STUDIA INFORMATICA 2006
Volume 27 Number 2 (67)

Rafał SZKOWRON, Andrzej HARLECKI
Akademia Techniczno-Humanistyczna w Bielsku-Białej, Katedra Mechaniki i Informatyki

KOMPUTEROWE WSPOMAGANIE NAUCZANIA Z ZAKRESU ZASTOSOWAŃ SIECI NEURONOWYCH I ALGORYTMÓW GENETYCZNYCH DO ROZWIĄZYWANIA ODWROTNEGO ZADANIA KINEMATYKI MANIPULATORÓW

Streszczenie. W pracy przedstawiono możliwości przygotowanego programu komputerowego o nazwie „NeuroManip", w którym wykorzystuje się sieć neuronową jako narzędzie do rozwiązywania odwrotnego zadania kinematyki manipulatorów. W proponowanej metodzie do „uczenia" przyjętej sieci używa się algorytmu genetycznego. Jako przykład rozważono „hipotetyczny", przestrzenny, trójczłonowy manipulator o sześciu stopniach swobody. Przygotowany program komputerowy daje również możliwość alternatywnego rozwiązania odwrotnego zadania kinematyki analizowanego manipulatora, przy wykorzystaniu, opartej na odmiennym formalizmie, „metody jakobianu pseudoodwrotnego".

Słowa kluczowe: komputerowy program wspomagający nauczanie, sieć neuronowa, algorytm genetyczny, manipulator, odwrotne zadanie kinematyki

COMPUTER AIDED TEACHING REGARDING APLLICATIONS OF NEURAL NETWORKS AND GENETIC ALGORITHMS FOR SOLVING INVERSE KINEMATICS PROBLEM OF MANIPULATORS

Summary. In this paper possibilities of the prepared computer program, called "Neuromanip", has been presented. In this program, a neural network is used to solve the inverse kinematics problem of manipulators. For training the neural network assumed, the genetic algorithm has been used. As an example, a chosen "hypothetical", spatial, three-links manipulator with 6 degrees-of-freedom has been analyzed. The program prepared enables also solving the inverse kinematics problem of the manipulator analyzed by using the "pseudoinverse Jacobian method" based on different formalism.

Keywords: computer program aided teaching, neural network, genetic algorithm, manipulator, inverse kinematics problem

1. Wiadomości wstępne

Szczegółowy opis sieci neuronowych, jako gałęzi sztucznej inteligencji, oraz sposobu ich działania i „uczenia" można znaleźć w wielu publikacjach, wśród których wymienić można także krajowe monografie [10, 13, 18].

Sieci neuronowe znajdują szerokie zastosowanie w różnych dziedzinach techniki, w tym głównie w elektrotechnice i automatyce, a także w medycynie. Spośród wielu implementacji należy wymienić przede wszystkim te, w których sieć neuronową zastosowano jako element generujący sygnały sterujące w różnych urządzeniach i mechanizmach. Szczególnie często sieci neuronowe używane są do sterowania ruchem członów manipulatorów robotów [1, 3, 4, 7, 8, 9, 11, 12, 14, 16, 17].

W przypadku rozwiązywania prostego zadania kinematyki (forward kinematics problem) manipulatorów [2] znane są wartości współrzędnych konfiguracyjnych (złączowych), opisujących ruch względny sąsiednich członów rozważanego manipulatora, natomiast określa się położenie, czyli pozycję i orientację, jego członu roboczego (tzw. efektora).

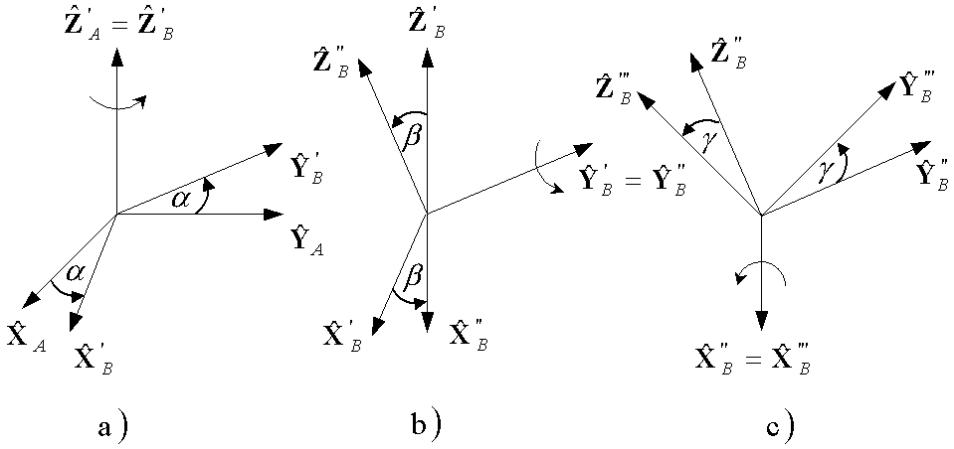
W przypadku rozwiązywania odwrotnego zadania kinematyki (inverse kinematics problem) manipulatorów [2, 6, 15] znane jest położenie efektora, natomiast wyznacza się wszystkie możliwe zbiory współrzędnych złączowych, gwarantujących uzyskanie tego położenia. W porównaniu z prostym zadaniem kinematyki wydaje się być ono trudniejsze ze względu na wielokrotność rozwiązań, wynikającą z faktu, iż istnieje kilka możliwych wariantów położeń członów manipulatora, odpowiadających określonemu położeniu efektora. Warunkiem koniecznym istnienia rozwiązania odwrotnego zadania kinematyki jest to, aby efektor, o zadanym położeniu, znajdował się w tzw. przestrzeni roboczej manipulatora [2].

2. Macierze transformacji na bazie kątów Eulera

Wzajemne położenie dwóch układów współrzędnych może być określone za pomocą tzw. macierzy transformacji jednorodnych, np. popularnej w robotyce macierzy Denavita--Hartenberga [2], elementy której buduje się w oparciu o cztery parametry geometryczne.

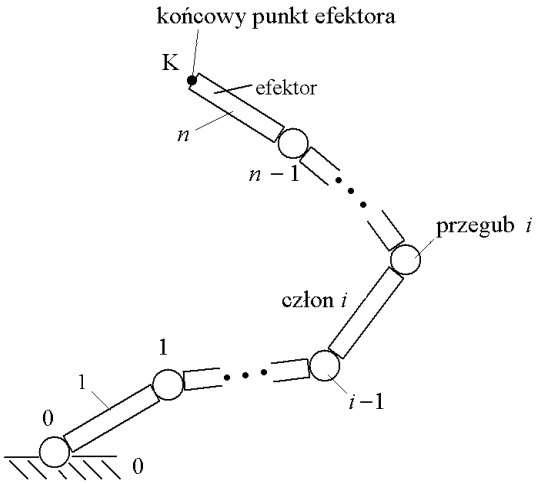
W opracowanym programie zaproponowano jednakże inny sposób budowania macierzy transformacji, bazujący na wykorzystaniu kątów Eulera (rys. 1) [2]. Z analizy przedstawionego rysunku widać, że rozważono tu przypadek wzajemnego położenia dwóch, utworzonych przez wersory, prawoskrętnych układów współrzędnych $\hat{\mathbf{X}}_B\hat{\mathbf{Y}}_B\hat{\mathbf{Z}}_B$ i $\hat{\mathbf{X}}_A\hat{\mathbf{Y}}_A\hat{\mathbf{Z}}_A$, o wspólnym początku, mających początkowo tę samą orientację. Końcowe położenie układów uzyskano dokonując ich kolejnych obrotów, przy czym – w zaproponowanym podejściu – każdy obrót jest wykonywany wokół osi (wersorów) układu $\hat{\mathbf{X}}_B\hat{\mathbf{Y}}_B\hat{\mathbf{Z}}_B$ (ruchomego), natomiast osie układu $\hat{\mathbf{X}}_A\hat{\mathbf{Y}}_A\hat{\mathbf{Z}}_A$ pozostają

nieruchome. Ze względu na fakt, iż trzy obroty wykonywane są kolejno wokół osi $\hat{\mathbf{Z}}_B, \hat{\mathbf{Y}}_B, \hat{\mathbf{X}}_B$ przedstawiony sposób określa się jako obroty o kąty Eulera $\hat{\mathbf{Z}} - \hat{\mathbf{Y}} - \hat{\mathbf{X}}$. Należy ponadto zwrócić uwagę, że każdy drugi i trzeci obrót wykonywany jest wokół osi, których aktualne położenie zależy od poprzednio wykonanego obrotu (obrotów).



Rys. 1. Kąty Eulera $\hat{\mathbf{Z}} - \hat{\mathbf{Y}} - \hat{\mathbf{X}}$
Fig. 1. Euler's angles $\hat{\mathbf{Z}} - \hat{\mathbf{Y}} - \hat{\mathbf{X}}$

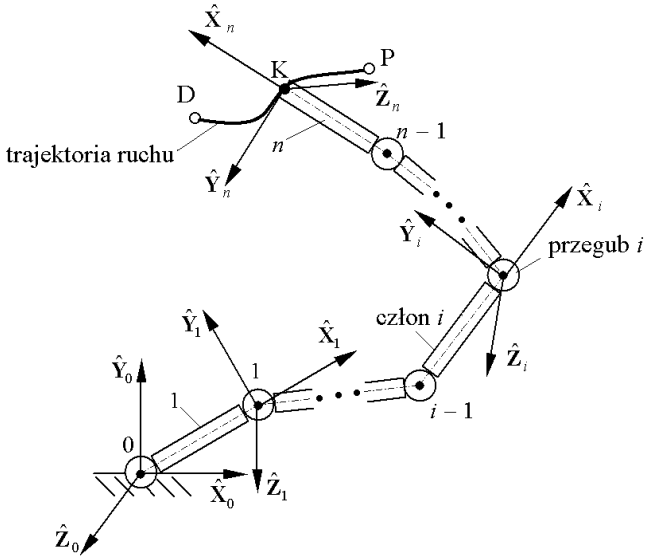
W przyjętej metodzie liczba członów manipulatora może być (teoretycznie) dowolna i mogą być one połączone ze sobą przegubami kulistymi (tworzą zatem pary kinematyczne III klasy). Ostatni człon, ze swobodnym końcem (końcowym punktem K), traktowany jest jako efektor. Wszystkie przeguby numeruje się kolejno, począwszy od przegubu 1 – najbliższego nieruchomej podstawy (rys. 2), oznaczonej numerem 0.



Rys. 2. Numerowanie członów i przegubów manipulatora
Fig. 2. Numbering of links and joints of manipulator

W środku przegubu 0 umieszcza się początek bazowego układu współrzędnych. Osie tego układu, tak jak osie pozostałych (lokalnych) układów współrzędnych, związanych z poszczególnymi członami, utworzone są przez prawoskrętny układ wersorów – w tym przypadku $\hat{\mathbf{X}}_0 \hat{\mathbf{Y}}_0 \hat{\mathbf{Z}}_0$.

Zakłada się przy tym, że orientacja wersorów tego układu może być dowolna. W punktach stanowiących środki wszystkich pozostałych przegubów, a także w końcowym punkcie K efektora, umieszcza się początki lokalnych układów współrzędnych, utworzonych przez wersory o numerach, zgodnych z numerem adekwatnego przegubu. Przyjmuje się, że wersory $\hat{\mathbf{X}}_i$ tych układów są skierowane wzdłuż prostych, przechodzących przez środki przegubów $i-1$ oraz i, które łączą człon i z sąsiednimi członami, w stronę od przegubu $i-1$ do przegubu i (rys. 3).



Rys. 3. Przyjęte układy współrzędnych
Fig. 3. Coordinate systems assumed

Aby określić pozycję i orientację dwóch sąsiednich układów współrzędnych, począwszy od podstawy do efektora, w przyjętej metodzie, buduje się macierze transformacji $\mathbf{T}_i$ (z układu $\hat{\mathbf{X}}_i\hat{\mathbf{Y}}_i\hat{\mathbf{Z}}_i$ do układu $\hat{\mathbf{X}}_{i-1}\hat{\mathbf{Y}}_{i-1}\hat{\mathbf{Z}}_{i-1}$) [15], bazujące na kątach Eulera, a następnie dokonuje się „złożenia" tych macierzy, korzystając z iloczynu:

$$\mathbf{K}(\alpha,\beta,\gamma)=\prod_{i=1}^{n}\mathbf{T}_i\,(\alpha_i,\beta_i,\gamma_i)\,, \tag{1}$$

gdzie: $\alpha_i,\beta_i,\gamma_i$ – kąty Eulera,

$$\mathbf{T}_i=\begin{bmatrix}\cos\alpha_i\cos\beta_i & \cos\alpha_i\sin\beta_i\sin\gamma_i-\sin\alpha_i\cos\gamma_i & \cos\alpha_i\sin\beta_i\cos\gamma_i+\sin\alpha_i\sin\gamma_i & l_i\cos\alpha_i\cos\beta_i\\ \sin\alpha_i\cos\beta_i & \sin\alpha_i\sin\beta_i\sin\gamma_i+\cos\alpha_i\cos\gamma_i & \sin\alpha_i\sin\beta_i\cos\gamma_i-\cos\alpha_i\sin\gamma_i & l_i\sin\alpha_i\cos\beta_i\\ -\sin\beta_i & \cos\beta_i\sin\gamma_i & \cos\beta_i\cos\gamma_i & -l_i\sin\beta_i\\ 0 & 0 & 0 & 1\end{bmatrix},$$

l_i – odległość środków przegubów i oraz $i+1$ (w przypadku gdy $i=n$ jest to odległość końcowego punktu K efektora od środka ostatniego przegubu $i-1$).

Pozycję efektora w bazowym układzie $\mathbf{X}_0\mathbf{Y}_0\mathbf{Z}_0$ (a zatem położenie początku związanego z nim układu współrzędnych $\hat{\mathbf{X}}_n\hat{\mathbf{Y}}_n\hat{\mathbf{Z}}_n$) określają elementy czwartej kolumny macierzy $\mathbf{K}(\alpha,\beta,\gamma)$, stanowiącej w istocie wektor $\mathbf{k}(\alpha,\beta,\gamma)$.

3. Budowa sztucznej sieci neuronowej

Z uwagi na sposób przemieszczania się końcowego punktu K efektora z początkowego punktu P trajektorii do jej punktu docelowego D (rys. 3), ogólnie rzecz biorąc, przyjęta sieć neuronowa powinna wariantowo umożliwiać realizację jednego z wymienionych przypadków:

1) gdy nie ma wymagań co do przebiegu trajektorii i czasu trwania ruchu końcowego punktu K efektora między początkowym punktem P i docelowym punktem D trajektorii,
2) gdy określona jest trajektoria końcowego punktu K efektora, ale nie ma wymagań co do czasu jego przemieszczania,
3) gdy oprócz wymagań co do przebiegu trajektorii, narzucone zostaną wymagania co do czasu przemieszczenia końcowego punktu K efektora.

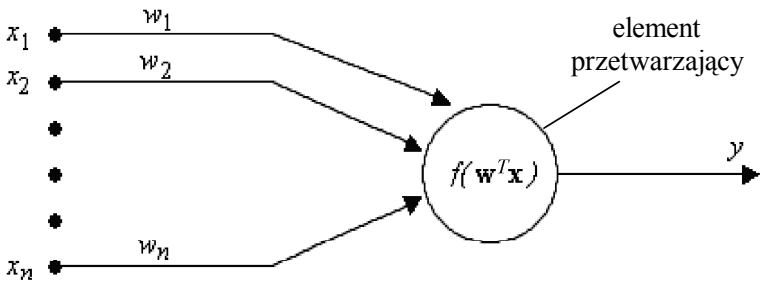
Przygotowując model (rys. 4) sztucznego neuronu, jako podstawowego elementu sieci neuronowej, należy przyjąć, że będzie się on składał z elementu przetwarzającego, który posiada n wejść x_i, gdzie $i = 1, 2, ..., n$ (każde scharakteryzowane przez tzw. wagę w_i, gdzie $i = 1, 2, ..., n$) oraz jedno wyjście y.

Zbiory sygnałów wejściowych x_i można przedstawić za pomocą, zaprezentowanego tu, w transponowanej postaci, wektora sygnałów wejściowych:

$$\mathbf{x}^T = [x_1, x_2, ..., x_n], \tag{2}$$

a zbiór wag za pomocą wektora wag:

$$\mathbf{w}^T = [w_1, w_2, ..., w_n]. \tag{3}$$



Rys. 4. Model sztucznego neuronu
Fig. 4. Model of artificial neuron

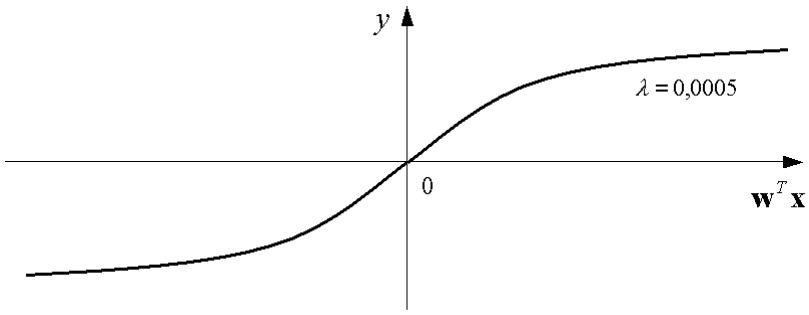
Sygnał wyjściowy y można wyrazić jako funkcję iloczynu skalarnego wektorów $\mathbf{w}^T$ i $\mathbf{x}$ (nazywaną funkcją aktywacji neuronu):

$$y = f(\mathbf{w}^T\mathbf{x}). \tag{4}$$

Spośród wielu rodzajów funkcji aktywacji [13, 18] autorzy programu wybrali sigmoidalną, bipolarną (tzn. przyjmującą zarówno wartości dodatnie, jak i ujemne) funkcję (rys. 5), określoną wzorem:

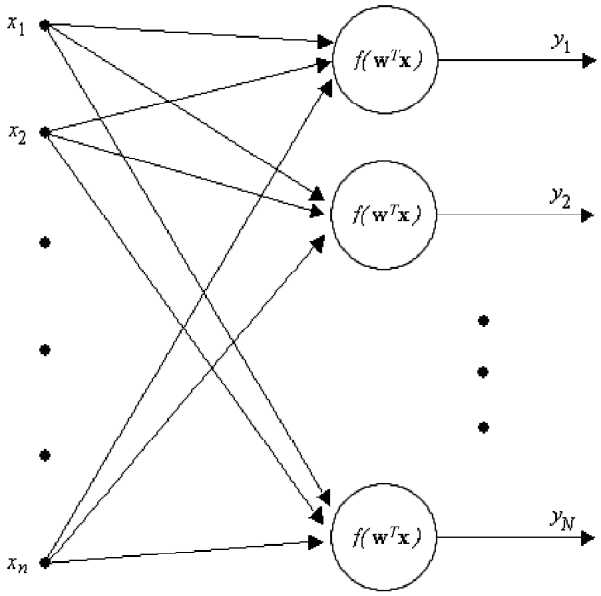
$$f\left(\mathbf{w}^T\mathbf{x}\right)=\frac{2}{1+e^{-\lambda\mathbf{w}^T\mathbf{x}}}-1. \tag{5}$$

gdzie: e – podstawa logarytmów naturalnych, λ – współczynnik nachylenia funkcji.



Rys. 5. Przyjęta postać sigmoidalnej, bipolarnej funkcji aktywacji neuronu
Fig. 5. Assumed form of sigmoidal, bipolar activation function of neuron

Pojedynczy neuron posiada ograniczone możliwości obliczeniowe – jego zadanie ogranicza się do tworzenia ważonej sumy sygnałów wejściowych i określenia, na jej podstawie, pojedynczego sygnału wyjściowego. Tworzy się więc warstwę składającą się z N neuronów (rys. 6). Wchodzące w jej skład neurony przetwarzają równocześnie n sygnałów, podanych na wejściu warstwy.



Rys. 6. Model warstwy neuronów
Fig. 6. Model of layer of neurons

Stwierdzono już, że z każdym neuronem związany jest pewien wektor wag. Zatem, zbiór wag związanych z całą warstwą można zapisać w postaci wektora:

$$\mathbf{w}_i^T=\left[w_{i,1},w_{i,2},...,w_{i,n}\right] \quad i = 1, 2, ..., N, \tag{6}$$

gdzie i – indeks kolejnego neuronu w rozważanej warstwie.

Wektor wyjściowy danej warstwy neuronów ma następującą postać:

$$\mathbf{y}^T = [y_1, y_2, ..., y_N]. \tag{7}$$

Wartości sygnałów wyjściowych (czyli wartości elementów wektora **y**) można obliczyć bezpośrednio ze wzoru:

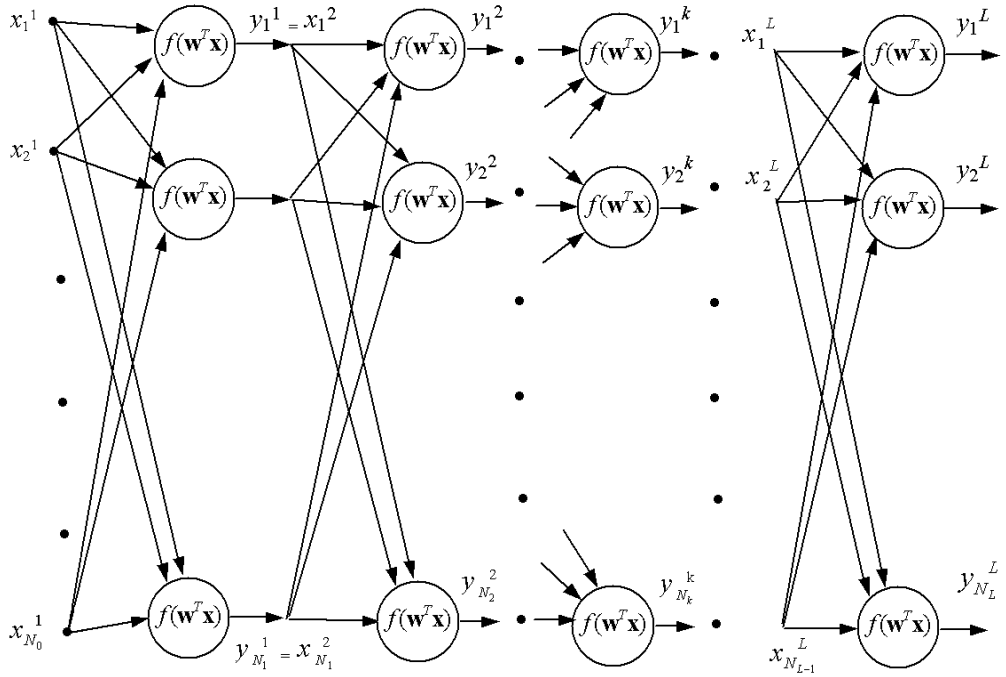
$$y_i = f(\mathbf{w}_i^T \mathbf{x}) \quad i = 1, 2, ..., N, \tag{8}$$

a po uszczegółowieniu:

$$y_i = f\left(\sum_{j=1}^{n} w_{i,j} x_j\right) \quad i = 1, 2, ..., N. \tag{9}$$

Jeżeli sygnały (9) potraktuje się jako sygnały wejściowe do następnej warstwy, otrzyma się sieć wielowarstwową.

Najczęściej używaną strukturą sieci jest taka, w której sygnały przekazywane są bezpośrednio z warstwy do warstwy, począwszy od pierwszej do ostatniej (sieć nazywana jest wówczas jednokierunkową, rys. 7).



Rys. 7. Model jednokierunkowej sieci wielowarstwowej (indeksy w przyjętych oznaczeniach odpowiadają numerom poszczególnych warstw)

Fig. 7. Model of one-way, multi-layer network (indices in notation assumed correspond to numbers of particular layers)

Pierwsza warstwa (przyjęto, że posiada ona N_1 elementów, a na jej wejściu podawanych jest N_0 sygnałów), której sygnały wejściowe są jednocześnie sygnałami wejściowymi do całej sieci, nazywana jest warstwą początkową. Ostatnia warstwa (przyjęto, że posiada numer L, wobec czego zawiera N_L elementów), podająca sygnały, będące odpowiedzią całej sieci, nosi nazwę warstwy wyjściowej. Warstwy pośrednie, o ile istnieją, stanowią tzw. warstwy ukryte.

Stosowane są również sieci neuronowe ze sprzężeniem zwrotnym, w których sygnały wyjściowe danej warstwy są równocześnie sygnałami wejściowymi do niej samej lub do warstw ją poprzedzających [13, 18].

Wektor wyjściowy sieci przyjmuje następującą postać:

$$\mathbf{y}^{L\,T} = \left[y_1^L,\ y_2^L,\ \ldots,\ y_{N_L}^L\right]. \tag{10}$$

Jego elementami są sygnały wyjściowe ostatniej warstwy (rys. 7).

Wektor wag związany z i-tym neuronem, znajdującym się w posiadającej N_k neuronów (gdzie $k = 1, 2, ..., L$) k-tej warstwie sieci, przedstawiono jako:

$$\mathbf{w}_i^{k\,T} = \left[w_{i,1}^k,\ w_{i,2}^k,\ \ldots,\ w_{i,N_{k-1}}^k\right] \quad i = 1, 2, ..., N_k. \tag{11}$$

Sygnały $y_1^L, y_2^L, \ldots, y_{N_L}^L$ są rzeczywistymi sygnałami wysyłanymi przez sieć. Tymczasem, na wyjściu sieci „oczekuje się" sygnałów wyjściowych o konkretnych (wymaganych) wartościach, które można przedstawić jako elementy wektora, „oczekiwanych" sygnałów wyjściowych sieci:

$$\mathbf{d}^T = \left[d_1,\ d_2,\ \ldots,\ d_{N_L}\right]. \tag{12}$$

Zatem, każdemu wektorowi N_0 sygnałów wejściowych sieci:

$$\mathbf{x}^T = \left[x_1,\ x_2,\ \ldots,\ x_{N_0}\right] \tag{13}$$

można przyporządkować wektor „oczekiwanych" sygnałów wyjściowych **d**. Mówi się, że tworzy się w ten sposób jeden „punkt uczący". Można oczywiście wygenerować dowolną liczbę m „punktów uczących".

Dysponując zbiorem „punktów uczących", przystępuje się do procesu doboru wag związanych z poszczególnymi neuronami, zwanego procesem „uczenia" sieci. Wagi te dobierane są losowo w postaci niewielkich liczb i przy ich użyciu, na podstawie elementów wektora sygnałów wejściowych **x**, oblicza się elementy wektora wyjściowego sieci $\mathbf{y}^L$ (stanowiącego w istocie wektor rzeczywistych sygnałów wyjściowych).

Po porównaniu wartości elementów wektora $\mathbf{y}^L$ z wartościami odpowiednich elementów wektora „oczekiwanych" sygnałów wyjściowych **d**, wyznaczonych w przypadku poszczególnych „punktów uczących" o numerach $l = 1, 2, ..., m$, można określić błąd sieci, wykorzystując zależność:

$$\varepsilon = \sum_{l=1}^{m} \sum_{i=1}^{N_L} \left[d_i(l) - y_i^L(l)\right]^2. \tag{14}$$

Proces „uczenia sieci" realizuje się dopóty, dopóki błąd ten nie stanie się mniejszy od przyjętego, dopuszczalnego błędu sieci.

Generalnie znane są trzy podstawowe metody „uczenia sieci" – metoda najmniejszych kwadratów [10], metoda propagacji wstecznej błędów [9, 10, 18] oraz metoda algorytmu genetycznego [5, 13], którą wykorzystuje się również w przypadku prezentowanego programu.

Należy pamiętać, że uwzględniona liczba m „punktów uczących" ma znaczący wpływ na efektywność metody – przyjęcie zbyt dużej liczby tych punktów powoduje wydłużenie czasu obliczeń, natomiast następstwem przyjęcia zbyt małej ich liczby jest ryzyko wystąpienia dużego błędu sieci.

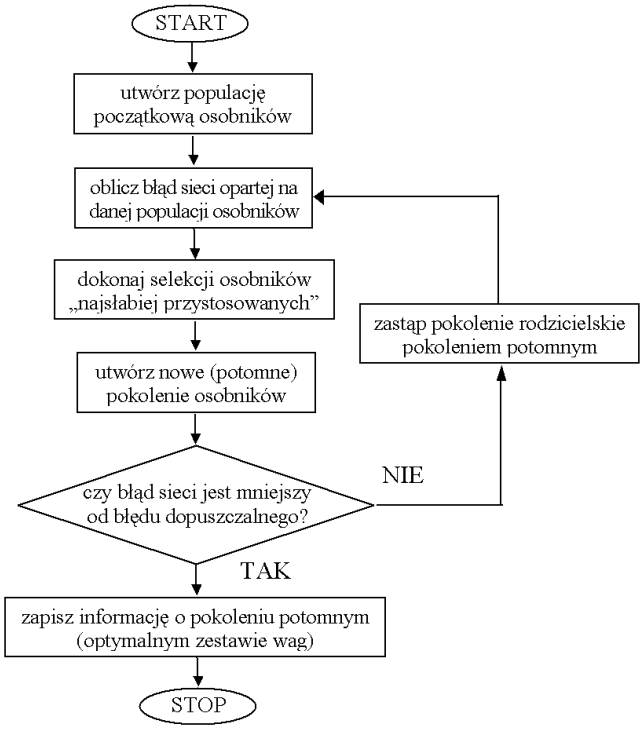
4. Ogólna charakterystyka algorytmu genetycznego

Działanie algorytmu genetycznego wzorowane jest na zachodzących w przyrodzie procesach ewolucji, pozwalających żywym organizmom na dostosowanie się do otaczającego je środowiska. Wnikając w istotę procesu, można zaobserwować kolejne etapy rozwoju organizmów. Najpierw całe pokolenie jest poddawane oddziaływaniu środowiska, w którym osobniki „lepiej przystosowane" mają większe szanse przeżycia. Osobniki, które przetrwały, mogą przystąpić do reprodukcji, w wyniku której powstaje nowe ich pokolenie (pokolenie potomne). Dziedziczy ono, w postaci zakodowanej, cechy swojego „pokolenia rodzicielskiego". Powstałe pokolenie potomne poddawane jest analogicznym procesom [5, 13].

Okazuje się, że opierając się na tak scharakteryzowanych ogólnych przesłankach, można dokonywać analogicznych zmian (ewolucji) wartości wybranych zmiennych matematycznych, tak aby uzyskać możliwie najlepszy (optymalny) wynik wyznaczanej zależności (funkcji). W przypadku prezentowanej metody zmiennymi, decydującymi o końcowym wyniku, określonym zależnością (14), są wagi związane z poszczególnymi neuronami sieci. W każdym pokoleniu tworzy się zatem wirtualnego osobnika, reprezentującego jeden zestaw wag. Można stwierdzić, że jego „przystosowanie" będzie tym większe, im mniejszy będzie błąd sieci, opartej na oferowanym przez niego zestawie wag.

Operacje tworzenia kolejnych pokoleń osobników (iteracje), w ramach których dokonuje się stosownych selekcji osobników „najsłabiej przystosowanych", powtarzają się aż do utworzenia osobnika, gwarantującego uzyskanie błędu sieci o wartości mniejszej od przyjętego błędu dopuszczalnego.

Użyty algorytm postępowania (algorytm genetyczny) można zatem przedstawić w postaci zaprezentowanej na rys. 8.



Rys. 8. Schemat działania przyjętego algorytmu genetycznego
Fig. 8. Scheme of genetic algorithm assumed

Kolejnym zagadnieniem jest dobór odpowiedniej funkcji, na podstawie której można by dokonać stosownej oceny „przystosowania” poszczególnych osobników (czyli w przypadku prezentowanej metody – ocenić jakość doboru wag poszczególnych neuronów, w aspekcie minimalizacji błędu, opartej na nich sieci). Przyjęto, że rolę tę będzie pełnić funkcja, określająca błąd rzeczywistego położenia końcowego punktu K efektora w stosunku do jego wymaganego położenia, którą zapisano wzorem:

$$f = \left(X_0^K - X_0^{K'}\right)^2 + \left(Y_0^K - Y_0^{K'}\right)^2 + \left(Z_0^K - Z_0^{K'}\right)^2, \tag{15}$$

gdzie:

X_0^K, Y_0^K, Z_0^K – wymagane współrzędne końcowego punktu K efektora w bazowym układzie $\hat{\mathbf{X}}_0\hat{\mathbf{Y}}_0\hat{\mathbf{Z}}_0$,

$X_0^{K'}, Y_0^{K'}, Z_0^{K'}$ – rzeczywiste (uzyskane w wyniku działania sieci) współrzędne tego punktu.

5. Opis przygotowanego programu „NeuroManip”

W procesie rozwiązywania odwrotnego zadania kinematyki wybranego manipulatora przy użyciu, napisanego w środowisku programistycznym Delphi 5.0, programu komputerowego, który nazwano „Neuromanip”, można wyróżnić następujące etapy:

- określenie parametrów geometrycznych manipulatora,
- budowa sieci neuronowej,
- realizacja procesu „uczenia” sieci,
- realizacja procesu rozwiązywania odwrotnego zadania kinematyki analizowanego manipulatora przy wykorzystaniu algorytmu genetycznego,
- alternatywne rozwiązanie zadania przy wykorzystaniu „metody jakobianu pseudoodwrotnego”.

Jak już stwierdzono, w przyjętej metodzie liczba członów rozważanego manipulatora może być (teoretycznie) dowolna. Zakłada się przy tym, że są one połączone przegubami kulistymi, przy czym dopuszcza się możliwość „odebrania” łączonym członom jednej lub dwóch możliwości ruchu względnego (a w konsekwencji odebranie tworzącemu manipulator łańcuchowi kinematycznemu stosownej liczby stopni swobody).

Opracowany program wykorzystano do rozwiązywania odwrotnego zadania kinematyki wybranego manipulatora. Na jego przykładzie, w dalszej części artykułu, „prześledzona” zostanie realizacja poszczególnych etapów programu.

5.1. Określenie parametrów geometrycznych manipulatora

Pierwszym etapem w przyjętym postępowaniu jest określenie struktury i geometrii rozważanego manipulatora, poprzez podanie liczby jego członów i ich długości. Wybrany do analizy manipulator posiadał trzy człony (oznaczono je numerami $i = 1, 2, 3$), których długości wynosiły odpowiednio 150, 100 i 150 cm. Człony połączone zostały przegubami kulistymi, przy czym przyjęto, że jedynie kąty α_i i β_i (gdzie $i = 1, 2, 3$) – rys. 1, mogą zmieniać się w czasie, natomiast kąty γ_i zostały przyjęte jako stałe. Łączonym członom odebrano zatem jedną możliwość ruchu względnego, wobec czego analizowany manipulator posiadał sześć stopni swobody.

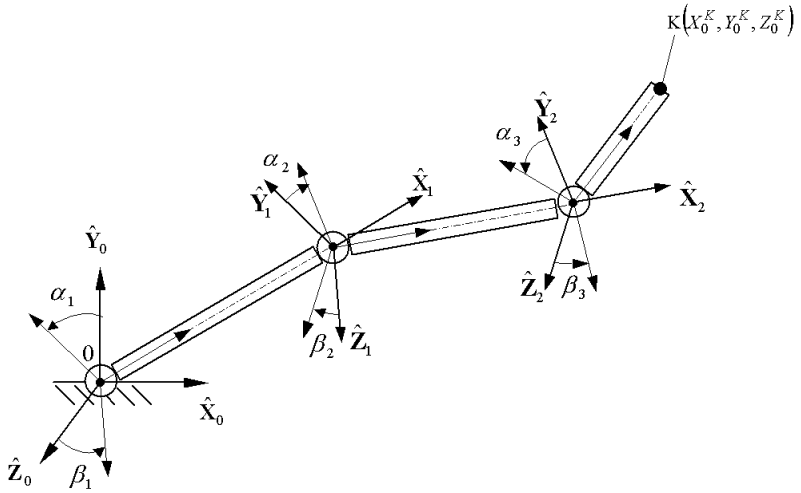
5.2. Budowa sieci neuronowej

Do obliczeń została wybrana jednokierunkowa, wielowarstwowa sieć neuronowa. Przyjęto następujące jej parametry: liczba warstw – 4 (warstwa wejściowa, dwie warstwy ukryte, warstwa wyjściowa); liczba neuronów w warstwie początkowej – 24, w każdej warstwie ukrytej – 24 i w warstwie wyjściowej – 6 (zatem sieć zbudowano przy wykorzystaniu 72 neuronów).

5.3. Realizacja procesu „uczenia” sieci

Do „uczenia” sieci wykorzystano zbiór $m = 1000$ losowo wygenerowanych „punktów uczących”. Generowanie pojedynczego „punktu uczącego” przebiegało w następującej kolejności:

- przy użyciu funkcji „Random” wylosowane zostały wartości kątów α_i i β_i, gdzie $i = 1, 2, 3$ (w programie przewiduje się, że maksymalne wartości tych kątów określane zostaną przez jego użytkownika),
- następnie, przy wykorzystaniu wzoru (1), obliczone zostały współrzędne (początkowe) X_0^K, Y_0^K, Z_0^K końcowego punktu K efektora (rys. 9) w bazowym układzie $\hat{\mathbf{X}}_0\hat{\mathbf{Y}}_0\hat{\mathbf{Z}}_0$,
- z kolei, przy użyciu funkcji „Random”, wylosowane zostały wartości przyrostów $\Delta\alpha_i$ i $\Delta\beta_i$ (gdzie $i = 1, 2, 3$) poszczególnych kątów (w programie przewiduje się, że maksymalne wartości tych przyrostów określane zostaną przez jego użytkownika),
- następnie obliczone zostały nowe wartości kątów α_i i β_i (gdzie $i = 1, 2, 3$), wyznaczono nowe współrzędne X_0^{K*}, Y_0^{K*}, Z_0^{K*} końcowego punktu K efektora i obliczono przyrosty ΔX_0^K, ΔY_0^K, ΔZ_0^K tych współrzędnych, w stosunku do jego współrzędnych początkowych (rys.10).



Rys. 9. Losowy wybór wartości kątów α_i i β_i (gdzie $i = 1, 2, 3$)
Fig. 9. Random selection of values of angles α_i and β_i (where $i = 1, 2, 3$)

Wygenerowany w ten sposób „punkt uczący” określony był przez wektor sygnałów wejściowych:

$$\mathbf{x}^T = \left[\alpha_1,\ \beta_1,\ \alpha_2,\ \beta_2,\ \alpha_3,\ \beta_3,\ \Delta X_0^K,\ \Delta Y_0^K,\ \Delta Z_0^K\right] \tag{16}$$

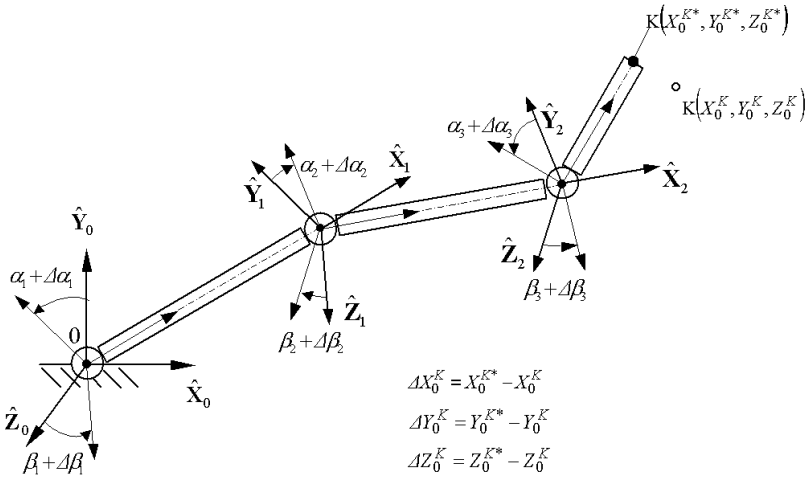
oraz wektor „oczekiwanych” sygnałów wyjściowych:

$$\mathbf{d}^T = \left[\Delta\alpha_1,\ \Delta\beta_1,\ \Delta\alpha_2,\ \Delta\beta_2,\ \Delta\alpha_3,\ \Delta\beta_3\right]. \tag{17}$$

Zadaniem utworzonej sieci neuronowej było wygenerowanie takich wartości przyrostów $\Delta\alpha_i$ i $\Delta\beta_i$ (gdzie $i = 1, 2, 3$) poszczególnych kątów, które zapewnić miały wymagane przemieszczenie końcowego punktu K efektora, zgodnie z zaplanowaną trajektorią.

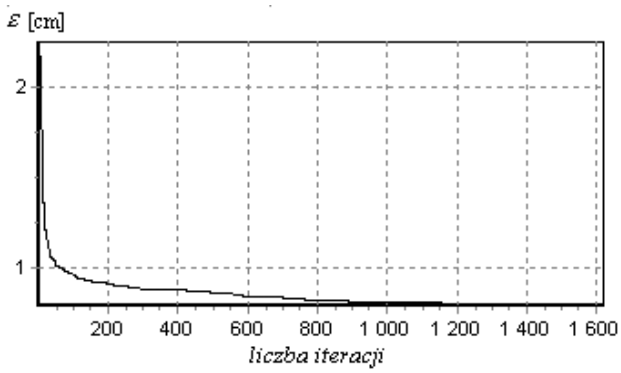
Do realizacji procesu „uczenia” sieci w programie wykorzystany został algorytm genetyczny o następujących parametrach [5, 13]: liczba osobników – 30, prawdopodobieństwo krzyżowania – 0,97, prawdopodobieństwo mutacji – 0,001. Przeprowadzone symulacje nu-

meryczne wykazały, że takie wartości parametrów gwarantują uzyskanie największej efektywności obliczeń (powodują zminimalizowanie czasu potrzebnego na „uczenie" sieci).



Rys. 10. Zmiany wartości kątów α_i i β_i po losowym wyborze wartości przyrostów $\Delta\alpha_i$ i $\Delta\beta_i$ (gdzie i = 1, 2, 3)

Fig. 10. Changes of values of angles after random selection of values of increases $\Delta\alpha_i$ and $\Delta\beta_i$ (where i = 1, 2, 3)



Rys. 11. Przebieg błędu sieci wyrażony w funkcji liczby zrealizowanych iteracji

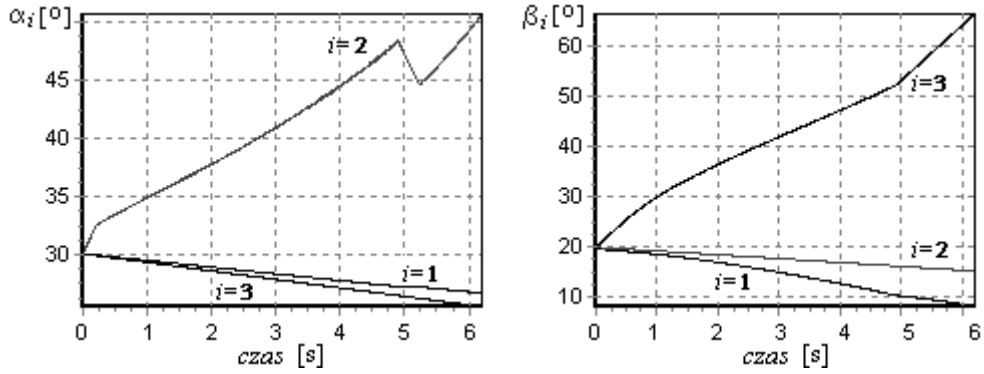
Fig. 11. Course of error of neural network expressed as a function of number of iterations realised

Na rys. 11 zilustrowano proces zmniejszania się, określonego zależnością (14), błędu ε utworzonej sieci – w trakcie kolejnych iteracji procesu jej „uczenia". Uwzględniając fakt, że czas trwania jednej iteracji wynosi około 2 s, można oszacować łączny czas trwania procesu „uczenia" sieci na około 50 min.

5.4. Realizacja procesu rozwiązywania odwrotnego zadania kinematyki analizowanego manipulatora przy wykorzystaniu algorytmu genetycznego

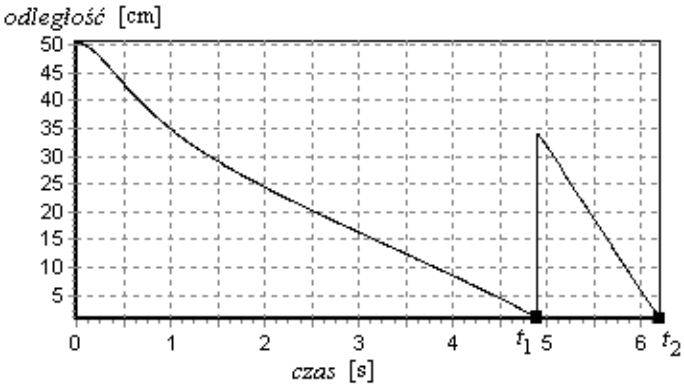
Dysponując dobranym zestawem wag, można było przystąpić do rozwiązywania odwrotnego zadania kinematyki rozważanego manipulatora.

Określenie planowanej trajektorii końcowego punktu K efektora następuje w programie poprzez podanie współrzędnych punktów (jednego lub kilku punktów pośrednich i punktu docelowego), przez które ma ona przechodzić. Ustalając początkowe położenie członów analizowanego manipulatora, przyjęto wartości poszczególnych kątów równe: $\alpha_i = 30^o$ i $\beta_i = 20^o$ (gdzie i=1,2,3). Odpowiednio do tych wartości określono, przy wykorzystaniu zależności (1), położenie końcowego punktu K efektora, w bazowym układzie $\hat{\mathbf{X}}_0\hat{\mathbf{Y}}_0\hat{\mathbf{Z}}_0$, jako: $X_0^K = X_0^P =$ $= 110{,}4$ cm, $Y_0^K = Y_0^P = 252{,}8$ cm, $Z_0^K = Z_0^P = -211{,}1$ cm. Położenie to było tożsame z położeniem początkowego punktu P trajektorii. Sformułowane zadanie polegało na przemieszczeniu końcowego punktu K efektora, najpierw do punktu pośredniego S trajektorii, o współrzędnych w bazowym układzie: $X_0^S = 100$ cm, $Y_0^S = 200$ cm, $Z_0^S = -200$ cm, a następnie do jej punktu docelowego D, o współrzędnych: $X_0^D = 110$ cm, $Y_0^D = 170$ cm, $Z_0^D = -200$ cm. Rozwiązując odwrotne zadanie kinematyki, wyznaczono czasowe przebiegi kątów α_i i β_i (gdzie i = 1, 2, 3) – rys. 12, zapewniające ruch końcowego punktu K efektora, zgodny z wymaganą trajektorią.



Rys. 12. Przebiegi kątów α_i i β_i (gdzie i = 1, 2, 3) uzyskane w przypadku przyjętej sieci neuronowej

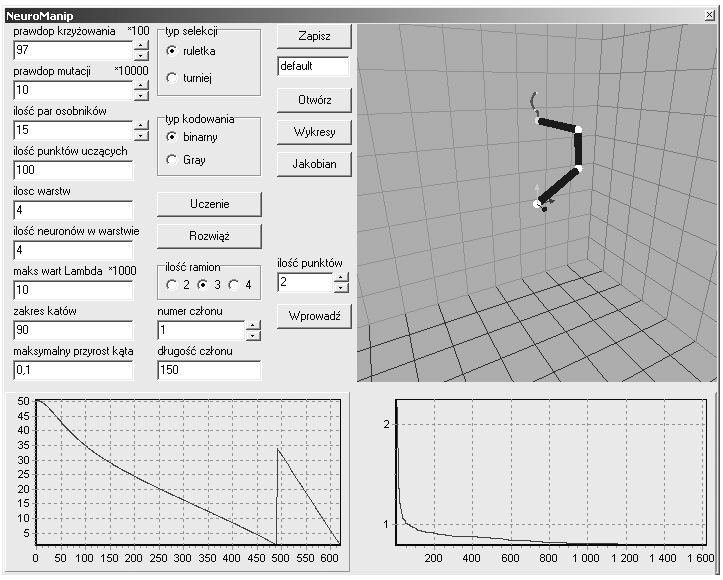
Fig. 12. Course of angles obtained in case of neural network assumed



Rys. 13. Ilustracja procesu „zbliżania się” końcowego punktu K efektora do kolejnych punktów trajektorii

Fig. 13. Illustration of process of bringing terminal point K of efector nearer to successive points of trajectory

Na rys. 13 zilustrowano, wyrażoną w czasie, zmianę odległości końcowego punktu K efektora od przyjętych punktów trajektorii (w chwili t_1 punkt K zajmował położenie S, w chwili t_2 zaś położenie D).



Rys. 14. Okno wyświetlanych wykresów
Fig. 14. Screen of graphs projected

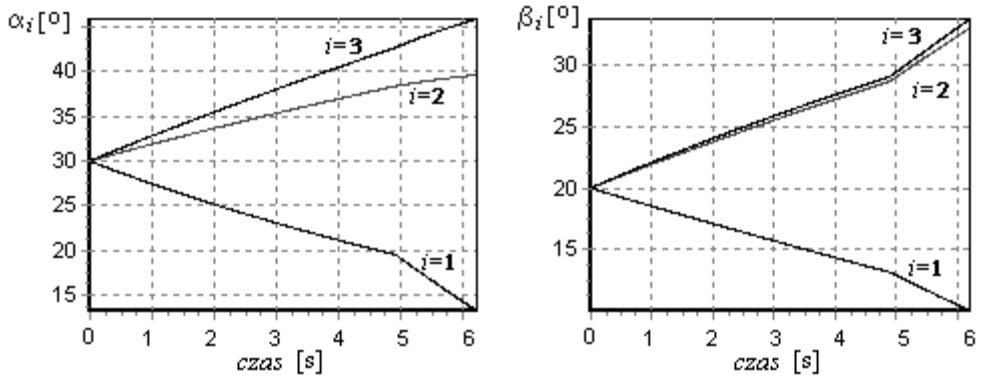
Program umożliwia prezentację animacji ruchu członów rozważanego manipulatora, którego końcowy punkt K porusza się zgodnie z zadaną trajektorią, dokonywaną za pośrednictwem „okna wyświetlanych wykresów" (rys.14).

5.5. Alternatywne rozwiązanie zadania

Opracowany program umożliwia także alternatywne rozwiązanie odwrotnego zadania kinematyki, w przypadku rozważanego manipulatora przy wykorzystaniu, opartej na odmiennym formalizmie, „metody jakobianu pseudoodwrotnego" [6, 15]. Na, zaprezentowanym wcześniej, rys. 12 przedstawiono, uzyskane przy użyciu sieci neuronowej, czasowe przebiegi kątów α_i i β_i (gdzie i = 1, 2, 3), a na rys. 16 a – współrzędnye końcowego punktu K efektora. Na rys. 15 i 16 b przedstawiono odpowiednio przebiegi tych samych wielkości, uzyskane przy wykorzystaniu „metody jakobianu pseudoodwrotnego".

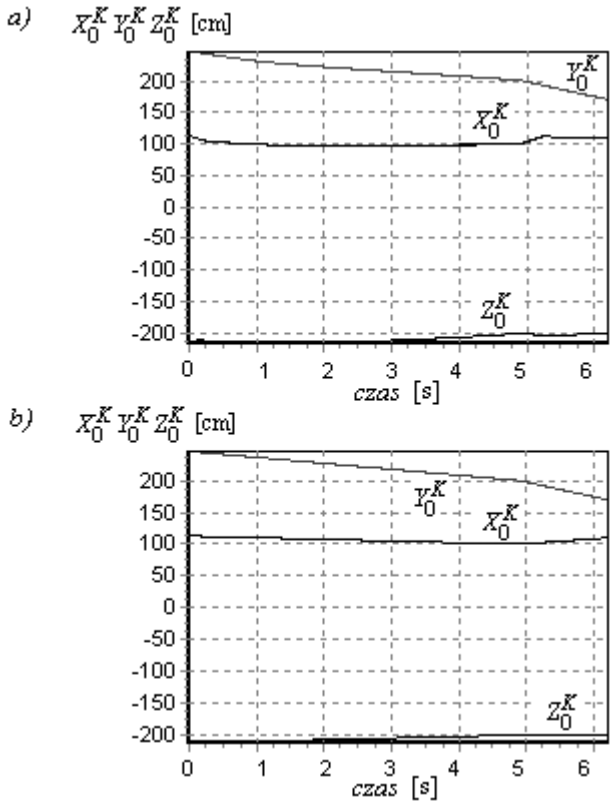
Można zauważyć, że przebiegi czasowe poszczególnych kątów α_i i β_i (gdzie i = 1, 2, 3), wyznaczone przy użyciu sieci neuronowej, różnią się od przebiegów tych samych kątów, uzyskanych przy zastosowaniu „metody jakobianu pseudoodwrotnego". Różnice w wyznaczonych przebiegach, a więc w zaoferowanych przez obie metody rozwiązaniach odwrotnego zadania kinematyki, wynikają z faktu, iż – jak już stwierdzono – nie ma jedynego (wyłącznego) rozwiązania tego zadania. Danemu położeniu końcowego punktu K efektora odpowiada bowiem kilka możliwych wariantów zbiorów kątów α_i i β_i (gdzie i = 1, 2, 3), gwarantują-

cych uzyskanie tego położenia. Rozwiązań postawionego zadania jest zatem kilka i każda z obu metod oferuje inne z nich.



Rys. 15. Przebiegi kątów α_i i β_i (gdzie i = 1, 2, 3) uzyskane w przypadku „metody jakobianu pseudoodwrotnego”

Fig. 15. Courses of angles obtained in case of „pseudoinverse Jacobian method”



Rys. 16. Przebiegi współrzędnych końcowego punktu K efektora uzyskane w przypadku: a) przyjętej sieci neuronowej, b) „metody jakobianu pseudoodwrotnego”

Fig. 16. Courses of coordinates of terminal point K of efector obtained in case of: a) neural network assumed, b) „pseudoinverse Jacobian method”

Natomiast, jak wynika z porównania rys. 16 a i 16 b, przebiegi czasowe współrzędnych końcowego punktu K efektora, uzyskane w przypadku uwzględnienia obu rodzajów czasowych przebiegów kątów α_i i β_i (gdzie i = 1, 2, 3), jedynie nieznacznie się różnią. Analizując obydwie grupy wykresów, można stwierdzić, że w obu rozważanych przypadkach końcowy

punkt K efektora, po rozpoczęciu ruchu z położenia określonego punktem P, osiągnął zadane położenia określone punktami S oraz D. Obie metody zapewniły zatem poprawny wynik.

6. Uwagi końcowe

Opracowany program komputerowy umożliwia wspomaganie nauczania z zakresu zastosowań sieci neuronowych i algorytmów genetycznych w zagadnieniach sterowania układów mechanicznych. Zdaniem autorów, program ten może być użyteczny w procesie dydaktycznym, realizowanym dla studentów „kierunków mechanicznych" uczelni technicznych (w tym głównie kierunku „Automatyka i Robotyka"), studiujących zagadnienia metod sterowania robotów.

Na podstawie obserwacji, poczynionych w wyniku przeprowadzenia eksperymentów obliczeniowych, można stwierdzić, że przygotowany program gwarantuje uzyskiwanie poprawnych wyników, niemniej jednak na jego efektywność ujemnie wpływa stosunkowo długi czas „uczenia" sieci neuronowej, wynoszący kilkadziesiąt minut. Okazało się przy tym, że czas ten był znacznym stopniu uzależniony od wartości przyjętych parametrów użytego algorytmu genetycznego, tzn. liczby osobników, prawdopodobieństwa krzyżowania oraz mutacji.

LITERATURA

1. Brudka M.: Sieci neuronowe w sterowaniu robotem na podstawie obrazów ultradźwiękowych. Praca doktorska, Politechnika Warszawska, Wydział Elektroniki i Technik Informacyjnych, Warszawa 2000.
2. Craig J. J.: Wprowadzenie do robotyki. Mechanika i sterowanie. WNT, Warszawa 1995.
3. Giergiel J., Hendzel Z., Trojnacki M.: Szybkie prototypowanie neuronowego algorytmu sterowania minirobotem. Przegląd Mechaniczny, Nr 4, 2005.
4. Giergiel J., Hendzel Z., Trojnacki M. :Szybkie prototypowanie neuronowego algorytmu sterowania mobilnym robotem. Materiały III Warsztatów Projektowania Mechatronicznego, Kraków 2003.
5. Goldberg D.: Algorytmy genetyczne i ich zastosowania. WNT, Warszawa 1995, 1998, 2003.
6. Harlecki A., Tengler S.: Komputerowy program dydaktyczny do numerycznego rozwiązywania odwrotnych zadań kinematyki manipulatorów. ZN Akademii Techniczno-Humanistycznej, Nr 17, Bielsko-Biała 2004.
7. Knosala R. (red.): Zintegrowany system wytwarzania modułowych zespołów maszyn. Skrypt Uczelniany Nr 1939, Wydawnictwa Politechniki Śląskiej, Gliwice 1995.

8. De Lope J., Zarraonandia T., González-Careaga R., Maravall D.: Solving the inverse kinematics in humanoid robots: a neural approach. http://gsyc.escet.urjc.es/robotica/publicaciones/rafaela-iwann03.pdf.
9. Macukow B.: Sieci neuronowe w układach sterowania robotów. Prace Naukowe Instytutu Cybernetyki Technicznej Politechniki Wrocławskiej, Nr 83, 1990.
10. Osowski S.: Sieci neuronowe do przetwarzania informacji, Oficyna Wydawnicza Politechniki Warszawskiej, Warszawa 2000.
11. Rivals I., Personnaz L., Dreyfus G., Canas D.: Modeling and control of mobile robots and intelligent vehicles by neural network. Proc. of IEEE Conference on Intelligent Vehicles, Paris 1994.
12. Rivals I., Canas D., Personnaz L., Dreyfus G.: Real-time control of an autonomous vehicle: a neural network approach to the path following problem. Proc. of 5th International Conference on Neural Networks and their Applications "NeuroNimes93", Nanterre 1993.
13. Rutkowska D., Piliński J., Rutkowski L.: Sieci neuronowe, algorytmy genetyczne i systemy rozmyte. Wydawnictwa Naukowe PWN, Warszawa 1997.
14. Tang W. S., Wang J.: A recurrent neural network for minimum infinity-norm kinematic control of redundant manipulators with an improved problem formulation and reduced architecture complexity. IEEE Transaction on Systems, Man, and Cybernetics – Part B: Cybernetics, Vol. 31, No. 1, 2001.
15. Tchoń K., Mazur A., Dulęba I., Hossa R., Muszyński R.: Manipulatory i roboty mobilne. Modele, planowanie ruchu, sterowanie. Akademicka Oficyna Wydawnicza PLJ, Warszawa 2000.
16. Zhang Y., Wang J.: Obstacle avoidance for kinematically redundant manipulators using a dual neural network. IEEE Transaction on Systems, Man, and Cybernetics – Part B: Cybernetics, Vol. 34, No. 1, 2004.
17. Zhou Y., Wilkins D., Cook R. P.: Neural network control for a fire-fighting robot, http://www.cs.wustl.edu/~zy/Robot.pdf.
18. Żurada J., Barski M., Jędruch W.: Sztuczne sieci neuronowe. Wydawnictwa Naukowe PWN, Warszawa 1996.

Recenzent: Dr inż. Marcin Skowronek

Wpłynęło do redakcji 8 czerwca 2006 r.

Abstract

After the introduction (the Chapter 1), the way of assuming coordinate systems, connected to particular links of manipulators considered, and defining transformation matrices, based on the Euler angles, has been presented (the Chapter 2).

The basic principles of preparing artificial neural networks for control of motion of manipulators' links have been characterised in the Chapter 3.

In the Chapter 4., the general scheme of genetic algorithm, which has been used for training the neural network assumed, has been shown.

The detailed description of the computer program worked out, which has been applied for solving a inverse kinematics problem in the case of the manipulator considered, has been presented in the Chapter 5. The chosen results of the task realised have been shown in fig.11, 12, 13 and 16a. In this Chapter, the results of the "pseudoinverse Jacobian method", applied alternatively for solving the inverse kinematics problem formulated, have been also shown (fig.15 and 16b). These results have been compared with results obtained in the case of the neural network assumed.

The final remarks have been presented in the Chapter 6.

Adresy

Rafał SZKOWRON, Akademia Techniczno-Humanistyczna, Katedra Mechaniki i Informatyki, ul. Willowa 2, 43-309 Bielsko-Biała, Polska, rafal.szkowron@poczta.fm.

Andrzej HARLECKI, Akademia Techniczno-Humanistyczna, Katedra Mechaniki i Informatyki, ul. Willowa 2, 43-309 Bielsko-Biała, Polska, aharlecki@ath.bielsko.pl.