mgr inż. Tomasz Kubik
Politechnika Wrocławska
e-mail: tkubik@@ict.pwr.wroc.pl

Zastosowanie specjalizowanych sieci neuronowych do inteligentnego sterowania robotem w dynamicznym otoczeniu

Rozprawa Doktorska

Promotor: dr hab. inż. Witold Jacak prof.

Politechnika Wrocławska **Wrocław 1998**
Instytut Cybernetyki Technicznej

Wszystkim Moim Najbliższym
poświęcam

Spis treści

Spis ważniejszych oznaczeń

$\beta_i, d_i, a_i, \alpha_i$ – parametry Denavit'a-Hartenberg'a

$\sigma_{\max}, \sigma_{\text{mean}}$ – wskaźniki jakości kalibracji

DOF – liczba stopni swobody manipulatora

$\hat{d}_q, \hat{d}_o$ – marginesy bezpieczeństwa (dla przegubów manipulatora przy zbliżaniu się, odpowiednio, do ograniczeń konstrukcyjnych i do przeszkód)

d_{qi}, d_{oi} – odległości i-tego przegubu manipulatora, odpowiednio, od ograniczeń konstrukcyjnych oraz od przeszkód

e, e_q, e_o – błąd położenia oraz błędy dodatkowe (przy uwzględnianiu, odpowiednio, ograniczeń konstrukcyjnych i obecności przeszkód)

$\epsilon^{\text{nom}}_{\max}, \epsilon^{\text{cal}}_{\max}$ – maksymalny błąd bezwzględny, odpowiednio, kinematyki nominalnej i skalibrowanej

$\epsilon^{\text{nom}}_{\text{mean}}, \epsilon^{\text{cal}}_{\text{mean}}$ – średni błąd bezwzględny, odpowiednio, kinematyki nominalnej i skalibrowanej

$\mathcal{I}$ – ilość punktów kontrolnych rozpatrywanych podczas kalibracji

J – macierz Jakobiego kinematyki

J_e, J_q, J_o – macierze pochodnych cząstkowych występujące w algorytmie gradientowym (bez ograniczeń, z ograniczeniami konstrukcyjnymi, z uwzględnianiem obecności przeszkód)

t_n – kinematyka manipulatora

$\mathcal{N}$ – liczba różnoargumentowych funkcji *sinus* występujących w rozwinięciu T_k

n, m – rozmiar, odpowiednio, przestrzeni konfiguracyjnej i przestrzeni roboczej manipulatora

$p^{\text{nom}}_{exi}, p^{\text{cal}}_{exi}, p^{\text{act}}_{exi}$ – x-owa współrzędna położenia efektora wyliczona, odpowiednio, z kinematykido nominalnej, skalibrowanej oraz kinematyki rzeczywistej (zaburzonej błędami kinematyki nominalnej) w i-tym punkcie kontrolnym

p_e, p_f – wektor współrzędnych położenia efektora oraz wektor zadanych (oczekiwanych) współrzędnych jego położenia

$\tilde{P}$ – stan kartezjański manipulatora

q – wektor konfiguracji manipulatora (wektor współrzędnych wewnętrznych)

$R_e = [n_e, o_e, a_e]$ – macierz orientacji efektora

T^n_0, T_i – kinematyka manipulatora w postaci macierzowej oraz jej i-ty element

P, R – oznaczenie, odpowiednio, przegubu translacyjnego i rotacyjnego

$v, \bar{v}, \tilde{v}, \hat{v}$ – funkcje kryterialne

Rozdział 1

Wstęp

Robotyka jest stosunkowo młodą gałęzią nauki, powstałą na styku obszarów zainteresowań matematyki, fizyki, mechaniki, lingwistyki, filozofii oraz innych nauk. Jej początki datuje się na lata sześćdziesiąte bieżącego stulecia, kiedy to badania nad robotami wstąpiły w fazę intensywnego rozwoju. Interdyscyplinarny charakter tej nowej dziedziny wynika z własności, jakimi chciano obdarzyć roboty, i ma swoje źródło w różnorodności napotykanych i rozwiązywanych przy tym problemów.

W wyniku przeprowadzonych doświadczeń, eksperymentów i analiz skonstruowano wiele robotów, aparatów i urządzeń, których liczbę przewyższa jeszcze ilość zaproponowanych modeli, opracowanych teorii, sprawdzonych algorytmów sterowania, itd. Jednak mimo tylu osiągnięć cel robotyki, jakim jest stworzenie inteligentnej, w pełni autonomicznej jednostki, nie został jeszcze osiągnięty i pozostaje nadal w sferze badań.

Wydaje się, że do osiągnięcia końcowego sukcesu przeszkodę stanowi nie tylko bariera technologiczna, ale także bariera ekonomiczna, socjologiczna, bariera starych paradygmatów. Bariery te próbuje się łamać, podpatrując i naśladując rozwiązania dostarczane przez samą przyrodę. Konstruowane są więc podobne do owadów maszyny, jednostki naśladujące ruchy zwierząt, roboty w skali mikro. Jako, że do stworzenia „prawdziwej" sztucznej inteligencji jest jeszcze daleko, aby maszyny te mogły zafunkcjonować w świecie rzeczywistym, wyposaża się je w różnego rodzaju algorytmy zachowań i systemy sterowania. W najprostszym przypadku umożliwiają one realizację zachowań, działanie których określa, podobnie jak przy odruchach bezwarunkowych, relacja akcja-reakcja (obiekty wyposażone w taki mechanizm noszą nazwę „obiektów reaktywnych"). W przypadku bardziej złożonym budowane są urządzenia, których struktura funkcjonalna odzwierciedla opracowany model zachowań (sterowanie takimi obiektami nosi nazwę „sterowania behawioralnego"). W przypadkach najbardziej skomplikowanych konstruowane są systemy inteligencji wielowarstwowej. Ich funkcjonowanie zdefiniowane jest przez sposób powiązania oraz możliwości wchodzących w ich skład podsystemów decyzyjnych, podsystemów sterowania, podsystemów monitorowania, itd. Racją przemawiającą za przyjęciem zcentralizowanej architektury takich systemów (tzn. systemów z centralną warstwą decyzyjną) jest osiąganie rozwiązań globalnych. Racją przemawiającą za przyjęciem zdecentralizowanej architektury (gdzie poszczególne warstwy są autonomiczne) jest większa szybkość w podejmowaniu decyzji.

W niniejszej rozprawie starano się dokonać syntezy inteligentnego systemu autonomicznego sterującego redundantnym manipulatorem robotycznym o przegubach obrotowych.

W obrębie rozważań znalazły się więc metody zwiększenia sprawności i szybkości obliczania kinematyki prostej i odwrotnej, planowania ruchu, bezkolizyjnego śledzenia ścieżki w czasie rzeczywistym. Wszystkie powstałe na tym gruncie idee osadzono w paradygmacie sieci neuronowych oraz obliczeń równoległych (wspomaganych obliczeniami symbolicznymi).

Zaproponowane rozwiązania zastosowane zostały w systemie reaktywnym, charakteryzującym się możliwością adaptacji do zmian otocznia. „Reaktywność" tego systemu, mimo iż anonsuje odruchowość, rozumiana jest nieco inaczej niż „reaktywność" wspomniana poprzednio. W systemie tym bowiem odruchy nie są już z góry określonymi działaniami, lecz stanowią wynik planowania zachowania na podstawie lokalnego stanu otoczenia. Ponieważ zasadnicza relacja akcja-reakcja pozostaje przy tym zachowana, przymiotnik „reaktywny" obecny jest w nazwie systemu. System reaktywny, mimo iż reaguje na wymuszenia środowiska (do obserwacji których służą mu sensory) sam w sobie nie ma zdolności gromadzenia doświadczeń. Z tego też powodu schematycznie odpowiada na każdą zaistniałą sytuację. Aby wzbogacić system reaktywny o brakującą umiejętność uczenia się, w niniejszej rozprawie zaproponowano zbudowanie nad nim warstwy nadrzędnej, tzw. warstwy edukacyjnej. Razem obie te warstwy tworzą inteligentny system autonomiczny.

Struktura niniejszej rozprawy odpowiada kolejności, w jakiej przeprowadzone zostały powyższe rozważania, począwszy od wspomnienia problemów podstawowych, kończąc zaś na propozycji inteligentnego systemu autonomicznego. I tak, w rozdziale 2, po wprowadzeniu podstawowych pojęć robotyki, przedstawione są metody obliczania kinematyki prostej i odwrotnej spotykane w literaturze (w dodatku A przedstawiona jest metoda Denavit'a-Hartenberg'a służąca do parametrycznego opisu kinematyki manipulatorów).

Następnie, w rozdziałach 3 i 4, przedstawione są propozycje nowych metod obliczania kinematyki prostej i odwrotnej dla manipulatorów redundantnych. W metodach tych kinematyka prosta obliczana jest przez zsyntetyzowane, z wykorzystaniem obliczeń symbolicznych, sinusoidalne sieci neuronowe (podrozdział 3.1). Dzięki zdolnościom sieci neuronowych do uczenia się, otrzymane kinematyki dają się łatwo i szybko kalibrować. Jak pokazano w podrozdziale 3.2, do kalibracji neuronowej kinematyki znakomicie nadaje się algorytm *BP* (dokładny jego opis znajduje się w dodatku B). Ponieważ synteza sieci neuronowych odbywa się z wykorzystaniem obliczeń symbolicznych, topologia tych sieci dokładnie odpowiada równaniom kinematyk. Z tego powodu są one sieciami specjalizowanymi do rozwiązywania konkretnego problemu jakim jest obliczanie kinematyki prostej, a nie traktowane jako „czarna skrzynka", którą można nauczyć dowolnych odwzorowań. Dalsza specjalizacja tych sieci polega na ich sprzęgnięciu w jeden system w celu otrzymania neuronowej implementacji iteracyjnego algorytmu obliczania kinematyki odwrotnej.

W podrozdziałach 4.1, 4.2, 4.3 oraz 4.4 pokazano, jak syntetyzuje się sprzężone sieci neuronowe implementujące algorytmy obliczania kinematyki odwrotnej bez ograniczeń oraz z ograniczeniami konstrukcyjnymi, a także ograniczeniami wynikającymi z obecności przeszkód w scenie roboczej manipulatora (w dodatku C przedstawiono modyfikację algorytmu obliczania kinematyki odwrotnej, tzw. algorytm tunelowy, zapewniającą osiąganie rozwiązań globalnych). Wykazano, że dzięki równoległej implementacji, wprowadzenie kolejnych modułów uwzględniających obecność ograniczeń nie wpływa na czas wykonywania jednej iteracji algorytmu.

W następnym z kolei rozdziale 5, obok przedstawienia metod reprezentacji sceny, pokazano zastosowanie zaproponowanych wcześniej algorytmów do planowania ruchów. Przy tej okazji zaprezentowano neuronowe metody obliczania odległości, planowania ścieżki

oraz dokonywania fuzji sygnałów sensorycznych.

Wszystkie zaproponowane metody wykorzystano w końcu jako części składowe warstwy reaktywnej inteligentnego systemu autonomicznego, którego koncepcję zawiera rozdział 6. System ten składa się z dwóch warst: warstwy edukacyjnej (gromadzącej informacje o otoczeniu manipulatora, dokonującej jej generalizacji i planującej akcje w oparciu o zgromadzone doświadczenia) oraz warstwy reaktywnej (odpowiedzialnej za wykonanie zaplanowanych akcji w czasie rzeczywistym). W podrozdziale 6.2 zaproponowano i szczegółowo omówiono strukturę, jaką powinna posiadać warstwa reaktywna. W podrozdziale tym przedstawiono opis zasady funkcjonowania takich elementów warstwy reaktywnej jak: system podejmowania decyzji, moduł planowania ruchu, moduł krokowego planowania dynamicznej trajektorii, reaktywny sterownik neuronowy.

W rozdziale 7 pokazano, do jakich zagadnień koordynacji pracy robotów można wykorzystać zaproponowane rozwiązania. Przedstawiono tam przykład koordynacji współpracy dwóch manipulatorów wspólnie trzymających obiekt oraz przykład współpracy manipulatorów we wspólnym gnieździe roboczym.

Poprawność i słuszność opracowanych metod potwierdzają wyniki symulacji zamieszczone w każdym z opisujących je rozdziałów. O możliwości zaimplementowania zaproponowanych metod oraz ich efektywności świadczą zamieszczone w rozdziale 8 wyniki eksperymentów przeprowadzonych na rzeczywistym robocie w specjalnie do tego celu stworzonym systemie badawczym. Opis charakterystyki techniczno-funkcjonalnej tego systemu (składającego się z trzech głównych elementów: manipulatora *AdeptOne*, systemu sensorycznego *SonaRanger*, komputera zarządzającego typu PC) zawiera podrozdział 8.1.

Reasumując, niniejsza rozprawa prezentuje następujące tezy:

1) zastosowanie specjalizowanych sieci neuronowych umożliwia efektywne i szybkie obliczanie dowolnych kinematyk prostych i odwrotnych dla manipulatorów redundantnych,

2) sprzężenie powyższych sieci z modułami analizującymi sygnały sensoryczne umożliwia planowanie bezkolizyjnych ruchów manipulatora w czasie rzeczywistym,

3) planowanie on-line bezpiecznych ruchów wraz z neuronowym sterownikiem dynamiki ruchu umożliwia syntezę autonomicznego systemu sterowania manipulatorem w środowisku dynamicznym.

Słuszność tez potwierdzają przedstawione w rozprawie teoretyczne rozważania oraz wyniki przeprowadzonych symulacji i eksperymentów. Podsumowanie całości osiągniętych rezultatów wraz z perspektywami ich wykorzystania oraz propozycjami kierunków dalszych badań zamieszczono w rozdziale 9.

Rozdział 2

Metody obliczania kinematyki prostej i odwrotnej

Niniejszy rozdział traktować będzie o kinematyce prostej i odwrotnej sztywnych manipulatorów robotycznych. Na początek, w podrozdziale 2.1, w skrócony sposób przedstawione zostaną niektóre z podstawych pojęć robotyki (jak manipulator, kinematyka), które najczęściej pojawiać się będą na stronach niniejszej rozprawy*. Następnie, w podrozdziale 2.2, przedstawione zostanie zestawienie różnych metod obliczania kinematyki manipulatorów.

2.1 Kinematyka prosta i odwrotna

Z punktu widzenia teorii mechanizmów *manipulator* jest to otwarty łańcuch kinematyczny, którego kolejne ogniwa (ramiona) połączone przegubami tworzą pary kinematyczne należące do piątej klasy, tzn. pary, posiadające tylko jeden stopień swobody†. Dlatego też liczba stopni swobody jaką dysponuje cały manipulator (*DOF -degree of freedom*), równa jest ilości jego przegubów.

Stan manipulatora można opisywać na dwa sposoby. Pierwszy z nich polega na podaniu wektora $q = (q_i \mid i = 1, \ldots, n)^T$, gdzie q_i reprezentuje kąt obrotu ramienia w przypadku, gdy i-ty przegub jest przegubem rotacyjnym, (taki przegub oznaczany jest literą R – *rotational*), lub wartość przesunięcia ramienia, gdy i-ty przegub jest przegubem translacyjnym, (oznaczay literą P – *prismatic*). Zakres zmian $q_i \in [q_i^{min}, q_i^{max}] = Q_i \in \mathbb{R}$ zależy od technicznych możliwości zmian położenia przegubu. Wektor $q \in Q = Q_1 \times Q_2 \cdots \times Q_n \in \mathbb{R}^n$ nazywany jest *wektorem konfiguracji manipulatora* lub po prostu *konfiguracją*. Zmienne q_i ze względu na ich charakter nazywane są *współrzędnymi wewnętrznymi*.

Drugi sposób wymaga wprowadzenia zewnętrznego kartezjańskiego układu bazowego. Stan manipulatora opisuje się wtedy wektorem punktów z przestrzeni bazowej $\tilde{P} = (P_i \mid i = 1, \ldots, n+1)$, reprezentującym położenia jego kolejnych przegubów i efektora. Tak opisany stan nazywany jest *stanem kartezjańskim manipulatora*.

Specjalne znaczenie przy opisie stanu manipulatora ma określenie położenia i orientacji jego efektora w zewnętrznym układzie bazowym (a więc we *współrzędnych zewnętrznych*)

*Pojęcia podstawowe dokładniej omówione są w literaturze, zobacz [Cra81, SV89].

†Wykluczenie występowania par kinematycznych o większej liczbie stopni swobody uwarunkowane jest trudnościami technicznymi przy przenoszeniu do nich napędu. Poza tym, parę o liczbie stopni swobody większej niż jeden można zawsze traktować jako złożenie par kinematycznych piątej klasy.

w zależności od wartości jego współrzędnych wewnętrznych. Zależność tą wyraża funkcja, nazywana *kinematyką prostą manipulatora* lub po prostu *kinematyką*. Samo zadanie wyznaczenia funkcji kinematyki nosi zaś nazwę *prostego zadania (problemu) kinematyki*.

Z formalnego punktu widzenia, kinematykę definiuje się (lokalnie) jako analityczne odwzorowanie z *przestrzeni wewnętrznej* $\mathbb{R}^n$ w *przestrzeń zewnętrzną (roboczą)* $\mathbb{R}^m$:

$$t_n : \mathbb{R}^n \to \mathbb{R}^m. \tag{2.1}$$

Przestrzeń wewnętrzna jednoznacznie określona jest przez dziedzinę współrzędnych wewnętrznych Q ($n = DOF$ jest rozmiarem tej przestrzeni). Przestrzeń zewnętrzna natomiast zdefiniowana być może na wiele sposobów. Jej definicja zależy bowiem od tego, jaki układ współrzędnych będzie wykorzystany do wyrażenia położenia i orientacji efektora.

Zazwyczaj współrzędne położenia efektora, $p_e = (p_{ex},\ p_{ey},\ p_{ez})^T$, wyraża się w przestrzeni kartezjańskiej $\mathbb{R}^3$, natomiast jego orientację wyraża się we współrzędnych oś/kąt, kątami Eulera lub kątami roll-pitch-yaw (RPY).

Przyjmując $(\mathbb{R}^3, RPY)$ za współrzędne zewnętrzne, kinematykę manipulatora wyrazić można następującym *równaniem kinematyki*:

$$t_n(q) = (p_{ex},\ p_{ey},\ p_{ez},\ \varphi,\ \theta,\ \psi)^T. \tag{2.2}$$

Równanie (2.2) zróżniczkowane po czasie przyjmuje postać:

$$\dot{t}_n(q) = J(q)\dot{q} \tag{2.3}$$

Występująca w nim macierz pochodnych cząstkowych $J(q) = \frac{\partial t_n}{\partial q}$ nosi nazwę *macierzy jakobiego kinematyki*.

Korzystając z rachunku macierzowego[‡], równanie kinematyki można również przedstawić w formie macierzowej (*macierzowe równanie kinematyki*):

$$T_0^n(q) = \begin{bmatrix} R_e & p_e \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} n_e & o_e & a_e & p_e \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{2.4}$$

w którym R_e jest macierzą orientacji o kolumnach $n_e = [n_{ex}, n_{ey}, n_{ez}]$, $o_e = [o_{ex}, o_{ey}, o_{ez}]$, $a_e = [a_{ex}, a_{ey}, a_{ez}]$, a p_e wektorem położenia efektora manipulatora.

W *odwrotnym zadaniu (problemie) kinematyki* za cel stawia się znalezienie takiej wewnętrznej konfiguracji manipulatora $q = (q_i \mid i = 1, \ldots, n)^T$, która zapewnia uzyskanie zadanej orientacji i położenia efektora w zewnętrznym układzie bazowym. Inaczej mówiąc, w zadaniu tym chodzi o znalezienie odwzorowania odwrotnego do kinematyki prostej (czyli *kinematyki odwrotnej*). Ponieważ funkcja kinematyki wiąże współrzędne zewnętrzne ze współrzędnymi wewnętrznymi w sposób nieliniowy, rozwiązanie odwrotnego zadania kinematyki jest bardziej skomplikowane niż zadania prostego. Co więcej, w przypadku *manipulatorów redundantnych* ($n > m$) rozwiązań odwrotnego problemu kinematyki może być nieskończenie wiele.

[‡]Macierzowe równanie kinematyki wyprowadza się, wymnażając macierze transformacji pomiędzy układami współrzędnych związanymi z kolejnymi przegubami. W dodatku A przedstawiony jest algorytm pozwalający na wyznaczenie macierzowego równania kinematyki, bazujący na metodzie Denavit'a-Hartenberg'a.

2.2 Podział metod obliczania kinematyki prostej i odwrotnej

Wykorzystaniu rachunku macierzowego, współrzędnych uogólnionych, metody Denavit'a-Hartenberg'a stało się pewnym standardem przy rozwiązywaniu prostego zadania kinematyki. Wiele metod obliczania kinematyki prostej opiera się właśnie na takich podstawach. Nieco inaczej wygląda sprawa z wyróżnieniem standardów przy rozwiązywaniu odwrotnego zadania kinematyki. Zadanie to bowiem doczekało się opracowania wielkiej różnorodności stosowanych w nim metod, a co za tym idzie, wielu standardów. W niniejszej rozprawie zdecydowano się wyróżnić, biorąc pod uwagę wykorzystywane narzędzia, następujące grupy metod obliczania kinematyki odwrotnej: metody symboliczne, metody numeryczne, metody z wykorzystaniem sieci neuronowych. Zaproponowany podział znajdzie swoje odbicie w kolejnych podrozdziałach. Ponieważ niejednokrotnie przy obliczaniu kinematyki odwrotnej występuje konieczność obliczenia kinematyki prostej, w podrozdziałach tych znajdzie się również miejsce dla metod obliczania kinematyki prostej.

2.2.1 Metody symboliczne

Historia powstania symbolicznych metod obliczania kinematyki prostej i odwrotnej, z punktu widzenia informatyki, wiąże się nierozerwalnie z powstaniem i rozwojem symbolicznych języków programowania (jak np. LISP). Jednak korzeni samych metod należy szukać dużo głębiej, sięgając wstecz przed narodziny wspomnianych języków. Kinematykę manipulatorów, traktowaną jako zagadnienie podrzędne, zaczęto obliczać przy okazji wyznaczania równań dynamiki.[§] W [KVTK93] przedstawiono krótki rys historyczny MCAE (*Mechanical Computer-Aided Engineering*) - powstałego już ponad 20 lat temu kierunku badań inżynierskich. W jego ramach, dzięki zastosowaniu komputerów i nowych technik programowania, zagadnienia obliczania dynamiki zautomatyzowano do tego stopnia, że podanie samych tylko parametrów (fizycznych i geometrycznych) manipulatora pozwala na wygenerowanie jego modelu matematycznego, wizualizację kształtu i ruchów, wyznaczanie sterowań.

Generalnie, pakiety programów MCAE można podzielić na dwie grupy:

- programy nie produkujące symbolicznego wyjścia,
- programy dostarczające symboliczne wyjście w postaci programów gotowych do kompilacji lub też w postaci równań opisujących manipulator.

W innym podziale pakietów programów wyróżnia się:

- programy oparte na *general-purpose computer algebra systems*,
- programy oparte na specyficznych strategiach manipulacji symbolicznych.

Kluczową rolę w powstaniu metod symbolicznych odegrało usystematyzowanie opisu modelu kinematycznego robota oraz wykorzystanie metod algebry liniowej (współrzędne uogólnione, rachunek macierzowy). W [DH55] zaproponowano metodę, w której każde

[§] Zostało dowiedzione, że współczynniki w równaniach dynamiki można wyrazić, podobnie jak równania kinematyki, za pomocą wielomianów trygonometrycznych [Kir84, VK85].

ramię manipulatora można opisać za pomocą zestawu czterech parametrów $(\beta_i, d_i, a_i, \alpha_i)$[¶]. Parametry te odpowiadają czterem homogenicznych transformacjom, jakich należy dokonać, aby przetransformować układ współrzędnych związany z poprzednim przegubem (lub układem bazowym) do układu związanego z przegubem bieżącego ramienia. Jednak cztery parametry w przestrzeni euklidesowej nie wystarczają do opisu wszystkich możliwych transformacji, co więcej, w niektórych przypadkach może dojść do niejednoznaczności w doborze parametrów. Dlatego też metoda z [DH55] jest często modyfikowana, [Cra81]. Jednak żadne modyfikacje metody opisu nie sprawią, aby możliwym stało się uzyskanie analitycznego rozwiązania odwrotnego problemu kinematyki dla dowolnego manipulatora. Jeśli więc mówi się o analitycznych metodach rozwiązywania odwrotnego problemu kinematyki, ma się na myśli manipulatory nieredundantne. Z teorii wiadomo, [Pie68], że rozwiązania takie istnieją, jeśli spełniony jest pewien warunek ograniczający ich konstrukcję. Klasyfikacji manipulatorów o 6 stopniach swobody spełniających ten warunek dokonano w [KB91]. Po wprowadzeniu zmodyfikowanych parametrów Denavit'a-Hartenberg'a wyróżniono tam pięć klas modeli manipulatorów: 1) xxxRRR, 2) RRRxxx, 3) xxRRRx, 4) xRRRxx, 5) PPPRRR, PRPRPR, ... (xxxRRR oznacza manipulator, którego pierwsze trzy przeguby są typu R lub P, następne zaś trzy przeguby są typu P). Jak widać, w pierwszych czterech klasach ujęto manipulatory spełniające warunek przecinania się osi trzech kolejnych przegubów rotacyjnych, ostatnia zaś klasa zawiera manipulatory posiadające 3 przeguby translacyjne. W sumie pięć wymienionych klas zawiera 52 struktury manipulatorów, charakteryzujących się możliwością podziału odwrotnego problemu kinematyki na dwa podproblemy. W pierwszym problemie szuka się trzech niewiadomych opisujących położenie, w drugim - trzech niewiadomych opisujących orientację manipulatora. Dla każdej z klas rozwiązanie odwrotnego problemu kinematyki podane jest w postaci analitycznej (konkretnego wzoru). Dlatego wystarcza zakwalifikować dany manipulator do którejś z klas, aby natychmiast uzyskać szukane rozwiązanie.

Zupełnie inne podejście do problemu rozwiązywania kinematyki odwrotnej zaprezentowano w [MOB94]. Metoda, którą tam przedstawiono (usystematyzowana metoda Raghavan'a-Roth'a) łączy obliczenia symboliczne z obliczeniami numerycznymi. W pierwszej jej fazie na podstawie macierzowego równania kinematyki (używa się parametrów Denavit'a-Hartenberg'a, zadana orientacja i położenie mają postać macierzy), budowany jest układ 14 równań. Po pewnych przekształceniach i podstawieniach ($\sin q_i = 2x_i/(1 + x_i^2), \cos q_i = (1 - x_i^2)/(1 + x_i^2), x_i = \tan(q_i/2)$, i-odpowiednie) równania te można sprowadzić do układu równań wielomianowych. Rozwiązanie tego układu znajduje się poprzez wyznaczenie wielomianu charakterystycznego manipulatora, zależnego tylko od jednej zmiennej pomocniczej x_i (manipulatory z wielomianem charakterystycznym stopnia nie większego niż 4 noszą nazwę manipulatorów analitycznych; dla manipulatorów typu 6R stopień tego wielomianu wynosi 16)[‖]. W następnej fazie numerycznie znajdowane są miejsca zerowe wielomianu charakterystycznego, a po podstawieniu odwrotnym obliczana jest wartość szukanej zmiennej wewnętrznej q_i. Mając znalezioną pierwszą niewiadomą, poprzez kolejne podstawienia do wyprowadzonych równań, znajdowane są wartości pozostałych niewiadomych. W rezultacie znajdowane są wszystkie rozwiązania odwrotnego problemu kinematyki. Podobnie, jak to było w przypadku poprzedniej metody, przytoczone postępowanie nie zapewnia uzyskania rozwiązania dla dowolnych manipulatorów.

[¶]Definicja parametrów znajduje się w dodatku A.

[‖]Pewnych analogii do tej metody można doszukać się w metodzie baz Gröbnera [Buc65, Buc85].

Ich ilość jest ograniczona do 42 typów manipulatorów o 6 stopniach swobody, ujętych w ramy 4 klas: 6R, 5R1P 4R2P 3R3P. Dzięki specjalnemu systemowi oznaczania geometrii manipulatorów, procedura znajdowania rozwiązań została zautomatyzowana i oprogramowana w systemie $Maple^{TM}$.

Automatyzacja obliczeń kinematyki odwrotnej w jeszcze innym aspekcie została ujęta w [RG94]. Według propozycji jej autorów automatyzacja znajdowania rozwiązań powinna zacząć się już na poziomie wyznaczania parametrów Denavit'a-Hartenberg'a. Użytkownik-konstruktor zobowiązany byłby tylko do wyznaczenia położenia układów współrzędnych związanych z przegubami manipulatora i do określenia, względem której osi danego układu odbywa się ruch, a zmodyfikowane parametry Denavit'a-Hartenberg'a wygenerowane zostałyby automatycznie. Poprzez porównanie otrzymanych parametrów z parametrami wcześniej skatalogowanych manipulatorów, wybrana zostałaby odpowiednia procedura liczenia kinematyki odwrotnej. Jeżeli nowo utworzony manipulator nie odpowiadałby żadnemu manipulatorowi z katalogu, biblioteka funkcji obliczających odwrotną kinematykę, jak również zawartość katalogu, byłaby poszerzana już przez samego użytkownika o odpowiadające nowemu manipulatorowi elementy. Zaproponowana metoda została zaimplementowana w systemie o nazwie ROBLINE.

2.2.2 Metody numeryczne.

Zastosowanie metod numerycznych związane jest głównie z rozwiązywaniem odwrotnego zadania kinematyki. Mimo niewątpliwych ich wad, jakimi są: duży nakład obliczeń i mała dokładność w porównaniu z rozwiązaniami otrzymywanymi drogą analityczną, ich użycie ma swoje głębokie uzasadnienie. Metody numeryczne stosowane są wszędzie tam, gdzie nie sposób otrzymać rozwiązania analitycznego, gdzie oprócz „czystego" odwrotnego problemu kinematyki rozpatrywane są problemy dodatkowe (np. unikanie osobliwości), gdzie spośród (nieskończenie)wielu rozwiązań wybierane jest jedno, optymalne w myśl obranego kryterium. Ze względu na sposób sformułowania problemu, numeryczne metody rozwiązywania odwrotnego zadania kinematyki można podzielić na:

- metody obliczanego sterowania, w których kinematyka odwrotna nie jest wyrażana *explicite*, lecz stanowi efekt zastosowanego sterowania;
- metody optymalizacyjne, w których kinematykę odwrotną definiuje się w terminach optymalizacji;
- metody obliczanej prędkości (przyspieszenia), występujące w literaturze od nazwą *Jacobian Control Methods*, [SS88], lub *Pseudo-Inverse Methods*, [Lie77];
- metody zbliżone do znanych z optymalizacji metod gradientowych, spotykane pod nazwą *Jacobian Transpose Methods*, [DSS88].

Przedstawiony powyżej podział nie jest ani podziałem jedynym, ani dokładnym czy też ostatecznym. O jego zgrubności decyduje fakt, iż istnieją metody, które, ze względu na narzędzie jakimi się posługują, można zakwalifikować do więcej niż jednej z przedstawionych grup. Niewykluczone, że powyższy podział trzeba będzie rozszerzyć o nowe, dopiero co powstające elementy. Zanim to jednak nastąpi, posłużymy się nim do usystematyzowania i omówienia istniejących już metod. Do tego celu wystarcza on w zupełności.

Metody obliczanego sterowania. Metody obliczanego sterowania zawdzięczają swoje pochodzenie metodom rozwijanym na gruncie teorii sterowania. Z uwagi na bardzo szeroki zakres problematyki, jaki się w nich porusza, nie sposób dokonać już nawet tylko częściowej ich analizy. Generalnie jednak można powiedzieć, że przy wyznaczaniu sterowania, zapewniającego śledzenie zadanej trajektorii w przestrzeni zewnętrznej z uwzględnieniem pewnych kryteriów dodatkowych, rozważa się w nich dynamikę manipulatora oraz sterowanie siłą i momentem, [Cra81, Kha86, AS86]. Przykład zastosowania metody obliczanego sterowania zawiera praca [Pot91]. Jej autor pokazał, jak można zdekomponować odwrotne zadanie kinematyki na dwa podzadania. Według niego o ruchu efektora powinno decydować złożenie dwóch ruchów w przegubach (*distributed positioning*): ruchu gładkiego (odpowiadającego ruchowi ramion o dużej masie) i ruchu z dużymi przyspieszeniami (ruch ramion o małej masie).

Metody optymalizacyjne. Idea zastosowania metod optymalizacyjnych zasadza się na przetransformowaniu odwrotnego problemu kinematyki w problem minimalizacji pewnego kryterium. Standardowo, dla odwrotnego problemu kinematyki bez ograniczeń przyjmuje się kwadratową postać tego kryterium, [Sas94]:

$$M(q) = f^T(q)f(q) = \sum_{i \in L} f_i(q) f_i(q), \quad f(q) \in \mathbb{R}^l, \tag{2.5}$$

gdzie $l \leq 12$, L jest zbiorem l różnych indeksów od 1 do 12 (odpowiednio do sposobu postawienia odwrotnego problemu kinematyki) oraz

$$\begin{array}{lll} f_1(q) = p_x - p_{ex}(q), & f_2(q) = p_y - p_{ey}(q), & f_3(q) = p_z - p_{ez}(q), \\ f_4(q) = n_x - n_{ex}(q), & f_5(q) = n_y - n_{ey}(q), & f_6(q) = n_z - n_{ez}(q), \\ f_7(q) = o_x - o_{ex}(q), & f_8(q) = o_y - o_{ey}(q), & f_9(q) = o_z - o_{ez}(q), \\ f_{10}(q) = a_x - a_{ex}(q), & f_{11}(q) = a_y - a_{ey}(q), & f_{12}(q) = a_z - a_{ez}(q). \end{array} \tag{2.6}$$

gdzie $p = [p_x, p_y, p_z]^T$, $n = [n_x, n_y, n_z]$, $o = [o_x, o_y, o_z]$, $a = [a_x, a_y, a_z]^T$ są zadanymi wektorami wyrażającymi, odpowiednio, pożądane położenie efektora oraz kolumny macierzy orientacji. Oczywiście, w przypadku postawienia odwrotnego problemu kinematyki z wymiarem przestrzeni zewnętrznej $m = 6$, wystarcza, aby w kryterium M zbiór indeksów L składał się z sześciu elementów (trzech pierwszych indeksów od 1 do 3 oraz trzech następnych, odpowiednio dobranych spośród indeksów od 4 do 12, [Sas94]).

W [Sas94] przedstawiono przegląd niektórych z metod optymalizacyjnych. W opracowaniu tym przedstawiono następujące metody (zobacz też [Wil90]): *Newton method, Gauss-Newton method, Levenberg and Marquardt idea, quasi-Newton method, David-Fletcher-Powell method, method of Broyden Family.* Wspomina się w nim również o hybrydowych metodach, w których korzysta się z technik dekompozycji problemu jak również z technik simpleksowych. Wynikiem przedstawionych rozważań jest tabela, porównująca poszczególne metody pod względem dokładności rozwiązania i liczby wykonywanych iteracji.

Ciekawym podejściem do zagadnień optymalizacji jest użycie algorytmów genetycznych, [MSB91], jednak ze względu na ograniczoną objętość niniejszej rozprawy, zagadnienie to nie będzie szerzej omawiane.

Metody pseudoinwersu. Jako pierwszy metodę pseudoinwersu do rozwiązania odwrotnego problemu kinematyki dla manipulatorów redundantnych zaproponował Whitnej, [Whi69]. Jednak pełne wykorzystanie metody pseudoinwersu wiąże się z nazwiskiem Liegeois, [Lie77]. Autor ten w swojej pracy zaproponował, aby, poprzez modyfikację „czystego" pseudoinwersowego rozwiązania, nadmiarowe stopnie swobody manipulatora wykorzystać do realizacji dodatkowych zadań, jak np. do omijania przeszkód, omijania osobliwości. Modyfikacja ta polega na przeprowadzeniu operacji rzutowania wektora odpowiadającego za minimalizację dodatkowego kryterium na przestrzeń zerową jakobianu. Jeśli więc równanie

$$\dot{q} = G_1 \dot{x} = dq1 \tag{2.7}$$

przedstawia „czyste" rozwiązanie pseudoinwersowe (przez q i x oznaczone są odpowiednio: wektor zmiennych wewnętrznych i wektor zmiennych zewnętrzych) i jeśli $H(q)$ jest gładką funkcją kryterialną, którą chcemy minimalizować, to

$$dq = G_1 dx + (G_2 J - I_n)\alpha \frac{\partial H}{\partial q} = dq1 + dq2 \tag{2.8}$$

przedstawia rozwiązanie zmodyfikowane. W przykładzie, który podał Liegeois, funkcja kryterialna H miała następującą postać:

$$H(q) = \frac{1}{n} \sum_{i=1}^{n} \left(\frac{q_i - a_i}{a_i - q_i^{max}} \right)^2, \tag{2.9}$$

gdzie $a_i = (q_i^{max} - q_i^{min})/2$, q_i^{max}, q_i^{min} - maksymalna i minimalna wartość zmiennych wewnętrznych.

Składniki $dq1$ i $dq2$ występujące w równaniu 2.8 nazywane są, odpowiednio, pseudoinwersowym składnikiem rozwiązania (*pseudo-inverse component*) i homogenicznym składnikiem rozwiązania (*homogeneous solution component*), [Mac89]. G_1 i G_2 są to pseudoinwersy macierzy J, $(G_2 J - I_n)$ jest operatorem rzutowania na przestrzeń zerową J, $\alpha \frac{\partial H}{\partial q} = Z$ jest przeskalowanym przez α, $(\alpha > 0)$, gradientem funkcji kryterialnej.

Dzięki rzutowaniu na przestrzeń zerową, homogeniczny składnik rozwiązania nie ma wpływu na położenie i orientację efektora manipulatora. Jednak, jak pokazał Maciejewski, [Mac89], w pewnych sytuacjach rozwiązania homogenicznego nie można zastosować. Ograniczenie to występuje przy specjalnych konfiguracjach manipulatora, w których prędkości, przyspieszenia i momenty w przegubach, obliczone z równania (2.8), przekraczają dopuszczalne wartości, [Yos84]. Mimo to, metoda pseudoinwersu jest szeroko stosowana w robotyce. Na przykład, w [KP93] stosuje się tą metodę, w połączeniu z metodami algebraicznymi, do obliczania kinematyki odwrotnej manipulatora o 7 stopniach swobody. Trudności z uzyskiwaniem nadmiernych przyspieszeń można pokonać poprzez minimalizację tych przyspieszeń (*joint torque minimization*), [CK88], jednak postępowanie to zahacza już o metody obliczanego sterowania, w których uwzględnia się dynamikę manipulatora.

Autorzy [WCM93] zauważyli celnie, że podstawą do opracowania metody rozwiązywania odwrotnego zadania kinematyki, w których dokonuje się optymalizacji jakiegoś kryterium, jest sposób jego zdefiniowania (sposób postawienia problemu). Dlatego też czasem trudno jest przystosować kryterium używane w jednej metodzie (w postaci np. równań różniczkowych 1,2-go rzędu, zbioru równań różniczkowych czy też algebraicznych) dla

potrzeb innej metody. W metodzie, którą sami zaproponowali kryterium (*performance index*) ma postać całkową:

$$minimize \quad \int_{t_0}^{t_1} G(q,\dot{q})dt. \tag{2.10}$$

Posługując się metodą mnożników Lagrange'a, dokonując pewnych podstawień, rozwiązanie odwrotnego problemu kinematyki dla powyższego kryterium otrzyma się w postaci:

$$\dot{q} = J^+\dot{x} + Z\alpha \stackrel{\triangle}{=} g_1(q,\alpha), \quad \dot{\alpha} = g_2(q,\alpha), \tag{2.11}$$

gdzie $g_1(.), g_2(.), \alpha$ zależą od sposobu zdefiniowania kryterium (2.10). W przypadkach, które rozpatrywał Won, tj. minimalizacji energii kinetycznej, $G(\theta,\dot{\theta})$, minimalizacji normy prędkości w przegubach, $G(\dot{\theta})$, oraz omijania osobliwości, $G(\theta)$, zostały obliczone dokładne formuły na g_1 i g_2.

Problem sterowania pseudoinwersem z punktu widzenia powtarzalności rozwiązań rozpatrywany był w [KH83]. W opracowaniu tym dowiedziono, że „czyste" rozwiązanie pseudoinwersowe z samej swojej natury nie pozwala na otrzymanie rozwiązań powtarzalnych, tj. rozwiązań, w których zamkniętej ścieżce w przestrzeni zewnętrznej odpowiada zamknięta ścieżka w przestrzeni wewnętrznej. Powtarzalność zapewnić natomiast może wprowadzenie do rozwiązania czynnika homogenicznego. Czynnik ten powoduje, że koszt energetyczny rozwiązania wzrasta, za to jednak minimalizuje się kryterium dodatkowe. Do oceny rozwiązań pod względem energetycznym w [KH83] zaproponowano wprowadzenie współczynnika „dobroci" (*efficiency*):

$$\mathit{eff} = \frac{||J^+\dot{x}||^2}{||J^+\dot{x}||^2 + ||\alpha Z||^2}. \tag{2.12}$$

Jak łatwo zauważyć, system (2.11) minimalizując kryterium dodatkowe dąży do zwiększenia swojej „dobroci".

Jak już powiedziano, rozwiązanie homogeniczne poprzez optymalizację może być zastosowane do wykorzystania redundancji w celu omijania przeszkód, unikania osobliwości, nie przekraczania ograniczeń konstrukcyjnych. Podobne cele można osiągnąć również w inny sposób. Zamiast poszukiwać minimum czy też maksimum można dążyć do utrzymania stałej wartości pewnej funkcji. W [WM88] został podany przepis na rozwiązanie homogeniczne, zapewniające spełnienie tak postawionego zadania. Według jego autorów, jeśli $\rho(q) = c$ jest wektorem funkcji stałych ($c \in \mathbb{R}^s$, $s \leq n$) i $\frac{\partial \rho(q)}{\partial q} = G(q)$, to wektor Z (zobacz równanie (2.11)) powinien mieć postać:

$$Z = -(I - J^+J)G^T\{G(I - J^+J)G^T\}^{-1}(GJ^T\dot{x}) + (1 - \bar{G}^T\bar{G})\delta, \tag{2.13}$$

gdzie $\bar{G} = G(I - J^+J)$, δ - dowolny wektor. Jako przykład został pokazany manipulator planarny o 3 stopniach swobody (typu potrójne wahadło), dla którego $\rho(q) = K_1(x^2 + y^2 - a^2)(x^2 + y^2 - b^2)^2 + K_2(\tan^{-1}(\frac{y}{x} - q_1))^2$, gdzie $a = r_1 + r_2 - r_3$, $b = r_1 - r_2 + r_3$, r_i - długość ramienia i, x, y współrzędne położenia efektora.

Pomysł z utrzymaniem stałej wartości funkcji kryterialnej wykorzystano również w [CSE91]. W tym przypadku metoda była zastosowana dla manipulatora o 7 stopniach swobody, z funkcją kryterialną zapewniającą zarazem omijanie osobliwości jak i zachowanie geometrii PUMY.

Metoda sterowania jakobianem była wielokrotnie usprawniana i modyfikowana. Jedną z jej odmian jest metoda rozszerzonej przestrzeni zadań (*extended task-space method*, [Lon92, Nen92, Ser93]). W metodzie tej zewnętrzną przestrzeń stanów rozszerza się o dodatkowe zmienne związane z zadaniem (*additional task-variables*). I tak równanie $J\dot{q} = \dot{x}$ agreguje się z równaniem $H\dot{q} = \dot{y}$ w jeden system równań różniczkowych ($q \in \mathbb{R}^n$, $x \in \mathbb{R}^m$, $y(q) \in \mathbb{R}^p$ - funkcja kryterialna lub wektor funkcji). W konsekwencji rozwiązanie odwrotnego problemu kinematyki otrzymuje się poprzez odwrócenie rozszerzonego jakobianu. Niestety, metoda ta nie jest wolna od wad. Wprowadza ona osobliwości związane z zadaniem (*task-space singularities*), które obok istniejących osobliwości algorytmu (*algorithm singularities*, [Lon92]) w pewnych sytuacjach uniemożliwiają znalezienie rozwiązania. Osobliwościom związanym z zadaniem próbuje się zaradzić poprzez przełączanie zadań lub przez ich ważenie, [Ser93].

Kolejną metodą z rozdziny metod sterowania jakobianem jest metoda ograniczonego jakobianu (*restricted Jacobian method*, [Nen92]). W metodzie tej definiuje się inwers uogólniony jakobianu manipulatora, $\bar{J}^+$, ograniczony przez przestrzeń zerową jakobianu zadania dodatkowego

$$\bar{J}^+ = [J(I - H^+H)]^+ \in \mathbb{R}^{n\times m} \tag{2.14}$$

oraz inwers uogólniony jakobianu zadania dodatkowego, $\bar{H}^+$, ograniczonego przez przestrzeń zerową jakobianu manipulatora

$$\bar{H}^+ = [H(I - J^+H)]^+ \in \mathbb{R}^{n\times p}. \tag{2.15}$$

Przy rozwiązywaniu odwrotnego problemu kinematyki korzysta się wtedy z równania:

$$\dot{q} = H^+\dot{y} + \bar{J}^+(\dot{x} - JH^+\dot{y}). \tag{2.16}$$

Wprawdzie również i ta metoda nie jest odporna na osobliwości, jednak przy właściwym doborze zmiennych zadania zapewnia powtarzalność.

Następną metodą, którą zalicza się do metod sterowania jakobianem jest metoda rzutowania na przestrzeń zerową gradientu funkcji kryterialnej (*null-space gradient-projection method*, [Nen92]). Metoda ta różni się od pozostałych sposobem wyznaczenia ograniczonego inwersu uogólnionego. Przy jego wyznaczaniu korzysta się z operatora rzutowania na przestrzeń zerową gradientu funkcji kryterialnej $h(q)$. Odpowiedni inwers uogólniony ma postać:

$$\tilde{J}^+ = J\left(I - \left(\frac{\partial h}{\partial q}\right)^+ \frac{\partial h}{\partial q}\right)^+. \tag{2.17}$$

Ideę rzutowania na przestrzeń zerową gradientu funkcji kryterialnej zastosowali również Lee i Kil, [LK94], z tym, że w metodzie transponowanego jakobianu.

Obliczanie pseudoinwersu macierzy jest procesem wymagającym dużych nakładów obliczeniowych. Aby nakłady te zmniejszyć, zaproponowano szereg algorytmów obliczających pseudoinwers nie wprost. Między innymi są to: Singular Value Decomposition, ([MK89]), LDL Decomposition, [Lon92], Cholesky decomposition, [Wil90, Too89]. W [CK88] zapropony został jeszcze inny sposób na obliczanie pseudoinwersu, w którym macierz jakobiego dekomponuje się zgodnie z równaniem:

$$J = J_m C, \tag{2.18}$$

gdzie J_m - regularna podmacierz $(m \times m)$ J, C_i - macierz $(m \times n)$. W efekcie dekompozycji macierzy jakobiego na dwie podmacierze pełnego rzędu, pseudoinwersowe rozwiązanie ma postać:

$$\dot{q} = C^+ J_M^{-1} \dot{x} \quad , \quad J^+ = C^+ J_m^{-1}. \tag{2.19}$$

Równanie (2.19) ma następującą interpretację fizyczną: $\dot{q}_p = J_m^{-1}\dot{x}$ - to rozwiązanie szczególne, osiągane za pomocą pierwszego zestawu przegubów, $\dot{q} = C^+\dot{q}_p$ - to rozwiązanie spełniające dodatkowe kryterium, realizowane przez pozostałe $n - m$ przegubów.

Z otrzymywaniem rozwiązań numerycznych metodą pseudoinwersu związany jest jeszcze jeden fakt, o którym zdawało sobie sprawę niewielu autorów. Otóż o rozwiązaniu, oprócz kryteriów dodatkowych, decydują również fizyczne jednostki, w których wyraża się położenie i orientację manipulatora, jego wymiary geometryczne oraz wartości prędkości. Jak pokazano w [DMB93] zmiana jednostek z [m] na [cm] w opisie długości ramion manipulatora powoduje otrzymanie zupełnie różnych rozwiązań dla każdej z nich. Dlatego też przy obliczaniu pseudoinwersu należy pamiętać o odpowiednim przeskalowywaniu jednostek.

Metody transponowanego jakobianu. Metoda transponowanego jakobianu jest owocem poszukiwań rozwiązania odwrotnego problemu kinematyki, które, posiadając zaletę rozwiązania otrzymanego metodą pseudoinwersu (ciągłość rozwiązań, uwzględnienie ograniczeń, itp.), nie wymagałoby tak dużych nakładów obliczeniowych. W metodzie transponowanego jakobianu nie oblicza się pseudoinwersu. Do znalezienia rozwiązania wystarcza znajomość jedynie kinematyki prostej, a więc $t(q)$ i $J(q)$. Jako pierwsi metodę tą, dla przypadku manipulatorów redundantnych, przedstawili Sciavicco i Siciliano, [SS88]. Zaproponowany przez nich algorytm, pracujący w układzie zamkniętym, zapewnia zbieżność zdefiniowanych wcześniej błędów do zera, a tym samym rozwiązuje odwrotny problem kinematyki. Istotę metody transponowanego jakobianu jej autorzy ujęli w poniższym twierdzeniu, [SS88]:

Twierdzenie 1 *Niech $\hat{q}(t)$, $(n \times 1)$, będzie rozwiązaniem odwrotnego problemu kinematyki odpowiadającym trajektorii zadanej $\hat{x}(t)$, $(m \times 1)$, $x(t)$ - bieżącym położeniem, $q(t)$ - bieżącą konfiguracją, $e(t) = \hat{x}(t) - x(t)$ - błędem położenia (i orientacji). Dynamika błędu e opisana jest równaniem:*

$$\dot{e} = \dot{\hat{x}} - J(q)\dot{q}. \tag{2.20}$$

Jeśli[**]

$$\dot{q} = \gamma J^T e, \quad gdzie \quad \gamma = \alpha + (e^T \dot{\hat{x}})(e^T J J^T e)^{-1}, \alpha > 0, \tag{2.21}$$

to błąd $e \to 0$, a więc $q \to \hat{q}$.

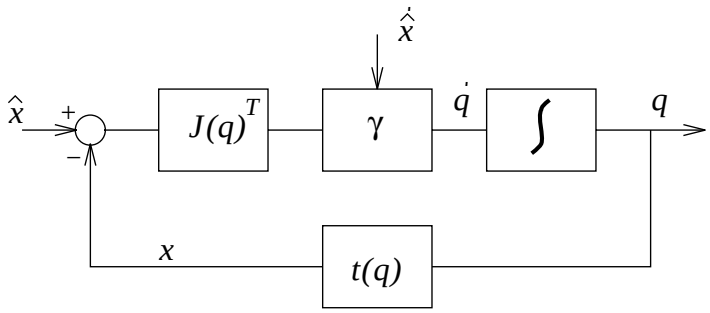
Dowód: Niech $v = 0.5 e^T e$ będzie funkcją Lapunowa błędu e. Dla $\dot{q}$ danego jak w (2.21) jej pochodna w czasie, $\dot{v} = e^T \dot{e} = -\alpha e^T J J^T e$, jest ujemnie określona w przestrzeni błędu wszędzie, poza obszarem $e = 0$, który jest atraktorem dla wszystkich trajektorii.

□

Sciavicco i Siciliano zauważyli, [SS88], że z obliczeniowego punktu widzenia bardziej znaczącym jest rozpatrywanie tylko proporcjonalnego czynnika w (2.21), tj.

$$\dot{q} = \alpha J^T e. \tag{2.22}$$

[**]Zobacz diagram na rysunku 2.1.



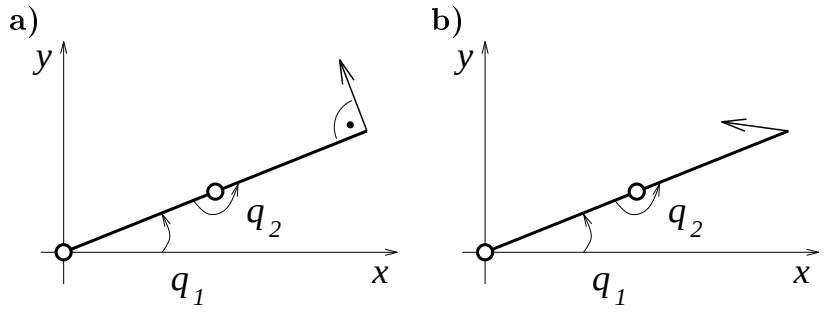
Rysunek 2.1: Diagram blokowy metody transponowanego jakobianu.

Rozwiązaniu (2.22) można nadać ciekawą fizyczną interpretację. Wiadomo, że zależność miedzy $(n \times 1)$ wektorem momentów w przegubach, τ, i wektorem sił, *zeta*, wyrażonych w przestrzeni kartezjańskiej, a przyłożonych do efektora manipulatora, wyraża się wzorem:

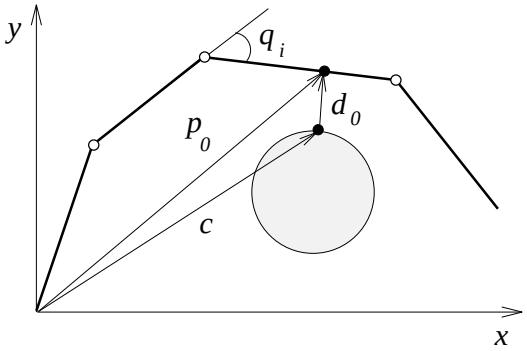
$$\tau = J^T \zeta. \tag{2.23}$$

Porównując (2.22) i (2.23) można zauważyć, że wektor αe odpowiada wektorowi sił (*elastic force vector*), który musi być przyłożony do efektora manipulatora o idealnej kinematycznej strukturze, aby zrealizowana została zadana trajektoria. Amplituda tej siły zależy od wartości błędu e, [DSS88]. Dzięki takiej interpretacji w łatwy sposób można opisać charakter osobliwości równania (2.22). Jeśli $rank(J) < m$, $J^T e = 0$ kiedy e należy do przestrzeni zerowej J^T. W świetle fizycznej interpretacji przestrzeni zerowej J^T odpowiadają takie kierunki sił w przestrzeni kartezjańskiej, że jeśli zostaną one przyłożone do efektora, to zrównoważą je mechaniczne ograniczenia manipulatora. Z punktu widzenia implementacji algorytmu, sytuacje te można rozpoznać po jednoczesnym spełnieniu dwóch warunków: $||\dot{q}|| = 0$ i $||e|| \neq 0$. Na rysunku 2.2 przedstawiono przykład manipulatora o 2 stopniach swobody, który a) bez problemu przechodzi przez konfigurację osobliwą ($J^T e \neq 0$, a więc można nadal korzystać z algorytmu (2.22)); b) musi osiągnąć wielkie prędkości w przegubach, aby zrealizować zadaną trajektorię, ($J^T e \approx 0$, a więc zbieżność algorytmu (2.22) jest zmniejszona). Różnice między przypadkami a) i b) wynikają z faktu, iż w osobliwości ($q_2 = \pi$) przestrzeni zerowej J^T odpowiada kierunek wzdłuż obu wyprostowanych ramion. Dlatego trajektoria prostopadła do tego kierunku (przypadek a)) jest „lepsza dla algorytmu" od trajektorii mającej składową wzdłuż tego kierunku (przypadek b)). Od metody transponowanego jakobianu wymaga się, aby, podobnie jak metoda pseudoinwersu, dawała możliwość uwzględniania ograniczeń. Aby warunek znajomości tylko kinematyki prostej został zachowany, ograniczenia te powinny być funkcjami zależnymi tylko od zmiennych wewnętrznych. Sciavicco i Siciliano zaproponowali dwie postacie takich ograniczeń, obie wyrażone w terminach błędu dodatkowego. Ich propozycje zostaną omówione poniżej.

1) Omijanie przeszkód. Niech $\hat{d}_0$ oznacza margines bezpieczeństwa, tzn. minimalną bezpieczną odległość manipulatora od przeszkody, c - wektor położenia najbliższego manipulatorowi punktu na przeszkodzie, p_0 - wektor położenia punktu najbliższego przeszkodzie, a leżącego na którymś z ramion manipulatora (zobacz rysunek 2.3). Niech $d_0 = p_0 - c$ oznacza wektor odległości manipulatora od przeszkody. Niebezpieczeństwo kolizji powstaje, gdy odległość manipulatora od przeszkody jest mniejsza niż przyjęty



Rysunek 2.2: Geometryczna interpretacja osobliwości w metodzie transponowanego jakobianu.



Rysunek 2.3: Położenie dwóch najbliższych sobie punktów, leżących na manipulatorze i przeszkodzie.

margines bezpieczeństwa $\hat{d}_0$. Jeśli więc e_0 oznacza błąd dodatkowy

$$e_0 = 0.5(\hat{d}_0^2 - d_0^T d_0), \quad (k \times 1), \quad k\text{-ilość aktywnych ograniczeń}, \tag{2.24}$$

to jego pochodna w czasie ma postać:

$$\dot{e}_0 = \hat{d}_0 \dot{\hat{d}}_0 - d_0^T \dot{c} - j_{d_0}^T \dot{q}, \tag{2.25}$$

gdzie $j_{d_0}^T = d_0^T J_{p_0}$, $J_{p_0} = \frac{\partial p_0}{\partial q}$. Przy założeniu, że $\dot{\hat{d}}_0 = 0$ (stały margines), i $\dot{c} = 0$ (statyczna przeszkoda), powyższe równanie upraszcza się do równania:

$$\dot{e}_0 = -j_{d_0}^T \dot{q}. \tag{2.26}$$

2) Ograniczenia na wartości zmiennych wewnętrznych. Niech q_i^{max} i q_i^{min} oznaczają, odpowiednio, maksymalną i minimalną wartość zmiennej wewnętrznej q_i. Niech $\hat{d}_q$ oznacza margines bezpieczeństwa, tzn. najmniejszą dopuszczalną odległość q_i od któregoś z ograniczeń q_i^{min} lub q_i^{max}. Jeśli więc e_q oznacza błąd dodatkowy ††

$$e_q = \hat{d}_q - d_q, (r \times 1), \quad r \text{ - ilość aktywnych ograniczeń}, \tag{2.27}$$

††Błąd e_q obliczany jest na bieżąco wzdłuż trajektorii.

przy czym $d_q = q_i - q_{imin}$ lub $d_q = q_{imax} - q_i$ (zależnie od tego, które z ograniczeń jest aktywne), to pochodna błędu w czasie ma postać:

$$\dot{e}_q = \dot{\hat{d}}_q - u_i^T \dot{q}, \tag{2.28}$$

gdzie $u_i^T = (00 \ldots \pm 1 \ldots 0)$ (+1 odpowiada q_i^{min}, -1 zaś q_i^{max}). Przy założeniu, że $\dot{\hat{d}}_q = 0$ (stały margines) powyższe równanie upraszcza się do równania:

$$\dot{e}_q = -u_i^T \dot{q}. \tag{2.29}$$

Rozszerzenie wektora błędu e z twierdzenia 1 o błędy dodatkowe przez analogię w prosty sposób prowadzi do uzyskania algorytmu rozwiązywania odwrotnego problemu kinematyki z ograniczeniami. Nietrudno dowieść, że algorytm ten, uwzględniający $v = k + r \leq n - m$ ograniczeń:

$$\dot{q} = \alpha J_e^T e_y, \quad e_y = [e, e_0, e_q]^T, \quad J_e = [J, J_{d_0}, U]^T, \tag{2.30}$$

gdzie wiersze $J_{d_0}(r \times n)$ i $U(r \times n)$ rozszerzonego jakobianu J_e zdefiniowane są odpowiednio przez j_{d0}^T i u_i^T, jest zbieżny. Sciavicco i Siciliano przeprowadzili dyskusję na temat przełączania aktywnych ograniczeń i analizę zbieżności dyskretnej wersji algorytmu.

Metoda transponowanego jakobianu bardzo szybko doczekała się modyfikacji. W jednej z nich, [DSS88], spełnianie dodatkowych kryteriów realizowane jest, podobnie jak to było przedstawiane w metodach pseudoinwersu, poprzez rzutowanie gradientu minimalizowanej funkcji kryterialnej na przestrzeń zerową jakobianu. Formułę takiego algorytmu wyraża wzór:

$$\dot{q} = -\alpha S J^T K_p \tilde{x} - \beta (I - J^T (J J^T)^{-1} J) \frac{\partial H(q)}{\partial q}, \tag{2.31}$$

gdzie $\alpha, \beta > 0 \in \mathbb{R}$, $\tilde{x} = x - x_d$, S - symetryczna, dodatnio określona macierz, $H(q)$ - minimalizowana funkcja kryterialna, $(I - J^T (J J^T)^{-1} J)$ - operator rzutowania.

O zbieżność algorytmu (2.31) decyduje właściwy dobór parametrów α, β, S. Odpowiedni dowód na zbieżność, wykorzystujący metodę funkcji Lapunowa, przedstawiono w [DSS88]. Praca [DSS88] zawiera również propozycję funkcji kryterialnej, powodującej omijanie przeszkód z zachowaniem zadanej konfiguracji. Gradient tej funkcji ma postać:

$$\frac{\partial H}{\partial q} = (K \tilde{q} + K_o \sum_{j=1}^{n} J_j^T \sum_{i=1}^{\kappa} \left(\frac{\tilde{x}_{ij}}{\tilde{x}_{ij}^T \tilde{x}_{ij}} \right)), \tag{2.32}$$

gdzie K_{obst} - współczynnik odpychania przegubów od przeszkód, K - *joint stiffness matrix*, $\tilde{q} = q - q_d$, J_j - jakobian manipulatora do przegubu j, $\tilde{x}_{ij}$ - wektor od przeszkody i do przegubu j, n i κ - odpowiednio liczba przegubów i przeszkód. W metodzie tej liczbę aktywnych ograniczeń nie zawężono do wielkości stopnia redundancji.

2.2.3 Metody z wykorzystaniem sieci neuronowych

Próby zastosowania sieci neuronowych do rozwiązywania prostego i odwrotnego problemu kinematyki zaowocowały powstaniem wielu metod, różniących się między sobą rodzajem zastosowanej sieci*, algorytmem uczenia, ideą postawienia problemu i jego rozwiązania. Ze względu na ostatnie z wymienionych kryteriów, metody te można podzielić na:

*Przegląd różnych architektur sieci neuronowych w zastosowaniach robotycznych podano w [KH89].

1) *direct inverse* - w metodach tych daną sieć neuronową uczy się odzwierciedlać wprost kinematykę odwrotną, [Kup87, LK90]. Uczenie to odbywa się w tak zwanym trybie „uczenia z nauczycielem" (*supervised learning*). Ciągiem uczącym jest zestaw par (położenia przegubów, oczekiwana pozycja efektora). Ponieważ sieci takich nie można nauczyć przekształcenia innego niż przekształcenie jeden do jeden (w parach uczących konkretnemu położeniu efektora manipulatora odpowiada dokładnie jedna konfiguracja), w przypadkach manipulatorów redundantnych za ich pomocą znajdowane jest tylko jedno (albo żadne) rozwiązanie. Porównaniem różnych algorytmów uczenia z nauczycielem, (*back-propagation*, *steepest-descent*, *grad-coniugate*, *quasi-Newton*) na przykładzie 3-warstwowej sieci z 10-ma neuronami w warstwie ukrytej (z funkcją aktywacji *tanh*), użytej do modelowania kinematyki odwrotnej dwuwahadła, przedstawiono w [RCPD95].

2) *distal learning* - w metodzie tej konstruowane są dwie sieci neuronowe, Net_f i Net_i, przeznaczone odpowiednio do obliczania kinematyki prostej i odwrotnej. Jako pierwsza uczeniu poddawana jest sieć Net_f, od której wymaga się dokładnego przybliżania kinematyki prostej. Kiedy wagi sieci Net_f są już ustalone, rozpoczyna się proces uczenia sieci Net_i. Jednak tym razem metoda uczenia jest już inna. Od ustawionych szeregowo sieci Net_f i Net_i wymaga się, aby realizowały przekształcenie tożsamościowe. Sieć Net_i uczona jest kinematyki odwrotnej przy okazji nauki właśnie tego przekształcenia. Metodą tą znajdowane jest tylko jedno rozwiązanie, [Jor92, LI95].

3) *regularization approach* - jest metodą, w której poprzez podział przestrzeni wewnętrznej i zewnętrznej oraz konstrukcję zestawu sieci neuronowych odpowiadających tym podziałom, uzyskać można co najmniej jedno rozwiązanie odwrotnego problemu kinematyki. W metodzie tej wyróżnia się cztery następujące etapy: etap 1 - podział przestrzeni wewnętrznej i zewnętrznej na regiony; etap 2 - konstrukcja modułowej sieci neuronowej, której każdy z modułów a) odpowiada jednemu regionowi b) jest nauczony kinematyki prostej danymi z regionu; etap 3 - znalezienie wielokrotnego rozwiązania kinematyki odwrotnej poprzez odwrócenie odpowiednich modułów sieci (sieci odwraca się korzystając z technik optymalizacyjnych, w których minimalizuje się jakieś kryterium z ograniczeniami wynikającymi z nauczonych wag sieci); etap 4 - wybranie spośród otrzymanych rozwiązań jednego, „najlepszego". Jakkolwiek metoda ta pozwala na otrzymanie wielu rozwiązań, jednak ich ilość jest ograniczona ilością modułów. Ponadto z uzyskaniem każdego, kolejnego rozwiązanie wiąże się problem odwracania sieci,[LI95].

4) metody obliczanego sterowania - podobnie jak to było w przypadku metod numerycznych, kinematykę odwrotną otrzymuje się tu pośrednio, poprzez znalezienie sterowania, zapewniającego realizację zadanej trajektorii. Neuronowo modeluje się dynamikę manipulatora, proponuje się też neuronowe sterowniki, [BGC94, San94].

5) Metoda sieci kontekstowych - w metodzie tej używa się sieci neuronowej o specjalnej strukturze, [YB89]. Wejścia do tej sieci podzielone są na dwie grupy. Pierwsza grupa stanowi wejście do sieci wykonującej podstawowe operacje matematyczne (sieć funkcyjna), podczas gdy druga grupa służy do określenia nastaw wag lub kontekstu, w jakim funkcja ta jest obliczana (druga grupa stanowi wejścia do sieci kontekstowej). Za pomocą takiej sieci oblicza się pożądane prędkości w przegubach przy zadanej prędkości efektora w układzie kartezjańskim, gdy kontekstem jest odwrócony jakobian.

Rozdział 3

Zastosowanie sinusoidalnych sieci neuronowych do obliczania kinematyki prostej

W standardowych metodach wykorzystujących sieci neuronowe do modelowania kinematyki manipulatora sieci te traktowane są jako „czarne skrzynki", które, dzięki zdolności do uczenia się, po pewnym okresie treningu są w stanie przybliżyć dowolną kinematykę z zadaną dokładnością. Ponadto, ze względu na równoległa architekturę, sieci neuronowe pozwalają na dokonywanie szybszych obliczeń niż miałoby to miejsce przy obliczaniu sekwencyjnym. Te dwie zalety, a więc zdolność do uczenia się oraz równoległość obliczeń, sprawiają, że w konkurencji z innymi metodami neuronowe modelowania kinematyki zajmuje pierwsze miejsce.

Zazwyczaj do neuronowego modelowania kinematyki używane są sieci typu *feedforward* z neuronami o sigmoidalnej funkcji aktywacji. W niniejszym rozdziale przedstawiony zostanie inny, niestandardowy sposób neuronowej implementacji kinematyki manipulatora. W sposobie tym architektura syntezowanej sieci predykowana będzie zadaniu, do rozwiązania którego ma ona posłużyć. Nowa sieć, dzięki zastosowaniu neuronów o sinusoidalnej funkcji aktywacji oraz strukturze dokładnie odpowiadającej równaniom kinematyki, w porównaniu z sieciami standardowymi odznaczać się będzie: 1) większą dokładnością (sieć ta nie przybliża, ale dokładnie oblicza kinematykę manipulatora); 2) krótszym procesem uczenia (dzięki temu, że synteza sieci odbywa się poprzez zastosowanie obliczeń symbolicznych, znane są nominalne, tj. odpowiadające kinematyce nominalnej, nastawienia jej wag).

Pomysł wykorzystania sieci neuronowych o specjalizowanych neuronach (z funkcją aktywacji *sinus* *) do modelowania równań kinematyki wiąże się ze spostrzeżeniem, że występujące w równaniach tych wyrażenia trygonometryczne po przekształceniach dają się wyrazić jako ważone sumy funkcji *sinus*. Ważone sumy funkcji *sinus* zaś w prosty sposób zamodelować można jako trzywarstwowe sieci neuronowe. Szczegóły wyjaśniające istotę tego pomysłu przedstawione zostaną w niniejszym rozdziale na odnośnym przykładzie.

*Stąd też nazwa *sinusoidalne sieci neuronowe*.

3.1 Synteza sieci neuronowej do obliczania kinematyki prostej

Analizując macierzowe równanie kinematyki, otrzymane z metody Denavit'a-Hartenberg'a[†], opisujące manipulator o n przegubach rotacyjnych

$$P = T_0^n(q) = \begin{bmatrix} T_1(q) & T_2(q) & T_3(q) & T_4(q) \\ T_5(q) & T_6(q) & T_7(q) & T_8(q) \\ T_9(q) & T_{10}(q) & T_{11}(q) & T_{12}(q) \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.1}$$

łatwo zauważyć, że poszczególne elementy macierzy $T_0^n(q)$, T_k (numerowanie elementów macierzy kinematyki tylko jednym indeksem jest podyktowane uproszczeniem zapisu wyrażeń w pojawiających się w dalszej części rozprawy), mają postać ważonej sumy następujących iloczynów:

$$\prod_{i=1}^{n} g(q_i), \tag{3.2}$$

gdzie $g(q_i)$ reprezentuje $\sin q_i$, $\cos q_i$ lub 1. Iloczyny te w prosty sposób przekształcić można do postaci addytywnej, w której każdy z sumowanych elementów wyraża się przez:

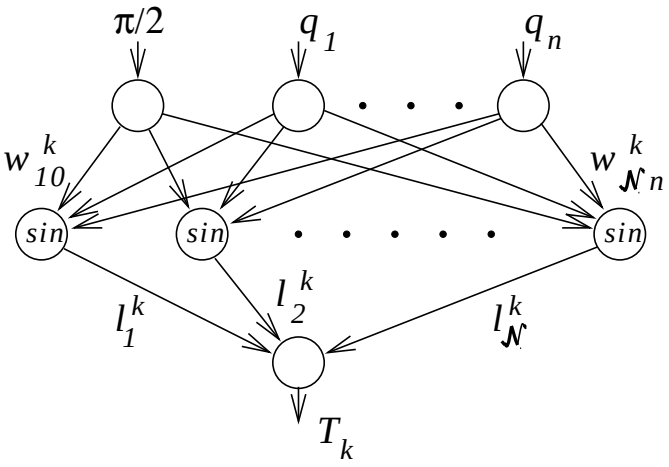
$$g(\sum_{j=1}^{n} w_j q_j), \quad \text{gdzie} \quad w_j \in \{-1, 0, 1\}. \tag{3.3}$$

W miejscu tym korzysta się z następujących jednoznaczności trygonometrycznych:
$\sin x \sin y = \frac{1}{2}(\cos(x-y) - cos(x+y))$, $\sin x \cos y = \frac{1}{2}(\sin(x-y) + \sin(x+y))$. Jeśli więc operację rozwinięcia do postaci addytywnej (wykorzystując dodatkowo fakt, iż $\cos x = \sin(x + \pi/2)$) przeprowadzi się dla wszystkich iloczynów ważonej sumy, odpowiednie elementy macierzy $T_0^n(q)$ będą miały postać:

$$T_k = \sum_{i=1}^{\mathcal{N}} l_i^k \sin((w_i^k)^T \tilde{q}), \tag{3.4}$$

gdzie k - numer elementu macierzy kinematyki, $\mathcal{N}$ - liczba różnoargumentowych funkcji *sinus* występujących w rozwinięciu T_k, $\tilde{q} = [\pi/2, q_1, q_2, \ldots, q_n]^T$, $w_i^k = [w_{i0}^k, w_{i1}^k, \ldots, w_{in}^k]^T$, $w_{ij}^k \in \{-1, 0, 1\}$. Równanie (3.4) z uwagi na konieczność wielokrotnego obliczania funkcji *sinus* nie może być, a raczej nie powinno być, wartościowane w sposób sekwencyjny. Znakomicie natomiast można je wartościować w sposób równoległy. Do tego celu wystarczy skonstruować trzywarstwową sieć neuronową typu *feedforward*, której $n+1$ wejść odpowiada kolejnym współrzędnym wektora $\tilde{q}$, $\mathcal{N} = 3^n$ neuronów warstwy ukrytej z sinusową funkcją aktywacji (3^n - maksymalna liczba możliwych kombinacji $(w_i^k)^T\tilde{q}$) odpowiada sinusom z równania (3.4), wyjście zaś odpowiada elementowi kinematyki T_k, [LB91, BDJ+94]. Jak łatwo zauważyć, równanie (3.4) odzwierciedla wprost architekturę takiej sieci, a występujące w nim współczynniki w_{ij}^k, l_i^k odpowiadają jej wagom (zobacz rysunek 3.1). Oczywiście, w przypadku, gdy sieć neuronowa posłużyć ma do obliczania kinematyki manipulatora z i-tym przegubem translacyjnym, sytuacja zmienia się. Wtedy

[†]Patrz metody symboliczne, podrozdział 2.2.1, oraz metoda Denavit'a-Hartenberg'a, dodatek A.



Rysunek 3.1: Sieć neuronowa modelująca element kinematyki T_k.

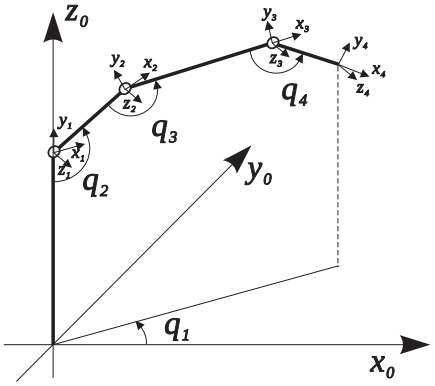
to i-te wejście do sieci ma wartość stałą, natomiast wagi l_i^k zależne od d_i (zobacz dodatek A) stają się parametrami zmiennymi. Sieć taka przestaje więc być siecią typu *feedforward*, a staje się siecią podobną do sieci kontekstowej. Z tego powodu sieć o sinusoidalnych neuronach powinna być stosowana przede wszystkim dla przypadków manipulatorów o przegubach rotacyjnych.

Z wagami omawianej sieci związany jest jeszcze jeden problem, a mianowicie problem właściwego ich nastawienia. W [LB91] proponuje się, aby wartości wag były nastawiane w procesie uczenia sieci i jako przykład podaje się trzy algorytmy uczenia: 1) *Least Mean Square (LMS) algorithm*, gdzie uczeniu poddawane są tylko wagi l_j^k, natomiast wagi w_{ij}^k wyznaczane są z równania (3.4), 2) *Back Propagation (BP) algorithm*, gdzie uczeniu poddawane są wagi l_j^k oraz w_{ij}^k, 3) *Hierarchically Self-Organizing Learning (HSOL) algorithm*, gdzie automatycznie rozpoznawane i kreowane są te neurony warstwy ukrytej, którym odpowiadają niezerowe wagi l_j^k.

Ostatni z przytoczonych algorytmów posiada przewagę nad innymi, gdyż dzięki niemu można zredukować rozmiar samej sieci neuronowej (liczba 3^n jest górnym ograniczeniem ilości sinusoidalnych neuronów, a przecież, przy szczególnej konstrukcji manipulatora, neuronów takich może być dużo mniej, [LK94]). Jednak nawet algorytm *HSOL* nie może równać się z metodami obliczeń symbolicznych, dzięki którym cały proces uczenia sieci i redukcja jej rozmiarów może być sprowadzony do zagadnienia „przepisania" (przekalkowania) uproszczonego równania (3.4) na tworzoną sieć, [BDJ+94, DK94, JDKS95]. Uproszczenie znaczy tu wyznaczenie na drodze symbolicznej wszystkich współczynników l_j^k, w_{ij}^k jako zmiennych zależnych od parametrów manipulatora (parametrów Denavit'a-Hartenberg'a), a następnie , po podstawieniu w miejsce parametrów ich wartości nominalnych, obliczenie ich wartości numerycznych. Po operacji upraszczania w równaniu (3.4) pozostaną tylko te wyrażenia, dla których odpowiednie wagi l_j^k będą niezerowe.

Wspomniane wyżej przepisanie jest niczym innym jak syntezą sieci neuronowej, której architektura i wartości wag dokładnie odpowiadają elementom równania uproszczonego. Jeśli sieć taka ma służyć do obliczania kinematyki nominalnej, wcale nie trzeba jej uczyć. O wyższości sieci sinusoidalnych nad innymi typami sieci neuronowych świadczą między innymi właśnie te dwa wyróżnione przymioty.

Posługiwanie się metodami obliczeń symbolicznych przy syntezie sieci neuronowych



i	β_i	d_i [m]	a_i [m]	α_i
1	q_1	0.7	0	$\pi/2$
2	$q_2 - \pi/2$	0	0.3	0
3	$q_3 - \pi/2$	0	0.6	0
4	$q_4 - \pi/2$	0	0.2	0

Rysunek 3.2: Manipulator typu 4R.

niesie ze sobą jeszcze jedną znaczącą zaletę. Na podstawie symbolicznego opisu kinematyki w prosty sposób można wygenerować równania reprezentujące jakobian, a w konsekwencji można również otrzymać jego neuronową reprezentację. Przy standardowych metodach neuronowej implementacji kinematyki takiego zabiegu wykonać nie można.

Jak łatwo zauważyć, architektura sieci implementujących elementy jakobianu będzie podobna do architektury sieci implementujących odpowiadające im elementy kinematyki. Różniczkując bowiem T_k po zmiennych q_m, $m = 1, \ldots, n$, oraz korzystając z tożsamości trygonometrycznych otrzymamy:

$$J_k = \frac{\partial T_k}{\partial q_m} = \sum_{i=1}^{\mathcal{N}} l_i^k w_{im}^k \cos((w_i^k)^T \tilde{q}) = \sum_{i=1}^{\mathcal{N}} \bar{l}_i^k \cos((w_i^k)^T \tilde{q}) = \sum_{i=1}^{\mathcal{N}} \bar{l}_i^k \sin((\bar{w}_i^k)^T \bar{q}). \qquad (3.5)$$

Oczywiście, rozważania na temat jakobianu są poprawne, jeśli jest on zdefiniowany dla macierzowej reprezentacji kinematyki, w której poszczególne elementy jego wierszy dane są jak w równaniu (3.5). Przy innych reprezentacjach, gdy do wyrażenia orientacji używa się np. kątów RPY, takiego podejścia bezpośrednio zastosować się nie da, [BDJ+94].

Przykład: Na rysunku 3.2 przedstawiony jest manipulator typu 4R wraz z tabelą opisujących go parametrów Denavit'a-Hartenberg'a. Pierwszym krokiem przy syntezie sieci neuronowej implementującej jego kinematykę (a później jego jakobian) jest wyznaczenie macierzowego równania kinematyki w postaci symbolicznej.

W czasie przeprowadzania obliczeń symbolicznych przypisania parametrom manipulatora ich numerycznych wartości można dokonać a) już na wstępie, b) dopiero na końcu obliczeń. Postępowanie jak w przypadku a) pozwala na znaczne uproszczenie obliczeń, jednak ze względu na eliminację zerowych elementów, zawęża otrzymywany wynik do nominalnej postaci macierzy kinematyki. W efekcie syntezowana później sieć neuronowa swoją strukturą odpowiadać będzie kinematyce nominalnej, a ta różnić się może od struktury odpowiadającej kinematyce rzeczywistej. Przeprowadzone doświadczenia pokazały, że sieć nominalna w porównaniu z siecią uwzględniającą istnienie niezerowych elementów gorzej kalibruje się (zobacz podrozdział 3.2). Postępowanie jak w przypadku b) pozwala natomiast na wyznaczenie macierzy kinematyki i syntezę sieci neuronowej w pełnej postaci. Odbywa się to jednak kosztem znacznego zwiększenia złożoności obliczeń.

Dla przyjętego modelu manipulatora w wyniku obliczeń symbolicznych otrzymuje się następującą postać macierzy kinematyki:

$$\begin{bmatrix} c_1A(q_2,q_3,q_4) & c_1B(q_2,q_3,q_4) & s_1 & c_1(0.3s_2 - 0.6(s_2c_3 + c_2s_3) + 0.2A(q_2,q_3,q_4)) \\ s_1A(q_2,q_3,q_4) & s_1B(q_2,q_3,q_4) & -c_1 & s_1(0.3s_2 - 0.6(s_2c_3 + c_2s_3) + 0.2A(q_2,q_3,q_4)) \\ -B(q_2,q_3,q_4) & A(q_2,q_3,q_4) & 0 & 0.7 - 0.3c_2 + 0.6(c_2c_3 - s_2s_3) - 0.2B(q_2,q_3,q_4)) \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

gdzie

$$\begin{aligned} A(q_2,q_3,q_4) &= c_2s_3c_4 + c_2c_3s_4 + s_2c_3c_4 - s_2s_3s_4, \\ B(q_2,q_3,q_4) &= c_2c_3c_4 - s_2s_3c_4 - s_2c_3s_4 - c_2s_3s_4, \end{aligned}$$

oraz $s_i = \sin(q_i), c_i = \cos(q_i)$, dla $i = 1, \ldots, 4$.

W kroku drugim prowadzącym do syntezy sieci neuronowej, wszystkie elementy macierzy kinematyki przekształcane są do postaci ważonej sumy funkcji *sinus*. W rozważanym przypadku przynosi to następującą formę macierzy kinematyki:

$$\begin{bmatrix} 0.5s_{1+2+3+4} - 0.5s_{1-2-3-4} & 0.5c_{1+2+3+4} + 0.5c_{1-2-3-4} & s_1 \\ 0.5c_{1-2-3-4} - 0.5c_{1+2+3+4} & 0.5s_{1+2+3+4} + 0.5s_{1-2-3-4} & -c_1 \\ -c_{2+3+4} & s_{2+3+4} & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{3.6}$$

$$\begin{bmatrix} 0.15s_{1+2} - 0.15s_{1-2} + 0.3s_{1-2-3} - 0.3s_{1+2+3} + 0.1s_{1+2+3+4} - 0.1s_{1-2-3-4} \\ 0.15c_{1-2} - 0.15c_{1+2} + 0.3c_{1+2+3} - 0.3c_{1-2-3} + 0.1c_{1-2-3-4} - 0.1c_{1+2+3+4} \\ 0.7 - 0.3c_2 + 0.6c_{2+3} + 0.2c_{2+3+4} \\ 1 \end{bmatrix}$$

gdzie

$$\begin{aligned} s_{i+j+k+l} &= \sin(q_i + q_j + q_k + q_l), \quad c_{i+j+k+l} = \sin(q_i + q_j + q_k + q_l + \tfrac{\pi}{2}), \\ s_{i-j-k-l} &= \sin(q_i - q_j - q_k - q_l), \quad c_{i-j-k-l} = \sin(q_i - q_j - q_k - q_l + \tfrac{\pi}{2}), \end{aligned}$$

dla $i, j, k, l = 1, \ldots, 4$.

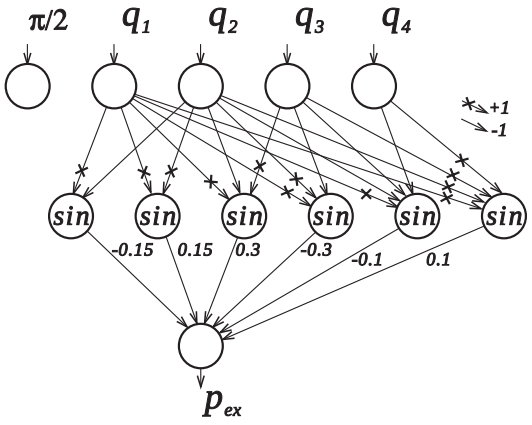
W kroku ostatnim każdy element powyższej macierzy jest „przepisywany" na sieć neuronową. Dla prostoty przepisanie to wyjaśnione zostanie na przykładzie tylko jednego z elementów macierzy, tj. elementu p_{ex} (x-owej współrzędnej wektora położenia), danego równaniem:

$$p_{ex} = 0.15s_{1+2} - 0.15s_{1-2} + 0.3s_{1-2-3} - 0.3s_{1+2+3} + 0.1s_{1+2+3+4} - 0.1s_{1-2-3-4}. \tag{3.7}$$

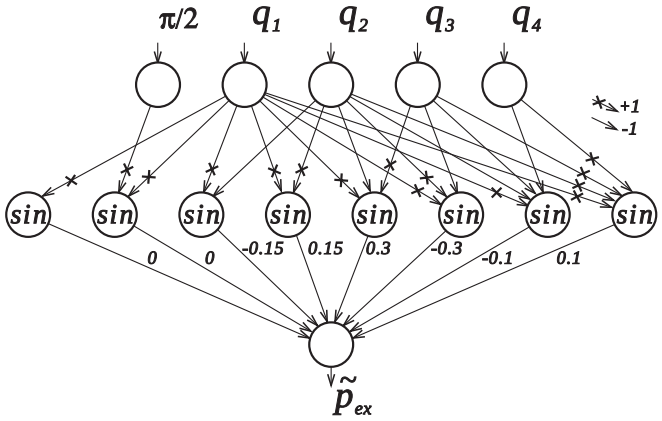
Na rysunku 3.3 przedstawiona jest sieć neuronowa implementująca p_{ex} (uproszczona sieć neuronowa). Jak widać, neurony warstwy ukrytej w sieci tej odpowiadają sinusom z równania (3.7). Współczynniki stojące przy argumentach q_i, odpowiednio dla każdego z sinusów, mają swoje odzwierciedlenie w wagach warstwy ukrytej. Współczynnikom stojącym przy samych sinusach odpowiadają zaś wagi warstwy wyjściowej.

Uwzględnienie w procesie symbolicznego obliczania wszystkich parametrów długościowych (d_i, a_i) prowadzi do uzyskania rozszerzonego równania kinematyki, a tym samym do syntezy rozszerzonej sieci neuronowej. W przypadku elementu p_{ex} rozszerzone równanie ma następującą postać:

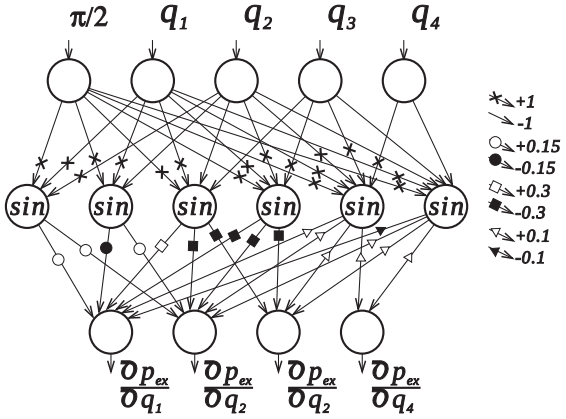
$$\tilde{p}_{ex} = 0s_1 + 0c_1 + p_{ex}, \tag{3.8}$$



Rysunek 3.3: Neuronowa implementacja p_{ex}.



Rysunek 3.4: Neuronowa implementacja $\tilde{p}_{ex}$.



Rysunek 3.5: Neuronowa implementacja fragmentu jakobianu $\frac{\partial p_{ex}}{\partial q}$.

zaś architekturę rozszerzonej, odpowiadającej $\tilde{p}_{ex}$, sieci neuronowej przedstawia rysunek 3.4.

Po dokonaniu syntezy sieci neuronowej implementującej kinematykę przychodzi kolej na syntezę sieci implementującej jakobian. Podobnie jak było to w przypadku kinematyki, synteza ta przedstawiona zostanie na przykładzie fragmentu jakobianu, mianowicie wektora $\frac{\partial p_{ex}}{\partial q}$. Aby jej dokonać wystarczy symbolicznie zróżniczkować równanie 3.7 (w przypadku nominalnym), a następnie przepisać otrzymany wynik na syntezowaną sieć. Ponieważ

$$\frac{\partial p_{ex}}{\partial q} = \begin{bmatrix} 0.15c_{1+2} - 0.15c_{1-2} + 0.3c_{1-2-3} - 0.3c_{1+2+3} + 0.1c_{1+2+3+4} - 0.1c_{1-2-3-4} \\ 0.15c_{1+2} + 0.15c_{1-2} - 0.3c_{1-2-3} + 0.3c_{1+2+3} + 0.1c_{1+2+3+4} + 0.1c_{1-2-3-4} \\ -0.3c_{1-2-3} - 0.3c_{1+2+3} + 0.1c_{1+2+3+4} + 0.1c_{1-2-3-4} \\ 0.1c_{1+2+3+4} + 0.1c_{1-2-3-4} \end{bmatrix} \tag{3.9}$$

sieć implementująca ten fragment jakobianu będzie miała postać jak na rysunku 3.5 (jej symboliczny opis w postaci ważonej sumy sinusów otrzyma się po dokonaniu podstawienia $\cos(x) = \sin(x + \pi/2)$).

Synteza sieci neuronowej implementującej skalibrowany jakobian wymaga nieco bardziej skomplikowanych zabiegów. Aby ją przeprowadzić należy najpierw skalibrować implementującą kinematykę sieć (zobacz podrozdział następny), potem odczytać ze skalibrowanej sieci wartości jej wag, następnie wpisać je do symbolicznych równań kinematyki, aż w końcu wychodząc ze zmodyfikowanych równań kinematyki dokonać syntezy jakobianu.

W celu porównania własności zaprezentowanych powyżej sieci neuronowych implementujących p_{ex} i $\tilde{p}_{ex}$ (zobacz rysunki 3.3 i 3.4), na ich przykładzie przeprowadzane zostały symulacyjne doświadczenia z kalibracją. Wyniki uzyskane podczas tych doświadczeń omówione zostaną w następnym podrozdziale.

3.2 Kalibracja kinematyki (uczenie sieci)

Synteza sieci neuronowych z zastosowaniem metod obliczeń symbolicznych, przy znanych parametrach manipulatora, pozwala na ominięcie żmudnego procesu ich uczenia. Ponieważ parametry, na podstawie których dokonuje się tej syntezy, są parametrami nominalnymi, tworzone sieci neuronowe odpowiadają kinematyce nominalnej. Jednakże kinematyka rzeczywista bywa różna od kinematyki nominalnej, podobnie jak różne bywają parametry rzeczywiste i nominalne manipulatora. Przyczyn występowania tych różnic należy szukać w błędach wykonania samego manipulatora, tj. błędach na długościach ramion, niedokładnościach przekładni, luzach, efektach cieplnych. Ponieważ błędów tych nie da się z góry przewidzieć, aby otrzymać kinematykę rzeczywistą dokonuje się kalibracji kinematyki nominalnej.

Istnieje wiele klasycznych już metod kalibracji kinematyk manipulatorów, [WHY88, ZD88, BM91, DS91, MRD91], jednak w przypadku neuronowej implementacji z nich się nie korzysta. Sama bowiem sieć, poprzez swoją zdolność do uczenia, dostarcza innej możliwości. Zadanie kalibracji kinematyki[‡] w przypadku „neuronowym" można wyrazić następująco, [BDJ+94, DJKM95]:

[‡]Kalibracja neuronowego jakobianu odbywa się poprzez kalibrację kinematyki. Wspomina się o tym przy okazji omawiania syntezy jakobianu (patrz przykład w poprzednim podrozdziale).

> Nauczyć sieć neuronową, reprezentującą nominalną kinematykę manipulatora, odwzorowania będącego jego kinematyką rzeczywistą, używając zarówno aktualnych wartości jego współrzędnych wewnętrznych, jak również aktualnych i nominalnych wartości jego współrzędnych zewnętrznych[§].

Tak postawione zadanie można zrealizować dzięki zastosowaniu wspomnianych uprzednio algorytmów LSM i BP. Ze względu jednak na możliwość korekcji zarówno wag l_j^k jak i w_{ij}^k (zobacz (3.4)), algorytm BP wydaje się być bardziej atrakcyjnym. Właśnie z tego powodu w poniższym przykładzie do kalibracji kinematyk został wykorzystany algorytm BP.

Przykład: Rezultaty uzyskane podczas kalibracji neuronowych kinematyk przedstawione zostaną na przykładzie sieci neuronowych implementujących równania (3.7) i (3.8). W celu porównania osiągniętych dla nich wyników kalibracji zdefiniowane zostały następujące błędy:

- maksymalny błąd bezwzględny kinematyki nominalnej

$$\epsilon_{\max}^{\text{nom}} = \max_i |p_{exi}^{\text{nom}} - p_{exi}^{\text{act}}|,$$

- maksymalny błąd bezwzględny kinematyki skalibrowanej

$$\epsilon_{\max}^{\text{cal}} = \max_i |p_{exi}^{\text{cal}} - p_{exi}^{\text{act}}|,$$

- średni błąd bezwględny kinematyki nominalnej

$$\epsilon_{\text{mean}}^{\text{nom}} = \frac{1}{\mathcal{I}} \sum_{i=1}^{\mathcal{I}} |p_{exi}^{\text{nom}} - p_{exi}^{\text{act}}|,$$

- średni błąd bezwględny kinematyki skalibrowanej

$$\epsilon_{\text{mean}}^{\text{cal}} = \frac{1}{\mathcal{I}} \sum_{i=1}^{\mathcal{I}} |p_{exi}^{\text{cal}} - p_{exi}^{\text{act}}|,$$

gdzie $\mathcal{I}$ oznacza ilość punktów kontrolnych[¶] (pomiarowych), p_{exi}^{nom} jest obliczonym przez sieć nominalnym położeniem efektora, p_{exi}^{cal} jest położeniem efektora obliczonym przez skalibrowaną sieć, p_{exi}^{act} natomiast jest rzeczywistym położeniem efektora.

Oprócz powyższych błędów zdefiniowane zostały, jako odpowiednie ich ilorazy, następujące wskaźniki jakości kalibracji: $\sigma_{\max} = \epsilon_{\max}^{\text{nom}}/\epsilon_{\max}^{\text{cal}}$, $\sigma_{\text{mean}} = \epsilon_{\text{mean}}^{\text{nom}}/\epsilon_{\text{mean}}^{\text{cal}}$.

Ponieważ przeprowadzone doświadczenia z kalibracją miały charakter symulacyjny, kinematykę rzeczywistą zamodelowano wprowadzając różnego rodzaju błędy do równania kinematyki nominalnej. W tabeli 3.1 przedstawione jest zestawienie tych błędów[‖] wraz ze sposobem ich binarnego kodowania dla celów prezentacji.

[§] Autor zdaje sobie sprawę, jak trudno jest dokonać pomiarów położenia i orientacji na rzeczywistym manipulatorze.

[¶] Przez punkt kontrolny rozumiana jest konfiguracja manipulatora, q, dla której obliczane były: p_{exi}^{act}, p_{exi}^{nom}, p_{exi}^{cal}.

[‖] Z oznaczaniami parametrów jak w metodzie Denavit'a-Hartenberg'a.

Źródło błędu	Kod błędu
błędna wartość parametru d_i	000001
błędna wartość parametru a_i	000010
błędna wartość parametru α_i	000100
błędy wynikające z niedokładności przekładni	001000
błędy wynikające z niedokładnego pozycjonowania układu bazowego	010000
błędy wynikające z zaszumienie pomiaru położenia efektora	100000

Tablica 3.1: Źródła błędów wprowadzanych do kinematyki oraz sposób ich kodowania.

Do celów kalibracji przygotowany został zestaw uczący, składający się ze zbioru stu par {konfiguracja, q; rzeczywiste położenie, p_{exi}^{act}}, których rozmieszczenie odpowiadało węzłom równomiernej siatki rozpiętej w przestrzeni wewnętrznej manipulatora. Liczba przeprowadzonych w czasie kalibracji cykli uczenia wynosiła tysiąc**.

Do celów weryfikacji wyników kalibracji wziętych zostało tysiąc punktów kontrolnych, których rozmieszczenie odpowiadało węzłom równomiernej siatki rozpiętej w przestrzeni wewnetrznej manipulatora (różnej od siatki wykorzystanej do kalibracji). W punktach tych obliczane były wartości zdefiniowanych wcześniej błędów bezwzględnych oraz wartości wskaźników jakości.

Wyniki weryfikacji, w postaci zestawienia wartości błędów bezwzględnych i wskaźników jakości (po tysięcznym cyklu uczenia) w zależności od błędów wprowadzonych do kinematyki nominalnej oraz zestawienia wartości błędów bezwzględnych i wskaźników jakości w zależności od ilości przeprowadzonych cykli uczenia, przedstawione zostały, odpowiednio, w tabeli 3.2 i tabeli 3.3.

Jak można zauważyć (patrz tabela 3.2), przy błędach kinematyki kodowanych przez: 000001, 000011, 010001, 100001, lepsze wyniki kalibracji uzyskano dla rozszerzonego modelu sieci neuronowej i to gdy korekcji poddawane były tylko wagi jej warstwy wyjściowej. Odwrotnie było w przypadku błędów kinematyki kodowanych przez: 000101, 001001. Przy takich źródłach błędów sieć rozszerzona gorzej poddawała się kalibracji, a zamiast uczenia wag warstwy wyjściowej sieci lepsze efekty dawało uczenie jej wszystkich wag.

Przyczynę uzyskiwania lepszych czy gorszych rezultatów dla różnych architektur sieci i algorytmów uczenia tłumaczy fakt, że zarówno sieci, jak i algorytmy uczenia nie zawsze „pasowały" do wprowadzonych do kinematyki błędów. Mówiąc dokładniej, o efektach kalibracji decydowało to, czy funkcja, jaką obliczała sieć odpowiadała z dokładnością do parametrów funkcji, jaką była kinematyka rzeczywista oraz czy algorytm uczenia pozwalał na zmianę parametrów różniących te dwie funkcje††. Jeśli więc wprowadzony błąd nie miał swojego odzwierciedlenia w architekturze sieci, sieć taka gorzej kalibrowała się. Jeśli natomiast architektura sieci uwzględniała istnienie błędów, a błędów takich w kinematyce rzeczywistej nie było, w czasie kalibracji z uczeniem wszystkich wag sieci niepotrzebnie

**Podczas symulacji zauważono, że po przeprowadzeniu tysiąca cykli uczenia dalsza kalibracja neuronowej kinematyki nie zmniejsza poziomu otrzymywanych błędów. Zauważono ponadto, że w rozpatrywanych przypadkach poziomy te ustalały się na różnych wartościach. Biorąc to pod uwagę zdecydowano, że wykonanie tysięcznego cyklu uczenia, a nie zmniejszenie się błędu poniżej wartości zadanej, kończyć będzie kalibrację.

††Architektura sieci, a więc ilość neuronów, ich typ oraz topologia połączeń decyduje o tym, jakiego rodzaju funkcje da się nimi aproksymować oraz jak dokładna może to być aproksymacja.

modyfikowano niektóre z nich.

Dlatego przy błędach kodowanych przez: 000001, 000011, 010001, 100001, sieć rozszerzona, uwzględniająca w swojej architekturze możliwość istnienia niezerowych parametrów długościowych, lepiej kalibrowała się od sieci uproszczonej. Dlatego uczenie wszystkich wag sieci przy błędach kodowanych przez 000101 i 001001 przynosiło lepsze rezultaty niż uczenie tylko wag warstwy wyjściowej. Dlatego też przy wystąpieniu błędu 000100 (co, przy wyprowadzaniu równań kinematyki, powoduje pojawienie się dodatkowych funkcji trygonometrycznych) efekty kalibracji były najgorsze.

Generalnie we wszystkich przypadkach (pomijając różnice w architekturach sieci i algorytmach uczenia), poza przypadkiem błędów kodowanych przez 100001 (gdzie głównym źródłem błędów były błędy pomiarowe), możliwym było zredukowanie maksymalnego błędu bezwzględnego kinematyki skalibrowanej do poziomu rzędu 10^{-2}mm. W porównaniu ze standardowymi metodami kalibracji kinematyki, [WHY88, ZD88, BM91, DS91, MRD91], jest to rezultat lepszy o co najmniej jeden rząd. Co więcej, wydaje się, że jeśli sieć modelująca kinematykę posiada wystarczającą liczbę neuronów, to przy jej kalibracji jedynym ograniczeniem jest dokładność urządzeń pomiarowych (w przypadku 100001 wprowadzony szum pomiarowy był nie większy niż 0,2 mm, a właśnie do tego poziomu sprowadzony został maksymalny błąd bezwzględny skalibrowanej kinematyki).

Efekty kalibracji w zależności od liczby przeprowadzonych cykli uczenia dla przypadku 000001 przedstawia tabela 3.3. Widać w niej, że po tysięcznym cyklu uczenia dalsza kalibracja znacząco nie wpływa na uzyskiwane rezultaty.

Kod błędu	Przed kalibracją		Po kalibracji				
				z korekcją wag l		z korekcją wag l i w	
	Parametr	Sieci p_{ex}, $\tilde{p}_{ex}$	Parametr	Sieć p_{ex}	Sieć $\tilde{p}_{ex}$	Sieć p_{ex}	Sieć $\tilde{p}_{ex}$
000001	$\epsilon^{\text{nom}}_{\text{max}}$	8.24	$\epsilon^{\text{cal}}_{\text{max}}$	0.00012	0.00009	0.00580	0.00670
	$\epsilon^{\text{nom}}_{\text{mean}}$	2.45	$\epsilon^{\text{cal}}_{\text{mean}}$	0.00003	0.00002	0.00175	0.00185
			σ_{max}	6.8 e4	9.2 e4	1.4 e3	1.2 e3
			σ_{mean}	9.5 e4	1.1 e5	1.4 e3	1.3 e3
000011	$\epsilon^{\text{nom}}_{\text{max}}$	21.7	$\epsilon^{\text{cal}}_{\text{max}}$	19.1	0.00011	2.56	0.0942
	$\epsilon^{\text{nom}}_{\text{mean}}$	11.1	$\epsilon^{\text{cal}}_{\text{mean}}$	4.1	0.00002	0.72	0.0197
			σ_{max}	1.1 e0	2.0 e5	8.5 e0	2.3 e2
			σ_{mean}	2.7 e0	4.8 e5	1.5 e1	5.6 e2
000101	$\epsilon^{\text{nom}}_{\text{max}}$	22.2	$\epsilon^{\text{cal}}_{\text{max}}$	20.9	30.4	0.0812	0.880
	$\epsilon^{\text{nom}}_{\text{mean}}$	5.6	$\epsilon^{\text{cal}}_{\text{mean}}$	5.5	7.3	0.0246	0.220
			σ_{max}	1.1 e0	0.7 e0	2.7 e2	2.5 e1
			σ_{mean}	1.0 e0	0.8 e0	2.3 e2	2.6 e1
001001	$\epsilon^{\text{nom}}_{\text{max}}$	22.8	$\epsilon^{\text{cal}}_{\text{max}}$	19.4	20.6	0.0750	0.740
	$\epsilon^{\text{nom}}_{\text{mean}}$	3.3	$\epsilon^{\text{cal}}_{\text{mean}}$	2.9	4.2	0.0249	0.185
			σ_{max}	1.2 e0	1.1 e0	3.0 e2	3.1 e1
			σ_{mean}	1.2 e0	0.8 e0	1.3 e2	1.8 e1
010001	$\epsilon^{\text{nom}}_{\text{max}}$	16.2	$\epsilon^{\text{cal}}_{\text{max}}$	0.00010	0.00009	0.00670	0.00569
	$\epsilon^{\text{nom}}_{\text{mean}}$	8.8	$\epsilon^{\text{cal}}_{\text{mean}}$	0.00003	0.00002	0.00150	0.00164
			σ_{max}	1.6 e5	1.8 e5	2.4 e3	2.9 e3
			σ_{mean}	3.1 e5	4.6 e5	5.9 e3	5.4 e3
100001	$\epsilon^{\text{nom}}_{\text{max}}$	8.24	$\epsilon^{\text{cal}}_{\text{max}}$	0.165	0.138	0.379	0.173
	$\epsilon^{\text{nom}}_{\text{mean}}$	2.45	$\epsilon^{\text{cal}}_{\text{mean}}$	0.059	0.036	0.096	0.061
			σ_{max}	5.0 e1	6.0 e1	2.2 e1	4.8 e1
			σ_{mean}	4.1 e1	6.9 e1	2.6 e1	4.0 e1

Wartości błędów ϵ podane są w milimetrach;
$xe^y = x \cdot 10^y$.

Tablica 3.2: Wyniki kalibracji w zależności od źródeł błędów.

Liczba cykli	Po kalibracji				
		z korekcją wag l		z korekcją wag l i w	
	Parametr	Sieć p_{ex}	Sieci $\tilde{p}_{ex}$	Sieć p_{ex}	Sieć $\tilde{p}_{ex}$
1	$\epsilon^{\text{cal}}_{\text{max}}$	0.218	0.383	0.529	0.519
	$\epsilon^{\text{cal}}_{\text{mean}}$	0.052	0.114	0.144	0.187
	σ_{max}	3.8 e1	2.2 e1	1.6 e1	1.6 e1
	σ_{mean}	4.7 e1	2.1 e1	1.7 e1	1.3 e1
10	$\epsilon^{\text{cal}}_{\text{max}}$	0.00011	0.00015	0.130	0.174
	$\epsilon^{\text{cal}}_{\text{mean}}$	0.00002	0.00004	0.035	0.035
	σ_{max}	7.5 e4	5.5 e4	6.4 e1	4.7 e1
	σ_{mean}	1.2 e5	6.6 e4	7.0 e1	7.0 e1
100	$\epsilon^{\text{cal}}_{\text{max}}$	0.00012	0.00010	0.0155	0.0183
	$\epsilon^{\text{cal}}_{\text{mean}}$	0.00002	0.00002	0.0046	0.0067
	σ_{max}	6.9 e4	8.2 e4	5.3 e2	4.5 e2
	σ_{mean}	1.3 e5	1.1 e5	5.4 e2	3.7 e2
1000	$\epsilon^{\text{cal}}_{\text{max}}$	0.00012	0.00009	0.00580	0.00670
	$\epsilon^{\text{cal}}_{\text{mean}}$	0.00003	0.00002	0.00175	0.00185
	σ_{max}	6.8 e4	9.2 e4	1.4 e3	1.2 e3
	σ_{mean}	9.5 e4	1.1 e5	1.4 e3	1.3 e3
10000	$\epsilon^{\text{cal}}_{\text{max}}$	0.00013	0.00009	0.00673	0.00788
	$\epsilon^{\text{cal}}_{\text{mean}}$	0.00002	0.00002	0.00187	0.00219
	σ_{max}	6.7 e4	9.1 e4	1.2 e3	1.0 e3
	σ_{mean}	11.6 e4	1.1 e5	1.3 e3	1.1 e3

Wartości błędów ϵ podane są w milimetrach;
x e$y = x \cdot 10^y$.

Tablica 3.3: Wyniki kalibracji w zależności od liczby cykli uczenia (przypadek 000100).

Rozdział 4

Wykorzystanie sieci neuronowych do obliczania kinematyki odwrotnej

W poprzednim rozdziale pokazano, jak na podstawie parametrów nominalnych manipulatora można skonstruować sieci neuronowe implementujące jego kinematykę i jakobian. Omówiono w nim również sposób kalibracji tych sieci, tj. metodę, dzięki której wspomniane sieci nauczyć można odzwierciedlać kinematykę i jakobian rzeczywistego manipulatora. Czy uzyskane dotychczas wyniki mogą posłużyć do neuronowego obliczania kinematyki odwrotnej? Czy istnieją metody, w których do obliczenia kinematyki odwrotnej wystarcza znajomość tylko kinematyki prostej? Przedstawieniu odpowiedzi na tak postawione pytania poświęcony jest niniejszy rozdział.

Wśród wielu przedstawionych w rozdziale 2 metod rozwiązywania odwrotnego problemu kinematyki metoda transponowanego jakobianu jako jedyna pozwala na obliczanie kinematyki odwrotnej na podstawie znajomości kinematyki prostej*, a tym samym pozwala na wykorzystanie zaprezentowanych w poprzednim rozdziale sieci neuronowych. Jednak do przeprowadzenia syntezy sieci neuronowej, obliczającej kinematykę odwrotną zgodnie z tą metodą, samo zastąpienie kinematyki i jakobianu ich neuronowymi implementacjami nie wystarcza. Nierozwiązanym bowiem pozostaje problem powiązania tych sieci w jedną sieć, nie mówiąc już o brakującej neuronowej implementacji oddziaływania ograniczeń. Aby uzupełnić syntezę sieci o te właśnie brakujące elementy, w kolejnych podrozdziałach problem kinematyki odwrotnej omówiony zostanie raz jeszcze, tym razem w terminach optymalizacji, oraz przedstawione zostaną rozwiązujące go neuronowo implementowalne algorytmy.

Metoda transponowanego jakobianu przedstawiona w podrozdziale 2.2.2 dostarcza rozwiązań obarczonych pewnym błędem. Wiąże się to z tym, że położenia efektora odpowiadające obliczanym z algorytmu konfiguracjom podążają za trajektorią zadaną w miarę upływu czasu. Jeśli popatrzeć na ten algorytm przy zdyskretyzowanym czasie, to okaże się, że w kolejnych chwilach rozwiązywane są osobne odwrotne problemy kinematyki, przy czym dla każdego z nich rozważany algorytm uruchamiany jest tylko raz. W jednym kroku zaś praktycznie nie można zniwelować różnicy pomiędzy położeniem bieżącym a zadanym. Dzięki dyskretyzacja czasu oraz potraktowaniu odwrotnego problemu kinematyki jako problemu optymalizacji statycznej możliwe jest zmniejszenie wspomnianego błędu do zadanego poziomu. Przedstawiane w niniejszym rozdziale metody obliczeniowe bazują właśnie na tym spostrzeżeniu.

*Pewne podobieństwo do tej metody można dostrzec w metodzie zaproponowanej w [Fag94].

4.1 Synteza sieci neuronowej do obliczania kinematyki odwrotnej bez ograniczeń

Niech $e(q) = p_f - p_e(q)$ oznacza błąd położenia, tzn. różnicę między bieżącym położeniem efektora $p_e(q)$, (obliczonym z kinematyki prostej $t_n(q)$), a położeniem pożądanym, p_f (wyrażonym w przestrzeni kartezjańskiej)[†]. Przy braku ograniczeń odwrotny problem kinematyki może być transformowany do następującego problemu optymalizacji:

Problem 1 *Dla danego położenia p_f, wyrażonego w przestrzeni zewnętrznej, znaleźć taką konfigurację przegubów manipulatora q^*, która minimalizuje kryterium:*

$$v(q) = \frac{1}{2} e(q)^T e(q). \tag{4.1}$$

Przy założeniu osiągalności p_f kryterium $v(q)$ może być traktowane jako funkcja Lapunowa. Zakładając, że $p_e(q)$ jest ciągła, $v(q)$ jest lokalnie pozytywnie zdefiniowaną funkcją q, gdyż:

- $v(q^*) = 0$ gdzie $e(q^*) = 0$;
- $v(q) > 0$ dla wszystkich q, takich, że $q \neq q^*$ i q należy do pewnej kuli $B(\epsilon) = \{q \mid \|q - q^*\| \leq \epsilon, \quad \epsilon > 0 \quad \text{i} \quad e(q^*) = 0\}$.

Fakt ten posłużyć może do wyprowadzenia algorytmu minimalizacji oraz wykazania jego zbieżności. Różniczkując $v(q)$ po czasie[‡] otrzymamy:

$$\dot{v} = \left(\frac{\partial v}{\partial q}\right)^T \dot{q} = -e^T \frac{\partial p_e}{\partial q} \dot{q} = -e^T J_e \dot{q}, \tag{4.2}$$

gdzie $J_e(q) = \frac{\partial p_e(q)}{\partial q}$ jest okrojoną do części translacyjnej macierzą Jakobiego kinematyki. Aby zapewnić zbieżność błędu e do zera (w sensie Lapunowa), $\dot{q}$ z równania (4.2) powinno mieć postać:

$$\dot{q} = \begin{cases} -\alpha \frac{\partial v}{\partial q} = \alpha J_e^T e, & \text{jeśli} \quad e \neq 0 \\ 0, & \text{jeśli} \quad e = 0 \end{cases}, \tag{4.3}$$

gdzie $\alpha \in \mathbb{R}$, $\alpha > 0$. Istotnie, jeśli $\dot{q}$ podane wzorem (4.3) wstawimy do równania (4.2) otrzymamy

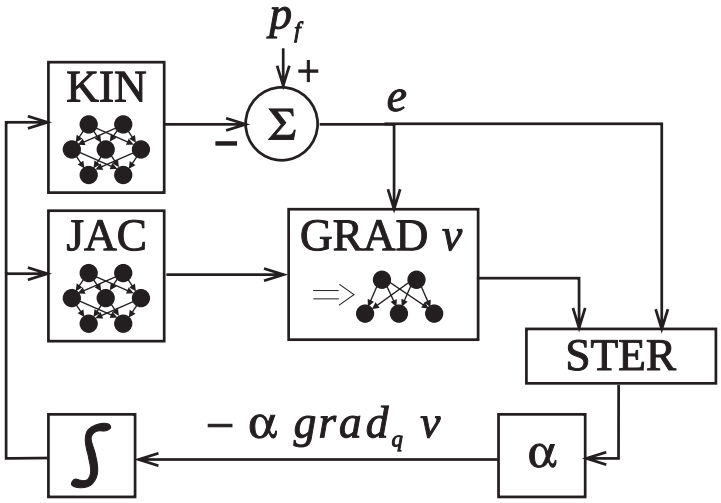
$$\dot{v} = -\alpha \left\| \frac{\partial v}{\partial q} \right\|^2 \leq 0, \tag{4.4}$$

co zapewnia zbieżność $e \to 0$, a więc $q \to q^*$. O stopniu zbieżności decyduje sposób obliczania występującego w równaniach (4.2) i (4.4) parametru α (zobacz podrozdział 4.5).

Równanie (4.3) stanowi tzw. system gradientowy, który można zaimplementować jako sprzężoną sieć neuronową (zobacz rysunek 4.1), składającą się z modułów: obliczania

[†]Pominięcie współrzędnych orientacji nie oznacza, że przedstawiana tu metoda pracuje tylko dla współrzędnych położenia. Aby uwzględnić orientację wystarczy rozszerzyć wektor błędu $e(q)$ o kolejne składowe, uzyskane z różnicy odpowiednich elementów macierzy orientacji (pożądanej i bieżącej, zobacz macierzowego równania kinematyki, (2.4), (3.1)), a z metody tej otrzyma się pełne rozwiązanie. Elementy rozszerzonego jakobianu wyrażać się wtedy będą jak w (3.5).

[‡]Czas wprowadzono w sposób sztuczny, uzależniając od niego q (q będzie się zmieniać w trakcie obliczeń).



Rysunek 4.1: Schemat sprzężonej sieci neuronowej służącej do obliczania kinematyki odwrotnej bez uwzględniania ograniczeń.

kinematyki, (KIN), obliczania jakobianu, (JAC), obliczania gradientu funkcji kryterialnej, (GRADv), obliczania kroku zbieżności, α, sterowania procesem obliczeń, (STER). Ostatni z modułów, (STER), odpowiedzialny jest za rozpoznawanie i omijanie minimów lokalnych (zobacz podrozdział 4.5) oraz za przerwanie obliczeń z chwilą spełnienia warunku stopu.

W rozdziale 6 omawiany jest system inteligentnego i reaktywnego sterowania robotem, w którego skład wchodzi moduł planowania ruchów on-line, MPR. W module tym zaimplementowany jest omawiany w niniejszym rozdziale algorytm neuronowego obliczania kinematyki odwrotnej (z uwzględnieniem ograniczeń). Zadaniem modułu STER w module MPR jest dostarczenie sygnału o istnieniu (lub nieistnieniu) rozwiązania. Sygnał taki będzie wygenerowany, gdy spełniony zostanie warunek stopu i obliczone zostanie (lub nie) wystarczająco dokładne rozwiązanie. Tylko w pozytywnym przypadku rozwiązanie to przekazane zostanie do modułu krokowego planowania trajektorii.

Przy standardowym warunku stopu $e(q) < \varepsilon$ (ε zadane), gdy p_f zadane jest w przestrzeni roboczej, sprzężona sieć iteracyjnie wygeneruje wartości zmian współrzędnych wewnętrznych, zgodnie z algorytmem(4.3), aż do chwili, kiedy bieżące położenie efektora manipulatora będzie wystarczająco bliskie położeniu zadanemu. Moduł STER zasygnalizuje wtedy fakt znalezienia rozwiązania odwrotnego problemu kinematyki. Generalnie jednak wcale tak być nie musi. Posługiwanie się algorytmem gradientowym w przypadku p_f zadanego poza przestrzenią roboczą manipulatora prowadzi do minimalizacji kryterium $v(q)$, jednak nie wiadomo, w którym kroku powinny zakończyć się iteracje. Warunek stopu, którym normalnie jest $e(q) < \varepsilon$, wtedy „nie pracuje". Wyjściem w takiej sytuacji byłoby określenie innego warunku stopu, np.: $e(q(k)) - e(q(k-1)) < \varepsilon$, bądź też: $k > limit$ (k - numer kolejnej iteracji, $limit$ - zadane ograniczenie na liczbę iteracji[§]). Jeśli spełniony zostanie warunek stopu, a otrzymane rozwiązanie odwrotnego problemu kinematyki dalekie będzie od rozwiązania oczekiwanego, moduł STER zasygnalizuje niemożność znalezienia rozwiązania.

[§]Warunkiem stopu związanym z liczbą wykonanych iteracji posłużono się w [YI95], gdzie, korzystając z algorytmu tunelowego, weryfikowano osiągalność (zobacz dodatek C).

Użycie miana „sprzężona sieć neuronowa" uzasadnia fakt, że wszystkie wspomniane wyżej moduły, poza modułem sterowania, mogą być zaimplementowane jako sieci neuronowe. Dzięki zaś neuronowej implementacji złożoność obliczeniowa samego algorytmu gradientowego w porównaniu ze złożonością obliczeniową przy standardowej (sekwencyjnej) jego implementacji wielokrotnie zmniejsza się. Jeśli przyjąć, że:

- czas propagacji obliczeń poprzez jedną warstwę sieci neuronowej jest równy 1 (a więc neuronowe obliczanie kinamatyki (jakobianu) zajmuje dwie jednostki czasu[¶], zaś jedną jednostkę czasu zajmuje obliczanie gradientu v przy pomocy sieci kontekstowej[‖]);

jeśli wziąć pod uwagę, że:

- obliczanie różnicy wektorów przy równoległej implementacji zajmuje 1 jednostkę czasu (a więc obliczenie wektora błędu e oraz wektora q zajmuje w obu przypadkach jedną jednostkę czasu);

jeśli założyć, że:

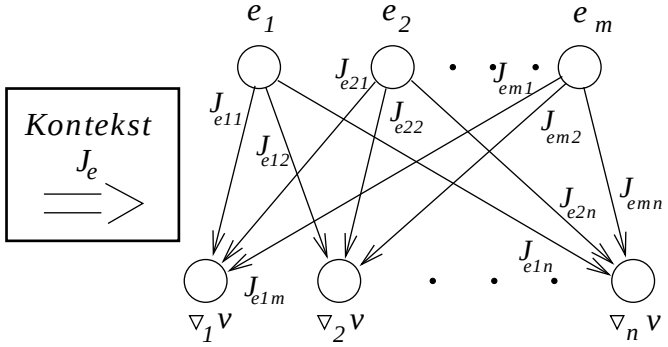
- obliczanie iloczynu skalarnego można zrealizować przy pomocy trzywarstwowej sieci neuronowej z neuronami mnożącymi w warstwie ukrytej oraz z jednym neuronem sumacyjnym w warstwie wyjściowej, tj. sieci o czasie propagacji równym 2 (dla α danego równaniem 4.33 obliczanie 2 iloczynów skalarnych oraz ich ilorazu zajmuje trzy jednostki czasu);

to czas potrzebny na wykonanie jednej iteracji algorytmu gradientowego oszacować można na $2+1+1+3+1=8$ jednostek (KIN równolegle z JAC, potem kolejno e, GRADv, α, q). Należy podkreślić, że oszacowanie to nie zależy ani od wymiaru wektora q (wymiaru przestrzeni wewnętrznej manipulatora), ani od wymiary wektora e (wymiaru przestrzeni zewnętrznej). Jak pokazały symulacje (zobacz podrozdział 4.5.1) wykonanie średnio 30 kroków obliczeń pozwala zmniejszyć wartość $||e||$ z poziomu rzędu kilkunastu (a nawet kilkudziesięciu) [cm] do poziomu rzędu kilku [μm] (przy czym ramiona manipulatora miały długość rzędu kilkudziesięciu [cm]). Całkowity czas potrzebny na obliczenie rozwiązania odwrotnego problemu kinematyki oszacować więc można na 240 jednostek. W przypadku sekwencyjnej implementacji samo obliczanie kinematyki szacowane jest na $m*3^n$ dodawań funkcji sinus.

Zagadnieniu neuronowego obliczania kinematyki i jakobianu poświęcony był już rozdział poprzedni. Wszystkie wymienione w nim zalety zaproponowanych sieci neuronowych automatycznie przenoszą się na neuronową implementację algorytmu gradientowego. Jednak do pełnej neuronowej implementacji tego algorytmu brakuje jeszcze przedstawienia neuronowego sposobu obliczania gradientu funkcji kryterialnej i parametru α. Postępując za przykładem Lee i Bekey'a, [LB91], Lee i Kil'a, [LK94], do konstrukcji modułów GRADv i α użyć można sieci kontekstowych (zobacz podrozdział 2.2.3). Dla przypomnienia, wagi w sieci kontekstowej mogą zmieniać się zgodnie z kontekstem, zdefiniowanym przez funkcję zewnętrzną lub stany sieci. Jeśli więc, jak na rysunku 4.2, wejściem do sieci jest błąd e, kontekstem – wyjście z sieci implementującej jakobian, to jej wyjściem jest gradient funkcji kryterialnej, $J_e^T e$. W ten oto sposób można dokonać syntezy sieci neuronowych wchodzących w skład modułów GRADv i α.

[¶] Ponieważ nie istnieją jeszcze rozwiązania sprzętowe z zaimplementowanymi w rzeczywistości sinusoidalnymi sieciami neuronowymi, trudno tu mówić o jakichś konkretnych wartościach czasów propagacji.

[‖] Komentarz w dalszej części podrozdziału.



Rysunek 4.2: Kontekstowa sieć neuronowa obliczająca $J_e^T e$.

4.2 Synteza sieci neuronowej do obliczania kinematyki odwrotnej z ograniczeniami konstrukcyjnymi

Niech q_i^{min} i q_i^{max} oznaczają, odpowiednio, minimalną i maksymalną wartość zmiennej q_i, tak że $q_i \in [q_i^{min}, q_i^{max}], i = 1, \ldots, n$. Niech $e(q) = p_f(q) - p_e(q)$. Przy tych założeniach odwrotny problem kinematyki z ograniczeniami konstrukcyjnymi może być transformowany do następującego problemu optymalizacji z ograniczeniami:

Problem 2 *Dla danego położenia p_f, wyrażonego we współrzędnych zewnętrznych, znaleźć taką konfigurację przegubów manipulatora q^*, która minimalizuje kryterium:*

$$v(q) = \frac{1}{2} e(q)^T e(q). \tag{4.5}$$

przy spełnieniu ograniczeń

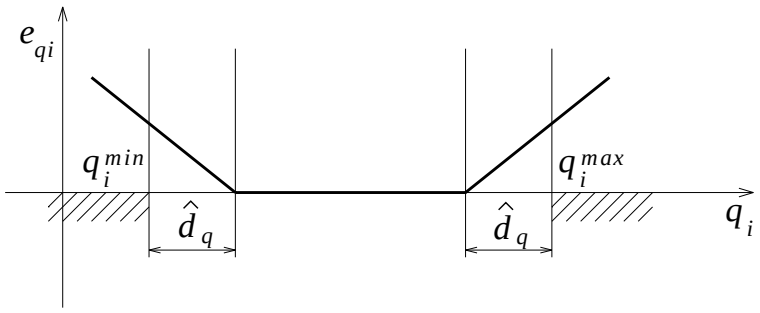
$$q_i^{min} \leq q_i^* \leq q_i^{max}, \quad i = 1, \ldots, n. \tag{4.6}$$

Powyższy problem, poprzez odpowiednią modyfikację funkcji kryterialnej, może być transformowany do problemu optymalizacji bez ograniczeń. Taka transformacja pozwali na powtórne skorzystanie z algorytmu (4.3), pod warunkiem, że miejsce $v(q)$ zajmie zmodyfikowana funkcja kryterialna.

Wydawać by się mogło, że jeśli p_f zadane jest w przestrzeni roboczej, to wprowadzanie ograniczeń konstrukcyjnych jest zabiegiem zbędnym. Jest to błędne przypuszczenie. Ponieważ zaproponowany algorytm generuje rozwiązanie w sposób iteracyjny, w którymś z kolejnych jego kroków mogłoby nastąpić przekroczenie tych ograniczeń i wygenerowanie nierealizowalnych rozwiązań. Właśnie stąd wynika konieczność ich uwzględniania, a w konsekwencji konieczność modyfikacji funkcji $v(q)$. W przypadku, gdy p_f leży poza przestrzenią roboczą, wprowadzone ograniczenia w oczywisty sposób nie pozwolą na obliczenie rozwiązania (zobacz rola modułu STER w podrozdziale poprzednim).

Zanim przedstawiony zostanie sposób modyfikacji funkcji $v(q)$ za przykładem Sciavicco i Siciliano ([SS88], zobacz rozdział 2) wprowadzone zostaną błędy dodatkowe. Jeśli $\hat{d}_q$ oznacza stały margines bezpieczeństwa (tj. jednakową dla wszystkich q_i bezpieczną odległość od ograniczeń), błędy dodatkowe e_{q_i}, $i = 1, \ldots, n$ dane są przez:

$$e_{q_i} = \begin{cases} \hat{d}_q - (q_i - q_i^{min}), & \text{dla } q_i < q_i^{min} + \hat{d}_q \\ 0, & \text{dla } q_i^{min} + \hat{d}_q \leq q_i \leq q_i^{max} - \hat{d}_q \\ \hat{d}_q - (q_i^{max} - q_i), & \text{dla } q_i > q_i^{max} - \hat{d}_q \end{cases}. \tag{4.7}$$



Rysunek 4.3: Wykres błędu $e_{q_i}(q)$.

Wykres błędu $e_{q_i}(q)$ przedstawia rysunek 4.3. Niech

$$\omega(q) = \frac{1}{2} e_q^T e_q \quad , \quad \text{gdzie} \quad e_q = (e_{q_1}, e_{q_2}, \ldots, e_{q_n})^T \tag{4.8}$$

oznacza kwadratową funkcję kary za zbliżanie się do ograniczeń. Funkcja ta przyjmuje wartość 0, gdy żaden z przegubów manipulatora nie znajduje się w odległości mniejszej od ograniczeń niż zadany margines $\hat{d}_q$. Większe wartości funkcja kary przyjmuje, gdy któryś z przegubów przekroczy ten margines (odpowiednio przesuwając wartości ograniczeń o $\hat{d}_q$ można sprawić, że funkcja kary zacznie działać dopiero po przekroczeniu pierwotnych ograniczeń). Nietrudno teraz pokazać, że jeśli tylko istnieje rozwiązanie odwrotnego problemu kinematyki przy ograniczeniach z zadanym marginesem bezpieczeństwa, minimalizacja zmodyfikowanego kryterium $\bar{v}(q)$, danego wzorem:

$$\bar{v}(q) = v(q) + \omega(q) = \frac{1}{2} e(q)^T e(q) + \frac{1}{2} e_q(q)^T e_q(q) \tag{4.9}$$

przy zastosowaniu algorytmu

$$\dot{q} = \begin{cases} -\alpha \frac{\partial \hat{v}}{\partial q}, & \text{jeśli } e \neq 0 \text{ lub } e_{q_i} \neq 0, \quad i = 1, \ldots, n \\ 0, & \text{jeśli } e, e_{q_i} = 0 \end{cases} \tag{4.10}$$

zapewni zbieżność błędów e, e_{q_i} do zera. W tym celu wystarczy, podobnie jak miało to miejsce przy problemie 1, posłużyć się metodą funkcji Lapunowa.

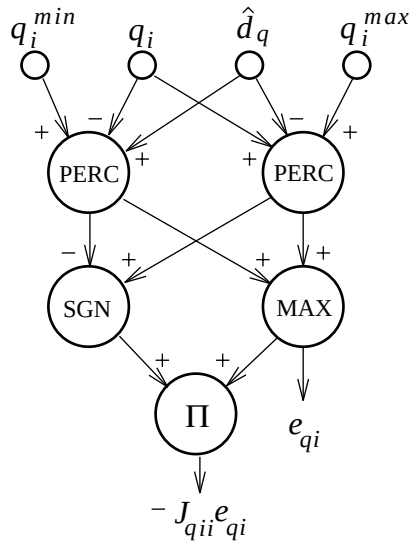
Równanie 4.10 można zaimplementować, podobnie do równania 4.3, jako sprzężoną sieć neuronową (przy zachowaniu wszystkich wynikających z tego faktu zalet). Ponieważ gradient zmodyfikowanej funkcji kryterialnej

$$\frac{\partial \hat{v}}{\partial q} = -J_e^T e - J_q^T e_q, \tag{4.11}$$

gdzie:

$$J_q = diag[a_i] \quad , \quad a_i = \begin{cases} 1, & \text{dla} \quad q_i < q_i^{min} + \hat{d}_q \\ 0, & \text{dla} \quad q_i^{min} + \hat{d}_q \leq q_i \leq q_i^{max} - \hat{d}_q \\ -1, & \text{dla} \quad q_i > q_i^{max} - \hat{d}_q \end{cases} \quad , \quad i = 1, \ldots, n \tag{4.12}$$

zawiera nowy składnik $-J_q^T e_q$, przy pełnej syntezie sieci neuronowej również i on powinien być obliczany neuronowo. Wartość i-tej współrzędnej tego składnika, $-J_{q_{ii}} e_{q_i} = a_i e_{q_i}$,



Rysunek 4.4: Obliczanie $-J_{q_{ii}}e_{q_i}$.

obliczyć można przy pomocy przedstawionej na rysunku 4.4 sieci neuronowej**. Funkcją aktywacji neuronów oznaczonych na rysunku symbolem PERC jest klasyczna funkcja perceptronu:

$$o_i = \begin{cases} net_i, & \text{gdy } net_i > 0 \\ 0, & \text{gdy } net_i \leq 0 \end{cases}, \tag{4.13}$$

gdzie $net_i = \sum_j w_{ji} in_{ji}$ jest wartością pobudzenia neuronu, w_{ij} – wagą wejścia in_{ij}, o_i – wyjściem z neuronu. Funkcją aktywacji neuronu oznaczonego symbolem SGN jest funkcja *signum*:

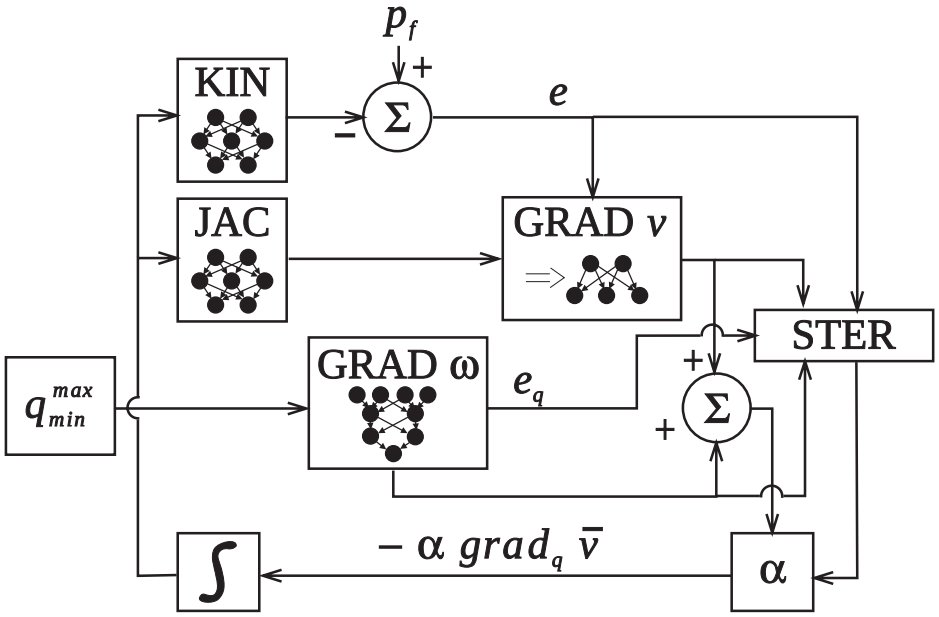
$$o_i = \begin{cases} 1, & \text{gdy } net_i > 0 \\ 0, & \text{gdy } net_i = 0 \\ -1, & \text{gdy } net_i < 0 \end{cases}. \tag{4.14}$$

Funkcją aktywacji neuronów oznaczonych symbolami MAX i $\prod$ jest funkcja identyczności, z tym, że w przypadku MAX pobudzenie neuronu wyraża się wzorem: $net_{imax} = max_j \ \ w_{ji} in_{ji}$, natomiast w przypadku $\prod$ wzorem: $net_{imax} = \prod w_{ji} in_{ji}$.

Pełną neuronowa implementacja algorytmu (4.10) przedstawia rysunek 4.5. Widoczna na nim sprzężona sieć neuronowa różni się od sieci służącej obliczaniu kinematyki odwrotnej bez ograniczeń (rysunek 4.1) modułami GRADω i q_{min}^{max}. W module GRADq, składającym się z n sieci neuronowych jak na rysunku 4.4, obliczane są poszczególne składniki gradientu funkcji kary, $-J_q^T e_q$. Moduł q_{min}^{max} dostarcza do modułu OGRq wartości ograniczeń q_i^{min}, q_i^{max}, $(i = 1, \ldots, n)$.

Wprowadzenie modułu GRADω do sprzężonej sieci neuronowej nie powoduje zwiększenia czasu wykonywania przez nią obliczeń. Przy założeniu, że czas propagacji poprzez jedną warstwę sieci zajmuje jedną jednostkę czasu, moduł GRADω wykona swoje obliczenia w czasie równym trzem jednostkom. Jednak obliczenia te odbywać się mogą równolegle z obliczaniem kinematyki i następującym po nim obliczaniem błędu e. Tak więc całkowity czas wykonania obliczeń w jednej iteracji algorytmu nie zmieni się i pozostanie na poziomie ośmiu jednostek. Na skutek wprowadzenia modułu GRADω nie

**Sieć ta służy wyłącznie do obliczeń równoległych, nie ma mowy o jej uczeniu.



Rysunek 4.5: Schemat sprzężonej sieci neuronowej służącej do obliczania kinematyki odwrotnej z ograniczeniami konstrukcyjnymi.

zmienia się też rola, jaką pełni w niej moduł STER. Ma on nadal czuwać nad procesem wykonywania obliczeń, z tym, że dodatkowo ma on kontrolować poziom błędu e_q.

W [BDJ+94, DK94, KM96] zaproponowano inną postać funkcji kary:

$$\omega(q) = \frac{1}{2} e_{\mathrm{m}}^T \Phi_{\mathrm{m}} e_{\mathrm{m}} + \frac{1}{2} e_{\mathrm{M}}^T \Phi_{\mathrm{M}} e_{\mathrm{M}}, \tag{4.15}$$

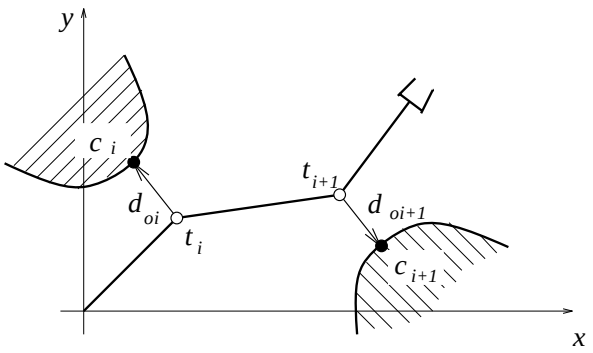
gdzie $e_{\mathrm{m}} = (e_i^{\mathrm{m}} \mid i = 1, \ldots, n)^T$, $e_{\mathrm{M}} = (e_i^{\mathrm{M}} \mid i = 1, \ldots, n)^T$, $e_i^{\mathrm{m}} = q_i^{min} - q_i$, $e_i^{\mathrm{M}} = q_i - q_i^{max}$, $\Phi_{\mathrm{m}} = diag[\varphi(e_i^{\mathrm{m}})]$, $\Phi_{\mathrm{M}} = diag[\varphi(e_i^{\mathrm{M}})]$, $\varphi(x) = 1/(1 + \exp(-\beta \cdot x))$. Przez przemnożenie funkcji kwadratowej przez funkcje sigmoidalną ominięto problem warunkowego definiowania błędu dodatkowego (porównaj z (4.7)). Gradient funkcji kary:

$$\frac{\partial \omega}{\partial q} = -\Phi_{\mathrm{m}} e_{\mathrm{m}} - D_{\mathrm{m}} \Phi'_{\mathrm{m}} e_{\mathrm{m}} + \Phi_{\mathrm{M}} e_{\mathrm{M}} + D_{\mathrm{M}} \Phi'_{\mathrm{M}} e_{\mathrm{M}}, \tag{4.16}$$

gdzie $D_{\mathrm{m}} = diag[e_i^{\mathrm{m}}]$, $D_{\mathrm{M}} = diag[e_i^{\mathrm{M}}]$, $\Phi'_{\mathrm{m}} = diag[\phi'(e_i^{\mathrm{m}})]$, $\Phi'_{\mathrm{M}} = diag[\phi'(e_i^{\mathrm{M}})]$, $\phi'(x) = \beta\phi(x)(1 - \phi(x))$, można zaimplementować jako sieć neuronową, w skład której wchodzą neurony o sigmoidalnej funkcji aktywacji.

4.3 Kinematyka odwrotna z omijaniem przeszkód jako problem optymalizacji

Niech $\hat{d}_o$ oznacza jednakową dla wszystkich przegubów manipulatora minimalną, bezpieczną odległość od przeszkód. Niech $d_{oi} = c_i - t_i$ będzie wektorem zaczepionym w środku układu współrzędnych związanym z i-tym przegubem manipulatora (kinematyka t_i wyraża jego położenie względem bazowego układu współrzędnych związanego z manipulatorem),



Rysunek 4.6: Manipulator w otoczeniu przeszkód.

kończącym się na najbliższym punkcie należącym do przeszkody (c_i wyraża położenie i-tego punktu należącego do przeszkody, wyrażone w bazowym układzie współrzędnych, zobacz rysunek 4.6).

Przy wprowadzonych oznaczeniach odwrotny problem kinematyki, w którym uwzględnia się obecność przeszkód w scenie roboczej manipulatora, może być transformowany do następującego problemu optymalizacji z ograniczeniami:

Problem 3 *Dla danego położenia p_f, wyrażonego we współrzędnych zewnętrznych, znaleźć taką konfigurację przegubów manipulatora q^*, która minimalizuje kryterium:*

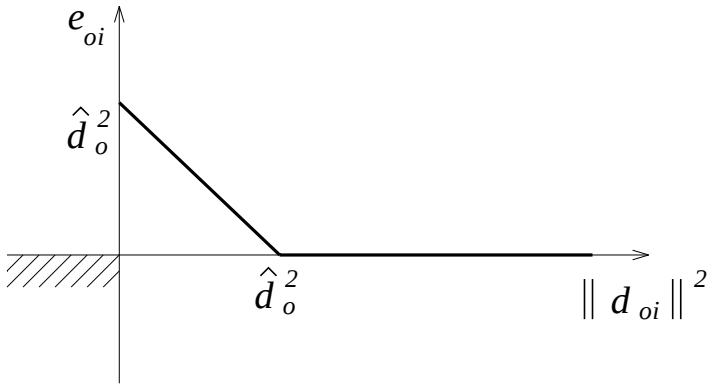
$$v(q) = \frac{1}{2} e(q)^T e(q) \tag{4.17}$$

przy spełnieniu ograniczeń

$$\|d_{oi}\| \geq \hat{d}_o, \quad i \in \{1 \ldots, n\}. \tag{4.18}$$

Uwaga: Założenie jednakowej wartości bezpiecznej odległości dla wszystkich przegubow jest uproszczeniem ogólniejszej sytuacji, w której dla różnych przegubów manipulatora zaproponowane byłyby różne wartości $\hat{d}_{oi}$. Podobnego założenia dokonano w poprzednim podrozdziale, deklarując jednakowy dla wszystkich przegubów margines bezpieczeństwa $\hat{d}_q$. Oczywiści, przy tym założeniu bezpośrednie zastosowanie prezentowanego poniżej algorytmu prowadzić może do otrzymywania niepoprawnych (kolizyjnych) rozwiązań (zwłaszcza, jeśli długości ramion manipulatora różnią się znacznie, gdy przeszkoda jest mała w porównaniu z ramieniem, itp.). Jeśli wziąć pod uwagę, że informacji o obecności przeszkód w scenie roboczej manipulatora dostarczać będą sensory ultradźwiękowe zamontowane na jego ramionach, to nawet agregacja uzyskiwanych sygnałów nie zawsze pozwoli na dostrzeżenie wszystkich niebezpieczeństw. Co więcej, zdarzyć się może, że ze względu na brak redundancji na odpowiednim kierunku, przeszkody, nawet doskonale widocznej, ominąć się nie da. Dlatego zakłada się, że manipulator (o proporcjonalnej budowie) wyposażony jest w sztuczną skórę, [CL89], i przechodzi w stan awaryjnego zatrzymania w przypadku zaistnienia kolizji. Oprócz sygnalizacji braku rozwiązań przez moduł sterowania procesem obliczań STER (zobacz rola modułu STER w podrozdziałach poprzednich) sztuczna skóra stanowić ma dodatkowy mechanizm zabezpieczeń przed kolizją. Wyjaśnienia wymaga tu jeszcze sposób zdefiniowania wektorów d_{oi}. Wyróżnienie przegubów jako punktów, względem których rozpatruje się obecność przeszkód, jest celowym zabiegiem*. Dzięki niemu pojawiające się w trakcie obliczeń ja-

*Porównaj z metodą transformowanego jakobianu, rozdział 2.



Rysunek 4.7: Wykres funkcji błędu dodatkowego e_{oi}.

kobiany można otrzymać wprost z kinematyk częściowych. Dla takiego podejścia „działa" przedstawiona poniżej metoda punktów wirtualnych. Do przegubów będą sprowadzane odczyty otrzymane z sensorów (zobacz rozdział 5.4.3). Można oczywiście, przy obliczaniu bezkolizyjnej kinematyki odwrotnej, wyróżniać inne punkty na manipulatorze (zobacz rysunek 2.3), jednak z uwagi na wiążące się z tym utrudnienia (zmienne jakobiany), tego się tutaj nie robi.

Rozwiązanie powyższego problemu można uzyskać poprzez jego transformację do problemu optymalizacji bez ograniczeń i, jak to było w przypadku problemu 2, ponowne skorzystanie z algorytmu (4.3) przy zmodyfikowanej postaci funkcji kryterialnej. W podobny też sposób syntezowana będzie implementująca algorytm gradientowy sieć neuronowa.
Niech e_{oi}, dane wzorem (4.19), oznacza błąd dodatkowy, powstały na skutek zbytniego zbliżenia się przegubu i do przeszkody (zobacz rysunek 4.7).

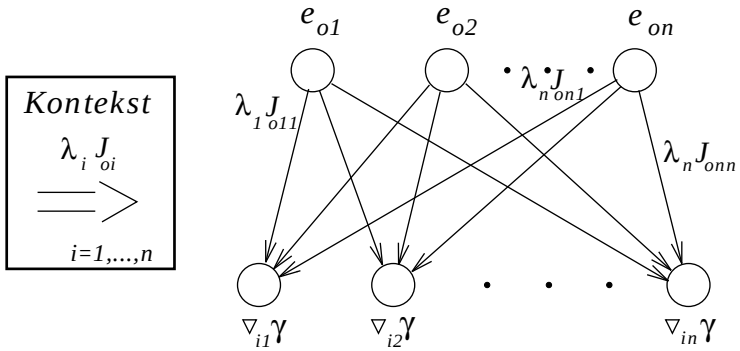
$$e_{oi} = \begin{cases} \hat{d}_o^2 - d_{oi}^T d_{oi}, & \text{gdy} \quad \|d_{oi}\| < \hat{d}_o \\ 0, & \text{gdy} \quad \|d_{oi}\| \geq \hat{d}_o \end{cases} . \tag{4.19}$$

Niech

$$\gamma(q) = \frac{1}{4} e_o^T \Lambda e_o, \text{ gdzie } \; e_o = (e_{o1}, \ldots, e_{on})^T, \Lambda = diag[\lambda_i], \lambda_i \geq 0 \tag{4.20}$$

oznacza funkcję kary za zbliżanie się przegubów manipulatora do przeszkód. Za pomocą parametrów λ_i można włączać, $(\lambda_i \geq 0)$, lub wyłączać, $(\lambda_i = 0)$, poszczególne składowe wektora e_o. Składową e_{oi} wyłącza się, gdy i-ty przegub nie przemieszcza się (np. jest kolumną manipulatora), gdy z góry wiadomo, że ścieżka jaką podąża jest bezkolizyjna (przypadek zaplanowanej ścieżki efektora) lub gdy nic nie wiadomo o najbliższym otoczeniu przegubu (odpowiada to sytuacji, gdy przegub nie jest wyposażony w żaden system sensoryczny).
Funkcja $\gamma(q)$ przyjmuje wartość 0, gdy przeguby manipulatora znajdują się względem przeszkody w odległości co najmniej równej $\hat{d}_o$. Nietrudno teraz pokazać, że w przypadku istnienia rozwiązania minimalizacja zmodyfikowanego kryterium

$$\tilde{v}(q) = v(q) + \gamma(q) = \frac{1}{2} e^T e + \frac{1}{4} e_o^T \Lambda e_o \tag{4.21}$$



Rysunek 4.8: Kontekstowa sieć neuronowa obliczająca $grad\gamma$.

przez zastosowanie algorytmu

$$\dot{q} = \begin{cases} -\alpha \frac{\partial \tilde{v}}{\partial q}, & \text{jeśli} \quad e \; lub \; e_{oi} \neq 0 \\ 0, & \text{jeśli} \quad e, e_{oi} = 0 \end{cases} \tag{4.22}$$

zapewni zbieżność błędów e, e_{oi} do zera[†], a więc przyniesie bezkolizyjne rozwiązanie odwrotnego problemu kinematyki.

Gradient zmodyfikowanej funkcji kryterialnej

$$\frac{\partial \tilde{v}}{\partial q} = -J_e^T e + J_o^T \Lambda e_o, \tag{4.23}$$

gdzie

$$J_o^T = [J_{o1}^T, \ldots, J_{on}^T], \quad J_{oi} = \begin{cases} d_{oi}^T J_i, & \|d_{oi}\| < \hat{d}_o \\ 0, & \|d_{oi}\| \geq \hat{d}_o \end{cases}, \quad J_i = \frac{\partial t_i}{\partial q} \tag{4.24}$$

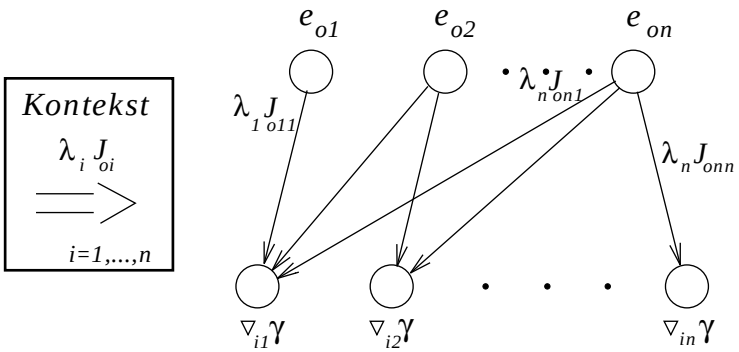
zawiera składnik $J_o^T \Lambda e_o$. Jego wartość obliczyć można przy pomocy przedstawionej na rysunku 4.8 kontekstowej sieci neuronowej[‡]. Kontekstem w sieci tej są współczynniki $\lambda_i J_{oi}$ (J_{oij}^T -j-ty wiersz i-tej kolumny J_o^T), zaś wejściem błędy e_{oi}. Wartość współczynników $\lambda_i J_{oi}$ obliczane są z kolei przy pomocy sieci kontekstowej, przedstawionej na rysunku 4.11. Wejściem do tej sieci są współrzędne wektorów d_i, kontekstem zaś przeskalowane przez λ_i elementy jakobianów J_i. Sposób równoległego obliczania samych błędów e_{oi} przedstawia rysunek 4.10.

Wnikliwie analizując wzory w (4.24) można zauważyć, że sieć tą można znacznie uprościć (część z wag sieci zawsze równa jest 0, $J_{oij} = 0$ dla $j > i$). Na rysunku 4.9 przedstawiono jej uproszczoną wersję.

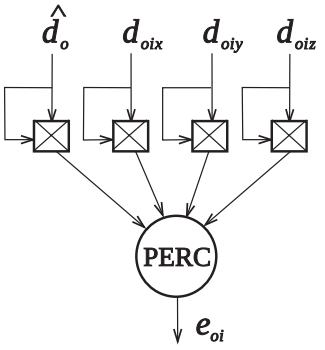
Sieć neuronową służącą do obliczania kinematyki odwrotnej z omijaniem przeszkód zgodnie z algorytmem (4.22), sprzęgającą wszystkie wspomniane powyżej sieci, przedstawia rysunek 4.12. Moduł oznaczony symbolem GRADγ zawiera sieci kontekstowe obliczające wyrażenia $\lambda_i J_{oi}^T d_{oi}$ (czyli gradient funkcji kary γ). Moduł $\Lambda, d_o, \hat{d}_0$ dostarcza do modułów λJ_o i e_o wektorów d_{oi} i $\hat{d}_{oi}$ oraz parametrów λ_i. Moduły λJ_o i e_0 są sieciami neuronowymi obliczającymi, odpowiednio do swoich nazw, wartości współczynników $\lambda_i J_{oi}$ i błędów e_{oi}.

[†]Dowód przez zastosowanie metody funkcji Lapunowa.

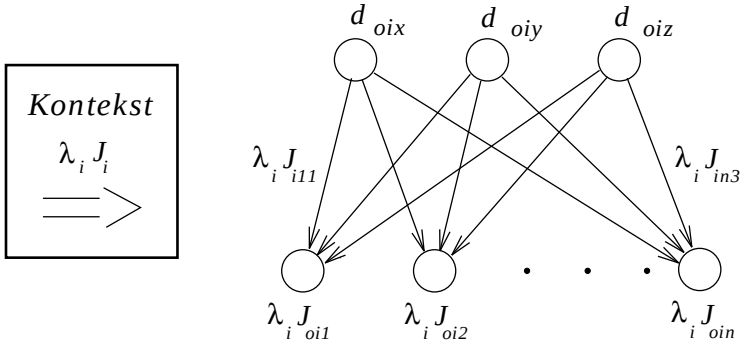
[‡]Sieć ta służy wyłącznie do równoległego obliczania, nie ma mowy o jej uczeniu.



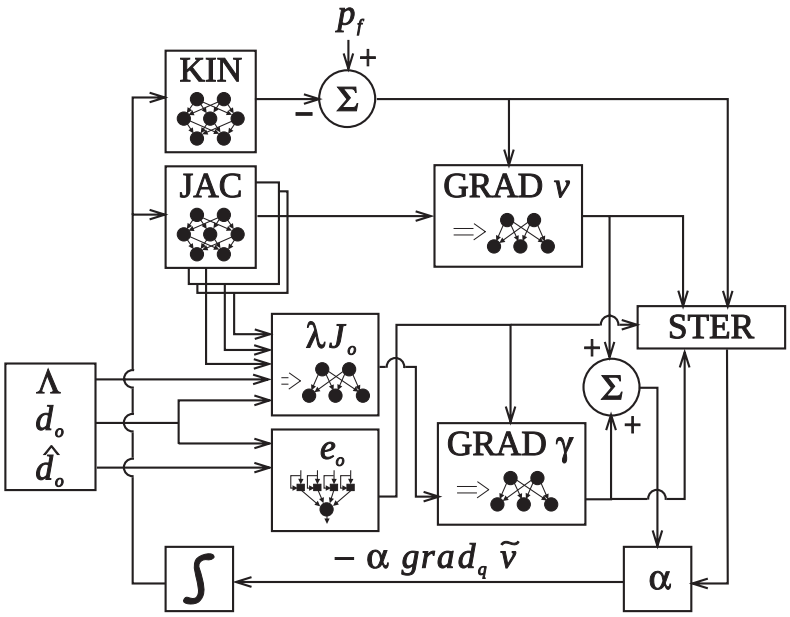
Rysunek 4.9: Uproszczona kontekstowa sieć neuronowa obliczająca $grad\gamma$.



Rysunek 4.10: Sieć neuronowa obliczająca e_{oi}.



Rysunek 4.11: Kontekstowa sieć neuronowa obliczająca $\lambda_i J_{oi}$.



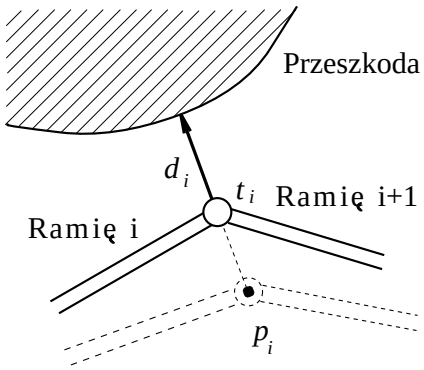
Rysunek 4.12: Schemat sieci neuronowej służącej do obliczania kinematyki odwrotnej z omijaniem przeszkód.

Czas potrzebny na wykonywanie obliczeń (przy założeniu, że propagacja danych poprzez jedną warstwę sieci zabiera jedną jednostkę czasu) jest taki sam dla modułów λJ_o i GRADγ i wynosi 1. Natomiast czas potrzebny na wykonanie obliczeń przez moduł e_o jest równy 2. Ponieważ obliczenia w module λJ_o mogą się odbywać równolegle z obliczaniem błędu e, ponieważ obliczenia w module e_o mogą się odbywać równolegle z obliczeniami w module KIN, ponieważ obliczenia w module GRADγ mogą odbywać się równolegle z obliczeniami w module GRADγ, oszacowanie czasu wymaganego na wykonanie jednej iteracji algorytmu gradientowego nie zmieni się (patrz poprzednie podrozdziały), pozostając na poziomie 8 jednostek. Niewiele zmieni się natomiast rola modułu sterowania procesem obliczeń STER. Dodatkowo będzie on musiał kontrolować poziom błędu e_o.

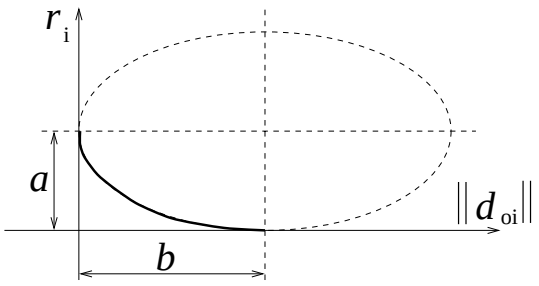
W celu uzyskania bezkolizyjnego rozwiązania, w [KM96] zaproponowano alternatywną funkcję kary, γ_p, wyrażającą się wzorem:

$$\gamma_p(q) = \sum_{i=1}^{n} \lambda_i e_{pi}^T e_{pi}, \tag{4.25}$$

gdzie $\lambda_i \geq 0$, $e_{pi} = p_i - t_i$, p_i - położenie i-tego punktu wirtualnego. Interpretacja parametrów λ_i w powyższym wzorze jest taka sama, jak w (4.20). Inaczej natomiast zdefiniowane są błędy dodatkowe. Porównując $v(q)$, (4.1), z $\gamma_p(q)$, (4.15), zauważyć można pewną analogię. Minimalizacja $v(q)$ prowadziła do zmniejszania się błędu e, aż do chwili, gdy $p_f = t_n(q)$. Podobnie minimalizacja $\gamma_p(q)$ prowadzić musi do zmniejszenia błędów e_{pi}, niekoniecznie jednak aż do chwili, gdy $p_i = t_i$. Wytłumaczenie tego faktu ukryte jest w sposobie zdefiniowania położenia punktów wirtualnych p_i i doborze parametrów λ_i. Na rysunku 4.13 przedstawiony jest jeden z przegubów manipulatora, który znalazł się niebezpiecznie blisko przeszkody. Aktualne położenie przegubu, t_i, dane jest z równań kinematyki częściowej (kinematyki do przegubu i). Położenie pożądane, p_i (tożsame z



Rysunek 4.13: Wyznaczanie położenie punktu wirtualnego.



Rysunek 4.14: Funkcja skalująca $r(d_{oi})$.

położeniem punktu wirtualnego), przy którym nie dojdzie do kolizji oblicza się raz[§], przed przystąpieniem do minimalizacji, ze wzoru:

$$p_i = t_i - \frac{d_{oi}}{\|d_{oi}\|} r_i(d_{oi}), \tag{4.26}$$

gdzie d_{oi} jest, jak poprzednio, wektorem skierowanym do najbliższej przegubowi przeszkody, $r_i(d_{oi}) = a(1 - \sqrt{1 - \frac{(\|d_{oi}\|-b)^2}{b^2}})$ jest funkcją skalującą, parametryzowaną przez a i b (zobacz rysunek 4.14).

Jeśli więc p_i jest położeniem osiągalnym przez i-ty przegub manipulatora[¶], zastosowanie algorytmu:

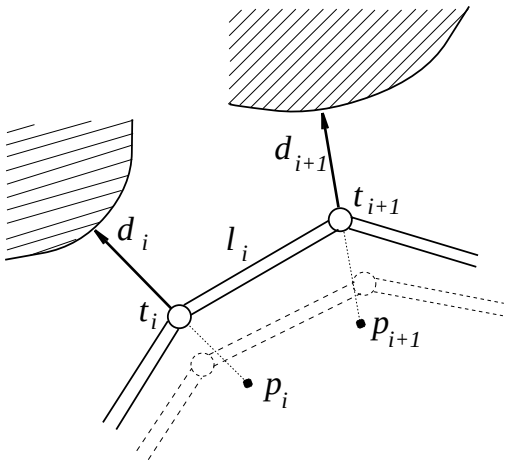
$$\dot{q} = \begin{cases} -\alpha \frac{\partial \tilde{v}_p}{\partial q}, & \text{jeśli} \quad e \text{ lub } e_{pi} \neq 0 \\ 0, & \text{jeśli} \quad e, e_{pi} = 0 \end{cases}, \quad \text{gdzie } \tilde{v}_p(q) = v(q) + \gamma_p(q) \tag{4.27}$$

pozwoli obliczyć bezkolizyjne rozwiązanie. Gradient zmodyfikowanej funkcji kryterialnej:

$$\frac{\partial \tilde{v}_p}{\partial q} = \frac{\partial v}{\partial q} + \frac{\partial \gamma_p}{\partial q} = -J_e^T e - \sum_{i=1}^{n} \lambda_i J_i^T e_{pi} \tag{4.28}$$

[§]Innym sposobem wyznaczenia położenia punktów wirtualnych jest skorzystanie z metod fuzji sygnałów sensorycznych (zobacz rozdział 5.4.3).

[¶]Założenie osiągalności położenia p_i przez i-ty przegub manipulatora potrzebne jest do przeprowadzenia dowodu zbieżności algorytmu (4.27) metodą funkcji Lapunowa.



Rysunek 4.15: Położenie nieosiągalnych punktów wirtualnych.

zawiera składnik $-\sum_{i=1}^{n} \lambda_i J_i^T e_{pi}$, który, podobnie jak $-J_e^T e$, można zaimplementować przy pomocy kontekstowych sieci neuronowych.

Na rysunku 4.15 pokazana jest sytuacja, w której punkty wirtualne p_i, p_{i+1} nie są osiągalne przez przeguby t_i, t_{i+1} manipulatora ($l_i > \|p_{i+1} - p_i\|$). Minimalizując kryterium $\tilde{v}_p(q)$ można jednak obliczyć nowe, bezpieczniejsze ze względu na kolizję położenie manipulatora (linia przerywana). Powtarzając wielokrotnie operacje wyznaczania położenia punktów wirtualnych i minimalizacji kryterium $\tilde{v}_p(q)$ uzyskuje się, jeśli jest to możliwe, wyzerowanie błędów e_{pi}, a więc bezkolizyjne rozwiązanie. Cel ten można również osiągnąć w inny sposób, przez odpowiednie włączanie i wyłączanie ograniczeń (sterowanie parametrami λ_i).

4.4 Kinematyka odwrotna z omijaniem przeszkód oraz ograniczeniami konstrukcyjnymi jako problem optymalizacji

W niniejszym podrozdziale do rozwiązania odwrotnego problemu kinematyki wykorzystane zostaną wyniki rozważań z podrozdziałów 4.1, 4.2, 4.3.

Niech $\omega(q)$ i $\gamma(q)$ będą funkcjami kary, danymi, odpowiednio, przez (4.8) oraz (4.20) lub (4.25). Niech $v(q)$ będzie dane równaniem (4.1). Odwrotny problem kinematyki, w którym poszukuje się bezkolizyjnego rozwiązania bez przekraczania ograniczeń konstrukcyjnych, może być transformowany do następującego problemu optymalizacji bez ograniczeń:

Problem 4 *Dla danego położenia p_f, wyrażonego we współrzędnych zewnętrznych, znaleźć taką konfigurację przegubów manipulatora q^*, która minimalizuje kryterium:*

$$\hat{v}(q) = v(q) + \omega(q) + \gamma(q). \tag{4.29}$$

Rozwiązanie powyższego problemu można uzyskać stosując następujący algorytm:

$$\dot{q} = -\alpha \frac{\partial \hat{v}}{\partial q}. \tag{4.30}$$

Zbieżność tego algorytmu, podobnie jak przy problemach 1, 2, 3 można pokazać, posługując się metodą funkcji Lapunowa[‖].

4.5 Zbieżność algorytmu

Występującemu w równaniach (4.3), (4.10), (4.22), (4.27) parametrowi α nie poświęcono jak dotąd zbyt dużej uwagi. Aby zapewnić zbieżność odpowiednich algorytmów wystarczyło zauważyć, że α powinno być nieujemne. Jeśli jednak, oprócz zbieżności, w grę wchodzi również szybkość, z jaką otrzymuje się rozwiązania, o parametrze α należy powiedzieć coś więcej.

Rozważane dotychczas algorytmy miały postać:

$$\dot{q} = -\alpha \operatorname{grad}_q V, \tag{4.31}$$

gdzie V - to pozytywnie zdefiniowana funkcja kryterialna. Ponieważ

$$\dot{V} = -\alpha \|\operatorname{grad}_q V\|^2 \tag{4.32}$$

algorytm (4.31) jest zbieżny, lecz prędkość z jaką się zbiega maleje wraz ze zbliżaniem się do rozwiązania. Sytuacji tej można zaradzić, przyjmując, [LB91, LK94]:

$$\alpha = -\frac{V^r}{\|\operatorname{grad}_q V\|^2}. \tag{4.33}$$

Przy takim α, startując z q^0, czas potrzebny na uzyskanie rozwiązania q^*, t_c, dany jest przez:

$$t_c = \int_{V(q^0)}^{V(q^*)} \frac{dV}{\dot{V}} = -\int_{V(q^0)}^{V(q^*)} V^{-r} dV = \begin{cases} \infty, & \text{gdy } r \geq 1 \\ \frac{V(q^0)^{1-\alpha}}{1-\alpha}, & \text{gdy } \frac{1}{2} < r < 1 \end{cases}. \tag{4.34}$$

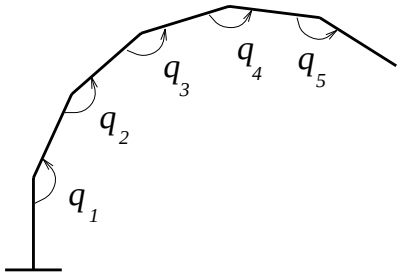
Widać stąd, że jeśli $r \geq 1$, zbieżność algorytmu (4.31) jest asymptotyczna ($V(t) = V(q^0)e^{-t}$, gdy $r = 1$). Jeśli natomiast $\frac{1}{2} < r < 1$, algorytm ten jest zbieżny w skończonym czasie.

Przedstawione powyżej propozycja obliczania parametru α niesie z sobą pewne zagrożenie. Przy zbliżaniu się do minimów lokalnych funkcji V, wartość α rośnie nieskończenie ($\|\operatorname{grad}_q V\| \approx 0$). W [LB91] własność tą wykorzystano do omijania minimów lokalnych. Aby takie minima rozpoznać, wystarczy kontrolować wartość błędu e i gradientu $\operatorname{grad}_q V$. Innym sposobem na omijanie minimów lokalnych może być zastosowanie algorytmu tunelowego, zaproponowanego w [LM80, LMGC82, GL82], a zastosowanego do zagadnień robotycznych w [YI95] (zobacz dodatek C).

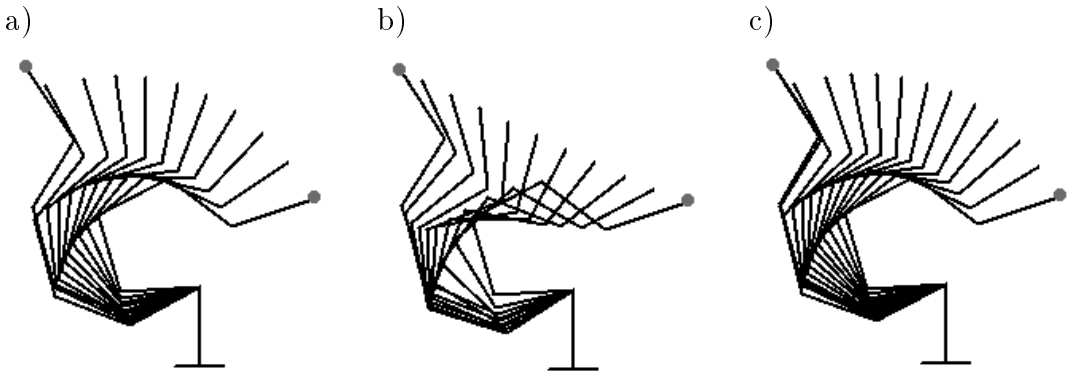
4.5.1 Przykłady neuronowego obliczania kinematyki odwrotnej

Podczas symulacji, których wyniki przedstawione są poniżej, starano się sprawdzić, czym charakteryzują się rozwiązania otrzymywane za pomocą sprzężonej sieci neuronowych oraz jaki wpływ na nie ma uwzględnienie ograniczeń. Symulacje te były przeprowadzane,

[‖]Niech $\hat{v}(q)$ będzie kandydatką na funkcję Lapunowa ($\hat{v}(q) > 0$, gdy $q \neq q^*$; $\hat{v}(q) = 0$, gdy $q = q^*$). Ponieważ $\dot{\hat{v}}(q) = \frac{\partial \hat{v}}{\partial q} \cdot \dot{q} = -\alpha \|\frac{\partial \hat{v}}{\partial q}\|^2$ jest ujemne wzdłuż trajektorii $\dot{q}$, algorytm 4.30 jest zbieżny.



Rysunek 4.16: Model manipulatora wykorzystywanego w czasie symulacji.



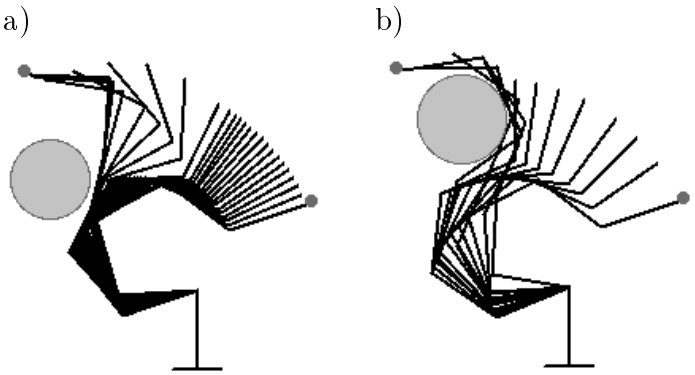
Rysunek 4.17: Rozwiązania odwrotnego problemu kinematyki otrzymane za pomocą: a) neuronowo zaimplementowanego algorytmu gradientowego; b) algorytmu GCM; c) metody całkowania Runge-Kutta.

między innymi, na przykładzie pokazanego na rysunku 4.16 manipulatora (manipulator ten posiada 5 przegubów rotacyjnych, o osiach obrotów prostopadłych do płaszczyzny rysunku, oraz nieruchomą podstawę).

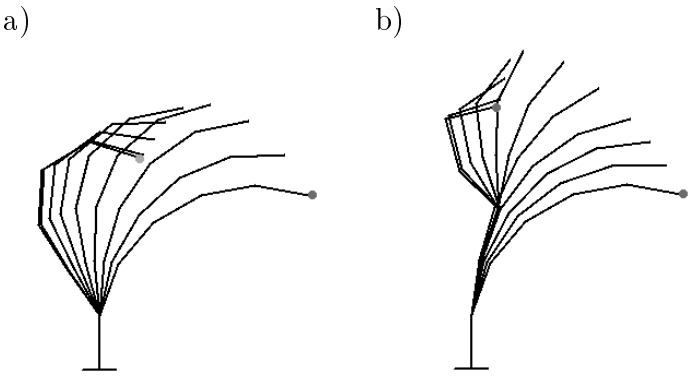
Na rysunku 4.17 przedstawione są kolejne położenia manipulatora odpowiadające pośrednim rozwiązaniom odwrotnego problemu kinematyki. W przypadku a) rozwiązania te były obliczone za pomocą neuronowo zaimplementowanego algorytmu gradientowego**, w przypadku b) – za pomocą algorytmu GCM (*Gradient Descent Method*, [Wil90]), w przypadku c) – przy użyciu metody całkowania Runge-Kutta. Ostatnie z położeń manipulatora jest rozwiązaniem końcowym††. Punkt znajdujący się po prawej stronie podstawy manipulatora odpowiada położeniu początkowemu jego efektora, natomiast punkt po lewej stronie podstawy jest położeniem zadanym. Jak można zauważyć, w przypadku a) i c) różnice między otrzymanymi rozwiązaniami są niewielkie. Ich podobieństwo wynika z posłużenia się przy minimalizacji kryterium v (zobacz problem 1) tym samym równaniem (zobacz równanie (4.3)), aczkolwiek całkowanym różnymi metodami. W wyniku całkowania rozwiązanie końcowe poszukiwane jest przy jak najmniejszych zmianach

**Moduł całkowania w neuronowej implementacji algorytmu gradientowego zrealizowany był przez jednowarstwową sieć neuronową z pamięcią, której wyjścia i stan opisać można równaniami: $out(j) = state(j-1) + in(j), state(j) = out(j)$.

††Rozwiązania pośrednie zostały tu wyróżnione, aby zademonstrować charakter pracy poszczególnych algorytmów. Normalnie, przy obliczaniu kinematyki odwrotnej, widocznym jest tylko rozwiązanie końcowe.



Rysunek 4.18: Bezkolizyjne rozwiązania odwrotnego problemu kinematyki otrzymane z neuronowo zaimplementowalnego algorytmu gradientowego.

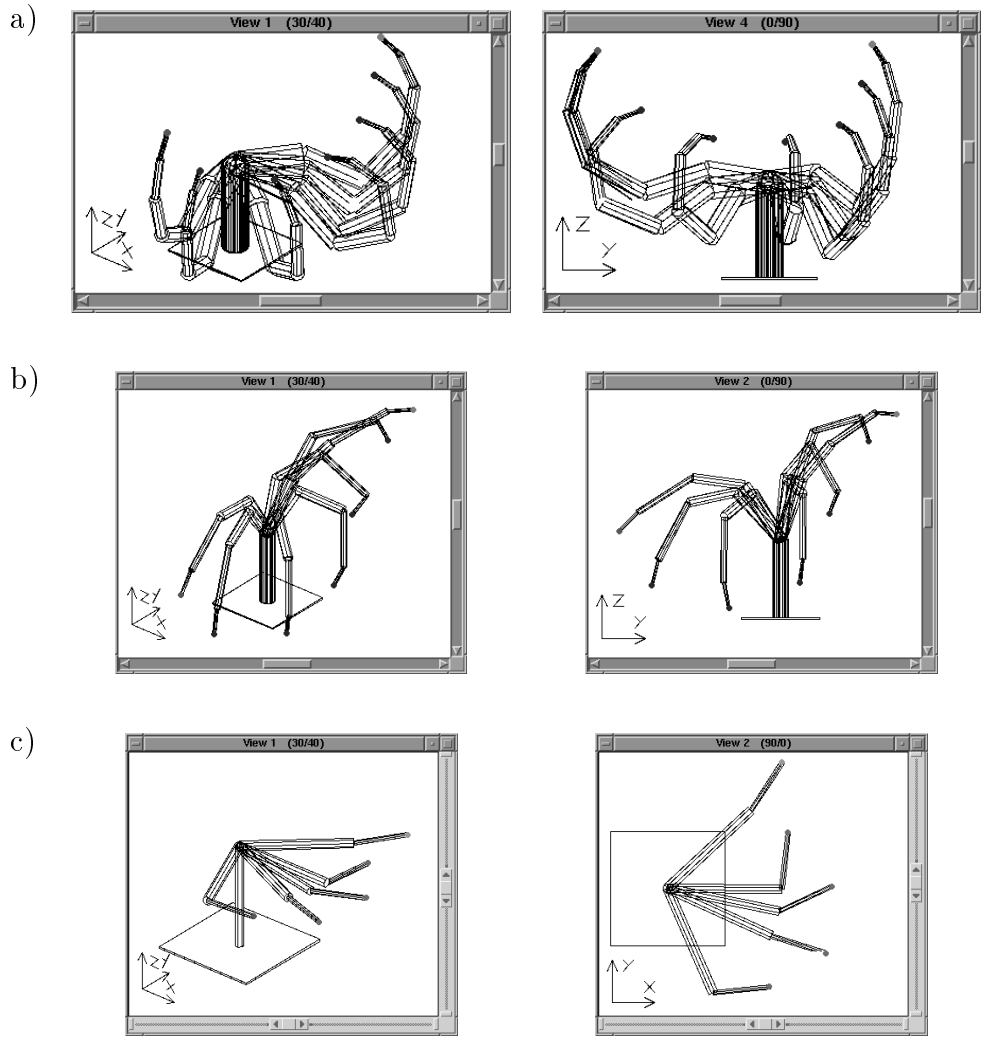


Rysunek 4.19: Rozwiązania odwrotnego problemu kinematyki otrzymane z neuronowo zaimplementowanego algorytmu gradientowego: a) bez uwzględniania ograniczeń; b) z uwzględnieniem ograniczeń na zakres zmian kątów w przegubach.

wartości współrzędnych wewnętrznych. Dlatego efektor manipulatora w rozwiązaniach pośrednich zatacza łuk w przestrzeni zewnętrznej. Efekt ten nie występuje, gdy do minimalizacji funkcji kryterialnej zostaje użyty algorytm GCM. Jak widać na omawianym rysunku (przypadek c)), algorytm GCM dostarcza rozwiązań pośrednich, w których efektor manipulatora podąża wzdłuż linii prostej w przestrzeni zewnętrznej. Efekt uwzględnienie ograniczeń (wynikających z obecności przeszkód oraz konstrukcji manipulatora) podczas obliczania kinematyki odwrotnej za pomocą neuronowo zaimplementowanego algorytmu gradientowego, zaobserwować można na rysunkach: 4.18 i 4.19. Przedstawiony na nich manipulator jest tym samym manipulatorem co poprzednio, z tym, że w przykładzie ilustrowanym rysunkiem 4.19 ma on inną konfigurację początkową[‡‡].

Jak widać (porównaj rysunek 4.17a) z rysunkiem 4.18), pojawienie się przeszkody na scenie roboczej nie wpływa na otrzymywane rozwiązania pośrednie, dopóki znajduje się ona w odległości większej niż zadany margines bezpieczeństwa. Gdy odległość manipulatora od przeszkody maleje poniżej tego progu, manipulator zaczyna się od niej odchylać,

[‡‡]Punkt leżący najdalej po prawej stronie podstawy manipulatora odpowiada początkowemu położeniu jego efektora.



Rysunek 4.20: Rozwiązania odwrotnego problemu kinematyki dla manipulatora: a) o siedmiu stopniach swobody; b) o pięciu stopniach swobody; c) o dwóch stopniach swobody.

podążając jednak niezmiennie efektorem w kierunku zadanego położenia. Część a) rysunku 4.18 pokazuje rozwiązania pośrednie, otrzymane z algorytmu (4.22), natomiast część b) - rozwiązania pośrednie otrzymane z algorytmu (4.27).

Wpływ ograniczeń konstrukcyjnych na otrzymywane rozwiązanie obrazuje rysunek 4.19. W części a) tego rysunku widoczne są kolejne rozwiązania pośrednie, otrzymane bez uwzględniania ograniczeń konstrukcyjnych. W części b) natomiast pokazane są rozwiązania, przy otrzymaniu których zakres zmienności współrzędnych wewnętrznych q_1 i q_2 ograniczony został do przedziału $[0°, 180°]$. Wartość marginesu bezpieczeństwa, $\hat{d}_q$, ustawiona była na $10°$, i jak widać, zastosowany algorytmu (4.10) nie dopuścił do przekroczenia tego marginesu.

Przedstawione do tej pory przykłady obliczania kinematyki odwrotnej dla prostoty rysunków umiejscowione były w przestrzeni dwuwymiarowej. Aby pokazać, że zaproponowane algorytmy działają równie dobrze w przestrzeni trójwymiarowej, przeprowadzono symulacje, których wyniki zostały zebrane na rysunku 4.20. Widoczne na nim kolejne położenia manipulatorów są końcowymi rozwiązaniami otrzymanymi z neuronowo zaim-

plementowanego algorytmu gradientowego. Jak widać, niezależnie od ilości stopni swobody (manipulator pokazany w części a) tego rysunku miał ich siedem, w części b) - pięć, w części c) - tylko dwa) każdy manipulator swoim efektorem osiągnął kolejno wszystkie zadane, widoczne w postaci punktów, położenia.

Rozdział 5

Planowanie ruchu manipulatora w wirtualnym otoczeniu

Wśród wielu zagadnień, jakimi zajmuje się ówczesna robotyka, sporą grupę stanowią zagadnienia związane pośrednio lub bezpośrednio z budową modelu otoczenia. Jako przykład wymienić tu można problemy rozpoznawania sceny i samolokalizacji, problemy montażowe i znajdowania optymalnego chwytu, problemy monitorowania, planowania ruchów i działań, czy też w końcu problemy planowania ruchu. Żaden też symulator, przeznaczony do testowania opracowanych metod, nie byłby symulatorem w pełnym znaczeniu tego słowa, gdyby nie dawał możliwości dowolnego modelowania sceny. Dlatego też, jeśli mówi się o planowaniu ruchu z omijaniem przeszkód, powinno się również powiedzieć coś więcej na temat przyjętej ich reprezentacji, metodzie liczenia odległości i rozpoznawaniu kolizji. Wypełnieniu tego właśnie obowiązku poświęcony jest niniejszy rozdział.

5.1 Metody reprezentacji przeszkód w modelu otoczenia

Problem modelowania sceny najszerzej chyba rozwinął się na gruncie zastosowań komputerowo wspomaganych systemów wytwarzania (CAD Systems). W zależności od celu, jakiemu mają służyć, systemy te charakteryzują odmienne sposoby reprezentacji obiektów. Do najczęściej spotykanych metod reprezentacji należą, [Hay86, BHW88]:

Metoda *wire frame geometry*: w metodzie tej dany obiekt reprezentowany jest przez zbiór punktów (trójek (x, y, z) odpowiadających wierzchołkom) i zbiór linii łączących te punkty. *Wire frame* jest właśnie nazwą obiektu, który powstanie przez połączenie punktów liniami.

Metoda *surface models*: w metodzie tej wierzchołki i krawędzie danego obiektu wprowadzane są w uporządkowany sposób, aby można było określić, które z nich ograniczają daną ścianę obiektu. Przez dodanie dodatkowego parametru, mówiącego o typie ściany (wydrążona, pełna), metodą tą można modelować obiekty bardziej złożone niż metodą *wire frame*.

Metoda *polygon modelling*: w metodzie tej dany obiekt reprezentowany jest przez uporządkowany zbiór wierzchołków, listę krawędzi (wskazującą na odpowiednie wierzchołki) i listę ścian (wskazującą na odpowiednie krawędzie). Metodę tą można też spotkać

pod nazwą *boundary representation*.

Metoda *solid modelling*: w metodzie tej korzystając z przygotowanej wcześniej (np. metodą *polygon modelling*) biblioteki obiektów podstawowych, tzw. prymitywów, można konstruować obiekty bardziej złożone. Jedną z najszerzej stosowanych metod tego typu jest metoda CSG (*Constructive Solid Modelling*), w której dozwolone są operacje dodawania, odejmowania, wydzielania części wspólnej dwóch obiektów.

Metoda *sweep representation*: w metodzie tej wykorzystany jest fakt, że dany obiekt można wygenerować „omiatając" przestrzeń jakąś powierzchnią. Jeśli więc powierzchnia zadana jest jakąś funkcją krzywoliniową i jeśli zadana jest również trajektoria tej powierzchni (położenie i orientacja), to funkcja ta wraz z trajektorią stanowią reprezentację obiektu.

Metoda *cell decomposition*: metoda ta zbliżona jest do trójwymiarowej triangularyzacji. Dany obiekt reprezentowany jest jako zbiór brył elementarnych (zazwyczaj tetrahedronów) stykających się ścianami, krawędziami lub wierzchołkami. Do grupy metod *cell decomposition* zalicza się metody aproksymacyjne (aproksymacja kulami, elipsami, sześcianami, itp.).

Metoda *spatial occupancy enumeration*: w metodzie tej obiekt reprezentowany jest przez listę zajmowanych przez siebie cel (czasem też nazywanych *voxels* - *volume elements*) trójwymiarowej siatki. Celami zazwyczaj są sześciany. Metoda ta jest szczególnym przypadkiem metody *cell decomposition* (wszystkie cele mają jednakowy kształt i rozmiar).

Metoda *octree representation*: metoda ta pochodzi bezpośrednio z metody *spatial occupancy enumeration*. Jedyną różnicą jest inny, oszczędniejszy sposób zapisu danych. Zamiast pamiętać każdą z zajętych przez obiekt cel, dane umieszcza się w strukturze drzewiastej. Struktura ta wynika z następującego sposobu postępowania: na początek przestrzeń zajmowaną przez obiekt dzieli się na osiem części (oktantów). Każda z tych części, jeśli nie jest zajęta w całości przez obiekt, jest rekurencyjnie dzielona na kolejne osiem części. Dzieje się tak aż do chwili, gdy dalszy podział nie jest już potrzebny lub gdy najmniejszy oktant osiągnie rozmiar siatki.

O części z tych metod wspominać się będzie raz jeszcze, przy okazji omawiania metod obliczania odległości i detekcji kolizji (patrz kolejny podrozdział). Jednak szczególnie dużo uwagi poświęcone zostanie metodzie *polygon modelling*, która doczeka się neuronowej implementacji (patrz podrozdział 5.3.1).

5.2 Standardowe metody obliczania odległości i detekcji kolizji

Kolizja (*łac.* collisio - zderzenie) jest pojęciem opisującym zdarzenie, jakie zachodzi podczas ruchu obiektów, gdy odległość między nimi maleje do zera i wskutek czego zmieniają się parametry ruchu. Powyższa definicja ma „fizyczny" charakter, nie dopuszcza do wzajemnego przenikania się obiektów. Nieco odmiennie interpretuje się kolizję przy formułowaniu wykrywających ją algorytmów. Ponieważ algorytmy te operują na modelach, a nie na rzeczywistych obiektach, kolizja rozpoznawana jest po pojawieniu się części wspólnej rozłącznych dotychczas zbiorów punktów reprezentujących obiekty. W [GH89] problem detekcji kolizji zdefiniowany został w następujący sposób:

Problem 5 *Niech* $K_1(\theta_b) \cap K_2(\theta_e) = \emptyset$, *gdzie* $K_i = R(\theta)C_i + \{p_i(\theta)\}$, $K_i \in \mathbb{R}^m$, $R_i(\theta)$

– macierz obrotu, p_i – wektor przesunięcia, $C_i \in \mathbb{R}^m$ – zbiór punktów odpowiadający przestrzeni zajmowanej przez obiekt i w lokalnym układzie współrzędnych, $\theta \in \Theta = [\theta_b, \theta_e]$ – parametr, którym parametryzowana jest ścieżka ruchu, θ_b i θ_e odpowiadają odpowiednio początkowemu i końcowemu punktowi ścieżki. Znaleźć punkt kolizji

$$\theta = \ min \ \{\theta \in \Theta; k_1(\theta)k_2(\theta) \neq \emptyset\} \tag{5.1}$$

lub pokazać, że

$$K_1(\theta_b)K_2(\theta_e) \neq \emptyset \quad dla\ wszystkich\ \theta \in \Theta. \tag{5.2}$$

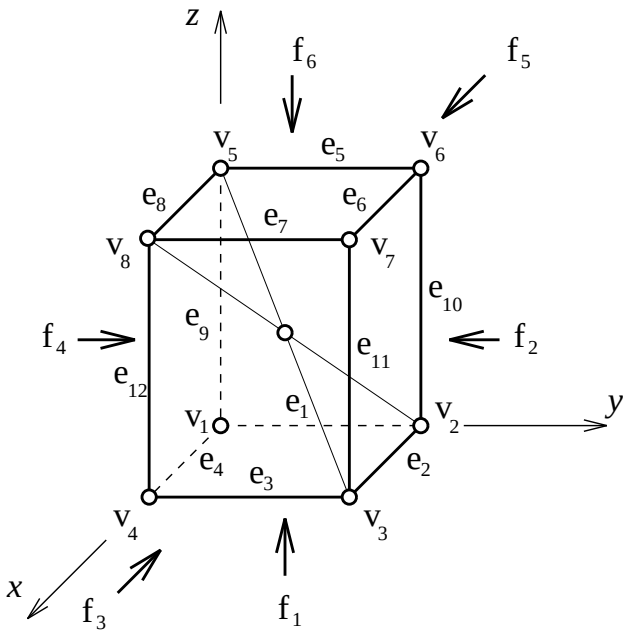
Jego rozwiązanie otrzymano zaś stosując iteracyjny algorytm BCDA (*Basic Collision Detection Algorithm*) oraz jego modyfikację, tj. algorytm PA (*Polytope Algorithm*). W obu algorytmach wykorzystuje się funkcje i przekształcenia pomocnicze, aby poprzez znalezienie najbliższych sobie punktów, należących do dwóch różnych obiektów, oraz poprzez rozwiązanie równania, w którym uwzględnia się ich odległość, można było stwierdzić, czy w danym kroku k (dla danego θ_k) nastąpiła kolizja. Algorytmy te operują na obiektach reprezentowanych jak w metodzie *polygon modelling*. W [GF89] zasięg działania procedury obliczania odległości rozszerzono z wielościanów na obiekty bardziej złożone, których opis wymaga już użycia metody *sweep representation*.

Inne podejście do problemu rozpoznawania kolizji przedstawiono w [Hay86, SH92]. Ponieważ do reprezentacji obiektów posłużono się tam metodą *octree representations*, detekcję kolizji utożsamiono z procesem testowania cel zajętych przez obiekty. Jeśli dwa obiekty miały jakąś celę wspólną, świadczyło to o wystąpieniu kolizji. Do podobnego zagadnienia sprowadza się testowanie kolizji przy aproksymacyjnych metodach reprezentacji. Jednak w tym przypadku o kolizji świadczy przecinanie się brył elementarnych, którymi obiekty aproksymowano*. W [Jac91] wspomina się o metodzie aproksymacji kulami, elipsoidami i prostopadłościanami w omawianym kontekście. Gotowy algorytm do testowania kolizji między prostopadłościanami przedstawiono w [Mey86].

Obliczanie odległości, jakkolwiek stanowi główną procedurę w algorytmach detekcji kolizji ([GH89]), jest problemem ważnym samym w sobie. Dlatego też częstokroć omawia się go osobno. W [Sch81] problem obliczania odległości pomiędzy obiektami O_1 i O_2 sprowadzono do problemu obliczania odległości między punktem $x = 0$ a tzw. *sumą Minkowskiego* obiektów O_1 i $-O_2$. Użycie *sumy Minkowskiego* jest standardowym chwytem, w którym: jeśli O_1 i O_2 wypukłe, to $O_1 \pm O_2$ też wypukłe; jeśli O_1 i O_2 mają, odpowiednio, n_1 i n_2 wierzchołków, to $O_1 \pm O_2$ ma wierzchołków $n_{12} \leq n_1 + n_2$; $O_1 \pm O_2 = \{x \pm y : x \in O_1, y \in O_2\}, -O = \{-x : x \in O\}$. Niestety, gotowe algorytmy podano tylko dla przypadków obiektów dwuwymiarowych. Wprowadzenie trzeciego wymiaru powoduje znaczne skomplikowanie rozwiązania.

Obliczaniem odległości w przestrzeni trójwymiarowej zajął się również Lumelski, [Lum85]. Algorytm, który przedstawił pozwala na szybkie obliczanie odległości pomiędzy dwoma dowolnymi odcinkami. Przy okazji obliczania odległości znajdowane są również współrzędne najbliższych sobie punktów, należących do tych odcinków.

*W [Hay86] wspomina się, że jeśli obiekty O_1 i O_2 są wielościanami wypukłymi o odpowiednio, O_{1E} i O_{2E} krawędziach oraz O_{1F} i O_{2F} ścianach, to trzeba wykonać $O_{1E}O_{2E} + O_{1F}O_{2E} + O_{1E}O_{2F}$ testów przecinania się ścian i krawędzi, aby stwierdzić kolizję.



Rysunek 5.1: Przykładowy sposób ponumerowania wierzchołków, krawędzi i ścian prostopadłościanu.

5.3 Neuronowy model otoczenia

Spośród wymienionych w podrozdziale 5.1 metod reprezentacji obiektów, metoda *polygon modelling* charakteryzuje się szczególną własnością. W metodzie tej bez większych problemów wyznaczyć można znormalizowane równania płaszczyzn, zawierających ściany obiektu (równania ścian)[†]. Bez większych też kłopotów można sprawić, aby dla punktów należących do wnętrza obiektu z równań tych otrzymano wartości ujemne[‡]. Własność tą skrzętnie wykorzystano przy stworzeniu neuronowej reprezentacji obiektu. Wykorzystano ją również w neuronowo implementowanych algorytmach liczenia odległości i detekcji kolizji.

5.3.1 Neuronowa reprezentacja obiektu

Stworzenie metodą *polygon modelling* reprezentacji dowolnego obiektu (wielościanu wypukłego) wymaga wprowadzenia jednoznacznej numeracji jego wierzchołków, krawędzi, ścian. Numeracja ta jest potrzebna do zachowania właściwego porządku na listach wierzchołków, krawędzi i ścian, stanowiących strukturę danych reprezentującą dany obiekt. Dla przykładu, na rysunku 5.1 przedstawiono prostopadłościan, którego wierzchołki ponumerowane są od v_1 do v_8, krawędzie od e_1 do e_2, ściany od f_1 do f_6, natomiast środek ciężkości oznaczony jest przez v_0. Strukturę danych reprezentującą ten prostopadłościan przedstawia rysunek 5.2.

[†]Do wyznaczenia równań danej ściany wystarcza znajomość współrzędnych położenia trzech jej wierzchołków. Współrzędne te można znaleźć przeglądając listę krawędzi, odpowiadających danej ścianie, wraz ze wskazywanymi przez krawędzie wierzchołkami.

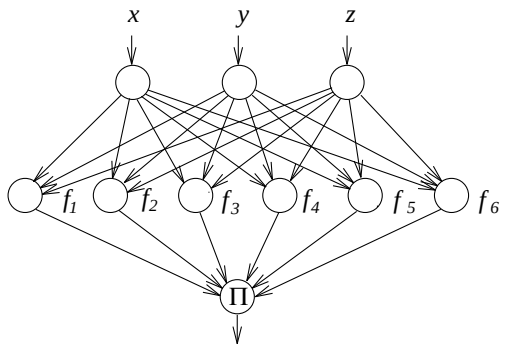
[‡]Odpowiada to skierowaniu wektorów normalnych ścian do wnętrza obiektu.

$$\text{Lista wierzchołków} \quad \{\overbrace{\{0.5,0.5,0.5\}}^{v_0}, \overbrace{\{0,0,0\}}^{v_1}, \overbrace{\{0,1,0\}}^{v_2}, \ldots, \overbrace{\{1,0,1\}}^{v_8}\}$$

$$\text{Lista krawędzi} \quad \{\overbrace{\{v_1,v_2\}}^{e_1}, \overbrace{\{v_2,v_3\}}^{e_2}, \ldots, \overbrace{\{v_4,v_8\}}^{e_{12}}\}$$

$$\text{Lista ścian} \quad \{\overbrace{\{e_1,e_2,e_3,e_4\}}^{f_1}, \overbrace{\{e_4,e_{11},e_6,e_9\}}^{f_2}, \ldots, \overbrace{\{e_5,e_6,e_7,e_8\}}^{f_6}\}$$

Rysunek 5.2: Struktura danych reprezentująca prostopadłościan.



Rysunek 5.3: Neuronowa reprezentacja prostopadłościanu.

Jak już wspomniano, przeglądając strukturę danych reprezentującą dany obiekt oraz wykorzystując współrzędne środka ciężkości, można bez trudu wyznaczyć znormalizowane równania ścian:

$$A_j x + B_j y + C_j z + D_j = 0 \tag{5.3}$$

spełniających warunek:

$$A_j v_{0x} + B_j v_{0y} + C_j v_{0z} + D_j < 0. \tag{5.4}$$

Równanie (5.3) można wprost zaimplementować za pomocą pojedynczego neuronu, mającego trzy wejścia (x, y, z) o wagach (A_j, B_j, C_j), którego wartość progowa (*bias*) równa jest D_j, a funkcją aktywacji jest standardowa funkcja perceptronu (zobacz podrozdział 4.2). Jeśli za pomocą takich neuronów zaimplementowane zostaną równania wszystkich ścian obiektu i jeśli wyjścia wszystkich neuronów skierowane zostaną do jednego, liniowego neuronu (z wagami jednostkowymi), to trzywarstwowa sieć neuronowa jaka wtedy powstanie będzie neuronową reprezentacją tego obiektu (na rysunku 5.3 pokazany jest przykład takiej sieci). Dzięki odpowiedniemu doborowi wag (zobacz warunek (5.4)) sieć taka znakomicie nadaje się do równoległego sprawdzania, czy dany punkt leży wewnątrz obiektu, który ona reprezentuje. O punkcie wewnętrznym świadczyć będzie zerowa wartość na wyjściu z sieci, o punkcie zewnętrznym – wartość niezerowa (> 0). Powyższy proces testowania można rozszerzyć na obiekty bardziej złożone, tj. na obiekty, w skład których wchodzą co najmniej dwa wielościany wypukłe. Wymaga to jednak połączenia sieci reprezentujących wielościany w jedną sieć. Cel ten osiąga się skierowując wszystkie ich wyjścia do jednego neuronu typu $\prod$ (zobacz podrozdział 4.2). Neuron ten realizować ma logiczną funkcję koniunkcji, tzn. zwracać ma 0, gdy testowany punkt leży wewnątrz któregoś z wielościanów, w przypadku przeciwnym zwracać ma wartość większą od zera, [MS94].

5.3.2 Neuronowe obliczanie odległości

Dzięki przeprowadzonej normalizacji, równania ścian, (5.3), posiadają następującą cechę: dla współrzędnych dowolnego punktu wartości przez nie zwracane równe są odległościom tego punktu (z dokładnością do znaku) od poszczególnych ścian. W poprzednim podrozdziale równania te były wykorzystane jedynie do przeprowadzania testów na kolizję punktu z obiektem. Do tego celu zaproponowana została odpowiednia sieć neuronowa. W niniejszym podrozdziale równania te posłużą do syntezy sprzężonej sieci neuronowej, obliczającej odległość pomiędzy dwoma wielościanami [§]. Niech O_1 i O_2 będą wielościanami wypukłymi o liczbie ścian równej odpowiednio M i N. Odległość $\rho(O_1, O_2)$ pomiędzy O_1 i O_2 jest równa odległości dwóch najbliższych sobie punktów $p_1 \in O_1$ i $p_2 \in O_2$. Problem znalezienia odległości $\rho(O_1, O_2)$ jest więc równoważny następującemu problemowi optymalizacji:

Problem 6 *Znaleźć punkty p_1^* i p_2^*, dla których funkcja kryterialna*

$$v(p_1, p_2) = \frac{1}{2}(p_1 - p_2)^T(p_1 - p_2) \tag{5.5}$$

osiąga swoje minimum, przy zachowaniu ograniczeń[¶]:

$$A_i p_{1x} + B_i p_{1y} + C_i p_{1z} + D_i < 0 \quad , \quad i = 1, \ldots, M, \tag{5.6}$$

$$A_j p_{2x} + B_j p_{2y} + C_j p_{2z} + D_j < 0 \quad , \quad j = 1, \ldots, N. \tag{5.7}$$

Wprowadzając odpowiednią funkcję kary powyższy problem można transformować do problemu optymalizacji bez ograniczeń, który, podobnie jak to było w przypadku odwrotnego problemu kinematyki, daje się rozwiązać za pomocy neuronowo implementowalnego algorytmu gradientowego. Niech więc dodatkowa funkcja kary, ω, zdefiniowana będzie jak niżej:

$$\omega(p_1, p_2) = \sum_{i=1}^{M} \frac{1}{2} r_i(p_1)^2 + \sum_{j=1}^{N} \frac{1}{2} r_j(p_2)^2, \tag{5.8}$$

gdzie przy $l \in \{i, j\}$

$$r_l(p) = \begin{cases} 0, & \text{jeśli } A_l p_x + B_l p_y + C_l p_z + D_l \leq 0 \\ A_l p_x + B_l p_y + C_l p_z + D_l, & \text{w przypadku przeciwnym} \end{cases} . \tag{5.9}$$

Przy ω jak wyżej, problem znalezienia odległości $\rho(O_1, O_2)$ będzie równoważny, z dokładnością do błędu zależnego od wag λ_1 i λ_2, następującemu problemowi optymalizacji bez ograniczeń:

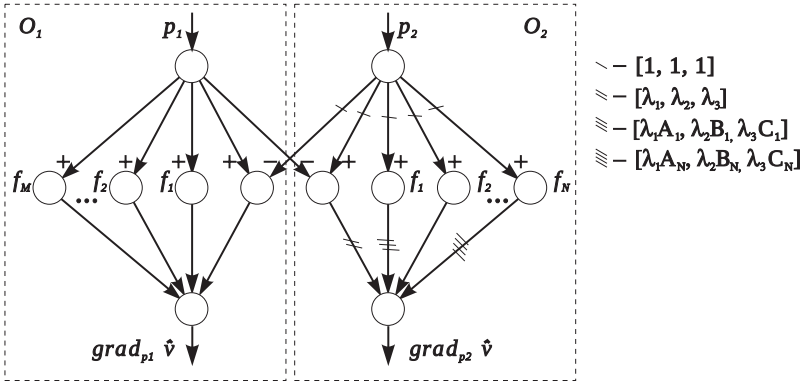
Problem 7 *Znaleźć punkty p_1^* i p_2^*, dla których*

$$\hat{v}(p_1, p_2) = \lambda_1 v(p_1, p_2) + \lambda_2 \omega(p_1, p_2) \tag{5.10}$$

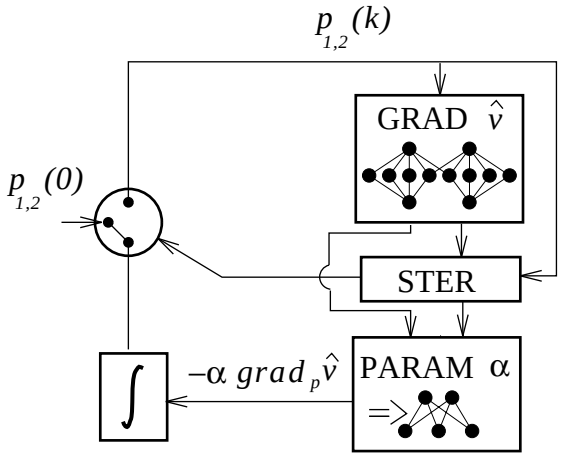
osiąga wartość minimalną.

[§]Sieć taka, po odpowiedniej modyfikacji może służyć do obliczania odległości pomiędzy obiektami dwuwymiarowymi, tj. pomiędzy wielobokami. Równania płaszczyzn zamienia się wtedy równaniami prostych, punkty $\in \mathbb{R}^3$ punktami $\in \mathbb{R}^2$.

[¶]Ograniczenia te zapewniają przynależność p_1 i p_2 odpowiednio do O_1 i O_2.



Rysunek 5.4: Schemat sieci neuronowej, obliczającej $\mathrm{grad}_{p_{1,2}} \hat{v}(p_{1,2})$



Rysunek 5.5: Neuronowa implementacja algorytmu obliczania odległości.

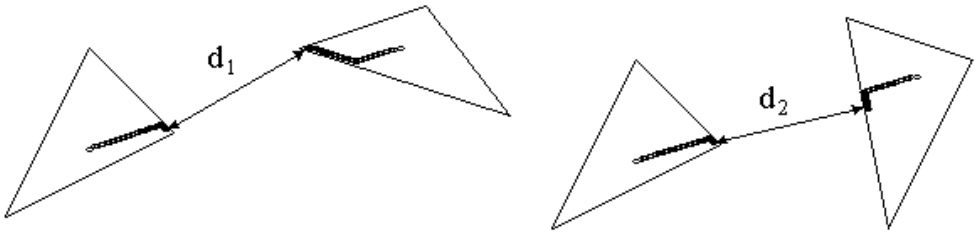
Problem 7 można rozwiązać następującym algorytmem gradientowym:

$$p_{1,2}(k+1) = p_{1,2}(k) - \alpha \ \mathrm{grad}_{p_{1,2}} \hat{v}(p_{1,2}(k)), \tag{5.11}$$

gdzie $p_{1,2} = (p_1^T, p_2^T)^T$, α – krok obliczeń, $\alpha > 0$,

$$\begin{aligned} \mathrm{grad}_{p_{1,2}} \hat{v}(p_{1,2}) &= \begin{bmatrix} \mathrm{grad}_{p_1} \hat{v}(p_{1,2}(k)) \\ \mathrm{grad}_{p_2} \hat{v}(p_{1,2}(k)) \end{bmatrix} \\ &= \begin{bmatrix} \lambda_1(p_1 - p_2) + \lambda_2 \sum_{i=1}^{M} r_i(p_1) \cdot [A_i, B_i, C_i]^T \\ -\lambda_1(p_1 - p_2) + \lambda_2 \sum_{j=1}^{N} r_j(p_2) \cdot [A_j, B_j, C_j]^T \end{bmatrix}. \end{aligned} \tag{5.12}$$

Na rysunku 5.4 pokazany jest schemat sieci neuronowej, służącej do równoległego obliczania gradientu $\mathrm{grad}_{p_{1,2}} \hat{v}(p_{1,2})$. Schemat sprzężonej sieci neuronowej, implementującej algorytm (5.11) przedstawia rysunek 5.5. Jak już zostało powiedziane, problem 7 równoważny jest problemowi znalezienia odległości $\rho(O_1, O_2)$ z dokładnością do pewnego błędu, zależnego od λ_1 i λ_2. Kryterium $\hat{v}(p_1, p_2)$ zdefiniowano bowiem w taki sposób, że jego minimum nie musi odpowiadać punktom z powierzchni obiektów O_1 i O_2, a więc może się zdarzyć, że $\rho(O_1, O_2) \neq \rho(p_1^*, p_2^*)$. Co więcej, tylko w przypadku, gdy O_1 i O_2 kolidują ze sobą,



Rysunek 5.6: Wynik neuronowego obliczania odległości na przykładzie pary trójkątów.

$\hat{v}(p_1^*, p_2^*) = 0$. W przeciwnym razie $\hat{v}(p_1^*, p_2^*) > 0$. Dlatego też odpowiedni dobór wag λ_1, λ_2 i kroku obliczeń α z właściwie określonym warunkiem stopu wpływa na dokładność otrzymanego z algorytmu (5.11) rozwiązania. Przykładowym warunkiem stopu może być spełnienie nierówności:

$$|\rho(p_1(k), p_2(k)) - \rho(p_1(k-1), p_2(k-2))| < \epsilon, \text{ gdzie } \epsilon << 1. \tag{5.13}$$

W [BDJ+94] przedstawiony został wariant neuronowego obliczania odległości, w którym funkcję kary ω zdefiniowano następująco[‖]:

$$\omega(p_1, p_2) = \sum_{i=1}^{M} \frac{\varphi_i(p_1)^2}{1 - \varphi_i(p_1)} + \sum_{j=1}^{M} \frac{\varphi_j(p_2)^2}{1 - \varphi_j(p_2)}, \text{ gdzie } \varphi_l(p) = \frac{1}{1 + e^{-\beta r_l(p)}}, \tag{5.14}$$

zaś parametr α obliczany był wg wzoru:

$$\alpha(p_{1,2}) = \frac{\text{grad}_{p_{1,2}} \hat{v}^T \text{ grad}_{p_{1,2}} \hat{v}}{\text{grad}_{p_{1,2}} \hat{v}^T H(\hat{v}) \text{ grad}_{p_{1,2}} \hat{v}}, \text{ gdzie } H(\hat{v}) = \left[\frac{\partial^2 \hat{v}}{\partial p_{1,2}^2} \right]. \tag{5.15}$$

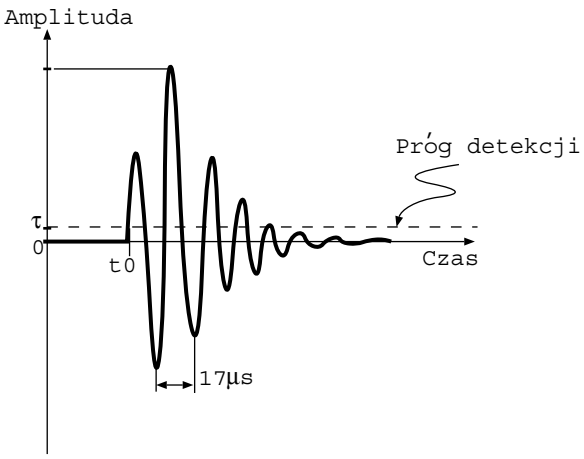
Przykład: Przedstawiony w powyższych rozważaniach algorytm obliczania odległości, (5.11), bazował na neuronowym modelu sceny, w którym każdy obiekt reprezentowany był przez zaszyte w wagach sieci równania jego ścian. Do celów demonstracyjnych własności tego algorytmu wybrano dwuwymiarowy model sceny, z trójkątami jako obiektami.

Na rysunku 5.6 pokazany jest wynik neuronowego obliczania odległości. Widoczne na nim sekwencje położeń punktów, zaczynające się we wnętrzu trójkątów, a kończące się przy ich krawędziach, stanowią ślad, jaki zostawiły po sobie punkty p_1 i p_2 (zobacz algorytm (5.11) oraz rysunek 5.5) po kolejnych iteracjach. Odległości wynikowe, d_1 i d_2, zdefiniowane są przez końcowe położenia tych punktów. Jak widać, iteracje nie kończą się z chwilą dotarcia przez któryś z punktów do krawędzi obiektu, lecz są kontynuowane aż do osiągnięcia rozwiązania.

5.4 Symulacja systemu sensorycznego w wirtualnym otoczeniu robota

Występujący w tytule niniejszego podrozdziału termin „sensory" należy rozumieć jako skróconą formę od „sensory akustyczne". Właśnie tego typu sensory użyte zostały w

[‖]Funkcja sigmoidalna $\varphi()$ jest jedną ze standardowych funkcji aktywacji nieliniowych neuronów.



Rysunek 5.7: Kształt fali typowego echa.

późniejszych eksperymentach, przeprowadzanych na rzeczywistym manipulatorze, i dlatego właśnie tylko o takich sensorach będzie mowa w niniejszym podrozdziale.

5.4.1 Własności fizyczne sensorów

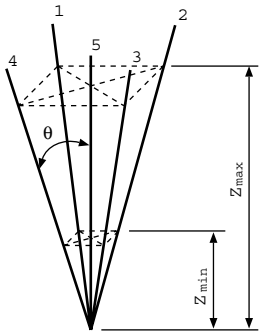
Większość konwencjonalnych akustycznych systemów sensorycznych składa się z pojedynczych przetworników, które mogą działać zarówno jako nadajniki jak i odbiorniki sygnałów. Zazwyczaj nadajnik po wyemitowaniu wiązki akustycznej staje się odbiornikiem, nasłuchującym echa powstającego po zderzeniu się wiązki z przeszkodą, [KV91]. Aby usprawnić systemy sensoryczne (wyeliminować szumy, zakłócenia, zjawisko podwójnego odbicia, itd.) budowane są systemy składające się z dwóch nadajników i dwóch odbiorników, [KK95] lub jednego nadajnika i trzech odbiorników (nadajnik jest jednym z odbiorników), [PAvC93].

Kształt fali typowego echa obserwowanego na wyjściu odbiornika ma oscylacyjny charakter, o malejących amplitudach (zobacz rysunek 5.7). Poziom sygnału oznaczony na rysunku przez z_0 jest progiem detekcji. Wartość z_0 dobiera się tak, aby wyeliminowane zostały ewentualne błędy, powstałe wskutek zaszumienia sygnału.

Pomiar odległości przy pomocy omawianych sensorów polega na pomiarze czasu, jaki upływa pomiędzy momentem nadania sygnału akustycznego, a chwilą odebrania echa (przyjmuje się, że odebranie echa następuje, gdy jego amplituda pierwszy raz przekroczy wartość progową,(detekcja amplitudowa)). Czas ten, oznaczony przez t_0, wstawia się następnie do równania:

$$z_0 = ct_0/2, \tag{5.16}$$

gdzie z_0 - wyliczana odległość, c - prędkość rozchodzenia się dźwięku w powietrzu. Z wielkości przedziału czasu, jaki potrzebny jest na przełączenie przetwornika z trybu nadawania na tryb odbierania wynika minimalny zasięg działania danego sensora. Zasięg maksymalny zazwyczaj jest zdeterminowany zdolnością tłumienia ośrodka, w którym rozchodzi się dźwięk i odpowiada odległości, przy której maksymalna wartość amplitudy powracającego echa, A, spada poniżej progu detekcji. W [KV91] pokazano, że wartość amplitudy A jest funkcją kąta, pod jakim względem nadajnika znajduje się przeszkoda, oraz odległości do tej przeszkody. Może się więc zdarzyć, że niektóre przeszkody nie będą



Rysunek 5.8: Model pojedynczego sensora.

dostrzegane, mimo iż znajdują się w „zasięgu widzenia" sensorów. Oczywiście, istnieją jeszcze inne źródła błędów, jak np.: źle dobrana częstotliwość nadawania, pochłanianie i rozrzut sygnałów, zjawisko wielokrotnych odbić, itd.

5.4.2 Modele sensorów

Fizyczne własności sensorów tylko w części dają się opisać za pomocą formuł matematycznych. Dlatego też trudno jest zbudować wystarczająco dokładny ich model, który mógłby zostać użyty do badań symulacyjnych. Co więcej, z każdą próbą budowy takiego modelu wiążą się trudności wynikające z obranego sposobu reprezentacji przeszkód w scenie (jak problem liczenia odległości, testowania kolizji itp.). Niebagatelną też sprawą pozostaje sposób dyskretyzacji wiązki wysyłanej przez pojedynczy sensor.

Na rysunku 5.8 pokazany jest trójwymiarowy model sensora, w którym starano się uwzględnić wszystkie wyżej wymienione okoliczności. Pięć widocznych na nim półprostych (krawędzie i oś symetrii ostrosłupa) to dyskretna wersja rzeczywistej wiązki wysyłanej przez sensor (stożek parametryzowany przez kąt θ (*beam angle*)). Półproste te wyznaczają kierunki, wzdłuż których odbywać się będzie obliczanie odległości do objętych wiązką przeszkód („kierunki detekcji"). Z_{min} i Z_{max} odpowiadają minimalnemu i maksymalnemu zasięgowi działania.

Przyjęcie powyższego modelu wymusza w czasie symulacji przeprowadzanie czteroetapowych obliczeń:

1. obliczanie odległości do przeszkody wzdłuż zadanego kierunku (dla każdego z kierunków detekcji);

2. obliczanie kąta padania promienia wiązki na powierzchnię przeszkody (dla każdego z kierunków detekcji);

3. generowanie szumów (dla każdej obliczonej odległości);

4. uśrednianie wyników.

W etapie pierwszym obliczanie odległości polega na znalezieniu punktów przecięcia danej półprostej ze ścianami rozpatrywanej przeszkody (korzysta się tu z metod modelowania, liczenia odległości i testowania kolizji, przedstawionych w podrozdziale 5.1). Spośród znalezionych punktów przecięcia wybiera się następnie jeden, leżący najbliżej punktu

zaczepienia półprostej. Jeśli odległość do niego mieści się w przedziale $[Z_{min}, Z_{max}]$, następuje przejście do etapu drugiego. W przeciwnym wypadku (lub gdy nie znaleziono punktów przecięcia), przyjmuje się, że przeszkoda wzdłuż zadanego kierunku nie jest widoczna.

W etapie drugim sprawdzane jest, czy kąt pomiędzy półprostą a przecinaną przez nią ścianą (kąt padania promienia) jest wystarczająco duży. Jeśli mieści się on w granicach $[\beta, \pi/2]$ (β – minimalny kąt padania), to następuje przejście do etapu trzeciego. W przeciwnym wypadku przyjmuje się, że przeszkoda wzdłuż zadanego kierunku nie jest widoczna.

W etapie trzecim na obliczoną odległość nakłada się zakłócenia, po czym następuje przejście do etapu czwartego.

W etapie czwartym wyniki zasymulowanych pomiarów (w postaci pięciu odczytów odpowiadających pięciu kierunkom) podlegają uśrednianiu, zgodnie z następującymi regułami: 1) gdy przeszkoda zauważona została na co najwyżej jednym z kierunków, wyjściem z symulatora jest wartość Z_{max} (tj. „nie widać przeszkód"); 2) gdy przeszkoda została zauważona na co najmniej dwóch kierunkach, wyjściem z symulatora jest wartość otrzymana po odpowiednim zsumowaniu, przeskalowaniu i zaszumieniu poszczególnych odległości na kierunkach.

Na rysunku 5.9 przedstawione są wyniki symulacji, w których na przykładzie pierścienia ośmiu sensorów badano własności opisanego wyżej modelu**. Jak widać, odległości otrzymane z poszczególnych sensorów zależały od kształtu, położenia i orientacji w jakiej znajdowała się względem nich umieszczona na scenie przeszkoda.††. Część a) rysunku 5.9 ilustruje sytuację, w której trzy sensory zaobserwowały przeszkodę. Różnica między otrzymaną odległością a odległością rzeczywistą wynika z faktu zaszumiania i uśredniania wyników w trzecim i czwartym etapie obliczeń. Część b) tego rysunku jest przykładem braku detekcji, pomimo obecności przeszkody w zasięgu działania sensorów.

Bazując na trójwymiarowym modelu sensora w prosty sposób można dokonać implementacji jego dwuwymiarowej wersji. Podobnie jak to było w przypadku trójwymiarowym, w dwuwymiarowym modelu wyróżnionych zostanie kilka kierunków detekcji. Analogicznie wykonywane też będą kolejne etapy obliczeń.

Istnieją jednak okoliczności, w których nie występuje konieczność wprowadzania aż tak złożonych modeli sensorów. Do testowania algorytmów obliczania kinematyki odwrotnej z omijaniem przeszkód wystarcza na przykład zwykłe, proste obliczanie odległości. Przy testach tych bowiem nie tyle chodzi o sprawdzenie zgodności modelu sensorów z ich fizycznymi własnościami, co raczej o wypracowanie metod agregacji (fuzji) ich sygnałów w odniesieniu do zastosowania w rozwiązywaniu odwrotnego problemu kinematyki.

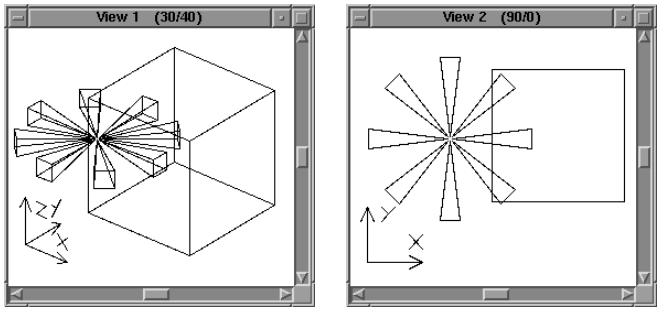
5.4.3 Metody fuzji sygnałów sensorycznych

Dokładnie analizując treść rozdziału 2 można przekonać się, że omawiane w nim metody obliczania kinematyki odwrotnej wykorzystują dwojakiego rodzaju informacje o otoczeniu manipulatora, a mianowicie: 1) informację o odległościach przegubów manipulatora od przeszkód wraz z kierunkiem do nich; 2) informację o położeniu punktów wirtual-

**Takie parametry pierścienia jak: jego średnica, ilość sensorów, wartość kąta θ oraz zakresu $[Z_{min}, Z_{max}]$ mogły być ustawiane dowolnie.

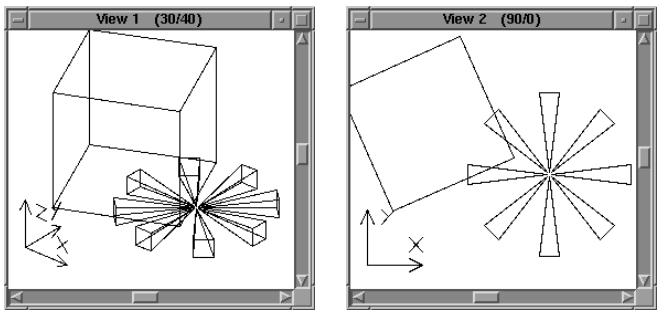
††Symulator umożliwiał umieszczenie na scenie wielu przeszkód o różnych kształtach. Dla prostoty na rysunku pokazano tylko przypadek pojedynczego prostopadłościanu.

a)



Nr	Odległość z symulatora	Odległość rzeczywista
1	1.20e+03	1.20e+03
2	4.20e+02	4.40e+02
3	3.24e+02	3.01e+02
4	4.20e+03	4.40e+03
5	1.20e+03	1.20e+03
6	1.20e+03	1.20e+03
7	1.20e+03	1.20e+03
8	1.20e+03	1.20e+03

b)



Nr	Odległość z symulatora	Odległość rzeczywista
1	1.20e+03	1.20e+03
2	1.20e+02	1.20e+02
3	1.20e+02	1.20e+02
4	1.20e+03	1.20e+03
5	1.20e+03	1.20e+03
6	1.20e+03	1.20e+03
7	1.20e+03	4.59e+03
8	1.20e+03	4.32e+03

Rysunek 5.9: Wyniki symulacji pracy pierścienia ośmiu sonarów: a) - sonary nr 2, 3 i 4 zaobserwowały przeszkodę; b) - żaden z sonarów nie zaobserwował przeszkody.

nych, które odpowiadają bezkolizyjnym położeniom przegubów. Stosunkowo prosto te równoważne sobie informacje daje się uzyskać ze zbudowanego wcześniej modelu sceny (zobacz metody reprezentacji sceny, metody liczenia odległości na początku bieżącego rozdziału). Jeśli jednak do dyspozycji nie ma żadnego modelu, a cała dostępna wiedza o scenie pochodzi z sensorów, zamontowanych bezpośrednio na ramionach manipulatora, wtedy powstaje problem fuzji (agregacji) sygnałów sensorycznych. Temu właśnie zagadnieniu poświęcony jest niniejszy podrozdział.

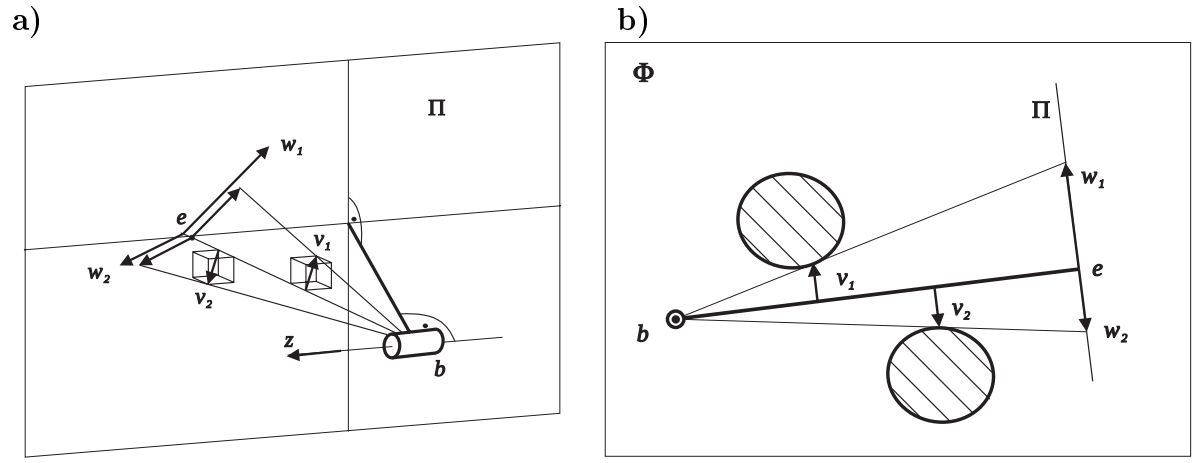
Metoda projekcji. Metodę tą można krótko scharakteryzować następującym zdaniem: jeśli dane ramię manipulatora uzbrojone jest w kilka sensorów, to pseudokierunek i pseudoodległość jego końca do przeszkody otrzyma się, dokonując projekcji wektorów, odpowiadających sygnałom sensorycznym, na płaszczyznę równoległą do osi przegubu i zaczepioną na końcu ramienia, wybierając następnie z rzutów tych wektor o minimalnym module. Moduł wybranego wektora odpowiadać będzie odległości od pseudoprzeszkody, natomiast on sam wskazywać będzie na przeszkodę, gdy jego początek zostanie zaczepieniu na końcu ramienia. Aby metodę tą można było zastosować, spełnione być muszą następujące warunki:

- manipulator jest reprezentowany przez model szkieletowy,
- sensory zamontowane są bezpośrednio na ramionach manipulatora,
- znane jest położenie i orientacja sensorów jak również położenie i orientacja ramion i przegubów manipulatora,
- wyjście o_i z sensora i jest równe o_i^{max}, jeśli w zakresie jego działania $[o_i^{min}, o_i^{max}]$ nie ma żadnych przeszkód, w przeciwnym wypadku o_i jest odległością do najbliższej przeszkody (na kierunku obserwacji),
- agregacji sygnałów dokonuje się tylko dla sensorów należących do tego samego ramienia (czyli dla każdego z ramion osobno),
- każde z ramion ma wyróżniony początek (b, gdzie umieszczony jest przegub) i koniec (e, gdzie umieszczony jest przegub następnego ramienia).

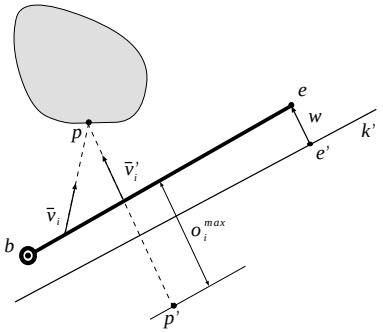
Na rysunku 5.10 a) pokazano graficzną interpretację metody projekcji. Symbolem Π oznaczona jest płaszczyzna projekcji. Przez v_i oznaczone są wektory jednostkowe, reprezentujące sensory. Przez w_i oznaczone są wektory kandydujące do miana pseudokierunku i pseudoodległości, otrzymane w wyniku projekcji i przeskalowania wartością o_i wektorów v_i. Wektorem wynikowym jest ten z wektorów w_i, który ma najmniejszy moduł.

Metoda projekcji bazuje na spostrzeżeniu, że przy obrocie ramienia wokół osi przegubu większą drogę zakreśla jego koniec niż jego środek. Dlatego też wektory v_i bliższe przegubowi „ważniejsze są" od tych, umieszczonych przy końcu ramienia. A poprzez projekcję właśnie można wpływać na „poziom ważności" poszczególnych wektorów.

Na rysunku 5.10 b) pokazany jest przykład ramienia manipulatora w rzucie na płaszczyznę prostopadłą do osi obrotu przegubu. Pomimo, że odległości ramienia od przeszkód są jednakowe ($o_1 = o_2$, $v_1 \parallel v_2 \parallel \Phi$), pseudokierunek i pseudoodległość definiuje wektor w_2.



Rysunek 5.10: Metoda projekcji a) geometryczna interpretacja b) przykład zastosowania.



Rysunek 5.11: Geometryczna interpretacja metody aproksymacji.

Metoda aproksymacji. Aby można było zastosować metodę aproksymacji, spełnione być muszą te same warunki co w metodzie projekcji. Dzięki temu założeniu, każdemu sensorowi przypisać można wyrażony w globalnym układzie współrzędnych punkt p_i, reprezentujący widzianą przez niego przeszkodę (współrzędne punktów p_i zdeterminowane są przez znaną pozycję ramienia, wektorów v_i oraz wartości o_i). Istota metody aproksymacji polega na utworzeniu zastawu par bratnich punktów $\{p_i, p_i'\}$, gdzie współrzędne p_i' oblicza się ze wzoru:

$$p_i' = p_i - 2 \cdot o_i^{max} \cdot v_i' \tag{5.17}$$

(v_i' - wektor jednostkowy, prostopadły do ramienia, wskazujący na p_i) i wyznaczeniu wektora pseudokierunku i pseudoodległości do przeszkody, w, na bazie wyników liniowej aproksymacji zbioru $\{\{p_i, p_i'\}, b, e\}$. Moduł wektora w odpowiadać będzie odległości od pseudoprzeszkody, natomiast on sam wskazywać na przeszkodę będzie, gdy jego początek zostanie zaczepieniu na końcu ramienia.

Operację aproksymacji można interpretować jako poszukiwanie bezkolizyjnego położenia ramienia w pobliżu rdzenia wolnej od przeszkód przestrzeni. Jeśli więc k' jest prostą aproksymującą zbiór punktów $\{\{p_i, p_i'\}, b, e\}$ (zobacz rysunek 5.11), to wektor w wyraża

się wzorem:

$$w = e - e', \tag{5.18}$$

gdzie e' jest rzutem prostopadłym punktu e na prostą k'.

5.5 Planowanie ścieżki w neuronowym modelu otoczenia robota

W podrozdziale 5.3 poruszone zostały problemy związane z modelowaniem sceny roboczej manipulatora. Była w nim mowa o neuronowej reprezentacji obiektów, o metodach detekcji kolizji, o obliczaniu odległości. Sprzęgnięcie przedstawionych tam rozważań z wynikami z rozdziału 4* , poprzez problem planowania ruchów, a w szczególności poprzez problem planowania ścieżki efektora, stanowi główną treść niniejszego podrozdziału.

W literaturze z problemem planowania ruchów, rozumianym jako zadanie znalezienia sekwencji konfiguracji manipulatora, zapewniających przeprowadzenie jego efektora z położenia początkowego do położenia zadanego, z omijaniem obecnych w scenie przeszkód, można się zetknąć przy okazji omawiania metod sztucznej inteligencji, programowania logicznego, systemów zdarzeń dyskretnych, [LP86, KV88, KH89, CL89, Jac89, Lat91a, Lat91b, LT91, Sat93]. Wyróżnia się tam dwa typy takiego planowania, mianowicie: 1) planowanie globalne, gdzie spośród wielu możliwych rozwiązań wybierane jest jedno, najlepsze w myśl przyjętego kryterium (do czego potrzebna jest znajomość całej sceny roboczej manipulatora); 2) planowanie lokalne, gdzie znajdowane są kolejne fragmenty rozwiązania (do czego wystarcza lokalna znajomość otoczenia manipulatora).

Z planowaniem globalnym wiąże się zazwyczaj konieczność transformowania przeszkód z przestrzeni roboczej do przestrzeni konfiguracyjnej manipulatora, [Lat91b]. Ze względu na duże nakłady obliczeniowe związane z użyciem geometrycznych technik poszukiwania rozwiązania, planowanie globalne nie może odbywać się na bieżąco.

Pewnym przeciwieństwem do planowania globalnego jest planowanie lokalne. Dostępna w nim informacja o przeszkodach dotyczy tylko lokalnego otoczenia manipulatora (i bywa uzyskiwana z sensorów zamontowanych na manipulatorze). Jej objętość jest na tyle mała, że planowanie ruchów, ograniczonych do „widzianego otoczenia", może już odbywać się na bieżąco. Na dużą szybkość przeprowadzanych obliczeń wpływa również fakt, że przy planowaniu lokalnym przeszkody reprezentowane są bezpośrednio w przestrzeni roboczej manipulatora. Niestety, zysk w postaci szybkości obliczeń może być zniwelowany przez straty, wynikające z natury lokalnego planowania (planowanie lokalne nie zapewnia zrealizowania każdego zadania).

Patrząc z perspektywy zastosowań osiągniętych dotychczas wyników, można zaproponować jeszcze jeden typ planowania ruchów, wiążący oba wyżej wymienione typy planowań. W planowaniu tym wyróżnia się dwie fazy: 1) fazę planowania bezkolizyjnej ścieżki dla efektora manipulatora, utożsamianego z punktem w przestrzeni trójwymiarowej (co odpowiada planowaniu globalnemu); 2) fazę planowania bezkolizyjnego ruchu manipulatora, podążającego efektorem wzdłuż zaplanowanej ścieżki (co odpowiada planowaniu lokalnemu,[BGC94]). Faza pierwsza pokrewna jest problemowi planowania ruchów dla robotów mobilnych, [Kod87, Sat93, CTL94]. Z uwagi na wprowadzony na początku tego rozdziału model otoczenia, spośród wielu opracowanych już w tej dziedzinie metod na

*W rozdziale tym przedstawiono neuronowy sposób obliczania kinematyki odwrotnej.

szczególną uwagę zasługuje metoda pól sztucznego potencjału. W podrozdziale 5.5.1 zastanie omówiony w szczegółach sposób jej przystosowania do realizacji fazy pierwszej. Faza druga w głównej mierze została już omówiona w rozdziale 4, gdzie przedstawiono metodę obliczania kinematyki odwrotnej. Kwestia sposobu pozyskiwania, generalizacji i wykorzystania informacji o lokalnym (zmieniającym się) otoczeniu manipulatora. wyjaśniona została w podrozdziale (5.4.3), gdzie omówione zostały metody fuzji sygnałów sensorycznych (z sensorów akustycznych). Wzajemne relacje pomiędzy obiema fazami planowań staną się bardziej zrozumiałe po lekturze rozdziału 6.

5.5.1 Neuronowa implementacja metody pól sztucznego potencjału

Planowanie bezkolizyjnej ścieżki dla punktu p w trójwymiarowej, statycznej (znanej) przestrzeni roboczej E_0 polega na znalezieniu najkrótszego, bezkolizyjnego przejścia pomiędzy położeniem początkowym tego punktu, p_S, a położeniem docelowym, p_F. Dzięki wprowadzeniu pól sztucznego potencjału, zadanie to sprowadzić można do jednego z dwóch następujących zadań minimalizacji: a) zadania znalezienia ścieżki leżącej wzdłuż największego spadku energii pola; b) zadanie znalezienia ścieżki, minimalizującej energię związaną z jej długością i karą za zbliżanie się do przeszkód. Standardowo, pole sztucznego potencjału w zadaniu a) definiuje się jako funkcję $U : E_0 \rightarrow \mathbb{R}^+$, która każdemu punktowi z przestrzeni E_0 przypisuje wartość energii potencjalnej, odpowiadającej jego położeniu. Funkcja ta stanowi sumę dwóch funkcji podstawowych, mianowicie potencjału przyciągającego, U_a, i potencjału odpychającego, U_r,

$$U(p) = U_a(p) + U_r(p). \tag{5.19}$$

Potencjał przyciągający definiuje się tak, aby w miejscu punktu decelowego utworzyła się „energetyczna dolina". Definicja natomiast potencjału odpychającego zapewnić ma utworzenie się w obszarze zajmowanym przez przeszkody „energetycznych wzgórz". Posługując się taką fizyczną interpretacją, poszukiwana ścieżka będzie drogą, jaką zakreśli tocząca się wśród tych wzgórz bezwymiarowa kulka o masie jednostkowej. Drogę kulki wyznacza się przez rozwiązanie równania ruchu (poprzez np. całkowanie metodą Runge-Kutta):

$$\ddot{p} = F = F_a + F_r, \tag{5.20}$$

gdzie $F_a = -\frac{\partial U_a}{\partial p}$, $F_r = -\frac{\partial U_r}{\partial p}$ - odpowiednio siła przyciągająca i siła odpychająca; lub przez zastosowanie algorytmu gradientowego:

$$\dot{p} = -\alpha \frac{\partial U}{\partial p}. \tag{5.21}$$

Zazwyczaj potencjał przyciągający ma postać kwadratową:

$$U_a(p) = \frac{1}{2}\beta d_f^2(p) \tag{5.22}$$

lub jest złożeniem funkcji kwadratowej i liniowej:

$$U_a(p) = \begin{cases} \frac{1}{2}\beta d_f^2(p), & \text{gdy} \quad d_f < s \\ 2\beta s d_f(p) - \beta s^2, & \text{gdy} \quad d_f \leq s \end{cases}, \tag{5.23}$$

gdzie $\beta > 0$ - waga, s - parametr przełączania, $d_f^2(p) = (p-p_f)^T(p-p_f)$, $d_f(p) = (p-p_f)$.

Większą różnorodnością definicji charakteryzuje się potencjał odpychający. Dla prostej sceny wyraża się go jako sumę następujących potencjałów, [Kha86, KV88]:

$$U_{ri}(p) = \begin{cases} \frac{1}{2}\beta(\frac{1}{d_{oi}(p)} - \frac{1}{\hat{d}_o}), & \text{gdy} \quad d_{oi}(p) < \hat{d}_o \\ 0, & \text{gdy} \quad d_{oi}(p) < \hat{d}_o \end{cases}, \tag{5.24}$$

gdzie $d_{oi}(p)$ - odległość punktu p od i-tej przeszkody, $\hat{d}_o$ - margines bezpieczeństwa. Wraz ze wzrostem ilości przeszkód w scenie rośnie niebezpieczeństwo pojawienia się w polu potencjałów minimów lokalnych. Aby zaradzić tej sytuacji wprowadzony został następujący potencjał odpychający, [KV88]:

$$U_{ri}(K(p)) = \begin{cases} \frac{A}{K_i(p)} \exp(-\alpha K_i(p)), & \text{dla} \quad K_i(p) > 1 \\ A \exp(-\alpha K_i^{1+\frac{1}{\alpha}}(p)), & \text{dla} \quad 0 \leq K_i(p) < 1 \end{cases}, \tag{5.25}$$

gdzie α - parametr odpowiedzialny za szybkość wzrostu potencjału, A - parametr skalujący, $K(p)$ - pseudoodległość punktu p od przeszkody i. Inną próbę skonstruowania pola potencjałów bez minimów lokalnych podjął Sato, [Sat93], proponując do tego celu użycie pól potencjałów Laplace'a.

Jak można zauważyć, najbardziej kosztowną operacją w planowaniu bezkolizyjnej ścieżki za pomocą równań (5.20) i (5.21) jest, występujące przy obliczaniu gradientu potencjału odpychającego, obliczanie odległości. Aby obniżyć koszty planowania, wystarczy w miejscu tym zastosować neuronowo implementowalne algorytmy z podrozdziału 5.2 (zysk osiąga się przez wprowadzenie równoległych obliczeń). Efektywniejszym jednak sposobem wykorzystania wyników z podrozdziału 5.2 jest ich zastosowanie do rozwiązania zadania b).

Dla przypomnienia, zadanie b) polega na znalezieniu ścieżki pomiędzy punktem startowym a punktem docelowym, o minimalnej energii, związanej z jej długością i karą za zbliżanie się do przeszkód (statycznych), [LB91]. Zadaniu temu można nadać następującą fizyczną interpretację: szukana bezkolizyjna ścieżka jest kształtem, jaki przyjmie rozciągnięta w polu potencjałów elastyczna nitka, uwiązana końcami w punkcie startowym i docelowym.

Energię elastycznej nitki zdefiniować można w następujący sposób. Jeśli na nitce wyróżnionych zostanie $\mathcal{L}+1$ kolejnych punktów p_i (p_0 i $p_\mathcal{L}$ to odpowiednio, punkt startowy i punkt docelowy), wtedy energia sprężystości nitki [†] wyrażać się będzie równaniem:

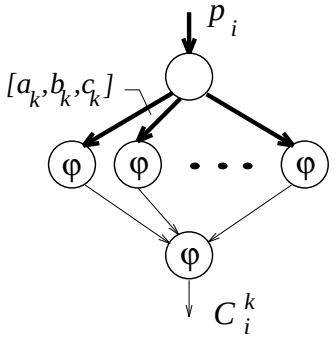
$$E_{sp} = \frac{1}{2}\sum_{i=1}^{\mathcal{L}} L_i^2 = \frac{1}{2}\sum_{i=1}^{\mathcal{L}}(p_i - p_{i-1})^T(p_i - p_{i-1}). \tag{5.26}$$

Energia związana z karą za zbliżanie się do przeszkód dana zaś będzie wzorem:

$$E_{ka} = \sum_{i=1}^{\mathcal{L}}\sum_{k=1}^{\mathcal{K}} C_i^k, \tag{5.27}$$

gdzie $\mathcal{K}$ - liczba przeszkód, C_i^k - kara za zbliżanie się punktu p_i (należącego do nitki) do przeszkody k. Natomiast energia całkowita nitki, E, będzie następującą ważoną sumą

[†] W rzeczywistości jest to energia sprężystości linii łamanej, składającej się z odcinków o długości L_i.



Rysunek 5.12: Sieć neuronowa obliczająca C_i^k.

energii sprężystości i energii związanej z karą za zbliżanie się do przeszkód:

$$E = w_{sp}E_{sp} + w_{ka}E_{ka}, \tag{5.28}$$

gdzie w_{sp}, w_{ka} są współczynnikami wagowymi.

Występującą we wzorze 5.27 karę C_i^k obliczyć można, jak w zadaniu a), zgodnie z formułami na potencjał odpychający, (5.24), (5.25). Aby jednak ominąć problem obliczania odległości, karę C_i^k otrzymać można bezpośrednio z przedstawionej na rysunku 5.12 sieci neuronowej. Sieć ta różni się od sieci przedstawionej na rysunku 5.3 tym, że: 1) neurony warstwy ukrytej i wyjściowej są tego samego typu, o sigmoidalnej funkcji aktywacji $\varphi(\cdot)$; 2) wszystkie wagi i wartości progowe (*bias'y*) warstwy ukrytej mają zmieniony znak (równania ścian są tak określone, że zwracają wartości dodatnie dla punktów należących do wnętrza danego obiektu); 3) wartość progowa (*bias*) neuronu warstwy wyjściowej równa jest liczbie neuronów warstwy ukrytej (liczbie ścian przeszkody k), M^k, pomniejszonej o 1/2. Analityczną postać funkcji kary przedstawia wzór:

$$C_i^k = \varphi(\sum_{j=1}^{M^k} \varphi(r_j^k) + M^k - 1/2), \tag{5.29}$$

gdzie $r_j^k(p_i) = A_j^k p_{ix} + B_j^k p_{iy} + C_j^k p_{iz} + D_j^k$ otrzymuje się z równania j-tej ściany k-tej przeszkody.

Mając zdefiniowaną energię całkowitą, zadanie znalezienia bezkolizyjnej ścieżki można utożsamić z zadaniem znalezienia takiego położenia punktów p_i, które tą energię minimalizuje. Do tego celu posłużyć się można, podobnie jak miało to miejsce przy liczeniu odwrotnej kinematyki, następującym algorytmem gradientowym:

$$\dot{p}_i = -\eta \left[\frac{\partial E}{\partial p_i}\right] \quad , \quad \eta > 0, i = 1, \ldots, \mathcal{L} - 1 \tag{5.30}$$

z warunkiem stopu:

$$|\dot{p}_i| < \epsilon \quad , \quad \epsilon \ll 1. \tag{5.31}$$

Zbieżność tego algorytmu wykazać można, sprawdzając znak $\dot{E}$ wzdłuż trajektorii (5.30). Ponieważ punkty p_0 i p_N są punktami statycznymi, pochodna energii w czasie ma postać:

$$\dot{E} = \sum_{i=1}^{\mathcal{L}-1} \left[\frac{\partial E}{\partial p_i}\right]^T \dot{p}_i = \sum_{i=1}^{\mathcal{L}-1} [w_{sp}\frac{\partial E_{sp}}{\partial p_i} + w_{ka}\frac{\partial E_{ka}}{\partial p_i}]^T \dot{p}_i, \tag{5.32}$$

gdzie

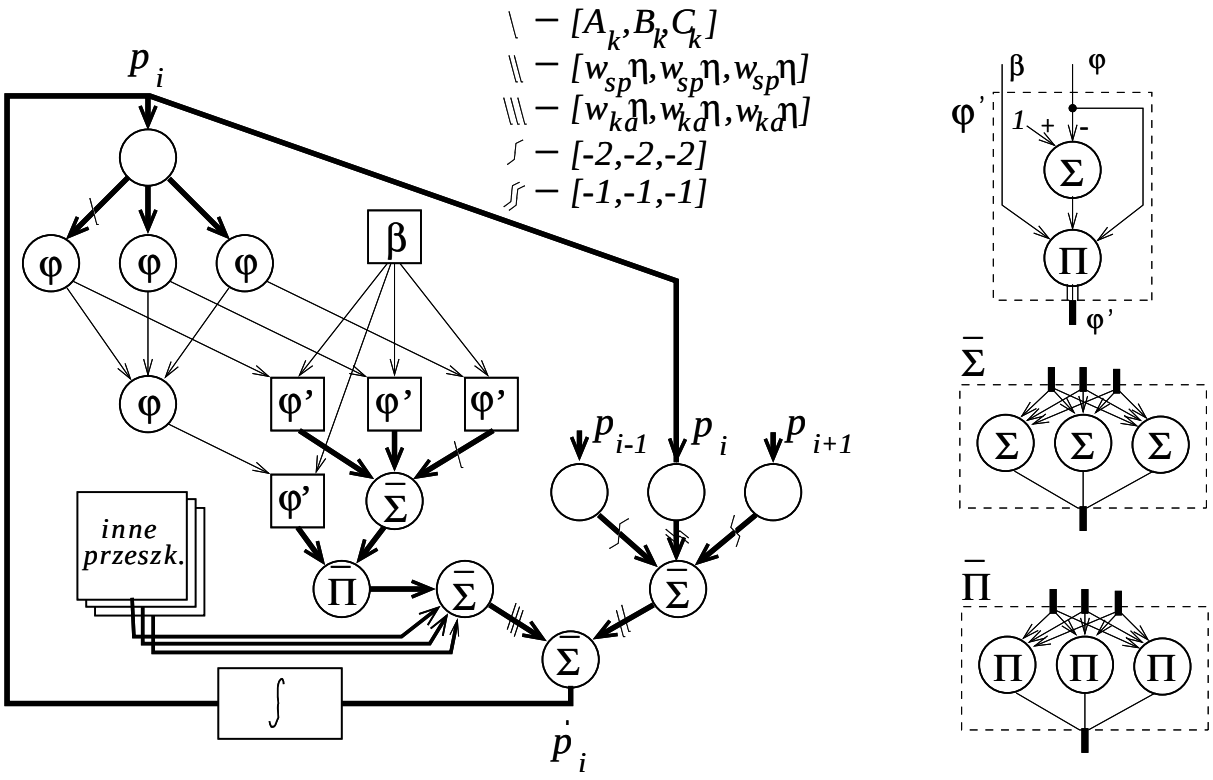
$$\begin{aligned}
\frac{\partial E_{sp}}{\partial p_i} &= 2p_i - p_{i-1} - p_{i+1}, \\
\frac{\partial E_{ka}}{\partial p_i} &= \sum_{k=1}^{\mathcal{K}} \frac{\partial C_i^k}{\partial p_i}, \\
\frac{\partial C_i^k}{\partial p_i} &= \varphi'(\sum_{j=1}^{M^k} \varphi(r_j^k(p_i)) + M^k - 1/2) \cdot (\sum_{j=1}^{M^k} \varphi'(r_j^k(p_i))) \cdot \frac{\partial r_j^k}{\partial p_i}, \\
\frac{\partial r_j^k}{\partial p_i} &= (A_j^k, B_j^k, C_j^k)^T, \\
\varphi'(\cdot) &= -\beta\varphi(\cdot)(1 - \varphi(\cdot)).
\end{aligned}$$

Jeśli $\dot{p}_i$ wybrane zostanie zgodnie z równaniem (5.30), to pochodna energii w czasie wzdłuż trajektorii tego systemu będzie ujemnie określona

$$\dot{E} = -\eta \sum_{i=1}^{\mathcal{L}-1} \dot{p}_i^T \dot{p}_i \leq 0 \tag{5.33}$$

wszędzie, z wyjątkiem punktów równowagi, gdzie $\dot{E} = 0$ ($\dot{E} = 0 \Leftrightarrow \dot{p}_i = 0, \forall i$). Fakt ten dowodzi, że zastosowanie algorytmu (5.30) zapewnia minimalizację energii E.

Aby ominąć problem minimów lokalnych (tzn. aby z algorytmu (5.30) uzyskać rzeczywiście najkrótszą, bezkolizyjną ścieżkę), obliczanie kolejnych położeń punktów p_i odbywać się musi równolegle z procesem wyżarzania (*annealing process*), [LB91]. Wyżarzanie to polega na zwiększaniu w miarę upływu czasu parametru β, występującego w wyrażeniu na funkcję sigmoidalną. Przy zmianie β (np. zgodnie z zależnością: $\beta = \beta_0 \log(1 + t)$ lub $\beta = \beta_0(1 + t)$, gdzie: β_0 - wartość początkowa, t - czas) zmienia się zasięg oddziaływania funkcji kary C_i^k. Im większe jest β, tym mniejszy jest zasięg oddziaływania. Małe β

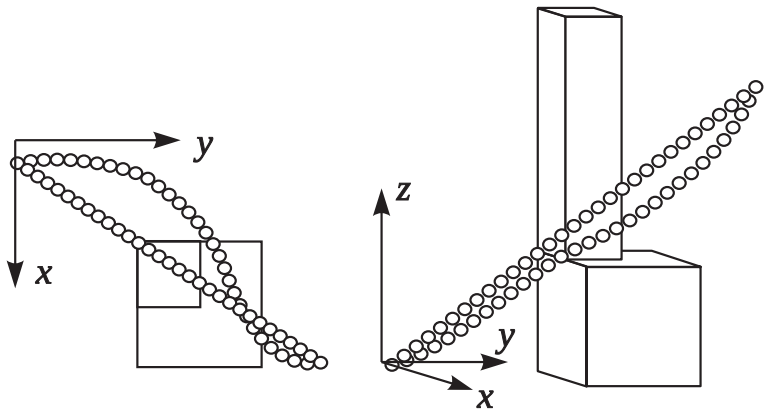


Rysunek 5.13: Neuronowa implementacja algorytmu $\dot{p}_i = -\eta \left[\frac{\partial E}{\partial p_i}\right]$.

w chwili początkowej powoduje, że wygładzone kary C_i^k nakładają się na siebie, co w efekcie przynosi likwidację lokalnych minimów E. Wraz ze wzrostem β kary „wyostrzają się", co sprawia, że „nitka zaczyna rozciągać się w kierunku minimum globalnego". Na rysunku 5.13 przedstawiono schemat sieci neuronowej, implementującej algorytm (5.30).

Przykład: „Neuronowość" procesu planowania ścieżki wynika z faktu przyjęcia neuronowej reprezentacji sceny. Model ten różni się od modelu przyjętego przy obliczaniu odległości odwrotnym skierowanie wektorów normalnych ścian oraz nieco zmienionymi parametrami sieci (inne typy neuronów i inne ich wartości progowe (*bias'y*)). Natomiast sama zasada funkcjonowania użytego w procesie planowania ścieżki algorytmu jest podobna do zasady funkcjonowania algorytmu obliczania odległości. Rozwiązanie bowiem otrzymuje się poprzez minimalizację zdefiniowanej wcześniej funkcji kryterialnej. W przypadku planowania ścieżki funkcją kryterialną jest funkcja definiująca energię „elastycznej nitki" (zobacz równanie (5.28)), zaś minimalizujący ją algorytm[‡] dany jest równaniem (5.30).

[‡]Jest to algorytm gradientowy.



Rysunek 5.14: Planowania bezkolizyjnej ścieżki metodą „elastycznej nitki".

Na rysunku 5.14 pokazany jest wynik poszukiwań bezkolizyjnej ścieżki w trójwymiarowej przestrzeni. Widoczne na nim punkty stanowią ślad, jaki zostawiła rozciągana w polu potencjałów nitka. W położeniu początkowym nitka ma kształt prostoliniowego odcinka, przecinającego obecną na scenie przeszkodę. Po uruchomieniu algorytmu (z parametrem β zmienianym zgodnie z zależnością $\beta = \beta_0(1 + t_0)$) nitka zaczęła rozciągać się, wysuwając się poza obszar zajmowany przez przeszkodę. Ostateczny kształt, jaki przyjęła nitka jest kształtem poszukiwanej ścieżki.

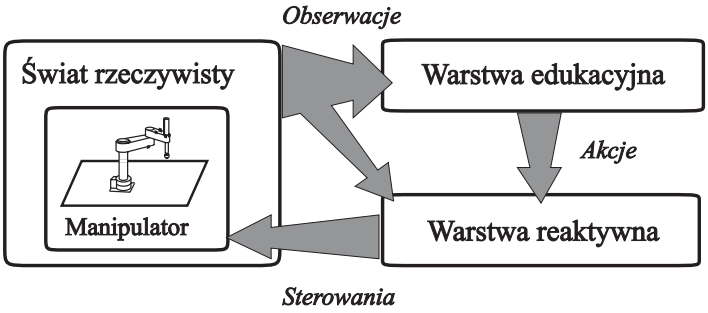
Rozdział 6

Inteligentny system planowania i sterowania autonomicznym agentem robotycznym

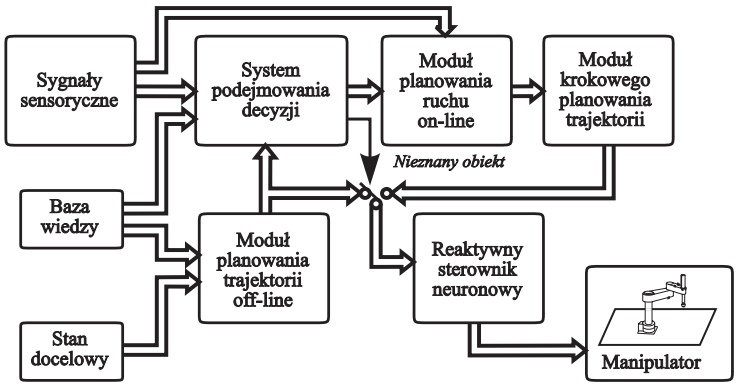
W nowoczesnych systemach robotycznych, w których współdziałanie wielu agentów koordynowane jest przez system nadrzędny, stosunek poziomu inteligencji koordynatora do poziomu inteligencji poszczególnych agentów można traktować jako pewnego rodzaju kryterium jakości. Kryterium to wyznacza, na jakim poziomie abstrakcji można określać zadania dla agentów, definiuje więc niejako ich autonomiczność. Dzięki posiadanej autonomiczności pojedynczy agent może wykonywać skomplikowane działania, samodzielnie bądź w grupie, bez konieczności wspierania przez system nadrzędny. Autonomiczność agenta można utożsamiać ze sposobem, w jaki reaguje on na zmiany otoczenia, w szczególności na zmiany postawionego zadania. Aby ją zdefiniować należy każdemu zaobserwowanemu stanowi przypisać odpowiednią akcję. Takie połączenie zbioru reguł (procedur) i odpowiadających im zachowań (reakcji) stanowi podstawę działania inteligentnego systemu autonomicznego. Niniejszy rozdział poświęcony jest przedstawieniu koncepcji takiego systemu, a w szczególności struktury i zasad funkcjonowania jednego z jego dwóch głównych elementów – systemu reaktywnego, [KM96].

6.1 Inteligentny robot autonomiczny

Na rysunku 6.1 przedstawiona jest struktura systemu inteligentnego robota autonomicznego. System ten, wykorzystując zgromadzoną wiedzę, bazując na sygnałach sensorycznych oraz na własnych sygnałach wewnętrznych, oblicza wartości momentów sterujących manipulatorem, zapewniając wykonanie postawionego zadania (etapem pośrednim w obliczaniu momentów sterujących jest rozwiązanie odwrotnego problemu kinematyki). W jego strukturze wyróżnić można dwie warstwy, [JBS96]: warstwę edukacyjną i warstwę reaktywną. Warstwa pierwsza odpowiedzialna jest za zbieranie informacji o otoczeniu manipulatora, jej generalizację i planowanie akcji w oparciu o wiedzę zgromadzoną w trakcie pracy. Natomiast warstwa druga (nazywana też systemem inteligentnego i reaktywnego sterowania robotem) odpowiada za wykonanie zaplanowanych akcji.



Rysunek 6.1: Struktura systemu inteligentnego robota autonomicznego.



Rysunek 6.2: Struktura systemu reaktywnego.

6.2 System inteligentnego i reaktywnego sterowania robotem

Warstwa reaktywna jest systemem, umożliwiającym sterowanie manipulatorem o przegubach obrotowych w otoczeniu niepewnym. Odpowiednie wejściowe momenty sterujące manipulatorem obliczane są w niej w oparciu o sygnały sensoryczne oraz własne sygnały wewnętrzne systemu. Innowacyjność tego systemu polega na zastosowaniu hybrydowych technik programowania, łączących metody numeryczne z wykorzystaniem sieci neuronowych*. W skład systemu reaktywnego wchodzą (zobacz rysunek 6.2):

- moduł planowania trajektorii off-line, MPT,
- system podejmowania decyzji, SPD,
- moduł planowania ruchu on-line, MPR, s z modułem krokowego planowania trajektorii, MKPT,
- reaktywny sterownik neuronowy, RSN.

Kiedy do systemu przekazany zostaje pożądany stan docelowy, wynik działania warstwy edukacyjnej, MPT rozpoczyna proces planowania bezkolizyjnej trajektorii, łączącej bieżący stan manipulatora z jego stanem docelowym. Planowanie to odbywa się z wyko-

*Sieci neuronowe konstruowane są przy pomocy metod symbolicznych, zobacz rozdziały 3, 4, 5.

rzystaniem zgromadzonej wiedzy o scenie[†]. Po zaplanowaniu trajektorii SPD uruchamia RSN, aby ten zaplanowaną trajektorię zrealizował. Jeśli poprzez system sensoryczny SPD zaobserwuje pojawienie się nieznanego obiektu, na wejście do RSN podana zostanie nowa trajektoria. Kolejne fragmenty tej trajektoria obliczane będą w trybie on-line, w trakcie współpracy MPR i MKPT.

Proces obliczania nowej trajektorii rozpoczyna się zaproponowaniem przez SPD bezpiecznego (ze względu na możliwość kolizji) i optymalnego (ze względu na postawione kryterium) kierunku ruchu efektora manipulatora. Na tym kierunku wyznaczany jest punkt, dla którego MPR obliczać będzie rozwiązanie odwrotnego zadania kinematyki. Jeśli w zadanym przedziale na kierunku poszukiwań nie uda się znaleźć bezkolizyjnego rozwiązania, MPR informuje o zaistniałym zagrożeniu SPD. Jeśli rozwiązanie takie zostanie znalezione, przekazane ono zostanie do MKPT.

Bezkolizyjna konfiguracja otrzymana z MPR oraz bieżąca konfiguracja manipulatora stanowią dwa punkty, przez które przechodzić ma planowana w MKPT trajektoria. Po każdorazowym zaplanowaniu fragmentu trajektorii, proces wyznaczania bezkolizyjnej konfiguracji i kolejnego jej fragmentu powtarza się. W czasie każdego takiego kroku obliczeń MPR i MKPT sprawdzają, czy bieżący punkt jest osiągalny ze względu na ograniczenia w przegubach i obecność przeszkód. Praca krokowa trwa do chwili, gdy: albo zostaje osiągnięty stan końcowy, albo niebezpieczeństwo kolizji znika i manipulator może wrócić na ścieżkę odpowiadającą trajektorii zaplanowanej off-line, albo też nastąpi blokada systemu. Wszystkie te trzy przypadki rozpoznawane są przez SPD, który wymusza na sterowniku manipulatora odpowiednie zachowanie. Wszystkie najważniejsze elementy systemu inteligentnego i reaktywnego sterownia robotem omówione zostaną dokładniej w kolejnych podrozdziałach.

6.3 Podstawowe elementy systemu inteligentnego i reaktywnego sterowania robotem

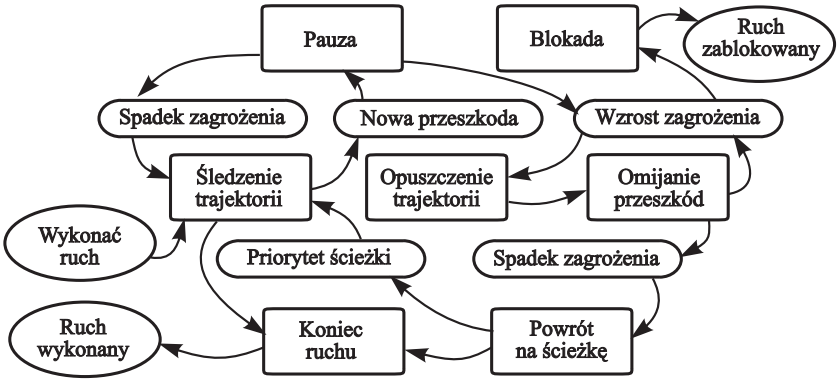
Jednym z elementów systemu inteligentnego i reaktywnego sterowania robotem jest moduł planowania trajektorii off-line. Ponieważ w literaturze opisanych jest wiele metod planowania trajektorii off-line, (jak np. w [Lat91b, RMR94, SI91]), możliwych do zaimplementowania w tym module, pozostawiono go bez opisu. Poniżej przedstawiona jest krótka charakterystyki pozostałych modułów systemu.

6.3.1 System podejmowania decyzji

System podejmowania decyzji jest miejscem, gdzie zgromadzona jest cała inteligencja reaktywnego agenta-manipulatora. Dzięki niej oraz systemowi percepcji, połączonego z bazą wiedzy o scenie, manipulator może samodzielnie, bez pomocy koordynatora, reagować na zmiany otoczenia. Zmiany te rozpoznawane są przez porównanie modelu zapisanego w bazie wiedzy z modelem wynikającym z bieżącej obserwacji.

Przeprowadzanie obserwacji umożliwia zespół sensorów zamontowanych bezpośrednio na ramionach manipulatora. Obserwacja nie oznacza tutaj tylko rejestracji obecności

[†]Jakkolwiek MPT stanowi element systemu inteligentnego i reaktywnego sterowania robotem, sam w sobie nie ma nic z „reaktywności". „Reaktywnością" odznaczają się dopiero pozostałe elementy systemy: SPD, MKP, MKPT, RSN.



Rysunek 6.3: Graf przejść pomiędzy kolejnymi stanami systemu podejmowania decyzji.

(nieobecności) przeszkody w zasięgu widzenia sensorów, lecz także ustalenie jej odległości od poszczególnych ramion i kierunku w jakim się ona znajduje. Zagadnienie to omawiane już było w podrozdziale 5.4.3.

Niech zaplanowana geometryczna (zobacz podrozdział 5.5) i dynamiczna trajektoria ruchu między punktami A i B, razem z lokalnym modelem sceny wyraża się przez, [JDKS95]:

$$L(A,B) := (q_{AB}(s), s(t), \delta_{AB}(s)), \quad \text{gdzie}$$

$q_{AB}(s)$ – ścieżka w przestrzeni wewnętrznej sparametryzowana parametrem, który nie jest czasem,

$s(t)$ – parametryzacja ścieżki czasem,

$\delta_{AB}(s) = (\delta_1(s), \delta_2(s), ..\delta_n(s))$ – przewidywany obraz otoczenia stworzony z wykorzystaniem bazy wiedzy, odpowiadający zaplanowanej ścieżce, gdzie $\delta_i(s)$ reprezentuje minimalny wektor odległości i-tego przegubu od statycznej (a więc znanej) przeszkody.

Dzięki tak przyjętemu modelowi sceny możliwym jest uzyskanie rzeczywistego obrazu otoczenia (zobacz rozdział 5) w trakcie pracy systemu. Podczas wykonywania ruchu generowane są, w wyniku porównania przewidywanego obrazu otoczenia z obrazem rzeczywistym, następujące zdarzenia w systemie podejmowania decyzji: **Priorytet ścieżki**, **Nowa przeszkoda**, **Wzrost zagrożenia**, **Spadek zagrożenia**.
System podejmowania decyzji jest zdarzeniowo zorientowanym systemem dynamicznym. Na rysunku 6.3 przedstawiony jest graf przejść pomiędzy jego stanami. Stan **Śledzenie** odpowiada sytuacji, w której manipulator podąża wzdłuż trajektorii zaplanowanej off-line. W chwili, gdy ruch ten kończy się, tzn. manipulator osiąga położenie końcowe, następuje przejście do stanu **Koniec ruchu**. Stan **Pauza** odpowiada chwili, w której manipulator nie porusza się. Kiedy na skutek pojawienia się nowej przeszkody manipulator musi zejść z trajektorii zaplanowanej, występuje stan **Opuszczenie trajektorii**. Przejście do tego stanu jest sygnałem do rozpoczęcia poszukiwania nowego kierunku ruchu (zobacz podrozdział 5.5) i wyznaczenia nowej trajektorii. Stan **Opuszczenie trajektorii** przechodzi w stan **Omijanie przeszkód** kiedy manipulator, realizując wyznaczaną on-line trajektorię, stara się ominąć przeszkodę. Z chwilą, gdy przeszkoda zostaje ominięta pojawia się stan **Powrót na ścieżkę**. Oznacza on, że manipulator powraca na ścieżkę odpowiadającą trajektorii zaplanowanej off-line (jeśli wskazuje na to **Priorytet ścieżki**) lub dąży

bezpośrednio do punktu docelowego. Stan **Blokada** odpowiada sytuacji, w której, ze względu na ograniczenia, żaden ruch nie może być wykonany.

6.3.2 Moduł planowania ruchów on-line

Jak wspomniano, w pewnych sytuacjach manipulator musi opuścić trajektorię zaplanowaną off-line znajdując nowe, bezkolizyjne położenia. W tym miejscu koniecznym staje się obliczenie, miedzy innymi, jego kinematyki prostej i odwrotnej. W rozdziałach 3 i 4 pokazano, jak wykorzystać do tego celu sieci neuronowe[‡]. W rozdziale 4 przedstawiony został neuronowo implementowalny algorytm obliczania kinematyki odwrotnej, w którym wyróżniono moduł STER jako element odpowiedzialny za sterowanie procesem obliczeń oraz za sygnalizowanie wyznaczenia (lub braku) rozwiązania. Pokazano ponadto, że struktura sieci może być wygenerowana automatycznie, przez moduł obliczeń symbolicznych[§]. Powodem, dla którego taki moduł nie występuje na schemacie przedstawiającym system reaktywny jest jego pełna autonomiczność i uniwersalność. Dokładniej mówiąc, moduł obliczeń symbolicznych posłużyć może do wygenerowania sieci neuronowych dla dowolnej ilości manipulatorów, jeśli tylko manipulatory te można opisać parametrami Denavit'a-Hartenberg'a. W tym sensie jest on niezależny i właśnie dlatego nie włącza się go do systemu inteligentnego i reaktywnego sterowania robotem.

6.3.3 Moduł krokowego planowania trajektorii

W systemie inteligentnego i reaktywnego sterowania robotem zadanie obliczania trajektorii ruchu, gdy manipulator znajduje się w stanie **Omijanie przeszkód** lub **Powrót na ścieżkę**, realizowane jest przez MKPT[¶]. Planowanie trajektorii przebiega przy założeniu, że ruchy manipulatora odbywają się z kawałkami stałymi przyspieszeniami w przestrzeni wewnętrznej[‖]. Trajektoria obliczana przez MKPT jest funkcją czasu, opisującą zmiany przyspieszeń, prędkości i położeń przegubów, łączącą stan bieżący manipulatora z położeniem otrzymanym z modułu planowania ruchu bez przekroczenia ograniczeń dynamicznych. W celu jej znalezienia w każdej iteracji MKPT oblicza:

- wartości ograniczeń na prędkościach i przyspieszeniach w przegubach,
- czas potrzebny na wykonanie ruchu, oraz
- następny fragment trajektorii.

Wyznaczenie ograniczeń dynamicznych. Aby obliczyć maksymalne przyspieszenia i prędkości na podstawie maksymalnych wartości momentów sterujących (jedynych znanych ograniczeń dynamicznych) zakłada się, że ruch do pośredniego punktu docelowego w bieżącej iteracji odbywa się prostoliniowo w przestrzeni wewnętrznej. Ponieważ dla

[‡]Sieci neuronowe umożliwiają kalibrację kinematyki (wykorzystana jest tu ich zdolność do uczenia się sieci) oraz pozwalają na dokonywanie obliczeń w niezależnym od liczbie stopni swobody manipulatorów czasie.

[§]Zapewnia się przez to większą dokładność, jak również skraca proces uczenia.

[¶]Krokowa praca tego modułu wynika z faktu, iż model sceny znany jest tylko lokalnie, niemożliwe jest więc obliczenie całej trajektorii.

[‖]Zaplanowanie trajektorii wymaga znajomości ograniczeń na prędkości i przyspieszenia w przegubach. Przy znanych jedynie wartościach maksymalnych sił i momentów sterujących, ograniczenia te obliczane są z modelu dynamiki.

typowych sytuacji kolejne punkty pośrednie leżą blisko siebie, rozwiązanie otrzymane przy tym założeniu jest dobrym przybliżeniem ograniczeń rzeczywistych. Na podstawie przyjętego założenia o ruchu z kawałkami stałymi przyspieszeniami, trajektoria w przestrzeni wewnętrznej sparametryzowana jest przez ściśle monotoniczną funkcję $s(t)$ (t - czas). Po podstawieniu tak sparametryzowanej trajektorii do równań dynamiki manipulatora (neuronowy nominalny model dynamiki, którym się posłużono, jest opisany podrozdziale 6.3.4), rozwiązuje się je ze względu na $\ddot{s}$ i $\dot{s}$ przy maksymalnych wartościach momentów sterujących. W konsekwencji wyznaczone zostają graniczne wartości przyspieszeń i prędkości przegubów w otoczeniu bieżącego fragmentu ścieżki.

Obliczenie czasu realizacji ruchu. Przedział czasu, w którym manipulator zdoła wykonać oczekiwany ruch, obliczany jest przy znanych już ograniczeniach na prędkości i przyspieszenia w przegubach. W pierwszej kolejności na podstawie pożądanej prędkości efektora w przestrzeni zewnętrznej wyznaczane jest dolne ograniczenie czasu realizacji ruchu $t_{d\,\min}$. Następnie, dla każdego z przegubów sprawdzane jest, czy $t_{d\,\min}$ wystarcza na wykonanie założonego ruchu bez przekroczenia ograniczeń. W przypadku, gdy warunek ten jest spełniony, $t_{d\,\min}$ jest poszukiwanym czasem ruchu. Jeśli dla któregoś z przegubów nastąpiło przekroczenie ograniczeń, obliczany jest nowy czas t_d, w którym założony ruch jest już realizowalny. Z każdym nowym t_d sprawdzane są warunki realizowalności ruchu dla pozostałych przegubów. Procedura ta jest powtarzana do chwili, kiedy czas t_d będzie odpowiedni do zrealizowania ruchu przez wszystkie przeguby.

Wyznaczanie fragmentu trajektorii. W poprzednich krokach obliczeń wyznaczany był czas realizacji ruchu t_d oraz ograniczenia na prędkości i przyspieszenia przegubów w postaci:

$$\begin{aligned} -\dot{q}_{\max} \le \dot{q} \le \dot{q}_{\max}, \\ \ddot{q}_{\min} \le \ddot{q} \le \ddot{q}_{\max}. \end{aligned} \tag{6.1}$$

Z założenia, że ruchy odbywają się z kawałkami stałymi przyspieszeniami w przestrzeni wewnętrznej wynika, że każdy z ruchów może być zrealizowany z nie więcej niż dwoma przełączeniami przyspieszeń (pierwsze przełączenie z początkowej wartości $\ddot{q}_0$ do wartości $\ddot{q}_1$ w chwili t_0, drugie z $\ddot{q}_1$ do $\ddot{q}_2$ w chwili t_1). Ponieważ w typowych przypadkach możliwych rozwiązań jest nieskończenie wiele, zdefiniowane zostały następujące kryteria wyboru rozwiązania optymalnego:

- **kryterium minimalizacji szarpnięć** w postaci:

$$Q_j = \int_{t_0}^{t_d} |\dddot{q}(t)| = |\ddot{q}_1 - \ddot{q}_0| + |\ddot{q}_2 - \ddot{q}_1|. \tag{6.2}$$

- **kryterium minimalizacji przeregulowań**, definiowane w dwojaki sposób:

$$Q_o = \|q(t) - \tilde{q}(t)\| = \begin{cases} \max\limits_{t_0 \le t \le t_d} |q(t) - \tilde{q}(t)| \\ \int_{t_0}^{t_d} (q(t) - \tilde{q}(t))^2 \mathrm{d}t \end{cases}, \tag{6.3}$$

gdzie $\tilde{q}(t)$ oznacza liniową trajektorię w przestrzeni wewnętrznej, łączącą bieżący stan z położeniem pożądanym.

Jak można zauważyć, oba proponowane kryteria optymalizacji mają postać:

$$Q = \Phi(\ddot{q}_1, \ddot{q}_2, t_1). \tag{6.4}$$

Problem znalezienia rozwiązania optymalnego można więc traktować jako problem minimalizacji Q_j i Q_o przy zachowaniu wszystkich wprowadzonych ograniczeń. Jest to typowy problem polioptymalizacji, który można przekształcić do problemu monooptymalizacyjnego z kryterium:

$$Q' = \lambda\tilde{Q}_j + (1-\lambda)\tilde{Q}_o \quad \text{z} \quad \tilde{Q} = \frac{Q - Q_{\min}}{Q_{\max} - Q_{\min}}, \quad \lambda \in [0,1]. \tag{6.5}$$

Występujący w równaniu (6.5) parametr λ określa, w jakim stosunku względem siebie wchodzą w kryterium mieszane kryteria (6.2) i (6.3). Zmieniając go zgodnie z wymogami zmieniającego się otoczenia manipulatora, bezpośrednio można wpływać na charakter planowanej w trybie on-line trajektorii.

6.3.4 Reaktywny sterownik neuronowy

Dynamika manipulatorów jest zazwyczaj wysoce nieliniowa i sprzężona, dlatego sterowanie manipulatorem wymaga zastosowania wyszukanych metod, [Cra81, SV89]. W omawianej aplikacji wykonanie trajektorii zaplanowanej przez moduł krokowego planowania trajektorii zapewnia typowy sterownik obliczanego momentu z pętlą sprzężenia typu PD i neuronowym modelem dynamiki (zobacz rysunek 6.4). Dla dynamiki manipulatora opisanej równaniem:

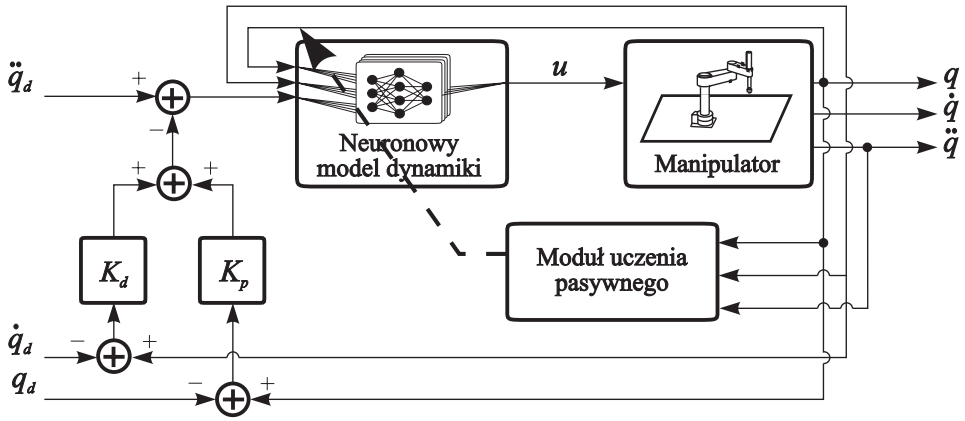
$$D(q)\ddot{q} + \dot{q}^T C(q)\dot{q} + R\dot{q} + G(q) = u, \tag{6.6}$$

gdzie D jest macierzą inercji, C macierzą współczynników sił odśrodkowych i Coriolisa, R jest macierzą współczynników tarcia, G jest wektorem sił grawitacyjnych, a u jest wektorem momentów i sił sterujących, sterownik obliczanego momentu z pętlą PD dany jest przez, [Cra81]:

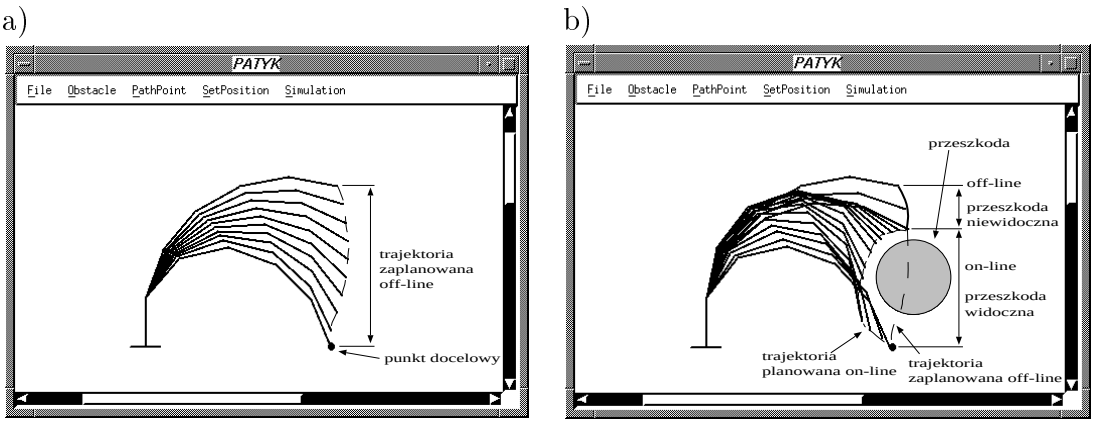
$$\begin{cases} u = D(q)w + \dot{q}^T C(q)\dot{q} + R\dot{q} + G(q) \\ w = \ddot{q}_d - K_d(\dot{q} - \dot{q}_d) - K_p(q - q_d) \end{cases}, \tag{6.7}$$

gdzie q_d, $\dot{q}_d$, $\ddot{q}_d$ są odpowiednio pożądanym położeniem, prędkością i przyspieszeniem przegubów, zaś K_p, K_d to dodatnio określone macierze wzmocnień sterownika.

Jak pokazano w [DJMB95], dynamikę manipulatora można zaimplementować za pomocą sieci neuronowych. Aby stworzyć jej neuronowy model skorzystać należy z obliczeń symbolicznych. Dzięki nim, na bazie danych technicznych manipulatora (długości ramion, ich masy itp.), posługując się formalizmem Euler'a-Lagrange'a, [SV89], wyznaczone zostały równania dynamiki. Równania te posłużyły następnie do stworzenia opisu sieci neuronowej i jej implementacji w sposób analogiczny do implementacji kinematyki. Ponieważ rzeczywista dynamika różni się od dynamiki nominalnej, konieczna jest kalibracja modelu. Kalibracja (polegająca na uczeniu sieci) przeprowadzana jest w dwóch trybach: aktywnym i pasywnym. Podstawowa różnica między tymi trybami polega na możliwości (tryb aktywny) lub niemożliwości (tryb pasywny) przeprowadzenia aktywnych eksperymentów na manipulatorze, [DJM95]. Aktywny tryb uczenia wykorzystany jest poza normalną pracą manipulatora. W wyniku jego zastosowania uzyskany zostaje model rzeczywistej dynamiki. Tryb pasywny uczenia włączany jest, gdy manipulator wykonuje już rzeczywiste zadania. Konieczność stosowania trybu pasywnego wynika z możliwości zmian warunków pracy manipulatora (np. poprzez zmianę obciążenia chwytaka).



Rysunek 6.4: Sterownik obliczanego momentu z neuronowym modelem dynamiki.



Rysunek 6.5: Wyniki symulacja pracy systemu inteligentnego i reaktywnego sterowania robotem.

Przykład: W niniejszym rozdziale przedstawiona została idea inteligentnego systemu planowania i sterowania autonomicznym agentem robotycznym. Ze szczególną uwagą omawiano w nim szczegóły związane z jednym z dwóch głównych elementów tego systemu, a mianowicie z systemem reaktywnym. Celem, jakiemu służyć ma niniejszy podrozdział jest dopełnienie przedstawionego tam opisu wynikami symulacji.

Efekt pracy systemu inteligentnego i reaktywnego sterowania robotem można zaobserwować na rysunku 6.5. W części a) tego rysunku pokazany jest manipulator, który wykonuje trajektorię zaplanowaną przez moduł planowania trajektorii off-line. Jako, że w czasie śledzenia tej trajektorii w otoczeniu manipulatora nie nastąpiły żadne zmiany, manipulator mógł prześledzić ją od początku do końca. W części b) rysunku 6.5 pokazana jest sytuacja, w której manipulator musiał, na skutek pojawienia się przeszkody, opuścić trajektorię zaplanowaną off-line (linia przerywana). Osiągnął on jednak położenie docelowe, gdyż dalej poruszał się wzdłuż trajektorii planowanej on-line.

Powyższe symulacje w ideowy sposób pokazały, że kombinacja metod użytych przy planowaniu ruchu off-line z metodami planowania trajektorii on-line jest bardzo efektywnym narzędziem do reaktywnego sterowania manipulatorem. Ponieważ w symulacjach tych, ze względu na dość skomplikowany model manipulatora, pominięto omawianie etapu neuro-

nowego obliczania dynamiki oraz reaktywnego sterowania, powstała w tym miejscu lukę uzupełniają wyniki kolejnych symulacji.

Wyniki te otrzymano bazując na modelu manipulatora typu podwójne wahadło**, o parametrach $l_1 = 0.6$, $l_2 = 0.3$ (l_1, l_2 są długościami ramion w bezwymiarowych jednostkach). Model dynamiki tego manipulatora został obliczony symbolicznie. Po podstawieniu w miejsce parametrów ich wartości numerycznych miał on następującą postać††:

$$D(q)\ddot{q} + G(q) + \dot{q}^T C(q)\dot{q} = u, \tag{6.8}$$

gdzie

$$D(q) = \begin{bmatrix} 81 + 36\sin(1.5708 + q_2) & 9 + 18\sin(1.5708 + q_2) \\ 9 + 18\sin(1.5708 + q_2) & 9 \end{bmatrix},$$

$$G(q) = \begin{bmatrix} 117.72\sin(q_1) + 29.43\sin(q_1 + q_2) \\ 29.43\sin(q_1 + q_2) \end{bmatrix},$$

$$C = \begin{bmatrix} [C_1] \\ [C_2] \end{bmatrix}, \quad C_1 = \begin{bmatrix} 0 & -18\sin(q_2) \\ -18\sin(q_2) & -18\sin(q_2) \end{bmatrix}, C_2 = \begin{bmatrix} 18\sin(q_2) & 0 \\ 0 & 0 \end{bmatrix}.$$

Równanie (6.8) zostało zaimplementowane, podobnie do równań kinematyki prostej, jako sieć neuronowa o sinusoidalnych neuronach.

Widoczne na rysunku 4.20 c) kolejne położenia manipulatora (począwszy od położenia najdalej wysuniętego w osi y, aż do położenia najmniej wysuniętego w tej osi) stanowią rozwiązania odwrotnego problemu kinematyki otrzymane z neuronowo zaimplementowanego algorytmu gradientowego. Przez konfiguracje te przechodzić miała trajektoria, planowana przez moduł krokowego planowania trajektorii‡‡, a realizację której wymusić miał reaktywny sterownik neuronowy, opisany równaniem (6.7), o wartościach współczynnikach wzmocnień $K_p = 50$ i $K_d = 20$.

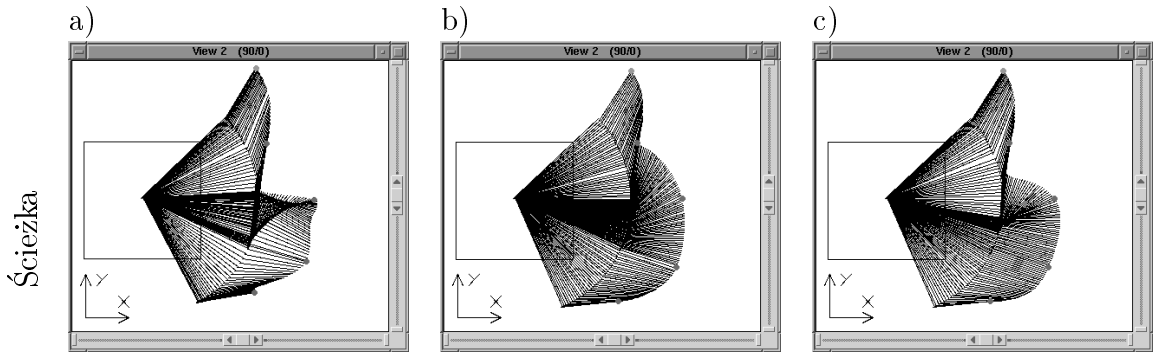
Ślad po przejściu manipulatora pomiędzy zadanymi położeniami przedstawia rysunek 6.6. W części a) tego rysunku pokazane są rozwiązania, które otrzymano, gdy trajektoria planowana była dla kryterium minimalizacji szarpnięć, (6.2). W części b) tego rysunku można zaobserwować rozwiązania, otrzymane przy planowaniu trajektorii z minimalizacją przeregulowań, (6.3). W części c) natomiast pokazane są rozwiązania otrzymane dla kryterium mieszanego, (6.5). Odpowiadnie czasowe wykresy sterowań u_1, przyspieszeń $\ddot{q}_1$, prędkości $\dot{q}_1$ oraz położeń q_1 i q_2 przedstawia rysunek 6.7*. Porównując rozwiązania a), b) i c) łatwo zauważyć, że: minimalizacja szarpnięć przy planowaniu trajektorii prowadzi do dużych przeregulowań (linia łamana na wykresach położeń q_1 i q_2 jest pożądaną, prostoliniową trajektorią przegubów, linia krzywa jest trajektorią otrzymaną), minimalizacja przeregulowań prowadzi do dużych szarpnięć (piki na wykresach przyspieszeń oraz

**Model kinematyki takiego manipulatora odpowiada modelowi kinematyki manipulatora typu SCARA. Manipulator *Adept One*, który został użyty w czasie eksperymentów, miał kinematykę właśnie tego typu.

††W modelu tym uwzględniono wpływ sił grawitacji, których kierunek działania był zgodny z kierunkiem osi x bazowego układu współrzędnych (w *Adepcie* siły grawitacji działają wzdłuż osi z, a więc nie wpływają na dynamikę). Nie uwzględniono natomiast oddziaływań sił tarcia.

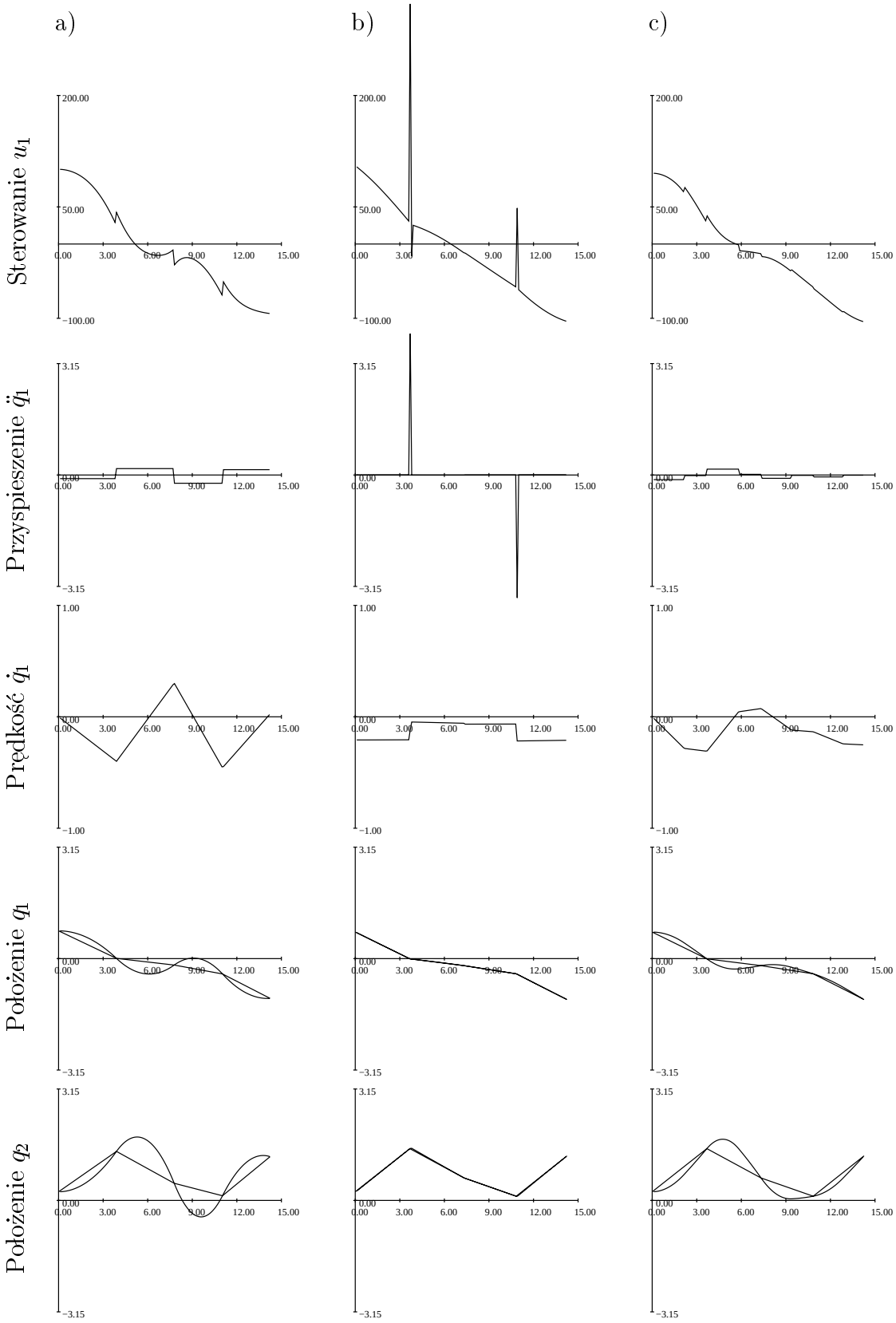
‡‡Zaplanowanie przejścia pomiędzy dwiema kolejnymi konfiguracjami stanowi jeden krok pracy modułu planowania trajektorii.

*Ze względu na czytelność rysunku pominięte zostały wykresy sterowań u_2, przyspieszeń $\ddot{q}_2$, prędkości $\dot{q}_2$.



Rysunek 6.6: Wyniki symulacja pracy systemu inteligentnego i reaktywnego sterowania robotem (ścieżki), gdy o wyborze trajektorii decydowało: a) kryterium minimalizacji przyspieszeń, b) kryterium minimalizacji przeregulowań, c) kryterium mieszane.

sterowań), natomiast minimalizacja mieszana, łącząc cechy obu wymienionych minimalizacji, dostarcza rozwiązań o małych przeregulowaniach i ograniczonych szarpnięciach. Kryterium mieszane, dzięki zmiennemu parametrowi λ (zobacz równanie (6.5) pozwala na elastyczne dostosowywanie charakteru planowanej trajektorii do wymogów postawionego przed manipulatorem zadania. Jeśli priorytetem przy planowaniu trajektorii jest dokładne jej śledzenie, wtedy λ przyjmuje wartość 0. Jeśli natomiast trajektoria zrealizowana być może z pewnymi błędami, $\lambda = 1$. Posługiwanie się taką strategią sprawia, że zaprezentowany moduł krokowego planowania trajektorii razem z neuronowym sterownikiem mogą dobrze spełniać swoje role w systemie inteligentnego i reaktywnego sterowania robotem, sam zaś system aktywnie może reagować na zmiany otoczenia.



Rysunek 6.7: Wyniki symulacja pracy systemu inteligentnego i reaktywnego sterowania robotem (sterowania, przyspieszenia, prędkości i położenia): gdy o wyborze trajektorii decydowało: a) kryterium minimalizacji przyspieszeń, b) kryterium minimalizacji przeregulowań, c) kryterium mieszane.

Rozdział 7

Metody koordynacji pracy wielu robotów

Patrząc poprzez pryzmat postawionego przed zespołem manipulatorów zadania, metody ich koordynacji podzielić można na dwie grupy: 1) metody koordynacji manipulatorów działających wspólnie (związanych poprzez trzymany obiekt) 2) metody koordynacji manipulatorów działających samodzielnie (nie związanych żadnym obiektem). Ze względu na możliwość wykorzystania zaprezentowanych wcześniej metod obliczania kinematyki odwrotnej, mając na uwadze przedstawioną w poprzednim rozdziale koncepcję inteligentnego systemu autonomicznego, omawianie metod koordynacji należących do drugiej grupy można by uznać za zakończone. Aby jednak ukazać problem koordynacji w szerszej perspektywie, w rozdziale tym znajdzie się miejsce na prezentację spotykanych w literaturze metod koordynacji należących do obu grup.

7.1 Koordynacja manipulatorów działających wspólnie

Problemy, jakie pojawiają się podczas koordynacji manipulatorów trzymających jakiś sztywny obiekt pokrewne są problemom sterowania zamkniętymi łańcuchami kinematycznymi. Ze względy jednak na specyficzne możliwości manipulatorów (niezależne sterowanie, możliwości zmiany chwytu, itp.) metody ich koordynacji posiadają własny, odmienny charakter.

Zazwyczaj zadanie koordynacji manipulatorów zdeterminowane jest przez podanie pożądanej trajektorii przenoszonego przez nie obiektu. Wraz z zapewnieniem realizacji tej trajektorii pojawić się również mogą zadania dodatkowe, jak np. żądanie równomiernego rozkładu sił pomiędzy manipulatorami, żądanie „delikatnego chwytu" (bez zgniatania trzymanego obiektu), żądanie „ruchomego chwytu" (obiekt może obracać się w chwytakach), żądanie minimalizacji zużytej energii. W zależności więc od nałożonych na zadanie ograniczeń (czy też jego specyfikacji) opracowane zostały różnego rodzaju metody koordynacji. I tak w [WFM91] zaproponowano metodę, w której wychodząc z równań ruchu (Newton'a i Euler'a) obiektu, wartości sił i momentów przyłożonych do niego przez poszczególne manipulatory* obliczane były poprzez odwrócenie tego równania. Jak pokazano, niepoślednią rolę przy obliczaniu sił i momentów odgrywa sposób, w jaki liczony jest inwers macierzy chwytu (*grasp matrix*). Posługując się odpowiednimi na inwers wzorami

*Założono tam, że wartości sił i momentów mogą być dowolne.

można całkowicie wyeliminować niebezpieczeństwo zgniecenia obiektu.

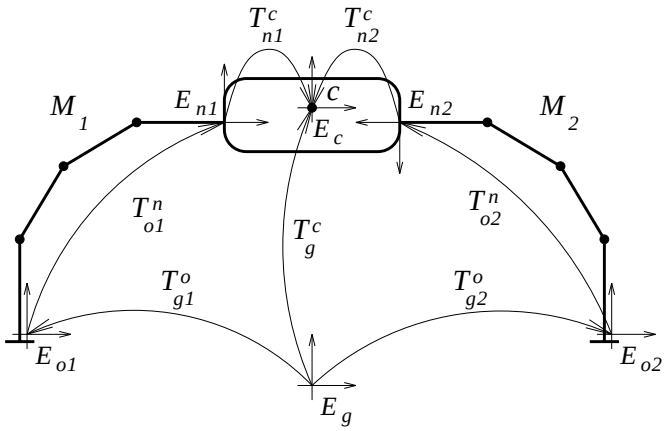
Nieco dalej w swoich rozważaniach posunęli się autorzy prac [Luh92, GDQ92]. Wychodząc z równania dynamiki obiektu (tego samego co w [WFM91]), posługując się równaniem jakobianowym, obliczali oni prędkości i przyspieszenia, a następnie siły i momenty rozwijane przez poszczególne manipulatory. W obu przypadkach odpowiednie równania zostały zagregowane w jeden system, przy czym w [Luh92] cała procedura (ograniczona do manipulatorów o 6 stopniach swobody) kończyła się eliminacją zmiennych i wyznaczeniem minimalnych sił i momentów. W [GDQ92] natomiast (dwa manipulatory o 6 stopniach swobody) obliczane jeszcze były sterowania (*robust position tracking controller* i *robust force tracking controller*).

Pomysł sterowania momentem przy śledzeniu zadanej dla obiektu ścieżki wykorzystano również w [JN94]. Jednak metoda tam przedstawiona różni się od zasadniczo od metody z [GDQ92]. Jej odmienność polega na skonstruowaniu systemu, na który składają się równolegle powiązane podsystemy, odpowiadające z osobna każdemu manipulatorowi. Dla tych właśnie podsystemów generowane są odpowiadające postawionemu zadaniu ruchy (trajektorie) przegubów. Ponieważ pomiędzy systemami istnieją iterakcje, wyznacza się je, wykorzystując do tego celu siły obliczone z wcześniej zdefiniowanej wirtualnych dynamik manipulatorów oraz wirtualnej dynamiki punktu reprezentującego obiekt. Uwzględniane są przy tym również ograniczenia wynikające z zależności kinematycznych pomiędzy manipulatorami i obiektem. W efekcie metoda umożliwia koordynację dowolnej liczby manipulatorów (ze względu na równoległość systemu, dodanie do niego nowych elementów nie stanowi żadnego problemu), przy czym postawione przed nimi zadanie może być utrudnione przez żądanie wykonywania dodatkowych ruchów manipulatorów względem siebie i obiektu (jak np. przesunięcie efektora wzdłuż krawędzi obiektu).

Przy okazji omawiania złożonych zadań koordynacji warto wspomnieć o metodzie zaproponowanej w [PYK94]. Umożliwia ona koordynację dwóch manipulatorów, o równej liczbie stopni swobody, obracających trzymany obiekt w chwytakach (*rolling contacts*). Wychodząc z równań opisujących pożądaną trajektorię obiektu, obliczane jest odpowiednie sterowanie siłą i momentem w pętli sprzężenia zwrotnego (linearyzującego i odsprzęgającego).

Na dużą uwagę zasługują również metody, w których korzysta się z zalet sieci neuronowych. Przykładem może tu być praca [CS91], której autorzy zaproponowali inteligentny sterownik o hierarchicznej strukturze. Zasada jego działania opiera się na „estymacji" przez predyktor efektów zastosowanych do podsystemów (manipulatorów z własnymi sterownikami) sterowań (*control commands*) i modyfikacji tych sterowań za pośrednictwem opartego o bazę wiedzy koordynatora, aż do wykonania postawionego zadania. Miejscem, w którym zaszyte zostały sieci neuronowe jest predyktor. Inteligencja natomiast zaszyta została w koordynatorze przeszukującym/budującym bazę wiedzy (z reguł IF ... THEN ...).

Przytoczone powyżej metody koordynacji charakteryzują się wspólną cechą. Uwzględnia się w nich mianowicie dynamikę manipulatorów jak i dynamiki samego obiektu. Zależności dynamiczne nie zawsze jednak muszą być brane pod uwagę. Na koordynację manipulatorów wspólnie trzymających obiekt można spojrzeć jak na problem czysto kinematyczny. W problemie tym nie ważny jest rozkład sił ani zależności czasowe. Poszukuje się natomiast rozwiązania odwrotnego problemu kinematyki dla wszystkich współdziałających manipulatorów, których położenia i orientacje zdeterminowane są przez położenie i orientację obiektu. Taka definicja koordynacji otwiera szerokie możliwości do



Rysunek 7.1: Rozmieszczenie układów współrzędnych.

zastosowania zaprezentowanych wcześniej metod neuronowego obliczania kinematyki odwrotnej. Jedynym koniecznym do zastosowania tych metod warunkiem jest wyrażenie położenia i orientacji obiektu (opisywanych normalnie w bazowym układzie współrzędnych) w układach współrzędnych, w których opisywana jest neuronowo implementowalna kinematyka poszczególnych manipulatorów.

Na rysunku 7.1 pokazany jest obiekt uchwycony przez manipulatory M_i $(i = 1, 2)$. Symbolem E_c na rysunku tym oznaczono układ współrzędnych związany z obiektem (umieszczony w jego środku ciężkości, c). Globalny układ współrzędnych oznaczony jest przez E_g. Symbolami zaś E_{oi} i E_{ni} oznaczono układy współrzędnych związanych odpowiednio z bazą i efektorem i-tego manipulatora. T^b_{ai} jest transformacją (opisaną macierzą) pomiędzy układam współrzędnych E_{oi} i E_b.

Znając trajektorię obiektu (a więc $T^c_g(t)$) bez trudu obliczyć można zadane macierzami położenia i orientacje P_i, w jakich w danej chwili muszą znaleźć się efektory manipulatorów, aby nie doszło do „upuszczenia" obiektu. Częstokroć wszelkie niedokładności (a co za tym idzie również niebezpieczeństwo zgniecenia obiektu) niwelowane jest przez specjalnie do tego celu skonstruowane chwytaki. Zależność P_i od T^c_g wyraża wzór:

$$P_i(t) = T^c_g(t)(T^c_{ni})^{-1}. \tag{7.1}$$

Z drugiej strony wiadomo, że każdy z manipulatorów w globalnym układzie współrzędnych opisany jest macierzowym równaniem kinematyki:

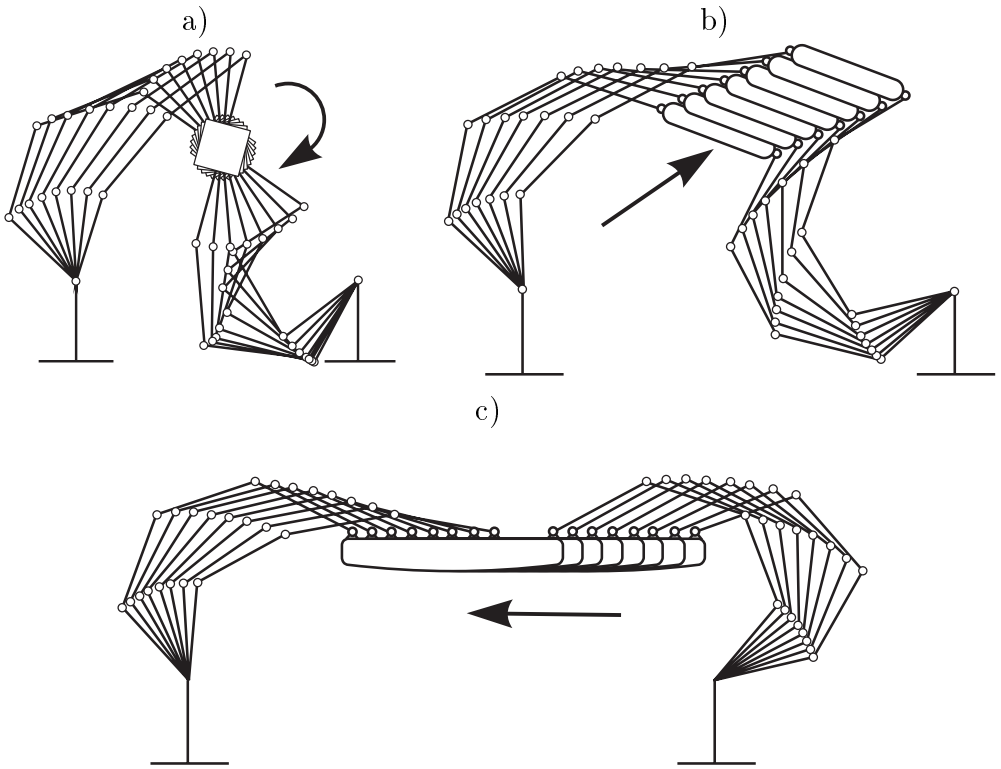
$$P_i = T^o_{gi} T^n_{oi}(q^i) \tag{7.2}$$

gdzie $T^n_{oi}(q^i)$ jest kinematyką manipulatora i wyrażoną w układzie współrzędnych z nim związanym (porównaj z równaniem 2.4). Ponieważ występująca w powyższym równaniu macierz T_{gi} jest macierzą pełnego rzędu o stałych, znanych wartościach elementów (dla stacjonarnego manipulatora), równanie to przekształcić można do postaci:

$$T^n_{oi}(q^i) = (T^o_{gi})^{-1} P_i, \tag{7.3}$$

co, po podstawieniu w miejsce P_i odpowiadającego mu znanego wyrażenia (równanie 7.1), przynosi:

$$T^n_{oi}(q^i) = (T^o_{gi})^{-1} T^c_g (T^c_{ni})^{-1}. \tag{7.4}$$



Rysunek 7.2: Koordynacja dwóch manipulatorów a) przy obracaniu trzymanego sztywno obiektu, b),c) przy przenoszeniu obiektu zawieszonego przegubowo.

Jak widać, jedyną niewiadomą w powyższym równaniu jest q_i. W powyższy sposób można więc sprowadzić problem koordynacji manipulatorów wspólnie trzymających obiekt do i odwrotnych problemów kinematyki zdefiniowanych w układach współrzędnych związanych z każdym z nich. Aby go rozwiązać, wystarczy teraz skonstruować i sprzężonych sieci neuronowych obliczających z osobna kinematykę odwrotną każdego z manipulatorów (takich jak sieć przedstawiona na rysunku 4.1).

Przykład: Efekty koordynacji dwóch manipulatorów (o czterech przegubach rotacyjnych, działających na płaszczyźnie) poprzez neuronowe obliczanie kinematyki odwrotnej dla każdego z nich ilustruje rysunek 7.2. W części a) tego rysunku pokazane są kolejne sekwencje ruchów manipulatorów podczas obracania trzymanego sztywno obiektu. W części b) i c) tego rysunku pokazane są kolejne sekwencje ruchów manipulatorów podczas przenoszenia obiektów przy ruchomym uchwycie. We wszystkich tych przypadkach kinematyka odwrotna manipulatorów obliczana była przez sprzężoną sieć neuronową (zobacz rozdział 4), po uprzednim obliczeniu zadanego położeń i orientacji efektorów na podstawie położenia i orientacji trzymanego obiektu.

7.2 Koordynacja manipulatorów działających samodzielnie

Z ujętą w tytule niniejszego podrozdziału koordynacją manipulatorów ma się do czynienia, gdy żadne z indywidualnie wykonywanych przez nie zadań nie obejmuje przypadku wspólnego (wspomaganego przez inny manipulator) przenoszenia obiektu. Zazwyczaj przez „samodzielne zadanie" rozumie się przeniesienie przez manipulator wyznaczonego przedmiotu z jednego miejsca w drugie, z jednoczesnym omijaniem przeszkód i unikaniem kolizji z pozostałymi manipulatorami. Partykularny interes, jaki przyświeca temu zadaniu, częstokroć bywa sprzeczny z interesem globalnym. Przy jego realizacji może bowiem dojść do blokady całego systemu. Dlatego też, gdy mowa jest o koordynacji manipulatorów, rzecz jest o metodach rozwiązywania następujących problemów[†]: 1) bezkolizyjne planowanie ruchu (indywidualnie dla każdego manipulatora, z uwzględnieniem tylko przeszkód statycznych); 2) bezkolizyjne planowanie trajektorii (w jaki sposób, jak szybko, przy uwzględnieniu jakich kryteriów optymalności wykonać ruch, aby nie doszło do kolizji); 3) planowanie działań (jak wykonać kolejne zadania, aby nie doszło do blokady systemu). Wszystkie te trzy problemy, w odniesieniu do pojedynczego manipulatora, rozwiązywane są przez przedstawiony w rozdziale 6 inteligentny system robotyczny. Już same moduły planowania ruchów on-line oraz krokowego planowania trajektorii (występujące w warstwie reaktywnej) wystarczają do rozwiązania prostych zadań koordynacji. Dowodzi tego przykład zamieszczonym poniżej. Jednak przed jego omówieniem przedstawiony zostanie krótki przegląd metod koordynacji manipulatorów spotykanych w literaturze.

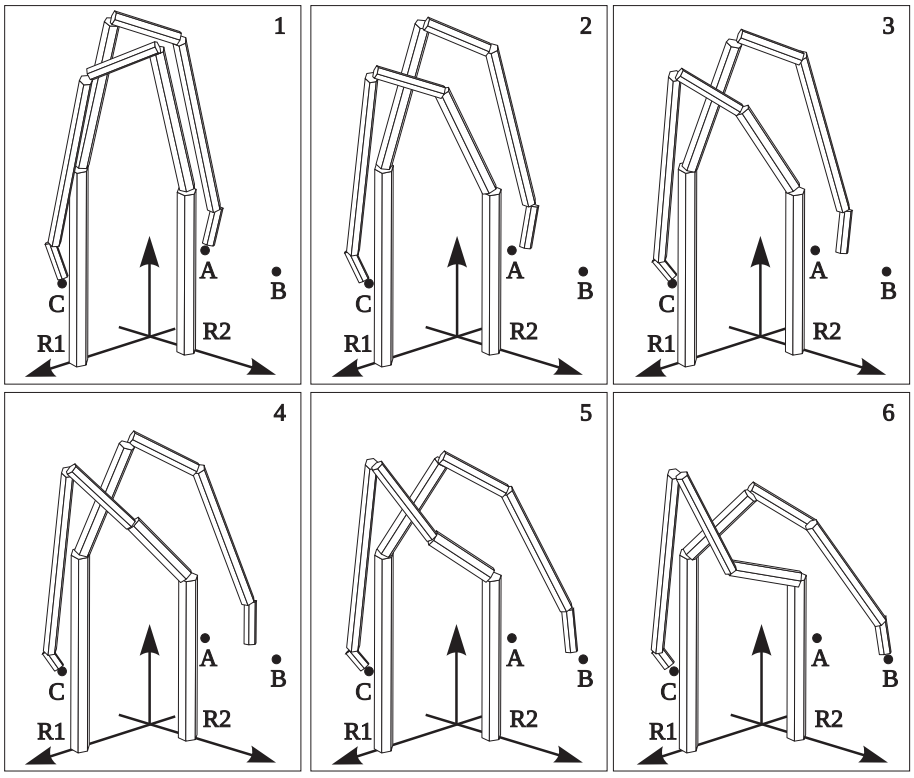
Problemowi koordynacji dwóch manipulatorów, których przestrzenie robocze przecinają się (*dual-robot system*) poświęcone są opracowania [SK92, SZ92]. W pierwszym z nich omawia się sposób planowania bezkolizyjnych, czasowo-optymalnych trajektorii, uwzględniając przy tym nałożone na manipulatory fizyczne ograniczenia (położenie i prędkości przegubów). Rozwiązania otrzymuje się poprzez minimalizację odpowiednio zdefiniowanych kryteriów jakości. Korzysta się przy tym z pomocy technik programowania nieliniowego[‡].
Natomiast w [SZ92] do planowania minimalno-czasowych trajektorii posłużył algorytmu aproksymacji dyskretnej (*perturbation trajectory improvement algorithm*). Jego zastosowanie uwarunkowane zostało założeniem, że bezkolizyjne ścieżki zaplanowane zostały przed przystąpieniem do planowania trajektorii, oraz w czasie śledzenia ścieżek manipulatory poruszają się tak szybko, jak to tylko możliwe, a w przypadku niebezpieczeństwo kolizji, jeden z nich zatrzymuje się.

W [LT91] przy planowania ruchów manipulatorów została zastosowana strategia *stop-and-go*. W opracowaniu tym kolizja występuje raczej w terminach szeregowania zadań niż w terminach obliczania odległości. Aby zaplanować bezkolizyjne ruchy (trajektorie) budowane są „mapy kolizji", tj. mapy, odzwierciedlające zależności kolizyjno-czasowe pomiędzy poszczególnymi manipulatorami i na podstawie map tworzony jest *disjunctive graph*. Samo planowanie sprowadza się do przeszukiwania tego grafu przy różnych strategiach[§] (bez priorytetów, z priorytetami, z uwzględnieniem prędkości manipulatorów).

[†]Pomijając zagadnienia związane z chwytem.

[‡]Użyciem technik optymalizacji z technikami dekompozycji zajmowano się również w [cH92], gdzie pokazano, że odległość między dwoma manipulatorami jest funkcją ciągła, Lipshitz'a i różniczkowalną.

[§]Planowanie ruchów jako problem szeregowania zadań częstokroć bywa rozwiązywany przy pomocy mechanizmów sieci Petri'ego, [BA91].



Rysunek 7.3: Kolejne sekwencje ruchu dwóch manipulatorów, których przestrzenie robocze przecinają się.

Przykład: Jak wyżej wspomniano, do koordynacji dwóch manipulatorów działających osobno można wykorzystać neuronowo implementowalne algorytmy odwracania kinematyki. Na potwierdzenie tych słów przeprowadzone zostały odpowiednie symulacje.

W zadaniu, którego kolejne etapy realizacji pokazane są na rysunku 7.3, manipulator **R1** przeprowadzić miał swój efektor z położenia początkowego **A** do położenia końcowego **B** (taki ruch odpowiada przeniesieniu jakiegoś obiektu z miejsca na miejsce), zaś manipulator **R2** nie mógł zmienić położenia swego efektora, zatrzymując go w punkcie **C** (można to interpretować jako przytrzymywanie jakiegoś obiektu w tym samym miejscu). Ponieważ przestrzenie robocze manipulatorów przecinają się, gdyby nie przyjęte postępowanie, przy realizacji tak postawionego zadania doszłoby do kolizji. Kolizję zdołano uniknąć poprzez zastosowanie neuronowo zaimplementowanego algorytmu obliczania kinematyki odwrotnej, z tym, że w przypadku manipulatora **R1** nie uwzględniano żadnych ograniczeń, natomiast w przypadku manipulatora **R2** ograniczeniem-przeszkodą był manipulator **R1**. Dodatkowo aspektem w przeprowadzanych symulacjach było sprawdzenie metod fuzji sygnałów sensorycznych, zaprezentowanych w rozdziale 5.4.3. Wyniki właśnie tej fuzji (uzyskane metodą aproksymacji) stanowiły dane wejściowe do uwzględniającego wpływ ograniczeń neuronowo zaimplementowanego algorytmu obliczania kinematyki odwrotnej.

Rozdział 8

Zastosowanie omówionych metod na stanowisku eksperymentalnym

Aby przetestować algorytmy obliczania kinematyki odwrotnej (z omijaniem przeszkód i uwzględnieniem ograniczeń konstrukcyjnych) na rzeczywistym manipulatorze, aby skonfrontować własności rzeczywistych sensorów z ich zasymulowanymi odpowiednikami oraz aby urzeczywistnić (częściowo) koncepcję systemu reaktywnego przeprowadzone zostały eksperymenty na specjalnie do tego stworzonym systemie badawczym. W niniejszym rozdziale przedstawiona zostanie krótka charakterystyka techniczno-funkcjonalna tego systemu (podrozdział 8.1). Podany również zostanie opis przeprowadzonych na nim eksperymentów wraz z dyskusją otrzymanych wyników (podrozdział 8.2).

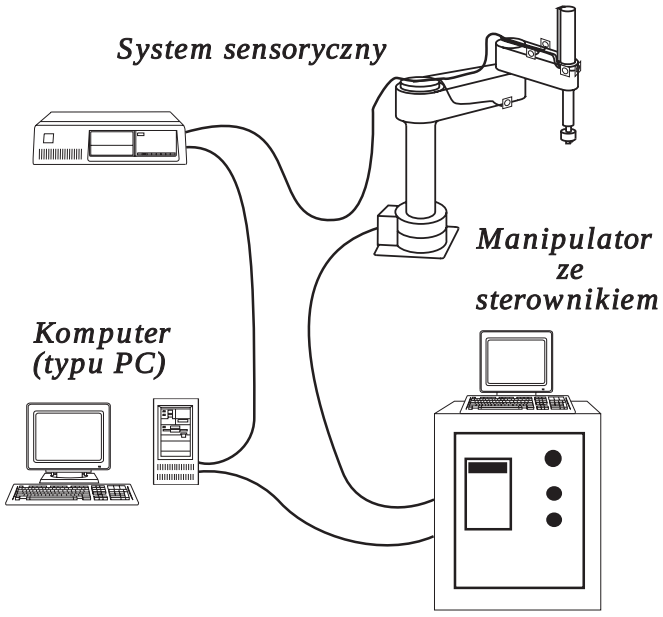
8.1 Charakterystyka techniczno-funkcjonalna systemu badawczego

Na rysunku 8.1 przedstawiony jest schemat systemu badawczego, na którym przeprowadzane były eksperymenty. System ten składa się z trzech głównych części: 1) manipulatora wraz ze sterownikiem, 2) systemu sensorycznego, 3) komputera zarządzającego (typu PC). Charakterystykę techniczno-funkcjonalną tych części przedstawiano kolejno w poniższych podrozdziałach.

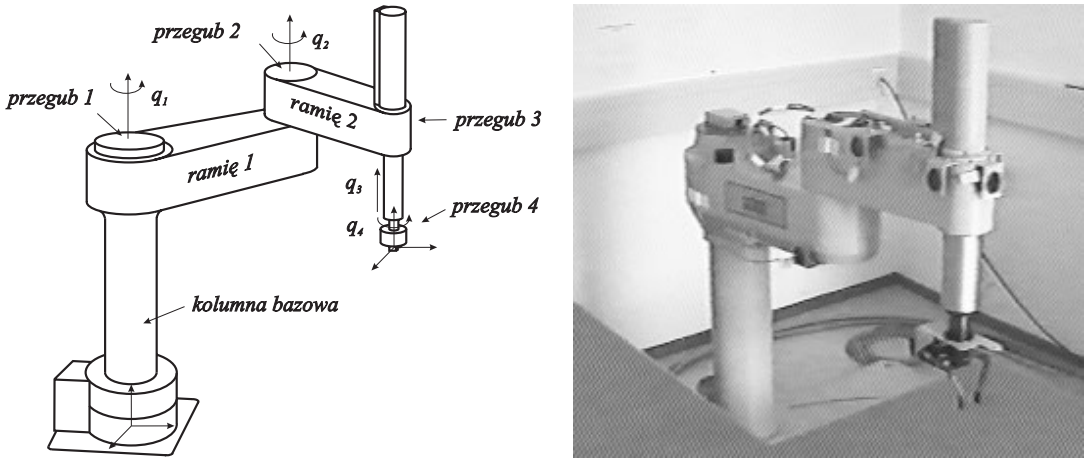
8.1.1 Manipulator i jego sterownik

Do eksperymentalnego testowania algorytmów obliczania odwrotnej kinematyki, wykorzystujących dane sensoryczne wybrany został manipulator przemysłowy *Adept One* (producentem manipulatora i jego sterownika jest Adept Technology Inc.).
Hardware: *Adept One* jest manipulatorem używanym głównie przy pracach montażowych. Jego konstrukcję określa się mianem *direct-drive*. Z kinematycznego punktu widzenia jego cztery stopnie swobody RRPR (zobacz rysunek 8.2) odpowiadają stopniom swobody manipulatora typu SCARA*. Najważniejsze parametry Adept'a, ze względu na rodzaj przeprowadzanych eksperymentów, to: długości ramion ($l_1 = 425$ mm, $l_2 = 375$ mm), zakres dopuszczalnych kątów w przegubach ($q_1 = \pm\frac{5\pi}{6}$, $q_2 = \pm\frac{49\pi}{60}$). Pozostałe parametry, jak również inne dane techniczne znaleźć można w [Ade86]. Razem z manipulatorem

* *Adept One* w przeciwieństwie do SCARY nie posiada przegubu translacyjnego na swojej kolumnie.



Rysunek 8.1: Schemat systemu badawczego.



Rysunek 8.2: Manipulator *Adept One*.

dostarczany jest jego sterownik wraz z systemem operacyjnym *Adept V*. W systemie tym urządzeniami we/wy są: terminal tekstowy (służący komunikacji użytkownik-sterownik); dysk twardy i stacje dysków miękkich (skąd uruchamiany jest system operacyjny, gdzie zapisywane są i skąd uruchamiane są programy robocze); zespół portów szeregowych (za pomocą których sterownik może komunikować się z urządzeniami zewnętrznymi, przesyłając znaki w kodzie ASCII); zespół we/wy binarnych (dwustanowe wejścia/wyjścia).
Software: System operacyjny *Adept V* umożliwia napisanie programów roboczych dla manipulatora w języku *Val II* i ich wykonanie. Pełną specyfikację języka *Val II* znaleźć można w [Ade88], zaś jego porównanie z innymi językami programowania robotów w [BW86]. Z punktu widzenia użytkownika wykonywanie poleceń definiujących zadanie, które wykonać ma manipulator, odbywać się może w dwóch trybach pracy: w trybie monitorowania (gdy kolejne komendy wydawane są bezpośrednio z klawiatury terminala) oraz w trybie programowym (kolejne komendy stanowią część aktualnie wykonywanego programu).
Funkcja: Zadaniem manipulatora ze sterownikiem było wykonywanie komend ruchu nadchodzących z komputera zarządzającego (a połączonego z jego sterownikiem przez złącze szeregowe) oraz przesyłanie komputerowi zarządzającemu danych o bieżącym położeniu przegubów w odpowiedzi na jego żądanie. Realizację tego zadania uzyskano uruchamiając bezpośrednio z terminala sterownika program, napisany w *Val II*[†]. W wyniku działania programu manipulator oczekiwał na komendę ruchu, a gdy taka komenda się pojawiła, natychmiast ją wykonywał. Podobnie było, jeśli przyszło żądanie wystawienia danych o położeniach przegubów. Sekwencje ruchu i oczekiwania powtarzane były aż do chwili przyjścia komendy zakończenia pracy. W przypadku, gdy kolejna komenda ruchu przyszła w chwili, gdy manipulator jeszcze się poruszał, komenda ta była buforowana i „sklejana" z poprzednią komendą ruchu. Dzięki zaletom sterownika (pojedyncze komendy ruchu wykonywane są równolegle (w tle) do komend głównego programu w *Val II*), wykonywanie następujących po sobie, „sklejonych" komend nie powodowało „drgawek" manipulatora. Dwa ruchy były bowiem sklejane w sposób gładki. Przepełnienie bufora komend ruchu (tzn. nadejście dwóch komend ruchu, gdy bieżąco wykonywana komenda jeszcze nie została zrealizowana) powodowało zastopowanie programu głównego na drugiej z nich. Blokada trwała dopóty, dopóki bieżąco wykonywana komenda ruchu nie została zrealizowana.

8.1.2 System sensoryczny

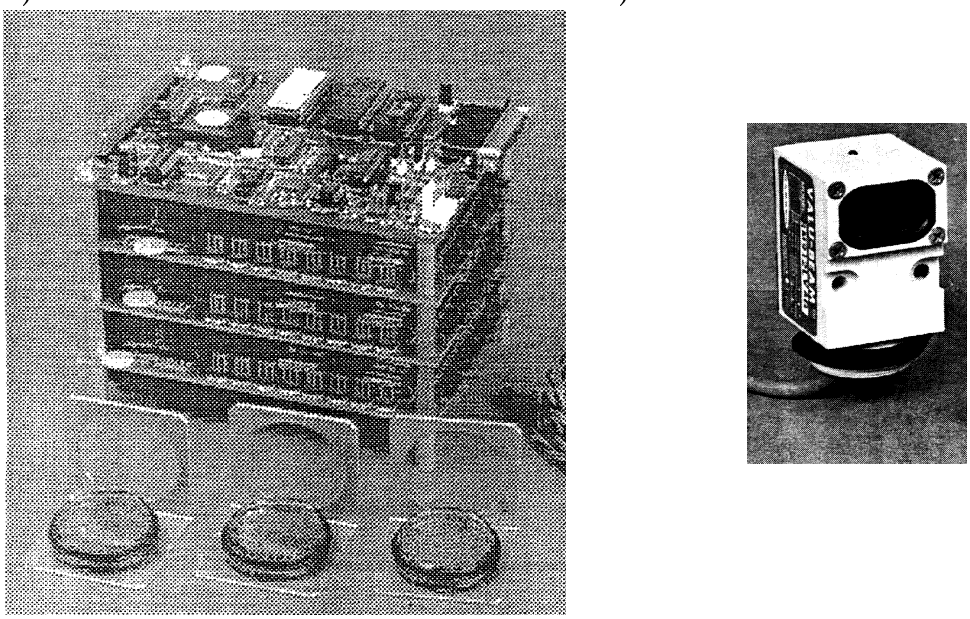
Do zbierania danych o otoczeniu manipulatora wykorzystywany był system sensoryczny ze sterownikiem sensorów o nazwie SonaRanger (producentem sterownika SonaRanger jest Transition Research Corporation (TRC)). System ten umożliwia dokonywanie pomiarów odległości za pomocą zestawu sensorów ultradźwiękowych[‡] oraz pozwala na rozpoznawanie faktu obecności lub nieobecności przeszkód (detekcja binarna) przy pomocy zestawu sensorów optycznych (działających w zakresie podczerwieni).
Hardware: Do elementów składowych systemu sensorycznego zaliczają się:

- płyta główna sterownika, wraz z jednostką centralną Motorola 68HC11, układami pamięci, gniazdami przyłączeniowymi oraz portem szeregowym RS-232;

[†]Do celów komunikacji opracowano prosty protokół transmisji danych. Ponadto, ustawienie dopuszczalnych wartości prędkości rozwijanych przez przeguby odbywało się już podczas procesu inicjalizacji sterownika.

[‡]Zobacz własności fizyczne sensorów, podrozdział 5.4.1.

a) b)

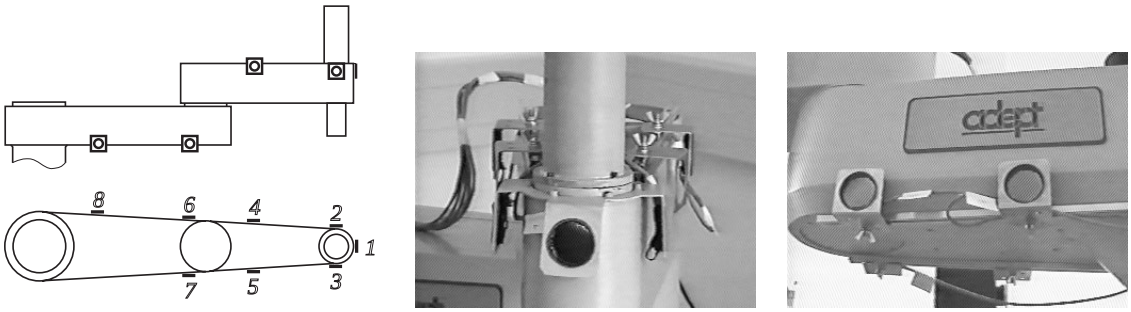


Rysunek 8.3: System sensoryczny a) SonaRanger w układzie 1 płyta główna + 3 płyty przyłączeniowe oraz trzy sensory ultradźwiękowe b) sensor optyczny.

- płyty przyłączeniowe, na których znajdują się gniazda przyłączeniowe sensorów;
- sensory ultradźwiękowe i optyczne;
- oprzyrządowanie pomocnicze, jak przewody, konektory, obudowa, zasilacz.

Ponieważ do płyty głównej sterownika podłączyć można maksymalnie 3 płyty przyłączeniowe, a do płyt przyłączeniowych podłączyć można do 8 sensorów ultradźwiękowych i do 8 sensorów optycznych (za pomocą osobnych, odpowiednio dwu i trzyżyłowych ekranowanych przewodów), maksymalna liczba obsługiwanych przez SonaRanger sensorów wynosi 2×24. Na rysunku 8.3 w części a) pokazany jest sterownik SonaRanger w konfiguracji 1 płyta główna + 3 płyty przyłączeniowe oraz trzy sensory ultradźwiękowe. W części b) tego rysunku pokazany jest sensor optyczny. Więcej szczegółów technicznych dotyczących sterownika znaleźć można w [Tra], dokumentacji dostarczanej przez TRC.

Software: Sterownik sonarów jest urządzeniem serwującym na żądanie urządzenia zewnętrznego bieżące odczyty z sensorów. Obsługa sensorów oraz wymiana informacji z urządzeniem zewnętrznym poprzez złącze szeregowe RS-232 możliwa jest dzięki programowi zaszytemu w pamięci na płycie głównej (*on-board software*), a wykonywanemu przez układ 68HC11. Cała komunikacja pomiędzy sterownikiem a urządzeniem zewnętrznym odbywa się zgodnie z zaproponowanym w [Tra] protokołem. W protokóle tym wyróżnione są: komendy wydawane sterownikowi (komendy inicjalizacji, tj. komendy ustalające kolejność „odpalania" sensorów, określające zasięg ich działania i partnerstwo; komendy żądające wystawienia bieżącego odczytu; komendy kończące pracę) oraz potwierdzenia i odpowiedzi na żądania, wysyłane przez sterownik. Sam proces filtracji zakłóceń i eliminacji fałszywych sygnałów realizowany jest na płycie głównej sterownika, dzięki czemu użytkownik obsługujący urządzenie zewnętrzne (komputer typu PC) nie musi się już o to



Rysunek 8.4: Umiejscowienie i uszeregowanie sensorów na ramionach manipulatora.

martwić.
Funkcja: Zadaniem systemu sensorycznego było dostarczanie do komputera zarządzającego danych (w postaci odczytów odległości z poszczególnych sensorów) o otoczeniu manipulatora. W tym celu wykorzystanych zostało osiem sensorów ultradźwiękowych, zamontowanych bezpośrednio na ramionach manipulatora. Maksymalny zasięg ich działania ustawiony był na odległość 1 m[§]. Dzięki uniwersalnemu mocowaniu, położenie i orientację sensorów można było zmieniać w szerokim zakresie. Na rysunku 8.4 przedstawione są sensory w położeniach i uszeregowaniu, jakie nadano im w czasie eksperymentów.

8.1.3 Komputer zarządzający

Do powiązania sterownika sensorów ze sterownikiem manipulatora w jeden system, realizujący wyznaczone wcześniej zadanie, wykorzystany został komputer typu PC.
Hardware: Z uwagi na charakter wykonywanych zadań oraz ilość przeprowadzanych w ich trakcie obliczeń, do eksperymentów wybrany został komputer z procesorem 486DX, taktowany zegarem 66 MHz. Na komunikację ze sterownikiem manipulatora i sterownikiem sensorów przeznaczone zostały dwa porty szeregowe (RS-232) tego komputera.
Software: Określone przez zadanie algorytmy zaimplementowane zostały w języku Visual C++.
Funkcja: Zadanie komputera zarządzającego polegało na:

1) inicjalizacji systemu sensorycznego;

2) zgłoszeniu żądania udostępnienia danych o otoczeniu manipulatora do sterownika sensorów oraz odebranie tych danych;

3) zgłoszeniu żądania udostępnienia danych o bieżącej konfiguracji manipulatora do jego sterownika oraz odebranie tych danych;

4) w zależności od postawionego zadania odpowiednim obliczeniu kinematyki odwrotnej z uwzględnieniem danych sensorycznych (Na podstawie bieżącej konfiguracji manipulatora oraz danych sensorycznych, przy znajomości geometrycznych manipulatora jak również położenia i orientacji sensorów na jego ramionach, obliczane było położenie przeszkód w scenie roboczej. Obliczenia te odbywały się względem

[§]We wszystkich przeprowadzonych eksperymentach zachowano tą samą wartość odległości maksymalnej.

układu bazowego manipulatora, w którym to układzie rozwiązywany był odwrotny problem kinematyki);

5) przesłaniu do sterownika manipulatora obliczonego rozwiązania w formie komendy ruchu (komenda ta zapewniała zmianę kątów w przegubach o pewien inkrement);

6) przesłaniu komendy zakończenia pracy do obu sterowników.

Wymienione wyżej punkty odzwierciedlają z grubsza (przy uwadze, że pomiędzy punktami 2 i 5 zapięto pętlę sprzężenia) strukturę wszystkich algorytmów, jakie zaimplementowano w języku Visual C++. Różnice pomiędzy poszczególnymi algorytmami występowały jedynie w punkcie 4, w którym obliczanie kinematyki odwrotnej zdeterminowane było przez postawione zadanie.

8.2 Opis eksperymentów

Konstrukcja Adepta, na którym przeprowadzane miały być eksperymenty, uniemożliwiała przetestowanie algorytmów obliczania kinematyki odwrotnej z jednoczesnym unikaniem przeszkód (odchylaniem ramion przy nie zmieniającym się położeniu efektora) jak miało to miejsce w czasie badań symulacyjnych. Dwa ramiona manipulatora (Adept faktycznie wykorzystany mógł być tylko jako dwuwahadło) z umieszczonymi na nich sensorami ograniczyły zakres badań eksperymentalnych do realizacji trzech następujących zadań: 1) zadania bezwzględnego unikania kolizji (pozwalającego na ucieczkę przed ruchomymi przeszkodami), 2) zadania „ inteligentntego" unikania kolizji, (umożliwiającego wyprowadzenie efektora z pułapki), 3) zadanie unikania kolizji „z nawrotem" (powodującego powrót do pozycji wyjściowej, opuszczonej wcześniej z powodu pojawienia się przeszkody). Przebieg eksperymentów, a właściwie kolejne kroki algorytmów, w myśl których się one odbywały, zdeterminowane był krokami algorytmów zaimplementowanych na komputerze zarządzającym (zobacz funkcje komputera zarządzającego, punkt 4, podrozdział poprzedni).

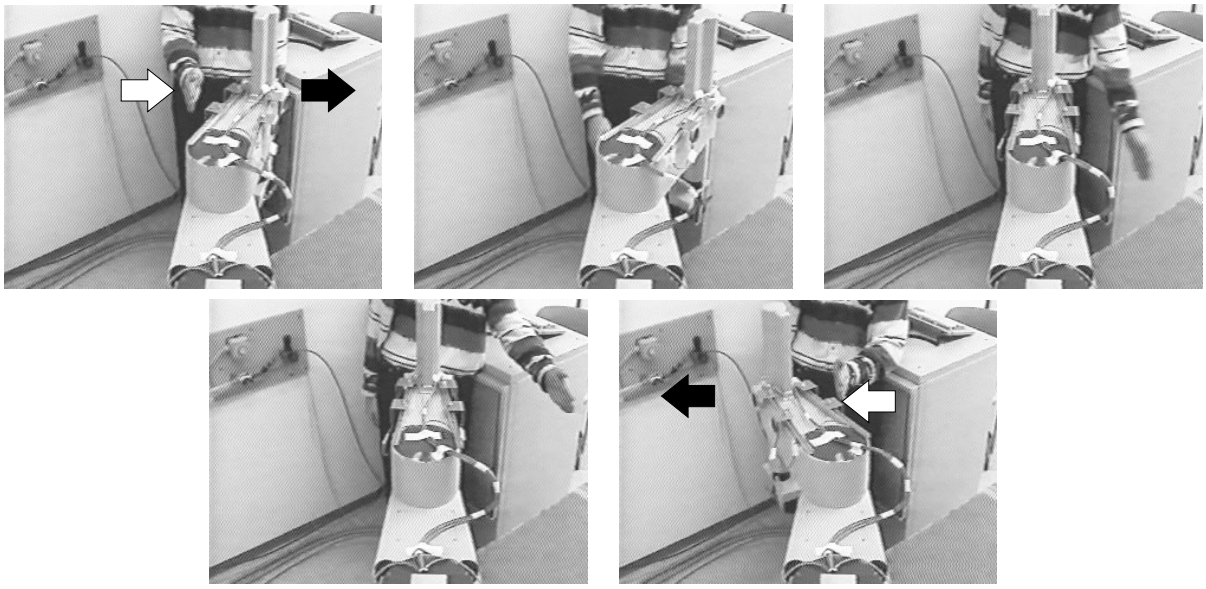
8.2.1 Zadanie bezwzględnego unikania kolizji

Głównym celem tego zadania było zapewnienie unikania kolizji manipulatora z przeszkodami, jeśli takowe pojawiały się w czasie jego pracy w polu widzenia zamontowanych na nim sensorów[¶].

Bezkolizyjne konfiguracje manipulatora obliczana była po uprzednim odczytaniu sygnałów sensorycznych (zobacz funkcje komputera zarządzającego i systemu sensorycznego, podrozdział poprzedni) i ich generalizacji (zobacz podrozdział 5.4.3). Jako, że w zadaniu bezwzględnego unikania kolizji położenie efektora nie ma szczególnego znaczenia, kinematyka odwrotna obliczana była przy pomocy algorytmów:

$$\dot{q} = \begin{cases} -\alpha J_o^T \Lambda e_o, & \text{jeśli} \quad e_{oi} \neq 0 \\ 0, & \text{jeśli} \quad e_{oi} = 0 \end{cases}, \tag{8.1}$$

[¶] Zarówno w tym, jak i w pozostałych zadaniach margines bezpieczeństwa, $\hat{d}_0$, ustawiony był na wartość 20 cm.



Rysunek 8.5: Realizacja zadania bezwzględnego unikania kolizji.

$$\dot{q} = \begin{cases} -\alpha J_i^T \Lambda e_{pi}, & \text{jeśli} \quad e_{pi} \neq 0 \\ 0, & \text{jeśli} \quad e_{pi} = 0 \end{cases} . \tag{8.2}$$

Powyższe algorytmy stanowią okrojoną wersją algorytmów (4.22) i (4.27). W ich skład wchodzą jedynie części pochodzące od funkcja kary za zbliżanie się do przeszkód (definicje funkcji kary, błędów dodatkowych i macierzy jakobiego zawierają równania: (4.20), (4.19), (4.24) oraz (4.25)).

Wyniki realizacji zadania bezwzględnego unikania kolizji na stanowisku eksperymentalnym, w postaci kolejno wykonanych zdjęć, przedstawia rysunek 8.5.

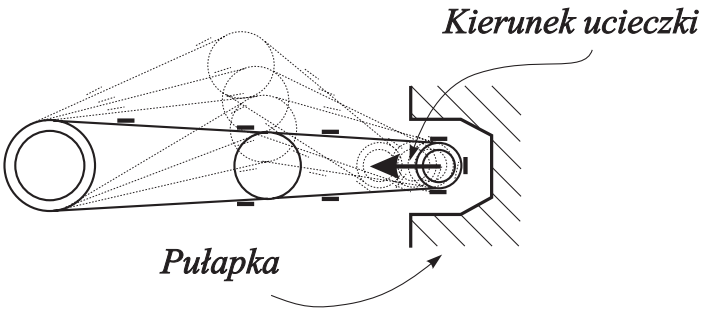
8.2.2 Zadanie „inteligentnego" unikania kolizji

W odróżnieniu od zadania bezwzględnego unikania kolizji, „inteligntne" unikanie kolizji wymagało wprowadzenia pewnych reguł na obliczanie kolejnych położeń efektora manipulatora. Reguły te wyrazić można następującym zdaniem: jeśli efektor manipulatora znajdzie się w pułapce, wyprowadzić go należy w kierunku do wewnątrz przestrzeni roboczej. Graficzną interpretację tak postawionego zadania przedstawia rysunek 8.6.

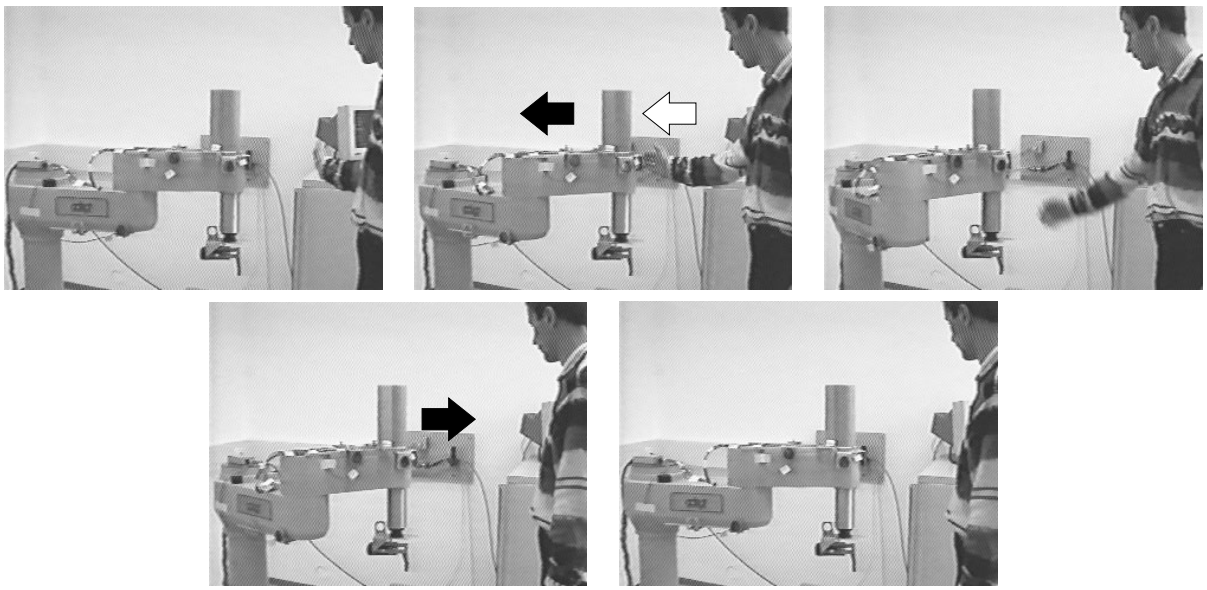
Podobnie jak przy zadaniu poprzednim, bezkolizyjne konfiguracje manipulatora obliczane były po uprzednim odczytaniu sygnałów sensorycznych i ich generalizacji. Jednak samo obliczanie kinematyki odwrotnej odbywało się zgodnie z algorytmami (4.22) i (4.27), w których za kolejne zadane położenia efektora przyjmowano punkty, leżące na kierunku wyprowadzającym z pułapki. Sam fakt złapania efektora w pułapkę rozpoznawany był na podstawie danych sensorycznych.

8.2.3 Zadanie unikania kolizji „z nawrotem"

W zadaniu tym wyróżniono dwie następujące fazy: fazę ucieczki przed przeszkodą (była ona równoważna zadaniu bezwzględnego unikania kolizji) oraz fazę powrotu do pozy-



Rysunek 8.6: Geometryczna interpretacja zadania „inteligentnego" unikania kolizji.



Rysunek 8.7: Realizacja zadania unikania kolizji „z nawrotem".

cji wyjściowej (była ona równoważna odwrotnemu zadaniu kinematyki, w którym zadane położenie efektora było jego położeniem sprzed ucieczki). Do realizacji tych faz, po uprzednim odczytaniu sygnałów sensorycznych i ich generalizacji, zostały użyte algorytmy: w fazie pierwszej – (8.1), (8.2), w fazie drugiej – (4.22), (4.27). Wyniki realizacji zadania unikania kolizji „z nawrotem" na stanowisku eksperymentalnym, w postaci kolejno wykonanych zdjęć, przedstawia rysunek 8.5.

8.3 Obserwacje i wnioski

Zdobyte podczas symulacji sensorów doświadczenia skonfrontowane były z eksperymentami na rzeczywistych sensorach. Stwierdzono, że, oprócz kształtu i wielkości przeszkody, kątów padania i odbicia wiązki akustycznej oraz powstających szumów, na wynik pomiarów wpływają (co było raczej oczywiste i do przewidzenia):

- rodzaj powierzchni (zdolność tłumienia, inaczej odbijała wiązkę tkanina, a inaczej metal),

- długości przewodów, którymi sensory podłączone zostały do sterownika oraz ich ekranowanie (dopiero podczas eksperymentów uwidocznione zostało, jakie bardzo indukcja elektromagnetyczna może zafałszować wyniki. Sensory, jako urządzenia elektroniczne, charakteryzują się dużą rezystancją wyjściową. Dlatego nawet małe prądy, powstające na skutek indukcji elektromagnetycznej, powodują powstanie dużych napięć na sensorach, a co za tym idzie, fałszują otrzymane wyniki. O indukcję zaś nie było trudno, gdy sterowano sensorami nadając impulsy o dużej częstotliwości oraz gdy uruchamiano napędzany silnikami elektrycznymi manipulator),

- drgania manipulatora i przeszkód.

Zakłamania w odczycie odległości spowodowały przyjęcie innego, niż w czasie symulacyjnego testów, potraktowania tych danych. Aby zaradzić zmieniającym się odczytom (przy tym samym wzajemnym położeniu sensor-przeszkoda, kolejne odczyty odległości różniły się od siebie) otrzymywane wartości musiano zdyskretyzować z odpowiednio dużym skokiem. Skutkiem takiego zabiegu było traktowanie odległości wpadających w przedział o przyjętej (5-cio centymetrowej) szerokości jako jedną i tą samą odległość. Powstała w ten sposób nieciągłość nie wpłynęła jednak na osiągane rezultaty. Rozwiązania odwrotnego problemu kinematyki, obliczane przy pomocy algorytmów (8.1), (8.2), (4.22), (4.27). były poprawne, i dzięki rozproszonemu przetwarzaniu danych *AdeptOne* mógł na bieżąco realizować postawione przed nim zadania. Niemała rolę przy realizacji zadań odegrał sterownik manipulatora. Dzięki wbudowanej zdolności do „sklejania ruchów", *AdeptOne* poruszał się płynnie, bez szarpnięć.
Podsumowując przeprowadzone eksperymenty można wygłosić ważkie stwierdzenie, że zaproponowane w niniejszej rozprawie algorytmy dają się zastosować w rzeczywistych aplikacjach. Przeprowadzone eksperymenty wykazały, że istnieje droga prowadząca od koncepcji inteligentnego systemu planowania i sterowania autonomicznym agentem robotycznym do jej implementacji.

Rozdział 9

Podsumowanie

W niniejszej rozprawie poruszony został bardzo istotny i wciąż na nowo rozpatrywany problem inteligentnego sterowania robotem w dynamicznym otoczeniu. O jego ważności i trudności zarazem świadczy fakt, iż w problemie tym wyróżnia się cały szereg złożonych podproblemów takich jak: obliczanie kinematyki prostej i odwrotnej, kalibracja kinematyki, planowanie ruchów i działań, wyznaczanie trajektorii i jej realizacja, itd. Aby więc rozwiązać problem inteligentnego sterowania robotem w dynamicznym otoczeniu należy znaleźć rozwiązania dla wszystkich wyróżnionych w nim podproblemów.

Na łamach niniejszej rozprawy w systematyczny sposób przedstawiono drogę prowadzącą do syntezy inteligentnego systemu autonomicznego, umożliwiającego sterowanie manipulatorem robotycznym o przegubach obrotowych w otoczeniu przeszkód. W systemie tym wielką rolę odgrywają specjalizowane sieci neuronowe, które, razem z propozycją struktury inteligentnego systemu autonomicznego oraz opracowaniem zasady funkcjonowania wchodzącej w jej skład warstwy reaktywnej, stanowią nowy wkład w dziedzinę badań nad inteligentnym sterowaniem.

Drogę prowadzącą do finalnego rozwiązania, tj. do syntezy inteligentnego systemu autonomicznego, zapoczątkowano omówieniem w rozdziale 2 podstawowych pojęć robotyki oraz dokonaniem przeglądu i klasyfikacji spotykanych w literaturze metod obliczania kinematyki prostej i odwrotnej. Wyróżniono następujące grupy metod obliczania kinematyki odwrotnej: metody symboliczne, metody numeryczne, metody z wykorzystaniem sieci neuronowych. Każdej z tych grup poświęcono osobny podrozdział (metody symboliczne opisano w podrozdziale 2.2.1, metody numeryczne i metody z wykorzystaniem sieci neuronowych opisano, odpowiednio, w podrozdziałach 2.2.2 i 2.2.3). Największą uwagą spośród wszystkich wymienionych metod obdarzono należącą do grupy metod numerycznych metodę transponowanego jakobianu. Na bazie właśnie tej metody rozwinięto później metodę obliczania kinematyki odwrotnej z wykorzystaniem specjalizowanych sieci neuronowych. Uprzednio jednak, w rozdziale 3, omówiono sposób, w jaki za pomocą sinusoidalnych sieci neuronowych można zamodelować kinematykę dowolnego manipulatora.

Sam fakt zastosowania sinusoidalnych sieci neuronowych do modelowania kinematyki wiąże się z dokonanym spostrzeżeniem, iż równania kinematyki mają postać wyrażeń trygonometrycznych, rozwijalnych do postaci ważonej sumy funkcji *sinus*. W rozdziale 3 przedstawiono, jak do takiej postaci przy pomocy obliczeń symbolicznych przekształca się równania kinematyki manipulatorów opisanych parametrami Denavit'a-Hartenberg'a. Przekształcenie to jest pierwszą fazą zaproponowanej w podrozdziale 3.1 metody syntezy sieci neuronowych implementujących kinematykę prostą. Wyższość zaproponowanej

sinusoidalnej sieci neuronowej nad innymi sieciami neuronowymi stosowanymi przy modelowaniu kinematyki prostej polega na większej dokładności obliczeń (z uwagi na dokładne odzwierciedlenie równań w strukturze sieci a nie tylko ich aproksymacji) oraz krótszym procesie uczenia (co osiąga się dzięki symbolicznemu wyznaczeniu wartości wszystkich wag sieci). Powiązanie obliczeń symbolicznych z procesem syntezy sinusoidalnych sieci neuronowych dostarcza dodatkowej zalety, której nie posiadają standardowe rozwiązania. Dzięki obliczeniom symbolicznym w prosty sposób, jak pokazano w rozprawie, otrzymać można sieci neuronowe implementujące jakobian manipulatora, a po kalibracji kinematyki, również i skalibrowany jakobian. Sam proces kalibracji kinematyki w przypadku jej neuronowej implementacji polega na nauczeniu sieci odwzorowania będącego kinematyką rzeczywistą. Dzięki zaletom sieci neuronowych kalibracja jest zabiegiem prostym i dającym doskonałe rezultaty (poziom błędów po kalibracji osiąga poziom szumów pomiarowych). W podrozdziale 3.2 zdefiniowano kryteria oceny efektów kalibracji jak również zaproponowano algorytmy uczenia sieci.

W celu dokładnego przedstawienia poszczególnych etapów syntezy sinusoidalnej sieci neuronowej implementującej kinematykę prostą, w podrozdziale 3.1 syntezy tej dokonano na przykładzie planarnego manipulatora typu 4R. Na przykładzie sinusoidalnej sieci neuronowej odpowiadającej x-owej współrzędnej położenia efektora tego manipulatora, w podrozdziale 3.2 omówiono efekty przeprowadzonej kalibracji. Podczas kalibracji rozważono przypadki, w których uczeniu (algorytmem *BP*) poddawane były tylko wagi warstwy wyjściowej, wagi warstwy wejściowej i wagi warstwy wejściowej, zaś uwzględnianymi źródłami błędów były: niedokładne wartość parametrów Denavit'a-Hartenberg'a, niedokładność przełożeń oraz pozycjonowania układu bazowego, zaszumienia pomiarów.

Metodę obliczania kinematyki odwrotnej przy użyciu specjalizowanych sieci neuronowych, jak już wspomniano, rozwinięto w niniejszej rozprawie na gruncie przygotowanym przez metodę transponowanego jakobianu (służącej do obliczania kinematyki odwrotnej) oraz metodę syntezy sinusoidalnych sieci neuronowych (służących do implementacji kinematyki prostej). W rozdziale 4, aby dokonać syntezy specjalizowanej sieci neuronowej obliczającej kinematykę odwrotną, zastosowano następującą metodologię postępowania:

- odwrotny problem kinematyki transformowano do problemu optymalizacji,
- wyznaczano algorytm zapewniający jego rozwiązanie,
- dokonywano neuronowej implementacji wyznaczonego algorytmu.

Obliczanie kinematyki odwrotnej w rozdziale 4 podzielono, w zależności od uwzględnianych w niej ograniczeń, na cztery osobne zagadnienia, mianowicie:

- problem bez ograniczeń,
- problem z ograniczeniami konstrukcyjnymi,
- problem z ograniczeniami wynikającymi z obecności przeszkód w scenie roboczej,
- problem z obydwoma typami ograniczeń.

W każdym z kolejnych podrozdziałów rozdziału 4 zdefiniowano osobny problem optymalizacji, odpowiadający wyżej przedstawionemu zagadnieniu (do funkcji kryterialnej wprowadzane były dodatkowe funkcje kary zależne od zdefiniowanych błędów dodatkowych).

Dla problemu wyznaczono rozwiązujący go algorytm i dokonano syntezy sprzężonej sieci neuronowej implementującej algorytm. W przypadku kinematyki odwrotnej bez ograniczeń pokazano (podrozdział 4.1), że wystarczy zapiąć pętlę sprzężenia zwrotnego na omówione w rozdziale 3 sinusoidalne sieci neuronowe, a powstały system, wzbogacony o sieć kontekstową obliczającą gradient funkcji kryterialnej oraz o moduł sterowania i moduł wyznaczający wartość kroku obliczeń, w którym w iteracyjny sposób generować będzie coraz dokładniejsze rozwiązania. W podrozdziałach 4.2, 4.3, 4.4 zademonstrowano, jakie moduły należy wprowadzić do systemu, aby uwzględniał on w rozwiązaniu wpływ ograniczeń. Założono przy tym, że dane o scenie roboczej dostarczane są w postaci wektorowej, informującej o odległości przegubów manipulatora od przeszkód. Dane te mogą być dostarczone przez system sensoryczny, bądź stanowić wynik obliczeń przeprowadzonych na modelu sceny. Dla każdego z modułów występujących w pętli sprzężenia opracowano jego neuronową (równoległą) implementację. Pokazano, że niezależnie od ilości modułów (a więc niezależnie od rodzaju ograniczeń w rozważanym odwrotnym problemie kinematyki) oraz niezależnie od liczby stopni swobody manipulatora, czas potrzebny na wykonanie jednej iteracji algorytmu oszacować można na osiem jednostek.

Na potwierdzenie poprawności zaproponowanych rozwiązań w podrozdziale 4.5 dokonano analizy zbieżności użytych algorytmów* oraz zademonstrowano wyniki przeprowadzonych symulacji (dwu i trzywymiarowych) uwzględniając z osobna wszystkie typy ograniczeń. Symulacyjnie również porównano efekt obliczania kinematyki odwrotnej z użyciem zaproponowanego algorytmu oraz algorytmu bazującego na metodzie całkowania Runge-Kutt'a i algorytmu GCM. Ostatecznie, wynikiem rozważań rozdziału 4 jest sprawdzona i efektywna (przy założeniu neuronowej (równoległej) jej implementacji) metoda obliczania kinematyki odwrotnej, uwzględniająca występowanie dowolnych ograniczeń.

Otrzymane w rozdziałach 3 i 4 wyniki można uogólnić, wychodząc poza klasę manipulatorów opisywanych parametrami Denavit'a-Hartenberg'a. Jedynym koniecznym warunkiem jest, aby równania kinematyki manipulatora spoza tej klasy były rozwijalne do postaci ważonej sumy funkcji *sinus*. Warunek ten pełni kluczową rolę w postawionej we wstępie niniejszej rozprawy i dowiedzionej właśnie omówionymi wynikami tezie: **zastosowanie specjalizowanych sieci neuronowych umożliwia efektywne i szybkie obliczanie dowolnych kinematyk prostych i odwrotnych dla manipulatorów redundantnych**.

Po opracowaniu neuronowo implementowalnych metod obliczania kinematyki prostej i odwrotnej kolejny krok prowadzący do syntezy inteligentnego systemu autonomicznego polegał na opracowaniu metod związanych z planowaniem ruchów. Ponieważ z każdym planowaniem ruchów wiąże się problem modelowania (czy to lokalnego czy też globalnego) sceny, pierwszy i drugi podrozdział rozdziału 5 poświęcone zostały omówieniu standardowych metod reprezentacji przeszkód w scenie oraz standardowych metod obliczania odległości i rozpoznawania kolizji. Następnie, w podrozdziale 5.3 zaproponowano nowy, neuronowy sposób reprezentacji przeszkód, wywodzący się z metody modelowania przeszkód wielościanami. Na bazie neuronowego modelu opracowano neuronowo implementowalną metodę obliczania odległości pomiędzy obiektami. Zasada, na podstawie której metoda ta została wyprowadzona, opiera się o te same założenia co zasada, według której opracowano metodę obliczania kinematyki odwrotnej. Na początek definiowany jest problem optyma-

*Ponadto w dodatku C zamieszczono opis algorytmu tunelowego, dającego możliwość modyfikacji omawianych algorytmów do takiej postaci, w której rozwiązania przez nie dostarczane będą rozwiązaniami globalnymi.

lizacji, następnie wyznaczany jest rozwiązujący go algorytm, na koniec zaś dokonywana jest synteza implementującej algorytm sieci neuronowej. Jak pokazały przeprowadzone w rozprawie analiza i symulacje, opracowany w ten sposób algorytm można użyć do szybkiego obliczania odległości (a co za tym idzie, do wyznaczania potencjalnych kolizji). Przy okazji neuronowego modelowania sceny, aby przygotować się do przeprowadzenia eksperymentów w środowisku naturalnym oraz aby przetestować nowe metody fuzji sygnałów sensorycznych, przeprowadzone zostały symulację systemu sensorycznego. Ich przebieg, począwszy od analizy fizycznych charakterystyk sensorów akustycznych aż po propozycję modelu i wyniki trójwymiarowych symulacji, przedstawiono w podrozdziale 5.4. W podrozdziale 5.4.3 omówione zostały, wpomniane już, dwie nowe metody fuzji sygnałów sensorycznych. Metody te dedykowane są do użycia przy planowaniu (czy też obliczaniu) bezkolizyjnego ruchu manipulatora z wykorzystaniem neuronowo zaimplementowanej kinematyki odwrotnej. Wynikiem przeprowadzonej fuzji w obu metodach, tj. w metodzie aproksymacji i metodzie projekcji, jest obliczenie pseudokierunku i pseudoodległości końca ramienia manipulatora, na którym umieszczone są sensory, do przeszkody. Metody te z powodzeniem zostały później zastosowane podczas symulacji pracy warstwy reaktywnej inteligentnego systemu autonomicznego oraz podczas eksperymentów przeprowadzonych na stanowisku badawczym. W ten sposób dowiedziona została druga teza niniejszej rozprawy: **sprzężenie specjalizowanych sieci neuronowych implementujących kinematykę prostą i odwrotną z modułami analizującymi sygnały sensoryczne umożliwia planowanie bezkolizyjnych ruchów manipulatora w czasie rzeczywistym**.

Bazując na neuronowym modelu otoczenia w podrozdziale 5.5 przedstawiono, obok metody standardowej, nową metodę planowania ścieżki dla punktu w trójwymiarowej przestrzeni. Zasada funkcjonowania nowej metody, podobnie jak miało to miejsce przy metodach obliczania odległości, opiera się na trzystopniowym schemacie. Na początek definiowany jest problem optymalizacji z zadaną funkcją kryterialną, następnie wyznaczany jest rozwiązujący go algorytm, aż wreszcie dokonywana jest synteza implementującej algorytm sieci neuronowej. Funkcja kryterialna dla tego przypadku ma postać energii sprężystości rozciąganej w polu potencjałów zdyskretyzowanej, elastycznej nitki. Jak pokazały przeprowadzone analiza i symulacje, zaproponowana metoda pozwala na zaplanowanie bezkolizyjnej ścieżki.

Wiążąc zaproponowane planowanie z metodami neuronowego obliczania odwrotnej kinematyki otrzymuje się system planowania bezkolizyjnych ruchów manipulatora (planowania bezkolizyjnej ścieżki dla punktu utożsamiane jest z planowaniem bezkolizyjnej ścieżki dla efektora manipulatora, uniknięcie kolizji ramion manipulatora z przeszkodami zapewniają algorytmy obliczania odwrotnej kinematyki). Przy globalnym modelu sceny, mimo iż zastosowano w systemie tym mechanizm obliczeń równoległych (sieci neuronowe), planowanie odbywać się musi w trybie off-line. Jeśli jednak założy się, że celem tego systemu jest znalezienie rozwiązań lokalnych przy lokalnej znajomości sceny, planowanie to może odbywać się w trybie on-line.

Ostateczny krok prowadzący do syntezy inteligentnego systemu autonomicznego w niniejszej rozprawie opisano w rozdziale 6. W rozdziale tym przedstawiono ideę struktury takiego systemu oraz omówiono zadania stojące przed wyróżnionymi w strukturze warstwami: warstwą edukacyjną (ma ona służyć gromadzeniu informacje o otoczeniu manipulatora, jej generalizowaniu oraz planowaniu akcji w oparciu o zgromadzone doświadczenia) i warstwą reaktywną (ma ona odpowiadać za wykonanie zaplanowanych akcji). Spośród tych dwóch warstw dokładnemu opracowaniu poddana została warstwa reaktywna. Przy

jej tworzeniu posłużono się wszystkimi zaproponowanymi do tej pory metodami oraz wprowadzono do niej kilka nowych elementów. Syntezę warstwy reaktywnej rozpoczęto od przedstawienia w podrozdziale 6.2 jej struktury wraz z opisem jej funkcjonowania. W kolejnych punktach podrozdziału 6.3 podano szczegółowy opis wchodzących w skład warstwy reaktywnej elementów:

- systemu podejmowania decyzji (oryginalna propozycja),
- modułu planowania ruchu (miejsce wykorzystania przedstawionych w poprzednich rozdziałach neuronowo implementowalnych algorytmów),
- modułu krokowego planowania trajektorii (nowa koncepcja, pozwalająca powiązać wyniki otrzymane przy planowanie ruchu ze sterowaniem zapewniającym ich realizację),
- reaktywny sterownik neuronowy (oryginalna metoda, w której skorzystano z doświadczeń zdobytych przy syntezie neuronowych kinematyk w celu otrzymania neuronowo implementowalnej dynamiki manipulatora).

Opis ten został poparty wynikami symulacji, pokazującymi istotę funkcjonowania wymienionych elementów. Symulacje te przeprowadzono dla manipulatora planarnego (reaktywne sterowanie zapewniło uniknięcie kolizji z dynamiczną przeszkodą, która się pojawiła na zaplanowanej uprzednio bezkolizyjnej ścieżce) oraz dla manipulatora typu podwójne wahadło (zastosowanie trzech różnych rodzajów strategii przy sterowaniu manipulatorem z uwzględnieniem jego neuronowej dynamiki zaowocowało trzema różnymi trajektoriami podczas krokowego ich planowania. Im bardziej zezwalano manipulatorowi oddalić się od krokowo zaplanowanej ścieżki ruchu, tym mniejsze były momenty sił nim sterujące). Rozdział 6 (razem z dwoma następnymi rozdziałami) potwierdza słuszność ostatniej z postawionych w rozprawie tez: **planowanie on-line bezpiecznych ruchów wraz z neuronowym sterownikiem dynamiki ruchu umożliwia syntezę autonomicznego systemu sterowania manipulatorem w środowisku dynamicznym.**

Dla pokazania pełnego obrazu możliwości wykorzystania zaprezentowanych w rozprawie wyników, neuronowe metody obliczania kinematyki odwrotnej, planowania ruchu, fuzji sygnałów sensorycznych przetestowane zostały na specjalnie do tego celu przygotowanym stanowisku doświadczalnym. Jednak przed opisaniem przeprowadzonych tam eksperymentów w rozdziale 7 pokazano, do jakich prostych zagadnień koordynacji pracy robotów (z kinematycznego punktu widzenia) można wykorzystać zaproponowane wcześniej rozwiązania. Dla celów ilustracyjnych w rozdziale tym zamieszczono przykładowe symulacje, w których koordynacja współpracy dwóch manipulatorów polegała na przenoszeniu wspólnie trzymanego obiektu oraz ustępowaniu sobie miejsca w przypadku pojawienia się niebezpieczeństwa kolizji.

Mając na uwadze potencjalne możliwości oferowane przez zaproponowany system reaktywny, pamiętając o omówionych zagadnieniach koordynacji, system badawczy został zbudowany jako system modułowy, składający się z trzech części: manipulatora i jego sterownika (*AdeptOne*), systemu sensorycznego (*SonaRanger*), komputera zarządzającego (typu PC). Na komputerze zarządzającym zaimplementowane zostały neuronowe metody obliczania kinematyki prostej i odwrotnej. Na nim również dokonywała się fuzja dostarczanych przez system sensoryczny sygnałów. Dzięki możliwości komunikacji ze sterownikiem manipulatora, komputer zarządzający wysyłał do sterownika komendy ruchu

i odczytywał z niego informacje o bieżącym położeniu manipulatora. Za realizacją komend ruchów przez manipulator odpowiedzialny był już sam sterownik. Jak pokazały eksperymenty (w których realizowano zadanie bezwzględnego unikania kolizji, zadanie „inteligentnego" unikania kolizji, zadanie unikania kolizji „z nawrotem") zaproponowane w niniejszej rozprawie metody z powodzeniem można wykorzystać w zagadnieniach sterowania manipulatorem w świecie rzeczywistym.

Warstwa edukacyjna inteligentnego systemu autonomicznego w niniejszej rozprawie została zaprezentowana w sposób ideowy. Rola, jaką powinna ona spełniać w tym systemie wyznacza kierunek dalszych badań. Opracowania powinny doczekać się metody zbieranie informacji o otoczeniu manipulatora, generalizacji tych informacji oraz metody planowania akcji w oparciu o wiedzę zgromadzoną w trakcie pracy. Przyszłe badania powinny dać odpowiedź na pytania, w jaki sposób powinno przebiegać uczenie się warstwy edukacyjnej w całym procesie „życia" oraz jakim poziomem inteligencji powinna warstwa ta się odznaczać. Na kierunku przyszłych badań wyróżnić można, wstępnie już poruszany, problem koordynacji pracy wielu robotów. W rozprawie pokazano prosty przykład koordynacji dwóch manipulatorów przenoszących wspólnie obiekt oraz grupy robotów pracujących w ciasnym otoczeniu jednego gniazda roboczego. Z problemem tym związane są jednak dużo bardziej skomplikowane zagadnienia, takie jak: opracowanie protokołów wymiany informacji pomiędzy robotami w celu ustalenie priorytetów i strategii współdziałania, wymiany doświadczeń, optymalizacji planowania, itp. Aby inteligentny system autonomiczny potrafił sobie z nimi poradzić, do jego struktury należałoby wprowadzić nowe komponenty. Takimi komponentami mogłyby być: agent zabezpieczający wykonanie ruchu, agent decydujący o bieżącym celu (interpretujący komendy zadawane np. słownie), agenci wyspecjalizowani w rozwiązywaniu szczególnych zadań planowania, agenci nawiązujący dialog z agentami innych systemów. W efekcie kontynuacja zaprezentowanych w rozprawie badań widziana jest jako opracowanie takich właśnie komponentów, rozszerzających możliwości opracowanego inteligentnego systemu autonomicznego.

Podziękowania

Pragnę serdecznie podziękować mojemu promotorowi, Panu Profesorowi Witoldowi Jacakowi, za wprowadzenie mnie w tematykę inteligentnego sterowania, za poświęcone na dyskusje ze mną wieczorne godziny, za umożliwienie mi pracy w laboratoriach Research Institute for Symbolic Computation, J.Kepler University, Austria, oraz za pomoc w realizacji niniejszej pracy.
Wyrazy wdzięczności chciałbym również skierować do Koleżanek i Kolegów z Zakładu Podstaw Cybernetyki i Robotyki, którzy szczodrze potrafili dzielić się ze mną swoją wiedzą i doświadczeniem. Dziękuję zwłaszcza Robertowi Muszyńskiemu i Ignacemu Dulębie za owocną współpracę przy rozwiązywaniu problemów związanych z przedstawioną tematyką. Dziękuję również moim austriackim Kolegom, w towarzystwie których powstawały niektóre fragmenty rozprawy, z których licznego grona wyróżnić chciałbym w tym miejscu Stephan'a Dreiseitl'a i Dietmar'a Schlosser'a.
Składam podziękowania Profesorowi Krzysztofowi Tchoniowi, promotorowi mojej pracy magisterskiej, za rady i pomoc okazaną podczas studiów doktoranckich, za zakładowe seminaria, które dzięki jego prowadzeniu były dla mnie nieocenionym źródłem wzbogacania wiedzy.

Dodatek A

Parametry Denavit'a-Hartenberg'a

W *metodzie Denavit'a-Hartenberg'a* transformacja pomiędzy układami współrzędnych związanymi z kolejnymi przegubami manipulatora ($i-1$ oraz i)* określona jest przez cztery parametry, [SV89]: β_i – obrót wokół osi z_{i-1}, d_i – przesunięcie wzdłuż osi z_{i-1}, a_i – przesunięcie wzdłuż osi x_i, α_i – obrót wokół osi x_i. Jak widać, parametry te odpowiadają czterem transformacjom podstawowym. Posługując się zapisem macierzowym, złożenie tych transformacji, A_i, wyrazić można następującym wzorem:

$$A_i = Rot(z_{i-1},\beta_i)\ Trans(z_{i-1},d_i)\ Trans(x_i,a_i)\ Rot(x,\alpha_i)$$
$$= \begin{bmatrix} \cos\beta_i & -\sin\beta_i\ \cos\alpha_i & \sin\beta_i\ \sin\alpha_i & a_i\ \cos\beta_i \\ \sin\beta_i & \cos\beta_i\ \cos\alpha_i & -\cos\beta_i\ \sin\alpha_i & a_i\ \sin\beta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{(A.1)}$$

gdzie $Rot()$ i $Trans()$ to, odpowiednio, macierz obrotu i macierz translacji. W przypadku, gdy i-ty przegub jest obrotowy, we wzorze (A.1) w miejsce β podstawia się współrzędną wewnętrzną q_i (kąt obrotu). Dla przegubu translacyjnego zamiast d_i wpisuje się q_i (przesunięcie).

Procedurę wyznaczania położenia poszczególnych układów, a w następstwie wyznaczania macierzowego równania kinematyki można przedstawić za pomocą następującego algorytmu:

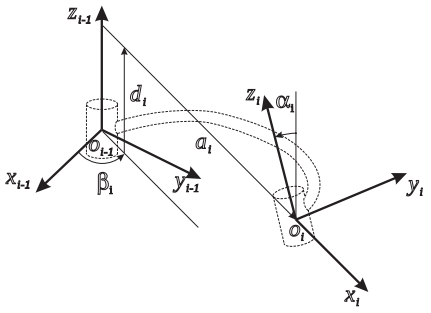
Krok 1: Umieścić osie układów $z_0,\ldots,z_{n-1}$ w kolejnych osiach przegubów manipulatora (zobacz rysunek A.1).

Krok 2: Umieścić początek układu bazowego E_0 w wybranym miejscu na osi z_0. Dopełnić układ osiami x_0, y_0 zgodnie z regułą prawoskrętną.
Dla $i=1,\ldots,n-1$ wykonać kroki algorytmu od 3 do 5.

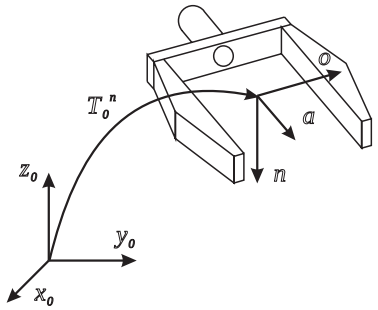
Krok 3: Umieścić o_i (początek układu E_i) w miejscu, gdzie normalna do osi z_i i z_{i-1} przecina z_i. Jeśli z_i oraz z_{i-1} przecinają się, o_i umieścić w punkcie przecięcia. Jeśli osie z_i, z_{i-1} są równoległe, początek układu E_i umieścić w przegubie i.

Krok 4: Położyć oś x_i wzdłuż normalnej między z_{i-1} i z_i przechodzącą przez o_i lub w kierunku normalnej do płaszczyzny $z_{i-1}-z_i$ jeśli z_{i-1} i z_i przecinają się.

*Przy założeniu, że manipulator składa się z $n+1$ ponumerowanych od 0 do n członów, gdzie podstawę manipulatora traktuje się jako człon o numerze 0, jego przeguby ponumerowane będą od 1 do n, tak że i-ty przegub będzie punktem w przestrzeni, gdzie łączą się człony $i-1$ oraz i.



Rysunek A.1: Parametry Denavit'a-Hartenberg'a



Rysunek A.2: Orientacja efektora

Krok 5: Dopełnić układ E_i osią y_i.

Krok 6: Ustalić położenie końcowego układu E_n. Zakładając, że n-ty przegub jest rotacyjny, ostateczne położenie układu efektora E_n uzyskać przez ustawienie osi $\bar{a}$ wzdłuż kierunku z_{n-1}. Początek układu o_e umieścić następnie na kierunku z_n. Zalecane jest by był to środek chwytaka albo koniec narzędzia, którym operować ma manipulator. Oś $\bar{o}$ ustawić na kierunku, w którym otwierają sie szczęki chwytaka (zobacz rysunek A.2. W przypadku skomplikowanych chwytaków , osie $\bar{o}$, $\bar{n}$ ustalić zgodnie z regułą prawoskrętną.

Krok 7 Utworzyć tablicę parametrów a_i, d_i, α_i, β_i.
a_i - odległość od o_i do miejsca przecięcia x_i i z_i mierzona wzdłuż x_i;
d_i - odległość od o_i do miejsca przecięcia x_i i z_{i-1} mierzona wzdłuż z_{i-1} (jeśli przegub i jest translacyjny, d_i jest zmienną q_i);
α_i - kąt między z_{i-1} a z_i mierzony wokół osi x_i;
β_i - kąt między x_{i-1} i x_i mierzony dokoła z_{i-1} (gdy przegub i jest rotacyjny, β_i jest zmienną q_i).

Krok 8: Znaleźć macierz transformacji A_i przez podstawienie wartości parametrów do wzoru (A.1).

Krok 9: Obliczyć macierz $T_0^n = \prod_{i=1}^{n} A_i$, reprezentującą położenie i orientację efektora w układzie bazowym. Macierz ta jest szukaną macierzą kinematyki.

Dodatek B

Algorytm *BP*

Algorytm wstecznej propagacji błędów, *Back Propagation algorithm*, jest jednym z najszerzej stosowanych algorytmów służących do tzw. uczenia sieci z nauczycielem. W uczeniu tym na wejścia i wyjścia sieci podaje się zestaw par uczących (są nimi wejście i wyjścia oczekiwane, odpowiadają odwzorowaniu, którego sieć ma się nauczyć), natomiast jej wagi trenowane są zgodnie z algorytmem.

Z matematycznego punktu widzenia algorytm *BP* zalicza się do algorytmów heurystycznych, które poprzez zmniejszanie gradientu w przestrzeni wag sieci minimalizują funkcję błędu.

Z implementacyjnego punktu widzenia, algorytm *BP*, poprzez systematyczną korekcję wag sieci, minimalizuje błąd pomiędzy otrzymywanym wyjściem a wyjściem oczekiwanym (w przypadku sieci modelującej kinematykę, wejście stanowią współrzędne wewnętrzne manipulatora, wyjście to jego współrzędne zewnętrzne, wyjście oczekiwane to rzeczywiste współrzędne zewnętrzne manipulatora, otrzymane w efekcie pomiaru). Swoją nazwę algorytm zawdzięcza temu, że mając wyznaczony błąd δ_j występujący podczas realizacji kolejnego kroku procesu uczenia w neuronie j, błąd ten jest rzutowany wstecz do wszystkich neuronów, których sygnały stanowiły wejścia do neuronu j.

Istota algorytmu *BP* wyrażona jest w poniższym wzorze na zmianę wagi w_{ij} pomiędzy neuronem i a neuronem j:

$$\triangle w_{ij} = -\eta \delta_j o_i, \tag{B.1}$$

gdzie δ_j zdefiniowana jest jako:

$$\delta_j = \begin{cases} (o_j^* - o_j) f'(net_j), & \text{jeśli } j\text{-ty neuron jest neuronem wyjściowym} \\ \sum_{k \in fanout} w_{kj} \delta_k f'(net_j), & \text{w przypadku przeciwnym} \end{cases}. \tag{B.2}$$

Zmienne o_j i o_j^* oznaczają, odpowiednio, aktualne i pożądane wyjście z neuronu j, η to współczynnik szybkości uczenia, $f'(net_j)$ to pochodna funkcji aktywacji, net_j jest ważoną sumą wejść do neuronu j, $fanout$ to numery neuronów, których wejścia połączone są z wyjściem z neuronu j.

Jak widać, algorytm *BP* wymaga znajomości pochodnej funkcji aktywacji. Przyjmując tradycyjny model neuronu McCulloch'a-Pitts'a, w którym dostępna jest tylko wartość jego wyjścia, o_j, pochodna funkcji aktywacji musi być wyrażona jako funkcja o_j. W przypadku sinusoidalnych neuronów postać pochodnej funkcji aktywacji, $f'(net_j) = \sqrt{1 - o_j^2}$, jest prawdziwa, jeśli $net_j \in [-\pi/2, \pi/2]$. Znaczy to, że przy tradycyjnym modelu neuronu,

jeśli $(w_j^k)^T\theta \notin [-\pi/2, \pi/2]$, wtedy algorytmu *BP* stosować nie można. W takim przypadku zestaw par uczących musi być wybierany z wielką ostrożnością.

Dodatek C

Algorytm Tunelowy.

Jednym z algorytmów, dzięki któremu można rozwiązać następujący problem:

Problem 8

$$\min_{x \in H} \quad f(x), \tag{C.1}$$

gdzie[*] $f : \mathbb{R}^n \to \mathbb{R}, x \in \mathbb{R}^n, H = [a_1, b_1] \times \ldots \times [a_n, b_n], a_i < b_i, i = 1, \ldots, n.$

jest algorytm tunelowy. Generalnie zasada funkcjonowania tego algorytmu opiera się na cyklicznym powtarzaniu dwóch faz obliczeń: lokalnej minimalizacji i tunelowania[†], aż do chwili, gdy znalezione zostanie minimum globalne. Na rysunku C.1 przedstawiono przykładowy schemat minimalizacji funkcji $f(x)$ za pomocą algorytmu tunelowego.

Jak można zauważyć, startując z punktu x_1^o podczas fazy minimalizacji pierwszego cyklu algorytmu znalezione zostało minimum lokalne x_1^*. W fazie drugiej, tj. w fazie tunelowania, z x_1^* otrzymano x_2^o. W punkcie x_2^o rozpoczął się drugi cykl pracy algorytmu. W fazie minimalizacji znalezione zostało x_2^*, a po tunelowaniu x_3^o. Trzeci cykl pracy algorytmu był cyklem ostatnim. Już w fazie minimalizacji znaleziony został punkt x_3^*, będący poszukiwanym minimum globalnym. Cyklicznie następujące po sobie fazy minimalizacji i tunelowania stanowią osobne procedury, przekazujące między sobą wyniki obliczeń.
Faza minimalizacji: w fazie tej poszukiwane jest minimum lokalne x_i^* funkcji f. W celu jego znalezienia można skorzystać z dowolnego algorytmu minimalizacji, w szczególności z algorytmu (4.31). Startując z zadanego punktu x_{i-1}^o obliczenia prowadzi się aż do momentu, gdy moduł z gradientu funkcji minimalizowanej bliski jest zeru ($\|f_x(x_i^*)\| < \epsilon$).
Faza tunelowania: w fazie tej dąży się do znalezienia takiego punktu x_i^o, dla którego $f(x_i^o) \leq f(x_i^*)$ i $x_i^o \neq x_i^*$ (x_i^o musi leżeć w obszarze innego zagłębienia funkcji f, zobacz rysunek C.1). Cel ten osiąga się, znajdując, przy pomocy dowolnego algorytmu, zera specjalnie skonstruowanej funkcji tunelowej, $T(x, x^*, \lambda^*)$.

Klasyczną funkcję tunelową definiuje się następującym ilorazem[‡]:

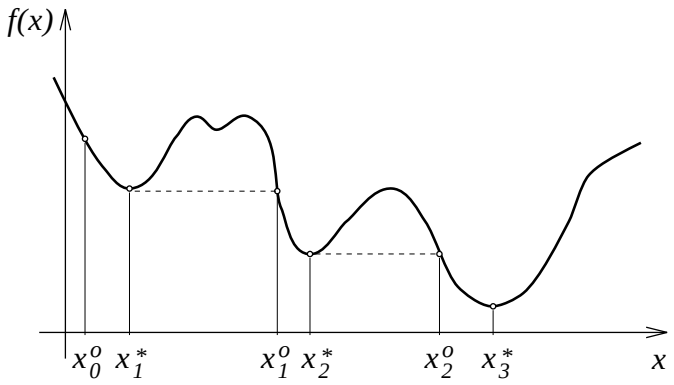
$$T(x, x^*, \lambda^*) = \frac{f(x) - f(x^*)}{\|x - x^*\|^{2\lambda^*}}, \tag{C.2}$$

gdzie λ^* nazywane jest mocą bieguna x^*. Znaczenie licznika i mianownika powyższego ilorazu obrazuje rysunek C.2. Licznik, tj. $f(x) - f(x^*)$ jest niczym innym, jak przesunięciem

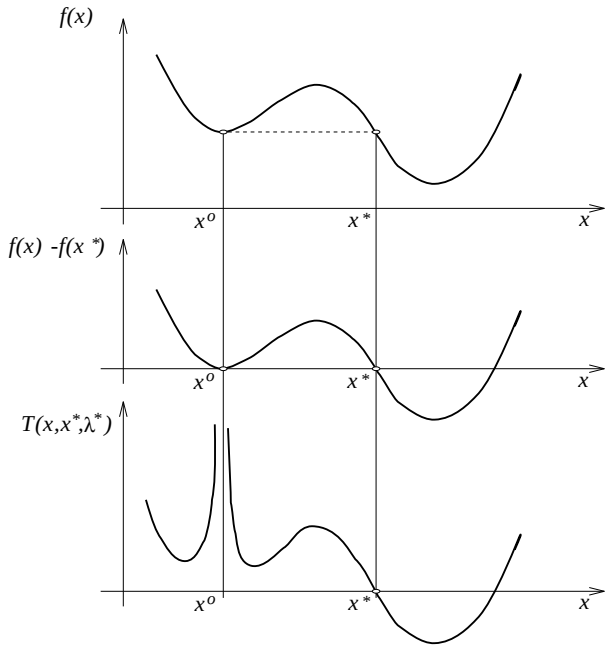
[*] Żądanie domkniętej dziedziny funkcji f potrzebne jest do zagwarantowania istnienia rozwiązania. Przy dowodzie zbieżności algorytmu zakłada się dodatkowo, że $f \in C^2$, [LM80].

[†] Tunelowanie ma tu podobne znaczenie jak tunelowanie w fizyce, gdy mówi się o przejściu przez barierę potencjału.

[‡] Obok klasycznej funkcji tunelowej opracowano również exponencjalną funkcję tunelową, [GB91].



Rysunek C.1: Przykładowy schemat pracy algorytmu tunelowego.



Rysunek C.2: Zasada konstruowania funkcji tunelowej.

wykresu funkcji f o wartość $-f(x^*)$. Dzięki temu poszukiwanie punktu $x^o \neq x^*$ sprowadza się do poszukiwania nowego zera funkcji tunelowej. Mianownik, tj. $\|x - x^*\|^{2\lambda^*}$ służy do wyeliminowania ze zbioru zer funkcji tunelowej znanego już x^* (minimum lokalnego f). Ze względu na charakter tej eliminacji, mówi się tu o wprowadzeniu bieguna w punkcie x^* o mocy[§] charakteryzowanej przez parametr λ^*. Ponieważ funkcja $f(x)$ może posiadać wielokrotne minima lokalne na tym samym poziomie, istnieje niebezpieczeństwo wielokrotnego przeskakiwania z jednego takiego minimum do drugiego. Dlatego też funkcję tunelową modyfikuje się tak, aby uwzględnić w niej wszystkie znalezione w bieżącej fazie wielokrotne minima x_i^* (w liczbie p) wraz z odpowiadającymi im parametrami λ_i^*:

$$T(x, x^*, \lambda^*) = \frac{f(x) - f(x^*)}{\prod_{p} \|x - x_i^*\|^{2\lambda^*}}. \tag{C.3}$$

Kiedy już znalezione zostanie nowe zero funkcji tunelowej, x^o, następna faza tunelowania rozpocznie się z wyjściową postacią funkcji T, (C.2).

Klasycznym algorytmem, z którego korzysta się podczas fazy tunelowania jest algorytm wywodzący się ze Stabilizowanej Metody Newtona, [LM80, YI95]. Metoda ta składa się z dwóch etapów. W pierwszym z nich, (iteracyjną metodą Newtona), rozwiązuje się układ równań

$$T_x(x_k, x^*, \lambda^*)\ dx_k = -T(x_k, x^*, \lambda^*) \tag{C.4}$$

ze względu na wektor przyrostów dx_k (na początek $x_0 = x^* + \epsilon r$, gdzie $\epsilon \ll 1$, $r \in \mathbb{R}^n$– losowo wybrany wektor jednostkowy, $T_x = \frac{\partial T}{\partial x}$) i z równania

$$x_{k+1} = x_k + \beta\ dx_k \tag{C.5}$$

oblicza się x_{k+1}– punkt startowy do kolejnej iteracji (rozpoczynającej się rozwiązaniem równania (C.4)). Krok obliczeń β dobierany jest w taki sposób, aby zapewnić zmniejszanie się normy błędu, $P(x)$, danej przez $P(x) = T(x)^T T(x)$. Do tego celu używa się metody bisekcji, tj. metody, w której startując z $\beta = 1$, β dzielone jest przez 2 dopóty, dopóki nie zostanie spełniony warunek $P(x_{k+1}) < P(x_k) < P(x^*)$. Iteracje (C.4), (C.5) przerywa się, gdy wartość β osiągnie wartość bliską zera. Jeśli otrzymany z ostatniej iteracji punkt x_k znalazł się w okolicy punktu osobliwego x_s, tzn. punktu, dla którego $T_x(x_s, x^*, \lambda^*) = 0$ i $T(x_s, x^*, \lambda^*) \neq 0$, wtedy (i tylko wtedy) następuje przejście do etapu drugiego.

Etap drugi rozpoczyna się zbudowaniem systemu $S(x)$ równoważnego $T(x)$ w następującym sensie:

a) $T(x) = 0 \Leftrightarrow S(x) = 0$,

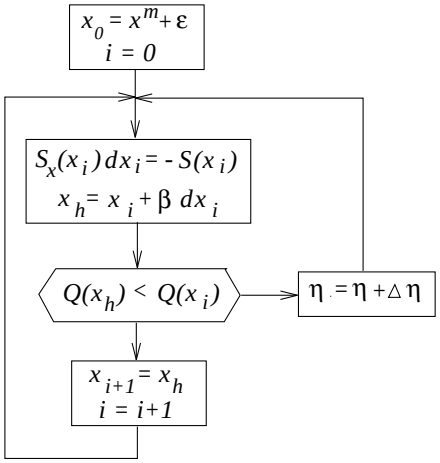
b) $T_x^{-1}(x_s)$ nie istnieje $\Rightarrow S_x^{-1}(x_s)$ istnieje.

System $S(x)$ definiowany jest poniższym równaniem:

$$S(x) = \frac{T(x)}{((x - x^m)^T (x - x^m))^\eta}, \tag{C.6}$$

gdzie x^m oznacza położenie *przesuwalnego bieguna* o mocy η . Nazwa *przesuwalny biegun* bierze się stąd, iż położenie tego bieguna zmienia się w trakcie dokonywanych obliczeń

[§]Moc bieguna, λ^* dobrana jest tak, że zachodzi: $T_x(x, x^*, \lambda^*) < 0$.



Rysunek C.3: Uproszczony schemat obliczania η podczas szukania zer $S(x)$.

(w chwili początkowej $x^m = x_k$). Kolejne położenia x^m oraz wartości η wyznaczane są w przedstawiony niżej sposób.
Na początek, startując z $\eta = 0$ oblicza się gradient funkcji $S(x)$:

$$S_x(x) = \frac{1}{((x - x^m)^T(x - x^m))^\eta} \cdot \left(f_x(x) - \frac{2\eta(x - x^m)f(x)}{(x - x^m)^T(x - x^m)} \right) \tag{C.7}$$

dla $x_0 = x^m + \epsilon$. Następnie z równań:

$$\begin{aligned} &S_x(x_i)\ dx_i = -S(x_i), \\ &x_{i+1} = x_i + \beta\ dx_i \quad , \quad 0 < \beta \leq 1, \end{aligned} \tag{C.8}$$

oblicza się nowe, hipotetyczne położenie punktu $x_h = x_{i+1}$. Jeśli kryterium $Q(x) = S(x)^T S(x)$ dla x_h nie zmniejsza się, tzn. jeśli $Q(x_h) \geq Q(x_i)$, powyższe obliczenia powtarza się dla zwiększonej o $\triangle\eta$ wartości η ($\eta = \eta + \triangle\eta$). Dopiero wtedy, gdy $Q(x_h) < Q(x_i)$, x_h i η mogą być wykorzystane w następnym kroku iteracji (zobacz rysunek C.3).

Podczas wykonywania obliczeń może dojść do sytuacji, w której $(x-x^m)^T(x-x^m) > 1$, co w połączeniu z dużymi wartościami η sprawia, że $S_x(x) \approx 0$. Powoduje to znaczne zmniejszenie stopnia zbieżności algorytmu (C.8). Stopień ten można zwiększyć, przesuwając biegun x^m do znalezionego w czasie ostatniej iteracji punktu x_i. Wraz z przesunięciem x^m w nowe miejsce, parametr η powinien zostać wyzerowany. Po takich zmianach, algorytm (C.8) wykaże się większą zbieżnością.

Warunki stopu. Warunki kończące fazę minimalizacji i fazę tunelowania wynikają bezpośrednio z natury użytych w nich algorytmów. W przypadku minimalizacji obliczenia kończy się z chwilą dotarcia w okolice minimum lokalnego (rozpoznawanego po wartości gradientu funkcji f lub po fakcie przekroczenia zadanego limitu ilości wykonanych iteracji). W przypadku tunelowania warunkiem stopu jest osiągnięcie okolicy zera funkcji $S(x)$ (lub też przekroczenie ograniczenia na ilość iteracji). Niestety, połączenie tych dwóch faz w jeden algorytm sprawia, ze podanie warunku stopu nie jest już tak oczywiste. Globalna minimalizacja jest bowiem problemem o wiele szerszym i trudniejszym niż poszukiwanie zer czy też minimów lokalnych. Zgodnie z sugestią autorów algorytmu, warunek stopu globalnej minimalizacji mógłby zależeć od ilości wykonanych iteracji. Jeśli

np. w najdłuższej fazie tunelowania wykonano N iteracji, przekroczenie w kolejnym cyklu algorytmu $10N$ iteracji mogłoby być takim warunkiem. Dokładniejsze szczegóły związane z implementacją algorytmu tunelowego (dla klasycznej i exponencjalnej funcji tunelowej) zamieszczone są w [GB91].

Literatura

[Ade86] Adept Technology, Inc., Sunnyvale, USA. *Adept Manipulator System, Installation Manual*, 1986.

[Ade88] Adept Technology, Inc., San Jose, USA. *Adept Manipulator System, V Reference Guide, version 6.2*, 1988.

[AS86] H. Asada i J.-J.E. Slotin. *Robot Analysis and Control*. Wiley-Interscience, New York, 1986.

[BA91] D. Ben-Arieh. Concurrent modeling and simulation of multi-robot systems. *Robotics & Computer-Integrated Manufacturing*, **8**(2):67–73, 1991.

[BDJ+94] B. Buchberger, S. Dreiseitl, W. Jacak, T. Kubik, R. Muszyński i D. Schlosser. RISC HYROB: The hybrid evolutionary method and system for itelligent robot control. Technical report, RWCP RISC-Linz, 1994.

[BGC94] L. Behera, M. Gopal i S. Chaudhury. Trajectory tracking of robot manipulator using gaussian networks. *Robotics and Autonomous Systems*, **13**:107–115, 1994.

[BHW88] D.D Bedworth, M.R. Henderson i P.M. Wolfe. *Computer-integrated design and manufacturing*. McGraw-Hill, Inc., 1988.

[BM91] J.-H. Borm i C.-H. Menq. Determination of optimal measurement configurations for robot calibration based on observability measure. *International Journal of Robotics Research*, **10**(1):51–63, 1991.

[Buc65] B. Buchberger. *An Algorithm for Finding a Basis for the Residue Class Ring of a Zero-dimentional Polynomial Ideal*. PhD thesis, Uniiversity of Innsbruck, Math. Inst., Austria, 1965. (in German).

[Buc85] B. Buchberger. Gröbner bases:an algorithmic method in polynomial ideal theory. W N.K. Bose, wydawca, *Multidimentional Systems Theory*, strony 184–232. D.Reidel Publishing Company, 1985.

[BW86] C. Blume i Jakob W. *Programming Languages for Industrial Robots*. Springer-Verlag, 1986.

[cH92] A. Çela i Y. Hamam. Optimal motion planning of multiple-robot system based on decomposition coordination. *IEEE Transactions on Robotics and Automation*, **8**(5):585–596, October 1992.

[CK88] C. Chevallereau i W. Khalil. A new method for the solution of the inverse kinematics of redundant robots. W *Proceedings of the IEEE Conference on Robotics and Automation*, strony 37–42. Computer Society Press, 1988.

[CL89] E. Cheung i V. Lumelsky. Development of sensitive skin for 3d robot arm operating in an uncertain environment. W *Proceedings of the IEEE Conference on Robotics and Automation*, strony 1056–1061. Computer Society Press, 1989.

[Cra81] J. J. Craig. *Introduction to Robotics*. MIT Press, Cambridge, 1981.

[CS91] X. Cui i K.G. Shin. Intelligent coordination of multiple systems with neural networks. *IEEE Transactions on Systems, Man, and Cybernetics*, **21**(6):1488–1496, November,December 1991.

[CSE91] S. Chiaverini, B. Siciliano i O. Egeland. Redundancy resoluiton for the human-arm-like manipulator. *Robotics and Autonomous Systems*, **8**:239–250, 1991.

[CTL94] R.H.T. Chan, P.K.S. Tam i D.N.K. Leung. Solving the motion planning problem by using neural networks. *Robotica*, **12**:323–333, 1994.

[DH55] J. Denavit i R.S. Hartenberg. A kinematic notation for lower-pair mechanisms based on matrices. *ASME Transaction Journal of Applied Mechanics*, **77**(2):215–221, 1955.

[DJKM95] S. Dreiseitl, W. Jacak, T. Kubik i R. Muszyński. Neural processing-based robot kinematics modeling and calibration for pose control. W *Proceedings of the 12th International Conference on Systems Science*, strony 288–295, Wroclaw, Poland, 1995.

[DJM95] I. Dulęba, W. Jacak i R. Muszyński. Hybrid computing approach to robot control in the presence of uncertainties. W *Proceedings of the International Federation of Automatic Control Conference on Motion Control*, strony 664–671, Munich, Germany, 1995.

[DJMB95] I. Dulęba, W. Jacak, R. Muszyński i Bruno Buchberger. Symbolic computation-based synthesis of neural network dynamics and controller for robots. W *Proceedings of the IEEE International Conference on Neural Networks (ICNN'95)*, strony 2720–2725, Perth, Australia, 1995.

[DK94] S. Dreiseitl i T. Kubik. Neural-processed inverse kinematics of robot manipulators. W *Proceedings of the Third International Conference on Automation, Robotics, and Computer Vision*, volume **3**, strony 1748–1751, Singapore, 1994.

[DMB93] K.L. Doty, C. Melchiorri i C. Bonivento. A theory of generalized inverses applied to robotics. *International Journal of Robotics Research*, **12**(1):1–19, February 1993.

[DS91] G. Duelen i K. Schröer. Robot calibration—method and results. *Robotics and Computer-Integrated Manufacturing*, **8**(4):223–231, 1991.

[DSS88] H. Das, J-J.E. Slotine i T.B. Sheridan. Inverse kinematics algorithms for redundant systems. W *Proceedings of the IEEE Conference on Robotics and Automation*, strony 43–48. Computer Society Press, 1988.

[Fag94] R. Faglia. The application to robotics of a new inversion method for non linear systems. W *Proceedings of the Third International Conference on Automation, Robotics, and Computer Vision*, volume **2**, strony 987–991, Singapore, 1994.

[GB91] S. Gomez i C. Barron. The expotential tunneling method. Reportes de Investigatión, **Vol.1**, No.3, IIMAS-UNAM, Mexico, 1991.

[GDQ92] X. Gao, D. Dawson i Z. Qu. On the robust control of two manipulators holding a rigid object. *Jurnal of Intelligent and Robotic Systems*, **8**:107–119, 1992.

[GF89] E.G Gilbert i C.P. Foo. Computing the distance between smooth objects in three dimentional space. W *Proceedings of the IEEE Conference on Robotics and Automation*, strony 158–163. Computer Society Press, 1989.

[GH89] E.G Gilbert i S.M. Hong. A new algorithm for detecting the collision of moving objects. W *Proceedings of the IEEE Conference on Robotics and Automation*, strony 8–14. Computer Society Press, 1989.

[GL82] S. Gomez i A. V. Levy. The tunneling method for solving the constrained global optimization problem with several non-connected feasible regions. *Lecture Notes in Mathematics*, (909):34–47, 1982.

[Hay86] V. Hayward. Fast collision detection scheme by recursive decomposition of a manipulator workspace. W *Proceedings of the IEEE Conference on Robotics and Automation*, strony 1044–1049. Computer Society Press, 1986.

[Jac89] W. Jacak. Strategies for searching collision free manipulators motions: Automata theory approach. *Robotica*, **7**(4):128–142, 1989.

[Jac91] W. Jacak. *Roboty inteligentne. Metody planowania działań i ruchów.* Prace Naukowe Instytutu Cybernetyki Technicznej P.Wr. 85, Seria: Monografie nr 18, Wydawnictwo Politechniki Wrocławskiej, Wrocław, 1991.

[JBS96] W. Jacak, B. Buchberger i S. Stifter. Intelligent systems combining reactive and learning capabilities. W *Proceedings of the 13th European Meeting on Cybernetics and Systems Research (EMCSR'96)*, Vienna, Austria, 1996. To appear.

[JDKS95] W. Jacak, S. Dreiseitl, T. Kubik i D. Schlosser. Distributed planning and control of intelligent robot's arm motion based on symbolic and neural processing. W *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics (SMC'95)*, volume **3**, strony 2898–2903, Vancouver, Canada, 1995.

[JN94] A. Jazidie i M. Nagamachi. Distributed planning of end-effectors' trajectories for multi-arm robots. W *Proceedings of the Third International Conference on Automation, Robotics, and Computer Vision*, volume **3**, strony 1213–1217, Singapore, 1994.

[Jor92] M.S. Jordan. Forward models: Supervised learning with distal teacher. *Cognitive Science*, **16**:307–354, 1992.

[KB91] W. Khalil i F. Bennis. Automatic generation of the inverse geometric model of robots. *Robotics and Autonomous Systems*, **7**:47–56, 1991.

[KH83] C.A. Klein i C.-H. Huang. Review of pseudoinverse control for use with kinematically redundant manipulators. *IEEE Transactions on Systems, Man, and Cybernetics*, **13**(3):245–250, March,April 1983.

[KH89] S.-Y. Kung i J.-N. Hwang. Neural network architectures for robotic applications. *IEEE Transactions on Robotics and Automation*, **5**(5):641–657, October 1989.

[Kha86] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *International Journal of Robotics Research*, **5**(1):90–98, 1986.

[Kir84] N. Kirćanski. *An Approach to mathematical modelling and control of manipulation robots.* PhD thesis, Belgrade University, 1984.

[KK95] L. Kleeman i R. Kuc. Mobile robot sonar for target localization and classification. *International Journal of Robotics Research*, **14**(4):295–318, August 1995.

[KM96] T. Kubik i R. Muszyński. Inteligentne, reaktywne sterowanie robotem działającym w naturalnym otoczeniu. W *V Krajowa Konferencja Robotyki. Prace naukowe ICT PWr. Nr 95, seria: konferencje*, volume **1**, strony 116–123, Wrocław, 1996. Oficyna wydawnicza PWr.

[Kod87] D.E. Koditschek. Exact robot navigation by means of potential functions: Some topological considerations. W *Proceedings of the IEEE International Conference on Robotics and Automation*, strony 1–6, 1987.

[KP93] M.V. Kirćanski i T.M. Petrović. Combined analytical-pseudoinverse inverse kinematics solution for simple redundant manipulators and singularity avoidance. *International Journal of Robotics Research*, **12**(2):188–196, April 1993.

[Kup87] M. Kuperstein. Adaptive visual motor coordination in multijoint robots using parallel architecture. W *Proceedings of the IEEE International Conference on Robotics and Automation*, strony 1595–1602, Releigh, North Carolina, 1987.

[KV88] P. Khosla i R. Volpe. Superquadric artificial potentials for obstacle avoidance and approach. W *Proceedings of the IEEE Conference on Robotics and Automation*, strony 1778–1784. Computer Society Press, 1988.

[KV91] R. Kuc i V.B. Viard. A physically based navigation strategy for sonar-guided vehicles. *International Journal of Robotics Research*, **10**(2):75–87, April 1991.

[KVTK93] N. Kirćanski, M. Vukobratović, A. Timćenko i M. Kirćanski. Symbolic modeling in robotics: Genesis, application and future prospects. *Jurnal of Intelligent and Robotic Systems*, **8**:1–19, 1993.

[Lat91a] J.C. Latombe. *Motion Planning: Theory and Applications.* In *Computer Applications in Production and Engineering: Integration Aspects* G. Doumeingts, J. Brown, M. Tomljanovich (ed.), Elsevier Science Publishers B.V., North-Holland, 1991.

[Lat91b] J.C. Latombe. *Robot Motion Planning.* Kluwer Academic Publications, Boston, Dordrecht, 1991.

[LB91] S. Lee i G.A. Bekey. Applications of neural networks to robotics. *Control and Dynamic Systems*, **39**:1–69, 1991.

[LI95] B.-L. Lu i K. Ito. Regularization of inverse kinematics for redundant manipulators using neural network inversion. W *Proceedings of the IEEE International Conference on Neural Networks*, strony 2726–2731, Perth, Australia, 1995.

[Lie77] A. Liegeois. Automatic supervisory control of the configuration and behavior of the multibody machanisms. *IEEE Transactions on Systems, Man, and Cybernetics*, **7**(12):868–871, December 1977.

[LK90] R. Lee i R.M. Kil. Robot kinematic control based on bi-directional mapping neural network. W *Proceedings of the International Joint Conference on Neural Networks*, volume **3**, strony 327–335, San Diego, California,USA, June 1990.

[LK94] S. Lee i R.M. Kil. Redundant arm kinematic control with recurrent loop. *Neural Networks*, **7**(4):643–659, 1994.

[LM80] A. V. Levy i A. Montavlo. A modification to the tunneling algorithm for finding the global minima of an arbitrary one dimentional scalar function. Communications Technicas, Serie Naranja, No. 240, IIMAS-UNAM, Mexico, 1980.

[LMGC82] A. V. Levy, A. Montavlo, S. Gomez i A. Calderon. Topics in global optimization. *Lecture Notes in Mathematics*, **909**:18–33, 1982.

[Lon92] M.K. Long. Task-directed inverse kinematics for redundant manipulators. *Jurnal of Intelligent and Robotic Systems*, **6**:241–261, 1992.

[LP86] T. Lozano-Perez. Motion planning for simple robot manipulators. W *Proceedings of the III International Symposium on Robotics Research*, strony 133–140, 1986.

[LT91] C.-F. Lin i W.-H. Tsai. Motion planning for multiple robots with multi-mode operations via disjunctive graphs. *Robotica*, **9**:393–408, 1991.

[Luh92] J.Y.S. Luh. Redundancy and coordination of multiple robots. *Journal of Intelligent and Robotic Systems*, **6**:95–105, 1992.

[Lum85] V.J. Lumelsky. On fast computation of distance between line segments. *Information Processing Letters*, (21):55–61, 1985.

[Mac89] A.A. Maciejewski. Kinetic limitation on the use of redundancy in robotic manipulators. W *Proceedings of the IEEE Conference on Robotics and Automation*, strony 113–118. Computer Society Press, 1989.

[Mey86] W. Meyer. Distances between boxes: Applications to collision detection and clipping. W *Proceedings of the IEEE Conference on Robotics and Automation*, strony 597–602. Computer Society Press, 1986.

[MK89] A.A. Maciejewski i C.A. Klein. The singular value decomposition: Computaiton and applications to robotics. *International Journal of Robotics Research*, **8**(6):63–79, 1989.

[MOB94] C. Mavroidis, F.B. Ouezdou i P. Bidau. Inverse kinematics of six-degree of freedom "general" and "spatial" manipulators using symbolic computation. *Robotica*, **12**:421–430, 1994.

[MRD91] B.W. Mooring, Z.S. Roth i M.R. Driels. *Fundamentals of Manipulator Calibration*. J. Wiley & Sons, 1991.

[MS94] R. Muszyński i D. Schlosser. Neural network representation of robot environment and its application to distance calculation. W *Proceedings of the Third International Conference on Automation, Robotics, and Computer Vision (ICARCV'94)*, volume **2**, strony 1324–1328, Singapore, 1994.

[MSB91] H. Mühlebein, M. Schomisch i J. Born. The parallel genetic algorithm as function optimizer. *Parallel Computing 17*, (17):619–632, 1991.

[Nen92] D.N. Nenchev. Restricted jacobian matrices of redundant manipulators in constrained motion tasks. *International Journal of Robotics Research*, **11**(6):584–597, December 1992.

[PAvC93] H. Peremans, K. Audenaert i J.M. van Campenhout. A high-resolution sensor based on tri-aural perception. *IEEE Transactions on Robotics and Automation*, **9**(1):36–48, February 1993.

[Pie68] D. L. Pieper. The kinematics of manipulators under computer control. Technical Report Memo AIM 72, Stanford Artificial Intelligence Laboratory, Stanford CA, 1968.

[Pot91] V. Pothonjak. New approach to the application of redundant robots. *Robotics & Computer-Integrated Manufacturing*, **8**(3):181–185, 1991.

[PYK94] E. Paljug, X. Yun i V. Kumar. Control of rolling contacts in multi-arm manipulation. *IEEE Transactions on Robotics and Automation*, **10**(4):441–452, August 1994.

[RCPD95] A. Ramdane-Cherif, V. Perdereau i M. Drouin. Optimization schemes for learning the forward and inverse kinematics equations with neural network. W *Proceedings of the IEEE International Conference on Neural Networks*, strony 2732–2737, Perth, Australia, 1995.

[RG94] W.E. Red i S.-W. Gong. Automated inverse-kinematics for robot off-line programming. *Robotica*, **12**:45–53, 1994.

[RMR94] H. Regendova, M. Markechova i J. Regenda. Time-suboptimal quasi-continous path generation for industrial robots. *Robotics and Autonomous Systems*, **13**:117–125, 1994.

[San94] T. D. Sanger. Neural network learning control of robot manipulators using gradually increasing task difficulty. *IEEE Transactions on Robotics and Automation*, **10**(3):323–333, June 1994.

[Sas94] S. Sasaki. On numerical techniques for kinematics problems of general serial-link robot manipulators. *Robotica*, **12**:309–322, 1994.

[Sat93] K. Sato. Deadlock-free motion planning using the laplace potential field. *Advanced Robotics*, **7**(5):449–461, 1993.

[Sch81] J.T. Schwartz. Finding the minimum distance between two convex polygons. *Information Processing Letters*, **13**(4,5):168–170, 1981.

[Ser93] H. Seraji. The configuration control approach. *IEEE Transactions on Robotics and Automation*, **9**(2):125–139, April 1993.

[SH92] C.A. Shaffer i G.M. Herb. A real-time robot arm collision avoidance system. *IEEE Transactions on Robotics and Automation*, **8**(2):149–160, April 1992.

[SI91] D. Simon i C. Isik. Optimal trigonometric robot joint trajectories. *Robotica*, **9**:379–386, 1991.

[SK92] S.-H. Suh i M.-S. Kim. An algebraic approach to collision-avoidance trajectory planning for dual-robot systems: Formulation and optimization. *Robotica*, **10**:173–182, 1992.

[SS88] L. Sciavicco i B. Siciliano. A solution algorithm to the inverse kinematic problem for redundant manipulators. *IEEE Transaction on Robotics and Automation*, **4**(4):403–410, August 1988.

[SV89] M.W. Spong i .Vidyasagar. *Robot Dynamics and Control*. J.Wiley & Sons, New York, 1989.

[SZ92] K.G. Shin i Q. Zheng. Minimum-time collision-free trajectory planning for dual-robot systems. *IEEE Transactions on Robotics and Automation*, **8**(5):641–644, October 1992.

[Too89] R.W. Toogood. Efficient robot inverse and direct dynamics algorithms using microcomputer based symbolic generation. W *Proceedings of the IEEE Conference on Robotics and Automation*, strony 1827–1832. Computer Society Press, 1989.

[Tra] Transition Research Corporation. *SonaRanger, User Manual, Release 4Da.*

[VK85] M. Vukobratović i N. Kirćanski. Computer assisted generation of robot dynamics models in an analytical form. *Acta Applied Mathematics*, (3):43–70, 1985.

[WCM93] J.H. Won, B.W. Choi i M.J.Chung. A unified approach to the inverse kinematics solution for a redundant manipulator. *Robotica*, **11**:159–165, 1993.

[WFM91] I.D. Walker, R.A. Freeman i S.I. Marcus. Analysis of motion and internal loading of objects grasped by multiple cooperating manipulators. *International Journal of Robotics Research*, **10**(4):396–409s, 1991.

[Whi69] D.E. Whitnej. Resolved motion rate control of manipulators and human prosthesis. *IEEE Transactions on Man Machine Systems*, **10**(2):47–53, 1969.

[WHY88] C.-H. Wu, J. Ho i K.-Y. Young. Design of robot accuracy compensator after calibration. W *Proceedings of the IEEE Conference on Robotics and Automation*, strony 780–785. Computer Society Press, 1988.

[Wil90] H. William. *Numerical recipies in C.* Cambridge Univ. Press, 1990.

[WM88] I.D. Walker i S.I. Marcus. Subtask performance by redundancy resolution for redundant robot manipulators. *IEEE Journal of Robotics and Automation*, **4**(3):350–353, June 1988.

[YB89] D.T. Yeung i G.A. Bekey. Using a context-sensitive learning network for robot arm control. W *Proceedings of the IEEE International Conference on Robotics and Automation*, strony 1441–1447, 1989.

[YI95] Z. Ying i S. S. Iyengar. Robot reachability problem: a nonlinear optimization approach. *Jurnal of Intelligent and Robotic Systems*, **12**:87–100, 1995.

[Yos84] T. Yoshikawa. Analysis amd control of robotic manipulators with redundancy. W Brady i Paul, wydawca, *Robotics Research: the First Int. Symp.*, strony 735–748. Cambridge MIT Press, 1984.

[ZD88] J. Ziegert i P. Datseris. Basic considerations for robot calibration. W *Proceedings of the IEEE International Conference on Robotics and Automation*, strony 932–938. Computer Society Press, Washington, 1988.