POLITECHNIKA WARSZAWSKA

Wydział Elektryczny

ROZPRAWA DOKTORSKA

mgr inż. Krystian Erwiński

Metoda doboru prędkości posuwu w układach sterowania numerycznego maszyn wieloosiowych z wykorzystaniem algorytmów sterowania predykcyjnego oraz sztucznych sieci neuronowych

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

Warszawa 2014

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 zagadnienia wyznaczania optymalnej prędkości posuwu w układach sterowania maszyn wieloosiowych w szczególności obrabiarek CNC (ang. Computerized Numerical Control - sterowanie numeryczne z użyciem komputera). W pracy omówiono problematykę generacji trajektorii ruchu w układach sterowania maszyn CNC. Zaproponowano nowatorskie rozwiązanie doboru maksymalnej prędkości posuwu przy zachowaniu zadanej tolerancji błędów konturu. Algorytm wykorzystuje neuronowy model napędów posuwu maszyny w celu estymacji błędów konturu powstałych w procesie obróbki. Na tej podstawie algorytm optymalizacyjny wyznacza optymalny profil prędkości. Nielinowy problem optymalizacyjny rozwiązywany jest z wykorzystaniem algorytmu optymalizacji rojem cząstek (ang. Particle Swarm Optimization - PSO) rozbudowanego o metodę mnożników Lagrange'a. Algorytm zaimplementowano w sterowniku CNC w formie komputera PC z systemem czasu rzeczywistego Linux RTAI. Na podstawie uzyskanych wyników badań symulacyjnych i doświadczalnych wykazano zmniejszenie czasu realizacji zadanej trajektorii ruchu przy jednoczesnym zapewnieniu nie przekraczania zadanego poziomu błędów konturu.

Słowa kluczowe: *CNC, optymalizacja prędkości posuwu, błędy konturu, model neuronowy, optymalizacja rojem cząstek.*

Abstract

The dissertation concerns the problem of optimal feedrate determination in multi-axis machine control systems - especially CNC machine tools. Problem of trajectory generation in CNC machine tool control system is described. A novel solution of maximum feedrate determination is presented which ensures constriction of contour error to arbitrary levels. The algorithm utilizes a neural model of machine feed drives in order to estimate contour errors during machining. Using this information an optimization algorithm determines the optimum feedrate profile. The nonlinear optimization problem is solved utilizing the Particle Swarm Optimization algorithm extended with the Augmented Lagrangian method. The algoritm was implemented in a PC based CNC controller with Linux RTAI real-time operating system. Simulation and experimental results confirm that ralization time of the given motion trajectory was decreased without violating given contour error limit.

Keywords: *CNC, feedrate optimization, contour error, neural network, particle swarm optimization.*

Spis treści

1. Wprowadzenie

Wieloosiowe obrabiarki CNC (ang. Computerized Numerical Control – komputerowe sterowanie numeryczne) są obecnie podstawowymi narzędziami procesu wytwórczego w wielu gałęziach przemysłu. Wykorzystywane są między innymi do obróbki skrawaniem, szlifowania, spawania, cięcia, wiercenia różnych materiałów takich jak metale, tworzywa sztuczne, drewno. W ostatnich latach wiele wysiłku poświęca się na udoskonalanie układów sterowania obrabiarek. Postęp w dziedzinie układów mikroprocesorowych pozwala na wprowadzanie nowych zaawansowanych algorytmów sterowania w celu poprawy dokładności i wydajności procesu obróbki.

Oś mechaniczna maszyny jest obecnie najczęściej budowana z wykorzystaniem silników synchronicznych prądu przemiennego z magnesami trwałymi (ang. Permanent Magnet Synchronous Motor - PMSM). W rozwiązaniach nisko-kosztowych i amatorskich stosuje się silniki krokowe. Ruch obrotowy silnika zamieniany jest na ruch postępowy suportu za pomocą przekładni śrubowej tocznej. Śruba może być połączona z silnikiem za pośrednictwem przekładni (np. pasowej) lub bezpośrednio z wykorzystaniem sprzęgła. W dużych maszynach lub jeżeli wymagane są duże prędkości posuwu zamiast przekładni śrubowej stosuje się układ koło zębate - listwa zębata (ang. rack and pinion). Spotyka się maszyny, w których ruch postępowy osi realizowany jest bez użycia przekładni poprzez silniki liniowe.

Najczęściej spotykanym typem maszyn CNC są maszyny 3-osiowe i 5-osiowe. Są to maszyny w układzie kartezjańskim tzn. osie mechaniczne maszyny realizujące ruch postępowy odpowiadają osiom kartezjańskiego układu współrzędnych. Maszyny 5-osiowe posiadają dwie dodatkowe osie obrotowe pozwalające na wychylanie narzędzia i obróbkę pod kątem. Rzadziej spotyka się maszyny o bardziej złożonej kinematyce np. typu „hexapod” lub „tripod”, w których posuw w poszczególnych osiach kartezjańskich jest nieliniową funkcją posuwu osi mechanicznych maszyny [48, 45].

Pozycjonowanie poszczególnych silników PMSM realizowane jest z wykorzystaniem cyfrowych serwonapędów. Układ regulacji składa się z kaskadowych regulatorów typu PID ze sprzężeniem w przód (ang. feed-forward). W celu zapewnienia sprzężenia zwrotnego do układu regulacji dokonywany jest pośredni pomiar położenia suportu z wykorzystaniem przetworników obrotowo-impulsowych (enkoderów) lub rezolwerów zamontowanych na wale silnika [41]. W przypadku występowania dużej podatności pomiędzy silnikiem a częścią

mechaniczną lub konieczności precyzyjnego pomiaru położenia suportu stosuje się liniały pomiarowe.

Położenia zadane dla poszczególnych serwonapędów są przesyłane z nadrzędnego sterownika CNC. Sterownik maszyny CNC ma za zadanie wygenerowanie w stałych odstępach czasu (np. co 1ms) przemieszczeń zadawanych serwonapędom poszczególnych osi mechanicznych maszyny. Złożenie posuwów poszczególnych osi prowadzi do uzyskania przemieszczenia narzędzia (np. frezu) wzdłuż zadanego toru ruchu, a w konsekwencji uzyskaniu pożądanego kształtu obrabianego przedmiotu. Jest to tak zwane sterowanie kształtowe (ang. contouring control lub continuous path control) [56, 57, 80]. Wartość przemieszczeń determinowana jest przez chwilową prędkość posuwu narzędzia, zadaną w danym punkcie toru ruchu. Wartość prędkości w kolejnych fazach realizacji toru ruchu wyznaczana jest w procesie generacji trajektorii ruchu. Wyznaczanie kolejnych przemieszczeń dokonywane jest w procesie interpolacji toru ruchu opisanego matematycznie za pomocą linii prostych, łuków lub krzywych na podstawie wygenerowanej uprzednio trajektorii.

Dawniej do opisu zadanego toru ruchu stosowano głównie linie proste i łuki ze względu na ograniczone możliwości obliczeniowe sterowników. Wadą tego rozwiązania była trudność reprezentacji złożonych kształtów co wymuszało aproksymację toru ruchu dużą liczbą segmentów liniowych. Nieciągłość toru ruchu powodowała nagłe zmiany prędkości i przyspieszenia co przekładało się na niższą dokładność procesu obróbki i wymuszało realizację toru ruchu z mniejszymi prędkościami. W ostatnich latach do definicji zadanego toru ruchu coraz częściej wykorzystuje się wielomianowe krzywe sklejane B-Spline i NURBS (ang. Non-Uniform Rational B-Spline) [77]. Ich zaletą jest możliwość dokładnej i zwartej reprezentacji bardzo złożonych kształtów oraz powszechne wykorzystanie w narzędziach projektowych typu CAD (ang. Computer Assisted Design - projektowanie wspomagane komputerowo). Najistotniejszą zaletą tej reprezentacji jest możliwość zdefiniowania toru ruchu o pożądanej gładkości (gwarancja ciągłości odpowiedniej pochodnej) co przekłada się na większe możliwe prędkości posuwu [64, 53, 61, 60]. Wadą jest o wiele większa złożoność obliczeniowa procesu interpolacji jednak wzrost mocy obliczeniowych układów mikroprocesorowych spowodował, że wada ta jest nieistotna.

W sterowaniu kształtowym maszyn CNC bardzo istotnym problemem jest wybór optymalnej prędkości posuwu narzędzia. Posuw z dużymi prędkościami i przyspieszeniami umożliwia krótszy czas obróbki co przekłada się na zwiększenie wydajności procesu produkcyjnego. Z drugiej strony większe prędkości i przyspieszenia powodują zwiększenie błędów nadążania w poszczególnych osiach. Jest to spowodowane dynamiką układu posuwu (serwonapędów, silników i części mechaniczej). Przy wyższych przyspieszeniach i prędkościach nadążne układy regulacji w serwonapędach (z reguły kaskadowe regulatory typu P-I) nie są w stanie skutecznie minimalizować błędów nadążania ze względu na ograniczone

pasmo, co prowadzi do powstawania przeregulowań. Zadane prędkości i przyspieszenia mogą doprowadzić do wystąpienia efektów nasyceniowych w serwonapędzie wynikających z ograniczonej mocy układu napędowego. Problem doboru odpowiedniej prędkości zadanej w układach sterowania CNC jest od dawna obecny w literaturze. W ostatnich latach nastąpił w tej dziedzinie znaczący rozwój związany ze zwiększeniem mocy obliczeniowych sterowników CNC.

1.1. Przegląd metod doboru prędkości posuwu

W najprostszym przypadku można przyjąć stałą prędkość posuwu dla całości bądź fragmentów toru ruchu. Rozpędzanie lub hamowanie realizowane jest poprzez prostokątne lub trapezoidalne profilowanie przyspieszenia. Maksymalne wartości prędkości, przyspieszenia i zrywu (w przypadku profilowania trapezoidalnego) zależą od właściwości dynamicznych poszczególnych osi, toru ruchu oraz przyjętej tolerancji błędu odtwarzania toru ruchu. Parametry te dobierane są arbitralnie przez operatora maszyny dla najgorszego możliwego przypadku. Poza najprostszymi kształtami bardzo trudno jest w ten sposób dobrać prędkość posuwu w pełni wykorzystującą możliwości maszyny i jednocześnie zapewniającą zachowanie wszystkich ograniczeń wynikających z dynamiki maszyny i układu sterowania.

Rosnące wymagania dotyczące wydajności produkcji i konieczność obróbki coraz bardziej złożonych kształtów połączone ze zwiększeniem możliwości obliczeniowych sterowników CNC spowodowało wzrost zainteresowania automatycznym doborem parametrów ruchu wśród badaczy.

Problem generacji czasowo-optymalnych trajektorii ruchu początkowo pojawiał się głównie w literaturze z zakresu sterowania robotami. Bobrow i in. [10] oraz Shin i McKay [91], niezależnie od siebie, jako pierwsi zaproponowali algorytmy generacji czasowo-optymalnych trajektorii ruchu z uwzględnieniem ograniczeń przyspieszenia. Metoda polegała na przełączaniu pomiędzy ekstremalnymi wartościami przyspieszenia w wybranych punktach (sterowanie typu “bang-bang”). Shiller i Lu [89, 90] opracowali wydajną metodę znajdowania punktów przełączania przyspieszenia. Inni badacze wykorzystywali w procesie optymalizacji prędkości programowanie dynamiczne [92] lub zasadę minimum Pontyagrin’a [39].

W dziedzinie sterowania kartezjańskimi maszynami CNC podejście podobne do Bobrowa zaproponowali Renton i Elbestawi [81]. Generacja optymalnej trajektorii ruchu następuje w wyniku analizy całej trajektorii ruchu od punktu początkowego do końcowego a następnie od punktu końcowego do początku przy jednoczesnej modyfikacji maksymalnej prędkości posuwu. Timar i in. [97] zaproponowali metodę optymalizacji prędkości dla torów ruchu maszyn CNC opisanych krzywymi wielomianowymi. Wyznaczyli optymalne wartości prędkości i przyspieszenia jako analityczną funkcję wymierną kawałkami ciągłą z punktami

przełączania będącymi pierwiastkami równań wielomianowych. Podobną metodę do opisanych powyżej zaproponowali Dong i Stori [23].

Zasadniczą wadą wszystkich powyższych metod jest uwzględnianie jedynie ograniczeń przyspieszenia lub momentu poszczególnych osi. Prowadzi to do nieciągłości przyspieszenia a tym samym uzyskana trajektoria nie jest gładka. Może to prowadzić do większych wibracji w układzie mechanicznym maszyny co przekłada się na zwiększone zużycia elementów mechanicznych. W ekstremalnych przypadkach znaczne skoki przyspieszenia mogą doprowadzić do występowania rezonansów mechanicznych. Aby tego uniknąć część badaczy zaproponowała uwzględnienie w procesie generacji trajektorii ruchu ograniczeń pochodnej przyspieszenia czyli zrywu (ang. jerk). Wpływ ograniczenia zrywu na strukturę mechaniczną został szczegółowo opisany przez Barre i in. w artykule [6]. Wadą uwzględnienia ograniczenia zrywu jest większa złożoność obliczeniowa algorytmów ze względu na konieczność stosowania rozwiązań iteracyjnych w odróżnieniu od metod uwzględniających jedynie ograniczenie przyspieszenia osi. W dziedzinie sterowania robotami, algorytmy generacji czasowo-optymalnej trajektorii ruchu uwzględniające ograniczenia zrywu zaproponowali Piazzi i Vistoli [76], Constantinescu i Croft [19] oraz Dong [22]. Zasadniczą wadą wyżej wymienionych metod jest konieczność dwukrotnego przetworzenia całej trajektorii ruchu (od początku do końca a następnie od końca do początku). W przypadku długich i złożonych trajektorii proces ten może trwać bardzo długo ze względu na znaczną ilość optymalizowanych zmiennych i uwzględnianych ograniczeń. Weck zaproponował metodę ograniczenia prędkości posuwu w danym segmencie trajektorii ruchu biorąc pod uwagę maksymalną lokalną krzywiznę toru ruchu [102]. Zaletą tej metody jest prostota i mała złożoność obliczeniowa, wadą natomiast stosunkowo zachowawczy dobór parametrów ruchu a przez to niewykorzystanie w pełni możliwości maszyny. Altinas i Erkorkmaz [3] zaproponowali metodę generacji czasowo-optymalnego profilu prędkości zapewniającą minimalizację zrywu. W tej metodzie profil prędkości opisany jest wielomianem sklejanym. W danym kroku algorytmu optymalizacji podlegają parametry jedynie fragmentu profilu prędkości dzięki czemu złożoność obliczeniowa jest niezależna od długości zadanego toru ruchu. Ponieważ zmiennymi optymalizowanymi są parametry wielomianu a nie wartości prędkości w poszczególnych chwilach czasu ilość optymalizowanych zmiennych jest mniejsza. Optymalizacja przeprowadzana jest metodą gradientową z uwzględnieniem ograniczeń przyspieszenia i zrywu stycznych do toru ruchu. Ponieważ profil prędkości posuwu ma postać wielomianu, wartości zadane prędkości i przyspieszenia poszczególnych osi są ciągłe. Podobną metodę dla manipulatorów robotycznych opracowały Macfarlane i Croft [69]. Heurystyczny algorytm doboru prędkości posuwu uwzględniający te same ograniczenia zaproponowali Heng i Erkorkmaz [42]. Algorytm polega na podziale trajektorii ruchu na segmenty i połączenie ich ustalonym, wielomianowym profilem

prędkości typu krzywa S (ang. S-curve). Prędkość na styku dwóch profili dobierana jest metodą bisekcji.

Nam i Yang przedstawili w [71] rekurencyjny algorytm generacji trajektorii ruchu polegający na estymacji w każdym punkcie trajektorii ruchu dopuszczalnego przemieszczenia uwzględniając ograniczenia prędkości, przyspieszenia i zrywu stycznych do toru ruchu. Ponadto prędkość jest ograniczana tak, że błąd cięciwowy (ang. chord error) nie przekracza zadanej maksymalnej wartości. Błąd cięciwowy oznacza odległość pomiędzy linią łączącą dwa sąsiednie punkty interpolowanej trajektorii ruchu a łukiem przechodzącym przez te dwa punkty o promieniu krzywizny równym promieniowi krzywizny toru ruchu. Podobne algorytmy uwzględniające ograniczenia prędkości, przyspieszenia, zrywu i błędu cięciwowego zostały zaproponowane w artykułach [59, 62].

Wadą powyższych rozwiązań jest uwzględnianie jedynie prędkości, przyspieszenia i zrywu w kierunku stycznym do toru ruchu. Nie są brane pod uwagę ograniczenia tych wielkości w poszczególnych osiach. Algorytm generacji trajektorii ruchu uwzględniający ograniczenia w osiach maszyny został zaproponowany przez Sencera [85]. W tej metodzie profil prędkości opisany jest krzywą sklejaną Beziera (B-spline). Gradientowy algorytm optymalizacyjny maksymalizuje sumę wartości punktów kontrolnych profilu prędkości jednocześnie uwzględniając ograniczenia prędkości, przyspieszenia i zrywu w poszczególnych osiach. Li i inni [63] zastosowali parametryzację wektora sterowania do sformułowania nieliniowego problemu optymalizacyjnego, którego rozwiązanie pozwala na znalezienie czasowo-minimalnej trajektorii ruchu uwzględniającej ograniczenia prędkości przyspieszenia i zrywu w osiach oraz błędu cięciwowego. Podobne rozwiązania zaproponowano w artykułach [114, 115, 31]. We wszystkich powyższych rozwiązaniach konieczne jest zastosowanie nieliniowego algorytmu optymalizacyjnego - najczęściej gradientowego (np. SQP). Wadą tego rozwiązania jest problem rozwiązania startowego, które musi być wystarczająco blisko rozwiązania optymalnego aby uniknąć problemów związanych z minimami lokalnymi.

W algorytmie opracowanym przez Beudaert'a i in. [8] optymalny profil prędkości posuwu generowany jest poprzez znalezienie w każdym kroku maksymalnego przemieszczenia, które nie spowoduje przekroczenia wartości prędkości, przyspieszenia i zrywu w osiach maszyny. Jeżeli znalezienie takiego przemieszczenia nie jest możliwe algorytm rozpoczyna pracę od wcześniejszego punktu (ang. backtracking) ograniczając maksymalne przemieszczenie. Algorytm nie korzysta z metod optymalizacji nieliniowej ale jego złożoność obliczeniowa zależy od złożoności zadanego toru ruchu. W przypadku złożonych trajektorii może być konieczne wielokrotne rozpoczynanie procesu generacji trajektorii od wcześniejszych punktów.

Wszystkie wyżej wymienione prace uwzględniały w procesie generacji trajektorii ruchu jedynie właściwości zadanego toru ruchu oraz przyjęte ograniczenia prędkości, przyspieszenia i zrywu. Nie brano pod uwagę dynamiki układu posuwu i układu sterowania, które mają

decydujący wpływ na błędy odtwarzania zadanej trajektorii ruchu. Uwzględnienie dynamiki w procesie generacji trajektorii ruchu zaproponowali Lin, Tsai i in. [98, 65]. W tym algorytmie profil prędkości składa się z segmentów połączonych krzywymi S. Liniowy model układu posuwu wykorzystywany jest do sprawdzania czy błąd odtwarzania trajektorii ruchu nie jest przekroczony. Jeżeli tak jest maksymalna prędkość w danym segmencie jest iteracyjnie zmniejszana do chwili gdy przewidywany błąd odtwarzania trajektorii spadnie poniżej zadanej wartości. Wadą tego rozwiązania jest wykorzystanie ustalonych profili prędkości w postaci krzywych S co wymaga przyjęcia stałych maksymalnych wartości prędkości, przyspieszenia i zrywu stycznych do toru ruchu. Wygenerowany profil prędkości nie wykorzystuje w pełni możliwości maszyny gdyż maksymalna prędkość modyfikowana jest w całym segmencie trajektorii ruchu nawet jeżeli wartość błędu konturu przekroczona jest tylko minimalnie. Ponadto wykorzystywany model jest modelem liniowym.

Przedstawiona powyżej koncepcja wykorzystania modelu układu posuwu w procesie generacji trajektorii ruchu skłoniła autora do podjęcia próby rozwiązania problemu wykorzystania nieliniowego modelu układu posuwu w procesie generacji czasowo-optymalnej trajektorii ruchu uwzględniającym dodatkowo ograniczenia maksymalnych wartości prędkości, przyspieszenia i zrywu w poszczególnych osiach maszyny.

1.2. Cel, teza i założenia pracy

Aby wykorzystać możliwości maszyny w maksymalnym stopniu, prędkość posuwu powinna być dobierana indywidualnie dla każdego fragmentu toru ruchu. Wymaga to analizy zadanej trajektorii narzędzia oraz uwzględnienia ograniczeń wartości prędkości i przyspieszenia dla poszczególnych osi maszyny.

Wyznaczając maksymalną prędkość posuwu narzędzia oprócz własności dynamicznych poszczególnych osi (bezwładność i tarcie w układzie mechanicznym, maksymalna moc napędów, pasmo przenoszenia układu regulacji) należy również wziąć pod uwagę geometrię zadanego toru ruchu. Wpływa ona na kształt profilu prędkości zadanego do serwonapędów ponieważ prędkość styczna jest złożeniem prędkości w poszczególnych osiach. Jeżeli tor ruchu charakteryzuje się małą krzywizną, zadana prędkość styczna może być większa niż w przypadku złożonego toru ruchu o dużej wartości krzywizny. W przypadku dużej krzywizny toru ruchu prędkość należy zmniejszyć.

W części procesów produkcyjnych dąży się do maksymalizacji prędkości posuwu przy zagwarantowaniu że trajektoria ruchu zostanie zrealizowana z pewną określoną dokładnością. Dokładność realizacji zadanego toru ruchu określa się mierząc błąd konturu (ang. contour error) czyli odległość pomiędzy torem zadanym a torem rzeczywiście realizowanym przez maszynę. W ogólności zależność pomiędzy zadaną prędkością posuwu a błędem konturu

ma charakter nieliniowy co powoduje, że uwzględnienie go w procesie generacji trajektorii ruchu jest problemem nietrywialnym. Analizując dostępną literaturę autor stwierdził że w procesie generacji optymalnej trajektorii ruchu nie uwzględniano w należytym stopniu błędów odtwarzania toru ruchu będących wynikiem własności dynamicznych układu posuwu. Skłoniło to autora do podjęcia próby rozwiązania tego problemu.

W rozprawie proponuje się zastosowanie algorytmów sterowania predykcyjnego do wyznaczenia czasowo-optymalnego profilu prędkości posuwu maszyny wieloosiowej. Sterowanie predykcyjne wykorzystuje model obiektu do wypracowania strategii sterowania minimalizującej współczynnik jakości odpowiedni dla danego problemu. W niniejszej rozprawie wykorzystuje się model napędu posuwu w formie sztucznych sieci neuronowych do przewidywania z wyprzedzeniem błędów konturu realizowanego toru ruchu i maksymalizacji prędkości posuwu bez przekraczania zadanej tolerancji błędu konturu.

Sformułowano następującą tezę rozprawy:

Możliwe jest skrócenie czasu realizacji zadanej trajektorii ruchu wieloosiowej maszyny sterowanej numerycznie, przy zachowaniu zadanej dokładności odtwarzania tej trajektorii, poprzez dobór prędkości posuwu z wykorzystaniem predykcyjnej procedury optymalizacyjnej oraz modelu maszyny w postaci sztucznej sieci neuronowej.

Celem pracy jest opracowanie i implementacja w sterowniku CNC algorytmu generacji trajektorii ruchu obrabiarek sterowanych numerycznie zapewniającego minimalny czas realizacji zadanego toru ruchu przy zapewnieniu nie przekraczania zadanej tolerancji błędów konturu. W szczególności za istotne punkty uznano:

— opracowanie neuronowego modelu układu posuwu służącego do predykcji błędu konturu
— opracowanie algorytmu generacji optymalnej trajektorii ruchu dla maszyn CNC;
— implementację sterownika CNC bazującego na komputerze PC z systemem czasu rzeczywistego, współpracującego z komercyjnymi serwonapędami prądu przemiennego;
— implementację proponowanego algorytmu w sterowniku CNC;
— przeprowadzenie badań doświadczalnych weryfikujących zasadność zaproponowanego rozwiązania;

W pracy zakłada się, że zadana trajektoria ruchu opisana jest za pomocą krzywych NURBS rzędu trzeciego (druga pochodna ciągła). Trajektorię w tej postaci można uzyskać bezpośrednio z plików CAD ponieważ krzywe NURBS są natywnym formatem wykorzystywanym w wielu programach projektowych. Alternatywnie można dopasować trajektorię NURBS do istniejącej trajektorii w formacie G-Code wykorzystując metodę najmniejszych kwadratów. Algorytm dopasowania krzywej NURBS do istniejącej trajektorii ruchu został opisany między innymi w pracach [30, 94].

W pracy nad algorytmem optymalizacji prędkości posuwu skupiono się na uwzględnieniu błędów wynikających z dynamiki maszyny. Nie uwzględniano błędów pochodzących od procesu obróbki (w szczególności sił skrawania). Założenie takie jest słuszne wszędzie tam gdzie obrabiane są materiały miękkie (tworzywa sztuczne, drewno) lub narzędzie nie wchodzi w kontakt z obrabianym materiałem (np. cięcie laserem, cięcie strumieniem wodnym). Uwzględnienie wpływu obciążenia wynikającego z procesu obróbki jest możliwe, wymagało by jednak opracowania modelu sił skrawania co jest problemem złożonym i wykracza poza ramy tej pracy. Model sił skrawania został zaproponowany między innymi w pracach [112, 27, 101].

W badaniach wykorzystywana jest dwuosiowa obrabiarka sterowana numerycznie w układzie kartezjańskim z przekładniami śrubowymi tocznymi i bezpośrednim połączeniem silników ze śrubą za pomocą sprzęgła. Maszyna wyposażona jest w silniki prądu przemiennego z magnesami trwałymi współpracujące z serwonapędami. Pomiar błędów odtwarzania trajektorii ruchu realizowany jest na wale silnika z wykorzystaniem inkrementalnych przetworników obrotowo-impulsowych (enkoderów) wbudowanych w silnik. Założenie to podyktowane jest faktem, iż wykorzystanie przekładni śrubowej tocznej zapewnia dużą sztywność układu posuwu i małe luzy a tym samym zmniejsza błąd pomiaru położenia na wale silnika w stosunku do położenia rzeczywistego mierzonego z wykorzystaniem liniałów. W praktyce większość maszyn dostępnych na rynku nie jest wyposażona w liniały pomiarowe gdyż koszt tych urządzeń zdecydowanie przewyższa korzyści wynikające ze zwiększenia dokładności pomiaru położenia w większości zastosowań.

Praca składa się z siedmiu rozdziałów uzupełnionych wykazem literatury. W pierwszym rozdziale zamieszczono wprowadzenie, przedstawiono przegląd literatury dotyczącej omawianego problemu, przedstawiono motywację autora do podjęcia tematu pracy. Ponadto przedstawiono tezę, założenia i cele pracy. Rozdział drugi poświęcono układom sterowania numerycznego maszyn. W rozdziale trzecim omówiono zagadnienie modelowania układu posuwu maszyn wieloosiowych. Przedstawiono opracowaną przez autora metodę identyfikacji nieliniowego modelu maszyny i przedstawiono wyniki badań zidentyfikowanego modelu. W rozdziale czwartym zawarto opis opracowanego algorytmu optymalizacji prędkości posuwu uwzględniającego ograniczenia błędu konturu. W rozdziale piątym opisano stanowisko laboratoryjne. W rozdziale szóstym przedstawiono wyniki badań opracowanego algorytmu na obiekcie rzeczywistym. Rozprawę zamyka rozdział siódmy, w którym sformułowano wnioski końcowe oraz wymieniono osiągnięcia własne. Fragmenty kodu źródłowego zaimplementowanego algorytmu zawarto w załączniku 1.

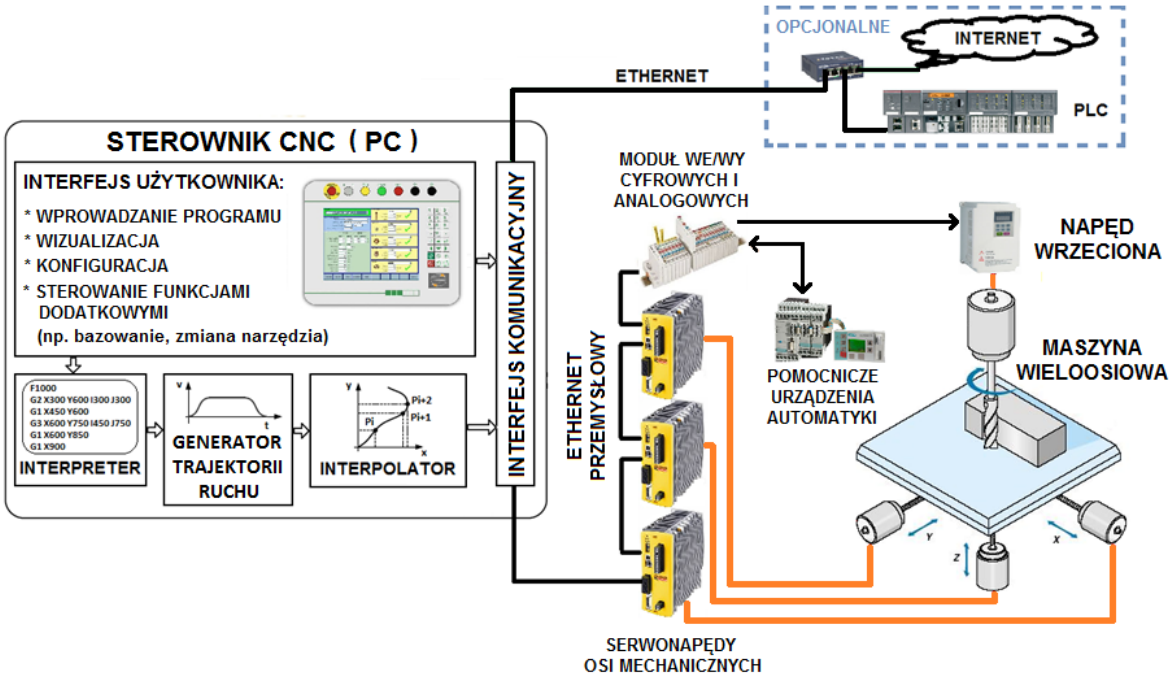
2. Układ sterowania numerycznego maszyn wieloosiowych

Pierwsze układy sterowania maszyn wieloosiowych powstały w Stanach Zjednoczonych na potrzeby przemysłu lotniczego w celu zautomatyzowania wytwarzania elementów samolotów odrzutowych. Zbudowane były na bazie lamp próżniowych i przekaźników, a program obróbki wczytywany był poprzez karty dziurkowane i musiał zostać wcześniej ręcznie napisany przez operatora. Wczesne sterowniki były urządzeniami bardzo kosztownymi i złożonymi, których rozmiar często dorównywał rozmiarowi sterowanych przez nie maszyn. Ponadto zmiana funkcjonalności takiego układu sterowania wymagała przebudowy całego systemu.

Rozwój komputerów umożliwił znaczne zwiększenie możliwości układów sterowania numerycznego maszyn. Obecnie program obróbki generowany jest automatycznie w specjalistycznym oprogramowaniu komputerowym typu CAD/CAM i przesyłany do maszyn w formie cyfrowej. Same sterowniki bazują najczęściej na komputerach PC dzięki czemu modyfikacja funkcjonalności układu sterowania sprowadza się do modyfikacji oprogramowania sterownika. Obecnie światowym trendem w dziedzinie rozwoju układów sterowania numerycznego maszyn jest opracowywanie i implementacja nowych metod i algorytmów minimalizujących błędy odtwarzania zadanego toru ruchu i zwiększających wydajność procesu obróbki.

2.1. Struktura układu sterowania CNC

Zadaniem układu sterowania CNC jest skoordynowane przemieszczenie osi mechanicznych maszyny tak aby złożenie przemieszczeń spowodowało ruch narzędzia po ściśle określonym torze ruchu. Schemat typowego układu sterowania maszyn CNC przedstawiono na rysunku 2.1. Układ sterowania numerycznego zbudowany jest ze sterownika, serwonapędów poszczególnych osi wraz z silnikami oraz samej maszyny. W skład sterownika maszyn CNC wchodzi interfejs użytkownika, umożliwiający wprowadzanie programu obróbki i kontrolę procesu obróbki przez operatora, interpreter programu, generator trajektorii ruchu, interpolator oraz interfejs komunikacji z serwonapędami poszczególnych osi[96]. Często w skład układu sterowania wchodzą pomocnicze układy automatyki pozwalające przykładowo na sterowanie



Rysunek 2.1. Struktura typowego układu sterowania CNC

zmianą narzędzi, urządzeniami bezpieczeństwa, dosyłaniem materiału do obróbki czy też monitorowanie procesu obróbki.

Funkcją interpretera jest zamiana programu technologicznego, zapisanego w odpowiednim formacie, na wewnętrzną reprezentację danego sterownika. Najważniejszą informacją zawartą w programie jest zadany tor ruchu narzędzia. Ponadto w programie często umieszczone są dodatkowe informacje dotyczące parametrów procesu obróbki (prędkość, tolerancje, układ odniesienia, jednostki odległości) i polecenia sterujące pomocniczymi urządzeniami automatyki. Program obróbki z reguły definiowany jest w języku "G-Code"ustandaryzowanym w normie ISO [?]. Program obróbki w standardzie G-Code składa się z wielu odcinków liniowych lub kołowych tworzących jeden tor ruchu. Wielu producentów stosuje niestandardowe rozszerzenia podstawowego języka "G-Code"w celu opisania bardziej złożonych kształtów np. z wykorzystaniem krzywych wielomianowych.

W systemach Rozproszonego Sterowania Numerycznego (ang. Distributed Numerical Control - DNC) gdzie program przesyłany jest do sterownika numerycznego poprzez sieć stosuje się dedykowane protokoły komunikacyjne właściwe dla producenta danego systemu. W ostatnich latach wdraża się nowy standard definicji obróbki - STEP-NC umożliwiający zdefiniowanie znacznie bardziej złożonych operacji niż podstawowy język "G-Code".

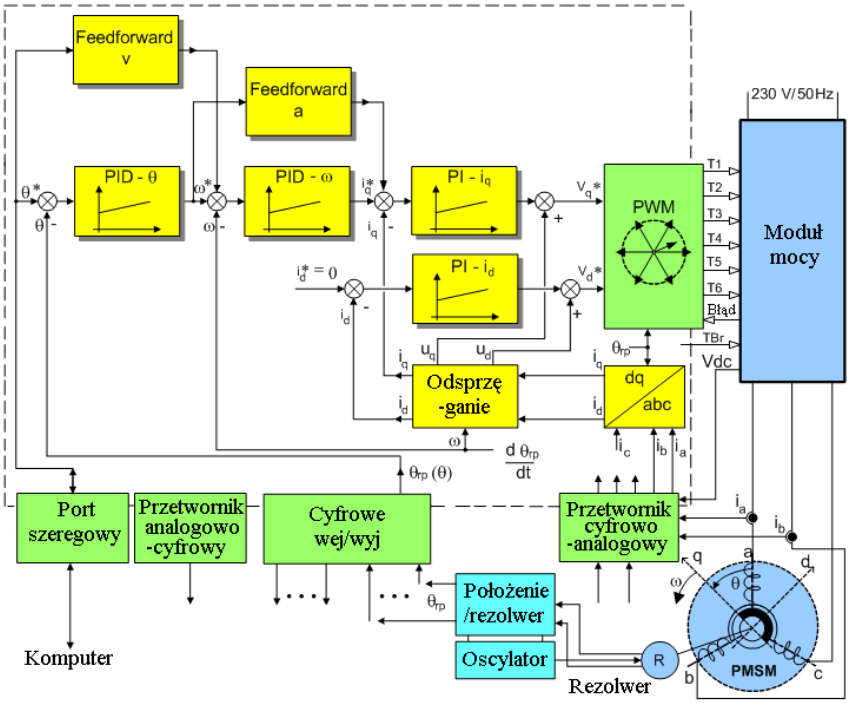
Generator trajektorii ruchu wyznacza profil prędkości z jaką realizowany jest tor ruchu odczytany przez interpreter. Profil prędkości jest to funkcja opisująca zależność prędkości narzędzia stycznej do toru ruchu od czasu lub drogi. Stosowanie profilowania jest niezbędne

gdyż układ mechaniczny jakim jest maszyna wieloosiowa posiada pewną bezwładność. Nie jest możliwe natychmiastowe rozpędzenie bądź zatrzymanie takiego układu ponieważ napędy osi posiadają ograniczoną moc.

Zadaniem interpolatora trajektorii ruchu jest wyznaczanie położeń zadanych do serwonapędów poszczególnych osi na podstawie zadanego toru ruchu oraz wygenerowanej trajektorii ruchu. Z reguły zakłada się, że kolejne punkty zadawane są do serwonapędów w stałych odstępach czasu, przeważnie co 1 milisekundę. Na podstawie profilu prędkości wyznaczane jest przemieszczenie narzędzia w ciągu jednego kroku czasowego a następnie aktualny punkt przesuwany jest wzdłuż zadanego toru ruchu o wyznaczoną odległość. Współrzędne nowego punktu leżącego na zadanym torze ruchu przesyłane są do serwonapędów odpowiednich osi mechanicznych jako wartości zadane położenia.

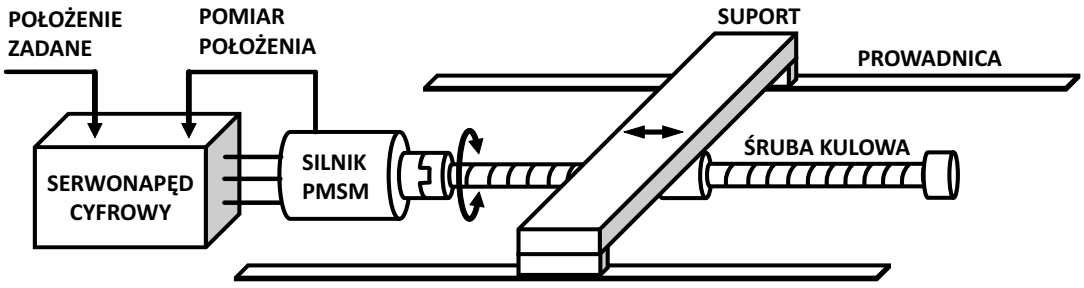
Istotnym elementem układu sterowania maszyn jest interfejs komunikacyjny sterownika CNC z serwonapędami poszczególnych osi. W starszych układach sterowania wartości położenia lub prędkości zadawane były poprzez interfejs analogowy w postaci napięcia. Innym nadal często spotykanym rozwiązaniem jest zadawanie przyrostów położeń poprzez interfejs step/dir znany z napędów silników krokowych. W nowoczesnych układach sterowania stosuje się magistrale szeregowe, dzięki którym można wymieniać dane w obu kierunkach pomiędzy serwonapędami a sterownikiem. W ostanich latach zaczęto stosować magistrale przemysłowe oparte na standardzie Ethernet takie jak np. Ethernet Powerlink lub Ethercat, które umożliwiają przesyłanie i odbieranie dużych ilości danych z dużą prędkością i dużym determinizmem czasowym. Sterownik CNC bazujący na komputerze PC z zaimplementowanym interfejsem komunikacyjnym Ethernet Powerlink, który posłużył jako stanowisko do badań doświadczalnych przedstawionych w niniejszej rozprawie został opracowany przez autora i opisany w [33].

W większości nowoczesnych maszyn osie mechaniczne napędzane są przez serwosilniki elektryczne prądu przemiennego z magnesami trwałymi (ang. Permanent Magnet Synchronous Motor - PMSM) pracujące pod kontrolą cyfrowych serwonapędów sterowanych poprzez sieć z nadrzędnego sterownika CNC. Strukturę układu regulacji typowego serwonapędu przedstawiono na rysunku 2.2. Z reguły stosuje się kaskadowy układ regulacji z regulatorami położenia, prędkości i prądu typu P lub PI ze sprzężeniami w przód (ang. feedforward) od prędkości i przyspieszenia. W literaturze opisywane są bardziej zaawansowane układy regulacji m.in regulatory ślizgowe, ze sprzężeniem od stanu, skrośne, neuronowe, rozmyte i inne. W praktyce, w komercyjnie dostępnych serwonapędach, takie rozwiązania nie są stosowane ze względu na ich dużą złożoność i często skomplikowaną procedurę strojenia. W większości przypadków do napędu osi mechanicznych maszyn CNC stosuje się silniki obrotowe. Konieczna jest więc zamiana ruchu obrotowego silników na ruch postępowy suportu za pomocą odpowiedniej przekładni. Najczęściej stosuje się przekładnie śrubowe toczne (śruby



Rysunek 2.2. Struktura serwonapędu współpracującego z silnikami PMSM

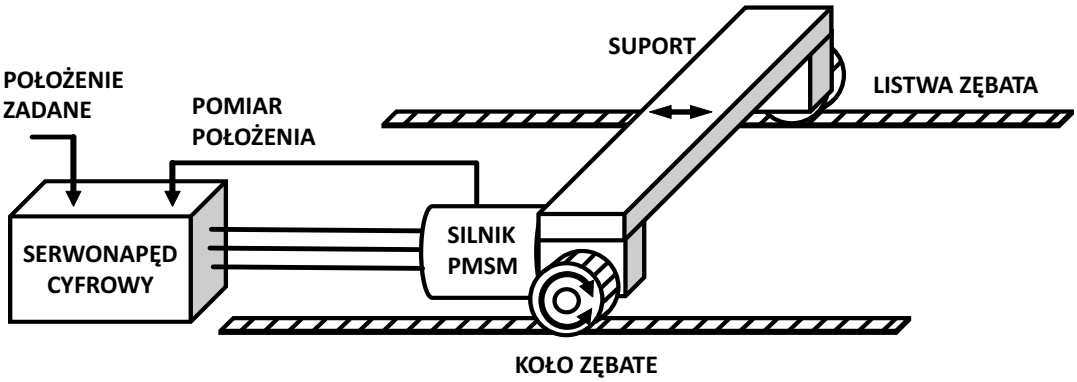
kulowe) lub układ listwa zębata - koło zębate. Połączenie silnika z przekładnią odbywa się poprzez sprzęgło lub inną przekładnię (np. pasową). Projektując układ posuwu maszyny dąży się do zapewnienia dużej sztywności, niewielkich oporów ruchu i likwidacji luzów. Schemat typowych układów posuwu osi mechanicznej maszyny wieloosiowej, wykorzystujących śrubę kulową i układ listwa zębata - koło zębate przedstawiono na rysunkach 2.3 i 2.4.



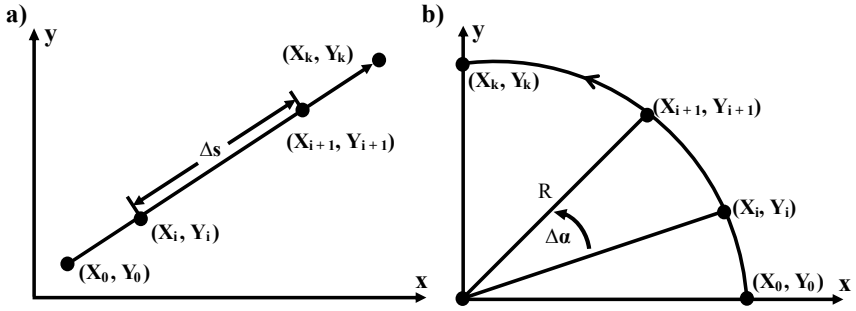
Rysunek 2.3. Schemat napędu posuwu osi mechanicznej wykorzystującego śrubę kulową.

2.2. Interpolacja toru ruchu opisanego za pomocą krzywych NURBS

W większości układów sterowania maszyn CNC do opisu toru ruchu stosuje się linie proste bądź łuki. Jest to podyktowane prostotą algorytmu interpolacji liniowej i kołowej co miało istotne znaczenie w czasach gdy sterowniki CNC nie dysponowały dużą mocą obliczeniową. Na rysunku 2.5 przedstawiono schemat działania algorytmów interpolacji liniowej i kołowej.



Rysunek 2.4. Schemat napędu posuwu osi mechanicznej wykorzystującego układ listwa zębata - koło zębate.



Rysunek 2.5. Interpolacja liniowa (a) i kołowa (b)

Przemieszczenie liniowe Δs bądź kątowe $\Delta\alpha$ wyznaczane jest na podstawie chwilowej prędkości wyznaczonej z wygenerowanego uprzednio profilu prędkości:

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

gdzie v - chwilowa prędkość posuwu, Δt - czas pomiędzy zadaniem kolejnych wartości położenia, R - promień okręgu.

W przypadku odcinka liniowego na płaszczyźnie, kolejne punkty wyznaczane są ze wzoru:

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

Dla odcinków łuków interpolacja na płaszyźnie realizowana jest z wykorzystaniem wzoru:

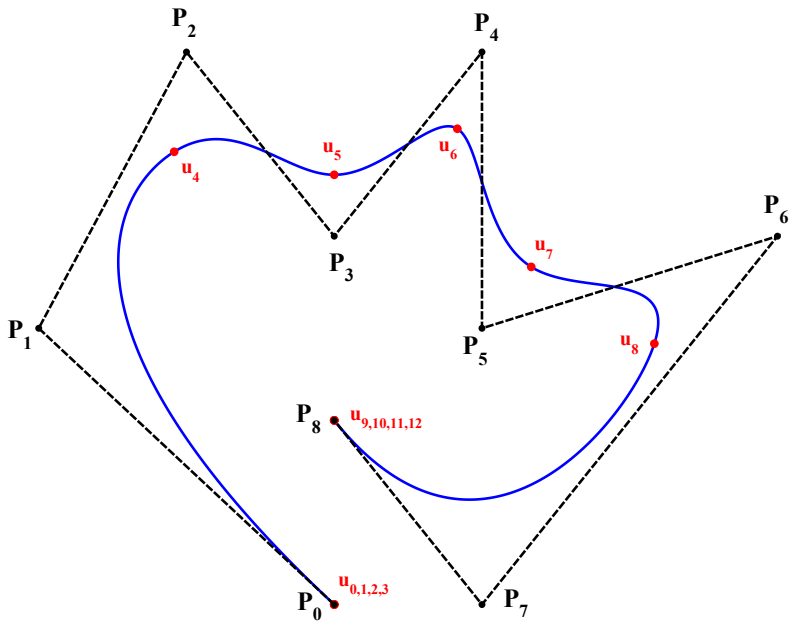
$$
\begin{aligned}
X_{i+1} &= \cos(\Delta\alpha)X_i - \sin(\Delta\alpha)Y_i \\
Y_{i+1} &= \cos(\Delta\alpha)Y_i + \sin(\Delta\alpha)X_i
\end{aligned}
\tag{2.3}
$$

O ile algorytm interpolacji linii prostych łatwo skaluje się do przestrzeni trójwymiarowej to algorytm interpolacji kołowej w trzech wymiarach jest zdecydowanie bardziej złożony[68]. Obydwie metody umożliwiają odwzorowanie tylko prostych kształtów. W praktyce złożone kształty aproksymuje się dużą liczbą krótkich segmentów liniowych. Tor ruchu opisany w ten sposób posiada nieciągłą pierwszą pochodną co prowadzi do nagłych zmian prędkości na styku segmentów a tym samym do pogorszenia jakości obróbki. Duża ilość segmentów liniowych konieczna do opisania złożonych kształtów prowadzi do znacznego rozmiaru programu. Niektóre sterowniki dodatkowo wygładzają tak zadany tor ruchu. Odbywa się to jednak kosztem zmiany pierwotnego kształtu zadanego toru ruchu i wymaga dodatkowego nakładu obliczeń.

W procesie tworzenia programu obróbki wykorzystuje się programy typu CAD (ang. Computer Asissted Design), w których kształty reprezentowane są z reguły za pomocą krzywych wielomianowych. Dzięki temu możliwa jest reprezentacja bardzo złożonych kształtów. Niektóre zaawansowane sterowniki CNC dysponują możliwością bezpośredniej interpolacji takich krzywych. Ma to szereg zalet w stosunku do klasycznej metody aproksymacji z wykorzystaniem segmentów liniowych. Tor ruchu jest gładki co przekłada się na mniejsze fluktuacje prędkości i przyspieszenia a tym samym prowadzi do mniejszych błędów odwzorowania toru ruchu. Unika się również błędów związanych z aproksymacją toru ruchu segmentami liniowymi i ponownym wygładzaniem w sterowniku. Program jest o wiele bardziej zwarty ponieważ jedna krzywa może opisać kształt wymagający aproksymacji wieloma krótkimi segmentami liniowymi. Interpolacja krzywych nie sprawia problemu zarówno w dwóch jak i trzech wymiarach. Wadą jest większy nakład obliczeń konieczny do zrealizowania algorytmów interpolacji krzywych wielomianowych co przy obecnie stosowanych, szybkich mikroprocesorach przestaje mieć znaczenie.

Najszerzej stosowanym rodzajem krzywych w programach CAD są krzywe B-sklejane (ang. Basis Spline lub B-Spline)[77]. Krzywe B-Spline są parametrycznymi wielomianowymi krzywymi sklejanymi. Parametr krzywej u jednoznacznie określa lokalizację punktu na krzywej. Z reguły przyjmuje się, że u należy do (0,1) gdzie 0 oznacza początek a 1 koniec krzywej. W celu zdefiniowania krzywej rzędu p należy zdefiniować n punktów kontrolnych P_i (zwanych również punktami de Boor'a) oraz $p + n + 1$ punktów węzłowych u_k. Punkty węzłowe stanowią niemalejący ciąg wartości parametru krzywej u określający miejsce połączenia segmentów wielomianowych tworzących krzywą. Zwielokrotnienie węzłów powoduje obniżenie stopnia ciągłości krzywej, która normalnie wykazuje ciągłość stopnia

$p-1$ (tzn. $p-1$ pochodna krzywej po jej parametrze jest ciągła). Zwielokrotnienie węzłów $p+1$ razy powoduje, że krzywa przechodzi przez dany punkt kontrolny. Właściwość tą często wykorzystuje się do wymuszenia początku i końca krzywej w odpowiednio pierwszym i ostatnim punkcie kontrolnym. Punkty kontrolne (będące skalarem bądź wektorem zależnie od wymiaru przestrzeni w której określona jest krzywa) definiują otoczkę wypukłą (ang. convex hull), w której zawiera się krzywa. Ważną własnością krzywych B-sklejanych jest lokalność oddziaływania punktów kontrolnych na kształt krzywej. Pozwala to na intuicyjną modyfikację kształtu przez przesuwanie punktów kontrolnych co jest jedną z przyczyn popularności krzywych b-sklejanych w oprogramowaniu CAD. Przykład krzywej b-sklejanej przedstawiono na rysunku 2.6.



P_i={(0;0), (-2;3), (-1;6), (0;4), (1;6), (1;3), (3;4), (1;0), (0;2)}

u_k={0, 0, 0, 0, 0.1667, 0.3333, 0.5, 0.6667, 0.8333, 1, 1, 1, 1}

Rysunek 2.6. Krzywa B-sklejana stopnia 3 (kolor niebieski) wraz z punktami kontrolnymi (kolor czarny) i punktami węzłowymi (kolor czerwony).

Matematycznie krzywą b-sklejaną można opisać wzorem:

$$\mathbf{C}(\mathbf{u}) = \sum_{i=0}^{n} N_{i,p}(u)\mathbf{P_i} \tag{2.4}$$

gdzie: $N_{i,p}(u)$ - wartość elementarnej funkcji bazowej wielomianu sklejanego dla zadanego parametru u.

Funkcja bazowa krzywej b-sklejanej określona jest rekurencyjnym wzorem Mansfielda - de Boor'a - Cox'a:

$$N_i^0(u) = \begin{cases} 1 & \text{dla } u \in [u_i, u_{i+1}) \\ 0 & \text{w przeciwnym razie} \end{cases}$$
$$N_i^n(u) = \frac{u - u_i}{u_{i+n} - u_i} N_i^{n-1}(u) + \frac{u_{i+n+1} - u}{u_{i+n+1} - u_{i+1}} N_{i+1}^{n-1}(u) \quad \text{dla } n > 0 \tag{2.5}$$

Wadą powyższego wzoru jest jego numeryczna niestabilność tzn. w pewnych warunkach może dojść do występowania znacznych błędów obliczeń wynikających ze skończonej precyzji liczb zmiennoprzecinkowych. Wady tej nie posiada algorytm De Boor'a, powszechnie stosowany w praktyce do wyznaczania współrzędnych punktów leżących na krzywej. Algorytm polega na wstawieniu węzła w danym punkcie p razy za każdym razem modyfikując i dodając punkty kontrolne tak aby kształt krzywej nie ulegał zmianie. W celu wstawienie węzła należy w pierwszej kolejności znaleźć przedział węzłów $[u_k, u_{k+1})$, w którym mieści się punkt, w którym ma być wstawiony nowy węzeł. W obliczeniach wykorzystane jest $p + 1$ punktów kontrolnych zaczynając od punktu P_{k-p}. Pierwszy i ostatni punkt pozostają bez zmian natomiast $p - 1$ punktów kontrolnych pomiędzy nimi zastępowane jest p nowymi punktami zgodnie ze wzorem:

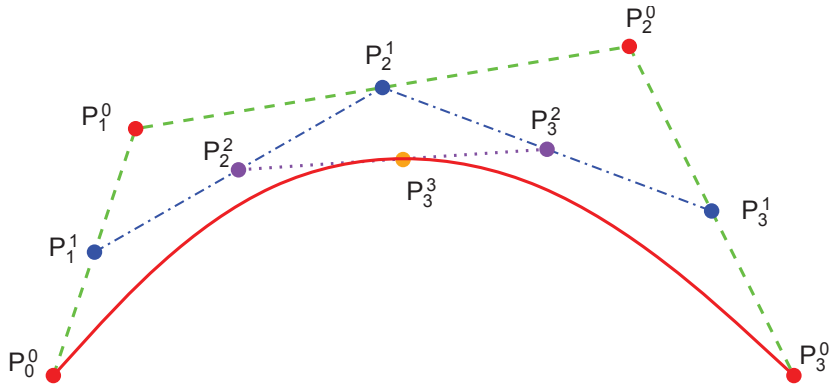
$$\begin{aligned} &\mathbf{P_i^r} = (1 - a_{i,r})\,\mathbf{P_{i-1}^{r-1}} + a_{i,r}\mathbf{P_i^{r-1}} \\ &a_{i,r} = \frac{u - u_i}{u_{i+p-r+1} - u_i} \\ &\text{dla } i = [k - p + r, k] \end{aligned} \tag{2.6}$$

W powyższym wzorze r oznacza indeks kolejnych generacji punktów kontrolnych powstałych przez wielokrotne wstawianie węzła w tym samym punkcie. Dla $r = p$ czyli po p-krotnym wstawieniu węzła wynikiem jest pojedynczy nowy punkt kontrolny, będący poszukiwanym punktem na krzywej odpowiadającym zadanej wartości parametru u. Algorytm ten pozwala również wyznaczać pochodne krzywej B-Sklejanej względem parametru u, wykorzystując fakt, że pochodna jest krzywą B-Sklejaną niższego rzędu. Punkty kontrolne pochodnej krzywej b-sklejanej wyznaczane są ze wzoru:

$$\mathbf{Q_i} = \left(\frac{p}{u_{p+i+1} - u_{i+1}} \right) (\mathbf{P_{i+1}} - \mathbf{P_i}) \tag{2.7}$$

Powyższy wzór można stosować rekurencyjnie do wyznaczania pochodnych wyższych rzędów.

Algorytm De Boor'a został przedstawiony na rysunku 2.7.



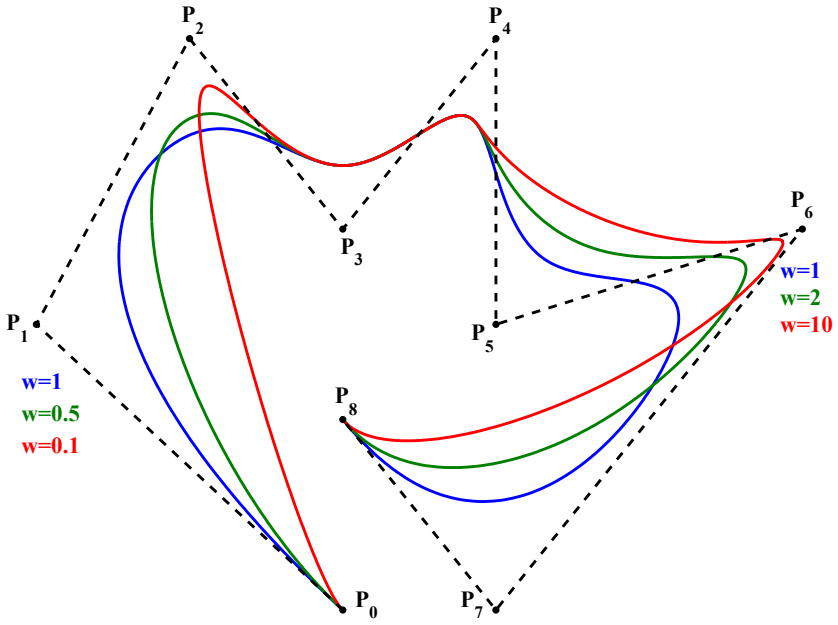
Rysunek 2.7. Algorytm De Boora wyznaczania współrzędnych punktu leżącego na krzywej B-Sklejanej rzędu trzeciego.

Wadą krzywych b-sklejanych jest niemożność dokładnego opisania dowolnych kształtów w szczególności krzywych stożkowych. Wady tej pozbawione są wymierne nierównomierne krzywe b-sklejane, określane powszechnie angielskim skrótem NURBS (ang. Non-Uniform Rational B-Spline), będące generalizacją krzywych B-Sklejanych. Krzywe NURSB zdefiniowane są wzorem:

$$\mathbf{C}(\mathbf{u}) = \frac{\sum_{i=0}^{n} N_{i,p}(u) w_i \mathbf{P_i}}{\sum_{i=0}^{n} N_{i,p}(u) w_i} \tag{2.8}$$

Krzywą NURBS definiuje się, podobnie jak krzywą B-Sklejaną, za pomocą punktów kontrolnych. Różnica polega na przypisaniu każdemu punktowi kontrolnemu, dodatkowego parametru zwanego wagą. Waga określa jak blisko danego punktu znajduje się krzywa. Wartość 0 oznacza, że dany punkt nie wpływa w ogóle na kształt krzywej. Wraz ze zwiększaniem wartości krzywa przechodzi coraz bliżej punktu kontrolnego. Jeżeli wszystkie wagi mają wartość 1 to krzywa NURBS jest równoważna krzywej B-Sklejanej. Wpływ wag na kształt krzywych NURBS pokazany został na rysunku 2.8 na przykładzie krzywej zdefiniowanej przez te same punkty kontrolne i węzły co krzywa na rysunku 2.6.

Wartość krzywej NURBS wyznacza się algorytmem De Boor'a przekształcając współrzędne kartezjańskie punktów kontrolnych do współrzędnych jednorodnych (ang. homogenous coordinates). Przekształcenie polega na pomnożeniu współrzędnych kartezjańskich przez wagę i dodaniu wagi jako dodatkowej współrzędnej. Dla krzywej trójwymiarowej punkt kontrolny we współrzędnych jednorodnych ma cztery współrzędne w postaci:



Rysunek 2.8. Wpływ różnych wartości wagowych na kształt krzywej NURBS. Punkty kontrolne i węzły identyczne jak na rysunku 2.6

$$\mathbf{H_i} = \begin{bmatrix} x_i w_i \\ y_i w_i \\ z_i w_i \\ w_i \end{bmatrix} \tag{2.9}$$

Wynik otrzymany we współrzędnych jednorodnych należy przekształcić do współrzędnych kartezjańskich poprzez podzielenie współrzędnych przez wagę będącą ostatnią współrzędną. W przypadku pochodnych krzywej NURBS przekształcenie jest bardziej złożone ponieważ NURBS jest funkcją wymierną. Należy skorzystać z reguły łańcuchowej otrzymując ogólne wyrażenie w postaci[77]:

$$\mathbf{C^{(k)}}(\mathbf{u}) = \frac{\mathbf{H_c^{(k)}}(\mathbf{u}) - \sum_{i=1}^{k} \binom{k}{i} H_w^{(i)}(u) \mathbf{H_c^{(i)}}(\mathbf{u})}{H_w^{(0)}(u)} \tag{2.10}$$

gdzie:

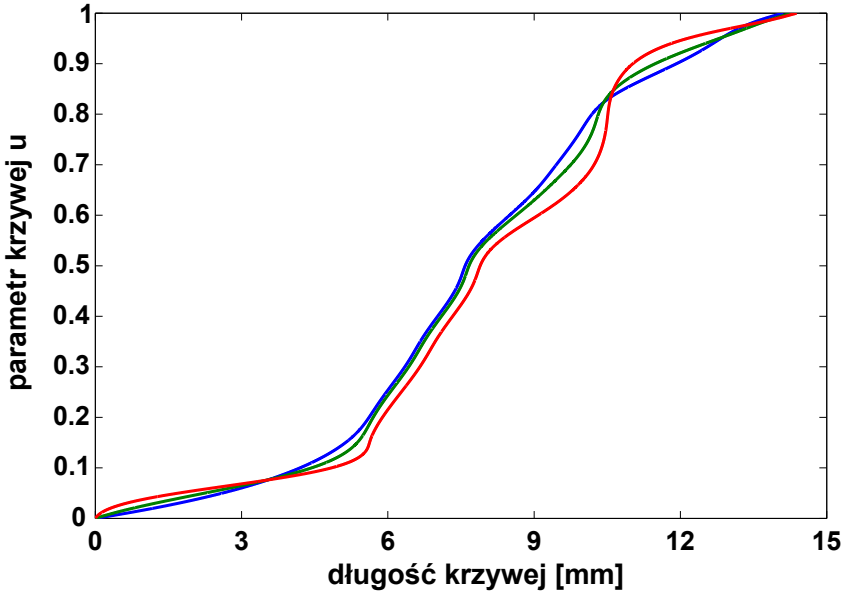
$\mathbf{C^{(k)}}(\mathbf{u})$ - k-ta pochodna krzywej NURBS we współrzędnych kartezjańskich,

$\mathbf{H_c^{(k)}}(\mathbf{u})$ - k-ta pochodna krzywej NURBS we współrzędnych jednorodnych bez ostatniej współrzędnej (wagi),

$H_w^{(k)}(u)$ - ostatnia współrzędna (waga) k-tej pochodnej krzywej NURBS we współrzędnych jednorodnych

Możliwość opisania bardzo złożonych kształtów czyni krzywe NURBS idealnym narzędziem do definiowania toru ruchu narzędzia w układach sterowania numerycznego maszyn wieloosiowych. Złożone kształty (w tym krzywe stożkowe) mogą być opisane dokładnie. Eliminuje to potrzebę aproksymacji toru ruchu dużą liczbą segmentów liniowych. Ponieważ krzywe NURBS charakteryzują się ciągłością do $p-1$ pochodnej ruch po torze opisanym taką krzywą jest płynny to znaczy, że nie występują nagłe zmiany prędkości. Skutkuje to większymi dopuszczalnymi prędkościami posuwu niż w przypadku aproksymacji segmentami liniowymi.

Zadanie interpolacji krzywych NURBS w układach sterowania maszyn jest bardziej złożone niż interpolacja liniowa lub kołowa. Głównym problemem jest wyznaczenie zmiany wartości parametru krzywej Δu odpowiadające przyrostowi drogi $\Delta s = v\Delta t$. Zależność pomiędzy tymi wielkościami ma charakter nieliniowy (rysunek 2.9).



Rysunek 2.9. Nieliniowa zależność parametru u krzywej od długości krzywej dla trzech krzywych NURBS przedstawionych na rysunku 2.8

Wyrażenie zależności $u = f(s)$ dla krzywych NURBS w formie analitycznej nie jest możliwe [36]. W celu przeprowadzenia procesu interpolacji należy więc tą zależność aproksymować. Działanie algorytmu interpolacji polega na rozwiązaniu w każdym kroku czasowym Δt interpolatora, następującego równania:

$$\int_{u_i}^{u_{i+1}} \frac{\sigma(u)}{V} du = \Delta t \tag{2.11}$$

gdzie: V - prędkość styczna do toru ruchu, $\sigma(u)$ - prędkość parametryczna czyli pochodna położenia na krzywej względem parametru krzywej.

$$\sigma(u) = \sqrt{\frac{dx}{du}^2 + \frac{dy}{du}^2 + \frac{dz}{du}^2} \tag{2.12}$$

Najczęściej stosowaną metodą rozwiązania powyżego problemu jest rozwinięcie w szereg Taylora funkcji $u(t_i)$ po raz pierwszy zaproponowane przez Yang'a i Kong'a [105]. Rozwinięcie ma w ogólnej formie postać[37]:

$$u_{i+1} = u_i + \dot{u}(t_i)\Delta t + \frac{\ddot{u}(t_i)}{2!}(\Delta t)^2 + \cdots + \frac{u^{(n)}(t_i)}{n!}(\Delta t)^n \tag{2.13}$$

Pochodne parametru u względem czasu wyrażają się wzorami:

$$\begin{aligned}
\dot{u}(t) &= \frac{V}{\sigma} \\
\ddot{u}(t) &= \frac{\sigma V' - \sigma' V}{\sigma^2}\dot{u}(t) \\
\dddot{u}(t) &= \frac{\sigma V' - 3\sigma' V}{\sigma^2}\ddot{u}(t) + \frac{\sigma V'' - \sigma'' V}{\sigma^2}\dot{u}(t)^2
\end{aligned} \tag{2.14}$$

Pochodne prędkości posuwu (stycznej do toru ruchu) względem parametru u zawarte we wzorach 2.14 zdefiniowane są następująco:

$$\begin{aligned}
V' &= \frac{dV}{du} = \frac{\sigma}{V}A \\
V'' &= \frac{d^2V}{du^2} = \frac{\sigma'}{V}A - \frac{\sigma^2}{V^3}A^2 + \frac{\sigma^2}{V^2}J
\end{aligned} \tag{2.15}$$

gdzie: A - przyspieszenie styczne do toru ruchu, J - zryw styczny do toru ruchu.

Podstawiając 2.15 do 2.14 otrzymuje się zmodyfikowane wyrażenia na pochodną parametru względem czasu. Wzory te nie zawierają prędkości stycznej w mianowniku co jest istotne biorąc pod uwagę, że ruch rozpoczyna się i kończy prędkością równą 0.

$$\begin{aligned}
\dot{u}(t) &= \frac{V}{\sigma} \\
\ddot{u}(t) &= \frac{A}{\sigma} - \frac{\sigma' V}{\sigma^2}\dot{u}(t) \\
\dddot{u}(t) &= \frac{J}{\sigma} - \frac{3\sigma' V}{\sigma^2}\ddot{u}(t) - \frac{\sigma'' V}{\sigma^2}\dot{u}(t)^2
\end{aligned} \tag{2.16}$$

Pochodne prędkości parametrycznej σ względem parametru krzywej wyrażone są następującymi równaniami:

$$\begin{aligned}
\frac{d\sigma}{du} = \sigma' &= \frac{\boldsymbol{C'(u)} \cdot \boldsymbol{C''(u)}}{\sigma} \\
\frac{d^2\sigma}{du^2} = \sigma'' &= \frac{\boldsymbol{C'}(u) \cdot \boldsymbol{C'''(u)} + |\boldsymbol{C''(u)}|^2 - \sigma'^2}{\sigma}
\end{aligned} \tag{2.17}$$

gdzie: $\boldsymbol{C'(u), C''(u), C'''(u)}$ - pierwsza, druga i trzecia pochodna krzywej względem jej parametru.

Powyższe wzory umożliwiają przeprowadzenie procesu interpolacji krzywej w zależności od zmiennej prędkości posuwu wykorzystując pierwszą, drugą i trzecią pochodną krzywej. Umożliwia to aproksymację funkcji $u(s)$ szeregiem Taylora rzędu trzeciego. Możliwe jest zastosowanie aproksymacji wyższych rzędów jednakże ma to sens jedynie jeżeli krzywa NURBS ma rząd większy niż 3. Krzywe rzędu wyższego niż 3 stosuje się rzadko gdyż nie zapewniają istotnie lepszej jakości odwzorowania pożądanego kształtu powodując jednocześnie wzrost ilości parametrów i nakładów obliczeń. Często aby ograniczyć liczbę koniecznych obliczeń ogranicza się rząd szeregu Taylora do 2 dzięki czemu wyznaczanie trzeciej pochodnej nie jest konieczne. Powoduje to większe błędy aproksymacji rozwiązania równania 2.11 i tym samym większe fluktuacje prędkości. Aproksymacja rzędu pierwszego jest najmniej wymagająca obliczeniowa jednak znaczne błędy aproksymacji powodują, że jest rzadko stosowana.

Alternatywną metodą rozwiązania problemu interpolacji krzywych NURBS jest zastosowanie metody "predyktor-korektor"[15, 13]. Metoda ta polega na wyznaczeniu zgrubnej aproksymacji wymaganego przyrostu parametru krzywej a następnie na iteracyjnej korekcji jego wartości zapewniającej minimalizację błędu aproksymacji. Metoda ta pozwala uzyskać bardzo mały błąd aproksymacji i nie wymaga wyznaczania pochodnych krzywej. Ze względu na iteracyjny charakter czas obliczeń jest o wiele dłuższy niż w metodzie szeregu Taylora.

Powyższe metody aproksymują lokalnie funkcję $u(s)$ w każdym kroku interpolacji. Możliwa jest również globalna aproksymacja tej funkcji poprzez dopasowanie tzw. wielomianu korekcji prędkości (ang. feed correction polynomial) [42, 103, 32]. Wyznaczenie parametrów wielomianu korekcji prędkości (np. metodą najmniejszych kwadratów) pozwala na sprowadzenie procesu interpolacji do wyznaczenia wartości wielomianu aproksymującego. Wadą tego rozwiązania jest konieczność aproksymacji wielomianu dla każdego toru ruchu. Ponieważ dokładna aproksymacja wymaga zastosowania wielomianów sklejanych wysokich rzędów (np. 7 [42]) ilość dodatkowych parametrów wymaganych do opisu zadanej trajektorii ruchu może być znaczna.

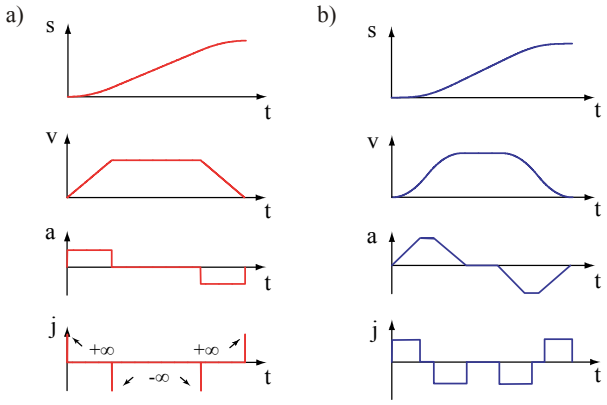
Ponieważ do opisu toru ruchu rzadko wykorzystuje się krzywe NURBS rzędu wyższego niż trzeci aproksymacja szeregiem Taylora drugiego lub trzeciego rzędu zapewnia w większości przypadków wystarczająco mały błąd aproksymacji funkcji $u(s)$. Dużą zaletą jest zamknięta forma metody co pozwala na uzyskanie rozwiązania w jednym kroku. Z tych względów metodę interpolacji krzywych NURBS wykorzystującą w szereg Taylora stosuje się najczęściej.

2.3. Generacja trajektorii ruchu w układach CNC

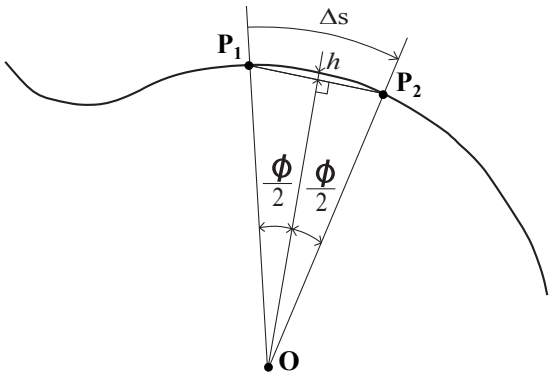
Najprostszą metodą profilowania prędkości jest prostokątne profilowanie przyspieszenia. W takim przypadku przyspieszenie zmienia się skokowo pomiędzy skrajnymi wartościami a profil prędkości ma kształt trapezoidalny. Algorytm prostokątnego profilowania przyspieszenia jest bardzo prosty i nie wymaga wielu obliczeń więc jest często stosowany w wielu układach sterowania CNC. W najprostszym wariancie generatora trajektorii ruchu przyjmuje się, że narzędzie porusza się ze stałą, zdefiniowaną przez programistę, prędkością posuwu w trakcie realizacji pojedynczego segmentu programu (linii prostej lub łuku). Na początku i końcu segmentu realizowane jest odpowiednio rozpędzanie i hamowanie zgodnie z maksymalnym przyspieszeniem stycznym przyjętym globalnie dla całego toru ruchu. Metoda ta sprawdza się, jeżeli program składa się z długich prostych odcinków. Często jednak skomplikowane kształty reprezentowane są w języku "G-Code"w formie wielu bardzo krótkich odcinków liniowych. W takim przypadku wyżej opisany algorytm wykonywałby szereg krótkich ruchów nigdy nie osiągając zadanej prędkości maksymalnej. Ponadto ciągłe hamowanie i rozpędzanie maszyny z prostokątnym profilowaniem przyspieszenia prowadzi do wibracji układu mechanicznego. Nadmierne wibracje prowadzić mogą do występowania rezonansów mechanicznych w układzie posuwu maszyny co przekłada się na niską jakość obróbki i zwiększone zużycie mechaniczne maszyny. W takim przypadku w procesie generacji trajektorii ruchu analizuje się z wyprzedzeniem wiele segmentów programu jednocześnie, scalając je w jeden płynny ruch z fazami rozpędzania i hamowania.

W bardziej zaawansowanych układach sterowania numerycznego maszyn ogranicza się wartość pochodnej przyspieszenia czyli zryw. Profilowanie z ograniczeniem zrywu charakteryzuje się trapezoidalnym profilem przyspieszenia i profilem prędkości opisanym wielomianem sklejanym drugiego rzędu. Ograniczenie zrywu powoduje ograniczenie w sygnale zadanym do napędów poszczególnych osi składowych wysokiej częstotliwości co przekłada się na zmniejszenie drgań w układzie posuwu a co za tym idzie mniejsze błędy odtwarzania zadanego toru ruchu i mniejsze zużycie mechaniczne układu posuwu. Wadą profilowania prędkości z ograniczeniem zrywu jest znacznie bardziej złożony algorytm generacji trajektorii ruchu. Profile prędkości, przyspieszenia i zrywu dla prostokątnego i trapezoidalnego profilowania przyspieszenia przedstawiono na rysunku 2.10.

Kolejnym problemem, który należy uwzględnić w procesie generacji trajektorii ruchu jest wpływ zadanej prędkości na błąd interpolacji krzywej wynikający z dyskretnego charakteru procesu interpolacji. Jako błąd interpolacji definiuje się błąd cięciwowy (ang. chord error) [96] stanowiący odległość pomiędzy linią łączącą dwa kolejne interpolowane punkty a łukiem przechodzącym przez te dwa punkty o środku pokrywającym się z lokalnym środkiem krzywizny krzywej NURBS. Definicję błędu cięciwowego ilustruje rysunek 2.11.



Rysunek 2.10. Profile położenia, prędkości, przyspieszenia i zrywu dla a) prostokątnego i b)trapezoidalnego profilowania przyspieszenia



Rysunek 2.11. Błąd cięciwowy h stanowiący odległość między linią prostą łączącą dwa interpolowane punkty P_1 i P_2 a łukiem aproksymującym krzywą NURBS o środku w punkcie O.

Środek krzywizny i każdy z interpolowanych punktów łączy promień krzywizny będący odwrotnością krzywizny. Krzywiznę krzywej NURBS dla danej wartości parametru definiuje się jako:

$$\kappa(u) = \frac{||\boldsymbol{C'(u)} \times \boldsymbol{C''(u)}||}{||\boldsymbol{C'(u)}||^3} \tag{2.18}$$

Aby ograniczyć wartość błędu cięciwowego ogranicza się prędkości posuwu zgodnie ze wzorem:

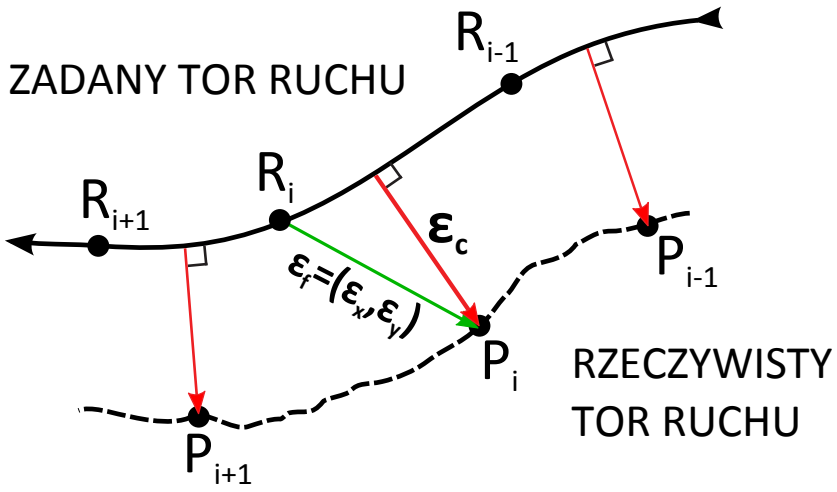
$$F_{max}(u) = \frac{2}{\Delta T}\sqrt{\frac{2\epsilon_{chord}}{\kappa(u)}} \tag{2.19}$$

gdzie: ϵ_{chord} maksymalny dopuszczalny błąd cięciwowy.

Samo ograniczenie błędu cięciwowego nie gwarantuje, że przyspieszenie i zryw będą się mieściły w dopuszczalnych granicach dlatego konieczne jest dodatkowe profilowanie trajektorii uwzględniające te ograniczenia.

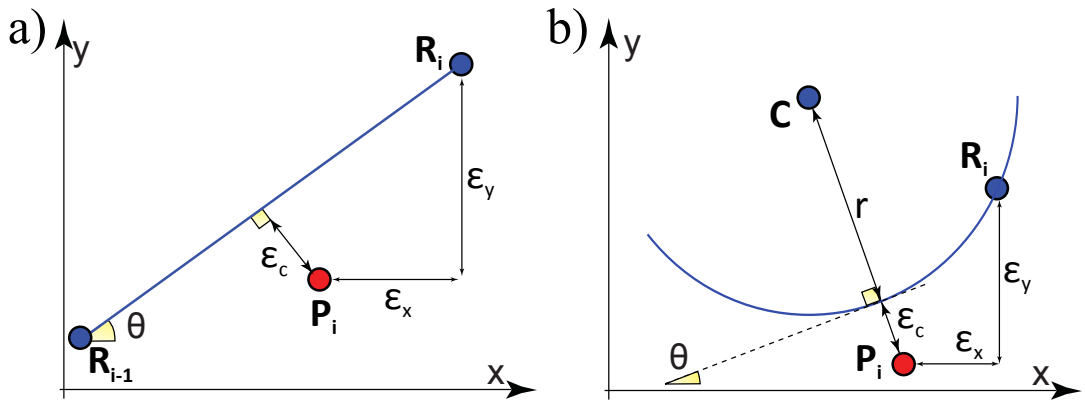
2.4. Błędy odtwarzania toru ruchu w układach sterowania maszyn

Zadany tor ruchu opisany jest idealnymi obiektami matematycznymi w formie prostych, łuków lub krzywych. Układ sterowania CNC może odwzorować zadane kształty z ograniczoną dokładnością. Utrata dokładności następuje już na etapie interpolacji na skutek dyskretyzacji toru ruchu i aproksymacji zależności $u(s)$ w przypadku toru ruchu opisanego krzywymi NURBS. Nadążne układy regulacji w serwonapędach działają na zasadzie kompensacji uchybu czyli różnicy między wartością zadaną a wartością rzeczywistą położenia osi. Układ regulacji reaguje więc dopiero gdy błąd wystąpi. Same regulatory są z reguły regulatorami liniowymi, nie mogą więc skompensować nieliniowości rzeczywistego układu posuwu. Definiuje się dwa rodzaje błędów określających jak dobrze układ sterowania numerycznego maszyn odwzorowuje zadany tor ruchu. Błąd nadążania (ang. following error) jest różnicą pomiędzy położeniem rzeczywistym osi a położeniem zadanym w danej chwili czasu. Błąd nadążania jest więc złożeniem uchybów poszczególnych osi i nie jest w żaden sposób powiązany z zadanym torem ruchu narzędzia. Najmniejszą odległość pomiędzy rzeczywistym położeniem osi a zadanym torem ruchu definiuje się jako błąd konturu (ang. contour error) [52]. Błąd konturu jest lepszym kryterium oceny jakości odwzorowania toru ruchu niż błąd nadążania gdyż określa faktyczną geometryczną różnicę pomiędzy zadanym i rzeczywistym torem ruchu. Definicję błędu konturu i błędu nadążania ilustruje rysunek 2.12.



Rysunek 2.12. Błąd nadążania ϵ_f, będący złożeniem uchybów poszczególnych osi (ϵ_x, ϵ_y), oraz błąd konturu ϵ_c stanowiący najkrótszą odległość pomiędzy zadanym i rzeczywistym torem ruchu. Punkty R_{i-1}, R_i, R_{i+1} oznaczają położenia zadane a P_{i-1}, P_i, P_{i+1} odpowiadające im położenia rzeczywiste.

Błąd konturu jest funkcją błędu nadążania (uchybów poszczególnych osi) oraz geometrii zadanego toru ruchu. W przypadku toru ruchu będącego linią prostą błąd konturu można wyznaczyć jako najkrótszą odległość pomiędzy linią a punktem (rys. 2.13a) Dla toru ruchu



Rysunek 2.13. Błąd konturu dla toru ruchu w postaci a) linii prostej i b) łuku

leżącego na płaszczyźnie można posłużyć się wzorem:

$$\epsilon_c = -\sin(\theta)\epsilon_x + \cos(\theta)\epsilon_y \tag{2.20}$$

Dla przypadku trójwymiarowego można wykorzystać wzór na odległość punktu od linii prostej wyznaczonej dwoma punktami:

$$\epsilon_c = \frac{||(\boldsymbol{P_i} - \boldsymbol{R_{i-1}}) \times (\boldsymbol{P_i} - \boldsymbol{R_i})||}{||\boldsymbol{R_i} - \boldsymbol{R_{i-1}}||} \tag{2.21}$$

Dla toru ruchu opisanego łukiem okręgu błąd konturu stanowi różnicę długości promienia okręgu i odcinka łączącego środek okręgu i rzeczywiste położenie suportu (rys. 2.13b). Dla okręgu na płaszczyźnie błąd konturu wyraża się wzorem[54]:

$$\epsilon_c = \sqrt{(r\sin(\theta) - \epsilon_x)^2 + (-r\cos(\theta) - \epsilon_y)} \tag{2.22}$$

Wyznaczenie błędu konturu dla kołowego toru ruchu w przestrzeni trójwymiarowej jest problemem bardziej złożonym gdyż punkt rzeczywisty nie musi leżeć w tej samej płaszczyźnie co tor ruchu. Konieczne jest więc najpierw rzutowanie punktu na płaszczyznę w której znajduje się okrąg a następnie wyznaczenie składowej błędu konturu w płaszczyźnie okręgu.

Powyższe wzory pozwalają na dokładne wyznaczenie błędu konturu dla torów liniowych i kołowych. Jeżeli tor ruchu opisany jest krzywą parametryczną np. NURBS dokładne wyznaczenie błędu konturu nie jest możliwe [28]. W literaturze zaproponowano szereg metod aproksymacji błędu konturu dla krzywoliniowego toru ruchu. W metodzie zaproponowanej w [108] błąd konturu wyznacza się jako najkrótszą odległość pomiędzy położeniem rzeczywistym a prostą styczną do krzywej w zadanym punkcie.

Wektory ortonormalne styczny $\hat{t}$, normalny $\hat{n}$ i binormalny $\hat{b}$ w zadanym punkcie $C(u)$ tworzą układ współrzędnych Freneta (ang. Frenet frame). Wektory wyznacza się ze wzorów:

$$\hat{\boldsymbol{t}}(\boldsymbol{u}) = \frac{\boldsymbol{C'(u)}}{||\boldsymbol{C'(u)}||} \qquad \hat{\boldsymbol{b}}(\boldsymbol{u}) = \frac{\boldsymbol{C'(u)} \times \boldsymbol{C''(u)}}{||\boldsymbol{C'(u)} \times \boldsymbol{C''(u)}||} \qquad \hat{\boldsymbol{n}}(\boldsymbol{u}) = \frac{\hat{\boldsymbol{b}}(\boldsymbol{u}) \times \boldsymbol{C'(u)}}{||\hat{\boldsymbol{b}}(\boldsymbol{u}) \times \boldsymbol{C'(u)}||} \tag{2.23}$$

Aproksymacja wektora błędu konturu ϵ_c stanowi sumę rzutów wektora błędu nadążania ϵ_t na wektory normalny i binormalny.

$$\vec{\boldsymbol{\epsilon_c}} = (\vec{\boldsymbol{\epsilon_t}} \cdot \hat{\boldsymbol{n}})\hat{\boldsymbol{n}} + (\vec{\boldsymbol{\epsilon_t}} \cdot \hat{\boldsymbol{b}})\hat{\boldsymbol{b}} \tag{2.24}$$

Przybliżoną wartość błędu konturu stanowi długość wektora aproksymującego błąd konturu:

$$\epsilon_c = ||\vec{\boldsymbol{\epsilon_c}}|| \tag{2.25}$$

W celu wyznaczenia błędu konturu powyższą metodą konieczna jest znajomość pierwszej i drugiej pochodnej krzywej dla parametru u, które to wartości wyznacza się w procesie interpolacji metodą Taylora. Przedstawiona metoda charakteryzuje się więc małą złożonością obliczeniową. Przybliżenie krzywej prostą styczną jest słuszne o ile punkt rzeczywisty leży blisko płaszczyzny tworzonej przez wektory normalny i binormalny. Założenie to może nie być spełnione jeżeli błędy nadążania mają dużą wartość.

Koren w artykule [54] zaproponował aproksymację krzywej łukiem o promieniu równym promieniowi krzywizny krzywej dla danej wartości parametru u. Błąd konturu wyznaczany jest wtedy tak jak dla toru ruchu opisanego łukiem. W przypadku toru trójwymiarowego metoda jest bardziej złożona. Ponadto łuk dobrze aproksymuje krzywą tylko wtedy kiedy jej krzywizna nie zmienia się gwałtownie w okolicach zadanego punktu.

Wielu badaczy zaproponowało różne metody poprawy jakości estymacji błędu konturu [106, 14, 29, 86]. Z reguły bazują one na aproksymacji krzywej opisującej tor zadany lub wprowadzeniu dodatkowego współczynnika korekcyjnego do podstawowego przybliżenia prostą styczną. Wadą tych metod jest z reguły większa złożoność obliczeniowa i konieczność wprowadzania dodatkowych informacji. Próbę rozwiązania tego problemu podjęto m.in w artykule [116] gdzie zaproponowano aproksymację funkcji błędu konturu za pomocą szeregu Taylora. W metodzie tej wektor błędu konturu wyznacza się ze wzoru:

$$\vec{\boldsymbol{\epsilon_c}} = \left[-\vec{\boldsymbol{c}} - \frac{1}{2} \frac{\kappa\left(\hat{\boldsymbol{c}} \cdot \hat{\boldsymbol{n}}\right)\left(\hat{\boldsymbol{t}} \cdot \vec{\boldsymbol{\epsilon_t}}\right)\hat{\boldsymbol{t}}}{1 - \kappa\left(\hat{\boldsymbol{c}} \cdot \hat{\boldsymbol{n}}\right)} \right] \cdot \vec{\boldsymbol{\epsilon_t}} \tag{2.26}$$

Wektor $\hat{\boldsymbol{c}}$ jest wektorem jednostkowym leżącym w kierunku prostej łączącej punkt zadany na krzywej z rzutem punktu rzeczywistego na płaszczyznę normalną do krzywej. Wektor ten jest

wyznaczany ze wzoru[108]:

$$\hat{\boldsymbol{c}} = -\frac{\vec{\boldsymbol{\epsilon_t}} \cdot \hat{\boldsymbol{t}}}{\sqrt{||\vec{\boldsymbol{\epsilon_t}}||^2 - \vec{\boldsymbol{\epsilon_t}} \cdot \hat{\boldsymbol{t}}}}\hat{\boldsymbol{t}} + \frac{1}{\sqrt{||\vec{\boldsymbol{\epsilon_t}}||^2 - \vec{\boldsymbol{\epsilon_t}} \cdot \hat{\boldsymbol{t}}}}\vec{\boldsymbol{\epsilon_t}} \tag{2.27}$$

Zaletą tej metody jest większa dokładność aproksymacji błędu konturu szczególnie dla krzywych o dużych fluktuacjach krzywizny. Kolejną zaletą jest złożoność obliczeniowa niewiele większa niż w przypadku metody aproksymacji styczną. Wymagane jest jedynie wyznaczenie pierwszej i drugiej pochodnej krzywej, które wyznaczane są w procesie interpolacji krzywej.

W literaturze zaproponowano iteracyjne wyznaczenia błędu konturu [18, 12, 84]. Metoda ta polega na poszukiwaniu wartości parametru u odpowiadającej punktowi na krzywej $C(u)$, będącemu prostopadłym rzutem pozycji rzeczywistej P na krzywą. Wartość tą można znaleźć wykorzystując metodę Newtona-Raphsona gdzie kolejne iteracje parametru krzywej wyrażają się wzorem [95]:

$$u_{i+1} = u_i - \frac{\boldsymbol{C'(u_i)} \cdot (\boldsymbol{C(u_i)} - \boldsymbol{P})}{\boldsymbol{C''(u_i)} \cdot (\boldsymbol{C(u_i)} - \boldsymbol{P}) + ||\boldsymbol{C'(u_i)}||^2} \tag{2.28}$$

Metodę tą cechuje duża dokładność aproksymacji rzeczywistego błędu konturu kosztem zwiększenia czasu obliczeń. W przypadku konieczności wielokrotnego wyznaczania błędu konturu czas obliczeń może ulec istotnemu zwiększeniu. Z tego względu metoda ta jest przydatna głównie w celu weryfikacji poprawności estymacji błędu konturu innymi metodami.

3. Neuronowy predyktor błędu konturu

Integralnym elementem predykcyjnego algorytmu optymalizacji prędkości posuwu w układach sterowania maszyn, zaproponowanego w niniejszej rozprawie, jest model układu posuwu maszyny wieloosiowej. Model taki pozwala na przewidywanie błędu konturu zadanego toru ruchu i optymalne dobranie prędkości posuwu bez przekraczania zadanej tolerancji błędu konturu. Błąd konturu wyznaczany jest na podstawie błędów nadążania poszczególnych osi maszyny i geometrii zadanego toru ruchu. W celu zbudowania predyktora błędu konturu konieczne jest opracowanie i identyfikacja modeli dynamicznych układów posuwu poszczególnych osi maszyny.

W najprostszym przypadku model układu posuwu maszyny wieloosiowej przyjmuje postać modelu liniowego rzędu pierwszego [52]. W takim modelu uwzględnia się tylko moment bezwładności osi mechanicznej zredukowany na wał silnika oraz współczynnik tarcia wiskotycznego (tarcie proporcjonalne do prędkości). Podstawowy model liniowy pierwszego rzędu można rozbudować tak aby uwzględnić sztywność połączeń pomiędzy poszczególnymi elementami mechanicznymi [26], tarcie nieliniowe [4, 47], luzy mechaniczne [110], nachylenie osi względem podłoża [46], deformacje termiczne [21] i inne. Uwzględnienie wielu różnych zjawisk fizycznych występujących w rzeczywistym układzie prowadzi do bardzo złożonych modeli wysokich rzędów. Model taki cechuje duża dokładność w przewidywaniu błędów nadążania osi lecz duża liczba parametrów opisująca poszczególne zjawiska fizyczne powoduje, że proces identyfikacji jest z reguły żmudny. Predykcja odpowiedzi obiektu z wykorzystaniem takiego złożonego modelu cechuje duża złożoność obliczeniowa. Z tego względu stosuje się modele typu wejście-wyjście o arbitralnie dobranej strukturze wewnętrznej. Ponieważ zakłada się brak wiedzy o wewnętrznej strukturze modelowanego procesu modele takie nazywa się modelami typu “czarne pudełko” (ang. black-box model) [67]. Zastosowanie modeli typu wejście pozwala na wykorzystanie w procesie identyfikacji danych uzyskanych podczas normalnej pracy maszyny.

Modele typu black-box mogą być modelami liniowymi lub nieliniowymi. Wśród modeli liniowych najczęściej stosuje się modele zmiennych stanu, modele autoregresyjne z wejściem zewnętrznym (ARX) oraz modele transmitancyjne. Modele takie są stosunkowo łatwe w identyfikacji lecz nie są w stanie w pełni odwzorować dynamiki obiektu nieliniowego jakim jest napęd posuwu osi mechanicznych maszyny CNC.

Do modelowania zjawisk fizycznych wykazujących charakter nieliniowy często stosowane są sztuczne sieci neuronowe [111, 73]. Sztuczne sieci neuronowe są strukturami matematycznymi umożliwiającymi aproksymację funkcji nieliniowych. Najczęściej stosowanym wariantem sztucznych sieci neuronowych jest wielowarstwowy perceprtron (ang. multilayer perceptron, MLP). Sieć taka składa się z neuronów realizujących liniową bądź nieliniową funkcję aktywacji, której argumentem jest suma wartości wejściowych pomnożonych przez wartości wagowe przypisane do każdego wejścia. Do sumy iloczynów wag i wejść dodawany jest „bias" będący dodatkową „wolną" wagą specyficzną dla danego neuronu. Jako nieliniową funkcję aktywacji często stosuje się funkcję tangens hiperboliczny (funkcja sigmoidalna bipolarna). Działanie pojedyńczego neuronu z sigmoidalną funkcją aktywacji można zapisać za pomocą następującego równania:

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

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

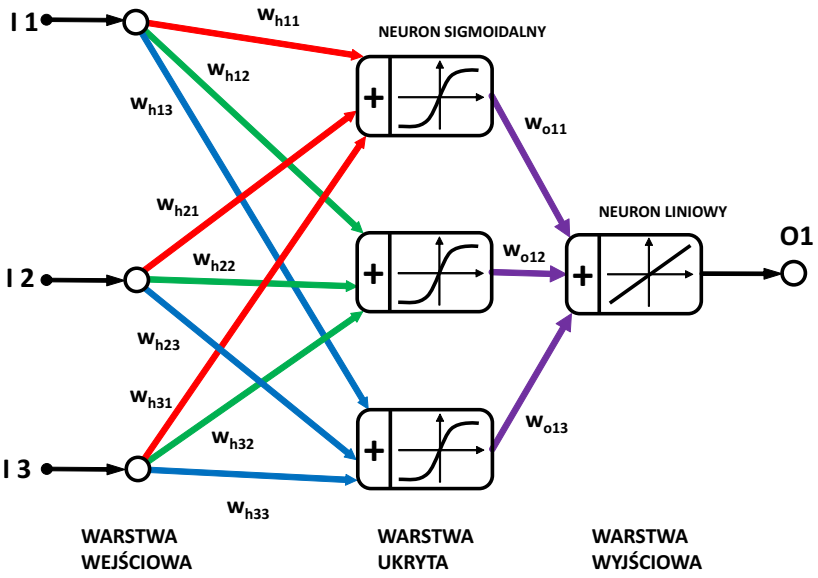
Neurony pogrupowane są w warstwy. Wartości wyjściowe neuronów jednej warstwy stanowią wejścia każdego z neuronów warstwy następnej. Pierwsza warstwa, której wejścia stanowią wejścia sieci, nosi nazwę warstwy wyjściowej. Ostatnia warstwa nazywana jest warstwą wyjściową. Warstwy pomiędzy warstwami wejściową i wyjściową nazywane są warstwami ukrytymi. W większości zastosowań stosuje się jedną warstwę ukrytą z sigmoidalną funckją aktywacji i liniową funkcją aktywacji w warstwie wyjściowej. Taka sieć posiada zdolność uniwersalnej aproksymacji funkcji nieliniowych [20, 44]. Istotnym problemem jest dobór ilości neuronów warstw ukrytych sieci. Nie istnieje żaden uniwersalny sposób optymalnego doboru ilości neuronów, przez co dla wielu przypadków konieczne jest stosowanie metody prób i błędów.

Wagi sieci dobierane są w procesie uczenia, który polega na iteracyjnej minimalizacji wskaźnika jakości aproksymacji danych uczących (tzw. uczenie z nauczycielem). Najczęściej stosowanym wskaźnikiem jakości jest błąd średniokwadratowy (ang. mean squared error, MSE) pomiędzy zbiorem danych uczących a wyjściami sieci.

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

gdzie: N_d - liczba danych uczących, d_i - dane uczące, $\hat{y_i}$ - odpowiedź sieci.

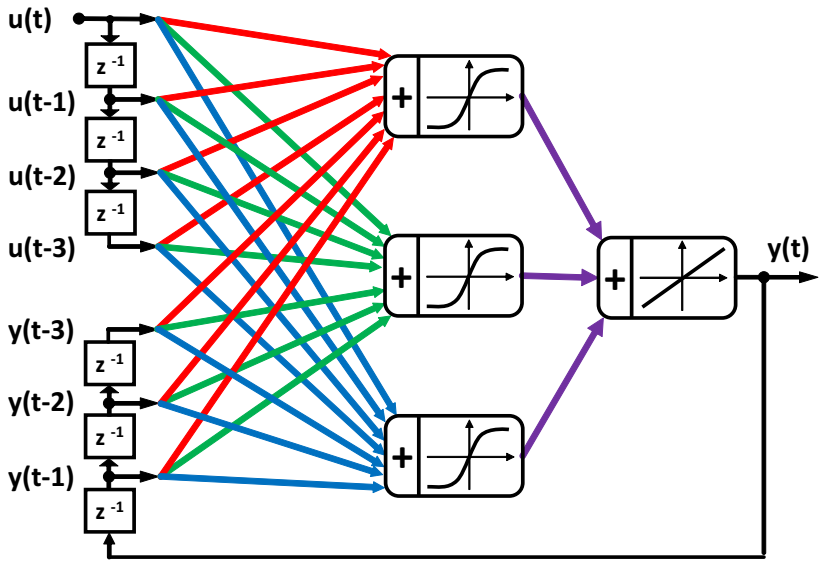
W literaturze przedstawiono wiele rodzajów algorytmów uczących. Najczęściej stosowane są algorytmy wykorzystujące gradient błędu np. powszechnie stosowana metoda



Rysunek 3.1. Graficzna reprezentacja sztucznej sieci neuronowej typu wielowarstwowy perceptron o jednej warstwie ukrytej, trzech wejściach i jednym wyjściu

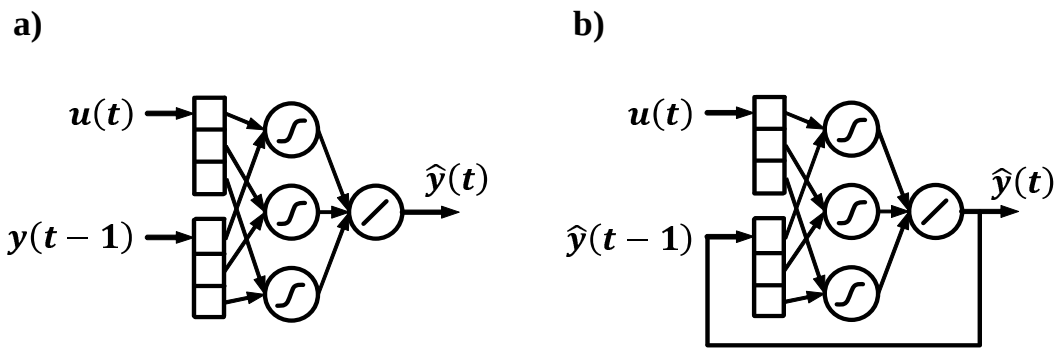
Levenberga-Marquardt'a. Istotnym problemem występującym w procesie uczenia jest "przeuczenie" sieci. Sieć przeuczona charakteryzuje się dobrym odwzorowaniem tylko dla danych wykorzystywanych w procesie uczenia. Aby sieć była użyteczna powinna wykazywać własności generalizacji tzn. dobrze odwzorowywać zależność nieliniową dla danych nie wykorzystywanych w procesie uczenia. Z tego względu do ostatecznej weryfikacji jakości uczenia stosuje się zbiór danych niezależny od od danych uczących - tzw. zbiór testowy.

Klasyczne jednokierunkowe sieci neuronowe typu wielowarstwowy perceptron są w stanie aproksymować bardzo złożone statyczne zależności nieliniowe. W celu modelowania obiektów fizycznych, które często są układami dynamicznymi, stosuje się rekurencyjne sieci dynamiczne. W tego typu sieciach występują wewnętrzne stany realizowane z reguły jako bloki opóźniające oraz sprzężenia zwrotne pomiędzy warstwami sieci. Często stosowanym rodzajem dynamicznej sieci rekurencyjnej jest sieć typu NARX (ang. Nonlinear AutoRegressive eXogenous input). Sieć typu NARX (rys. 3.2) jest generalizacją liniowego modelu autoregresyjnego (ARX). Od klasycznej sieci jednokierunkowej różni się sprzężeniem zwrotnym pomiędzy warstwą wyjściową i wejściową (wyjście sieci podawane jest na jej wejście) oraz blokami opóźniającymi na wejściu i w torze sprzężenia zwrotnego. Sieć NARX nie posiada bloków opóźniających ani sprzężeń wewnątrz sieci co znacznie przyspiesza trening ze względu na łatwiejsze wyznaczanie gradientu błędu w porównaniu z innymi rodzajami sieci rekurencyjnych i dynamicznych [43]. Struktura takiej sieci dobierana jest z reguły metodą prób i błędów. Konieczny jest wybór ilości warstw ukrytych, liczby neuronów w poszczególnych warstwach oraz ilości bloków opóźniających.



Rysunek 3.2. Graficzna reprezentacja przykładowej sieci neuronowej typu NARX.

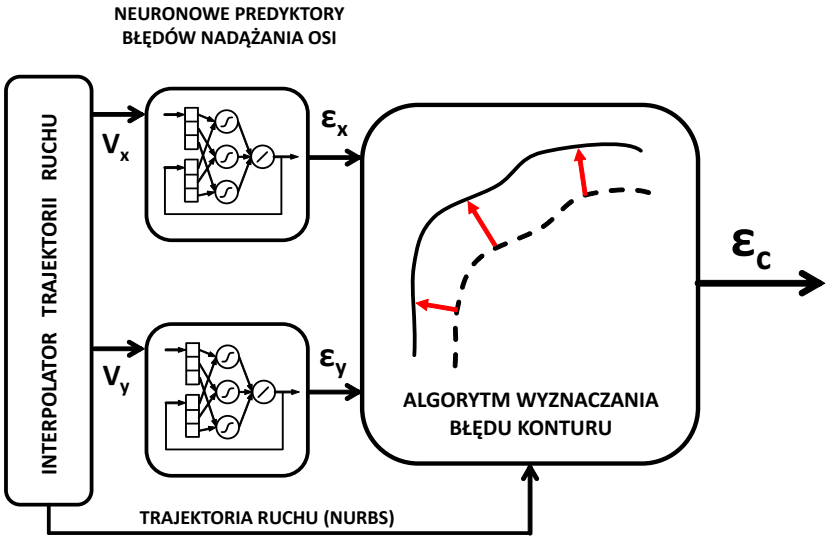
Sieć neuronowa typu NARX może działać w dwóch różnych architekturach, szeregowo-równoległej i równoległej [72]. Oba warianty architektury sieci NARX przedstawiono na rysunku 3.3. W architekturze szeregowo-równoległej nie występuje sprzężenie zwrotne pomiędzy wyjściem i wejściami sieci neuronowej. Zamiast poprzednich wartości wyjściowych sieci na wejście podawane są poprzednie wartości wyjściowe rzeczywistego obiektu. Rozwiązanie to pozwala na uzyskanie dużej dokładności predykcji odpowiedzi obiektu na sygnał wejściowy. Jednocześnie trening takiej sieci przebiega szybko ze względu na zastosowanie standardowych metod uczenia sieci jednokierunkowych [104]. Architektura ta może być zastosowana tylko jeżeli dane wyjściowe rzeczywistego obiektu są dostępne na bieżąco. Ma to miejsce w przypadku uczenia sieci lub w zastosowaniach on-line do predykcji odpowiedzi w krótkim horyzoncie czasowym. W zastosowaniach off-line, gdzie konieczna jest predykcja w dłuższym horyzoncie czasowym, bardziej odpowiednia jest architektura równoległa, gdyż predykcja realizowana jest tylko na podstawie wartości wejściowych. Ponieważ wartości wyjściowe sieci podawane na wejście w formie sprzężenia zwrotnego zawsze obarczone są pewnym błędem predykcji, sieć NARX w architekturze równoległej charakteryzuje się większym błędem predykcji niż sieć w architekturze szeregowo-równoległej. Trening sieci w architekturze równoległej za pomocą powszechnie stosowanych algorytmów gradientowych wymaga zastosowania bardziej złożonej metody wyznaczania gradientu (tzw. gradient dynamiczny). Proces treningu trwa dłużej i jest bardziej podatny na zatrzymanie w minimum lokalnym ze względu na bardziej złożoną funkcję błędu. Ze względu na większą złożoność funkcji błędu wynik treningu sieci NARX w architekturze równoległej silnie zależy od wartości początkowych wag.



Rysunek 3.3. Architektury sieci NARX - szeregowo-równoległa (a) i równoległa (b).

3.1. Identyfikacja modelu neuronowego

W niniejszej rozprawie wykorzystany został dwuosiowy neuronowy predyktor błędu konturu wykorzystujący sieci neuronowe typu NARX. Każda z osi maszyny reprezentowana jest przez sieć NARX, która dokonuje predykcji błędu nadążania w odpowiedzi na prędkość zadaną wyznaczoną w procesie interpolacji toru ruchu opisanego krzywą NURBS. Predykowane błędy nadążania wraz z informacjami o trajektorii uzyskanymi z interpolatora wykorzystywane są do wyznaczenia błędu konturu. Schemat blokowy zaproponowanego predyktora błędu konturu przedstawiono na rysunku 3.4.

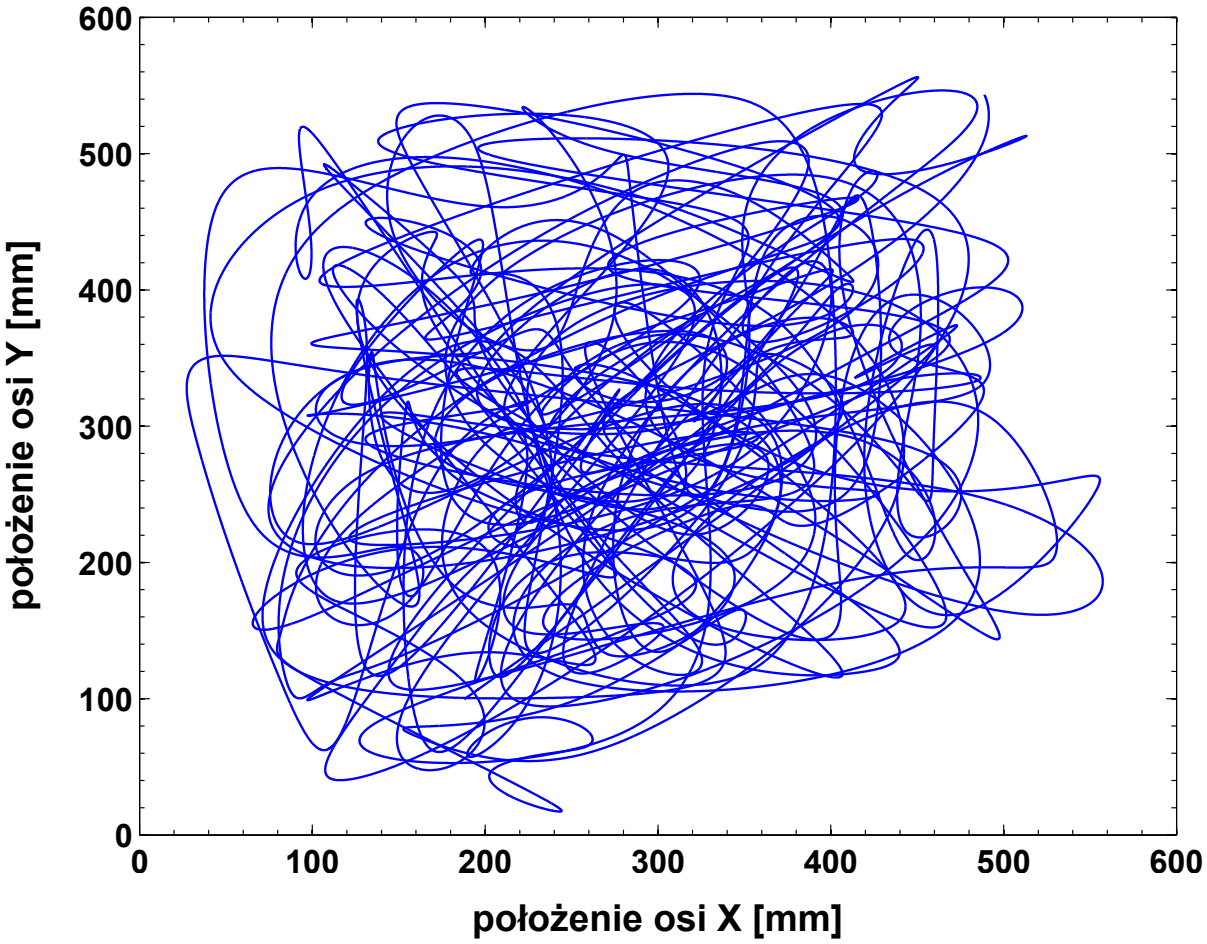


Rysunek 3.4. Schemat blokowy neuronowego predyktora błędu konturu

Każda z sieci NARX reprezentująca poszczególne osi maszyny uczona jest niezależnie od siebie na podstawie danych uzyskanych z maszyny wieloosiowej. Dane odczytywane są podczas normalnej pracy maszyny tzn. sieć NARX reprezentuje dynamikę układu posuwu pracującego w zamkniętej pętli regulacji. W takim rozwiązaniu nie ma konieczności zmiany

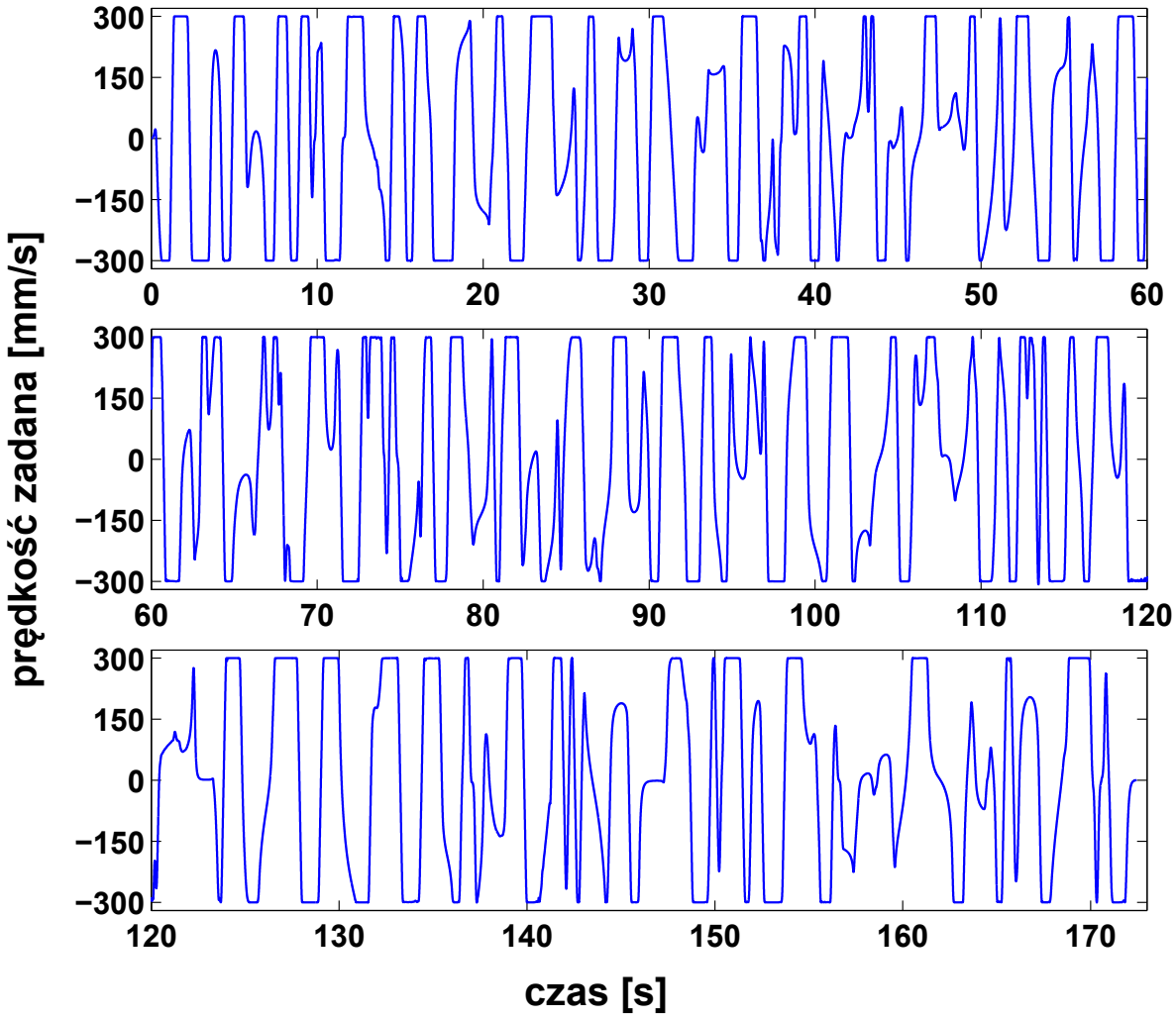
konfiguracji oprogramowania serwonapędów czy też sterownika CNC, co może być niemożliwe w przypadku zastosowania komercyjnych komponentów układu sterowania.

W celu uzyskania zbioru danych uczących wygenerowano w sposób losowy tor ruchu w formie krzywej NURBS (Rys.3.5). Do wygenerowanego toru ruchu dobrano profil prędkości zapewniający nie przekraczanie dopuszczalnych wartości prędkości, przyspieszenia i zrywu dla identyfikowanych osi. Maksymalne dopuszczalne wartości tych wielkości wynoszą dla obu osi odpowiednio 500 mm/s, 2000 mm/s^2 i 50000 mm/s^3. Prędkości zadane będące sygnałami wejściowymi wykorzystanymi do treningu sieci NARX przedstawiono na rysunkach 3.6 i 3.7.



Rysunek 3.5. Krzywa NURBS definiująca tor ruchu wykorzystany do treningu neuronowego predyktora błędu konturu

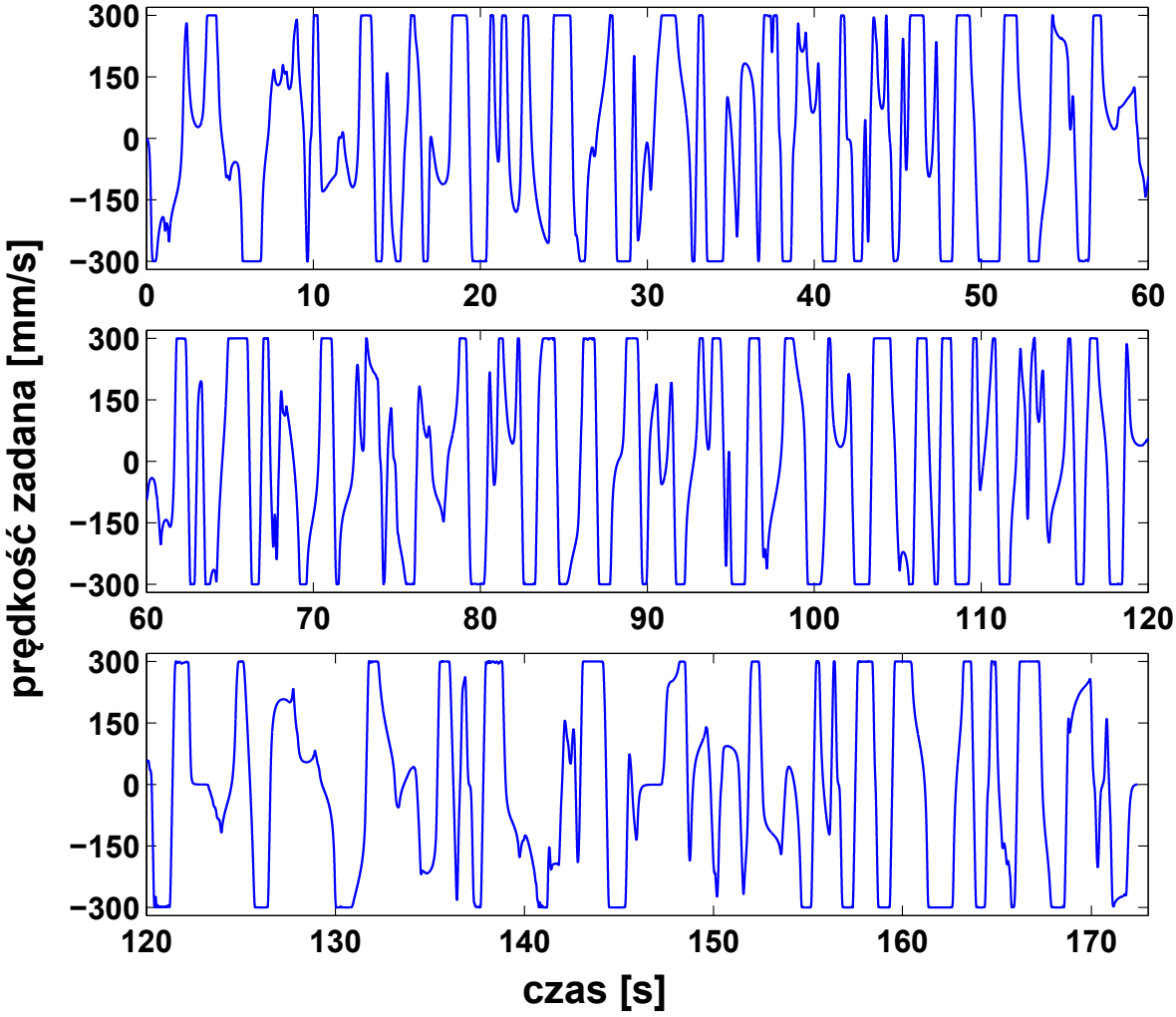
Wygenerowana trajektoria ruchu została odtworzona na maszynie dwuosiowej CNC. Obie Osie maszyny zbudowane są z wykorzystaniem śrub kulowych o skoku 10 mm/obr, napędzanymi przez silniki PMSM (T_n=1,5Nm ω_n=3000 obr/min) współpracujące z serwonapędami Microflex e100. Pomiar położenia rzeczywistego realizowany jest za pośrednictwem enkodera inkrementalnego o rozdzielczości 2500 impulsów na obrót. Położenia zadane do serwonapędów generowane są przez oprogramowanie



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

zaimplementowane na komputerze PC z systemem czasu rzeczywistego Linux RTAI. W systemie RTAI zaimplementowano interpolator trajektorii ruchu opisanej krzywymi NURBS i oprogramowanie komunikacyjne obsługujące standard Ethernet Powerlink. Położenia zadane wysyłane są do serwonapędów z częstotliwością 1kHz. W każdym cyklu serwonapędy przesyłają do komputera aktualną wartość błędu nadążania. Odczytany błąd nadążania zapisywany jest do pliku tekstowego w celu dalszej obróbki w programie Matlab. Szczegółowy opis stanowiska badawczego zawarto w rozdziale 5.

Ponieważ każda maszyna wykazuje błędy powtarzalności, błąd nadążania różni się pomiędzy różnymi odtworzeniami tej samej trajektorii ruchu. Z tego względu pomiar błędu nadążania powtórzono 10 razy a zebrane próbki uśredniono. Ze względu na ograniczoną rozdzielczość pomiaru i proces uśredniania uzyskany przebieg nie jest gładki i występuje w nim szum. Cechy te mogą negatywnie wpływać na proces treningu sieci. Z tego względu uśredniony przebieg błędu nadążania jest wygładzany poprzez dopasowanie wygładzającego wielomianu

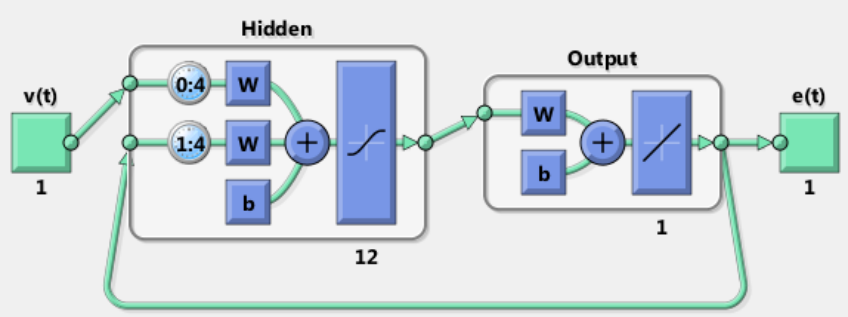


Rysunek 3.7. Przebieg prędkości zadanej w osi Y wykorzystany jako sygnał wejściowy w procesie treningu sieci neuronowej

sklejanego 3-go stopnia (ang. cubic smoothing spline). Proces ten przeprowadzony jest w programie Matlab z wykorzystaniem polecenia "csaps" z pakietu "Curve Fitting Toolbox" ze współczynnikiem wygładzania równym $5e^{-2}$.

Proces treningu przeprowadzony został w oprogramowaniu Matlab z wykorzystaniem pakietu Neural Network Toolbox. Strukturę sieci NARX wykorzystanej w procesie treningu wybrano metodą prób i błędów przeprowadzając wielokrotny trening dla różnych architektur i wybierając sieć charakteryzującą się najniższym błędem predykcji. Wybrana sieć neuronowa posiada jedną warstwę ukrytą oraz po 4 bloki opóźniające na wejściu i w torze sprzężenia zwrotnego. Graficzną reprezentację sieci NARX wykorzystanej w procesie identyfikacji predyktora neuronowego przedstawiono na rysunku 3.8. Tą samą strukturę sieci wykorzystano do modelowania obydwu osi maszyny.

Dane dla każdej osi podzielono na dwa równe zbiory - uczący i testowy. Zbiór testowy posłużył do weryfikacji jakości generalizacji sieci neuronowej. Do treningu



Rysunek 3.8. Graficzna reprezentacja sieci neuronowej NARX wykorzystanej w procesie identyfikacji predyktora neuronowego wygenerowana przez pakiet Neural Network Toolbox programu Matlab.

wykorzystano algorytm "trainbr" z pakietu Neural Network Toolbox, będący wariantem algorytmu Levenberga-Marquardt'a z automatyczną regularyzacją Bayes'owską [70, 38]. W tej metodzie zamiast błędu średniokwadratowego minimalizacji podlega następująca funkcja:

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

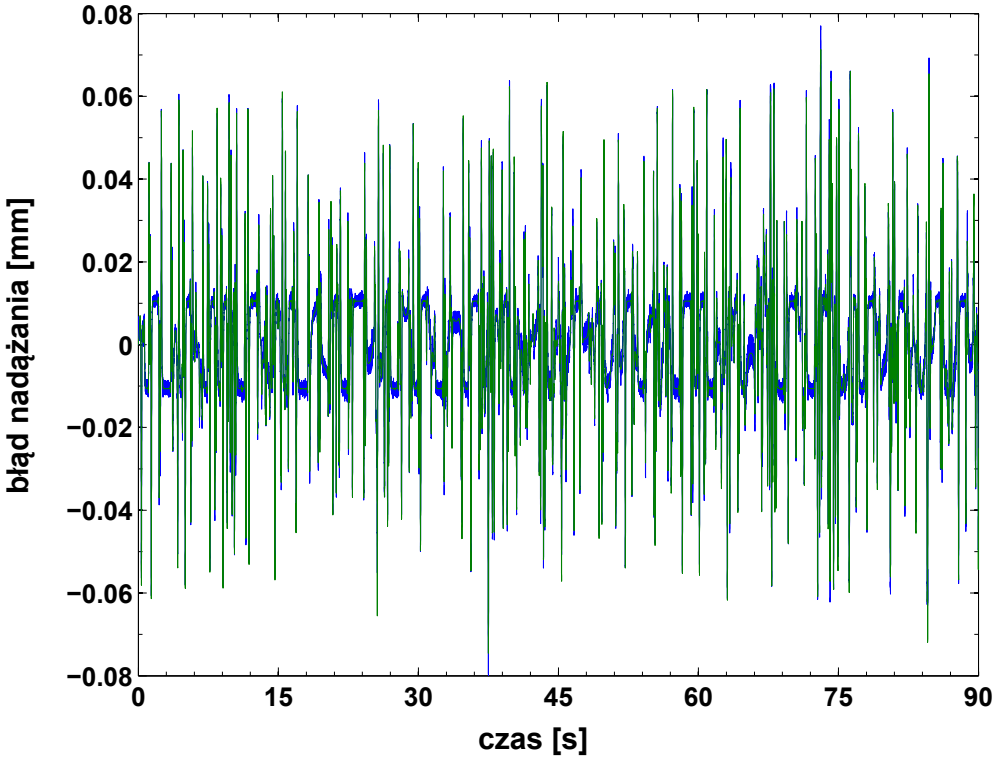
gdzie: N_d - liczba danych uczących, d_i - dane uczące, $\hat{y}_i$ - odpowiedź sieci, N_w - liczba wag sieci, w_j - wagi sieci, α,β - automatycznie dobierane współczynniki wagowe

Ideą regularyzacji jest niedopuszczenie do uzyskania w procesie uczenia zbyt dużych wartości wag co z reguły jest oznaką przeuczenia sieci. Suma kwadratów wag sieci pełni rolę funkcji kary za zbyt duże wartości wag. W praktyce algorytm regularyzacji powoduje spowolnienie procesu uczenia lecz w większości przypadków skutecznie zapobiega przeuczeniu poprzez ograniczenie wartości współczynników wagowych sieci.

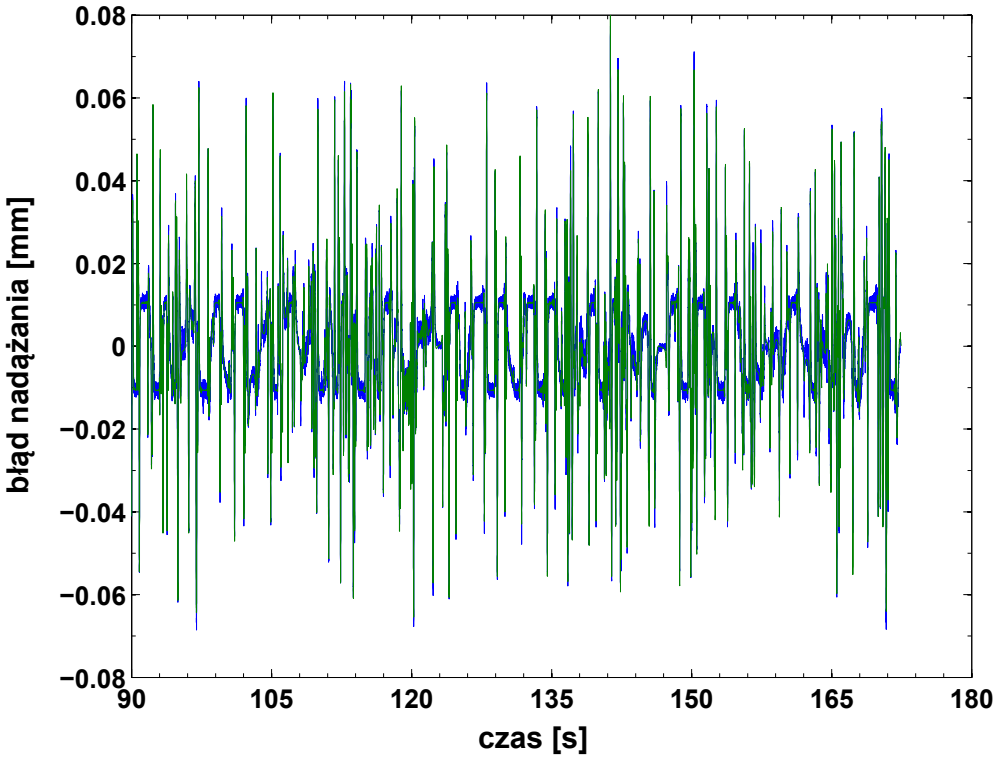
Uczenie sieci neuronowej NARX przebiegało dwuetapowo. Uczenie rozpoczęto w architekturze szeregowo-równoległej i po osiągnięciu minimalnej wartości gradientu ($1e^{-8}$) lub w przypadku braku postępu w procesie uczenia, architektura została zmieniona na równoległą. Uczenie sieci w architekturze równoległej kontynuowano do czasu aż kolejne iteracje algorytmu uczącego nie powodowały dalszego zmniejszania wartości błędu średniokwadratowego. Zastosowanie początkowego etapu uczenia sieci w architekturze szeregowo-równoległej z wykorzystaniem danych rzeczywistych w torze sprzężenia zwrotnego pozwala na uzyskanie korzystnych wartości początkowych wag dla uczenia z zamkniętą pętlą sprzężenia zwrotnego. Możliwe jest uczenie sieci bezpośrednio w architekturze równoległej lecz ze względu na losową inicjalizację wartości wagowych sieci istnieje prawdopodobieństwo niepowodzenia procesu uczenia ze względu na zatrzymanie w minimum lokalnym funkcji celu. Proces uczenia przeprowadzono niezależnie dla obydwu osi maszyny.

Odpowiedzi nauczonych sieci neuronowych modelujących osie X i Y porównane z rzeczywistym błędem nadążania przedstawiono na rysunkach 3.9 (zbiór treningowy dla osi X), 3.10 (zbiór testowy dla osi X), 3.11 (zbiór treningowy dla osi Y), 3.12 (zbiór testowy dla osi Y). Błąd predykcji, czyli różnicę pomiędzy rzeczywistym przebiegiem błędu nadążania a odpowiedzią sieci neuronowych przedstawiono na rysunkach 3.13, 3.14, 3.15, 3.16. Na wykresach 3.17 i 3.19 przedstawiono przebiegi rzeczywistego błędu konturu i wartości błędu konturu przewidywane przez predyktor neuronowy dla zbioru uczącego i testowego. Błąd predykcji błędu konturu czyli różnicę pomiędzy błędem konturu przewidywanym przez predyktor neuronowy a rzeczywistym przedstawiono na rysunkach 3.18 i 3.20.

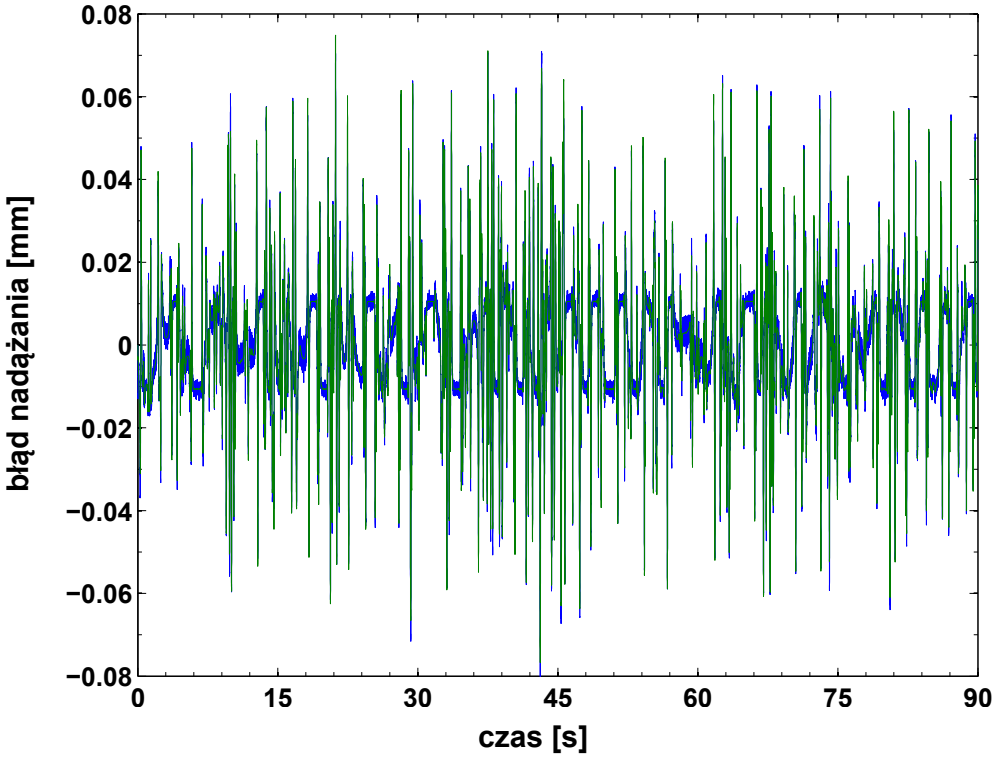
Z przedstawionych wyników badań wynika, że dynamiczna rekurencyjna sieć neuronowa typu NARX może skutecznie odwzorowywać dynamikę osi mechanicznych maszyn wieloosiowych. W szczególności sieć neuronowa pozwala na predykcję błędu nadążania poszczególnych osi. Na podstawie wartości przewidywanego błędu nadążania w poszczególnych osiach oraz znajomości geometrii zadanego toru ruchu możliwa jest predykcja błędu konturu realizowanej trajektorii ruchu. Z przedstawionych wykresów wynika, że predykcja błędu konturu obarczona jest błędem wynikającym z błędów predykcji błędu nadążania przez sieci NARX odpowiadające poszczególnym osiom. Z przeprowadzonych badań wynika, że błąd predykcji błędu konturu nie przekracza 0.01 mm. Ponieważ całkowite usunięcie błędu predykcji nie jest możliwe, wykorzystując neuronowy predyktor błędu konturu w algorytmie optymalizacji prędkości należy uwzględnić występowanie błędu predykcji. Pomimo występowania błędu predykcji przedstawiony w niniejszym rozdziale neuronowy predyktor błędu konturu pozwala na przewidywanie błędu z wystarczającą dokładnością do wykorzystania w predykcyjnym algorytmie optymalizacji prędkości posuwu.



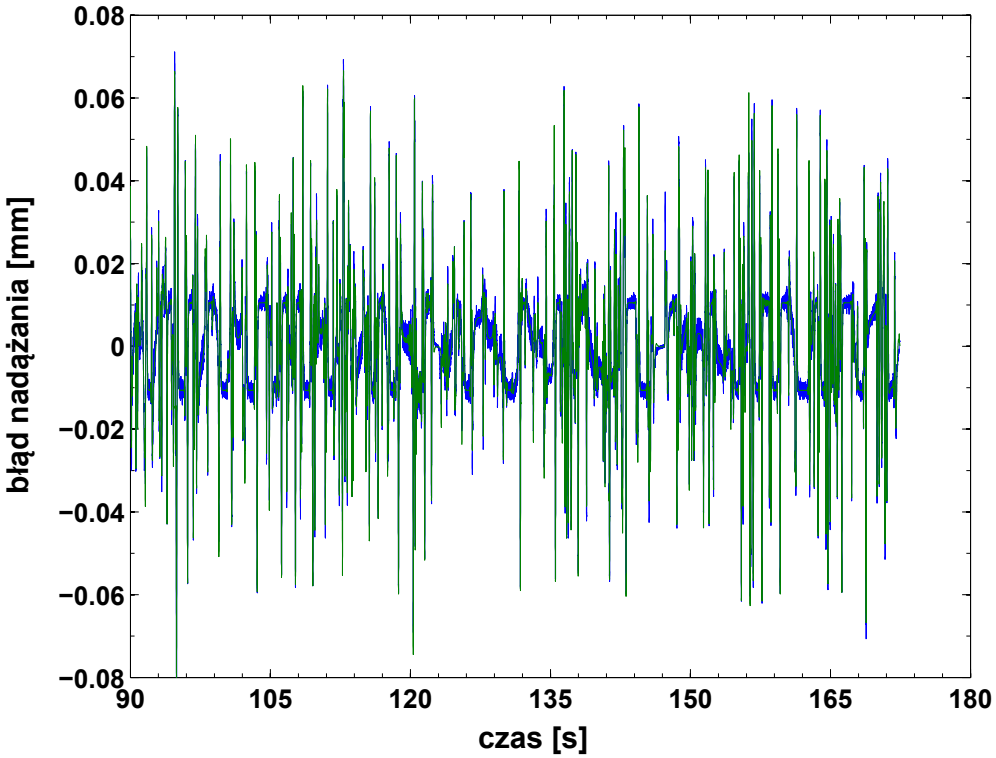
Rysunek 3.9. Przebieg rzeczywistego błędu nadążania w osi X (wykres niebieski) i odpowiedzi sieci neuronowej (wykres zielony). Zbiór treningowy t = 0 - 90 s.



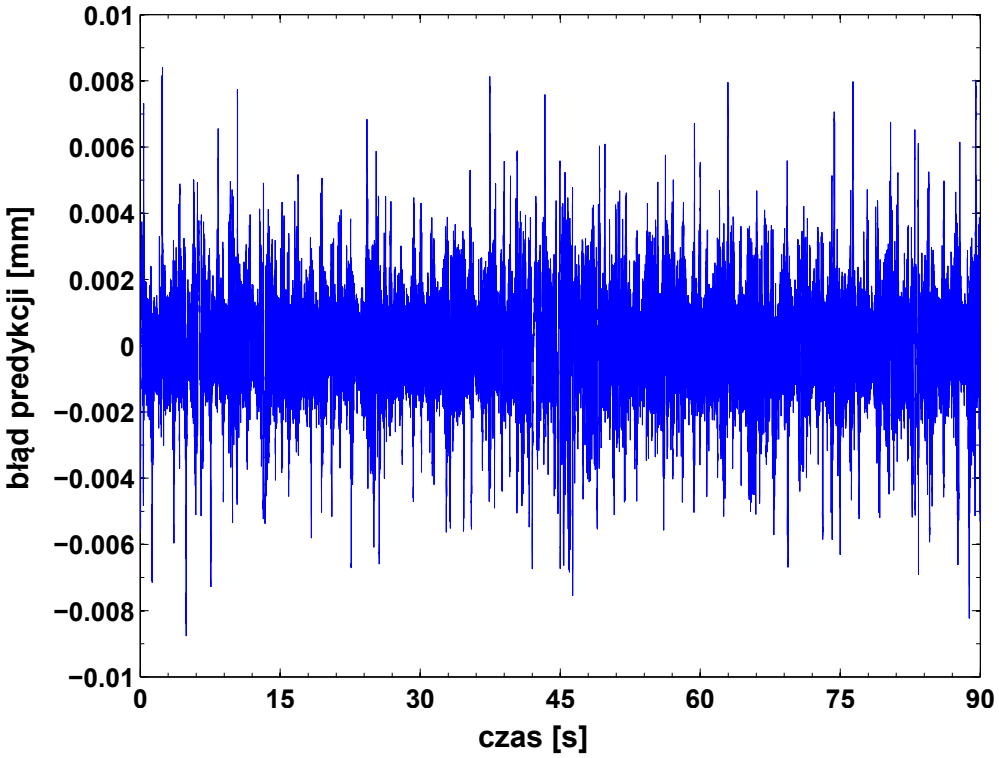
Rysunek 3.10. Przebieg rzeczywistego błędu nadążania w osi X (wykres niebieski) i odpowiedzi sieci neuronowej (wykres zielony). Zbiór testowy t = 90 - 180 s.



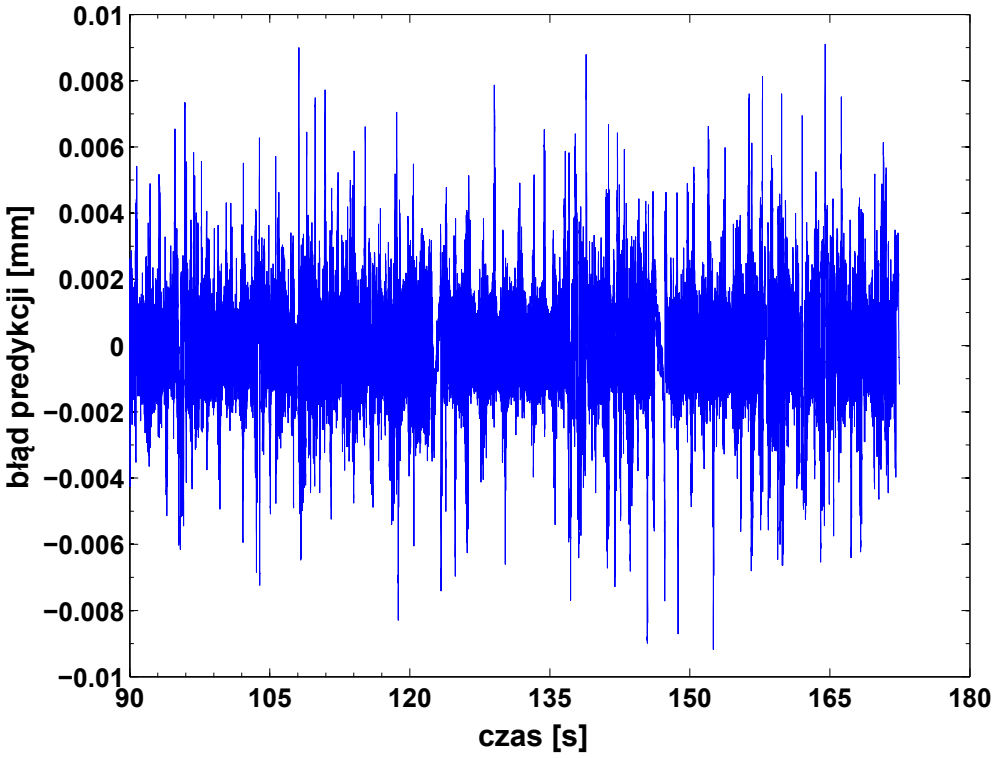
Rysunek 3.11. Przebieg rzeczywistego błędu nadążania w osi Y (wykres niebieski) i odpowiedzi sieci neuronowej (wykres zielony). Zbiór treningowy t = 0 - 90 s.



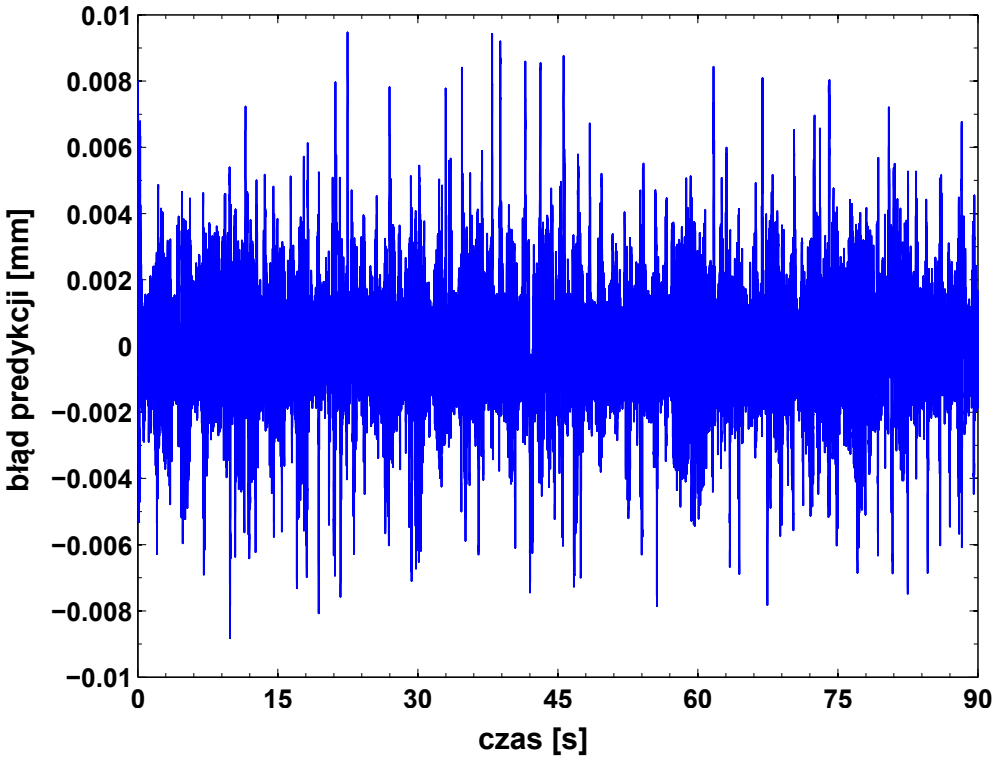
Rysunek 3.12. Przebieg rzeczywistego błędu nadążania w osi Y (wykres niebieski) i odpowiedzi sieci neuronowej (wykres zielony). Zbiór testowy t = 90 - 180 s.



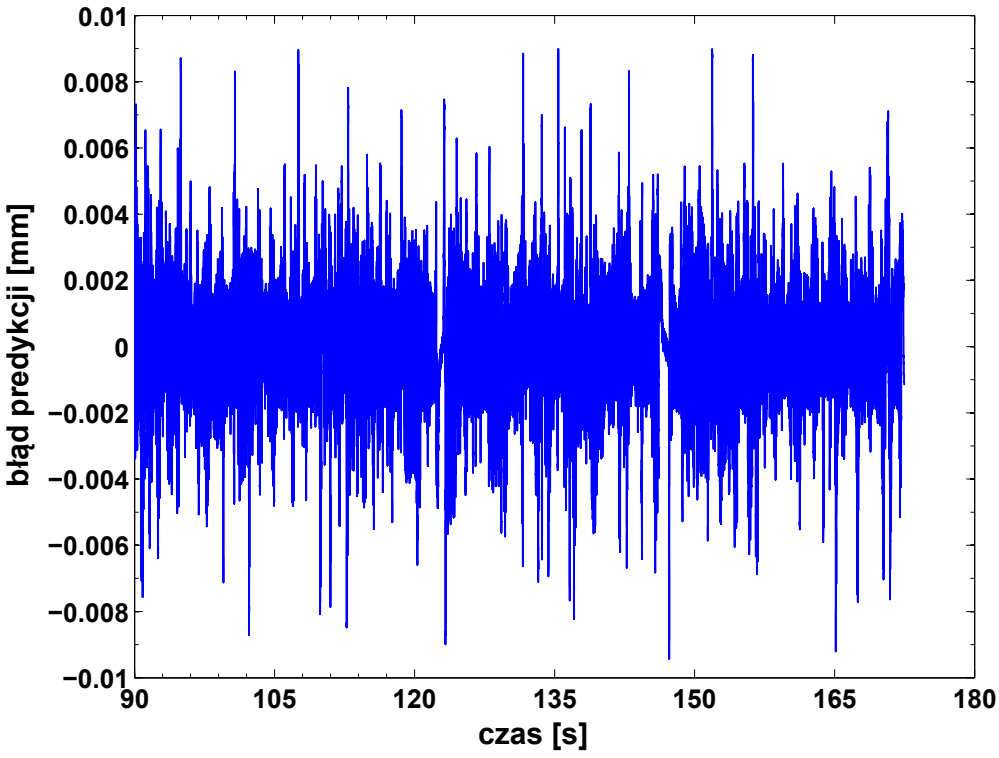
Rysunek 3.13. Błąd predykcji sieci neuronowej dla osi X (zbiór danych uczących t= 0 - 90 s).



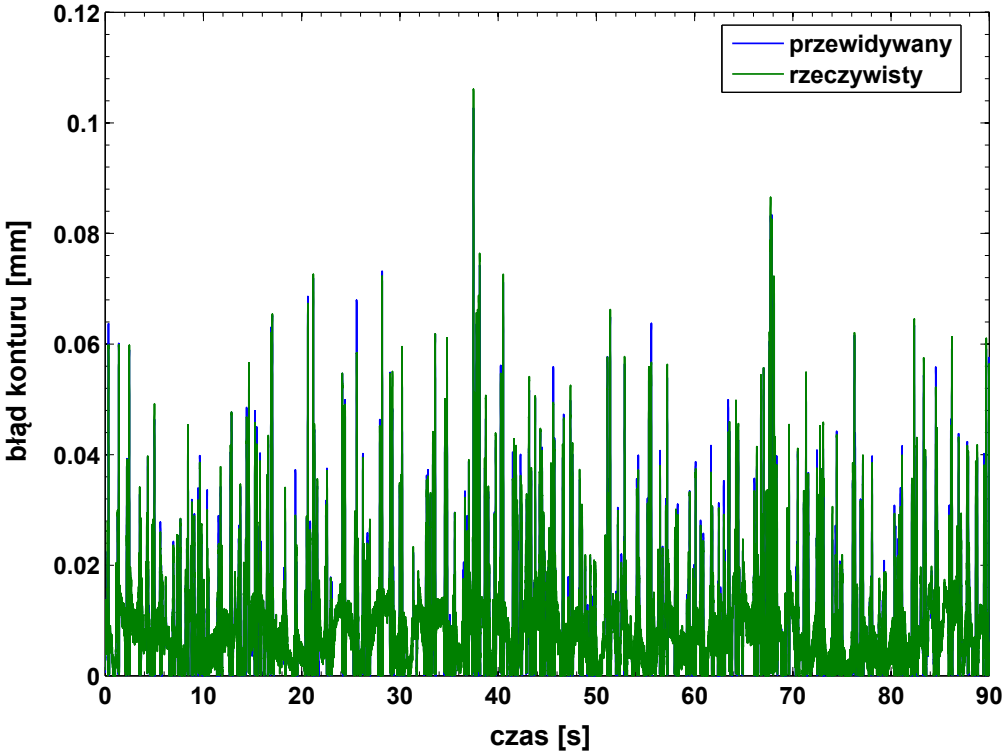
Rysunek 3.14. Błąd predykcji sieci neuronowej dla osi X (zbiór danych testowych t= 90 - 180 s).



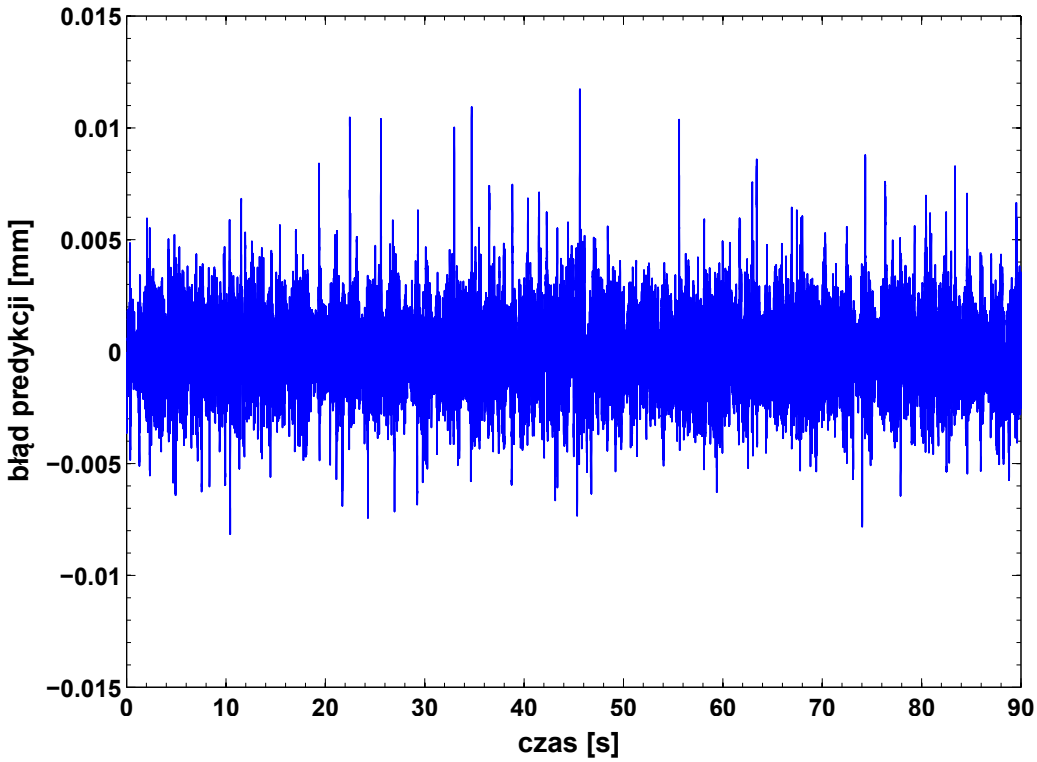
Rysunek 3.15. Błąd predykcji sieci neuronowej dla osi Y (zbiór danych uczących t= 0 - 90 s).



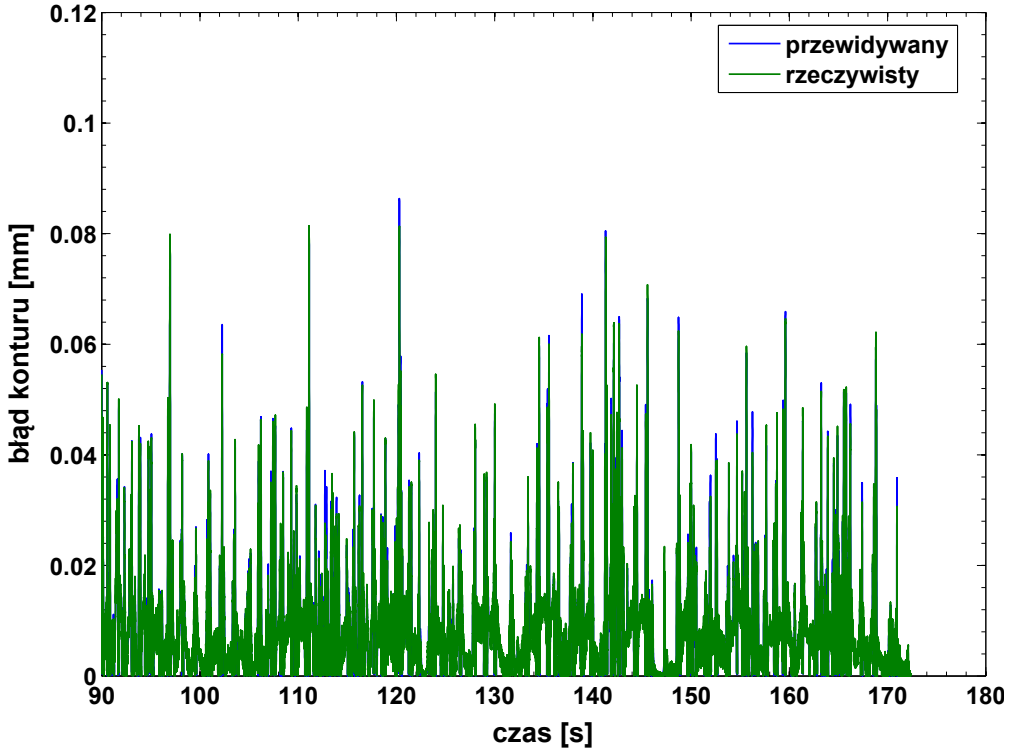
Rysunek 3.16. Błąd predykcji sieci neuronowej dla osi Y (zbiór danych testowych t= 90 - 180 s).



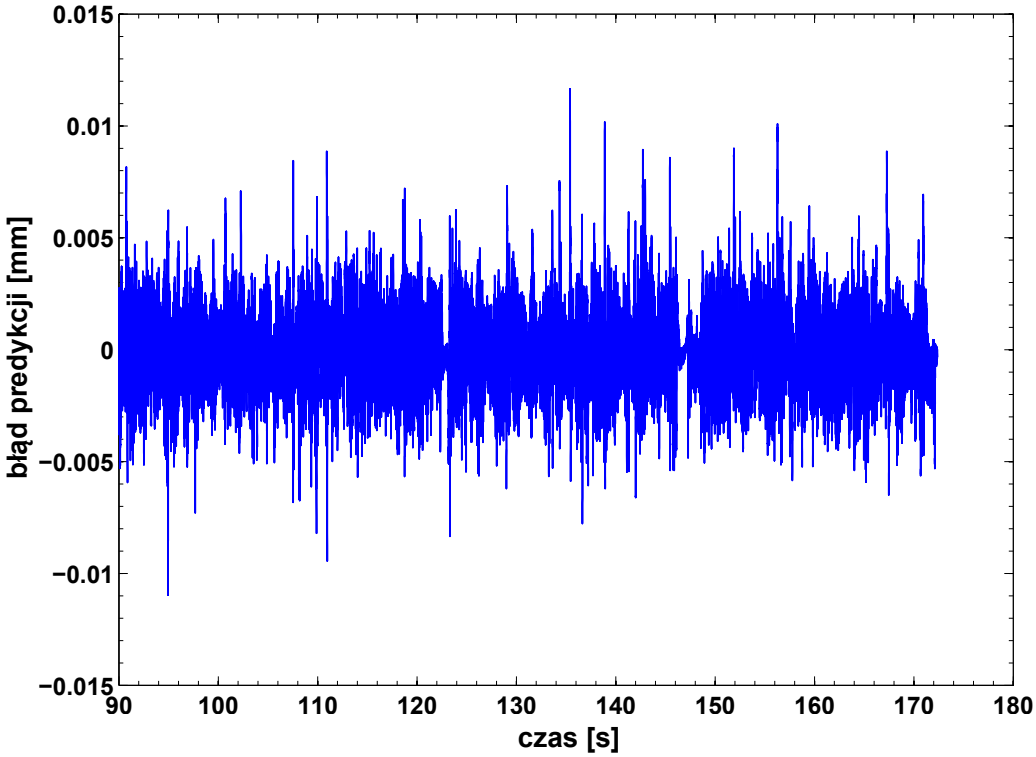
Rysunek 3.17. Błąd konturu (zbiór danych uczących t= 0 - 90 s).



Rysunek 3.18. Błąd predykcji błędu konturu (zbiór danych uczących t= 0 - 90 s).



Rysunek 3.19. Błąd konturu (zbiór danych testowych t= 90 - 180 s).



Rysunek 3.20. Błąd predykcji błędu konturu (zbiór danych testowych t= 90 - 180 s).

4. Predykcyjny algorytm doboru optymalnej prędkości posuwu zapewniający odtworzenie toru ruchu z zadaną dokładnością

Celem algorytmu doboru optymalnej prędkości posuwu jest znalezienie profilu prędkości posuwu maszyny wieloosiowej zapewniającego minimalny czas realizacji trajektorii ruchu. Profil prędkości określa chwilową wartość prędkości stycznej do toru ruchu (prędkość posuwu) w każdej chwili czasu, która wykorzystywana jest w procesie interpolacji zadanego toru ruchu opisanego krzywą NURBS. W procesie minimalizacji czasu realizacji zadanego toru ruchu należy uwzględnić fizyczne ograniczenia rzeczywistego obiektu jakim jest maszyna wieloosiowa. Ograniczenia obejmują maksymalne wartości prędkości, przyspieszenia i zrywu w każdej z osi maszyny oraz błąd konturu. Problem ten można sformułować następująco:

$$min(T_{\sum}) = min\left(\int_0^L \frac{ds}{v(s)}\right) \tag{4.1}$$

gdzie:
$T_{\sum}$ - całkowity czas realizacji zadanej trajektorii ruchu,
$v(s)$ - profil prędkości posuwu w funkcji położenia na krzywej reprezentującej tor ruchu,
L - całkowita długość zadanego toru ruchu.

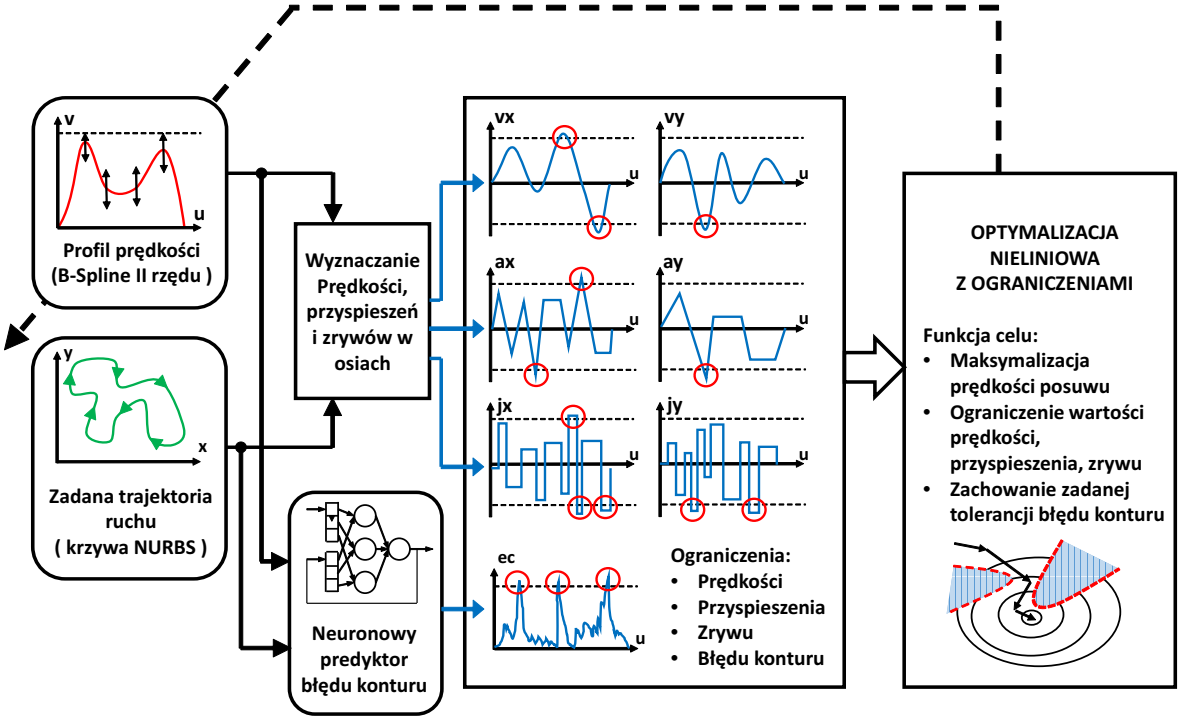
Powyższy problem podlega następującym ograniczeniom (dla maszyny dwuosiowej):

$$\begin{aligned} &v_x(s) < v_{x_{max}}, \quad v_y(s) < v_{y_{max}} \\ &a_x(s) < a_{x_{max}}, \quad a_y(s) < a_{y_{max}} \\ &j_x(s) < j_{x_{max}}, \quad j_y(s) < j_{y_{max}} \\ &\epsilon_c(s) < E_{c_{max}} \end{aligned} \tag{4.2}$$

gdzie:
$v_x(s), v_y(s), a_x(s), a_y(s), j_x(s), j_y(s)$ - chwilowe wartości prędkości, przyspieszenia i zrywu w osiach maszyny w funkcji położenia na krzywej reprezentującej tor ruchu,
$v_{x_{max}}, v_{y_{max}}, a_{x_{max}}, a_{y_{max}}, j_{x_{max}}, j_{y_{max}}$ - maksymalne dopuszczalne wartości prędkości, przyspieszenia i zrywu w osiach maszyny,

$\epsilon_c(s)$ - chwilowa wartość błędu konturu,
$E_{c_{max}}$ - maksymalna dopuszczalna wartość błędu konturu.

W celu zagwarantowania spełnienia ograniczeń związanych z błędem konturu, konieczne jest wykorzystanie modelu matematycznego odwzorowującego dynamikę układu posuwu maszyny. Zastosowanie modelu matematycznego do przewidywania błędu konturu w sterowaniu prędkością posuwu maszyny mieści proponowany algorytm w klasie algorytmów sterowania predykcyjnego. W algorytmach sterowania predykcyjnego model obiektu wykorzystywany jest do przewidzenia odpowiedzi obiektu na zadany sygnał sterujący. Na podstawie odpowiedzi modelu sygnał sterujący jest iteracyjnie modyfikowany, w celu minimalizacji lub maksymalizacji odpowiedniej funkcji celu. Do wyznaczenia odpowiedniego sygnału sterującego stosowane są algorytmy optymalizacji z ograniczeniami. W przypadku gdy zastosowany model jest modelem nieliniowym (np. sztuczna sieć neuronowa), stosowany algorytm optymalizacyjny musi być również algorytmem optymalizacji nieliniowej. Schemat blokowy algorytmu wyznaczającego optymalny profil prędkości posuwu, zaproponowanego w tej pracy, przedstawiono na rysunku 4.1.



Rysunek 4.1. Schemat blokowy algorytmu optymalizacji prędkości posuwu

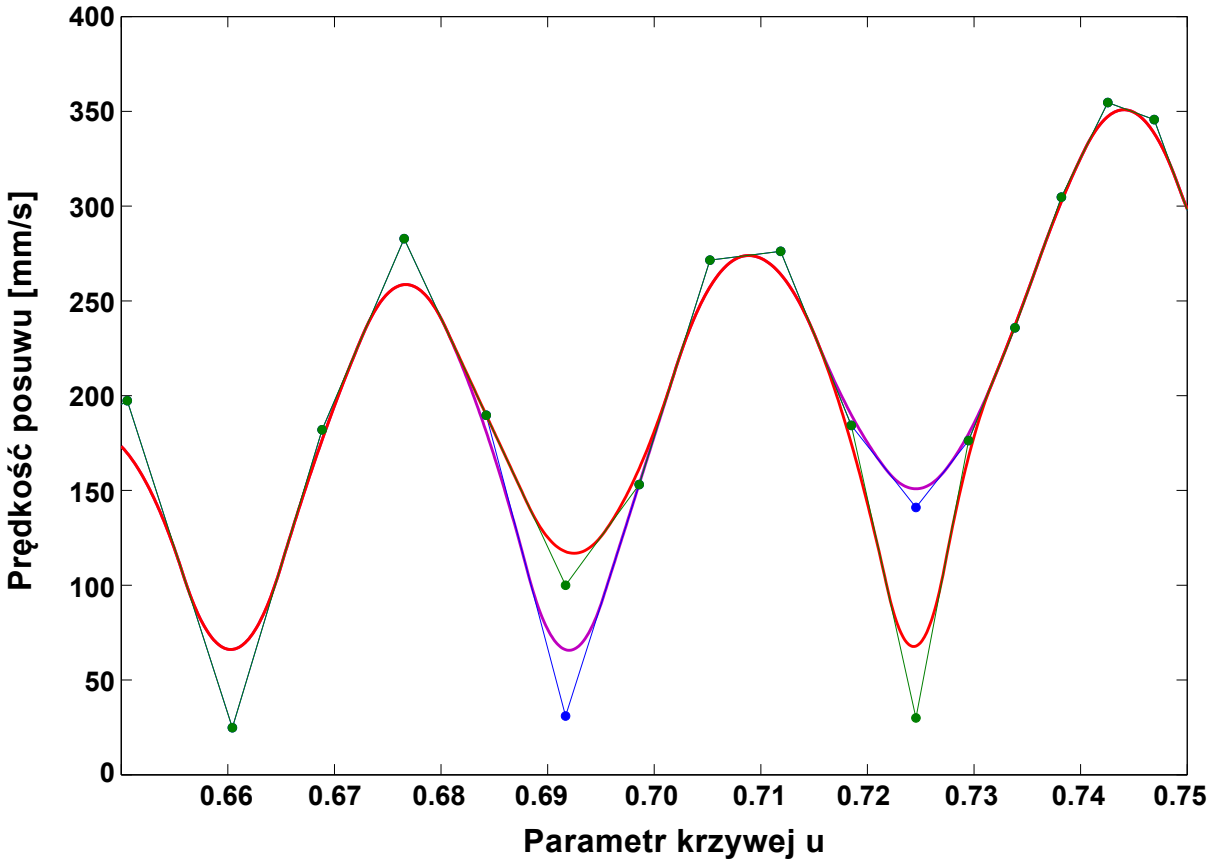
Na wejście algorytmu wprowadzony jest zadany tor ruchu w postaci krzywej NURBS, początkowy profil prędkości oraz wartości ograniczeń. W każdej iteracji algorytmu sprawdzana jest wartość funkcji celu odpowiadającej kryterium minimalizacji czasu realizacji

trajektorii ruchu oraz wartość funkcji ograniczeń zależnej od wartości wielkości podlegających ograniczeniom. Wartości kinematyczne (prędkości, przyspieszenia i zrywy) wyznaczane są na podstawie aktualnego profilu prędkości i zadanego toru ruchu. Błąd konturu wyznaczany jest za pomocą neuronowego predyktora błędu konturu opisanego w rozdziale 3. Algorytm optymalizacji nieliniowej z ograniczeniami iteracyjnie modyfikuje profil prędkości do czasu uzyskania profilu prędkości zapewniającego minimalny czas realizacji trajektorii ruchu, bez naruszenia zadanych ograniczeń.

W tradycyjnych algorytmach sterowania predykcyjnego zmiennymi optymalizowanymi są poszczególne wartości próbek sygnału sterującego w pewnym skończonym horyzoncie czasowym. W każdej chwili czasu należy też uwzględnić ograniczenia narzucone na układ. W opisywanym tu problemie sygnałem sterującym jest prędkość posuwu. Stosunkowo niska dynamika sygnału sterującego w porównaniu z krótkim okresem próbkowania (typowo nie więcej niż 1 ms) powoduje, że tradycyjne podejście wymagałoby długiego horyzontu czasowego. Skutkowałoby to dużą liczbą optymalizowanych zmiennych i ograniczeń co powodowałoby, że problem optymalizacyjny byłby bardzo wymagający obliczeniowo.

Alternatywnym rozwiązaniem jest zdefiniowanie sygnału sterującego w formie funkcji (najczęściej wielomianowej) i optymalizacja parametrów tej funkcji zamiast pojedynczych próbek. Zdefiniowanie profilu prędkości w postaci wielomianu pozwala na znaczne ograniczenie ilości optymalizowanych zmiennych. Ten sposób reprezentacji profilu prędkości został zaproponowany przez Altinasa [3]. W wyżej wymienionej pracy profil prędkości jest wielomianem sklejanym piątego rzędu, którego argumentem jest czas. Zmiennymi optymalizowanymi są parametry wielomianu. Wadą tego rozwiązania jest trudny do przewidzenia wpływ zmiany parametrów na kształt profilu prędkości, co dodatkowo komplikuje problem optymalizacyjny. Idea wykorzystania wielomianu do reprezentacji profilu prędkości została rozwinięta przez Sencera [85], który do tego celu wykorzystał krzywą b-sklejaną. Ten sposób opisu profilu prędkości został przyjęty w niniejszej pracy i posłużył jako punkt wyjścia do opracowania proponowanego algorytmu optymalizacji prędkości. Głównym atutem tak zdefiniowanego profilu prędkości jest intuicyjne powiązanie zmiennych optymalizacyjnych, którymi są wartości punktów kontrolnych, z jej kształtem. Modyfikacja danego punktu kontrolnego wpływa jedynie na kształt krzywej w bezpośrednim otoczeniu tego punktu tj. dla fragmentu krzywej zawartego pomiędzy węzłami u_i i u_{i+k+1} dla punktu kontrolnego P_i i krzywej rzędu k. Na rys. 4.2 przedstawiono fragmenty dwóch przykładowych profili prędkości zdefiniowanych jako krzywe B-sklejane oraz ich wielokąty kontrolne.

Oś odciętych wykresu stanowi parametr krzywej NURBS opisującej zadany tor ruchu i jednocześnie parametr krzywej B-sklejanej opisującej profil prędkości. Oś rzędnych stanowi wartość prędkości posuwu. Wartości parametru u odpowiadające poszczególnym punktom



Rysunek 4.2. Fragment profili prędkości rzędu drugiego opisanych krzywą b-sklejaną.

kontrolnym profilu prędkości określają punkty Greville'a [35], wyznaczane ze wzoru:

$$G_i = \frac{1}{k}\left(\sum_{j=i+1}^{i+k+1} u_j\right) \tag{4.3}$$

gdzie:
k - rząd krzywej B-sklejanej opisującej profil prędkości,
u_j - j-ty węzeł.

Obydwa profile przedstawione na wykresie różnią się jedynie wartościami dwóch punktów kontrolnych. Dla krzywej drugiego rzędu wpływ zmiany jednego punktu kontrolnego powoduje zmianę kształtu profilu prędkości w zakresie co najwyżej dwóch sąsiednich punktów.

W przyjętym rozwiązaniu argumentem funkcji profilu prędkości nie jest czas lecz bezwymiarowy parametr, oznaczany jako u, jednoznacznie określający położenie na krzywej NURBS, definiującej zadany tor ruchu. Tak zdefiniowany profil prędkości określa wartość prędkości posuwu w zależności od aktualnego położenia na torze ruchu a nie czasu. Jest to istotne ponieważ czas, w którym realizowane są poszczególne fragmenty trajektorii ruchu

ulega zmianie wraz ze zmianą profilu prędkości. Zastosowanie parametru u jako argumentu funkcji profilu prędkości powoduje uproszczenie problemu optymalizacji prędkości posuwu. W niniejszej pracy przyjęto, że profil prędkości jest krzywą b-sklejaną rzędu 2, gdyż jest to najniższy możliwy rząd, który zapewnia ograniczenie wartości zrywu. Możliwe jest zastosowanie wyższych rzędów krzywej, a przez to ograniczenie pochodnych położenia wyższych rzędów. W praktycznych zastosowaniach ograniczanie pochodnych położenia wyższych niż zryw spotyka się rzadko, gdyż prowadzi to do pogorszenia dynamiki ruchu maszyny bez wyraźnych korzyści.

W praktycznych zastosowaniach zadany tor ruchu może mieć znaczną długość. Optymalizacja całego profilu prędkości może nie być możliwa, ze względu na znaczną ilość zmiennych i ograniczeń, które należałoby uwzględnić w procesie optymalizacji. Z tego względu zastosowano skończony horyzont predykcji. W danym czasie optymalizacji podlega tylko fragment profilu prędkości, tak aby ograniczyć ilość zmiennych i ograniczeń, które są jednocześnie rozpatrywane w procesie optymalizacji. Po zakończeniu optymalizacji danego fragmentu horyzont predykcji i optymalizacji ulega przesunięciu tak aby objąć kolejny fragment profilu prędkości. Nowy i poprzednio optymalizowany horyzont częściowo się pokrywają tak aby umożliwić korektę poprzednio optymalizowanych punktów kontrolnych uwzględniając nowe dane w przesuniętym horyzoncie predykcji. Pierwsze dwa punkty kontrolne optymalizowanego fragmentu profilu prędkości nie podlegają optymalizacji a ich wartości są równe wartościom dwóch ostatnich punktów poprzednio optymalizowanego fragmentu. W przypadku początkowego fragmentu trajektorii ruchu ustalony jest jedynie pierwszy punkt kontrolny, którego wartość wynosi 0 co wymusza zerową początkową prędkość posuwu. Pozostałe punkty kontrolne mogą być swobodnie modyfikowane przez algorytm optymalizacyjny. Wyjątkiem jest końcowy fragment trajektorii ruchu gdzie wartość ostatniego punktu kontrolnego równa jest 0, wymuszając zerową końcową wartość prędkości posuwu. W algorytmie optymalizacji prędkości posuwu przyjęto długość horyzontu wynoszącą 40 punktów kontrolnych przy czym 10 pierwszych punktów kontrolnych stanowi 10 punktów poprzedniego horyzontu. Wartości te zostały dobrane eksperymentalnie tak aby uzyskać odpowiedni kompromis pomiędzy szybkością obliczeń a precyzją modyfikacji profilu prędkości. Jako odległość pomiędzy punktami kontrolnymi przyjęto 7,5 mm.

4.1. Funkcje celu i ograniczeń w algorytmie optymalizacji prędkości

Celem algorytmu optymalizacji prędkości posuwu jest minimalizacja czasu realizacji trajektorii ruchu. Problem można przeformułować jako problem maksymalizacji prędkości posuwu. W opracowanym rozwiązaniu przyjęto następującą postać funkcji celu:

$$F_c = \frac{1}{N} \sum_{i=1}^{N} \left(1 - \frac{v_i}{V_{max}}\right)^2 \tag{4.4}$$

gdzie:

V_{max} - maksymalna wartość prędkości posuwu,

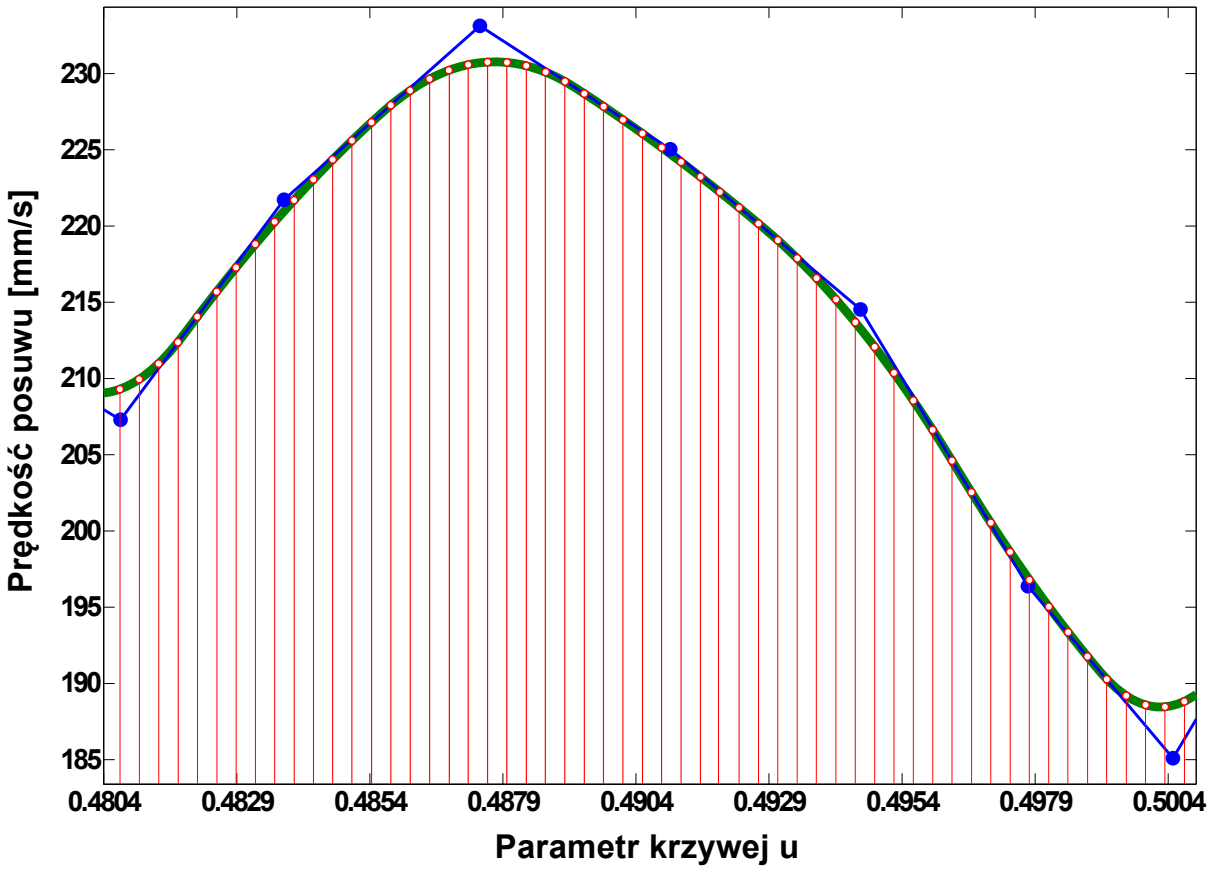
v_i - wartość i-tego punktu kontrolnego definiującego profil prędkości, zmienianego w zakresie $(0, V_{max})$,

N - ilość punktów kontrolnych optymalizowanych w danym horyzoncie predykcji.

Funkcja celu została sformułowana tak aby jej wartość spadała wraz ze zwiększaniem sumarycznej wartości punktów kontrolnych. Ma to na celu dostosowanie postaci funkcji celu do wymagań większości algorytmów optymalizacyjnych, które minimalizują funkcję celu. Wartości zmiennych optymalizacyjnych (wartości punktów kontrolnych profilu prędkości) zostały unormowane w zakresie od 0 do 1 w celu uniezależnienia wartości funkcji celu od maksymalnej wartości prędkości posuwu, zależnej od parametrów danej maszyny.

Oprócz minimalizowanej funkcji celu w procesie optymalizacji uwzględnione są ograniczenia wielkości kinematycznych, czyli prędkości, przyspieszenia i zrywu poszczególnych osi oraz ograniczenie błędu konturu. Minimalizując funkcję celu algorytm optymalizacyjny zapewnia nie przekraczanie narzuconych ograniczeń. W celu zredukowania ilości ograniczeń podlegających optymalizacji, ograniczenia nie są sprawdzane w każdym punkcie horyzontu predykcji lecz w określonych punktach odpowiadających zawsze tym samym wartościom parametru krzywej. Punkty rozmieszczone są równomiernie względem długości krzywej. W opisywanym algorytmie przyjęto odległość pomiędzy punktami wynoszącą 0.25 mm. Odległość ta została dobrana eksperymentalnie w celu zapewnienia najlepszego kompromisu pomiędzy wymaganiami obliczeniowymi a skutecznością uwzględniania ograniczeń. Fragment profilu prędkości z zaznaczonymi punktami weryfikacji ograniczeń przedstawiono na rysunku 4.3.

Funkcja ograniczeń zwraca ciąg wartości dla każdego punktu weryfikacji ograniczeń i każdej wielkości podlegającej ograniczeniom. Wielkości kinematyczne w osiach zależą wyłącznie od aktualnej prędkości posuwu i geometrii zadanego toru ruchu. Dla ustalonego profilu prędkości wartości ograniczeń kinematycznych są więc funkcjami parametru krzywej. W celu wyznaczenia wartości kinematycznych należy w pierwszej kolejności wyznaczyć wartości prędkości, przyspieszenia i zrywu stycznych do toru ruchu dla danej wartości parametru u. Wartość prędkości stycznej wyznaczana jest bezpośrednio jako wartość krzywej B-Sklejanej reprezentującej profil prędkości $v(u)$. Wartości przyspieszenia $a_t(u)$ i zrywu $j_t(u)$ są pochodnymi wartości profilu prędkości względem czasu nie są więc tożsame z pierwszą



Rysunek 4.3. Fragment profilu prędkości z zaznaczonymi punktami weryfikacji ograniczeń (kolor czerwony).

i drugą pochodną krzywej względem parametru, otrzymywanych wykorzystując algorytm de Boor'a. Przyspieszenie i zryw styczne do toru ruchu wyznacza się za pomocą wzorów:

$$a_t(u) = \frac{v(u)\frac{dv(u)}{du}}{\sigma(u)} \tag{4.5}$$

$$j_t(u) = \frac{v^2(u)}{\sigma^2(u)}\frac{d^2v(u)}{du^2} - \frac{\sigma' v(u)}{\sigma^2 a_t(u)} + \frac{a_t^2(u)}{v(u)} \tag{4.6}$$

Wielkość σ w powyższych wzorach oznacza prędkość parametryczną opisującą zmianę położenia w funkcji parametru krzywej NURBS opisującej zadany tor ruchu. Prędkość parametryczną oraz jej pierwszą pochodną σ' wyznacza się ze wzorów:

$$\sigma(u) = \left\| \frac{\mathbf{dC(u)}}{\mathbf{du}} \right\| \tag{4.7}$$

$$\sigma'(u) = \frac{\frac{\mathbf{dC(u)}}{\mathbf{du}} \cdot \frac{\mathbf{d^2C(u)}}{\mathbf{du^2}}}{\sigma(u)} \tag{4.8}$$

Prędkości, przyspieszenia i zrywy w osiach wyznaczane są ze wzorów:

$$\mathbf{v_a}(\mathbf{u}) = \begin{bmatrix} v_x(u) \\ v_y(u) \end{bmatrix} = \frac{\mathbf{dC(u)}}{\mathbf{ds}} v(u) \tag{4.9}$$

$$\mathbf{a_a}(\mathbf{u}) = \begin{bmatrix} a_x(u) \\ a_y(u) \end{bmatrix} = \frac{\mathbf{d^2C(u)}}{\mathbf{ds^2}} v^2(u) + \frac{\mathbf{dC(u)}}{\mathbf{ds}} a_t(u) \tag{4.10}$$

$$\mathbf{j_a}(\mathbf{u}) = \begin{bmatrix} j_x(u) \\ j_y(u) \end{bmatrix} = \frac{\mathbf{d^3C(u)}}{\mathbf{ds^3}} v^3(u) + 3\frac{\mathbf{d^2C(u)}}{\mathbf{ds^2}} a_t(u)v(u) + \frac{\mathbf{dC(u)}}{\mathbf{ds}} j_t(u) \tag{4.11}$$

Wyrażenia $\frac{\mathbf{dC(u)}}{\mathbf{ds}}$, $\frac{\mathbf{d^2C(u)}}{\mathbf{ds^2}}$, $\frac{\mathbf{d^2C(u)}}{\mathbf{ds^2}}$ są kolejnymi pochodnymi krzywej NURBS $C(u)$ względem długości krzywoliniowej. Powyższe pochodne są tożsame z pochodnymi krzywej NURBS tylko wtedy gdy krzywa parametryzowana jest długością krzywoliniową (ang. arc-length parametrization). Z reguły stosowana jest parametryzacja bezwymiarowa (parametr $u \in [0, 1]$). W takim przypadku pochodne krzywej względem długości krzywoliniowej wyrażają się wzorami:

$$\frac{\mathbf{dC(u)}}{\mathbf{ds}} = \frac{\mathbf{dC(u)}}{\mathbf{du}} \frac{du}{ds} \tag{4.12}$$

$$\frac{\mathbf{d^2C(u)}}{\mathbf{ds^2}} = \frac{\mathbf{d^2C(u)}}{\mathbf{du^2}} \left(\frac{du}{ds}\right)^2 + \frac{\mathbf{dC(u)}}{\mathbf{du}} \frac{d^2u}{ds^2} \tag{4.13}$$

$$\frac{\mathbf{d^3C(u)}}{\mathbf{ds^3}} = \frac{\mathbf{d^3C(u)}}{\mathbf{du^3}} \left(\frac{du}{ds}\right)^3 + 3\frac{\mathbf{d^2C(u)}}{\mathbf{du^2}} \frac{d^2u}{ds^2} \frac{du}{ds} + \frac{\mathbf{dC(u)}}{\mathbf{du}} \frac{d^3u}{ds^3} \tag{4.14}$$

Pochodne parametru krzywej względem długości krzywoliniowej wyznaczane są ze wzorów:

$$\frac{du}{ds} = \frac{1}{\left\| \frac{\mathbf{dC(u)}}{\mathbf{du}} \right\|} \tag{4.15}$$

$$\frac{d^2u}{ds^2} = -\frac{\mathbf{dC(u)}}{\mathbf{du}} \cdot \frac{\mathbf{d^2C(u)}}{\mathbf{du^2}} \left(\frac{du}{ds}\right)^4 \tag{4.16}$$

$$\begin{aligned} \frac{d^3u}{ds^3} = 4 \left(\frac{\mathbf{dC(u)}}{\mathbf{du}} \cdot \frac{\mathbf{d^2C(u)}}{\mathbf{du^2}}\right)^2 \left(\frac{du}{ds}\right)^7 - \\ \left(\frac{\mathbf{d^2C(u)}}{\mathbf{du^2}} \cdot \frac{\mathbf{d^2C(u)}}{\mathbf{du^2}} + \frac{\mathbf{dC(u)}}{\mathbf{du}} \cdot \frac{\mathbf{d^3C(u)}}{\mathbf{du^3}}\right) \left(\frac{du}{ds}\right)^5 \end{aligned} \tag{4.17}$$

Oprócz wartości kinematycznych ograniczeniu podczas optymalizacji podlega błąd konturu. Błąd konturu wyznaczany jest przez neuronowy predyktor błędu konturu opisany w poprzednim rozdziale. Ponieważ do wyznaczania błędu konturu wykorzystywane są dynamiczne sieci neuronowe, jego wartość zależy zarówno od aktualnego parametru u jak również od stanów opóźnień na wejściach sieci. W celu wyznaczenia błędu konturu dla aktualnie optymalizowanego fragmentu profilu prędkości przed rozpoczęciem procesu optymalizacji przeprowadzana jest predykcja dla wcześniejszych fragmentów profilu prędkości. Ma to na celu zainicjowanie bloków opóźniających dynamicznych sieci neuronowych odpowiednimi wartościami początkowymi. W trakcie trwania procesu optymalizacji konieczne jest wielokrotne przeprowadzenie procesu predykcji błędu konturu. Za każdym razem sieci neuronowe inicjalizowane są wartościami początkowymi wyznaczonymi przed rozpoczęciem procesu optymalizacji dzięki czemu predykcja błędu konturu realizowana jest jedynie dla aktualnie optymalizowanego fragmentu profilu prędkości.

Predykcja błędu konturu przy każdym wywołaniu funkcji ograniczeń polega na przeprowadzeniu interpolacji krzywej NURBS dla optymalizowanego horyzontu predykcji. W wyniku interpolacji dla każdej z osi otrzymywane są szeregi czasowe prędkości zadanych. Szeregi te podawane są na wejścia neuronowego predyktora błędu konturu przedstawionego w rozdziale 3. Na wyjściu predyktora otrzymywany jest szereg czasowy błędu konturu. Każdemu elementowi szeregu czasowego przyporządkowana jest wartość parametru krzywej otrzymana w trakcie interpolacji. Błąd konturu musi zostać wyznaczony w ściśle określonych punktach weryfikacji ograniczeń, które z reguły nie pokrywają się z elementami szeregu czasowego. Aby uzyskać wartość błędu konturu w zadanych punktach weryfikacji ograniczeń do punktów szeregu czasowego dopasowywany jest wielomian interpolacyjny trzeciego rzędu. Jako argument interpolowanego wielomianu przyjmuje się wartości parametru odpowiadające poszczególnym punktom szeregu czasowego. Otrzymany w ten sposób wielomian sklejany pozwala na wyznaczenie wartości błędu konturu dla dowolnej wartości parametru krzywej w optymalizowanym przedziale. Interpolacja przeprowadzana jest metodą Akimy [1, 2]. Jest to metoda interpolacji lokalnej, która nie wymaga rozwiązywania dużych układów równań liniowych dzięki czemu wymagania obliczeniowe są niewielkie. Lokalny charakter metody pozwala również na dokonywanie interpolacji tylko w przedziałach pomiędzy punktami szeregu czasowego, które zawierają punkty weryfikacji ograniczeń.

Wyznaczone wartości prędkości, przyspieszeń i zrywów w osiach oraz błędu konturu są przeskalowane względem odpowiednich wartości maksymalnych. Zastosowane skalowanie powoduje, że w przypadku przekroczenia przez ograniczaną wielkość wartości maksymalnej ograniczenie przyjmuje wartość dodatnią. W przeciwnym wypadku, ograniczenia mają wartość ujemną. Ograniczenia są dzięki temu niezależne od przyjętych wartości maksymalnych

dla poszczególnych wielkości. Wartości ograniczeń zwracane przez funkcje ograniczeń dla przypadku dwuosiowego przyjmują postać:

$$
\begin{aligned}
c_{vx}(u_i) &= \frac{v_x(u_i)}{V_{x_{max}}} - 1 \\
c_{vy}(u_i) &= \frac{v_y(u_i)}{V_{y_{max}}} - 1 \\
c_{ax}(u_i) &= \frac{a_x(u_i)}{A_{x_{max}}} - 1 \\
c_{ay}(u_i) &= \frac{a_y(u_i)}{A_{y_{max}}} - 1 \\
c_{jx}(u_i) &= \frac{j_x(u_i)}{J_{x_{max}}} - 1 \\
c_{jy}(u_i) &= \frac{j_y(u_i)}{J_{y_{max}}} - 1 \\
c_{ce}(u_i) &= \frac{\epsilon_c(u_i)}{E_{c_{max}}} - 1
\end{aligned}
\tag{4.18}
$$

gdzie:
$c_{vx}(u_i)$, $c_{vy}(u_i)$, $c_{ax}(u_i)$, $c_{ay}(u_i)$, $c_{jx}(u_i)$, $c_{jy}(u_i)$, $c_{ce}(u_i)$ - wartości ograniczeń zwracane przez funkcję ograniczeń dla każdego punktu weryfikacji ograniczeń u_i,
$v_x(u_i)$, $v_y(u_i)$, $a_x(u_i)$, $a_y(u_i)$, $j_x(u_i)$, $j_y(u_i)$, $\epsilon_c(u_i)$ - prędkości, przyspieszeń i zrywów w osiach oraz błędu konturu dla każdego punktu weryfikacji ograniczeń u_i,
$V_{x_{max}}$, $V_{y_{max}}$, $A_{x_{max}}$, $A_{y_{max}}$, $J_{x_{max}}$, $J_{y_{max}}$, $E_{c_{max}}$ - maksymalne wartości prędkości, przyspieszeń i zrywów w osiach oraz błędu konturu.

4.2. Optymalizacja prędkości posuwu rojem cząstek

W celu rozwiązania problemu optymalizacji prędkości przedstawionego w poprzednim podrozdziale konieczne jest zastosowanie algorytmów optymalizacji nieliniowej do minimalizacji funkcji celu. Najczęściej stosowanymi algorytmami optymalizacyjnymi są algorytmy wykorzystujące gradient funkcji celu i ograniczeń (algorytmy gradientowe) np. algorytm sekwencyjnego programowania kwadratowego (SQP)[85]. W przypadku zastosowania metod gradientowych, konieczne jest wyznaczenie gradientu funkcji celu i funkcji ograniczeń. Zastosowanie w funkcji ograniczeń modelu dynamicznego, niezbędnego do uwzględnienia błędu konturu powoduje, że trudno jest znaleźć gradient funkcji ograniczeń w postaci analitycznej. Możliwa jest aproksymacja gradientu metodą różnic skończonych lecz wymaga to wyznaczania dodatkowych wartości funkcji celu i funkcji ograniczeń. Ponadto błędy w wyznaczaniu gradientu powodują mniej skuteczne działanie algorytmu

optymalizacyjnego i mogą doprowadzić do błędnych rozwiązań. Istotnym problemem w przypadku wszystkich gradientowych algorytmów optymalizacyjnych jest przedwczesne zakończenie optymalizacji w optimum lokalnym. Optimum lokalne jest rozwiązaniem lepszym od wszystkich pobliskich rozwiązań lecz może być znacznie gorsze od najlepszego możliwego rozwiązania (optimum globalnego). Jakość uzyskanego rozwiązania zależy silnie od rozwiązania początkowego (ang. initial guess), od którego algorytm zaczyna poszukiwania rozwiązania optymalnego. W problemie optymalizacji prędkości trudno jest podać dobre rozwiązanie początkowe, które będzie zbliżone do rozwiązania optymalnego i jednocześnie będzie spełniać narzucone ograniczenia ze względu na dużą ilość zmiennych i ograniczeń, które należy uwzględnić.

Alternatywne podejście do problemu optymalizacji prędkości posuwu w postaci zastosowania programowania liniowego zaproponowano w [34, 40]. Algorytmy programowania liniowego, które były jednymi z pierwszych algorytmów optymalizacyjnych, charakteryzują się dużą efektywnością w znajdowaniu rozwiązań i nie wymagają znajomości gradientu funkcji celu i ograniczeń. Warunkiem koniecznym ich zastosowania jest liniowa postać funkcji celu i funkcji ograniczeń. Wymusza to przyjęcie szeregu założeń upraszczających, linearyzujących problem optymalizacyjny. Przede wszystkim nie jest możliwe wykorzystanie modelu nieliniowego do predykcji błędów nadążania osi. Nie jest również możliwe bezpośrednie uwzględnienie ograniczeń błędu konturu, który jest nieliniową funkcją błędów nadążania w przypadku toru ruchu zdefiniowanego jako krzywa NURBS. Z tego względu to podejście nie może zostać zastosowane w niniejszej pracy.

Mając na uwadze powyższe fakty w niniejszej pracy postanowiono wykorzystać algorytmy optymalizacji globalnej nie wymagające wyznaczania gradientu. Algorytmy optymalizacji globalnej starają się znaleźć najlepsze możliwe rozwiązanie poprzez ekstensywne przeszukiwanie przestrzeni rozwiązań bez wykorzystywania informacji o gradiencie funkcji celu i ograniczeń. Z tego względu algorytmy te są odporne na przedwczesne zatrzymanie optymalizacji w optimum lokalnym. Algorytmy tego typu nie wymagają również podania rozwiązania początkowego przez użytkownika. Zamiast tego pewna liczba rozwiązań początkowych generowana jest losowo przez algorytm optymalizacyjny. W wielu algorytmach optymalizacyjnych rozwiązanie początkowe istotnie wpływa na jakość końcowego rozwiązania. Część algorytmów wymaga również by podane rozwiązanie początkowe spełniało narzucone ograniczenia. Powoduje to, że podanie odpowiednio dobrego rozwiązania początkowego jest często konieczne aby problem rozwiązać. Dla rozpatrywanego problemu optymalizacji prędkości posuwu, wygenerowanie takiego rozwiązania początkowego, które spełniałoby wszystkie ograniczenia jest trudne. W algorytmach optymalizacji globalnej ten problem nie występuje, gdyż rozwiązania początkowe generowane są automatycznie przez algorytm.

Z pośród wielu algorytmów optymalizacji globalnej do najpopularniejszych należy algorytm optymalizacji rojem cząstek (ang. Particle Swarm Optimization, PSO). Popularność algorytmu wynika z prostoty działania, łatwości implementacji i skuteczności w znajdowaniu rozwiązań wielu praktycznych problemów. Z tego względu w algorytmie optymalizacji prędkości opracowanym przez autora niniejszej rozprawy wykorzystano algorytm PSO. Algorytm PSO zaproponowany przez Kennedy'ego i Eberhart'a [50, 24, 51] miał początkowo służyć symulacji ruchu ławic ryb i stad ptaków poszukujących pożywienia. W trakcie prac odkryto, że algorytm skutecznie minimalizuje złożone funkcje wielu zmiennych z wieloma minimami lokalnymi. Działanie algorytmu polega na przeszukiwaniu przestrzeni rozwiązań przez wirtualne „cząstki" w celu znalezienia najlepszego możliwego rozwiązania (minimum funkcji celu). Każda cząstka charakteryzuje się pozycją, której współrzędne są zmiennymi optymalizacyjnymi oraz wektorem prędkości. Początkowa pozycja i prędkość są generowane losowo, z reguły przy użyciu rozkładu równomiernego, w całym zakresie przestrzeni zmiennych optymalizacyjnych. Z tego względu w algorytmie PSO nie ma konieczności podania rozwiązania początkowego, w odróżnieniu od większości klasycznych algorytmów. W każdej iteracji algorytmu modyfikacji ulega prędkość oraz pozycja każdej z cząstek. W oryginalnej wersji algorytmu współrzędne wektorów prędkość i pozycji każdej cząstki wyznaczane są na podstawie równań:

$$v^*_{i,j} = v_{i,j} + \phi_1 \cdot rand^U_{(0,1)} \cdot (p_{i,j} - x_{i,j}) + \phi_2 \cdot rand^U_{(0,1)} \cdot (g_i - x_{i,j}) \tag{4.19}$$

$$x^*_{i,j} = x_{i,j} + v^*_{i,j} \tag{4.20}$$

gdzie:
$v^*_{i,j}$ - nowa i-ta współrzędna wektora prędkości j-tej cząstki
$v_{i,j}$ - aktualna i-ta współrzędna wektora prędkości j-tej cząstki
$x^*_{i,j}$ - nowa i-ta współrzędna położenia j-tej cząstki
$x_{i,j}$ - aktualna i-ta współrzędna położenia 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-ta współrzędna najlepszej osobistej pozycji j-tej cząstki
g_i - i-ta współrzędna najlepszej globalnej pozycji

Jakość poszczególnych rozwiązań określana jest przez wartość funkcji celu. Im mniejsza wartość funkcji celu odpowiada danemu rozwiązaniu tym rozwiązanie to uznawane jest za lepsze. Każdej cząstce przypisana jest „najlepsza osobista pozycja" (ang. personal best

position, pbest), która jest najlepszą dotychczasową pozycją tej cząstki w przestrzeni rozwiązań. Jeżeli w kolejnych iteracjach algorytmu nowa pozycja cząstki będzie odpowiadała mniejszej wartości funkcji celu niż dotychczasowa pozycja pbest, to nowo znaleziona pozycja staje się nową pozycją pbest. Jeżeli nowa pozycja odpowiada wartości funkcji celu niższej niż jakakolwiek ze znalezionych dotąd pozycji przez którąkolwiek z cząstek roju to ta pozycja zapisywana jest jako „najlepsza globalna pozycja" (ang. global best position, gbest). Pozycje te są wykorzystywane do wyznaczenia wektorów prędkości i pozycji każdej cząstki w kolejnych iteracjach algorytmu. Nowe pozycje cząstek nie są zawsze lepsze od pozycji wyznaczonych w poprzednich iteracjach, w odróżnieniu od algorytmów gradientowych, w których kolejne rozwiązania wyznaczana są na podstawie spadku wartości funkcji celu. Z tego względu algorytm PSO jest znacznie bardziej odporny na przedwczesne zatrzymanie w minimum lokalnym funkcji celu.

Cząstki poruszają się początkowo losowo dokonując eksploracji przestrzeni rozwiązań. Prędkość modyfikowana jest w sposób stochastyczny w każdej iteracji powodując przyciąganie do najlepszych znalezionych pozycji. Stopniowo rój przechodzi od fazy eksploracji do fazy eksploatacji. Oznacza to, że krąg poszukiwań optimum zawęża się co skutkuje coraz dokładniejszym przeszukiwaniem coraz węższego obszaru przestrzeni rozwiązań. Po pewnej liczbie iteracji rój skupia się w jednym, bardzo wąskim obszarze. Najlepsze znalezione rozwiązanie (pozycja gbest) jest wtedy zwracane przez algorytm jako rozwiązanie końcowe.

O tym jak szybko algorytm zbiega do rozwiązania i jak dokładnie przeszukiwana jest przestrzeń rozwiązań decydują wartości parametrów algorytmu. Parametrami algorytmu są współczynniki przyspieszenia oraz ilość cząstek (rozmiar roju). W większości przyjmuje się $\phi_1 = \phi_2 = 2.0$ oraz liczbę cząstek w zakresie od 20 do 50 [78]. Współczynniki przyspieszenia określają kompromis pomiędzy eksploracją a eksploatacją. Modyfikacja ich wartości prowadzi do wydłużenia bądź skrócenia poszczególnych faz. Zły dobór parametrów może doprowadzić do niekontrolowanego wzrostu prędkości a w konsekwencji braku zbieżności roju. Z tego powodu stosuje się ograniczenia maksymalnej wartości prędkości. Wartość tego ograniczenia jest zależna od danego problemu i stanowi kolejny parametr wpływający na zachowanie roju.

W celu zapobieżenia problemowi nadmiernego wzrostu prędkości Shi i Eberhart [87] zaproponowali dodanie do równania zmiany prędkości współczynnika bezwładności ω. Równanie 4.19 przyjmuje wtedy postać:

$$v^*_{i,j} = \omega \cdot v_{i,j} + \phi_1 \cdot rand^U_{(0,1)} \cdot (p_{i,j} - x_{i,j}) + \phi_2 \cdot rand^U_{(0,1)} \cdot (g_i - x_{i,j}) \tag{4.21}$$

Wprowadzenie współczynnika bezwładności umożliwia wprowadzenie dodatkowego, obok ograniczania prędkości, sposobu na tłumienie nadmiernych wartości prędkości. Jednocześnie wprowadzenie inercji zmniejsza wpływ ograniczenia prędkości na dynamikę roju przez co

przy odpowiedniej wartości współczynnika parametry te można pominąć. Współczynnik ω przyjmuje z reguły wartości z zakresu [0,1]. Wysoka wartość tego współczynnika (np. 0.9) powoduje, że rój cząstek bardzo wolno zbiega do ostatecznego rozwiązania poświęcając wiele iteracji na eksplorację przestrzeni rozwiązań. Niska wartość współczynnika (np. 0.4) powoduje, że cząstki szybko zbiegają do jednego rozwiązania lecz zdolność roju do eksploracji jest mocno ograniczona. Z tego względu w wielu pracach współczynnik ω zaimplementowano jako liniowo malejącą funkcję liczby iteracji algorytmu PSO. W literaturze zaproponowano wiele innych, bardziej złożonych, metod adaptacji współczynnika bezwładności m.in poprzez wprowadzenie nieliniowej zależności współczynnika od liczby iteracji [11], stochastyczną modyfikację współczynnika [113] czy też adaptację współczynnika z wykorzystaniem logiki rozmytej [88]. Nowe warianty algorytmów prezentowały z reguły szybszą zbieżność lub większą skuteczność w znajdowaniu minimum funkcji celu niż algorytm klasyczny lecz jednocześnie wprowadzały dodatkowe parametry, których wartość należy dobrać dla danego problemu.

Wprowadzenie nowego współczynnika, nie rozwiązało problemu trudnej od przewidzenia interakcji poszczególnych parametrów i ich łącznego wpływu na zachowanie roju. Zły dobór współczynników może doprowadzić do utraty stabilności roju. Problem jest jeszcze bardziej złożony jeżeli współczynnik inercji podlega adaptacji. Analiza dynamiczna zachowania roju przeprowadzona przez Clerc'a [16] skłoniła go do przeformułowania równania modyfikacji prędkości i wprowadzenia tzw. współczynnika ścisku (ang. constriction coefficient) oznaczanego jako χ. Równanie (4.19) przyjmuje wtedy postać:

$$v^*_{i,j} = \chi \left(v_{i,j} + \phi_{c1} \cdot rand^U_{(0,1)} \cdot (p_{i,j} - x_{i,j}) + \phi_{c2} \cdot rand^U_{(0,1)} \cdot (g_i - x_{i,j}) \right) \tag{4.22}$$

$$\chi = \frac{2}{\phi - 2 + \sqrt{\phi^2 - 4\phi}} \tag{4.23}$$

$$\phi = \phi_1 + \phi_2 \tag{4.24}$$

Clerc zaproponował wartość współczynników $\phi_{c1} = \phi_{c2} = 2.05$ co skutkuje wartością współczynnika ścisku równą w przybliżeniu 0.7298. Jest to równoważne przyjęciu w równaniu 4.21 współczynnika bezwładności $\omega = \chi \approx 0.7298$ oraz współczynników przyspieszenia $\phi_1 = \phi_2 \approx 1.4962$. Clerc udowodnił, że z powyższymi parametrami rój nigdy nie utraci stabilności, co czyni ograniczenie prędkości zbędnym. Dalsze badania [25] wykazały jednak, że ograniczenie prędkości cząstek równe maksymalnej wartości danej zmiennej ($V_{max,i} = X_{max,i}$) zapewnia lepsze właściwości dynamiczne roju cząstek niż brak ograniczenia. Wariant

algorytmu PSO wykorzystujący współczynnik ścisku z parametrami zaproponowanymi przez Clerc'a z ograniczeniem prędkości równym ograniczeniu wartości zmiennej uznawany jest obecnie za standardowy, podstawowy wariant algorytmu.

Standardowy algorytm PSO ma istotną wadę. Jeżeli aktualna pozycja cząstki jest globalną najlepszą pozycją (gbest) wtedy nowa wartość prędkości zależy jedynie od poprzedniej prędkości i współczynnika inercji lub ścisku. Ponieważ wartość współczynnika inercji/ścisku jest mniejsza niż jeden, prędkość cząstki będzie się zmniejszać aż osiągnie wartość bardzo bliską zera. Jeżeli pozostałe cząstki zbiegną do tej zatrzymanej cząstki cały rój przestanie się poruszać. Zjawisko to nosi nazwę stagnacji roju. Nie ma gwarancji, że najlepsza globalna pozycja roju, który uległ stagnacji jest minimum lokalnym funkcji celu. Szczegółowa analiza i dowód tej właściwości algorytmu PSO zawarto w pracy [99]. Standardowy algorytm PSO został początkowo wybrany przez autora jako składnik proponowanego algorytmu optymalizacji prędkości. W trakcie prób okazało się, że dla problemu optymalizacji prędkości występuje zjawisko stagnacji a otrzymywane rozwiązania są niekiedy dalekie od optymalnych. W celu poprawy jakości otrzymywanych rozwiązań konieczne stało się wykorzystanie zmodyfikowanego wariantu algorytmu PSO.

W celu rozwiązania problemu stagnacji roju van den Bergh zaproponował nowy wariant algorytmu PSO nazwany optymalizacją rojem cząstek z gwarantowaną zbieżnością (ang. Guaranteed Convergence Particle Swarm Optimization, GCPSO) [100]. Modyfikacje wprowadzone w nowym wariancie algorytmu PSO mają na celu zagwarantowanie, że rój odnajdzie rozwiązanie, które będzie rzeczywistym minimum funkcji celu (lokalnym lub globalnym) a nie jedynie ostatnim najlepszym punktem odnalezionym przez rój w momencie wystąpienia stagnacji. Modyfikacja polega na wprowadzeniu nowego równania modyfikacji prędkości dla cząstki, która odnalazła ostatnią globalnie najlepszą pozycję (najlepsza cząstka). Równanie ma postać:

$$v^*_{i,\tau} = \chi \cdot v_{i,\tau} - x_{i,\tau} + g_i + \rho \cdot rand^U_{(-1,1)} \tag{4.25}$$

gdzie:
τ - indeks najlepszej cząstki czyli tej, która odnalazła najlepszą globalną pozycję
ρ - promień obszaru przeszukiwania losowego wokół pozycji najlepszej cząstki
$rand^U_{(-1,1)}$ - liczba pseudolosowa o rozkładzie równomiernym z przedziału [-1,1]

Nowe równanie modyfikacji prędkości powoduje, że wokół globalnej najlepszej pozycji przeprowadzone jest przeszukiwanie losowe w celu zapobieżenia przedwczesnemu zatrzymaniu algorytmu w punkcie nie będącym minimum funkcji celu. Wartość współczynnika skalującego ρ, wyznacza promień obszaru wokół najlepszej globalnej pozycji, w którym

prowadzone jest przeszukiwanie losowe. Za każdym razem gdy w wyniku przeszukiwania losowego odnalezione jest lepsze rozwiązanie (odpowiadające mniejszej wartości funkcji celu) inkrementowana jest liczba sukcesów N_s. Jeżeli w danej iteracji nie uda się znaleźć lepszego rozwiązania inkrementowana jest liczba porażek N_f. W przypadku, gdy liczba sukcesów bądź porażek przekroczy wartości progowe $N_{s,MAX}$ i $N_{f,MAX}$ współczynnik ρ jest modyfikowany zgodnie z następującą regułą:

$$\rho^* = \begin{cases} 2\rho & dla\ N_s > N_{s,MAX} \\ 0.5\rho & dla\ N_f > N_{f,MAX} \\ \rho & dla\ pozostaych \end{cases} \tag{4.26}$$

Algorytm GCPSO został zaimplementowany przez autora. Po przeprowadzeniu wielu prób uzyskano wyniki wskazujące na znaczącą poprawę w porównaniu do standardowej wersji algorytmu PSO. Problem zbiegania algorytmu do punktów nie będących minimami przestał występować. Jednocześnie zdecydowanej poprawie uległa jakość otrzymywanych rozwiązań. Wartości progowe liczby sukcesów i porażek przyjęto zgodnie z sugestiami autora algorytmu jako $N_{s,MAX} = 15$ i $N_{f,MAX} = 5$. Jednocześnie przyjęto wartość minimalną współczynnika ρ równą 10^{-5} oraz maksymalną równą 1. Wraz z modyfikacją współczynnika ρ zerowana jest wartość sukcesów bądź porażek w zależności od tego, która z nich przekroczyła założoną wartość progową. Dla wszystkich testowanych wariantów algorytmu przyjęto empirycznie dobraną liczbę cząstek równą 25.

Wśród wielu odmian algorytmu PSO prezentowanych w literaturze, istotną część stanowią odmiany ze zmodyfikowaną topologią wymiany informacji wewnątrz roju. Standardowa topologia zakłada występowanie jednej globalnej najlepszej pozycji, którą „znają" wszystkie cząstki stąd taką topologię nazywa się globalną. Odmienne topologie zakładają, że grupy cząstek mają swoje lokalne najlepsze pozycje dlatego takie topologie noszą nazwę lokalnych. Przyjęcie takiej topologii spowalnia przepływ informacji wewnątrz roju, co w założeniu powoduje wolniejszą zbieżność roju ale równocześnie dokładniejsze przeszukiwanie przestrzeni rozwiązań. To z kolei skutkuje większym prawdopodobieństwem znalezienia minimum globalnego złożonych funkcji celu. Z prób przeprowadzonych nad zastosowaniem topologii lokalnej z algorytmem GCPSO oraz wyników przedstawionych w [75] wynika, że zastosowanie topologii lokalnej nie powoduje znalezienia lepszych rozwiązań niż te uzyskane przez algorytm o topologii globalnej. Przeprowadzono również testy wykorzystując warianty algorytmu, w których wartość parametrów (współczynniki przyspieszenia i bezwładności lub ścisku) podlegała adaptacji. Próby nie wykazały istotnej wyższości tych wariantów nad algorytmem GCPSO. Z tego względu w prezentowanym algorytmie optymalizacji

prędkości wykorzystano wyłącznie algorytm GCPSO o topologii globalnej bez adaptacji współczynników.

Osobnym problemem występującym w algorytmach PSO jest problem kryterium zakończenia optymalizacji. Najprostszym a zarazem najczęściej stosowanym kryterium jest maksymalna liczba iteracji algorytmu lub ewaluacji funkcji celu. W praktyce trudno jest znaleźć optymalną liczbę iteracji zapewniającą zbieżność algorytmu. Zbyt mała liczba iteracji powoduje, że rój cząstek może nie zdążyć zbiec do minimum funkcji celu. Zwiększenie liczby iteracji zwiększa prawdopodobieństwo zbieżności roju lecz czas obliczeń ulega niepotrzebnemu wydłużeniu. W klasycznych algorytmach optymalizacyjnych jako kryterium zbieżności przyjmuje się spadek wartości gradientu (w przypadku algorytmów gradientowych) lub różnicy wartości funkcji celu pomiędzy kolejnymi iteracjami poniżej pewnej założonej wartości. Proste zastosowanie tego kryterium w algorytmie PSO nie jest możliwe ze względu na występowanie wielu potencjalnych rozwiązań, dla których wartości funkcji celu nie zawsze ulegają zmniejszeniu pomiędzy iteracjami. Zastosowanie tego kryterium tylko do najlepszej globalnej pozycji również nie jest dobrym rozwiązaniem ponieważ wartość funkcji celu odpowiadająca tej pozycji często nie ulega zmianie przez wiele iteracji.

Znaczenie lepszym rozwiązaniem problemu terminacji algorytmu jest kryterium zbieżności uwzględniające stan roju. W niniejszej pracy przyjęto kryterium zbieżności wykorzystujące średnią odległość każdej z cząstek od najlepszej globalnej pozycji. Wartość ta wyznaczana w każdej iteracji algorytmu zgodnie ze wzorem:

$$\bar{d}_{gbest} = \frac{1}{N} \sum_{i=0}^{N-1} \sum_{j=0}^{D-1} (x_{i,j} - g_j)^2 \tag{4.27}$$

gdzie:
N - ilość cząstek w roju
D - wymiar przestrzeni rozwiązań

Optymalizacja jest przerywana jeżeli jednocześnie spełnione są poniższe dwie poniższe nierówności:

$$\left|\bar{d}_{gbest}(k) - \bar{d}_{gbest}(k - k_{stop})\right| \leq DTOL \cdot \bar{d}_{gbest}(k - k_{stop}) \tag{4.28}$$

$$F(\bar{d}_{gbest}(k - k_{stop})) - F(\bar{d}_{gbest}(k)) \leq FTOL \cdot F(\bar{d}_{gbest}(k - k_{stop})) \tag{4.29}$$

Spełnienie pierwszego kryterium oznacza, że średnia odległość cząstek od najlepszej globalnej

pozycji nie ulega znaczącej zmianie przez określoną liczbę iteracji. Spełnienie drugiego kryterium oznacza, że przez pewną liczbę iteracji wartość funkcji celu odpowiadająca najlepszej globalnej pozycji nie uległa znaczącemu zmniejszeniu. Jednoczesne spełnienie obu kryteriów świadczy o tym, że w ciągu założonej liczby iteracji rój poczynił względnie niewielkie postępy co skutkuje zakończeniem procesu optymalizacji. W obydwu kryteriach postępy roju mierzone są w sposób względny dzięki czemu kryteria mają zastosowanie niezależnie od początkowego rozkładu cząstek. Parametrami kryterium zakończenia optymalizacji jest liczba iteracji k_{stop} pomiędzy którymi badane są postępy roju oraz współczynniki $DTOL$ i $FTOL$ determinujące względne wartości progowe dla których nierówności są spełnione. Do rozwiązania problemu optymalizacji prędkości przyjęto wartości parametrów $k_{stop} = 5$, $DTOL = 5^{-3}$, $FTOL = 5^{-3}$. Wartości te dobrano empirycznie tak aby uzyskać zadowalające rozwiązania w wciągu kilkudziesięciu iteracji algorytmu. Niższe wartości współczynników powodują uzyskanie dokładniejszego rozwiązania kosztem jednak większej liczby iteracji.

Cechą charakterystyczną algorytmu PSO jest stosunkowo szybkie znajdowanie otoczenia rozwiązania problemu optymalizacyjnego. Poprawa dokładności rozwiązania (eksploatacja) zajmuje jednak o wiele więcej iteracji. Stochastyczny charakter algorytmu PSO dodatkowo utrudnia dokładną minimalizację funkcji celu. Z tego względu w prezentowanym algorytmie optymalizacji prędkości zastosowano podejście hybrydowe. Rozwiązanie zwrócone przez algorytm optymalizacji rojem cząstek jest wykorzystywane do zainicjowania algorytmu optymalizacji lokalnej, który jest w stanie znaleźć dokładniejsze rozwiązanie znacznie szybciej niż sam algorytm PSO. W niniejszej pracy zastosowano bezgradientowy algorytm optymalizacji lokalnej BOBYQA autorstwa M. J. D. Powell'a [79]. Połączenie obu algorytmów pozwala na uzyskanie dokładniejszych rozwiązań szybciej niż ma to miejsce w przypadku zastosowania samego algorytmu PSO.

Algorytm optymalizacji rojem cząstek w swojej podstawowej formie, podobnie jak inne algorytmy wieloagentowe, nie umożliwia rozwiązywania zadań optymalizacji z ograniczeniami. Istnieje szereg metod pozwalających na uwzględnienie ograniczeń przez algorytm PSO, których przegląd zawarto w [17, 58]. Jedną z często stosowanych metod jest metoda funkcji kary. Metoda polega na rozszerzeniu funkcji celu $F(\mathbf{x})$ o sumę kar pochodzących od każdego aktywnego ograniczenia. Najczęściej stosowaną funkcją kary jest funkcja zwracająca kwadrat ograniczenia dla wartości ograniczenia większej od 0 (tzn. gdy ograniczenie jest przekroczone) oraz wartość 0 w przeciwnym wypadku. Algorytm minimalizacyjny minimalizuje więc zarówno wartość funkcji celu jak również wartość przekraczanych ograniczeń. Suma kar pomnożona jest przez współczynnik proporcjonalności ρ, który reguluje stopień w jakim kara wpływa na wartość rozszerzonej funkcji celu $\Phi(\mathbf{x})$.

$$\Phi(\mathbf{x}) = F(\mathbf{x}) + \rho \sum_{i=1}^{M} \left(\max\left\{0, c_i(\mathbf{x})\right\}\right)^2 \tag{4.30}$$

Problemem w metodzie funkcji kary jest odpowiedni dobór współczynników proporcjonalności kary dla każdego z ograniczeń. Zbyt duża wartość współczynnika zwiększa prawdopodobieństwo zatrzymania algorytmu w minimum lokalnym. Zbyt mała wartość może spowodować pominięcie danego ograniczenia w procesie optymalizacji. Proces optymalizacji powtarzany jest kilkukrotnie dla coraz większych wartości współczynnika proporcjonalności ρ. Aby uzyskać rozwiązanie będące rozwiązaniem pierwotnego problemu optymalizacji z ograniczeniami współczynnik proporcjonalności musi zmierzać do nieskończoności, co czyni problem źle uwarunkowanym. W praktyce wielkość współczynnika musi być ograniczona a otrzymane rozwiązanie jest jedynie przybliżeniem rozwiązania oryginalnego zadania optymalizacji z ograniczeniami.

Powyższy problem został rozwiązany w metodzie Rozszerzonego Lagranżianu (ang. Augmented Lagrangian Method, ALM), której zastosowanie w połączeniu algorytmem PSO zaproponowali Kai i Sedlaczek [83]. Algorytm rozszerzonego Lagranżjanu działa podobnie do metody kwadratowej funkcji kary, różnica polega na odmiennym sformułowaniu rozszerzonej funkcji celu. Rozszerzona funkcja celu, zwana rozszerzonym Lagranżjanem Powell'a-Hestenes'a-Rockafellar'a, przyjmuje postać: [82]:

$$L(\mathbf{x}) = F(\mathbf{x}) + \frac{\rho}{2} \sum_{i=1}^{M} \left(\max\left\{0, c_i(\mathbf{x}) + \frac{\lambda_i}{\rho}\right\}\right)^2 \tag{4.31}$$

W porównaniu do rozszerzonej funkcji celu z kwadratową funkcją kary rozszerzony Lagranżjan posiada dodatkowe parametry λ_i, przyporządkowane niezależnie do każdego ograniczenia, zwane mnożnikami Lagrange'a. Dzięki wprowadzeniu mnożników rozwiązanie zadania minimalizacji bez ograniczeń rozszerzonego Lagranżjanu jest dokładnym rozwiązaniem zadania minimalizacji funkcji celu z ograniczeniami dla skończonych wartości współczynnika kary ρ. Optymalne wartości mnożników Lagrange'a λ_i oraz współczynnika kary ρ są nieznane i muszą zostać wyznaczone w toku procesu optymalizacji.

Optymalizacja prędkości posuwu metodą rozszerzonego lagranżjanu odbywa się w dwóch pętlach - zewnętrzej i wewnętrznej. W pętli wewnętrznej, dla ustalonych wartości parametrów λ_i oraz ρ, minimalizowany jest rozszerzony Lagranżjan przy użyciu algorytmu PSO a następnie BOBYQA. W zewnętrznej pętli wartości parametrów są aktualizowane po czym proces minimalizacji jest ponawiany. Kolejne minimalizacje rozszerzonego lagranżjanu wykorzystują jako wartość początkową poprzednie rozwiązanie. Ponieważ kolejne rozwiązania otrzymywane w kolejnych iteracjach zewnętrznej pętli algorytmu znajdują się w tym samym otoczeniu przestrzeni zmiennych optymalizacyjnych algorytm PSO stosuje się jedynie w pierwszej

iteracji. W kolejnych iteracjach stosowany jest jedynie algorytm optymalizacji lokalnej BOBYQA. Rozwiązanie to pozwala na istotne przyspieszenie procesu optymalizacji gdyż pomija się fazę eklsploracji przestrzeni rozwiązań, która w kolejnych iteracjach algorytmu jest zbędna.
Mnożniki Lagrange'a aktualizowane są zgodnie ze wzorem:

$$\lambda_i^* = \min\left\{\max\left\{0, \lambda_i + \rho c_i(\mathbf{x})\right\}, 10^{20}\right\} \tag{4.32}$$

Początkowa wartość mnożników wynosi 0. Przyjęto maksymalną wartość mnożników Lagrange'a równą 10^{20} a minimalną równą 0.

Wartość współczynnika kary w pierwszej iteracji wyznaczana jest ze wzoru [9]:

$$\rho^1 = max\left\{10^{-6}, min\left\{10, \frac{2|F(\mathbf{x_0})|}{\sum\limits_{i=0}^{M} c_i^2(\mathbf{x_0})}\right\}\right\} \tag{4.33}$$

W kolejnych iteracjach współczynnik kary wyznaczany jest zgodnie ze wzorem:

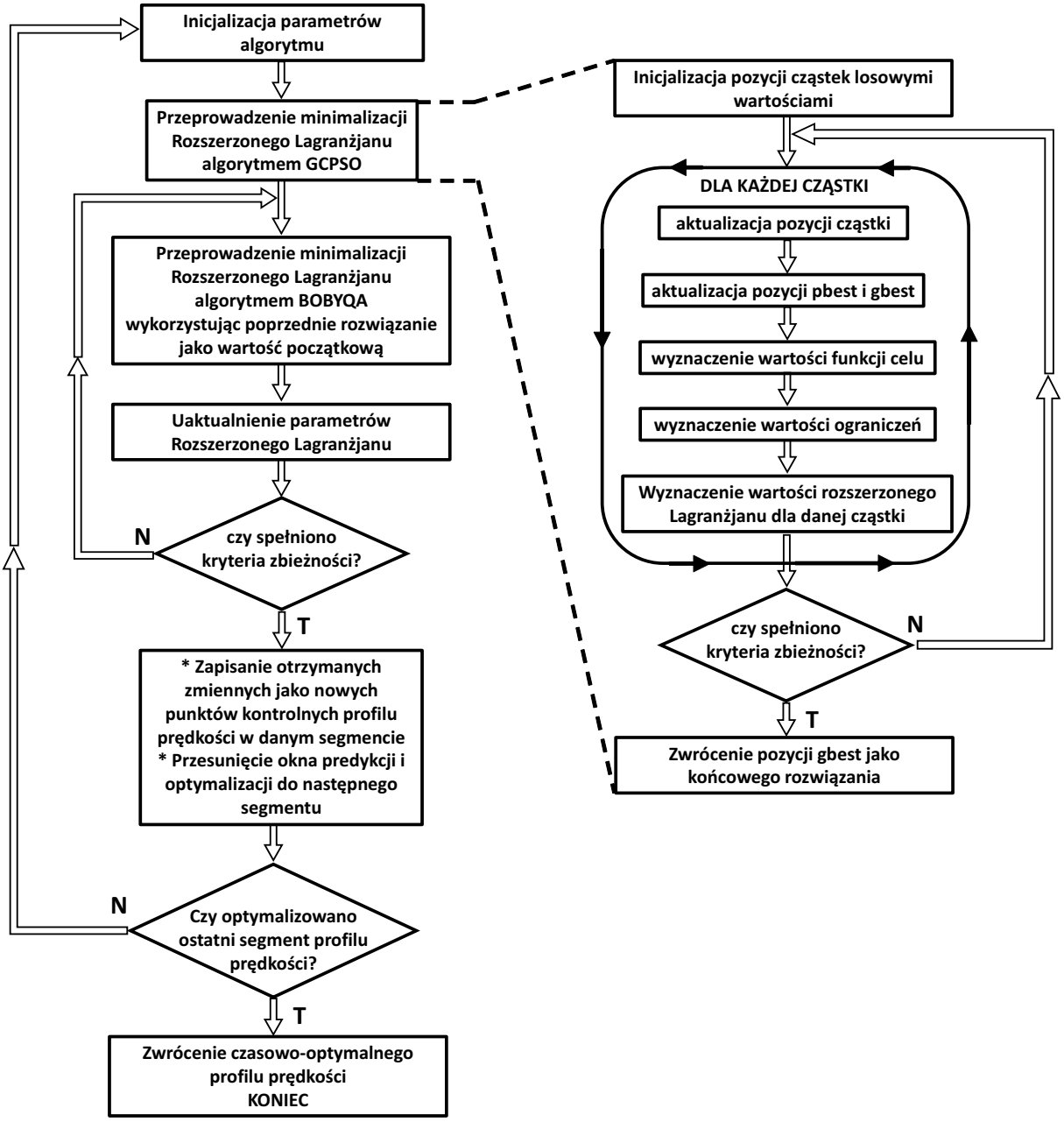
$$\rho^* = \begin{cases} 10\rho & ICM^* > 0.5 \cdot ICM \\ \rho & ICM^* \leq 0.5 \cdot ICM \end{cases} \tag{4.34}$$

$$ICM^* = \max\left\{ICM, \left|\max\left\{c_i(\mathbf{x}), -\frac{\lambda_i}{\rho}\right\}\right|\right\} \tag{4.35}$$

Parametr ICM (ang. Infeasibility - Complementarity Measure) jest miarą spełnienia ograniczeń problemu optymalizacyjnego. Wartość $ICM = 0$ oznacza, że wszystkie ograniczenia są spełnione. W przypadku gdy nowa wartość współczynnika ICM jest większa niż połowa wartości tego współczynnika w poprzedniej iteracji, współczynnik kary ρ zwiększany jest dziesięciokrotnie. Współczynnik ten jest również kryterium terminacji zewnętrznej pętli algorytmu. Dla problemu optymalizacji prędkości posuwu przyjęto, że algorytm optymalizacji prędkości kończy działanie dla $ICM < 10^{-12}$ lub dla liczby iteracji zewnętrznej pętli algorytmu równej 40. Szczegółowe omówienie metody rozszerzonego Lagranżjanu od strony teoretycznej, zostało zawarte m.in w [7].

Algortym GCPSO w połączeniu z metodą rozszerzonego Lagranżjanu został zaimplementowany w całości przez autora w języku C i wykorzystany do rozwiązania problemu optymalizacji prędkości posuwu dla toru ruchu zdefiniowanego w formie krzywej NURBS. Implementując algorytm BOBYQA zaadaptowano otwarty kod źródłowy algorytmu

autorstwa V. Oikonena [74]. Implementując metodę rozszerzonego Lagranżjanu bazowano na kodzie źródłowym otwartego oprogramowania optymalizacyjnego NLOPT autorstwa S.G. Johnsona [49]. Schemat przepływowy zaimplementowanego algorytmu przedstawiono na rysunku 4.4. Fragmenty kodu źródłowego zaimplementowanego algorytmu przedstawiono w załączniku 1.



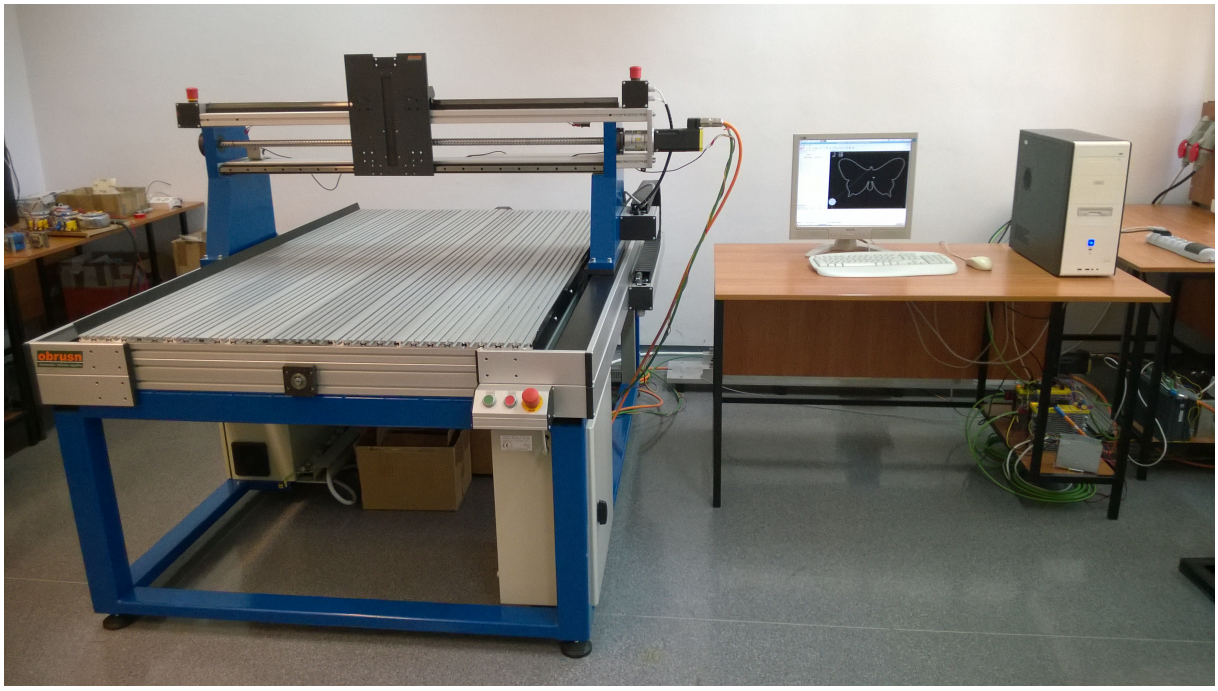
Rysunek 4.4. Schemat przepływowy algorytmu optymalizacji prędkości posuwu

W rozdziale przedstawiono algorytm optymalizacji prędkości posuwu wykorzystujący algorytm optymalizacji rojem cząstek z gwarantowaną zbieżnością(GCPSO) oraz metodę rozszerzonego Lagranżjanu. Przedstawiono funkcję celu oraz ograniczeń wykorzystaną w procesie optymalizacji prędkości posuwu. Omówiono zastosowany wariant algorytmu PSO

wraz z uzasadnieniem wyboru tego wariantu. Przedstawiono zastosowane kryteria terminacji. Omówiono metodę rozszerzonego Lagranżjanu umożliwiającą uwzględnienie ograniczeń przez algorytm PSO.

5. Stanowisko badawcze

W celu przeprowadzenia badań doświadczalnych algorytmu optymalizacyjnego konieczne było zbudowanie układu sterowania CNC. Zbudowano stanowisko badawcze złożone z dwuosiowej maszyny CNC wyprodukowanej przez PIAP OBRUSN Toruń oraz sterownika CNC bazującego na komputerze PC z systemem czasu rzeczywistego. Zdjęcie stanowiska przedstawiono na rysunku 5.1.



Rysunek 5.1. Zdjęcie stanowiska badawczego

Każda z osi maszyny napędzana jest silnikiem PMSM o momencie nominalnym 1,5 Nm. Ruch obrotowy silników zamieniany jest na ruch postępowy osi poprzez przekładnie śrubowe toczne (śruby kulowe) o skoku 10 mm/obr. Pomiar położenia realizowany jest poprzez przetwornik obrotowo-impulsowy o rozdzielczości 2500 imp./obr. wbudowany w silnik. Silniki sterowane są za pomocą serwonapędów Baldor Microflex e100. Serwonapędy wykorzystują klasyczną, kaskadową strukturę układu regulacji z regulatorami położenia (P), prędkości (PI) i prądu (PI) ze sprzężeniami w przód od prędkości i przyspieszenia.

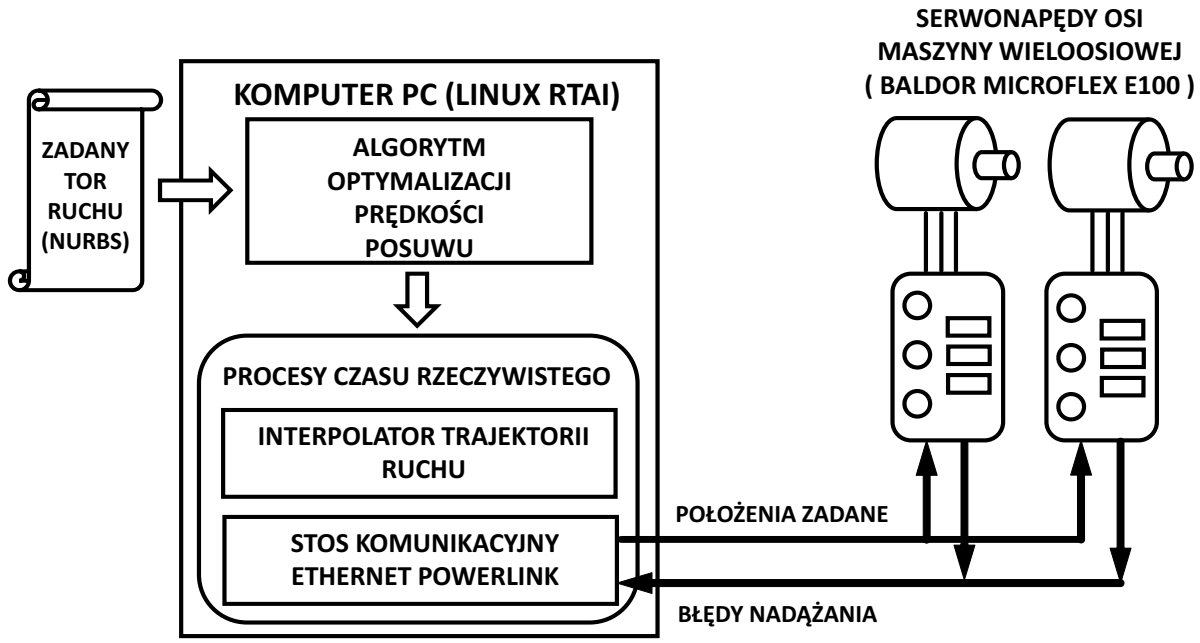
Celem autora było zbudowanie otwartego układu sterowania, niezależnego od rozwiązań sprzętowych w postaci kart rozszerzeń czy sterowników mikroprocesorowych. Wynikało

to z konieczności zaimplementowania opracowanego algorytmu optymalizacyjnego, który pełni rolę generatora trajektorii ruchu oraz interpolatora trajektorii ruchu dla torów ruchu opisanych krzywymi NURBS. W komercyjnych układach sterowania wprowadzenie takich modyfikacji nie jest możliwe. Z tego względu postanowiono zaimplementować opracowane algorytmy na standardowym komputerze klasy PC. Ponieważ w układach sterowania CNC kluczowym problemem jest zadawanie położeń do serwonapędów w stałych odstępach czasu, konieczne było zastosowanie systemu operacyjnego czasu rzeczywistego. Taki system jest w stanie zapewnić realizację programów z determinizmem czasowym, nieosiągalnym dla standardowych systemów operacyjnych takich jak Windows czy Linux. Dzieje się tak dlatego, że standardowy system operacyjny zaprojektowany jest do realizowania wielu zadań zapewniając im równy dostęp do zasobów sprzętowych. Celem systemu czasu rzeczywistego jest realizacja kluczowych zadań o wysokim priorytecie (np. sterowania) w sposób deterministyczny tzn. w możliwie stałym przewidywalnym czasie. Układ sterowania maszyn wieloosiowych pełniący rolę stanowiska badawczego w niniejszej pracy pracuje pod kontrolą systemu Linux RTAI. Jest to darmowy i otwarty system operacyjny będący modyfikacją standardowego Linux'a, zapewniający wysoki determinizm czasowy (tzw. twardy czas rzeczywisty)[5].

W systemie czasu rzeczywistego zainstalowano oprogramowanie LinuxCNC, będące darmowym programem do sterowania maszynami wieloosiowymi. Oprogramowanie to nie umożliwia sterowania nowoczesnymi serwonapędami poprzez sieć Ethernet oraz nie posiada możliwości interpolacji trajektorii ruchu opisanej krzywymi NURBS. Umożliwia natomiast pisanie własnych modułów czasu rzeczywistego w interfejsie programistycznym o nazwie HAL i zarządzanie nimi co znacznie upraszcza korzystanie z możliwości systemu Linux RTAI. Autor postanowił wykorzystać te możliwości do implementacji algorytmu generującego czasowo-optymalną trajektorię ruchu oraz oprogramowania realizującego tą trajektorię w czasie rzeczywistym.

Algorytm optymalizacyjny zaimplementowano w języku C jako proces przestrzeni użytkownika systemu operacyjnego tzn. proces, który nie pracuje w reżimie czasu rzeczywistego. Jako proces czasu rzeczywistego zaimplementowany został interpolator czasowo-optymalnej trajektorii ruchu dla toru zdefiniowanego w formie krzywych NURBS oraz stos komunikacyjny protokołu Ethernet Powerlink. Obydwa powyższe moduły programowe również zaimplementowano w języku C. Schemat blokowy opracowanego układu sterowania przedstawiono na rysunku 5.2.

Standard Ethernet Powerlink pozwala na komunikację z szeroką gamą komercyjnych serwonapędów różnych producentów (np. ABB, Parker). Protokół Powerlink działa z wykorzystaniem mechanizmu szczelin czasowych, który polega na przydzieleniu każdemu z węzłów sieci (np. napędowi) ściśle określonego okna komunikacyjnego, w którym możliwa jest



Rysunek 5.2. Schemat blokowy stanowiska badawczego

wymiana danych pomiędzy węzłem a sterownikiem nadrzędnym. Wymiana danych z węzłami odbywa się cyklicznie. Zastosowany mechanizm zapewnia niski rozrzut czasowy co pozwala na zminimalizowanie błędów spowodowanych opóźnieniami komunikacyjnymi i kolizjami danych w sieci. Zaimplementowany stos komunikacyjny oparto na otwartym stosie openPowerlink przeznaczonym dla standardowego systemu Linux. Aby możliwe było działanie w reżimie czasu rzeczywistego pod kontrolą systemu Linux RTAI konieczne było wprowadzenie zmian w kodzie źródłowym stosu. Zmiany te obejmowały:

— opracowanie sterowników czasu rzeczywistego dla użytkowanych kart sieciowych
— zastąpienie przerwań Linuxa przerwaniami RTAI, które podlegają priorytetom systemu czasu rzeczywistego
— zastosowanie cyklicznych przerwań zegara systemowego RTAI cechującego się niskim rozrzutem czasowym
— zastąpienie buforów programowych bezpośrednimi wywołaniami funkcyjnymi w celu zmniejszenia czasu reakcji

Ponadto konieczne było wprowadzenie zmian w konfiguracji programowej i sprzętowej komputera na którym zaimplementowano omawiane oprogramowanie. Zmiany te obejmowały:

— rozdzielenie procesów czasu rzeczywistego i procesów użytkownika, wraz ze stowarzyszonymi przerwaniami, na osobne rdzenie procesora
— przyporządkowanie karcie sieciowej odpowiedzialnej za komunikację z napędami przerwania o najwyższym możliwym priorytecie

— wyłączenie zbędnych funkcji i sprzętu np. DMA dla dysków twardych czy karty dźwiękowej
— wyłączenie niemaskowalnych sprzętowych przerwań serwisowych na płycie głównej
— wyłączenie funkcji skalowania zegara procesora i funkcji skalowania prędkości wentylatorów w zależności od obciążenia procesora

Powyższe zmiany miały na celu zminimalizowania wpływu sprzętu (poprzez niemaskowalne przerwania sprzętowe) oraz oprogramowania przestrzeni użytkownika na czasowo-krytyczne procesy komunikacji i interpolacji trajektorii ruchu. Wprowadzone zmiany pozwoliły na uzyskanie wysokiego determinizmu komunikacji. Zmierzone rozrzuty czasowe cyklu komunikacyjnego wynoszące kilka mikrosekund dla okresu cyklu komunikacyjnego wynoszącego 1 milisekundę, pozwalają pominąć wpływ komunikacji na błędy nadążania mierzone przez serwonapędy.

Oprócz zmian mających na celu minimalizację rozrzutów czasowych, konieczne było zaadaptowanie stosu do pracy zgodnej ze standardem CANOpen CIA402, w którym pracuje większość serwonapędów wykorzystujących do komunikacji standardy Ethernet Powerlink lub EtherCAT. Zmiany te polegały na zdefiniowaniu po stronie sterownika adresów danych przewidzianych przez standard i odpowiadających np. wartości zadanej czy błędowi nadążania. Ponadto zaimplementowano maszynę stanów zgodną ze standardem CIA402, zarządzającą inicjalizacją napędu, procedurą bazowania i obsługi błędów. Dzięki temu stos może się komunikować z każdym napędem obsługującym ten powszechnie stosowany standard. Szczegółowy opis przedstawionego układu sterowania i wprowadzonych modyfikacji wraz z wynikami pomiarów rozrzutu czasowego został zawarty przez autora w artykule [33].

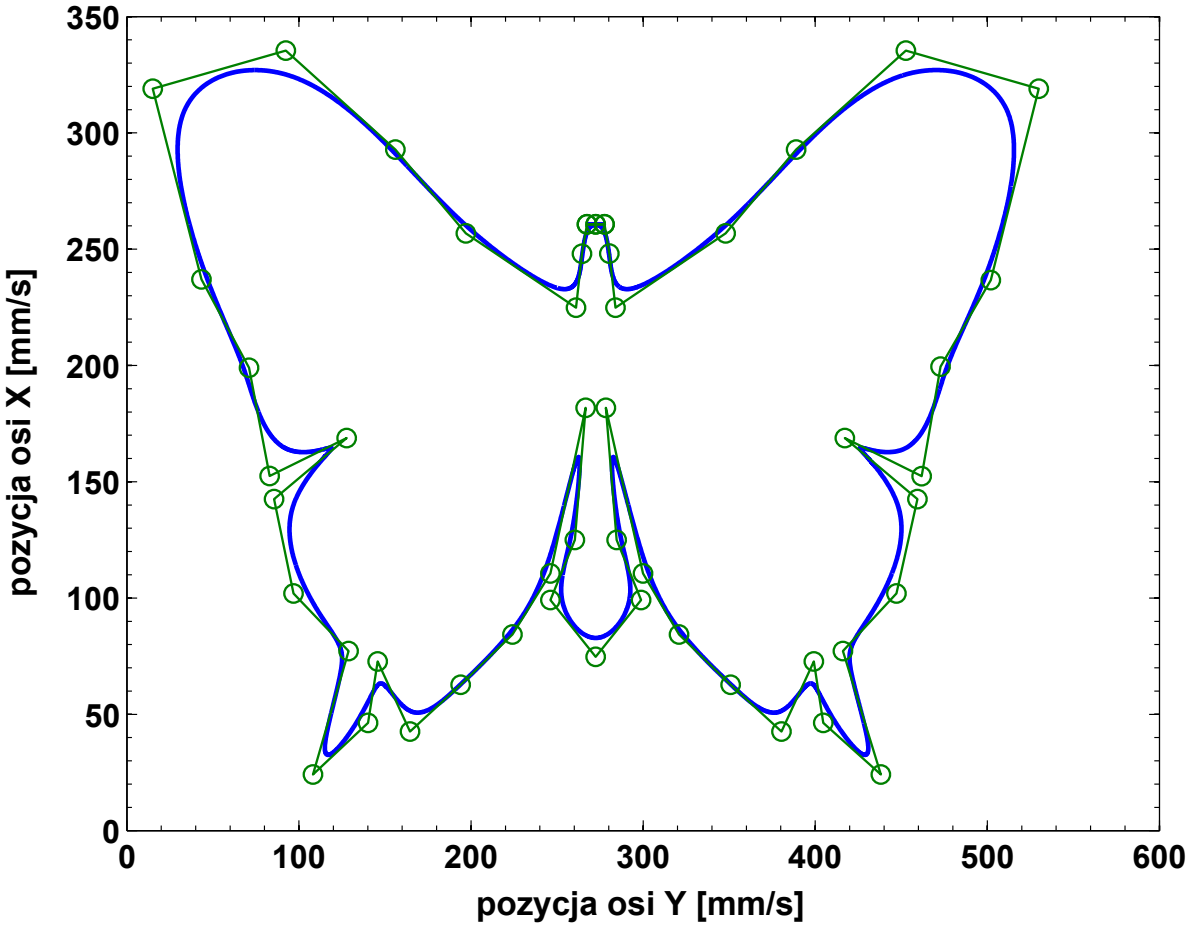
Do serwonapędów sterownik przesyła położenie zadane, prędkość zadana i przyspieszenie zadane. Prędkość i przyspieszenie, wyznacznone analitycznie z profilu prędkości, służą jako wartości zadane do sprzężeń w przód. Serwonapędy przesyłają do sterownika położenie rzeczywiste oraz błąd nadążania wyznaczone na podstawie pomiaru z przetwornika obrotowo-impulsowego. Błędy nadążąnia odczytane z obu serwonapędów służą do wyznaczenia błędu konturu. Pomiar błędu nadążania obarczony jest błędem ponieważ położenie wału silnika różni się od położenia suportu maszyny ze względu na luzy i sprężystości występujące w układzie mechanicznym. Aby dokonać pomiaru faktycznego przemieszczenia suportu konieczne byłoby zastosowanie liniałów pomiarowych. W opracowanym układzie sterowania liniały nie zostały zastosowane gdyż wpływ sprężystości i luzów na pomiar położenia można pominąć. Zastosowana konstrukcja maszyny zapewnia dużą sztywność i minimalne luzy. Z tego względu, w tej klasie maszyn, liniały stosuje się rzadko w praktyce.

Układ sterowania, omówiony w niniejszym rozdziale, został zbudowany przez autora w celu przeprowadzenia badań doświadczalnych opracowanego algorytmu optymalizacji prędkości posuwu. Układ sterowania składa się z maszyny wieloosiowej wyposażonej w

komercyjne serwonapędy oraz sterownika w formie komputera klasy PC. Komputer pracuje pod kontrolą systemu operacyjnego czasu rzeczywistego, w którym zaimplementowano interpolator czasowo-optymalnej trajektorii ruchu dla toru zdefiniowanego w formie krzywej NURBS oraz stos protokołu komunikacyjnego Ethernet Powerlink. Interpolator i stos zostały zaimplementowane w języku C jako moduły czasu rzeczywistego. Zastosowanie standardowego protokołu umożliwia sterowanie komercyjnymi maszynami. Wykorzystanie komputera PC jako platformy sprzętowej umożliwia implementację dowolnego oprogramowania sterującego. Dzięki temu opracowany układ sterowania ma duże znaczenie praktyczne.

6. Wyniki badań doświadczalnych algorytmu optymalizacji prędkości posuwu

Badania doświadczalne algorytmu optymalizacji prędkości przeprowadzone zostały z wykorzystaniem toru ruchu zadanego w formie krzywej NURBS przedstawionej na rysunku 6.1.

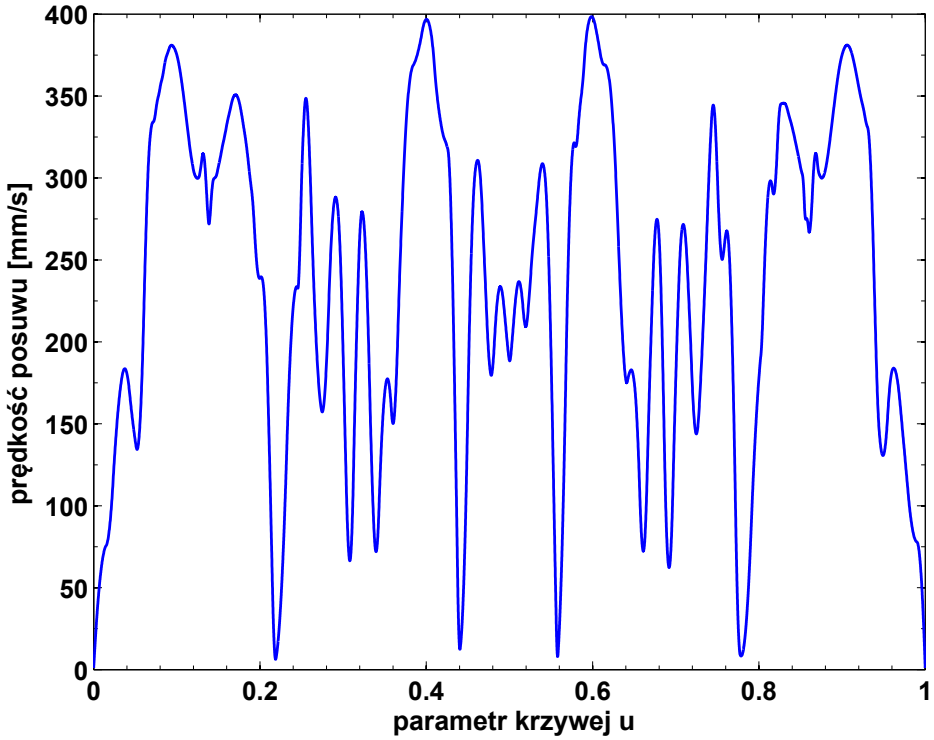


Rysunek 6.1. Zadany testowy tor ruchu w formie krzywej NURBS (niebieski) z zaznaczonymi punktami kontrolnymi (zielony)

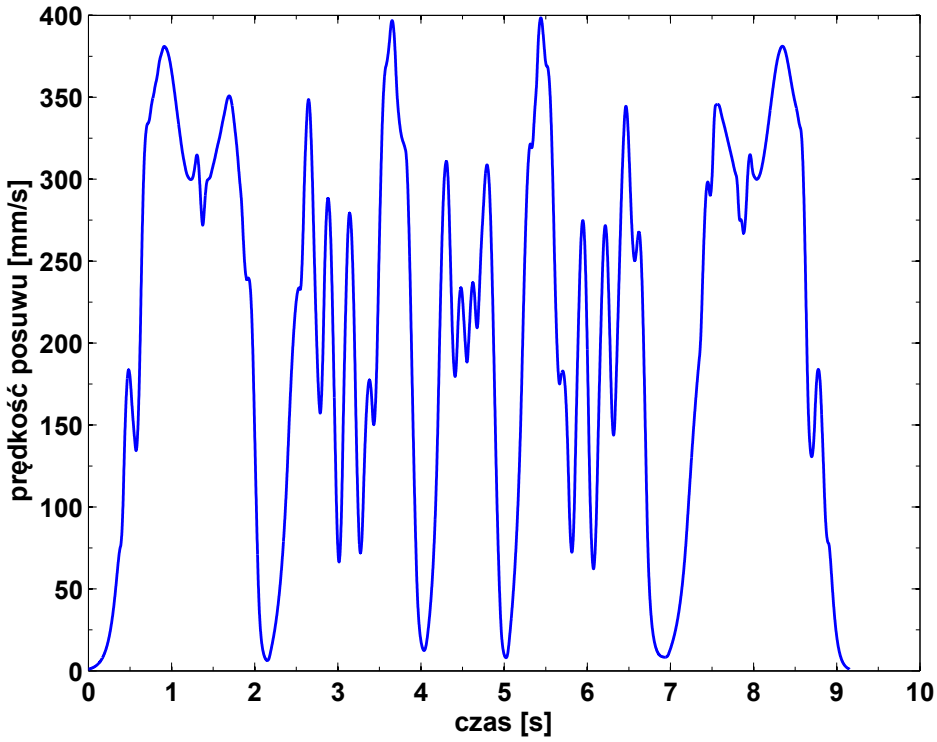
Dla powyższego toru ruchu przeprowadzono optymalizację prędkości posuwu przyjmując dla obydwu osi maksymalną wartość prędkości równą 300 mm/s, maksymalną wartość przyspieszenia równą 2500 mm/s^2 oraz maksymalną wartość zrywu równą 50000 mm/s^3. Przyjęte ograniczenia wynikały z maksymalnych możliwości zastosowanych serwonapędów i

silników oraz z konstrukcji maszyny. Czasowo-optymalne profile prędkości wygenerowano przyjmując maksymalną wartość błędu konturu wynoszącą 0,05 mm, 0,03 mm i 0,02 mm. Zadany tor ruchu w formie krzywej NURBS ma długość 1897,4 mm. Oznacza to, że dla przyjętej odległości pomiędzy punktami kontrolnymi równej około 7,5mm, profil prędkości został zdefiniowany przez 253 punkty kontrolne. Długość horyzontu predykcji została przyjęta na 40 punktów kontrolnych profilu prędkości przy czym 10 pierwszych punktów pochodziło z poprzedniego horyzontu. Całkowita liczba optymalizowanych przedziałów wynosiła 9.

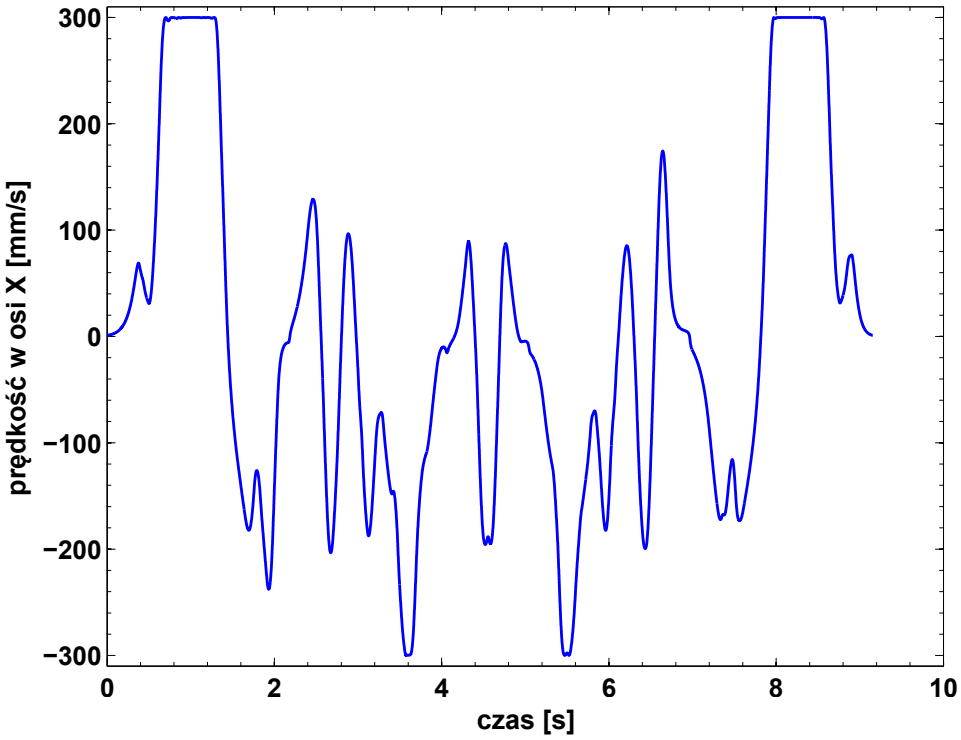
Ze względu na rozrzut rzeczywistej wartości błędów nadążania wynikający z ograniczonej powtarzalności maszyny oraz błędy predykcji wynikające z niedokładności modelu zadane wartości progowe zostały pomniejszone o 0.005 mm. Zadany tor ruchu został zrealizowany dla każdego z wygenerowanych profili 10 razy a otrzymany błąd nadążania został uśredniony. Na podstawie uśrednionych błędów nadążania wyznaczono błąd konturu. Na wykresach przedstawiono profil prękości w funkcji parametru krzywej i czasu oraz prędkość, przyspieszenie i zryw w obydwu osiach w funkcji czasu. Przedstawiono również wykres rzeczywistego błędu konturu w funkcji czasu wyznaczony na podstawie pomiaru błędu nadążania oraz wykres błędu konturu wyznaczony przez predyktor neuronowy. W trakcie optymalizacji błąd konturu wyznaczano z wykorzystaniem metody drugiego rzędu przedstawionej w rozdziale 2 (wzory 2.26 i 2.27). Wartości błędu konturu przedstawione na wykresach wyznaczano przy pomocy metody iteracyjnej (wzór 2.28), która zapewnia najlepsze przybliżenie rzeczywistego błędu konturu.



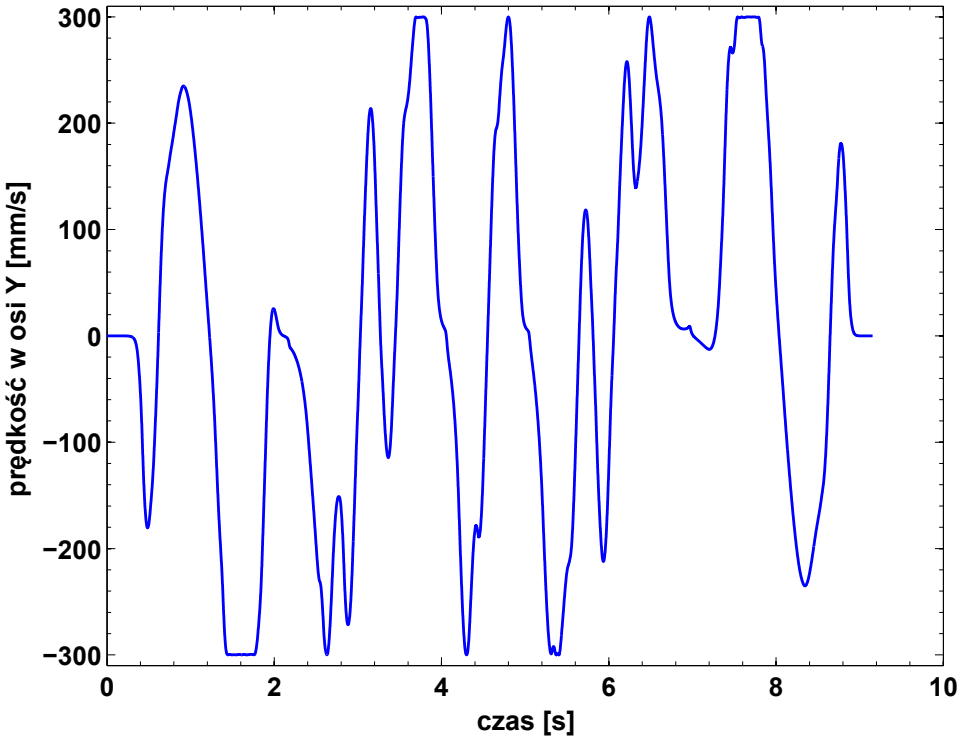
Rysunek 6.2. Profil prędkości posuwu w funkcji parametru krzywej dla tolerancji błędu konturu 0.045 mm



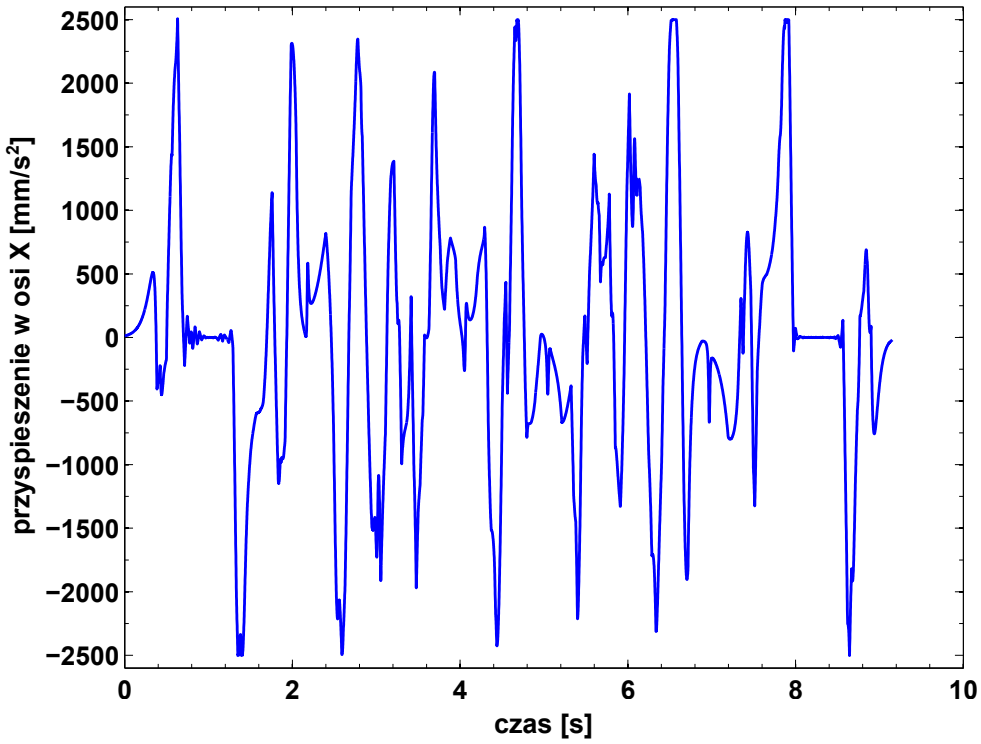
Rysunek 6.3. Profil prędkości posuwu w funkcji czasu dla tolerancji błędu konturu 0.045 mm



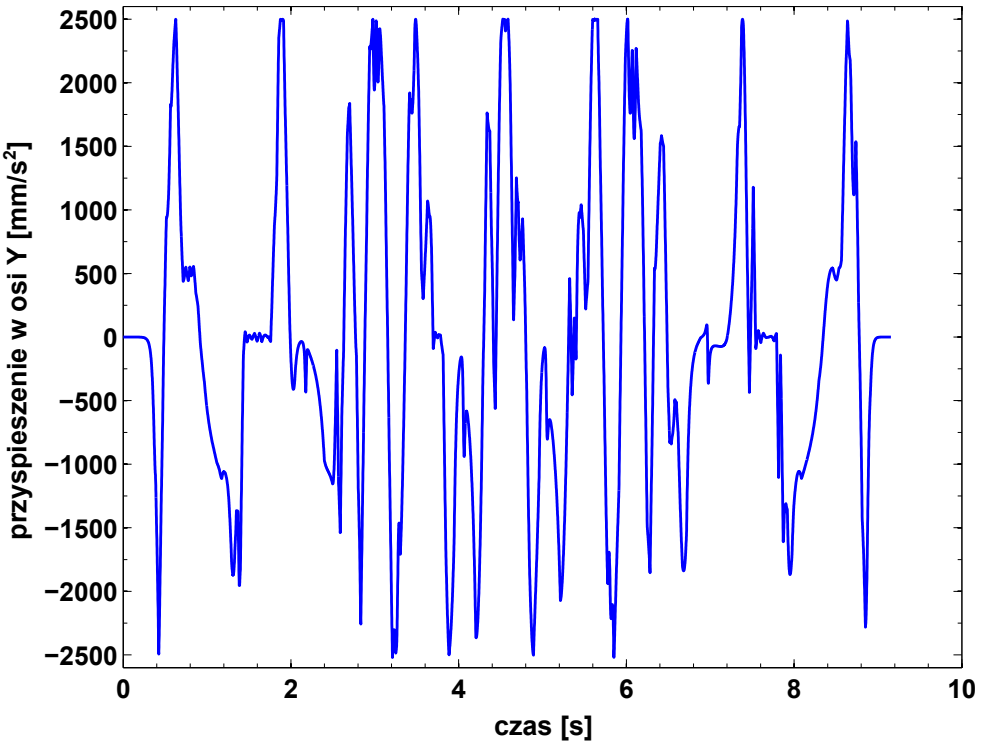
Rysunek 6.4. Prędkość zadana w osi X dla tolerancji błędu konturu 0.045 mm



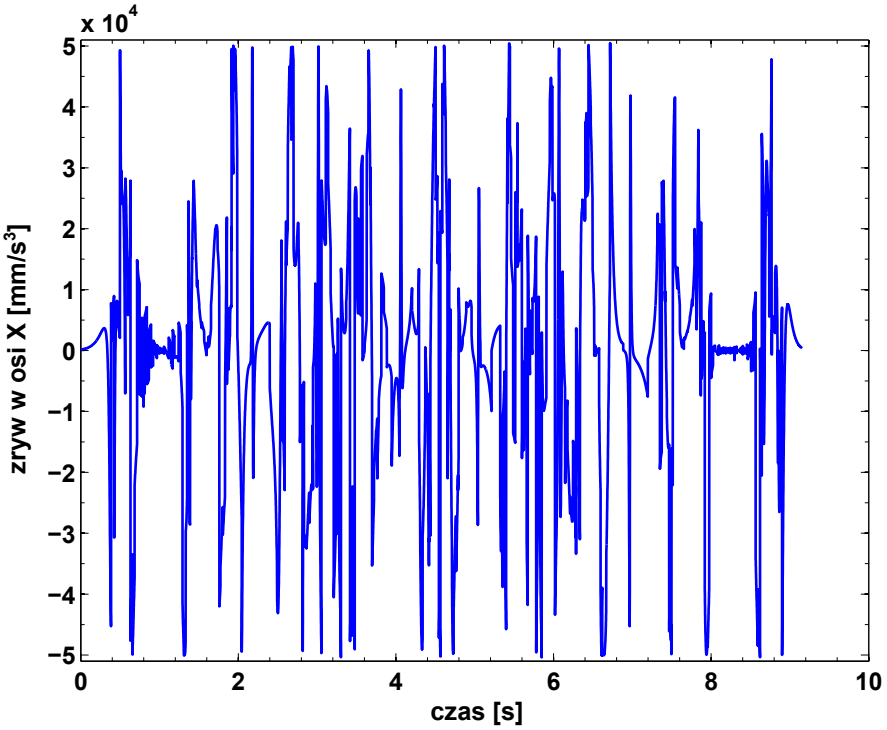
Rysunek 6.5. Prędkość zadana w osi Y dla tolerancji błędu konturu 0.045 mm



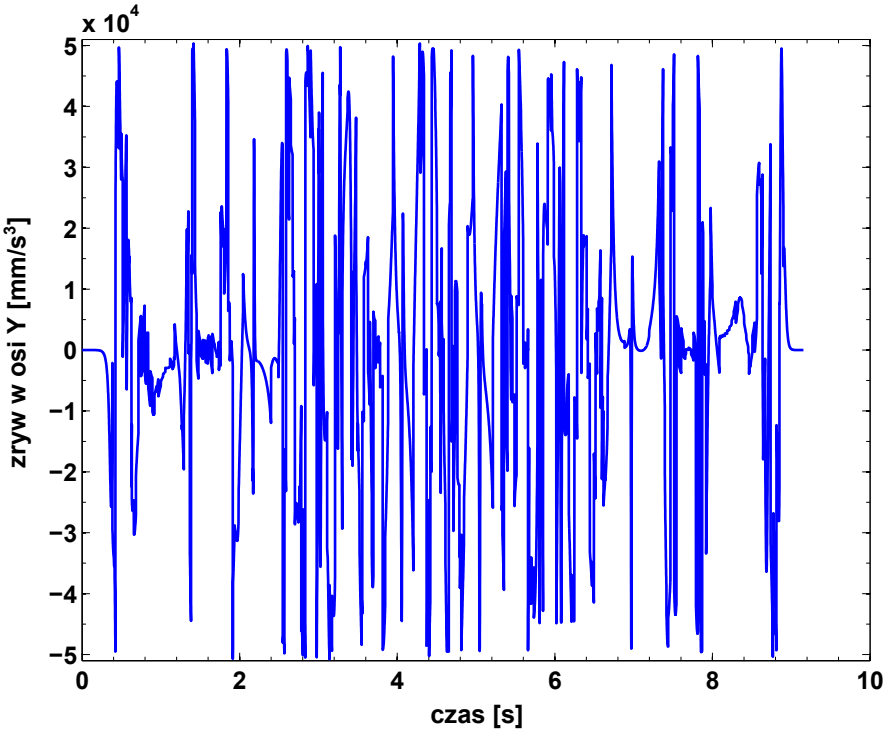
Rysunek 6.6. Przyspieszenie zadane w osi X dla tolerancji błędu konturu 0.045 mm



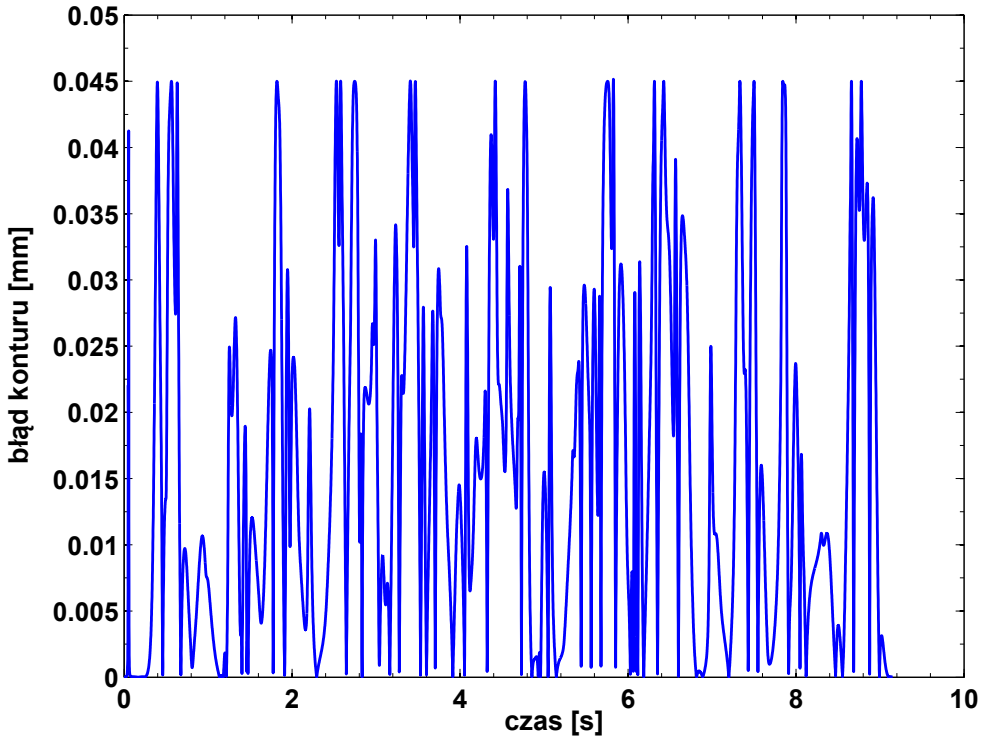
Rysunek 6.7. Przyspieszenie zadane w osi Y dla tolerancji błędu konturu 0.045 mm



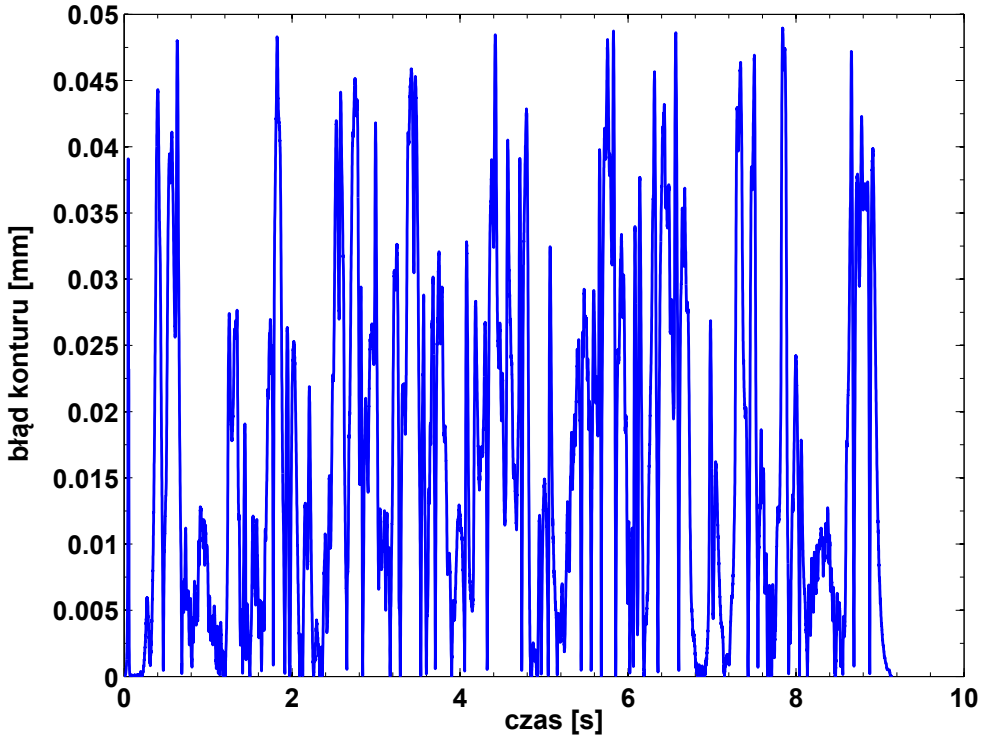
Rysunek 6.8. Zryw zadany w osi X dla tolerancji błędu konturu 0.045 mm



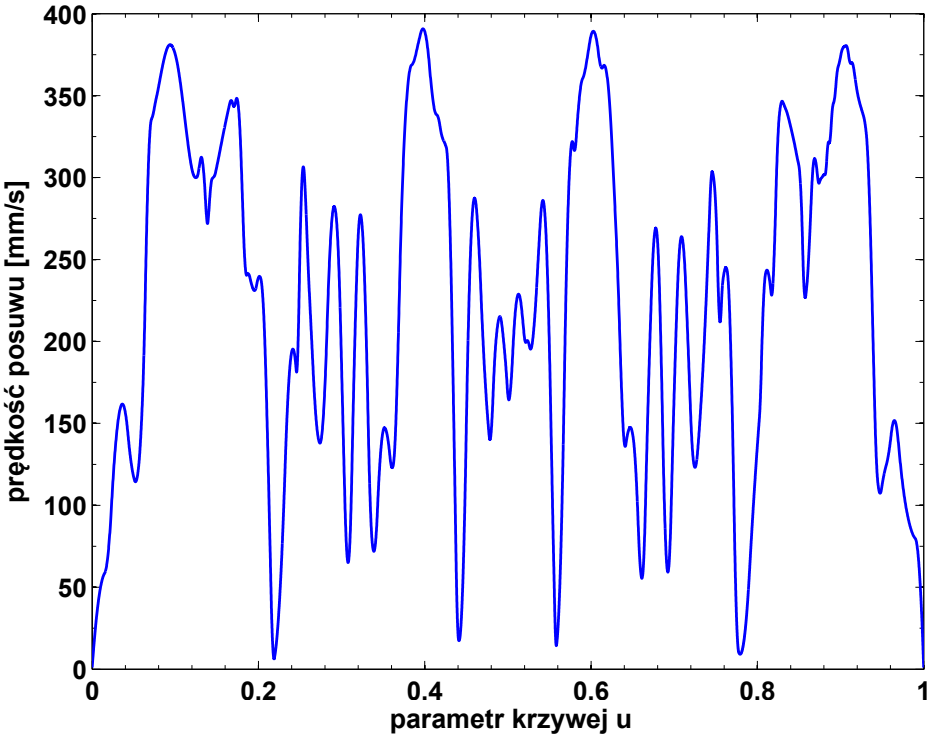
Rysunek 6.9. Zryw zadany w osi Y dla tolerancji błędu konturu 0.045 mm



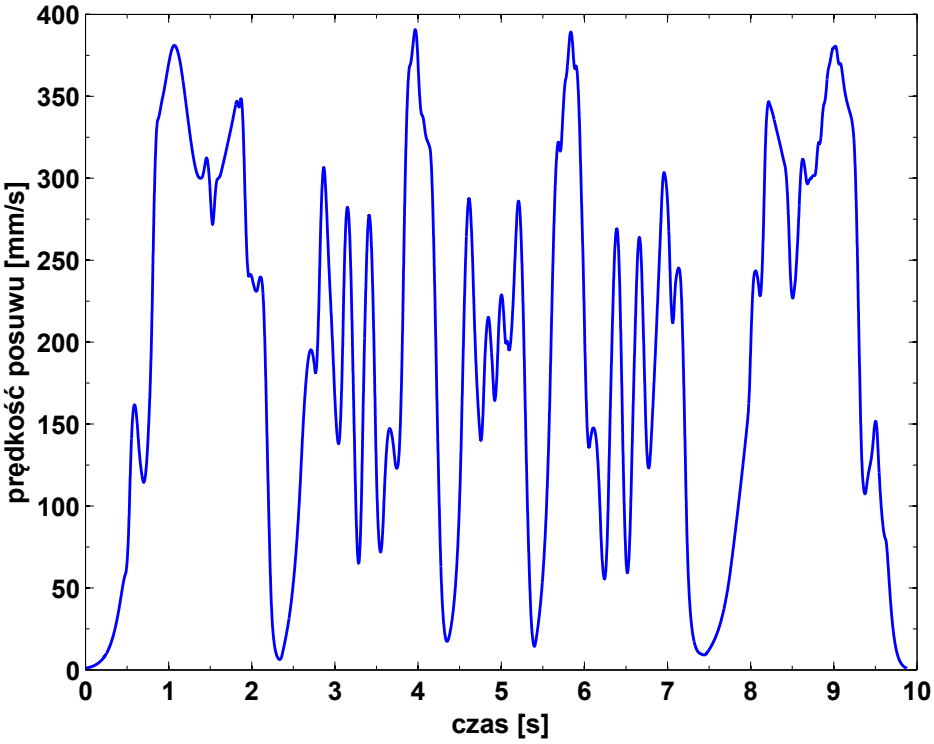
Rysunek 6.10. Przewidywany błąd konturu dla tolerancji błędu konturu 0.045 mm



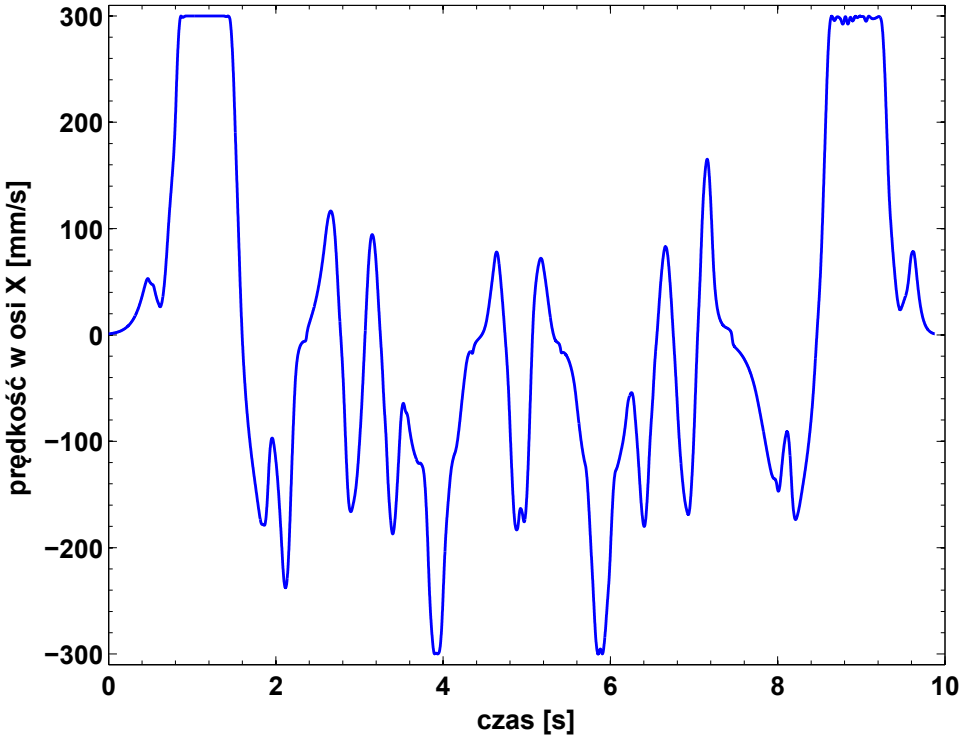
Rysunek 6.11. Rzeczywisty błąd konturu dla tolerancji błędu konturu 0.045 mm



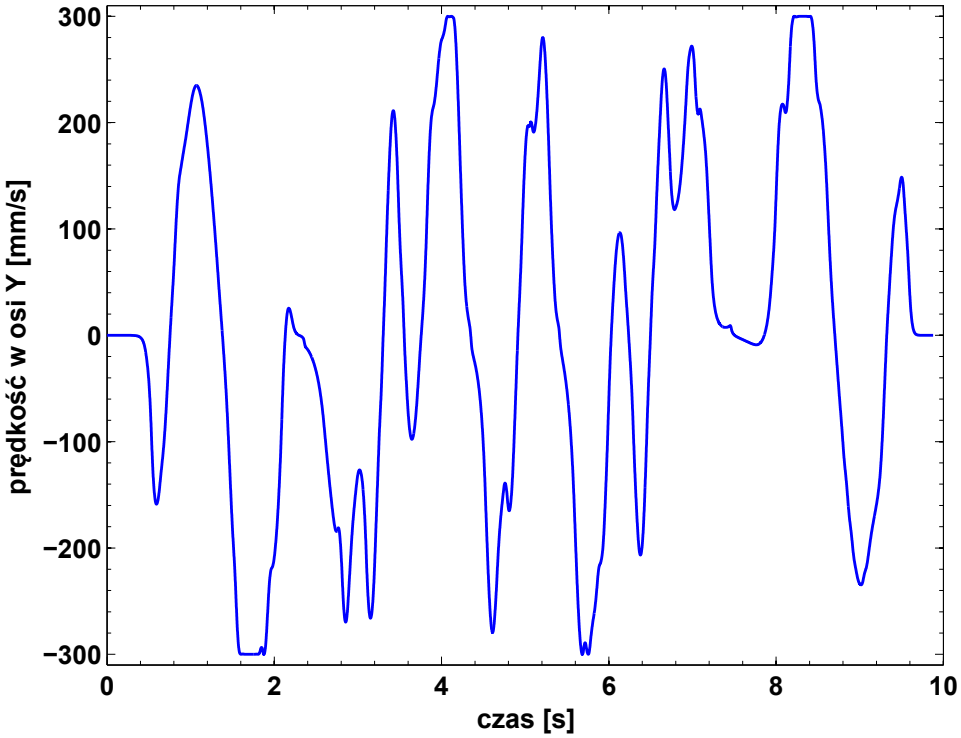
Rysunek 6.12. Profil prędkości posuwu w funkcji parametru krzywej dla tolerancji błędu konturu 0.025 mm



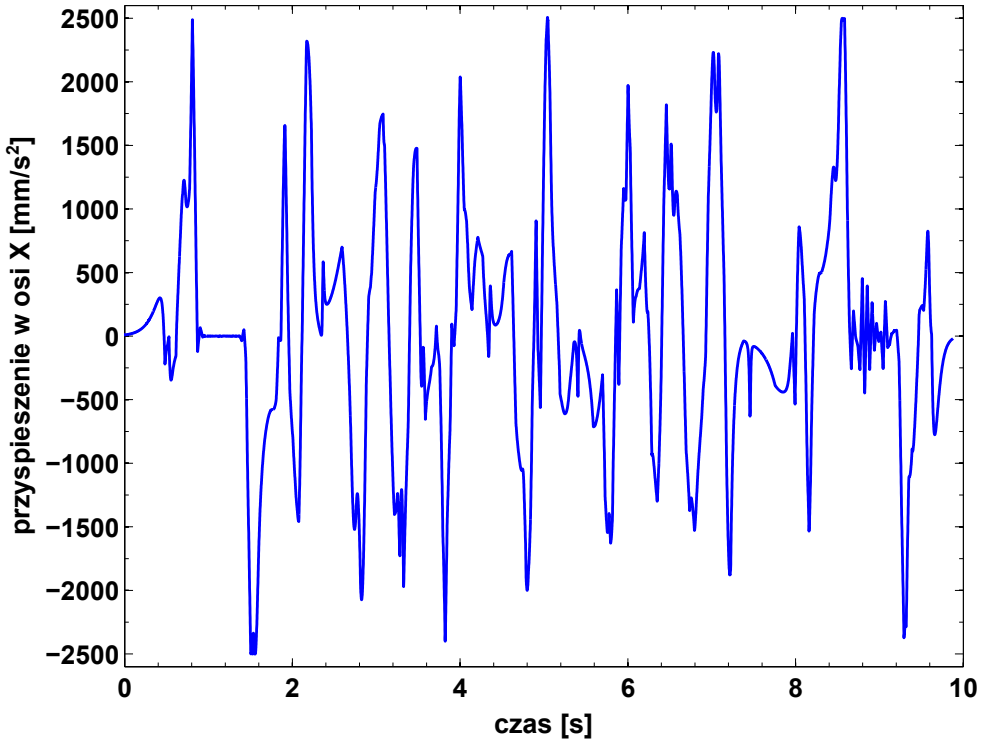
Rysunek 6.13. Profil prędkości posuwu w funkcji czasu dla tolerancji błędu konturu 0.025 mm



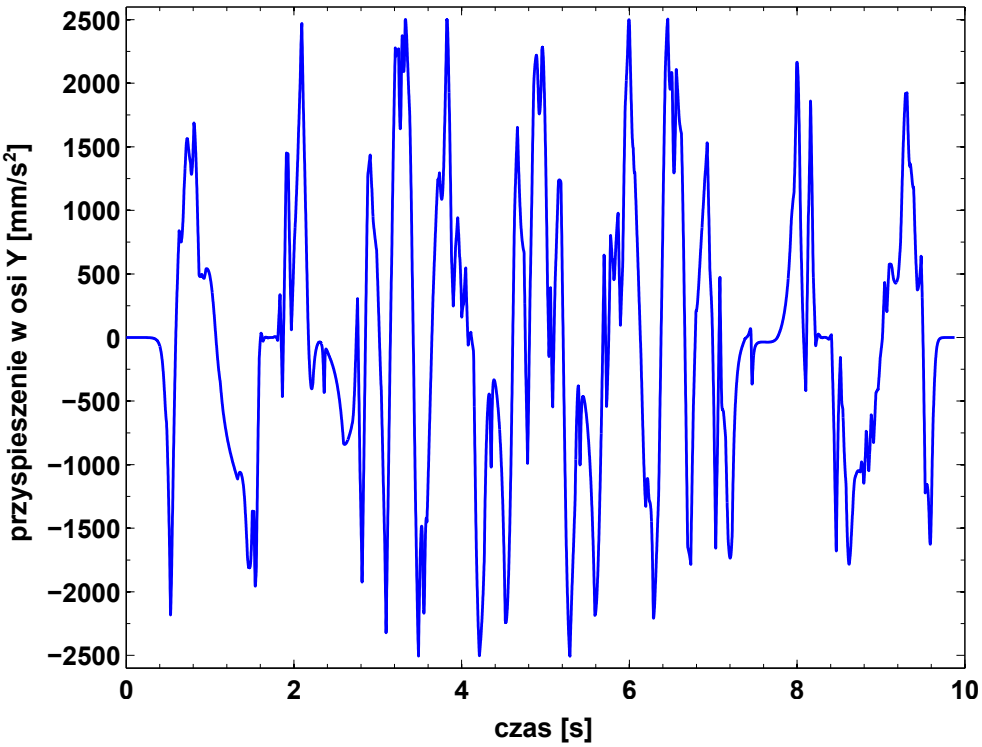
Rysunek 6.14. Prędkość zadana w osi X dla tolerancji błędu konturu 0.025 mm



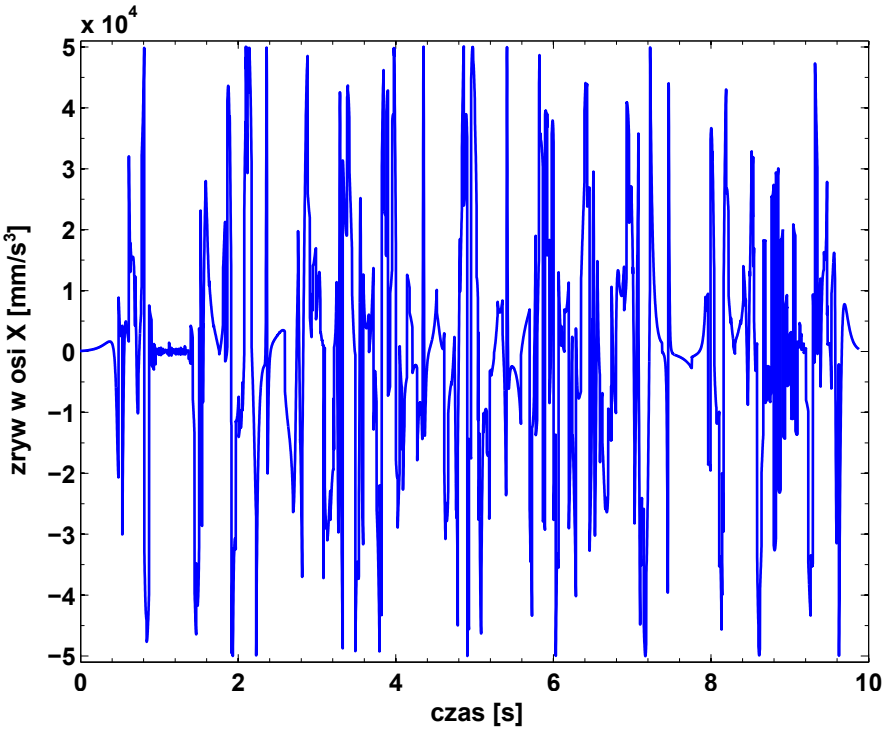
Rysunek 6.15. Prędkość zadana w osi Y dla tolerancji błędu konturu 0.025 mm



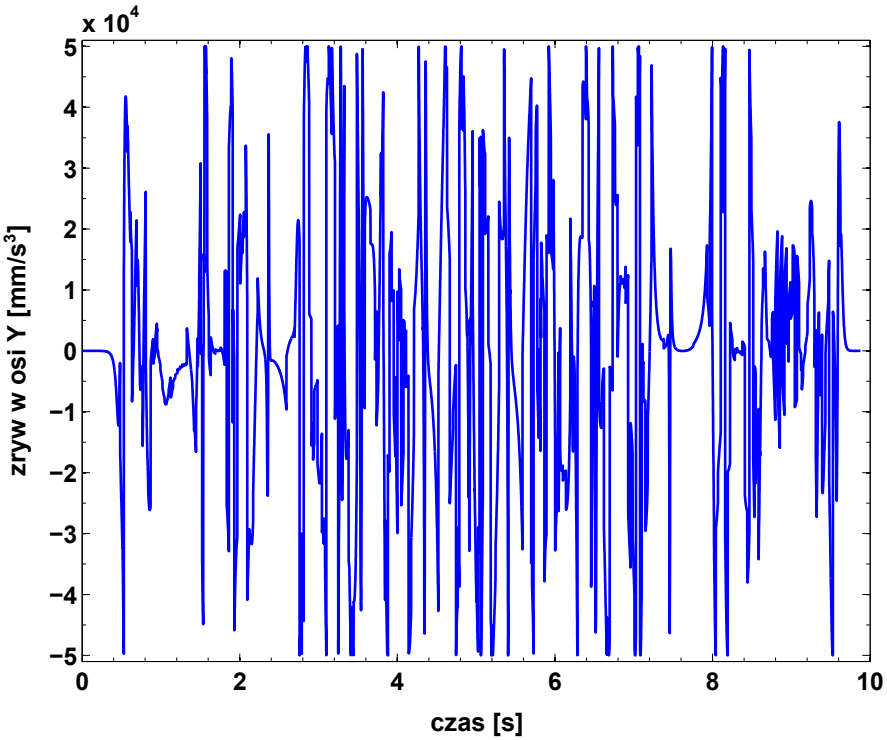
Rysunek 6.16. Przyspieszenie zadane w osi X dla tolerancji błędu konturu 0.025 mm



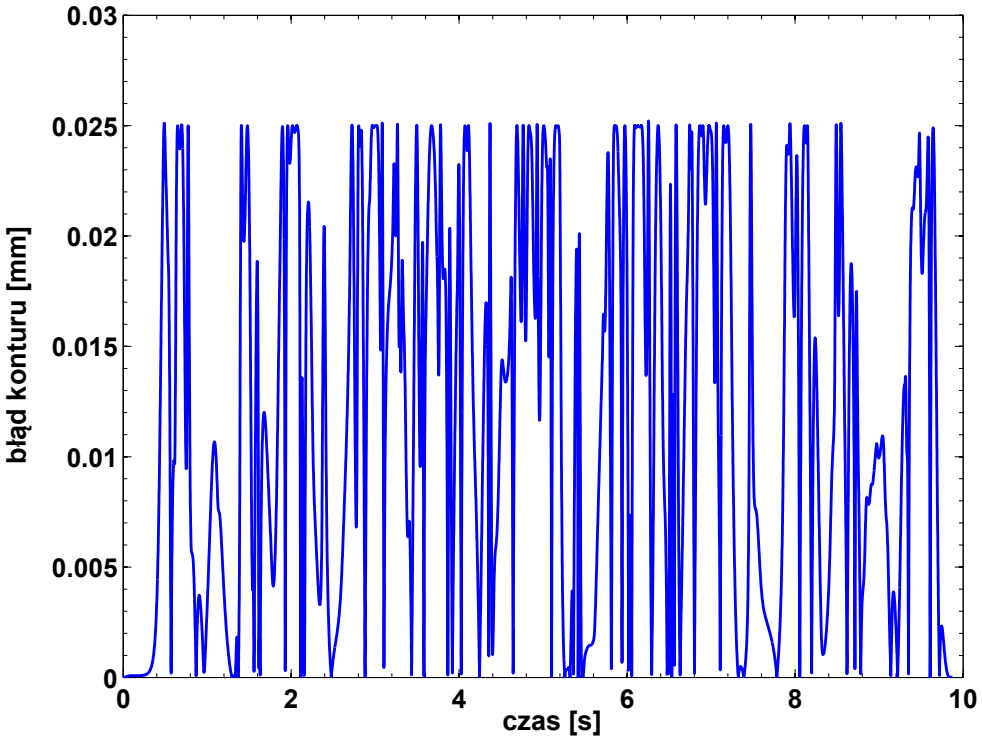
Rysunek 6.17. Przyspieszenie zadane w osi Y dla tolerancji błędu konturu 0.025 mm



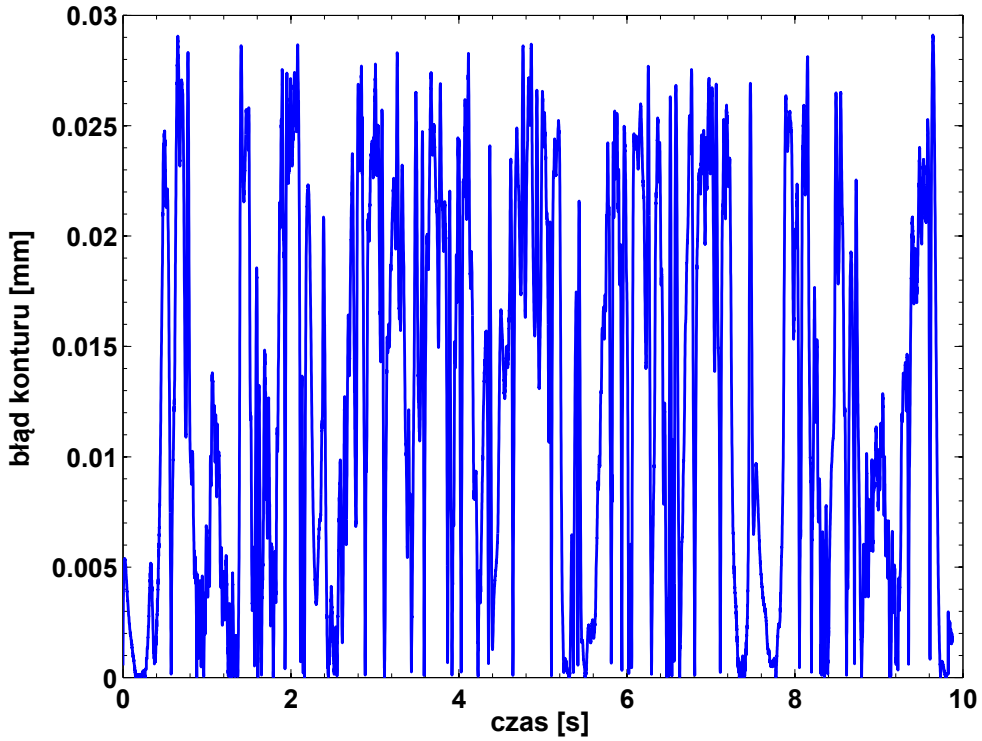
Rysunek 6.18. Zryw zadany w osi X dla tolerancji błędu konturu 0.025 mm



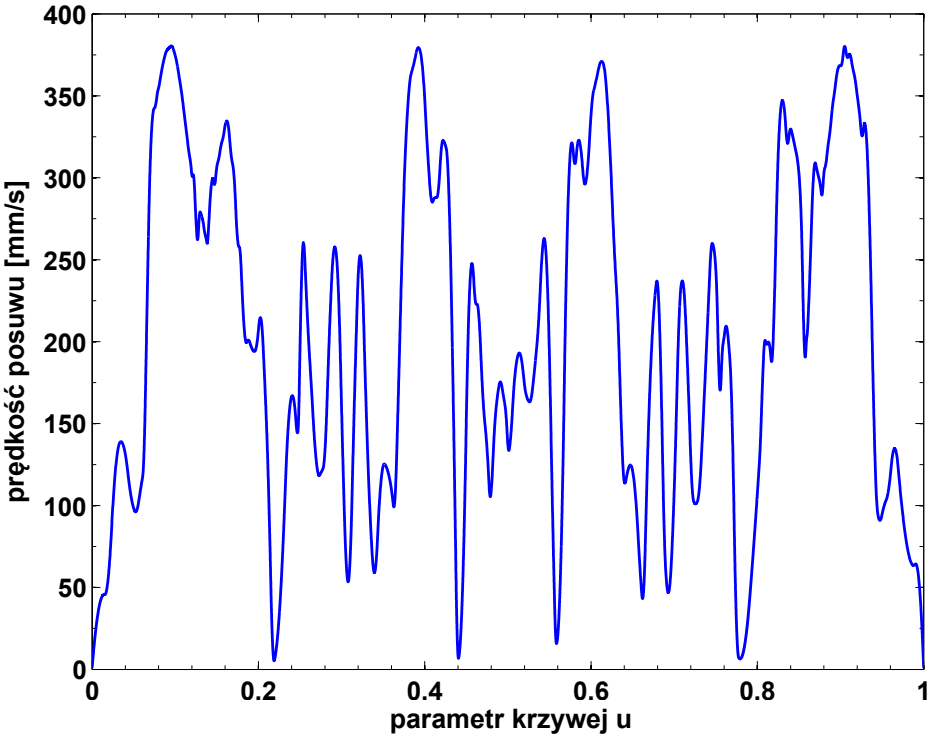
Rysunek 6.19. Zryw zadany w osi Y dla tolerancji błędu konturu 0.025 mm



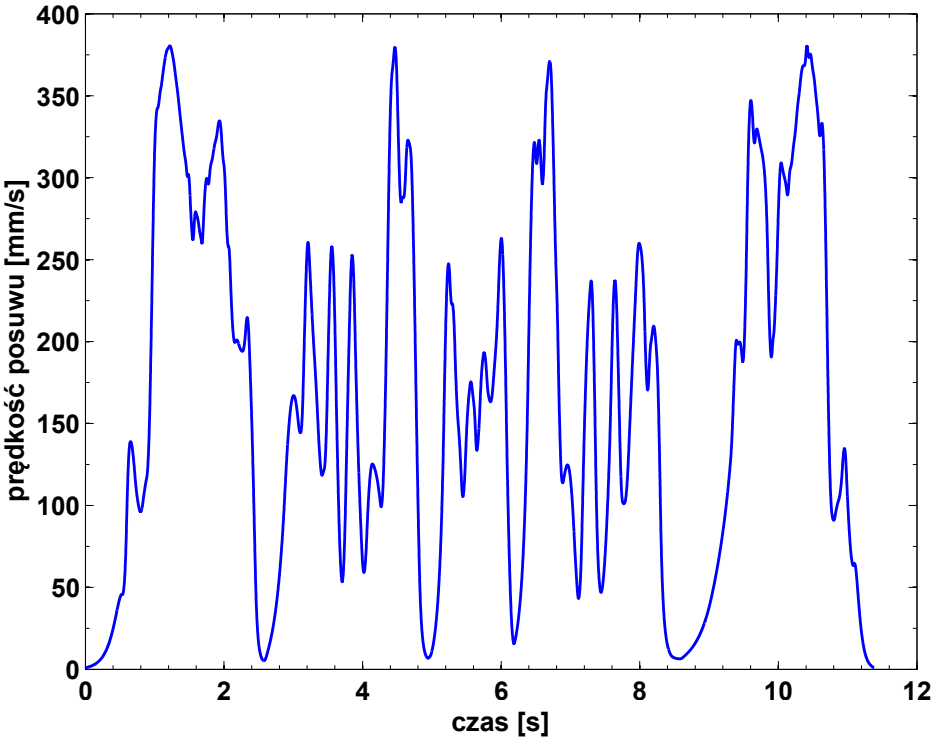
Rysunek 6.20. Przewidywany błąd konturu dla tolerancji błędu konturu 0.025 mm



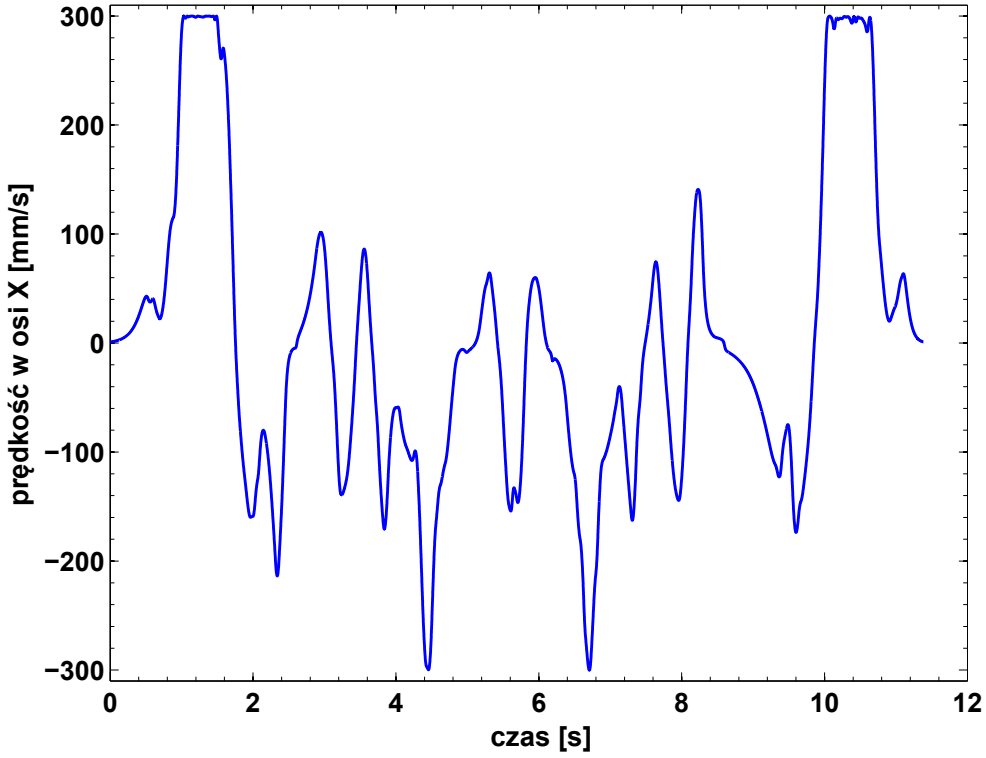
Rysunek 6.21. Rzeczywisty błąd konturu dla tolerancji błędu konturu 0.025 mm



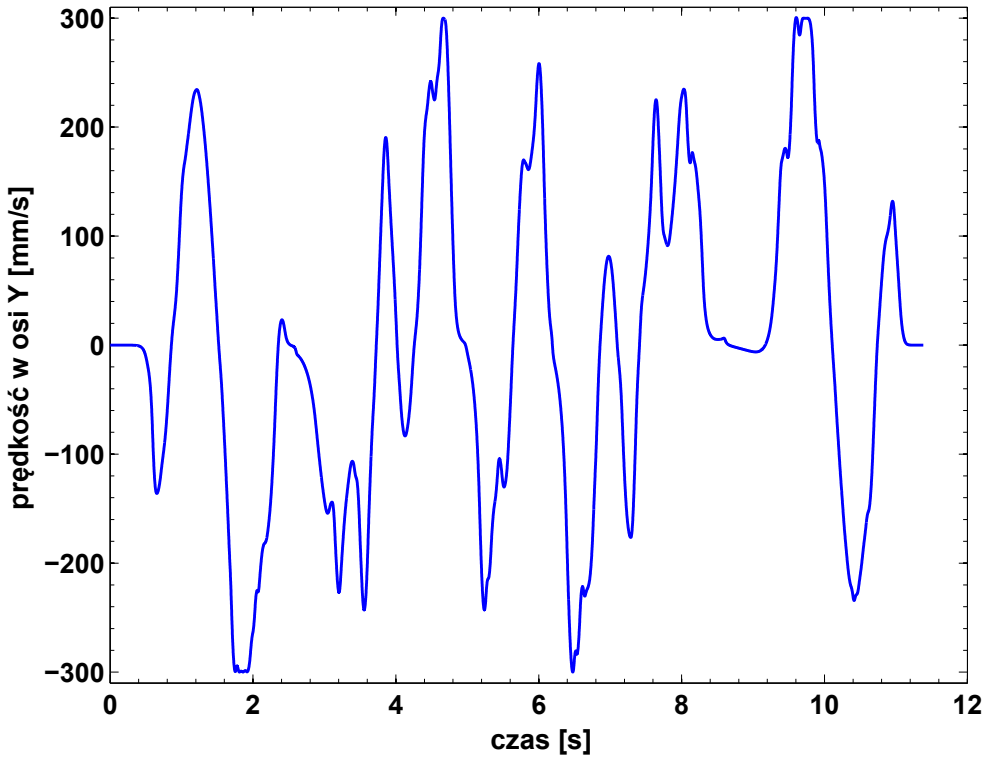
Rysunek 6.22. Profil prędkości posuwu w funkcji parametru krzywej dla tolerancji błędu konturu 0.015 mm



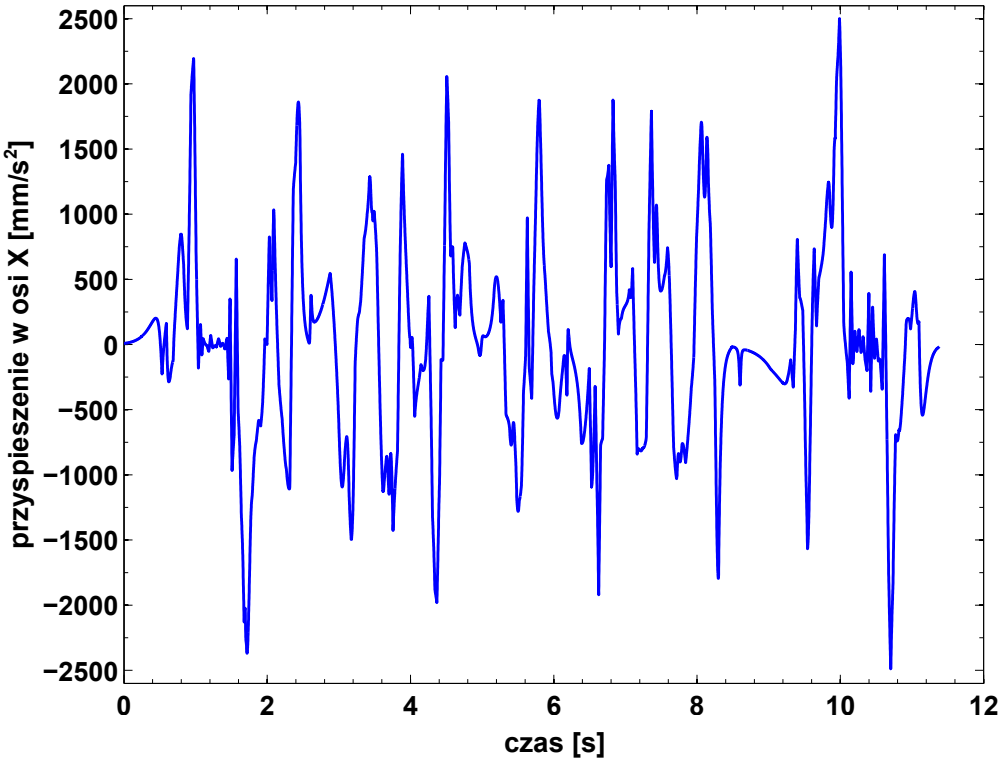
Rysunek 6.23. Profil prędkości posuwu w funkcji czasu dla tolerancji błędu konturu 0.015 mm



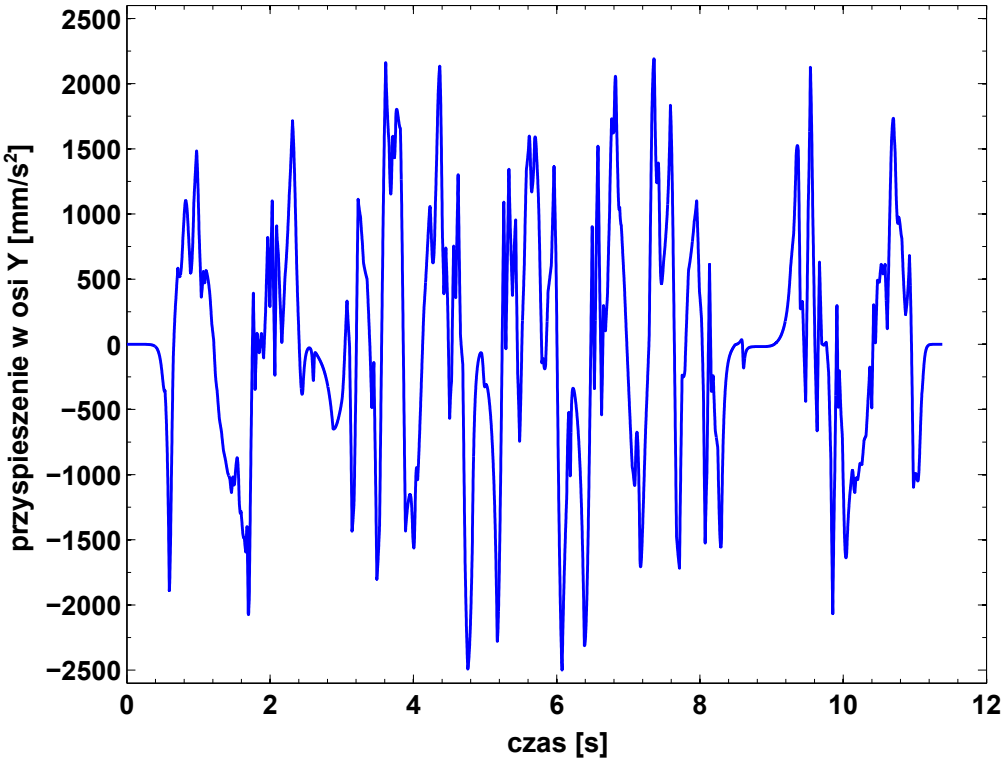
Rysunek 6.24. Prędkość zadana w osi X dla tolerancji błędu konturu 0.015 mm



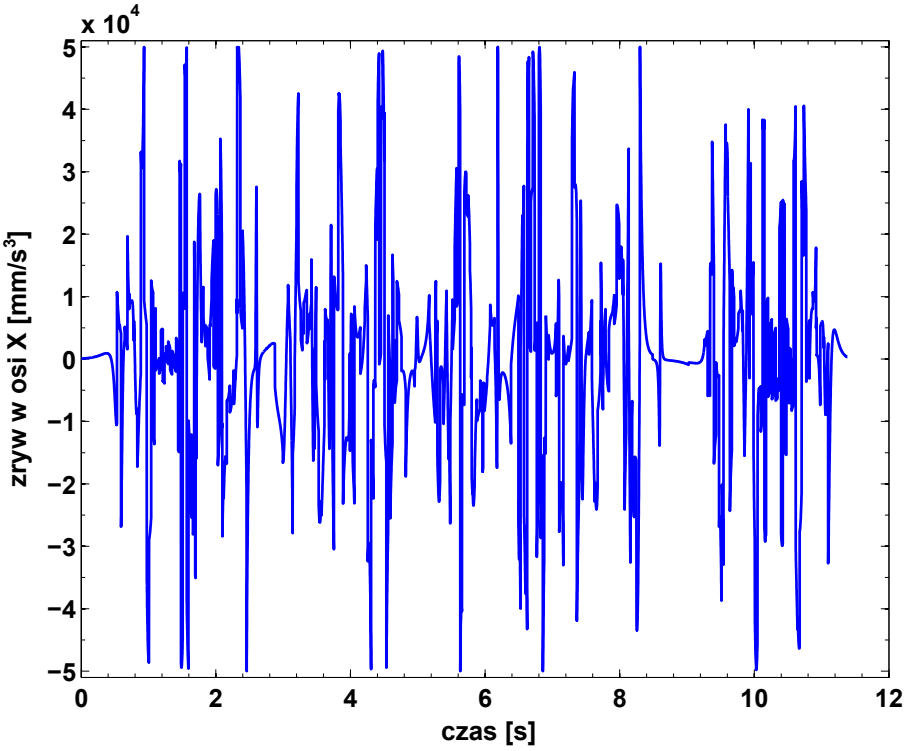
Rysunek 6.25. Prędkość zadana w osi Y dla tolerancji błędu konturu 0.015 mm



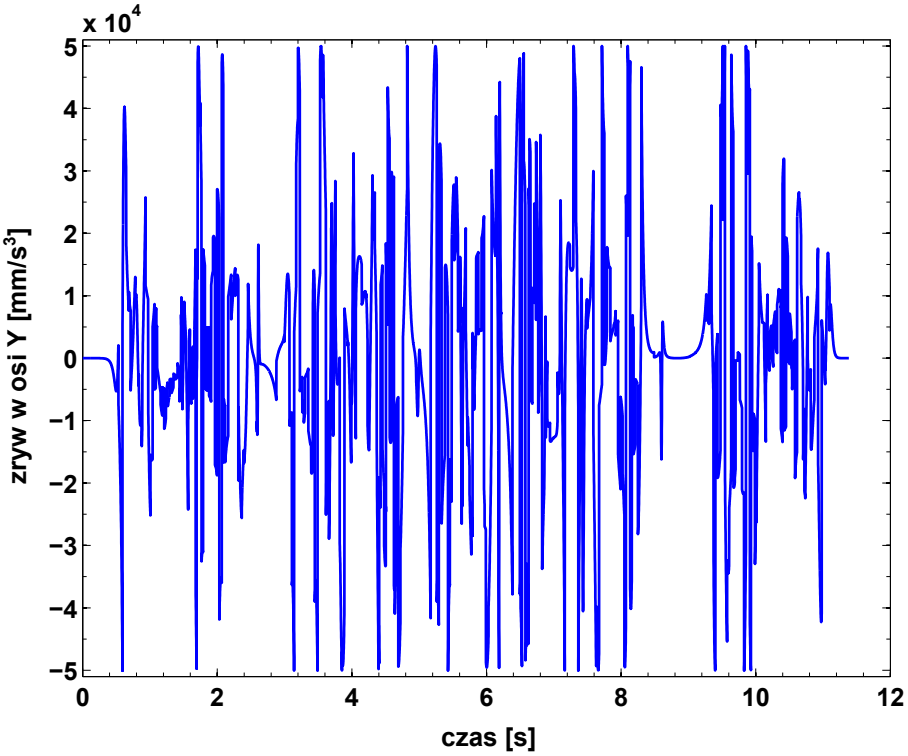
Rysunek 6.26. Przyspieszenie zadane w osi X dla tolerancji błędu konturu 0.015 mm



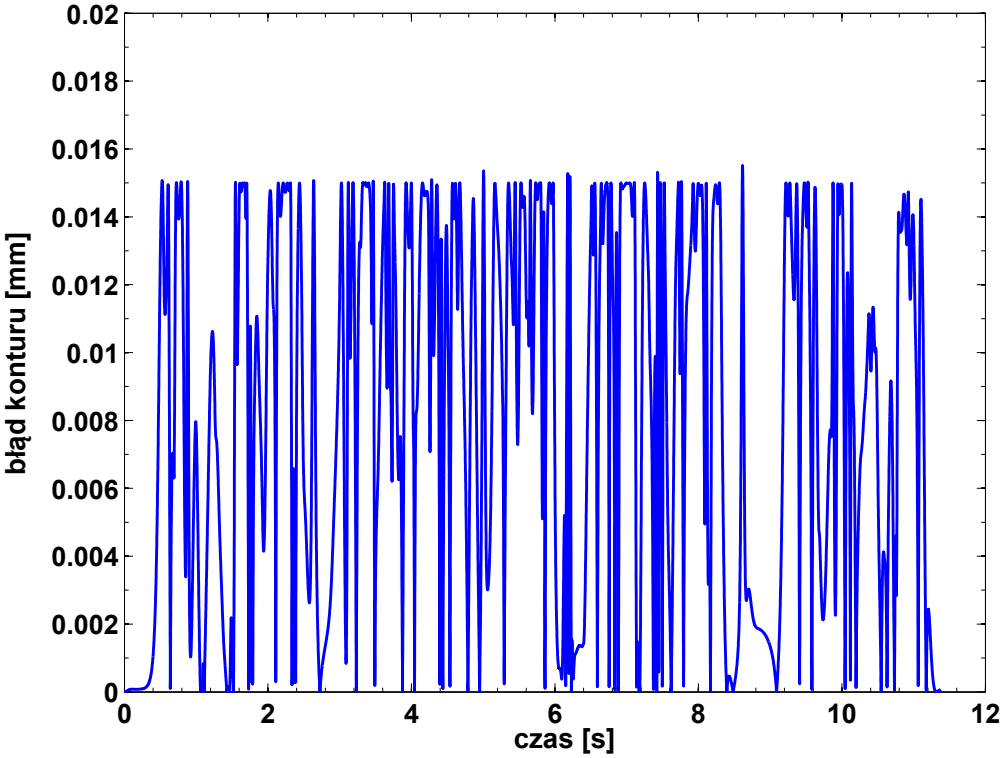
Rysunek 6.27. Przyspieszenie zadane w osi Y dla tolerancji błędu konturu 0.015 mm



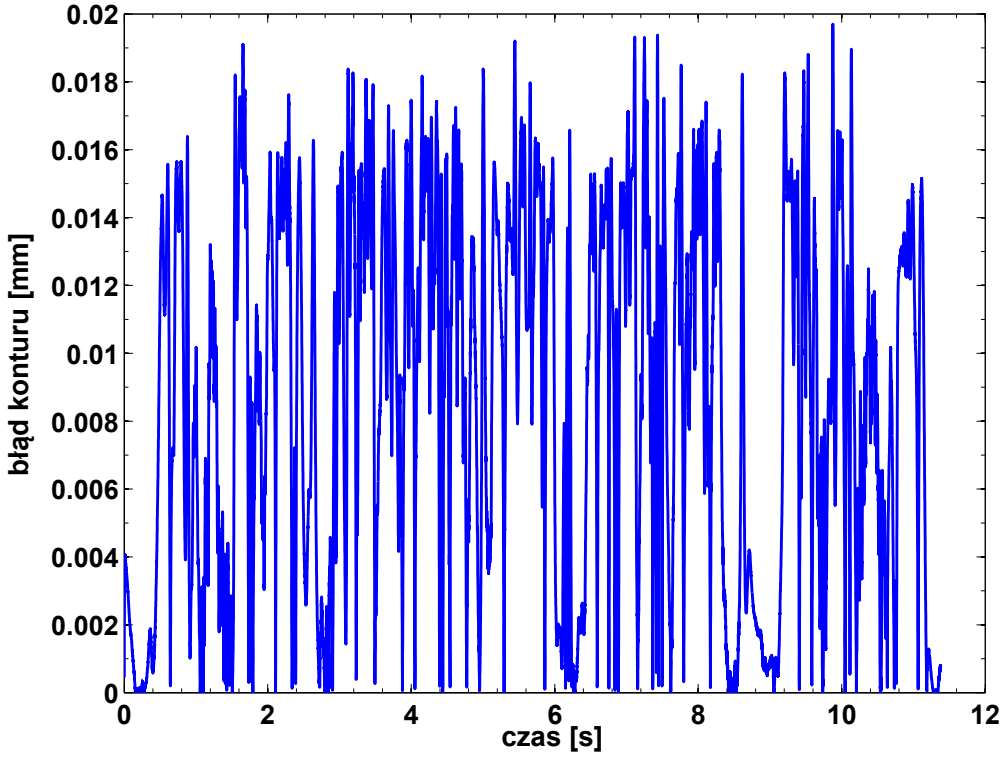
Rysunek 6.28. Zryw zadany w osi X dla tolerancji błędu konturu 0.015 mm



Rysunek 6.29. Zryw zadany w osi Y dla tolerancji błędu konturu 0.015 mm

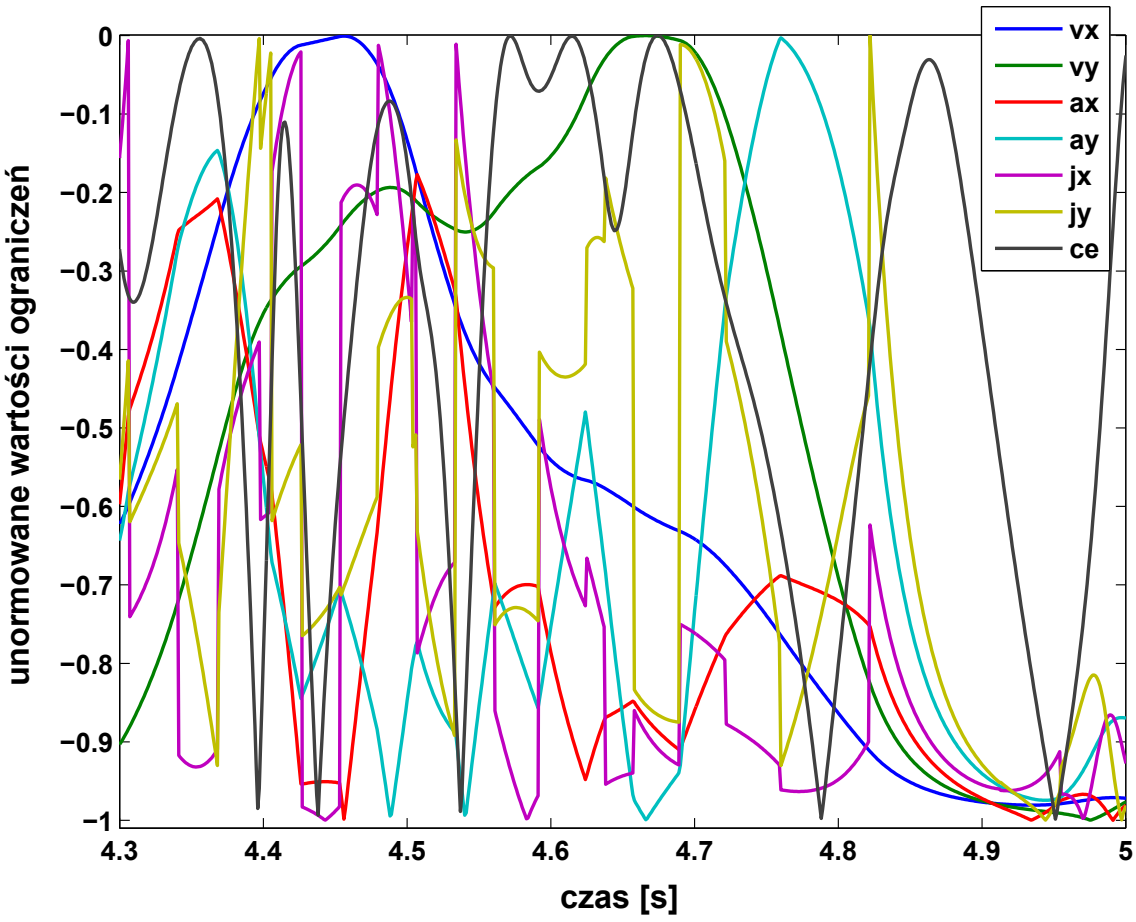


Rysunek 6.30. Przewidywany błąd konturu dla tolerancji błędu konturu 0.015 mm



Rysunek 6.31. Rzeczywisty błąd konturu dla tolerancji błędu konturu 0.015 mm

Na podstawie przedstawionych wyników można stwierdzić, że opracowana metoda optymalizacji prędkości posuwu w układach sterowania maszyn CNC pozwala na wygenerowanie czasowo-optymalnego profilu prędkości uwzględniając ograniczenia maksymalnej prędkości, przyspieszenia i zrywu w osiach. Ponadto uwzględnione jest ograniczenie maksymalnej wartości błędu konturu, który jest miarą dokładności odtwarzania zadanej trajektorii ruchu przez maszynę CNC. Rzeczywisty błąd konturu przekracza nieznacznie wartości maksymalne pomimo, że błąd konturu wyznaczony przez predyktor neuronowy mieści się w zadanej tolerancji. Wynika to z niedokładności modelu oraz ograniczonej powtarzalności działania maszyny CNC. Nie jest możliwe opracowanie idealnego modelu pozwalającego na bezbłędną predykcję błędu konturu w złożonym układzie mechanicznym jakim jest maszyna CNC. Nie jest również możliwe zbudowanie maszyny, która za każdym razem odtworzy zadaną trajektorię ruchu dokładnie tak samo. Z tego względu w opracowanym algorytmie zadana maksymalna wartość błędu konturu jest obniżona o 0.005 mm co w większości przypadków zagwarantuje zachowanie zadanej tolerancji.



Rysunek 6.32. Fragment przebiegów prędkości, przyspieszenia, zrywu oraz błędu konturu przeskalowanych względem wartości maksymalnych

Na rysunku 6.32 przedstawiono fragment przebiegów prędkości, przyspieszenia, zrywu oraz błędu konturu przeskalowane zgodnie ze wzorem $C_N = \frac{|C|}{C_{MAX}} - 1$, gdzie C jest jedną z ograniczanych wielkości a C_{MAX} jej wartością maksymalną. Skalowanie ma na celu umożliwienie porównania wszystkich ograniczanych wielkości, których zakres różni się o rzędy wielkości. Przedstawione przebiegi zostały wygenerowane na podstawie trzeciego, z przedstawionych, powyżej czasowo-optymalnych profili prędkości posuwu. Na przedstawionym wykresie można zaobserwować, że różne z ograniczanych wielkości są w pewnych fragmentach przebiegów bliskie wartości maksymalnej. Oznacza to, że wielkość, której wartość jest blisko wartości maksymalnej, stanowi w danej chwili główne ograniczenie wpływające na prędkość posuwu. Ponadto wszystkie z ograniczeń mają wpływ na ostateczny kształt profilu prędkości w różnych jego fragmentach. Wygenerowanie takiego profilu prędkości bez zastosowania algorytmu optymalizacyjnego nie jest możliwe ze względu na dużą ilość ograniczeń, które należy uwzględnić.

Uzyskany w wyniku optymalizacji profil prędkości cechuje się dużymi wahaniami prędkości posuwu. Prędkość posuwu jest zmniejszana tak aby spełnić narzucone ograniczenia błędu konturu, prędkości, przyspieszenia i zrywu. Minima prędkości posuwu odpowiadają minimom krzywizny zadanego toru ruchu. W tych punktach występują znaczne zmiany prędkości, przyspieszenia i zrywu co prowadzi do większych błędów nadążania a w konsekwencji większych błędów konturu. Bez zastosowania algorytmu optymalizacyjnego uzyskanie profilu prędkości spełniającego narzucone warunki było by trudne i wymagało by znacznego ograniczenia maksymalnej prędkości posuwu. Dobór odpowiednich wartości przyspieszeń i zrywów bez zastosowania algorytmu optymalizacyjnego jest zadaniem trudnym, wymagającym dużego doświadczenia i kilku prób. Nawet wtedy uzyskane wartości będą dobrane zachowawczo. Dzięki zastosowaniu proponowanego algorytmu możliwa jest realizacja zadanej trajektorii ruchu z maksymalnymi dopuszczalnymi prędkościami, przyspieszeniami i zrywami co prowadzi do minimalizacji czasu jej realizacji a tym samym do zwiększenia wydajności procesu produkcji.

7. Wnioski

Przedstawioną rozprawę doktorską poświęcono zagadnieniu generacji czasowo-optymalnego profilu prędkości posuwu maszyny CNC z uwzględnieniem ograniczeń prędkości, przyspieszenia, zrywu oraz błędu konturu. Omówiono strukturę układu sterowania CNC wraz z najważniejszymi zagadnieniami dotyczącymi problemu interpolacji toru ruchu zadanego w formie krzywej wielomianowej NURBS. Opracowano neuronowy predyktor błędu konturu pozwalający na przewidywanie błędu konturu w procesie optymalizacji prędkości posuwu. Opracowano algorytm optymalizacji prędkości posuwu wykorzystujący algorytm optymalizacji rojem cząstek (PSO) uwzględniający ograniczenia maksymalnych wartości prędkości, przyspieszenia i zrywu w osiach maszyny oraz błędu konturu. 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:

— Opracowany neuronowy predyktor błędu konturu skutecznie modeluje dynamikę układu posuwu maszyny CNC pozwalając na przewidzenie wartości błędu konturu.
— Algorytm optymalizacyjny umożliwia wygenerowanie czasowo-optymalnego profilu prędkości posuwu uwzględniającego maksymalne wartości prędkości, przyspieszenia, zrywu oraz błędu konturu co pozwala na skrócenie czasu realizacji zadanej trajektorii ruchu.
— Uzyskany błąd konturu różni się od przewidywanego błędu konturu ze względu na nieuniknione błędy modelowania oraz ograniczoną powtarzalność maszyny CNC. Obniżenie wartości progowej błędu konturu o 0.005mm pozwala na skompensowanie tych rozbieżności.

Autor za najcenniejsze osiągnięcia własne uważa:

— Opracowanie neuronowego predyktora błędu konturu umożliwiającego przewidywanie błędu konturu.
— Opracowanie innowacyjnego algorytmu optymalizacji błędu konturu wykorzystującego opracowany predyktor neuronowy w celu ograniczenia maksymalnej wartości błędu konturu.
— Budowę układu sterowania CNC wykorzystujący komputer PC z systemem czasu rzeczywistego, w którym zaimplementowano opracowany algorytm.

— Eksperymentalną weryfikację działania opracowanego algorytmu optymalizacji prędkości posuwu w układzie sterowania CNC.

Zdaniem autora najważniejszym elementem innowacyjnym w opracowanym algorytmie jest uwzględnienie błędu konturu w procesie optymalizacji prędkości posuwu dzięki zastosowaniu sztucznych sieci neuronowych do predykcji błędu konturu. Posiadana przez autora wiedza z zakresu optymalizacji parametrów pracy maszyn CNC wykazuje, że problem ograniczenia błędu konturu w procesie generacji czasowo-optymalnych profili prędkości nie został rozwiązany w dostępnej literaturze. Innowacyjnym wkładem autora w dziedzinę optymalizacji prędkości posuwu maszyn wieloosiowych jest także zastosowanie sztucznych sieci neuronowych do predykcji błędu konturu wraz z opracowaną metodologią uczenia w zamkniętej pętli układu sterowania. Całość opracowanego algorytmu została zaimplementowana w języku C dzięki czemu możliwe jest jego zastosowanie na różnych platformach sprzętowych bez konieczności stosowania specjalistycznego oprogramowania.

Duże znaczenie praktyczne ma zbudowany przez autora układ sterowania maszyn CNC wykorzystujący komputer PC i system operacyjny czasu rzeczywistego. W środowisku czasu rzeczywistego autor zaimplementował stos protokołu komunikacyjnego Ethernet Powerlink do komunikacji z serwonapędami. Pozwala to na wykorzystanie opracowanego układu do sterowania różnego rodzaju maszynami wyposażonymi w komercyjne napędy różnych producentów. Dzięki opracowanemu układowi sterowania algorytm optymalizacji prędkości posuwu może zostać wykorzystany do sterowania wieloma komercyjnymi maszynami wieloosiowymi.

Badania doświadczalne opracowanego algorytmu ujawniły rozbieżność pomiędzy przewidywanym a rzeczywistym błędem konturu. Rozbieżność ta wynika z niedoskonałości predyktora neuronowego oraz ograniczonej powtarzalności maszyny wieloosiowej, która posłużyła za stanowisko doświadczalne. Ponieważ występowanie błędów predykcji oraz błędów powtarzalności jest nieuniknione, dopuszczalna wartość błędu konturu wprowadzana do algorytmu jest obniżana o 0.005 mm w stosunku do wartości zadanej. Wartość ta została dobrana eksperymentalnie tak aby zapewnić nie przekraczanie zadanej wartości błędu konturu pomimo występowania błędów predykcji i powtarzalności. Badania doświadczalne wykazały, że korekcja wartości progowej błędu konturu powoduje, że rzeczywisty błąd konturu mieści się w zadanej tolerancji.

Testując opracowany algorytm stwierdzono stosunkowo długi czas obliczeń związanych z wyznaczaniem optymalnego profilu prędkości (kilka minut), w stosunku do czasu realizacji pojedynczej trajektorii ruchu (kilkanaście sekund). Fakt ten uniemożliwia użycie algorytmu, w obecnej postaci, do optymalizacji on-line tzn. w trakcie pracy maszyny. Nie jest to problemem w produkcji wielkoseryjnej gdzie raz wygenerowaną czasowo-optymalną trajektorię ruchu

realizuje się wielokrotnie. Łączny zaoszczędzony czas przy każdym powtórzeniu znacznie przewyższa czas potrzebny na przeprowadzenie jednokrotnej optymalizacji.

Rozwiązanie powyższych problemów stanowi, według autora, interesujący kierunek dalszych badań. Czas obliczeń może zostać znacznie skrócony dzięki wykorzystaniu obliczeń równoległych na wielu rdzeniach procesora lub przy użyciu układów graficznych. Zastosowanie algorytmu PSO oraz sieci neuronowych istotnie ułatwia proces zrównoleglenia obliczeń. Problem rozbieżności pomiędzy rzeczywistym a przewidywanym błędem konturu może zostać rozwiązany poprzez udoskonalenie neuronowego predyktora błędu konturu. Kolejnym możliwym kierunkiem dalszych badań jest rozszerzenie opracowanego algorytmu optymalizacji prędkości posuwu dla maszyn 3 i 5 osiowych.

Autor uważa, że opracowany algorytm może znaleźć zastosowanie wszędzie tam gdzie konieczna jest wielokrotna realizacja zadanej trajektorii narzędzia z dużymi prędkościami np. w procesie cięcia blach, tworzyw sztucznych czy skóry. Prace nad wdrożeniem opracowanego przez autora systemu sterowania wraz z opracowaną metodą optymalizacji prędkości posuwu będą kontynuowane we współpracy z Przemysłowym Instytutem Automatyki i Pomiarów Oddział Badawczo-Rozwojowy Urządzeń Sterowania Napędów w Toruniu (PIAP-OBRUSN Toruń).

Bibliografia

[1] H. Akima. A new method of interpolation and smooth curve fitting based on local procedures. *Journal of the ACM (JACM)*, 17(4):589–602, 1970.

[2] H. Akima. Algorithm 526: Bivariate interpolation and smooth surface fitting for irregularly distributed data points [e1]. *ACM Transactions on Mathematical Software (TOMS)*, 4(2):160–164, 1978.

[3] Y. Altintas and K. Erkorkmaz. Feedrate optimization for spline interpolation in high speed machine tools. *CIRP Annals-Manufacturing Technology*, 52(1):297–302, 2003.

[4] 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.

[5] A. Barbalace, A. Luchetta, G. Manduchi, M. Moro, A. Soppelsa, and C. Taliercio. Performance comparison of vxworks, linux, rtai and xenomai in a hard real-time application. In *Real-Time Conference, 2007 15th IEEE-NPSS*, pages 1–5. IEEE, 2007.

[6] P.J. Barre, R. Bearee, P. Borne, and E. Dumetz. Influence of a jerk controlled movement law on the vibratory behaviour of high-dynamics systems. *Journal of Intelligent and Robotic Systems*, 42(3):275–293, 2005.

[7] D.P. Bertsekas. *Constrained optimization and Lagrange Multiplier methods*. Academic Press, 1982, 1982.

[8] X. Beudaert, P.Y. Pechard, and C. Tournier. 5-axis tool path smoothing based on drive constraints. *International Journal of Machine Tools and Manufacture*, 51(12):958–965, 2011.

[9] E.G. Birgin and J.M. Martinez. Improving ultimate convergence of an augmented lagrangian method. *Optimization Methods and Software*, 23(2):177–195, 2008.

[10] J.E. Bobrow, S. Dubowsky, and J. Gibson. Time-optimal control of robotic manipulators along specified paths. *The International Journal of Robotics Research*, 4(3):3 – 17, 1985.

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

[12] X.D. Chen, J.H. Yong, G. Wang, J.C. Paul, and G. Xu. Computing the minimum distance between a point and a nurbs curve. *Computer-Aided Design*, 40(10):1051–1054, 2008.

[13] C.W. Cheng, M.C. Tsai, and J. Maciejowski. Accurate feedrate control of cnc machine tools along nurbs curves. In *Decision and Control, 2004. CDC. 43rd IEEE Conference on*, volume 3, pages 3053–3058. IEEE, 2004.

[14] M.Y. Cheng and C.C. Lee. On real-time contour error estimation for contour following tasks. *Proceedings of IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, pages 1047–1052, 2005.

[15] M.Y. Cheng, M.C. Tsai, and J.C. Kuo. Real-time nurbs command generators for cnc servo controllers. *International Journal of Machine Tools and Manufacture*, 42(7):801–813, 2002.

[16] 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.

[17] 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.

[18] J.R. Conway, C.A. Ernesto, R.T. Farouki, and M. 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.

[19] D. Costantinescu and EA Croft. Smooth and time-optimal trajectory planning for industrial manipulators along specified paths. *Journal of Robotic Systems*, 17(5):233–249, 2000.

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

[21] H. Dehnavi, M.R. Movahhedy, A. Naebi, and S. Pasban. Prediction of the effect of heat generation in ballscrew on the accuracy of cnc milling machine. In *14th International Conference on Computer Modelling and Simulation (UKSim)*, pages 278–282. IEEE, 2012.

[22] J. Dong, P.M. Ferreira, and J.A. Stori. Feed-rate optimization with jerk constraints for generating minimum-time trajectories. *International Journal of Machine Tools and Manufacture*, 47(12):1941–1955, 2007.

[23] J. Dong and J.A. Stori. A generalized time-optimal bidirectional scan algorithm for constrained feed-rate optimization. *Journal of dynamic systems, measurement, and control*, 128(2):379–390, 2006.

[24] 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.

[25] R. Eberhart and Y. 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.

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

[27] C. Eksioglu, ZM Kilic, and Y. Altintas. Discrete-time prediction of chatter stability, cutting forces, and surface location errors in flexible milling systems. *Journal of Manufacturing Science and Engineering*, 134(6):061006, 2012.

[28] A.M. El-Khalick and N. Uchiyama. Contouring controller design based on iterative contour error estimation for three-dimensional machining. *Robotics and Computer-Integrated Manufacturing*, 27(4):802–807, 2011.

[29] K Erkorkmaz and Y Altintas. High speed contouring control algorithm for cnc machine tools. *Proceedings of the ASME International Mechanical Engineering Congress and Exposition (IMECE'98), Anaheim, California*, pages 463–469, 1998.

[30] K. Erkorkmaz and Y. Altintas. Quintic spline interpolation with minimal feed fluctuation. *Journal of manufacturing science and engineering*, 127(2):339–349, 2005.

[31] K Erkorkmaz, A Alzaydi, A Elfizy, and S Engin. Time-optimal trajectory generation for 5-axis on-the-fly laser drilling. *CIRP Annals-Manufacturing Technology*, 60(1):411–414, 2011.

[32] K. Erkorkmaz and M. Heng. A heuristic feedrate optimization strategy for nurbs toolpaths. *CIRP Annals-Manufacturing Technology*, 57(1):407–410, 2008.

[33] K. Erwinski, M. Paprocki, L. Grzesiak, K. Karwowski, and A. Wawrzak. Application of ethernet powerlink for communication in a linux rtai open cnc control system. *IEEE Transactions on Industrial Electronics*, 60(2):628–636, 2013.

[34] 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.

[35] G.E. Farin. *Curves and Surfaces for Computer-Aided Geometric Design: A Practical Code*. Academic Press, Inc., 1996.

[36] R.T. Farouki and T. Sakkalis. Real rational curves are not unit speed. *Computer Aided Geometric Design*, 8(2):151–157, 1991.

[37] R.T Farouki and Y.F. Tsai. Exact taylor series coefficients for variable-feedrate cnc curve interpolators. *Computer-Aided Design*, 33(2):155–165, 2001.

[38] 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.

[39] R. Gourdeau and H.M. Schwartz. Optimal control of a robot manipulator using a weighted time-energy cost function. *Proceedings of the 28th IEEE Conference on Decision and Control*, pages 1628–1631, 1989.

[40] J.X. Guo, K. Zhang, Q. Zhang, and X.S. Gao. Efficient time-optimal feedrate planning under dynamic constraints for a high-order cnc servo system. *Computer-Aided Design*, 45(12):1538–1546, 2013.

[41] W. Habrat. *Obsługa i programowanie obrabiarek CNC. Podręcznik Operatora.* Wydawnictwo KaBe, 2007.

[42] M. Heng and K. Erkorkmaz. Design of a nurbs interpolator with minimal feed fluctuation and continuous feed modulation capability. *International Journal of Machine Tools and Manufacture*, 50(3):281–293, 2010.

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

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

[45] M. Hu. Kinematics models and development of control system of 2-prr parallel machine. *Mechanical Engineering and Technology*, pages 367–372, 2012.

[46] Y. Hun Jeong, B.K. Min, and D.W. 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.

[47] 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.

[48] Y. Jin, Z. Bi, C. Higgins, M. Price, W. Chen, and T. Huang. Optimal design of a new parallel kinematic machine for large volume machining. *Advances in Reconfigurable Mechanisms and Robots I*, pages 343–354, 2012.

[49] S.G. Johnson. *The NLopt nonlinear-optimization package*, 2014 (dostęp 11.06.2014). http://ab-initio.mit.edu/wiki/index.php/NLopt.

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

[51] J.F. Kennedy, J. Kennedy, and R. Eberhart. *Swarm intelligence*. Morgan Kaufmann, 2001.

[52] Y. Koren. *Computer control of manufacturing systems*. McGraw-Hill New York et al., 1983.

[53] Y. Koren and R.S. Lin. Five-axis surface interpolators. *CIRP Annals-Manufacturing Technology*, 44(1):379–382, 1995.

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

[55] Y. Koren, C.C. Lo, and M. Shpitalni. Cnc interpolators: Algorithms and analysis. *ASME Manufacture Science and Engineering*, 64:83 – 92, 1993.

[56] J. Kosmol. *Serwonapędy obrabiarek sterowanych numerycznie*. Wydawnictwa Naukowo-Techniczne, 1998.

[57] J. Kosmol. *Automatyzacja obrabiarek i obróbki skrawaniem*. Wydawnictwa Naukowo-Techniczne, 2000.

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

[59] J.Y. Lai, K.Y. Lin, S.J. Tseng, and W.D. Ueng. On the development of a parametric interpolator with confined chord error, feedrate, acceleration and jerk. *The International Journal of Advanced Manufacturing Technology*, 37(1):104–121, 2008.

[60] J.M. Langeron, E. Duc, C. Lartigue, and P. Bourdet. A new format for 5-axis tool path computation, using bspline curves. *Computer-Aided Design*, 36(12):1219–1229, 2004.

[61] C. Lartigue, C. Tournier, M. Ritou, and D. Dumur. High-performance nc for hsm by means of polynomial trajectories. *CIRP Annals-Manufacturing Technology*, 53(1):317–320, 2004.

[62] A.C. Lee, M.T. Lin, Y.R. Pan, and W.Y. Lin. The feedrate scheduling of nurbs interpolator for cnc machine tools. *Computer-Aided Design*, 43(6):612–628, 2011.

[63] S.R. Li, Q. Zhang, X.S. Gao, and H. Li. Minimum time trajectory planning for five-axis machining with general kinematic constraints. *Mathematics Mechanization Research Preprints KLMM, Chinese Academy of Sciences*, 31:1–20, 2012.

[64] K.Y. Lin, W.D. Ueng, and J.Y. Lai. Cnc codes conversion from linear and circular paths to nurbs curves. *The International Journal of Advanced Manufacturing Technology*, 39(7):760–773, 2008.

[65] M.T. Lin, M.S. Tsai, and H.T. Yau. Development of a dynamics-based nurbs interpolator with real-time look-ahead algorithm. *International Journal of Machine Tools and Manufacture*, 47(15):2246–2262, 2007.

[66] X. Liu, F. Ahmad, K. Yamazaki, and M. Mori. Adaptive interpolation scheme for nurbs curves with the integration of machining dynamics. *International Journal of Machine Tools and Manufacture*, 45(4):433–444, 2005.

[67] L. Ljung. *System Identification: Theory for the User,*. Prentice Hall, New Jersey,, 1999.

[68] A.J. Lubbe. Mathematical basis for three dimensional circular interpolation on cnc machines. *Southern African Journal of Industrial Engineering*, 8(2):47–59, 1997.

[69] S. Macfarlane and E.A. Croft. Jerk-bounded manipulator trajectory planning: Design for real-time applications. *Robotics and Automation, IEEE Transactions on*, 19(1):42–52, 2003.

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

[71] S.H. Nam and M.Y. Yang. A study on a generalized parametric interpolator with real-time jerk-limited acceleration. *Computer-Aided Design*, 36(1):27–36, 2004.

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

[73] M.G.A. Nassef, C. Schenck, and B. Kuhfuss. Simulation-based parameter identification of a reduced model using neural networks. In *9th IEEE International Conference on Control and Automation (ICCA)*, pages 974–978. IEEE, 2011.

[74] V. Oikonen. *Turku PET Centre*, 2012 (dostęp 11.06.2014). http://www.turkupetcentre.net.

[75] 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.

[76] A. Piazzi and A. Visioli. Global minimum-time trajectory planning of mechanical manipulators using interval analysis. *International journal of Control*, 71(4):631–652, 1998.

[77] L. A. Piegl and W. Tiller. *The NURBS book (2nd ed.)*. Springer-Verlag New York, Inc., New York, NY, USA, 1997.

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

[79] M.J.D. Powell. The bobyqa algorithm for bound constrained optimization without derivatives. *Cambridge NA Report NA2009/06, University of Cambridge, Cambridge*, 2009.

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

[81] D. Renton and M.A. Elbestawi. High speed servo control of multi-axis machine tools. *International Journal of Machine Tools and Manufacture*, 40(4):539–559, 2000.

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

[83] 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.

[84] I. Selimovic. Improved algorithms for the projection of points on nurbs curves and surfaces. *Computer Aided Geometric Design*, 23(5):439–445, 2006.

[85] 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.

[86] B. Sencer, Y. Altintas, and E. Croft. Modeling and control of contouring errors for five-axis machine tools. part i: Modeling. *Journal of manufacturing science and engineering*, 131(3), 2009.

[87] 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.

[88] 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.

[89] Z. Shiller and H.H. Lu. Robust computation of path constrained time optimal motions. *IEEE International Conference on Robotics and Automation,Cincinnati, OH, May 13–18*, pages 144–149, 1990.

[90] Z. Shiller and H.H. Lu. Computation of path constrained time optimal motions with dynamic singularities. *ASME Journal of dynamic systems, measurement, and control*, 114(1):34–40, 1992.

[91] K. Shin and N. McKay. Minimum-time control of robotic manipulators with geometric path constraints. *IEEE Transactions on Automatic Control*, 30(6):531–541, 1985.

[92] K. Shin and N. McKay. A dynamic programming approach to trajectory planning of robotic manipulators. *IEEE Transactions on Automatic Control,*, 31(6):491–500, 1986.

[93] M. Shpitalni, Y. Koren, and C.C. Lo. Realtime curve interpolators. *Computer-Aided Design*, 26(11):832 – 838, 1994.

[94] A. Siu. Real time trajectory generation and interpolation. Master's thesis, University of British Columbia, 2011.

[95] H.C. Song, X. Xu, K.L. Shi, and J.H. Yong. Projecting points onto planar parametric curves by local biarc approximation. *Computers & Graphics*, 38:183–190, 2014.

[96] S.H. Suh, S.K. Kang, D.H. Chung, and I. Stroud. *Theory and design of CNC systems*. Springer, 2008.

[97] S.D. Timar, R.T. Farouki, T.S. Smith, and C.L. Boyadjieff. Algorithms for time–optimal control of cnc machines along curved tool paths. *Robotics and Computer-Integrated Manufacturing*, 21(1):37–53, 2005.

[98] M.S. Tsai, H.W. Nien, and H.T. Yau. Development of an integrated look-ahead dynamics-based nurbs interpolator for high precision machinery. *Computer-Aided Design*, 40(5):554–566, 2008.

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

[100] 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.

[101] H. Wang, X. Qin, C. Ren, and Q. Wang. Prediction of cutting forces in helical milling process. *The International Journal of Advanced Manufacturing Technology*, 58(9):849–859, 2012.

[102] M. Weck, A. Meylahn, and C. Hardebusch. Innovative algorithms for spline-based cnc controller. *Annals of the German Academic Society for Production Engineering*, 6(1):83–86, 1999.

[103] J. Wu, H. Zhou, X. Tang, and J. Chen. A nurbs interpolation algorithm with continuous feedrate. *The International Journal of Advanced Manufacturing Technology*, 59(5-8):623–632, 2012.

[104] H. Xie, H. Tang, and Y.H. Liao. Time series prediction based on narx neural networks: An advanced approach. In *2009 International Conference on Machine Learning and Cybernetics*, volume 3, pages 1275–1279. IEEE, 2009.

[105] D.C.H Yang and T. Kong. Parametric interpolator versus linear interpolator for precision cnc machining. *Computer-Aided Design*, 26(3):225 – 234, 1994.

[106] J. Yang and Z. Li. A novel contour error estimation for position loop-based cross-coupled control. *IEEE/ASME Transactions on Mechatronics*, 16(4):643–655, 2011.

[107] S.S. Yeh and P.L. Hsu. Adaptive-feedrate interpolation for parametric curves with a confined chord error. *Computer-Aided Design*, 34(3):229–237, 2002.

[108] S.S. Yeh and P.L. Hsu. Estimation of the contouring error vector for the cross-coupled control design. *IEEE/ASME Transactions on Mechatronics*, 7(1):44–51, 2002.

[109] T. Yong and R. Narayanaswami. A parametric interpolator with confined chord errors, acceleration and deceleration for nc machining. *Computer-Aided Design*, 35(13):1249–1259, 2003.

[110] G.W. Younkin. Modeling machine tool feed servo drives using simulation techniques to predict performance. *IEEE Transactions on Industry Applications*, 27(2):268–274, 1991.

[111] A.M. Zain, H. Haron, and S. Sharif. Prediction of surface roughness in the end milling machining using artificial neural network. *Expert Systems with Applications*, 37(2):1755–1768, 2010.

[112] N. Zeroudi, M. Fontaine, and K. Necib. Prediction of cutting forces in 3-axes milling of sculptured surfaces directly from cam tool path. *Journal of Intelligent Manufacturing*, pages 1–15, 2012.

[113] 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.

[114] Q. Zhang, S. Li, and J. Guo. Smooth time-optimal tool trajectory generation for cnc manufacturing systems. *Journal of Manufacturing Systems*, 2012.

[115] Q. Zhang and S.R. Li. Efficient computation of smooth minimum time trajectory for cnc machining. *The International Journal of Advanced Manufacturing Technology*, pages 1–10, 2013.

[116] L.M. Zhu, H. Zhao, and H. Ding. Real-time contouring error estimation for multi-axis motion systems using the second-order approximation. *International Journal of Machine Tools and Manufacture*, 68(5):75–80, 2013.

Załącznik 1: Kod źródłowy wybranych funkcji

```
/* FUNKCJA CELU */
/* x przeskalowane [0,1] */
double v_opt_objective(
unsigned int n, const double *x, double *grad, void *data)
{
  double obj_val=0.0;
  unsigned int i;

  for(i=0;i<n;i++){
    obj_val += (1-x[i])*(1-x[i]);
  }

  obj_val = obj_val/((double)n);

  return(obj_val);
}
```

```
/* FUNKCJA OGRANICZEN */
/* x przeskalowane do [0,1] */
void v_opt_constraints(unsigned m, double *result, unsigned n,
            const double* x, double* grad, void* data)
{
  ts_con_data* datstr;// struktura danych
  nurbs_curve* curve;// struktura krzywej NURBS definiujacej tor ruchu
  ts_feed_profile* fprof;// struktura profilu predkosci
  ts_akima* akistr;//struktura danych do interpolacji metoda Akima
  cart_point Pt,dPt,ddPt,dddPt,eout,v,a,j;
  double u, u_old, cerr,cerr_old, cerr_itp;
  double tv,ta,tj;
  unsigned int i,k,l;

  datstr = (ts_con_data*)data;
  curve = &(datstr->curve);
  fprof = &(datstr->fprof);
  akistr = &(datstr->akima);
  eout.x=0.0;
  eout.y=0.0;
  eout.z=0.0;
```

```
22
23  /* zapisanie obecnego stanu profilu predkosci */
24  memcpy(fprof->tmp_v_pts,&(fprof->v_pts[fprof->v_start]),n*sizeof(double));
25
26  /* przeskalowanie zmiennych optymalizacyjnych
27   * i zapisanie ich w profilu predkosci */
28  for(i=0;i<n;i++){
29    fprof->v_pts[fprof->v_start + i] =
30    x[i]*(fprof->maxtv - fprof->mintv) + fprof->mintv;
31  }
32
33  /* wyznaczenie wartosci ograniczen predkosci, przyspieszenia i zrywu
34   * w pkt. weryfikacji ograniczen */
35  for(i=0;i<(fprof->ncep);i++)
36  {
37    /* wyznaczenie wartosci krzywej NURBS w wybranych punktach */
38    nurbs_deBoor_eval(fprof->cep[i], -1, curve, &Pt, &dPt, &ddPt, &dddPt);
39
40    /* wartosc wypadkowej predkosci, przyspieszenia i zrywu */
41    eval_feed_profile(fprof->cep[i],-1,fprof,&tv,&ta,&tj,&dPt, &ddPt);
42
43    /* wyznaczenie unormowanych wartosci ograniczen */
44    compute_vaj_constraints(
45        &(result[i]),&(result[fprof->ncep + i]),NULL,
46        &(result[2*fprof->ncep + i]),&(result[3*fprof->ncep + i]),NULL,
47        &(result[4*fprof->ncep + i]),&(result[5*fprof->ncep + i]),NULL,
48        &dPt, &ddPt, &dddPt, tv,ta,tj);
49  }
50
51  /* zapisanie stanu modelu neuronowego */
52  narx_store_states_double(datstr->netx,datstr->nety);
53  akima_save(akistr);
54  akistr->n = 0;
55  akistr->p = 0;
56  i=0; k=0; l=0;
57  u = datstr->last_u;
58  u_old = datstr->last_u;
59  cerr = datstr->last_ce;
60  cerr_old = datstr->last_ce;
61
62  /* symulacja modelu neuronowego i wyznacznie bledu konturu */
63  while(u < fprof->u_pts[fprof->v_end])
64  {
65    /* wyznaczenie predkosci w osiach -> wartosc wejsciowa do NN*/
66    nurbs_deBoor_eval(u, -1, curve, &Pt, &dPt, &ddPt, &dddPt);
```

```
67      eval_feed_profile(u,-1,fprof,&tv,&ta,&tj,&dPt,&ddPt);
68      compute_axial_vaj(&v,&a,&j,&dPt,&ddPt,&dddPt,tv,ta,tj);
69
70      /* odp. NN w kolejnej dyskretnej chwili czasu
71      bledy nadazania w osiach */
72      narx_1run_double(datstr->netx,datstr->nety,
73              v.x,v.y,&(eout.x),&(eout.y),0);
74
75      /* wyznaczenie bledu konturu */
76      cerr = contour_error4(&eout, &Pt, &dPt, &ddPt);
77
78      /* interpolacja bledu konturu
79       * wielomianem 3-go stopnia metoda Akima
80       * pozwala wyznaczyc wartosci bledu konturu
81       * w punktach weryfikacji ograniczen */
82      akima_shift_store(akistr,u,cerr);
83      if((akistr->n >= 4)){
84        while((fprof->cep[i] >= akistr->x[2]) &&
85         (fprof->cep[i] < akistr->x[3]))
86        {
87          akima_calc(akistr);
88          cerr_itp = akima_eval(akistr,fprof->cep[i]);
89          result[6*fprof->ncep + i] = cerr_itp/MAX_CONERR - 1.0;
90          i++;
91        }
92      }
93
94      cerr_old = cerr;
95      u_old = u;
96      /* interpolacja NURBS - kolejny punkt na krzywej */
97      u = get_u_Taylor3(tv, ta, tj, u, fprof->dT, &dPt, &ddPt, &dddPt);
98      k++;
99    }
100
101   /* przywrocenie stanu poczatkowego */
102   akima_restore(akistr);
103   narx_restore_states_double(datstr->netx,datstr->nety);
104   memcpy(&(fprof->v_pts[fprof->v_start]),fprof->tmp_v_pts,n*sizeof(double));
105 }
```

```
1  /* SYMULACJA PARY SIECI NEURONOWYCH */
2  void narx_1run_double(ts_narx* netx, ts_narx* nety,
3    double inx, double iny, double* outx, double* outy,
4    unsigned int no_state_save)
5  {
```

```
6   unsigned int i=0,j=0,p=0;
7   unsigned int oi = netx->num_h1;
8   const unsigned int num_inp = netx->num_inp;
9   const unsigned int num_h1 = netx->num_h1;
10  double* out_ptr_x = netx->h1_out;
11  double* out_ptr_y = nety->h1_out;
12
13  if(no_state_save){
14  /* zapis stanu sieci */
15    memcpy(netx->inputs_bckp,netx->inputs,num_inp*sizeof(double));
16    memcpy(nety->inputs_bckp,nety->inputs,num_inp*sizeof(double));
17  }
18
19  netx->inputs[0] = inx*(netx->in_scale);
20  nety->inputs[0] = iny*(nety->in_scale);
21
22  /* 1sza warstwa */
23  for(i=0;i<num_h1;i++){
24    netx->h1_out[i] = 0.0;
25    nety->h1_out[i] = 0.0;
26    for(j=0;j<num_inp;j++){
27      netx->h1_out[i] +=
28        netx->inputs[j]*netx->w[i*(num_inp+1)+j];// dotprod(IW,inp)
29      nety->h1_out[i] +=
30        nety->inputs[j]*nety->w[i*(num_inp+1)+j];// dotprod(IW,inp)
31    }
32    /* tanh+bias */
33    netx->h1_out[i] =
34      2.0/(1.0+exp(-2.0*(netx->h1_out[i] + netx->w[i*(num_inp+1)+j])))-1.0;
35    nety->h1_out[i] =
36      2.0/(1.0+exp(-2.0*(nety->h1_out[i] + nety->w[i*(num_inp+1)+j])))-1.0;
37  }
38
39  p = (i-1)*(num_inp+1)+j + 1;
40
41  /* 2ga warstwa */
42  if(netx->num_h2 > 0){
43    for(i=0;i<(netx->num_h2);i++){
44      netx->h2_out[i] = 0.0;
45      nety->h2_out[i] = 0.0;
46      for(j=0;j<num_h1;j++){
47        netx->h2_out[i] +=
48          netx->h1_out[j]*netx->w[p + i*(num_h1+1)+j];// dotprod(IW,inp)
49        nety->h2_out[i] +=
50          nety->h1_out[j]*nety->w[p + i*(num_h1+1)+j];// dotprod(IW,inp)
```

```
51        }
52        netx->h2_out[i] =
53          2.0/(1.0+exp(-2.0*(netx->h2_out[i] +
54           netx->w[p + i*(num_h1+1)+j])))-1.0;
55        nety->h2_out[i] =
56          2.0/(1.0+exp(-2.0*(nety->h2_out[i] +
57           nety->w[p + i*(num_h1+1)+j])))-1.0;
58      }
59      oi = netx->num_h2;
60      out_ptr_x = netx->h2_out;
61      out_ptr_y = nety->h2_out;
62      p = p + (i-1)*(netx->num_h1+1) + j + 1;
63    }
64
65    /* warstwa wyjsciowa */
66    netx->output = 0.0;
67    nety->output = 0.0;
68    for(j=0;j<oi;j++){
69      netx->output += out_ptr_x[j]*netx->w[p + j];// dotprod(IW,inp)
70      nety->output += out_ptr_y[j]*nety->w[p + j];// dotprod(IW,inp)
71    }
72    netx->output += netx->w[p + j];// add bias, purelin
73    nety->output += nety->w[p + j];// add bias, purelin
74
75    if(no_state_save){
76      /* przywrocenie stanu sieci */
77      memcpy(netx->inputs,netx->inputs_bckp,num_inp*sizeof(double));
78      memcpy(nety->inputs,nety->inputs_bckp,num_inp*sizeof(double));
79    }
80    else{
81      for(i=(num_inp-1);i>0;i--){
82        netx->inputs[i] = netx->inputs[i-1];
83        nety->inputs[i] = nety->inputs[i-1];
84      }
85      netx->inputs[netx->num_id] = netx->output;
86      nety->inputs[nety->num_id] = nety->output;
87    }
88
89    *outx = netx->output*(netx->out_scale);
90    *outy = nety->output*(nety->out_scale);
91  }
```

```
1  /* INTERPOLACJA WIELOMIANEM 3 STOPNIA */
2  /* METODA AKIMA */
3  void akima_calc(ts_akima* akistr)
```

```
{
  int i=2,j;
  double bp1;
  double yr1,yr2,yl1,yl2;

  if(akistr->n >= 4)
  {
    if(akistr->p>=0)
    {
      if(akistr->n > 5){
        for(j=0;j<5;j++){
          akistr->dx[j] = akistr->x[j+1] - akistr->x[j];
          akistr->dy[j] = (akistr->y[j+1] - akistr->y[j])/akistr->dx[j];
        }
      }
      else if(akistr->n == 4){
        for(j=2;j<5;j++){
          akistr->dx[j] = akistr->x[j+1] - akistr->x[j];
          akistr->dy[j] = (akistr->y[j+1] - akistr->y[j])/akistr->dx[j];
        }
        yl1 = akistr->dx[3]*(akistr->dy[3] -
          2*(akistr->dy[2])) + akistr->y[2];//yl1
        akistr->dy[1] = (akistr->y[2] - akistr->dy[1])/(akistr->dx[3]);
        yl2 = akistr->dx[2]*(akistr->dy[2] -
          2*(akistr->dy[1])) + akistr->y[2];//yl2
        akistr->dy[0] = (yl1 - yl2)/(akistr->dx[2]);//sl2
      }
      else if(akistr->n == 5){
        for(j=1;j<5;j++){
          akistr->dx[j] = akistr->x[j+1] - akistr->x[j];
          akistr->dy[j] = (akistr->y[j+1] - akistr->y[j])/akistr->dx[j];
        }
        yl1 = akistr->dx[2]*(akistr->dy[2] -
          2*(akistr->dy[1])) + akistr->y[1];//yl1
        akistr->dy[0] = (akistr->y[1] - yl1)/(akistr->dx[2]);//sl1
      }
    }
    else
    {
      if(akistr->p == -1)
      {
        for(j=0;j<4;j++){
          akistr->dx[j] = akistr->x[j+1] - akistr->x[j];
          akistr->dy[j] = (akistr->y[j+1] - akistr->y[j])/akistr->dx[j];
        }
```

```
49          yr1 = (akistr->dx[2])*(2*akistr->dy[3] -
50            akistr->dy[2]) + akistr->y[4];
51          akistr->dy[4] = (yr1 - akistr->y[4])/(akistr->dx[2]);
52        }
53        if(akistr->p == -2)
54        {
55          for(j=0;j<3;j++){
56            akistr->dx[j] = akistr->x[j+1] - akistr->x[j];
57            akistr->dy[j] = (akistr->y[j+1] - akistr->y[j])/akistr->dx[j];
58          }
59          yr1 = (akistr->dx[1])*(2*akistr->dy[2] -
60            akistr->dy[1]) + akistr->y[3];
61          akistr->dy[3] = (yr1 - akistr->y[3])/(akistr->dx[1]);
62          yr2 = (akistr->dx[2])*(2*akistr->dy[3] -
63            akistr->dy[2]) + yr1;
64          akistr->dy[4] = (yr2 - yr1)/(akistr->dx[2]);
65        }
66      }
67
68      /* wyznaczenie wspolczynnikow wielomianu */
69      akistr->b = akima_slope(akistr->dy[i-2], akistr->dy[i-1],
70            akistr->dy[i], akistr->dy[i+1]);
71      bp1 = akima_slope(akistr->dy[i-1], akistr->dy[i],
72          akistr->dy[i+1], akistr->dy[i+2]);
73      akistr->d = (bp1 + akistr->b - 2 * akistr->dy[i])/
74            ((akistr->dx[i])*(akistr->dx[i]));
75      akistr->c = (3*akistr->dy[i] - 2*akistr->b - bp1)/
76            (akistr->dx[i]);
77    }
78    else
79    {
80      printf("ERROR(akima_calc): za malo punktow danych",akistr->n);
81    }
82  }
83
84  inline double akima_slope(double s1, double s2, double s3, double s4) {
85    double sla, sra;
86
87    sla = fabs(s2 - s1);
88    sra = fabs(s4 - s3);
89    if (sla == 0.0 && sra == 0.0)
90      return (0.5 * (s2 + s3));
91    else
92      return ((sra * s2 + sla * s3) / (sla + sra));
93  }
```

```
/* WYZNACZANIE PREDKOSCI, PRZYSPIESZENIA I ZRYWU W OSIACH */
void compute_axial_vaj(cart_point* v, cart_point* a, cart_point* j,
             cart_point* dPt,cart_point* ddPt,cart_point* dddPt,
             double Vt, double At, double Jt)
{
  cart_point C_s, C_ss, C_sss;

  /* pochodne du/ds, d^2u/ds^2, d^3u/ds^3 */
  double u_s = 1/modulus(dPt);
  double u_ss = -dotprod(dPt,ddPt)*pow(u_s,4);
  double u_sss = 4*pow(dotprod(dPt,ddPt),2)*pow(u_s,7) -
          (dotprod(ddPt,ddPt) +
          dotprod(dPt,dddPt))*pow(u_s,5);


  /* pochodne dC(u)/ds,d^2C(u)/ds^2,d^3C(u)/ds^3 */
  C_s.x = dPt->x*u_s;
  C_s.y = dPt->y*u_s;
  C_s.z = dPt->z*u_s;

  C_ss.x = ddPt->x*u_s*u_s + dPt->x*u_ss;
  C_ss.y = ddPt->y*u_s*u_s + dPt->y*u_ss;
  C_ss.z = ddPt->z*u_s*u_s + dPt->z*u_ss;

  C_sss.x = dddPt->x*u_s*u_s*u_s + 3*ddPt->x*u_s*u_ss + dPt->x*u_sss;
  C_sss.y = dddPt->y*u_s*u_s*u_s + 3*ddPt->y*u_s*u_ss + dPt->y*u_sss;
  C_sss.z = dddPt->z*u_s*u_s*u_s + 3*ddPt->z*u_s*u_ss + dPt->z*u_sss;

  /* predkosc w osiach */
  v->x = C_s.x*Vt;
  v->y = C_s.y*Vt;
  v->z = C_s.z*Vt;

  /* przyspieszenie w osiach */
  a->x = C_ss.x*Vt*Vt + C_s.x*At;
  a->y = C_ss.y*Vt*Vt + C_s.y*At;
  a->z = C_ss.z*Vt*Vt + C_s.z*At;

  /* zryw w osiach */
  j->x = C_sss.x*Vt*Vt*Vt + 3*C_ss.x*Vt*At + C_s.x*Jt;
  j->y = C_sss.y*Vt*Vt*Vt + 3*C_ss.y*Vt*At + C_s.y*Jt;
  j->z = C_sss.z*Vt*Vt*Vt + 3*C_ss.z*Vt*At + C_s.z*Jt;
}
```

```
/* MINIMALIZACJA ROZSZERZONEGO LAGRANZJANU */
```

```
/* ALGORYTMEM GCPSO */
result pso_gbest_minimize(int n,/* ilosc zm. opt. */
            func f,/* wsk. na f-cje rozsz. lag. */
            void *f_data,/* dane pomocn. */
            const double *lb,/*g. granica zm. opt*/
            const double *ub,/*d. granica zm. opt*/
            double *x, /* rozwiazanie */
            double *minf,/* rozwiazanie */
            stopping *stop,/* struk. war. stopu */
            int population, /* ilosc czastek */
            int gc_pso, /* flaga GCPSO */
            pso_workspace_t* work/* dane pomocn. */
            )
{
  result ret = NLOPT_FAILURE;
  bobyqa_result retbq;
  int i,j,k;
  int pop=population;
  int loc_alloc=0;
  double fval=0.0, tmp_dist=0.0;
  double regpso_swarm_rad=0.0;
  double nm_minf;
  double nm_x[n];
  int gbest_idx_old = 0;
  double gbest_val_old = work->part[0].pbest_val;
  double rho=1.0;
  int no_fail=0, no_succ=0;

  if(work==NULL)
  {/* alokacja przestrzeni roboczej */
    work = pso_alloc_workspace(pop,n);
    loc_alloc = 1;
  }

  /* losowa inicjalizacja pozycji czastek */
  pso_initialize(work,x,lb,ub,f,f_data);

  /* ALGORYTM GCPSO */
  /* minimalizacja rozszerzonego lagranzjanu metoda GCPSO */
  while((ret<0) && ((work->pso_first_run)!=0))
  {

    if(gc_pso){
      if (((work->gbest_idx) ==
        (gbest_idx_old)) && ((work->gbest_val[0]) >= gbest_val_old)){
```

```
47        no_fail++;
48        no_succ = 0;
49      }
50      else if (((work->gbest_idx) ==
51           gbest_idx_old) && ((work->gbest_val[0]) < gbest_val_old)){
52        no_succ++;
53        no_fail = 0;
54      }
55      else{
56        no_succ = 0;
57        no_fail = 0;
58      }
59
60      if(no_succ > GC_NS){
61        rho *= 2.0;
62        no_succ = 0;
63      }
64      else if (no_fail > GC_NF){
65        rho *= 0.5;
66        no_fail = 0;
67      }
68
69      if (rho < 10e-5)
70        rho = 10e-5;
71      else if (rho > 1.0)
72        rho = 1.0;
73
74      gbest_idx_old = work->gbest_idx;
75      gbest_val_old = work->gbest_val[0];
76    }
77
78    /* iteracja po czastkach */
79    for(i=0;i<pop;i++)
80    {
81      /* iteracja po wymiarach */
82      for(j=0;j<n;j++)
83      {
84        if(lb[j]!=ub[j])
85        {
86          {
87
88            if((gc_pso==1) && (i==(work->gbest_idx))){
89              /*aktualizacja predkosci GCPSO */
90              work->part[i].v[j] =
91              (work->w)*(work->part[i].v[j]) - work->part[i].x[j] +
```

```
92              work->gbest[j] + rho*urand(1.0,-1.0);
93            }else{
94              /* aktualizacja predkosci PSO */
95              work->part[i].v[j] = (work->w)*(work->part[i].v[j]) +
96              urand(0.0,work->c1)*((work->part[i].pbest[j]) -
97              (work->part[i].x[j])) +
98              urand(0.0,work->c2)*(work->gbest[j] - work->part[i].x[j]) ;
99            }
100
101           /* ograniczenie predkosci czastek */
102           if(work->part[i].v[j] > work->vmax[j]){
103             work->part[i].v[j] = work->vmax[j];
104           }
105           else if(work->part[i].v[j] < -(work->vmax[j])){
106             work->part[i].v[j] = -(work->vmax[j]);
107           }
108
109           work->part[i].x[j] =
110           (work->part[i].x[j]) + (work->part[i].v[j]);
111
112           /* ograniczenie pozycji czastek */
113           if((work->part[i].x[j])>(work->cub[j])){
114             work->part[i].v[j] = 0.0;
115             work->part[i].x[j] = work->cub[j];
116           }
117           else if((work->part[i].x[j])<(work->clb[j])){
118             work->part[i].v[j] = 0.0;
119             work->part[i].x[j] = work->clb[j];
120           }
121         }
122       }/* END: if(lb[j]!=ub[j]) */
123     }/* END: for(j=0;j<n;j++) */
124   }/* END: for(i=0;i<pop;i++) */
125
126   for(i=(STOPITER-1);i>0;i--){
127     work->gdist[i] = work->gdist[i-1];
128     work->gbest_val[i] = work->gbest_val[i-1];
129   }
130   work->gdist[0] = 0.0;
131   regpso_swarm_rad = 0.0;
132
133   for(i=0;i<pop;i++)
134   {
135     /* wyznaczenie wartosci rozszerzonego lagranzjanu */
136     fval = f(n, work->part[i].x, NULL, f_data);
```

```
137
138       /* aktualizacja pbest */
139       if(fval<(work->part[i].pbest_val)){
140         work->part[i].pbest_val = fval;
141         memcpy(work->part[i].pbest,work->part[i].x,n*sizeof(double));
142       }
143
144       /* aktualizacja gbest */
145       if(fval < (work->gbest_val[0])){
146         work->gbest_idx = i;
147         work->gbest_val[0] = fval;
148         memcpy(work->gbest, work->part[i].x,n*sizeof(double));
149       }
150       work->part[i].fval = fval;
151
152       /* srednia odleglosc czastek od gbest */
153       tmp_dist = 0.0;
154       for(j=0;j<n;j++){
155         if(ub[j]!=lb[j]){
156           tmp_dist +=
157           (work->part[i].x[j] - work->gbest[j])*
158           (work->part[i].x[j] - work->gbest[j]);
159           if(i==0){
160             work->max_dev[j] =
161             fabs(work->part[i].x[j] - work->gbest[j]);
162           }
163           else if(
164           fabs(work->part[i].x[j] - work->gbest[j]) >
165           work->max_dev[j]){
166             work->max_dev[j] =
167             fabs(work->part[i].x[j] - work->gbest[j]);
168           }
169         }
170       }
171       work->gdist[0] += sqrt(tmp_dist);
172       if(work->gdist[0] > regpso_swarm_rad) regpso_swarm_rad =
173         work->gdist[0];
174     }
175     work->gdist[0] = work->gdist[0]/((double)pop);
176
177     /* inkrementacja licznika iteracji */
178     stop->nevals++;
179
180     /* warunki zbieznosci */
181     if ((work->gbest_val[0]) < (stop->minf_max))
```

```
182     {
183       ret = STOPVAL_REACHED;
184       work->pso_first_run=0;
185     }
186     else if (stop_evals(stop))
187     {
188       ret = MAXEVAL_REACHED;
189       work->pso_first_run=0;
190     }
191     else if( (fabs((work->gdist[0])-(work->gdist[STOPITER-1])) <=
192           PSO_DTOL*(work->gdist[STOPITER-1])) &&
193          (fabs((work->gbest_val[0])-(work->gbest_val[STOPITER-1])) <=
194           PSO_RTOL*(work->gbest_val[STOPITER-1])))
195     {
196       ret = FTOL_REACHED;
197       work->pso_first_run=0;
198     }
199     else
200       ret = FAILURE;
201
202     /* zapis akturalnej pozycji gbest jako rozwiazania */
203     *minf = work->gbest_val[0];
204     memcpy(x,work->gbest,n*sizeof(double));
205
206   }/* END: while(ret<0) */
207
208   /* poprawa dokladnosci rozwiazania algorytmem BOBYQA */
209   *minf = f(n, x, NULL, f_data);
210   memcpy(work->sub_x,x,n*sizeof(double));
211   work->sub_minf = *minf;
212   ret = new_bobyqa(n,2*n+1,work->sub_x,lb,ub,work->sub_istep,
213           -1.0,1e-8,0.0,1e-8,1e-8,15000,NULL,
214           &(work->sub_minf),f,f_data,NULL,1);
215
216   ret = retbq;
217   *minf = work->sub_minf;
218   memcpy(x,work->sub_x,n*sizeof(double));
219   pso_free_workspace(work);
220
221   return ret;
222 }
```