresów we-wy PC pod adresami (hex) 320 – 32F (tab. 4.1).

| **Adres (hex)** | **odczyt** | **zapis** | |
|---|---|---|---|
| 320 | DMKPC | DPCMK1, zgłoszenie przerwania do MK | Dostęp 16-bit. |
| 322 | SMKPC | DPCMK2 | |
| 324 | Wytworzenie zbocza sygnału zgłoszenia przerwania do PC | Skasowanie zgłoszenia przerwania do PC (dane ignorowane) | |
| 328 | Regulator 1 (silnik lewy) – sterowanie | | dostęp 8 – bitowy (gdy SELREG = 0) |
| 329 | Regulator 1 (silnik lewy) – dane | | |
| 32A | Regulator 2 (silnik prawy) – sterowanie | | |
| 32B | Regulator 2 (silnik prawy) – dane | | |
| 32C | | Wyświetlacz – sterowanie | dostęp 8 – bitowy (gdy SELDIS = 0) |
| 32D | | Wyświetlacz – dane | |
| 32E | Wyświetlacz – sterowanie | | |
| 32F | Wyświetlacz – dane | | |

Tabela 4.1: Rozmieszczenie rejestrów komunikacyjnych karty PC/104 w przestrzeni adresowej PC

**DPCMK1, DPCMK2** – 16-bitowe rejestry do zapisu przez PC, do odczytu przez MK, służące do przesyłania informacji od PC do MK. Zapis danych do DPCMK1 powoduje zgłoszenie przerwania do MK.

**DMKPC** – 16-bitowy rejestr do odczytu przez PC, do zapisu przez MK, służący do przesyłania informacji w stronę od MK do PC.

**SMKPC** – 8-bitowy rejestr stanu przerwań i MK. Bity SMKPC:
7,..,4 – STATMK – bity ustawiane programowo przez MK przez zapis do rejestru STATMK,
3 – INTREG2 – zgłoszenie przerwania przez regulator 1,
2 – INTREG1 – zgłoszenie przerwania przez regulator 2,
1 – INTPCFF – zgłoszenie przerwania przez MK,

0 – MKREADY – gotowość MK do przyjęcia danych (zanegowany stan przerzutnika zgłoszenia przerwania do MK).

Sygnał zgłoszenia przerwania do PC (IRQ5 albo IRQ10 ustawiane zworką) jest sumą sygnałów przerywających od regulatorów INTREG2 i INTREG1 (jeżeli PC ma dostęp do regulatorów, czyli SELREG = 0) oraz przerwania od MK (INTPCFF zgłaszanego przez MK programowo).

Skasowanie zgłoszenia przerwania od MK (INTPCFF) następuje przez zapis PC pod adres 324H (dane są ignorowane). Dodatkowo operacja odczytu przez PC z pod adresu 324H powoduje wytworzenie zbocza sygnału IRQ (o ile jest on aktywny). Jest to niezbędne dla prawidłowego działania systemu przerwań PC, gdy jest ustawione zgłaszanie przerwań zboczem sygnału, a sterownik zgłasza jednocześnie więcej niż jedno przerwanie (np. od regulatorów i od MK).

Przesyłanie danych miedzy PC i MK realizowane jest według zasad podanych niżej:

- PC wpisuje dane do rejestrów komunikacyjnych w kolejności najpierw do DPCMK2 (jeżeli jest to potrzebne) potem do DPCMK1 tylko wtedy gdy MKREADY (SMKPC.0) = 1. Zapis do DPCMK1 powoduje zgłoszenie przerwania do MK i skasowanie bitu MKREADY.

- PC, jako skutek zgłoszenia przerwania lub po stwierdzeniu, że INTPCFF (SMKPC.1) = 1, odczytuje dane z rejestru DMKPC, następnie kasuje zgłoszenie przerwania przez wykonanie zapisu (dane dowolne) pod adres we-wy 324H. W przypadku, gdy są też wykorzystywane (i odmaskowane) przerwania od regulatorów, powinien zostać dodatkowo wykonany odczyt z adresu we-wy 324H (dane do zignorowania) w celu wytworzenia zbocza sygnału zgłoszenia przerwania IRQ, w sytuacji jednoczesnego zgłoszenia przerwania przez MK i regulator.

Komunikacja inicjowana jest zawsze i wyłącznie przez PC. Przy poprawnym wykonaniu komendy (udana komunikacja z regulatorem, aktualne dane z ADC, itp.) MK ustawia 4bity rejestry STATMK (SMKPC) na 0000b. W przypadku wystąpienia błędu, w młodszej połówce tego rejestru znajdzie się jego kod (tab. 4.2).

| **Kod błędu** | **Opis** |
|---|---|
| 0 | Operacja przebiegła pomyślnie |
| 1 | Niepoprawny kod rozkazu |
| 2 | Niepoprawna wartość parametru (np. żądana prędkość jest spoza dopuszczalnego zakresu) |
| 3 | Dane nie gotowe (np. brak nowego odczytu z ADC, czujników odległości) |
| 4 | Wystąpił błąd podczas komunikacji z regulatorem 1 (lewy) |
| 5 | Wystąpił błąd podczas komunikacji z regulatorem 2 (prawy) |
| 15 | Aktywny STOP AWARYJNY, konieczność ręcznego skasowania |

Tabela 4.2: Kody błędów komunikacji z kartą PC/104

Opis dostępnych rozkazów mikrokontrolera znajduje się w poniższej tabeli (tab. 4.3).

| **DPCMK1 – bity 16..8** | | | | | | | | **DPCMK1 – bity 7..0** | **Funkcja** |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | Żądanie resetu (inicjalizacji) MK |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | | Odczyt statusu:<br>– sygnałów zewnętrznych KEY1, KEY2, TERMAL FLAG1, TERMAL FLAG2 (ustawiane na bitach 11..8 DMKPC)<br>– sygnałów zewnętrznych EXTIN, młodsze 8bitów DMKPC |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | EXTVAL | Zapis stanu sygnałów zewnętrznych EXTOUT (z ew. maskowaniem części bitów przez MK). Parametr EXTVAL (8bit) |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | P | | Odczyt wejść analogowych. Wartość zwracana jest poprzez rejestr DMKPC:<br>P = 0: odczyt z wejść P1.0, P1.1: |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | – prąd silnika 1 (lewego), starsze 8bitów DMKPC<br>– prąd silnika 2 (prawego), młodsze 8bitów DMKPC<br>P = 1: odczyt z wejść P1.2, P1.3:<br>– napięcie zasilania (akumulatora lub zasilacza zewnętrznego), starsze 8bitów DMKPC<br>– napięcie zasilania dalmierza laserowego, młodsze 8bitów DMKPC |
| 0 | 0 | 0 | 0 | 1 | 0 | R | L | | Kontrola sterowania regulatorami LM629 oraz LCD przez PC:<br>R – bit określający sterowanie regulatorami LM629<br>L bit określający sterowanie LCD<br>1 = sterowanie z PC, 0 = sterowanie przez MK |
| 0 | 0 | 0 | 0 | 1 | 1 | P | S | SERVAL | Sterowanie serwami modelarskimi:<br>S wybór serwo 0 lub 1<br>P = 0 sterowanie pozycją, parametr SERVAL w zakresie 1...255, dla 0 = sygnał sterujący nie jest generowany<br>P = 1 sterowanie prędkością, parametr SERVAL w zakresie 1...255, dla 0 = następuje skokowa zmiana położenia |
| 0 | 0 | 0 | 0 | 0 | 0 | m | m | 00iiiiii | Sterowanie odpytywaniem o wejścia analogowe: mmiiiiii |
| R | L | 0 | 0 | 0 | 0 | 0 | 0 | | Natychmiastowe zatrzymanie silników (podanie zerowego sterowania) |
| R | L | 0 | 0 | 0 | 0 | 0 | 1 | | Łagodne zatrzymanie silników |
| R | L | 0 | 0 | 0 | 0 | 1 | 0 | | Załadowanie parametru *il* regulatora (DPCMK2 – 16-bit) |
| R | L | 0 | 0 | 0 | 0 | 1 | 1 | | Załadowanie parametru *kd* regulatora (DPCMK2 – 16-bit) |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| R | L | 0 | 0 | 0 | 1 | 0 | 0 | | Załadowanie parametru *ki* regulatora (DPCMK2 – 16-bit) |
| R | L | 0 | 0 | 0 | 1 | 0 | 1 | | Załadowanie parametru *kp* regulatora (DPCMK2 – 16-bit) |
| R | L | 0 | 0 | 0 | 1 | 1 | 0 | DSIVAL | Załadowanie parametru *Derivative Sampling Interval* regulatora (parametr DSIVAL, 8bit) |
| R | L | 0 | 0 | 1 | 0 | 0 | 0 | | Odczyt aktualnego *położenia* |
| R | L | 0 | 0 | 1 | 0 | 0 | 1 | | Odczyt aktualnej *prędkości* |
| R | L | 0 | 0 | 1 | 1 | 0 | 0 | ACC_H | Załadowanie wartości *przyśpieszenia.*<br>Parametr 24bit:<br>ACC_H - starsze 8bit, DPCMK2 młodsze 16bit. |
| R | L | 0 | 0 | 1 | 1 | 0 | 1 | VEL_H | Załadowanie wartości *prędkości* dla sterowania pozycyjnego.<br>Parametr 24bit:<br>VEL_H - starsze 8bit, DPCMK2 młodsze 16bit. |
| R | L | 0 | 0 | 1 | 1 | 1 | F | VEL_H | Załadowanie wartości prędkości dla sterowania prędkościowego:<br>F = 0 – kierunek do tyłu,<br>F = 1 – kierunek do przodu,<br>Parametr 24bit:<br>VEL_H - starsze 8bit, DPCMK2 młodsze 16bit. |
| R | L | 0 | 0 | | | | | POSVAL_H | Załadowanie wartości prędkości dla sterowania prędkościowego:<br>F = 0 – kierunek do tyłu,<br>F = 1 – kierunek do przodu,<br>Parametr 28bit:<br>POSVAL_H - starsze 12bit, DPCMK2 młodsze 16bit. |

Tabela 4.3: Zestawienie dostępnych rozkazów mikrokontrolera

### 4.1.5 Obsługa komunikacji przez komputer PC

Komputer PC robota mobilnego pracuje pod kontrolą systemu operacyjnego Linux. W tym systemie nie jest możliwa obsługa przerwań przez programy użytkownika, a jedynie sterowniki urządzeń pracujące jako część jądra systemu operacyjnego.

Na potrzeby komunikacji z mikrokontrolerem znajdującym się na karcie PC/104 został stworzony taki sterownik, zaimplementowany jako moduł jądra [6]. Dzięki temu możliwe jest dynamiczne (tj. w czasie pracy systemu) jego ładowanie i usuwanie za pomocą skryptu, który tworzy niezbędne pliki w katalogu `/dev`.

Moduł udostępnia interfejs komunikacji z mikrokontrolerem w postaci urządzenia znakowego (ang. *char device*) o dynamicznie przydzielanych numerach głównym i porządkowym (ang. *major* i *minor*). Przesyłanie danych odbywa się za pośrednictwem funckji *ioctl()* przyjmującej jako parametry: identyfikator żądanej funkcji oraz wskaźnik do zmiennej. Moduł obsługuje tylko jeden taki identyfikator, natomiast numer realizowanej funkcji zakodowany jest w zmiennej 32-bitowej wraz z jej argumentami i przesyłany bezpośrednio do mikrokontrolera. Uzasadnieniem takiego rozwiązania (w odróżnieniu od implementowania osobnego wywołania *ioctl()* dla każdej realizowanej funkcji) jest prostota i szybkość, gdyż testowanie modułów jądra jest znacznie bardziej czasochłonne a ewentualne pomyłki bardziej kosztowne w porównaniu z biblioteką programu użytkownika. Rezultat zapytania zwracany jest przez tę samą zmienną 32–bitową. Możliwe jest jednoczesnie korzystanie z pliku urządzenia znakowego związanego z mikrokontrolerem przez wiele programów bez zagrożenia przepisywaniem rejestrów komunikacyjnych, przy czym synchronizacja dostępu zagwarantowana jest dzięki wykorzystaniu semaforów. Takie rozwiązanie zostało podyktowane potrzebą elastyczności w późniejszym tworzeniu sterowników dla różnych urządzeń obslugiwanych przez mikrokontroler - silników, serwomechanizmów modelarskich, układu kontroli zasilania oraz zestawu modułów akwizycji danych z wejść analogowych. Możliwe jest dekompozycja obsługi wymienonych modułów funkcjonalnych na osobne procesy bez konieczności dalszego zapewniania synchronizacji dostępu do zasobu.

### 4.1.6 Programy PC obsługi regulatorów

Dla potrzeb przetestowania działania komunikacji z mikrokontrolerem przez urządzenie implementowane jako moduł jądra zostały przygotowane dwa programy testujące. Pierwszy z nich służy do przesyłania pojedynczych komend do regulatorów LM629, zaś drugi służy do cyklicznego odczytu zmiany kąta

obrotu kół – co po pomnożeniu przez obwód koła i podzieleniu przez okres próbkowania w pewnym przybliżeniu opisuje prędkość robota.

### Program sterowania regulatorami

Program obsługi regulatorów umożliwia bezpośrednie modyfikowanie parametrów regulatora PID oraz wartości zadanych. Składnia wywołania jest następująca:

```
usage: ./test /dev/devname motor_id command_id value
motor_id: 1 = left, 2 = right, 3 = both
command_id: A   = acceleration
            V   = speed (positional mode)
            v   = speed (velocity mode)
            p   = relative position
            P   = read position
            il  = integration limit
            kd  = derivative term
            ki  = integral term
            kp  = proportional term
            dsi = derivational sampling interval
```

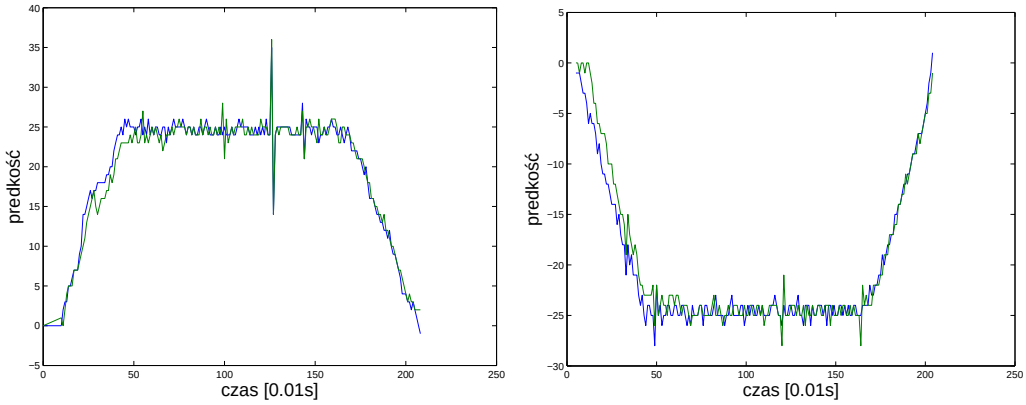
### Program raportujący ruch silników

W celu weryfikacji działania układu regulacji został stworzony prosty program, którego składnia wywołania jest identyczna z wcześniej przedstawioną, zaś po realizacji komendy cyklicznie (co 10 milisekund) odczytywane jest przemieszczenie robota.

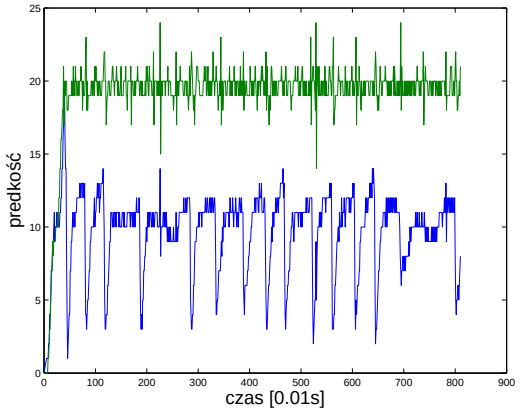
Dla zachowania regularności odczytu wykorzystane zostało wywołanie systemowe *setitimer()*. Przy częstotliwości przerwań zegarowych 100 Hz (czyli co 10ms) gwarantuje ono regularne odpytywanie układów LM629 (za pośrednictwem mikrokontrolera).

Poniżej zostały zamieszczone wykresy prędkości obydwu kół przy jeździe na wprost i do tyłu dla nastaw regulatorów $kp = 200, ki = kd = 0$ (rysunek 4.3). Widoczny jest trapezowy kształt przebiegu prędkości, co jest zgodne z oczekiwaniami.

Jednocześnie przy skręcaniu (kiedy koła poruszają się z różnymi prędkościami) występują bardzo znaczące uchyby w prędkości koła znajdującego się po wewnętrznej stronie łuku – strojenie parametrów nie doprowadziło tutaj do poprawy pracy regulatorów.




Rysunek 4.3: Prędkości kół przy jeździe na wprost i do tyłu




Rysunek 4.4: Prędkości kół przy skręcaniu

Zgodnie ze wskazówkami [12] zalecane jest iteracyjne dobieranie kolejno parametrów $K_p$, $K_i$ i $K_d$. Metody opierające się na badaniu odpowiedzi układu w zamkniętej pętli sprzężenia mają tutaj ograniczone zastosowanie ze względu na arytmetykę cyfrowej realizcji algorytmu PID w układzie LM629. Problemem jest zmiejszanie i rozszerzanie precyzji (reprezentacji) wyników pośrednich. Druga z zalecanych metod strojenia – badanie zachowania wału silnika przy jego ręcznym odchylaniu od ustalonego położenia jest nierealizowalne ze względu na zastosowane przekładnie. Także badanie oscyloskopem sygnału sterującego bezpośrednio na zaciskach silników przy generowanych małych przesunięciach jest praktycznie nierealizowalne.

Problemy z działaniem regulatorów występują jedynie przy skręcaniu (objawia się to widocznymi szarpnięciami koła znajdującego się po wewnętrzenej

stronie łuku) i są spowodowane w dużej mierze oporami wynikającymi z mechanicznej konstrukcji napędu.

## 4.2 Podłączenie czujników IR

Czujniki odległości IR firmy *Sharp* – *GP2D120* i *GP2Y0A02* umieszczone na obwodzie robota dają na wyjściu napięciowy sygnał analogowy, związany zależnością nieliniową z rzeczywistą odległością czujnika od obiektu. Zamontowanych w każdym robocie jest 18 takich czujników – 13 krótkiego i 5 dalekiego zasięgu. Ze względu na dużą liczbę sygnałów analogowych oraz praktyczne względy prowadzenia kabli w korpusie robota czujniki podłączone są do czterech modułów wejścia analogowego zrealizowanych na mikrokontrolerze *PIC16F688*.

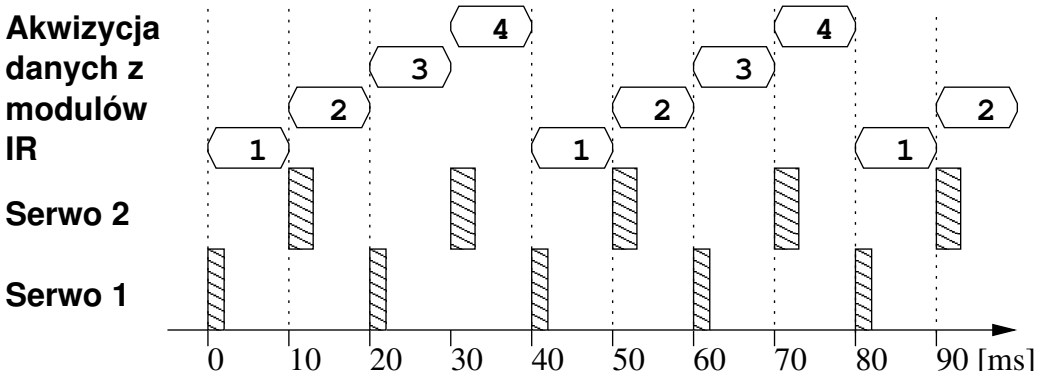
Układ *PIC16F688* wyposażony jest w 6 wejść konwersji analogowo cyfrowej, do których podłączone są czujniki IR oraz w układ transmisji szeregowej podłączony do magistrali RS485. Mikrokontroler umieszczony na karcie PC/104 cyklicznie odpytuje 4 moduły akwizycji danych analogowych o stan wejść i te dane udostępnia dalej dla komputera PC. Żądaniem przesłania danych jest bajt z adresem modułu (2 bity) oraz wejściami które mają być zasilane (pozostałe 6 bitów) i ustawionym 9-tym bitem transmisji. Bajty odpowiedzi mają 9-ty bit zgaszony – w ten sposób zagwarantowana jest bezkolizyjność na magistrali RS485.

Oprogramowanie mikrokontrolera *AT89C51AC2* firmy *Atmel* do odpytywania modułów wykorzystuje moduł transmisji szeregowej oraz układy czasowe. Czujniki IR udostępniają pomiar z częstotliwością 25Hz, stąd optymalne jest ich odpytywanie co 40ms każdy. Dwa serwomechanizmy podłączone do procesora *Atmel* wymagają sterowania sygnałem o długości proporcjonalnej do żądanego wychylenia o częstotliwości 50Hz. Stąd optymalny rozkład obsługi modułów opiera się na pętli 10ms, w której cyklicznie obsługiwane są czujniki IR oraz generowany jest sygnał sterujący dla serwomechanizmów modelarskich (rys. 4.5). Do obsługi zależności czasowych wykorzystywane są następujące elementy:

**Timer0** – pracujący w trybie 8-bitowym odlicza czas trwania impulsu dla serwomechanizmu modelarskiego w zakresie $0.9 \ldots 2.1$ ms,

**Timer1** – pracujący w trybie 16-bitowym generuje przerwanie będące podstawą cyklu obsługi modułów IR oraz serwomechanizmów o okresie 10ms,

**Timer2** – pracujący w trybie 16-bitowym jest użyty do generowania częstotliwości pracy portu szeregowego 19200bps.




Rysunek 4.5: Cykl obsługi modułów IR oraz serwomechanizmów

## 4.3 Szybka komunikacja z dalmierzem laserowym

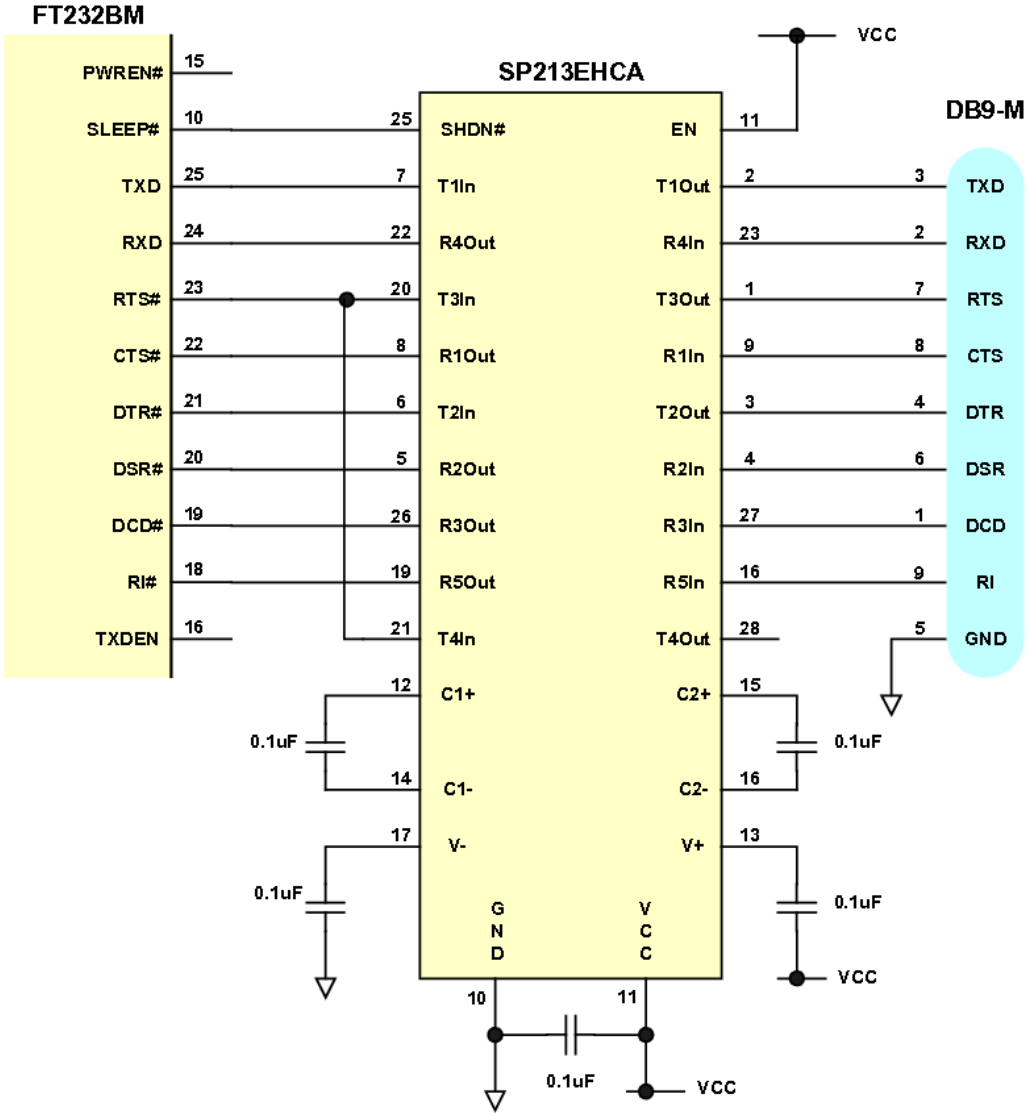
Dalmierz laserowy będący na wyposażeniu robota – model *LMS200* firmy *SICK*, standardowo wyposażony jest w interfejsy RS232 oraz RS422 do komunikacji szeregowej z komputerem. Komunikacja odbywa się na zasadzie wymienianych żądań i potwierdzeń konfiguracyjnych, po czym urządzenie przechodzi w tryb ciągłego przesyłania pomiarów odległości w promieniu skanowania.

Komunikacja z komputerem może przebiegać z prędkościami 9600bps, 38400bps oraz 500000bps, przy czym ta ostatnia, najszybsza transmisja możliwa jest tylko przy wykorzystaniu połączenia RS422 [21]. Od prędkości transmisji zależy maksymalna częstotliwość oraz rozdzielczość uzyskiwania danych i tak możliwe są tryby pracy:

- rozdzielczość 0.5 stopnia, 361 odczytów,
  5Hz (38400bps) lub 32Hz (500000bps),
- rozdzielczość 1 stopień, 181 odczytów,
  10Hz (38400bps) lub 75Hz (500000bps).

Jak widać częstotliwość przesyłania odczytów jest wprost proporcjonalna do prędkości na łączu RS232/RS422 i odwrotnie proporcjonalna do ilości przesyłanych danych (rozdzielczości skanowania).

Transmisja 500000bps może być realizowana z użyciem specjalizowanych kart PCI i PCMCIA produkowanych przez firmę *SICK*, które są stosunkowo kosztowne i trudno dostępne, bądź też może być realizowana z użyciem konwerterów USB pozwalających w dowolny sposób ustawiać prędkość transmisji

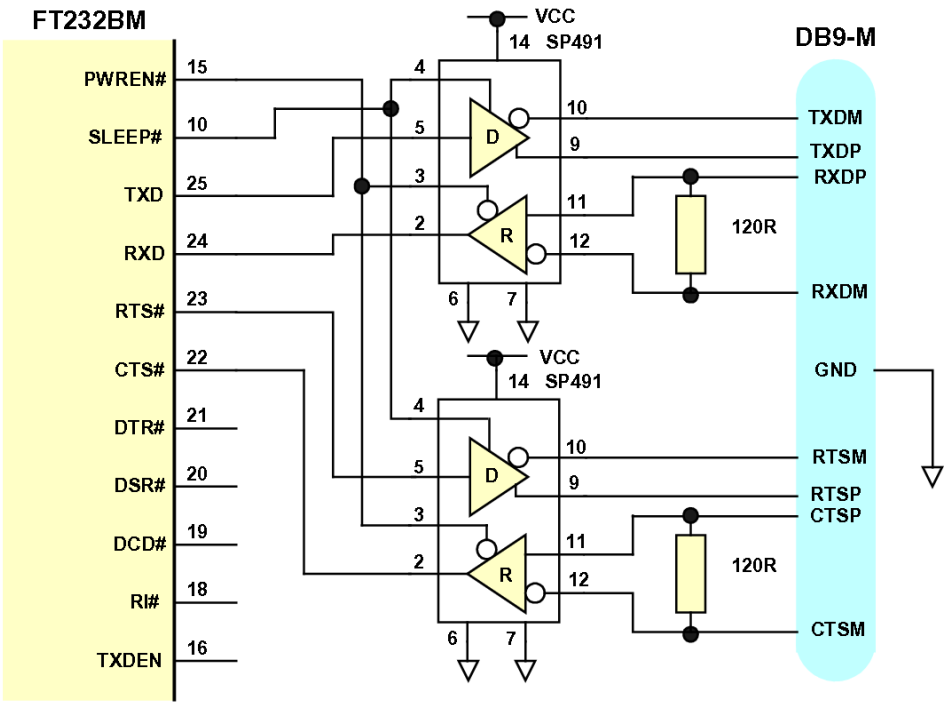



Rysunek 4.6: Schemat konwertera USB↔RS232 [9]

szeregowej w granicach (w zależności od producenta układu) rzędu 300bps – 3Mbps. Przemysłowe wersje konwerterów USB z wyprowadzeniami standardu RS422 są około 7-krotnie droższe niż powszechnie używane, obsługujące tylko standard RS232 z powodu galwanicznych elementów zabezpieczających oraz szybkich optoizolatorów.

W trakcie prac nie udało się nawiązać komunikacji z dalmierzem korzystając z kart PCI szybkiej transmisji szeregowej firm *Advantech* oraz *Moxa* z po-

wodu braku możliwości konfiguracji prędkości dokładnie 500kbps. Ze względu na sposób konstrukcji – użyte oscylatory kwarcowe i binarne dzielniki częstotliwości możliwa jest jedynie konfiguracja 460.8kbps a dalej 921kbps.




Rysunek 4.7: Schemat konwertera USB↔RS422, tylko linie RXD/TXD są wykorzystywane (jeden układ SP491) [9]

Została wykonana modyfikacja powszechnie dostępnego konwertera standardów USB↔RS232 opartego na układzie *FT232BM*, zgodnie ze schematami z karty katalogowej [9]. Układ konwersji na poziomy RS232 został zastąpiony układem *MAX490* (odpowiednik widniejącego na schematach *SP491*) – umożliwiającym dwukierunkową transmisję z prędkością do 2500kbps, przy czym tylko linie RXD/TXD są wykorzystywane do komunikacji z dalmierzem i tylko one zostały podłączone do nowego układu konwersji (rys. 4.6, 4.7).

## 4.4 Obraz systemu Linux dla robota mobilnego

Robot mobilny *ELEKTRON* wyposażony jest w komputer PC klasy *embedded PC* z procesorem (w zależności od egzemplarza) *Celeron ULV 650MHz*

oraz *Pentium III LV 933MHz* oraz 256MB i 512MB pamięci opracyjnej RAM. Funkcję pamięci masowej pełni karta Compact Flash o pojemności 512MB.

Ze względu na specyfikę zastosowania oraz ograniczone zasoby sprzętowe konieczne było przygotowanie specjalizowanego obrazu systemu *Linux*. Kluczowe w takiej konfiguracji są:

- ograniczenie do minimum pamięci na karcie *Compact Flash* zajmowanej przez system operacyjny – przez zainstalowanie tylko faktycznie wykorzystywanych pakietów, a wręcz wyłącznie ich niezbędnych elementów, czyli np. z pominięciem dokumentacji,
- ograniczenie do minimum ilości pamięci wykorzystywanej przez działające w systemie usługi (demony),
- optymalizacja konfiguracji jądra systemu dla posiadanego zestawu sprzętu – tj. kompilacja sterowników tylko dla urządzeń znajdujących się na płycie głównej i zainstalowanych jako karty rozszerzeń,
- kompilacja całości systemu (jądra, bibliotek oraz programów) z optymalizacją dla procesora Pentium, co zapewnia możliwie najpełniejsze wykorzystanie mocy obliczeniowej jednostki,
- przygotowanie systemu do pracy z nośnika *Compact Flash* w trybie *read–only* – spowodowane ograniczoną ilością cykli zapisu w tego typu pamięci masowej,
- możliwość łatwej modyfikacji zestawu pakietów w obrazie systemu i ich aktualizacji,
- możliwość utrzymania spójności pomiędzy obrazami systemu na wszystkich egzemplarzach robotów.

W celu spełnienia wyżej wymienionych wymagań obraz systemu *Linux* dla robota został zbudowany na dystrybucji *Gentoo*, której główną cechą charakterystyczną jest system pakietów (ang. *portage*) nastawiony na instalację przez kompilację z kodów źródłowych w odróżnieniu od pakietów z plikami wykonywalnymi. Kompilacja wszystkich elementów systemu odbywa się ze zdefiniowanymi flagami optymalizacji oraz dyrektywą nakazującą niedołączanie części dokumentacji:

```
# plik /etc/make.conf
CFLAGS="-O3 -pipe -march=pentium3 \
        -fomit-frame-pointer -funroll-loops"
CXXFLAGS="${CFLAGS}"
FEATURES="nodoc noinfo noman"
```

Obraz tworzony był na bazie o drzewa katalogów standardowej, podstawowej konfiguracji *Gentoo*, tzw. *stage2* zainstalowanej w katalogu `/s2` pełnej konfiguracji tej samej dystrybucji. Takie podejście zapewnia dostęp do systemu typu stacja robocza na komputerze stacjonarnym, na którym jest możliwość przetestowania oprogramowania, skorzystania z jego dokumentacji, itp. Instalacja, bądź aktualizacja pakietu w obrazie odbywa się za pomocą polecenia:

```
root@lenin# ROOT="/s2" emerge -av _nazwa_pakietu_
```

Synchronizacja bazy pakietów między pełnym systemem stacji roboczej oraz systemem robota zapewniona jest przez zamontowanie katalogu bazy (`/usr/portage`) w drzewie obrazu systemu docelowego:

```
mount -o bind /usr/portage /s2/usr/portage
```

Spójność pomiędzy obrazem systemu przechowywanym na komputerze stacjonarnym, a zawartością kart pamięci *Comapct Flash* robotów utrzymywana jest za pomocą narzędzia `rsync`. Służy ono do synchronizowania zawartości plików i katalogów, przy czym porównywanie odbywa się w pierwszej kolejności na podstawie rozmiaru i czasu modyfikacji, a dodatkowo wartości specjalnej funkcji skrótu (ang. *hash*) zawartości plików. W celu aktualizacji przesyłane są bloki danych tylko tych plików, które wymagają uspójnienia i to tylko te fragmenty, które faktycznie się różnią. W ten sposób przez łącze WLAN przesyłana jest możliwie mała ilość danych, a opracje zapisu na karcie pamięci ograniczone są do minimum. Dodatkowo z transferu wykluczone są katalogi zawierające te zbędne elemeny pakietów oprogramowania, które nie mogą być pominięte przez flagi używane przy ich instalacji (opcje `USE` i `FEATURES` w pliku `/etc/make.conf`).

W celu zsynchronizowania zawartości karty *Compact Flash* robota z katalogiem na komputerze stacjonarnym należy skorzystać ze skrytpu:

```
root@pcm1# ./rsync.cmd
```

# Rozdział 5

# Oprogramowanie uzupełniające

## 5.1 Model robota na potrzeby symulacji

Możliwość badania algorytmów sterowania robotami z wykorzysteniem symulatorów znacząco ułatwia i przyśpiesza czas tworzenia oprogramowania. Korzystne zatem jest posiadanie modelu robota oraz środowiska symulacyjnego, które w możliwie pełny sposób oddawałyby zachowanie rzeczywistego robota. Istotną cechą jest tutaj zgodność interfejsu programistycznego do sterowania robotem i jego modelem w symulatorze – w idealnym przypadku sprawia ona, że ten sam program można bez modyfikacji uruchamiać w obydwu środowiskach.

Oprogramowanie *Player/Stage*, dzięki swej modularnej budowie, dobrze spełnia wymienione wymaganie – program sterujący robotem komunikuje się z serwerem *Player*, a nie odwołuje się bezpośrednio do sensorów i efektorów robota. To od konfiguracji serwera zależy, czy klient bedzie korzystał z rzeczywistych urządzeń, czy działał wykorzystując symulator.

Dostępne są trzy symulatory przystosowane do działania z serwerem *Player*: symulator dwuwymiarowy *Stage* uwzględniający wyłącznie kinematykę robotów oraz symulatory trójwymiarowe *Gazebo* oparty całkowicie na wolnodostępnym oprogramowaniu oraz *USARsim* oparty na silniku popularnej gry *Unreal Tournament* [4].

### 5.1.1 Model robota w symulatorze *Stage*

W symulatorze *Stage* możliwe jest modelowanie robotów w statycznym płaskim środowisku dwuwymiarowym – jak np. piętro budynku, którego plan jest wczytywany w postaci graficznej (binarna mapa zajętości). Symulowane są odczyty m.in. z czujników odległości (sonary), dalmierza laserowego oraz

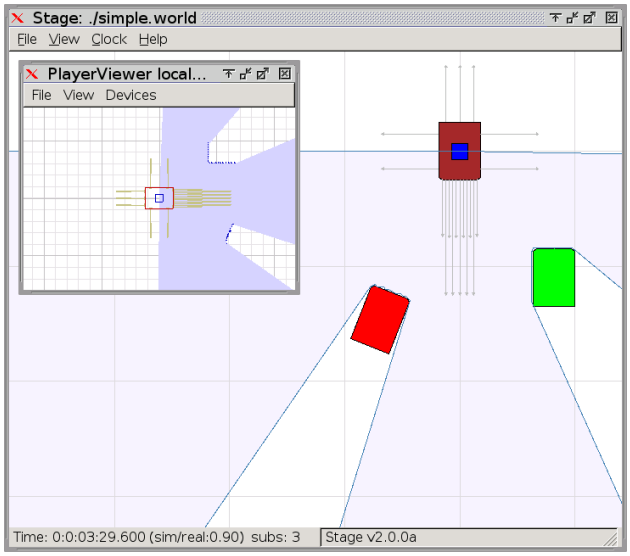
pomiary odometryczne. Plik z konfiguracją modelu `elektron.inc` uwzględnia rozmieszczenie i zasięg poszczególnych czujników odległości (tab. 5.1) oraz wymiary (tab. 5.2) robota (rys. 5.1).

```
# The ELEKTRON sonar array
define elektron_sonar ranger
(
  scount 18

  # define the pose and field of view of each transducer
  # [xpos ypos heading] [range_min range_max view_angle]
  spose[0]  [ 0.24  0.15 0 ]   sview[0]  [ 0.04 0.5 1 ]
  spose[1]  [ 0.25  0.12 0 ]   sview[1]  [ 0.20 1.0 1 ]
  spose[2]  [ 0.25  0.09 0 ]   sview[2]  [ 0.04 0.5 1 ]
  spose[3]  [ 0.25  0.06 0 ]   sview[3]  [ 0.20 1.0 1 ]
  spose[4]  [ 0.25  0.03 0 ]   sview[4]  [ 0.04 0.5 1 ]
  spose[5]  [ 0.25  0.00 0 ]   sview[5]  [ 0.20 1.0 1 ]
  spose[6]  [ 0.25 -0.03 0 ]   sview[6]  [ 0.04 0.5 1 ]
  spose[7]  [ 0.25 -0.06 0 ]   sview[7]  [ 0.20 1.0 1 ]
  spose[8]  [ 0.25 -0.09 0 ]   sview[8]  [ 0.04 0.5 1 ]
  spose[9]  [ 0.25 -0.12 0 ]   sview[9]  [ 0.20 1.0 1 ]
  spose[10] [ 0.24 -0.15 0 ]   sview[10] [ 0.04 0.5 1 ]
  spose[11] [ 0.15  0.18 90]   sview[11] [ 0.04 0.5 1 ]
  spose[12] [-0.15  0.18 90]   sview[12] [ 0.04 0.5 1 ]
  spose[13] [ 0.15 -0.18 -90]  sview[13] [ 0.04 0.5 1 ]
  spose[14] [-0.15 -0.18 -90]  sview[14] [ 0.04 0.5 1 ]
  spose[15] [-0.25  0.12 180]  sview[15] [ 0.04 0.5 1 ]
  spose[16] [-0.25  0.00 180]  sview[16] [ 0.04 0.5 1 ]
  spose[17] [-0.25 -0.12 180]  sview[17] [ 0.04 0.5 1 ]

  # define the size of each transducer
  # [xsize ysize] in meters
  ssize [0.01 0.01]
)
```

Tabela 5.1: Konfiguracja rozmieszczenia i zasięgu poszczególnych czujników robota dla symulatora Stage

```
# a ELEKTRON in standard configuration
define elektron position
(
  # actual size
  size [0.50 0.36]

  # the ELEKTRON's center of rotation
  # is offset from its center of area
  origin [0.0 0.0 0]

  # draw a nose on the robot so we can see which way it points
  gui_nose 1
  gui_boundary 0

  # estimated mass in KG
  mass 15.0

  # this polygon approximates the shape of a ELEKTRON
  polygons 1
  polygon[0].points 6
  polygon[0].point[0] [  0.25  0.15 ]
  polygon[0].point[1] [  0.23  0.18 ]
  polygon[0].point[2] [ -0.25  0.18 ]
  polygon[0].point[3] [ -0.25 -0.18 ]
  polygon[0].point[4] [  0.23 -0.18 ]
  polygon[0].point[5] [  0.25 -0.15 ]
  polygon[0].filled 1

  # differential steering model
  drive "diff"

  # use the sonar array defined above
  elektron_sonar()
)
```

Tabela 5.2: Konfiguracja wymiarów robota dla symulatora Stage



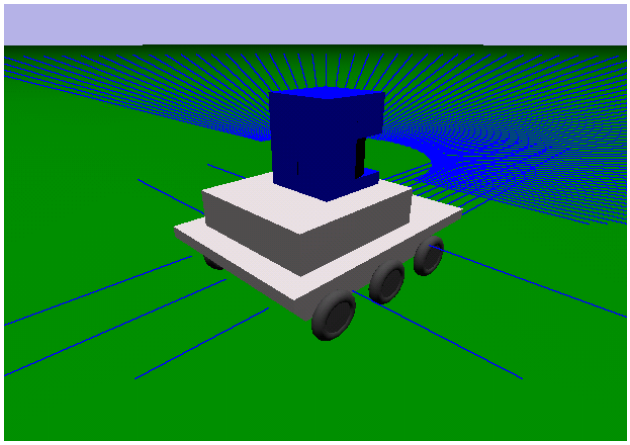

Rysunek 5.1: Model robota *ELEKTRON* z czujnikami w symulatorze 2D

### 5.1.2 Model robota w symulatorze *Gazebo*

W symulatorze trójwymiarowym *Gazebo* możliwe jest nie tylko modelowanie robotów połączone z wizualizacją, ale także uwzględniana jest dynamika ruchu obiektów (a zatem także tarcie) oraz możliwe jest oddziaływanie robota na środowisko (takie jak przesuwanie czy podnoszenie obiektów).

Silnikiem symulacji jest biblioteka ODE (*Open Dynamics Engine* [22]), zaś roboty są modelowane jako zestaw brył połączonych przegubami (ang. *joints*). W przypadku robota *ELEKTRON* model stanowią prostopadłościenne bryły korpusu, do których przegubami obrotowymi dołączone zostały koła. Tak jak w rzeczywistym robocie napęd przenoszony jest z osi silnika na 3 koła za pomocą paska, w modelu koła otrzymują taką samą prędkość obrotową, co odpowiada pracy układów LM629 w trybie regulacji prędkości (rys. 5.2).

Symulator *Gazebo* obsługuje wiele jednoczesnych widoków z kamer (co jest niezbędne np. przy parach stereowizyjnych) stąd w założeniu autorów rendering sceny przeprowadzany jest do bufora pamięci, a nie widocznego okna systemu graficznego. W przypadku systemu *Linux* akceleracja oferowana przez układy większości kart graficznych działa jedynie przy renderingu bezpośrednio na ekran. Wprowadzenie modyfikacji polegającej na bezwzględnym wizualizowaniu sceny znacząco poprawiło wydajność symulatora. Należy zaznaczyć, że nawet z wykorzystaniem sprzętowej akceleracji przez kartę graficzną to właśnie wizualizacja jest najbardziej czasochłonnym elementem – z jej pominięciem możliwe jest symulowanie nawet bardzo złożonych układów



Rysunek 5.2: Model robota *ELEKTRON* z czujnikami w symulatorze 3D

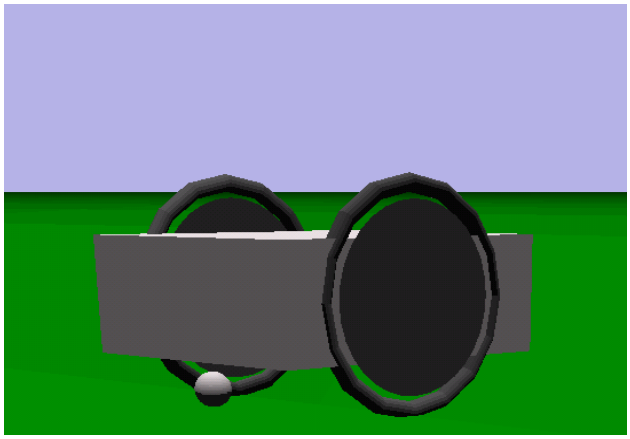
w czasie rzeczywistym.

Wprowadzony został także model innych robotów, będących w posiadaniu IAIiS – o robczej nazwie HMT. Ich przeznaczeniem jest przede wszystkim gra w zawodach piłkarskich RoboCup, jednak planowane także jest ich wykorzystanie w pracach nad strukturami sterowania zespołem robotów heterogenicznych. Jednym z postulatów tych prac jest stworzenie jednolitych narzędzi programowania różnych typów robotów – zatem oczywistym krokiem jest stworzenie możliwości kontroli wszystkich robotów z poziomu serwera *Player*.

Model robota HMT stanowią prostopadłościenny korpus, do którego przyłączonę są dwa niezależnie napędzane koła. Do utrzymywania równowagi wykorzystywane są dwie kuliste podpory – także uwzględnione w modelu trójwymiarowym (rys. 5.3).

Istotną zaletą symulatora *Gazebo* jest możliwość modelowania wielu obiektów tego samego typu – dodawanie do środowiska kolejnych egzemplarzy tego samego robota polega na dopisaniu kolejnej sekcji w pliku konfiguracyjnym XML (tab. 5.3) oraz dodaniu do konfiguracji serwera *Player* sekcji sterowników kolejnych urządzeń (tab. 5.4).

Pozycja robota określana jest przez położenie (`<xyz>`) oraz zestaw kątów *roll–pitch–yaw* (`<rpy>`). Identyfikator (`<id>`) służy do skojarzenia modelu z interfejsem serwera *Player*. Jeden z modelowanych robotów jest wyposażony w dalmierz laserowy SICK – co uzyskane zostało tylko przez dodanie sekcji `<model:SickLMS200>`. Definicje obydwu robotów dodatkowo zawierają sekcję `<model:TruthWidget>` – dzięki której możliwe jest uzyskanie rzeczywistej pozycji, nieobarczonej błędami odometrii.



Rysunek 5.3: Model robota *HMT* w symulatorze 3D

```
<model:Protonek>
  <plugin>Protonek.so</plugin>
  <id>watcher</id>
  <xyz>0.0 8.0 0.0</xyz>
  <rpy>0 0 -90</rpy>
  <model:SickLMS200>
    <id>laser0</id>
    <xyz>0.0 0.0 0.30</xyz>
  </model:SickLMS200>
  <model:TruthWidget>
    <id>watcher_truth</id>
  </model:TruthWidget>
</model:Protonek>
```

```
<model:Protonek>
  <plugin>Protonek.so</plugin>
  <id>pusher1</id>
  <xyz>-0.8 -0.33 0.0</xyz>
  <rpy>0 0 90.0</rpy>

  <model:TruthWidget>
    <id>pusher1_truth</id>
  </model:TruthWidget>
</model:Protonek>
```

Tabela 5.3: Plik XML z konfiguracją robotów dla symulatora Gazebo

| | |
|---|---|
| **driver**<br>(<br>  **name** "gz_position"<br>  **provides** ["position:0"]<br>  gz_id "watcher"<br>) | **driver**<br>(<br>  **name** "gz_position"<br>  **provides** ["position:1"]<br>  gz_id "pusher"<br>) |
| **driver**<br>(<br>  **name** "gz_sonar"<br>  **provides** ["sonar:0"]<br>  gz_id "watcher"<br>) | **driver**<br>(<br>  **name** "gz_sonar"<br>  **provides** ["sonar:1"]<br>  gz_id "pusher1"<br>) |
| **driver**<br>(<br>  **name** "gz_laser"<br>  **provides** ["laser:0"]<br>  gz_id "laser0"<br>) | |
| **driver**<br>(<br>  **name** "gz_truth"<br>  **provides** ["truth:0"]<br>  gz_id "watcher_truth"<br>) | **driver**<br>(<br>  **name** "gz_truth"<br>  **provides** ["truth:1"]<br>  gz_id "pusher1_truth"<br>) |

Tabela 5.4: Plik z konfiguracją dla serwera Player odpowiadający konfiguracji symulatora Gazebo

W celu uruchomienia symulatora należy jako parametr wywołania podać ścieżkę do pliku XML z definicją świata, np.:

```
$ gazebo protonek.world
```

W zależności od tego, czy w pliku zostały skonfigurowanane kamery (sekcje `<model:ObserverCam>`) zostaną wyświetlone okna wizualizacji. Następnie należy uruchomić serwer *Player* podając jako parametry identyfikator symulatora (jednocześnie może działać wiele niezależnych symulacji) oraz ścieżkę do pliku z odpowiadającą konfiguracją urządzeń serwera:

```
$ player protonek.cfg
```

Traktowanie robota *ELEKTRON* jako monocyklu wprowadza błędy tak

w rzeczywistości jak i w symulatorze. Przy jeździe na wprost przemieszczenie wyznaczone na podstawie odometrii bardzo dobrze pokrywa się ze stanem faktycznym. W czasie skrętu modelowany pojazd tak w rzeczywistości, jak i w symulacji ślizga się na powierzchni.

## 5.2 Budowanie map

Posiadanie mapy terenu, po którym porusza się robot jest niezbędne do implementowania, testowania oraz weryfikacji algorytmów nawigacji globalnej. Jest przy tym ważne, aby prace odbywały się na mapie możliwie dokładnie oddającej rzeczywisty teren czy obiekt, w jakim będzie działał robot.

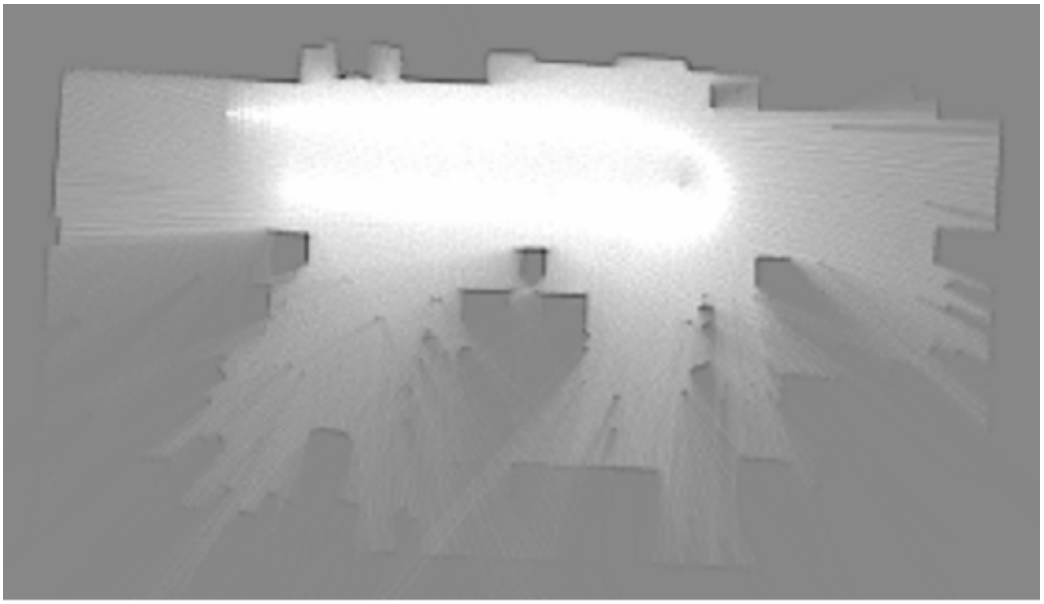
Do sporządzenia mapy laboratorium robotyki gdzie znajdują się roboty zostało wykorzystane oprogramowanie *PMAP* [26]. Działa ono w oparciu o zapis pomiarów dalmierza laserowego i odometrii wykonywany przy eksploracji (w tym przypadku sterowanej zdalnie przez operatora) pomieszczenia. Zebrane dane są analizowane z wykorzystaniem filtru cząsteczkowego w celu minimalizacji wpływu błędów odometrii na tworzoną mapę. Pomiary z dalmierza są wykorzystywane jednocześnie do korekcji danych odometrycznych (położenia robota) oraz do modelowania otoczenia robota. Od ustawień parametrów filtru zależy dokładność i rozdzielczość wynikowej mapy.

Poniżej przedstawiono wynik działania programu na danych z jednego przejazdu robotem po terenie laboratorium (rys. 5.4). Robot w trakcie przejazdu zatoczył jedną pełną pętlę. Mapa wynikowa jest uzyskiwana w odcieniach szarości – ciemne obszary to wolna przestrzeń, jasne to ściany, przeszkody, itp. Obszary pośrednie to miejsca, z których zebrano mniejsze ilości odczytów bądź też uzyskane pomiary są sprzeczne, co może mieć miejsce w przypadku występowania odbić od niektórych powierzchni i materiałów. W przypadku algorytmów działających z wykorzystaniem map zajętości obszarów konieczne jest przeprowadzenie operacji progowania na tak uzyskanej mapie (rys. 5.5).

## 5.3 Akwizycja obrazu z kamery dookólnej

W celu praktycznej weryfikacji jakości obrazu uzyskiwanego z kamery dookólnej robota został stworzony program realizujący perspektywiczny rzut obrazu. Program wykorzystuje biblioteki *OpenCV* oraz *IPP* [17, 13].

Rzutowanie perspektywiczne opiera się na przekształceniu, gdzie dla każdego punktu $(x', y')$ płaszczyzny rzutu wyznaczany jest odpowiadający punkt o współrzędnych $(x, y)$ obrazu wyjściowego. Liczby $(x, y)$ nie są liczbami cał-



Rysunek 5.4: Mapa laboratorium w postaci surowej (odcienie szarości)

Rysunek 5.5: Mapa laboratorium po progowaniu na poziomie 152/255

kowitymi – stąd możliwe są różne metody interpolacji w wyznaczaniu wartości piksela obrazu docelowego. Prostsza i szybszba metoda najbliższego sąsiedztwa polega na wstawieniu do obrazu docelowego piksela najbliższemu współrzednym $(x, y)$ w obrazie wyjściowym. Metoda interpolacji bilinearnej polega na uśrednieniu wartości pikseli w sąsiedztwie punktu $(x, y)$ proporcjonalnie do ich odległości od tego punktu – daje ona lepsze rezultaty (obraz jest bardziej gładki), jednak wymaga dłuższego czasu obliczeń.

Operacja rzutowania perspektywicznego na komputerze PC klasy tego, w który wyposażony jest robot - PIII, 850MHz przy stosowaniu zoptymalizowanych bibliotek IPP zajmuje na tyle mało czasu, że jest możliwe jej wykonywanie oraz dalsze proste przetwarzanie tak uzyskanego obrazu w czasie rzeczywistym (tab. 5.5).

| | OpenCV | IPP 5.0 |
|---|---|---|
| nearest-neigbor interpolation | 90ms | 33ms |
| bilinear interpolation | 86ms | 43ms |

Tabela 5.5: Porównanie czasów rzutowanie perspektywicznego przy korzystaniu z bibliotek OpenCV i IPP

Rysunek 5.6: Rzut perspektywiczny obrazu z kamery dookólnej

# Rozdział 6

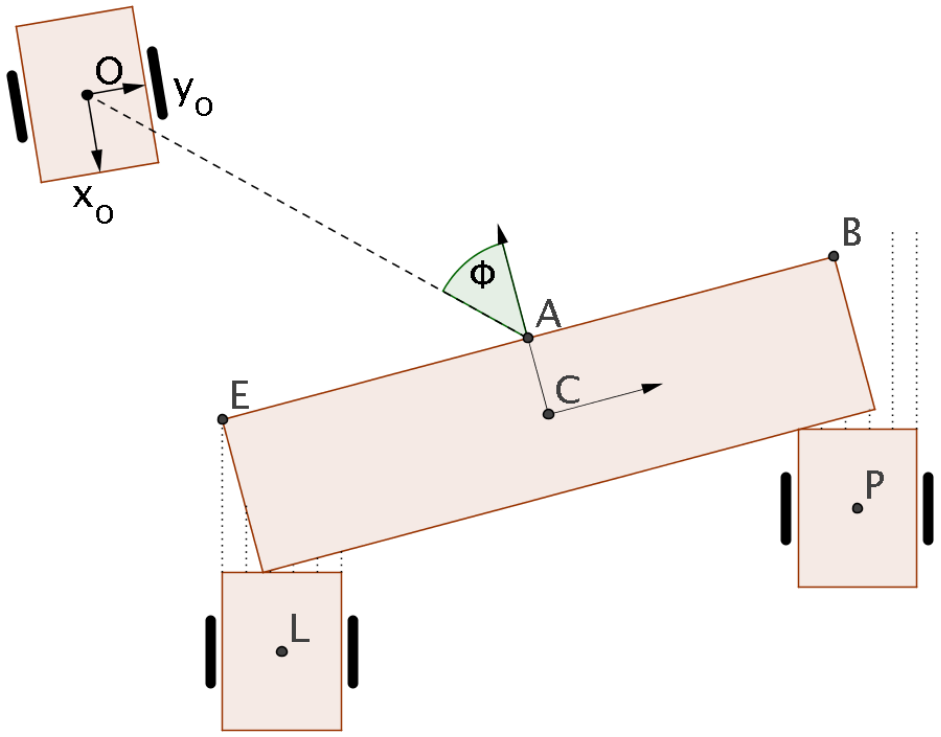
# Realizacja zadania wielorobotowego

Jako sposób na weryfikację działania oprogramowania sterującego robotem wybrałem zadanie pchania pudła, jako przykład gdzie wykonanie możliwe 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 [20, 24, 5]. Inne sposoby to np. sztywne uchwycenie obiektu przez roboty, czy oplecenie go liną [7].

Metoda, którą zdecydowałem się stosować jest rozwinięciem podejścia określanego jako *pusher–watcher* – gdzie zadanie jest realizowane przez grupę robotów z podziałem na role [10]. 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,




Rysunek 6.1: Schemat zadania pchania pudła

- układu jezdnego robotów,
- modułu komunikacji WLAN,
- systemu oprogramowania, który musi pozwalać na spójne sterowanie działaniem wielu robotów.

## 6.1 Opis zadania

Rozważane jest następujące zadanie: robot $O$ wyposażony w dalmierz laserowy określa odległość $(OA)$ i orientację $(\phi)$ obiektu. Roboty pchające ($L$ i $P$) wyposażone są w m.in. 6 czujników odległości dających odczyt w zakresie ok. $5 \ldots 50cm$ (rys. 6.1).

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

- utrzymanie położenia przy krawędzi pudła,

- pchanie z zadaną prędkością.

Pierwszy z nich jest realizowany 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 $\omega$.

Realizacja drugiego celu 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. 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.

## 6.2 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 $\phi$ 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 $\phi$ wyrażony w stopniach:

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

Zadanie można traktować jako układ regulacji, gdzie uchybem jest kąt $\phi$, 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 okres 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:

- $\omega_L, \omega_P$ – 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 iteracją algorytmu dyskretnego):

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

Prędkości obrotowe $w_L$ i $w_P$ 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:

- 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.

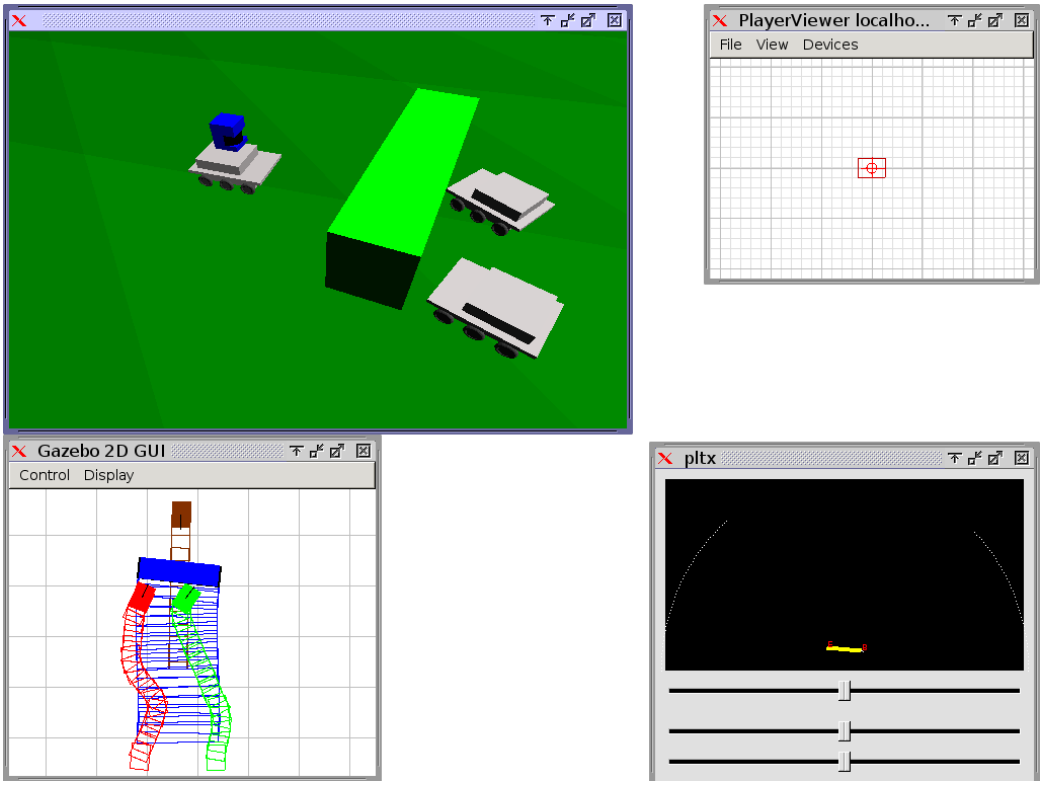
Komunikacja programu sterującego z robotami realizowana jest z użyciem klasy *PlayerMultiClient* biblioteki C++ z pakietu *Player/Stage*, która jest konieczna do zapewnienia działania pętli głównej programu jako przesłanie sterowań w reakcji na przetworzenie danych z czujników wszystkich robotów (tab. 6.1).

```
#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 6.1: Schemat programu sterującego wieloma robotami

## 6.3 Realizacja zadania na symulatorze

Zadanie zostało przygotowane z wykorzystaniem wcześniej opisywanego symulatora *Gazebo*. 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. 6.2).




Rysunek 6.2: Realizacja pchania pudła na symulatorze

## 6.4 Eksperymenty

W trakcie eksperymentów robot wodzący był nieruchomy, zaś zadaniem robotów pchających była współpraca mająca na celu minimalizowanie kąta $\phi$ pomiędzy prostopadłą do pudła poprowadzoną z jego środka – a półprostą poprowadzoną z tego punktu w kierunku środka robota wodzącego (rys. 6.1). Pchane pudło miało wymiary ok. $180 \times 20 \times 20$cm i wagę kilkunastu kilogramów (rys. 6.9, 6.10).

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. 6.4, 6.6, 6.8). 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.

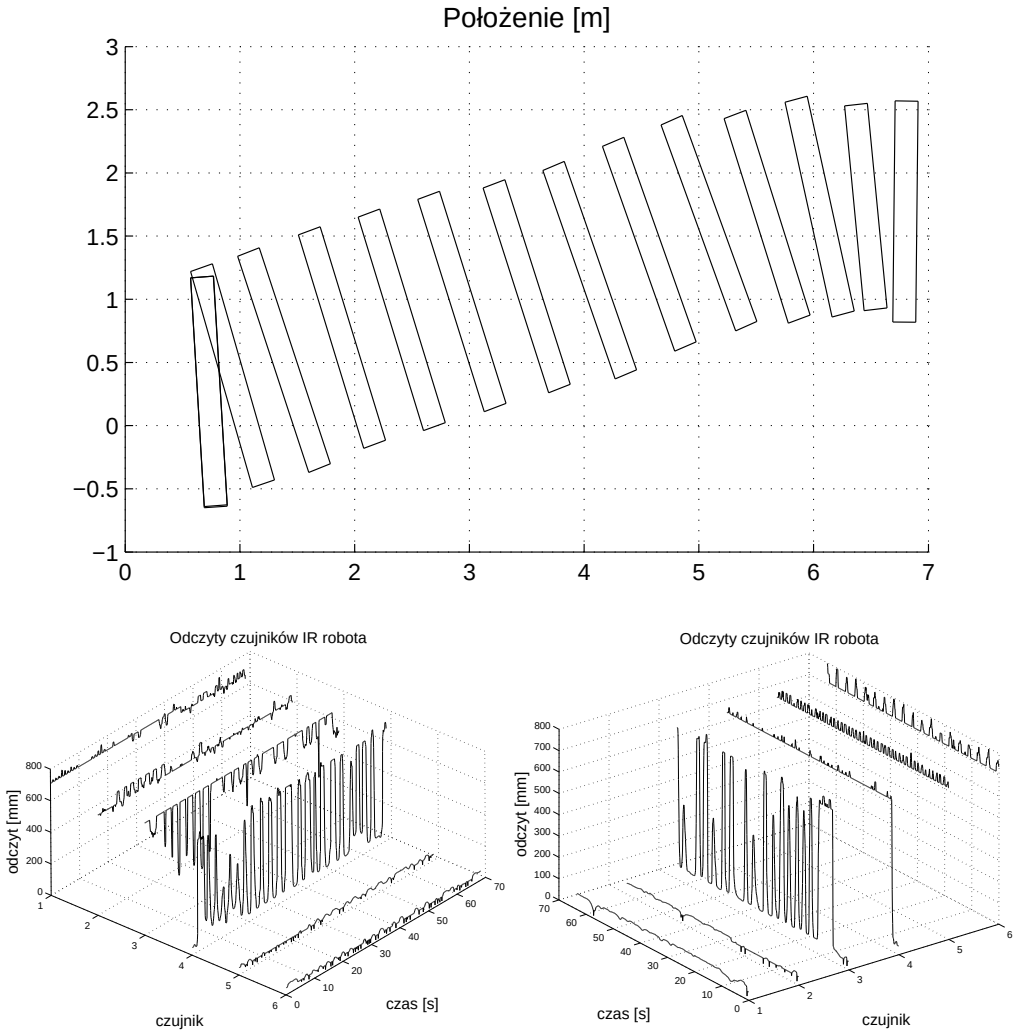
Małe wartości odczytów z czujników IR odpowiadają czujnikom zasłoniętym przez pudło, natomiast duże to odczyty z czujników, przed którymi znajdowała się wolna przestrzeń. Odczyty z czujników znajdujących się od strony środka pudła nie przekraczają kilku centymetrów, natomiast odczyty z czujników znajdujących 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 $\phi$ traktowanego w tym przypadku jako uchyb regulacji (rys. 6.3, 6.5, 6.7). W celu likwidacji stałej różnicy kąta $\phi$ konieczne okazało się wprowadzenie do równań stanowiących w istocie regulator proporcjonalny członu całkującego – czego konsekwencją jest także widoczne przeregulowanie przebiegu na wykresach.

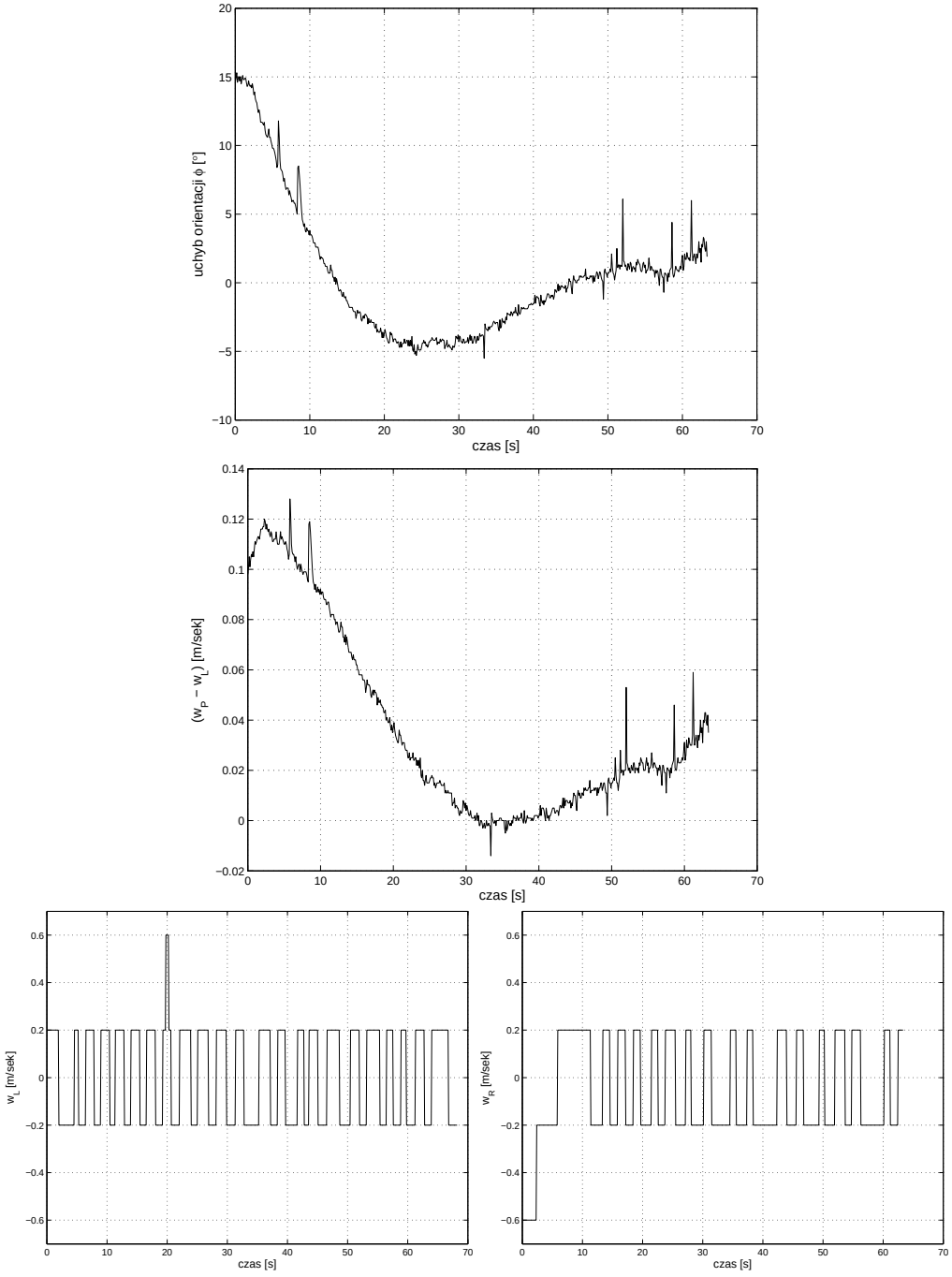
Wykresy prędkości obrotowej robota ($\omega_L$ i $\omega_R$) przedstawiają proces regulacji orientacji robota tak, aby krawędź pudła znajdowała się na środku jego zderzaka. Wartości prędkości oscylują pomiędzy $\pm 0.2$ [rad/sek], co odpowiada skrętom w prawo i lewo korygującym ustawienie robota względem 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.

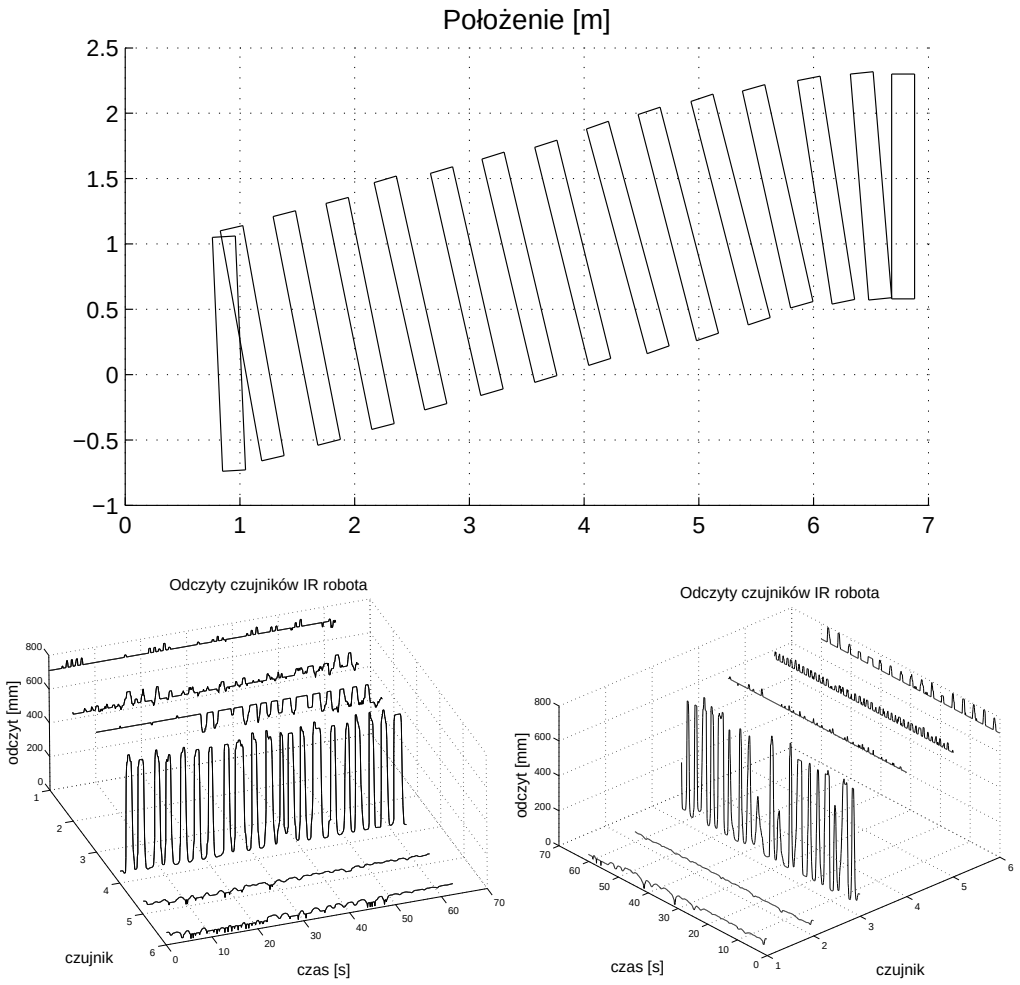
W trakcie realizacji zadania przez rzeczywiste roboty prędkości $w^i$ oraz $\omega^i$ uzyskiwane na podstawie odometrii z enkoderów umieszczonych na osiach kół obarczone były dużym błędem – co wynikało z faktu poślizgu robota w trakcie pchania ciężkiego pudła. Zadanie było wykonywane przez roboty praktycznie 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.



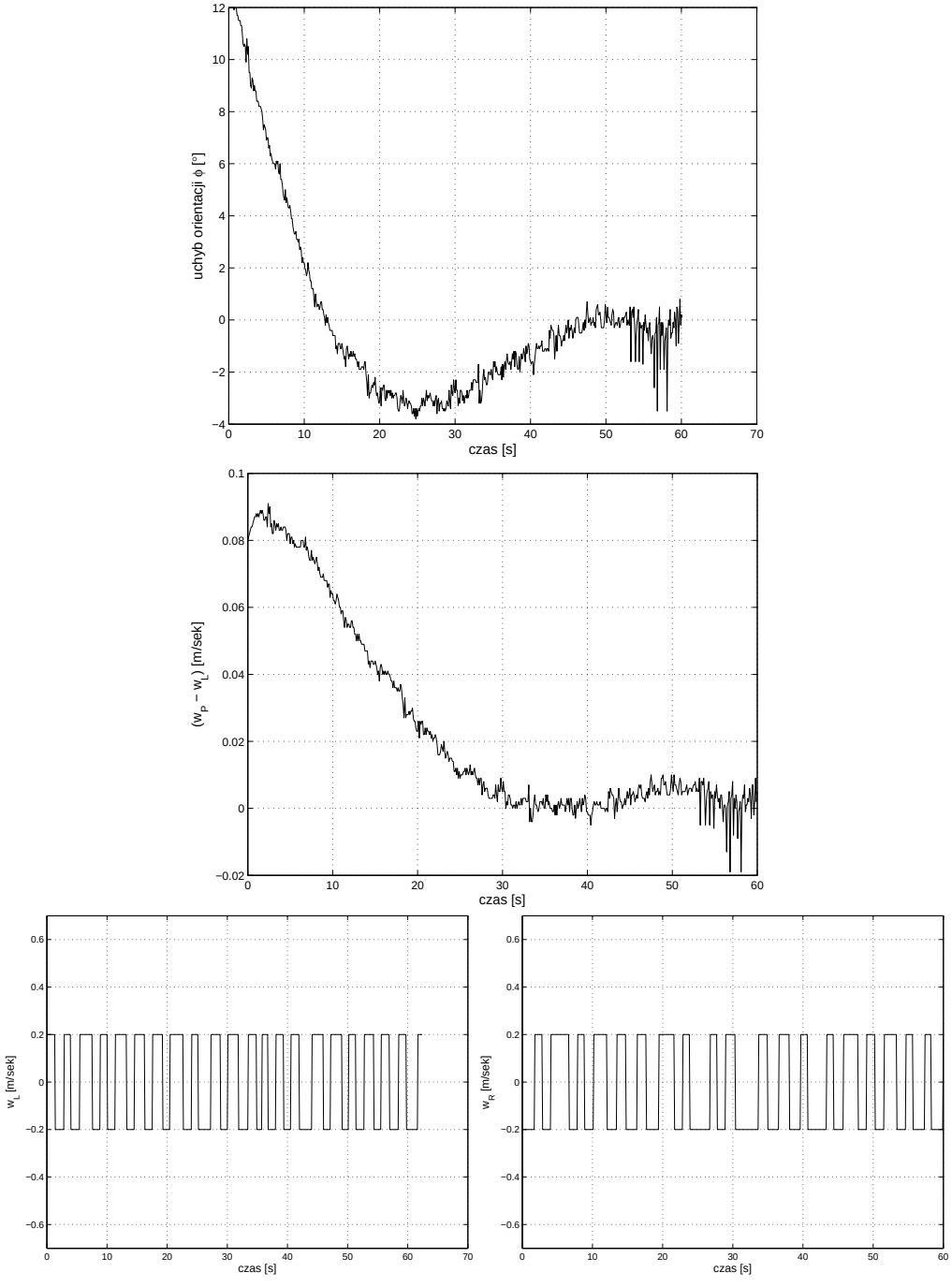

Rysunek 6.3: Wykresy położenia pudła oraz odczytów czujników IR robotów (przebieg 1)



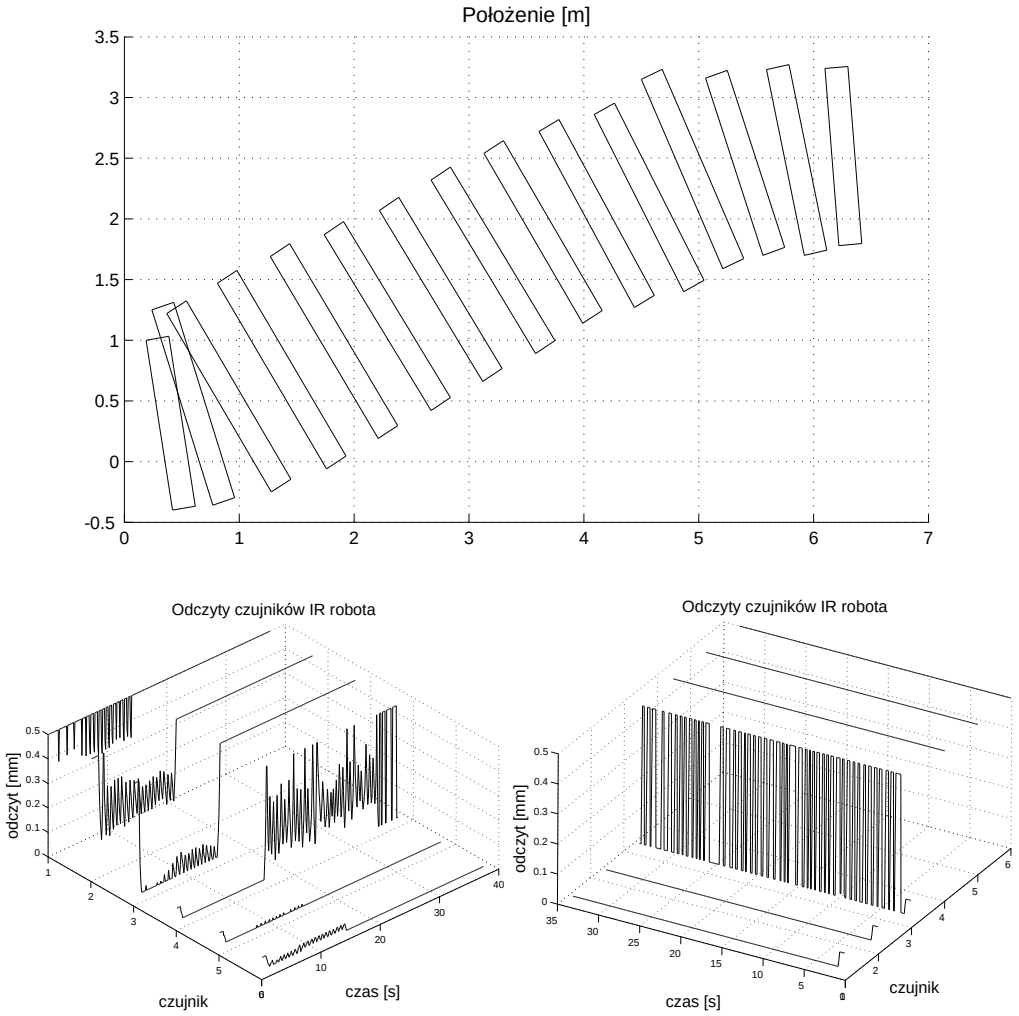

Rysunek 6.4: Wykres uchybu regulacji, różnicy zadawanych prędkości liniowych oraz prędkości obrotowych lewego i prawego robota pchającego (przebieg 1)



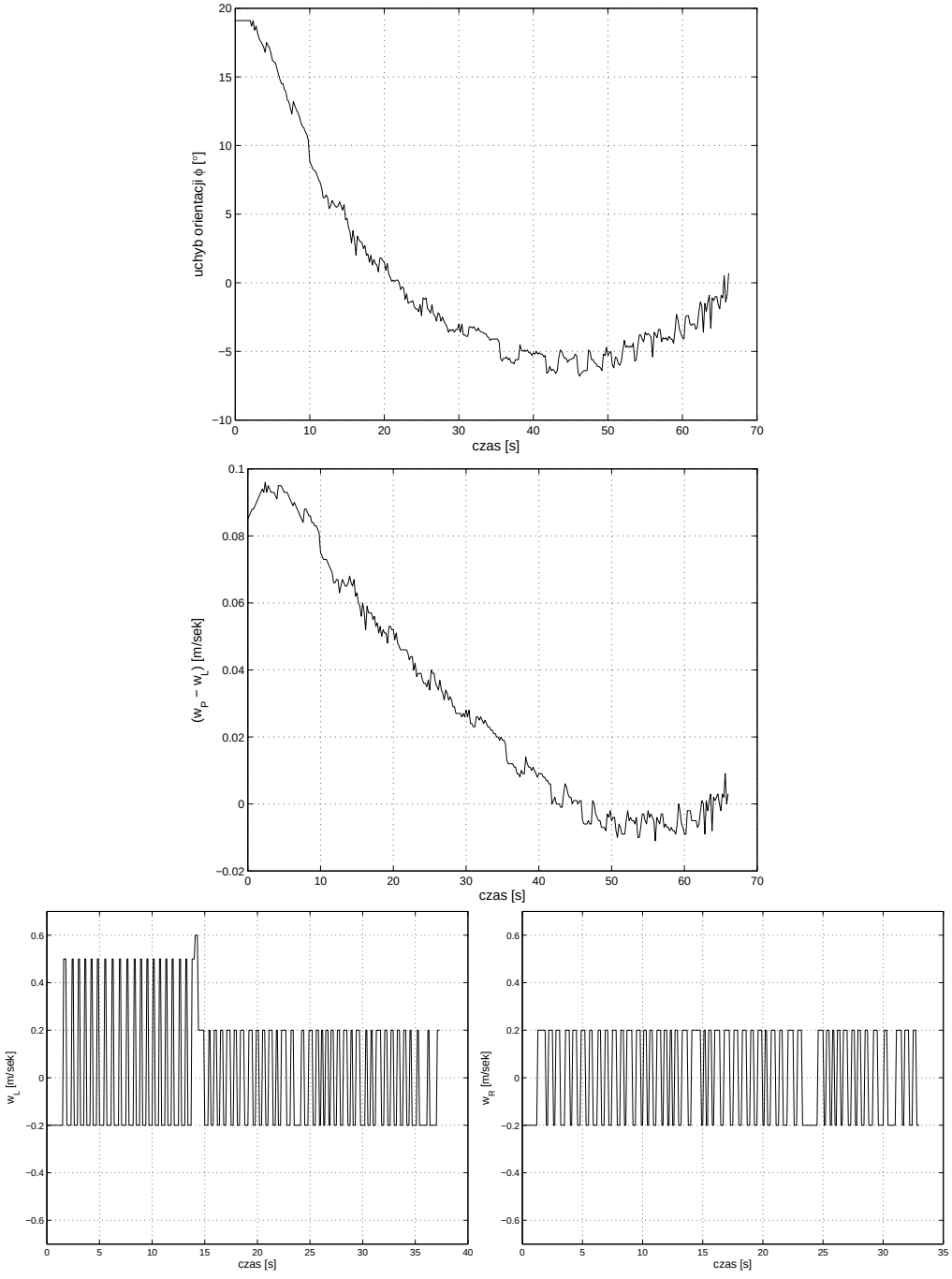

Rysunek 6.5: Wykresy położenia pudła oraz odczytów czujników IR robotów (przebieg 2)



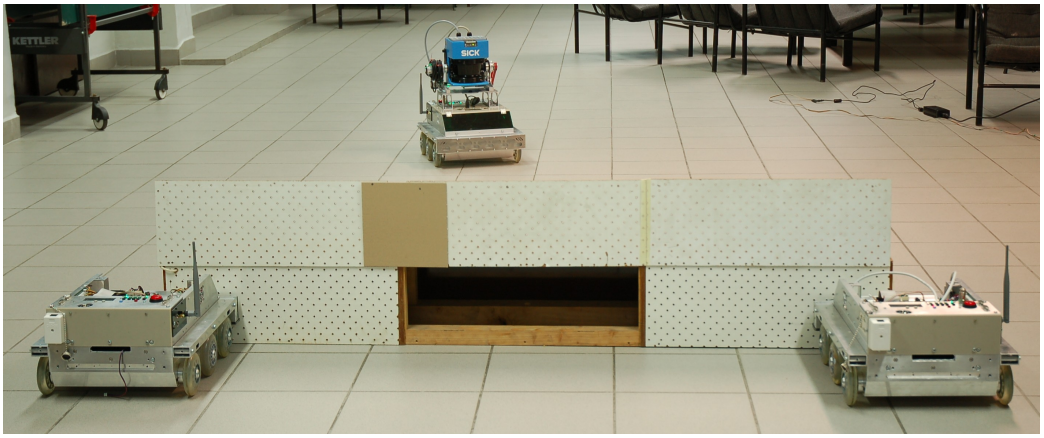

Rysunek 6.6: Wykres uchybu regulacji, różnicy zadawanych prędkości liniowych oraz prędkości obrotowych lewego i prawego robota pchającego (przebieg 2)



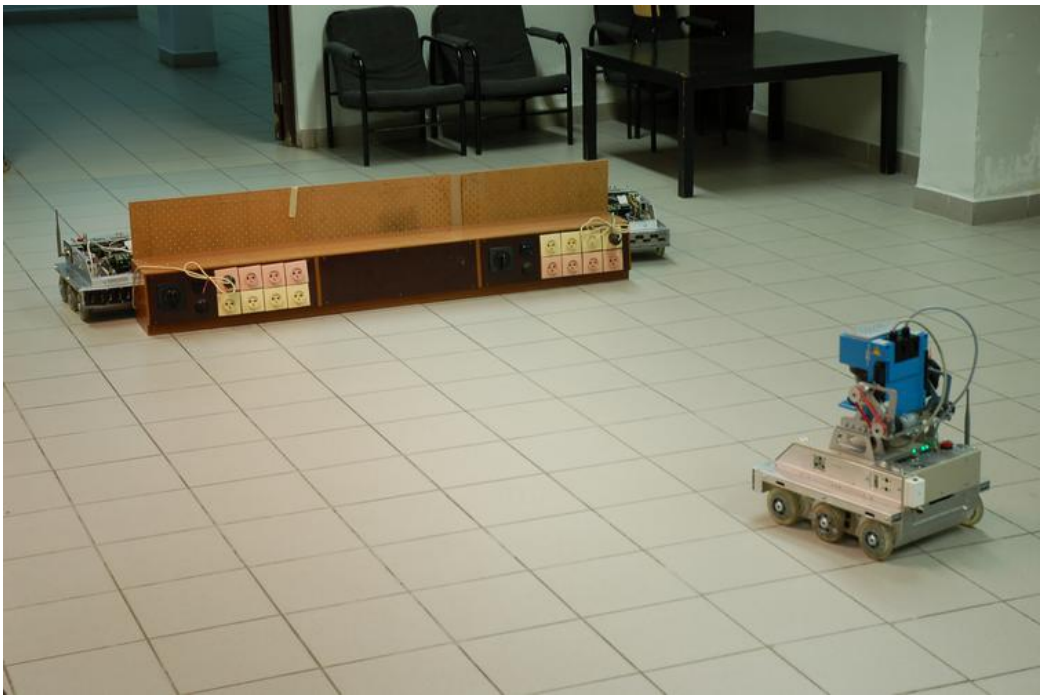

Rysunek 6.7: Wykresy położenia pudła oraz odczytów czujników IR robotów w trakcie pchania pudła (przebieg na podstawie symulacji)




Rysunek 6.8: Wykres uchybu regulacji, różnicy zadawanych prędkości liniowych oraz prędkości obrotowych lewego i prawego robota pchającego (przebieg na podstawie symulacji)




Rysunek 6.9: Realizacja pchania pudła – widok od strony robotów pchających



Rysunek 6.10: Realizacja pchania pudła – widok od strony robota wodzącego

# Rozdział 7

# Podsumowanie

Praca poświęcona jest tworzeniu i uruchomieniu środowiska do programowania robotów mobilnych. Na podstawie przeglądu istniejących struktur oprogramowania robotów mobilnych została sporządzona charakterystyka wymagań dla takiego systemu. Kierując się wyszczególnionymi w pracy kryteriami dokonany został wybór pakietu *Player/Stage* jako tej z dostępnych struktur, która spełniała określone w pracy wymagania. W konsekwencji został wybrany jako system operacyjny robota *LINUX*. Warto tutaj podkreślić, że dzięki wykorzystaniu modelu komunikacji klient–serwer programy aplikacyjne robota nie są związane z żadnym konkretnym systemem operacyjnym, a jedynym wymaganiem jest możliwość komunikacji z serwerem urządzeń *Player* z wykorzytaniem rodziny protokołów TCP/IP. Oprogramowanie klienta zostało uruchomione jako natywne aplikacje i przetestowane dodatkowo na systemach *FreeBSD* oraz *Windows*. Zwłaszcza to ostatnie spełnia ważną praktyczną funkcję z racji możliwości szybkiej i prostej instalacji na komputerach typu *laptop* na potrzeby prezentacji możliwości robotów.

W ramach pracy zostało stworzone oprogramowanie podstawowego modułu sterującego m.in. układem jezdnym robota – mikroprocesora zapewniającego komunikację z układami regulatorów silników. Ta część może być wykorzystana przy adaptowaniu robota do pracy z innymi strukturami oprogramowania czy pod kontrolą innego systemu operacyjnego.

Zestaw uruchomionych i przetestowanych elementów robota obejmuje:

- układ jezdny – sterowanie silnikami i dostęp do informacji z enkoderów optycznych umieszczonych na osiach kół robota,
- układ sterowania serwomechanizmami modelarskimi,
- układy akwizycji danych analogowych,

- oprogramowanie sterujące kamerami (możliwość zmiany parametrów takich jak zoom, ostrość, nasycenie barw),
- oprogramowanie akwizycji obrazu, w tym z kamery dookólnej,
- układ szybkiej komunikacji z dalmierzem laserowym,
- układ sterowania obrotnicą dalmierza laserowego wykorzystywaną do budowania map trójwymiarowych.

Większość z wymienionych modułów oraz ich oprogramowanie zostały zaadoptowane do pracy z wykorzystaniem serwera *Player/Stage*.

Przygotowane środowisko umożliwia badania z wykorzystaniem symulatora dwuwymiarowego (głównie do prac związanych z algorytmami planowania ścieżki oraz budowania map) oraz trójwymiarowego (do prac związanych z budowaniem map trójwymiarowych oraz oddziaływaniem na środowisko).

Całość przygotowanego systemu została zweryfikowana przez implementację zadania wymagającego współpracy wielu robotów. Zaproponowany został nowatorski algorytm transportu obiektów bazujący na przydzieleniu ról do robotów oraz charakteryzujący się dużą odpornością na występujące błędy i szumy pomiarowe oraz niedokładności modelu robota.

Przygotowane środowisko jest już obecnie używane przy realizacji prac dydaktycznych oraz naukowo badawczych IAiIS, co stanowi także pozytywną weryfikację omówionych w pracy decyzji oraz spełnienia przedstawionych wymagań. Czas potrzebny na uruchomienie środowiska dzięki wykorzystaniu wolnodostępnego oprogramowania *Open Source* – zarówno systemu operacyjnego z gotowymi sterownikami urządzeń, bibliotek jak i samego pakietu sterowania robotami był stosunkowo krótki. Decyzja o bazowaniu na szeroko stosowanym oprogramowaniu pozwoliła na łatwiejsze porównanie algorytmów nowo tworzonych w ramach prac dyplomowych z już istniejącymi – takimi jak unikanie kolizji czy budowanie map.

# Bibliografia


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

[2] Rodney A. Brooks. A robust layered control system for a mobile robot. IEEE Trans. on Robotics and Automation, 1986, wolumen 2, numer 1, s. 14–23.

[3] James Bruce, Tucker Balch, Manuela Veloso. Fast and inexpensive color image segmentation for interactive robots. In: Proceedings of IROS-2000. *Proceedings*, Japan, October, 2000.

[4] Stefano Carpin et al. High fidelity tools for rescue robotics: Results and perspectives. Robocup 2005 Symposium, 2005.

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

[6] Jonathan Corbet, Alessandro Rubini, Greg Kroah-Hartman. *Linux Device Drivers, Third Edition.* O'Reilly & Associates, Inc. February, 2005.

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

[8] *eCos (embedded Configurable operating system)*, http://ecos.sourceware.org/

[9] Future Technology Devices Intl. Ltd. *FT232BM USB USART (USB–SERIAL) I.C.*, 2005.

[10] Brian P. Gerkey, 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). *Proceedings*, 2002, s. 464–469.

[11] Hitec RCD USA, Inc. *General Servo FAQ's*.

[12] Steven Hunt. *LM628 Programming Guide*. National Semiconductor, January, 1999. Application Note 693.

[13] *Intel Integrated Performance Primitives for Intel Architecture. Reference Manual. Volume 2: Image and Video Processing.*, 2005.

[14] Jianqiang Jia, Weidong Cheng, Yugeng Xi. Design and implementation of an open autonomous mobile robot system. In: Proc. IEEE Int. Conf. Robotics and Automation, New Orleans, LA. *Proceedings*, 2004.

[15] M. Majchrowski. Algorytm unikania kolizji przez robota mobilnego bazujący na przeszukiwaniu przestrzeni prędkości. Praca magisterska, Politechnika Warszawska, 2006.

[16] National Semiconductor. *LM628/LM629 Precision Motion Controller.*, January, 2003.

[17] *Intel Open Source Computer Vision Library*, http://www.intel.com/technology/computing/opencv/index.htm

[18] *RTAI, Real-Time Application Interface*, http://www.rtai.org/

[19] *RTLinuxFree*, http://www.fsmlabs.com/rtlinuxfree.html

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

[21] SICK AG – Division Auto Ident, Germany. *Telegrams for Operating/Configuring the LMS 2xx Laser Measurement Systems.*, 2003.

[22] Russell Smith. Open dynamics engine v0.5 user guide. http://www.ode.org/, February, 2006.

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

[24] Z-D. Wang et al. A pushing leader based decentralized control method for cooperative object transportation. In: Proceedings of the IEEE/RSJ International Conference on Intelligent Robot Systems (IROS2004). *Proceedings*, 2004.

[25] Cezary Zieliński. The MRROC++ system. First Workshop on Robot Motion and Control, RoMoCo 99, 1999, s. 147–152.

[26] Andrew Howard. *Simple mapping utilities (pmap)*, http://robotics.usc.edu/∼ahoward/pmap/

[27] Carle Côté, François Michaud. *Mobile and Autonomous Robotics Integration Environment*, http://marie.sourceforge.net/

[28] Herman Bruyninckx. *Orocos Open Robot Control Software*, http://www.orocos.org/