POLITECHNIKA WARSZAWSKA

Wydział Elektryczny

ROZPRAWA DOKTORSKA

mgr inż. Marcin Paprocki

Sterowanie adaptacyjne maszyn wieloosiowych z wykorzystaniem elementów sztucznej inteligencji

Promotor
prof. dr hab. inż. Lech M. Grzesiak

Warszawa 2015

mojej żonie Justynie

Składam serdeczne podziękowania panu prof. dr. hab. inż. Lechowi M. Grzesiakowi, dr. inż. Kazimierzowi Karwowskiemu oraz mgr. inż. Andrzejowi Wawrzakowi za pomoc, cenne uwagi i poświęcony czas.

Streszczenie

Rozprawa doktorska dotyczy minimalizacji błędów nadążania w wieloosiowych maszynach CNC (ang. Computerized Numerical Control). W pracy rozważono zagadnienia realizacji zadanych przemieszczeń poszczególnych osi mechanicznych maszyny CNC. Zaproponowano innowacyjne rozwiązanie wyznaczania zmodyfikowanych przemieszczeń osi mechanicznych minimalizujące ich błędy nadążania. Opracowany algorytm wykorzystuje, w każdej ze struktur sterowania osi, blok Kompensatora Trajektorii Zadanej. Blok na podstawie neuronowego modelu osi maszyny dokonuje predykcji błędów nadążania powstałych w procesie obróbki. Następnie algorytm optymalizacyjny wyznacza zmodyfikowane przemieszczenia osi w celu minimalizacji błędu nadążania. Rozwiązanie nieliniowego problemu optymalizacyjnego dokonywane jest z wykorzystaniem algorytmu optymalizacji rojem cząstek (ang. Particle Swarm Optimization - PSO). Blok Kompensatora Trajektorii Zadanej zintegrowano ze sterownikiem CNC, który zaimplementowano w komputerze PC z systemem czasu rzeczywistego Linux RTAI. Na podstawie uzyskanych wyników badań wykazano zmniejszenie błędów nadążania w poszczególnych osiach maszyny. Opracowana strategia sterowania poprawia dokładność wykonania elementów obrabianych w maszynach CNC.

Słowa kluczowe: *CNC, zmniejszanie błędów odtwarzania zadanej trajektorii ruchu, sztuczne sieci neuronowe, optymalizacja rojem cząstek.*

Abstract

The thesis concerns minimization of following errors in multi-axis CNC machines (Computerized Numerically Controlled). In the thesis the problem of realizing demanded displacements of every axis of the CNC machine was presented. An innovative solution was proposed that modyfies those displacement in order to minimise the following errors. The algorithm developed utilizes a Trajectory Compensator Block for every axis control structure. This block predicts the following errors that arise during machining by utilizing a neural network model of the machines axis. The optimization algorithm then computes modified displacements of the machine's axes so that the following error is minimized. Solution of the nonlinear optimization problem is achieved by using Particle Swarm Optimization (PSO) algorithm. The Trajectory Compensation Block was implemented in a CNC controller. The CNC controller is implemented in a PC with Linux RTAI real-time operating system. Based on results obtained a decrease of following error in each machine axis was shown. The developed control strategy improves machining accuracy in CNC machine tools.

Keywords: *CNC, decrease motion trajectory errors, Artificial Neural Networks, Particle Swarm Optimization algorithm.*

Praca naukowa współfinansowana ze środków:

— Europejskiego Funduszu Społecznego, Budżetu Państwa i Budżetu Województwa Kujawsko-Pomorskiego w ramach Programu Operacyjnego Kapitał Ludzki Priorytetu VIII, działanie 8.2 poddziałanie 8.2.2 „Regionalne Strategie Innowacji", projektu systemowego Samorządu Województwa Kujawsko-Pomorskiego „Krok w przyszłość 2 – stypendia dla doktorantów",

— Europejskiego Funduszu Społecznego i Budżetu Państwa w ramach Zintegrowanego Programu Operacyjnego Rozwoju Regionalnego, Działania 2.6 „Regionalne Strategie Innowacyjne i transfer wiedzy" projektu własnego Województwa Kujawsko-Pomorskiego „Stypendia dla doktorantów 2008/2009 – ZPORR",

— Europejskiego Funduszu Społecznego, Budżetu Państwa i Budżetu Województwa Kujawsko-Pomorskiego w ramach Programu Operacyjnego Kapitał Ludzki Priorytetu VIII, działanie 8.2 poddziałanie 8.2.2 „Regionalne Strategie Innowacji", projektu systemowego Samorządu Województwa Kujawsko-Pomorskiego „Krok w przyszłość IV – stypendia dla doktorantów".

Spis treści

1. Wprowadzenie

Maszyny wieloosiowe sterowane numerycznie są stosowane w wielu gałęziach przemysłu. Spektrum tego typu rozwiązań jest ogromny - od specjalizowanych urządzeń technologicznych, poprzez obrabiarki numeryczne do maszyn manipulacyjnych (robotów). Obrabiarki numeryczne umożliwiają obróbkę mechaniczną materiału za pomocą narzędzia wykonawczego (np. wrzeciono z frezem, laser, itp.).

W budowie obrabiarek sterowanych numerycznie jako podstawowe elementy wykonawcze wyróżnić można osie mechaniczne maszyny, w których budowie wyodrębnia się układ przeniesienia napędu oraz napęd. Uzyskanie dużego zakresu ruchów roboczych w maszynach wieloosiowych w przestrzeni 3D wymaga zastosowania przynajmniej kilku osi mechanicznych. W maszynach wieloosiowych sterowanych numerycznie najczęściej spotykaną konfiguracją osi mechanicznych jest układ kartezjański. Takie ustawienie osi pozwala uzyskać dowolny punkt w przestrzeni 3D w ramach możliwych liniowych przemieszczeń każdej z osi mechanicznych. Uzyskanie każdego punku przez narzędzie wykonawcze maszyny wieloosiowej (np. wrzeciono wraz z frezem) wymaga zastosowania jeszcze dodatkowych dwóch osi obrotowych. Tworzą one układ 5-cio osiowy wraz z pozostałymi trzema osiami. Zastosowanie dwóch osi obrotowych umożliwia obróbkę materiału przez narzędzie wykonawcze pod zadanym kątem, który w niektórych procesach technologicznych jest niezbędny [7]. Oprócz układów trójosiowych i pięcioosiowych w maszynach wieloosiowych stosowane są również konfiguracje osi mechanicznych o kinematyce równoległej np. typu „hexapod” lub „tripod” [98].

W latach 60. i 70. XX w. struktura sterowania numerycznego (ang. Numerical Control - NC) maszyn wieloosiowych bazowała na sprzętowych blokach funkcyjnych obróbki i przetwarzania informacji, których modyfikacja wymagała przebudowy układu sterowania. Zadaniem sterownika NC była realizacja z góry określonych (niezmiennych) ruchów osi mechanicznych opisanych w sposób symboliczny (np. w postaci liczbowej). Ich modyfikacja nierzadko wymagała modyfikacji bloków sterownika NC, a niekiedy jego całkowitej wymiany.

Przełomem w sterowaniu takimi maszynami było opracowanie komputerowego sterowania - CNC (ang. Computerized Numerical Control – komputerowe sterowanie numeryczne) oraz nowoczesnych cyfrowych serwonapędów. Budowa układów CNC możliwa była dzięki pojawieniu się zaawansowanych układów mikroprocesorowych (komputerów). Osiągalna stała się modyfikacja oraz podmiana programowo zadanych ruchów osi mechanicznych w samym

sterowniku (np. poprzez wykorzystanie kart dziurkowanych). Wzrosła także długość oraz złożoność tych ruchów, którą mógł obsłużyć sterownik CNC.

Głównym zadaniem układów sterowania w maszynach wieloosiowych CNC jest realizacja zadanej trajektorii ruchu na podstawie zadanego kodu maszynowego (ang. Numerical Code)[44]. W procesie interpolacji [44] z kodu numerycznego wyznaczane są przemieszczenia poszczególnych osi mechanicznych. Przesyłane są one z układu sterownika CNC do układów napędowych osi mechanicznych. Przemieszczenia wszystkich osi mechanicznych maszyny przekładają się na przesunięcie narzędzia roboczego wzdłuż zadanego toru ruchu. Ten sposób sterowania nazywany jest sterowaniem kształtowym (ang. contouring control lub continuous path control) [44]. Dopiero wprowadzenie oprogramowania CAD/CAM umożliwiło wykorzystanie zaawansowanego sterowania kształtowego, co znacząco wpłynęło na dokładność i wydajność produkcji.

Na dokładność przemieszczeń danej osi mechanicznej obrabiarki numerycznej ma wpływ rodzaj zastosowanego napędu. Napędy w obrabiarkach najczęściej pracują w trybie regulacji położenia. Wymagane jest by napędy osiągały zadane wartości położeń przy jednoczesnym nałożeniu ograniczeń na wielkości prędkości, przyspieszenia oraz zrywu.

W trakcie realizacji trajektorii powstają błędy odtwarzania zadanej trajektorii ruchu. Na powstawanie tych błędów mają wpływ czynniki związane z elementami mechanicznymi maszyn wieloosiowych (konstrukcja maszyn oraz stopień zużycia elementów mechanicznych) jak i czynniki związane z układami sterowania. Wzrost temperatury prowadnic w czasie pracy maszyn przekłada się na zmianę współczynnika tarcia poszczególnych elementów maszyn[118]. Również zużycie elementów mechanicznych, zmiana własności układów sprężystych czy okresowe konserwacje maszyn wpływają na zmiany ich własności dynamicznych[109]. Dodatkowo w procesie obróbki pojawiają się trudne do przewidzenia zakłócenia[59].

Dodatkowym źródłem błędów są niedoskonałości w wykonaniu silników i urządzeń pomiarowych w układach regulacji oraz osiąganie granicznych wartości prędkości i prądu w napędzie. Występuje to najczęściej w przypadku zadawania skrajnych wartości prędkości i przyspieszeń posuwu.

Istnieje ciągła potrzeba udoskonalania systemów sterowania w celu jeszcze większej wydajności pracy jak i ich elastycznego dostosowania do wymagań technologicznych.

1.1. Przegląd metod minimalizacji błędów odtwarzania zadanej trajektorii ruchu

Metody minimalizacji błędów odtwarzania zadanej trajektorii można podzielić pod względem ich metodologii na trzy grupy [41]. Metody zorientowane na zmniejszanie błędów

nadążania (ang. tracking control)[87], metody sterowania zorientowane na zmniejszanie błędu konturu (ang. contouring control)[87] oraz metody korekcji zadanej trajektorii ruchu (kompensacji trajektorii ruchu względem warunków pracy maszyny).

Zmniejszanie błędów nadążania w układach sterowania można realizować poprzez zastosowanie sterowania śledzącego (nadążnego). Takie sterowanie występuje najczęściej w serwonapędach i jest to w większości przypadków kaskadowy układ regulacji z regulatorami PID (ang. Proportional-Integral-Derivative) lub PIV (ang. Proportional position loop Integral and proportional Velocity loop). Problemem jest odpowiedni (optymalny) dobór nastaw w tych regulatorach. Oprócz klasycznych metod doboru nastaw regulatorów PID (m. in.: symetryczne i modułowe optimum, kryteriów całkowych, lokowania biegunów [10]), wielu badaczy opracowało alternatywne rozwiązania, które wspomagają dobór optymalnych nastaw regulatora PID. Wei-Der Chang [14] proponuje zastosowanie technik z wykorzystaniem algorytmów genetycznych (ang. genetic algorithms - GA). Natomiast Y. Zhang i in. [129] opracowali w tym celu blok dostrajający regulator PID z użyciem logiki rozmytej (ang. fuzzy logic - FL). H. Nurhadi i in. [80] proponuje technikę optymalizacji nastaw z wykorzystaniem funkcji jakości (strat) bazującej na algorytmie Taguchi. Muguo Lia i Da Liua [67] w swoim rozwiązaniu zastosowali optymalizację nastaw regulatora za pomocą odpowiednio nauczonej falkowej rekurencyjnej sieci neuronowej (ang. recurrent wavelet neural network - RWNN). Inne podejście opracowali L. Abdullah i in. [5], którzy poprawę działania regulatora PID uzyskują poprzez włączenia w pętle regulacji nieliniowego kompensatora regulatora PID. Natomiast umieszczenie w pętli regulacji bloku filtra Kalmana proponuje Weibing Wang i Pengbing Zhao [114]. Wspomniane metody doboru wymagają znajomości parametrów dynamicznych obiektu regulacji. Jest to konieczne w celu budowy i nastrojenia układu sterującego lub kompensującego działanie regulatora PID, który zminimalizuje błędy nadążania.

W celu zwiększenia dynamiki serwonapędów, stosuje się dodatkowy blok sprzężenia w przód (Feed-Forward, FF). Sygnałem wejściowym do tego bloku jest zawsze wartość zadana (wejściowa) do układu regulacji. W przypadku kaskadowego układu regulacji, każda z pętli może posiadać swój odrębny układ sprzężenia w przód [5][116]. W klasycznej postaci blok sprzężenia FF może przyjmować postać modelu odwrotnego do transmitancji obiektu regulacji (co jest możliwe jedynie w przypadku gdy układ jest odwracalny). Wartość wyjściowa z tych bloków dodawana jest do węzła sumatora sprzężenia zwrotnego. W wyniku iloczynu transmitancji bloku sprężania w przód i obiektu regulacji uzyskuje się wartość równą jeden. Tym samym prowadzi to do wyeliminowania błędów - w szczególności błędów nadążania.

Niestety, w wielu przypadkach mianownik transmitancji bloku sprzężenia w przód zawiera zera (nieredukowalne), które mogą prowadzić do oscylacji wielkości regulowanej. Rozwiązanie tego problemu zaproponował Tomizuka [105] [103] - układ regulacji z zerowym przemieszczeniem fazowym błędu nadążania (ang. Zero Phase Error Tracking Controller –

ZPETC). Ulepszony algorytm ZPETC opracował N. Uchiyama [106] oraz R.Seethaler i in. [86].

Sprzężenie w przód w znaczący sposób może pomóc w redukcji błędów nadążania, jednak poprawność implementacji tego bloku bardzo zależy od znajomości właściwości dynamicznych osi maszyny numerycznej. Drugim problemem wynikającym z istoty działania tego bloku jest sygnał wyjściowy, który wpływa na wartość sygnału sterującego. W wielu przypadkach może to doprowadzić do wymuszenia na serwonapędzie uzyskania wartości dla niego nieosiągalnych, ze względu na ograniczenia dynamiczne i mechaniczne. Regulatory serwonapędów pracują wtedy w obszarze nieliniowym (nasycenia).

W celu jeszcze lepszej minimalizacji błędów nadążania, autorzy prac naukowych proponują rozwiązania alternatywne do standardowego działania kaskadowej regulacji bazującej na układach PID. Jedną z propozycji jest zastosowanie sterowania ślizgowego (ang. Sliding mode control - SMC), które należy do typu rozwiązań sterowania odpornego (ang. robust control - RC). Sterowanie ślizgowe umożliwia poprawę działania serwonapędu, którego parametry dynamiczne nie są do końca znane lub zgrubnie przybliżone. Wartości sygnału sterującego otrzymywane są poprzez przełączanie się z jednej struktury sterowania do innej struktury sterowania. Trajektoria fazowa układu regulacji dąży do „ślizgania się" wzdłuż „granicy", która określa warunki przełączania się pomiędzy tymi strukturami sterowania. W efekcie końcowym sterowanie to umożliwia poszerzenie pasma przenoszenia danego serwonapędu, uwzględniając jego ograniczania oraz kompensację zakłóceń zewnętrznych na niego działających.

N. Uchiyama i in. zastosowali sterowanie ślizgowe w rozwiązaniach dla maszyn CNC trójosiowych [107], jak i pięcioosiowych [108]. H. Geok-Soon i in. [121] zaproponowali budowę zintegrowanego sterowania ślizgowego, w którym skupili się głównie na poprawie działania samego regulatora minimalizując efekt nadmiernej zmiany jego parametrów podczas pracy serwonapędu (ang. problem of chattering). Sencer i in. [93] zastosowali rozwiązanie, w którym sterowanie ślizgowe nie tylko minimalizuje błędy nadążania poszczególnych osi układu wieloosiowego, ale także minimalizuje błędy powstających na skutek braku synchronizacji pomiędzy tymi osiami. Zhao Pengbing i Shi Yaoyao [83] zaproponowali rozwiązanie sterowania ślizgowego, w którym skupili się na eliminacji błędów nadążania powstających na skutek występowania luzów (ang. dead-zone).

Jednym ze znaczących zjawisk nieliniowych powodujących generację błędów nadążania, występujących podczas pracy maszyny wieloosiowej jest zjawisko tarcia oraz luzy mechaniczne. Zjawiska te są trudne w identyfikacji, co wpływa na budowę poprawnego modelu matematycznego danej maszyny, a w dalszej kolejności budowę i strojenie odpowiedniego układu regulacji. Najprostszym sposobem minimalizacji wpływu tych zjawisk jest odpowiednie zaprojektowanie i budowa samej konstrukcji mechanicznej maszyny, np. zastosowanie prowadnic tocznych. Ze względu na wysokie koszty elementów konstrukcyjnych,

wielu badaczy proponuje alternatywne podejście mające ograniczyć negatywny wpływ wyżej wymienionych zjawisk na pracę maszyn. Wzrost wydajności obliczeniowej układów sterowników CNC, umożliwia budowę odpowiednich bloków kompensujących te zjawiska i ich integrację w samym sterowniku CNC.

Y.S. Tarng i H.E. Cheng [101] proponują budowę analitycznego modelu maszyny numerycznej z uwzględnieniem zjawiska tarcia typu „stick-slip". Następnie w wyniku symulacji dokonują korekty ustawień regulatora prędkości serwonapędów każdej z osi numerycznych. X. Mei i in. [73] opracowali podobne rozwiązanie, ale z wykorzystaniem bardziej złożonego modelu tarcia. Alternatywnym podejściem wykazali się K. Zhang i in. [127]. Zaproponowali oni minimalizację wpływu zjawiska tarcia poprzez budowę bloku kompensującego w postaci układu Feed Forward. Podobne rozwiązanie opracowali X.C. Xi [122]. Rozwinięcie tej idei wykorzystali Z. Jamaludin i in.[52] oraz N.A Rafan i in.[85] jednak w obu przypadkach zastosowali bardziej złożony model Maxwella (ang. General Maxwell-slip - GMS). Nieszablonowym podejściem wykazali się I.B. Tijani i R. Akmeliawati [102], proponując wykorzystanie nieliniowej funkcji tarcia (zależnej od prędkości), w której dobór współczynników dokonywany jest za pomocą algorytmu v-SVR (ang. v-Support Vector Regression). L. Wang i in. [113] zamodelowali tarcie jako proces dynamiczny - w postaci modelu LuGre.

Jedną z metod na minimalizację błędów konturu jest zastosowanie sterowania skośnie sprzężonego – CCC (ang. Cross Coupling Control). Ideę takiego sterowania jako pierwszy zaproponował Y. Koren [63] w roku 1992. W sterowaniu tym można wyróżnić dwa główne człony funkcyjne: blok modelowania błędu konturu oraz blok sterowania. Zadaniem bloku modelowania jest na podstawie aktualnych błędów nadążania (każdej z osi) wyliczanie aktualnego błędu konturu. Wartość błędu wykorzystywana jest w bloku sterowania CCC do obliczenia poprawek położenia zadanego dla każdej z osi, w celu minimalizacji błędu konturu. Wyliczone poprawki niekoniecznie wpływają na zmniejszenie błędów nadążania w poszczególnych osiach mechanicznych.

Klasyczną implementację sterowania CCC zastosowali Y.S. Shieh i in. [96] oraz Y.T. Shih i in. [97]. W tych rozwiązaniach kompensacja błędu konturu następuje w wyniku wyliczenia poprawek do zadanego położenia, na podstawie aktualnych błędów nadążania osi mechanicznych. Pewną innowację zaproponowali Y. Altintas i M.R. Khoshdarregi [6], w której uwzględnili i zminimalizowali wpływ występowania efektów drgań maszyny wieloosiowej, na skutek braku synchronizacji w wykonywaniu przemieszczeń pomiędzy osiami mechanicznymi. Z.M. Yeh i in. [125] w bloku sterowania CCC zaimplementowali elementy logiki rozmytej. J.K. Chin i in. [20] zaproponowali rozwiązanie, w którym blok modelowania błędu konturu umożliwia jego predykcję, co przekłada się na wcześniejszą reakcję układu sterowania CCC. Predykcja błędu konturu wyliczana jest pod warunkiem, że zadana trajektoria ruchu

jest złożona z przemieszczeń liniowych i łuków. J.K. Chin i in. [19] dokonali ulepszenia tego rozwiązania poprzez implementację elementów logiki rozmytej. M.T. Yan i in. [124] zaproponowali parametryczne prawo sterowania w układzie sterownika CCC. Parametry dobierane były w wyniku znalezienia minimum zdefiniowanej wcześniej funkcji kosztów. F. Huo i A.N. Poo [47] zbudowali działający on-line układ sterowania CCC, w którym błąd konturu estymowany był na podstawie aktualnych błędów nadążania i pozycji zadanych poszczególnych osi mechanicznych. C.S. Chen i L.Y. Chen [17] opracowali układ sterowania CCC, który bezpośrednio kompensuje trajektorię zadaną. Osiągane jest to w wyniku predykcji błędów nadążania otrzymanych na podstawie działania opracowanego wcześniej modelu matematycznego maszyny. M. Rahaman i in. [86] zintegrowali sterowanie CCC z układem wykrywającym opóźnienia pomiędzy osiami numerycznymi, w celu ich kompensacji. J.R. Conway i in. [25] zaproponowali rozwiązanie, w którym błąd konturu może być wyliczony dla trajektorii ruchu opisanej krzywą parametryczną. Natomiast J. Wu i in. [120] przedstawili szybki, działający on-line algorytm wyliczania błędu konturu, bazujący na aproksymacji krzywizny trajektorii ruchu w danym punkcie. W innej pracy J. Wu i in. [119] dokonali integracji sterowania z uczeniem iteracyjnym (ang. iterative learning control - ILC)[115] w układzie sterowania CCC. H.Z. Moghadam i in. [74] opracowali „sterowanie hierarchiczne", w którym układ sterowania CCC jest dostrajany w celu zapewnienia najpierw najmniejszych błędów nadążania dla każdej z osi, a następnie zmniejszenia błędu konturu. F. Huo i in. [49] zbudowali układ CCC dla dwóch osi, w którym uwzględniono analityczny model maszyny do obliczania aktualnych oraz predykcji przyszłych błędów konturu. W celu zmniejszenia nakładu obliczeniowego wykorzystano metodę Taylora (ang. Taylor series expansion error compensation - TSEEC) do aproksymacji tego błędu.

Oprócz metod bezpośrednio wpływających na zmniejszanie błędów nadążania w osiach oraz błędu konturu, istnieją metody pośrednie wpływające na ich minimalizację. Jedną z tych metod jest kompensacja trajektorii zadanej. W szczególności wyróżnić tu można sterowanie powtarzalne (ang. repetitive control) oraz kompensacje wartości zadanych posuwów osi numerycznych (ang. path precompensation).

Sterowanie powtarzalne [115] przeznaczone jest w ogólności do procesów, w których sygnał sterujący ma charakter periodyczny. W przypadku maszyn wieloosiowych ograniczałoby się to do powtarzania tej samej trajektorii ruchu. W tym sterowaniu dąży się do osiągnięcia sygnału sterującego, który minimalizuje błąd w kolejnych powtórzeniach procesu, poprzez dodanie odpowiedniej poprawki do samego sygnału sterującego. Wyliczane poprawki bazują na wartościach błędów z wcześniejszych procesów. Taki rodzaj sterowania dla maszyny wieloosiowej bazującej na silnikach liniowych zaproponowali S.L. Chen i T.H. Hsieh [18], którzy zintegrowali je wraz z układem sprzężenia w przód (FF).

Podobną metodą, do sterowania powtarzalnego, jest sterowanie z uczeniem iteracyjnym (ILC)[115]. W tym wypadku nie jest wymagane aby sygnał sterujący miał charakter periodyczny, ale był on związany ze skończonym w czasie procesem. Wymagane jest by proces ten był powtarzany do uzyskania satysfakcjonującego poziomu błędu. Tak jak w sterowaniu powtarzalnym, zmianie podlega sygnał sterujący modyfikowany na podstawie błędów z aktualnego procesu jak i z błędów z procesów wcześniejszych. D.I. Kim i S.Kim [61] zaproponowali integrację ICL w sterowaniu PID w układzie maszyny CNC. Natomiast H.S. Li i in.[66] zintegrowali ICL z układem CCC.

W metodach kompensacji wartości zadanych posuwów osi numerycznych dochodzi do modyfikacji trajektorii ruchu w wyniku czego ograniczane są błędy nadążania i konturu. Uzyskuje się to poprzez zwiększanie lub obniżenie chwilowej prędkości danej osi numerycznej (w tym wartości przyspieszenia, zrywu, itp.). Y.F. Chang i in. [15] zbudowali układ sterownika CNC bazujący na sterowniku PLC z zaimplementowanym układem predykcyjnym - minimalizacji zrywu w postaci filtra (ang. look-ahead linear jerk filter - LALJF). Układ filtra, umożliwia na bieżąco modyfikację trajektorii ruchu (ograniczanie wartości zrywu) podczas pracy maszyny. Natomiast A.C. Lee i in. [65] opracowali rozwiązanie, w którym sterownik CNC w procesie interpolacji generuje trajektorię ruchu z narzuconymi ograniczeniami co do maksymalnego błędu konturu, wartości przyspieszenia oraz zrywu. Trajektoria ruchu opisana jest z wykorzystaniem krzywych NURBS (ang. Non-Uniform Rational B-Spline). W sterowniku CNC w pierwszej kolejności w trybie off-line generowany jest profil prędkości z uwzględnieniem ograniczeń i punktów krytycznych (nieciągłości geometryczne). Profil prędkości następnie dzielony jest na segmenty. W czasie pracy maszyny, segmenty te są przesyłane do bloku interpolatora, gdzie następuje generacja trajektorii ruchu (na bazie NURBS). Autorzy zaimplementowali predykcyjno-korygujący algorytm (ang. predictor–corrector interpolation - PCI), w celu efektywnego wyliczenia trajektorii z krzywych NURBS. J.X. Guo i in. [43] zaproponowali algorytm kompensacji trajektorii ruchu przebiegający w dwóch etapach. Najpierw na podstawie matematycznego modelu osi numerycznych maszyny wieloosiowej dokonywana jest predykcja błędów nadążania na podstawie zadanej trajektorii ruchu. W dalszym etapie, autorzy proponują minimalizację błędów z wykorzystaniem opracowanych nieliniowych ograniczeń dotyczących wartości prędkości oraz zrywu. W celu uproszczenia obliczeń, zaproponowana została aproksymacja liniowa, nieliniowej funkcji ograniczeń. P. Bosetti i E. Bertolazzi [13] opracowali rozwiązanie, w którym głównym zadaniem było ograniczenie błędu konturu, dla zadanej tolerancji. Na tej podstawie dobierane były ograniczenia związane z prędkością, przyspieszeniem i zrywem dla poszczególnych osi numerycznych. J. Dong i in. [30] zaproponowali rozwiązanie, które ma na celu zmniejszenie błędu konturu w przypadku gdy trajektoria ruchu składa się krótkich przemieszczeń liniowych. Uzyskano minimalizację błędu dzięki opracowanemu

filtrowi prędkości (ang. target feedrate filter - TFF). Podobnym problemem zajęli się W. Fan i in. [40]. Rozwiązanie polega na aproksymacji krótkich liniowych segmentów trajektorii ruchu, krzywą wielomianową trzeciego stopnia.

Wiele z metod zmniejszania błędów nadążania proponowanych w literaturze polega na zastosowaniu nowych algorytmów regulacji w serwonapędzie. W większości praktycznych zastosowań wykorzystuje się serwonapędy komercyjne. Struktura regulacji takich serwonapędów to w przytłaczającej większości klasyczne, kaskadowo połączone regulatory PID prądu, prędkości i położenia. Możliwość modyfikacji struktury regulatorów jest przeważnie bardzo ograniczona. Praktyczne zastosowanie większości metod, w których proponowane są nowe struktury regulacji, wymagałoby opracowania dedykowanych serwonapędów. W praktycznych zastosowaniach jest to rzadko spotykane podejście ze względu na znaczące koszty wdrożenia takiego rozwiązania.

W metodach zmniejszania błędu konturu CCC, wielu autorów umieszcza zaproponowany algorytm minimalizacji błędu w zewnętrznym sterowniku CNC, do którego podłączone są serwonapędy. Opracowane algorytmy obliczają zmodyfikowane przyrosty trajektorii zadanej. Do ich wyliczenia wykorzystywane są informacje o już zaistniałych błędach trajektorii ruchu, w celu ich dalszej minimalizacji. Autorzy prac zazwyczaj proponują budowę obserwatora błędu konturu, który najczęściej bazuje na modelach osi mechanicznych maszyny CNC. W większości przypadków identyfikacja serwonapędu nie jest trudna ze względu na znaną strukturę regulacji. Natomiast identyfikacja układu przeniesienia napędu w wielu przypadkach jest problemem bardziej złożonym. Najczęściej elementy układu przeniesienia przybliżane są modelami liniowym. W procesie identyfikacji osi mechanicznych autorzy prac zmuszeni są przyjąć z góry ustaloną dokładność opracowanego modelu matematycznego. Część autorów w modelowaniu uwzględniała nawet takie zjawiska, zachodzące w osiach mechanicznych, jak tarcie typu „stick-slip” czy luzy mechaniczne. Tylko niektórzy autorzy zaproponowali modelowanie osi mechanicznych bazujące na modelach nieliniowych.

Ciekawe podejście modelowania osi mechanicznych maszyny wieloosiowej przedstawili Feng Huo i Aun-Neow Poo [48], którzy zaproponowali standardowy układ CCC. Jednak do modelowania każdej z osi wykorzystali model typu BlackBox. Do tego celu wykorzystali sieci neuronowe typu NARX (ang. ang. Neural ARX). Wartością wyjściową każdej sieci był błąd nadążania danej osi mechanicznej. Wartości błędów nadążania wykorzystano do wyznaczenia wartości błędu konturu. Feng Huo i Aun-Neow Poo potwierdzili w badaniach bardzo dobrą zgodność otrzymanych wyników sieci z błędami występującymi w osiach maszyny wieloosiowej. Sieć umożliwiała predykcję tylko o jedną wartość w przód błędu nadążania. Autorzy jednak nie przetestowali możliwości predykcji błędów nadążania przez sieci neuronowe dla dłuższego horyzontu predykcji, w celu sprawdzenia ich poprawności działania.

W wielu przypadkach korekcja zadanej trajektorii ruchu, poprzez jej kompensację, jest dokonywana ze względu na określone warunki pracy maszyny. Wliczają się w to zarówno ograniczenia związane z posuwem osi mechanicznych, jak i wymogi procesu technologicznego. Modyfikacja trajektorii ruchu dokonywana jest najczęściej tylko w ramach modyfikacji zadanych profili prędkości, przyspieszenia i zrywu. Algorytmy kompensacyjne nie bazują na bezpośredniej modyfikacji przyrostów trajektorii zadanej. W wielu rozwiązaniach zmniejszenie błędów nadążania czy błędu konturu osi mechanicznych następuje pośrednio na skutek tych modyfikacji. Często algorytmy sterowania kompensacji trajektorii zadanej są złożone i wymagają dużego nakładu obliczeń numerycznych. W wielu przypadkach obliczenia dokonywane są w trybie „off-line". Bardzo często dotyczy to trajektorii ruchu opisanych za pomocą krzywych NURBS.

Metody korekcji zadanej trajektorii ruchu poprzez jej kompensację mają zastosowanie przede wszystkim w obrabiarkach, w których występują duże prędkości i przyspieszenia suportów (wycinarki laserowe, plazmowe itp). W maszynach tych występują duże błędy nadążania związane z dynamiką układu posuwu. Tego typu metody nie mają dużego zastosowania w maszynach CNC służących do obróbki skrawaniem twardych materiałów (np. stali). W maszynach tego typu stosuje się niskie prędkości i przyspieszenia suportu, które narzuca sam proces technologiczny. Z tego względu błędy odtwarzania trajektorii wynikają głównie ze zmiennego działania sił skrawania. Korekcja tych błędów wymagałaby modelowania procesu skrawania dla danego materiału i narzędzia [88] [69] [56].

Przedstawione powyżej koncepcje modelowania osi mechanicznych oraz kompensacji trajektorii zadanej, skłoniły autora do podjęcia próby rozwiązania problemu minimalizacji błędów odtworzenia zadanej trajektorii ruchu poprzez wykorzystanie nieliniowego modelu układu posuwu osi numerycznej z uwzględnieniem ograniczeń maksymalnych wartości prędkości i przyspieszenia.

1.2. Cel, teza i założenia pracy

Poprawa jakości pracy maszyny wieloosiowej możliwa jest poprzez zwiększenie dokładności odtwarzania zadanej trajektorii ruchu. Jednym z rozwiązań jest optymalizacja przemieszczeń wyznaczonych przez sterownik CNC, przesyłanych do serwonapędów osi mechanicznych.

W optymalnej generacji przemieszczeń osi mechanicznych powinny być uwzględnione właściwości dynamiczne maszyny (bezwładność i tarcie w układzie mechanicznym, maksymalna moc napędów, pasmo przenoszenia układu regulacji). Poprawna praca serwonapędów wymaga odpowiedniego doboru nastaw regulatorów w zależności od implementacji (w niektórych rozwiązaniach serwonapędów zmiana tych nastaw podczas

eksploatacji maszyny jest niemożliwa). Często zakłada się liniowość układu mechanicznego oraz układu regulacji serwonapędu. Niekiedy jest to założenie błędne, co przekłada się na powstawanie błędów odtworzenia trajektorii ruchu. Celowe jest uwzględnienie tych nieliniowości w sposób kompleksowy, poprzez ich modelowanie. Ze względów praktycznych, modelowanie danego zespołu mechanicznego odbywa się niekiedy w układzie pracującym (układ zamkniętych pętli sprzężeń zwrotnych). Często stosowanymi wtedy modelami matematycznymi są modele typu Black Box[68].

W wielu przypadkach dobór właściwych parametrów regulatorów jest dokonywany jednorazowo (etap uruchamiania i testowania maszyny u producenta), bez korekcji tych parametrów, w czasie późniejszej pracy maszyny. W niektórych układach serwonapędowych nie ma zaimplementowanych regulatorów sprzężeń do przodu, a ich bezpośrednia implementacja jest niemożliwa. Minimalizacja błędów nadążania dokonywana może być wówczas poprzez korekcję położenia zadanego.

W rozprawie proponuje się zastosowanie algorytmów zmniejszających błędy nadążania w osiach maszyny CNC, podczas odtwarzania zadanej trajektorii ruchu. W celu znalezienia wartości minimalnej funkcji kryterialnej jakości sterowania (funkcji celu), wykorzystano algorytm optymalizacji rojem cząstek (ang. Particle Swarm Optimization)[21]. W procesie optymalizacji zastosowano nieliniowe modele poszczególnych osi mechanicznych do predykcji błędów nadążania. Jako modele osi mechanicznych wykorzystano sztuczne sieci neuronowe typu NARX.

Sformułowano następującą tezę rozprawy:

Możliwe jest zmniejszenie błędów nadążania w osiach maszyny sterowanej numerycznie, podczas realizacji zadanej trajektorii ruchu, poprzez modyfikację zadanych przyrostów położenia osi mechanicznych maszyny z wykorzystaniem predykcyjnej procedury optymalizacyjnej oraz neuronowego modelu maszyny.

Celem rozprawy jest dokonanie analizy pracy maszyny CNC w aspekcie realizacji zadanej trajektorii ruchu oraz opracowanie nowych, oryginalnych struktur i metod sterowania minimalizujących błędy nadążania w maszynie dwuosiowej. Sformułowane zostały następujące zadania badawcze i konstrukcyjne:

— opracowanie neuronowych modeli układu posuwu osi mechanicznych maszyny numerycznej, zdolnych do predykcji błędów nadążania;
— opracowanie oryginalnego algorytmu wyznaczania optymalnych przemieszczeń dla serwonapędów maszyn CNC;

— zaprojektowanie i budowa sterownika CNC, zaimplementowanego w komputerze PC z systemem czasu rzeczywistego Linux RTAI, komunikującego się serwonapędami PMSM za pośrednictwem magistrali Ethernet POWERLINK;
— implementację proponowanego algorytmu w opracowanym sterowniku CNC;
— weryfikację działania zaproponowanego rozwiązania poprzez badania doświadczalne;

Algorytm modyfikacji posuwów osi mechanicznych wykorzystuje model NARX do modelowania dynamiki maszyny wieloosiowej. Opracowany algorytm sterowania zaprojektowano dla maszyn wieloosiowych o zespołach osi mechanicznych w układzie kartezjańskim. Przyjęto, że obciążenie osi mechanicznej wynikające z działania sił skrawania na narzędzie jest pomijalne a dominujący wpływ na błędy nadążania ma dynamika osi mechanicznych. Założenie to jest spełnione dla maszyn typu wycinaki laserowe, plazmowe itp. Zakłada się, że parametry dynamiczne i statyczne układu regulacji są nieznane. Weryfikacja poprawności działania algorytmu została przeprowadzona na maszynie, wyposażonej w dwie osie mechaniczne w układzie kartezjańskim. W każdej z osi, liniowe przemieszczenie realizowane jest za pomocą przekładni śrubowej tocznej, połączonej z wałem silnika poprzez sprzęgło. W obu osiach wykorzystano napędy PMSM. Pomiar położenia i prędkości dokonywany jest na wale silnika poprzez odczyt wartości z przetwornika obrotowo-impulsowego zintegrowanego z silnikiem. Zakłada się wysoką sztywność i minimalne luzy przekładni śrubowej tocznej, a błędy z nimi związane uznaje się za pomijalnie małe. Błędy nadążania poszczególnych osi wyznaczane są na podstawie wartości z przetworników obrotowo-impulsowych silników. Jest to rozwiązanie bardzo często spotykane w rozwiązaniach komercyjnych.

Rozprawa zawiera siedem rozdziałów oraz wykaz literatury. W rozdziale pierwszym umieszczono wprowadzenie wraz z przeglądem literatury związanym z tematem pracy. Przedstawiono problem badawczy oraz umotywowano jego podjęcie przez autora. Określono cel, tezę i założenia pracy. W rozdziale drugim omówiono zagadnienia związane z architekturą i działaniem układów sterowania numerycznego maszyn wieloosiowych. Przedstawiono proces generacji trajektorii ruchu dla układu posuwów osi mechanicznej oraz błędy nadążania. W rozdziale trzecim zaprezentowano model układu posuwu z przekładnią śrubową toczną. Przedstawiono zaproponowany przez autora nieliniowy model neuronowy układu posuwu osi mechanicznej z przekładnią śrubową toczną. W rozdziale czwartym przedstawiono predykcyjny algorytm minimalizacji błędów nadążania układu posuwu osi mechanicznej. Zaprezentowano wyniki badań symulacyjnych proponowanego algorytmu. Rozdział piąty zawiera opis stanowiska laboratoryjnego. W rozdziale szóstym omówiono wyniki badań opracowanego algorytmu, które potwierdzają tezę pracy. W ostatnim rozdziale siódmym zaprezentowano wnioski końcowe oraz osiągnięcia autora.

2. Sterowanie numeryczne maszyn wieloosiowych

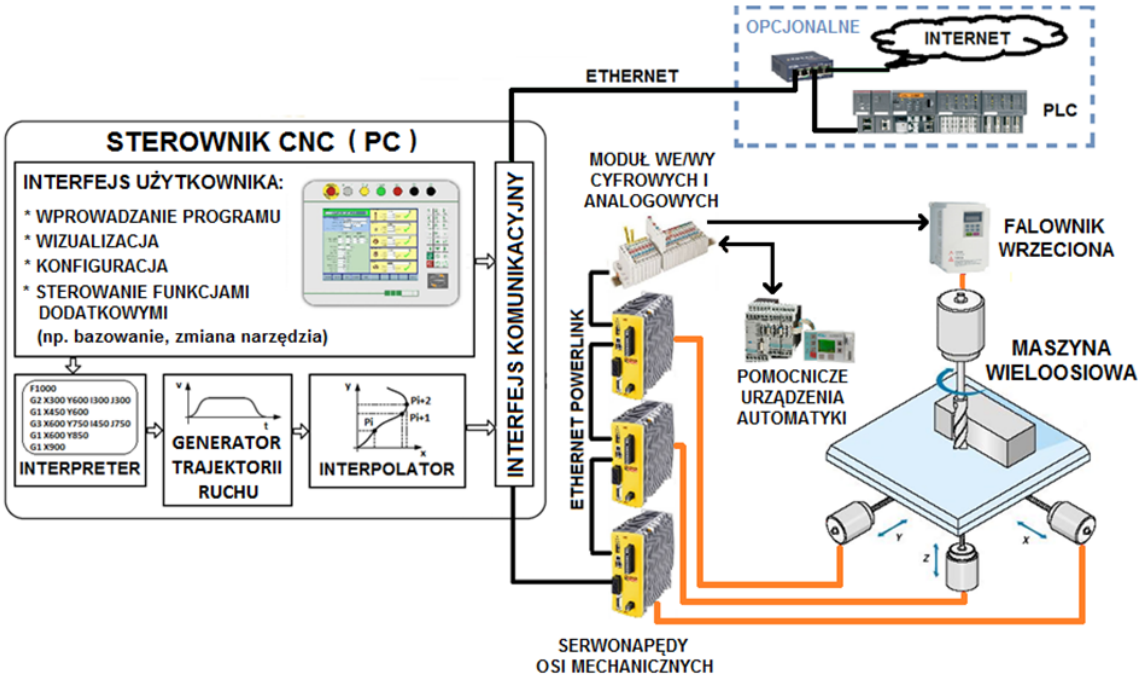
W obecnie stosowanych układach sterowania CNC coraz więcej jest rozwiązań typu otwartego (ang. open equipment manufacturing - OEM), w których odchodzi się od modyfikacji czysto sprzętowych, na rzecz modyfikacji typowo softwarowych. Niebagatelną rolę odegrały tu komputery przemysłowe PC. Umożliwiają one indywidualne dostosowanie funkcjonalności oraz oprogramowania do danej konstrukcji maszyny wieloosiowej. Przykładem rozwiązań bazujących na otwartym sterowaniu CNC jest sterowanie adaptacyjne (ang. adaptive control - AC), które uwzględnia parametry mechaniczne i dynamiczne maszyny wieloosiowej podczas jej pracy. W tej koncepcji sterowania dąży się do minimalizacji błędów odtwarzania zadanej trajektorii ruchu oraz poprawy wydajności procesu obróbki w trakcie jego trwania.

2.1. Struktura otwartych układów sterowania CNC

We współczesnych strukturach otwartych układów sterowania CNC można wyróżnić część wykonawczą, czyli układy osi mechanicznych wraz z napędami oraz sterownik CNC. Przykładową architekturę maszyny sterowanej numerycznie ze sterownikiem CNC zaimplementowanym w komputerze klasy PC przedstawiono na rysunku 2.1 .

Sterownik CNC ma bezpośredni wpływ na realizację zadanej trajektorii ruchu przez narzędzie wykonawcze. Zadana trajektoria ruchu określona jest w programie obróbki. W budowie sterownika CNC wyróżnić można dwa główne bloki funkcyjne: interfejs użytkownika oraz tor generacji zadanej trajektorii ruchu.

W bloku interfejsu użytkownika następuje wprowadzenie programu wykonawczego. Operator obsługujący maszynę wieloosiową w większości rozwiązań poprzez interfejs użytkownika ma możliwość wizualizacji trajektorii zadanej oraz konfiguracji maszyny. Dotyczy to parametrów bezpośrednio związanych z trajektorią ruchu, np. prędkości przestawczych osi maszyny, procesu bazowania maszyny, ustalanie punktu początkowego trajektorii ruchu w przestrzeni ze względu na wymiary materiału roboczego. Konfigurowalne są także parametry narzędzia wykonawczego (np. średnica frezu), prędkość wrzeciona, dobór oraz wymiana narzędzi podczas pracy maszyny. Interfejs użytkownika pozwala także operatorowi na stałe monitorowanie maszyny podczas jej pracy (w tym możliwość wstrzymania jak i wznowienia pracy maszyny w dowolnym momencie).



Rysunek 2.1. Schemat przykładowego układu sterowania numerycznego CNC

Proces wyznaczania przemieszczeń realizowany jest wieloetapowo. Następuje to w trzech kolejnych blokach funkcyjnych: interpretera kodu numerycznego, generatora trajektorii ruchu (układu profilowania prędkości) oraz interpolatora.

Najpierw następuje interpretacja kodu maszynowego (programu obróbki) dostarczonego do układu sterowania maszyny CNC. Opis trajektorii ruchu przedstawiony jest w postaci kodu numerycznego. Program obróbki zdefiniowany jest najczęściej w języku „G-code" [3]. W podstawowej formie język „G-code" umożliwia opis trajektorii ruchu za pomocą odcinków liniowych oraz łuków. Opis trajektorii w takim wypadku składa się z przemieszczeń liniowych (ruchy robocze i przestawcze) lub w kształcie łuków. Dodatkowo w kodach G mogą wystąpić dodatkowe znaczniki opisujące parametry procesu obróbki (prędkość posuwu, układy odniesienia, tolerancję wykonania, jednostki odległości). W celu odwzorowania bardziej skomplikowanych trajektorii ruchu (niemożliwych do opisania tylko za pomocą przemieszczeń liniowych i kołowych) stosuje się modyfikacje kodu G. Wielu producentów rozszerzyło podstawowy standard języka „G-code" o kodowanie opisu trajektorii za pomocą krzywych wielomianowych NURBS [100]. Przekłada się to na możliwość odwzorowania bardziej złożonej trajektorii ruchu przez maszynę CNC, w przeciwieństwie do trajektorii opisanej z wykorzystaniem podstawowego języka „G-code".

Stosunkowo nowym podejściem, w opisie trajektorii ruchu maszyn wieloosiowych, jest wykorzystanie standardu STEP-NC [100]. Standard ten poza podstawowymi danymi,

określającymi trajektorię ruchu, umożliwia zapis np. informacji związanych z technologią obróbki.

Zinterpretowane dane przemieszczeń przekazywane są do układu profilowania trajektorii ruchu. Na tym etapie wyznaczana jest trajektoria ruchu w przestrzeni trójwymiarowej. Następnie do zadanego toru ruchu dopasowywana jest trajektoria narzędzia wykonawczego (profil prędkości stycznej do toru ruchu). Uwzględniane są ograniczenia związane z maksymalnymi parametrami ruchu danych osi mechanicznych. Ograniczenia wynikają z parametrów dynamicznych każdej z osi, w szczególności bezwładności osi, uniemożliwiającej uzyskanie skokowej zmiany prędkości posuwu. Ograniczenia ruchu poszczególnych osi rzutują na docelową prędkość wykonania trajektorii ruchu, realizowaną przez maszynę wieloosiową.

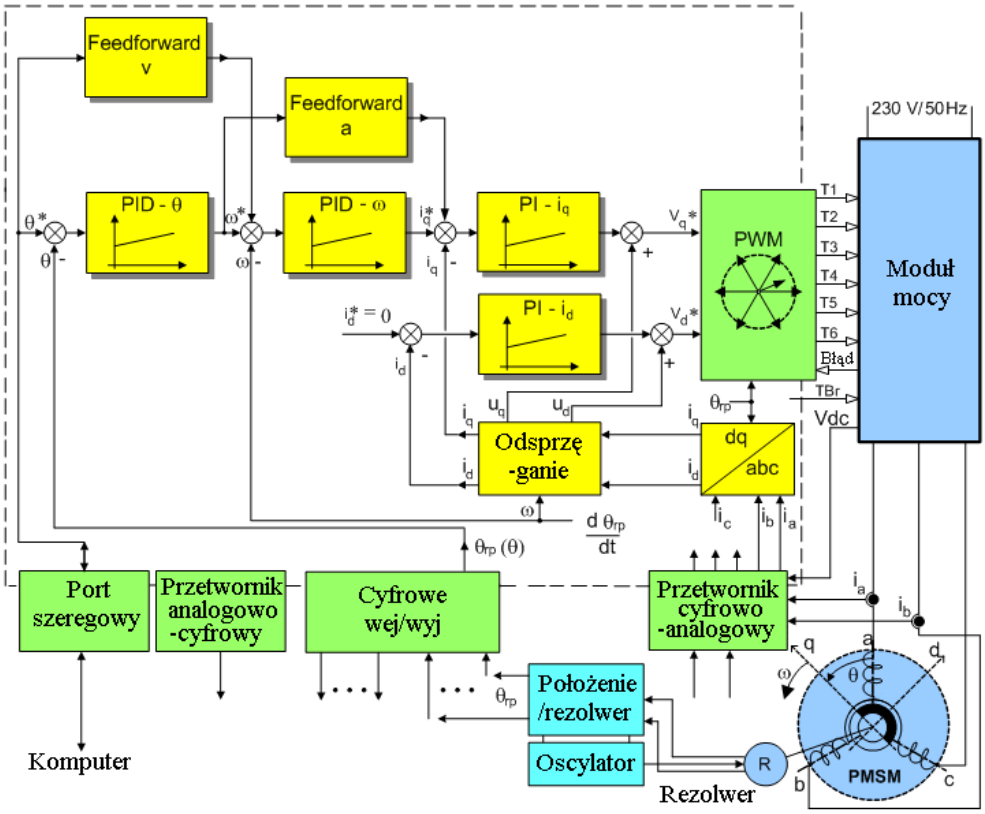
Ostatnim etapem jest interpolacja zadanego toru ruchu zgodnie z wygenerowaną trajektorią ruchu. Ponieważ kolejne przemieszczenia wysyłane są do napędów w stałych odstępach czasu, konieczna jest dyskretyzacja wygenerowanej trajektorii ruchu. Wartości prędkości stycznej do toru ruchu, w każdej z dyskretnych chwil czasu, są rozkładane na przemieszczenia w poszczególnych osiach mechanicznych maszyny. Wartości przemieszczeń są przesyłane do napędów jako wartości zadane położenia.

Informacje ze sterownika CNC przesyłane są poprzez interfejs komunikacyjny do układów wykonawczych. Najczęściej jako układy wykonawcze stosowane są cyfrowe serwonapędy. Ze względu na potrzebę nadzoru nad serwonapędami podczas pracy oraz weryfikację aktualnego położenia osi mechanicznej, wymagane jest przesyłanie informacji zwrotnej do układu sterownika CNC. W przedstawionych interfejsach komunikacyjnych nie ma takiej możliwości. Obecnie w celu spełnienia tego wymogu stosuje się magistrale szeregowe. Coraz częściej w rozwiązaniach napędowych magistrale te bazują na interfejsie Ethernet. W celu zapewnienia niskiego rozrzutu czasowego (jitter'u) cyklu komunikacyjnego, stosuje się takie rozwiązania bazujące na Ethernet jak: EtherCAT[117] czy Ethernet Powerlink[117].

W nowoczesnych maszynach wieloosiowych, jako układy napędowe, coraz częściej stosowane są cyfrowe serwonapędy z silnikami PMSM (ang. Permanent Magnet Synchronous Motor). Powszechnie stosowaną strukturą regulacji serwonapędów z silnikiem PMSM jest struktura sterowania polowo-zorientowanego (Field Oriented Control – FOC). W tym rozwiązaniu występują dwa tory regulacji, a kaskadowo połączone człony dotyczą jedynie toru z regulatorem położenia. Przykład struktury regulacji typu FOC, z kaskadowym torem regulacji położenia, przedstawiono na rysunku 2.2. W układzie najczęściej zawarte są (kolejno) regulatory położenia, prędkości i prądu - typu PI lub P[127]. Działanie kaskadowego układu regulacji może być poprawione, poprzez zastosowanie układów sprzężenia w przód (ang. Feed Forward - FF) [72] [76]. W komercyjnych serwonapędach spotykane są również inne rozwiązania, takie jak PIV [54]. W tego typu układach stosuje się sprzężenie zwrotne od wartości prądu, prędkości i położenia. Wartości prądu wyznaczane są na podstawie

pomiarów prądów w uzwojeniu silnika. Stosowane są do tego czujniki wykorzystujące zjawisko Halla. Wartości położenia wału silnika mierzone są z wykorzystaniem przetworników obrotowo-impulsowych, obrotowo-kodowych, rezolwerów lub innych rozwiązań [26]. Wartości prędkości wyznaczane są na podstawie wartości położeń.

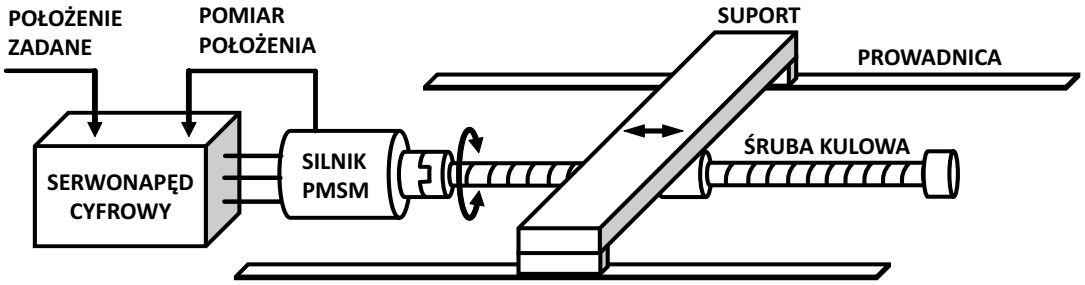
Inne regulatory takie jak: skrośne, rozmyte, neuronowe, od stanu, ślizgowe, nie są popularne w komercyjnie dostępnych serwonapędach.



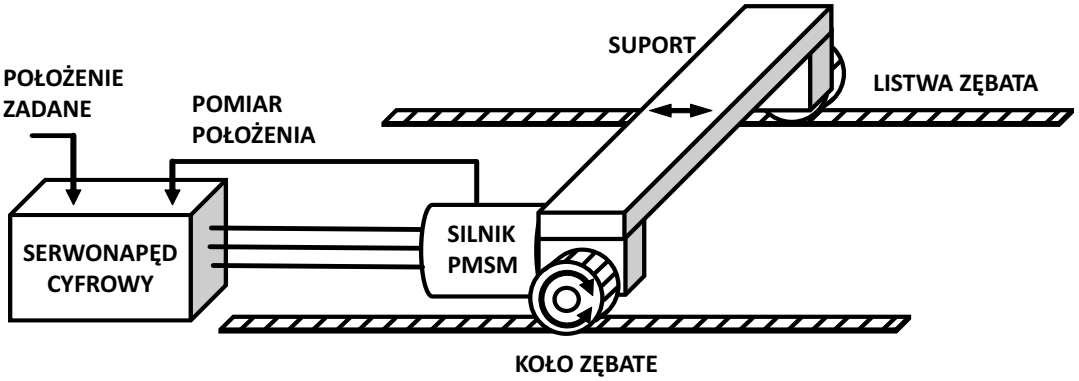
Rysunek 2.2. Przykład struktury regulacji typu FOC z kaskadowym torem regulacji położenia

Układ napędowy wraz z zespołem mechanicznym tworzy oś mechaniczną maszyny wieloosiowej. Na podstawie zadanych sygnałów sterujących, realizowane jest przesunięcie odpowiedniego elementu konstrukcyjnego wzdłuż osi prowadnic. Złożenie przesunięć osi maszyny przekłada się na ruch w przestrzeni narzędzia wykonawczego. Najczęściej w osiach mechanicznych stosowane są napędy obrotowe. Zadaniem układ przeniesienia napędu jest zamiana ruchu obrotowego silnika na ruch postępowy suportu. Do popularnych rozwiązań należą konstrukcje bazujące na przekładni śrubowej tocznej, która zamienia ruch obrotowy na postępowy (rysunek 2.3). Jeżeli ze względu na długość suportu istnieje ryzyko, że podczas pracy może dojść do wyboczenia śruby tocznej, to stosuje się inne rozwiązanie np. bazujące na wykorzystaniu przekładni zębatej (rysunek 2.4). Ze względu na dużą sprężystość oraz przenoszenie ograniczonych momentów obrotowych, do rzadszych rozwiązań należą suporty mechaniczne bazujące na przekładniach pasowych [130]. Całkowicie innym podejściem

jest zastosowanie silników liniowych, które umożliwiają realizację przemieszczeń liniowych suportu bez zastosowania układu przeniesienia napędu.



Rysunek 2.3. Suport mechaniczny wykorzystujący jako przekładnie śrubę kulową



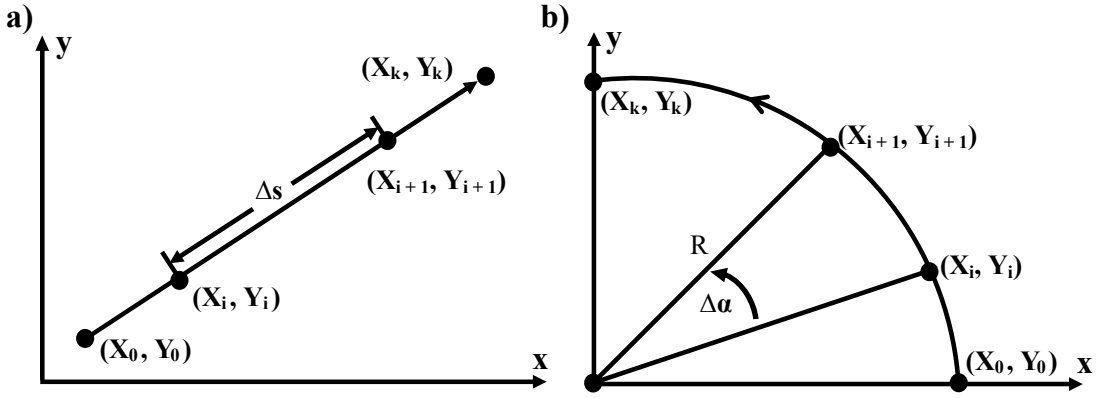
Rysunek 2.4. Suport mechaniczny wykorzystujący jako przekładnie zębatą

Dodatkowo maszyny wieloosiowe wyposażone są w struktury nadzorujące pracę narzędzia wykonawczego oraz pomocnicze układy automatyki. Opcjonalnie maszyna wieloosiowa wyposażona może być w interfejs komunikacyjny, umożliwiający wykorzystanie maszyny jako jednej z części większej linii technologicznej. Grupa maszyn wieloosiowych CNC danej linii technologicznej najczęściej nadzorowana jest z wykorzystaniem koncepcji Bezpośredniego Sterowania Numerycznego (ang. Direct Numerical Control - DNC).

2.2. Generacja trajektorii ruchu dla układu posuwu osi mechanicznej

Pomimo, że opis trajektorii ruchu przy pomocy „G-code" składa się tylko z linii prostych oraz łuków, to w procesie interpolacji, każda z osi mechanicznych dokonuje i tak krótkich przemieszczeń liniowych. Przykład generacji trajektorii liniowej i łukowej przedstawiono na rysunku 2.5.

Do ich wyznaczenia potrzebna jest informacja na temat prędkości chwilowej, otrzymanej z wcześniej wygenerowanego profilu prędkości. Przemieszczenie kątowe $\Delta\alpha$ wyznacza się z zależności:



Rysunek 2.5. Przykład generacji trajektorii liniowej(a) i łukowej(b)

$$\Delta s = v\Delta t \quad \Delta\alpha = \frac{v\Delta t}{R} \tag{2.1}$$

gdzie v - prędkość chwilowa posuwu, Δt - czas pomiędzy kolejnymi zadanymi wartościami położeń, R - promień okręgu.

W przypadku przemieszczeń liniowych Δs na płaszczyźnie, wyznacza się je z zależności:

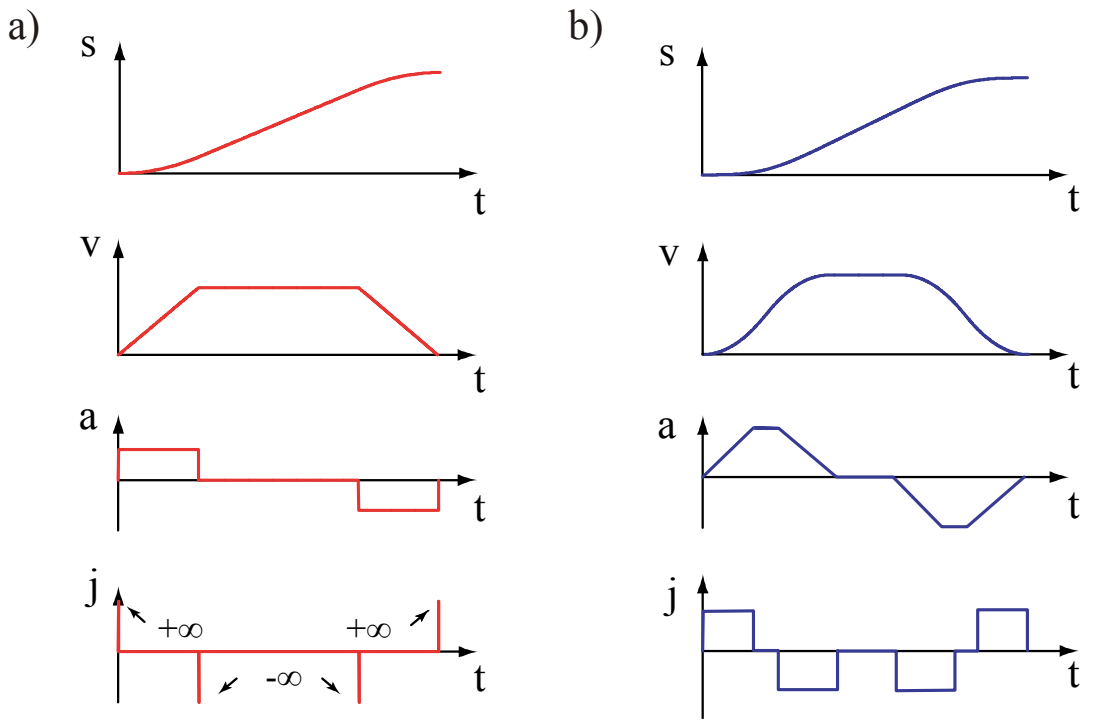
$$\begin{aligned} X_{i+1} &= X_i + \frac{(X_k - X_0)\Delta s}{\sqrt{(X_k - X_0)^2 + (Y_k - Y_0)^2}} \\ Y_{i+1} &= Y_i + \frac{(Y_k - Y_0)\Delta s}{\sqrt{(X_k - X_0)^2 + (Y_k - Y_0)^2}} \end{aligned} \tag{2.2}$$

interpolacja na płaszczyźnie odcinków łuków realizowana jest na podstawie wzoru:

$$\begin{aligned} X_{i+1} &= \cos(\Delta\alpha)X_i - \sin(\Delta\alpha)Y_i \\ Y_{i+1} &= \cos(\Delta\alpha)Y_i + \sin(\Delta\alpha)X_i \end{aligned} \tag{2.3}$$

Generacja przyrostów przemieszczeń dla poszczególnych osi mechanicznych maszyn CNC realizowana jest w sterowniku CNC (w bloku interpolatora). Przyrosty położenia wyznaczane są na podstawie profilu prędkości zadanej trajektorii ruchu.

Profilowanie trapezowe, w którym przyspieszenie zmienia się skokowo, a prędkość przyjmuje profil trapezu jest stosunkowo prostym obliczeniowo algorytmem. Zakłada się realizację trajektorii ruchu z możliwie maksymalną prędkością. Dąży się do ustawienia maksymalnego przyspieszenia na początku i końcu realizacji takiego odcinka trajektorii. Taka strategia realizacji trajektorii jest odpowiednia gdy składa się ona z długich odcinków linii prostych. W wypadku realizacji łuków i krótkich odcinków liniowych jest już strategią



Rysunek 2.6. Profile położenia, prędkości, przyspieszenia i zrywu dla a) nieograniczonej i b) ograniczonej wartości zrywu.

nieefektywną. Ze względu na trapezoidalny przyrost prędkości na krótkich odcinkach, nie jest możliwe uzyskanie maksymalnej prędkości (profil prędkości przypomina wówczas profil trójkątny). W wielu przypadkach może to doprowadzić do wystąpienia rezonansów mechanicznych, które powodują pogorszenie jakości obróbki i nadmierne zużycie elementów konstrukcyjnych maszyny CNC. Rozwiązaniem tego problemu jest profilowanie prędkości i przyspieszeń z uwzględnieniem wielu segmentów kodu trajektorii ruchu.

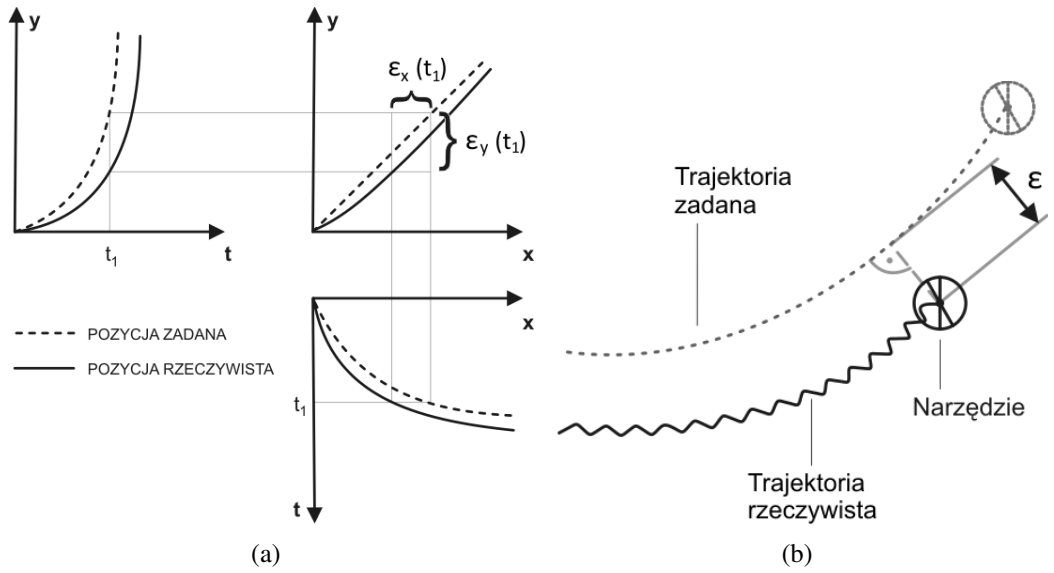
Alternatywnym rozwiązaniem profilowania prędkości, jest analiza wyższych pochodnych trajektorii ruchu - w szczególności ograniczanie wartości zrywu. Wartość przyspieszenia w takim wypadku przybiera kształt trapezoidalny, zaś prędkość jest wielomianem sklejanym drugiego rzędu. Przekłada się to na zmniejszenie składowych wysokich częstotliwości w sygnale zadanym, przekazywanym do poszczególnych osi mechanicznych. Skutkuje to zmniejszeniem drgań w układzie posuwu osi mechanicznych oraz wpływa na ograniczenie zużycia elementów mechanicznych. Rozwiązanie to (ograniczanie zrywu) wymaga jednak bardziej złożonego algorytmu generacji trajektorii ruchu. Na rysunku 2.6 przedstawiono profile położenia, prędkości, przyspieszenia i zrywu dla nieograniczonej i ograniczonej wartości zrywu.

2.3. Błędy nadążania w układzie posuwu osi mechanicznej

Trajektoria ruchu składa się z linii prostych lub łuków w przypadku opisu za pomocą „kodów G" lub krzywych wyższych rzędów (np. NURBS). Sterownik CNC odtwarza trajektorie zawsze z określoną dokładnością. W bloku interpolatora dochodzi do powstawania błędów odtwarzania trajektorii ruchu, na skutek dyskretyzacji toru ruchu. Dodatkowo same układy regulacji serwonapędów wprowadzają błędy związane z działaniem kompensacji uchybu położenia (różnica pomiędzy wartością zadaną a rzeczywistym położeniem osi mechanicznych). Układ regulacji (np. położenia) wyznacza sygnał zadany na podstawie wartości uchybu, które już zaistniały. Regulatory te w większości implementacji są regulatorami liniowymi i niekiedy nie kompensują poprawnie nieliniowości rzeczywistego układu posuwu.

W celu oszacowania parametrów odtwarzania zadanej trajektorii ruchu, przez układ sterownika CNC, wprowadzono dwa pojęcia błędów. Błędy te dzielimy na:

— Błędy nadążania (rys. 2.7(a)) – definiowane jako różnica pomiędzy pozycją zadaną a pozycją rzeczywistą danej osi mechanicznej.
— Błędy konturu (rys. 2.7(b)) – definiowane jako różnica odległości między trajektorią zadaną a trajektorią narzędzia wykonawczego.

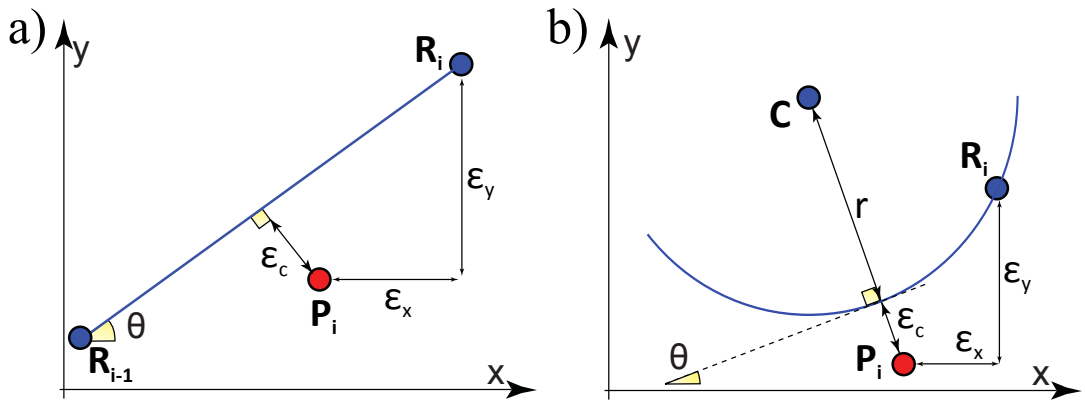


Rysunek 2.7. Błędy nadążania (a) i błędy konturu (b)

Błąd konturu zależy od błędów nadążania w poszczególnych osiach (ich uchybów) oraz kształtu zadanego toru ruchu. Błąd konturu, dla toru ruchu będącego w kształcie linii prostej, wyznaczany jest jako najkrótsza odległość pomiędzy linią a punktem (rys. 2.8a)

Dla toru ruchu leżącego na płaszczyźnie zależność można wyznaczyć ze wzoru:

$$\epsilon_c = -\sin(\theta)\epsilon_x + \cos(\theta)\epsilon_y \tag{2.4}$$



Rysunek 2.8. Błąd konturu wyznaczony dla trajektorii ruchu w kształcie a) linii prostej i b) łuku

błąd konturu dla toru ruchu opisanego za pomocą łuku lub okręgu wyznaczany jest jako różnica długości promienia okręgu i odcinka łączącego środek okręgu oraz rzeczywistego położenia suportu (rys. 2.8b). Błąd konturu w przypadku okręgu na płaszczyźnie wyznaczyć można z następującego wzoru[62]:

$$\epsilon_c = \sqrt{(r\sin(\theta) - \epsilon_x)^2 + (-r\cos(\theta) - \epsilon_y)} \tag{2.5}$$

W przypadku trójwymiarowym, wyznaczenie błędu konturu dla trajektorii ruchu w kształcie łuku jest bardziej złożone. Często rzeczywiste położenie suportu leży w innej płaszczyźnie niż sam tor ruchu. W celu obliczenia błędu wymagane jest najpierw rzutowanie punktu na płaszczyznę w której znajduje się łuk, a następnie wyliczenie składowej błędu konturu w płaszczyźnie łuku.

3. Neuronowe modele układu posuwu osi mechanicznej z przekładnią śrubową toczną

Jednym z głównych modułów zaproponowanego układu sterowania CNC, są bloki modelowania zespołu posuwu osi mechanicznych z przekładnią śrubową toczną, służące do predykcji błędów nadążania osi.

W wielu wypadkach modele osi mechanicznych buduje się w celu weryfikacji poprawności działania obranej strategii sterowania. Często spotykanym podejściem jest osobna identyfikacja samego obiektu regulacji bez uwzględnienia dynamiki regulatorów, których struktura jest znana. Model całego układu regulacji powstaje z połączenia znanej struktury regulatorów oraz zidentyfikowanego modelu obiektu regulacji [35].

Opracowanie precyzyjnych modeli matematycznych obiektów regulacji jest w wielu przypadkach trudne, gdyż są to układy dynamiczne o nieliniowościach wysokiego rzędu [75]. W celu rozwiązania tych niedogodności stosuje się uproszczenia w postaci założenia o liniowości obiektów. Jeśli przyjęcie liniowego modelu obiektu regulacji jest niewystarczające, to model uzupełniany jest wówczas o część nieliniową, która uwzględniać może tarcie nieliniowe [8][51], luzy mechaniczne [126], nachylenie osi względem podłoża [53], sztywność połączeń pomiędzy poszczególnymi elementami mechanicznymi [34], odkształcenia termiczne [29] i inne nieliniowości. W ostateczności uwzględnienie wszystkich tych zjawisk fizycznych, prowadzi do powstania złożonych modeli matematycznych wysokiego rzędu. Bardzo dobra predykcja błędów nadążania wymaga wyznaczenia wielu parametrów takiego modelu, co wiązać się może z długim procesem zbierania danych potrzebnych w procesie identyfikacji. Dodatkowo tak rozbudowany model obarczony jest często dużą złożonością obliczeniową.

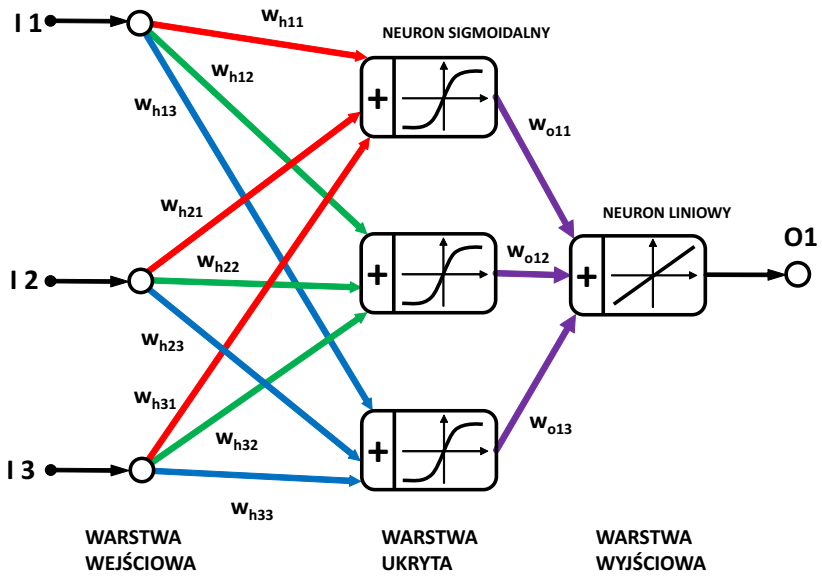
W przypadku trudności w procesie identyfikacji i modelowaniu obiektów regulacji, alternatywnym podejściem jest wykorzystanie modeli typu wejścia/wyjścia. Głównym założeniem w takim modelu jest brak wiedzy o wewnętrznej budowie obiektu regulacji - model typu Black Box [68]. Proces identyfikacji takiego modelu może bazować na danych uzyskanych podczas normalnej pracy maszyny. Najczęściej są to modele liniowe takie jak AR, ARX [60], ARMAX [104] odpowiedniego stopnia, modele transmitancyjne lub inne [59]. Proces identyfikacji tych modeli nie cechuje się nadmierną złożonością, jednak w wielu przypadkach finalny model nie dość dobrze odwzorowuje dynamikę obiektu nieliniowego, jakim jest napęd posuwu osi maszyny CNC.

Jednym z rozwiązań takiego problemu jest modelowanie nieliniowe z zastosowaniem sztucznych sieci neuronowych [27][55]. Opis matematyczny, pojedynczego neuronu dany jest zależnością:

$$y = \tanh\left(\sum_{i=1}^{n} x_i * w_i + b\right) \tag{3.1}$$

gdzie: x_i - wejścia neuronu, w_i - wagi neuronu, b - bias, n - liczba wejść neuronu.

Neurony pogrupowane są w warstwach. Wyjścia poszczególnych neuronów danej warstwy są danymi wejściowymi warstwy następnej. Sieć neuronowa może składać się z kilku warstw. Pierwsza warstwa neuronów nazwana jest warstwą wejściową, zaś warstwa ostatnia - warstwą wyjściową . Warstwy pomiędzy tymi warstwami określane są jako warstwy ukryte. W większości przypadków wykorzystywane sieci neuronowe posiadają jedną warstwę ukrytą. Sieci tego typu określane są w literaturze jako „wielowarstwowy perceptron" (ang. Multi-Layered Perceptron - MLP). Przykładowy schemat budowy sztucznej sieci neuronowej przedstawia rysunek 3.1. Każdy neuron w takiej sieci ma sigmoidalną funkcję aktywacji, prócz neuronów z warstwy wyjściowej, gdzie funkcja aktywacji jest funkcją liniową. Struktura takiej sieci umożliwia aproksymowanie funkcji nieliniowych[28] [46]. Ilość neuronów w każdej warstwie sieci neuronowej dobierana jest arbitralnie, gdyż nie istnieje żaden uniwersalny sposób optymalnego doboru ilości neuronów. W większości przypadków ilość ta jest dobierana metodą prób i błędów. Ze względu na kierunek przetwarzania danych, sieci te są nazywane także jako sieci jednokierunkowe.



Rysunek 3.1. Schemat budowy sztucznej sieci neuronowej posiadającej jedną warstwę ukrytą (trzy wejścia, jedno wyjście)

Jakość odtworzenia zadanej funkcji nieliniowej, przez sieć neuronową, zależy bezpośrednio od wartości wagowych. W celu jak najlepszej aproksymacji wartości wag, dobierane są one w procesie uczenia (tzw. uczenie z nauczycielem). Proces uczenia polega na minimalizacji wskaźnika jakości aproksymacji danych uczących. Wskaźnik ten może być dowolnie określany, jednak najczęściej przyjmuje on postać błędu średniokwadratowego (ang. mean squared error - MSE) pomiędzy wartościami danej funkcji nieliniowej (danymi uczącymi) a wartościami wyjściowymi z sieci (aproksymacją). MSE przedstawić można następującą zależnością:

$$\epsilon_{MSE} = \frac{1}{N_d} \sum_{i=1}^{N_d} (d_i - \hat{y}_i)^2 \tag{3.2}$$

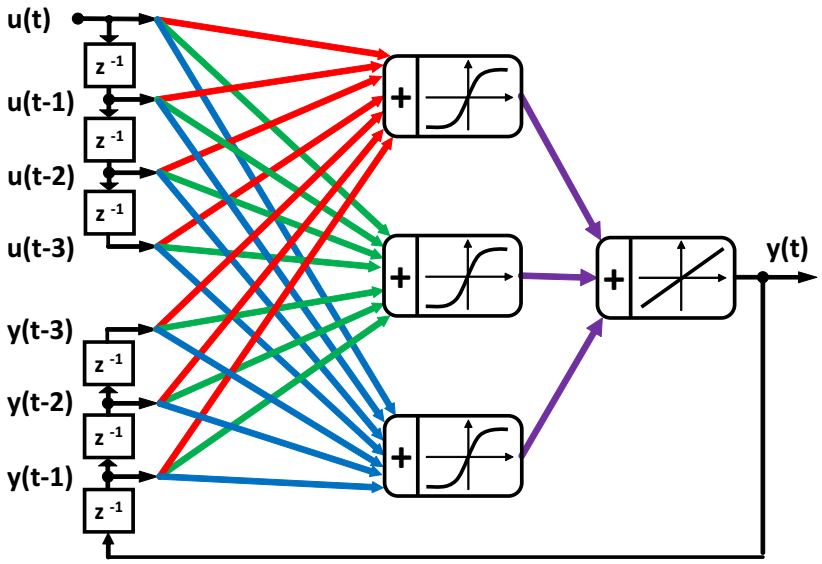
gdzie: N_d - liczba danych wykorzystanych w procesie uczenia, d_i - wartości funkcji nieliniowej (dane wykorzystane w procesie uczenia), $\hat{y}_i$ - wartości wyjściowe z sieci (aproksymacja wartości funkcji nieliniowej).

Występuje wiele algorytmów uczących sztuczne sieci neuronowe. Najpopularniejsze to algorytmy, które wykorzystują gradient wskaźnika jakości pomiędzy poszczególnymi cyklami nauczania sieci (np. algorytm Levenberga-Marquardt). Sieć neuronową powinna charakteryzować własność generalizacji, tzn sieć powinna dobrze aproksymować wartości funkcji nieliniowej, również dla argumentów (danych wejściowych), które nie były wykorzystywane w procesie uczenia. Jeśli dla tych danych sieć błędnie aproksymuje funkcję nieliniową, to sieć została błędnie „dostrojona" i nastąpił tzw. efekt „przeuczenia" sieci. Dlatego w procesie nauczania ogół danych potrzebnych do nauczenia sieci, dzielony jest na dane używane do „strojenia sieci" (zbiór danych uczących) oraz zbiór danych walidacyjnych.

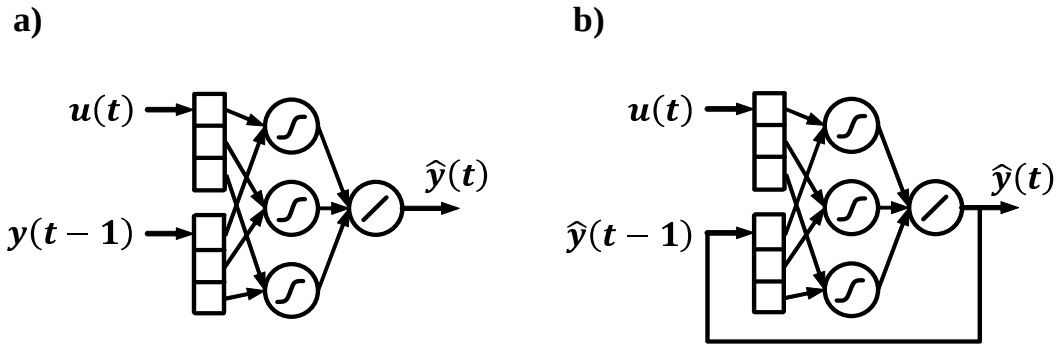
Sieć MLP umożliwia aproksymowanie funkcji nieliniowej z dowolną zadaną dokładnością. Nieliniowe obiekty dynamiczne można modelować za pomocą sieci MLP, dodając bloki opóźniające na wejściach sieci oraz sprzężenie zwrotne pomiędzy warstwą wyjściową a wejściową sieci. Sieci tego typu określane są jako sieci NARX (ang. Neural ARX)[4] [48]. Sieć typu NARX jest rozszerzeniem typowego liniowego modelu ARX, który w przeciwieństwie do niego umożliwia aproksymację nieliniowości. Schemat sieci typu NARX przedstawia rysunek 3.2. Dobór ilości warstw ukrytych, liczba neuronów w warstwach, ilość bloków opóźniających najczęściej jest dokonywany metodą prób i błędów.

W zależności od potrzeb, sieć typu NARX może być uczona w trybie szeregowo-równoległym lub równoległym [77]. Schemat poglądowy obu przypadków przedstawiono na rysunku 3.3.

W trybie szeregowo-równoległym brak jest sprzężenia zwrotnego pomiędzy wyjściem sieci a jej wejściem. Wartości wejściowe otrzymywane są z danych już zgromadzonych wcześniej. Największą zaletą tego trybu uczenia jest możliwość zastosowania tych samych algorytmów



Rysunek 3.2. Schemat przykładowej sieci neuronowej typu NARX.



Rysunek 3.3. Schemat sieci NARX uczonej w trybie szeregowo-równoległym (a) i równoległym (b).

uczenia, stosowanych dla sieci jednokierunkowych MLP [123]. Gradient wskaźnika jakości jest znacznie prostszy do wyznaczenia niż w przypadku sieci rekurencyjnych dzięki czemu proces uczenia przebiega znacznie szybciej.

Trening sieci w trybie równoległym, w przeciwieństwie do treningu szeregowo-równoległego, jest bardziej skomplikowany, gdyż wymaga wykorzystania złożonej obliczeniowo metody tzw. gradientu dynamicznego. Uczenie trwa dłużej, a ze względu na skomplikowaną funkcję błędu, występuje dość duże ryzyko zatrzymania algorytmu gradientowego w minimum lokalnym. Dodatkowo wynik treningu dla trybu równoległego sieci NARX mocno zależy od wartości początkowych wag [45].

Sieć NARX jako układ modelujący dynamikę obiektu umożliwia uzyskanie aproksymacji odpowiedzi obiektu, który modeluje w dłuższym horyzoncie czasowym. Ze względu na brak informacji wyjściowych z obiektu rzeczywistego, wartości te otrzymywane są za pomocą sprzężenia zwrotnego z wyjścia sieci NARX. Takie podejście sprawia, że wartości wyjściowe z

sieci są obarczone błędem aproksymacji. Alternatywnie sieć NARX może być zainicjalizowana danymi wejściowymi, które wcześniej zostały zgromadzone z modelowanego obiektu. W takim przypadku, sieć neuronowa będzie mogła dokonać bardzo dobrej predykcji odpowiedzi obiektu na sygnał wejściowy. Takie działanie sieci NARX jest możliwe do zastosowania w wypadku, gdy otrzymywane są na bieżąco informacje wyjściowe z modelowanego obiektu. Sieć umożliwia wtedy bardzo dobrą predykcję odpowiedzi obiektu w krótkim horyzoncie czasowym.

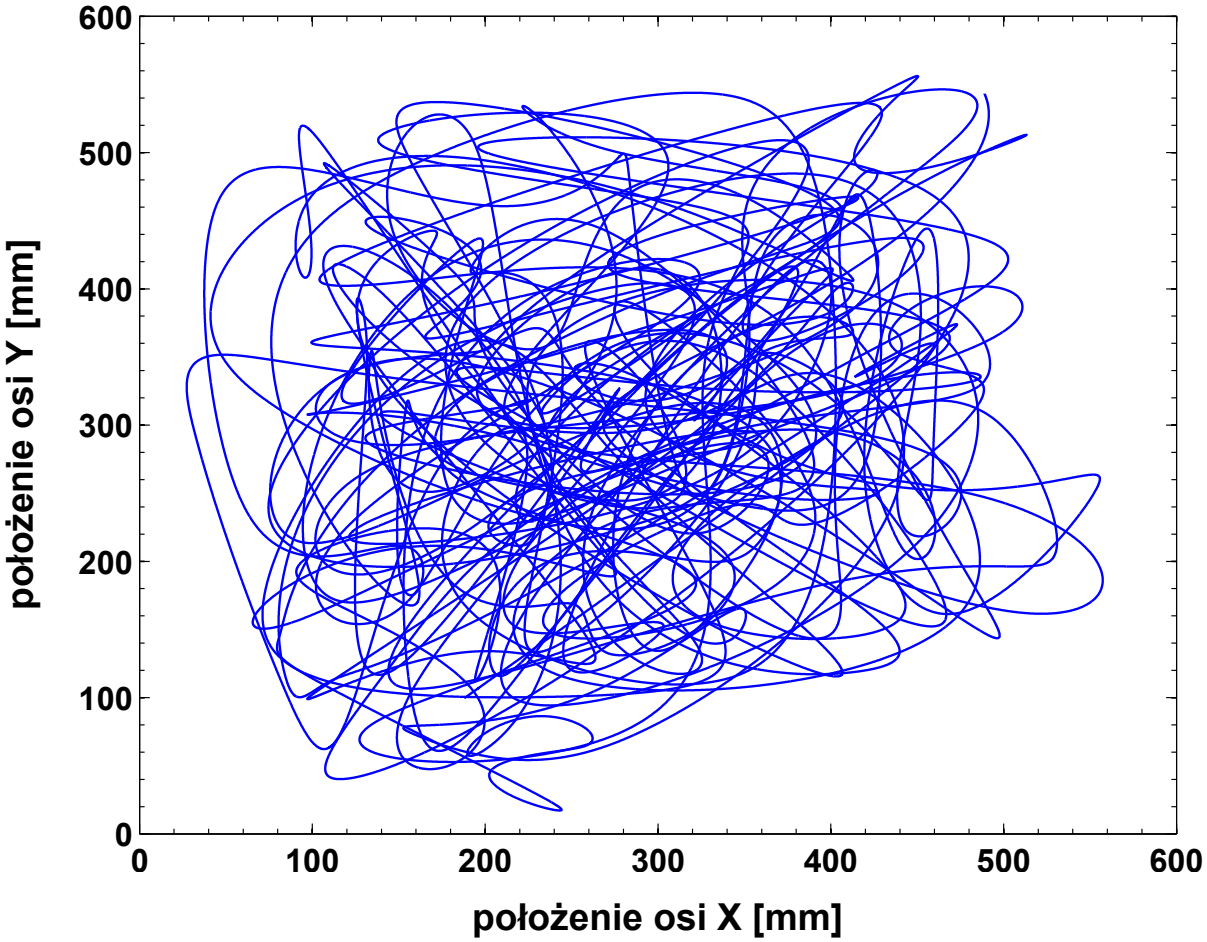
3.1. Wybór architektury i metody uczenia dla neuronowych modeli układów posuwu osi mechanicznych

W niniejszej rozprawie zostały opracowane dwa neuronowe modele układu posuwu osi mechanicznych. Każdy z modeli bazował na sieci neuronowej typu NARX, która dokonuje predykcji błędu nadążania na podstawie wartości przyrostów posuwu, obliczonych w bloku interpolatora sterownika CNC.

Sieci NARX uczone są niezależnie, na podstawie danych otrzymanych z każdej osi mechanicznej, podczas pracy maszyny wieloosiowej. Dane do treningu sieci odczytywane są z układ posuwu osi mechanicznych, które pracują w swojej domyślnej konfiguracji - w zamkniętej pętli regulacji. Nie dokonuje się zmiany ustawień w samym sterowniku CNC, jak i w oprogramowaniu serwonapędów. Podejście to ma szczególne znaczenie w rozwiązaniach praktycznych, gdzie taka modyfikacja pociąga za sobą przestoje w produkcji i z tego względu jest trudna do przeprowadzenia.

Dane treningowe, do nauki sieci NARX, otrzymano w wyniku realizacji losowego toru ruchu - rysunek 3.4. Podczas realizacji toru ruchu zadbano by profil prędkości posuwu osi mechanicznych nie przekraczał dopuszczalnych wartości prędkości (500 mm/s), przyspieszenia (2000 mm/s^2) oraz zrywu (50000 mm/s^3). Zadane profile prędkości użyte do treningu sieci NARX przedstawiają rysunki 3.5 i 3.6.

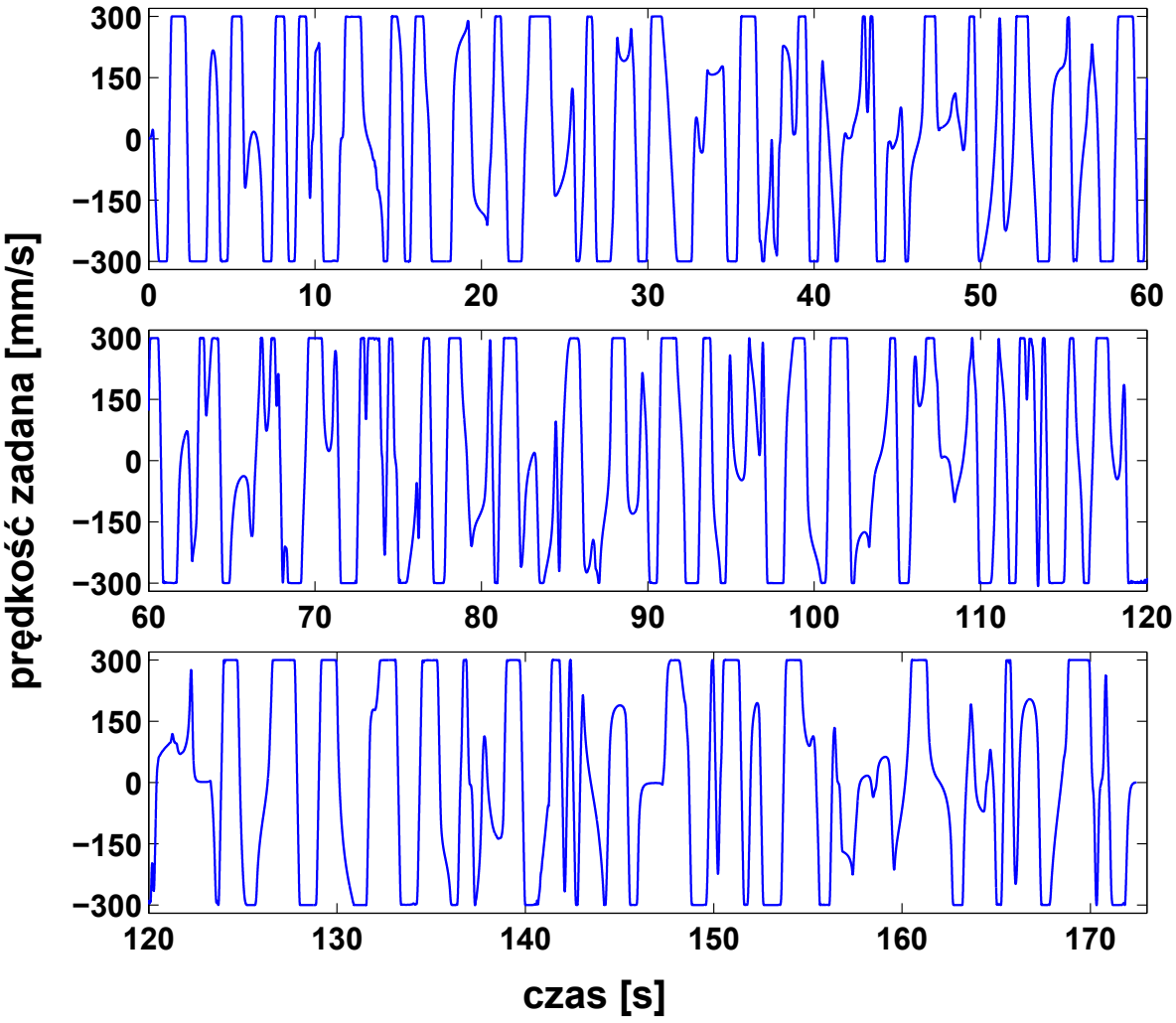
Trajektoria ruchu przeznaczona do wygenerowania danych testowych sieci NARX została odtworzona na dwuosiowej maszynie CNC. Suporty osi mechanicznych bazują na śrubach tocznych (o skokach 10 mm/obr.), które współpracują z serowonapędami Microflex e100 firmy Baldor. W serwonapędach wykorzystywane są silniki PMSM (T_n=1,5Nm ω_n=3000 obr/min), umożliwiające dokonanie pomiaru położenia suportu osi za pomocą zintegrowanych przetworników obrotowo-impulsowych (enkoderów inkrementalnych). Rozdzielczość enkoderów wynosi 2500 impulsów na obrót, co daje 10000 jednostek na obrót po przetworzeniu w napędzie (kwadratura). Każdy z serwonapędów połączony jest ze sterownikiem CNC za pomocą interfejsu komunikacyjnego Ethernet Powerlink. W sterowniku CNC z wykorzystaniem bloku interpolatora generowane są położenia zadane dla każdej z osi. Sterownik CNC zaimplementowano w komputerze PC z zainstalowanym systemem



Rysunek 3.4. Krzywa definiująca tor ruchu, który został wykorzystany do treningu neuronowych modeli układów posuwu osi mechanicznych

operacyjnym czasu rzeczywistego - Linux RTAI. Kolejne wartości posuwu przesyłane są ze Sterownika CNC do serwonapędów co 1ms. Serwonapędy, w każdym z tych cykli komunikacyjnych, odsyłają informację o aktualnych wartościach błędów nadążania. Wartości błędów zapisywane są w sterowniku CNC (w postaci pliku tekstowego). W celu wyznaczenia danych testowych do sieci NARX dalsze przetwarzanie tego pliku dokonywane jest w MATLAB.

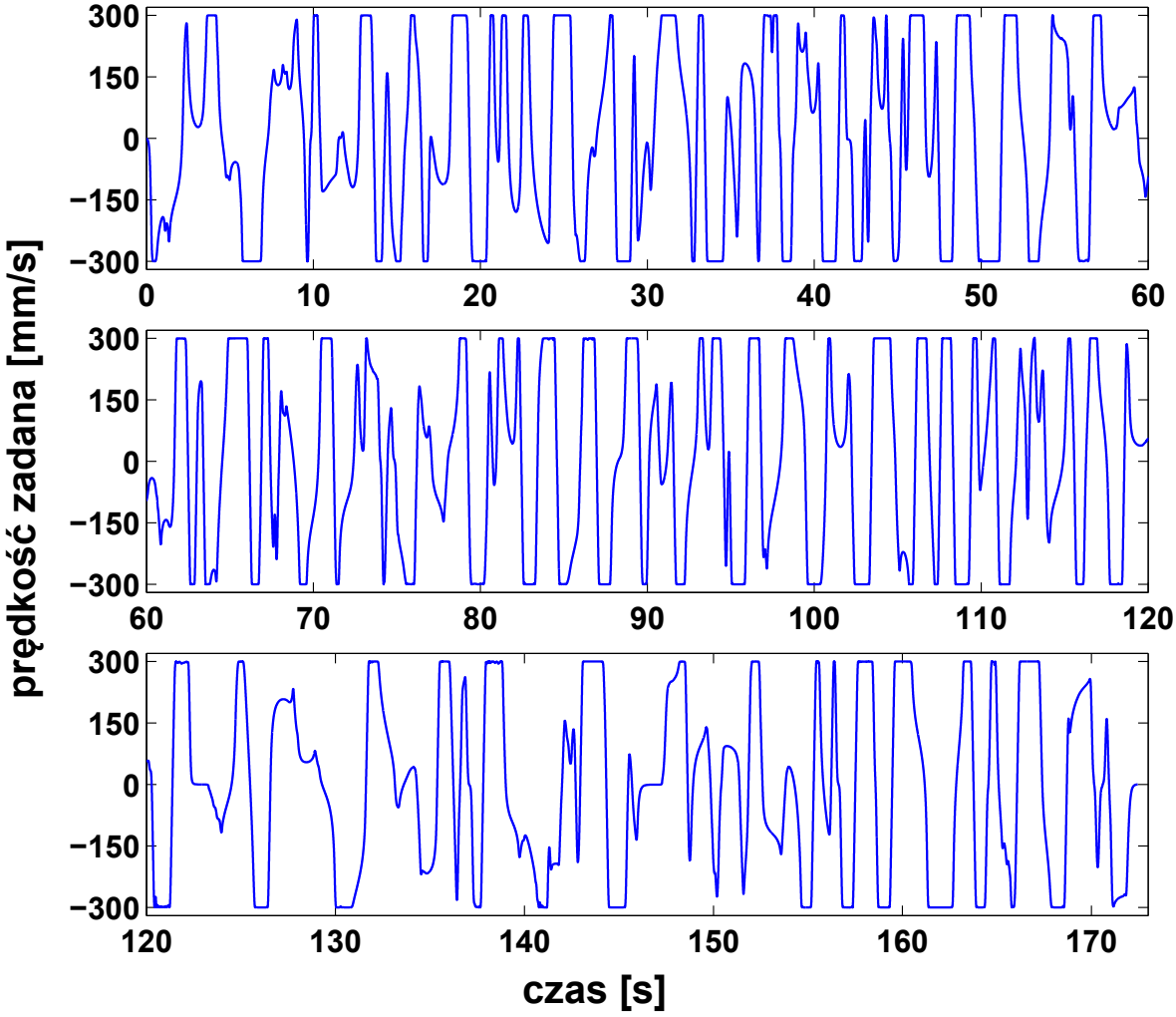
W używanej do testów maszynie pojawiły się błędy powtarzalności - występowały różnice w otrzymywanych błędach nadążania pomiędzy kolejnymi odtwarzaniami zadanej trajektorii ruchu (rys 3.4). W celu minimalizacji tego efektu, trajektoria zadana była wykonana 10 razy, a otrzymane błędy uśrednione w celu uzyskania reprezentatywnego zbioru uczącego dla sieci NARX. W uzyskanym przebiegu błędu nadążania występują nieciągłości i szum. Przyczyną jest ograniczona rozdzielczość układa pomiarowego (enkodera inkrementalnego). Taka postać danych wpływa niekorzystnie na trening sieci NARX - wymagana jest dalsza obróbka danych. W tym celu do wyników uśrednionych błędów nadążania zostały dopasowane



Rysunek 3.5. Przebieg prędkości zadanej w osi X wykorzystany jako sygnał wejściowy w procesie treningu sieci neuronowej

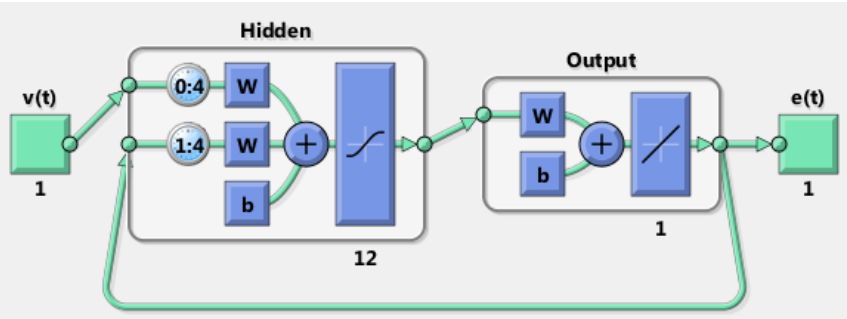
wielomiany sklejane 3-go stopnia (ang. Cubic Smoothing Spline). Dane zostały wygładzone, w szczególności zminimalizowany został efekt skokowych zmian wartości błędów. Uzyskano to za pomocą MATLAB (polecenie „csaps", toolbox „Curve Fitting Toolbox", współczynnik wygładzania $5e^{-2}$).

Treningu sieci NARX dokonano przy użyciu MATLB (toolbox „Neural Network Toolbox"). Liczba neuronów sieci oraz ilość bloków opóźniających, została określona metodą prób i błędów w procesie treningu sieci. Na podstawie wyników predykcji błędu nadążania każdej z sieci o różnych architekturach, wybrano tę o najmniejszym błędzie predykcji. Testy wykazały, że zadowalające właściwości ma sieć NARX o architekturze z jedną warstwą ukrytą, która posiada cztery bloki opóźniające na wejściu i w torze sprzężenia zwrotnego. Dwie sieci o tej samej ilości neuronów oraz bloków opóźniających, modelowały działanie dwóch osi numerycznych maszyny CNC. Wartością wyjściową każdej z sieci była wartość błędu



Rysunek 3.6. Przebieg prędkości zadanej w osi Y wykorzystany jako sygnał wejściowy w procesie treningu sieci neuronowej

nadążania każdej z osi maszyny. Schemat sieci neuronowej NARX wykorzystanej do predykcji błędów nadążania osi mechanicznej przedstawia rysunek 3.7.



Rysunek 3.7. Schemat sieci neuronowej NARX wykorzystanej do predykcji błędów nadążania osi mechanicznej(MATLB - toolbox „Neural Network Toolbox”).

W celu weryfikacji poprawności predykcji sieci neuronowych NARX, dane testowe zostały podzielone na dwa zbiory - zbiór danych uczących i walidacyjnych. Trening sieci NARX realizowany był z pomocą MATLAB (toolbox „Neural Network Toolbox") z wykorzystaniem funkcji „trainbr", w której zaimplementowano algorytm Levenberga-Marquardt'a z automatyczną regularyzacją Bayes'owską [71][42]. W takim wypadku wskaźnik jakości podlegający minimalizacji w trakcie uczenia sieci ma następującą postać:

$$\epsilon_{BR} = \beta \sum_{i=1}^{N_d} (d_i - \hat{y}_i)^2 + \alpha \sum_{j=1}^{N_w} w_j^2 \tag{3.3}$$

gdzie: N_d - liczba danych uczących, d_i - dane uczące, $\hat{y}_i$ - odpowiedź sieci, N_w - liczba wag sieci, w_j - wagi sieci, α,β - automatycznie dobierane współczynniki wagowe

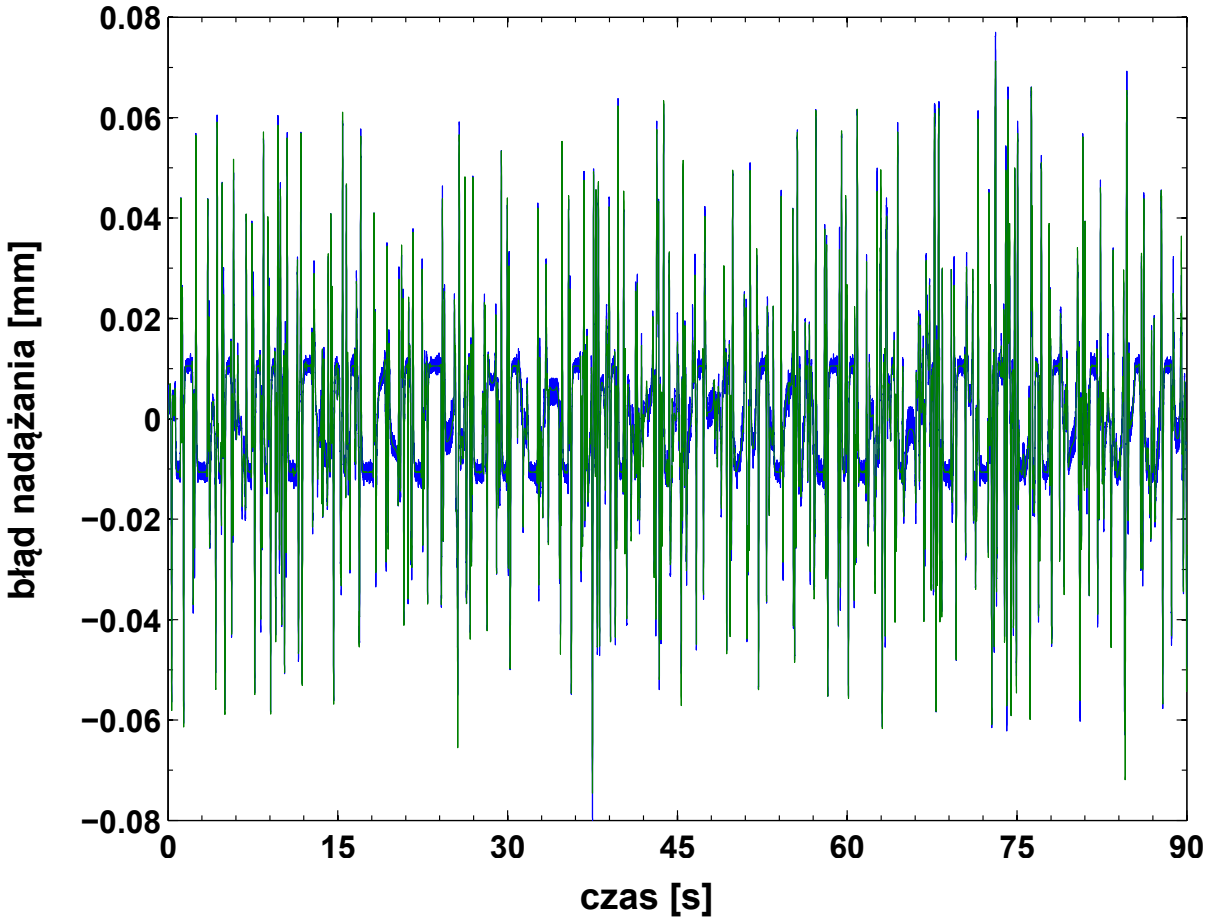
W regularyzacji Bayes'owskej, oprócz wskaźnika jakości w postaci błędu średniokwadratowego, wprowadzono dodatkowy składnik - sumę kwadratów wag sieci. W algorytmie tym dąży się do minimalizacji także tego składnika, gdyż zbyt duże wartości wag sieci w wielu przypadkach oznaczają „przeuczenie" sieci. Uczenie sieci z wykorzystaniem regularyzacji Bayes'owskiej, wymaga większej liczby iteracji uczenia sieci NARX niż z wykorzystaniem np. współczynnika jakości typu MSE.

Proces treningu sieci NARX o architekturze równoległej, ze względu na gradient dynamiczny, jest bardzo podatny na zatrzymanie się w minimum lokalnym - głównie ze względu na źle dobrane początkowe wartości wag sieci. Wyznaczone wartości wag sieci NARX w procesie treningu dla architektury szeregowo-równoległej (bez sprzężenia zwrotnego, za to z danymi rzeczywistymi na drugim wejściu do sieci) nadają się lepiej do uczenia sieci o architekturze równoległej, niż te otrzymane w wyniku losowej inicjalizacji. Dodatkowo uczenie sieci szeregowo-równoległej, umożliwia zastosowanie wydajnych obliczeniowo metod treningu znanych z sieci typu MLP. Dlatego trening sieci NARX odbywał się dwuetapowo. Najpierw poddano uczeniu sieci NARX o architekturze szeregowo-równoległej. Proces ten zatrzymał się w momencie osiągnięcia minimalnej wartości gradientu ($1e^{-8}$) lub zaprzestania jego zmian w kolejnych iteracjach. Następnie sieci NARX zmieniały swą postać na architekturę równoległą. Trening kontynuowany był, aż kolejne iteracje algorytmu uczącego spowodowały osiągnięcie stałej (niezmniejszającej się) wartości błędu średniokwadratowego. Trening sieci przeprowadzono niezależnie dla obydwu osi maszyny.

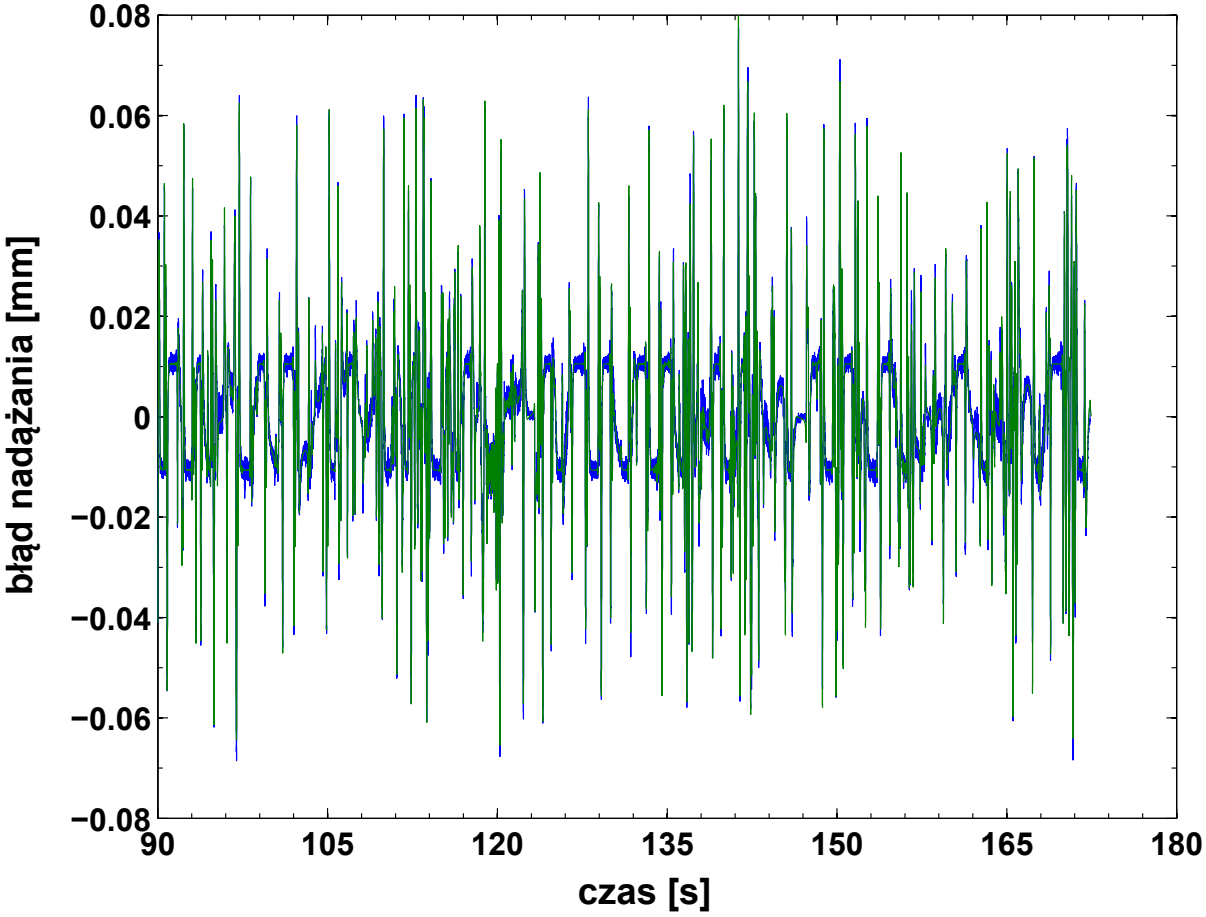
Na rysunkach 3.8 oraz 3.10 kolorem niebieskim zostały przedstawione wykresy błędów nadążania (odpowiednio dla osi X i Y), w wyniku realizacji fragmentu toru ruchu (realizacja od 0-ej do 90-tej sekundy). Fragment ten został wykorzystany do treningu neuronowych modeli układów posuwu osi mechanicznych. Natomiast kolorem zielonym zostały przedstawione wykresy predykcji błędu nadążania, otrzymane w wyniku działania neuronowych modeli

układów posuwu osi mechanicznych. W celu sprawdzenia sieci NARX ich działanie zostało przetestowane na zbiorze danych, który nie został wykorzystany w procesie uczenia sieci (fragment toru ruchu zrealizowanego pomiędzy 90-tą a 180-tą sekundą). Otrzymane wyniki walidacji zostały przedstawione na rysunkach 3.9 oraz 3.11. Różnicę pomiędzy wykresem błędu nadążania a ich predykcją otrzymaną z działania neuronowego modelu układu posuwu osi X dla zbioru danych uczących (rysunek 3.8) i walidacyjny (rysunek 3.9) przedstawiają rysunki: 3.12 i 3.13. Natomiast różnicę pomiędzy wykresem błędu nadążania a ich predykcją otrzymaną z działania neuronowego modelu układu posuwu osi Y dla zbioru danych uczących (rysunek 3.10) i walidacyjny (rysunek 3.11) przedstawiają rysunki: 3.14 i 3.15.

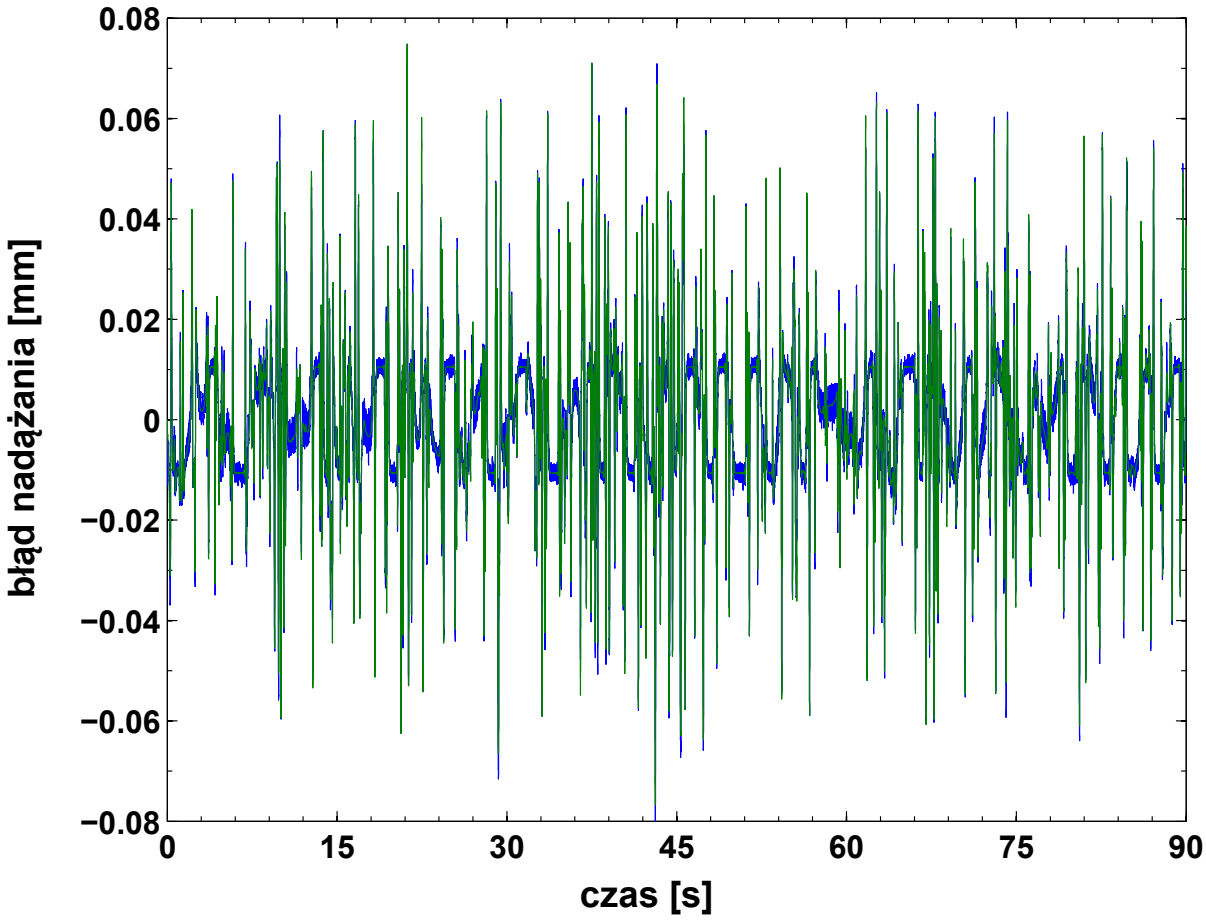
Otrzymane wyniki badań potwierdzają, że neuronowe sieci NARX bardzo dobrze modelują dynamikę układu posuwu osi mechanicznych z przekładnią śrubową toczną. Opracowane sieci NARX dokonują predykcji błędów nadążania układu poszczególnych osi mechanicznych. Błędy predykcji tych wartości nie przekraczały 0,01mm. Przedstawione w tym rozdziale sieci neuronowe NARX pozwalają przewidywać błędy nadążania z wystarczającą dokładnością, co umożliwia wykorzystanie ich w predykcyjnym algorytmie minimalizacji błędów nadążania układu posuwu osi mechanicznych.



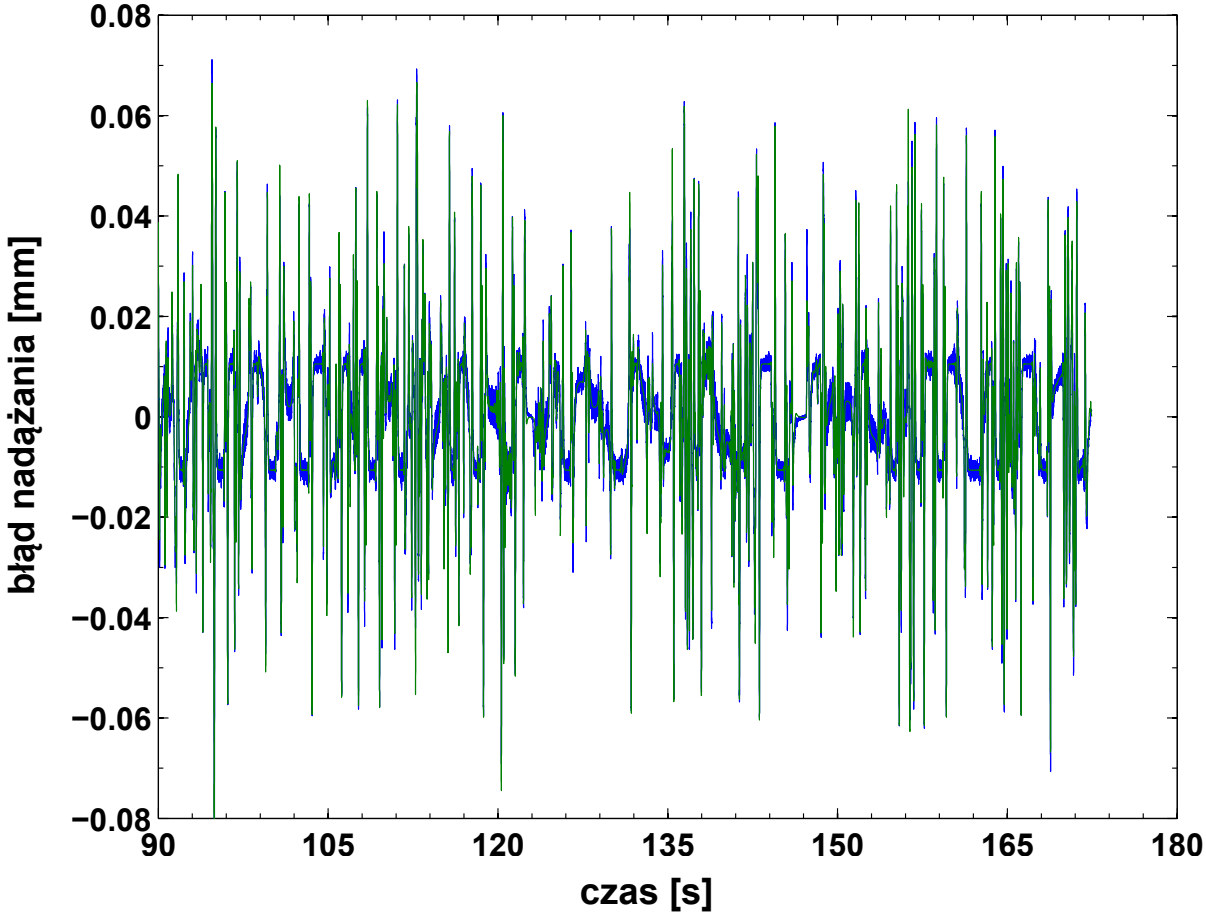
Rysunek 3.8. Błąd nadążania w osi X (niebieski wykres) i predykcja błędu nadążania przez sieci NARX (zielony wykres). Zbiór danych uczących (czas 0 - 90 s).



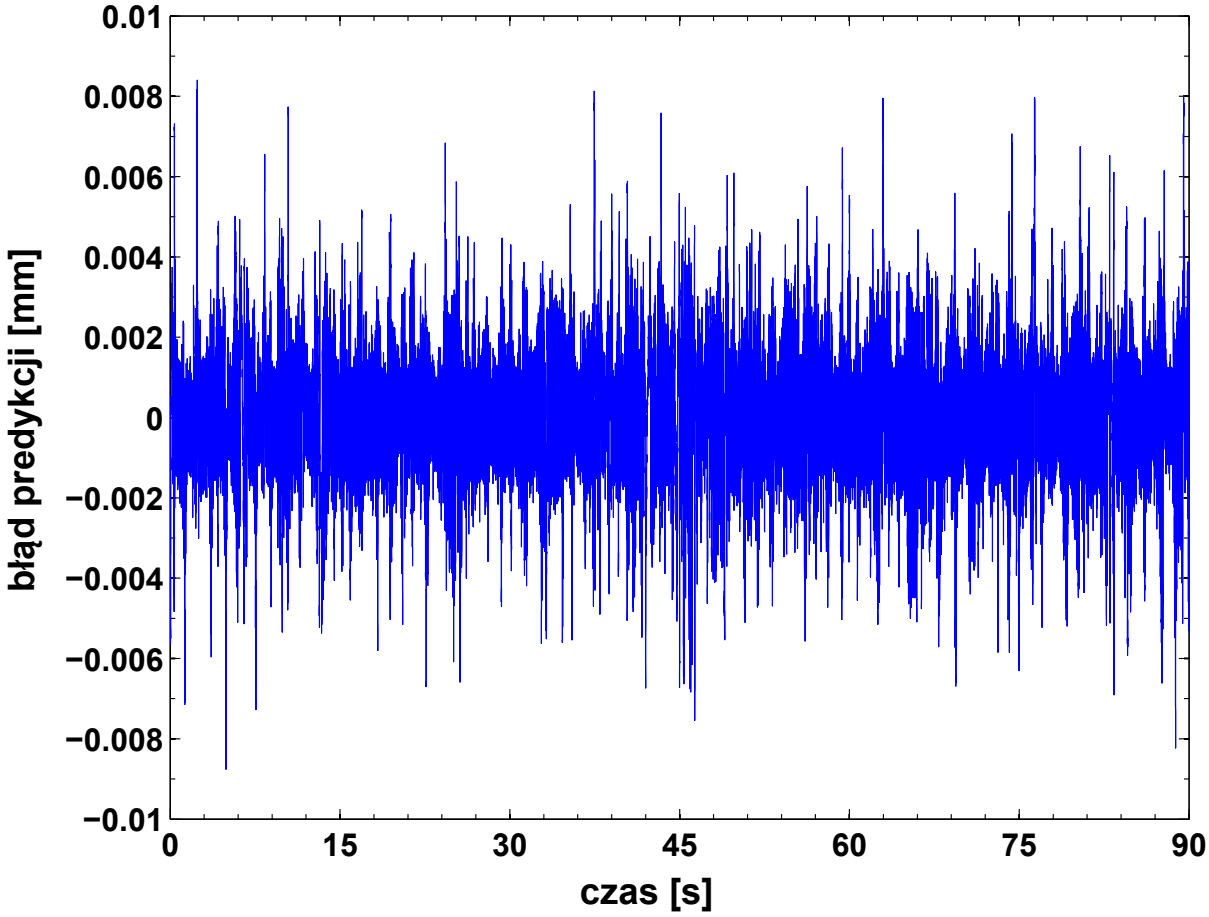
Rysunek 3.9. Błąd nadążania w osi X (niebieski wykres) i predykcja błędu nadążania przez sieci NARX (zielony wykres). Zbiór danych walidacyjny (czas 90 - 180 s).



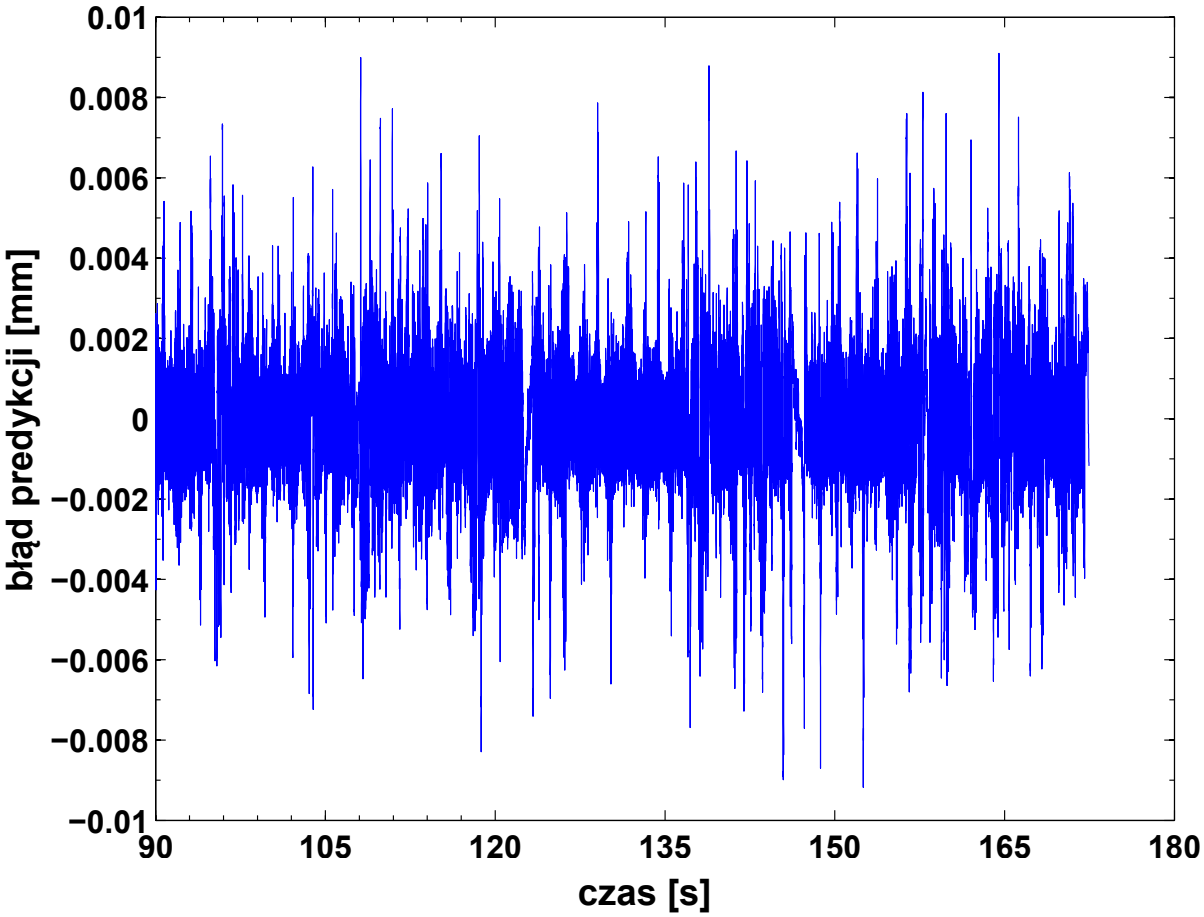
Rysunek 3.10. Błąd nadążania w osi Y (niebieski wykres) i predykcja błędu nadążania przez sieci NARX (zielony wykres). Zbiór danych uczących (czas 0 - 90 s).



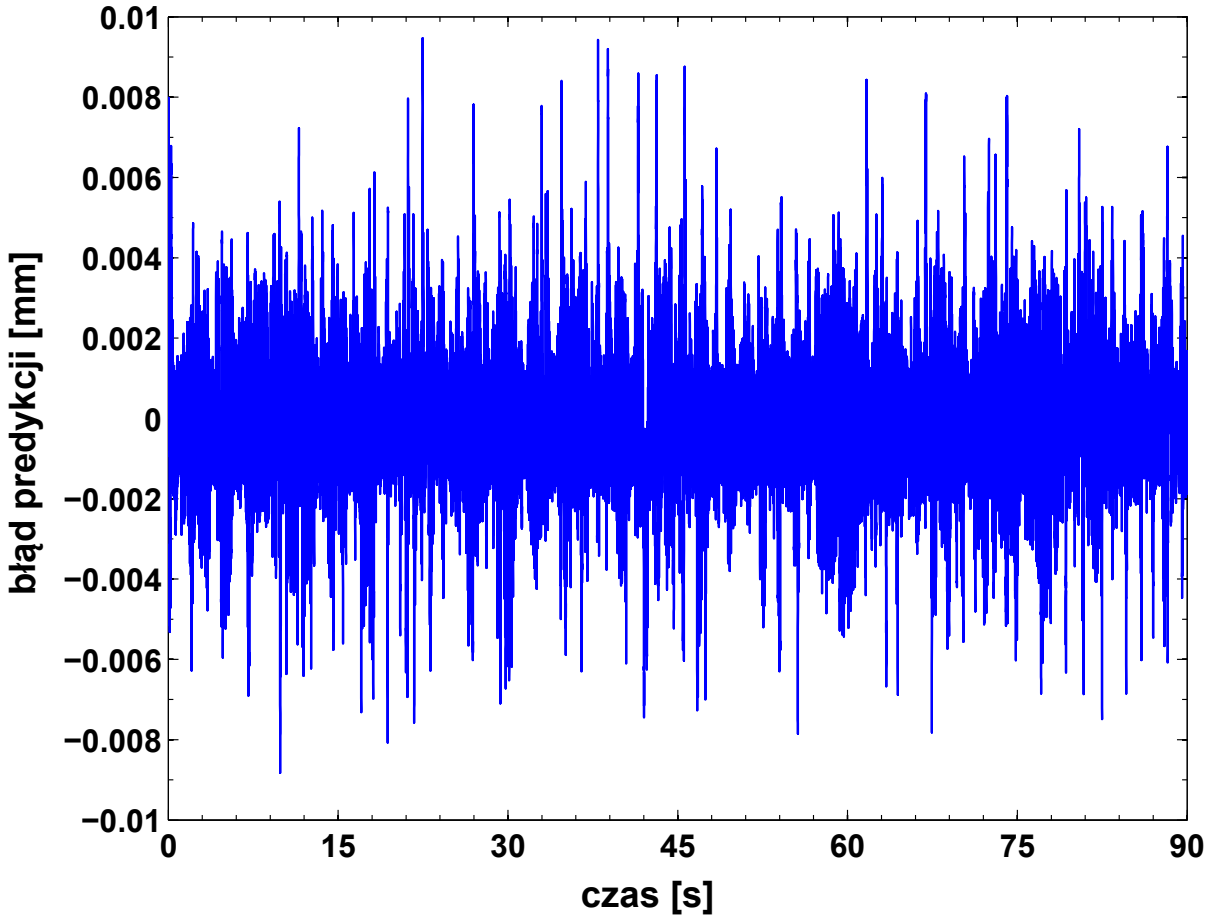
Rysunek 3.11. Błąd nadążania w osi Y (niebieski wykres) i predykcja błędu nadążania przez sieci NARX (zielony wykres). Zbiór danych walidacyjny (czas 90 - 180 s).



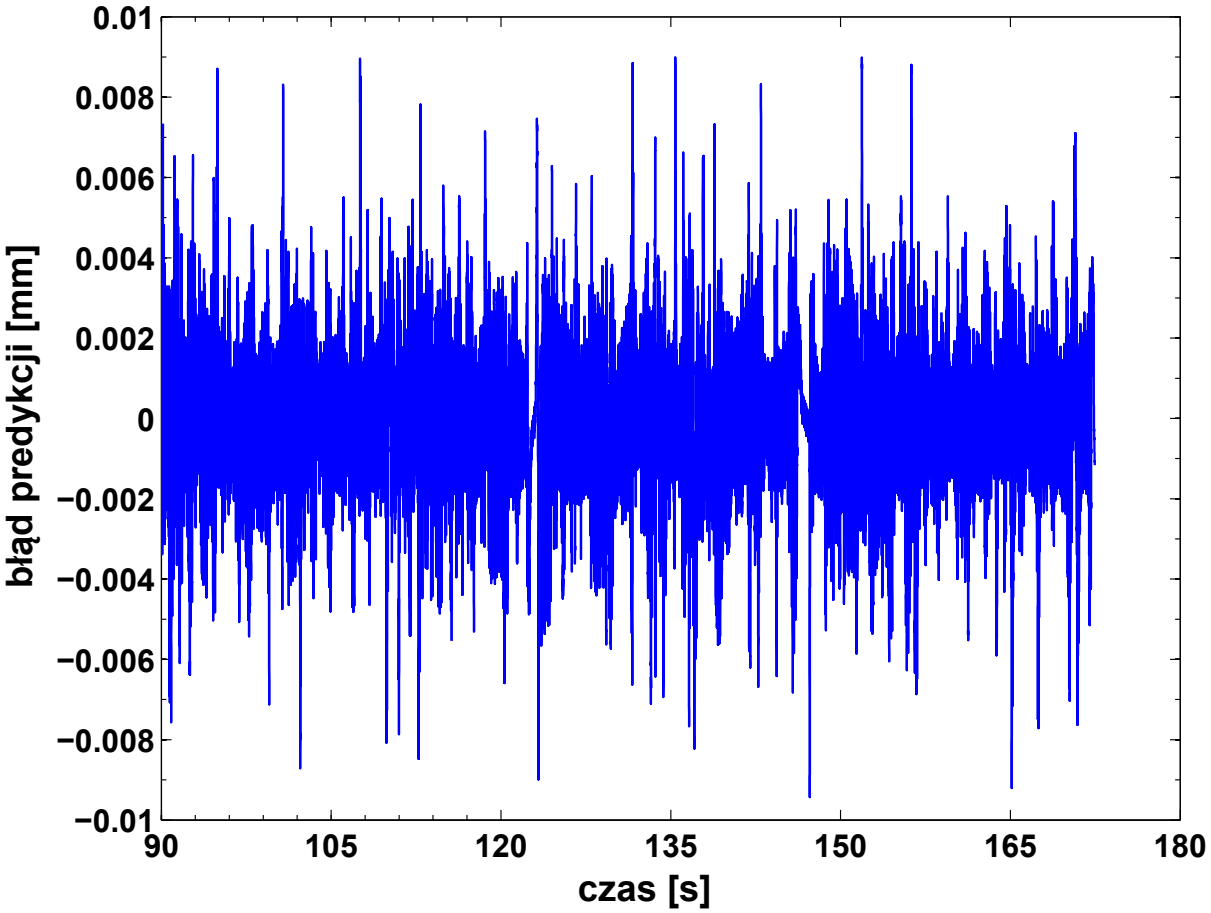
Rysunek 3.12. Błąd predykcji sieci neuronowej dla osi X (zbiór danych uczących - czas 0 - 90 s).



Rysunek 3.13. Błąd predykcji sieci neuronowej dla osi X (zbiór danych walidacyjnych - czas 90 - 180 s).



Rysunek 3.14. Błąd predykcji sieci neuronowej dla osi Y (zbiór danych uczących - czas 0 - 90 s).



Rysunek 3.15. Błąd predykcji sieci neuronowej dla osi Y (zbiór danych walidacyjnych - czas 90 - 180 s).

4. Predykcyjny algorytm minimalizacji błędów nadążania układu posuwu osi mechanicznej

Celem algorytmu minimalizacji błędów nadążania układu posuwu osi mechanicznej, jest zwiększenie dokładności odwzorowania zadanej trajektorii ruchu, przez układ posuwu osi mechanicznej. Zwiększenie dokładności odwzorowania trajektorii ruchu dokonuje się poprzez minimalizację błędów nadążania w każdej z osi mechanicznych układu posuwu maszyny wieloosiowej. W procesie minimalizacji błędów nadążania, każdej z osi mechanicznych, uwzględnić należy parametry fizyczne (ograniczenia) maszyny wieloosiowej. W szczególności dotyczy to maksymalnych wartości prędkości oraz dopuszczalnego błędu nadążania każdej z osi. Dla maszyny dwuosiowej powyższy problem można zdefiniować następująco:

$$\begin{aligned} min(\epsilon_x(i)) &= min\left(\sum_{i=1}^{L_x} |r_x(i) - s_x(i)|\right) \\ min(\epsilon_y(i)) &= min\left(\sum_{i=1}^{L_y} |r_y(i) - s_y(i)|\right) \end{aligned} \tag{4.1}$$

gdzie:
$\epsilon_x(i)$ - błąd nadążania osi X w i-tej chwili,
$\epsilon_y(i)$ - błąd nadążania osi Y w i-tej chwili,
$r_x(i)$ - zadany przyrost przemieszczenia dla osi X w i-tej chwili,
$r_y(i)$ - zadany przyrost przemieszczenia dla osi Y w i-tej chwili,
$s_x(i)$ - rzeczywisty przyrost przemieszczenia dla osi X w i-tej chwili,
$s_y(i)$ - rzeczywisty przyrost przemieszczenia dla osi Y w i-tej chwili,
L_x - ostatni zadany przyrost przemieszczenia dla osi X,
L_y - ostatni zadany przyrost przemieszczenia dla osi Y.

Powyższy opis podlega następującym ograniczeniom (dla maszyny dwuosiowej):

$$\begin{aligned} v_x(i) < v_{Xmax}, \quad v_y(i) < v_{Ymax} \\ \epsilon_x(i) < E_{Xmax}, \quad \epsilon_y(i) < E_{Ymax} \end{aligned} \tag{4.2}$$

gdzie:
$v_x(i), v_y(i)$ - chwilowe wartości prędkości w i-tej chwili,
v_{Xmax}, v_{Ymax} - maksymalne dopuszczalne wartości prędkości w osiach maszyny,
E_{Xmax} - maksymalna dopuszczalna wartość błędu nadążania dla osi X,
E_{Ymax} - maksymalna dopuszczalna wartość błędu nadążania dla osi Y.

Nieprzekraczanie wyszczególnionych powyżej ograniczeń, dla maksymalnych wartości błędów nadążania, wymaga dobrej znajomości dynamiki maszyny wieloosiowej. Konieczne jest posiadanie informacji o błędach w każdej chwili czasowej, podczas realizacji zadanych posuwów osi mechanicznych. Wykorzystując modele matematyczne układów posuwu osi mechanicznych, można przewidywać przyszłe błędy nadążania dla każdej z tych osi. Znajomość wartości błędów nadążania można wykorzystać do modyfikacji obliczonych przyrostów posuwu dla każdej z osi (obecnych i przyszłych), w celu zmniejszenia wartości błędów nadążania. Podejście to mieści się w klasie rozwiązań określanych jako sterowanie predykcyjne.

Predykcyjne algorytmy sterowania bazują na modelu obiektu sterowania. Model umożliwia wyznaczenie przyszłych odpowiedzi obiektu (predykcja) na podstawie sygnału sterującego. Wartość sygnału sterującego jest modyfikowana w celu minimalizacji lub maksymalizacji określonego kryterium (funkcji celu) przy uwzględnieniu zadanych ograniczeń. Zadanie to realizowane jest przez algorytmy optymalizacji z ograniczeniami. Jeśli użyty model będzie nieliniowy (np. sztuczna sieć neuronowa), to wymagane jest by użyty algorytm optymalizacji umożliwiał optymalizację nieliniową.

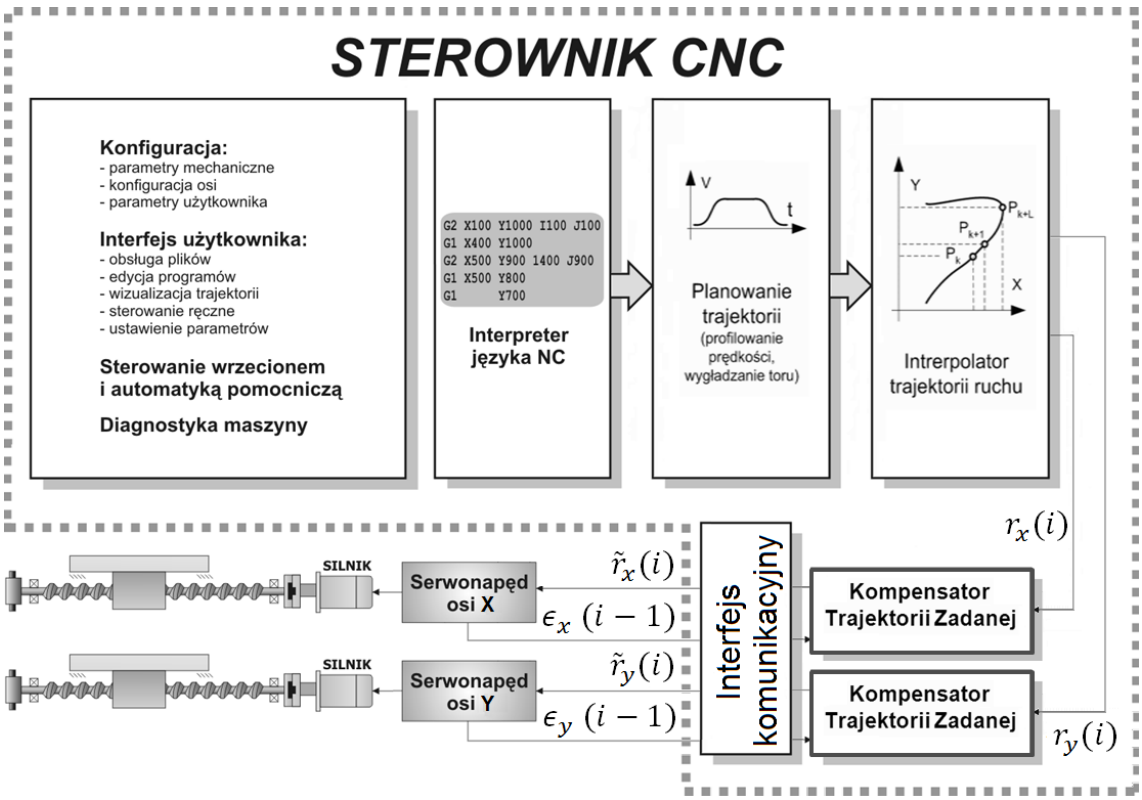
W proponowanym algorytmie optymalizacyjnym przyrostów posuwu, celem jest minimalizacja błędów nadążania (kolejnych przyszłych N błędów) dla każdej z osi mechanicznych. Powyższy problem optymalizacyjny sformułowano w postaci następującej funkcji celu dla każdej z osi mechanicznych:

$$\begin{aligned} F_c &= \sum_{i=1}^{N} (\hat{\epsilon}(i))^2 \\ \hat{\epsilon}(i) &= f_{NN}(\tilde{r}(i)) \end{aligned} \tag{4.3}$$

gdzie:
$\hat{\epsilon}(i)$ - predykowany błąd nadążania i-tej próbki w danym horyzoncie predykcji, zależny od zmodyfikowanego przyrostu położenia zadanego $\tilde{r}(i)$,
f_{NN} - wartość wyjściowa sieci neuronowej,
N - ilość próbek optymalizowanych w danym horyzoncie predykcji.

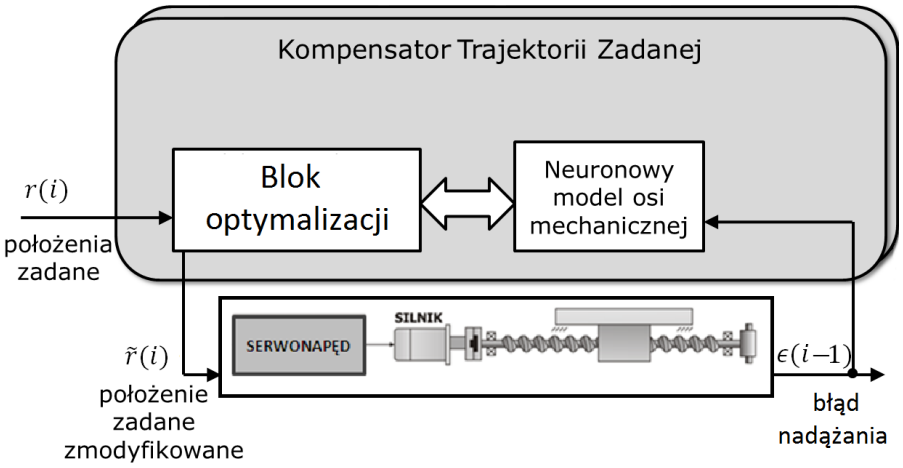
4.1. Kompensator Trajektorii Zadanej

Proponowany układ sterowania CNC (rys. 4.1) steruje zespołem dwóch osi mechanicznych, z których każda zbudowana jest z suportu mechanicznego oraz serwonapędu. W sterowniku CNC zaimplementowano standardowe funkcje związane z konfiguracją maszyny CNC, takie jak: diagnostyka maszyny, obsługa interfejsu użytkownika, sterowanie wrzecionem i automatyką pomocniczą. Ponadto zaimplementowane zostały klasyczne algorytmy generacji trajektorii ruchu z trapezoidalnym profilowaniem prędkości oraz interpolacji liniowej i kołowej, występujące w większości sterowników CNC. W proponowanym rozwiązaniu pomiędzy blok Interpolatora trajektorii ruchu a moduł interfejsu komunikacji z serwonapędami wbudowano dwa bloki Kompensatorów Trajektorii Zadanej (rys. 4.2), które realizują algorytm minimalizacji błędów nadążania układu posuwu każdej z osi mechanicznych. Bloki Kompensatorów mają za zadanie wyznaczać nowe zmodyfikowane wartości przyrostów położeń zadanych (odpowiednio $\tilde{r}_x(i)$ oraz $\tilde{r}_y(i)$) w stałych odstępach czasu.



Rysunek 4.1. Układ sterownika CNC z zaimplementowanym układem sterowania predykcyjnego (Kompensatora Trajektorii Zadanej)

W każdym bloku Kompensatora Trajektorii Zadanej wyszczególnić można neuronowy model osi mechanicznej oraz blok optymalizacji.

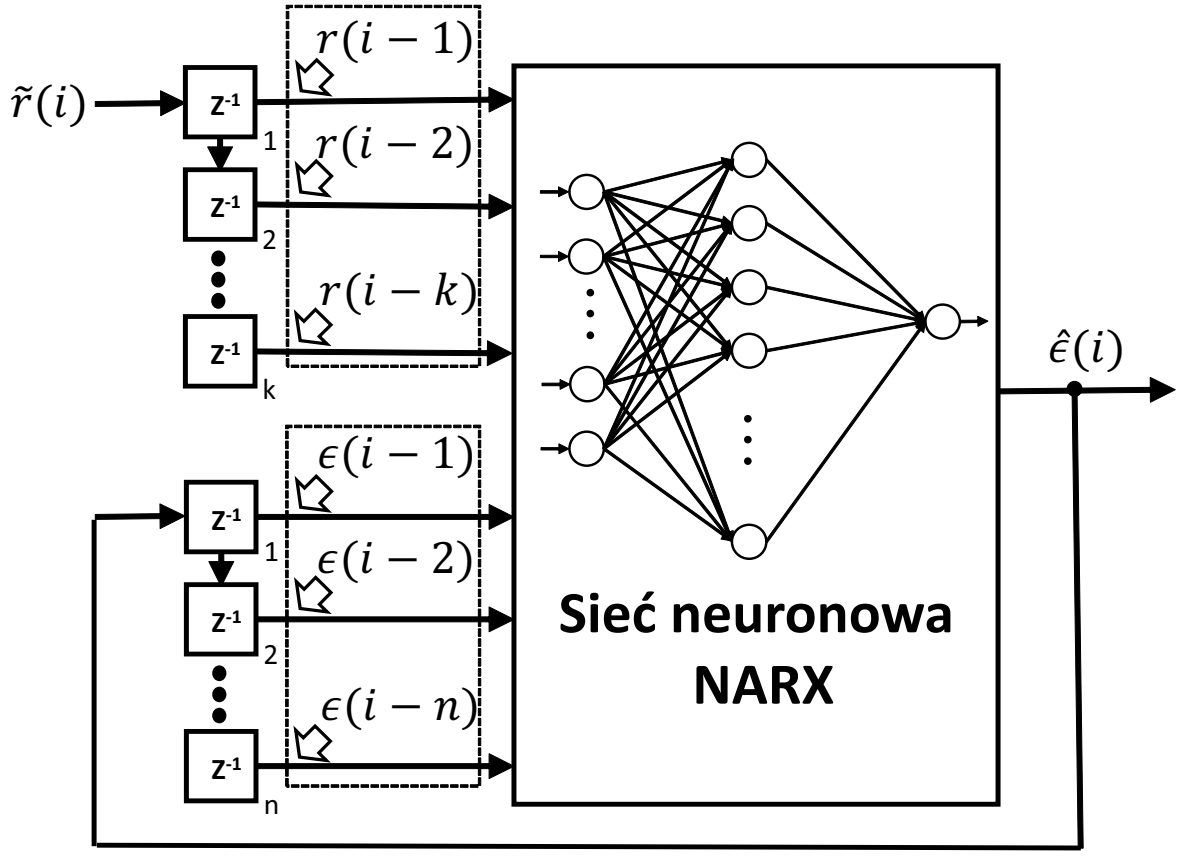


Rysunek 4.2. Kompensator Trajektorii Zadanej dla jednej z osi mechanicznych

Zadaniem neuronowego modelu osi mechanicznej, w postaci sztucznej sieci neuronowej typu NARX (szczegółowo opisanego w rozdziale 3), jest predykcja błędu nadążania. W każdym bloku Kompensatora Trajektorii Zadanej występuje sieć neuronowa, modelująca dynamikę danej osi. Schemat sieci neuronowej typu NARX, będącej modelem osi mechanicznej, przedstawia rysunek 4.3. Sieci neuronowe NARX posiadają n wejść opóźniających, związanych z wartościami błędów nadążania oraz k wejść związanych z wartościami przyrostów trajektorii zadanej. Sieci neuronowe umożliwiają predykcję błędów nadążania od $\hat{\epsilon}(i)$ do $\hat{\epsilon}(i+N)$ wartości w przód. W celu zmniejszenia błędu predykcji, przed każdym wyliczaniem wartości $\hat{\epsilon}(i)$, ustalane są warunki początkowe sieci NARX w następujący sposób:

— wejścia opóźniające związane z wartościami przyrostów trajektorii zadanej inicjalizowane są wektorem $[r(i-1), ..., r(i-k)]$.
— wejścia opóźniające związane z wartościami błędów nadążania inicjalizowane są wektorem $[\epsilon(i-1), ..., \epsilon(i-n)]$,

Schemat przepływowy zaproponowanego algorytmu przedstawiono na rysunku 4.4. W Kompensatorze, w stałych odstępach czasu, wyznaczana jest wartość optymalnego zmodyfikowanego położenia zadanego $r(i)$. W tym celu z bloku Interpolatora otrzymywany jest wektor położeń zadanych $[r(i), ..., r(i+N)]$ (horyzont predykcji o długości $N+1$). Przekazywany jest on do bloku optymalizacji, który wykorzystuje neuronowy model osi mechanicznej. W bloku optymalizacji realizowany jest algorytm optymalizacji, który wykonywany jest cyklicznie. W każdym cyklu, wartości wektora $[r(i), ..., r(i+N)]$ są modyfikowane i uzyskiwany jest nowy wektor $[\tilde{r}(i), ..., \tilde{r}(i+N)]$. Ilość cykli algorytmu optymalizacji jest z góry określona. Finalnie wybierany jest ten wektor $[\tilde{r}(i), ..., \tilde{r}(i+N)]$, który



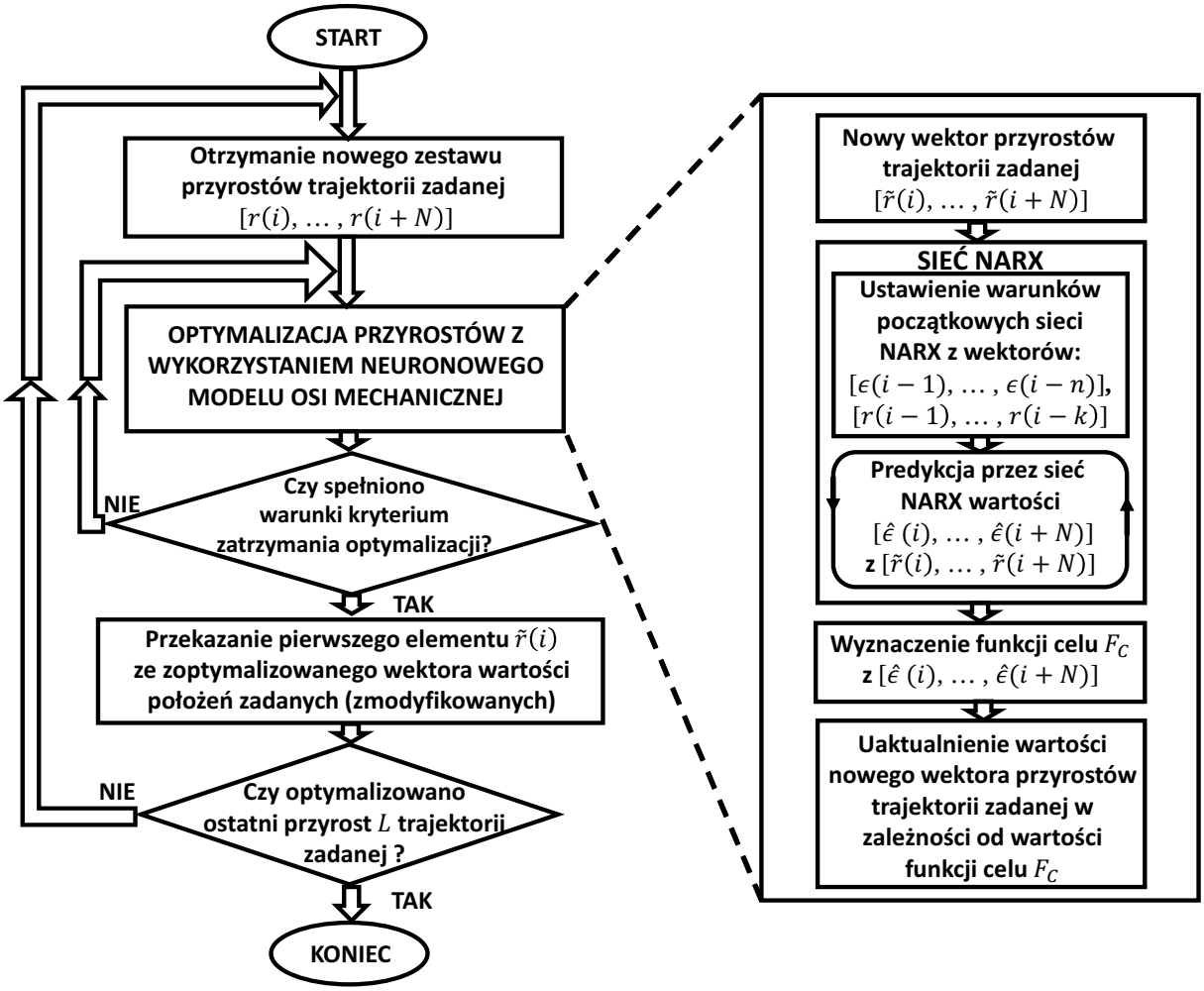
Rysunek 4.3. Schemat sieci neuronowej typu NARX będącej modelem osi mechanicznej.

spośród wszystkich wygenerowanych wektorów, powoduje uzyskanie najmniejszej wartości funkcji celu F_c (równanie 4.3). Wartość $\tilde{r}(i)$ z tego wektora staje się następną wartością położenia zadanego zmodyfikowanego. W tym momencie kończy się jedna iteracja działania Kompensatora Trajektorii Zadanej i rozpoczyna następna.

4.2. Przegląd metod optymalizacji nieliniowej

W celu wyboru odpowiedniego algorytmu optymalizacji, działającego w bloku optymalizacji, została przeprowadzona analiza algorytmów optymalizacyjnych opisanych w literaturze.

Problemy optymalizacyjne z ograniczeniami często rozwiązuje się wykorzystując algorytmy gradientowe z ograniczeniami np. algorytm sekwencyjnego programowania kwadratowego (ang. Sequential Quadratic Programming - SQP)[92]. Zasada działania algorytmu, polega na rozwiązywaniu szeregu podproblemów, z których każdy jest wynikiem lokalnej aproksymacji funkcji celu, funkcją kwadratową oraz linearyzacji ograniczeń. Algorytm SQP jest wykonywany sekwencyjnie, gdyż każde rozwiązanie danego podproblemu stanowi punkt startowy dla kolejnego podproblemu.



Rysunek 4.4. Schemat przepływowy algorytmu optymalizacji przyrostów trajektorii zadanej.

Innym możliwym rozwiązaniem problemu optymalizacji jest wykorzystanie algorytmu programowania liniowego (ang. Linear Programming - LP)[39][43]. LP nie wymaga wyliczania wartości gradientu oraz ograniczeń, co przekłada się na dużą efektywność obliczeniową algorytmu. Wymaganiem algorytmu jest by funkcja celu oraz ograniczenia były liniowe, a w ostateczności były zlinearyzowane w przypadku problemu nieliniowego.

Alternatywnym podejściem do wyżej wymienionych algorytmów może być zastosowanie algorytmów optymalizacji globalnej. W algorytmach tych optimum globalne znajdywane jest w wyniku ekstensywnego przeszukiwania przestrzeni rozwiązań. Algorytmy tego typu są niepodatne na zatrzymanie się w optimum lokalnym, a w szczególności nie są zależne od początkowych wartości argumentów funkcji celu. W większość przypadków wartości te są generowane przez sam algorytm. Dodatkowo algorytmy te uwzględniają narzucone ograniczenia, nawet dla początkowych wartości argumentów funkcji celu.

Jednym z przykładów algorytmów optymalizacji globalnej jest algorytm optymalizacji rojem cząstek (ang. Particle Swarm Optimization - PSO). Algorytm został opracowany przez Kennedy'ego i Eberhart'a [57] [32][58] jako symulator ruchów stad ptaków i ławic ryb

poszukujących pożywienia. Okazało się, że opracowany algorytm można wykorzystać także jako skuteczne narzędzie do minimalizacji funkcji wielu zmiennych, których zbiór rozwiązań posiada minima lokalne. Algorytm PSO jest prosty w implementacji i nieskomplikowany obliczeniowo, dlatego został zaimplementowany przez autora i wykorzystany do rozwiązania problemu optymalizacji przyrostów położeń zadanych.

Zasada działania algorytmu optymalizacji rojem cząstek, bazuje na przeszukiwaniu zbioru rozwiązań zadanej funkcji, w celu znalezienia jej optimum globalnego. Spektrum jest przeszukiwane przez wirtualne cząstki, które zajmują konkretne położenie w tym zbiorze rozwiązań (określone przez konkretny zestaw argumentów zmiennych funkcji celu). Każda z cząstek określa konkretne rozwiązanie funkcji celu. Informacje na temat wartości funkcji celu każdej z cząsek są wymieniane pomiędzy nimi w celu określenia kierunku przeszukiwania zbioru rozwiązań każdej cząstki. Kierunek ten określany jest przez wektor prędkości. Jego początkowa wartość, dla każdej z cząstek, otrzymywana jest losowo przy użyciu rozkładu równomiernego, w całej przestrzeni możliwych argumentów funkcji celu. Opis zmiany położenia oraz wektora prędkości każdej z cząstki można przedstawić w postaci równań:

$$v^*_{i,j} = v_{i,j} + \phi_1 \cdot rand^U_{(0,1)} \cdot (p_{i,j} - x_{i,j}) + \phi_2 \cdot rand^U_{(0,1)} \cdot (g_i - x_{i,j}) \tag{4.4}$$

$$x^*_{i,j} = x_{i,j} + v^*_{i,j} \tag{4.5}$$

gdzie:
$v^*_{i,j}$ - nowa i-te położenie wektora prędkości j-tej cząstki
$v_{i,j}$ - aktualna i-te położenie wektora prędkości j-tej cząstki
$x^*_{i,j}$ - nowa i-te położenie j-tej cząstki
$x_{i,j}$ - aktualna i-te położenie j-tej cząstki
$rand^U_{(0,1)}$ - liczba pseudolosowa o rozkładzie równomiernym z przedziału [0,1]
ϕ_1,ϕ_2 - współczynniki przyspieszenia
$p_{i,j}$ - i-te najlepsze osobiste położenie j-tej cząstki
g_i - i-te najlepsze globalne położenie cząstek

Powyższe równania są oryginalną postacią tego algorytmu, przedstawioną przez Kennedy'ego i Eberhart'a. Podczas działania algorytmu, każda cząstka przechowuje informację o swoim najlepszym dotychczasowym położeniu (ang. personal best position - pbest). Pbest określa najlepszy zestaw argumentów dla których funkcja celu względem dotychczasowych położeń cząstki ma najniższą wartość. Jeśli w kolejnej iteracji tego algorytmu, dla tej samej cząstki, aktualne położenie stanie się lepsze (nowa wartość funkcji celu będzie niższa od

poprzedniej), to w kolejnej iteracji wartość ta staje się nową wartością pbest. Dodatkowo w każdej iteracji porównywane są wszystkie wartości pbest wszystkich cząstek, w celu znalezienia tej odpowiadającej najniższej wartości funkcji celu. Następnie jest ona porównywana z wartością najlepszego globalnego położenia cząstek (ang. global best position - gbest). Jeśli wartość ta jest lepsza niż gbest, to w kolejnej iteracji zastępuje ona wartość dotychczasowego gbest. W przeciwnym wypadku, wartość gbest zostaje bez zmian.

Z powyższych równań wynika, że decydujący wpływ na wartość wektora prędkości każdej z cząstek, mają wartości najlepszego osobistego i globalnego położenia oraz współczynniki losowe. Skutkuje to w każdej iteracji niezapewnieniem spadku wartości funkcji celu dla każdej z cząstek. Jest to sytuacja inna niż ma to miejsce w algorytmach gradientowych, gdzie co iteracje wymagany jest spadek wartości funkcji celu. Takie działanie algorytmu PSO powoduje, że jest on bardziej odporny na przedwczesne zatrzymanie w minimum lokalnym.

W pierwszej fazie algorytmu PSO cząstki rozmieszczane są losowo w zbiorze rozwiązań. Początkowo, ich położenia zmieniają się losowo w celu eksploracji przestrzeni rozwiązań. Jednak w każdej kolejnej iteracji modyfikowany wektor prędkość (w sposób stochastyczny) powoduje zbliżanie roju cząstek w okolice najlepszego położenia gbest. Przestrzeń poszukiwań roju cząstek powoli zawęża się w celu jego eksploatacji. Skutkuje to bardziej dokładnym przeszukaniem przestrzeni z znajdującym się najlepszym położeniem (minimum funkcji celu). Rój cząstek po skończonej liczbie iteracji skupia się w bardzo wąskim obszarze zbioru rozwiązań. Otrzymana wartość gbest przyjmowana jest jako rozwiązanie końcowe.

Oprócz wartości pbest oraz gbest, na wektor prędkości cząstki wpływ mają parametry algorytmu, takie jak: ilość cząstek oraz współczynniki przyspieszenia ϕ_1,ϕ_2. Parametry rzutują na szybkość zbiegania się w okolicach najlepszego rozwiązania oraz na dokładność przeszukiwania zbioru rozwiązań roju cząstek. Decydujący wpływ na długość trwania fazy eksploracji i eksploatacji mają współczynniki przyspieszenia. Przyjęte jest [84], że parametry przyspieszenia przyjmują wartości $\phi_1 = \phi_2 = 2.0$, zaś liczba cząstek roju waha się pomiędzy 20 a 50. Nieodpowiedni dobór parametrów algorytmu skutkuje nadmiernym wzrostem wektora prędkości każdej z cząstek, co finalnie spowodować może brak zbieżności w okolicach najlepszego położenia całego roju cząstek. Rozwiązaniem tego problemu jest wprowadzenie ograniczenia długości wektora prędkości, które można zrealizować na różne sposoby, w zależności od problemu optymalizacyjnego. Shi i Eberhart [94] zaproponowali uwzględnienie w równaniu 4.4 współczynnika bezwładności ω, który minimalizował wpływ wzrostu nadmiernej wartości wektora prędkości. Zmodyfikowane równanie ma postać:

$$v^*_{i,j} = \omega \cdot v_{i,j} + \phi_1 \cdot rand^U_{(0,1)} \cdot (p_{i,j} - x_{i,j}) + \phi_2 \cdot rand^U_{(0,1)} \cdot (g_i - x_{i,j}) \tag{4.6}$$

Obok bezpośredniego ograniczenia wartości wektora prędkości, współczynnik bezwładności wpływa dodatkowo na ograniczanie jego wartości. W konsekwencji odgórne ograniczenie prędkości można pominąć ze względu na fakt, że wprowadzenie inercji zmniejsza wpływ ograniczenia wektora prędkości na dynamikę roju cząstek. Wartość współczynnika bezwładności najczęściej wybierana jest losowo z przedziału [0,1]. Wysokie lub niskie wartości współczynnika mogą mieć różne skutki na rój cząstek. Wysokie wartości (rzędu 0.9) powodować mogą wolniejszą zbieżność roju do najlepszego rozwiązania, przyczyniając się do zwiększonej eksploracji. Natomiast niska wartość współczynnika (rzędu 0.4) wpływa na szybszą zbieżność (eksploatację) wokół najlepszego rozwiązania, kosztem przeszukiwania większego obszaru zbioru rozwiązań. Dlatego wielu autorów postanowiło zaimplementować współczynnik bezwładności, jako wartość funkcji zależnej od kolejnych iteracji algorytmu PSO (funkcji przeważnie malejącej). Przykładami może być tu współczynnik ω jako funkcja nieliniowej zależności od liczby iteracji [16], modyfikowany na bazie logiki rozmytej [95] oraz stochastyczna modyfikacja współczynnika ω [128]. Wymienione przykłady modyfikacji współczynnika bezwładności, poprawiają zdolności roju cząstek do bardziej efektywnego znajdywania najlepszego rozwiązania w porównaniu z rozwiązaniem klasycznym (równanie 4.6). Problemem jest jednak sposób wyrażenia funkcji wraz z doborem jej parametrów, która określa wartość współczynnika ω. Wyznaczenie funkcji z jej parametrami, wymaga przeanalizowania zadanego problemu optymalizacyjnego, a sam algorytm zwiększa swoją złożoność obliczeniową. W większości powyższych przykładów, konieczny jest eksperymentalny dobór wielu współczynników algorytmu PSO, w celu znalezienia odpowiedniej równowagi między fazą eksploracji i eksploatacji roju cząstek. Błędne określenie współczynników może doprowadzić do dezintegracji roju cząstek, w szczególności gdy parametrem modyfikowanym jest współczynnik bezwładności.

Alternatywnym podejściem wykazał się Clerc'a [22], który zmodyfikował klasyczne równanie wektora prędkości. Modyfikacja polegała na wprowadzeniu współczynnika ścisku (ang. constriction coefficient), co wpłynęło na postać równania 4.4 w następujący sposób:

$$v^*_{i,j} = \chi \left(v_{i,j} + \phi_{c1} \cdot rand^U_{(0,1)} \cdot (p_{i,j} - x_{i,j}) + \phi_{c2} \cdot rand^U_{(0,1)} \cdot (g_i - x_{i,j}) \right) \tag{4.7}$$

$$\chi = \frac{2}{\phi - 2 + \sqrt{\phi^2 - 4\phi}} \tag{4.8}$$

$$\phi = \phi_1 + \phi_2 \tag{4.9}$$

gdzie:

χ - współczynnik ścisku

ϕ_1, ϕ_2 - współczynniki przyspieszenia

Dla powyższych równań współczynniki ϕ_{c1} i ϕ_{c2} są równe wartości 2.05 [22]. Współczynnik ścisku w takim wypadku równy jest ≈ 0.7298. W porównaniu z równaniem 4.6 by uzyskać taką samą postać jak równanie 4.7, trzeba przyjąć wartość współczynnika bezwładności równą $\omega = \chi \approx 0.7298$ oraz współczynników przyspieszenia $\phi_1 = \phi_2 \approx 1.4962$. Takie wartości współczynników powodują, że rój cząstek nigdy nie ulegnie dezintegracji (udowodnione przez Clerc). W konsekwencji narzucanie odgórnie ograniczenia wartości wektora prędkości nie jest potrzebne. W szczególności późniejsza weryfikacja algorytmu [33] doprowadziła do wyników, w których dzięki brakowi odgórnych ograniczeń można uzyskać bardzo dobre właściwości dynamiczne roju cząstek. Zaproponowano wprowadzenie ograniczenia co do wartości wektora prędkości równe maksymalnej wartości danej zmiennej ($V_{max,i} = X_{max,i}$). Obecnie zaproponowany przez Clerc'a algorytm PSO ze współczynnikami ścisku oraz parametrami ograniczającymi prędkość równymi ograniczeniom wartości zmiennej jest wariantem najczęściej wykorzystywanym.

Zjawiskiem często występującym w algorytmach PSO jest tzw. stagnacja roju. Zjawisko to ma miejsce gdy najlepsza cząstka wskazuje gbest, które jest optimum lokalnym, zaś pozostałe cząstki roju zbliżają się powoli do tej wartości. Dzieje się to ze względu na to, że wartość wektora prędkości dla cząstki przebywającej w gbest zależy od jej poprzedniej wartości, a także poprzednich wartości współczynnika bezwładności lub ścisku. Ze względu na wartości współczynnika bezwładności lub ścisku, które są mniejsze od 1, wartość wektora prędkości będzie dążyć do zera, zbliżając do siebie pozostałe cząstki roju wokół tej wartości. Zjawisko to szerzej zostało opisane w [110].

Częściowym rozwiązaniem powyższego problemu jest zaproponowany przez van den Bergh'a algorytm PSO z gwarantowaną zbieżnością (ang. Guaranteed Convergence Particle Swarm Optimization - GCPSO) [111]. W proponowanym rozwiązaniu położono nacisk na faktyczne znalezienie minimum funkcji celu (lokalne lub globalne), a nie tylko na ostatnią najlepszą pozycję gbest roju cząstek, w przypadku wystąpienia stagnacji roju. Van den Bergh zaproponował nowe równanie wyznaczania wektora prędkości dla cząstki, która znalazła ostatnią najlepszą pozycję gbest, w następującej postaci:

$$v^*_{i,\tau} = \chi \cdot v_{i,\tau} - x_{i,\tau} + g_i + \rho \cdot rand^U_{(-1,1)} \tag{4.10}$$

gdzie:

τ - indeks cząstki, która odnalazła najlepszą globalną pozycję we wszystkich iteracjach PSO,

ρ - promień okręgu wyznaczający obszar przeznaczony do przeszukiwania losowego o środku w gbest,

$rand^U_{(-1,1)}$ - liczba pseudolosowa o rozkładzie równomiernym z przedziału [-1,1].

W celu zapobiegnięcia skupienia się oraz docelowym zatrzymaniu się roju cząstek w optimum lokalnym, wprowadzono zmodyfikowane równanie wektora prędkości. Wokół najlepszej pozycji gbest uruchamiane jest dodatkowe przeszukiwanie losowe, które ogranicza się do obszaru wyznaczonego przez sferę o promieniu współczynnika skalującego ρ. W przypadku, gdy podczas przeszukiwania losowego, znajdzie się w zbiorze rozwiązań położenie lepsze od dotychczasowego, to inkrementowany jest współczynnik sukcesów N_s. Jeżeli w kolejnych iteracjach nie uda się znaleźć lepszego położenia od dotychczasowego, to inkrementowany jest współczynnik porażek N_f. Jeśli sumy współczynników sukcesów N_s lub sumy współczynników porażek N_f przekroczą - odpowiednio - granicę $N_{s,MAX}$ i $N_{f,MAX}$, to współczynnik ρ wyznaczany jest z następującej zależności:

$$\rho^* = \begin{cases} 2\rho & dla\ N_s > N_{s,MAX} \\ 0.5\rho & dla\ N_f > N_{f,MAX} \\ \rho & dla\ pozostaych \end{cases} \tag{4.11}$$

Wśród wielu wariantów algorytmu PSO występują takie, w których modyfikowana jest topologia wymiany informacji pomiędzy cząstkami. W standardowych topologiach PSO (globalnych) informacja na temat położenia gbest jest wymieniana z całym rojem cząstek. W topologiach lokalnych tworzone są grupy cząstek, w których informacja na temat najlepszego położenia jest wymieniana tylko pomiędzy nimi. Modyfikacja ta powoduje, że faza eksploracji przez wszystkie cząstki trwa dłużej, a zatem zbiór możliwych rozwiązań przeszukiwany jest dokładniej. Wpływa to na wydłużenie czasu, w którym cząstki znajdują najlepszą pozycję, co powinno skutkować większym prawdopodobieństwem znalezienia globalnego optimum. Z analizy literatury [82] wynika jednak, że wyniki działania topologi lokalnej nie są znacząco lepsze od działania algorytmów PSO w topologii globalnej.

Innym ważnym aspektem związanym z algorytmem PSO jest określenie momentu zatrzymania optymalizacji. Najczęściej spotykanym rozwiązaniem jest zatrzymanie optymalizacji po wykonaniu, z góry określonej, liczby iteracji algorytmu lub ewaluacji funkcji celu. Najczęściej liczba ta dobierana jest eksperymentalnie, co wymaga bardzo dobrej znajomości zadanego problemu optymalizacyjnego. Wybór zbyt małej liczby iteracji algorytmu może spowodować, że rój cząstek nie będzie w stanie osiągnąć okolic minimum globalnego funkcji celu. Ustawienie zbyt dużej ilości iteracji w wielu przypadkach powoduje niepotrzebne wydłużenie działania algorytmu PSO. Wybór innego kryterium zatrzymania optymalizacji (znanych ze standardowych algorytmów), takich jak spadek wartości gradientu (algorytm gradientowy), spadek wartości funkcji celu pomiędzy kolejnymi iteracjami w przypadku PSO jest niemożliwy. W algorytmach PSO pomiędzy kolejnymi jego iteracjami wartości funkcji celu są niekoniecznie niższe niż w poprzednich iteracjach, gdyż zbiór rozwiązań tej funkcji przeszukiwany jest w wielu losowych miejscach. Alternatywnym do wymienionych wcześniej

podejściem, co do zatrzymania optymalizacji, jest kryterium zbieżności badające zbieżność cząstek roju. Kryterium to można wyrazić za pomocą następujących wzorów:

$$\bar{d}_{gbest} = \frac{1}{N} \sum_{i=0}^{N-1} \sum_{j=0}^{D-1} (x_{i,j} - g_j)^2 \tag{4.12}$$

gdzie:
N - ilość cząstek roju
D - wymiar zbioru rozwiązań

Po jednoczesnym spełnieniu dwóch poniższych nierówności, proces optymalizacji jest przerywany.

$$\left|\bar{d}_{gbest}(k) - \bar{d}_{gbest}(k - k_{stop})\right| \leqslant DTOL \cdot \bar{d}_{gbest}(k - k_{stop}) \tag{4.13}$$

$$F_c(\bar{d}_{gbest}(k - k_{stop})) - F_c(\bar{d}_{gbest}(k)) \leqslant FTOL \cdot F(\bar{d}_{gbest}(k - k_{stop})) \tag{4.14}$$

gdzie:
k_{stop} - liczba iteracji pomiędzy którymi badane są postępy roju cząstek,
$DTOL, FTOL$ - współczynniki determinujące względne wartości progowe dla których nierówności są spełnione.

Z nierówności 4.13 wynika, że jest ona spełniona, gdy przez określoną ilość iteracji, średnia odległości pomiędzy cząstkami a położeniem gbest nie ulega znaczącej zmianie. Natomiast z nierówności 4.14 wynika, że jest ona spełniona, gdy przez określoną ilość iteracji wartość gbest zmniejsza się nieznacznie. Spełnienie obu nierówności powoduje zakończenie procesu optymalizacji, gdyż rój cząstek w skończonej ilości iteracji nie dokonał znaczącego postępu w zawężaniu zbioru rozwiązań. Jednocześnie, rój cząstek nie doprowadził do znacznego spadku wartości gbest. W obydwu nierównościach kryterium zatrzymania optymalizacji, wyznaczane jest w sposób względny oraz jest niezależne od początkowych położeń cząstek roju.

Wymienione dotychczas odmiany algorytmów PSO, nie uwzględniają ograniczeń innych niż zakres zmienności zmiennych optymalizacyjnych. Konieczność uwzględnienia bardziej złożonych ograniczeń, skłoniła niektórych badaczy do opracowania wariatów PSO, w których uwzględniono ograniczenia bezpośrednio w samym algorytmie, przykładami mogą być tu rozwiązania opisane w [23] [64]. Alternatywną metodą uwzględniania ograniczeń w algorytmie PSO jest metoda polegająca na zastosowaniu funkcji kary. W metodzie tej,

minimalizacji podlega suma funkcji celu oraz funkcji kary. Często spotyka się funkcję kary w postaci sumy kwadratów poszczególnych ograniczeń, które przyjmują wartości niezerowe w wypadku ich przekroczenia lub wartości równe zero, gdy nie są one przekroczone. W takim wypadku algorytm PSO dąży zarówno do minimalizacji funkcji celu jak i funkcji kary (dąży do minimalizacji wartości, które powodują przekroczenie zadanych ograniczeń). Przykład implementacji funkcji kary przedstawiono w równaniu 4.15. W funkcji suma kar mnożona jest przez współczynnik proporcjonalności ρ, który decyduje jak mocno przekroczenie wartości ograniczeń wpływa na wartość rozszerzonej funkcji celu $\Phi(\mathbf{x})$.

$$\Phi(\mathbf{x}) = F_c(\mathbf{x}) + \rho \sum_{i=1}^{M} \left(\max\left\{0, c_i(\mathbf{x})\right\}\right)^2 \tag{4.15}$$

Istotnym problemem, w powyższej metodzie, jest ustalenie odpowiedniej wartości dla współczynnika proporcjonalności kary ρ wszystkich zdefiniowanych ograniczeń. Jeśli wartość współczynnika ρ będzie za duża, spowodować to może przypadki, w których algorytm PSO będzie zatrzymywał się w optimum lokalnym. Natomiast za mała wartość współczynnika ρ może nie wpływać znacząco na sam proces optymalizacji w algorytmie PSO. W tym celu optymalizacja powtarzana jest wielokrotnie, a wartość współczynnika proporcjonalności przyjmuje coraz większe wartości. W konsekwencji współczynnik proporcjonalności ρ dąży do nieskończoności, by znaleźć rozwiązanie zbliżone do standardowego problemu optymalizacyjnego z ograniczeniami. Prowadzi to do problemu optymalizacji , który jest źle uwarunkowany. Ze względów praktycznych, współczynnik proporcjonalności ρ musi posiadać skończoną wartość. Rzutuje to na rozwiązanie problemu optymalizacyjnego, gdyż wynik otrzymany, z wykorzystaniem tej metody, jest jedynie aproksymacją rozwiązania problemu optymalizacji z ograniczeniami.

Jednym z rozwiązań powyższego problemu jest zastosowanie metody Rozszerzonego Lagranżianu (ang. Augmented Lagrangian Method - ALM) w algorytmie PSO [91][38]. Metoda jest zbliżona do metody kwadratowej funkcji kary lecz różni się inną definicją funkcji celu. Rozszerzona funkcja celu, zwana rozszerzonym Lagranżjanem Powell'a-Hestenes'a-Rockafellar'a [90], ma następującą postać:

$$L(\mathbf{x}) = F(\mathbf{x}) + \frac{\rho}{2} \sum_{i=1}^{M} \left(\max\left\{0, c_i(\mathbf{x}) + \frac{\lambda_i}{\rho}\right\}\right)^2 \tag{4.16}$$

gdzie:
λ_i - mnożniki Lagrange'a.

Porównując powyższe równanie z równaniem 4.15, w funkcji kary są dodatkowo uwzględnione mnożniki Lagrange'a λ_i, które dotyczą każdego ograniczenia z osobna. Wprowadzenie

mnożników skutkuje tym, że rozwiązanie problemu minimalizacji, bez ograniczeń rozszerzonego Lagranżjanu, jest identyczne z rozwiązaniem problemu minimalizacji funkcji celu z ograniczeniami ze skończoną wartością współczynnika kary ρ. Dobór współczynnika kary ρ oraz mnożników Lagrange'a λ_i dokonywany jest automatycznie w procesie optymalizacji.

4.3. Algorytm optymalizacyjny zaimplementowany w bloku Kompensatora Trajektorii Zadanej

W bloku optymalizacji Kompensatora Trajektorii Zadanej, wyznaczane są optymalne położenia zadane dla serwonapędów, dla N-elementowego horyzontu predykcji. Poprawa jakości sterowania uzyskiwana jest w wyniku działania algorytmu optymalizacyjnego, który dąży do znalezienia takich nowych przyrostów położeń zadanych (argumentów funkcji celu), dla których błędy nadążania w poszczególnych osiach przyjmują wartości najmniejsze. Na podstawie przeprowadzonej analizy algorytmów optymalizacyjnych, w bloku Kompensatora Trajektorii Zadanej, postanowiono zastosować algorytm optymalizacji rojem cząstek (PSO). Z pośród wielu wariantów algorytmu PSO, opisywanych w literaturze, konieczne było wybranie takiego, który najskuteczniej rozwiązuje problem minimalizacji błędów nadążania w bloku Kompensatora Trajektorii Zadanej. W tym celu autor przeprowadził testy kilku wariantów algorytmów optymalizacyjnych, opisanych w poprzednim rozdziale.

W bloku optymalizacji funkcja celu podlegająca optymalizacji jest określona wzorem 4.3. Ograniczenia wyznaczające jej dopuszczalne rozwiązania określają nierówności 4.2.

W pierwszej kolejności, autor podjął próbę implementacji algorytmu sekwencyjnego programowania kwadratowego (SQP). Jednak zastosowanie tego rozwiązania do omawianego problemu optymalizacyjnego, ze względu na dynamiczny neuronowy model osi maszyny, wyklucza obliczanie gradientu w sposób analityczny. Alternatywnie, gradient może być wyznaczony metodą różnic skończonych. Takie podejście powoduje zwiększenie złożoności obliczeniowej algorytmu oraz wzrost niedokładności obliczeń, wynikających z błędów aproksymacji gradientu. Dodatkowym problemem, w algorytmie gradientowym, jest możliwość znalezienia rozwiązania funkcji celu, które jest optimum lokalnym w jej zbiorze rozwiązań. Takie rozwiązanie funkcji celu może być zdecydowanie gorsze od optimum globalnego. Czas znalezienia optimum globalnego (o ile algorytm nie na trafi na optimum lokalne), jest ściśle uzależniony od warunków początkowych argumentów funkcji celu (ang. initial guess). W omawianym problemie optymalizacji przyrostów położenia, trudno jest wyznaczyć odpowiednie warunki początkowe potrzebne do startu algorytmu gradientowego.

Koleją próbą rozwiązania problemu optymalizacji przyrostów położenia zadanego, było wykorzystanie algorytmu programowania liniowego (LP). Wymaganiem algorytmu jest by

funkcja celu oraz ograniczenia były liniowe, a w ostateczności były zlinearyzowane w przypadku problemu nieliniowego. Użycie algorytmu programowania liniowego, wymagałoby linearyzacji nieliniowego modelu predykcji błędów nadążania, co prowadzi do problemów z błędami aproksymacji. Z tego powodu algorytm ten nie został wykorzystany przez autora do rozwiązania problemu optymalizacji.

Powyższe dwa algorytmy ze względu na omówione ograniczenia nie zostały użyte przez autora w bloku optymalizacji. Postanowiono wykorzystać algorytmy, w których nie ma konieczności wyliczania wartości gradientu funkcji celu lub jej aproksymacji.

Kolejnym krokiem, było przetestowanie przez autora algorytmu optymalizacji rojem cząstek z gwarantowaną zbieżnością (GCPSO) wraz z ograniczeniami uaktualnianymi metodą rozszerzonego Lagranżjanu. W wyniku przeprowadzenia wielokrotnych badań otrzymano satysfakcjonujące wyniki w procesie optymalizacji funkcji celu. Problem zbiegania cząstek roju do położeń wyznaczających lokalne optima nie został zaobserwowany.

W metodzie uwzględniono ograniczenia z zastosowaniem funkcji kary, określonej za pomocą metody rozszerzonego Lagranżjanu. Jednak określone ograniczenia w nierównościach 4.2 umożliwiają rezygnację z tej metody, na rzecz zastosowania prostszej oraz szybszej obliczeniowo metody ograniczeń typu Box. Wyniki badań algorytmu GCPSO, z zastosowaniem tego typu ograniczeń, nie odbiegały znacząco od wyników z użyciem algorytmu uzupełnionego o metodę rozszerzonego Lagranżjanu.

Autor również eksperymentował z wariantem algorytmu GCPSO, w którym współczynniki przyspieszenia oraz bezwładności podlegają adaptacji w trakcie trwania procesu optymalizacji. Otrzymane wyniki nie potwierdziły zmniejszonej ilości iteracji algorytmu, potrzebnej do otrzymania wartości wyjściowej w porównaniu z algorytmem GCPSO, w którym te współczynniki nie były adaptowane. W związku z tym w dalszych badaniach zrezygnowano z adaptacji współczynników algorytmu.

Ze względu na implementację Kompensatora Trajektorii Zadanej w układzie sterownika CNC oraz związane z tym wymagania, przebadany algorytm GCPSO nie mógł być wykorzystany. Czas działania zaproponowanego algorytmu był zbyt długi do wykorzystania go w procesie optymalizacji potrzebnym w sterowaniu predykcyjnym, który działał w trybie on-line. Szczegółowe wymagania dotyczące działania Kompensatora Trajektorii Zadanej omówiono w rozdziale 5.

Z tego względu do badań wybrano standardowy algorytm PSO. Wariant wybranego algorytmu Optymalizacji Rojem Cząstek przedstawiają wzory 4.6 oraz 4.5. Wykorzystywany współczynnik bezwładności miał ustaloną wartość równą 0,7, natomiast współczynniki przyspieszeń miały takie same wartości równe 2,0.

Podczas badań okazało się, że proces optymalizacji ulegał przedwczesnemu zatrzymaniu, zanim znalezione zostało zadowalające rozwiązanie. Przyczyną problemu okazała się zbyt

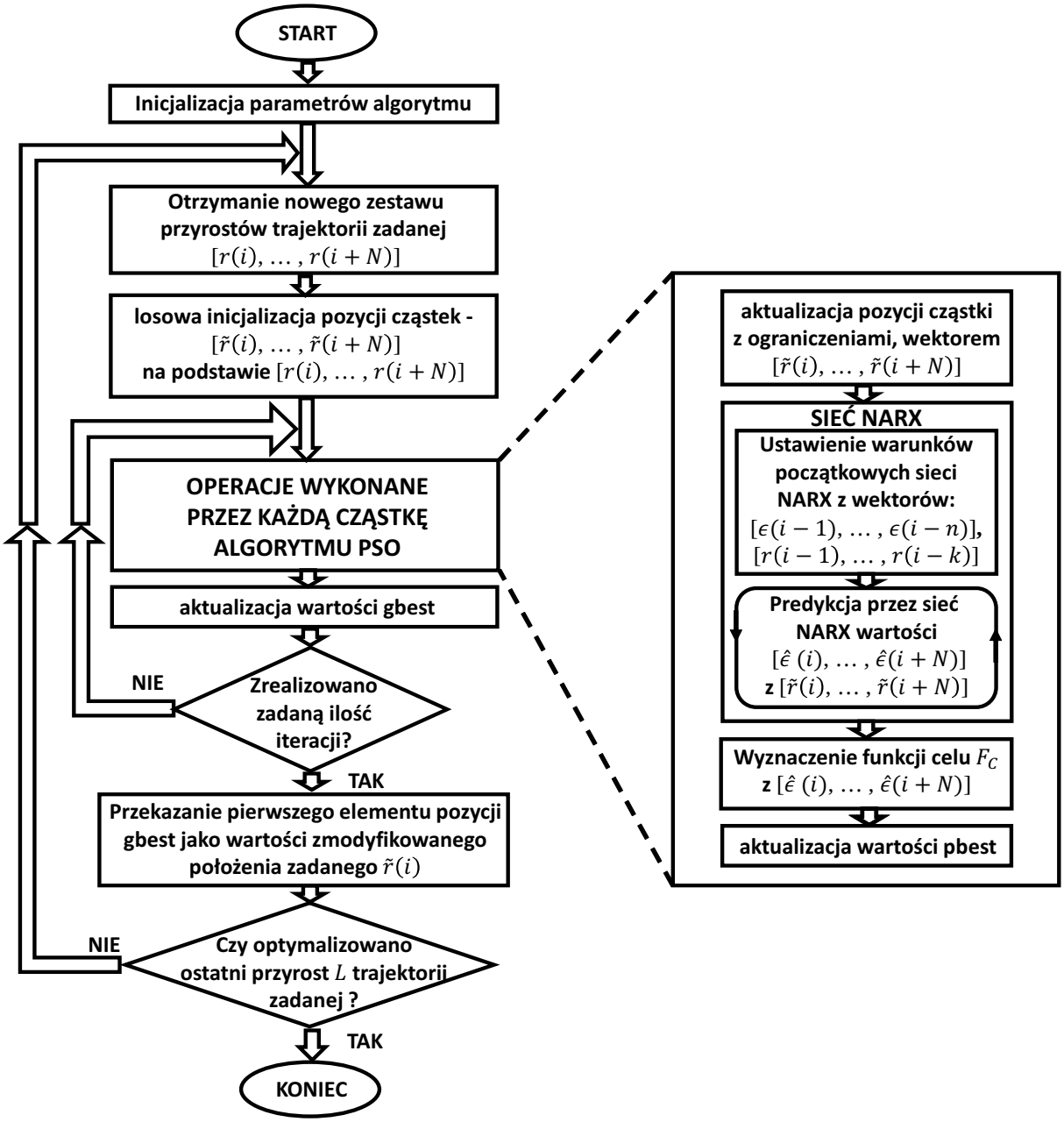
Tabela 4.1. Parametry zaimplementowanego algorytmu PSO

Parametry algorytmu PSO	
Ilość cząstek	10
Horyzont predykcji N	5
Ilość iteracji i	100
Wartość współczynnika bezwładności ω	0,7
Wartości współczynników przyspieszeń ϕ_1, ϕ_2	2,0

duża ilość cząstek oraz niewystarczająca ilość wykonywanych iteracji algorytmu PSO. Metodą prób i błędów ustalono optymalną liczbę cząstek na 10 a liczbę iteracji na 100, które były najmniejszymi wartościami parametrów, pozwalającymi uzyskać zadowalające wyniki optymalizacji. Jednocześnie parametry te zapewniały, że obliczenia zostaną zrealizowane w ustalonym czasie, wymaganym do pracy w trybie on-line. Parametry zaimplementowanego algorytmu PSO dla obu Kompensatorów Trajektorii Zadanej przedstawia tabela 4.1.

Schemat przepływowy działania bloku Kompensatora Trajektorii Zadanej z zaimplementowanym algorytmem optymalizacyjnym PSO przedstawiono na rysunku 4.5.

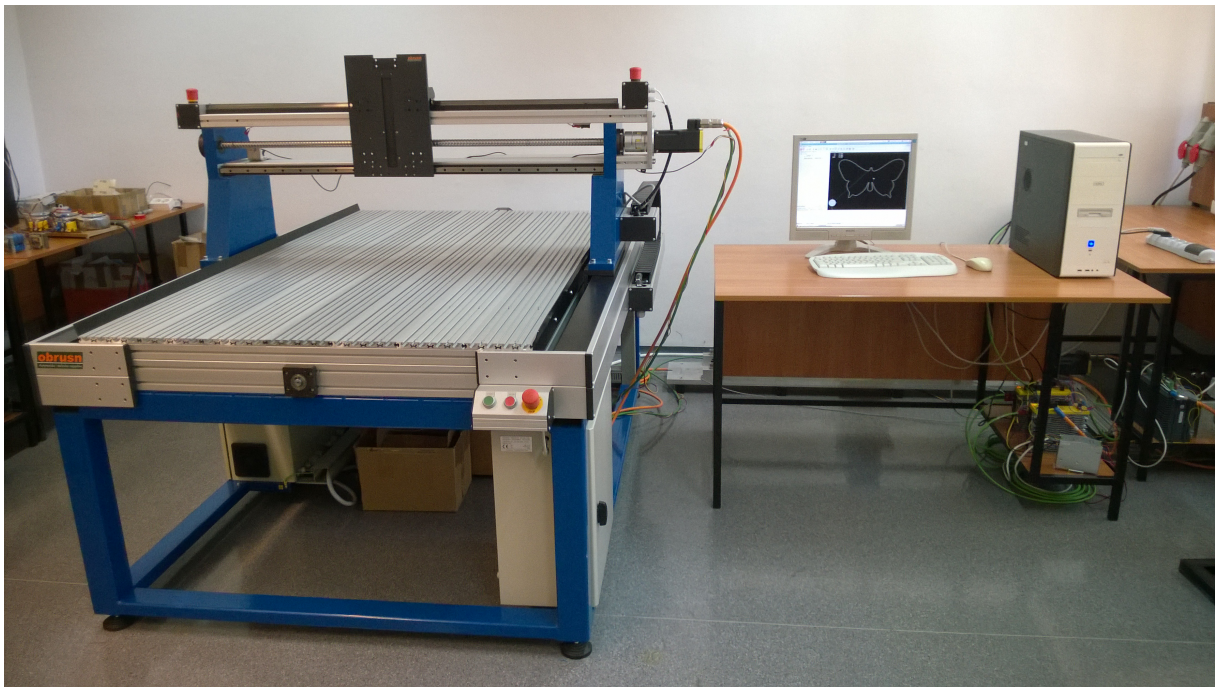
W niniejszym rozdziale zaprezentowano strukturę proponowanego Kompensatora Trajektorii Zadanej (KTZ). Optymalizacji podlegały przyrosty trajektorii zadanej. Działanie algorytmu realizowane było w bloku optymalizacji, który wraz z neuronowym modelem osi mechanicznej składał się na Blok Kompensatora Trajektorii Zadanej. Przedstawiono algorytm Optymalizacji Rojem Cząstek (PSO), zastosowany w KTZ oraz uzasadniono wybór określonego wariantu tego algorytmu. Przedstawiono optymalizowaną funkcję celu oraz przyjęte ograniczenia. Wyniki doświadczalne, weryfikujące działanie Kompensatora Trajektorii Zadanej zaimplementowanego w sterowniku CNC, zostały opisane w rozdziale 6-tym.



Rysunek 4.5. Schemat przepływowy algorytmu optymalizacji przyrostów trajektorii zadanej z wykorzystaniem algorytmu PSO.

5. Stanowisko badawcze

Badania doświadczalne algorytmu minimalizacji błędów nadążania układu posuwu osi mechanicznych, zostały przeprowadzone na stanowisku badawczym. Stanowisko badawcze składa się z otwartego sterownika CNC, zaimplementowanego w komputerze PC [37][36], z zainstalowanym systemem operacyjnym czasu rzeczywistego Linux RTAI[31] oraz komercyjnej dwuosiowej maszyny wieloosiowej, wyprodukowanej przez PIAP OBRUSN Toruń. Zdjęcie stanowiska badawczego przedstawia rysunek 5.1.



Rysunek 5.1. Stanowisko badawcze

5.1. Dwuosiowy zespół osi mechanicznych maszyny CNC

Podstawowymi elementami wykonawczymi części mechanicznej maszyny CNC są dwie osie mechaniczne. W ich budowie można wyodrębnić cześć mechaniczną, w której następuje zamiana ruchu obrotowego na ruch liniowy i serwonapęd. W części napędowej zastosowano silniki obrotowe PMSM o momencie znamionowym 1,5Nm. Układ regulacji w serwonapędzie (Microflex e100 firmy Baldor) jest klasycznym układem kaskadowo połączonych regulatorów

położenia, prędkości i prądu. Każdy z silników PMSM połączony jest za pomocą sprzęgieł kłowych z przekładniami śrubowymi tocznymi (śruby kulowe), w wyniku czego ruch obrotowy silnika zamieniany jest na ruch postępowy suportu części mechanicznej. Parametry osi mechanicznych zostały zestawione w tabeli 5.1.

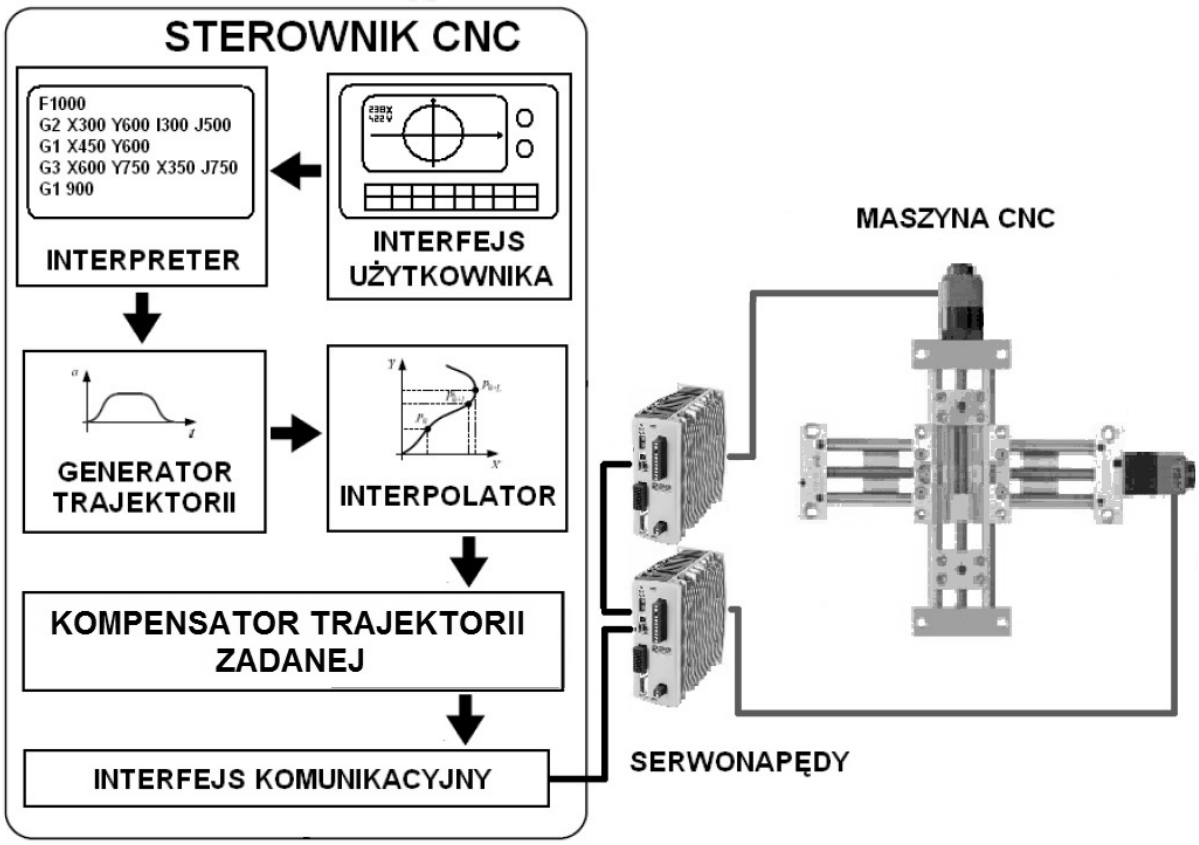
Tabela 5.1. Parametry osi mechanicznych i serwonapędów osi X i Y

Suport mechaniczny	**Oś X**	**Oś Y**
długość suportu	1450 mm	750 mm
przekładnia	1:1(sprzęgło)	1:1(sprzęgło)
przełożenie ruchu	śruba kulowa z nakrętką	śruba kulowa z nakrętką
skok śruby kulowej	10 mm/obr.	10 mm/obr.
Serwonapęd		
typ regulacji	kaskadowy	kaskadowy
regulator prądu	PI	PI
regulator prędkości	PI	PI
regulator położenia	P	P
regulator sprzężenia w przód	brak	brak
układ pomiaru położenia	przetwornik obrotowo-impulsowy (2500 imp./obr)	przetwornik obrotowo-impulsowy (2500 imp./obr)
parametry silnika	PMSM, 3-fazowy, 230VAC, 1,5Nm	PMSM, 3-fazowy, 230VAC, 1,5Nm

5.2. Sterownik CNC zaimplementowany w komputerze PC

Zaprojektowany i zbudowany sterownik CNC, zaimplementowany w komputerze PC, komunikuje się z serwonapędami z wykorzystaniem magistrali komunikacyjnej typu Ethernet Powerlink (EPL) [24]. Magistrala EPL, o topologii liniowej, składa się ze sterownika CNC oraz dwóch serwonapędów Microflex e100 firmy Baldor, posiadających interfejs EPL. Interfejs od strony sterownika CNC bazuje na standardowej karcie sieciowej PCI Ethernet 10/100 Mb/s. Jest to karta RTL8139DL firmy Realtek [89]. Schemat wykorzystywanego systemu sterowania maszyn wieloosiowych CNC został przedstawiony na rysunku 5.2.

W celu zapewnienia odpowiednich parametrów pracy oraz integracji z oprogramowaniem sterownika CNC maszyn wieloosiowych, autor musiał dokonać modyfikacji standardowego stosu EPL. W procesie projektowania i budowy sterownika, jednym z ważniejszych problemów do rozwiązania jest synchroniczne przesyłanie pozycji zadanych do serwonapędów. Wymagane jest by wszystkie informacje były przesyłane oraz przetwarzane przez serwonapędy w tym samym czasie. Dodatkowo częstość przesyłania tych informacji powinna być zawsze stała. W przedstawionym układzie, odstępy czasu przesyłania tych informacji (w tym danych na temat położeń zadanych) zostały określone na 1ms. Sterownik CNC powinien zapewnić



Rysunek 5.2. Schemat wykorzystywanego systemu sterowania maszyn wieloosiowych CNC

wysoki priorytet obsługi tego działania, a zadania o niższym priorytecie (np. obsługa interfejsu graficznego), nie powinny wpływać na fluktuacje tego czasu. Chcąc sprostać tym wszystkim wymogom, sterownik CNC musi działać na bazie systemu operacyjnego czasu rzeczywistego (ang. Real Time Operating System - RTOS). Autor zainstalował i dostosował do potrzeb sterownika CNC otwarty (ang. open source) system operacyjny czasu rzeczywistego Linux RTAI. Wykorzystywany RTOS jest systemem twardego czasu rzeczywistego (ang. Hard Real Time Operating System) co oznacza, że zadania czasu rzeczywistego muszą być zawsze realizowane w określonym, nieprzekraczalnym czasie. Wątki czasowo-krytyczne realizowane są z rozrzutami czasowymi (ang. jitter) oraz opóźnieniami (ang. latency) rzędu pojedynczych mikrosekund. Determinizm czasowy zapewniany przez RTOS Linux RTAI jest porównywalny z komercyjnymi rozwiązaniami, takim jak: VxWorks czy QNX [9].

Wysoki determinizm czasowy systemu Linux RTAI jest wynikiem modyfikacji jądra standardowego systemu Linux. Uproszczoną strukturę systemu Linux RTAI przedstawiono na rysunku 5.3. Budowa RTOS bazuje na architekturze mikrojądra (ang. microkernel). Standardowe zadania sytemu Linux (jądro systemu Linux oraz aplikacje przestrzeni użytkownika), nie wymagają działania w czasie rzeczywistym - działają jako proces o najniższym priorytecie (tzw. zadania w tle). Wszystkie zadania wymagające determinizmu

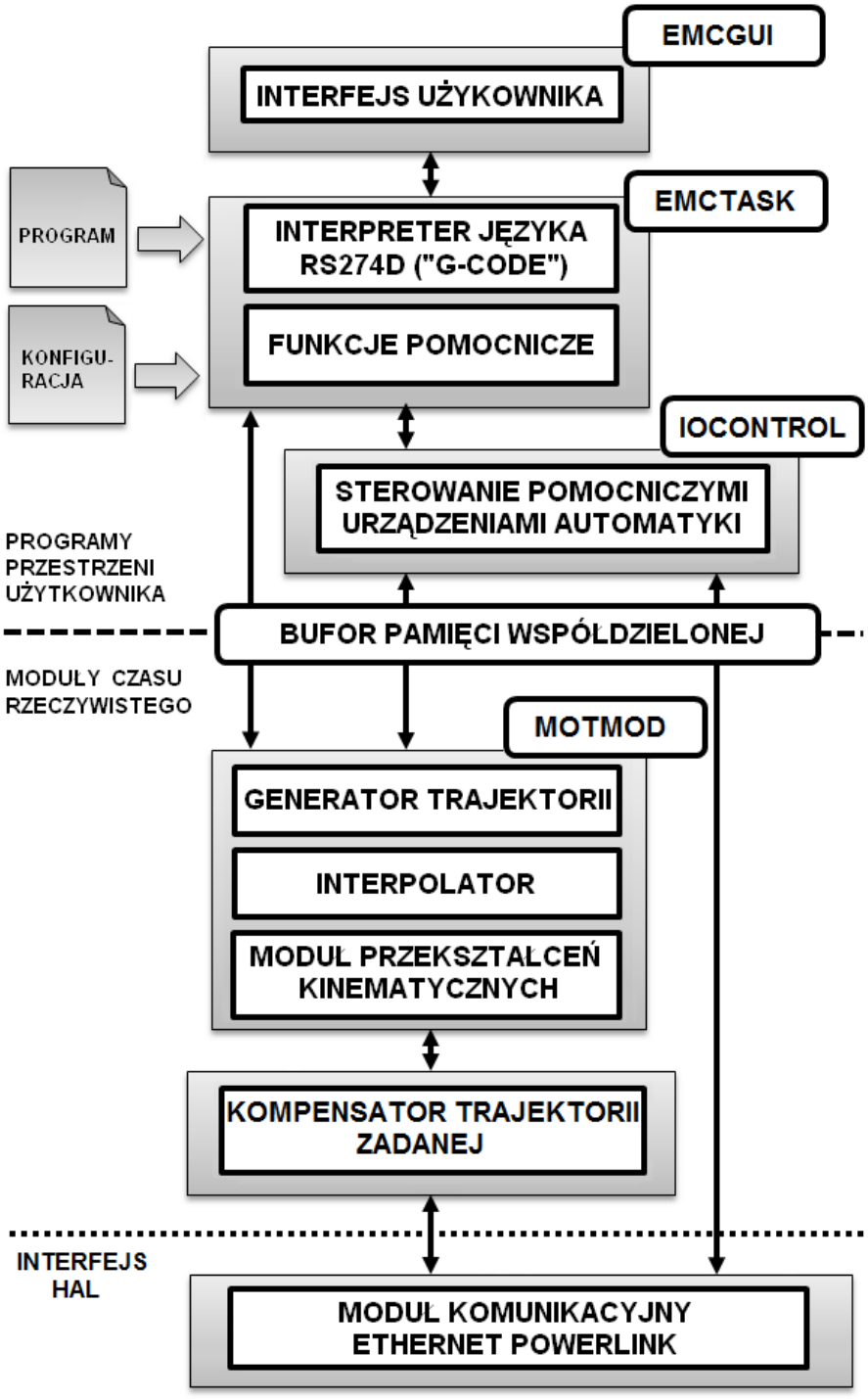
czasowego (zadania czasu rzeczywistego), działają z wyższym priorytetem niż system Linux. Procesy te obsługiwane są bezpośrednio przez mikrojądro, w odpowiedniej kolejności (uszeregowaniu) na bazie przyznanych im wcześniej priorytetów. Procesy o priorytecie wyższym, realizowane są przed zadaniami o priorytecie niższym i nie mogą być przez nie wywłaszczone. Szeregowaniu podlega również obsługa przerwań. Przerwania o niższym priorytecie (obsługiwane przez system Linux) są realizowane dopiero po zakończeniu realizacji wszystkich przerwań i zadań czasu rzeczywistego w danym cyklu.

Rysunek 5.3. Uproszczona struktura systemu Linux RTAI

Aplikacją bezpośrednio zarządzającą pracą maszyny wieloosiowej w sterowniku CNC jest aplikacja LinuxCNC [99]. Schemat budowy aplikacji LinuxCNC jest przedstawiony na rysunku 5.4. W aplikacji można wyróżnić podział na dwie części: bloki obsługiwane w przestrzeni użytkownika i moduły czasu rzeczywistego. Bloki przestrzeni użytkownika wykonywane są z niższym priorytetem, na równi z pozostałymi programami w przestrzeni systemu Linux. Moduły czasu rzeczywistego obsługiwane są jako zadania czasu rzeczywistego przez mikrojądro.

Blok EMCGUI jest najwyżej położoną częścią składową aplikacji LinuxCNC i bezpośrednio realizuje zadania związane z interfejsem użytkownika. W bloku wizualizowana jest trajektoria ruchu w trzech wymiarach oraz dodatkowe specjalne funkcyjne kontrole (ustalanie offsetów programowych trajektorii ruchu, ograniczania globalne na prędkość realizacji trajektorii ruchu, itp.). Jeśli jest to konieczne, istnieje możliwość dodatkowych modyfikacji interfejsu graficznego użytkownika do własnych potrzeb.

Blok EMCTASK zawiera blok interpretera języka „G-code”. W bloku odczytywane są pliki konfiguracyjne ustawień maszyny CNC oraz pliki trajektorii ruchu w postaci „G-code”. Blok EMCTASK realizuje również dodatkowe funkcje określone przez użytkownika. Blok EMCTASK komunikuje się z blokiem IOCONTROL, w którym realizowane są zadania związane z obsługą portów wejściowych i wyjściowych maszyny CNC. Bloki przestrzeni użytkownika komunikują się z modułami czasu rzeczywistego przez bufor pamięci współdzielonej.



Rysunek 5.4. Schemat budowy aplikacji LinuxCNC

Wszystkie moduły czasu rzeczywistego działają w przestrzeni jądra systemu w czasie rzeczywistym. Obsługiwane są przez mikrojądro, które dopuszcza do ich wywłaszczenia przez zadnia systemu Linux lub inne procesy przestrzeni czasu rzeczywistego z niższym priorytetem. W mikrojądrze przechwytywane są wszystkie przerwania zgodnie z ich zaszeregowaniem względem priorytetu obsługi.

W bloku MOTMOD realizowane są czasowo krytyczne zadania związane z generacją trajektorii zadanej, dlatego blok ten działa w przestrzeni czasu rzeczywistego. Do bloku z bufora pamięci współdzielonej przekazywane są informacje na temat trajektorii ruchu, gdzie w bloku generatora trajektorii dochodzi do jej generacji. Następnie dane przekazywane są do bloku interpolatora a finalnie do bloku przekształceń kinetycznych, gdzie wyznaczane są posuwy dla dwóch osi mechanicznych. Blok MOTMOD zawiera również funkcje umożliwiające wygładzenie trajektorii ruchu z ostrych krawędzi, w ramach zadanej tolerancji jej realizacji.

Do opracowanego przez autora bloku Kompensatora Trajektorii Zadanej przesyłane są zbuforowane informacje o aktualnym przemieszczeniu oraz kolejnych przyszłych zadanych położeniach osi mechanicznych. Informacje te są potrzebne na użytek zaimplementowanego algorytmu optymalizacji trajektorii ruchu. Ze względu na złożoność i wymaganą szybkość wykonywania obliczeń, blok ten działa również w przestrzeni czasu rzeczywistego.

W bloku interfejsu komunikacyjnego EPL został zaimplementowany przez autora stos komunikacyjny Ethernet Powerlink. Stos ma postać modułu jądra systemu Linux i działa w przestrzeni czasu rzeczywistego. W celu wymiany informacji z pozostałymi blokami tej przestrzeni (blok Kompensatora Trajektorii Zadanej oraz MOTMOD) został użyty specjalny interfejs aplikacyjny HAL (ang. hardware abstraction layer), który jest częścią składową aplikacji LinuxCNC. HAL umożliwia uruchomienie modułów czasu rzeczywistego, przydzielenie im priorytetu działań oraz wymianę informacji pomiędzy nimi (ang. interprocess communication - IPC).

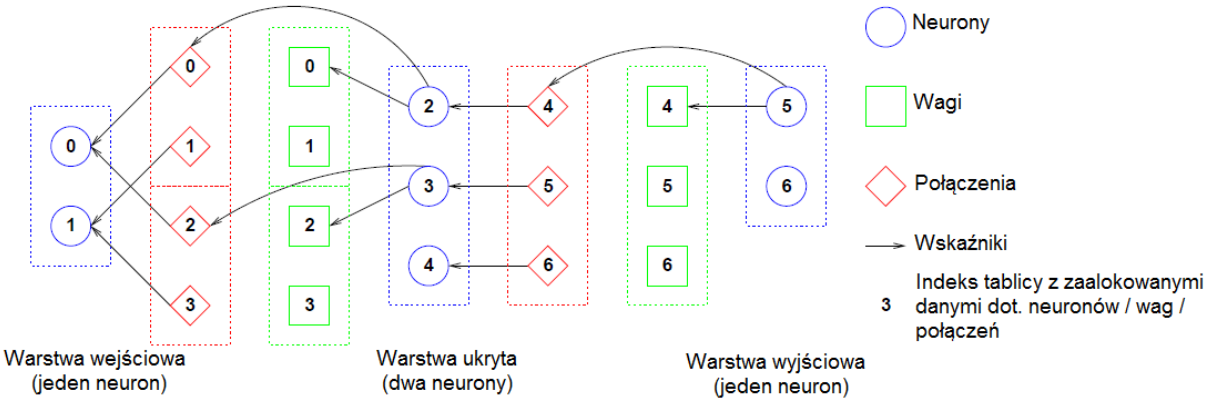
Komunikacja IPC pomiędzy modułami w przestrzeni HAL jest możliwa dzięki wirtualnym portom. Z punktu widzenia programisty porty stanowią zmienne w pamięci współdzielonej (ang. shared memory) pomiędzy różnymi modułami. Zmienne te są widoczne dla pozostałych modułów w przestrzeni HAL oraz mogą być użyte do wymiany danych pomiędzy nimi. Konfiguracja połączeń portów oraz priorytet modułów, definiowane są w specjalnym pliku konfiguracyjnym, uruchamianym podczas startu aplikacji LinuxCNC.

5.3. Bloki Kompensatorów Trajektorii Zadanej

Bloki Kompensatorów Trajektorii Zadanej, zbudowane są z bloków optymalizacji i neuronowych modeli osi mechanicznych.

Blok optymalizacji, bazuje na standardowym algorytmie Optymalizacji Rojem Cząstek (PSO). Szczegóły dotyczące działania tego bloku zostały opisane w rozdziale 4-tym. Blok w całości został opracowany i zaimplementowany przez autora w języku ANSI C.

W celu implementacji sieci NARX w strukturach neuronowego modelu osi mechanicznej, autor pracy zaadaptował otwarte biblioteki FANN autorstwa S. Nissen'a [79]. Biblioteki umożliwiają zaalokowanie struktury sieci NARX w oprogramowaniu sterownika CNC z wykorzystaniem języka programowania ANSI C. Struktura biblioteki FANN umożliwia implementację sieci neuronowej zoptymalizowanej pod kątem szybkości obliczeń, co ma szczególne znaczenie podczas działania sterownika CNC. Przykład rozmieszczenia struktury danych sieci neuronowej typu FF (1-2-1) z wykorzystaniem biblioteki FANN przedstawia rysunek 5.5.



Rysunek 5.5. Przykład rozmieszczenia struktury danych sieci neuronowej typu FF (1-2-1) z wykorzystaniem biblioteki FANN

W celu adaptacji biblioteki FANN do potrzeb neuronowego modelu osi mechanicznych, autor dokonał wymaganych modyfikacji. Konieczne było dostosowanie funkcji alokacji pamięci i struktur biblioteki do potrzeb i wymagań modułów czasu rzeczywistego. Usunięto również z biblioteki FANN funkcji służących do uczenia sieci neuronowych, gdyż nie były one wykorzystywane podczas pracy neuronowego modelu osi mechanicznych. Fragmenty kodu źródłowego zaimplementowanego bloku Kompensatora Trajektorii Zadanej przedstawiono w załączniku 1.

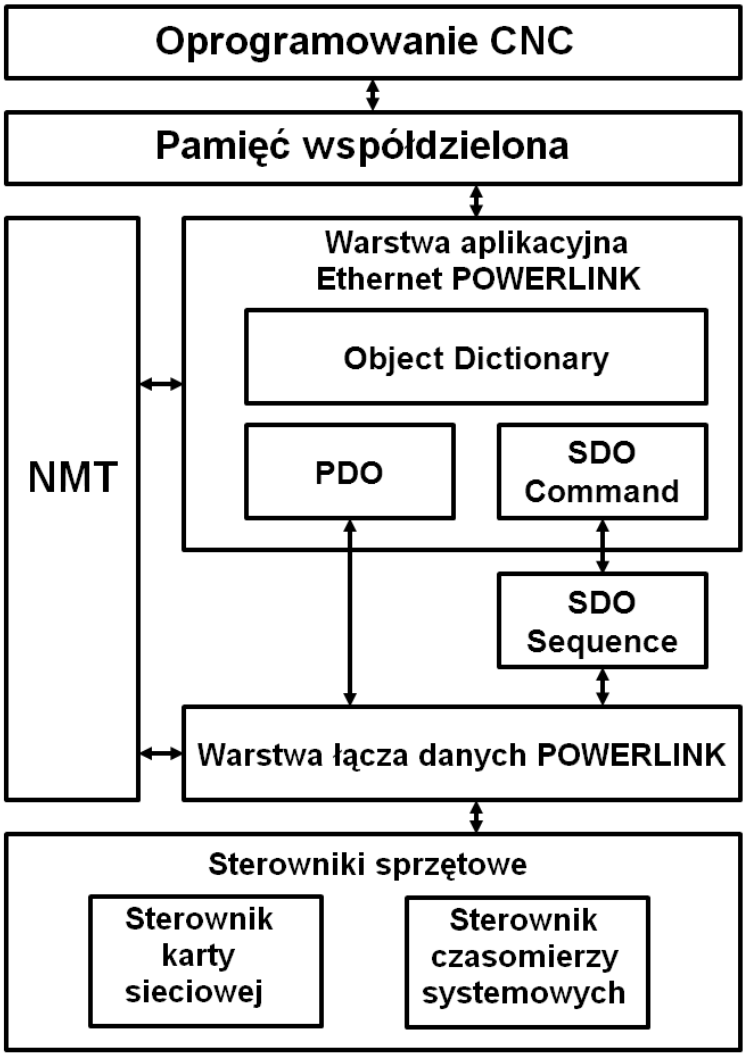
5.4. Blok interfejsu komunikacyjnego Ethernet Powerlink

Blok interfejsu komunikacyjnego Ethernet Powerlink zawiera stos komunikacyjny składający się z następujących modułów:

— sterowniki niskiego poziomu (obsługa karty sieciowej PCI Ethernet e8139 Realtek);

— stos protokołu Ethernet Powerlink (w tym biblioteka Object Dictionary - OBD, warstwa łącza danych oraz maszyna stanów - network management - NMT);
— maszyny stanów profilu CiA402 [2] oraz interfejs komunikacyjny z HAL.

Schemat bloku interfejsu komunikacyjnego Ethernet Powerlink przedstawia rysunek 5.6.



Rysunek 5.6. Schemat bloku interfejsu komunikacyjnego Ethernet Powerlink

Standardowy protokół komunikacyjny Ethernet TCP/IP wykorzystuje protokół wielodostępu CSMA/CD (ang. Carrier Sense Multiple Access with Collision Detection)[50]. Jego zadaniem jest zarządzanie dostępem do sieci oraz wykrywanie kolizji pakietów przesyłanych przez urządzenia (węzły) tworzące tę sieć. Kolizje pakietów mogą wystąpić w przypadku ich jednoczesnego nadawania przez przynajmniej dwa węzły. W momencie wystąpienia kolizji, transmisja jest przerywana a następnie wznawiana w sposób losowy. Z tego powodu transmisja z wykorzystaniem Ethernetu TCP/IP jest niedeterministyczna i nie może być stosowana do synchronicznego sterowania serwonapędami maszyn wieloosiowych.

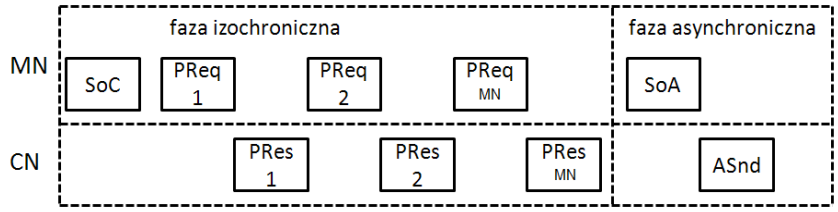
Ethernet Powerlink (EPL) w przeciwieństwie do Ethernet TCP/IP jest izochronicznym protokołem komunikacyjnym, który spełnia wysokie wymagania stawiane w aplikacjach sterowania ruchem. Porównując EPL do Ethernet TCP/IP można zauważyć, że warstwa fizyczna (warstwa 1 OSI[1]) pozostaje niezmieniona. Dzięki temu możliwe jest zastosowanie standardowych interfejsów kart sieciowych dostępnych w komputerach PC. Warstwa łącza danych (OSI warstwa 2) została rozszerzona o mechanizm obsługi zadań w trybie czasu rzeczywistego. Warstwa sieci (warstwa 3 modelu OSI) nie jest używana. Reszta warstw została zastąpiona przez analogiczne warstwy stosu EPL. Stos EPL zarządzany jest przez maszynę stanów NMT.

W standardowej sieci EPL wyróżnia się jedno urządzenie nadzorcze (ang. Managing Node - MN) oraz urządzenia podrzędne (ang. Control Node - CN), których w sieci może być maksymalnie 240. Sieć EPL może być zbudowana w topologi liniowej, gwiazdy, lub topologi drzewa. Urządzenie MN zarządza komunikacją i konfiguracją pracy urządzeń CN w sieci Powerlink.

Przesyłanie danych za pośrednictwem magistrali EPL odbywa się w cyklicznych sekwencjach komunikacyjnych. Pojedynczy cykl komunikacyjny składa się dwóch faz: fazy izochronicznej i fazy asynchronicznej. Faza izochroniczna działa na zasadzie szczelin czasowych (krótkich przedziałów czasowych przeznaczonych do obsługi danego urządzenia). Tylko jeden węzeł typu CN może dokonać transmisji danych do węzła MN, w ściśle określonym przedziale czasowym. Protokół ten nazywany jest wielodostępem z podziałem czasowym (ang. Time Division Multiple Access - TDMA). Wykorzystanie TDMA powoduje całkowite wyeliminowanie kolizji pakietów sieciowych, co zapewnia wysoki poziom determinizmu czasowego. Podczas fazy asynchronicznej, MN przyznaje prawo do transmisji pojedynczemu CN lub sobie, w zależności od potrzeb.

Schemat cyklu komunikacyjnego EPL dla dwóch węzłów typu CN przedstawiony jest na rysunku 5.7. Cykl rozpoczyna się od ramki „początek cyklu” (ang. Start of Cycle - SoC), transmitowanej przez urządzenie MN. Ramka ta wyznacza start fazy izochronicznej oraz dokonuje synchronizacji działania wszystkich urządzeń typu CN. Następnie urządzenie MN wysyła ramkę typu „żądania odpowiedzi” (ang. Poll Request - PRq) do pojedynczego węzła typu CN. Urządzenie CN w odpowiedzi na ramkę typu PRq przesyła ramkę typu „odpowiedź” (nag. Poll Response - PRS) z odpowiednimi danymi. Sekwencja ta jest powtarzana dla każdego węzła typu CN w sieci EPL. Jeśli wszystkie urządzenia CN zostaną obsłużone, urządzenie MN wysyła ostatnią ramkę typu PRq zaadresowaną do siebie. Następnie MN odpowiada na tę ramkę, wysyłając ramkę typu PRs, sygnalizując koniec fazy izochronicznej. W fazie asynchronicznej, MN wysyła jako pierwszy ramkę typu „początek fazy asynchronicznej” (ang. start of start of asynchronous - SoA) do wszystkich węzłów sieci. Dane zawarte w tej ramce zawierają informację, który z węzłów powinien odpowiedzieć na zapytanie MN. Następnie

wyznaczony węzeł wysyła ramkę typu „odpowiedź asynchroniczna” (ang. Asynchronous send - Asnd) z odpowiednimi danymi. Urządzenie MN, po otrzymaniu danych czeka określony czas, by znowu poprzez wysłanie ramki typu Soc rozpocząć następny cykl komunikacyjny EPL.



Rysunek 5.7. Cykl komunikacyjny w Ethernet Powerlink dla dwóch węzłów typu CN

Format ramki w protokole Ethernet Powerlink jest przedstawiony na rysunku 5.8. Ramka EPL bazuje na standardowej ramce Ethernet. Dane Ethernet Powerlink są enkapsulowane w polu danych ramki Ethernet. Pola danych związane z protokołem EPL składają się z czterech głównych części:

— typ ramki (SoC, PRQ, PRS, SOA, lub Asnd)
— adres węzła docelowego
— adres węzła źródłowego
— informacja o ilości przesyłanych danych.

	Ofset bitowy								format ramki
Ofset oktetowy	7	6	5	4	3	2	1	0	
0 .. 5	Adres MAC adresata								Ethernet II
6 .. 11	Adres MAC nadawcy								
12 .. 13	Typ ramki Ethernet								
14	zast.	Typ wiadomości							Ethernet POWERLINK
15	Adresat								
16	Nadawca								
17 .. n	Dane								
n+1 .. n+4	Suma kontrolna								Ethernet II

Rysunek 5.8. Format ramki w protokole Ethernet Powerlink

Warstwa aplikacji stosu EPL bazuje na standardzie CANOpen [2], której głównym składnikiem jest biblioteka Object Dictionary (OBD). W strukturze OBD zawarte są dane o wszystkich parametrach, odnoszących się do stosu komunikacyjnego Powerlink (np. czas cyklu, tolerancje jitteru , ilość węzłów, itp.), a także zmienne procesowe przesyłane pomiędzy urządzeniem MN a urządzeniami CN. Każda zmienna (obiekt), ma swój własny szesnastkowy identyfikator (indeks) w bibliotece OBD. Strukturę EPL OBD przedstawia tabela 5.2. Object Dictionary jest pośrednikiem między niższymi warstwami stosu EPL, a aplikacją.

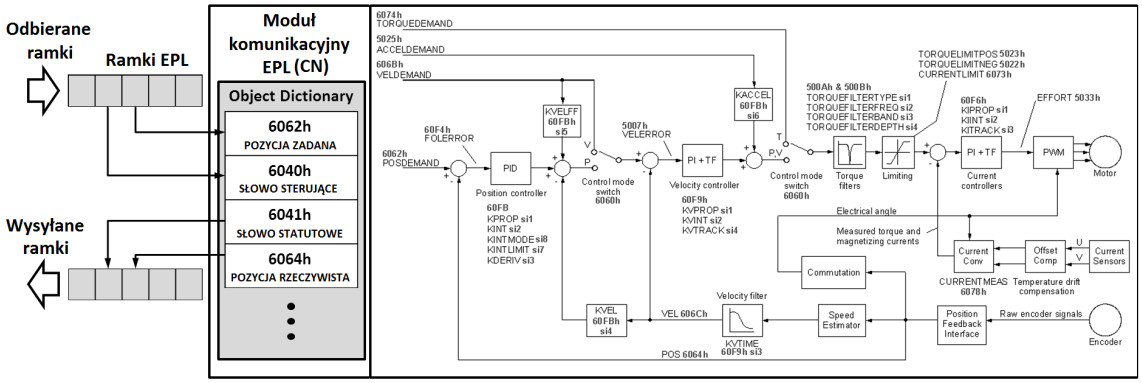
Tabela 5.2. Struktura biblioteki OBD w Ethernet Powerlink

0000h	Nie używane
0001h – 001Fh	Standardowe typy danych np. Boolean, Integer16, itp.
0020h – 001Fh	Struktury składające się ze standardowych typów danych
0040h – 005Fh	Sekcja wykorzystywana przez producenta
0060h – 007Fh	Sekcja danych standardowych dla urządzenia
0080h – 009Fh	Sekcja struktur danych standardowych dla urządzenia
00A0h – 03FFh	Zarezerwowane dla przyszłych zastosowań
0400h – 041Fh	Sekcja standardowych danych służących do konfiguracji Powerlink
0420h – 04FFh	Sekcja struktur danych wykorzystywana przez Powerlink
0500h – 0FFFh	Zarezerwowane dla przyszłych zastosowań
1000h – 1FFFh	Obszar profilu komunikacji
2000h – 5FFFh	Obszar profilu producenta
6000h – 9FFFh	Ustandaryzowany profil urządzenia
A000h – BFFFh	Ustandaryzowany profil interfejsu
C000h – FFFFh	Zarezerwowane dla przyszłych zastosowań

Obiekty biblioteki OBD mogą być przypisane (mapowane) do „obiektu danych procesowych” (ang. process data object - PDO) lub do „obiektu danych serwisowych” (ang. service data object - SDO). Obiekty wysyłane i odbierane w fazie izochronicznej są mapowane w PDO, natomiast obiekty przesyłane w fazie asynchronicznej w SDO. Obiekty mapowane do SDO przeważnie związane są danymi konfiguracyjnymi węzłów (np. nastawami regulatorów serwonapędu), niekiedy kilku urządzeń CN. Warstwa SDO służy do sekwencyjnego wysyłania kolejnych obiektów mapowanych w SDO, ponieważ w fazie asynchronicznej przesyłany jest jeden obiekt do jednego urządzenia CN. Standard EPL alternatywnie umożliwia wysyłanie obiektów mapowanych w SDO za pomocą protokołu TCP/IP z pominięciem stosu EPL. Każde urządzenie w sieci EPL posiada własną bibliotekę OBD z odpowiednio mapowanymi obiektami w PDO i SDO. Komunikacja izochroniczna i asynchroniczna polega na wymianie danych z odpowiedniego obiektu biblioteki poprzez ramki PRq i PRs, w celu uaktualnienia wartości tych obiektów (czy to w MN, czy węzłach CN).

Oprócz standardowych obiektów wynikających z dokumentacji EPL, istnieje przestrzeń indeksów (sekcja) w OBD, w której możliwe jest zdefiniowanie własnych obiektów użytkownika. Sekcje te są zdefiniowane w odrębnych normach, w zależności od typu urządzenia CN. Standardy te, znane jako profile urządzeń (ang. device profiles), określają zestaw obiektów zawierających konfigurację zmiennych procesu, typowych dla danego typu CN. Profil urządzenia dla seronapędów jest zdefiniowany w standardzie CiA402. Każdy serownapęd obsługujący protokół EPL, musi mieć zaimplementowaną bibliotekę OBD w standardzie CiA402. Obiektami definiowanymi przez CiA402 mogą być dane dotyczące parametrów pozycji zadanej, pozycji rzeczywistej, wartości prędkość, pomiaru wartości prądu, stanu logicznego cyfrowych wejść/wyjść, wartości współczynników wzmocnień regulatorów

typu PID, konfiguracji trybu pracy seronapędu (praca w trybie momentowym, zadanej prędkości lub położenia) i wiele innych. Opracowany sterownik CNC w pełni wspiera standard CiA402. Struktura zdefiniowana w CiA402 jest w pełni obsługiwana. Schemat przykładowego mapowania biblioteki OBD CiA402 dla serwonapędu Baldor MicroFlex e100 pokazano na rysunku5.9.



Rysunek 5.9. Schemat przykładowego mapowania biblioteki OBD CiA402 dla serwonapędu Baldor MicroFlex e100.

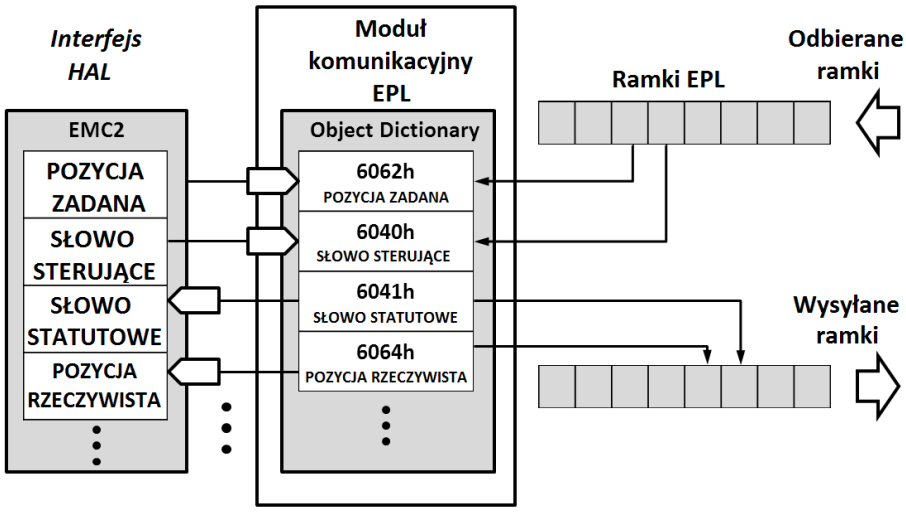
Oprócz zdefiniowanych obiektów OBD, wynikających z samego standardu EPL, biblioteka CiA402 definiuje strukturę maszyny stanów, która kontroluje działania każdego serwonapędu. Aktualny stan odczytywany jest na podstawie wartości obiektu „słowo statusowe” (ang. status word). Zmiana stanu maszyny stanów w serwonapędzie dokonywana jest przez MN poprzez modyfikację obiektu „słowo sterujące” (ang. control word). Wymiana informacji na temat wartości tych dwóch obiektów jest konieczna do prawidłowego uruchomienia i sterowania serwonapędami.

W celu sterowania dwoma serwonapędami na stanowisku badawczym, autor zaimplementował i dostosował moduł komunikacyjny EPL w sterowniku CNC. Opracowano również bibliotekę CiA402 wraz z maszyną stanów dla każdego z seronapędów.

Schemat poglądowy wymiany danych (mapowanie obiektów z biblioteki OBD CiA402) pomiędzy aplikacją LinuxCNC, a modułem komunikacyjnym EPL przedstawia rysunek 5.10.

5.5. Integracja modułu EPL z układem sterownia LinuxCNC

Stos EPL został zaimplementowany w systemie Linux RTAI jako moduł czasu rzeczywistego. Stos bazuje na otwartym kodzie źródłowym OpenPOWERLINK w wersji 1.6 [112]. Konieczne było zaadaptowanie stosu EPL oraz dokonanie szeregu modyfikacji kodu źródłowego wraz z rozbudową o nowe funkcje, w celu sterowania serwonapędami z poziomu systemu Linux RTAI. W sterowniku w warstwie aplikacji stosu EPL autor zaimplementował bibliotekę OBD z profilem sprzętowym CIA402 do komunikacji z serwonapędami. Obiekty



Rysunek 5.10. Schemat poglądowy wymiany danych (mapowanie obiektów z biblioteki OBD CiA402) pomiędzy aplikacją LinuxCNC, a modułem komunikacyjnym EPL.

OBD zawierające zmienne procesowe (PDO) są wysyłane w ramkach PRq i PRs. Indeksy biblioteki OBD transmitowane pomiędzy sterownikiem CNC a serwonapędami w fazie izochronicznej przedstawia tabela 5.3.

Tabela 5.3. Indeksy biblioteki OBD (PDO) transmitowane pomiędzy sterownikiem CNC a serwonapędami w fazie izochronicznej.

Dane ramki PRq	Dane ramki PRs
Pozycja zadana (6062h)	Pozycja rzeczywista (6064h)
Stan wyjśc cyfrowych(60hFE)	Błąd nadążania (60F4h)
Tryb pracy (6060h)	Stan wejść cyfrowych(60FDh)
Słowo sterujące (6040h)	Słowo statusowe(6041h)

Wprowadzono zmiany w stosie EPL, w systemie Linux RTAI a także konfiguracji podstawowej komputera PC (ustawieniach BIOS), w celu dostosowania stosu EPL na bazie biblioteki OpenPOWERLINK do wymogów systemu operacyjnego Linux RTAI oraz zapewnienia stabilnej komunikacji.

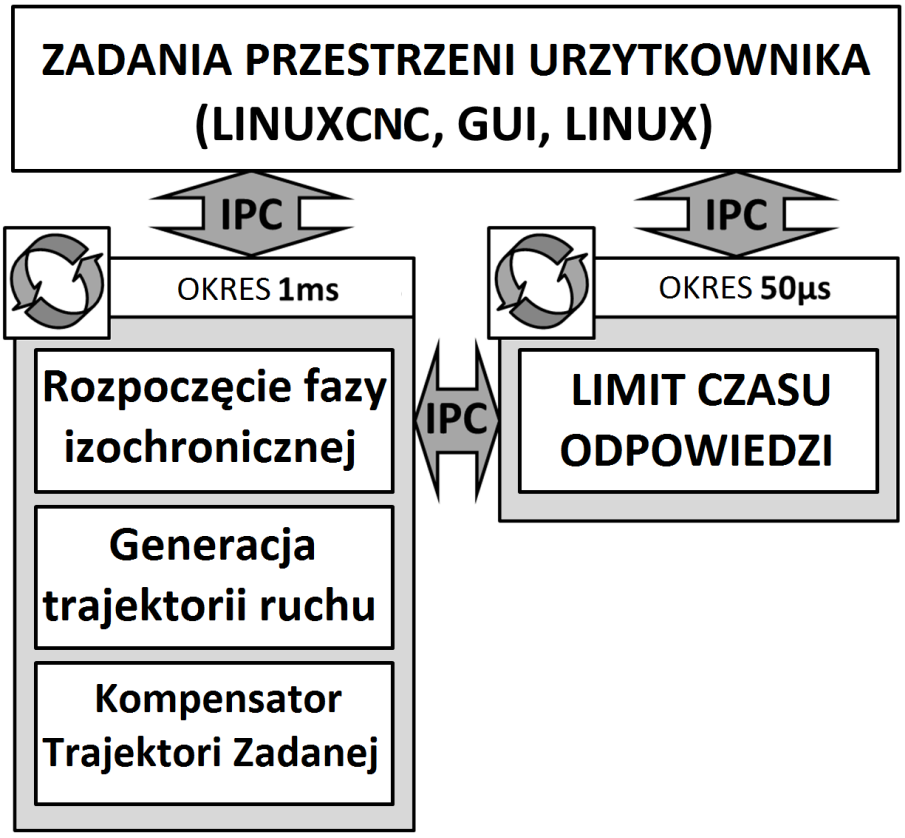
Dokonano następujących zmian stosie EPL:

— Współdzielone bufory pamięci zostały zastąpione przez bezpośrednie wywołania funkcji, w celu zwiększenia wydajności odczytu i zapisu danych.

— Funkcje przestrzeni jądra Linux zostały zastąpione przez ich odpowiedniki biblioteki RTAI (alokacja pamięci, blokady pętlowe spinlock i operacje atomowe).

— Standardowa obsługa przerwań karty sieciowej w systemie Linux została zastąpiona przez obsługę przerwań z wykorzystaniem biblioteki RTAI. Zabieg ten miał na celu natychmiastową obsługę przerwania karty sieciowej przez mikrojądro RTAI. Ze względu na szeregowanie przerwań, przerwania od karty sieciowej były obsługiwane w pierwszej kolejności.

— Struktury liczników systemu Linux zostały zastąpione ich odpowiednikami w bibliotece RTAI. Wykorzystywane są dwa wątki w przestrzeni HAL, realizowane w zdefiniowanych odstępach czasowych. Rozpoczęcie fazy izochronicznej (SoC) realizuje jeden cykliczny wątek o okresie 1 ms. Wątek ten jest wspólny dla zadań związanych z generacją trajektorii ruchu oraz jej dalszą interpolacją. W tym wątku działa również blok Kompensatora Trajektorii Zadanej. Drugi wątek o okresie 50μs służy do monitorowania czasów pracy węzłów sieci EPL, w celu wykrycia przekroczenia dozwolonego limitu czasu odpowiedzi. Wątki te wykorzystują Zaawansowany Programowalny Kontroler Przerwań (ang. Advanced Programmable Interrupt Controller - APIC), który pracuje w trybie okresowym. Schemat poglądowy wątków w przestrzeni czasu rzeczywistego użytych w sterowniku CNC przedstawia rysunek 5.11.



Rysunek 5.11. Schemat poglądowy wątków w przestrzeni czasu rzeczywistego użytych w sterowniku CNC.

Wprowadzono następujące zmiany w konfiguracji systemu Linux RTAI:

— Zbędne w procesie sterowania maszyny CNC urządzenia i zasoby komputera PC (np. karta dźwiękowa) zostały wyłączone.

— W konfiguracji magistrali PCI w komputerze PC (BIOS), zostało przydzielone przerwanie o wysokim priorytecie obsługi dla karty sieciowej. Przerwanie to nie jest współdzielone z innymi przerwaniami występującymi w magistrali PCI.
— Zadania czasu rzeczywistego (generacja trajektorii ruchu, obsługa stosu komunikacyjnego EPL oraz blok Kompensatora Trajektorii Zadanej) skonfigurowano do działania na oddzielnym, izolowanym rdzeniu procesora dwurdzeniowego (poprzez wykorzystanie parametru ISOLCPUS przekazywanego do systemu operacyjnego w trakcie uruchamiania komputera). Ponadto obsługę przerwania karty sieciowej przypisano tylko do rdzenia pracującego w trybie pracy czasu rzeczywistego. Obsługa pozostałych przerwań została przypisana do drugiego rdzenia, w celu uniknięcia zakłóceń wykonywania zadań i wątków czasu rzeczywistego.

Zmodyfikowano ustawienia komputera PC (BIOS), takie jak wyłączenie:
— zintegrowanej karty dźwiękowej;
— funkcje oszczędzania energii;
— skalowania częstotliwość procesora;
— monitorowania temperatury;
— dynamicznej regulacji prędkości wentylatora;
— obsługi portów USB;
— systemu monitoringu dysków twardych S.M.A.R.T..

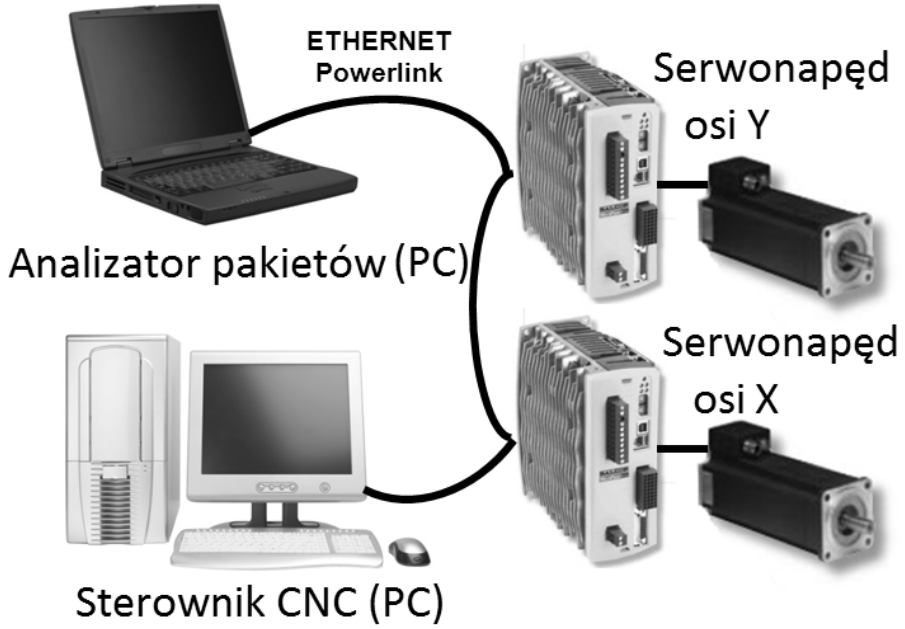
Większość z powyższych ustawień BIOS obsługiwana jest przez niemaskowane przerwania sprzętowe, które nie mogą być zablokowane z poziomu działającego systemu operacyjnego. Obsługa tych przerwań, podczas działania systemu Linux RTAI w trybie czasu rzeczywistego, powoduje powstawanie niedopuszczalnych jitterów i latencji w czasie działania wątków RTOS.

Autor prewencyjnie wyłączył działanie bloku bezpośredniego dostępu do pamięci (ang. Direct Memory Acces - DMA) dla dysków twardych. Wyłączony został również system zarządzania przerwaniami (ang. system management interrupts - SMI) w układzie chipsetu płyty głównej. Układ ten w nowoczesnych płytach głównych wykonywany jest do różnych zadań, takich jak kontrola temperatury układów płyty głównej, system raportowania błędów sprzętowych, itp.

5.6. Badania sterownika CNC zaimplementowanego w komputerze PC

W celu zbadania poprawności działania sterownika CNC, autor zaimplementował go na trzech komputerach PC o różnej konfiguracji sprzętowej. Dodatkowo wykorzystano czwarty komputer z systemem Linux RTAI, na którym zaimplementowano opracowany przez autora moduł analizatora pakietów magistrali Ethernet Powerlink. Komputer został włączony w łańcuch komunikacyjny Ethernet Powerlink wraz ze sterownikiem CNC oraz serwonapędami

osi mechanicznych. Zadaniem tego komputera był pomiar czasu cyklu komunikacyjnego pomiędzy pakietami SoC EPL. Zestawienie konfiguracji sprzętowej wykorzystywanych komputerów przedstawia tabela 5.4. Konfiguracja komputera będącego analizatorem pakietów Ethernet Powerlink była identyczna jak komputera nr 2 z tabeli 5.4.



Rysunek 5.12. Schemat poglądowy stanowiska badawczego analizatora pakietów Ethernet Powerlink.

Tabela 5.4. Parametry sprzętowe komputerów użytych w badaniach eksperymentalnych

Parametry PC	**1. komputer**	**2. komputer**	**3. komputer**
Procesor	Core 2 Duo E7500 (2,93GHz)	Core 2 Quad Q8200 (2,33GHz)	Core i5 760 (2,80GHz)
Płyta główna	IEI IMBA-9454ISA	ASUS P5E Deluxe	ASUS P7H55
Pamięć operacyjna RAM	2GB	4GB	4GB
Karta graficzna	Matrox G550 PCIe	ATI Radeon HD 4600	ATI Radeon HD 5670
Wersja systemu Linux RTAI	3.8.1	3.8.1	3.8.1
Wersja jądra systemu Linux	2.6.32-122-rtai	2.6.32-122-rtai	2.6.32-122-rtai
Częstotliwość zegara APIC	16,667138 MHz	20,871775 MHz	8,362397 MHz

Opracowany analizator pakietów został skonfigurowany tylko do odbioru ramek EPL. Analizator dokonuje pomiaru czasu odbioru każdej z ramek EPL. Pomiar odbywa się z wykorzystaniem opracowanego modułu czasu rzeczywistego, który bazuje na funkcjach obsługi przerwań karty sieciowej Ethernet. Pomiar czasu przeprowadzono z użyciem 64-ro bitowego rejestru czasu procesora (Time Stamp Counter - TSC). Pomiar czasu z użyciem TSC jest

bardzo dokładny, gdyż licznik jest inkrementowany z częstotliwością taktowania procesora (2.33GHz). Każdy z rdzeni procesora wielordzeniowego posiada indywidualny licznik TSC. Odczyt wartości z rejestru TSC dokonywany jest z użyciem instrukcji RDTSC, której realizacja jest bardzo szybka i nie wpływa znacząco na dokonany pomiar czasu. Opracowany moduł czasu rzeczywistego do analizy pakietów przetwarzany jest wyłącznie na jednym, wyizolowanym rdzeniu procesora. Zabieg ten ma na celu wyeliminowanie wpływu procesów systemu Linux, obsługiwanych przez drugi rdzeń, na moduł pomiarowy. Analizator pakietów oblicza również w trybie on-line wartość średnią i odchylenie standardowe zmierzonych okresów czasu ramek EPL.

W przedstawionym na rysunku 5.12 stanowisku badawczym, urządzenia w sieci EPL połączone są w topologii liniowej. Używane serwonapędy mają zintegrowane dwa porty Ethernet Powerlink. Opóźnienia wprowadzone przez te dwa porty, są z reguły znacznie mniejsze, niż całkowity czas trwania cyklu komunikacyjnego.

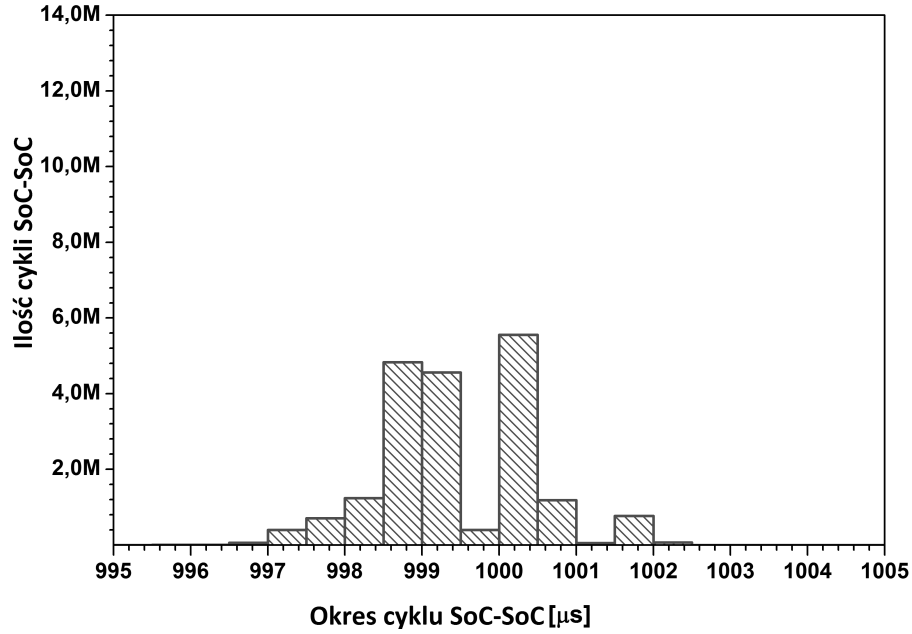
W przeprowadzonych przez autora badaniach, mierzono rozrzut (jitter) cyklu komunikacyjnego EPL. Wartość jitteru wyznaczano w wyniku różnicy pomiarów czasu pomiędzy kolejnymi ramkami wyznaczającymi początek cyklu EPL (ramki SoC). Odebranie ramki SoC powoduje synchronizacje wewnętrznych pętli regulacji każdego z seronapędów (wykonanie zadanego przemieszczenia). Dlatego ważne jest by odstępy czasu pomiędzy kolejnymi ramkami SoC nie różniły się miedzy sobą. Jitter ramek SoC jest bardzo ważnym parametrem dla systemu sterowania serwonapędu, gdyż pośrednio wpływa na błąd nadążania danej osi.

Głównym czynnikiem wpływającym na pojawienie się jitteru, w cyklu komunikacyjnym ramek SoC, są operacje obliczeniowe związane z realizacją aplikacji LinuxCNC. W celu zbadania wpływu obliczeń na jitter komunikacyjny ramek SoC, autor przeprowadził badania na każdym z trzech testowych komputerów. Dla każdej implementacji sterownika CNC została zinterpretowana, a następnie wykonana trajektoria ruchu, zdefiniowana w języku „G-code". Dla zadanej trajektorii ruchu przeprowadzono pomiary czasu, pomiędzy 20-ma milionami kolejnych ramek SoC, dla każdego z komputerów testowych. Okres cyklu komunikacyjnego został ustawiony na 1ms. Wyniki eksperymentu przedstawiono w tabeli 5.5 oraz w formie histogramów na rysunkach 5.13, 5.14 i 5.15.

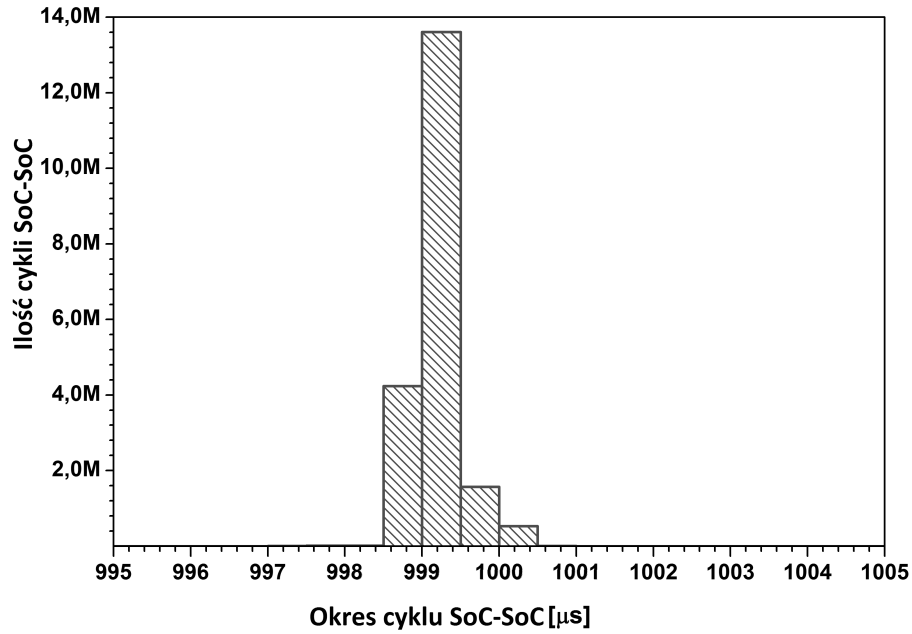
Tabela 5.5. Parametry czasowe dla 20-tu milionów cykli komunikacyjnych SoC

	1. komputer	**2. komputer**	**3. komputer**
wartość średnia [μs]	999,4565	999,2829	997,2869
maksymalny rozrzut [μs]	11,7918	9,9695	11,5113
odchylenie standardowe [μs]	0,9721	0,3433	0,4540

Maksymalny jitter cyklu komunikacji SoC, dotyczący wszystkich trzech komputerów, nie przekracza 12 μs, a wartości odchylenia standardowego były poniżej 1μs. Wszystkie wartości

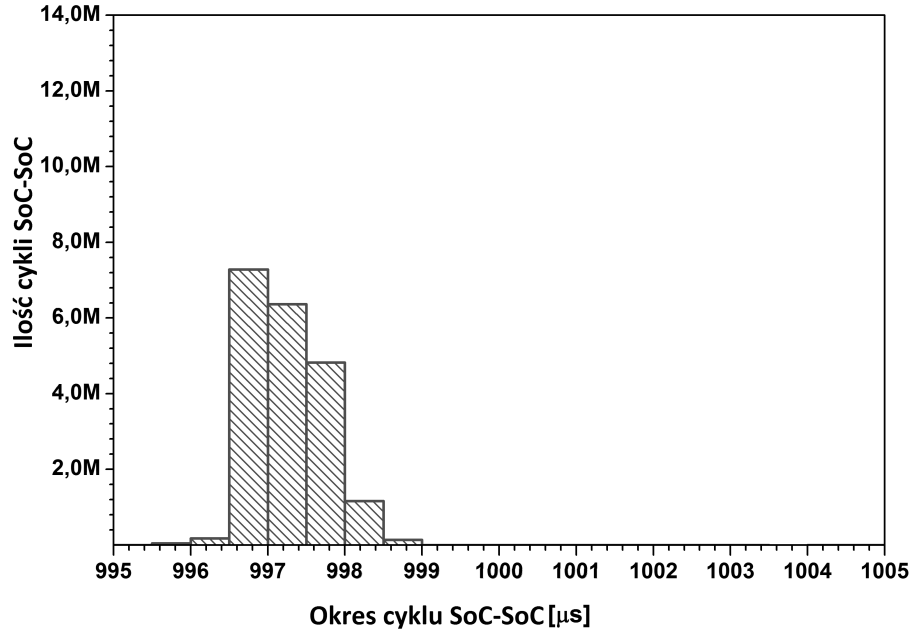


Rysunek 5.13. Histogram czasu pomiędzy ramkami SoC (20 mln. cykli) dla komputer nr 1.



Rysunek 5.14. Histogram czasu pomiędzy ramkami SoC (20 mln. cykli) dla komputer nr 2.

średnie okresów występowania ramek SoC są nieco niższe od wartości 1ms. Wynika to z ograniczonej rozdzielczości zegarów APIC, które wyznaczają początek cyklu ramek EPL. Wyższa częstotliwość pracy APIC umożliwiłaby lepszą rozdzielczość zegarów APIC oraz zmniejszyłaby fluktuacje rozpoczęcia cyklu EPL. Pomimo tego niewielkie różnice w pomiarze czasu cykli nie mają wpływu na prawidłową pracę sterownika CNC oraz generowane w nim przyrosty trajektorii zadanej.



Rysunek 5.15. Histogram czasu pomiędzy ramkami SoC (20 mln. cykli) dla komputer nr 3.

Wartości maksymalnego jitter'u oraz odchylenia standardowego są różne w każdym z badanych komputerów. Spowodowane jest to różnymi architekturami chipsetu płyty głównej oraz częstotliwością ich pracy. Komputer nr 2, który ma najniższy jitter, wykorzystuje układ chipsetu wysokiej klasy (X48) o wysokiej częstotliwości taktowania. Natomiast komputer nr 1 (najgorszy jitter) wykorzystuje najstarszy układ chipsetu (945G), który taktowany jest zegarem o niższej częstotliwości, co wprowadza opóźnienia w jego obsłudze. Pomimo, że komputer nr 3 wykorzystuje najnowszy układ chipsetu (H55) spośród badanych komputerów, to jakość pracy tego układu jest gorsza niż układu chipsetu komputera nr 2 (wyższa klasa układu). Nowsze oraz wysokiej klasy układy chipsetu działają przy wyższych częstotliwościach zegara taktującego, co przekłada się na mniejsze opóźnienia oraz mniejsze wartości jitter'u w pracy modułów czasu rzeczywistego.

Jeżeli w osi mechanicznej wirnik silnika obraca się z maksymalną prędkością równą 3000obr/min, zaś zastosowana przekładnia śrubowa toczna ma skok śruby równy 10mm/obr, to prędkość posuwu osi mechanicznej wynosi 30m/min. W takich warunkach jitter równy $12\mu s$ spowoduje błąd nadążania równy wartości $6\mu m$. Przy niższych prędkościach pracy maszyny CNC (ok. 5m/min) błąd nadążanie nie będzie przekraczać wartości $1\mu m$. W przypadku maszyn, gdzie wymagana tolerancja błędu nadążania nie powinna przekraczać 0,01mm, opóźnienia w komunikacji spowodowane jitter'em nie maja tak dużego znaczenia.

W rozdziale omówiono stanowisko badawcze z maszyną CNC. W maszynie CNC zostały wykorzystane dwa komercyjnie dostępne serwonapędy Baldor Microflex e100. Oprogramowanie sterownika wykorzystuje otwarty system czasu rzeczywistego Linux RTAI oraz aplikację sterującą LinuxCNC. Komputer PC wyposażony jest w kartę sieciową, która

wykorzystywana jest do komunikacji z serwonapędami przy użyciu magistrali komunikacyjnej Ethernet Powerlink.

Autor dokonał:

— Opracowania i implementacji sterownika CNC w komputerze PC.
— Implementacji stosu protokołu EPL w systemie operacyjnym Linux RTAI z jego gruntowną modyfikacją, w celu umożliwienia jego poprawnego działania w tym systemie.
— Rozbudowy stosu EPL o profil komunikacji CiA402 oraz jego integrację z oprogramowaniem sterującym LinuxCNC.
— Implementacji opracowanego bloku Kompensatora Trajektorii Zadanej w sterowniku CNC zgodnie z wymaganiami stawianymi przez system Linux RTAI.
— Modyfikacji w stosie EPL, konfiguracji systemu Linux RTAI oraz BIOS'u (wraz z konfiguracją płyty głównej komputerów PC), w celu zapewnienia stabilnej i precyzyjnej pracy sterownika CNC.

Badania sterownika CNC bazującego na komputerze PC wykazały stabilność pracy i poprawną komunikacje z wykorzystanymi serwonapędami, przy użyciu magistrali Ethernet Powerlink. Pomiary jitter'u przeprowadzone w trakcie działania programu sterującego LinuxCNC wykazały, że maksymalny jitter okresu cyklu komunikacyjnego Ethernet Powerlink jest mniejszy niż $12\mu s$, a wartość odchylenia standardowego tej wielkości jest mniejsza od $1\mu s$, dla 3 różnych konfiguracji sprzętowych komputerów PC. Otrzymane jitter'y w minimalny stopniu wpływają na wartości błędów nadążania.

6. Wyniki badań algorytmu minimalizacji błędów nadążania układu posuwu osi mechanicznej

Badania dotyczyły sprawdzenia poprawności działania dwóch bloków Kompensatora Trajektorii Zadanej, zintegrowanych z układem sterownika CNC maszyny dwuosiowej. Przeprowadzono badania neuronowego modelu osi mechanicznej oraz bloku optymalizującego dla każdego z Kompensatorów.

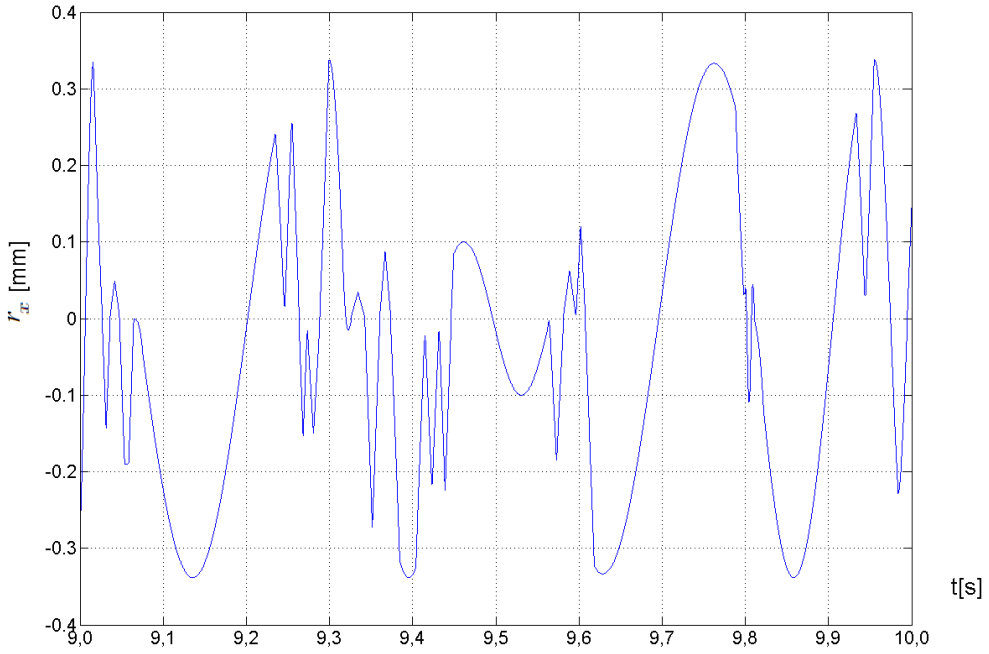
W tabeli 6.1 przedstawiono parametry neuronowych modeli osi mechanicznych dla każdej z osi. Zaimplementowane sieci NARX posiadały jedną warstwę ukrytą, w której znajdowało się 12 neuronów o sigmoidalnej funkcji aktywacji. Warunki początkowe wejść sieci (bloków opóźniających) ustalane były na podstawie wartości rzeczywistych, zapamiętanych z poprzednich iteracji algorytmu.

Tabela 6.1. Parametry sieci neuronowych NARX

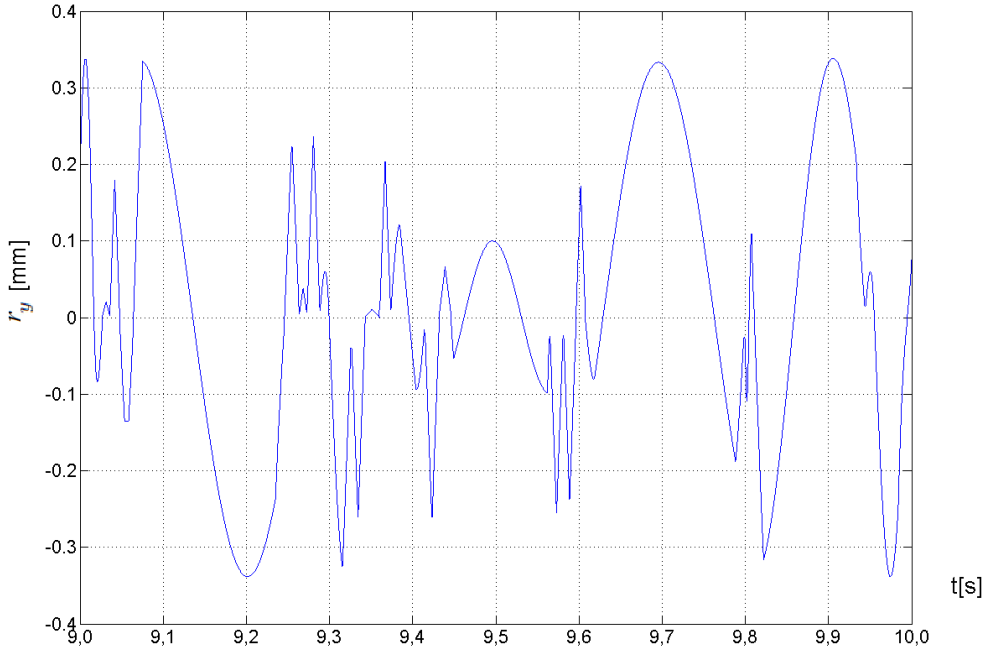
Parametry sieć NARX	**Sieć NARX dla osi X**	**Sieć NARX dla osi Y**
Ilość warstw ukrytych	1	1
Ilość neuronów w warstwie ukrytej	12	12
Funkcja aktywacji w warstwie ukrytej	funkcja sigmoidalna	funkcja sigmoidalna
Ilość wejść r	4	4
Ilość wejść ϵ	4	4

Przeprowadzono testy predykcji błędów nadążania w poszczególnych osiach maszyny wieloosiowej, w których wykorzystano osie X i Y maszyny CNC stanowiska badawczego. Na rysunkach 6.1 i 6.2 przedstawiono fragmenty przebiegów przyrostów położenia zadanego r_x oraz r_y zrealizowane w osiach mechanicznych X i Y. Przebiegi błędów nadążania, będące odpowiedzią układu na zadane przyrosty położenia zadanego r_x oraz r_y, przedstawiono na rysunkach 6.3 i 6.4 oznaczając je kolorem niebieskim. Kolorem czerwonym oznaczono przebiegi błędu nadążania $\hat{\epsilon}$ uzyskane na wyjściu sieci NARX odpowiednio dla osi X i Y. Błędy predykcji wartości ϵ z wykresów 6.3 i 6.4, będące różnicą przebiegów rzeczywistego i predykowanego błędu nadążania, przedstawiono odpowiednio na rysunkach 6.5 i 6.6. Na podstawie otrzymanych wyników można stwierdzić, że sieci neuronowe NARX bardzo dobrze

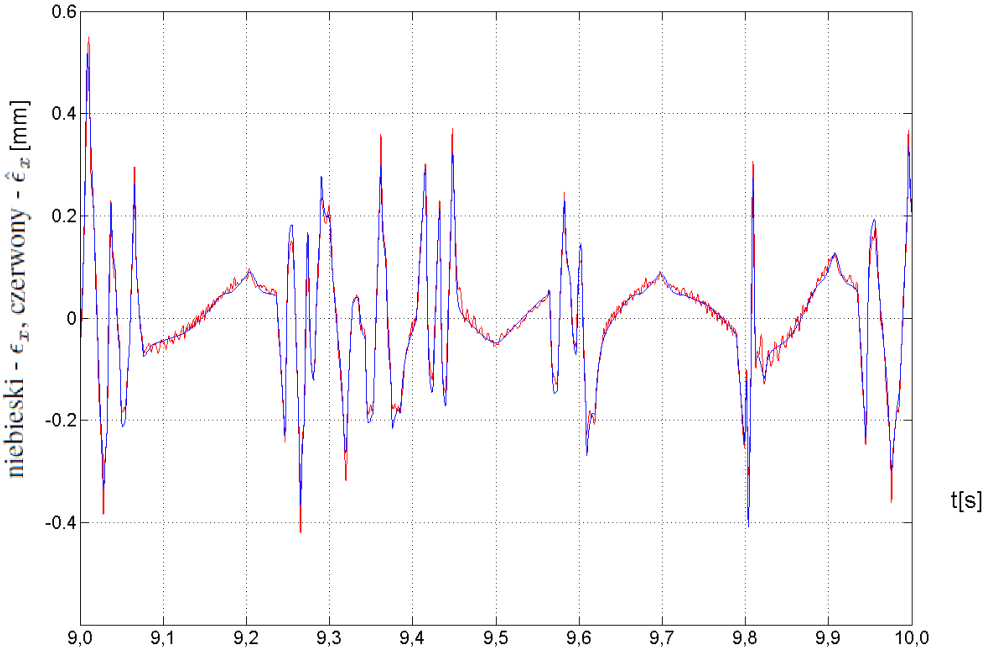
odwzorowują błędy nadążania w osiach X i Y. Wartości błędów estymowanych dla osi X i Y stanowią od 1% do 10% wartości samych wartości estymowanych.



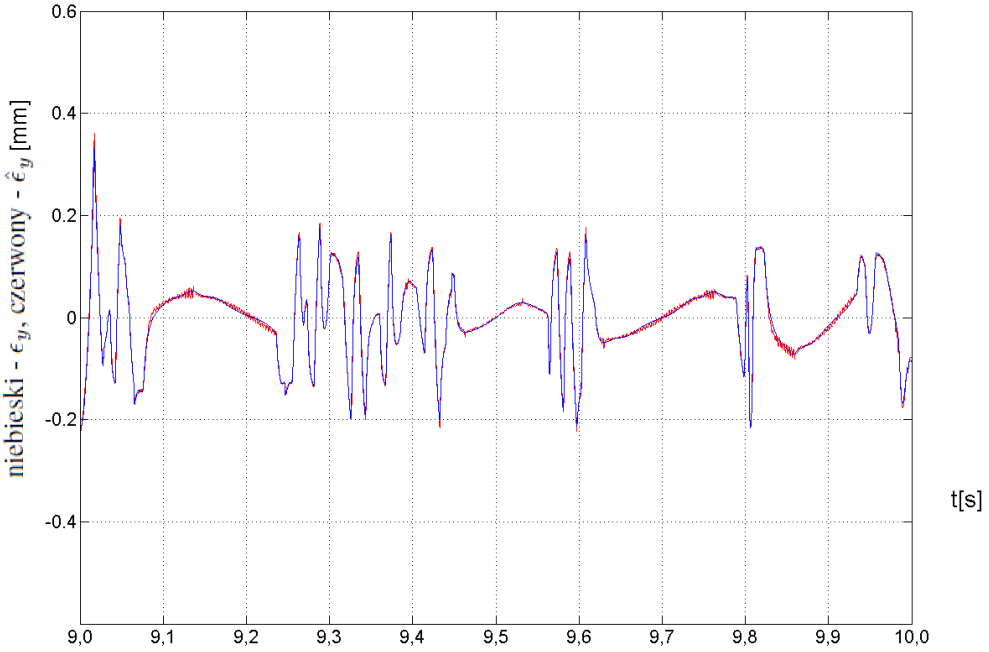
Rysunek 6.1. Wykres przyrostów położenia zadanego r_x zrealizowanych przez oś X



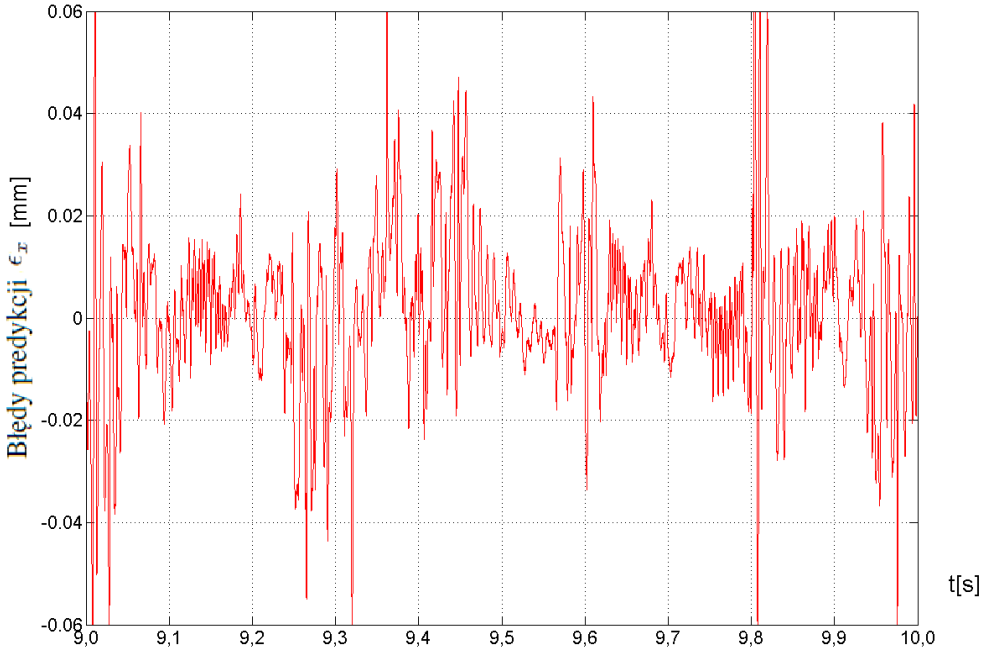
Rysunek 6.2. Wykres przyrostów położenia zadanego r_y zrealizowanych przez oś Y



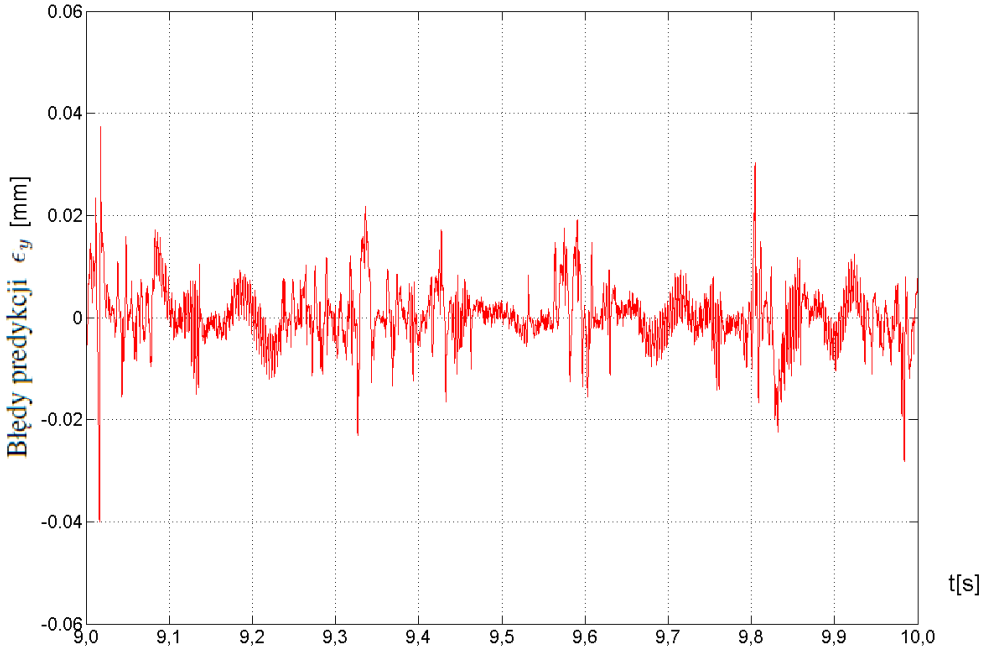
Rysunek 6.3. Błędy nadążania wynikające z realizacji przyrostów położenia zadanego r_x (kolor czerwony) oraz ich predykcja obliczona z sieci NARX (kolor niebieski)



Rysunek 6.4. Błędy nadążania wynikające z realizacji przyrostów położenia zadanego r_y (kolor czerwony) oraz ich predykcja obliczona z sieci NARX (kolor niebieski)



Rysunek 6.5. Błędy predykcji ϵ sieci NARX w osi X

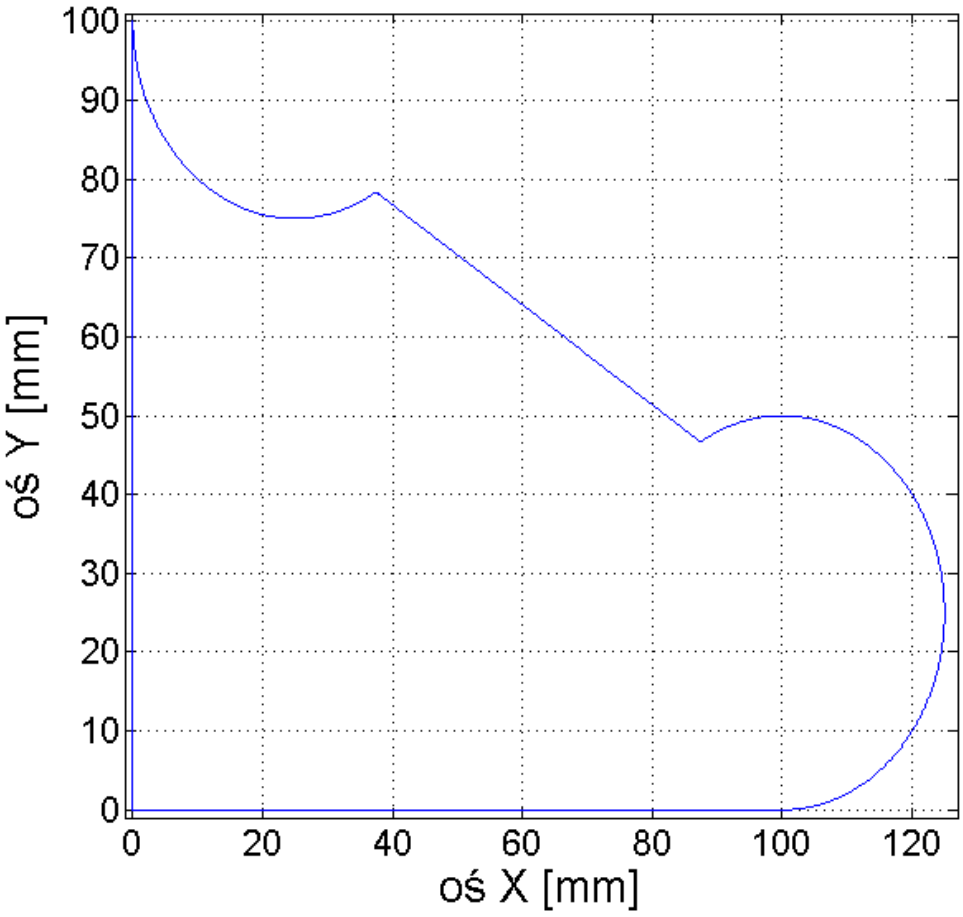


Rysunek 6.6. Błędy predykcji ϵ sieci NARX w osi Y

Badania poprawności działania bloku optymalizacji, zostały przeprowadzone w ramach kompleksowych badań Kompensatorów Trajektorii Zadanej osi X i Y. Funkcja celu podlegająca optymalizacji jest określona wzorem 4.3. Funkcja celu posiada ograniczony zbiór rozwiązań wyznaczony przez nierówności 4.2. Parametry zastosowanego algorytmu optymalizacji rojem cząstek zostały przedstawione w tabeli 6.2. Badania doświadczalne minimalizacji błędów nadążania układu posuwu osi mechanicznej, przeprowadzone zostały z wykorzystaniem trajektorii zadanej w formie przedstawionej na rysunku 6.7. Błędy nadążania niemodyfikowanej trajektorii zadanej oraz błędy nadążania zmodyfikowanej trajektorii zadanej dla osi X przedstawiono na rysunku 6.8. Błędy nadążania niemodyfikowanej trajektorii zadanej oraz błędy nadążania zmodyfikowanej trajektorii zadanej dla osi Y przedstawiono na rysunku 6.9.

Tabela 6.2. Parametry algorytmu optymalizacji rojem cząstek

Parametry algorytmu PSO	**Algorytm PSO dla osi X**	**Algorytm PSO dla osi Y**
Ilość cząstek	10	10
Horyzont predykcji N	5	5
Ilość iteracji i	100	100
Wartość współczynnika bezwładności ω	0,7	0,7
Wartości współczynników przyspieszeń ϕ_1, ϕ_2	2,0	2,0
Maksymalna wartość prędkości	300 mm/s	300 mm/s
Maksymalna wartość przyspieszenia	2500 mm/s^2	2500 mm/s^2
Ograniczenia przyrostu położenia (przeskalowana prędkość)	$\pm$0,1mm aktualnego przyrostu	$\pm$0,1mm aktualnego przyrostu
Ograniczenie błędu nadążania	$\pm$0,06mm aktualnego przyrostu	$\pm$0,06mm aktualnego przyrostu



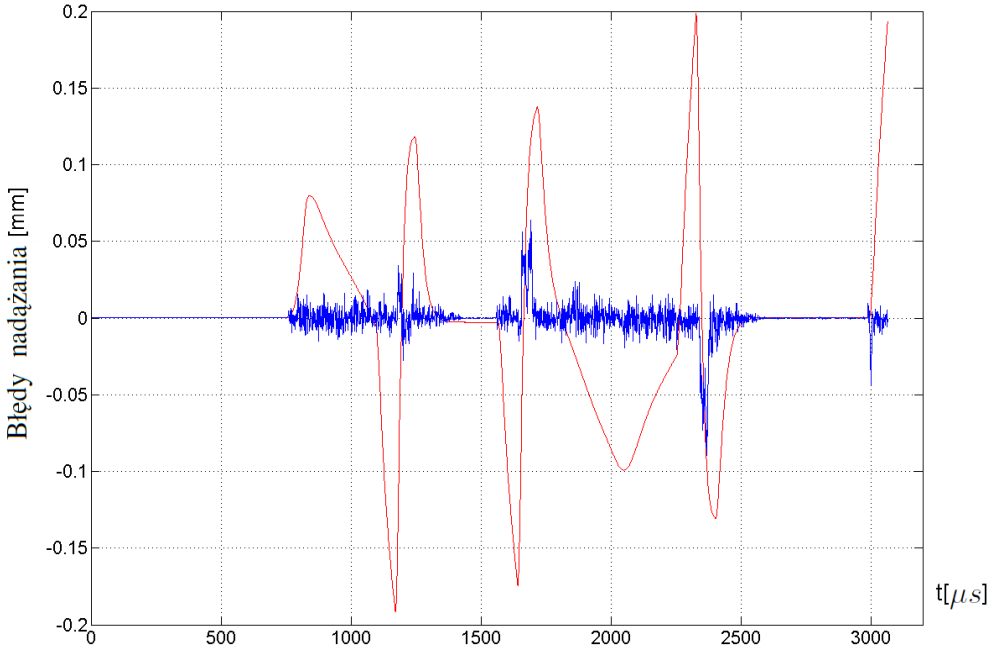
Rysunek 6.7. Trajektoria zadana w płaszczyźnie X-Y

Na podstawie rysunku 6.8 i 6.9 wyliczono błędy średniokwadratowe błędów nadążania, dla dwóch realizacji trajektorii zadanej, dla obydwu osi mechanicznych. Wyniki zestawiono w tabeli 6.3. Wartości MSE dla trajektorii zadanej zmodyfikowanej przez Kompensatory Trajektorii Zadanej, są wielokrotnie mniejsze w stosunku do błędów średniokwadratowych trajektorii zadanej bez modyfikacji.

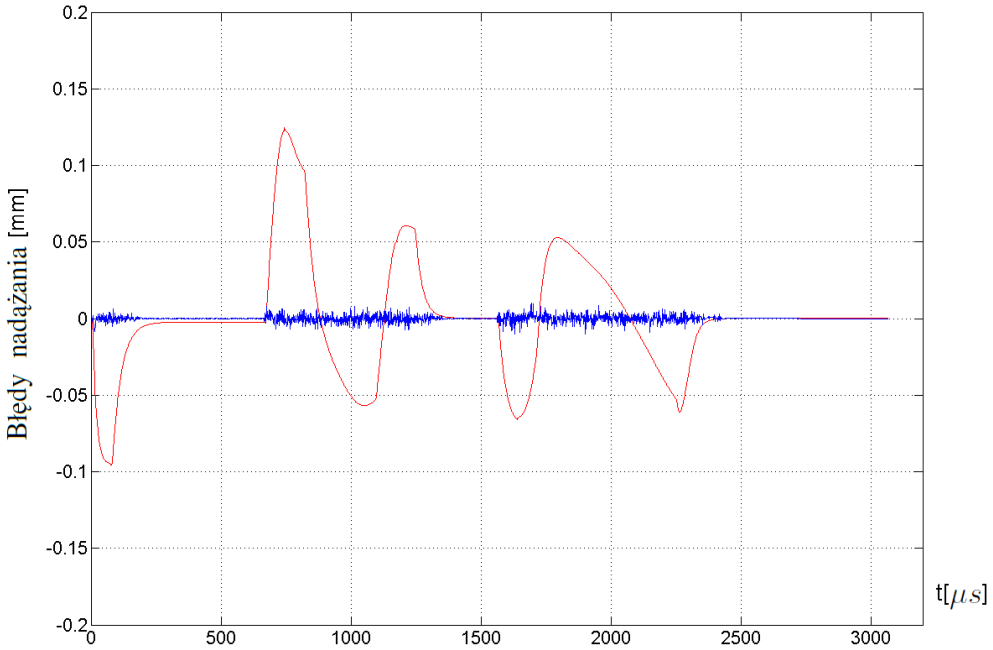
Tabela 6.3. MSE błędów nadążania dla dwóch realizacji trajektorii zadanej (obydwie osie mechaniczne)

MSE błędów nadążania	**dla osi X**	**dla osi Y**
niezmodyfikowanej trajektorii ruchu	$0,0033$	$0,0014$
zmodyfikowanej trajektorii ruchu	$7,6401 \times 10^{-5}$	$3,7540 \times 10^{-5}$

Na podstawie przedstawionych wyników można stwierdzić, że opracowana metoda optymalizacji przyrostów położenia zadanego w układach sterowania maszyn CNC, pozwala na minimalizację błędów nadążania w układach posuwu osi mechanicznych, uwzględniając jego maksymalną wartość. Rzeczywiste błędy nadążania poszczególnych osi nie przekraczają określonych wartości maksymalnych pomimo, że predykcja tych wartości obarczona jest



Rysunek 6.8. Błędy nadążania niemodyfikowanej trajektorii zadanej (kolor czerwony) oraz błędy nadążania zmodyfikowanej trajektorii zadanej (kolor niebieski) dla osi X



Rysunek 6.9. Błędy nadążania niemodyfikowanej trajektorii zadanej (kolor czerwony) oraz błędy nadążania zmodyfikowanej trajektorii zadanej (kolor niebieski) dla osi Y

błędami predykcji. Wynika to z niedoskonałości modelu oraz ograniczonej powtarzalności działania maszyny CNC. Bardzo trudne jest opracowanie modeli osi mechanicznych, pozwalających na bezbłędną predykcję błędów nadążania, w tak skomplikowanym układzie mechanicznym jakim jest maszyna CNC. Trudne jest również zbudowanie maszyny CNC, która za każdym razem przy odtwarzaniu zadanego toru ruchu, wykonywałaby go dokładnie tak samo.

Opracowany algorytm umożliwia uzyskanie jeszcze mniejszych błędów nadążania niż te, które otrzymano w wyniku przeprowadzonych badań. Ograniczeniem była konieczność działania algorytmu w trybie on-line. Zwiększenie ilości iteracji algorytmu optymalizacyjnego, w celu zmniejszenia błędów nadążania, pociągało za sobą ryzyko wydłużenia czasu wyznaczania kolejnych przyrostów zadanych do serwonapędów, poza przewidziany okres 1ms. W wyniku przeprowadzenia licznych testów, parametry algorytmu PSO dobrano w taki sposób, by uzyskać możliwie małe błędy nadążania przy zagwarantowaniu nie przekraczania założonego reżimu czasowego.

Dzięki zastosowaniu Kompensatora Trajektorii Zadanej możliwa jest realizacja toru ruchu, w którym błędy nadążania w poszczególnych osiach są minimalizowane. Prowadzi to do poprawienia dokładności wykonania elementów obrabianych w procesie produkcji.

7. Podsumowanie

W zaprezentowanej rozprawie doktorskiej omówiono zagadnienie minimalizacji błędów nadążania układu posuwu osi mechanicznych. Przedstawiono strukturę układu sterowania CNC wraz z głównymi zagadnieniami, odnoszącymi się do problemu poprawnego odwzorowania zadanego toru ruchu. Opracowano neuronowe modele układów posuwu osi mechanicznych, które dokonują predykcji błędów nadążania. Predykcja wartości błędów nadążania jest niezbędna w procesie optymalizacji przyrostów posuwu. Opracowano algorytm optymalizacji przyrostów posuwu, wykorzystujący algorytm optymalizacji rojem cząstek (PSO), uwzględniający ograniczenia maksymalnych wartości przyrostów posuwu oraz błędów nadążania. Zbudowano układ sterowania maszyn CNC wykorzystujący komputer PC z systemem czasu rzeczywistego, w którym zaimplementowano opracowany algorytm w postaci programu komputerowego.

Wyniki przeprowadzonych badań doświadczalnych pozwalają na wysunięcie następujących wniosków:

— Opracowane neuronowe modele układów posuwu osi mechanicznych, skutecznie modelują dynamikę układu posuwu osi mechanicznych maszyny CNC, pozwalając na przewidzenie wartości błędów nadążania z małymi błędami predykcji.
— Algorytm optymalizacyjny umożliwia minimalizację wartości błędów nadążania w poszczególnych osiach, co pozwala na zmniejszenie błędów odtwarzania zadanej trajektorii ruchu.
— Uzyskane przebiegi błędów nadążania różnią się od przebiegów uzyskanych na wyjściu sieci neuronowej, ze względu na niedoskonałe modelowanie rzeczywistego obiektu przez sieć NARX. Ponadto na rozbieżność pomiędzy modelem a rzeczywistością wpływ ma ograniczona powtarzalność maszyny CNC. Pomimo niedoskonałości modelu neuronowego, rzeczywiste błędy nadążania poszczególnych osi nie przekraczają określonych wartości maksymalnych.

Autor za najcenniejsze osiągnięcia własne uważa:

— Opracowanie neuronowych modeli osi mechanicznych, które umożliwiają przewidywanie błędów nadążania.
— Opracowanie innowacyjnego algorytmu optymalizacji przyrostów wartości zadanej toru ruchu, wykorzystującego opracowane neuronowe modele osi mechanicznych, w celu ograniczenia maksymalnej wartości błędów nadążania.

— Opracowanie i budowę układu sterowania CNC, wykorzystującego komputer PC z systemem czasu rzeczywistego, w którym zaimplementowano opracowany algorytm.
— Eksperymentalną weryfikację działania opracowanego algorytmu optymalizacji wartości zadanej toru ruchu w układzie sterowania CNC.

Zdaniem autora najważniejszym elementem innowacyjnym, w opracowanym algorytmie, jest uwzględnienie błędów nadążania poszczególnych osi mechanicznych, w procesie optymalizacji przyrostów wartości zadanej toru ruchu, dzięki zastosowaniu sztucznych sieci neuronowych do predykcji błędów nadążania. Posiadana przez autora wiedza z zakresu optymalizacji toru ruchu maszyn CNC pozwala stwierdzić, że zaproponowana metoda minimalizacji błędów nadążania nie została przedstawiona w dostępnej literaturze. Innowacyjnym wkładem autora w dziedzinę optymalizacji przyrostów wartości zadanej toru ruchu maszyn wieloosiowych, jest także zastosowanie sztucznych sieci neuronowych do predykcji błędów nadążania wraz z opracowaną metodologią uczenia sieci w zamkniętej pętli układu sterowania. Całość opracowanego algorytmu została zaimplementowana w języku „ANSI C", dzięki czemu możliwe jest jego zastosowanie na rożnych platformach sprzętowych, bez konieczności stosowania specjalistycznego oprogramowania.

Zbudowany przez autora układ sterowania maszyn CNC, wykorzystując komputer PC i system operacyjny czasu rzeczywistego, ma duże znaczenie praktyczne. Autor zaimplementował stos protokołu komunikacyjnego Ethernet Powerlink do komunikacji z serwonapędami w środowisku czasu rzeczywistego. Sterowanie różnego rodzaju maszynami, wyposażonymi w komercyjne napędy różnych producentów, jest możliwe z wykorzystaniem opracowanego układu sterowania.

Badania zastosowanego algorytmu wykazały, że rzeczywiste błędy nadążania poszczególnych osi nie przekraczają ograniczeń, pomimo że predykcja wartości błędów nadążania przez neuronowe modele osi cechuje pewna niedokładność. Wynika to z niedoskonałości modelu oraz ograniczonej powtarzalności działania maszyny CNC.

Konieczność wyznaczania kolejnych zmodyfikowanych przyrostów położenia on-line, narzucała istotne ograniczenia co do czasu obliczeń. W wyniku przeprowadzonych prób parametry algorytmu PSO zostały dobrane tak, aby zapewnić możliwie niskie wartości błędów nadążania bez przekraczania założonego czasu obliczeń (1ms).

Rozwiązanie powyższego problemu stanowi, według autora, interesujący kierunek dalszych badań. W celu skrócenia czasu generacji kolejnych położeń zadanych, konieczne będzie skrócenie czasu obliczeń. Osiągnąć to można poprzez zrównoleglenie obliczeń na wielu rdzeniach procesora lub przy wykorzystaniu dodatkowych zasobów obliczeniowych (np. wykorzystanie mocy obliczeniowej układów graficznych). Zastosowanie algorytmu PSO oraz sieci neuronowych istotnie ułatwia proces zrównoleglenia obliczeń. W celu zmniejszenia

rozbieżności pomiędzy rzeczywistym a przewidywanym błędem nadążania, konieczne będzie udoskonalenie modelu osi mechanicznej.

Opracowany algorytm nie uwzględnia modelowania błędów powstałych w trakcie pracy narzędzia wykonawczego maszyny wieloosiowej (np. obróbki skrawaniem). Z tego względu zakres jego stosowania ogranicza się do maszyn typu wycinarki laserowe, plazmowe itp. Zastosowanie algorytmu do maszyn, w których obciążenie wynikające z sił skrawania ma istotny wpływ na błędy nadążania, wymagałoby opracowania modelu sił skrawania co stanowi interesujący kierunek dalszych badań.

Zdaniem autora opracowany algorytm mógłby znaleźć zastosowanie w większości układów sterowania maszyn wieloosiowych. Prace nad wdrożeniem opracowanego przez autora systemu sterowania, wraz z opracowaną metodą optymalizacji przyrostów wartości zadanej toru ruchu, będą kontynuowane we współpracy z Przemysłowym Instytutem Automatyki i Pomiarów Oddział Badawczo-Rozwojowy Urządzeń Sterowania Napędów w Toruniu (PIAP-OBRUSN Toruń).

Bibliografia

[1] ISO/IEC 7498-1: 1994. Information technology–open systems interconnection–basic reference model: The basic model. Iso, International Organization for Standardization, Geneva, Switzerland, 1994.

[2] IEC 61800-7-201. Adjustable speed electrical power drive systems – part 7-201: Generic interface and use of profiles for power drive systems – profile type 1 specification. IEC ISBN 2–8318–9375–5, International Electrotechnical Commission, Geneva, Switzerland, 2007.

[3] ISO 6983-1:2009. Numerical control of machines – program format and definition of address words – part 1: Data format for positioning, line motion and contouring control systems. Iso, International Organization for Standardization, Geneva, Switzerland, 2009.

[4] Dzieliński A. Neural network-based narx models in non-linear adaptive control. *International Journal of Applied Mathematics and Computer Science*, 12(2):235–240, 2002.

[5] L Abdullah, Z Jamaludin, MN Maslan, J Jamaludin, I Halim, NA Rafan, and TH Chiew. Assessment on tracking performance of cascade p/pi, npid and ncasff controller for precise positioning of xy table ballscrew drive system. *Procedia CIRP*, 26:212–216, 2015.

[6] Y Altintas and MR Khoshdarregi. Contour error control of cnc machine tools with vibration avoidance. *CIRP annals-manufacturing technology*, 61(1):335–338, 2012.

[7] Karlo Apro. *Secrets of 5-axis machining*. Industrial Press Inc., 2008.

[8] B. Armstrong-Hélouvry, P. Dupont, and C.C. De Wit. A survey of models, analysis tools and compensation methods for the control of machines with friction. *Automatica*, 30(7):1083–1138, 1994.

[9] Rafael V Aroca, Glauco Caurin, and Sao Carlos-SP-Brasil. A real time operating systems (rtos) comparison. In *XXIX Congresso da Sociedade Brasileira de Computação*, 2009.

[10] Karl Johan Åström and Tore Hägglund. *Advanced PID control*. ISA-The Instrumentation, Systems, and Automation Society; Research Triangle Park, NC 27709, 2006.

[11] VV Athani. *Stepper motors: fundamentals, applications and design*. New Age International, 1997.

[12] Xavier Beudaert, Sylvain Lavernhe, and Christophe Tournier. Feedrate interpolation with axis jerk constraints on 5-axis nurbs and g1 tool path. *International Journal of Machine Tools and Manufacture*, 57:73–82, 2012.

[13] Paolo Bosetti and Enrico Bertolazzi. Feed-rate and trajectory optimization for cnc machine tools. *Robotics and computer-integrated manufacturing*, 30(6):667–677, 2014.

[14] Wei-Der Chang. A multi-crossover genetic approach to multivariable pid controllers tuning. *Expert Systems with Applications*, 33(3):620–626, 2007.

[15] Yih-Fang Chang, Truong-Giang Nguyen, and Chia-Pin Wang. Design and implementation of look-ahead linear jerk filter for a computerized numerical controlled machine. *Control Engineering Practice*, 18(12):1399–1405, 2010.

[16] Amitava Chatterjee and Patrick Siarry. Nonlinear inertia weight variation for dynamic adaptation in particle swarm optimization. *Computers & Operations Research*, 33(3):859–871, 2006.

[17] Chin-Sheng Chen and Li-Yeh Chen. Cross-coupling position command shaping control in a multi-axis motion system. *Mechatronics*, 21(3):625–632, 2011.

[18] Shang-Liang Chen and Tsung-Hsien Hsieh. Repetitive control design and implementation for linear motor machine tool. *International Journal of Machine Tools and Manufacture*, 47(12):1807–1816, 2007.

[19] Jih-Hua Chin, Yuan-Ming Cheng, and Jin-Huei Lin. Improving contour accuracy by fuzzy-logic enhanced cross-coupled precompensation method. *Robotics and Computer-Integrated Manufacturing*, 20(1):65–76, 2004.

[20] Jih-Hua Chin and Tsung-Ching Lin. Cross-coupled precompensation method for the contouring accuracy of computer numerically controlled machine tools. *International Journal of Machine Tools and Manufacture*, 37(7):947–967, 1997.

[21] Maurice Clerc. *Particle swarm optimization*, volume 93. John Wiley & Sons, 2010.

[22] Maurice Clerc and James Kennedy. The particle swarm-explosion, stability, and convergence in a multidimensional complex space. *Evolutionary Computation, IEEE Transactions on*, 6(1):58–73, 2002.

[23] C.A. Coello Coello. Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: a survey of the state of the art. *Computer methods in applied mechanics and engineering*, 191(11):1245–1287, 2002.

[24] International Electrotechnical Commission et al. *Industrial Communication Networks-Profiles-Part 2, Additional Fieldbus Profiles for Real-time Networks Based on ISO*. International Electrotechnical Commission, 2007.

[25] Jeremy R Conway, Charlie A Ernesto, Rida T Farouki, and Mei Zhang. Performance analysis of cross-coupled controllers for cnc machines based upon precise real-time contour error measurement. *International Journal of Machine Tools and Manufacture*, 52(1):30–39, 2012.

[26] Dynamic Research Corporation. *Techniques for Digitizing Rotary and Linear Motion: Encoder Division*. Drc, 1992.

[27] M Correa, C Bielza, and J Pamies-Teixeira. Comparison of bayesian networks and artificial neural networks for quality detection in a machining process. *Expert systems with applications*, 36(3):7270–7279, 2009.

[28] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[29] Hojat Dehnavi, Mohammad Reza Movahhedy, Ahmad Naebi, and Soleyman Pasban. Prediction of the effect of heat generation in ballscrew on the accuracy of cnc milling machine. In *Computer Modelling and Simulation (UKSim), 2012 UKSim 14th International Conference on*, pages 278–282. IEEE, 2012.

[30] Jingchuan Dong, Taiyong Wang, Bo Li, and Yanyu Ding. Smooth feedrate planning for continuous short line tool path with contour error constraint. *International Journal of Machine Tools and Manufacture*, 76:1–12, 2014.

[31] Lorenzo Dozio and Paolo Mantegazza. Linux real time application interface (rtai) in low cost high performance motion control. *Motion Control*, 2003, 2003.

[32] R. Eberhart and J. Kennedy. A new optimizer using particle swarm theory. *Proceedings of the sixth international symposium on micro machine and human science*, 1:39–43, 1995.

[33] Russ C Eberhart and Yuhui Shi. Comparing inertia weights and constriction factors in particle swarm optimization. In *Evolutionary Computation, 2000. Proceedings of the 2000 Congress on*, volume 1, pages 84–88. IEEE, 2000.

[34] M Ebrahimi and R Whalley. Analysis, modeling and simulation of stiffness in machine tool drives. *Computers & industrial engineering*, 38(1):93–105, 2000.

[35] Kaan Erkorkmaz and Yusuf Altintas. High speed cnc system design. part ii: modeling and identification of feed drives. *International Journal of Machine Tools and Manufacture*, 41(10):1487–1509, 2001.

[36] Krystian Erwinski, Marcin Paprocki, Lech Grzesiak, Kazimierz Karwowski, and Andrzej Wawrzak. Otwarty system sterowania maszynami cnc bazujący na komputerze pc z protokołem ethernet powerlink. In *X Konferencja Naukowa Sterowanie w Energoelektronice i Napędzie Elektrycznym SENE 2011 Łódź*, 2011.

[37] Krystian Erwinski, Marcin Paprocki, Lech M Grzesiak, Kazimierz Karwowski, and Andrzej Wawrzak. Application of ethernet powerlink for communication in a linux rtai open cnc system. *Industrial Electronics, IEEE Transactions on*, 60(2):628–636, 2013.

[38] K. Erwiński. *Metoda doboru prędkości posuwu w układach sterowanianumerycznego maszyn wieloosiowych z wykorzystaniem algorytmów sterowania predykcyjnego oraz sztucznych sieci neuronowych*. PhD thesis, Politechnika Warszawska, 2014.

[39] W. Fan, X.S. Gao, C.H. Lee, K. Zhang, and Q. Zhang. Time-optimal interpolation for five-axis cnc machining along parametric tool path based on linear programming. *The International Journal of Advanced Manufacturing Technology*, 69(5-8):1373–1388, 2013.

[40] Wei Fan, Chen-Han Lee, and Ji-Hong Chen. A realtime curvature-smooth interpolation scheme and motion planning for cnc machining of short line segments. *International Journal of Machine Tools and Manufacture*, -(0):–, 2015.

[41] HUO FENG. *Controlling Contour Errors in CNC Machines*. PhD thesis, National University of Singapore, 2012.

[42] D.F. Foresee and M.T. Hagan. Gauss-newton approximation to bayesian learning. In *Neural Networks, 1997., International Conference on*, volume 3, pages 1930–1935. IEEE, 1997.

[43] Jian-Xin Guo, Ke Zhang, Qiang Zhang, and Xiao-Shan Gao. Efficient time-optimal feedrate planning under dynamic constraints for a high-order cnc servo system. *Computer-Aided Design*, 45(12):1538–1546, 2013.

[44] Jerzy Honczarenko. *Obrabiarki sterowane numerycznie*. Wydawnictwa Naukowo-Techniczne, 2008.

[45] Bill G Horne and C Lee Giles. An experimental comparison of recurrent neural networks. *Advances in neural information processing systems*, pages 697–704, 1995.

[46] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.

[47] Feng Huo and Aun-Neow Poo. Improving contouring accuracy by using generalized cross-coupled control. *International Journal of Machine Tools and Manufacture*, 63:49–57, 2012.

[48] Feng Huo and Aun-Neow Poo. Nonlinear autoregressive network with exogenous inputs based contour error reduction in cnc machines. *International Journal of Machine Tools and Manufacture*, 67:45–52, 2013.

[49] Feng Huo, Xue-Cheng Xi, and Aun-Neow Poo. Generalized taylor series expansion for free-form two-dimensional contour error compensation. *International Journal of Machine Tools and Manufacture*, 53(1):91–99, 2012.

[50] IEEE. Part 3: Carrier sense multiple access with collision detect on (csma/cd) access method and physical layer specifications. *IEEE Std 802.3, 2000 Edition*, pages i–1515, 2000.

[51] Z. Jamaludin, H. Van Brussel, G. Pipeleers, and J. Swevers. Accurate motion control of xy high-speed linear drives using friction model feedforward and cutting forces estimation. *CIRP Annals-Manufacturing Technology*, 57(1):403–406, 2008.

[52] Zamberi Jamaludin, Hendrik Van Brussel, and Jan Swevers. Quadrant glitch compensation using friction model-based feedforward and an inverse-model-based disturbance observer. In *Advanced Motion Control, 2008. AMC'08. 10th IEEE International Workshop on*, pages 212–217. IEEE, 2008.

[53] Young Hun Jeong, Byung-Kwon Min, and Dong-Woo Cho. Estimation of machine tool feed drive inclination from current measurements and a mathematical model. *International Journal of Machine Tools and Manufacture*, 46(12):1343–1349, 2006.

[54] David Kaiser and Parker Compumotor. Fundamentals of servo motion control. *Parker Compumotor*, 11, 2001.

[55] Girish Kant and Kuldip Singh Sangwan. Predictive modelling and optimization of machining parameters to minimize surface roughness using artificial neural network coupled with genetic algorithm. *Procedia CIRP*, 31:453–458, 2015.

[56] Yung-Chou Kao, Nhu-Tung Nguyen, Mau-Sheng Chen, and Shin-Tzong Su. A prediction method of cutting force coefficients with helix angle of flat-end cutter and its application in a virtual three-axis milling simulation system. *The International Journal of Advanced Manufacturing Technology*, 77(9-12):1793–1809, 2015.

[57] J. Kennedy and R. Eberhart. Particle swarm optimization. *Proceedings of IEEE international conference on neural networks*, 4(2):1942–1948, 1995.

[58] James Kennedy, James F Kennedy, Russell C Eberhart, and Yuhui Shi. *Swarm intelligence*. Morgan Kaufmann, 2001.

[59] Dohyun Kim and Doyoung Jeon. Fuzzy-logic control of cutting forces in cnc milling processes using motor currents as indirect force sensors. *Precision Engineering*, 35(1):143–152, 2011.

[60] Dohyun Kim, Do Hyeon Son, and Doyoung Jeon. Feed-system autotuning of a cnc machining center: Rapid system identification and fine gain tuning based on optimal search. *Precision Engineering*, 36(2):339–348, 2012.

[61] Dong-II Kim and Sungkwun Kim. An iterative learning control method with application for cnc machine tools. *Industry Applications, IEEE Transactions on*, 32(1):66–72, 1996.

[62] Y. Koren and C.C. Lo. Variable-gain cross-coupling controller for contouring. *CIRP Annals-Manufacturing Technology*, 40(1):371–374, 1991.

[63] Y Koren and CC Lo. Advanced controllers for feed drives. *CIRP Annals-Manufacturing Technology*, 41(2):689–698, 1992.

[64] S. Koziel and Z. Michalewicz. Evolutionary algorithms, homomorphous mappings, and constrained parameter optimization. *Evolutionary computation*, 7(1):19–44, 1999.

[65] An-Chen Lee, Ming-Tzong Lin, Yi-Ren Pan, and Wen-Yu Lin. The feedrate scheduling of nurbs interpolator for cnc machine tools. *Computer-Aided Design*, 43(6):612–628, 2011.

[66] HongSheng Li, Xingpeng Zhou, and YangQuan Chen. Iterative learning control for cross-coupled contour motion systems. In *Mechatronics and Automation, 2005 IEEE International Conference*, volume 3, pages 1468–1472. IEEE, 2005.

[67] Muguo Li and Da Liu. A novel adaptive self-turned pid controller based on recurrent-wavelet-neural-network for pmsm speed servo drive system. *Procedia Engineering*, 15:282–287, 2011.

[68] Lennart Ljung, editor. *System Identification (2Nd Ed.): Theory for the User*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1999.

[69] Roberto Augusto Gómez Loenzo, Pedro Daniel Alaniz Lumbreras, René de Jesús Romero Troncoso, and Gilberto Herrera Ruiz. An object-oriented architecture for sensorless cutting force feedback for cnc milling process monitoring and control. *Advances in Engineering Software*, 41(5):754–761, 2010.

[70] Jan Marian Maciejowski. *Predictive control: with constraints*. Pearson education, 2002.

[71] D.J.C. MacKay. A practical bayesian framework for backpropagation networks. *Neural computation*, 4(3):448–472, 1992.

[72] Atsushi Matsubara, Kotaro Nagaoka, and Tomoya Fujita. Model-reference feedforward controller design for high-accuracy contouring control of machine tool axes. *CIRP Annals-Manufacturing Technology*, 60(1):415–418, 2011.

[73] Xuesong Mei, Maosaomi Tsutsumi, Takanori Yamazaki, and Nuogang Sun. Study of the friction error for a high-speed high precision table. *International Journal of Machine tools and manufacture*, 41(10):1405–1415, 2001.

[74] Hesam Zomorodi Moghadam, Robert G Landers, and SN Balakrishnan. Hierarchical optimal contour control of motion systems. *Mechatronics*, 24(2):98–107, 2014.

[75] Luis Morales-Velazquez, Rene de Jesus Romero-Troncoso, Roque Alfredo Osornio-Rios, Gilberto Herrera-Ruiz, and J Jesus de Santiago-Perez. Special purpose processor for parameter identification of cnc second order servo systems on a low-cost fpga platform. *Mechatronics*, 20(2):265–272, 2010.

[76] Kotaro Nagaoka and Atsushi Matsubara. Improving motion accuracy of tool center point using model-reference feedforward controller. *Procedia {CIRP}*, 1(0):605 – 608, 2012.

[77] Kumpati S Narendra and Kannan Parthasarathy. Learning automata approach to hierarchical multiobjective analysis. *Systems, Man and Cybernetics, IEEE Transactions on*, 21(1):263–272, 1991.

[78] MGA Nassef, Christian Schenck, and Bernd Kuhfuss. Simulation-based parameter identification of a reduced model using neural networks. In *Control and Automation (ICCA), 2011 9th IEEE International Conference on*, pages 974–978. IEEE, 2011.

[79] Steffen Nissen. Implementation of a fast artificial neural network library (fann). *Report, Department of Computer Science University of Copenhagen (DIKU)*, 31, 2003.

[80] Hendro Nurhadi and Yeong-Shin Tarng. Experimental pc based tgpid control method for 2d cnc machine. *Expert Systems with Applications*, 38(7):8949–8962, 2011.

[81] Andrzej W Ordys et al. Predictive control for industrial applications. *annual reviews in control*, 25:13–24, 2001.

[82] E.S. Peer, F. Van den Bergh, and A.P. Engelbrecht. Using neighbourhoods with the guaranteed convergence pso. In *Swarm Intelligence Symposium, 2003. SIS'03. Proceedings of the 2003 IEEE*, pages 235–242. IEEE, 2003.

[83] Zhao Pengbing and Shi Yaoyao. Adaptive sliding mode control of the a-axis used for blisk manufacturing. *Chinese Journal of Aeronautics*, 27(3):708–715, 2014.

[84] R. Poli, J. Kennedy, and T. Blackwell. Particle swarm optimization. *Swarm intelligence*, 1(1):33–57, 2007.

[85] NA Rafan, Zamberi Jamaludin, Tsung Heng Chiew, Lokman Abdullah, and Mohd Nazmin Maslan. Contour error analysis of precise positioning for ball screw driven stage using friction model feedforward. *Procedia CIRP*, 26:712–717, 2015.

[86] Mostafizur Rahaman, Rudolf Seethaler, and Ian Yellowley. A new approach to contour error control in high speed machining. *International Journal of Machine Tools and Manufacture*, 88:42–50, 2015.

[87] R Ramesh, MA Mannan, and AN Poo. Tracking and contour error control in cnc servo systems. *International Journal of Machine Tools and Manufacture*, 45(3):301–326, 2005.

[88] CJ Rao, D Nageswara Rao, and P Srihari. Influence of cutting parameters on cutting force and surface finish in turning operation. *Procedia Engineering*, 64:1405–1415, 2013.

[89] Realtek. *RTL8139DL, Single Chip Multifunction 10/100Mbps Ethernet Controller with Power Management*, 8 2005. Rev. 1.2.

[90] R Tyrrell Rockafellar. Augmented lagrange multiplier functions and duality in nonconvex programming. *SIAM Journal on Control*, 12(2):268–285, 1974.

[91] K. Sedlaczek and P. Eberhard. Using augmented lagrangian particle swarm optimization for constrained problems in engineering. *Structural and Multidisciplinary Optimization*, 32(4):277–286, 2006.

[92] B. Sencer, Y. Altintas, and E. Croft. Feed optimization for five-axis cnc machine tools with drive constraints. *International Journal of Machine Tools and Manufacture*, 48(7):733–745, 2008.

[93] Burak Sencer, Tatsuya Mori, and Eiji Shamoto. Design and application of a sliding mode controller for accurate motion synchronization of dual servo systems. *Control Engineering Practice*, 21(11):1519–1530, 2013.

[94] Y. Shi and R. Eberhart. A modified particle swarm optimizer. In *The 1998 IEEE International Conference on Evolutionary Computation Proceedings, IEEE World Congress on Computational Intelligence*, pages 69–73. IEEE, 1998.

[95] Yuhui Shi and Russell C Eberhart. Fuzzy adaptive particle swarm optimization. In *Evolutionary Computation, 2001. Proceedings of the 2001 Congress on*, volume 1, pages 101–106. IEEE, 2001.

[96] Yaw-Shih Shieh, An-Chen Lee, and Chin-Sheng Chen. Cross-coupled biaxial step control for cnc edm. *International Journal of Machine Tools and Manufacture*, 36(12):1363–1383, 1996.

[97] Yi-Ti Shih, Chin-Sheng Chen, and An-Chen Lee. A novel cross-coupling control design for bi-axis motion. *International Journal of Machine Tools and Manufacture*, 42(14):1539–1548, 2002.

[98] Seungkil Son, Taejung Kim, Sanjay E Sarma, and Alexander Slocum. A hybrid 5-axis cnc milling machine. *Precision Engineering*, 33(4):430–446, 2009.

[99] Tomislav Staroveški, Danko Brezak, Toma Udiljak, and Dubravko Majetić. Implementation of a linux-based cnc open control system. In *12th International Scientific Conference on Production Engineering, Computer Integrated Manufacturing and High Speed Machining, CIM 2009*. Hrvatska znanstvena bibliografija i MZOS-Svibor, 2009.

[100] Suk-Hwan Suh, Seong Kyoon Kang, Dae-Hyuk Chung, and Ian Stroud. *Theory and design of CNC systems*. Springer Science & Business Media, 2008.

[101] YS Tarng and HE Cheng. An investigation of stick-slip friction on the contouring accuracy of cnc machine tools. *International Journal of Machine Tools and Manufacture*, 35(4):565–576, 1995.

[102] Ismaila B Tijani and Rini Akmeliawati. Support vector regression based friction modeling and compensation in motion control system. *Engineering Applications of Artificial Intelligence*, 25(5):1043–1052, 2012.

[103] Masayoshi Tomizuka. Robust digital motion controllers for mechanical systems. *Robotics and autonomous systems*, 19(2):143–149, 1996.

[104] N Tounsi, T Bailey, and MA Elbestawi. Identification of acceleration deceleration profile of feed drive systems in cnc machines. *International Journal of Machine Tools and Manufacture*, 43(5):441–451, 2003.

[105] Tsu-Chin Tsao and Masayoshi Tomizuka. Adaptive zero phase error tracking algorithm for digital control. *Journal of dynamic systems, measurement, and control*, 109(4):349–354, 1987.

[106] Naoki Uchiyama. Adaptive two-degree-of-freedom control of feed drive systems. *International Journal of Machine Tools and Manufacture*, 48(3):437–445, 2008.

[107] Naoki Uchiyama, Shigenori Sano, et al. Sliding mode contouring control design using nonlinear sliding surface for three-dimensional machining. *International Journal of Machine Tools and Manufacture*, 65:8–14, 2013.

[108] Naoki Uchiyama, Shigenori Sano, et al. Sliding mode contouring control for biaxial feed drive systems with a nonlinear sliding surface. *Procedia CIRP*, 14:506–510, 2014.

[109] Zuperl Uros, Cus Franc, and Kiker Edi. Adaptive network based inference system for estimation of flank wear in end-milling. *journal of materials processing technology*, 209(3):1504–1511, 2009.

[110] F. Van Den Bergh. *An analysis of particle swarm optimizers*. PhD thesis, University of Pretoria, 2002.

[111] F. van den Bergh and A.P. Engelbrecht. A new locally convergent particle swarm optimizer. In *Proceedings of the IEEE international conference on systems, man, and cybernetics*, pages 96–101, 2002.

[112] Wolfgang Wallner and Josef Baumgartner. openpowerlink in linux userspace: Implementation and performance evaluation of the real-time ethernet protocol stack in linux userspace. *Bernecker+ Rainer Industrie-Elektronik Ges. mbH, Austria*, 2011.

[113] Limei Wang, Shu Lin, and Hao Zheng. Precision contour control of xy table based on lugre model friction compensation. In *Intelligent Control and Information Processing (ICICIP), 2011 2nd International Conference on*, volume 2, pages 1124–1128. IEEE, 2011.

[114] Weibing Wang and Pengbing Zhao. Application of kalman filter in the cnc servo control system. *Procedia Engineering*, 7:442–446, 2010.

[115] Youqing Wang, Furong Gao, and Francis J Doyle. Survey on iterative learning control, repetitive control, and run-to-run control. *Journal of Process Control*, 19(10):1589–1600, 2009.

[116] Manfred Weck and Guohong Ye. Sharp corner tracking using the ikf control strategy. *CIRP Annals-Manufacturing Technology*, 39(1):437–441, 1990.

[117] Bogdan M Wilamowski and J David Irwin. *Industrial communication systems*. CRC Press, 2011.

[118] Cheng-Hsien Wu and Yu-Tai Kung. Thermal analysis for the feed drive system of a cnc machine center. *International Journal of Machine Tools and Manufacture*, 43(15):1521–1528, 2003.

[119] Jianhua Wu, Chao Liu, Zhenhua Xiong, and Han Ding. Precise contour following for biaxial systems via an a-type iterative learning cross-coupled control algorithm. *International Journal of Machine Tools and Manufacture*, 93:10–18, 2015.

[120] Jianhua Wu, Zhenhua Xiong, and Han Ding. Integral design of contour error model and control for biaxial system. *International Journal of Machine Tools and Manufacture*, 89:159–169, 2015.

[121] Xue-Cheng Xi, Geok-Soon Hong, and Aun-Neow Poo. Improving cnc contouring accuracy by integral sliding mode control. *Mechatronics*, 20(4):442–452, 2010.

[122] Xue-Cheng Xi, Aun-Neow Poo, and Geok-Soon Hong. Tracking error-based static friction compensation for a bi-axial cnc machine. *Precision Engineering*, 34(3):480–488, 2010.

[123] Hang Xie, Hao Tang, and Yu-He Liao. Time series prediction based on narx neural networks: An advanced approach. In *Machine Learning and Cybernetics, 2009 International Conference on*, volume 3, pages 1275–1279. IEEE, 2009.

[124] Mu-Tian Yan, Ming-Hung Lee, and Ping-Lang Yen. Theory and application of a combined self-tuning adaptive control and cross-coupling control in a retrofit milling machine. *Mechatronics*, 15(2):193–211, 2005.

[125] Zong-Mu Yeh, YS Tarng, and YS Lin. Cross-coupled fuzzy logic control for multiaxis machine tools. *Mechatronics*, 7(8):663–681, 1997.

[126] George Younkin. Modeling machine tool feed servo drives using simulation techniques to predict performance. In *Industry Applications Society Annual Meeting, 1989., Conference Record of the 1989 IEEE*, pages 1699–1706. IEEE, 1989.

[127] Ke Zhang, Alexander Yuen, and Yusuf Altintas. Pre-compensation of contour errors in five-axis cnc machine tools. *International Journal of Machine Tools and Manufacture*, 74:1–11, 2013.

[128] Liping Zhang, Huanjun Yu, and Shangxu Hu. A new approach to improve particle swarm optimization. In *Genetic and Evolutionary Computation—GECCO 2003*, pages 134–139. Springer, 2003.

[129] Yanshun Zhang, Tao Yang, Chunyu Li, Shanshan Liu, Chaochao Du, Ming Li, and Hailing Sun. Fuzzy-pid control for the position loop of aerial inertially stabilized platform. *Aerospace Science and Technology*, 36:21–26, 2014.

[130] Jan Żółtowski. *Podstawy konstrukcji maszyn: przekładnie*. Oficyna Wydawnicza Politechniki Warszawskiej, 2004.

Załącznik 1: Kod źródłowy wybranych funkcji

```
/*****************************************************
*                          REALTIME  FUNCTIONS          *
*****************************************************/

static void opt_compute(void *arg, long period)
{
unsigned int i;
unsigned int num_net_inputs, num_input_delays,
num_arx_inputs, num_outputs, num_output_delays;

for(i=0;i<num_pso;i++)
{
num_net_inputs = fann_get_num_input(neural_network[i]);
num_input_delays = shm_opt_fann_info[i]->num_input_delays;
num_arx_inputs = shm_opt_fann_info[i]->num_arx_inputs;
num_outputs = fann_get_num_output(neural_network[i]);
num_output_delays = shm_opt_fann_info[i]->num_output_delays;

if(online[i]==1)
{
/*      On-line Training */

/* Get value of network inputs number */
num_net_inputs = fann_get_num_input(neural_network[i]);
num_input_delays = shm_opt_fann_info[i]->num_input_delays;

/* shift input delays before update */
memmove(&input_network[i][1],
&input_network[i][0], (num_net_inputs-1)
* sizeof(fann_type) );
```

```
/* update input value from actual input value */
input_network[i][0]  = (*(io_array[i]->input_delta[0]));

/* update input value from previous output value */
input_network[i][num_input_delays]  = output_network[i][0];

/* update input value from previous output value */
output_network[i][0]  =
(*(io_array[i]->input_real_otput_err[0]));

/* Update train data via new incremental input and output
vector */
fann_update_train_data(train_data[i], input_network[i],
output_network[i]);

/* Train neural network */
fann_train_on_data(neural_network[i], train_data[i],
max_epochs[i], epochs_between_reports[i],
desired_error[i]);

}
else
{
/* update neural NARX model - input_real_otput
(incremental value) update */
model_update_input_real_otput(i);

/* update initial state for initial guess */
update_initial_state(pso_p[i], i);

/* PSO computation */
pso_run(pso_p[i], pso_dimension[i], fp_Fitnessfunction , i);

/* Final output incremental  and absolute values from
Global best values */
output_abs[i][0] = output_abs[i][0] + pso_p[i]->g[0];
//write actual absolute position
```

```
*(io_array[i]->output_abs[0]) = output_abs[i][0];
//write actual absolute position to pin
*(io_array[i]->output_delta[0]) = pso_p[i]->g[0];
//write actual incremental position to pi

/* Save actual velocity and acceleration as "old velocity"
and "old acceleration" for next iteration */
update_final_state(pso_p[i], i);
}

/* Save to file procedure */
if( (io_array[0]->save_to_file) == 1)/* check for
'save to file' param */
{
if((*shm_opt_fann_rtstatus) == FIO_READY) /* check if userspace
module isnt reading shm right now, we cant write if it does */
{
*shm_opt_fann_rtstatus = FIO_SAVING_SHM;/* save to memory
started, userspace module shouldnt read shm now */
for(i=0;i<num_nets;i++)
{
/* copy weight array to shm */
memcpy(shm_opt_fann_weigths[i],(neural_network[i]->weights),
((shm_opt_fann_info[i]->num_weights)*sizeof(fann_type)));
}
*shm_opt_fann_rtstatus = FIO_SAVED_SHM;/* save to memory
finished, userspace module can save to shm */
}
else if((*shm_opt_fann_rtstatus) == FIO_SAVED_FILE)
{
(io_array[0]->save_to_file) = 0;
(*shm_opt_fann_rtstatus) = FIO_READY;
rtapi_print("fann_halrt:␣nets␣saved␣to␣file␣\n");
}
}
/* End of save to file procedure */
}
```

```
}

/*************************************************************
*                    LOCAL FUNCTION DECLARATIONS             *
*************************************************************/

pso_type FeedOptFun(pso_type* x, unsigned int dimension,
unsigned int current_pso)
{

unsigned int k;
pso_type fitness = 0.0; /* returned value of fitness */
pso_type local_absolute_position =
(*(io_array[current_pso]->input_real_otput_abs_err[0]));
// actual real absolute position

/* Compute model */
compute_model(x, dimension, current_pso);

/* compute fitness value */
for(k=0;k<dimension;k++)
{
local_absolute_position =   local_absolute_position
// last absolute position
+ x[k]
// next proposal position increment
+ output_network[current_pso][k];
/* next computed position displacement of proposal
position increment */

fitness = fitness +
( ( (*(io_array[current_pso]->input_abs[k]))
 - (local_absolute_position) )
*( (*(io_array[current_pso]->input_abs[k]))
 - (local_absolute_position) ) );
}
```

```
return fitness;
}

void model_update_input_real_otput(unsigned int current_pso)
{
unsigned int num_net_inputs, num_input_delays;

num_net_inputs =
fann_get_num_input(neural_network[current_pso]);
num_input_delays =
shm_opt_fann_info[current_pso]->num_input_delays;

/* shift input delays before update */
memmove(&input_network_store[current_pso][1],
&input_network_store[current_pso][0], (num_net_inputs-1)
* sizeof(fann_type) );

/* update input position (feedback)*/
input_network_store[current_pso][num_input_delays] =
 (*(io_array[current_pso]->input_real_otput_err[0]));
}

void update_initial_state(pso_all* pso,
unsigned int current_pso)
{
int i;

for(i=0;i<(pso->fDim);i++)
{
/* Set velocity increment from trajectory generator */
(pso->pso_velo_pred_horiz[i]) =
(*(io_array[current_pso]->input_delta[i]));
}

(pso->pso_acce_pred_horiz[0]) =
(*(io_array[current_pso]->input_delta[0]))
 - (pso->velo_old_state);
```

```
for(i=1;i<(pso->fDim);i++)
{
/* Set velocity increment from trajectory generator */
(pso->pso_acce_pred_horiz[i]) =
(*(io_array[current_pso]->input_delta[i])) -
(*(io_array[current_pso]->input_delta[i-1]));
}
}

void update_final_state(pso_all* pso, unsigned int current_pso)
{
/* acce_odl_state = velo_next_state - velo_odl_state */
(pso->acce_old_state)
 = (*(io_array[current_pso]->input_delta[0]))
 - (pso->velo_old_state);

/* velo_odl_state = velo_next_state */
(pso->velo_old_state) =
(*(io_array[current_pso]->input_delta[0]));
}

void compute_model(pso_type* imput_values_vector,
unsigned int dimension, unsigned int current_pso)
{
unsigned int i;
unsigned int num_net_inputs, num_input_delays, num_arx_inputs,
num_outputs, num_output_delays;
fann_type *temp;

/* FEED OPTIMIZATION FITNESS FUNCTION
* x - array of optimized variables = consecutive feedrates
* dimension - optimization problem dimension = number of
* timesteps to optimize ahead
*
* 1) interpolate trajectory fragment in order to get speed
* profiles for each axis
```

```
* 2) pass feedrates to neural networks in order to compute
* errors for each axis
* 3) compute contour error
* 4) compute value of fitness function
*
* PSO minimizes difference between error and tolerance which
* should maximize speed
* because error increases with speed
* Fitness = sum_0^dimension((eps_tol(i) - est_eps(i))^2)
* est_eps = nnarx(x)
*/

num_net_inputs =
fann_get_num_input(neural_network[current_pso]);
num_input_delays =
shm_opt_fann_info[current_pso]->num_input_delays;
num_arx_inputs =
shm_opt_fann_info[current_pso]->num_arx_inputs;
num_outputs =
fann_get_num_output(neural_network[current_pso]);
num_output_delays =
shm_opt_fann_info[current_pso]->num_output_delays;

/* update input_network vector from input_network_store
vector before starting model computation */
memmove(input_network[current_pso],
input_network_store[current_pso],
(dimension) * sizeof(fann_type) );

for(i=0;i<dimension;i++)
{
/* update next input position from imput_values_vector */
input_network[current_pso][0] =
(fann_type)imput_values_vector[i];

/* compute FANN ARX */
temp = fann_run(neural_network[current_pso],
```

```
input_network[current_pso]);

output_network[current_pso][i] = temp[0];

/* shift inputs before next iteration computation */
memmove(&input_network[current_pso][1],
&input_network[current_pso][0],
(num_net_inputs-1) * sizeof(fann_type) );

/* update input position (feedback) from output_network*/
input_network[current_pso][num_input_delays]
 = output_network[current_pso][i];
}
}

/* ------------------------------------------------------------ */

pso_all* pso_create_standard(unsigned int num_particles,
unsigned int dimension, parameters param)
{
pso_all* pso;
pso_particle* particles;
pso_type* particle_values_x;
pso_type* particle_values_v;
pso_type* particle_values_p;
pso_type* gbest_p;
pso_type* gbest_pTemp;
pso_type* pso_velo_pred_horiz;
/* Set velocity increment from trajectory generator */
pso_type* pso_acce_pred_horiz;
/* Set acceleration increment from trajectory generator */
unsigned int i;

/* allocate and initialize the main pso structure */
pso = (pso_all *) rt_malloc(sizeof(pso_all));
if(pso == NULL)
{
rtapi_print("%s: Error: Out of memory.",__func__);
```

```
return NULL;
}

pso->fDim                                   = dimension;
/* Dimensionality of problem. */
pso->numIterations                  = 0;
/* Number of fitness-evaluations. */

pso->acce_inc_max                   = param.acce_inc_max;
/* Maximum acceleration increment
(Jerk constant maximum value). */
pso->acce_old_state         = param.acce_old_state;
/* Acceleration old value - from previous iteration. */
pso->acce_mul_bound         = param.acce_mul_bound;

pso->velo_old_state         = param.velo_old_state;
/* Velocity old value - from previous iteration. */
pso->velo_mul_bound                 = param.velo_mul_bound;

/* the initial velocity is allocated in one long array */
pso_velo_pred_horiz =
(pso_type*) rt_malloc(dimension * sizeof(pso_type));

if(pso_velo_pred_horiz == NULL)
{
rtapi_print("%s: Error: Out of memory.",__func__);
pso_destroy(pso);
return NULL;
}
for(i = 0; i != dimension; i++)
{
pso_velo_pred_horiz[i] = 0.0;
}

/* the initial acceleration is allocated in one long array */
pso_acce_pred_horiz =
(pso_type*) rt_malloc(dimension * sizeof(pso_type));
```

```
if(pso_acce_pred_horiz == NULL)
{
rtapi_print("%s: Error: Out of memory.",__func__);
pso_destroy(pso);
return NULL;
}
for(i = 0; i != dimension; i++)
{
pso_acce_pred_horiz[i] = 0.0;
}

/* the initial guess is allocated in one long array */
gbest_p = (pso_type*) rt_malloc(dimension * sizeof(pso_type));
if(gbest_p == NULL)
{
rtapi_print("%s: Error: Out of memory.",__func__);
pso_destroy(pso);
return NULL;
}
for(i = 0; i != dimension; i++)
{
gbest_p[i] = 0.0;
}

/* the initial Temporary guess is allocated
in one long array */
gbest_pTemp =
(pso_type*) rt_malloc(dimension * sizeof(pso_type));
if(gbest_pTemp == NULL)
{
rtapi_print("%s: Error: Out of memory.",__func__);
pso_destroy(pso);
return NULL;
}

pso->pso_velo_pred_horiz = pso_velo_pred_horiz;
pso->pso_acce_pred_horiz = pso_acce_pred_horiz;
```

```
pso->g = gbest_p;
/* All-time best found solution to problem. */
pso->gTemp = gbest_pTemp;
/* Temporary best found solution to problem. */
pso->gFitness = 0.0;
/* Fitness for best-found position. */
pso->gFitnessTemp = 0.0;
/* Temporary for best-found position. */

pso->particles = num_particles;
pso->inertiaWeight = param.inertiaWeight;
pso->weightParticleAttraction = param.weightParticleAttraction;
pso->WeightSwarmAttraction = param.WeightSwarmAttraction;

/* all the particles is allocated in one long array */
particles = (pso_particle*) rt_malloc((num_particles)
* sizeof(pso_particle));
if(particles == NULL)
{
rtapi_print("%s: Error: Out of memory.",__func__);
pso_destroy(pso);
return NULL;
}

pso->first_particle = particles;
pso->last_particle  = particles + num_particles;

/* all the particles values x is allocated in one long array */
particle_values_x = (pso_type*) rt_malloc(num_particles
* dimension * sizeof(pso_type));
if(particle_values_x == NULL)
{
rtapi_print("%s: Error: Out of memory.",__func__);
pso_destroy(pso);
return NULL;
}
```

```
/* all the particles values v is allocated in one long array */
particle_values_v = (pso_type*) rt_malloc(num_particles
* dimension * sizeof(pso_type));
if(particle_values_v == NULL)
{
rtapi_print("%s: Error: Out of memory.",__func__);
pso_destroy(pso);
return NULL;
}

/* all the particles values p is allocated in one long array */
particle_values_p = (pso_type*) rt_malloc(num_particles
* dimension * sizeof(pso_type));
if(particle_values_p == NULL)
{
rtapi_print("%s: Error: Out of memory.",__func__);
pso_destroy(pso);
return NULL;
}

for(i = 0; i != num_particles; i++)
{
particles[i].x_first = particle_values_x + (i * dimension);
particles[i].x_last  = particle_values_x + (i * dimension)
+ dimension;
particles[i].v_first = particle_values_v + (i * dimension);
particles[i].v_last  = particle_values_v + (i * dimension)
+ dimension;
particles[i].p_first = particle_values_p + (i * dimension);
particles[i].p_last  = particle_values_p + (i * dimension)
+ dimension;
particles[i].pFitness = 0;
particles[i].omega = 0;
}

return pso;
```

```
}

void pso_destroy(pso_all *pso)
{
if(pso == NULL)
return;

pso_safe_free(pso->pso_acce_pred_horiz);
rtapi_print("pso_destroy␣pso->pso_acce_pred_horiz\n");

pso_safe_free(pso->pso_velo_pred_horiz);
rtapi_print("pso_destroy␣pso->pso_velo_pred_horiz\n");

pso_safe_free(pso->g);
rtapi_print("pso_destroy␣pso->g\n");

pso_safe_free(pso->gTemp);
rtapi_print("pso_destroy␣pso->gTemp\n");

pso_safe_free(pso->first_particle->x_first);
rtapi_print("pso_destroy␣pso->first_particle->x_first\n");

pso_safe_free(pso->first_particle->v_first);
rtapi_print("pso_destroy␣pso->first_particle->v_first\n");

pso_safe_free(pso->first_particle->p_first);
rtapi_print("pso_destroy␣pso->first_particle->p_first\n");

pso_safe_free(pso->first_particle);
rtapi_print("pso_destroy␣pso->first_particle\n");

pso->last_particle = NULL;
rtapi_print("pso_destroy␣pso->last_particle\n");

pso_safe_free(pso);

}
```

```
void Guess_destroy(pso_type *Guess)
{
pso_safe_free(Guess);
}

void pso_run(pso_all* pso, unsigned int numIterations,
pso_type (*fp_Fitnessfunction)(pso_type*, unsigned int,
unsigned int), unsigned int current_pso)
{
pso_particle* last_particle = pso->last_particle;
pso_particle* particle_it;
pso_type* x_it;
pso_type* v_it;
pso_type* p_it;
pso_type* global = pso->g;
pso_type* globalTemp = pso->gTemp;

unsigned int dimension = pso->fDim;
unsigned int lastIteration = numIterations;

pso_type r1;
pso_type r2;

pso_type* v_pred_horiz = pso->pso_acce_pred_horiz;
/* Set acceleration increment from trajectory generator */
pso_type v_max_iter =  (pso->acce_inc_max);
/* Maximum acceleration increment
(Jerk constant maximum value). */
pso_type vos = pso->acce_old_state;
/* Acceleration old value - from previous iteration. */
pso_type vol;
/* Acceleration old value - from previous local iteration. */
pso_type v_max = 0.0;
/* Acceleration upper bound. */
pso_type v_min = 0.0;
/* Acceleration lower_bound. */
```

```
pso_type v_mul = pso->acce_mul_bound;
pso_type vu;
/* Acceleration local upper bound. */
pso_type vl;
/* Acceleration local lower_bound. */

pso_type* x_pred_horiz = pso->pso_velo_pred_horiz;
/* Set velocity increment from trajectory generator */
pso_type x_max_iter =  (pso->velo_inc_max);
/* Maximum velocity increment
(Acceleration constant maximum value). */
pso_type xos = pso->velo_old_state;
/* Velocity old value - from previous iteration. */
pso_type xol;
/* Velocity old value - from previous local iteration. */
pso_type x_max = 0.0;
/* Velocity upper bound. */
pso_type x_min = 0.0;
/* Velocity lower_bound. */
pso_type x_mul = pso->velo_mul_bound;

pso_type omega = pso->inertiaWeight;
pso_type phi1 = pso->weightParticleAttraction;
pso_type phi2 = pso->WeightSwarmAttraction;

pso_type newFitness;

pso_type kDenormalLimit =  1e-30;

//////////////////////////////////////////////////
//      pso_type initialGuessOld;
//////////////////////////////////////////////////

unsigned int i;
unsigned int j;
size_t dim_size = (pso->fDim) * sizeof(pso_type);
```

```
//////////////////////////////INIT////////////////

pso->numIterations = numIterations;

for (particle_it = pso->first_particle;
 particle_it != last_particle; particle_it++)
{
x_it = particle_it->x_first;
v_it = particle_it->v_first;
p_it = particle_it->p_first;

vol = vos;
xol = xos;

//////////////////////////////////////////////////
//              initialGuessOld = xos;
//////////////////////////////////////////////////

for(i = 0; i != dimension; i++)
{

v_max = v_pred_horiz[i] + ((v_max_iter) * (v_mul));
v_min = v_pred_horiz[i] - ((v_max_iter) * (v_mul));

x_max = x_pred_horiz[i] + ((x_max_iter) * (x_mul));
x_min = x_pred_horiz[i] - ((x_max_iter) * (x_mul));

//x_max = x_pred_horiz[i] + ( (x_max_iter) * (x_mul) );
//x_min = x_pred_horiz[i] + ( (x_min_iter) * (x_mul) );

/* Update lower and upper random bound of pso velocity
for current constrains */
pso_bounds_update(&vu, &vl, vol, v_max, v_min, v_max_iter,
 kDenormalLimit);

/* Initialize velocity. */
v_it[i] = pso_rand_min_max(vl, vu);
```

```
/* Compute pso value */
pso_compute_values(&x_it[i], xol, x_max, x_min, &v_it[i],
 kDenormalLimit);


/* Initialize best known position.
* Contents must be copied because the agent
* will likely move to worse positions. */
p_it[i] = x_it[i];

/* Actual value of v_it[i] becomes old value vol of v_it[i]
for next iteration*/
vol = v_it[i] = x_it[i] - xol;

/* Actual value of x_it[i] becomes old value xol of x_it[i]
for next iteration*/
xol = x_it[i];
}

particle_it->pFitness =
(*fp_Fitnessfunction)(particle_it->x_first,
 dimension, current_pso);
 //call to function pointer passed from callee

/* Update swarm's best known position.
* This must reference the agent's best-known
* position because the current position changes. */
if ( (particle_it == (pso->first_particle))
|| ((pso->gFitness) > (particle_it->pFitness)) )
{
memcpy(global, particle_it->p_first, dim_size);
memcpy(globalTemp, particle_it->p_first, dim_size);
pso->gFitness = particle_it->pFitness;
pso->gFitnessTemp = particle_it->pFitness;
}
}
```

```
//////////////////COMPUTE////////////////////////

/* Perform actual optimization iterations. */
for (j = 0;      j != lastIteration; j++)
{
//                k = 0;

for (particle_it = pso->first_particle; particle_it
!= last_particle; particle_it++)
{
x_it = particle_it->x_first;
v_it = particle_it->v_first;
p_it = particle_it->p_first;

vol = vos;
xol = xos;


for(i = 0; i != dimension; i++)
{

v_max = v_pred_horiz[i] + ((v_max_iter) * (v_mul));
v_min = v_pred_horiz[i] - ((v_max_iter) * (v_mul));

x_max = x_pred_horiz[i] + ((x_max_iter) * (x_mul));
x_min = x_pred_horiz[i] - ((x_max_iter) * (x_mul));

/* Pick random weights. */
r1 = pso_rand(); //rand(0,1)
r2 = pso_rand(); //rand(0,1)

/* Update velocity. */
v_it[i] = (omega * v_it[i])
+ (phi1 * r1 * (p_it[i] - x_it[i]) )
+ (phi2 * r2 * (global[i] - x_it[i]));
```

```
/* Compute pso velocity value */
pso_compute_velocity_values(&v_it[i], vol, v_max, v_min,
 v_max_iter,
 kDenormalLimit);

/* Compute pso value */
pso_compute_values(&x_it[i], xol, x_max, x_min, &v_it[i],
 kDenormalLimit);

/* Actual value of v_it[i] becomes old value vol of v_it[i]
 for next iteration*/
vol = v_it[i] = x_it[i] - xol;

/* Actual value of x_it[i] becomes old value xol of x_it[i]
 for next iteration*/
xol = x_it[i];
}

/* Compute new fitness. */
newFitness = (*fp_Fitnessfunction)(particle_it->x_first,
dimension, current_pso);

/* Update best-known position
in case of fitness improvement. */
if (newFitness < particle_it->pFitness)
{
/* Update best-known position.
* Contents must be copied because the agent
* will likely move to worse positions. */
memcpy(particle_it->p_first, particle_it->x_first, dim_size);
particle_it->pFitness = newFitness;

/* Update swarm's best known position.
* This must reference the agent's best-known
* position because the current position changes. */
if (particle_it->pFitness < pso->gFitness)
{
```

```
if (particle_it->pFitness < pso->gFitnessTemp)
{
        memcpy(globalTemp, particle_it->p_first, dim_size);
        pso->gFitnessTemp = particle_it->pFitness;
}
}

}

}

memcpy(global, globalTemp, dim_size);
pso->gFitness = pso->gFitnessTemp;

}

}

void pso_bounds_update(pso_type* upper,
pso_type* lower, pso_type old,
pso_type max_limit, pso_type min_limit,
pso_type max_iter, pso_type kDenormalLimit)
{
/* Determine new bounds */
*lower = (old - max_iter);
*upper = (old + max_iter);

/* Fix renormalized floating-point values. */
*lower = (fabs(*lower) > ( kDenormalLimit)) ? (*lower) : (0.0);
*upper = (fabs(*upper) > ( kDenormalLimit)) ? (*upper) : (0.0);

/* Enforce value bounds before updating value. */
*lower = (*lower > min_limit) ? (*lower) : (min_limit);
*upper = (*upper < max_limit) ? (*upper) : (max_limit);
}

void pso_compute_values(pso_type* new_val, pso_type old,
```

```
pso_type max_limit, pso_type min_limit, pso_type* iter,
pso_type kDenormalLimit)
{
/* Compute new value */
*new_val = (old + *iter);

/* Fix renormalized floating-point values. */
*new_val = (fabs(*new_val) > ( kDenormalLimit)) ? (*new_val)
: (0.0);

/* Enforce value bounds before updating value. */
*new_val = (*new_val > min_limit) ? (*new_val) : (min_limit);
*new_val = (*new_val < max_limit) ? (*new_val) : (max_limit);
}

void pso_compute_velocity_values(pso_type* actual,
pso_type prev, pso_type max_limit, pso_type min_limit,
pso_type max_iter, pso_type kDenormalLimit)
{
/* Enforce value bounds before updating value. */
*actual = ((*actual - prev) > (-max_iter)) ? (*actual)
: (prev - max_iter);
*actual = ((*actual - prev) < ( max_iter)) ? (*actual)
: (prev + max_iter);

/* Fix renormalized floating-point values. */
*actual = (fabs(*actual)>( kDenormalLimit))?(*actual):(0.0);

/* Enforce value bounds before updating value. */
*actual = (*actual > min_limit) ? (*actual) : (min_limit);
*actual = (*actual < max_limit) ? (*actual) : (max_limit);
}

/* ---------------------------------------------------------- */
```