SYSTEM STEROWANIA O STRUKTURZE RÓWNOLEGŁEJ – ARCHITEKTURA BROOKSA

(wg. IEEE JOURNAL OF ROBOTICS AND AUTOMATION. VOL. RA-2.NO.1. 1986)

Spis treści

Dekompozycja behawioralna zadania sterowania lokomocją

Działanie lokomocyjne robota można traktować jako złożenie wielu prostych, niezależnych działań (przykładowo, takich jak: podążanie do lokalnego celu, ruch korytarzem, itp.), które dają się zaobserwować jako odrębne ***„zachowania”*** robota.

Zadanie sterowania lokomocją można więc zdekomponować na szereg niezależnych od siebie zadań, związanych ze sterowaniem wyróżnionymi zachowaniami.
Taka dekompozycja prowadzi do systemu sterowania o ***architekturze równoległej***.

Koncepcja architektury Brooksa

Szczególną koncepcję budowy systemu o architekturze równoległej zaproponował Rodney Brooks. Zgodnie z tą koncepcją zadanie sterowania lokomocją dekomponujemy przez wyróżnienie nie samych zachowań, lecz tzw. „poziomów kompetencji” działania robota.

Def. ***Poziom kompetencji*** jest opisem pożądanej klasy zachowań robota w oczekiwanych sytuacjach.

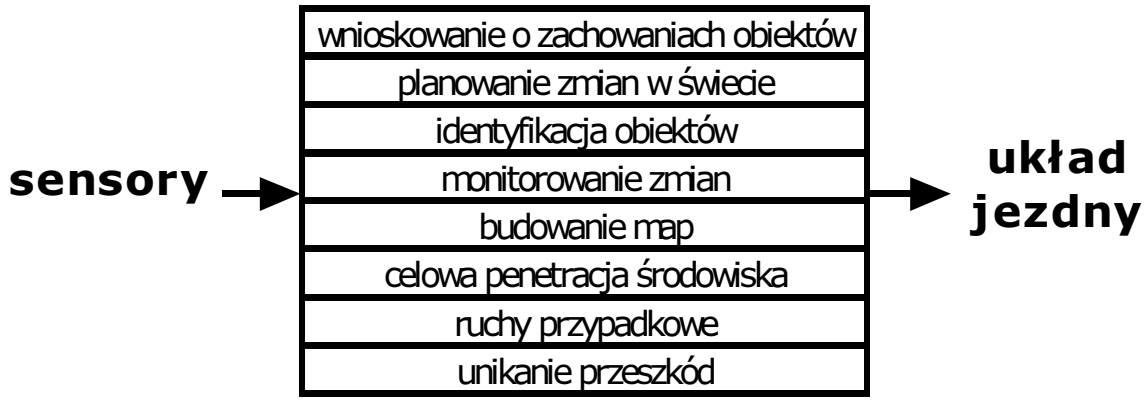
POZIOMY KOMPETENCJI I STRUKTURA SYSTEMU

Brooks wyróżnił następujące poziomy kompetencji:

0. - unikanie kontaktu z obiektami (ruchomymi lub stacjonarnymi);
1. - ”przypadkowe” poruszanie się w otoczeniu bez kolizji z obiektami;
2. - „poznawanie” świata, poprzez rozpoznawanie i przemieszczanie się w kierunku najbardziej odległych miejsc (korytarze);
3. - budowanie mapy otoczenia i planowanie dróg pomiędzy poznanymi miejscami;
4. - wykrywanie zmian w otoczeniu statycznym;
5. - wnioskowanie o otoczeniu w kategoriach „identyfikowalności” obiektów i realizacja zadań związanych z pewnymi obiektami;
6. - tworzenie i wykonywanie planów prowadzących do pożądanej zmiany stanu otoczenia;
7. - wnioskowanie o zachowaniu się obiektów i odpowiednie do tego modyfikowanie planów.

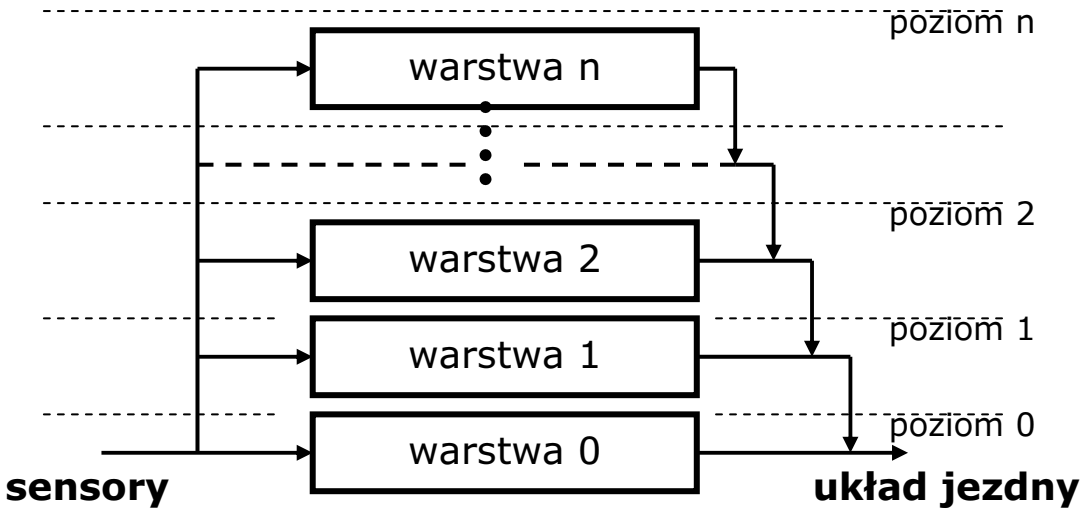
Można zauważyć, że poziomy te tworzą hierarchię, w której każdy wyższy poziom zawiera wszystkie poprzednie jako podzbiory.
Każdy wyższy poziom kompetencji pociąga za sobą lepiej określoną (węższą) klasę dopuszczalnych zachowań.

Struktura systemu sterowania uwzględniająca opisaną koncepcję dekompozycji ma następująca postać:



Warstwy sterowania i architektura Subsumption

System sterowania tworzą równoległe warstwy, które współpracując ze sobą w określony sposób, sterują działaniem na poszczególnych poziomach kompetencji. Podstawowa korzyść wynikająca z idei poziomów kompetencji polega na tym, że można najpierw zbudować system złożony z kilku warstw zapewniających jakiś początkowy poziom kompetencji, a potem dodawać nowe warstwy osiągając przez to kolejne wyższe poziomy.



Działający system sterowania otrzymamy już po zbudowaniu warstwy osiągającej „0" poziom kompetencji. Po uruchomieniu i przebadaniu systemu nie trzeba go już więcej zmieniać - jest to tzw. system sterowania zerowego poziomu.
Następnie można zbudować i dodać do systemu kolejną warstwę (1-szą). Warstwa ta we współpracy z warstwą zerową osiąga pierwszy poziom kompetencji. Jest ona w stanie badać dane systemu zerowego poziomu oraz oddziaływać na jego linie wyjściowe, zmieniając pierwotny przepływ danych.

Warstwa zerowa działa dalej, nieświadoma istnienia warstw wyższych.

Proces dodawania kolejnych warstw może być kontynuowany, aż do osiągnięcia pożądanego zachowania robota.

Pojedyncza warstwa składa się z asynchronicznie działających modułów a każdy moduł jest prostą maszyną obliczeniową. Warstwy o wyższych poziomach kompetencji, mogą przejąć działanie niższych poprzez blokadę ich wyjść, pomimo tego niższe kontynuują swoje działanie.

Taka struktura została określona mianem ***architektury Subsumption***

Zalety architektury Subsumption

Osiąganie wielu celów Poszczególne warstwy mogą pracować nad osiągnięciem różnych celów - czasami wykluczających się. W takim wypadku mechanizm blokowania pośredniczy w wyborze podejmowanej akcji. Zaletą takiego rozwiązania jest brak konieczności wcześniejszego podejmowania decyzji o wyborze celu – można ją podjąć dopiero w oparciu o wyniki pracy poszczególnych warstw.

Wykorzystanie wielu sensorów Można częściowo pominąć problem fuzji sensorów. Nie wszystkie odczyty sensorów muszą trafiać do centralnej reprezentacji w systemie, a jedynie te, które przetwarzanie percepcyjne uzna za wysoce niezawodne. Jednocześnie odrzucone odczyty mogą być wykorzystywane w innych warstwach dla osiągania ich celów. Można też powiązać określone sensory z określonymi warstwami.

Odporność Wielość sensorów zwiększa odporność systemu o ile ich odczyty mogą być inteligentnie wykorzystane.
Dodatkowym, specyficznym źródłem odporności architektury Subsumption jest to, że wprowadzenie nowej, nawet źle funkcjonującej warstwy nie wprowadza zakłócenia do pracy warstw niższych. One kontynuują swoje działanie i w przypadku gdy nowa warstwa sobie nie radzi, sterują robotem.

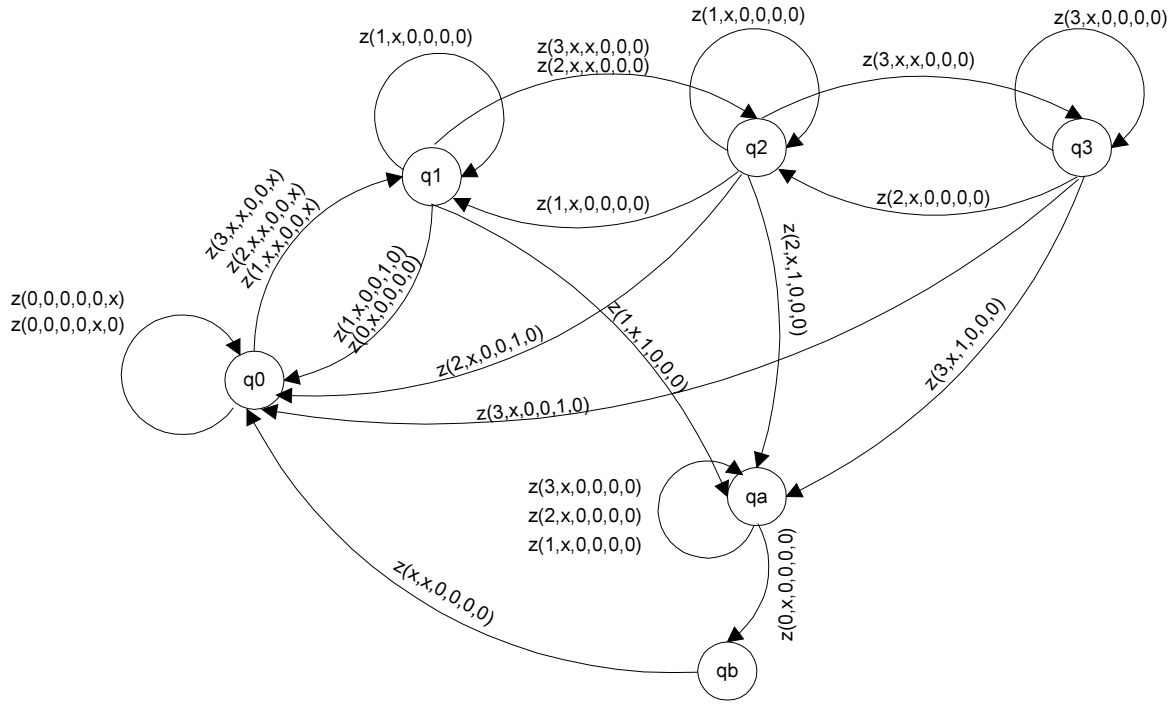
Rozszerzalność Każdą warstwę można zbudować na osobnym procesorze a nawet rozprzestrzenić na kilka procesorów. To jest proste, gdyż wymagania szerokości pasma komunikacyjnego pomiędzy warstwami (i wewnątrz nich) są niewielkie. Nie ma także potrzeby koordynowania współpracy procesorów.

Struktura warstwy

Warstwę można zbudować z małych procesorów przesyłających pomiędzy sobą komunikaty.

Każdy ***procesor*** jest maszyną typu ***automat skończony***.

Przykład Automatu Skończonego



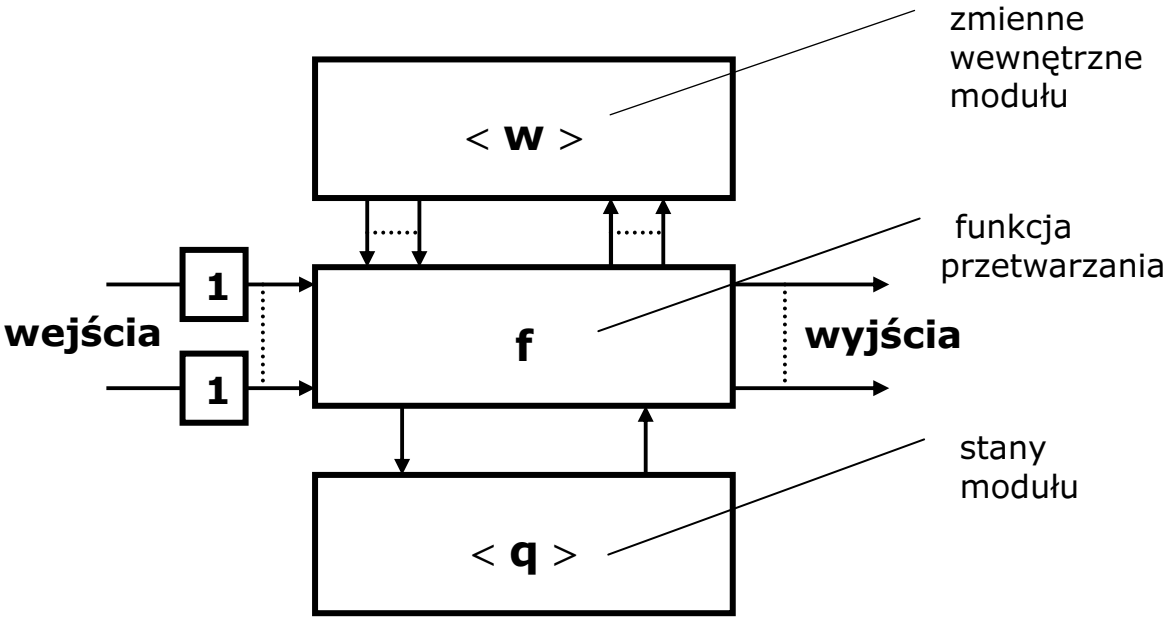
U Brooksa każdy ***procesor*** jest maszyną typu ***automat skończony***, ale rozszerzoną o możliwość pamiętania pewnej struktury danych i realizacji prostych funkcji geometrycznych.

Procesory przesyłają komunikaty po łączących je liniach, asynchronicznie i bez potwierdzenia. Monitorują swoje wejścia, i wysyłają wiadomości na swoje wyjścia. Komunikaty mogą być tracone, (gdy nie zostaną w porę odebrane). Nie ma innej formy komunikacji pomiędzy procesami, a w szczególności nie ma wspólnej pamięci. Te procesory określamy mianem ***modułów***.

Wszystkie moduły są równorzędne w tym sensie, że w warstwie nie ma centralnego sterowania. Linie wejściowe modułów mogą być ***„tłumione”***, a wyjścia ***„blokowane”***, co daje mechanizm, w którym wyższa warstwa może ***„subsumować”*** niższą, przejmując jej rolę.

Budowa modułu

Każdy moduł jest automatem o skończonej liczbie stanów, rozszerzonym dodatkowo o zmienne wewnętrzne które mogą obsłużyć struktury danych Lispowych.
Moduł ma pewną liczbę wejść i wyjść. Linie wejściowe wyposażone są w jednoelementowe bufory, tak że ostatnia wiadomość jest zawsze dostępna, jednak może ona zostać zgubiona, gdy nie zostanie wykorzystana przed pojawieniem się kolejnej.



Każdy stan automatu ma swój symbol. Istnieje wyróżnione wejście RESET. Po odebraniu sygnału RESET, moduł znajduje się w stanie NIL.

Wyróżniamy cztery rodzaje stanów:

- ***Output – stan związany z generowaniem wyjścia***

Na linie wyjściową jest wysyłany komunikat obliczony jako funkcja zawartości buforów wejściowych i zmiennych wewnętrznych, po czym moduł przechodzi do kolejnego stanu

- ***Side effect - stan pośredni***

Jedna ze zmiennych wewnętrznych modułu przyjmuje nową wartość obliczoną jako funkcja wejść i innych zmiennych; następnie moduł przechodzi w kolejny stan.

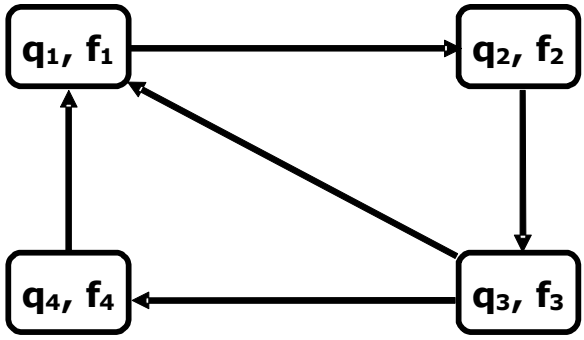
- ***Conditional dispatch - stan z przejściem warunkowym***

Na zmiennych wewnętrznych i wejściach jest obliczany pewien predykat, i w zależności od jego spełnienia (prawdziwości) następuje przejście do jednego z dwu kolejnych stanów.

- ***Event dispatch - stan z przejściem zdarzeniowym***

Sprawdzane są sekwencje par typu warunek - stan, aż jakieś zdarzenie (warunek) jest prawdziwe. Zdarzenia są wyrażeniami and/or na sygnałach wejściowych. Wyrażenia mogą dodatkowo obejmować czynnik opóźnienia, który jest równy 1 przez pewien czas po przejściu do tego stanu. Wyjście ze stanu następuje po spełnieniu jednego z wyrażeń.

Graf automatu skończonego realizującego moduł AVOID



Symbole q_i, f_i charakteryzują odpowiednio stan oraz realizowaną w nim funkcję.

q_1 - (NIL; stan z przejściem zdarzeniowym) - przejście następuje gdy na wejściu modułu pojawi się sygnał: *siła kierująca*.

q_2 - (PLAN; stan pośredni) – realizowana jest funkcja f_2 - (*wybór kierunku*), obliczająca *chwilowy kierunek ruchu* na podstawie *wypadkowej siły odpychającej* i *siły kierującej*.

q_3 - (GO; stan z przejściem warunkowym) - realizowana jest funkcja f_3 - (*poziom siły*), sprawdzająca czy *wypadkowa siła* (*chwilowy kierunek ruchu*) przekracza zadany próg; gdy TAK następuje przejście do q_4, gdy NIE - powrót do q_1.

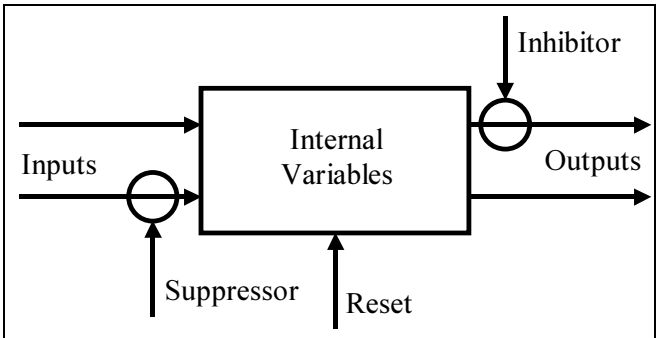
q_4 - (START; stan typu wyjście) - jest realizowana funkcja f_4 (*ruch w kierunku siły*) przekształcająca kierunek i wartość *wypadkowej siły* na komendy *sterowania silnikami*.

Komunikacja pomiędzy modułami

Rysunek ilustruje komunikacyjne własności modułu. Każdy moduł posiada pewną liczbę wejść i wyjść. Linia wychodząca z wyjścia jednego modułu prowadzi do wejścia innego modułu lub wejść wielu innych modułów. Linie można traktować jak przewody. Wyjścia mogą być blokowane (linią ***inhibit***), a wejścia tłumione (linią ***suppress***).

Pojawienie się dowolnego sygnału na linii *inhibit* powoduje zablokowanie (zawieszenie) danego wyjścia na pewien określony czas.

Podobnie działa sygnał na linii *suppress*, z tym, że w tym przypadku, ten sygnał wchodzi na wejście modułu zastępując sygnał stłumiony.



Wejścia i wyjścia modułu

Przykłady implementacji architektury Subsumption

Brooks opracował i zaimplementował pierwsze trzy warstwy architektury Subsumption osiągające drugi poziom kompetencji. Przedstawił też system sterowania dla robota poruszającego się wzdłuż ściany.

Poziom zerowy

Celem sterowania poziomu zerowego jest niedopuszczenie do kolizji robota z przeszkodą. Jeśli coś zbliży się do robota, to robot powinien się odsunąć. Jeśli bezpośrednio na jego kursie znajdzie się przeszkoda, to robot powinien się zatrzymać. Połączenie tych dwóch taktyk jest wystarczające do zapewnienia ucieczki od przeszkód ruchomych oraz unikania kolizji z przeszkodami stacjonarnymi (choć może wymagać wielu ruchów).
Działanie poszczególnych modułów przedstawiają się następująco:

Moduł Sonar - zbiera i wstępnie przetwarza dane pomiarowe z czujników robota a następnie tworzy lokalną mapę otoczenia (we współrzędnych biegunowych, z robotem umieszczonym centralnie).

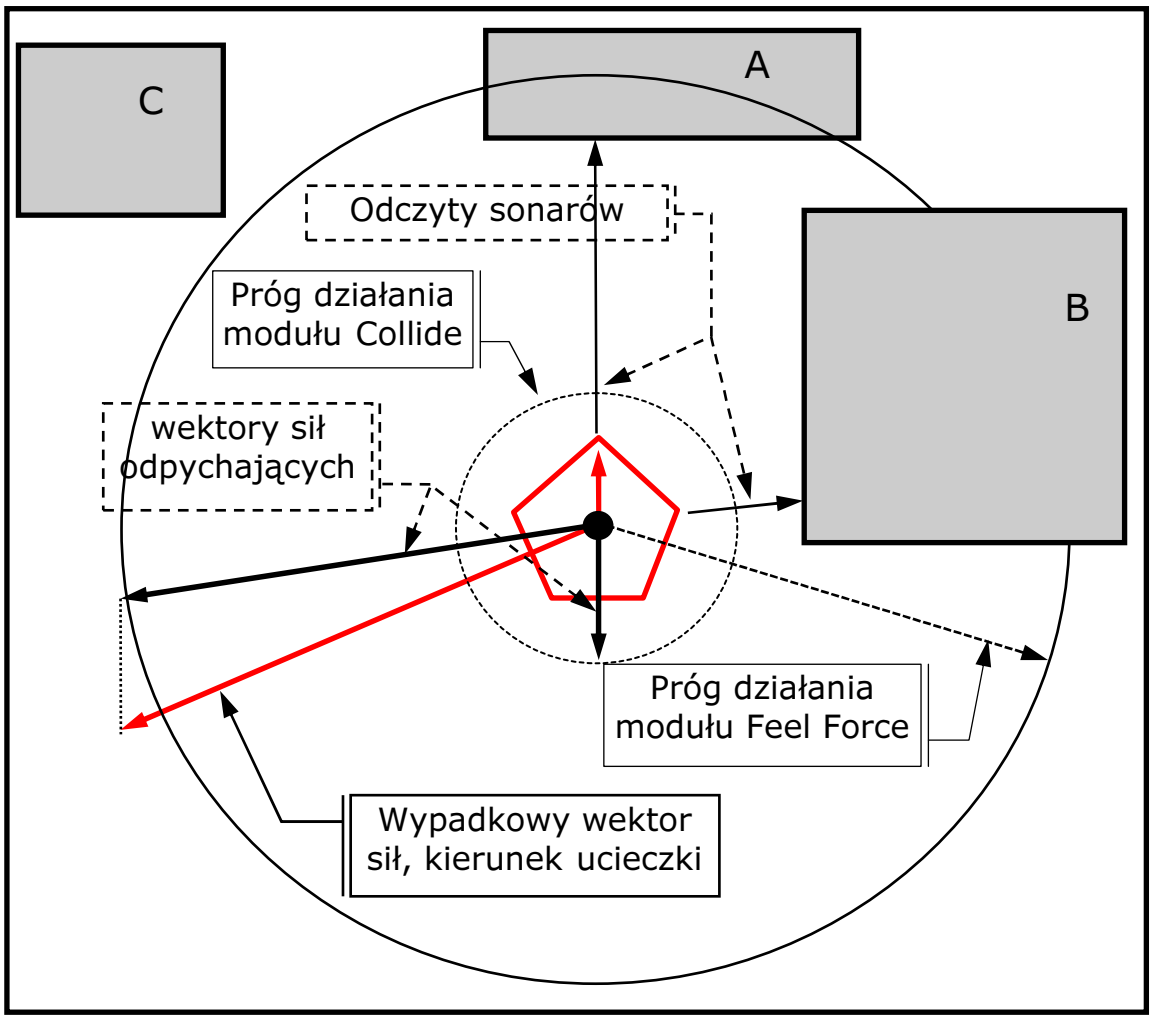
Moduł Feel Force - przetwarza odległości od przeszkód (z mapy) na siły oraz oblicza wypadkowy wektor sił oddziałujących na robota. Siła jest odwrotnie proporcjonalna do kwadratu odległości od przeszkody.

Moduł Collide - sprawdza czy bezpośrednio przed robotem nie pojawiła się przeszkoda, jeżeli tak, to wysyła sygnał *halt* do modułu sterowania silnikami, zatrzymujący robota. Jeśli robot stoi to sygnał *halt* jest ignorowany.

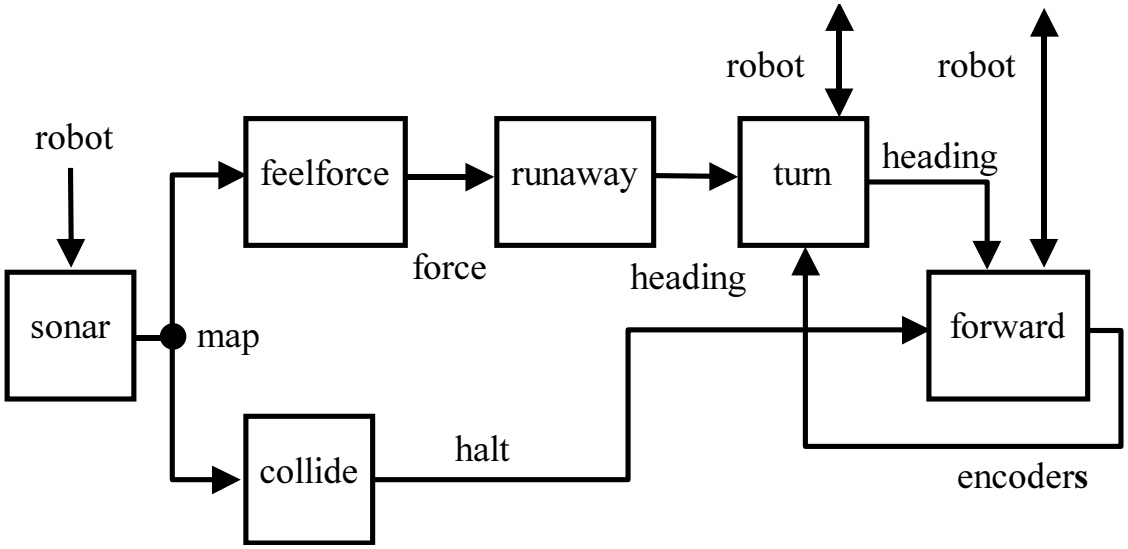
Moduł Run Away – bada wypadkowy wektor sił i jeśli wartość siły przekracza odpowiedni próg - określa dla modułu *Turn* komendę *heading* zawierająca kąt obrotu robota (w miejscu) oraz długość ruchu do przodu.

Moduł Turn - odbiera komendę *heading*. Steruje realizacją obrotu i w tym samym czasie nadaje komunikat *busy.* Po zakończeniu ruchu podaje odczyty enkodera i przekazuje sterowanie do modułu *Forward*. Nie przyjmuje następnej komendy dopóki nie zostanie zresetowany odczytem enkodera z modułu *Forward.*

Moduł Forward - steruje ruchem do przodu ale przerywa ruch po otrzymaniu sygnału *halt*. Po zakończeniu ruchu wysyła odczyty enkoderów. Ten komunikat resetuje moduł *Turn,* który jest znowu gotowy przyjąć nową komendę ruchu.



Model sterowania poziomu zerowego



Najniższy poziom sterowania struktury „Subsumption”.

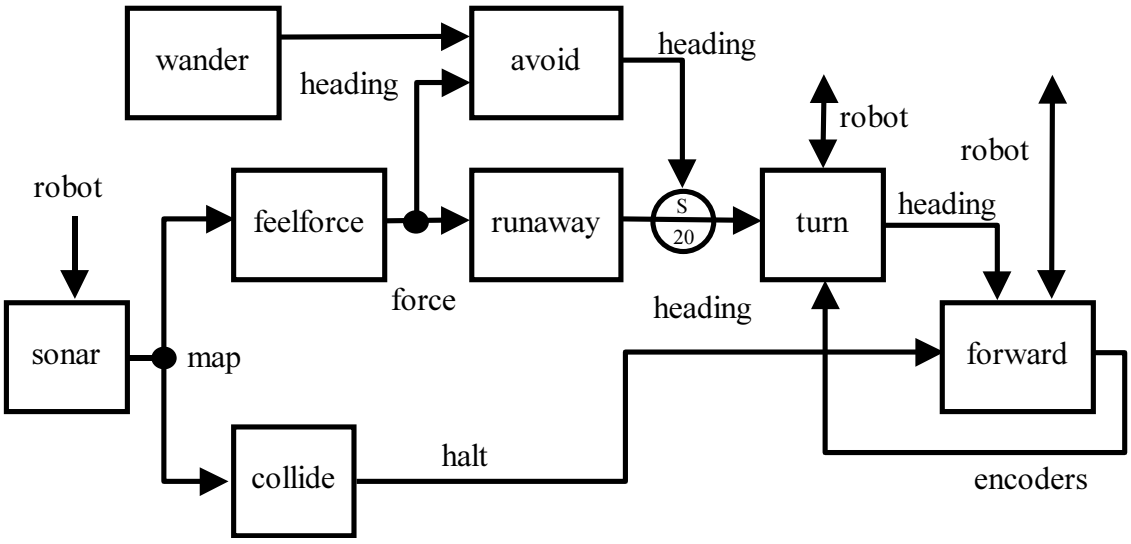
Poziom Pierwszy

Pierwsza warstwa sterowania w połączeniu z zerową daje możliwość losowego błądzenia robota bez kolizji z przeszkodami, czyli zapewnia pierwszy poziom kompetencji. Jej działanie w dużej mierze opiera się na unikaniu przez warstwę zerową kontaktu z przeszkodami.
Ponadto wykorzystuje ona prostą heurystykę do planowania ruchu tak, aby uniknąć potencjalnych kolizji, które by następnie angażowały poziom zerowy.

Do systemu dołączone są tylko dwa nowe moduły.

Moduł Wander - generuje, mniej więcej co 10 sekund, nowy kierunek dla robota.

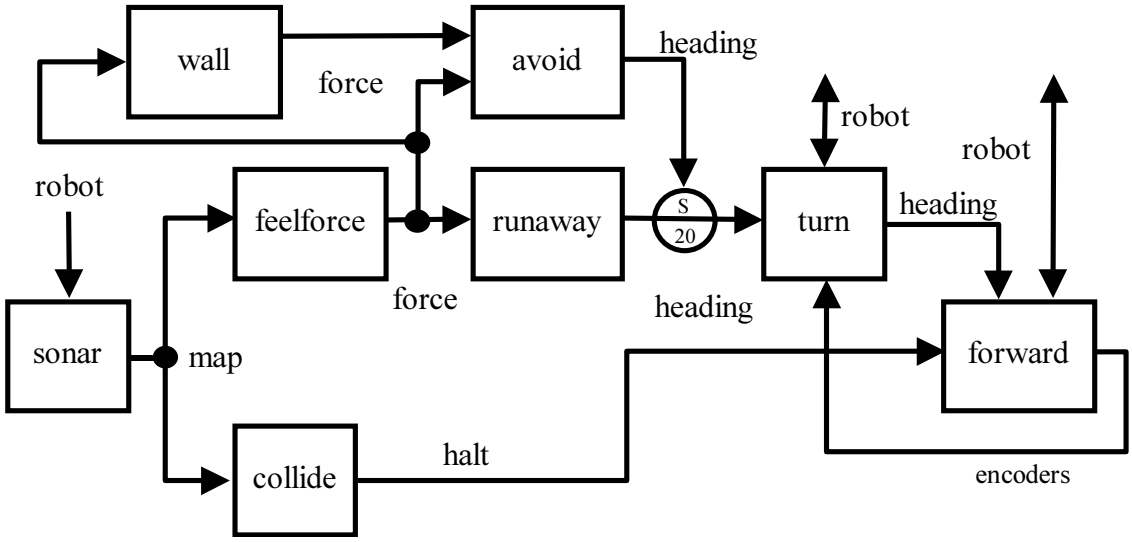
Moduł Avoid - pobiera wektor siły obliczony w warstwie zerowej i łączy go z wymaganym kierunkiem jazdy, co w wyniku daje kierunek omijający przeszkody.



System sterowania pierwszego poziomu

Robot poruszający się wzdłuż ścian

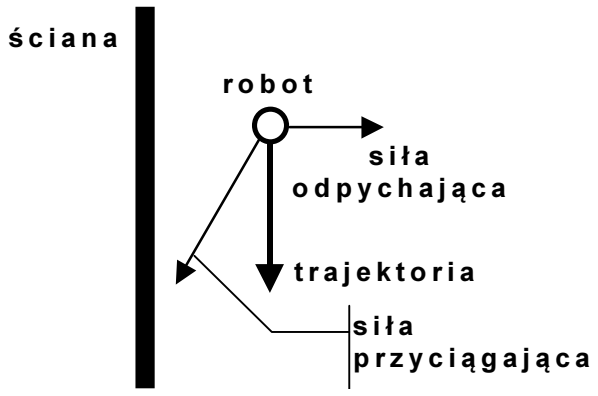
Sterowanie, które zapewnia poruszanie się robota wzdłuż ścian odpowiada pierwszemu poziomowi kompetencji, z tym że bezcelowe błądzenie zostało zastąpione świadomym penetrowaniem środowiska.



Warstwa 1 systemu śledzącego ścianę.

System ten, różni się od przedstawionego wcześniej tylko tym, że zamiast modułu *wander* zastosowano moduł *wall*.

Moduł Wall generuje wektor siły przyciągania o ustalonej wartości i kącie (zwykle 120°) względem siły odpychania dostarczonej z poziomu zerowego. Wektor ten jest następnie dodawany do wektora siły odpychającej z modułu *feelforce* w module *avoid*.

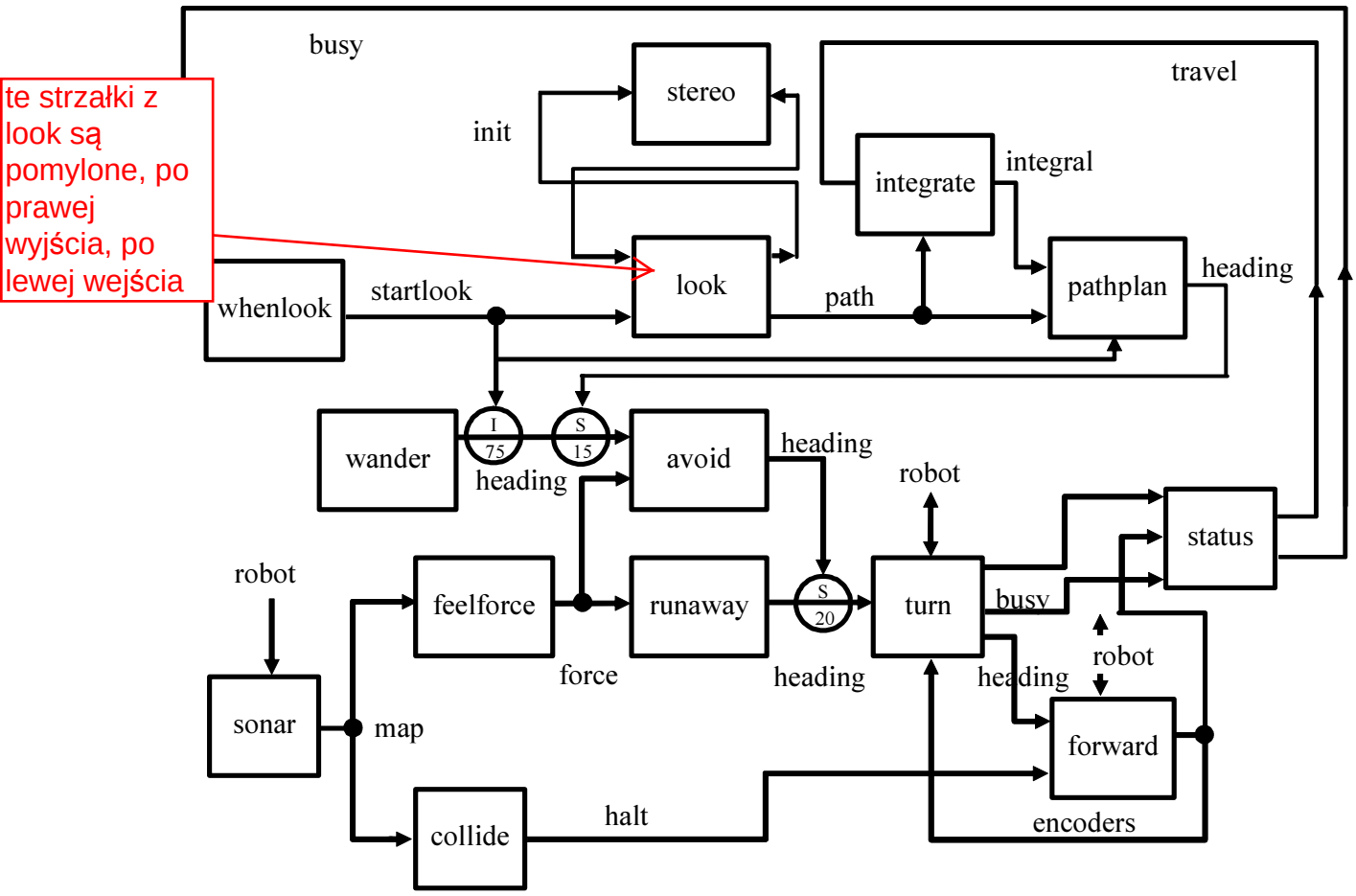


Model siłowy sterowania ruchem wzdłuż ściany.

Gdy robot jest we właściwej odległości od ściany, siła odpychająca i składowa normalna siły przyciągającej znoszą się, i pozostaje składowa styczna do ściany, która powoduje że robot porusza się równolegle do ściany.
Jeśli robot znajdzie się zbyt blisko ściany, siła odpychająca staje się silniejsza i siła wypadkowa spowoduje ruch od ściany. Natomiast, gdy robot jest za daleko od ściany, siła przyciągająca bierze górę i powoduje ruch w kierunku ściany.

Poziom drugi

Druga warstwa umożliwia „poznawanie” świata, poprzez rozpoznawanie i przemieszczanie się w kierunku najbardziej odległych miejsc (korytarze). W połączeniu z warstwą pierwszą i zerową daje to możliwość aktywnego działania robota.



Sposób zachowania się robota wyposażonego w sterowanie o drugim poziomie kompetencji pokazują ryciny zamieszczone poniżej (8, 9 i 10).

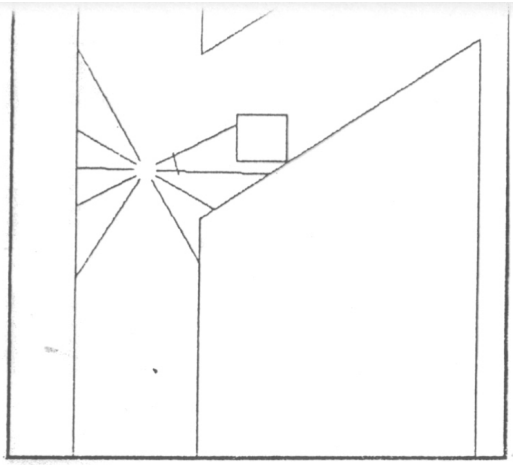


Fig. 8. Simulated robot receives 12 sonar readings. Some sonar b glance off walls and do not return within a certain time.

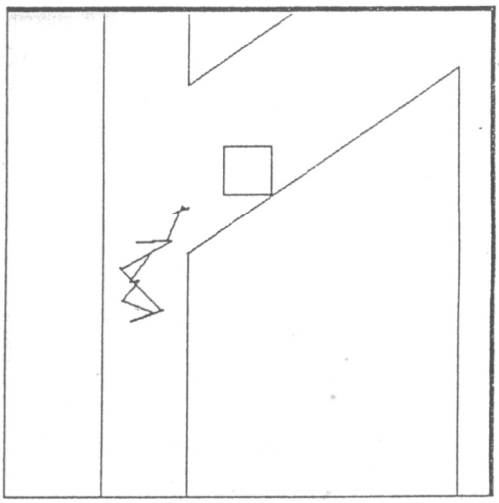
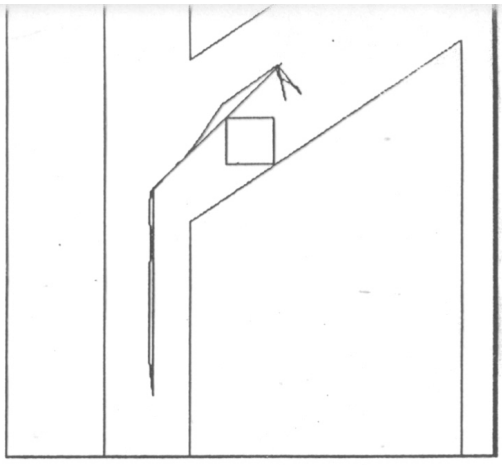
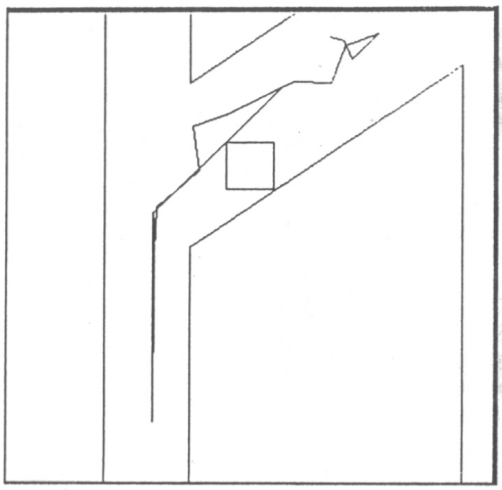


Fig. 9. Under levels 0 and 1 control the robot wanders around aimless does not hit obstacles.



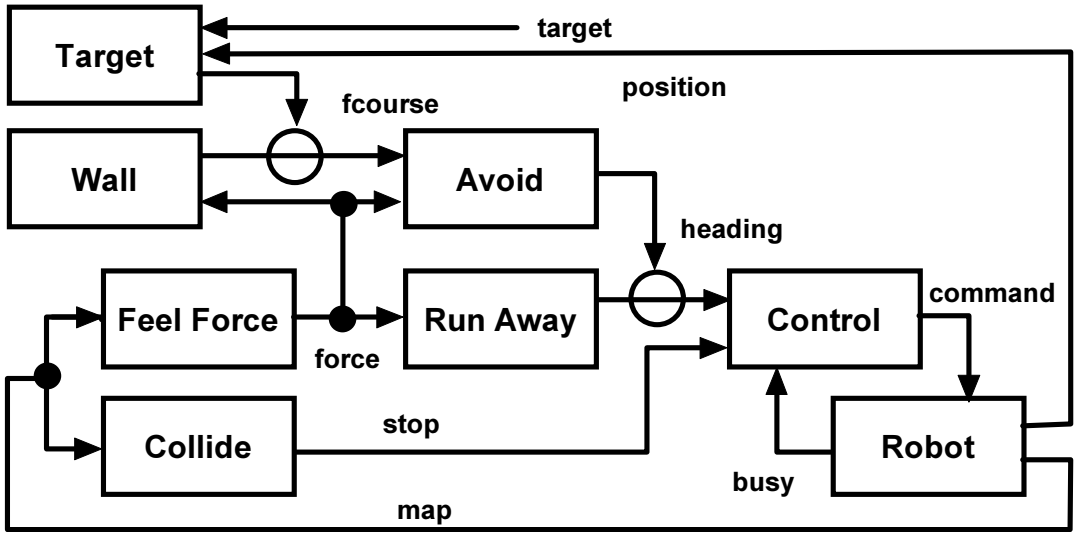
(a)

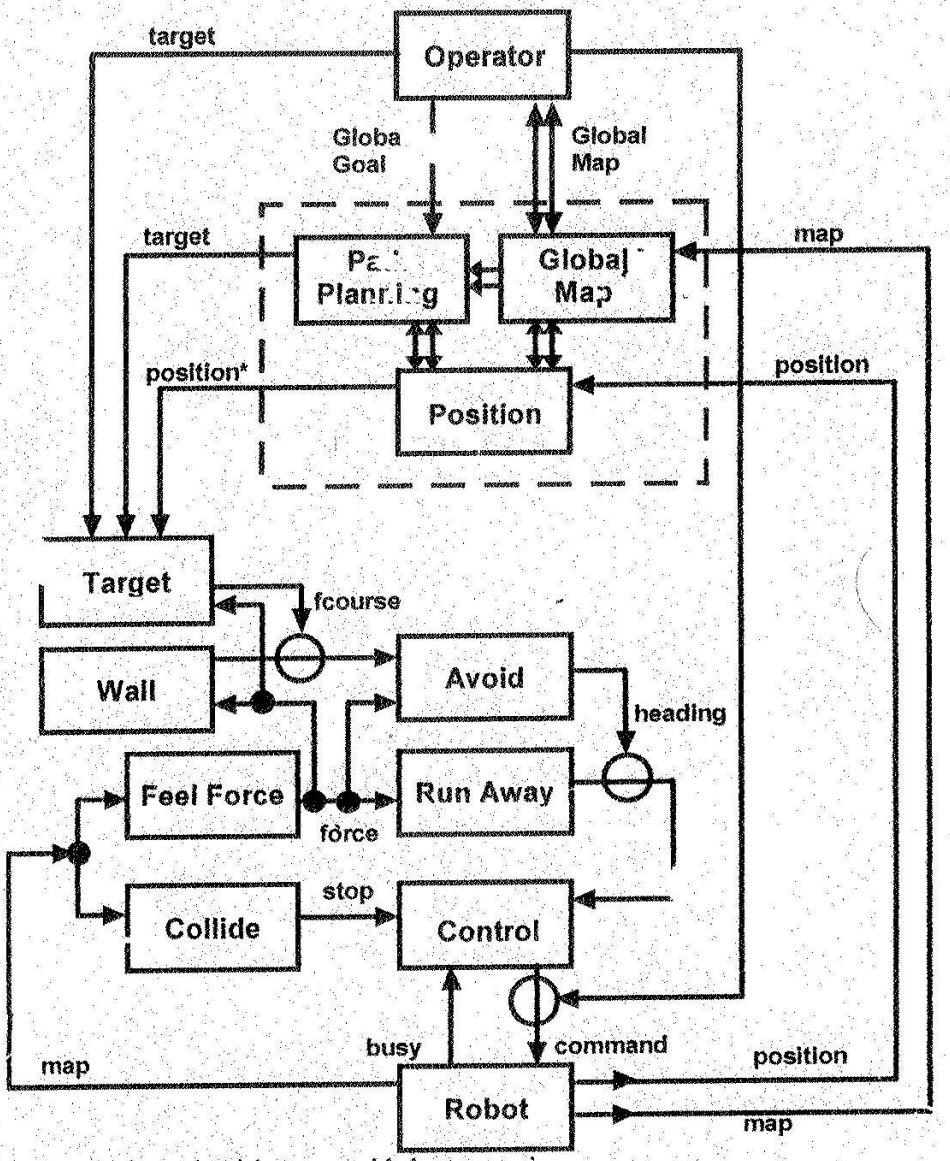


(b)

Fig. 10. (a) With level 2 control the robot tries to achieve commanded goals. The nominal goals are the two straight lines. (b) After reaching the second goal, since there are no new goals forthcoming, the robot reverts to aimless level 1 behavior.

Implementacja architektury Brooksa na robocie Ulisses





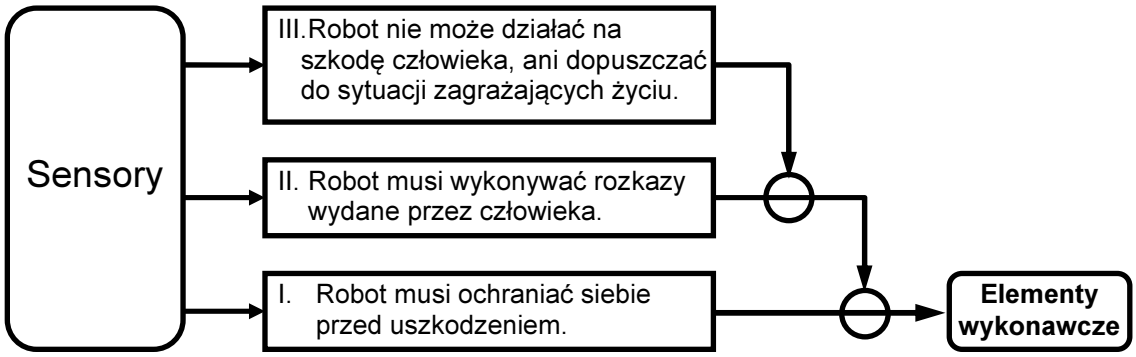
Rys.2.7. Schemat logiczny drugiej warstwy układu sterowania.

Działanie nowych modułów w układzie sterowania:

- *Global Map* - przechowuje i aktualizuje globalną mapę otoczenia. Jak wspomniano wcześniej, mapa ta ma postać tablicy, której elementy odpowiadają komórkom równomiernej siatki, odwzorowującej całą przestrzeń roboczą. Wartości elementów przedstawiają prawdopodobieństwo istnienia przeszkody w danej komórce. Mapa może być z góry wprowadzona przez *Operatora*. *Operator* może również nanosić zmiany w mapie w trakcie działania robota. Ingerencja operatora nie jest jednak konieczna, bowiem zasadniczym (koniecznym) wejściem modułu *Global Map* jest sygnał *map*, budujący mapę otoczenia robota w postaci odczytów z czujników odległości (lub innych). Oznacza to, że w razie konieczności mapa globalna może być budowana „od zera" przy użyciu samego sygnału *map*. Oczywiście wykorzystanie tego sygnału jest możliwe jedynie wtedy, gdy znana jest aktualna konfiguracja robota. Dlatego też, moduł *Global Map* ściśle współpracuje z modułem *Position*, podającym pozycję i orientację robota. Mapa otoczenia zawarta w *Global Map* jest wykorzystywana przez moduł *Path Planning* w procesie planowania ścieżki.
- *Position* - oblicza rzeczywistą konfigurację robota. Wejściem tego modułu jest sygnał *position* z robota, przedstawiający aktualną pozycję i orientację otrzymaną z pomiarów

Konkluzje na temat architektury „Subsumption”

Prawa robotyki Isaaca Asimov’a, sformułowane u jej zarania przez długi czas wydawały się bardzo futurystyczne i w życie wcielane były jedynie przez twórców literatury science-fiction. Tymczasem właśnie architektura „Subsumption” pozwala na wprowadzenie praw Asimov’a.



Architektura „Subsumption” naturalnie odzwierciedla prawa robotyki Asimov’a.

Zerowy poziom kompetencji realizuje pierwsze prawo Asimov’a nakazujące robotowi chronić swoje istnienie.
Następne poziom odpowiada drugiemu prawu, ponieważ robot jest posłuszny człowiekowi i wykonuje postawione przed nim zadania.
Rozwój prac nad architekturą „Subsumption” powinien doprowadzić również do urzeczywistnienia prawa trzeciego, które odpowiadałoby największemu poziomowi kompetencji systemu sterowania robotem.