Politechnika Wrocławska

Instytut Informatyki Stosowanej

Praca Magisterska

# ZASTOSOWANIE WYBRANYCH METOD SZTUCZNEJ INTELIGENCJI DO STEROWANIA ROBOTA MOBILNEGO

BUDOWA SYSTEMU WIZYJNEGO DLA RZECZYWISTEGO ROBOTA

Bartosz Szurgot

**Promotor:** prof. dr hab. inż. Halina Kwaśnicka

**Ocena:**

Wrocław 2007

**Streszczenie**

W niniejszej pracy został omówiony, zbudowany w celach badawczych, system robotyczny. Oprócz części programowej (sterownik) zbudowano rzeczywistego robota gąsienicowego o nazwie „TIER", na którym przeprowadzono badania praktyczne zastosowanego podejścia.

Znaczną część pracy stanowi opis systemu wizyjnego, analizującego dane przychodzące z jedynego czujnika w jaki jest wyposażony robot – czarnobiałej kamery monowizyjnej. Wśród stosowanych metod analizy znajdują się między innymi: filtry, sieci neuronowe i algorytmy segmentacji grafiki rastrowej. Zaproponowano także nietypowe podejście do pobierania informacji o otoczeniu, na podstawie pojedynczych klatek obrazu.

Dużo uwagi poświęcono możliwościom dalszego rozwoju systemu. Autor proponuje liczne zmiany i usprawnienia, które po wprowadzeniu mogą znacząco wpłynąć na możliwości realizowanych przez robota zadań.

**Abstract**

In this paper robotic system, build for research purposes, has been described. Beside the software part (the controller) real tracked robot named „TIER" has been build and proposed approach has been practically tested on it.

A lot of place has been given for a vision system analysing data coming from the only sensor robot has – black-and-white mono-vision video-camera. Some of methods used in this task: filters, neural networks and algorithms for raster images segmentation. Also an unusual approach for collecting information about environment surrounding robot has been proposed, making use of single video-camera images only.

High priority has been given to possibilities of further development of system. Author proposes many changes and rationalisations that may have big influence on possible tasks for robot, if introduced.

# Spis treści

**Wstęp** . . . . . 1

**Rozdział 1. Sztuczna inteligencja i robotyka mobilna** . . . . . 4

1.1. Roboty i ich zastosowania . . . . . 4
1.2. Sztuczna inteligencja w robotyce mobilnej . . . . . 6
1.3. Zakres niniejszej pracy . . . . . 9

**Rozdział 2. Zastosowane podejście** . . . . . 11

2.1. Wprowadzenie do projektu . . . . . 11
2.2. System wizyjny . . . . . 13
2.2.1. Dlaczego system wizyjny? . . . . . 14
2.2.2. Normalizacja . . . . . 14
2.2.3. Segmentacja „hybrydowa” . . . . . 15
2.2.4. Segmentacja „grafowa” . . . . . 22
2.2.5. Porównanie metod segmentacji . . . . . 25
2.2.6. Wektoryzacja . . . . . 27
2.2.7. Odczytywanie głębi . . . . . 29
2.2.8. Reprezentacja przestrzeni . . . . . 34
2.3. System sterujący . . . . . 35
2.3.1. Komunikacja i sterowanie . . . . . 35
2.3.2. Ruch i wykrywanie kolizji . . . . . 36

**Rozdział 3. Eksperymenty weryfikujące poprawność i efektywność zaproponowanego rozwiązania** . . . . . 40

3.1. Implementacja systemu . . . . . 40
3.2. Przeprowadzone eksperymenty . . . . . 41
Jazda na wprost w statycznym środowisku . . . . . 41
Jazda na wprost w środowisku zmiennym . . . . . 41
Omijanie przeszkód statycznych . . . . . 42
Omijanie przeszkód w ruchu . . . . . 42
Manewrowanie w ciasnym środowisku . . . . . 42
Wpływ oświetlenia . . . . . 44

**Rozdział 4. Podsumowanie** . . . . . 45

4.1. Wyniki prac . . . . . 45

4.2. Dalsze prace . . . . . 47
Ruch w „ciasnym” środowisku . . . . . 47
Dobór czujników . . . . . 48
Szumy i komunikacja międzywarstwowa . . . . . 48
Autonomia robota . . . . . 49
Sprzętowy system wizyjny . . . . . 49

**Bibliografia** . . . . . 50

**Dodatek A. Budowa robota** . . . . . 53

A.1. Stawiane cele . . . . . 53
A.2. Wizja całościowa . . . . . 54
A.3. Schemat komunikacji . . . . . 54
A.4. Mechanika . . . . . 55
A.5. Elektronika . . . . . 56
A.5.1. Odsprzęganie zasilania . . . . . 56
A.5.2. Układy nawrotne . . . . . 57
A.5.3. Mikrokontroler sterujący . . . . . 59
A.6. Kamera wideo . . . . . 63
A.7. Końcowe przemyślenia . . . . . 65

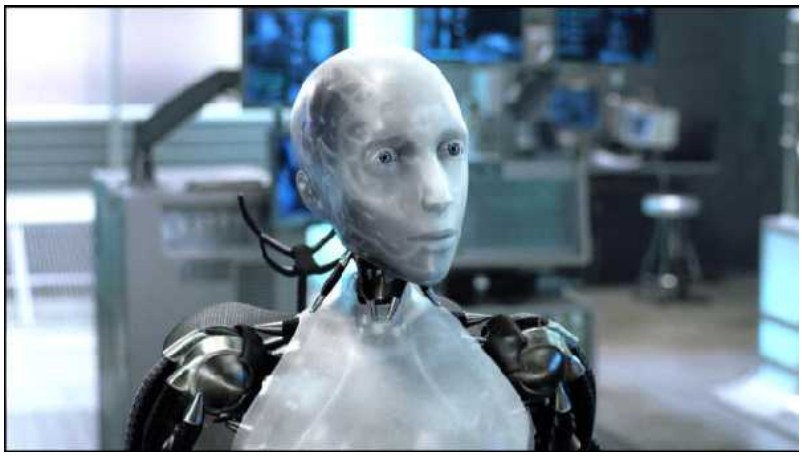
**Dodatek B. Podziękowania** . . . . . 69

**Spis rysunków** . . . . . 70

**Spis tablic** . . . . . 73

# Wstęp

Większości osób niezwiązanych na codzień z nauką roboty kojarzą się głównie z filmami science-fiction – czymś niespotykanym na codzień (rys. 0.1). Pogląd ten nie jest prawdą. Choć roboty znane z filmów oraz możliwości, jakie posiadają są przeważnie wytworem wyobraźni reżyserów, to jednak mamy okazję spotkać się z nimi pod postacią projektów naukowych (np: projekty realizowane przez studentów w ramach kół naukowych [38][39]), zabawek dla dzieci [36] lub też z efektami ich pracy: patrząc na drogę na przejeżdżające pojazdy.



Rys. 0.1. Kadr z popularnego filmu „I, robot" (zaczerpnięty z [34]).

Kiedy w przemyśle pojawiły się pierwsze roboty, zrewolucjonizowały one działanie linii produkcyjnych. Powierzano im coraz więcej żmudnych bądź niebezpiecznych prac, odciążając tym samym ludzi, którzy mogli w tym czasie zająć się bardziej wymagającymi zadaniami. Od tamtej pory minęło już wiele lat – roboty na liniach montażowych potrafią samodzielnie malować karoserie pojazdów, nawiercać precyzyjnie otwory w elementach składowych, zależnie od ich położenia na taśmie, czy też spawać metalowe komponenty.

Choć w dziedzinie robotyki osiągnięto już wiele, [51][49] możliwości dzisiejszych robotów są nadal ograniczone – są one przeważnie przystosowane do pracy w zamkniętych, ustalonych środowiskach, wykonując tylko pewien szereg z góry ustalonych czynności. Kolejnym, naturalnym, krokiem w tej „ewolucji" jest przystosowanie ich do auto-

nomicznego działania w zmiennym środowisku. Zadanie to, choć wykonywane przez nas na co dzień, okazuje się być niezwykle skomplikowanym do opisania tradycyjnymi metodami algorytmicznymi. Zwykłe mieszkanie jest dla robotów skrajnie niebezpiecznym miejscem – otoczenie się ciągle zmienia, domownicy poruszają się relatywnie szybko z miejsca na miejsce, wiele osób posiada też w domach zwierzęta, lecz i to nadal jest dopiero namiastka wszystkich czynników wpływających na trudność interakcji robota ze światem zewnętrznym [2].

Zbierając razem wszystkie wymienione czynniki, okazuje się, iż otrzymujemy praktycznie nieskończoną ilość różnorakich sytuacji (o „stanach", w klasycznym ujęciu, niewspominając), w jakich może zaleźć się robot. Lecz i to nie jest jeszcze koniec, gdyż tak naprawdę rozwiązanie powyższych problemów to dopiero nauczenie maszyny metod(y) „przetrwania" w nieprzyjaznym dla niej środowisku – aby fakt istnienia w nim robota miał sens, musi on być w stanie wykonywać użyteczne dla ludzi czynności.

Pojawiają się tu kolejne trudności, zawiązane z interakcją – jeżeli mielibyśmy programować robota za każdym razem kiedy chcielibyśmy zrobić kawę, czy rozwiesić pranie, prawdopodobnie lepiej byłoby zrobić tę czynność samemu.

Reasumując więc – aby nasz nowy „pomocnik" rzeczywiście służył nam pomocą, musi być on:

1. w pełni mobilny – by móc się zawsze dostać w miejsce, gdzie będzie potrzebny w danej chwili.
2. inteligentny – musi potrafić zachować się „odpowiednio" (lub przynajmniej bezpiecznie dla otoczenia i siebie samego (zgodnie z kolejnością wymieniania)) [40] w zupełnie nowej dla niego sytuacji.
3. łatwo sterowalny – naturalnym dla człowieka sposobem komunikacji jest dialog. Jeżeli maszyna ma być intuicyjna i prosta w obsłudze, musi posiadać możliwość komunikacji głosowej.
4. elastyczny – jego budowa nie może (znacząco) ograniczać możliwości jego zastosowania w wyręczaniu ludzi w różnych zadaniach.

Poziom technologiczny, jakim dysponujemy, ma różny wpływ na różne z podanych czynników. Szczególnie intensywny rozwój jednak miał, w ciągu ostatnich 20-30 lat, miejsce w informatyce – z relatywnie prostych komputerów, o niewielkiej mocy obliczeniowej w przeciągu jednego pokolenia przeszliśmy na potężne superkomputery, mogące wykonywać miliardy operacji w ciągu sekundy. Nawet przeciętny komputer domowy potrafi wykonywać zadania o jakich jeszcze niedawno można było jedynie marzyć, zaś konstrukcje z przed 20 lat (głównie adaptowane do roli mikrokontrolerów jedno układowych) znajdują obecnie z powodzeniem zastosowanie we wszelakim sprzęcie AGD itp (pralki, lodówki, kuchenki mikrofalowe, wierze Hi-fi...) [62].

Właśnie taki stan rzeczy umożliwia praktyczne zastosowanie, wymagających znacznych nakładów obliczeniowych, licznych metod sztucznej inteligencji (ang. *AI – Artificial Intelligence*), takich jak choćby sztuczne sieci neuronowe (ang. *ANN - Artificial Neural Networks*), czy też algorytmy genetyczne (ang. *GA – Genetic Algorithms*). Daje to robotom nowe możliwości analizy danych przychodzących, za pośrednictwem czujników, owocując większymi możliwościami inteligentnej interakcji z otaczającym środowiskiem, oraz rozwiązywania powierzonych im problemów (zleconych zadań).

W niniejszej pracy skupimy się głównie na możliwościach związanych z inteligentnym przetwarzaniem informacji docierających ze środowiska zewnętrznego oraz ich praktycznym stosowaniem. Dla potrzeb przeprowadzanych eksperymentów

powstał prosty robot (nazwany „TIER” od ang. *Tracked Impulse Electric Robot* – napędzany elektrycznie robot sterowany impulsowo) z napędem gąsienicowym, sterowany drogą radiową przez zewnętrzny komputer (więcej informacji na jego temat można znaleźć w dod. A). Choć w praktycznych zastosowaniach lepszym (bardziej uniwersalnym) rozwiązaniem zdaje się być zbudowanie w pełni autonomicznego robota, to przyjęte podejście upraszcza znacząco konstrukcję (położony jest większy nacisk na oprogramowanie), oraz daje możliwość implementacji rozproszonego sterownika, nie powodując dodatkowego obciążenia baterii robota.

Ponieważ zdecydowano się na pobieranie informacji ze środowiska za pomocą systemu wizyjnego znaczna część tejże pracy jest poświęcona przetwarzaniu obrazów. Pośród omawianych podejść znajdą się zarówno metody klasycznej grafiki komputerowej, jak i typowe metody sztucznej inteligencji, takie jak sieci neuronowe. Podczas prac nad systemem wizyjnym zaproponowano także nietypowe podejście do pobierania informacji z obrazu $2D$. Zostanie ono szerzej omówione w kolejnych rozdziałach.

Duży nacisk został także położony na praktyczność omawianych rozwiązań, ponieważ całość systemu jest zaimplementowana i sprawdzona na rzeczywistym robocie, nie zaś na symulatorze komputerowym. Posiadając sprawdzoną i działającą platformę sprzętową można (po pewnych modyfikacjach) zastosować ją do użytecznych celów. Przykładem rozwiązań zbliżonych funkcjonalnie do prezentowanego są roboty-odkurzacze sprzątające mieszkania i supermarkety [25][26].

W kolejnych rozdziałach niniejszej pracy omówione są poszczególne etapy realizacji postawionych celów – począwszy od zaprezentowania kilku przykładowych podejść (rozdz. 1), jakie spotkano w literaturze. Następnie przedstawiona jest własna propozycja realizacji systemu (rozdz. 2) na jaką się zdecydowano. Kolejnym krokiem jest jej implementacja wraz z przeprowadzaniem testów oraz analiza uzyskanych w ten sposób wyników (rozdz. 3). W ostatnim rozdziale pracy (rozdz. 4) całość została podsumowana z głównym naciskiem położonym na kierunki dalszych prac.

Na koniec autor pragnie zaznaczyć, iż całość kodu aplikacji oraz dokumentacji powstała wykorzystując jedynie wolne, ogólnie dostępne oprogramowanie (ang. *open-source*) zarówno jako fragmenty kodu, jak i w roli narzędzi (programistycznych). W szczególności oznacza to nie korzystanie z Windows’a ani żadnych innych produktów Microsoft’u.

# Rozdział 1

# Sztuczna inteligencja i robotyka mobilna

W rozdziale tym omówione są podstawowe zagadnienia związane zarówno z robotyką mobilną jak i sztuczną inteligencją. Przedstawiono również pokrótce jakie zagadnienia będą poruszane w ramach niniejszej pracy.

## 1.1. Roboty i ich zastosowania

„Robotyka" jako taka to bardzo szerokie pojęcie – jest to przede wszystkim nauka interdyscyplinarna: składają się na nią między innymi: mechanika, automatyka i elektronika, informatyka.

Budując robota należy mieć na uwadze jego zastosowanie, które to prawdopodobnie zdeterminuje jego typ i kształt. Typów robotów jest wiele – zgrubnie można je przypisać do pewnych klas:

1. lądowe (rys. 1.1)
   - gąsienicowe
   - kołowe
   - kroczące
2. wodne (rys. 1.2)
   - nawodne
   - podwodne
3. latające (rys. 1.3)
   - sterowce
   - poduszkowce
   - poruszające się przy zerowym ciążeniu
   - samoloty i śmigłowce

Można by długo pisać o ich wadach, zaletach i zastosowaniach [53]. W warunkach domowych jednak najprościej jest zająć się robotami lądowymi (tak też stało się w przypadku tejże pracy – więcej informacji na ten temat znajduje się w dod. A).

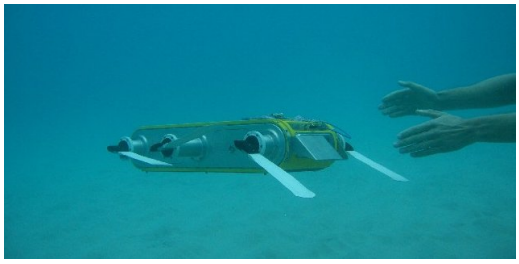
Roboty lądowe znajdują zastosowania w wielu dziedzinach życia, często służąc pomocą człowiekowi w docieraniu do trudnodostępnych lub też niebezpiecznych (np: skażonych) miejsc.

(a)


(b)

Rys. 1.1. Przykładowe roboty lądowe (zaczerpnięte kolejno z [30] oraz [33]).

(a)



(b)

Rys. 1.2. Przykładowe roboty wodne (zaczerpnięte z [31] oraz [32]).

(a)

(b)

Rys. 1.3. Przykładowe roboty latające (zaczerpnięte z [28] oraz [29]).

Przykładem może tu być projekt robota badającego wnętrze rur o zadanej średnicy [55]. Choć jest to projekt z końca XX wieku (1999 r.) widać, iż już w tamtym okresie pojawiały się konstrukcje praktyczne, gotowe do zastosowania. Roboty penetrujące orurowania mogą służyć do kontroli ich stanu od wewnątrz. W przypadku rur zakopanych pod ziemią nie ma innego (prostego) sposobu aby sprawdzać je pod kątem pęknięć. Dodatkowo wyposażenie mobota w odpowiednie manipulatory pozwala na wykonywanie czynności konserwacyjnych takich jak czyszczenie czy też lokalizacja uszkodzonych miejsc [55].

Doskonałym przykładem zastosowania robota mobilnego jako pomocy dla człowieka jest „Gizmo" – robot stworzony dla Szpitala Dziecięcego w Bostonie [35]. Pracuje on od 3 lat (stan na dzień powstania artykułu źródłowego – 2005 r.) rozwożąc dokumentacje i leki po budynku szpitalnym, samodzielnie korzystając z wind, omijając ludzi dzięki czujnikom podczerwieni lub też prosząc ich delikatnym damskim głosem o przepuszczenie w przypadku braku możliwości ominięcia.

Oprócz korzyści wynikających z odciążenia pracowników placówki z mozolnej pracy przenoszenia dokumentów medycznych robot okazał się atrakcją dla pacjentów – jest bardzo lubiany przez dzieci, dla których stanowi niebywałą atrakcję.

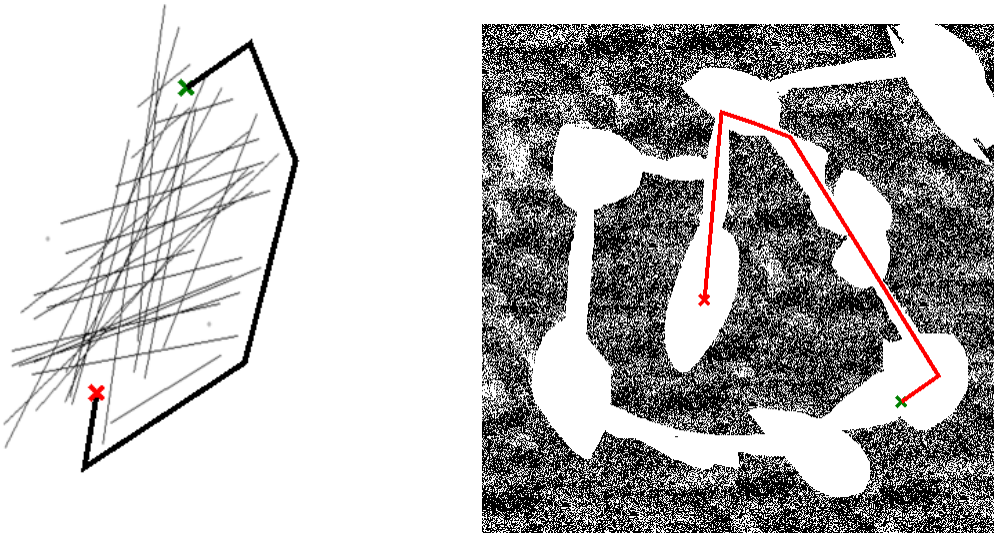
Roboty są również powszechnie stosowane do sprzątania pomieszczeń. Ciekawym projektem na skalę przemysłową jest ACRO (ang. *ACRO – Automation of Floor Cleaning*) [27]. Zbudowany w ramach niego robot jest w pełni autonomicznym czyścicielem podłóg przewidzianym do zastosowania w dużych halach takich jak supermarkety, stacje kolejowe czy lotniska. Posiada on także prosty interface pozwalający an obsługę przez niewykwalifikowany personel – jest to ważna cecha w praktycznych zastosowaniach.

Choć robot ten jest przewidziany dla dużych powierzchni i jest zbyt kosztowny dla przeciętnego obywatela, nie oznacza to, iż nie są tworzone rozwiązania dedykowane dla użytkowników indywidualnych. Sprzątające roboty do użytku domowego można kupić w sklepach internetowych już za kilkaset złotych (stan na chwilę pisania niniejszej pracy) [25][26].

## 1.2. Sztuczna inteligencja w robotyce mobilnej

Sztuczna inteligencja znajduje wiele zastosowań w robotyce mobilnej – począwszy od planowania ruchu i sterowania na projektowaniu samych robotów skończywszy [37][43][52].

Dużą popularnością cieszą się algorytmy genetyczne w zadaniach znajdowaniu optymalnej trasy na mapie łączącej dwa punkty [47][52]. Przykładowe trasy znalezione za pomocą algorytmów genetycznych dla kilku planszy przedstawiono na rys. 1.4 [47]. Choć algorytmy te nie optymalizują wyłącznie lokalnie (istnieje pewne prawdopodobieństwo znalezienia optimum globalnego) i przeszukują dużą przestrzeń, są one bardzo kosztowne obliczeniowo, przez co trudne do zrealizowania praktycznego w aplikacjach czasu rzeczywistego, jakimi bez wątpienia są moboty. Dużą zaletą ich jest zaś podejście „any-time" wynikające bezpośrednio z zasady ich działania – w każdej chwili można pobrać wynik jaki do tej pory udało się uzyskać i założyć, iż to właśnie będzie naszym bieżącym rozwiązaniem.



(a) (b)

Rys. 1.4. Przykładowe wyniki działania algorytmów genetycznych do wyszukiwania tras na mapie (zaczerpnięte z implementacji koncepcji zaprezentowanej w [47]).

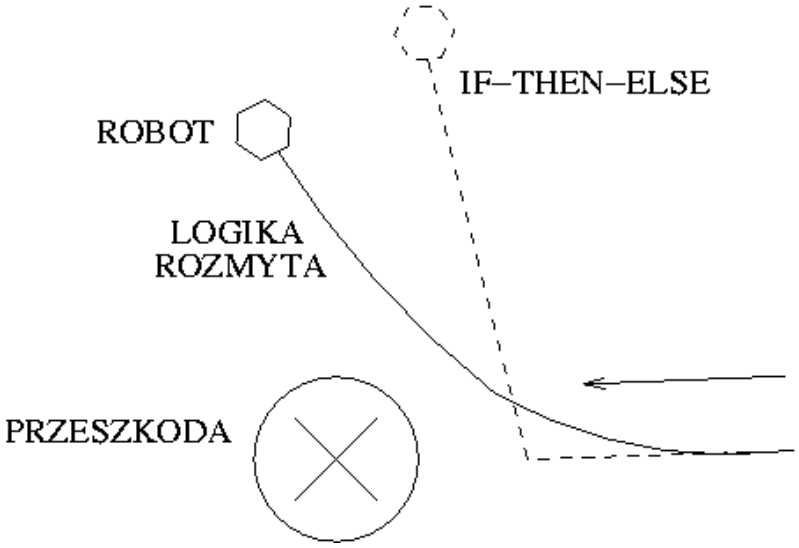
Zaprezentowane w [47] podejście jest o tyle ciekawe, iż reprezentacja przestrzeni jest ciągła – niestety czas wymagany dla obliczenia bardziej skomplikowanych tras jest zbyt długi, przez co algorytm nadaje się wyłącznie do planowania „offline".

Do sterowania robotem mobilnym bardzo powszechnie stosuje się sztuczne sieci neuronowe [50] – przeważnie perceptron wielowarstwowy (zwyczajowo nazywany siecią propagacji wstecznej – ang. *BPN – Back Propagation Network*). Aby wykorzystać możliwie w pełni zalety sieci neuronowych często używa się ich do całościowego sterowania robotem – na wejścia sieci są podawane odczyty z czujników, zaś jej wyjścia są połączone z motoryką robota, wprawiając go w ruch [41][50]. Różnica pomiędzy takim sterownikiem a klasycznym systemem informatycznym opartym o model warstwowy jest zasadnicza. W przypadku podejścia warstwowego poszczególne etapy analizy są skupione w pewnych elementach systemu – każda z warstw zamyka w sobie szczegóły pewnej funkcjonalności, udostępniając jedynie pewien interface dostępowy doń. Choć taki system jest przejrzysty, z dobrze wydzielonym podziałem zadań, gubiona jest w nim pewna „globalność". Problem podzielony na podproblemy wymaga pewnych uogólnień i przetwarzania (często stratnego) w celu uzyskania wybiórczych informacji, na podstawie których można podjąć we w miarę prosty sposób decyzję. Sieć neuronowa posiada pełną informację, zaszytą w postaci wag i sygnałów, w całej swojej strukturze – przetwarzanie odbywa się więc globalnie (odpowiadałoby to modelowi oprogramowania gdzie każda z „warstw" komunikuje się (obustronnie) z każdą inną).

Z drugiej jednak strony, uczenie sieci jest dość problematyczne: jest bardzo czasochłonne, podatne na minima lokalne oraz niedeterministyczne. Praktycznie aż do momentu testowania nauczonej już sieci, na przypadkach spoza zbioru uczącego nie ma pewności, czy sieć nauczyła się tego, czego od niej oczekiwaliśmy, czy też proces nauczania wyuczył sieć podziału według innego kryterium niż planowane przez autora.

Ciekawą techniką do sterowania ruchem robotów jest stosowanie logiki rozmytej (ang. *FL – Fuzzy Logic*) umożliwiającej proste przejście od „skokowych" reguł typu if-then-else do sterowania płynnego (ciągłego). Logika rozmyta łączy w sobie przejrzystość podobną do systemu regułowego oraz „gładkość" reakcji systemu na otoczenie – lepiej oddaje rzeczywistość niż przetwarzanie dyskretne, nie zwiększając przy tym znacznie poziomu skomplikowania.

Powszechnie stosuje się tę technikę do generowania „odpowiedzi" mobota (ang. *mobot – MObile roBOT*) na środowisko [54][56]. Przykładem może tu być reguła mówiąca o omijaniu przeszkody: im bliżej do przeszkody tym mocniej robot powinien skręcić aby ją ominąć zachowując przy tym płynność ruchu. Sytuację taką ilustruje rys. 1.5.




Rys. 1.5. Robot omija przeszkodę stosując logikę rozmytą (im bliżej przeszkody, tym mocniej skręca by ją ominąć) w porównaniu z prostym systemem regułowym.

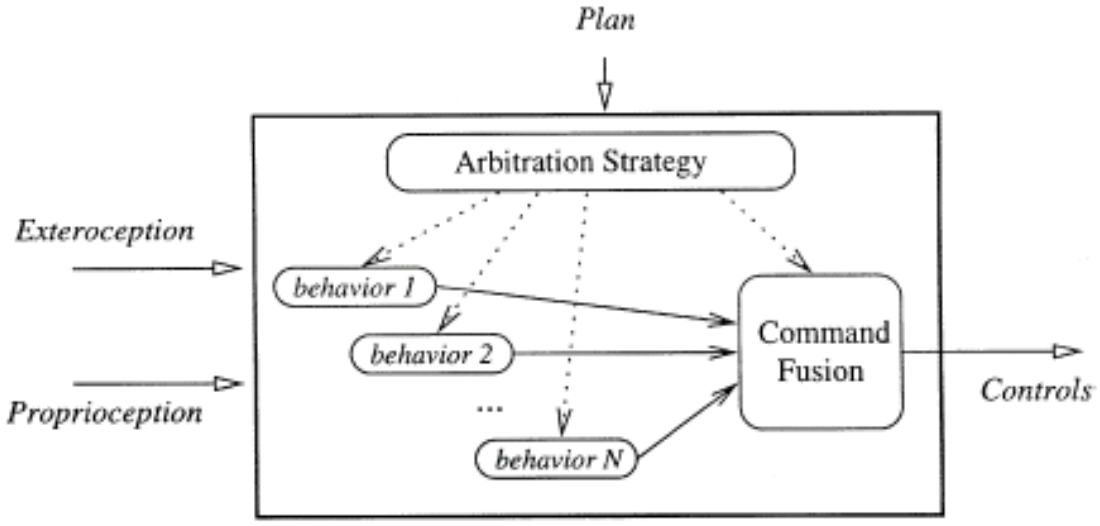
Logika rozmyta sporadycznie występuje „samodzielnie" – przeważnie jest ona łączona z innymi metodami, stanowiąc fragment większego systemu przetwarzającego.

Dobrym przykładem na zastosowanie logiki rozmytej w połączeniu z inną techniką jest dekompozycja behawioralna. Polega ona na rozłożeniu (dekompozycji) złożonego zadania sterowania posiadającego wiele podcelów (np: poszukiwanie obiektu o zadanym kształcie, jednocześnie omijając przeszkody w terenie i jadąc wzdłuż ścian).

Aby uzyskać przejrzysty system biorący pod uwagę wszystkie wymienione (wymagane) czynniki należy potraktować je jako osobne cele, stworzyć moduły je realizujące na podstawie danych wejściowych (np: jeżeli nie trzeba omijać przeszkody, nie ma sensu żeby moduł unikania kolizji podejmował jakąkolwiek akcję) oraz, po wykonaniu stosownych obliczeń, przeprowadzić fuzję decyzji podjętych lokalnie przez każdy z modułów z osobna. Tak uzyskaną decyzję „wypadkową" traktujemy jako odpowiedź globalną w systemie i realizujemy za pomocą motoryki robota.

Przedstawiono obrazowo ideę tegoż podejścia na rys. 1.6.

Widać wyraźnie, iż logika rozmyta jest tu dobrze pasującym elementem do reprezentacji informacji wewnątrz systemu oraz łączenia częściowych odpowiedzi z




Rys. 1.6. Ideowy rysunek pokazujący zasadę działania dekompozycji behawioralnej (zaczerpnięty z [56]).

modułów w jedną całość.

W zastosowaniach praktycznych rzadko zdarza się stosować pojedynczą metodę – przeważnie pełnowartościowy system powstaje poprzez zastosowanie kilku różnych podejść, tak aby możliwie uwypuklić silne strony każdej z nich. Jest też faktem powszechnie znanym, iż nie ma sensu „wymuszać" na sztucznej inteligencji radzenia sobie z problemami doskonale znanymi i łatwo rozwiązywalnymi za pomocą klasycznych algorytmów. Podejście takie jedynie wydłuża czas pracy systemu oraz niepotrzebnie go komplikuje nie dając w zamian żadnych konkretnych korzyści.

Metody sztucznej inteligencji, jako kosztowne obliczeniowo, powinny być wykorzystywane tam, gdzie inne podejścia zawodzą lub są trudne do praktycznego zrealizowania.

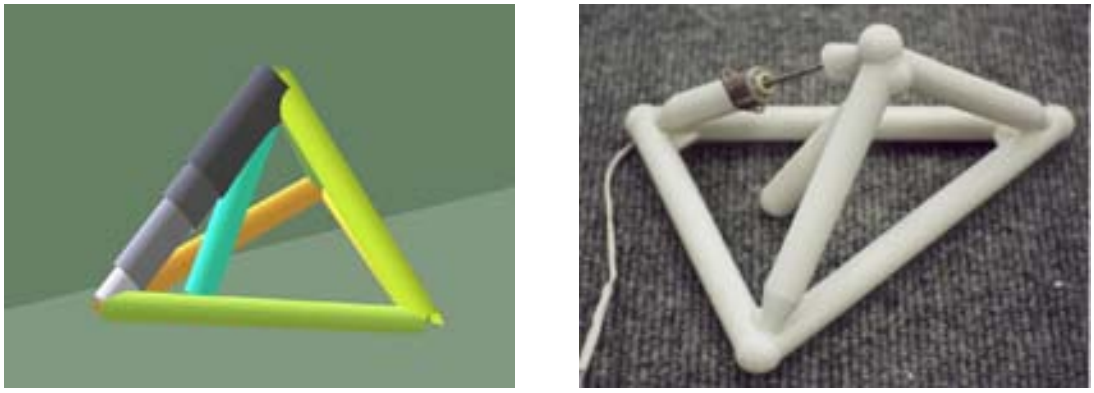
Warto również wspomieć o możliwości stosowania metod sztucznej inteligencji do projektowania. Jako przykład można przytoczyć tu projekt „GOLEM" (ang. *GOLEM – Genetically Organized Lifelike Electro Mechanics*), w którym wykorzystuje się algorytmy ewolucyjne do tworzenia robotów wewnątrz wirtualnego świata. Ich zadaniem jest poruszanie się przed siebie [37].

Ciekawą cechą projektu jest fakt, iż najlepsze z osobników są realizowane w fizycznym świecie (w sposób półautomatyczny) i tam testowane. Przykładowego osobnika wirtualnego oraz jego fizycznego odpowiednika przedstawia rys. 1.7

Wybrane wyniki można obejrzeć na stronie projektu [37]. Istnieje także możliwość pobrania używanego w procesie ewolucji wirtualnych mobotów programu.

## 1.3. Zakres niniejszej pracy

Z powodu ograniczonych możliwości czasowych i finansowych w niniejszej pracy pokryto jedynie pojedyncze zagadnienia związane z robotyką mobilną. Główną tematyką poruszaną jest analiza danych z rzeczywistej (czarnobiałej) kamery wizyjnej w



(a) (b)

Rys. 1.7. Przykładowy robot zaprojektowany przez algorytmy ewolucyjne (a) oraz jego fizyczna realizacja (b).

celu odczytania informacji o przestrzeni dookoła jadącego robota. Informacje te są wykorzystane przez dość prosty system decyzyjny, który jest tylko namiastką pokazującą możliwości związane z planowaniem ruchu w terenie otwierając pole dalszym pracom.

Końcowy system robotyczny, jaki udało się stworzyć jest na tyle kompletny aby zapewnić bezkolizyjne poruszanie w terenie oraz, po drobnej rozbudowie, realizacje funkcji porównywalnych z robotami sprzątającymi łatwo dostępnymi dla przeciętnie zasobnego obywatela [25][26].

# Rozdział 2

# Zastosowane podejście

W bieżącym rozdziale opisane zostały metody jakie są używane w zaimplementowanym systemie robotycznym. Główną częścią jest system wizyjny z którego pobierane są informacje o otaczającym robota świecie. Druga część rozdziału poświęcona została logice decydującej o reakcji robota na otoczenie.

## 2.1. Wprowadzenie do projektu

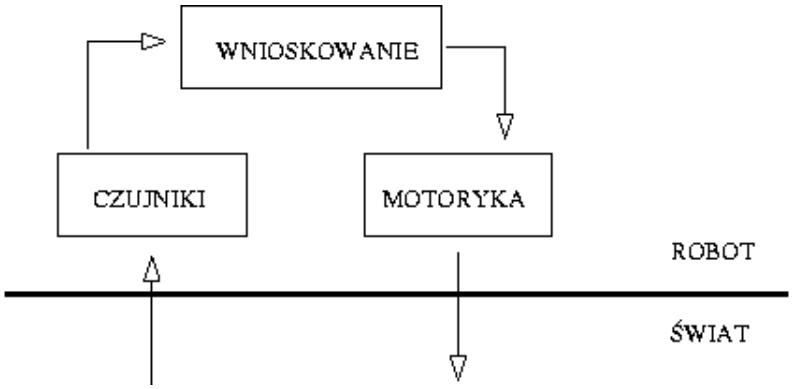
Bieżący rozdział jest w całości poświęcony systemowi, jaki został zaproponowany do sterowania robotem TIER[1]. Znajduje się tu szczegółowy opis każdego ważnego etapu analizy i podejmowania decyzji, nim jednak zostaną zaprezentowane szczegóły omówione będzie podejście w sposób całościowy.

Jedynym czujnikiem, za pomocą którego robot rejestruje informacje o otaczającym go świecie jest czarnobiała kamera monowizyjna, z kolei oddziaływanie na środowisko może odbywać się wyłącznie poprzez ruch realizowany przez silniki elektryczne. Takie postrzeganie tegoż zagadnienia pozwala sprowadzić całość do postaci autonomicznego elementu (bytu) oddziałującego ze środowiskiem. Ilustruje takie podejście w sposób ogólny rys. 2.1 zaś na rys. 2.2 widać z jakich konkretnie elementów składa się omawiany system.

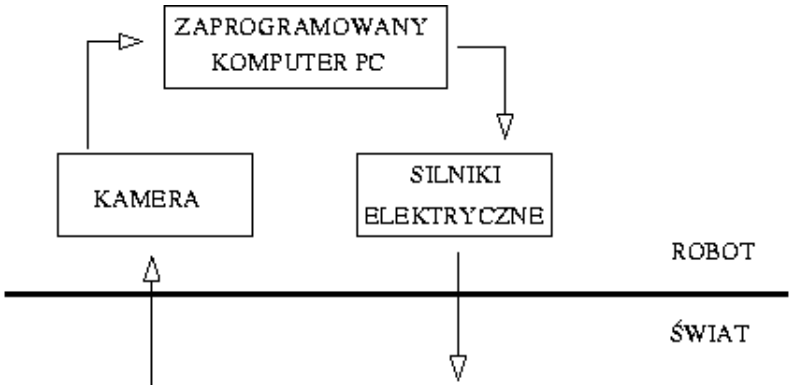
Widać tu wyraźnie podział systemu na pewne fragmenty (podsystemy): blok odpowiadający za pobieranie danych z czujników („blok wejściowy”), blok przetwarzający uzyskane informacje (główna część systemu) oraz blok sterujący mechaniką robota („blok wyjściowy”).

Powołując się na rys. 2.2: sposób analizowania danych pobranych z kamery jest opisany w sekcji podroz. 2.2 zaś winoskowanie z tak uzyskanych danych zostało omówione w podrozdziale podroz. 2.3.2. Sama realizacja podsystemu sterowania fizycznym robotem zostanie tu pominięta ze względu na jej bardzo sprzętowy (niskopoziomowy) charakter[2]. Przepływ informacji w systemie jako całości przedstawia rys. 2.3.

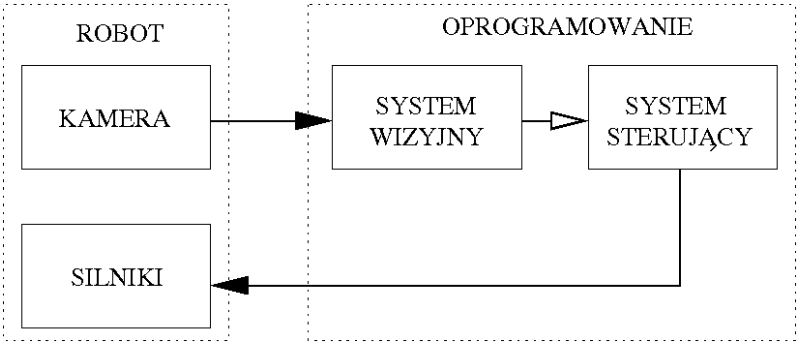
1. Więcej informacji odnośnie budowy mechanicznej oraz elektronicznej robota można znaleźć w dod. A.
2. Więcej informacji można znaleźć w dod. A.




Rys. 2.1. Ogólny model oddziaływania robota ze światem.




Rys. 2.2. Model oddziaływania robota ze światem zastosowany w prezentowanym rozwiązaniu.

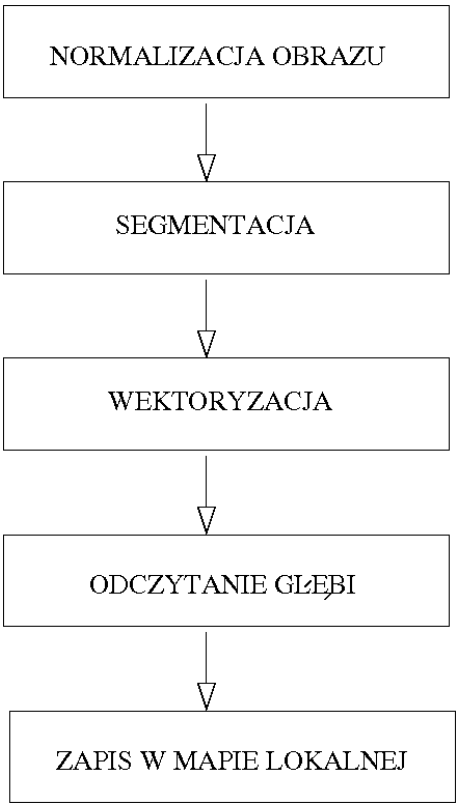



Rys. 2.3. Przepływ danych w systemie jako całości.

Ponieważ robot jest wyposażony tylko[3] w pojedynczą kamerę wideo analiza danych wejściowych sprowadza się do przetwarzania obrazów (oraz pobierania z nich „ukrytych" informacji). Analizę taką można przeprowadzić na wiele różnych sposobów: zależnie od typu obrazu wejściowego, informacji jakie chcemy z niego wydobyć/uwypuklić, etc… W przypadku sterowania robotem mobilnym[4] tak, aby unikał on przeszkód na swojej drodze, kluczowe jest pobranie z otoczenia informacji gdzie takowe przeszkody się znajdują.

## 2.2. System wizyjny

Tematyką tego podrozdziału jest proponowany system wizyjny oparty na kamerze monowizyjnej. Mając na uwadze stawiane cele oraz dostępne środki zaproponowano przepływ informacji, jak przedstawiono na rys. 2.4. W kolejnych podrozdziałach opisano poszczególne etapy analizy oraz zilustrowano ich efekty konkretnymi przykładami.




Rys. 2.4. Przepływ danych w systemie wizyjnym jako całości.

Na wstępie, w podroz. 2.2.2 jest omówiona normalizacja obrazu, jaka ma miejsce aby możliwie ujednolicić dane wejściowe z czujnika. Znaczną część pracy poświęcono problematyce dzielenia obrazu na fragmenty (obszary mające cechę wspólną: podobne kolorystycznie, o podobnym wzorze, itp…) – segmentacji. W podsekcjach podroz. 2.2.3 oraz podroz. 2.2.4 zaprezentowano dwa alternatywne podejścia do tego problemu jakie zostały zaimplementowane i sprawdzone na potrzeby systemu. W podroz. 2.2.5 zostaną

3. Określenie „tylko" nie jest tu w pełni na miejscu, gdyż pobierany obraz w rozdzielczości 640x480 przy 8 bitowej palecie kolorów daje 78643200 różnych stanów wejściowych!
4. Czasem w literaturze spotyka się także skrótowe określenie robotów mobilnych jako „mobotów".

one porównane i na podstawie tego wybrane będzie podejście lepiej spełniające stawiane wymagania. Mając już wybraną metodę segmentacji obrazu w następnych podsekcjach przedstawione zostały kolejne etapy przetwarzania danych wejściowych czyli: wektoryzacja (podroz. 2.2.6), odczytywanie głębi z obrazu (podroz. 2.2.7) oraz sposób, w jaki ostatecznie jest reprezentowane otoczenie robota po transformacjach (podroz. 2.2.8).

### 2.2.1. Dlaczego system wizyjny?

Człowiek za pośrednictwem swoich zmysłów otrzymuje informacje o otaczającym go świecie – do dyspozycji mamy: słuch, węch, dotyk, smak i wzrok. Różne zmysły pełnią różne funkcje poznawcze i niosą ze sobą różną ilość informacji, jednak zdecydowanie dominuje tu postrzeganie wizyjne[5]. Za jego pośrednictwem otrzymujemy większość docierających do mózgu bodźców.

Obraz wideo daje maszynie (podobnie jak człowiekowi) dostęp do ogromu informacji, którą trzeba umiejętnie wykorzystać. Pierwszym napotkanym problemem jest ilość danych potrzebna do analizy. Zbyt mała rozdzielczość obrazu wejściowego nie pozwoli na rozpoznanie pewnych obiektów, z kolei nadmiernie szczegółowy obraz w wysokiej rozdzielczości spowoduje znaczące wydłużenie się czasu potrzebnego na jego przetworzenie[6].

Jeżeli do tego będziemy wykorzystywać algorytmy analizy o ponad liniowej złożoności ze względu na liczbę pikseli, czas analizy wzrośnie tak gwałtownie, iż przeanalizowany obraz może się okazać bezużyteczny, bo rzeczywistość jaką reprezentuje, uległa zbyt dużej zmianie i uzyskane informacje są nieadekwatne do faktycznego stanu otoczenia (w chwili obecnej).

Mamy tu więc do czynienia ze skomplikowanym zadaniem analizy komputerowej, wymagającej dość ścisłego przestrzegania norm czasowych. Odpowiedź systemu, aby była poprawna, musi być nie tylko prawidłowa merytorycznie (proces przetwarzania musi dawać poprawne rezultaty) ale i udzielona w zadanej „szczelinie czasowej" (nie możemy przekraczać pewnego założonego czasu obliczeń, żeby zachować spójność z rzeczywistością i ciągłość analizy). Mamy tu więc do czynienia z systemem miękkiego czasu rzeczywistego (ang. *soft-realtime*) [48].

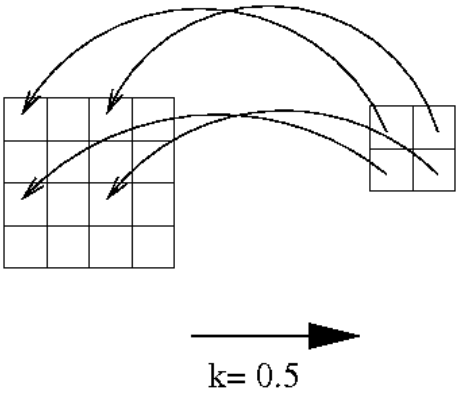
### 2.2.2. Normalizacja

Zespół operacji nazwany tu wspólnie „normalizacją" (wstępne przetwarzanie – ang. *preprocessing*) ma zapewnić możliwie dobre podstawowe parametry obrazu, takie jak ilość kolorów, czy jasność. Przykładowo, dla doboru wielu parametrów stosowanych algorytmów kluczowa jest rozdzielczość obrazu wejściowego; prawie zawsze też zakłada się iż obraz jest „odpowiedniej" jasności, czy też ostrości [60]. Ponieważ prezentowany system ma działać w czasie rzeczywistym oraz jest dopiero podstawą dla dalszych (potencjalnie bardziej kosztownych obliczeniowo) operacji, musi być wydajny.

Aby skrócić czas pełnego cyklu przetwarzania w systemie oraz zapewnić stabilne dane wejściowe, pierwszą operacją wykonywaną na obrazie jest jego skalowanie celem zmniejszenia liczności punktów do pewnej, ustalonej z góry liczby (rozdzielczości). Do skalowania wykorzystywany jest obliczeniowo najprostszy z możliwych algorytmów.

5. Konkretniej – stereowizyjne.
6. Warto w tym miejscu zauważyć, iż 2-krotne zwiększenie rozdzielczości spowoduje 4-krotne zwiększenie powierzchni (liczby punktów na obrazie).

Iteruje on po kolejnych punktach obrazu wyjściowego (kolory punktów nie są wtedy jeszcze ustalone) dla każdego z nich wyliczając pozycję pojedynczego piksela z obrazu wejściowego. Pozycja ta jest wyliczana na podstawie skali – dzięki niej wiadomo ile punktów obrazu wejściowego odpowiada powierzchni jednego piksela wyjściowego. Nazwijmy ten współczynnik $w_x$ dla osi $OX$ oraz $w_y$ dla osi $OY$. Mając tę wiedzę, można przyjąć, iż punktowi $(0;0)$ obrazu wyjściowego odpowiada punkt $(0;0)$ obrazu wejściowego. Przesuwając się kolejno o $w_x$ po osi $OX$ oraz $w_y$ po osi $OY$ wyliczamy pozycje punktów do pobrania koloru z obrazu wejściowego dla odpowiednich punktów obrazu wyjściowego. Przykładowo dla punktu $(1;2)$ obrazu wyjściowego odpowiadający punkt obrazu wejściowego ma współrzędne $(w_x, 2 * w_y)$. Kolor danego punktu wyjściowego jest kopią koloru wyliczonego punktu wejściowego. Przykładowe skalowanie dla obrazu o rozmiarze $4x4$ punkty do obrazu $2x2$ punkty ilustruje rys. 2.5.




Rys. 2.5. Przykład skalowania obrazu dla skali $k = 0.5$.

Takie podejście, mimo swej prostoty (i szybkości) powoduje od razu poprawienie ostrości[7] oraz daje dobry czas przetwarzania.

W przypadku obrazu z kamery w dobrych warunkach oświetleniowych większość operacji (oprócz skalowania) nie jest tak naprawdę konieczna (ilustruje to rys. 2.6(a) oraz rys. 2.6(b)) – przydają się one dopiero przy słabym/silnym oświetleniu, kiedy obraz jest niedoświetlony (zbyt ciemny) lub „przepalony" i posiada niewielką ilość kolorów (rys. 2.7(a) oraz rys. 2.7(b)[8]) [60].

Warto zaznaczyć w tym miejscu, iż rozciągając histogram, od razu poprawiamy jasność (zwiększamy odległości pomiędzy kolorami) – miejsca ciemne zaczynają wtedy ujawniać swoje szczegóły.

### 2.2.3. Segmentacja „hybrydowa"

Proponowany algorytm składa się z sekwencji kroków do wykonania na obrazie wejściowym, aby uzyskać obraz wyjściowy z zaznaczonymi segmentami[9].

7. Dzieje się tak, ponieważ w obrazach z rzeczywistych kamer mamy do czynienia z nieostrymi krawędziami (obszary przeważnie nachodzą na siebie na odległość kilku pikseli) – jeżeli teraz wybierzemy z takich obszarów „co n-ty" punkt mamy bardzo dużą szansę trafienia, kiedy jeden piksel jest w jednym obszarze, drugi zaś w drugim co z kolei powoduje usunięcie „płynnego" przejścia pomiędzy obszarami – gradienty kolorów są więc większe niż na obrazie wejściowym.
8. rys. 2.7(b) powstał poprzez silne ściemnienie programowe rys. 2.6(a).
9. Wewnętrznie za pomocą odpowiednich znaczników, dla celów wizualizacyjnych zaś za pomocą kolorów.

(a) (b)

Rys. 2.6. Obraz przy dobrych warunkach oświetleniowych: (a) obraz wejściowy; (b) obraz po przetworzeniu (normalizacji).

(a) (b)

Rys. 2.7. Obraz przy złych warunkach oświetleniowych (zbyt ciemny): (a) obraz wejściowy; (b) obraz po przetworzeniu (normalizacji).

Dobrany eksperymentalnie ciąg składa się z następujących operacji: [60]

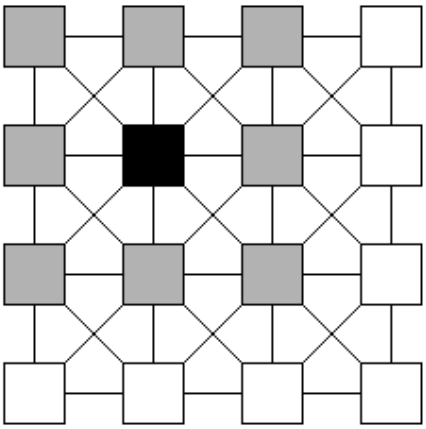
1. Wykrywanie krawędzi – sieć CNN
2. Końcowe przetwarzanie (ang. *postprocessing*) krawędzi
   - binaryzacja obrazu
   - operacja „denoise"
   - operacja „connector"
   - domknięcie obrazu
   - obróbka krawędzi obrazu
3. Zaznaczanie segmentów (obiektów)

Jednym z najważniejszych kroków prezentowanego podejścia do segmentacji jest detekcja krawędzi na obrazie, na podstawie których określana jest później przynależność danego punktu do konkretnego segmentu.

Do detekcji krawędzi zdecydowano się zastosować sieć neuronową komórkową (ang. *Cellular Neural Network*). Mózg ludzki doskonale radzi sobie z przetwarzaniem obrazów – jest to jedna z podstawowych operacji realizowanych przez niego. Wybór wydaje się więc naturalnym w tym przypadku. Dodatkowo przemawiają za tym wyborem efekty uzyskane przez innych w dziedzinie przetwarzania grafiki za pomocą sieci neuronowych [50].

Budowa sieci neuronowej komórkowej jest bardzo zbliżona do klasycznych filtrów, znanych w grafice komputerowej – każdy neuron odpowiadający pojedynczemu pikselowi jest połączony w każdym ze swoich sąsiadów w zadanym promieniu $r$ (rys. 2.8). Wagi zapisujemy w postaci macierzy o wymiarach $(2 * r + 1) \times (2 * r + 1)$. Wyróżniamy osobno macierz połączeń z wejściami sąsiednich neuronów („macierz sterowania") oraz połączeń z ich wyjściami („macierz sprzężenia zwrotnego"). Całość ilustruje rys. 2.9.

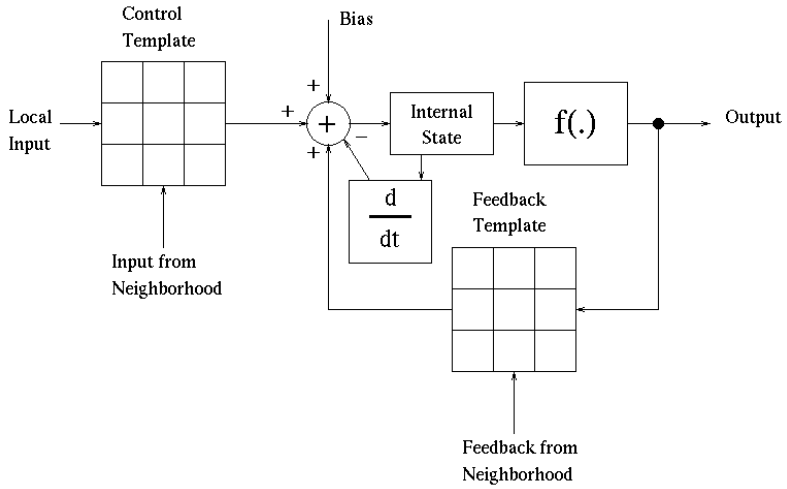
Wagi połączeń są ustalone z góry i nie ulegają zmianie w trakcie działania programu.



Rys. 2.8. Połączenia wewnątrz sieci CNN (rysunek zaczerpnięty z [20]).

Jako szablon sieci zastosowano zmodyfikowaną maskę zaznaczania krawędzi (promień sąsiedztwa zwiększony z $r = 1$ do $r = 2$) w wersji oryginalnej zaczerpnięty z [10]. (tab. 2.1 i tab. 2.2).

Jak wykazały testy [60], krawędzie były wykrywane w większości poprawnie, metoda zaś wykazywała znaczną odporność na szumy. Jako przykład efektu przetwarzania ob-




Rys. 2.9. Macierze sprzężenia zwrotnego i sterowania (rysunek zaczerpnięty z [19]).

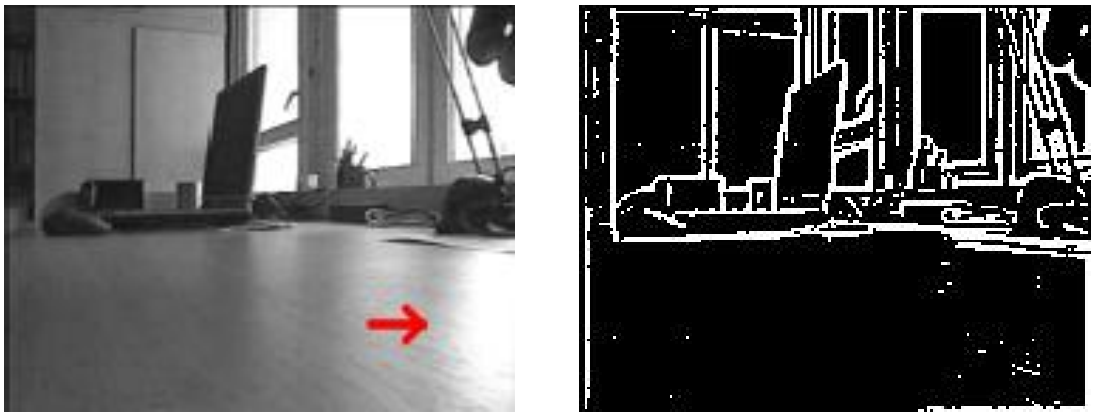
| | | | | |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 2 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 |

Tabela 2.1. Macierz sterowania dla $r = 2$.

| | | | | |
|---|---|---|---|---|
| $-1$ | $-1$ | $-1$ | $-1$ | $-1$ |
| $-1$ | $-1$ | $-1$ | $-1$ | $-1$ |
| $-1$ | $-1$ | 24 | $-1$ | $-1$ |
| $-1$ | $-1$ | $-1$ | $-1$ | $-1$ |
| $-1$ | $-1$ | $-1$ | $-1$ | $-1$ |

Tabela 2.2. Macierz sprzężenia zwrotnego dla $r = 2$.

razu (preprocessing + sieć CNN) może posłużyć rys. 2.10(a) i rys. 2.10(b) – szczególnie godny uwagi fragment zaznaczono na obrazie wejściowym czerwoną strzałką. Następuje w tym miejscu bardzo szybkie przejście szarości w niemal idealną biel (silny refleks światła słonecznego od gładkiej powierzchni blatu stołu). Testowane wcześniej klasyczne algorytmy miały poważne problemy (ze względu na lokalny charakter działania) – sieć neuronowa poradziła sobie bez tu problemu.



(a) (b)

Rys. 2.10. Detekcja krawędzi przy użyciu sieci neuronowej na przykładzie obrazu „stół": obraz wejściowy (a) oraz obraz wyjściowy (b).

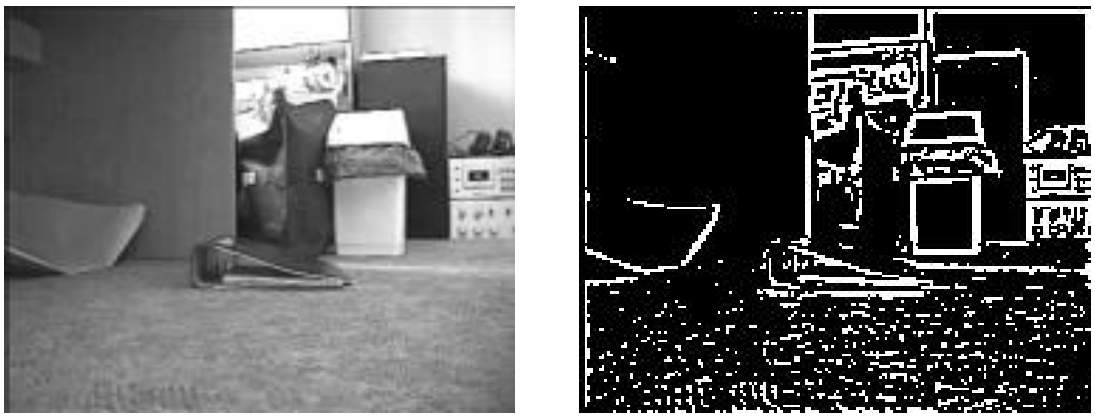
Wynik prezentowany na rys. 2.10(b) uzyskano już po 2 krokach dynamiki (synchronicznej) sieci. Prezentowana sieć jest bardzo szybko zbieżna, przez co po krótkim czasie można pobrać obraz wyjściowy.

Po uzyskaniu krawędzi z obrazu wejściowego należy dokonać jego binaryzacji, która ułatwia późniejszą pracę nie powodując przy tym praktycznie żadnych zauważalnych zmian wizualnych [60].

Kolejnym krokiem jest odszumianie obrazu. Mimo dobrych wyników sieci neuronowych w minimalizacji szumu, na powierzchniach „pstrokatych" pojawiają się często liczne drobne pseudo-krawędzie. Sytuację tę widać na przykładzie wielokolorowej wykładziny podłogowej (rys. 2.11(a)) i jej obrazu krawędzi (rys. 2.11(b)).

Drobne zakłócenia, jak te pokazane na rys. 2.11(b) są dość łatwe do usunięcia za pomocą prostej heurystyki – jeżeli powierzchnia pojedynczego obiektu jest mniejsza niż założony próg, należy go usunąć. Na takiej właśnie zasadzie działa operacja „denoise" [60]. Jak widać na rys. 2.12 radzi ona sobie dość dobrze z drobnym szumem. Ponieważ jednak występuje w niej element progujący (z progiem ustalonym apriori) algorytm ten trudno jest dostroić. Czasem jest to wręcz niemożliwe, kiedy w jednej części obrazu drobne krawędzie są istotne, w innym zaś nie.

Na koniec postprocessingu wykonywanych jest kilka drobnych operacji, takich jak łączenie krawędzi (operacja „connector" [60]), domknięcie obrazu [53] i otaczanie całego obrazu krawędzią (jest to wygodne przy definiowaniu segmentów). Końcowy efekt, dla przykładowych par wejście/wyjście ilustrują rysunki rys. 2.13 i rys. 2.14.

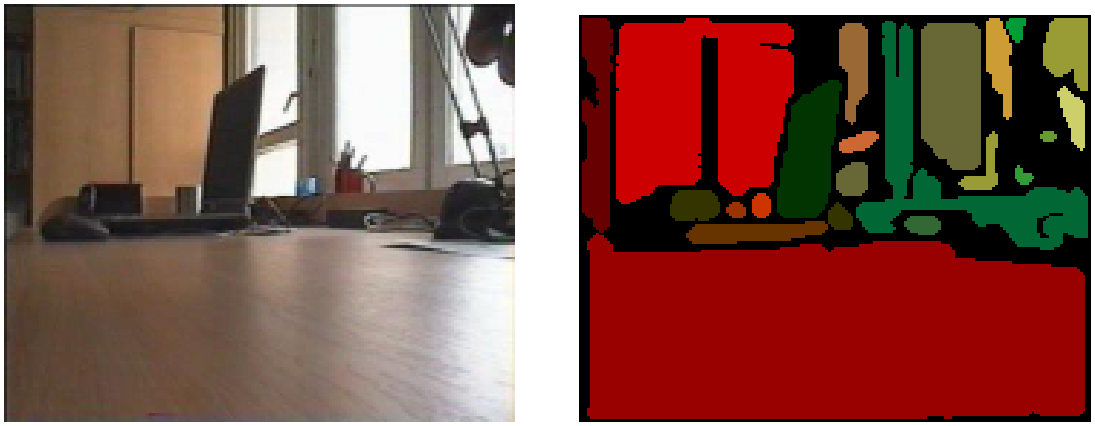


(a) (b)

Rys. 2.11. Detekcja krawędzi przy użyciu sieci neuronowej dla „pstrokatej" powierzchni: obraz wejściowy (a) oraz obraz wyjściowy (b).

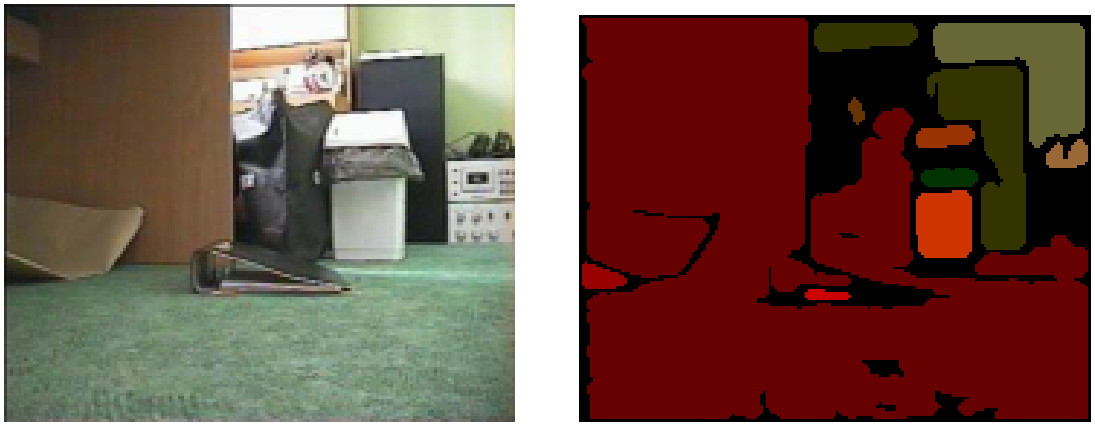
(a) (b)

Rys. 2.12. Efekt operacji „denoise" dla obrazu krawędzi „pstrokatej" powierzchni: (a) obraz wejściowy oraz (b) obraz wyjściowy.



(a) (b)

Rys. 2.13. Efekt całości procesu przetwarzania obrazu wejściowego „stół” metodą hybrydową: (a) obraz wejściowy oraz (b) obraz wyjściowy.



(a) (b)

Rys. 2.14. Efekt całości procesu przetwarzania obrazu wejściowego „pstrokaty” metodą hybrydową: (a) obraz wejściowy oraz (b) obraz wyjściowy.

### 2.2.4. Segmentacja „grafowa"

Drugie ze stosowanych podejść opiera się na reprezentacji grafowej. Podejście to zostało zaprezentowane w [5]. Cechuje się ono uwzględnianiem zarówno lokalnych, jak i nielokalnych aspektów analizowanego obrazu oraz dobrą złożonością obliczeniową ($O(n \cdot \log n)$), gdzie $n$ oznacza liczbę pikseli na obrazie).

Graf $G$ tworzony na podstawie obrazu wejściowego zdefiniowany został jako $G = (V, E)$ gdzie $V$ jest zbiorem węzłów (reprezentujących poszczególne segmenty). Początkowo każdy z pikseli obrazu znajduje się w osobnym węźle. $E$ jest zbiorem krawędzi łączących sąsiednie węzły. Graf jest grafem nieskierowanym, ważonym. Nieujemne wagi reprezentują różnicę pomiędzy łączonymi węzłami. Waga ta jest współczynnikiem mówiącym o tym jak bardzo dane fragmenty obrazu są do siebie podobne.

Nielokalność działania algorytmu polega na braniu pod uwagę przy porównywaniu dwóch segmentów ich wewnętrznej różnicy (dla każdego z osobna – jak bardzo segment jest różnorodny wewnątrz) oraz różnicy między nimi w relacji do ich różnic wewnętrznych [5]. Daje to w efekcie możliwość łączenia dwóch dość różnych segmentów, jeżeli są one wewnętrznie zróżnicowane (np: dwa fragmenty dywanu z pstrokatym wzorem mogą zostać połączone w jeden segment), z drugiej zaś strony segmenty o wysokim poziomie różnic między nimi, ale wewnętrznie spójne (np: brązowa pufa na zielonym dywanie) nie zostaną połączone. Daje to dużą przewagę nad algorytmami ze „statycznym" progiem akceptacji, kiedy to segmenty łączone są poniżej pewnego poziomu wzajemnych różnic (tj. dla wartości współczynnika podobieństwa mniejszego od zadanego, obszar jest uznawany za przynależny do pierwszej grupy, w przeciwnym przypadku zaś do drugiej).

Na początek zdefiniowana jest miara odległości wewnętrznej segmentu, jako największa z wag minimalnego drzewa rozpinającego (ang. *MST – Minimum Spanning Tree* [59]) tegoż segmentu $C$:

$$Itl(C) = \max_{e \epsilon MST(C,E)} w(e) \tag{2.1}$$

gdzie:

$$Itl(C) \quad - \quad \text{współczynnik różnic wewnętrznych segmentu } C \tag{2.2}$$
$$MST(C, E) \quad - \quad \text{minimalne drzewo rozpinające segmentu } C \tag{2.3}$$
$$w(e) \quad - \quad \text{waga krawędzi } e \tag{2.4}$$

Dla dwóch segmentów, jako ich wzajemną różnicę wewnętrzną, użyte jest minimum z ich wewnętrznych różnic zmodyfikowanych o funkcję skalującą, na podstawie rozmiaru segmentu:

$$MItl(C_1, C_2) = \min\{Itl(C_1) + \tau(C_1), Itl(C_2) + \tau(C_2)\} \tag{2.5}$$

gdzie:

$$\tau(C) = \frac{k}{|C|} \tag{2.6}$$

gdzie:

$$MItl(C_1, C_2) \quad - \quad \text{wzajemna różnica wewnętrzna } C_1 \text{ i } C_2 \tag{2.7}$$
$$k \quad - \quad \text{współczynnik proporcjonalności} \tag{2.8}$$
$$|C| \quad - \quad \text{rozmiar segmentu jako ilość węzłów (punktów)} \tag{2.9}$$

Funkcja $\tau(C)$ jest współczynnikiem powodującym, iż mniejsze komponenty wymagają większych różnic pomiędzy sobą, aby nie zostać połączonymi. Jest to potrzebne w krańcowych przypadkach, kiedy $|C| = 1$, co z kolei implikuje $Itl(C) = 0$ (różnica pojedynczego punktu z samym sobą).

Jak widać z powyższego, $k$ jest współczynnikiem proporcjonalności – dla większych jego wartości preferowane są większe segmenty. Stała ta zależy (nie wprost) od wielkości obrazu – obraz prezentujący tę samą scenę, lecz w wyższej rozdzielczości będzie wymagał większego $k$ aby uzyskać zbliżoną segmentację.

Następnym krokiem jest zdefiniowanie różnicy pomiędzy dwoma sąsiednimi segmentami. Jako różnicę przyjęto minimalną wagę łączącą dwa sąsiadujące segmenty:

$$Dif(C_1, C_2) = \min_{v_1 \epsilon C_1, v_2 \epsilon C_2, (v_1, v_2) \epsilon E} w\Big((v_1, v_2)\Big) \tag{2.10}$$

Jeżeli takowe połączenie nie istnieje (segmenty nie sąsiadują) jako wagę przyjmujemy nieskończoność. Ze względu na specyfikę algorytmu (wszystkie wagi są posortowane w sposób niemalejący) przyjęcie takiej właśnie definicji znacząco skraca czas działania algorytmu – przyjęcie mediany, lub innej podobnej funkcji jako $Dif$ powoduje, iż problem staje się NP-zupełny [5]. Badania wykazały też, iż taka definicja różności, mimo swej prostoty, sprawdza się bardzo dobrze w praktycznych przypadkach [5].

Mając zdefiniowane powyższe funkcje możemy zapisać predykat $D$, sprawdzający istnienie granicy pomiędzy 2 zadanymi segmentami, jak następuje:

$$D(C_1, C_2) = Dif(C_1, C_2) > MItl(C_1, C_2) \tag{2.11}$$

Na koniec warto zaznaczyć, iż istnieje ciekawy sposób na realizację segmentacji według dodatkowych kryteriów, poprzez definiowanie innej funkcji, która przykładowo preferowałaby zadany kształt, bądź rozmiar (zwracając dlań małe wartości) [5].

Segmentacja odbywa się w czasie $O(n \cdot \log(n))$ od liczby punktów obrazu według następującego algorytmu: [5]

1. Posortuj zbiór krawędzi $E$ w sposób niemalejący.
2. Niech $S_0$ będzie początkową segmentacją, taką, że każdy z wierzchołków $v_i$ jest w osobnym segmencie.
3. Powtarzaj krok 4. dla $q = 1, \ldots, m$, gdzie $m$ jest liczbą wszystkich krawędzi.
4. Zbuduj $S_q$ na podstawie $S_{q-1}$ w następujący sposób:
   - Niech $v_i$ oraz $v_j$ oznaczają wierzchołki połączone $q$-tą krawędzią (połączenie to zostanie oznaczone jako $o_q = (v_i, v_j)$).
   - Jeżeli $v_i$ oraz $v_j$ należą do różnych komponentów w segmentacji $S_{q-1}$ oraz waga $w(o_q)$ jest mała w porównaniu do wewnętrznych różnic poszczególnych segmentów, połącz segmenty.
5. Wynikiem jest $S = S_m$.

Interesującą kwestią jest fakt, iż segmentacja nie zawsze musi być jednoznaczna – przy postawionych założeniach i wybranym algorytmie, może okazać się że będzie

kilka „poprawnych" (alternatywnych) wariantów [5].

Na koniec jeszcze kilka słów o implementacji. Podobnie jak w poprzednim przypadku, obraz jest poddawany pewnemu przetwarzaniu wstępnemu (preprocessing) oraz końcowemu (postprocessing) według sekwencji:

1. Rozciągnij histogram (głównie ma to na celu skorygowanie kiepskiej jakości oświetlenia).
2. Przeprowadź wygładzanie Gauss'a (usuwamy w ten sposób drobny szum).
3. Wykonaj segmentację (główny krok – segmentacja jest przeprowadzana zgodnie z przedstawionym wyżej algorytmem).
4. Połącz ze sobą małe komponenty, aby uniknąć drobnych „plam".

W powyższych rozważaniach przez cały czas używany był termin wagi połączeń pomiędzy wierzchołkami. W praktyce można wykorzystać różne metody liczenia różnic pomiędzy danymi wierzchołkami/segmentami.

Autorzy [5] pokazują dwa podejścia:

1. Przestrzeń obrazu – waga jest liczona pomiędzy połączonymi wierzchołkami według różnicy w ich kolorze $w(v_i, v_j) = |I(p_i) - I(p_j)|$, gdzie $I(p)$ jest natężeniem koloru danego punktu $p$. Dla każdego z wierzchołków połączenia są realizowane tylko dla sąsiadów w sensie odległości na odpowiadających pozycjach pikseli w obrazie (np: $r = 1$ daje $8$ punktów).
2. Przestrzeń cech, gdzie waga jest liczona nie tylko na podstawie różnicy intensywności kolorów, lecz także na podstawie pozycji, czyli $w(v_i, v_j) = |f(x, y, I(p_i)) - f(x, y, I(p_j))|$. W takim przypadku połączenia pomiędzy węzłami teoretycznie mogłyby być zupełne (każdy z każdym), lecz z przyczyn wydajnościowych przerzedza się je.

Choć drugie z prezentowanych podejść jest ogólniejsze od pierwszego, nastręcza ono pewnych niedogodności z punktu widzenia przetwarzania na potrzeby sterowania robotem mobilnym. Ze względu na większy zasięg działania i inne kryterium oceny odległości w przestrzeni obrazu, może dojść do sytuacji, kiedy dwa obszary o bardzo podobnej kolorystyce nie sąsiadujące ze sobą zostaną połączone. Stanie się tak jeśli odległość (w przestrzeni cech) wyjdzie relatywnie mała, ze względu na zbliżony kolor. Może to prowadzić do znajdowania segmentów składających się z kilku obszarów nie będących w bezpośrednim sąsiedztwie [5].

Na potrzeby omawianego systemu wizyjnego, w którym chcemy stwierdzić gdzie znajduje się przeszkoda, aby móc ją ominąć, przyjęcie, że każdy z segmentów jest ciągły nie ogranicza w żaden sposób systemu jednocześnie upraszczając algorytm analizy.

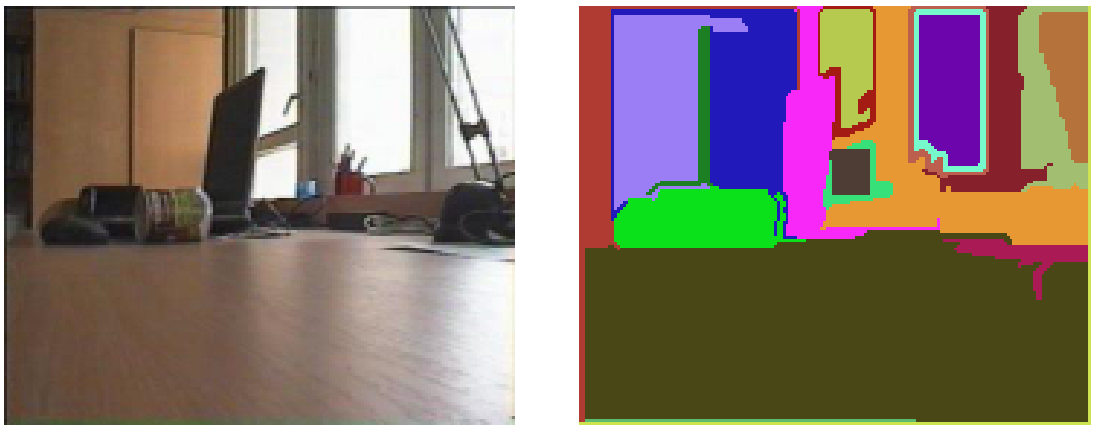
Na potrzeby wyjaśnienia działania samego algorytmu posługiwano się intensywnością $I$ danego punku $p$. Ma to przełożenie $1 : 1$ dla obrazów w odcieniach szarości. Czasem jednak ważna informacja przekazywana jest za pomocą koloru – dlatego właśnie pojawia się potrzeba określenia algorytmu dla reprezentacji zachowującej kolorystykę.

W przedstawionym rozwiązaniu zostało to zrobione poprzez uruchomienie algorytmu dla trzech składowych $R$, $G$, $B$ z osobna i połączenie rozwiązań (jeżeli dany punkt znalazł się w tym samym segmencie na wszystkich trzech jednobarwnych obrazach, trafia on do tegoż samego segmentu wyjściowego).

Z doświadczeń autorów wynika, iż takie podejście daje lepsze rezultaty niż przetwarzanie koloru $RGB$ na inne sposoby [5].

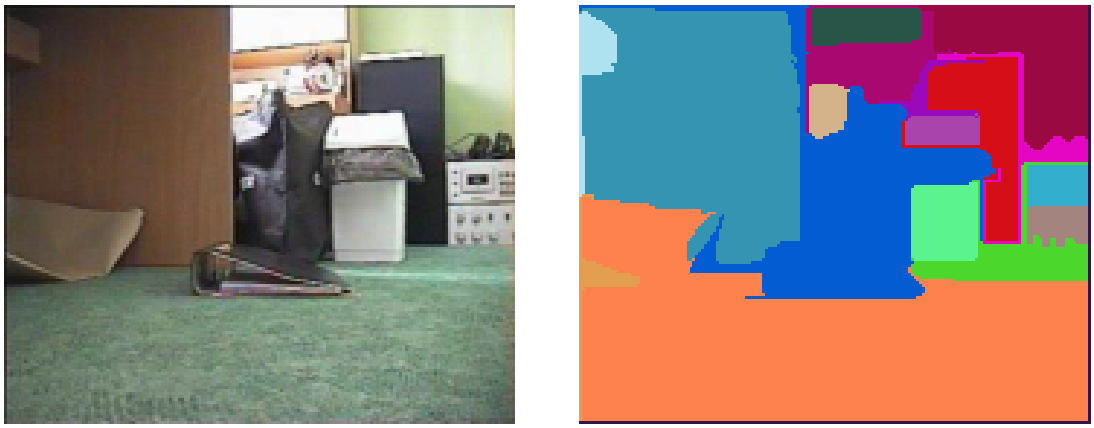
Do przeprowadzenia segmentacji algorytmem grafowym wykorzystano obrazy wejściowe z tej samej serii co, te z rys. 2.13 i rys. 2.14. Jako parametry algorytmu użyto

$\gamma = 0.9$ (parametr dla wygładzania Gauss'a), $k = 500$ (parametr proporcjonalności) oraz $min\_size = 130$ (minimalny rozmiar segmentu). Uzyskane w ten sposób obrazy wynikowe prezentują rys. 2.15 i rys. 2.16.



(a) (b)

Rys. 2.15. Efekt całości procesu przetwarzania obrazu wejściowego „stół" metodą grafową: (a) obraz wejściowy oraz (b) obraz wyjściowy.



(a) (b)

Rys. 2.16. Efekt całości procesu przetwarzania obrazu wejściowego „pstrokaty" metodą grafową: (a) obraz wejściowy oraz (b) obraz wyjściowy.

### 2.2.5. Porównanie metod segmentacji

Przeprowadzono szereg badań mających na celu porównania obu prezentowanych metod: „grafowej" i „hybrydowej". Z przyczyn objętościowych, w niniejszej pracy zostały zamieszczone tylko przykładowe wyniki.

Z przeprowadzonych badań wynika, iż prezentowany jako drugi, algorytm grafowy poradził sobie lepiej z zadaniem segmentacji obrazu. Wypadł on również znacznie lepiej czasowo – dla tych samych obrazów różnice w czasie przetwarzania były rzędu $400\%$. Na korzyść pierwszego z podejść („hybrydowego") przemawia trafniejsze

dobieranie krawędzi w niektórych miejscach testowanych obrazów. Prawdopodobną przyczyną takiego zachowania jest obecność sieci neuronowej w procesie przetwarzania. Dodatkową zaletą jest pojawianie się pewnej korelacji pomiędzy kolejnymi klatkami obrazu podczas przetwarzania sekwencji. Segmentacja „grafowa" przetwarza dane za każdym razem od początku. Sieć neuronowa zaś posiada swój stan (w postaci sygnałów jakie zostały podane na jej wejścia i wygenerowane na wyjściach) co powoduje istnienie pewnej relacji pomiędzy kolejnymi obrazami jakie są do niej wprowadzane. Jest to zaleta w przypadku kiedy kolejne klatki obrazu nie różnią się od siebie znacząco co ma miejsce przy ciągłym przetwarzaniu danych z kamery wideo.

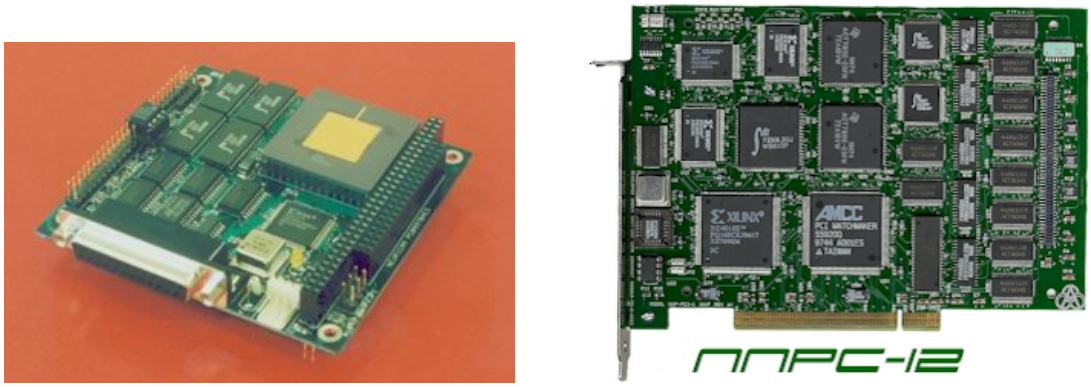
Lepsze (sumarycznie) wyniki segmentacji „grafowej" to w sporej mierze zasługa nielokalnego działania samego algorytmu grafowego. Rozwiązanie hybrydowe, prócz części neuronowej, bazuje niemal zupełnie na lokalnych podejściach, dlatego choć samo wyznaczanie krawędzi działa skutecznie, proces segmentacji, jako całość daje średnie wyniki.

W przypadku podejścia „hybrydowego", w niektórych miejscach pojawiały się również niedokładności w wyznaczaniu krawędzi. Dla porównania można przeanalizować łączenie linii stołu z dywanem na rys. 2.14(a) oraz rys. 2.16(b)). Prawdopodobnym powodem takiego zachowania sieci neuronowej jest kiepska rozróżnialność dwóch zupełnie różnych kolorów (ciemno brązowy i jasno zielony) po przejściu na odcienie szarości (rys. 2.11(a)). Ciekawym podejściem byłoby rozwinięcie metody „hybrydowej" do zespolonego przetwarzania danych w sieciach neuronowych czy też zastosowania trzech wymiarów (macierz/wektor wejściowy zamiast liczby). Możliwa byłaby także inna reprezentacja koloru, taka aby „mieściła się" w jednym wymiarze. Przykładowo zamiast trójkolorowej reprezentacji $RGB$ można zastosować pojedynczą liczbę reprezentującą długości fali świetlnej danej barwy.

Dla Tych samych obrazów wejściowych algorytm hybrydowy uzyskiwał na komputerze $P4/2.4GHz$ i $512MB$ RAM'u wydajność rzędu $4$-$5$ klatek na sekundę (dość długo też zajmowało przygotowanie algorytmu do działania, kiedy to trzeba było zbudować dużą sieć neuronową w pamięci), zaś algorytm grafowy około $16$-$20$ klatek. Zważywszy na konieczność pracy algorytmu w czasie rzeczywistym, ale w niezbyt szybko zmieniającym się środowisku, $4$-$5$ klatek na sekundę jest akceptowalną szybkością dla końcowego algorytmu, zawierającego przetwarzanie wyznaczonych segmentów i cały mechanizm wnioskujący, nie zaś dla niskopoziomowego przetwarzania obrazu wejściowego z kamery.

Z całą pewnością sytuacja mogłaby wyglądać inaczej, gdyby zastosowano do obróbki obrazu sprzętową realizację sieci neuronowej komórkowej. Można takie urządzenia nabyć w postaci kart rozszerzeń do komputerów klasy IBM-PC, PC104 i nie tylko [11][12][13][16]. Przykładowe zdjęcia tego typu kart przedstawiono rys. 2.17(a) i rys. 2.17(b).

Kolejną bardzo istotną różnicą pomiędzy prezentowanymi algorytmami była regularność (przewidywalność) czasu przetwarzania kolejnych klatek przychodzących do przetworzenia. Zagadnienie to jest ważne dla systemów czasu rzeczywistego, gdyż jeśli wszystkie elementy wchodzące w jego skład będą się zachowywały bardzo przewidywalnie całość systemu również będzie przewidywalna. Gwarantuje to zachowanie ciągłości analizy w czasie.




(a) (b)

Rys. 2.17. Wybrane karty rozszerzeń realizujące sieć CNN sprzętowo – karta dla komputera w standardzie PC104 (a – zaczerpnięte z [15]) i karta PCI dla standardowego komputera domowego PC (b – zaczerpnięte z [14]).

Podczas licznych testów jakie zostały przeprowadzone algorytm „hybrydowy" wykazywał bardzo wysoką stabilność – czas przetwarzania był praktycznie niezmienny niezależnie od podawanych na jego wejście obrazów. Działo się tak gdyż około $70-80\%$ całego czasu przetwarzania stanowiło wykrywanie krawędzi przy pomocy sieci neuronowej. Ponieważ zawsze trzeba było policzyć zadaną z góry ilość razy stan dla stałej liczby neuronów ta właśnie część systemu wykazywała praktycznie idealną stabilność czasową. Jedyne zauważalne różnice były powodowane przez operacje odszumiania obrazów wyjściowych oraz łączenia niektórych krawędzi.

Znacznie mniej przewidywalna okazała się metoda „grafowa". Odwrotnie niż metoda „hybrydowa", posiada ona tylko jeden etap wykonywany w stałym (dla zadanej wielkości obrazu) czasie – wygładzanie obrazu. Pozostała część bazuje na operacjach na zbiorach, te zaś charakteryzują się znacznymi różnicami w czasie wyszukiwania zależnie od liczby zawieranych elementów. Efektem końcowym jest bardzo szybka segmentacja obrazów zawierających duże, jednolite obszary oraz znacznie wolniejsza segmentacja dla obrazów z dużą ilością szczegółów. Tak duża różnica w ilości czasu potrzebnego na uzyskanie wyniku mogłaby stanowić utrudnienie przy tworzeniu systemu wizyjnego. Badania praktyczne pokazały jednak, iż w znaczącej większości sytuacji jakie można „zaobserwować" (przy pomocy kamery) poziom skomplikowania obrazów jest na tyle zbliżony, że nagłe, znaczące zmiany w czasie segmentacji praktycznie się nie zdarzają.

Podsumowując powyższe przemyślenia oraz mając na uwadze ograniczenia w dostępie do sprzętu, zdecydowano się pozostać przy rozwiązaniu jednocześnie wydajnym i czysto programowym. Wybór padł na podejście grafowe i to ono jest używane w kolejnych etapach pracy jako baza do dalszych analiz.

### 2.2.6. Wektoryzacja

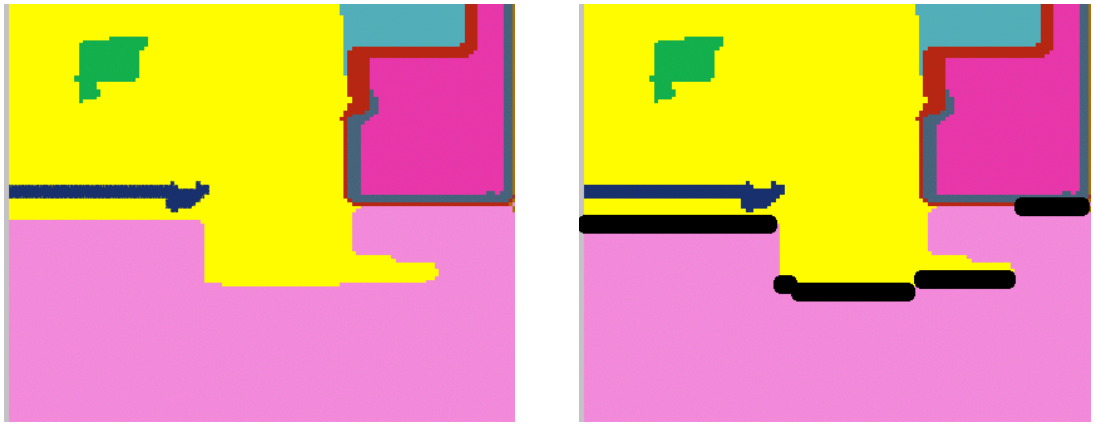
Po wstępnej obróbce obrazu (tj. normalizacji i segmentacji) kolejnym krokiem jest przetworzenie danych z postaci rastrowej na wektorową (rys. 2.4).

Na tym etapie analizy wybierając sposób wektoryzacji warto wziąć pod uwagę wygodę dalszego przetwarzania w późniejszych etapach. Przedstawiony wynik końcowy omawianego tu podejścia do wektoryzacji ułatwi odczytywanie głębi z obrazu – mechanizm ten zostanie dokładnie omówiony w podroz. 2.2.7.

Na obraz wejściowy składa się kilkanaście tysięcy punktów[10], jednak tak naprawdę tylko niewielka część okazuje się być dla nas istotną podczas odczytywania głębi. Wszystkie one stanowią linię krawędzi pomiędzy podłogą a przeszkodą (dolny obrys przeszkody).

Z dość dobrym przybliżeniem możemy przyjąć, iż dla każdego punktu na osi $OX$ (poziomej) istnieje co najwyżej jeden punkt krawędzi, znajdujący się na obrazie, który jest istotny dla dalszej analizy. Punkty te układają się w łamaną linię (niekoniecznie ciągłą). Przykład takiej linii przedstawiono na rys. 2.19 (obraz powstały po wektoryzacji rys. 2.18) po uprzednim podzieleniu go na segmenty. Linię oznaczono na czarno.

Rys. 2.18. Obraz z kamery robota – kosz zagradzający drogę.



(a) (b)

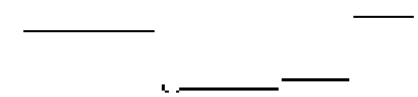
Rys. 2.19. (a) Obraz po segmentacji rys. 2.18 (b) z zaznaczoną na czarno linią istotną dla dalszej analizy (odczytywania głębi).

Widać iż niezwykle łatwo jest znaleźć takie linie w sposób automatyczny – wystarczy dla każdego punktu na osi $OX$ znaleźć pierwszy punkt, który należy do innego

10. Rozmiar analizowanych podczas testów obrazów wynosił typowo około $150 \times 120$ punktów.

obszaru niż początkowy (nawiązując do rysunku – jest innego koloru). Mając listę takich punktów w bardzo prosty sposób można przekształcić ją do postaci wektorowej. Metoda również nie musi być optymalna czasowo ponieważ uzyskanych w ten sposób punktów jest niewiele (dla prezentowanych rozmiarów obrazów wejściowych typowo jest ich od kilkudziesięciu do stukilkudziesięciu) – wystarczy łączyć je jeżeli leżą na jednej linii i tworzyć nowy wektor w przypadku załamania linii.

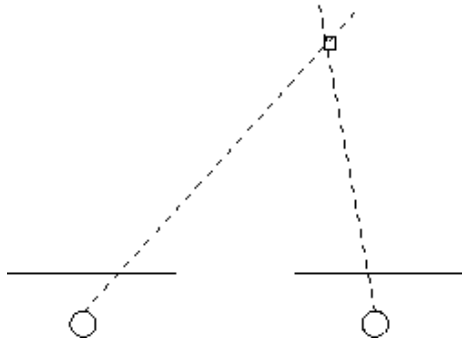
Efekt takiej transformacji pokazuje rys. 2.20. Obraz ten prezentuje już postać wektorową.



Rys. 2.20. Obraz z rys. 2.19(a) po zaznaczeniu ważnych punktów i ich wektoryzacji.

### 2.2.7. Odczytywanie głębi

W literaturze szeroko opisane są różne metody odczytywania głębi z obrazów wideo na podstawie danych z systemów stereowizyjnych [44][53]. Pozyskanie takich informacji z dwóch kamer jest możliwe ponieważ dzięki posiadaniu obrazów nieco różniących się perspektywą jesteśmy w stanie wyznaczyć punkt przecięcia dwóch prostych a tym samym jednoznaczny punkt w przestrzeni. Ilustrację tej sytuacji widzianej z góry prezentuje rys. 2.21.



Rys. 2.21. Wyznaczanie głębi w obrazie na podstawie obrazu stereowizyjnego. Rysunek zrobiony na podstawie [53].

Z goła inna sytuacja ma miejsce w przypadku posiadania tylko jednego obrazu – mając jeden obraz, mamy tylko jedną prostą wyznaczającą nam nieskończenie wiele punktów w przestrzeni. Odczytanie głębi z takiego obrazu nie jest więc możliwe bez dodatkowej wiedzy (najbardziej ogólny przypadek).

Ciekawą kwestią do wyjaśnienia jest tu fakt (często przytaczany podczas dyskusji jakie autor miał okazję przeprowadzić odnośnie niniejszej pracy), iż człowiek, zamykając jedno oko, jest nadal w stanie bezbłędnie wychwytywać głębię z obrazu jaki widzi. Jest to często popełniany błąd – to co człowiekowi wydaje się, że widzi jest tak naprawdę obrazem silnie przetworzonym przez jego mózg. „Głębia" jaką widzimy zamykając jedno oko jest wynikiem pracy umysłu łączącego ze sobą ogromne ilości faktów w jedną całość – wiemy jak wyglądają pewne przedmioty, mimo iż w danej chwili nie widzimy ich w całości, widząc cień na korytarzu i wiedząc skąd pada światło jesteśmy w stanie ocenić jaki obiekt go rzuca, etc… Do tego dochodzi jeszcze fakt posiadania przez człowieka innych zmysłów, z których zupełnie nieświadomie w takiej sytuacji korzystamy.

Znając środowisko jesteśmy w stanie „uzupełnić" informacje brakujące w obrazie przez fakt patrzenia tylko przez jedno oko – sytuacja zmienia się diametralnie, kiedy stajemy przez zadaniem odczytania informacji o odległościach w widzianym obrazie, gdy nie mamy żadnego punktu odniesienia, który pozwalałby nam odwołać się do posiadanej wiedzy.

Dobrym przykładem jest tu łódź na plaży – wszędzie dookoła jest piasek zaś pośrodku znajduje się mała łódka wiosłowa. Jeżeli zobaczymy taki obraz na widokówce (płaski obraz) nie wiemy czy to prawdziwa łódź, do której zmieściłby się człowiek, czy też może sprytnie uchwycone przez fotografa makro-zdjęcie modelu/zabawki. Tego typu wątpliwości nie wystąpiłyby, jeżeli patrzylibyśmy na „obiekt na plaży" na własne oczy[11].

Jeżeli więc nie da się wyznaczyć głębi z obrazu monowizyjnego, zaś ogrom wiedzy, jaką należałoby uwzględnić aby uczynić to realnym dla świata w jakim żyjemy na codzień jest wręcz niewyobrażalny, pytanie brzmi: czy istnieje możliwość przyjęcia pewnych dodatkowych założeń odnośnie charakteru otoczenia, które czyniłyby problem rozwiązywalnym?

W niniejszym podrozdziale zaprezentowane zostanie własne podejście do tego problemu, gdzie wyznaczenie głębi odbywać się będzie w dość prosty sposób matematyczny – kluczowe są przyjęte założenia:

1. Podłoże, po jakim porusza się robot jest płaskie.
2. Przeszkody jakie robot napotyka są prostopadłe do podłogi i jej dotykają (linia przeszkody na podłodze jest linią graniczną na każdej wysokości – patrząc z góry nic powyżej nie „wystaje" poza tę linię).

Są to założenia podstawowe – znając je jesteśmy w stanie stwierdzić, jaka jest odległość danego punktu przecięcia (krawędzi) przeszkody z podłogą (warunek pierwszy). Aby taka informacja wystarczała do bezkolizyjnego poruszania się, wymagane jest, by zarejestrowana przez robota krawędź była linią, przed którą droga jest przejezdna (drugie założenie).

Na pierwszy rzut oka założenia te wydają się bardzo silne. Tak jest w istocie – ograniczają one w sposób znaczący możliwą przestrzeń, w jakiej robot może się poruszać. Z drugiej jednak strony, założenia te wręcz idealnie spełnia znacząca

11. Tego typu efekty często są wykorzystywane w przemyśle filmowym aby obniżyć koszty produkcji pewnych scen [1].

większość powierzchni wnętrz budynków[12].

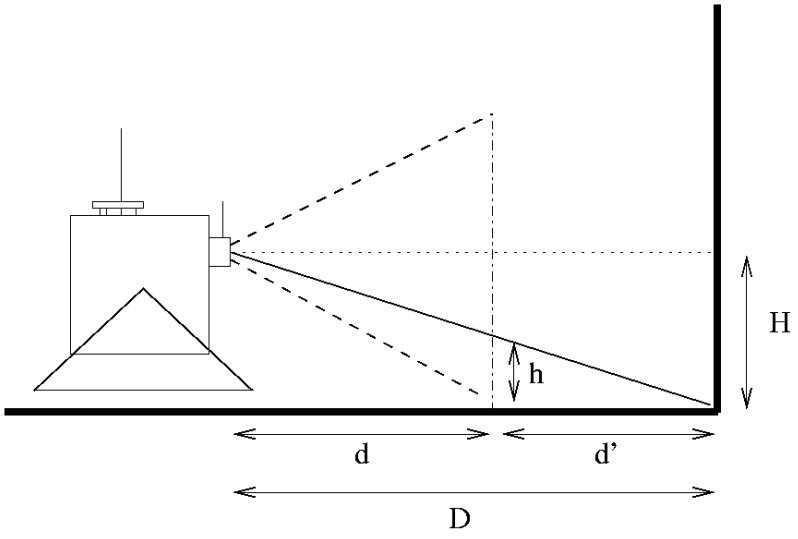
Dodatkowym założeniem, jakie przyjęto w niniejszej pracy jest punkt mocowania kamery wideo. Musi się on znajdować na pewnej, znanej wysokości nad podłogą, sama kamera zaś musi patrzeć „na wprost", tak aby jej oś optyczna była równoległa do płaszczyzny podłogi.

Takie wymagania, choć nie są konieczne, upraszczają nieco część matematyczną, a przede wszystkim, wymagają prostych parametrów wejściowych, jakie należy zmierzyć (empirycznie) i dostarczyć do algorytmu.

Znając wymagania stawiane prezentowanemu systemowi wizyjnemu, można przejść do konkretnych wyliczeń matematycznych mających na celu przekształcenie punktu $(x_0; y_0)$ z układu współrzędnych uzyskanego obrazu wektorowego w punkt $(x_1; y_1)$ wyjściowego obrazu wektorowego, reprezentującego głębię[13].

Dla bardziej intuicyjnego rozumienia współrzędnych obu przestrzeni (wejściowej i wyjściowej) przyjmijmy, że dla obrazu wejściowego $OX$ jest osią horyzontalną zaś $OY$ wertykalną, oraz $x_0 < 0$ oznacza punkty leżące na lewo od osi optycznej, zaś $x_0 > 0$ leżące na prawo od niej. Najniższe punkty możliwe do zaobserwowania przy użyciu kamery to punkty o współrzędnej $y_0 = 0$. Podobne oznaczenia zostaną przyjęte dla obrazu wyjściowego – oś $OX$ reprezentuje kierunki lewo/prawo (kolejno $x_1 < 0$ i $x_1 > 0$) zaś oś $OY$ odległość „na wprost" od robota na mapie (czyli robot znajduje się w punkcie $(0; 0)$ i „patrzy" w stronę $[0; 1]$).

Na rys. 2.22 przedstawiono widok sceny z robotem widzianym z jego boku. Zaznaczono na nim również wszystkie używane w dalszych rozważaniach parametry.




Rys. 2.22. Schemat wyznaczania głębi z obrazu – widok z boku.

12. Choć bywają problematyczne miejsca, takie jak na przykład schody lub krzesła (jeżeli robot jest wysoki).
13. Dalej obraz ten będzie nazywany „mapą", gdyż daje on obraz podobny do tego, jaki widziałby robot gdyby kamera znajdowała się nad nim i patrzyła na wszystko z góry. Dokładny opis reprezentacji rzeczywistości w systemie znajduje się w podroz. 2.2.8.

Informacje jakie algorytm musi znać apriori to wysokość, na jakiej znajduje się kamera ($H$) oraz odległość, w jakiej znajduje się początek podłogi widziany przez kamerę ($d$). Z obrazu wejściowego pobieramy wysokość, na jakiej znajduje się zaobserwowany punkt (jego współrzędna $y_0$). Docelowo chcemy wyznaczyć odległość, w jakiej znajduje się nasz punkt od robota: $D = d + d'$.

Z twierdzenia Talesa możemy zapisać:

$$\frac{D}{H} = \frac{d}{H-h} = \frac{d'}{h} \tag{2.12}$$

Czyli:

$$D = \frac{d \cdot H}{H-h} = y_1 \tag{2.13}$$

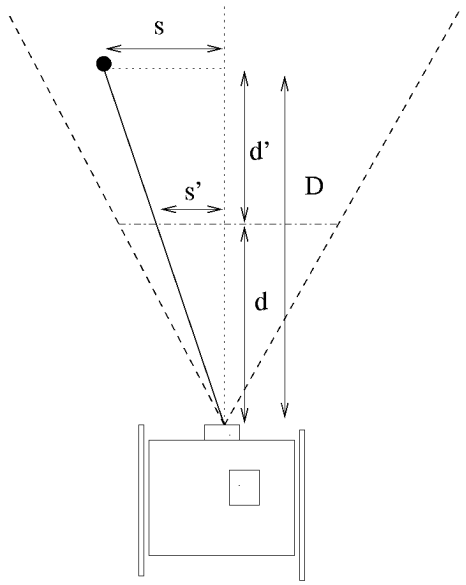
W ten sposób mamy odczytaną odległość od robota „na wprost" – gdyby badany punkt leżał dokładnie na przeciw kamery, byłaby to nasza odległość od celu. Jeżeli jednak punkt ten nie leży idealnie w osi kamery to licząc odległość wzdłuż osi $OX$ należy dodatkowo uwzględnić odległość na osi $OY$, w jakiej widziany obiekt się znajduje. Łatwo sobie wyobrazić taką sytuację patrząc na prostą drogę asfaltową na długim odcinku – z powodu perspektywy, z jakiej patrzymy, wydaje się nam ona zwężać i mieć kształt stożkowaty, mimo iż w rzeczywistości jej brzegi są liniami równoległymi (rys. 2.23).

Rys. 2.23. Prosta droga – równoległe linie krawędzi wydają się zbieżne (zaczerpnięte z [21]).

Pomocniczy szkic poglądowy wraz z zaznaczonymi wielkościami używanymi w wyprowadzeniu przedstawiono na rys. 2.24.

Ponownie korzystając z Twierdzenia Talesa, analogicznie do równ. 2.12, na podstawie rys. 2.24 możemy zapisać:

$$\frac{d}{s'} = \frac{D}{s} \tag{2.14}$$




Rys. 2.24. Schemat wyznaczania głębi z obrazu – widok z góry.

Po prostych przekształceniach otrzymujemy końcowy wzór na wyliczenie odległości punktu od osi optycznej kamery:

$$s = \frac{D}{d} \cdot s' = x_1 \tag{2.15}$$

Korzystając z równ. 2.13 i równ. 2.15 jesteśmy w stanie przekształcić współrzędne „obrazu" na współrzędne „mapy" (czyli stworzyć mapę, z której można pobrać informacje o odległościach).

Na koniec tego podrozdziału warto wspomnieć dwa słowa o dokładności prezentowanej metody. Na rys. 2.22 widać wyraźnie, iż w miarę przesuwania obserwowanego przedmiotu (krawędzi) dalej od robota, kolejnym przyrostom odległości $D$ odpowiadają coraz to mniejsze przyrosty wysokości $h$ na obrazie obserwowanym.

Teoretycznie nie jest to problemem, jednak w praktycznych zastosowaniach nasze dane wejściowe nie są ciągłe lecz skwantowane (obraz wejściowy jest rastrem), czyli tak naprawdę, im dany obiekt jest dalej, tym większe zmiany odległości są konieczne, aby pokazał się on na kolejnym (wyższym) pikselu obrazu. Efekt ten powoduje, iż od pewnej wysokości $h$ obrazu wejściowego informacja o odległości jest tak niepewna, że jej analiza wogóle nie ma sensu. Da się to zjawisko wykorzystać w pewien sposób – zakładając maksymalny błąd odczytu odległości $\Delta_d$ (np: $\Delta_d = 5[cm]$) można obliczyć, jaka część obrazu jest użyteczna dla potrzeb analizy. Wystarczy iterować począwszy od połowy obrazu podążając ku jego dolnej krawędzi i przeprowadzać transformację odległości dla kolejnych punktów, zgodnie z równ. 2.13 oraz badać zamiany odległości kolejnych (sąsiednich) punktów. Kiedy znajdziemy punkt $y_i$ spełniający nierówności:

$$|D(y_{i+1}) - D(y_i)| < \Delta_d \tag{2.16}$$

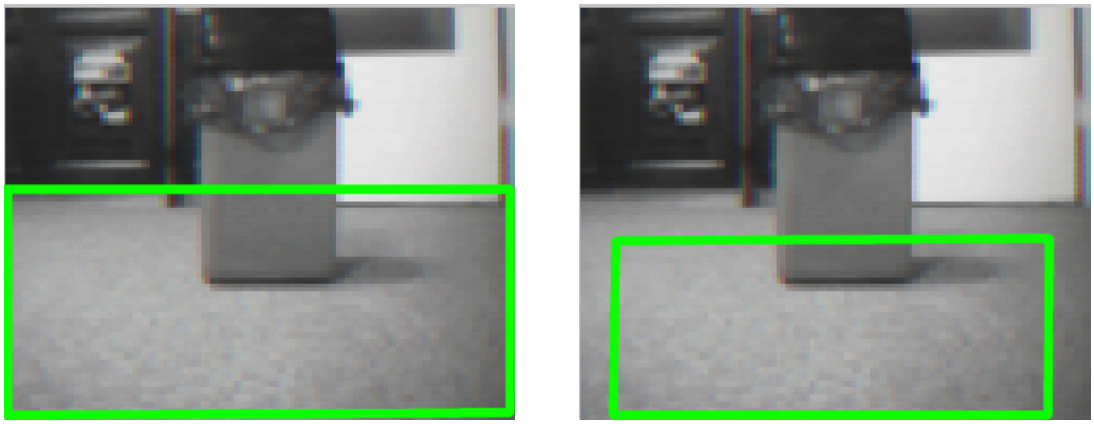
$$|D(y_{i-1}) - D(y_i)| < \Delta_d \tag{2.17}$$

gdzie:

$$D(y) - \text{Transformacja głębokości zgodna z równ. 2.13} \tag{2.18}$$

oznaczać to będzie znalezienie maksymalnej wysokości $h$ na obrazie wejściowym, dla którego pomiary będą się mieściły w granicy przyjętego błędu[14]. Punkt ten oznaczmy $y_i = y_{max}$.

Analogiczne wyliczenia można przeprowadzić dla osi $OX$ (zakładając $y = y_{max}$ z poprzednich wyliczeń) uzyskując prostokątny fragment obrazu, który jest istotny dla obliczeń, zachowując zadaną dokładność. Po przeprowadzeniu prezentowanych operacji, oraz uwzględnieniu dokładności uwidacznia się ogromna wada przedstawionego podejścia – do rzeczywistej analizy teoretycznie użyteczne jest zaledwie $50\%$ powierzchni obrazu wejściowego. Po uwzględnieniu pewnej dokładności powierzchnia ta maleje jeszcze bardziej – w omawianym systemie używano jedynie około $40\%$ całości. Przypadek ten został obrazowany na rys. 2.25.



(a) (b)

Rys. 2.25. Obraz z kamery robota (a) z zaznaczoną częścią użyteczną teoretycznie oraz (b) częścią użyteczną przy pewnej, założonej dokładności.

### 2.2.8. Reprezentacja przestrzeni

Po przetworzeniu obrazu z kamery pod kątem uzyskania głębi otrzymano mapę tego, co aktualnie robot widzi. Mapa taka jest tworzona przy każdym cyklu obróbki obrazu od nowa, mimo iż robot przemieścił się jedynie nieznacznie od tamtej chwili[15].

W przypadku idealnym, kiedy robot stoi w miejscu, każda kolejna klatka obrazu powinna dawać takie same wyniki po przetwarzaniu. W rzeczywistości każdy z tych obrazów, choć wizualnie jest niemal identyczny, okazuje się różnić binarnie na tyle wyraźnie od pozostałych, iż dla wielu algorytmów daje on znacząco inne rezultaty. W prezentowanym przypadku widać tę sytuację doskonale podczas segmentacji – kiedy obraz jest analizowany, po zwizualizowaniu wyników w czasie rzeczywistym, efekty przetwarzania ujawniają pewne niedoskonałości – pewne mniejsze obszary raz należą do jednego segmentu, innym razem do sąsiedniego, jeszcze innym razem zaś są zupełnie osobnym fragmentem. Okazjonalnie trafia się też skrajnie inna segmentacja, kiedy to np: duża płaszczyzna jest raz całością a raz dwiema osobnymi częściami. Dzieje się tak ponieważ decyzja o połączeniu (lub nie) dwóch sąsiednich segmentów jest podejmowana

14. Tzn: przemieszczenie piksela o jedną pozycję w bok nie spowoduje błędu większego niż założony.
15. Badania pokazały, iż prezentowany system osiąga prędkość przetwarzania około $4$-$5$ klatek na sekundę.

na zasadzie progowania (ang. *thresholding*). Jeżeli więc trafią się dwa sąsiednie obszary, nieznacznie różniące się (na skutek niedokładności kamery i zmienności warunków zewnętrznych) może się przytrafić sytuacja, w której system raz uzna, iż różnica pomiędzy tymi obszarami jest nieco powyżej progu łączenia, zaś przy następnym wykonaniu zakwalifikuje różnicę nieco poniżej progu. Efektem tego są dwie bardzo różne segmentacje, dla wizualnie identycznych obrazów. Choć problem ten występuje sporadycznie dla systemu przetwarzającego wiele obrazów w ciągu sekundy oznacza to, że okresowo trafia się niepoprawna segmentacja niosąca błędną wiedzę o obserwowanym otoczeniu.

Jednym z możliwych rozwiązań jest wprowadzenie pewnej zależności pomiędzy tym co zostało właśnie zaobserwowane (bieżący obraz) a tym co było zaobserwowane wcześniej (poprzednie obrazy). Aby uzależnić w jakiś sposób kolejne odczyty między sobą, wprowadzono dla każdej wykrytej i przetransformowanej krawędzi czas życia (ang. *TTL – Time To Live* – na wzór stosu protokołów TCP/IP [57]). Każda krawędź znaleziona na mapie „lokalnej" (chwilowej) dostaje licznik czasu („TTL") i jest dodawana do mapy „globalnej". Podczas wykonywania cyklu działania systemu, „TTL" każdego elementu na mapie globalnej jest zmniejszany – kiedy wartość ta osiągnie $0$ element jest z niej usuwany. Dzięki takiemu podejściu system zabezpieczony jest przed sytuacją, kiedy w wyniku błędnej segmentacji krawędź nie zostanie zauważona w jednej z klatek (cykli) i robot będzie jechał w stronę przeszkody[16].

Empirycznie zostało stwierdzone, iż dla systemu dobrą wartością „TTL" jest $4$ (co odpowiada czasowi około $1s$) – daje ona możliwość zapamiętywania ostatnich wyników uodparniając tym samym system na błędy pojedynczych segmentacji. Jednocześnie nie powoduje to pojawiania się zbyt dużych zależności czasowych, które wyraźnie przeszkadzają podczas skręcania, kiedy to na mapie zmienia się najwięcej.

Podejście z „czasem życia" spełnia dobrze swoje zadanie „uśredniając" wyniki uzyskane z kilku ostatnich klatek obrazu, wpływając pozytywnie na ilość informacji, na podstawie których podejmowana jest decyzja w systemie.

## 2.3. System sterujący

Podrozdział ten opisuje sposób w jaki robot jest sterowany, na jakiej podstawie podejmowane są decyzje co do kierunku ruchu, jak wykrywane są kolizje oraz jak robot ich unika. Opisy zostały wzbogacone przykładami ilustrującymi poruszane problemy.

### 2.3.1. Komunikacja i sterowanie

Komunikacja z robotem odbywa się 2-kierunkowo: do robota są wysyłane rozkazy z komputera, robot zaś odsyła odpowiedzi. Odbywa się to za pośrednictwem wspominanego w dod. A modułu radiowego [4].

Ponieważ w transmisji komputer-mikrokontroler wykorzystywany jest port RS232, kwantem danych jest tu $1$ bajt. Jest to ilość danych zdecydowanie niewystarczająca na potrzeby sterowania opisywanym robotem, gdyż samo ustawianie prędkości i kierunku obrotu pojedynczej gąsienicy wymaga, aby dało się przesłać około $400$ różnych wartości (czyli minimum $9$ bitów).

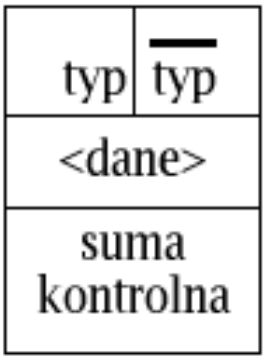
Kolejnym problemem jest stratność transmisji – podczas przesyłania danych drogą radiową istnieje duże ryzyko wystąpienia przekłamania wiadomości.

16. W skrajnym przypadku mogłoby to zaowocować uderzeniem w przeszkodę.

Aby sprostać stawianym wymaganiom opracowany został (relatywnie prosty) protokół transmisyjny, wysyłający dane w paczkach. Nadawane paczki są stałej długości – ułatwia to i znacznie przyśpiesza detekcję błędów w powolnym mikrokontrolerze. Każda z paczek protokołu składa się z 3 podstawowych elementów:

1. Nagłówek – informuje on jakiego rodzaju dane są transmitowane (jaki rozkaz dla robota).
2. Dane – blok danych o stałej długości. Blok ten jest opcjonalny – jego obecność zależy od rodzaju transmitowanego pakietu.
3. Suma kontrolna – element znacznie zmniejszający ryzyko przekłamania danych podczas przesyłania potencjalnie stratnym kanałem radiowym. Daje ona szansę wykrycia nawet kilkubitowego błędu transmisji.

Aby uniknąć błędu w kluczowej informacji, jaką jest rodzaj pakietu, nagłówek opatrzony jest dodatkową restrykcją – młodsza połówka bajtu nagłówka jest odwrotnością bitową starszej połówki. Jest to dodatkowa suma kontrolna samego nagłówka. Budowę pakiety przedstawia rys. 2.26.




Rys. 2.26. Budowa pojedynczego pakietu protokołu stosowanego do komunikacji z robotem drogą radiową.

Kamera Przechwytująca obraz, zainstalowana w przedniej części robota, posiada własny nadajnik [6]. Z przyczyn technicznych, oba nadajniki (kamery oraz radiowy – sterujący) znajdują się blisko siebie oraz w pobliżu generujących znaczne zakłócenia stopni mocy (układów nawrotnych omawianych w podroz. A.5.2) i silników.

Wybrany sprzęt nie interferuje ze sobą, ponieważ oba nadajniki nadają na znacząco różnych częstotliwościach oraz są małej mocy[17] [6][4].

Na koniec warto zaznaczyć, iż kamera jest czarnobiała oraz posiada własne, dodatkowe oświetlenie w postaci 4 diod LED[18] świecących w zakresie bliskiej podczerwieni [6]. Poprawia to nieco jasność obrazu przy niewielkich odległościach w przypadku bardzo słabego światła zewnętrznego.

### 2.3.2. Ruch i wykrywanie kolizji

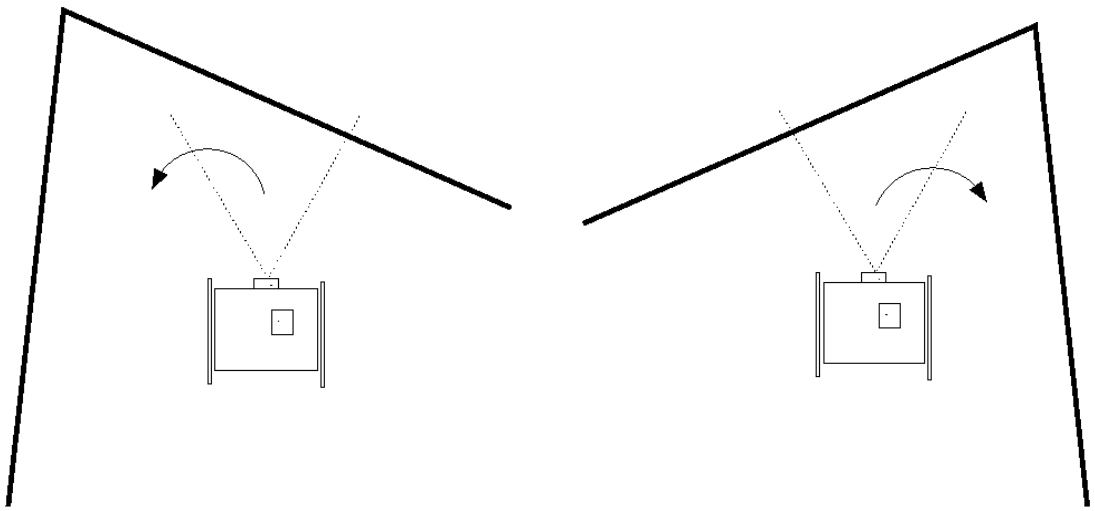
Zadaniem stawianym przed robotem jest samodzielne poruszanie się w terenie, musi on więc omijać przeszkody jakie się znajdują w otaczającym go środowisku.

17. Taka konfiguracja byłaby nie do zaakceptowania w przypadku, gdyby robot miał mieć większy zasięg i miał gwarantować wysoką niezawodność przekazu, nie mniej, dla potrzeb przeprowadzanych testów, omawiane rozwiązanie zdaje egzamin.
18. LED - ang. *Light Emitting Diode*.

Sposób w jaki są identyfikowane przeszkody oraz jak robot je „zapamiętuje" opisano we wcześniejszych podrozdziałach, jednak samo wykrycie przeszkody nie rozwiązuje problemu sterowania – potrzebna jest wiedza, w jaki sposób najlepiej będzie ją ominąć.

Ponieważ system w danej chwili czasowej jest w posiadaniu pojedynczego obrazu otoczenia, nie ma możliwości podjęcia globalnej decyzji – konieczne jest stosowanie pewnych heurystyk mających na celu podjęcie „prawdopodobnie dobrej" decyzji.

Pierwszym pomysłem, jaki się nasuwa jest skręcanie w tę stronę, gdzie znajduje się mniej przeszkód, czyli tam, gdzie łączna suma długości wykrytych krawędzi jest mniejsza. Pojawia się tutaj podstawowy problem: może się zdarzyć sytuacja, kiedy robot wjedzie w zaułek, np: do narożnika – każdy kolejny zakręt będzie powodował zmianę kierunku na przeciwny i robot zacznie się „odbijać", raz w jedną, raz w drugą stronę (rys. 2.27).



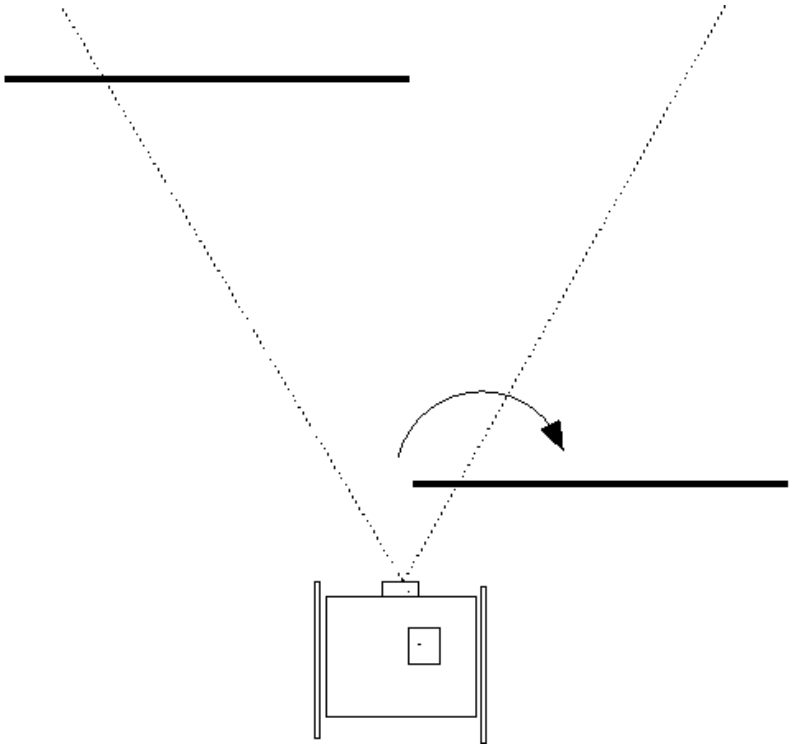
(a) (b)

Rys. 2.27. Sytuacja zakleszczenia dla najprostszej heurystyki sterującej, kiedy robot zaczyna skręcać naprzemiennie w obie strony bez końca: (a) sygnał skręcania w lewo; (b) sygnał skręcania w prawo.

Problem ten rozwiązuje dodanie kolejnego elementu pamiętającego, zapewniającego informacje o poprzednio podjętych decyzjach odnośnie skręcania. Dodana heurystyka modyfikuje schemat podejmowania decyzji – jeżeli droga jest zablokowana, skręt odbywa się w wybraną ze stron (na podstawie omawianej wcześniej „łącznej długości przeszkód"), jednak jeżeli robot już jest w trakcie wykonywania skrętu, należy go utrzymać w stanie skręcania w tę właśnie stronę, aż do napotkania wolnej drogi, kiedy znów może on jechać przed siebie.

Choć rozwiązanie jest znacznie lepsze od poprzedniego, nadal istnieje sytuacja, w której robot zachowuje się niepoprwanie. Tym razem problematyczna jest metoda podejmowania decyzji o kierunku skrętu – jeżeli przykładowo przeszkoda jest blisko po prawej stronie robota, lecz jest mała, zaś po stronie lewej jest dużo więcej długich krawędzi, lecz daleko robot i tak skręci w lewo – prosto na przeszkodę… Ilustruje to rys. 2.28.

Ostateczna wersja heurystyki wybierającej kierunek skręcania została wzbogacona o wagi – odległości krawędzi były do tej pory sumowane „liniowo", czyli liczyła się wy-



Rys. 2.28. Zbyt naiwna heurystyka skrętu powoduje skręt wprost na przeszkodę.

łącznie ich sumaryczna długość. Po dodaniu wag krawędzie znajdujące się dalej (a konkretniej ich długość) były mniej znaczące niż te bliżej, co zapewniało „rozsądniejsze" decyzje – omijanie przede wszystkim przeszkód będących w bezpośrednim sąsiedztwie robota.

W praktycznej realizacji, jako wagę każdej krawędzi zastosowano odwrotność jej zaobserwowanej odległości od robota. Praktyka pokazała, iż krawędzie są bardzo krótkie i przeważnie nie ma większego znaczenia jak się odległość do krawędzi policzy – można to zrobić przykładowo biorąc za odległość krawędzi:

1. odległość bliższego wierzchołka krawędzi
2. odległość dalszego wierzchołka krawędzi
3. średnią odległość obu wierzchołków krawędzi
4. odległość dowolnego (przypadkowego) wierzchołka krawędzi – to podejście jest najwygodniejsze z punktu widzenia implementacji i czasu obliczeń[19].

Końcową wersję podsystemu odpowiadającego za sterowanie robotem przedstawia poniższy pseudo-kod:

```
Strona={lewa,prawa}
Kierunek={w_lewo,w_prawo,do_przodu}

double suma_ważona_krawędzi_po_stronie(Strona s, Mapa m)
{
  double wynik=0
  for( Krawędź k IN zbiór_krawędzi(m) )
```

19. Po tym jak zapisaliśmy krawędź w postaci odcinka, bez wykonywania dodatkowych wyliczeń, nie wiadomo, który z jego punktów jest bliżej, a który dalej…

```
    // Czy krawędź 'k' jest po stronie 's' robota?
    if( jest_po_stronie(k, s) )
      // waga() - funkcja malejąca wraz ze wzrostem
      // odległości krawędzi 'k' od robota.
      wynik+= długość(k) * waga(k)
  return wynik
}

Kierunek wybierz_kierunek_ruchu(Kierunek obecny, Mapa m)
{
  if( droga_zablokowana()==false )
    return do_przodu
  if(obecny!=do_przodu)
    return obecny
  if( suma_ważona_krawędzi_po_stronie(lewa, m) <
      suma_ważona_krawędzi_po_stronie(prawa, m) )
    return w_lewo
  return w_prawo
}

Mapa aktualizuj_mapę_globalną(Mapa globalna, Mapa lokalna)
{
  int TTL=4
  for( Krawędź k IN zbiór_krawędzi(lokalna) )
    dodaj_krawędź(globalna, k, TTL)
  for( Krawędź k IN zbiór_krawędzi(globalna) )
  {
    zmniejsz_ttl(k)
    if( pobierz_ttl(k)==0 )
      usuń_krawędź(globalna, k)
  }
  return globalna
}

// Główna pętla sterująca:
Kierunek k=do_przodu
Mapa globalna=EMPTY
while( wyjdź_z_programu()==false )
{
  Mapa lokalna=system_wizyjny_wyznacz_mapę_lokalną()
  globalna=aktualizuj_mapę_globalną(globalna, lokalna)
  k=wybierz_kierunek_ruchu(k, globalna)
  robot_jedź_w_kierunku(k)
}
```

# Rozdział 3

# Eksperymenty weryfikujące poprawność i efektywność zaproponowanego rozwiązania

Poniższy rozdział został poświęcony głównie opisowi eksperymentów jakie zostały przeprowadzone celem sprawdzenia zachowania robota zarówno w typowych jak i wyjątkowych sytuacjach.

Na wstępie, pokrótce, została przybliżona implementacja oraz środowisko uruchomieniowe. Potem omówione są kolejne eksperymenty, jakie przeprowadzono. Każdy z nich jest podsumowany komentarzem wyjaśniającym wyniki.

## 3.1. Implementacja systemu

Na potrzeby testów prezentowanego w rozdz. 2 systemu powstała dedykowana platforma sprzętowa – robot gąsienicowy TIER. Robot jest sterowany zdalnie za pośrednictwem modemów radiowych. Całość przetwarzania informacji ma miejsce na komputerze klasy PC. Elektronika robota przetwarza jedynie proste rozkazy sterujące jazdą w konkretnym kierunku[1].

Całość systemu omawianego w rozdz. 2 została zaimplementowana na darmowej platformie systemowej GNU/Linux [7][8] przy użyciu narzędzi z pakietu gcc [22] oraz edytora vim [23].

Architektura oprogramowania umożliwia pobieranie kolejnych klatek z kamery za pośrednictwem dowolnej sieci TCP/IP (o odpowiedniej przepustowości)[2], gdyż całość jest podzielona na dwie części – serwer udostępniający pojedynczy (bieżący) obraz przechwycony na żądanie z kamery oraz część sterującą wysyłającą zapytania do serwera i przetwarzającą odpowiedzi (obrazy rastrowe).

Ponieważ program przetwarza duże ilości danych w czasie rzeczywistym wymaga więc dość silnej konfiguracji sprzętowej do pracy – testy przeprowadzano na komputerze $P4/2.4GHz$ z $512MB$ pamięci RAM.

Oprogramowanie sterujące komunikuje się z robotem (z punktu widzenia kodu) bezpośrednio za pomocą portu LPT w trybie „surowym" – program wymaga więc do poprawnego działania uprawnień administratora.

1. Więcej szczegółów odnośnie budowy i działania części sprzętowej można znaleźć w dod. A.
2. Więcej informacji na temat mechanizmów komunikacji w systemie można znaleźć w podroz. A.3.

## 3.2. Przeprowadzone eksperymenty

Zadaniem robota jest samodzielne poruszanie się w zamkniętym środowisku (ang. *indoor environment*) tak więc i testy zostały pomyślane pod kątem sprawdzenia zachowania się robota w typowych przypadkach. Kolejno zostanie sprawdzona jego zdolność do:

1. wykrywania statycznych przeszkód na swej drodze
2. wykrywania i reagowania na przeszkody dynamiczne
3. ruchu swobodnego w statycznym środowisku
4. ruchu swobodnego w dynamicznym środowisku
5. poruszania się w ciasnym środowisku
6. niwelowania wpływ oświetlenia na pracę systemu sterującego

Po sprawdzeniu w/w sytuacji znane będzie zachowanie się robota w różnych przypadkach na jakie może on trafić w trakcie ruchu. Zostanie również wyjaśnionych kilka zachowań jakie zostały zaobserwowane wraz z ewentualnymi metodami przeciwdziałania niekorzystnym zjawiskom.

### Jazda na wprost w statycznym środowisku

Aby zweryfikować czy robot poprwanie wykrywa przeszkody w terenie zastosowano uproszczoną wersję systemu sterującego, który nie wykonuje skrętów. Robot jechał przed siebie aż do napotkania przeszkody, po czym się zatrzymywał. Dla uproszczenia badań środowisko robota było statyczne (jedynym ruchomym elementem był sam robot).

Z zadaniem tym robot radził sobie doskonale – zawsze zatrzymywał się w ustalonej odległości od przeszkody i tam stał. Dzięki zastosowaniu „czasu życia" obserwowanych krawędzi nie zdarzały się sytuacje, kiedy robot stał aby zaraz potem na ułamek sekundy ruszyć, po czym znów się zatrzymać wykonując swoiste „doskoki" do przeszkody. Zgodnie z oczekiwaniami, wyłączenie czasu życia krawędzi ($TTL = 0$) kończyło się występowaniem tego efektu, gdyż decyzja była wtedy zależna wyłącznie od bieżącej segmentacji obrazu, podatnej na losowe przekłamania.

### Jazda na wprost w środowisku zmiennym

Po sprawdzeniu poprawności reakcji robota na środowisko statyczne, kolejnym krokiem było zbadanie, jak się zachowa przy takich samych założeniach jak poprzednio ale dla zmiennego w czasie środowiska (przeszkody mogą się pojawiać i znikać).

Aby przeprowadzić taki test robot przemieszczał się po długim pomieszczeniu, gdzie co chwilę człowiek zastępował mu drogę uniemożliwiając przejazd.

Za każdym razem robot zatrzymywał się przed nowo napotkaną przeszkodą oraz ruszał przed siebie w przypadku usunięcia się przeszkody z drogi.

Choć podejmowane decyzje były poprawne, zastosowanie „czasu życia krawędzi" odbiło się negatywnie na czasie reakcji robota w przypadku nagłego usunięcia się przeszkody z drogi – po zastąpieniu drogi, zatrzymanie następowało niemal natychmiast, zaś po jej zwolnieniu robot ponownie ruszał po czasie około $1s$. Był to czas potrzebny na usunięcie z mapy globalnej krawędzi, które jeszcze przed chwilą były obserwowane.

Przypadek jazdy na wprost w zmiennym środowisku prezentuje jeden z filmów na załączonej do pracy płycie CD-ROM.

### Omijanie przeszkód statycznych

Po podstawowych testach jazdy na wprost w środowisku statycznym jak i zmiennym włączono mechanizm sterowania powodujący skręcanie w celu ominięcia napotkanej przeszkody (podroz. 2.3) i pozwolono robotowi na swobodne poruszanie się w zamkniętym pomieszczeniu o powierzchni około $15m^2$.

Robot radził sobie dobrze również i w tym przypadku, choć niektóre z podejmowanych przezeń decyzji jednoznacznie wskazywały, iż mimo stosowanych heurystyk, wykonywane ruchy nadal są bardzo naiwne – gdyby miał on do dyspozycji mapę terenu, budowaną w trakcie poruszania się po nim oraz globalny algorytm planowania trasy, decyzje byłby z całą pewnością bardziej trafne.

Podczas jazdy okazjonalnie dokładano również pewne przeszkody terenowe. Przeszkody te były statyczne oraz umieszczane w momencie kiedy robot nie mógł ich „widzieć". Ze względu na budowę systemu powodowało, iż nie robiło to żadnej różnicy z punktu widzenia założeń eksperymentu.

Podczas testów pozwalano robotowi jeździć do kilkunastu minut nie obserwując przy tym kolizji z obiektami środowiska. Film pokazujący bezkolizyjną jazdę robota po pomieszczeniu został zawarty na załączonej do pracy płycie CD-ROM.

### Omijanie przeszkód w ruchu

Eksperyment ten jest fuzją dwóch prezentowanych wcześniej: jazdy na wprost w środowisku zmiennym oraz swobodnego poruszania się w środowisku statycznym.

Ponieważ robot w żaden sposób nie buduje modelu zachowania się ruchomych elementów otoczenia, nie jest on w stanie uniknąć kolizji w przypadku, kiedy obiekt manewruje szybciej od niego oraz „chce" w niego uderzyć.

Jeżeli jednak założyć, iż inne obiekty pojawiające się w zasięgu nie będą dążyć do kolizji robot, TIER również się z nimi nie zderzy. Dzieje się tak ponieważ w przypadku napotkania przeszkody zatrzymuje się on w miejscu i próbuje skręcić (w prawdopodobnie „czystszą" stronę) aby znaleźć kierunek, w którym może się poruszać bezkolizyjnie.

Praktyka pokazała, iż chodzenie po pokoju, w którym jeździ robot, nie powoduje problemów, gdyż traktuje on napotkaną osobę jako przeszkodę i próbuje ją zawsze ominąć.

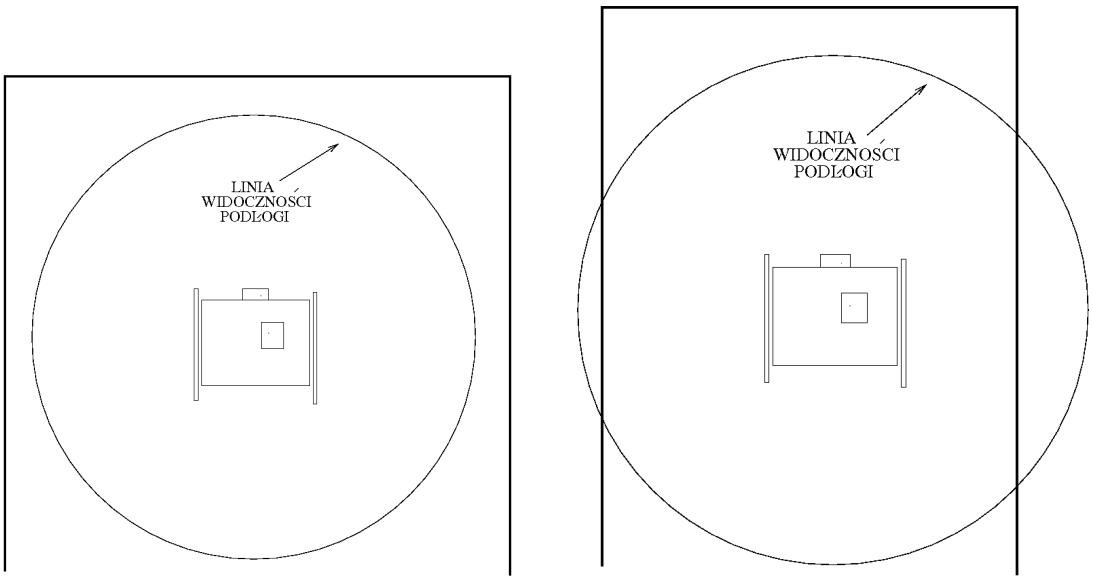
Na koniec trzeba wspomnieć, iż podczas określania kolizji należy mieć na uwadze fakt ograniczonego pola widzenia robota – jeżeli ruchoma przeszkoda nadejdzie z boku maszyna fizycznie nie będzie miała możliwości jej zauważenia a, co za tym idzie, odpowiedniego zareagowania.

### Manewrowanie w ciasnym środowisku

Choć swobodna jazda po otoczeniu nie sprawiała robotowi trudności, problemy zaczęły się pojawiać przy manewrowaniu w ciasnym środowisku (zawierającym dużo przeszkód na małym obszarze). System stwierdza czy nastąpi kolizja z zaobserwowaną przeszkodą na podstawie analizy jego wymiarów względem przerwy między przeszkodami, w którą ma wjechać. Ustalono także pewien margines błędu związany z niedokładnościami pomiarowymi oraz faktem, iż robot nie jest w stanie zakręcić idealnie w miejscu (w przypadku ogólnym może się zdażyć, iż jedna z gąsienic będzie się poruszać

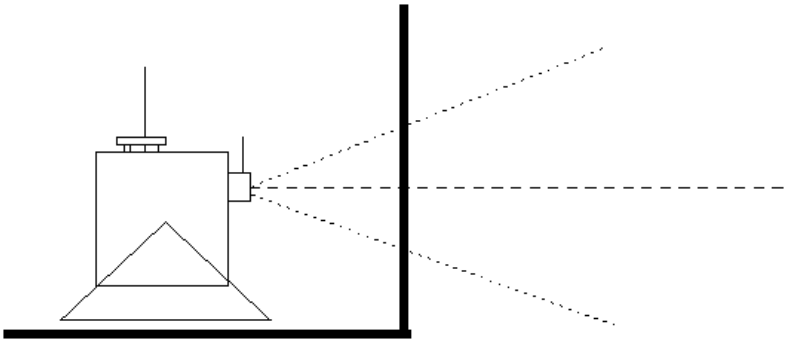
po obszarze o mniejszym tarciu niż druga). Za ów margines przyjęto połowę przekątnej robota (najszerszy jego wymiar) plus kilkanaście centymetrów dla bezpieczeństwa.

Choć sprawdzało się to w pewnych przypadkach, okazywało się być zgubne kiedy robot musiał zawrócić po wjechaniu w ciasny zaułek w kształcie litery „U" – po stwierdzeniu, iż dalej nie da się jechać zaczynał on skręcać w bok. Kiedy jednak ustawił się juz pod pewnym kątem okazywało się, iż ze względu na minimalną odległość robota od gładkiej ściany przestawał on widzieć krawędź między nią a podłogą. Segmentacja obrazu natychmiast zwracała informację, iż cały obszar obrazu jest jedną powierzchnią więc z punktu widzenia systemu wizyjnego wyglądało to, jakby droga była pusta, robot zaprzestawał więc skręcania i zaczynał jechać wprost na ścianę. Sytuację tą ilustruje rys. 3.1. Przypadek kiedy robot po skręceniu widzi ścianę bez krawędzi prezentuje rys. 3.2.




(a) (b)

Rys. 3.1. Poruszanie się robota w ciasnym korytarzu: (a) jest dość miejsca; (b) obrót spowoduje wyciągnięcie błędnych decyzji i kolizję ze ścianą.



Rys. 3.2. Problem z systemem wizyjnym przy nadmiernym zbliżeniu się do przeszkody.

Jedynym sposobem, aby programowo zapobiec temu było ustalenie takiego marginesu bezpiecznej odległości od przeszkody, by nawet po zakręceniu nic nie mogło się znaleźć bliżej niż linia, od której robot widzi podłogę (odległość $d$ na rys. 2.22). Odpowiada to przypadkowi (a) z rys. 3.1.

W praktyce oznaczało to przyjęcie promienia równego około $65cm$. Znacząco ogranicza to możliwości poruszania się w typowym, ciasnym środowisku, gdyż „bezpieczna" przerwa pomiędzy przeszkodami musi wynosić ponad $130cm$, co przykładowo uniemożliwia przejazd robota przez typowej wielkości drzwi domowe (o średnicy $80cm$).

**Wpływ oświetlenia**

Przy okazji omawiania systemu wizyjnego, w podroz. 2.2.2 wspomniana była procedura korekcji jasności obrazu poprzez rozciąganie histogramu. Zdaje ona dobrze egzamin w przypadku, kiedy jasność jest zbyt słaba bądź też zbyt silna. W realnym środowisku znacznie częściej spotykanym problemem są cienie. Ze względu na brak informacji o źródłach światła i generowanych cieniach, nie sposób jest odróżnić ostrej krawędzi przeszkody na dywanie od cienia jakiegoś gładkiego obiektu.

Efektem tego jest zła segmentacja, pociągająca za sobą błędną detekcję przeszkód i zbyteczną próbę ominięcia czegoś co nie istnieje (w skrajnym przypadku może to spowodować mylne uznanie danego obszaru za nieprzejezdny).

Przykład takiej sytuacji pokazuje rys. 3.3. Prezentowany system wizyjny, niestety, nie radzi sobie z tego typu sytuacjami optycznymi. Aby poprawić tę sytuację należałoby gruntownie go przebudować, dodając pewną wiedzę o świecie, lub potencjalnie dodać inny rodzaj czujników weryfikujących, czy podana segmentacja jest poprawna.

Rys. 3.3. Przykładowa sytuacja kiedy gwałtowna zmiana natężenia światła wygląda jak krawędź powodując niepoprawną segmentację.

# Rozdział 4

# Podsumowanie

W niniejszym rozdziale omówione zostały wyniki jakie udało się uzyskać dzięki prezentowanemu podejściu. Poddane są one również zweryfikowane pod kątem początkowych planów. W końcowej części rozdziału poruszona jest kwestia kierunków dalszych prac, jakie warto podjąć.

## 4.1. Wyniki prac

Jako podstawowy cel badań postawiono przygotowanie i praktyczną implementację systemu robotycznego mającego zapewniać samodzielne, bezkolizyjne poruszanie się w nieznanym apriori terenie. Ze względu na specyfikę systemu wizyjnego przyjęto pewne założenia odnośnie środowiska, w jakim robot będzie się poruszał. Uzyskanie takiego poziomu zaawansowania jest dobrym punktem wyjściowym dla dalszych, bardziej wyszukanych funkcji jakie robot może pełnić.

W niniejszej pracy zaprezentowano dość nietypowe podejście do analizy głębi nieznanego apriori otoczenia za pomocą systemu monowizyjnego – autor nie spotkał się z żadną publikacją gdzie podobny system byłby omawiany. Choć stosowanie jednokamerowych systemów wizyjnych w robotyce przemysłowej jest bardzo powszechne (stosuje się tam również sieci neuronowe [50]) używane są one w znanych z góry środowiskach.

Do celów testów został zbudowany robot mobilny (jego dokładna budowa jest opisana w dod. A), na którym testowano oprogramowanie napisane na podstawie opracowanego podejścia (rozdz. 2). Jako podsumowanie całości prac opisane zostały przeprowadzone praktyczne próby oraz wyciągnięto wnioski odnośnie dalszych kierunków, jakie warto wypróbować, lub też należałoby porzucić jako nierokujące sukcesów.

Podczas praktycznych testów skonstruowanego systemu robotycznego jako całości, napotkano na trudności z wydajnością niektórych ze stosowanych metod. Głównym problemem była sieć neuronowa pochłaniająca około $3/4$ całego czasu przewidzianego na pełny cykl analizy. Z powodu braku dostępu do klastra obliczeniowego, będącego w stanie przetworzyć odpowiednią ilość danych w czasie rzeczywistym, konieczne było zrezygnowanie ze stosowania tego podejścia.

Z testów, jakie przeprowadzono wynika, iż robot dobrze sobie radzi z samodzielnym poruszaniem się w płaskim terenie i nie zderza się z przeszkodami statycznymi ani

dynamicznymi. Problemem są ciasne przejścia pomiędzy elementami otoczenia – w przypadku napotkania takowego robot stwierdza, iż jest zbyt ciasno aby mógł tam wjechać i bezpiecznie się poruszać (podroz. 3.2).

Choć udało się zrobić dużo, system nadal ma ograniczone możliwości działania – aby wykonywać większość terenowych prac potrzebna jest wiedza o otoczeniu (mapa). Dodatkowo, aby nie ograniczać robota do przewidzianego z góry środowiska, mapa otoczenia musiałaby być tworzona dynamicznie.

Posiadając odczyty z czujników na tyle dokładne, aby umożliwić tworzenie mapy globalnej na podstawie większej niż 4-5 liczby klatek obrazu możnaby wiernie odwzorowywać środowisko i planować trasę przemieszczania się w nim ze znacznym wyprzedzeniem. Poświęcono tym zagadnieniom kilka ciekawych prac, z czego większość omawia algorytmy „anytime”[1], bardzo praktyczne w przypadku robotyki [42][46][61].

Efekty uzyskane podczas przeprowadzonych badań stanowią dobrą podstawę pod rozbudowę oraz dalsze prace w dziedzinie robotyki mobilnej. Opracowana metoda odczytywania głębi z obrazów mono-wizyjnych posiada także pewną zaletę nad macierzą czujników odległości – nie jest ona tak bardzo kierunkowa dzięki czemu potrafi „zauważyć” drobne przeszkody znajdujące się tuż przy ziemi. Wykrycie takiego obiektu za pomocą zwykłego czujnika odległości byłoby trudne, lub wręcz niemożliwe.

Na obecnym poziomie zaawansowania prac możnaby zastosować zaproponowany system (po drobnych przeróbkach programowo-sprzętowych) jako samobieżny odkurzacz domowy. Rozwiązania tego typu stają się ostatnio bardzo popularne, ze względu na swoją praktyczność – moboty sprzątające można kupić na polskim runku począwszy od kilkuset złotych (stan na początek 2007 r.) [25][26]. Zdjęcie przykładowego robota przeznaczonego do tego typu zastosowań domowych przedstawiono na rys. 4.1.

Rys. 4.1. Zdjęcie przykładowego robota samodzielnie odkurzającego pomieszczenia (zaczerpnięte z [24]).

1. Algorytmy „anytime” to klasa algorytmów, w których istnieje możliwość pobrania rozwiązania nie w pełni gotowego, ale za to w dowolnej chwili czasowej. W miarę wydłużania czasu pracy algorytmu możemy uzyskać dokładniejsze przybliżenie rzeczywistego rozwiązania.

## 4.2. Dalsze prace

Podrozdział ten prezentuje, pokrótce, kierunki jakie autor uznał za obiecujące. Wprowadzenie ich do istniejącego systemu, choć byłoby czasochłonne, znacznie podniosłoby jego możliwości oraz rozszerzyło zakres potencjalnych zastosowań.

### Ruch w „ciasnym" środowisku

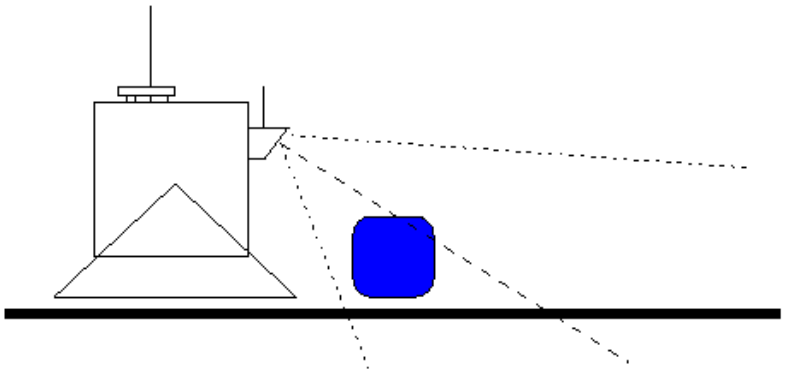
W podroz. 3.2 omówiono wyniki uzyskane dla ciasnego środowiska w jakim robot się poruszał podczas prowadzonych testów. Rezultaty okazały się być bardzo słabe, ze względu na odległość, jaka dzieliła robota od pierwszego punktu na podłodze widocznego na obrazie z kamery.

Praktyczna odległość (promień) do poruszania się przez robota była więc ograniczona do przestrzeni na której widać było podłogę. Aby poprawić tę sytuację należałoby umożliwić robotowi widzenie podłogi dużo wcześniej – w sytuacji idealnej punkt ten znajdowałby się tuż przy początku robota. Można to osiągnąć na dwa sposoby:

1. kamera musiałaby mieć bardzo duży promień widzenia, umożliwiając postrzeganie przedmiotów będących poza zasięgiem kamery zastosowanej w projekcie, zachowując przy tym geometrię (utrata tejże informacji wymuszałaby dodanie kolejnej transformacji „prostującej" przechwycony obraz).
2. kamera musiałaby być skierowana nieco w dół, aby widziała podłogę bliżej.

Pierwsze rozwiązanie jest trudniejsze i bardziej kosztowne od drugiego, więc nie będzie ono dalej omawiane.

Drugie z podejść wymagałoby relatywnie niskiego nakładu – należałoby skonstruować mocowanie kamery, umożliwiające umiejscowienie jej pod odpowiednim kątem względem pionu oraz zmodyfikować sposób wyliczania głębi tak aby uwzględniała tenże fakt. Sytuację tę widać na rys. 4.2 – warto porównać go z rys. A.17 oraz rys. 3.2: różnica w wykrywalności przeszkód jest znacząca.



Rys. 4.2. Pole widzenia kamery skierowanej pod pewnym kątem względem pionu.

Pewnym problemem mogłoby być przy takim podejściu widzenie przez robota części samego siebie (wysuniętych bardziej z przodu). Można by problem ten rozwiązać dwojako:

1. jeżeli nie pogorszyłoby to efektów uzyskanych poprzez zwiększenie kąta odchylenia kamery od pionu, można by ów kąt nieco zmniejszyć, tak aby wystające elementy znalazły się poza polem widzenia.

2. można by „wycinać" pewne fragmenty obrazu programowo tak aby wyższe warstwy nie widziały ich już na swoim wejściu.

### Dobór czujników

Przy kolejnych projektach znacznie większą uwagę należy poświęcić problemowi doboru czujników, w jakie robot ma zostać wyposażony. Choć kamera posiada dużą rozdzielczość i przekazuje dzięki temu znaczne ilości informacji, nie nadaje się ona dobrze do każdego zastosowania.

Lepszym wyborem byłaby para kamer tworzących system stereowizyjny [53], potencjalnie wzbogacona jeszcze o inne czujniki dedykowane do pewnych zastosowań, jak na przykład ultradźwiękowe czujniki odległości.

Posiadanie systemu składającego się z kilku różnych źródeł informacji działających na innych zasadach fizycznych daje większą pewność informacji niż dane z tylko jednego systemu. Przykładową sytuację „wykrycia" fałszywej krawędzi, jaką pokazuje rys. 3.3, możnaby zweryfikować mając odczyt z czujnika odległości, który by zanegował możliwość istnienia ściany na drodze robota w spodziewanej odległości.

### Szumy i komunikacja międzywarstwowa

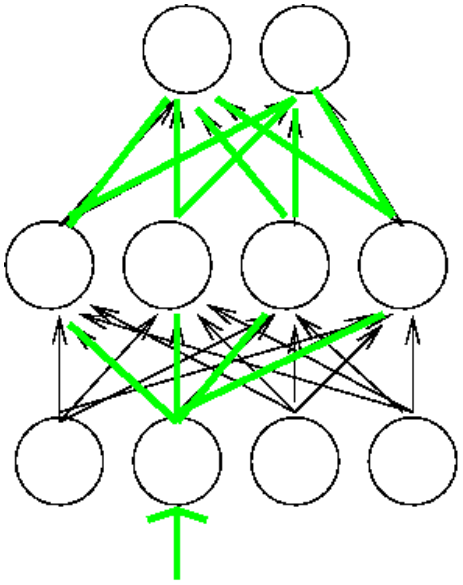
Dużym problemem przy realizacji przetwarzania danych z rzeczywistych czujników okazały się być szumy i drobne zakłócenia – kiedy pojawiały się one na najniższych warstwach systemu przetwarzającego, mających pośrednio wpływ na wszystkie dalsze analizy, błędy z nich wynikające zaczynały się kumulować i zwiększać w miarę jak proces przetwarzania przechodził do wyższych warstw.

Aby zapobiec propagowaniu się błędów należałoby wykrywać je jak najszybciej i korygować na podstawie innych informacji posiadanych przez system. Należałoby wprowadzić tu system warstwowy, jak zaprezentowano to na rys. 2.4 jednak zapewniający komunikację obustronną między warstwami, nie zaś jednostronną, jak to zrealizowano: warstwy niższe dostarczałyby danych do analizy dla warstw wyższych (tak jak do tej pory) otrzymując jednocześnie od nich „wsparcie" w postaci informacji o tym, co prawdopodobnie należałoby poprawić, lub też pod jakim kątem analizować dane.

Przykładowo, dla omawianego przypadku konieczności wprowadzenia czasu życia („TTL") dla krawędzi na mapie globalnej warstwa odpowiadająca za tworzenie mapy mogłaby przesyłać „w dół" hierarchii informację, iż w określonych miejscach obrazu znaleziono przeszkodę, więc należy się jej również spodziewać przy kolejnej segmentacji, prawdopodobnie nie różniącej się znacznie od tej, dla bieżącego obrazu wejściowego.

Do realizacji takiego systemu z połączeniami między-warstwowymi doskonale nadają się sieci neuronowe. Ze względu na swoją specyfikę działania w sposób naturalny realizują one pełne połączenie pomiędzy wszystkimi etapami analizy (połączenie każdy-z-każdym). W sieci, w której neurony są ze sobą połączone w sposób pełny informacja jest rozproszona po całym systemie jednocześnie. Dla sieci warstwowych połączenie jest bliskie zupełnemu, ponieważ informacja z niższych warstw jest propagowana do warstw wyższych (połączenie jest więc pośrednie). Propagację informacji widać na rys. 4.3.

Ciekawy przykład zastosowania sieci neuronowej do sterowania mobotem zaprezentowano w [50], gdzie sztuczna sieć neuronowa samodzielnie kierowała robotem kołowym omijając przeszkody napotykane na drodze w laboratorium.



Rys. 4.3. Propagacja informacji w sieci neuronowej (pogrubione, zielone linie).

### Autonomia robota

Warto również poświęcić czas na przedstawienie kwestii autonomiczności robota mobilnego. Robot autonomiczny nie ma potrzeby komunikowania się z zewnętrznym komputerem wykonującym na jego rzecz obliczeń. Taka komunikacja wymaga przesyłania ogromnych (w przypadku systemu wizyjnego) ilości danych, co staje się dość znaczącym czynnikiem czasowym. Gdyby całość tej komunikacji odbywała się wewnątrz pojedynczej jednostki, czas ten możnaby zupełnie wyeliminować.

Z drugiej jednak strony umieszczenie komputera o podobnych możliwościach obliczeniowych co maszyna „stacjonarna" na platformie mobilnej stawia duże wymagania odnośnie zasilania i zajmowanego miejsca na docelowym robocie.

### Sprzętowy system wizyjny

Ciekawym projektem byłoby zrealizowanie systemu wizyjnego opartego (przynajmniej w części najbardziej niskopoziomowej) o rozwiązania czysto sprzętowe: obraz bezpośrednio z kamery mógłby być przetwarzany przez prosty automat realizujący podstawowe filtracje (wyrównywanie histogramu, poprawa ostrości, etc…) później zaś dane przekazywane byłby bezpośrednio do karty realizującej sprzętowo sieć neuronową komórkową.

Prostszą (łatwą do realizacji, korzystając wyłącznie z istniejących elementów) wersją powyższego podejścia byłoby pobieranie obrazu z kamery, podstawowa obróbka na komputerze i przekazanie takiego rastra z powrotem do sprzętowej sieci CNN.

# Bibliografia


[1] *Jak to jest?* Przegląd Reader's Digest, 1996.

[2] *http://www.abc.net.au/science/news/stories/s414220*. 2005.11.10.

[3] *http://www.cadsoft.de*. 2006.04.02.

[4] *http://www.aries-rs.com.pl/produkt/arm/arm2.html*. 2006.06.06.

[5] *http://people.cs.uchicago.edu/ pff/segment/*. 2006.08.25.

[6] *http://www.allegro.pl/item122322557_mikro_kamera_b*. 2006.09.04.

[7] *http://debian.org*. 2006.10.01.

[8] *http://suse.de*. 2006.10.01.

[9] *http://felektr.katalogi.pl/temat9309/*. 2006.10.12.

[10] *http://lab.analogic.sztaki.hu/Candy/csl.html*. 2006.11.17.

[11] *http://server.eletel.p.lodz.pl/tele/crit2/index.html*. 2006.12.14.

[12] *http://www.avaye.com/index.php/neuralnets/applied/hardware*. 2006.12.14.

[13] *http://www.cs.york.ac.uk/arch/NeuralNetworks/cmm.html*. 2006.12.14.

[14] *http://www.modulusfe.com/nnpc12/*. 2006.12.14.

[15] *http://www.neuricam.com/assets/images_prd/TotemPC104.jpg*. 2006.12.14.

[16] *http://www.particle.kth.se/ lindsey/nnwAtm.html*. 2006.12.14.

[17] *http://www.atmel.com*. 2007.03.04.

[18] *http://www.atmel.com/atmel/acrobat/doc0265.pdf*. 2007.03.04.

[19] *http://www.ce.unipr.it/pardis/CNN/cnn_cont.gif*. 2007.03.11.

[20] *http://www.isi.ee.ethz.ch/ haenggi/CNN_web/CNN_figures/network.gif*. 2007.03.11.

[21] *http://www.caip.rutgers.edu/ ksen/Snaps/dv/empty_road.jpg*. 2007.03.15.

[22] *http://gcc.gnu.org*. 2007.03.18.

[23] *http://www.vim.org*. 2007.03.18.

[24] *http://amex.waw.pl/foto/odkurzacz_rv10/odk2_r.jpg*. 2007.03.23.

[25] *http://www.allegro.pl/item174627507_odkurzacz_robot_zdalnie_sterowany_na_pilota_.html*. 2007.03.23.

[26] *http://www.allegro.pl/item178025386_odkurzacz_robot_automat_cleanmate_365.html*. 2007.03.23.

[27] *http://cordis.europa.eu/esprit/src/results/res_area/iim/iim2.htm*. 2007.03.24.

[28] *http://links999.net/robotics/robots/images/UAV-Hellfire-Missile.jpg*. 2007.03.24.

[29] *http://pdv.cs.tu-berlin.de/MARVIN/MarkII-Pics/Heli1_flying.jpg*. 2007.03.24.

[30] *http://static.howstuffworks.com/gif/military-robot-1.jpg*. 2007.03.24.

[31] *http://ucsdnews.ucsd.edu/graphics/images/2004/UnderWaterRobot_lg.jpg*. 2007.03.24.

[32] *http://www.cim.mcgill.ca/ yiannis/Photography/Research/TetherlessAqua.JPG*. 2007.03.24.

[33] *http://www.cyclingnews.com/photos/2005/tech/shows/interbike05/interbike056/sram-robot-1.jpg*. 2007.03.24.

[34] *http://www.fsf.nl/uploadafbeeldingen/MPOKDCRI*. 2007.03.24.

[35] *http://www.news.harvard.edu/gazette/2005/08.25/99-gizmo.html*. 2007.03.24.

[36] *http://www.robonet.pl*. 2007.03.24.

[37] *http://www.demo.cs.brandeis.edu/golem*. 2007.03.25.

[38] *http://sun10.ci.pwr.wroc.pl/ cjant/projekty/khepera.html*. 2007.05.16.

[39] *http://www.konar.pwr.wroc.pl/*. 2007.05.16.

[40] I. Asimov. *Runaround*. Street & Smith, 1942.

[41] B. Ribeiro C. Silva. *Navigating mobile robots with a modular neural architecture*. Springer-Verlag, 2003.

[42] A. Stentz D. Ferguson. *The delayed D* algorithm for efficient path replanning*. Proceedings of the IEEE International Conference on Robotics and Automation, 2005.

[43] R. Dain. *Developing mobile robot wall-following algorithms using genetic programming*. Kluwer Academic Publishers, 1998.

[44] E. Davies. *Machine vision: theory, algorithms, practicalities*. Morgan Kaufmann, 2004.

[45] P. Górecki. *Układy cyfrowe - pierwsze kroki*. BTC, 2004.

[46] J. Kuffner J. Berg, D. Ferguson. *Anytime path planning and replanning in dynamic environments*. Proceedings of the IEEE International Conference on Robotics and Automation, 2006.

[47] B. Szurgot K. Kujawski. *Wyszukiwanie ścieżki na zadanej planszy*. 2005.

[48] K. Orkisz K. Lal, T. Rak. *RTLinux - system czasu rzeczywistego*. Helion, 2003.

[49] D. Kortenkamp. *Artificial intelligence and mobile robots: case studies of successful robotic systems*. MIT Press, 1998.

[50] R. Kosiński. *Sztuczne sieci neuronowe: dynamika nieliniowa i chaos*. Wydawnictwa Naukowo-Techniczne, 2002.

[51] P. Menzel. *Robo sapiens: czy roboty mogą myśleć?* G+J Gruner+Jahr Polska, 2002.

[52] P. Melin O. Castillo, L. Trujillo. *Multiple objective genetic algorithms for path-planning optimization in autonomous mobile robots*. Springer, 2006.

[53] A. Szmigielski P. Ciesielski, J. Sawoniewicz. *Elementy robotyki mobilnej*. Wydawnictwo PJWSTK, 2004.

[54] D. Parhi. *Navigation of mobile robots using a fuzzy logic controller*. Springer, 2005.

[55] T. Mitsui S. Hirose, H. Ohno. *Design of in-pipe inspection vehicles for* $\phi 25$*,* $\phi 50$*,* $\phi 150$ *pipes*. IEEE International Conference on Robotics & Automation, 1999.

[56] A. Saffiotti. *The uses of fuzzy logic in autonomous robot navigation*. 1997.

[57] M. Sportack. *Sieci komputerowe – księga eksperta*. Helion, 1999.

[58] T. Starecki. *Mikrokontrolery 8051 w praktyce*. BTC, 2004.

[59] R. Stephens. *Algorytmy i struktury danych z przykładami w Delphi*. Helion, 2000.

[60] B. Szurgot. *Wykrywanie obiektów na obrazach z kamery analogowej*. 2006.

[61] A. Hilman U. Roth, M. Walker. *Dynamic path planning with spiking neural networks*. 1997.

[62] J. Watson. *Elektronika*. WKŁ, 1999.

# Dodatek A

# Budowa robota

Dodatek ten poświęcono całkowicie bardzo rozbudowanemu zagadnieniu, jakim jest budowa robota mobilnego. Ze względu na tematykę niniejszej pracy rozdział ten został miejscami potraktowany bardzo skrótowo, a wiele szczegółów pominiętych. Mimo przyjętych uproszczeń, dodatek ten nadal jest dość rozległy, dając wyobrażenie jak skomplikowanym i czasochłonnym zadaniem jest stworzenie rzeczywistej platformy mobilnej.

## A.1. Stawiane cele

Podstawowym celem stawianym przed robotem jest samodzielne przemieszczanie się. Ważnym aspektem jest określenie środowiska, w jakim „przemieszczanie się" będzie miało miejsce – do wyboru jest cała gama: począwszy od pomieszczeń zamkniętych, przez teren otwarty, na wodzie skończywszy. W niniejszej pracy wybór padł na pomieszczenia zamknięte, ze względu na ich przystępność (a co za tym idzie łatwość testowania) i znaczne uproszczenie wymagań konstrukcyjnych (takich jak chociażby wodoodporność).

Mając środowisko, kolejnym krokiem jest zdecydowanie się na sposób w jaki robot będzie się w nim poruszał. Spośród najbardziej popularnych rodzajów konstrukcji robotów można nadmienić roboty kroczące i jeżdżące (kołowe i gąsienicowe) [53]. Roboty kroczące nie były brane pod uwagę ze względu na stopień skomplikowania zarówno mechaniki, elektroniki, jak i samego algorytmu sterującego[1]. Choć napęd kołowy jest podobnie skomplikowany do gąsienicowego z punku widzenia mechaniki, wybór padł na gąsienice (napęd typu „czołg" [53]) jako napęd znacznie prostszy w sterowaniu[2].

Aby umożliwić łatwe tworzenie oprogramowania zachowując przy tym niskie koszta wytworzenia fizycznego robota zdecydowano się zastosować model robota mobilnego, ale nie autonomicznego – elektronika pokładowa pozwala jedynie na porozumiewanie się z zewnętrznym komputerem podejmującym decyzje na podstawie otrzymywanych odczytów z czujników, sam mikrokontroler zaś jedynie implementuje funkcje klienta protokołu komunikacyjnego oraz mechanizm obsługi silników.

---

1. Aby mieć pełną kontrolę nad każdą kończyną z osobna potrzeba po 2 serwomechanizmy [62] na każdą z nich, co pomnożone przez minimum 4 kończyny daje 8 sterowanych częstotliwościowo układów do synchronizowania w czasie rzeczywistym.
2. W przeciwieństwie do napędu kołowego (typu „samochód" [53]) umożliwia on skręcanie w miejscu.

Dysponując określonym środowiskiem oraz sposobem przemieszczania się, następnym w kolejności etapem jest wybór rodzaju czujników w jakie robot będzie wyposażony. Do najpopularniejszych spośród dostępnych w sprzedaży należą: systemy wizyjne, czujniki odległości (soniczne, podczerwone, radiowe[3]), diody reagujące na odpowiedni kolor. Mając na uwadze możliwości oraz ilość przekazywanych informacji zdecydowano się zastosować prosty system wizyjny złożony z jednej kamery bezprzewodowej.

Na koniec wprowadzenia warto zaznaczyć, iż niektóre z prezentowanych schematów są na tyle szczegółowe, iż trudno je zamieścić w postaci czytelnej na pojedynczej stronie formatu $A4$. Jeżeli czytelnik napotka trudności z odczytaniem szczegółów, wszystkie prezentowane grafiki znajdują się również na dołączonej płycie CD-ROM, skąd można je pobrać i oglądać w dowolnym powiększeniu.

## A.2. Wizja całościowa

Nim zostanie omówiona bardziej szczegółowa konstrukcja elektroniczno-mechaniczna robota przedstawiony zostanie zamysł konstrukcyjny – z jakich podstawowych elementów składa się prezentowany robot oraz jakie podejścia zastosowano.

Robot posiada dwie fizycznie niezależne drogi (kanały) komunikacyjne: jednym z nich przesyłany jest obraz wideo, drugim zaś odbywa się przesyłanie poleceń do wykonania przez maszynę.

Na pokładzie robota, oprócz odbiornika rozkazów znajduje się mikrokontroler odpowiadający za protokół komunikacyjny oraz za sterowanie silnikami elektrycznymi. Bezpośrednim sterowaniem silnikami zajmują się odpowiednie układy mocy, otrzymujące jedynie informację czy silnik ma się kręcić i, jeśli tak, to w którą stronę.

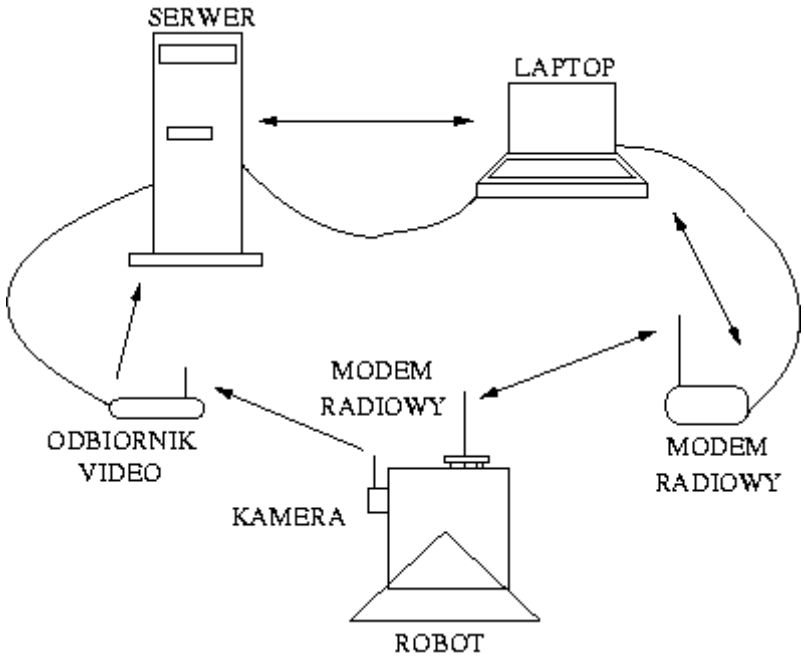
Jak widać z powyższego robot, pełni „jedynie" funkcję wykonawczą – cały program sterujący znajduje się po stronie komputera. Podejście takie daje dużą elastyczność w zakresie zastosowanych metod, powoduje jednak wyraźny narzut związany z czasem komunikacji[4].

## A.3. Schemat komunikacji

Z powodu ograniczonego budżetu robot posiada dość nietypowy system sterowania i komunikacji: Obraz z kamery jest przekazywany drogą radiową do odbiornika[5], z którego idzie drogą kablową do zainstalowanej w serwerze karty telewizyjnej. Zważywszy na znikomą moc obliczeniową wspominanego serwera, obraz jest przesyłany poprzez połączenie sieciowe do laptopa pełniącego rolę sterownika robota. Na podstawie tak otrzymanego obrazu analizowane jest otoczenie oraz podejmowana jest decyzja o dalszych ruchach (przekładana na konkretne rozkazy dla silników elektrycznych). Rozkazy są wysyłane drogą radiową (po innym kanale radiowym, niż sygnał z kamery, aby uniknąć interferencji [62]) z powrotem do robota[6] gdzie są przekładane na odpowiednie sygnały dla silników, powodujących ruch konstrukcji.

3. Konkretnie mowa tu o radarze.
4. Więcej informacji na temat komunikacji między robotem a komputerem znajduje się w podroz. A.3
5. Kamera i odbiornik stanowią autonomiczny zestaw zakupiony drogą internetową [6].
6. Transciever radiowy umożliwia komunikację obustronną a co za tym idzie – potwierdzanie odbioru danych.

Pełny przebieg komunikacji pomiędzy elementami składowymi (wraz z zaznaczonymi kierunkami transmisji) przedstawia rys. A.1.




Rys. A.1. Przebieg komunikacji robot-komputer.

## A.4. Mechanika

Całość konstrukcji nośnej robota jest metalowa, skręcana śrubami. Dzięki takiemu podejściu szkielet jest przystosowany do przenoszenia większych ciężarów[7] i jest dość wytrzymały.

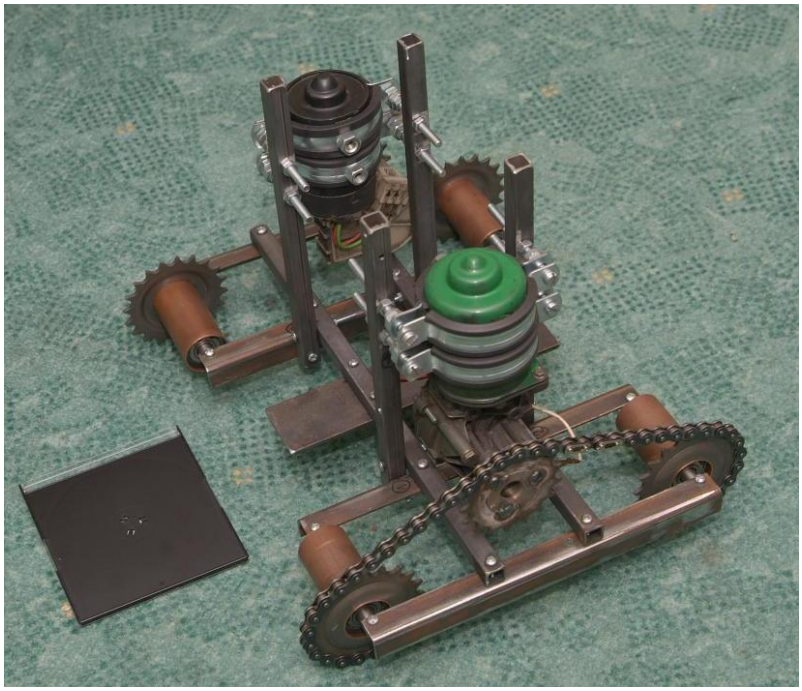
Zdjęcie na rys. A.2 zostało wykonane po założeniu pierwszej z gąsienic napędowych na zębatki, przed instalacją obsługującej elektroniki i akumulatora żelowego. Punktem odniesienia, odnośnie rozmiaru jest położone obok robota pudełko od płyty kompaktowej.

Jako element jezdny (gąsienice) zastosowano łańcuch rowerowy, obłożony na specjalnie dorabianych, łożyskowanych kółkach zębatych do przerzutek rowerowych. Koła zębate są połączone parami za pomocą kątowników, te zaś połączone poprzez dwa poprzecznie przykręcone profile zamknięte. Pośrodku profili zamkniętych, od spodu, przykręcony jest płaskownik uniemożliwiający konstrukcji rotowanie się.

Między kątowniki a profile zamknięte, łączące lewą i prawą stronę konstrukcji, przykręcone są krótsze odcinki takich samych profili, służące za mocowanie silników. Dodatkowo zastosowanie wysokich „pali" umożliwia przytwierdzenie do nich płaskiej płytki, służącej do mocowania mniejszych płytek konkretnych podzespołów elektronicznych.

Do napędzania zastosowano komutatorowe silniki prądu stałego, wraz z odpowiednim przełożeniem dającym dużą moc przy względnie niewielkich obrotach. Postawione

7. Mowa tu głównie o ważącym około $2.8kg$ akumulatorze żelowym



Rys. A.2. Metalowy szkielet (konstrukcja nośna) robota TIER z założoną jedną gąsienicą.

założenia spełniały doskonale silniki od wycieraczek samochodowych, po niewielkich przeróbkach[8]. Silniki te pracują pod napięciem $12V$, co implikuje $12V$ źródło zasilania[9].

## A.5. Elektronika

Cała elektronika pokładowa robota składa się z kilku odrębnych części, podłączonych do centralnego mikrokontrolera, odpowiadającego za sterowanie silnikami oraz komunikację ze stacjonarnym komputerem.

W tej części pracy zostanie szerzej opisana elektronika robota wraz z załączonymi schematami[10].

### A.5.1. Odsprzęganie zasilania

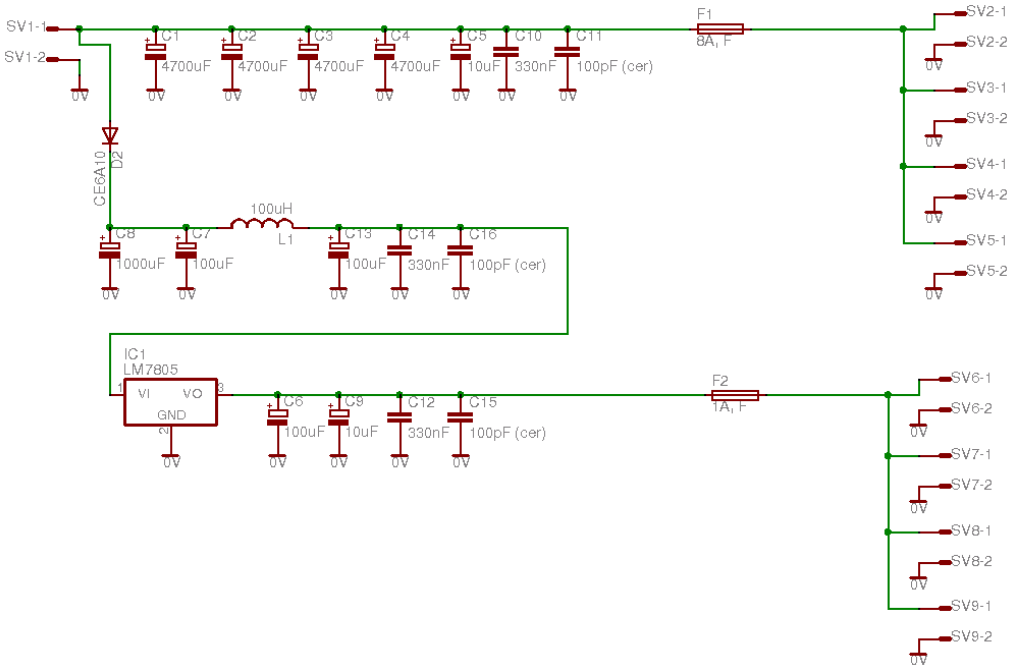
Podstawą do pracy każdego układu elektronicznego jest jego odpowiednie zasilanie. Tyczy się to przede wszystkim elektroniki sterującej (cyfrowej) – nagły spadek napięcia zasilania może spowodować zacięcie się mikrokontrolera i uniemożliwienie dalszej pracy [58][45].

Głównym źródłem zakłóceń są silniki elektryczne, które podczas pracy, pobierają około $2A$ prądu każdy. Ponieważ sterowanie ich szybkością przebiega za pomocą wypełnienia przebiegu prostokątnego (ang. *PWM – Phase Width Modulation*), załączającego i

8. Wymagały one jedynie nieznacznego przerobienia połączeń klocków komutatora oraz zamocowania zębatek rowerowych celem przełożenia napędu na gąsienice.
9. Zastosowano tu wspominany wcześniej akumulator żelowy od motocykla.
10. Wszystkie przedstawione schematy wykonano przy pomocy programu Eagle [3] w wersji na system operacyjny GNU Linux [8].

wyłączającego układy nawrotne, powoduje to częste „skoki" prądu i napięcia w całym układzie. Aby temu możliwie skutecznie zapobiec, zasilanie silników jest odsprzęgane przez kilka kondensatorów elektrolitycznych o dużej pojemności, oraz kilka mniejszych, znacznie szybszych (w sensie czasu potrzebnego na naładowanie/rozładowanie), kondensatorów ceramicznych. Zasilanie elektroniki sterującej odbywa się poprzez stabilizator scalony $5V$, otoczony zespołami filtrów[11], zarówno po stronie wejściowej, jak i wyjściowej.

Na rys. A.3 oraz rys. A.4 zamieszczono kolejno schemat ideowy układu oraz płytkę z zaznaczonymi elementami.




Rys. A.3. Schemat ideowy płytki układu odsprzęgania zasilania.

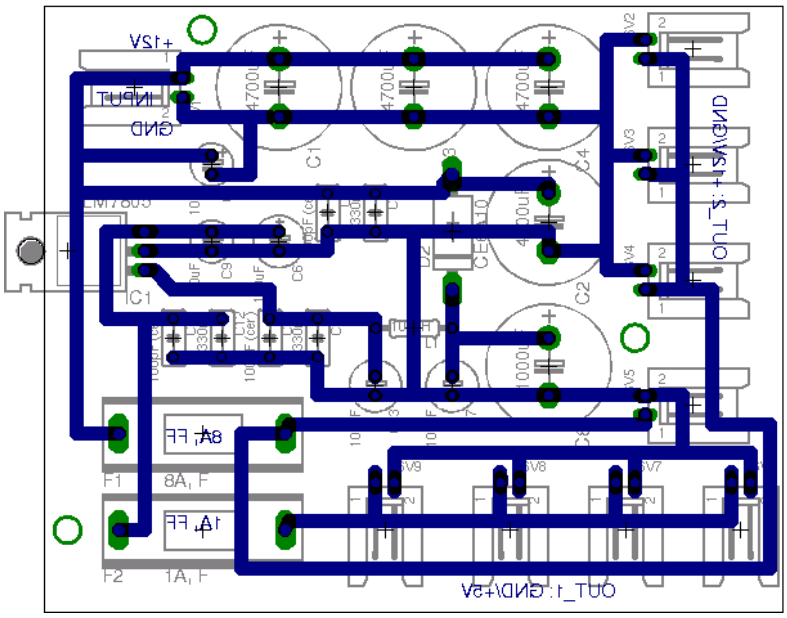
### A.5.2. Układy nawrotne

Aby umożliwić gąsienicom niezależne obracanie się w obu kierunkach, wymagany jest układ nawrotny, mogący przebiegunowywać silniki zgodnie z zapotrzebowaniem. Aby uprościć konstrukcję zdecydowano się zastosować dwa niezależne układy (po jednym na każde z wyprowadzeń pojedynczego silnika). Dzięki takiemu podejściu sterowanie można prosto sprowadzić do „wystawiania" sygnałów binarnych (2-bitowych) na odpowiednie wejścia układów tak, że odpowiednie stany będą odpowiadały za odpowiednie kierunki pracy (obrotów). Schemat ideowy takiego podsystemu (tzn: silnik + 2 sterowniki + mikrokontroler) przedstawia rys. A.5.

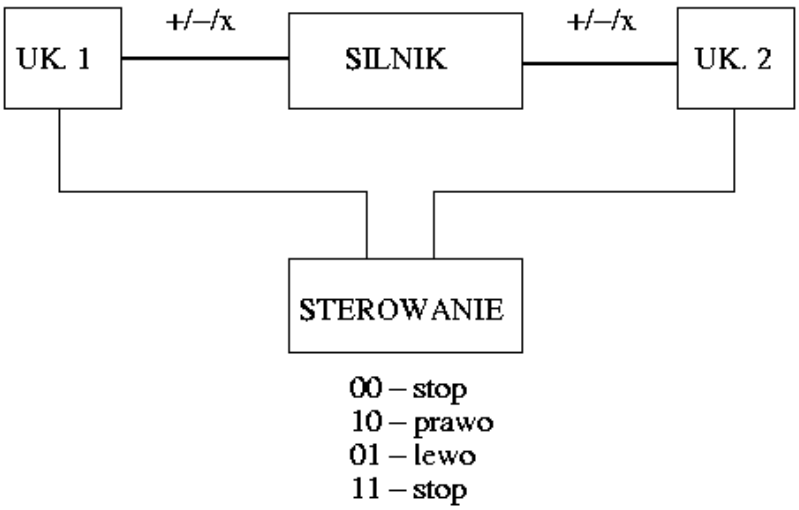
Jako realizację tegoż pomysłu skonstruowano układ dający na wyjściu mocy napięcie bliskie napięciu zasilania ($12V$) lub masie ($0V$) w zależności od wybranego zewnętrznie trybu pracy [9].

Jako elementy wykonawcze zastosowano parę komplementarnych tranzystorów mocy BD911 oraz BD912 sterowanych za pomocą tranzystorów małosygnałowych. Ze względu

11. Kondensatorów filtrujących.



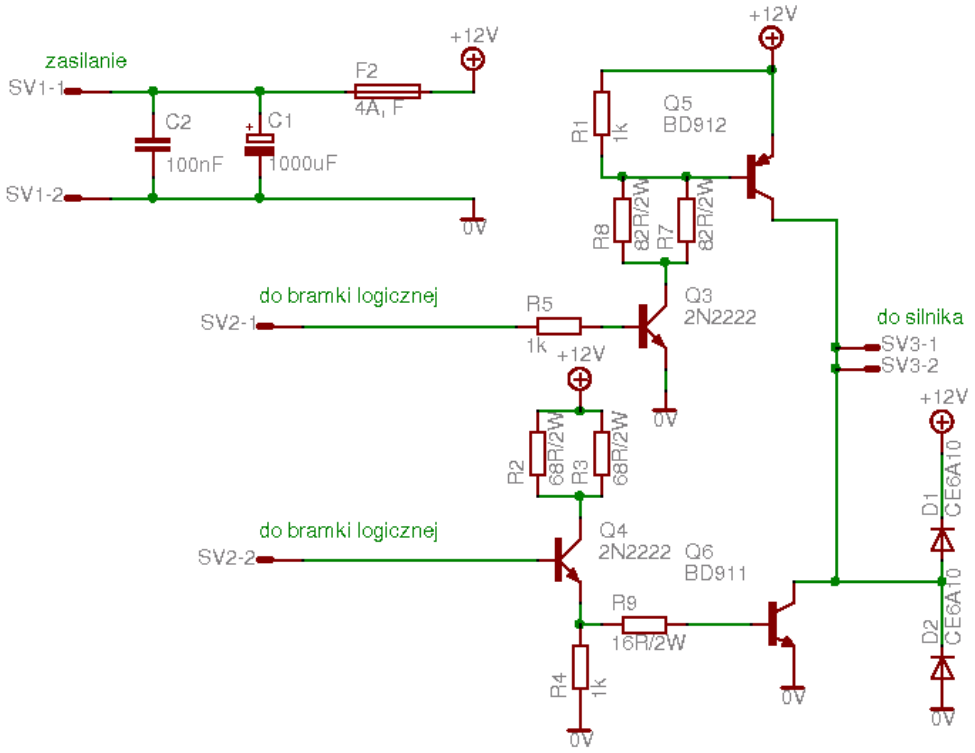
Rys. A.4. Schemat montażowy płytki układu odsprzęgania zasilania.




Rys. A.5. Podsystem sterowania nawrotnego pojedynczym silnikiem.

na indukcyjny charakter silników elektrycznych potrzebne były również diody prostownicze, zabezpieczające układ przed nagłymi skokami napięcia podczas odłączania zasilania silnika [9][62].

Schemat ideowy układu oraz rysunek płytki drukowanej zamieszczono na rys. A.6 oraz rys. A.7.




Rys. A.6. Schemat ideowy płytki układu nawrotnego.

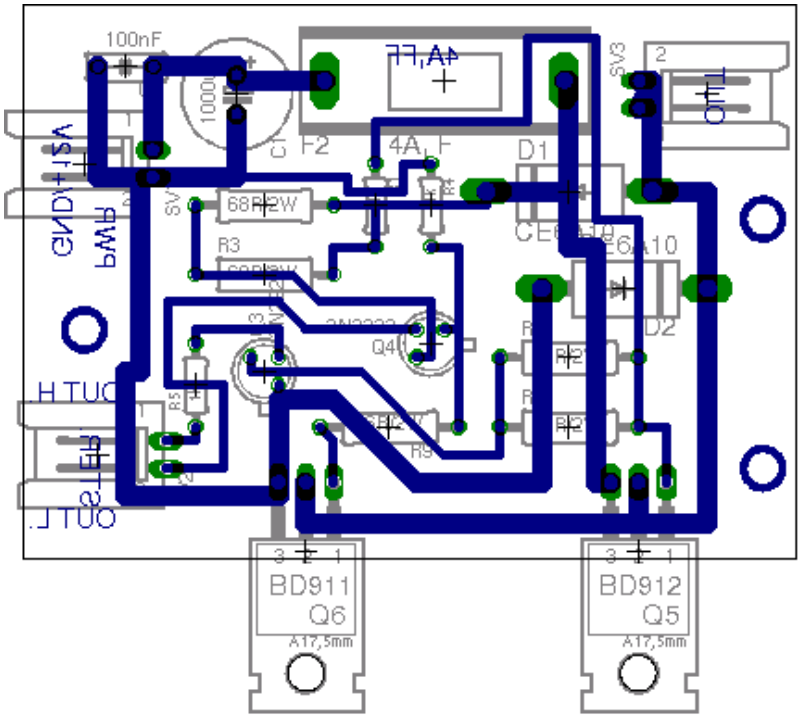
Posiadając dwa takie układy, łącząc wyjście każdego z nich z jednym z wyprowadzeń silnika, otrzymujemy możliwość ustawiania, jakie napięcie ma się znaleźć na którym z wyprowadzeń. Jeżeli napięcia te są równe (lub też wszystkie tranzystory mocy są w stanie odcięcia) silnik nie kręci się. Pojawienie się różnych napięć powoduje obrót silnika w jedną ze stron (zależną od polaryzacji).

Podpięcie po jednym takim zestawie (2 układy) na każdy z 2 silników (razem 4 układy) umożliwia niezależne sterowanie każdą z gąsienic[12].

### A.5.3. Mikrokontroler sterujący

Do sterowania poszczególnymi elementami konstrukcji oraz do komunikowania się z komputerem sterującym wykorzystany został mikrokontroler firmy Atmel [17] AT89C51 [18] zaprogramowany w asemblerze owego procesora[13] [58].

12. Tak naprawdę, z przyczyn realizacyjnych, stan „1 1” (oba tranzystory mocy znajdują się w stanie przewodzenia) jest zablokowany elektronicznie poprzez układ selektora znajdujący się na płytce z mikrokontrolerem, gdyż stan taki spowodowałby przegrzanie i uszkodzenie tranzystorów wyjściowych (nastąpiłoby zwarcie). Widać ten problem na schemacie ideowym omawianego układu rys. A.6. Układ zabezpieczający jest szerzej opisany w podroz. A.5.3
13. Kod źródłowy tego programu ma kilkaset linii i jego przytaczanie tutaj nie jest celowe – można go znaleźć w katalogu z projektem na załączonej płycie CD-ROM.




Rys. A.7. Schemat montażowy płytki układu nawrotnego.

Jego dwie podstawowe funkcje to nadawanie i odbieranie danych oraz sterowanie mocą silników poprzez wypełnienie przebiegu prostokątnego (ang. *PWM*) generowanego na odpowiednich wyprowadzeniach.

Nadawanie i odbieranie danych odbywa się (z punktu widzenia mikrokontrolera) za pomocą portu RS232, o napięciach zgodnych ze standardem TTL. Fizycznym kanałem komunikacyjnym jest droga radiowa realizowana przez układy mini-ARM 2 produkcji warszawskiej firmy Aries-RS[14] [4].

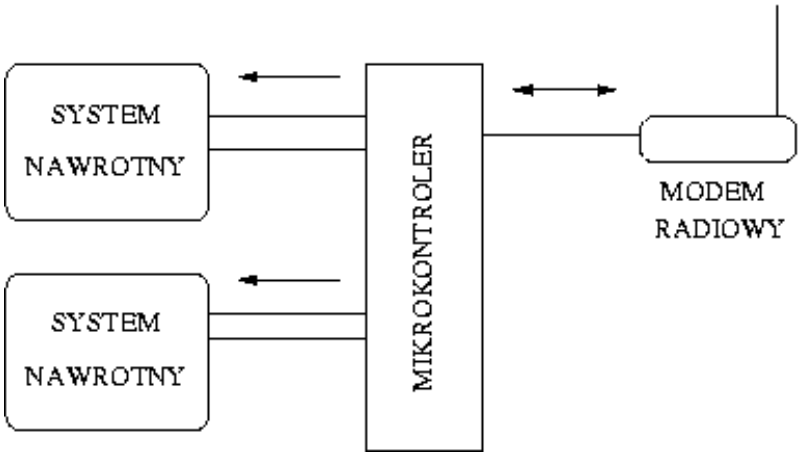
Schemat ideowy przesyłu informacji w robocie ilustruje rys. A.8.

Fizyczne połączenie pomiędzy układem sterującym a sterownikami nawrotnymi jest realizowane poprzez kable dwużyłowe, biegnące ponad płytkami układów elektronicznych, widoczne na rys. A.9, przedstawiającym widok z góry na całość elektroniki robota.

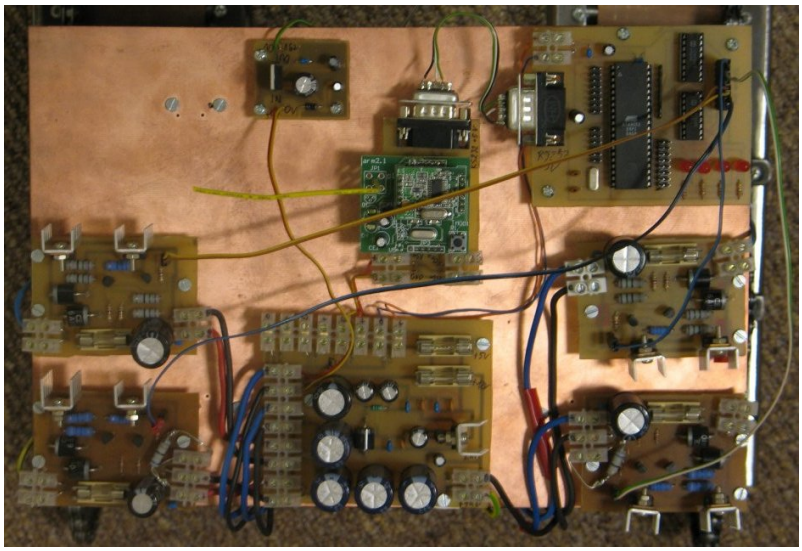
Jako „układ sterujący” należy rozumieć tu mikrokontroler wraz z układami zabezpieczającymi przed podaniem na wejście układów nawrotnych stanu powodującego zwarcie (wspominanego w podroz. A.5.2 stanu „1 1”).

Ponieważ na wyjściu mikrokontrolera pojawiają się stany wysokie na wszystkich wyprowadzeniach w momencie resetowania (co ma miejsce np: podczas startu układu) nie ma możliwości programowego zabezpieczenia przed podaniem niepoprawnego stanu[15]. Aby więc rozwiązać problem eliminacji niepoprawnej „kombinacji” zastosowano układ

14. Po stronie komputera PC do komunikacji z modułem został użyty układ mikroprocesorowy komunikujący się przez port RS232 z modemem oraz przez port LPT (za pomocą dedykowanego protokołu) z komputerem. Szczegóły dotyczące tego układu można znaleźć w katalogu projektowym na załączonej płycie CD-ROM.
15. Nawet gdyby istniała taka możliwość, to istniałoby ryzyko zacięcia się układu, lub też zawieszenia programu, co skończyłoby się spaleniem elementów wyjściowych. Rozwiązanie „programowe” jest więc nie tylko niemożliwe, ale i nieakceptowalne.



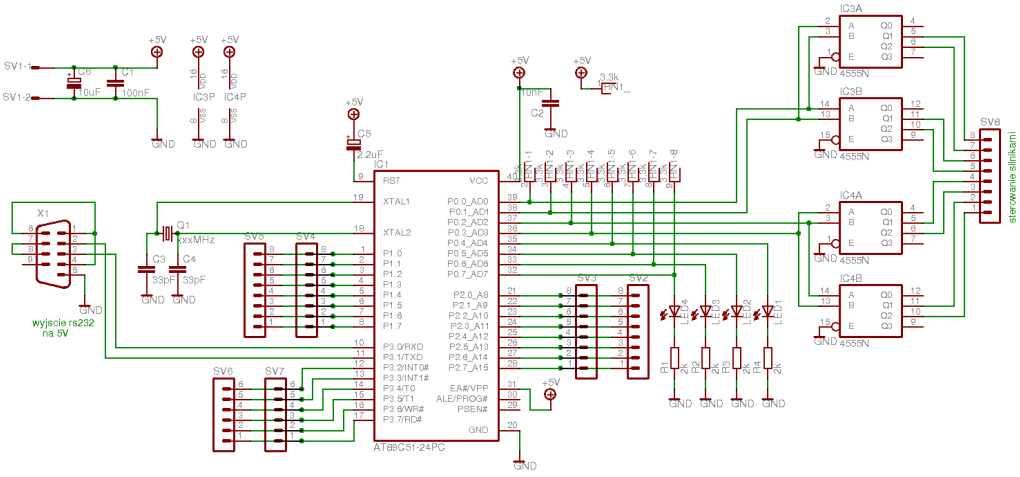

Rys. A.8. Schemat ideowy przepływu danych pomiędzy układami elektronicznymi robota.



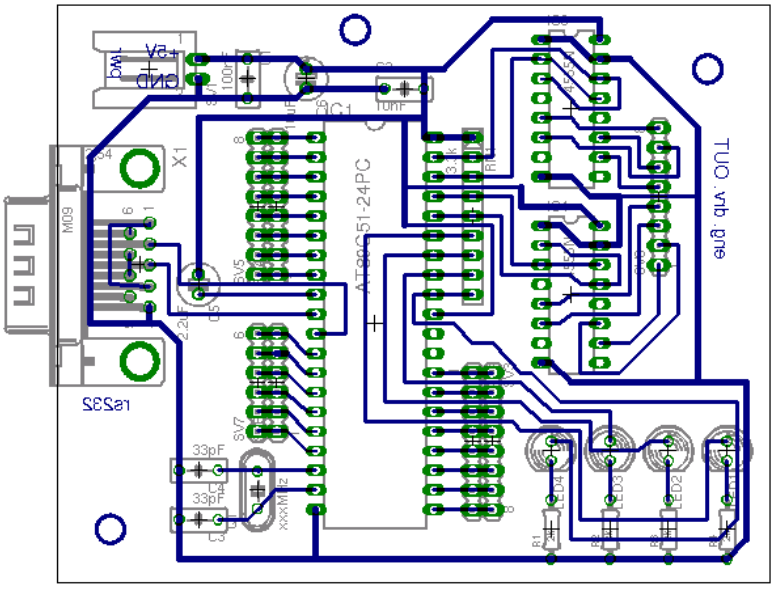
Rys. A.9. Całość elektroniki pokładowej robota – widok z góry.

dwóch selektorów „1 z 4” – układy te otrzymują na wejściu dwa bity interpretowane jako liczbę binarną generując sygnał wysoki tylko na określonym wyjściu (1 z 4 możliwych), zależnym od wartości podanej liczby. Łącząc dwa wyjścia takiego selektora (odpowiadające liczbom binarnym „0 1” oraz „1 0”) do wejść układów nawrotnych oraz wejścia selektora do wyprowadzeń mikrokontrolera zapewniamy, iż tylko jeden stan może być „włączony” w danej chwili czasowej. Dwa pozostałe stany wejściowe („0 0” i „1 1”), choć mogą być wygenerowane przez mikrokontroler, powodują „zapalenie” niepodłączonych wyjść selektora, tym samym są więc ignorowane (z punktu widzenia układu nawrotnego zawsze jest to równoważne podaniu stanu „0 0” na wejście, ponieważ na selektorze oba „podpięte” wyprowadzenia są w tym czasie w stanie niskim).

Schemat ideowy oraz montażowy płytki sterownika (mikrokontrolera i selektorów) przedstawiają kolejno rys. A.10 i rys. A.11.



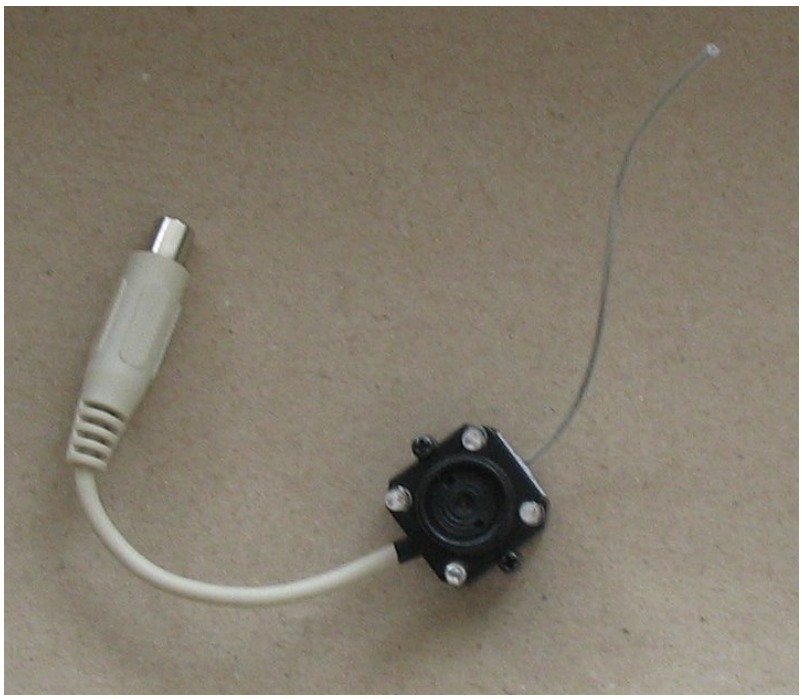

Rys. A.10. Schemat ideowy układu bezpośredniego sterowania robotem.




Rys. A.11. Schemat montażowy układu bezpośredniego sterowania robotem.

## A.6. Kamera wideo

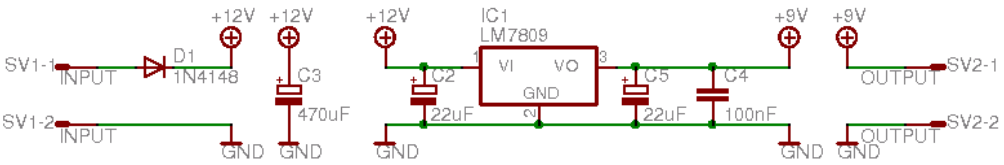
Jedyny czujnik, w jaki jest wyposażony robot to kamera (widoczna na rys. A.12). Została ona umiejscowiona z przodu pośrodku całej konstrukcji. Zasilana jest pojedynczym napięciem $9V$ i posiada własny nadajnik radiowy[16] [6].



Rys. A.12. Kamera bezprzewodowa (bez odbiornika) zastosowana w projekcie.

Do stabilizacji napięcia $9V$ wykorzystano scalony stabilizator LM7809 wraz z odpowiednimi kondensatorami filtrującymi, wymaganymi ze względu na fakt, iż stabilizacja odbywa się bezpośrednio z wyjść $12V$ podłączanych pod układy mocy (czyli prawie bezpośrednio pod silniki).

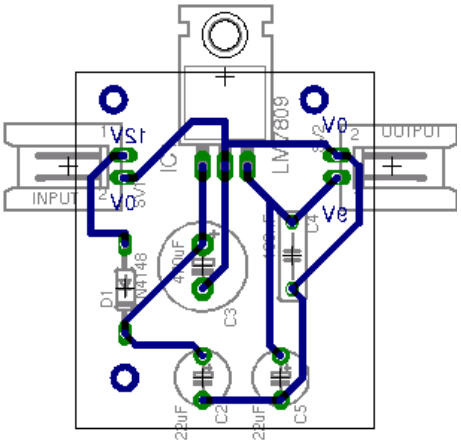
Schemat ideowy i montażowy układu stabilizatora dla kamery widać na rys. A.13 oraz rys. A.14.



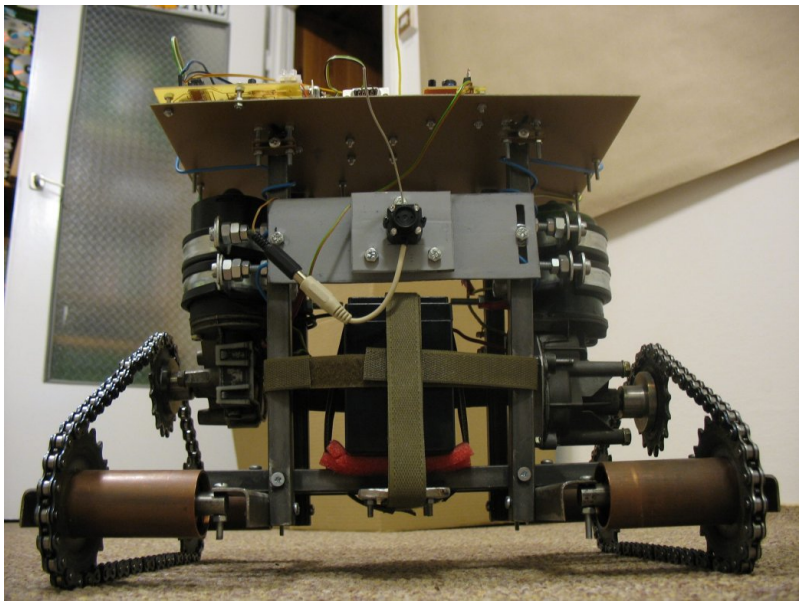

Rys. A.13. Schemat ideowy układu stabilizatora 9V dla kamery wideo.

rys. A.15 i rys. A.16 przedstawiają fizyczny sposób mocowania kamery do konstrukcji nośnej wraz z elementami regulacyjnymi (trzy śruby widoczne po bokach kamery).

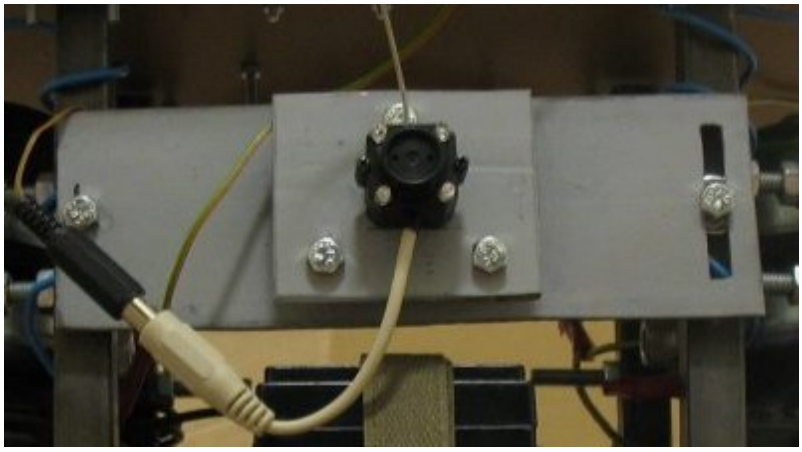
16. Więcej inforacji odnośnie mechanizmu komunikacyjnego i miejsca kamery w nim znajduje się w sekcji podroz. A.3




Rys. A.14. Schemat montażowy układu stabilizatora 9V dla kamery wideo.



Rys. A.15. Sposób fizycznego zamocowania kamery na robocie.

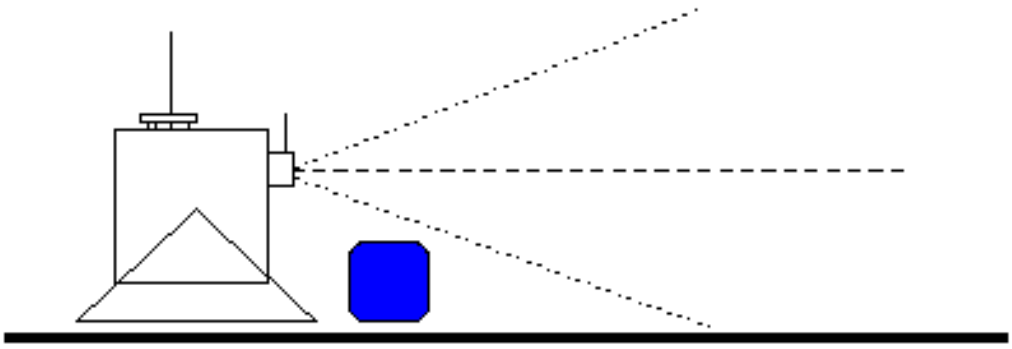


Rys. A.16. Sposób fizycznego zamocowania kamery na robocie – zbliżenie na kamerę.

## A.7. Końcowe przemyślenia

W niniejszym rozdziale omówiona została pobieżnie budowa fizycznego robota, który został zbudowany na potrzeby badań. Praktyka pokazała, iż posiada on wciąż wiele (nie zawsze oczywistych) wad konstrukcyjnych, które utrudniały późniejsze prace lub też powodowały zagrożenia dla samej konstrukcji w trakcie jej eksploatacji. Pokrótce zostaną przybliżone najpoważniejsze niedopracowania, z jakimi się spotkano.

Jedną z ważniejszych wad konstrukcyjnych jest sposób mocowania kamery – ze względu na zastosowany mechanizm matematyczny i przyjęte założenia została ona wykalibrowana tak aby filmowała możliwie dokładnie „na wprost" robota, równolegle do płaszczyzny podłogi[17]. W praktyce okazało się to być bardzo dużym ograniczeniem, gdyż ze względu na wąski kąt obserwacji, jaki oferuje zastosowany sprzęt, istotne obszary dookoła robota (podłoga) są widoczne nie bliżej niż około $50cm$ od samego robota – powoduje to oczywisty problem z wykrywaniem przeszkód, które znajdą się zbyt blisko, ponieważ nie zostaną wogóle zauważone. Przedstawiono taką sytuację na rys. A.17.



Rys. A.17. Przykład przeszkody niezauważonej przez robota ze względu na zbyt małą odległość od niej.

17. Dokładny opis podsystemu wizyjnego znajduje się w podroz. 2.2.7.

Kolejnym słabym punktem jest miejscowe wystawanie konstrukcji nośnej elektroniki (duża płyta laminatowa, do której są przytwierdzone poszczególne układy) poza linię stalowej konstrukcji robota. Problem polega na tym, że w momencie, kiedy w wyniku błędu w oprogramowaniu robot uderza w przeszkodę, to istnieją miejsca, w których, patrząc od góry, pierwszym elementem jaki jest narażony na uderzenie jest nie metalowa konstrukcja robota, lecz laminatowa, luźno zawiszona płytka z elektroniką pokładową – ostry, a zarazem bardzo delikatny element.

Podczas testów raz zdarzył się przypadek, kiedy wada ta ujawniła się praktycznie – robot został dość szybko wyłączony więc skończyło się na wgnieceniu w drewnianej nodze od biurka. Szczęśliwie żaden z podzespołów elektronicznych się nie uszkodził.

Na etapie testowania i kalibrowania kamery kilkukrotnie zachodziła potrzeba włożenia (metalowego) śrubokręta do wewnętrznych części konstrukcji, celem dokręcenia/poluzowania śrub odpowiadających za kierunek patrzenia. Zawsze podczas tej czynności robot był wyłączony, jednak mimo to zdarzyło się raz, iż omyłkowe zetknięcie śrubokrętem śruby z elementem zasilającym spowodowało zwarcie i przepalenie początkowego odcinka ścieżki na płytce drukowanej odsprzęgania zasilania. Problem ten ujawnił, iż brakowało w układzie jeszcze jednego bezpiecznika (zwłocznego) – powinien to być pierwszy element układu, nie tak zaś jak zostało to zrobione, kiedy wszystkie bezpieczniki są szybkie ale już „za" kondensatorami wejściowymi[18].

Ze względu na wibracje sztywnego, metalowego szkieletu robota, celem osłony układów elektronicznych przed niekorzystnymi drganiami, mogącymi powodować obluzowanie się elementów, a co za tym idzie, trudne do zdiagnozowania przypadkowe zachowania układów, zastosowano mocowanie elastyczne (elastycznej już samej z siebie) płytki laminatowej poprzez zawieszenie jej na kilku wycinkach laminatu (mocowanie takie przedstawia rys. A.18).

Niestety, takie mocowanie okazało się nadal zbyt sztywne i choć podczas testów żaden z elementów nie obluzował się, to kilkukrotnie zdarzało się znajdować nakrętki mocujące poszczególne elementy do laminatu na podłodze – odkręcały się one właśnie z powodu wibracji.

Aby możliwie najbardziej zniwelować ten niekorzystny efekt mocowanie powinno być dodatkowo wyłożone gumą, zaś wszystkie śrubki mocujące poszczególne elementy musiałyby mieć podkładki z twardej gumy i być mocno dokręcone.

Podczas początkowego testowania oprogramowania sterującego układami nawrotnymi zdarzały się zawieszenia oprogramowania owocujące „migotaniem" naprzemiennym losowych stanów na wyjściach portów mikrokontrolera. Choć układy selektorów nie dopuściły do bezpośredniego pojawienia się stanów niepoprawnych, to fizyka tranzystorów mocy powodowała, iż ich inercyjność przy wyłączaniu powodowała praktycznie jednoczesne przewodzenie obu przy zmianach stanów wejściowych z częstotliwością

18. Zostały one tam umieszczone, aby uniknąć problemów z przepalaniem się ich podczas uruchamiania robota, kiedy trzeba naładować znaczną pojemność kondensatorów elektrolitycznych, w krótkim czasie, co owocuje krótkim przepływem dużego prądu.

Rys. A.18. Mocowanie elastycznego laminatu na układy elektroniczne do metalowego szkieletu robota.

rzędu $0.5MHz$[19] – efekt był zbliżony do zwarcia i jedynie szybkie wyłączanie zasilania ratowało tranzystory przed spaleniem.

Rozwiązanie idealne posiadałoby w tym miejscu dodatkowy układ czasowy, niepozwalający na zmianę stanu częściej niż np: sto razy na sekundę ($100Hz$).

Problemy ze sterowaniem pojawiały się również na etapie ustawiania mocy silników. Jak już wspominano w podroz. A.5.1, moc jest regulowana poprzez zmianę wypełnienia przebiegu sygnału włączającego silnik. Takie rozwiązanie nie zapewnia jednak liniowego przełożenia procentowego wypełnienia przebiegu na ilość obrotów koła. Aby obejść ten problem wymagana była programowa korekcja wartości (poprzez zastosowanie empirycznie dobranej funkcji, „uliniawiającą" tę zależność), jednak problem powracał podczas zakręcania, kiedy to jedna gąsienica była mniej obciążona od drugiej. Dodatkowo podanie zbyt małego wypełnienia fazy powodowało przepływ prądu bez wykonywania jakiegokolwiek obrotu co było czystą stratą i groziło przegrzaniem elementów mocy.

Jedynym wydającym się zdawać tu egzamin rozwiązaniem jest przerobienie mechanizmu sterującego prędkością, tak aby na wejściu podsystemu podawać nie abstrakcyjne procentowe wypełnienie przebiegu („abstrakcyjne" gdyż jest ono zależne od zbyt wielu nieoczywistych i trudno mierzalnych czynników aby posługiwać się ową miarą w prosty sposób), lecz mającą fizyczne odzwierciedlenie ilość obrotów koła napędowego w ustalonej jednostce czasu. Aby jednak taka wielkość miała realne odbicie w rzeczywistości wymagane jest (szybkie) sprzężenie zwrotne w postaci odczytów obrotów, jakie rzeczywiste koło wykonuje. Mając takie informacje można odpowiednio sterować wypełnieniem aby utrzymać zadane parametry wejściowe.

Ostatnim zagadnieniem, jakie zostało pochopnie zbagatelizowane w procesie projektowania robota, było monitorowanie napięcia zasilania. Choć założenie to początkowo

19. Mikrokontroler jest taktowany kwarcem $11.059MHz$, co przy konstrukcji rdzenia wykorzystującej 12 cykli zegarowych na jeden cykl maszynowy i rozkazach (w większości) jednocyklowych daje około miliona instrukcji na sekundę [58].

zdawało egzamin, gdyż całość elektroniki była zasilana napięciem $5V$ wystabilizowanym z $12V$[20] zasilających silniki, więc rozładowany akumulator nie oddawał już wystarczającej mocy do wysterowania silników, to niskomocowa elektronika nadal działała poprawnie. Problemy zaczęły się po dodaniu kamery bezprzewodowej, wymagającej napięcia $9V$. Kiedy napięcie na akumulatorze zaczynało opadać to na obrazie wysyłanym przez kamerę pojawiały się „pływające" białe paski, powodujące często błędną segmentację obrazu (a co za tym idzie „nielogiczne" decyzje o omijaniu nieistniejących przeszkód). Kiedy akumulator był już bardzo rozładowany (napięcie na nim wynosiło około $10 - 11V$) pojawiały się też czasowe przerwy w strumieniu wideo oraz szumy w przypadkowych fragmentach rejestrowanego obrazu, praktycznie uniemożliwiające dalszą pracę robota.

20. Napięcie świeżo naładowanego akumulatora było nawet wyższe i sięgało niemal $13V$.

## Dodatek B

# Podziękowania

Na zakończnie autor chciałby szczególnie podziękować kilku osobom, których praca i pomysły miały szczególny wpływ na charakter całego projektu. Dzięki pomocy tych osób udało się zrobić to co zostało zaplanowane w założonym czasie.

Przede wszystkim pragnę serdecznie podziękować promotorce niniejszej pracy Pani **prof. dr hab. inż. Halinie Kwaśnickiej** za nieocenioną pomoc zarówno od strony merytorycznej jak i technicznej, zarówno podczas badań jak i podczas pisania samej pracy. Liczne uwagi jakie poczyniła wielokrotnie były punktami zwrotnymi.

Wyrazy wdzięczności należą się również **Mieczysławowi Felcenlobenowi**, który poświęcił wiele godzin swojej pracy i włożył wiele wysiłku aby stworzyć kluczowe elementy składowe konstrukcji mechanicznej robota. Przez cały czas prac służył on również techniczną radą i niezliczoną ilością uwag przyczyniając się do ostatecznego funkcjonowania budowanego robota.

Podziękowania należą się również **Andrzejowi Szurgotowi** oraz **Jakubowi Szurgotowi** za pomoc merytoryczną z zakresu mechaniki i elektroniki oraz udostępnienie narzędzi niezbędnych do stworzenia metalowej konstrukcji mechanicznej robota gąsienicowego.

Wyrazy wdzięczności autor pragnie również przekazać kolegom **Zbysiowi** oraz **Arnlodowi_S.** z „forum elektronika” [9] za pomoc merytoryczną przy opracowywaniu i prototypowaniu układów nawrotnych dla silników napędzających robota. Dzięki ich trafnym spostrzeżeniom oraz fachowym radom udało się doprowadzić ten etap do końca.

Na przebieg prac znaczący wpływ miał również **Michał Przewoźniczek** – jako absolwent wydziału elektroniki, doświadczony i zainteresowany robotyką kilkukrotnie wskazał trafnie kierunki warte zgłębiania oraz nakierował myśli autora w dobrą stronę.

# Spis rysunków


0.1 Kadr z popularnego filmu „I, robot” (zaczerpnięty z [34]). . . . . 1

1.1 Przykładowe roboty lądowe (zaczerpnięte kolejno z [30] oraz [33]). . . . . 5
1.2 Przykładowe roboty wodne (zaczerpnięte z [31] oraz [32]). . . . . 5
1.3 Przykładowe roboty latające (zaczerpnięte z [28] oraz [29]). . . . . 5
1.4 Przykładowe wyniki działania algorytmów genetycznych do wyszukiwania tras na mapie (zaczerpnięte z implementacji koncepcji zaprezentowanej w [47]). . . . . 7
1.5 Robot omija przeszkodę stosując logikę rozmytą (im bliżej przeszkody, tym mocniej skręca by ją ominąć) w porównaniu z prostym systemem regułowym. . . . 8
1.6 Ideowy rysunek pokazujący zasadę działania dekompozycji behawioralnej (zaczerpnięty z [56]). . . . . 9
1.7 Przykładowy robot zaprojektowany przez algorytmy ewolucyjne (a) oraz jego fizyczna realizacja (b). . . . . 10

2.1 Ogólny model oddziaływania robota ze światem. . . . . 12
2.2 Model oddziaływania robota ze światem zastosowany w prezentowanym rozwiązaniu. 12
2.3 Przepływ danych w systemie jako całości. . . . . 12
2.4 Przepływ danych w systemie wizyjnym jako całości. . . . . 13
2.5 Przykład skalowania obrazu dla skali $k = 0.5$. . . . . 15
2.6 Obraz przy dobrych warunkach oświetleniowych: (a) obraz wejściowy; (b) obraz po przetworzeniu (normalizacji). . . . . 16
2.7 Obraz przy złych warunkach oświetleniowych (zbyt ciemny): (a) obraz wejściowy; (b) obraz po przetworzeniu (normalizacji). . . . . 16
2.8 Połączenia wewnątrz sieci CNN (rysunek zaczerpnięty z [20]). . . . . 17
2.9 Macierze sprzężenia zwrotnego i sterowania (rysunek zaczerpnięty z [19]). . . . . 18
2.10 Detekcja krawędzi przy użyciu sieci neuronowej na przykładzie obrazu „stół”: obraz wejściowy (a) oraz obraz wyjściowy (b). . . . . 19
2.11 Detekcja krawędzi przy użyciu sieci neuronowej dla „pstrokatej” powierzchni: obraz wejściowy (a) oraz obraz wyjściowy (b). . . . . 20
2.12 Efekt operacji „denoise” dla obrazu krawędzi „pstrokatej” powierzchni: (a) obraz wejściowy oraz (b) obraz wyjściowy. . . . . 20
2.13 Efekt całości procesu przetwarzania obrazu wejściowego „stół” metodą hybrydową: (a) obraz wejściowy oraz (b) obraz wyjściowy. . . . . 21
2.14 Efekt całości procesu przetwarzania obrazu wejściowego „pstrokaty” metodą hybrydową: (a) obraz wejściowy oraz (b) obraz wyjściowy. . . . . 21

2.15 Efekt całości procesu przetwarzania obrazu wejściowego „stół" metodą grafową: (a) obraz wejściowy oraz (b) obraz wyjściowy. . . . . 25
2.16 Efekt całości procesu przetwarzania obrazu wejściowego „pstrokaty" metodą grafową: (a) obraz wejściowy oraz (b) obraz wyjściowy. . . . . 25
2.17 Wybrane karty rozszerzeń realizujące sieć CNN sprzętowo – karta dla komputera w standardzie PC104 (a – zaczerpnięte z [15]) i karta PCI dla standardowego komputera domowego PC (b – zaczerpnięte z [14]). . . . . 27
2.18 Obraz z kamery robota – kosz zagradzający drogę. . . . . 28
2.19 (a) Obraz po segmentacji rys. 2.18 (b) z zaznaczoną na czarno linią istotną dla dalszej analizy (odczytywania głębi). . . . . 28
2.20 Obraz z rys. 2.19(a) po zaznaczeniu ważnych punktów i ich wektoryzacji. . . . . 29
2.21 Wyznaczanie głębi w obrazie na podstawie obrazu stereowizyjnego. Rysunek zrobiony na podstawie [53]. . . . . 29
2.22 Schemat wyznaczania głębi z obrazu – widok z boku. . . . . 31
2.23 Prosta droga – równoległe linie krawędzi wydają się zbieżne (zaczerpnięte z [21]). 32
2.24 Schemat wyznaczania głębi z obrazu – widok z góry. . . . . 33
2.25 Obraz z kamery robota (a) z zaznaczoną częścią użyteczną teoretycznie oraz (b) częścią użyteczną przy pewnej, założonej dokładności. . . . . 34
2.26 Budowa pojedynczego pakietu protokołu stosowanego do komunikacji z robotem drogą radiową. . . . . 36
2.27 Sytuacja zakleszczenia dla najprostszej heurystyki sterującej, kiedy robot zaczyna skręcać naprzemiennie w obie strony bez końca: (a) sygnał skręcania w lewo; (b) sygnał skręcania w prawo. . . . . 37
2.28 Zbyt naiwna heurystyka skrętu powoduje skręt wprost na przeszkodę. . . . . 38

3.1 Poruszanie się robota w ciasnym korytarzu: (a) jest dość miejsca; (b) obrót spowoduje wyciągnięcie błędnych decyzji i kolizję ze ścianą. . . . . 43
3.2 Problem z systemem wizyjnym przy nadmiernym zbliżeniu się do przeszkody. . . 43
3.3 Przykładowa sytuacja kiedy gwałtowna zmiana natężenia światła wygląda jak krawędź powodując niepoprawną segmentację. . . . . 44

4.1 Zdjęcie przykładowego robota samodzielnie odkurzającego pomieszczenia (zaczerpnięte z [24]). . . . . 46
4.2 Pole widzenia kamery skierowanej pod pewnym kątem względem pionu. . . . . 47
4.3 Propagacja informacji w sieci neuronowej (pogrubione, zielone linie). . . . . 49

A.1 Przebieg komunikacji robot-komputer. . . . . 55
A.2 Metalowy szkielet (konstrukcja nośna) robota TIER z założoną jedną gąsienicą. . . 56
A.3 Schemat ideowy płytki układu odsprzęgania zasilania. . . . . 57
A.4 Schemat montażowy płytki układu odsprzęgania zasilania. . . . . 58
A.5 Podsystem sterowania nawrotnego pojedynczym silnikem. . . . . 58
A.6 Schemat ideowy płytki układu nawrotnego. . . . . 59
A.7 Schemat montażowy płytki układu nawrotnego. . . . . 60
A.8 Schemat ideowy przepływu danych pomiędzy układami elektronicznymi robota. . . 61
A.9 Całość elektroniki pokładowej robota – widok z góry. . . . . 61
A.10 Schemat ideowy układu bezpośredniego sterowania robotem. . . . . 62
A.11 Schemat montażowy układu bezpośredniego sterowania robotem. . . . . 62
A.12 Kamera bezprzewodowa (bez odbiornika) zastosowana w projekcie. . . . . 63
A.13 Schemat ideowy układu stabilizatora 9V dla kamery wideo. . . . . 63
A.14 Schemat montażowy układu stabilizatora 9V dla kamery wideo. . . . . 64
A.15 Sposób fizycznego zamocowania kamery na robocie. . . . . 64

A.16 Sposób fizycznego zamocowania kamery na robocie – zbliżenie na kamerę. . . . . 65
A.17 Przykład przeszkody niezauważonej przez robota ze względu na zbyt małą odległość od niej. . . . . 65
A.18 Mocowanie elastycznego laminatu na układy elektroniczne do metalowego szkieletu robota. . . . . 67

# Spis tablic

2.1 Macierz sterowania dla $r = 2$. . . . . 18
2.2 Macierz sprzężenia zwrotnego dla $r = 2$. . . . . 18