POLITECHNIKA POZNAŃSKA

WYDZIAŁ INŻYNIERII MECHANICZNEJ

ROZPRAWA DOKTORSKA

STEROWANIE ROBOTEM ZA POMOCĄ ODPORNEGO NA AWARIE OSI ALGORYTMU BAZUJĄCEGO NA SZTUCZNEJ INTELIGENCJI

Patryk NOWAK

Poznań 2023

Promotor
Prof. dr hab. inż. Andrzej Milecki

Promotor pomocniczy:
Dr inż. Dominik Rybarczyk

Politechnika Poznańska
Instytut Technologii Mechanicznej
Zakład Urządzeń Mechatronicznych

Samemu idzie się szybciej,

ale we dwójkę dociera się dalej.

G. Musso

Mojej żonie i rodzicom.

Streszczenie

W niniejszej rozprawie podjęto badania nad opracowaniem odpornego na uszkodzenia osi sterowania robotem, w którym przewidziano zastosowanie metod sztucznej inteligencji. Głównym celem pracy jest opracowanie i przebadanie sterowania odpornego na awarię osi manipulatora, zapewniające pozycjonowanie w zadanym punkcie, z jednoczesnym omijaniem przeszkód w strefie roboczej robota. Opracowano połączenie algorytmu genetycznego, sztucznej sieci neuronowej oraz modelu robota do pozycjonowania ramienia robota, w przypadku awarii jednej albo dwóch osi. Opracowane rozwiązanie zostało sprawdzone w badaniach symulacyjnych i doświadczalnych, w zadaniach typu „złap i odłóż" wykonywanych w warunkach laboratoryjnych.

W pracy wykonano przegląd literatury dotyczący zastosowań algorytmów FTC w sterowaniu manipulatorami. Na tej podstawie stwierdzono, że nie ma algorytmu, który pozwalał by na sterowanie robotem w momencie uszkodzenia więcej niż jednej osi z jednoczesnym omijaniem przeszkód. Na początku opracowano algorytm przygotowywania danych uczących przez połączenie modelu robota oraz algorytmu genetycznego. Zaproponowano zastosowanie funkcji nagrody dzięki, której algorytm oceniał każdy krok robota i decydował który ruch ma być wykonany. Do wyznaczania parametrów funkcji nagrody zastosowano algorytm genetyczny oraz sztuczną sieć neuronową w postaci wielowarstwowego perceptronu. Opracowano model kinematyczny robota, który był wykorzystywany w algorytmie do szybkiego wyznaczania wszystkich ruchów jakie robot może wykonać w każdym kroku. Model ten wykorzystywano też w trakcie badań symulacyjnych. Opracowano również model robota uwzględniający jego dynamikę, który był wykorzystywany do obliczania energii jaką robot zużywa w trakcie ruchu. Ten model był wykorzystywany w sterowaniu z kryterium energetycznym. Przedstawiono również algorytmy, dzięki którym przetestowano system na przemysłowym robocie rzeczywistym. Zaproponowany algorytm zweryfikowano w badaniach symulacyjnych i doświadczalnych.

Abstract

In this dissertation, research is undertaken on the development of an axis-failure-tolerant robot control, which envisions the use of artificial intelligence methods. The main objective of the thesis is to develop and test a fault-tolerant robotic axis control that provides positioning at a preset point while avoiding obstacles in the robot's work zone. a combination of a genetic algorithm, an artificial neural network and a robot model was developed for positioning the robot arm, in case of failure of either one or two axes. The developed solution was tested in simulation and experimental studies, in catch-and-put tasks performed in laboratory conditions.

In this paper, a literature review was performed on the applications of FTC algorithms in manipulator control. Based on this, it was found that there is no algorithm that allows controlling a robot when more than one axis is damaged with simultaneous obstacle avoidance. At first, an algorithm was developed to prepare learning data by combining a robot model and a genetic algorithm. It was proposed to use a reward function by which the algorithm evaluated each step of the robot and decided which move to make. a genetic algorithm and an artificial neural network in the form of a multilayer perceptron were used to determine the parameters of the reward function. a kinematic model of the robot was developed and used in the algorithm to quickly determine all the moves the robot could make in each step. This model was also used during simulation studies. a model of the robot taking into account its dynamics was also developed and used to calculate the energy the robot consumes during movement. This model was used in energy-criteria control. The algorithms by which the system was tested on an industrial real robot were also presented. The proposed algorithm was verified in simulation and experimental studies.

SPIS TREŚCI

SPIS WAŻNIEJSZYCH SYMBOLI i SKRÓTÓW

Symbole

$P_s\,(x_s, y_s, z_s)$	- pozycja początkowa oraz bezpieczna dla TCP robota
$P_{t1}\,(x_{t1}, y_{t1}, z_{t1})$	- pierwsza pozycja docelowa dla TCP robota
$P_{t2}\,(x_{t2}, y_{t2}, z_{t2})$	- druga pozycja docelowa dla TCP robota
$O_1 \quad O_2$	- pozycje pierwszej i drugiej przeszkody
α_s	- kąt obrotu osi dla ruchu normalnego
β_s	- kąt obrotu osi dla ruchu dokładnego
$j0 \div j5$	- aktywność osi robota 0 nieaktywna, 1 aktywna
p_m, p_t	- parametry funkcji nagrody
r	- funkcja nagrody
d_e	- odległość do przeszkód [m]
d_g	- odległość do pozycji docelowych [m]
P_{ji}	- punkty charakterystyczne w osi robota służące do określania odległości poszczególnych członów robota od przeszkód gdzie j określa numer członu robota, a i numer punktu na osi robota.
O_{ji}	- punkty charakterystyczne w osi przeszkody służące do określania odległości poszczególnych przeszkód od członów robota, gdzie j określa numer przeszkody, a i numer punktu na osi przeszkody.
θ_{iL}	- przesunięcia kątowe osi i-tej w lewo [°]
θ_{iR}	- przesunięcia kątowe osi i-tej w prawo [°]
$j_0 \div j_5$	- położenia kątowe osi [°]
S	- zbiór wszystkich możliwych przemieszczeń kątowych
t	- czas [s]
MES	- błąd średniokwadratowy
e	- liczba epok uczących
Δ	- błąd będący różnicą wartości nominalnej oraz rzeczywistej
v	- prędkość [m/s]
α	- kąt obrotu [rad]
e	- błąd
a_i	- pozycja osi robota w przestrzeni kartezjańskiej, gdzie i to numer osi
s_{si}	- pozycja centrum masy segmentu i-tego
s_{fi}	- pozycja centrum masy figury i-tej

M_f	- moment siły [Nm]
r	- promień wodzący [m]
F	- siła [N]
I	- moment bezwładności [kg·m^2]
m	- masa
f	- współczynnik zależny od rodzaju i wielkości łożyska oraz współczynnika dopuszczalnego obciążenia statycznego
P_z	- obciążenie zastępcze (równoważne)
d_m	- średnica podziałowa łożyska
P	- moc
U	- napięcie międzyfazowe
I_f	- prąd fazy
α_w	- krok zmiany położenia kątowego wyrażany w stopniach
k	- liczba ustawień pozycji TCP robota w celu sprawdzenia przestrzeni roboczej robota
j_{is}	- liczba możliwych pozycji położenia osi i-tej

Skróty

TCP	- punkt środkowy narzędzia (tool center point)
FTC	- sterowanie odporne na błędy (Fault Tolerant Control)
PFTCS	- pasywne systemy sterowania odpornego na błędy
AFTCS	- aktywne systemy sterowania opornego na błędy
HFTCS	- hybrydowe systemy sterowania odpornego na błędy
FDI	- system wykrywania i izolowania usterek
AFT2BC	- adaptive fuzzy type-2 backstepping control
NTSC	- non-singular terminal synergetic control
IT2FSBO	- interval type-2 fuzzy satin bowerbird optimization
NF	- neuro fuzzy
MLP	- multilayer perceptron
FD	- fault detection
SFT-PID-NFTSM	- self-tuning fuzzy proportional–integral–derivative non-singular fast terminal sliding-mode control
TDE	- time delay estimation

ANHWSO	- adaptive high-order variable structure observer
AMFBVSC	- adaptive modern fuzzy backstepping variable structure controller
SVM	- support vector machine
VSO	- variable structure observer
DNN	- dual neural network
SOMNN	- self-organizing map neural network
RBFNN	- radial basis function neural network
AFTFTCC	- adaptive fixed-time fault-tolerant constraint control
SMC	- sliding mode control
SMO	- sliding mode observer
STW-TOSM	- super-twisting third-order sliding-mode
STW-SOSM	- super-twisting second-order sliding-mode
CTC	- computed torque controller
FxTSOSMO	- fixed-time second-order sliding-mode observer
FxTSMC	- fixed-time sliding-mode controller
ABNFTSMC	- adaptive backstepping nonsingular fast terminal sliding mode control
STRCL	- super-twisted reaching control law
AFTC-SSMC	- active fault tolerant control with synchronous sliding mode control
NRM	- Newtona-Raphson method
RFTTC	- robust fault-tolerant tracking control
ORRCF	- operator theory-based robust right coprime factorization
PPC	- prescribed performance control
AIC	- active inference controller
u-AIC	- unsigned AIC
SVD	- singular value decomposition
SSN	- sztuczna sieć neuronowa
AG	- algorytm genetyczny

1 Wstęp

Pierwszy robot przemysłowy został zastosowany w przemyśle w roku 1961. Obecnie, na świecie zainstalowanych jest w przemyśle ponad 3,5 miliona robotów. Ich liczba rośnie w tempie ok. 0,5 mln sztuk rocznie. w literaturze można znaleźć określenia „robot przemysłowy" i „manipulator przemysłowy", które często występują zamiennie. Główna różnica między nimi dotyczy tego, że manipulatory, kształtem i działaniem odpowiadają ręce człowieka, natomiast roboty to pojęcie bardziej ogólne, obejmujące także inne konfiguracje. w dzisiejszych czasach roboty i manipulatory są powszechnie stosowane w wielu zakładach produkcyjnych [1], na wszelkiego rodzaju zautomatyzowanych liniach produkcyjnych [2]. Zatrzymanie całej linii produkcyjnej w wyniku awarii robota, stanowi poważny problem i jest źródłem znacznych strat przedsiębiorstw. Roboty mogą być również stosowane w środowisku niebezpiecznym dla człowieka [3, 4] lub w miejscach, w których serwisowanie takiego robota może być niemożliwe, jak np. w kosmosie [5]. w takich zastosowaniach powinny one być możliwie najmniej podatne na awarię, ze względu na problemy z dostępem do nich i wysokie koszty naprawy. w takich przypadkach, uzasadnione jest stosowanie redundancji albo algorytmów odpornych na błędy, które można łatwo i nisko kosztowo zaimplementować w sterownikach. Może to znacząco zmniejszyć koszty związane z przestojem. Stanowi to poważny argument do rozwijania algorytmów odpornych na błędy, w zastosowaniu do manipulatorów. Sterowanie odporne na błędy (ang. Fault Tolerant Control – FTC), było najpierw wprowadzone w samolotach, a później przeniesiono je także do innych urządzeń, w których niezawodność pracy jest kluczowa.

W niniejszej rozprawie zaproponowano rozwiązanie, pozwalające na pozycjonowanie robota sześcioosiowego, w przypadku awarii jednej lub dwóch osi. Założono również, że opracowany algorytm będzie zdolny do jednoczesnego omijania przeszkód, znajdujących się w przestrzeni roboczej robota. Postanowiono zainteresować się tą tematyką badawczą, ponieważ nie znaleziono żadnego dobrego rozwiązania, które w przypadku wykrycia awarii osi robota, umożliwia kontynuowanie ruchu z omijaniem przeszkód i zakończenie tego ruchu pozycjonowaniem punktu centralnego narzędzia (ang. Tool Central Point – TCP), zlokalizowanego na ostatnim ramieniu robota . z tego względu rozwiązanie tego problemu uznano równocześnie za potrzebne i za nowatorskie.

W założeniu opracowany system sterowania ma zapewniać wykonanie jednego z najczęściej stawianych robotom przemysłowym zadań, tj. „złap, podnieś i odłóż", także

w przypadku uszkodzenia osi. Algorytm powinien umożliwiać pracę robota także wtedy, gdy w jego przestrzeni roboczej znajdują się maksymalnie dwie przeszkody. Aby rozwiązać takie zadanie należy opracować specjalny algorytmu, który przejmie sterowanie poszczególnymi osiami robota i będzie generował kolejne przemieszczenia kątowe aktywnych osi, w taki sposób, aby przesunąć TCP robota do punktu końcowego, po jak najkrótszej trajektorii, omijając przy tym znajdujące się na niej nieruchome przeszkody. Tematyka niniejszej pracy skupia się na osiągnięciu tak postawionego zadania. Przyjęto również aby w algorytmie zastosować metody bazujące na sztucznej inteligencji.

W rozprawie zaproponowano algorytm krokowy, w którym zastosowano funkcję nagrody. Do wyznaczenia jej optymalnych parametrów użyto algorytmu genetycznego, a do ich uogólnienia użyto sztuczną sieć neuronową. w badanym w pracy rozwiązaniu algorytm genetyczny zastosowano do generowania danych uczących sztuczną sieć neuronową. Zaproponowana strategia sterowania odpornego na błędy wykorzystuje też model symulacyjny sterowanego robota, który służył do szybkiego wyznaczania wszystkich możliwych do wykonania przez niego ruchów. Były one następnie oceniane z użyciem funkcji nagrody. Najlepiej oceniony ruch był wykonywany przez robota.

W rozdziale 2. przedstawiono dotychczasowy stan wiedzy z zakresu sterowania robotami odpornego na błędy. Rozdział 3. zawiera ogólny opis zaproponowanego rozwiązania. Obejmuje on również przedstawienie celów pracy oraz postawienie jej tezy. w rozdziale 4. omówiono metody i narzędzia badawcze, wykorzystywane do udowodnienia postawionej tezy pracy. Opisano w nim również szczegółowe zaproponowane rozwiązania, dotyczące działania algorytmu w sytuacji wystąpienia błędu osi robota. Zawarto w nim także opis architektury sieci neuronowej, zastosowanej w proponowanym rozwiązaniu oraz opis algorytmu genetycznego, służącego do generowania danych uczących. Przedstawiono również metody uczenia oraz model symulacyjny robota, który był wykorzystywany w trakcie działania algorytmu. Opisano również integrację wszystkich omówionych systemów. w rozdziale 5. zamieszczono wyniki badań symulacyjnych oraz doświadczalnych, wraz z opisem metodologii ich wykonania. Ostatni rozdział zawiera podsumowanie pracy.

Zastosowany w pracy algorytm stanowi nowe rozwiązanie, które umożliwia dostosowywanie akcji robota do aktualnego stanu jego osi i otoczenia. Zaproponowane rozwiązanie jest uniwersalne i może być stosowane dla różnych robotów o różnej liczbie

stopni swobody, wykonujących zadnia przemieszczania TCP robota do zadanego punktu, po jak najkrótszej trajektorii.

2 Rozpoznanie stanu wiedzy z zakresu odpornego na błędy sterowania robotami

2.1 Systemy sterowania odporne na błędy

Wszystkie wytworzone przez człowieka urządzenia nie są doskonałe i dlatego występują w nich różnego rodzaju uszkodzenia i błędy. Podlegają one też starzeniu się, w rezultacie którego następuje zużycie najbardziej obciążonych elementów. Rezultatem tego jest nieprawidłowe działanie tych urządzeń oraz ich unieruchomienie. Tymczasem współcześnie powszechnie wymaga się, aby urządzenia pracowały niezawodnie przez określony, najlepiej jak najdłuższy czas. Dlatego podjęto prace badawcze nad opracowaniem różnych rozwiązań, zapewniających niezawodną pracę urządzeń. Jedna z metod polega na zastosowaniu specjalnych rozwiązań w zakresie sterowania, które powinny zapewniać tolerowanie, tzn. poprawną pracę urządzenia mimo wystąpienia błędu, typu awaria jakiegoś elementu składowego. Takie sterowanie nazywa się odpornym na błędy. w terminologii angielskojęzycznej, określa się je następująco: fault tolerant control (FTC). w systemie sterowania tolerującym błędy, konieczne jest rozróżnienie takich pojęć jak: definicja usterki i odporność na błąd. Usterka jest definiowana jako odchylenie parametru od wartości akceptowalnej, a odporność na błędy, to zdolność systemu do kontynuowania pracy niezależnie od błędów. Początki systemów FTC, związane były z koniecznością stosowania specjalnego sterowania, w urządzeniach i aparaturze o krytycznym znaczeniu dla bezpieczeństwa. Przykładami takich urządzeń są samoloty i reaktory bądź całe elektrownie jądrowe. Obecnie techniki FTC są stosowane w wielu urządzeniach przemysłowych, wśród których znajdują się np. roboty. Ich awaria na zautomatyzowanej linii produkcyjnej oznacza zwykle zatrzymanie całej linii, a to jest zwykle wysoce kosztowne. Dlatego możliwość działania urządzenia, chociażby w ograniczony sposób albo wolniej, jest niezwykle pożądana w przemyśle.

Podstawowe pojęcia i techniki dotyczące odporności na błędy, stosowane w systemach komputerowych zostały sformułowane w pracy [6]. w publikacji tej przedstawiono na wstępie różne klasyfikacje błędów oraz aplikację technik redundancji, w celu zapewnienia niezawodnej pracy komputerów. Opisano tam również metody modelowania i przewidywania błędów. Na koniec pokazano przykłady komputerów, tolerujących błędy. Na początku lat 90-tych Stengel [7], Veillette [8] i Patton [9] opublikowali wyniki różnych zastosowań systemów FTC. z kolei w pracy [10] przedstawiono stan wiedzy, na temat zastosowań sterowania tolerującego błędy do 1997 roku.

Jedno z pierwszych, opisanych w literaturze zastosowań sterowania tolerującego błędy w manipulatorach robotów, zostało opisane w pracy [11], opublikowanej w roku 1990. Określono tam miarę wpływu uszkodzenia przegubu, na „pozostałą" sprawność kinematycznie redundantnego manipulatora. Miara ta została wykorzystana jako kryterium i technika obliczania optymalnych konfiguracji, odpornych na uszkodzenia manipulatorów redundantnych. Pokazano przykład robota trójwymiarowego, który po awarii przegubu używał czwartego siłownika, mogącego zagwarantować połowę pierwotnej zręczności.

W tym samym okresie, tj. w latach 1990. podjęto szeroko zakrojone badania nad opracowaniem układów FTC, przeznaczonych do zastosowań w manipulatorach przeznaczonych do badań kosmicznych [12, 13, 14]. Podstawy projektowania manipulatorów tolerujących awarie, zostały przedstawione w pracy [15]. w tej pracy opisano narzędzia modelowania oraz metody oceny niezawodnościowej robotów. Przedstawiono aspekty projektowe wraz z oceną niezawodności robotów. Na podstawie przeprowadzonych analiz, zaproponowano metodykę projektowania manipulatorów, odpornych na uszkodzenia. Szczególną uwagę zwrócono na opracowanie zaleceń do projektowania ramienia o 10. stopniach swobody. Zaproponowano zastosowanie dwóch siłowników na każdym przegubie.

W celu zapewnienia pomyślnej tzn. niezawodnej pracy robotów autonomicznych w odległych lub niebezpiecznych środowiskach, odporność na błędy jest niezbędna. w rozdziale książki [16] zaproponowano różne algorytmy zapewniające odporności na błędy. Podczas ich opracowywania, należy najpierw określić możliwe awarie komponentów robota oraz zdefiniować współzależność tych awarii. Do tego celu bardzo przydatnym narzędziem może być analiza drzew błędów (Fault Tree Analysis). Opracowane za pomocą tej metody struktury drzewiaste, umożliwiają wyznaczenie zarówno schematu blokowego możliwych wystąpień usterek robota, jak i określenie związków przyczynowo-skutkowych pomiędzy uszkodzeniami.

W pracy [17] zastosowano metodę sterowania, w której wykorzystano algorytm odniesienia do modelu oraz regulator proporcjonalnej pochodnej integralnej, w celu zapewnienia odporności na uszkodzenia manipulatorów. Metoda ta została przetestowana symulacyjnie, wykazując skuteczność algorytmu, który umożliwił kontynuację ruchu po założonej trajektorii i osiągnięcie zadanego efektu końcowego.

Już w 1987 roku Selkäinaho i Halme [18] zaproponowali zastosowanie sztucznej inteligencji, czyli systemu eksperckiego działającego w czasie rzeczywistym,

w rozwiązywaniu problemów kontroli błędów. Użyto go w algorytmie nadzorującym wykrywanie i lokalizację uszkodzeń. w przypadku uszkodzenia czujnika, system automatycznie zastępował uszkodzony pomiar uaktualnionym sygnałem wyjściowym z modelu predykcyjnego. Badany system wykorzystywał sztuczną inteligencję. Został przetestowany z powodzeniem w procesie pilotażowym do usuwania uszkodzeń fizycznych.

W pracy [19] zaproponowano zastosowanie inteligentnego sterowania odpornego na błędy do elastycznej komórki montażowej. Sterownik tolerujący błędy natychmiast rozpoznawał błędy i reagował w czasie rzeczywistym na takie sytuacje. w opisywanym rozwiązaniu zastosowano połączenie zaawansowanego autonomicznego systemu nadzoru, z generatorem działań kierowanych przez czujniki. Badania eksperymentalne wykonano z wykorzystaniem mobilnego robota dwuramiennego.

Autorzy pracy [20] opisali sterowanie w czasie rzeczywistym, odporne na błędy, opracowane dla kinematycznie redundantnego manipulatora. Wykorzystali miarę tolerancji błędów do ciągłego podążania po zadanej trajektorii przez efektor końcowy manipulatora. Opracowano algorytm, który został wykorzystany w czasie rzeczywistym do sterowania, dostępnym na rynku manipulatorem o siedmiu stopniach swobody.

Celem stosowania systemów FTC jest zapewnienie niezawodności, która powinna być utrzymana pomimo występowania błędów o dopuszczalnym charakterze. Obecnie trwają intensywne prace nad rozwojem tego typu sterowania w urządzeniach wykorzystywanych w procesie produkcyjnym. Jedną z zalet zastosowania tego typu sterowania w procesie produkcyjnym, jest nie tylko bezpieczeństwo urządzeń krytycznych, ale również redukcja kosztów. Na obniżenie kosztów produkcji ma wpływ ciągłość i niezawodność pracy stosowanych urządzeń. w momencie wystąpienia akceptowalnej przez FTC awarii, nie ma potrzeby zatrzymywania produkcji w celach naprawczych. Skutkuje to zarówno zmniejszeniem kosztów, związanych z koniecznością wezwania serwisu, jak również oszczędnością czasu, wynikającą z zatrzymania procesu produkcyjnego.

Shin i Lee [21] przedstawili odporny na błędy system sterowania, który może być wykorzystany do „przezwyciężenia" awarii siłowników manipulatorów. Sterownik wykorzystywał algorytmy dla sterowania normalnego (bez awarii), wykrywania awarii i sterowania w przypadku awarii, w celu osiągnięcia założonej realizacji zadania.

Systemy sterowania odpornego na uszkodzenia można podzielić na trzy główne kategorie: pasywne FTCS (PFTCS), aktywne FTCS (AFTCS) oraz hybrydowe FTCS

(HFTCS) [22]. Algorytm typu PFTCS charakteryzuje się tym, że działa w trybie offline i jest w stanie dostosować się tylko do usterek zdefiniowanych na etapie projektowania. Pasywne systemy sterujące odporne na błędy, nie wymagają wykrywania i izolowania błędów i są mniej złożone obliczeniowo, niż systemy aktywne. Algorytm AFTCS składa się z mechanizmu rekonfiguracji oraz podsystemu identyfikacji uszkodzeń (FDI). Podsystem FDI dostarcza kontrolerowi informacji o usterce, a ten reaguje poprzez rekonfigurację siebie lub kontrolowanego urządzenia. Podsystem FDI jest kluczowym elementem aktywnego FTC, ponieważ określa usterkę i wysyła informację do kontrolera. Szybka rekonfiguracja kontrolera po otrzymaniu informacji z FDI, pozwala na nazwanie tej metody aktywną. Znane są również hybrydowe regulatory HFTCS, które są połączeniem PFTCS i AFTCS.

2.2 Metody sztucznej inteligencji w sterowaniu odpornym na błędy manipulatorów

Metody sztucznej inteligencji są obecnie silnie rozwiniętym narzędziem, które ma swoje zastosowanie także w systemach do tolerancji błędów, w sterowaniu ramionami robotów. Jedną z metod sztucznej inteligencji wykorzystywaną w układach sterowania odpornych na błędy, są sterowniki z logiką rozmytą, które są w stanie skutecznie radzić sobie z układami nieliniowymi, wykorzystując funkcje przynależności, które oceniają analogowe sygnały wejściowe za pomocą zmiennych logicznych, przyjmujących wartości od 0 do 1. w pracy [23] opisano tzw. adaptacyjne sterowanie rozmyte typu 2 (ang. adaptive fuzzy type-2 backstepping control - AFT2BC). Metoda backsteppingu została również opisana w publikacji [24]. Sterowanie typu AFT2BC przetestowano na modelu ramienia robota PUMA560, wykonanym w środowisku MATLAB, w którym konfiguracja kinematyczna robota może być zmieniana równocześnie z uszkodzeniem osi i zmianą obciążenia ramienia. Zaproponowany algorytm sterowania nie wymagał apriorycznej znajomości modelu dynamicznego robota, dzięki czemu sterownik może działać zarówno w przypadku, gdy uszkodzenie wynika z niepewności modelu, jak i w przypadku zakłóceń zewnętrznych. Na rysunku 1 przedstawiono schemat blokowy zaproponowanej metody sterowania adaptacyjnego. w poszczególnych blokach zamieszczono numery, określające zbiór równań Lyapunova (1) - (6):

$$\dot{V} = e_1\dot{e}_1 = e_1(\dot{q}_d - x_2), \tag{1}$$

gdzie: $\dot{V}$ jest pochodną funkcji Lyapunova [25], e_1 jest błędem śledzenia, q_d jest zadanym położeniem efektora robota a $\dot{q}_d$ jest pochodną tego sygnału, x_2 jest wektorem stanu bieżącego $x_2 = \dot{x}_1$, a $x_1 = [q_1 q_2 q_3]^T$ jest wektorem stanu.

Wektor sterujący osiami manipulatora u określa zadane położenie ramion robota, które składa się z trzech elementów - sygnałów. Pierwszy z nich to u_a, określający adaptacyjne wyrażenie sterowania rozmytego typu 2, które zostało zaprojektowane w celu estymacji idealnego prawa sterowania typu backstepping, opisane następująco

$$u_a = W^T(e_1\dot{e}_1)\Theta, \tag{2}$$

gdzie: W^T reprezentuje średnie funkcje bazowe [23] uzyskane przez system rozmyty typu 2, w którym każda funkcja bazowa jest dana przez średnią odpowiadających jej lewych i prawych funkcji bazowych, Θ oznacza adaptowane parametry wektorowe, wyznaczane ze wzoru

$$\dot{\Theta} = \gamma e_2 W(e_1\dot{e}_1) - \sigma_1\Theta, \Theta(0) = 0 \tag{3}$$

gdzie: $\gamma > 0$, $\sigma_1 > 0$, $e_2 = \dot{q}_d + c_1e_1 - x_2$, c_1 jest dodatnim stałym wektorem.

Druga składowa wektora sterującego u_r jest odpornym wyrażaniem sterującym wprowadzonym w celu redukcji efektów błędu estymacji rozmytej typu 2

$$u_r = \hat{\varepsilon}\, tanh\,(\tfrac{e_2}{\chi}), \tag{4}$$

gdzie:,

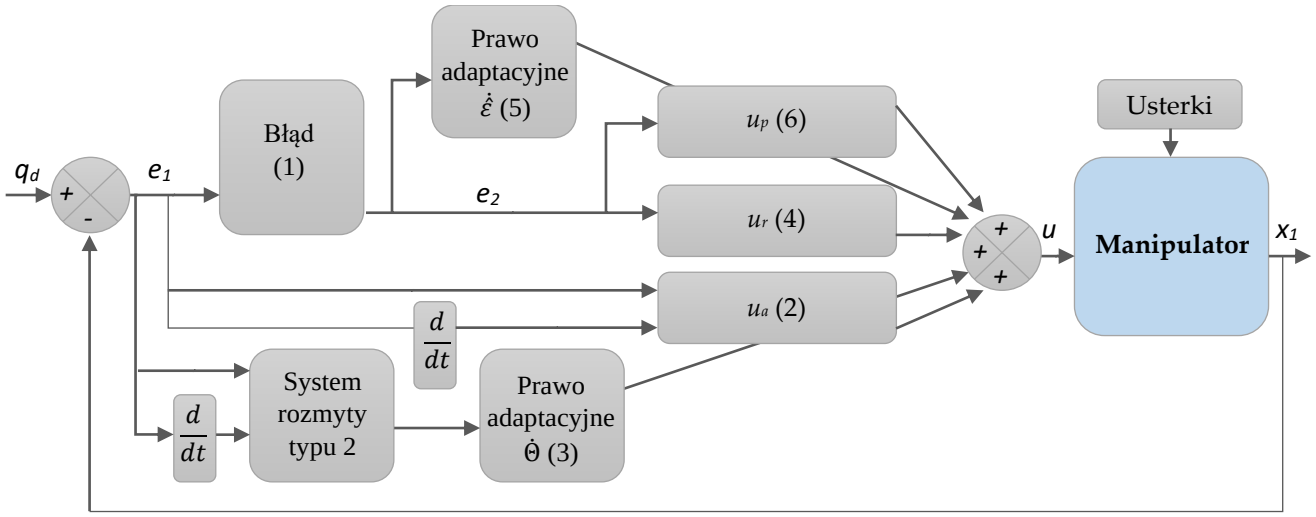
$$\dot{\hat{\varepsilon}} = \eta\hat{e}_2\, tanh\left(\frac{e_2}{\chi}\right) - \sigma_2\hat{\varepsilon}, \tag{5}$$

gdzie: $\eta > 0, \sigma_2 > 0, \chi > 0, \hat{\varepsilon}(0) = 0$

Trzecia składowa opisana jest wzorem

$$u_p = c_2e_2, \tag{6}$$

gdzie: $c_2 > 0$,



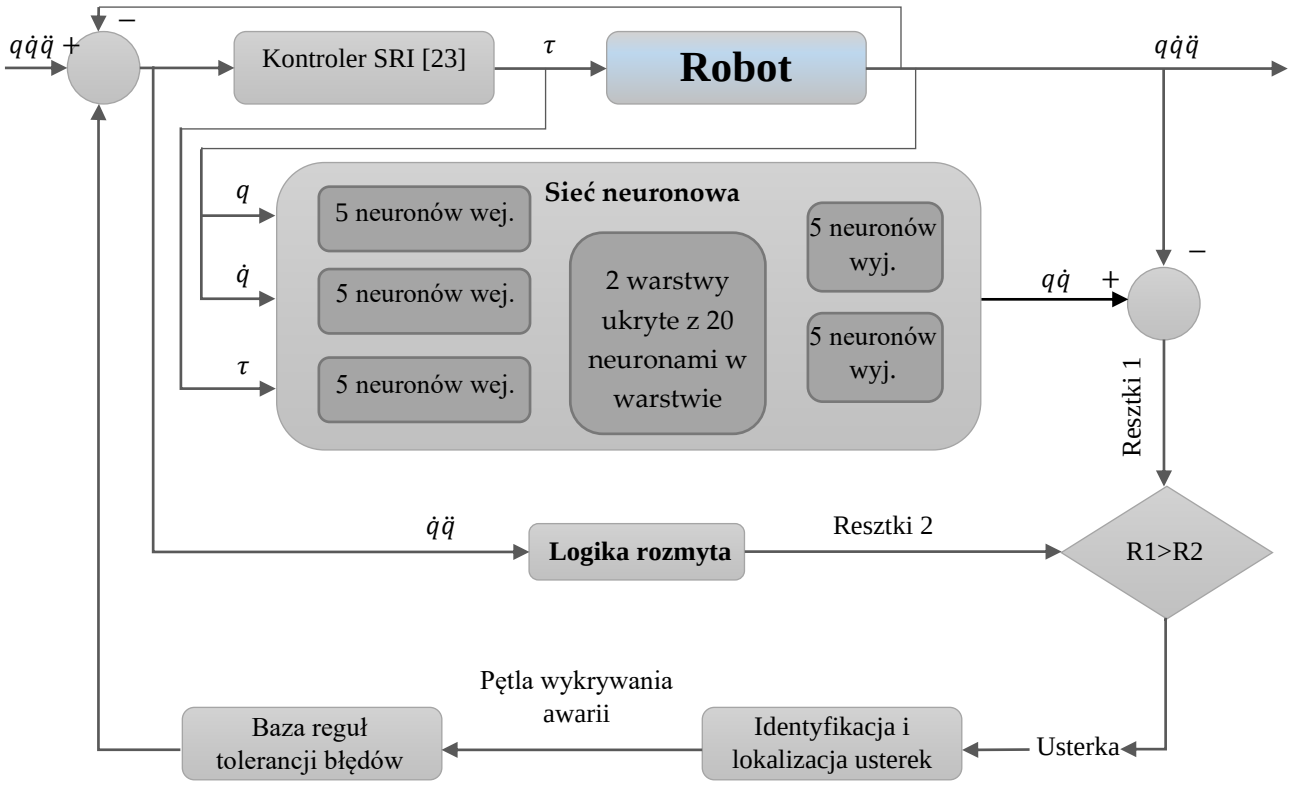
Rysunek 1. Schemat sterowania AFT2BC, gdzie: u – zadane położenie osi robota [23]

Autorzy pracy [26] przedstawili tolerancyjny na uszkodzenia, schemat sterowania w postaci (ang. non-singular terminal synergetic control - NTSC), połączony z (ang.

interval type-2 fuzzy satin bowerbird optimization - IT2FSBO). w pracy opisano również adaptacyjny rozszerzony filtr Kalmana, który wykrywał, identyfikował i izolował usterki siłownika nawet w obecności szumu [27].

W pracy [28] przedstawiono algorytm wykrywania usterek robota z wykorzystaniem funkcji neuro-rozmytych (ang. neuro-fuzzy - NF), który pozwala na sterowanie robotem, pomimo awarii czujników lub siłowników osi. Na rysunku 2 przedstawiono architekturę zaproponowanego w omawianej publikacji, sterownika tolerującego błędy. Architektura detekcji i tolerancji błędów tego rozwiązania, zbudowana jest z wielowarstwowego perceptronu uczonego metodą wstecznej propagacji błędu oraz z bloku logiki rozmytej. Warstwa wejściowa sztucznej sieci neuronowej składa się z 15-tu neuronów, zorganizowanych w trzy grupy. w każdej grupie znajduje się 5 neuronów, po jednym dla każdej osi. Te trzy grupy służą do wstępnej oceny kolejno: położenia, prędkości i przyspieszenia. Sieć posiada dwie grupy neuronów wyjściowych. Każda grupa ma 5 neuronów, po jednym dla każdej osi. Grupy te generują pozycje i prędkości kolejno dla każdej osi. Wynik działania perceptronu wielowarstwowego (ang. multilayer perceptron - MLP) w postaci iloczynu pozycji i prędkości, tj. $q\dot{q}$ jest sumowany z parametrami wyjściowymi robota. Suma ta trafia do bloku wnioskowania wraz z wynikiem działania bloku logiki rozmytej. Blok wnioskowania, otrzymując dane, daje odpowiedź czy i która oś jest uszkodzona. Algorytm został z powodzeniem przetestowany na symulacji robota ER5u w środowisku MATLAB.

W pracy [30] przedstawiono system, w którym za monitorowanie i wykrywanie usterek (FD) odpowiada sztuczna sieć neuronowa w postaci perceptronu wielowarstwowego (MLP). System ten składa się z bloku, w którym MLP podaje informacje o uszkodzeniu urządzenia, natomiast baza reguł logiki rozmytej podejmuje decyzje o rodzaju uszkodzenia i jego lokalizacji. Autor zademonstrował działanie systemu detekcji uszkodzeń, oparte na aplikacji neuro-rozmytej (NF), zastosowanej dla robotycznego ramienia o 5. stopniach swobody, gdy jedna z osi robota ulegnie awarii. Architektura systemu jest podobna do rozwiązania opisanego w [29], jednak w tym rozwiązaniu wnioskowanie przez układ logiki rozmytej, odbywa się na podstawie wyjścia sieci neuronowej.



Rysunek 2. Architektura układu sterowania, wykrywania i usuwania uszkodzeń z jednostką neuro-rozmytą [28]

Autorzy pracy [31] zaproponowali nowe rozwiązanie w postaci samo strojenia rozmytego regulatora proporcjonalno-całkująco-różniczkującego, do szybkiego, terminalowego sterowania w trybie ślizgowym (ang. self-tuning fuzzy proportional–integral–derivative non-singular fast terminal sliding-mode control - SFT-PID-NFTSM), z estymacją opóźnienia czasowego (ang. time delay estimation - TDE). Proponowana metoda stosuje samo strojenie bloku PID regulatora, z wykorzystaniem algorytmów logiki rozmytej. w porównaniu z innymi znanymi metodami opisanymi w literaturze, takimi jak regulator PID lub regulator z blokiem logiki rozmytej, odpowiedzialnym za samo strojenie bloku PID tj. (PID-NFTSM), wyniki pracy [31] pokazują, że kontroler zaproponowany w omawianej pracy był stabilny, miał mniejsze przeregulowanie i poprawiał odpowiedź przejściową. Ponadto integracja TDE zmniejszyła wymaganą moc obliczeniową regulatora, a także pozwoliła wyeliminować wymóg znajomości dokładnej dynamiki układu. Architektura tego systemu została przedstawiona na rysunku 3. Aby w pełni zrozumieć jej działanie , należy pokazać również wzory (7) - (15), według których działał układ:

$$S = e + k_1 e^{[\lambda]} + k_2 \dot{e}^{[\frac{p}{q}]}, \tag{7}$$

gdzie: *S* jest powierzchnią ślizgową NFTSM, *e* jest błędem, $k_1 = diag(k_{11}, k_{12}, \dots, k_{1n}) \in R^{nxn}$ i $k_2 = diag(k_{21}, k_{22}, \dots, k_{2n}) \in R^{nxn}$ są macierzami dodatnio definiowanymi, p i q są dodatnimi liczbami nieparzystymi dobranymi tak, aby spełniały warunki $1 < \frac{p}{q} < 2 \quad and \quad \lambda > p/q$,

$$S_{PID}(t) = K_p s(t) + K_i \int_0^t s(t)dt + K_d \frac{ds(t)}{dt}, \tag{8}$$

gdzie: S_{PID}powierzchnia ślizgowa oparta na PID-NFTSM, K_p, K_i, K_d to wzmocnienia proporcjonalne, całkowe oraz różniczkowe regulatora PID,

$$|\Delta(x_1, x_2, u, \tau_d)| \leq \Delta_0, \tag{9}$$

gdzie: $x_1 = q, x_2 = \dot{q}, u = \tau$, τ_d jest momentem zakłócającym, Δ_0 jest stałą, $\Delta(x_1, x_2, u, \tau_d)$ obejmuje wszystkie efekty niepewności, zakłóceń i usterek,

$$\left|\dot{\Delta}(x_1, x_2, u, \tau_d)\right| \leq \Delta_1, \tag{10}$$

gdzie: Δ_1 jest stałą,

$$\wedge (\dot{e}, \Delta)_{(t)} \cong \wedge (\dot{e}, \Delta)_{(t-L)}, \tag{11}$$

gdzie: $\wedge (\dot{e}, \Delta)_{(t)}$ jest funkcją nieznaną, $(t - L)$ jest opóźnieniem czasowym,

$$u_{TDE_t} \cong \widehat{\wedge} (\dot{e}, \Delta)_{(t)}, \tag{12}$$

gdzie: u_{TDE_t} jest estymacją nieznanej funkcji,

$$\dot{u}_{ar} = \left(\widehat{K} + a\right) sign(S_{PID}), \tag{13}$$

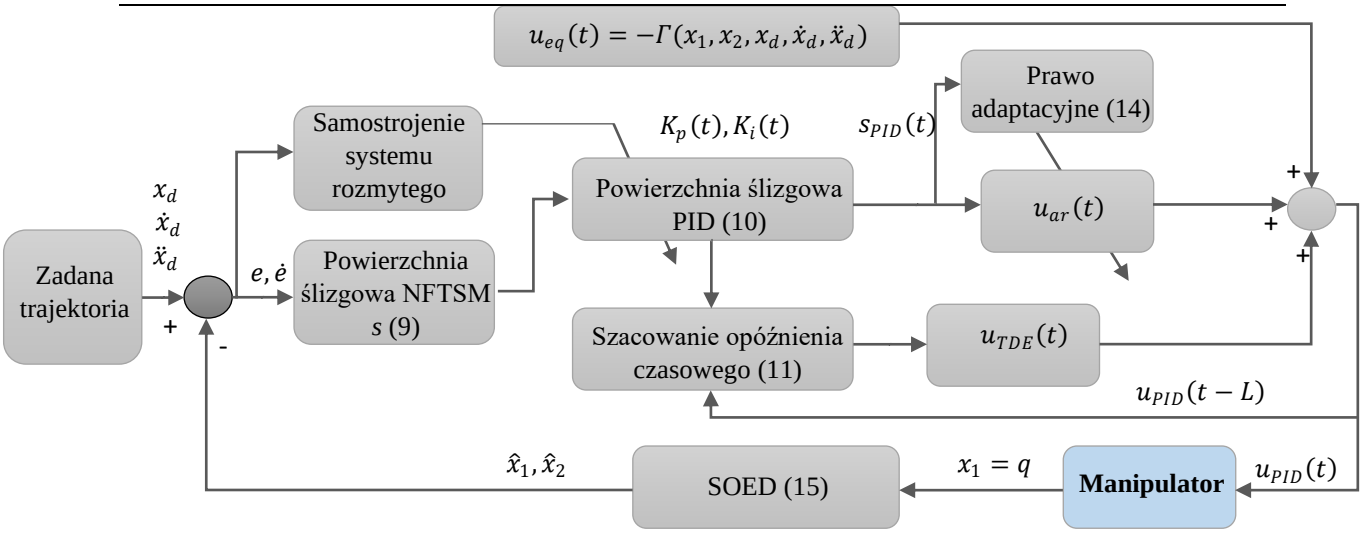
gdzie: $\dot{u}_{ar}$ jest prawem adaptacyjnym, *K* jest nieznaną stałą, $\widehat{K}$ jest estymacją wartości *K*,

$$u_{PID} = -\Omega^+(\dot{e}x_1)(u_{eq} + u_{TDE} + u_{ar}), \tag{14}$$

gdzie: u_{PID} to proponowany kontroler, $-\Omega^+(\dot{e}x_1)$ jest niewiadomą funkcją,

$$V = \frac{1}{2} S_{PID}^T S_{PID} + \frac{1}{2} C \breve{K}^T \breve{K}, \tag{15}$$

gdzie: V jest funkcją Luapunov'a, $\breve{K} = K - \widehat{K}$ to błąd adaptacyjny, C jest wzmocnieniem adaptacyjnym

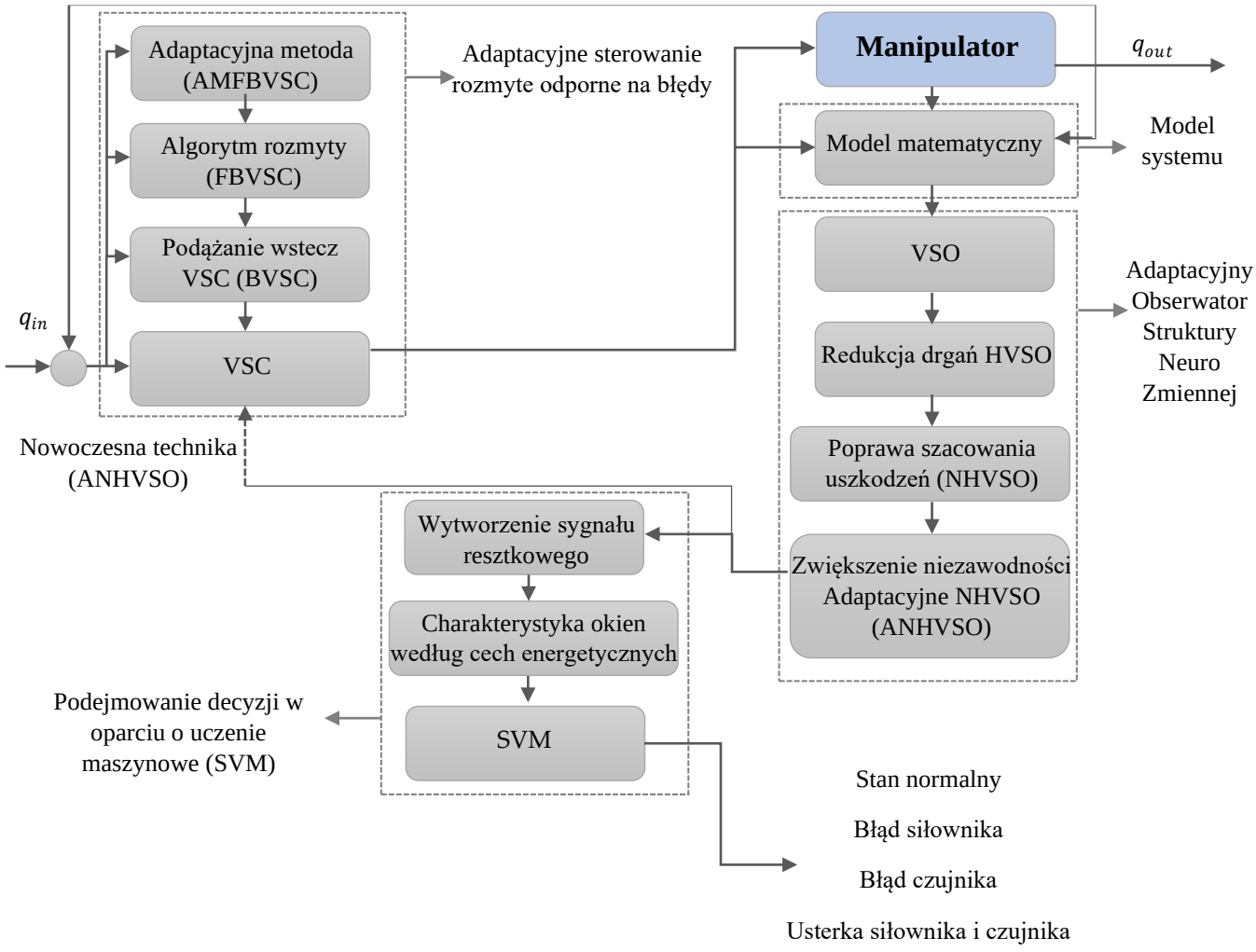


Rysunek 3. (SFT-PID-NFTSM) z (TDE) [31]

W pracy [32] przedstawiono nową metodę diagnozowania uszkodzeń i sterowania odpornego na uszkodzenia manipulatora, wykorzystującą kombinację neuronowego adaptacyjnego obserwatora struktury zmiennej wysokiego rzędu (ang. adaptive high-order variable structure observer - ANHWSO) oraz adaptacyjnego regulatora struktury zmiennej, typu rozmytego cofającego się (ang. adaptive modern fuzzy backstepping variable structure controller - AMFBVSC). Architektura tego układu sterowania została przedstawiona na rysunku 4. w metodzie tej, do detekcji i identyfikacji błędu, zastosowano technikę uczenia maszynowego wektorowego (ang. support vector machine - SVM). Jak napisano, zaproponowana metoda sterowania tj. ANHVSO, znacznie poprawiła wydajność identyfikacji błędów, w porównaniu do układu neuronowego obserwatora zmiennej, struktury wysokiego rzędu (NHVSO) oraz obserwatora zmiennej struktury (ang. variable structure observer - VSO). Badania opisane w [32] są kolejnym przykładem, zastosowania metod sztucznej inteligencji w identyfikacji uszkodzeń. w tym przypadku sieć neuronowa oraz uczenie maszynowe były wykorzystywane w sterowaniu, natomiast logika rozmyta była użyta w strojeniu proponowanej metody.

W pracy [33] Autorzy zaproponowali wykorzystanie podwójnej sieci neuronowej (ang. dual neural network DNN) [34], do diagnostyki i usuwania usterek. Jest to pierwsze znalezione w literaturze opracowanie, które przewidywało możliwość sterowania redundantnym manipulatorem w przypadku awarii więcej niż dwóch osi. w pracy tej przedstawiono realizację zadania kreślenia okręgu, gdy aż 3 z 7 dostępnych osi ulegną awarii w różnych okresach czasu. We wcześniejszych rozwiązaniach, zadanie sterowania w momencie awarii osi polegało na wyłączeniu pojedynczej osi lub pojedynczego czujnika.

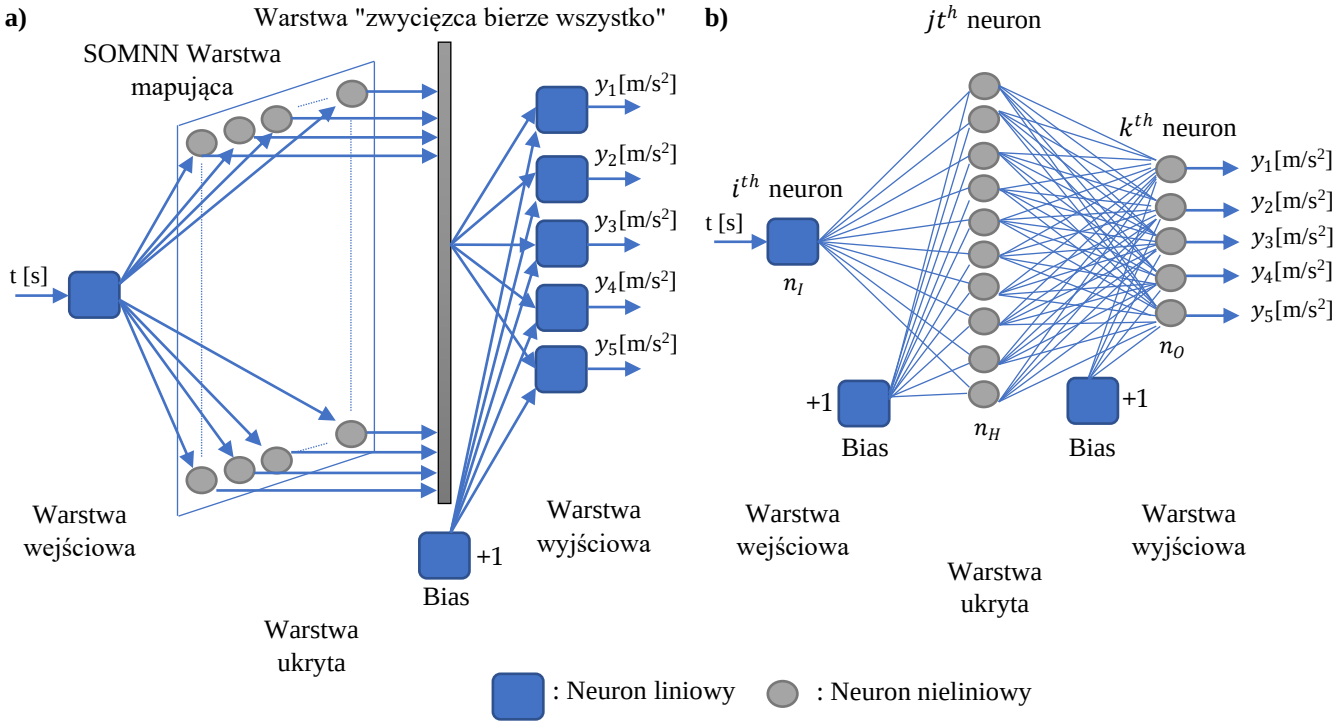
Rozwiązanie to jest jednak dedykowane ściśle określonym trajektoriom ruchu tzn. trajektorią która jest wyznaczona wcześniej np. za pomocą planera ruchu. Algorytm nie generuje trajektorii optymalnej tj. najkrótszej, a jego zadaniem jest tylko podążanie po zdefiniowanej wcześniej trajektorii, która w przedstawionych badaniach miała kształt okręgu.



Rysunek 4. Architektura sterowania z obserwatorem (ANHVSO) i regulatorem (AMFBVSC) [32]

W pracy [35] zaproponowano i porównano ze sobą dwa podejścia do detekcji błędów robota spawalniczego, oparte na predykcji przyspieszeń poszczególnych osi. Pierwsze podejście do predykcji błędów, polegało na wykorzystaniu sieci neuronowej z mapą samoorganizującą się (ang. self-organizing map neural network - SOMNN). Sieć ta została przedstawiona na rysunku 5a. w drugim podejściu, do tego samego celu, wykorzystano sieć neuronową o radialnej funkcji bazowej (ang. radial basis function neural network - RBFNN), którą przedstawiono na rysunku 5b. w pracy wykazano możliwość wykorzystania obu sieci tj. RBFNN i SOMNN, do wstępnego przewidywania

przyspieszenia poszczególnych osi w celu wykrywania usterek robota lub przewidywania pozostałego czasu eksploatacji urządzenia.



Rysunek 5. a) Self organizing map neural network (SOMNN) [35], b) Radial basis function neural network (RBFNN) [35]

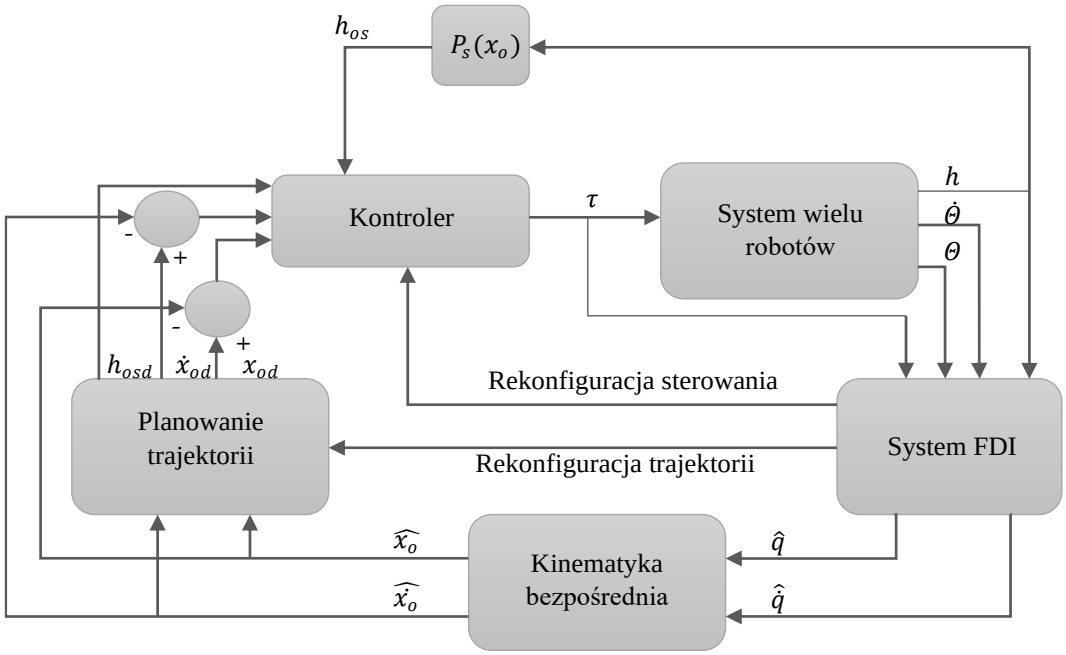
W pracy [36], przedstawiono zdecentralizowane strategie sterowania oparte na sieci neuronowej RBFNN. w tej metodzie sterowania każdy przegub traktowany był jako osobny podsystem. Usterka w przegubie była wykrywana za pomocą obserwatora prędkości. Przedstawione prawo sterowania adaptacyjnego opierało się na informacji z poszczególnych stawów. Może ono być zastosowane do dowolnej budowy ramienia robota.

Autorzy pracy [37] zaproponowali sieć neuronową o radialnej podstawie (RBFNN), stosowaną do estymacji uszkodzeń systemu, w celu uniknięcia błędu diagnostycznego. Zadaniem sieci neuronowej była kompensacja zakłóceń zewnętrznych i awarii siłowników, które w proponowanej metodzie rozpoznawane były poprzez adaptacyjny obserwator zakłóceń.

W pracy [38] zaproponowano adaptacyjny system sterowania z ograniczeniami w czasie (ang. adaptive fixed-time fault-tolerant constraint control - AFTFTCC) do śledzenia trajektorii. System został przetestowany przez Autorów dla dwóch manipulatorów, pracujących razem. Opisana metoda różni się od innych metod tym, że nie potrzebuje takich

komponentów jak estymator stanu czy obserwator błędów. Metoda wykorzystuje RBFNN do określania górnej granicy niepewności.

W pracy [39] przedstawiono układ sterowania odporny na cztery rodzaje błędów: stawu swobodnie wahającego się, stawu zablokowanego, błędnie zmierzoną pozycję stawu oraz niepoprawnie zmierzoną prędkości stawu. Sztuczne sieci neuronowe w postaci MLP i RBFNN były wykorzystywane do wykrywania usterek wolnozmiennych oraz zablokowanych osi robota. Działanie sieci neuronowych w tym rozwiązaniu opierało się na wykrywaniu usterek przez MLP, gdzie każda oś diagnozowana była przez osobny MLP. Informacja w postaci odwzorowanego wektora błędów była następnie przekazywana do sieci RBFNN, która służyła do klasyfikacji informacji o błędach.



Rysunek 6. Architektura odpornego systemu sterowania [39]

Dzięki temu, że proponowany układ sterowania wykrywał błąd, a następnie izolował go i dokonywał rekonfiguracji sterownika, był on, jak stwierdzono w publikacji, w stanie sterować także ramieniem w przypadku wielu usterek. Nie potwierdziły tego jednak testy przedstawione w omawianej pracy. Opisany układ sterowania został przedstawiony na rysunku 6.

2.3 Odporne na błędy sterowanie trybem ślizgowym

Jedną z najpopularniejszych metod stosowanych do sterowania układami nieliniowymi, w tym także ramionami robotycznymi, jest sterowanie w trybie ślizgowym (ang. sliding mode control - SMC). Sterowanie to oparte jest na układach o zmiennej strukturze,

składających się z niezależnych układów o różnych właściwościach i logice przełączania pomiędzy nimi. w SMC stosuje się nieciągły sygnał sterujący, na przykład „+*k*" i „–*k*", który wymusza "ślizganie się" układu wzdłuż sygnału zadanego Gdy układ, a dokładniej jego parametry bądź zmienne stanu, porusza się wzdłuż krzywej lub powierzchni ślizgowej, mówi się, że układ jest sterowany metodą ślizgową. Aktualny przegląd wszystkich dostępnych metod sterowania z wykorzystaniem trybu ślizgowego, przedstawiono w pracy [40]. Najprostsza metoda sterowania w trybie ślizgowym może być opisana wzorem – regułą:

$$u = k * sign(e), \tag{16}$$

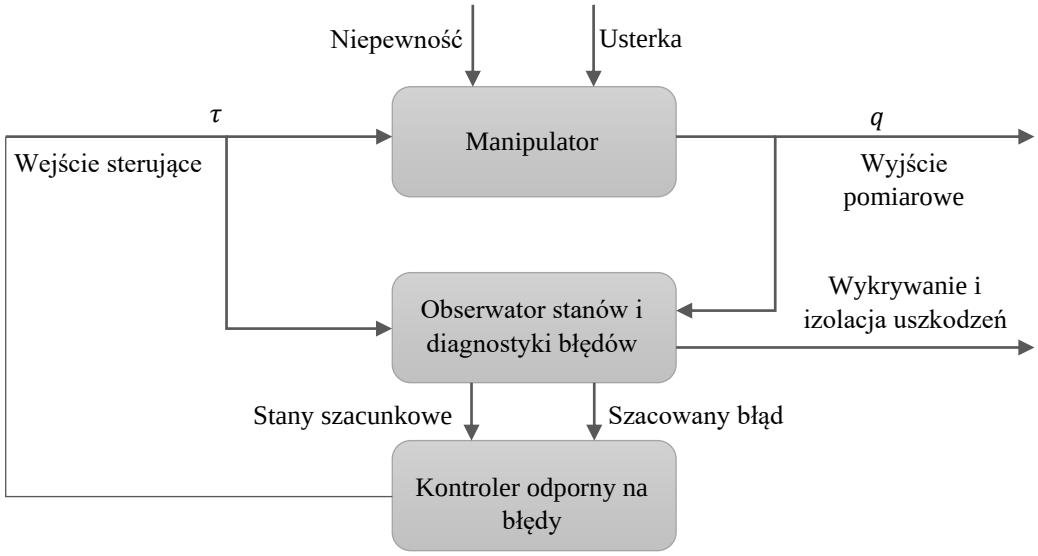
gdzie: u to sygnał obliczony – sterujący obiektem, k to stała, która jest wartością zmiany, e to błąd sygnału sterującego, $sign()$ to funkcja znaku.

Równanie (16) opisuje prostą metodę sterowania w trybie ślizgowym. Zaletą tego typu sterowania jest jego prostota i niewrażliwość na niepewność systemu. Ze względu na fakt, że metoda jest nieciągła, a większość układów sterowania nie jest dyskretna, główną wadą prezentowanej metody są oscylacje sygnału wyjściowego układu, zwane *chatteringiem.* Niemniej jednak metoda ta stała się atrakcyjna ze względu na swoją prostotę, dlatego też projektowane i implementowane są różne układy sterowania w trybie ślizgowym dla różnych obiektów. Metoda SMC, ze względu na swoją odporność na niepewności, jest również często stosowana w tolerujących błędy układach sterowania. w związku z tym, znajduje ona także zastosowanie w manipulatorach, dla których wymagana jest niezawodna praca.

Jedną z odpornych na błędy metod SMC, jest zastosowanie obserwatora trybu przesuwnego (ang. sliding mode observer - SMO) [41], który służy do wykrywania błędów. Zastosowanie nieliniowego obserwatora, umożliwia estymację stanu niemierzalnego i modelowanie niepewności, co pozwala na skonstruowanie algorytmu estymacji błędu.

W pracy [42] zaprezentowano odporny na błędy układ sterowania w postaci obserwatora super-skrętnego trybu przesuwnego trzeciego rzędu (ang. super-twisting third-order sliding-mode - STW-TOSM) oraz regulatora super-skrętnego trybu przesuwnego drugiego rzędu (ang. super-twisting second-order sliding-mode - STW-SOSM). Metoda ta umożliwia estymację błędów i niepewności bez pomiaru prędkości członów robota. Do poprawnej pracy układu potrzebne są jedynie pomiary położenia. w artykule Autorzy przedstawili porównanie następujących systemów: tradycyjnego tzn. regulator momentu obrotowego (ang. computed torque controller - CTC) [43], aktywny kontroler CTC-FTC [44], pasywny

kontroler SM-FTC [45] oraz proponowany w artykule kontroler STW-SOSM-FTC w formie pasywnej i aktywnej. Badania zaproponowanego regulatora wykazały, że jego użycie zapewnia znacznie mniejszy błąd, lepszą stabilność oraz brak wibracji na uzyskanych przebiegach wyjściowych w porównaniu z innymi regulatorami. Schemat blokowy z połączeniami poszczególnych bloków w procesie sterowania przedstawiono na rysunku 7.



Rysunek 7. Schemat blokowy metody przedstawionej przez Mien Van, Pasquale Franciosa i Dariusza Ceglarka [42]

Autorzy pracy [46] przedstawili układ w postaci stało-czasowego obserwatora trybu przesuwnego drugiego rzędu (ang. fixed-time second-order sliding-mode observer - FxTSOSMO) oraz stało-czasowego regulatora trybu przesuwnego (ang. Fixed-time sliding-mode controller - FxTSMC). w tej pracy porównano zaproponowany obserwator (FxTSOSMO) z zaprojektowanym i opisanym w ich wcześniejszej pracy, obserwatorem trybu przesuwnego o stałym czasie (FxSMO). Porównano kontroler obliczający moment obrotowy, nonsingularny szybki terminalny kontroler trybu ślizgowego (NFTSMC) [47] oraz proponowaną metodę (FxTSMC). Stwierdzono, że ten ostatni układ zapewniał większą precyzję estymacji, a błąd proponowanego rozwiązania zbiegał się w ustalonym czasie.

Rozwiązanie stabilizacji i zbieżności systemu zamkniętego z FTC po ustalonym czasie, przedstawiono na rysunku 8 [46]. Jest ono opisane następującymi wzorami (17) - (20):

$$u_{stw} = \mu_1 |e|^{\frac{1}{2}} sign(e) + \xi, \tag{17}$$

gdzie: u_{stw} jest przybliżeniem nieznanego składnika Ω, $\mu_1 > 0$, $e = x_2 - v$, $\xi = k(t)sign(e)$ oraz

$$\dot{v} = \wedge u + f(x_1, x_2) + u_{stw} + k_1[e]^{\gamma_1} + k_2[e]^{\gamma_2}, \quad (18)$$

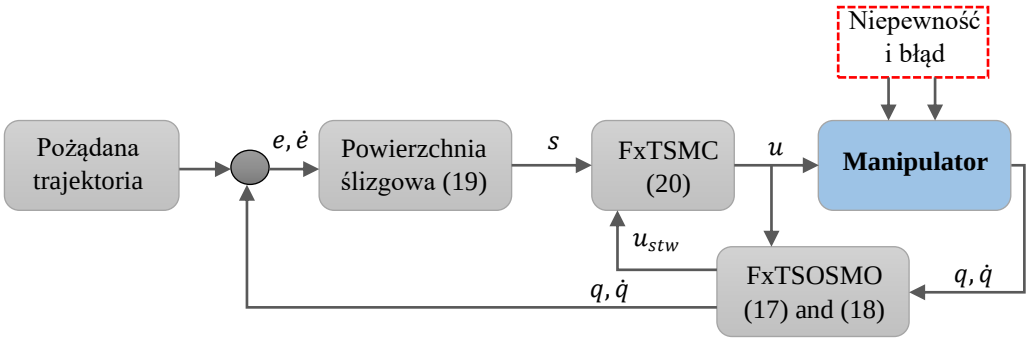
gdzie: $\dot{v}$ jest obserwatorem zaprojektowanym w oparciu o algorytm super skręconego trybu przesuwnego wysokiego rzędu, $\wedge = M^{-1}(q)$, M jest macierzą bezwładności, $k_1 > 0$, $k_2 > 0$, $[e]^{\gamma_1} = [e]^{\gamma_1} sign(e)$, v jest estymacją prędkości x_2, $\gamma_1 = \frac{1}{2}$, $\gamma_2 > 2$,

$$s = e + \frac{1}{k_2^{\gamma}}[\dot{e} + k_1[e]^{\alpha}]^{\frac{1}{\gamma}}, \quad (19)$$

gdzie: s jest proponowaną powierzchnią ślizgową, $\alpha > 1$, $\frac{1}{2} < \gamma < 1$,

$$u = u_0 + u_c + u_s, \quad (20)$$

gdzie: u jest proponowanym kontrolerem, $u_0 = \wedge^{+} (-f(x_1, x_2) - u_{stw} + \ddot{x}_d - k_1|e|^{\alpha-1}\dot{e})$, $\wedge^{+}$ jest pseudo odwrotnością $\wedge$, $u_c = -\wedge^{+} k_2^{\gamma}|T(e,\dot{e})|^{1-\frac{1}{\gamma}}\dot{e}$, $u_s = \wedge^{+} (-(\Gamma + \alpha)sign(s) - \lambda_1[s]^{m_1} - \lambda_2[s]^{m_2})$, Γ jest stałą dodatnią, $0 < m_1 < 1$, $m_2 > 1$



Rysunek 8. Schemat blokowy pętli zamkniętej układu FxTSOSMO-FxTSMC [46]

W pracach [48, 49] Autorzy przedstawili odporny na błędy układ sterowania manipulatorem z elastycznym złączem (ang. single-link flexible joint manipulator - SFJM). Udowodniono, że możliwe jest sterowanie elastycznym złączem zastosowanym w przegubie robota, przy użyciu SMC z trybem ślizgowym SMTO trzeciego rzędu. Przedstawione wyniki pokazały, że SMC jest metodą, która może być stosowana do obiektów charakteryzujących się nieliniowością i niepewnością. Rozwiązanie zostało jednak zwalidowane tylko dla manipulatora jednoosiowego, ale jak stwierdzono z artykule, możliwe jest opracowanie podobnego układu dla manipulatora o większej liczbie stopni swobody.

W pracy [50] zaproponowano nowatorską metodykę adaptacyjnego sterowania trybem ślizgowym manipulatora (ang. adaptive backstepping nonsingular fast terminal sliding mode control - ABNFTSMC). Zaproponowane podejście łączy metodę NFTSMC z mechanizmem projektowania backsteppingowego. Połączenie to skutkuje niskim błędem śledzenia, niskim *chatteringiem* na wyjściu oraz zapewnia szybką odpowiedź. Opisany

układ został porównany z regulatorami CTC, PID, PID-SMC oraz NFTSMC. Wyniki wskazują na poprawę parametrów manipulatora.

W pracy [51] zaproponowano układ składający się z kombinowanego sterowania (ang. combined non-singular fast terminal sliding mode control - NFTSMC) oraz regulatora (ang. high-order sliding mode - HOSM). Przedstawiony układ sterowania wykorzystywał również algorytm oparty na estymacji opóźnienia czasowego (TDE), do określenia błędów. Opisany w pracy wynik wskazuje, że układy Fast Terminal Sliding Mode Control (FTSMC) oraz NFTSMC charakteryzują się szybszą zbieżnością, w porównaniu do metody sterowania NTSMC. Jednak sterowanie FTSMC może napotkać problem osobliwości podczas pracy. Dlatego w pracy wykazano przewagę systemu NFTSMC nad innymi, wymienionymi wyżej. w pracy oceniono również inne metody, zarówno aktywnych układów sterowania tolerujących błędy (AFTCS), jak i pasywnych układów sterowania tolerujących błędy (PFTCS).

W pracy [52] przedstawiono połączenie regulatora w postaci NFTSMC z obserwatorem w postaci (TOSM) (ang. third order sliding mode). Wyniki badań pokazują, że obserwator TOSM może estymować prędkość sygnału wyjściowego układu (np., robota), dzięki czemu układ nie musi mierzyć tej wielkości. Metoda ta została porównana z metodami SMC, NTSMC, NFTSMC. Wykazano, że zaproponowana metoda NFTSMC charakteryzuje się najmniejszym błędem śledzenia trajektorii. Obserwator TOSM okazał się dokładniejszy i miał mniejszy *chattering*, niż obserwator SOSM.

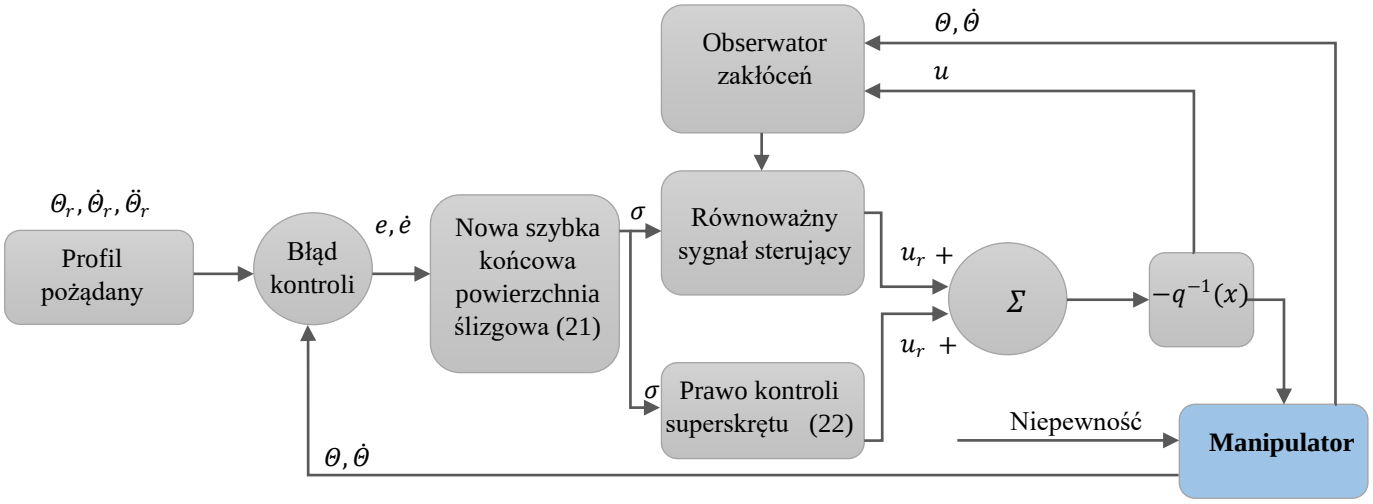
W pracy [53] przedstawiono kombinację szybkiego terminalowego trybu przesuwnego (FTSMC) (STRCL) (ang. super-twisted reaching control law) oraz obserwatora zakłóceń (ang. disturbance observer - DO). Zaproponowana metoda sterowania została porównana z metodami SMC i NFTSMC. Zapewnia ona zbieżność w skończonym czasie i skutecznie redukuje zjawisko *chatteringu*. Dzięki zastosowaniu obserwatora zaburzeń, zmniejsza się też złożoność obliczeń. Metoda DO jest również wykorzystywana do estymacji niepewności dotyczącej dynamiki, zakłóceń zewnętrznych i awarii. Schemat zaproponowanej metody przedstawiono na rysunku 9. w pracy posługiwano się nową szybką terminalną powierzchnią ślizgową, opisaną wzorem:

$$\sigma_i = \dot{e}_i + \frac{2\gamma_1}{1+E^{\mu_1(|e_i|-\Phi)}} e_i + \frac{2\gamma_2}{1+E^{\mu_2(|e_i|-\Phi)}} |e_i|^{\alpha} sign(e_i), \tag{21}$$

gdzie: $\sigma \in R$ to FTSMS, E jest funkcją wykładniczą, e_i reprezentuje błąd położenia, $\dot{e}_i$ reprezentuje błąd prędkości, $\gamma_1, \gamma_2, \mu_1, \mu_2$ są stałymi dodatnimi, $0 < \alpha < 1$, $\Phi = \left(\frac{\gamma_1}{\gamma_2}\right)^{1/(1-\alpha)}$, oraz

$$u_r = \Upsilon_1 |\sigma|^{0.5} sign(\sigma) + \eta, \tag{22}$$

gdzie: u_r to STRCL, $\Upsilon_1 = diag(\Upsilon_{11}, \ldots, \Upsilon_{1n})$.



Rysunek 9. Schemat FTSMS-STRCL-DO [53]

Autorzy pracy [54] zaproponowali nową technikę sterowania w trybie przesuwnym, a mianowicie (AFTC-SSMC) (ang. active fault tolerant control with synchronous sliding mode control). Sterowanie to opiera się na fakcie, że w tradycyjnym regulatorze SMC tylko błąd położenia zbiega się do zera. Natomiast w sterowaniu synchronicznym do zera zbiega się zarówno błąd położenia, jak i kinematyczna zależność między tymi błędami. Metodę porównano ze standardowym aktywnym i pasywnym sterowaniem w trybie przesuwnym, czyli AFTC-SMC i PFTC-SMC. Wykazano, że badana metoda charakteryzowała się lepszą odpornością na błędy oraz prowadziła do lepszych wyników śledzenia trajektorii.

W pracy [55] przedstawiono odporną na błędy strategię sterowania manipulatorem w postaci połączenia metody NFTSMC z zaproponowanym szybkim obserwatorem trybu ślizgowego trzeciego rzędu TOSMO. Takie połączenie jest w stanie obsłużyć nieznane dane wejściowe, dodatkowo redukując zjawisko drgań i poprawiając dokładność śledzenia trajektorii. Zaproponowany szybki TOSMO może estymować zarówno sygnał prędkości jak i dane wejściowe, szybciej niż standardowy TOSMO. Struktura takiego układu jest podobna do układu FTSMS-STRCL-DO [47], ale posiada inny model obserwatora. Struktura blokowa proponowanej metody została przedstawiona na rysunku 10. Rozwiązanie zaproponowane przez Autorów opisane jest następującymi wzorami:

$$\hat{s} = \hat{e} + \int [\beta_1 |e|^{\gamma_1} sign(e) + \beta_2 \hat{e}^{\gamma_2} sign(\hat{e}) + \beta_3 e + \beta_4 e^3]\, dt, \quad (23)$$

gdzie: $\hat{s}$ to NFTS Surface - płaszczyzna, $\beta_1, \beta_2, \beta_3, \beta_4$ to stałe dodatnie, $0 < \gamma_1 < 1$, $\gamma_2 = 2\gamma_1/(1+\gamma_1)$

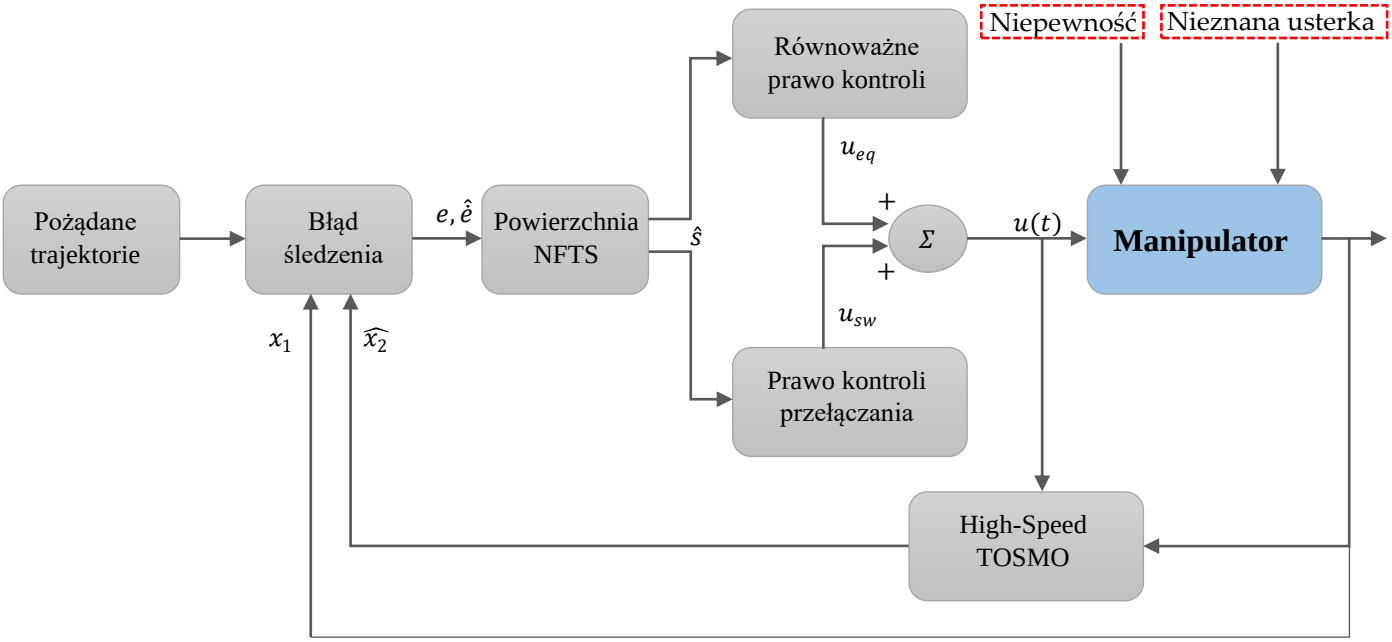
Prawo sterowania opisane jest równaniem

$$u_{eq} = \psi(x) + k_2 |\tilde{x}_1|^{\frac{1}{3}} sign(\tilde{x}_1) + \Gamma(\dot{\hat{x}}_1 - x_2) + \int k_3 sign(\tilde{x}_1) + A - \ddot{x}_d, \quad (24)$$

gdzie: u_{eq} jest równoważnym prawem sterowania, $A = \beta_1 |e|^{\gamma_1} sign(e) + \beta_2 \hat{e}^{\gamma_2} sign(\hat{e}) + \beta_3 e + \beta_4 e^3$, $\psi(x) = M(q)^{-1}[-C(q,\dot{q}) - G(q)]$ jest nominalnym modelem manipulatora, Γ jest stałą dodatnią, k_2, k_3 oznaczają wzmocnienie ślizgowe, $\ddot{x}_d$ jest oczekiwanym przyspieszeniem,

$$u_{sw} = (\Delta_\delta + \mu) sign(\hat{s}), \quad (25)$$

gdzie: u_{sw} jest prawem sterowania przełączaniem, μ jest stałą dodatnią, Δ_δ jest błędem estymacji.



Rysunek 10. Struktura metody sterowania NFTSMC- TOSMO [55]

2.4 Inne metody sterowania odpornego na błędy

Oprócz omówionych wyżej, wiodących odpornych na uszkodzenia metod sterowania manipulatorami, takich jak SMC i metod bazujących na sztucznej inteligencji, w literaturze można znaleźć również inne różne dedykowane metody sterowania manipulatorami. Niektóre z nich zostaną ogólnie opisane poniżej.

W pracy [56] przedstawiono wykorzystanie znanej metody Newtona-Raphsona (NRM) [57], do sterowania manipulatorem w przypadku awarii jednej z osi robota. Autorzy pracy nakleili na ramieniu końcowym robota kod QR. Kod ten był odczytywany przez kamerę umieszczoną u podstawy robota, a następnie pozycja końcówki robota była wprowadzana do oprogramowania sterującego. Dzięki temu algorytm mógł określić moment, w którym doszło do awarii, nie dopasowując pozycji śledzonej przez kamerę do pozycji obliczonej. Następnie system, który w obliczeniach kinematyki odwrotnej, wykorzystuje metodę Newtona-Raphsona do znalezienia dobrego przybliżenia pierwiastka funkcji o wartości rzeczywistej, określił który przegub uległ awarii. Znalezienie uszkodzonej osi odbywało się poprzez porównanie pozycji z kamery, z pozycją obliczoną przez NRM. Jeśli wykryto różnicę, sprawdzono która oś została uszkodzona. w przypadku wykrycia uszkodzenia stawu, traktowano go jako sztywny i przeliczano kinematykę odwrotną robota metodą Newtona-Raphsona. Korzystając z tego rozwiązania, Autorzy pracy udowodnili, że jeśli robot jest w stanie osiągnąć zadaną pozycję pomimo utraty osi, to algorytm pomaga w osiągnięciu celu.

W pracy [58] zaproponowano nową metodę hybrydowego sterowania predykcyjnego odpornego na błędy (ang. hybrid fault tolerant predictive control HFTPC). Metoda ta została przetestowana na manipulatorze hybrydowym będącym połączeniem elementów hyraulicznych i mechanicznych. w pracy porównano proponowaną metodę HFTPC, której jednym z elementów jest model sterowania predykcyjnego (MPC) [59], ze sterowaniem, w którym działa tylko część MPC. Zaprezentowany system był w stanie sterować manipulatorem hybrydowym w momencie wystąpienia awarii.

W pracy [60] przedstawiono silne sterowanie nadążne odporne na błędy (ang. robust fault-tolerant tracking control RFTTC) dla niepewnych nieliniowych układów ze sprzężeniem zwrotnym, wykorzystujące opartą na teorii operatorów solidną faktoryzacje liczb względnie pierwszych (ORRCF) (ang. operator theory-based robust right coprime factorization). Wykazano, że zarówno RFTTC oparte na modelu wewnętrznym jak i RFTTC oparte na kompensacji operatorowej, są skuteczne w radzeniu sobie z zakłóceniami i błędnymi sygnałami. Zaproponowany system został przetestowany w zadaniu śledzenia trajektorii, przez 2-osiowego robota.

W pracy [61] przedstawiono adaptacyjny system sterowania odporny na błędy, w przypadku wystąpienia nieznanych usterek siłowników. Szczegółowa informacja o usterce nie jest wymagana do działania algorytmu. Zaproponowana metoda pomija

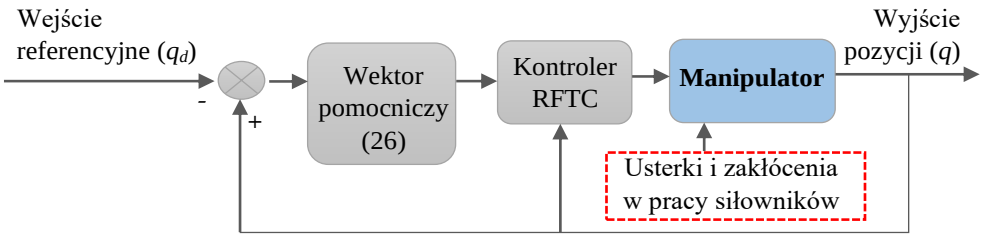
skomplikowane przekształcenia logarytmiczne, w strukturze wstępnej kontroli wydajności (PPC) (ang. pre-scribed performance control) [62], dzięki czemu sterownik jest wydajny i prosty w implementacji.

W pracy [63] zaproponowano wytrzymały układ sterowania odporny na błędy w czasie stałym (RFTFTC) (ang. robust fixed time fault tolerant control system), wprowadzając dodatkowo wektor pomocniczy opisany wzorem:

$$\chi = \dot{e} + \lambda e, \tag{26}$$

gdzie: $\lambda \in R^+$, $e = q - q_d$ to błąd pozycji, $\dot{e} = \dot{q} - \dot{q}_d$ to błąd prędkości

Schemat proponowanego systemu przedstawiono na rysunku 11.



Rysunek 11. RFTC z wektorem pomocniczym [63]

W pracach [64] i [65] przedstawiono FTCS dla manipulatorów, oparty na kontrolerze aktywnego wnioskowania (AIC) (ang. active inference controller), który wykorzystuje sensoryczny błąd predykcji energii swobodnej, do generacji reszt i progów dla FDI. Nie wymaga on dodatkowych kontrolerów do usuwania błędów. w pracy [65] udoskonalono AIC i wprowadzono tzw. bezstronny AIC (u-AIC), w celu zmniejszenia prawdopodobieństwa wystąpienia fałszywych wyników i umożliwienia łatwego zdefiniowania probabilistycznie odpornych progów wykrywania uszkodzeń.

W pracy [49] przedstawiono sterowanie odporne na uszkodzenia z wykorzystaniem regulatora proporcjonalno-integralno-różniczkującego (PID). Oprócz regulatora PID układ składa się on z obserwatora Luenbergera [66], który jest w stanie oszacować błędy analizy FDI. Na podstawie informacji dostarczonej przez FDI, wielkość błędu jest oceniana za pomocą dekompozycji wartości pojedynczych (SVD) (ang. singular value decomposition) [67]. System został przetestowany na jednolinkowym elastycznym manipulatorze.

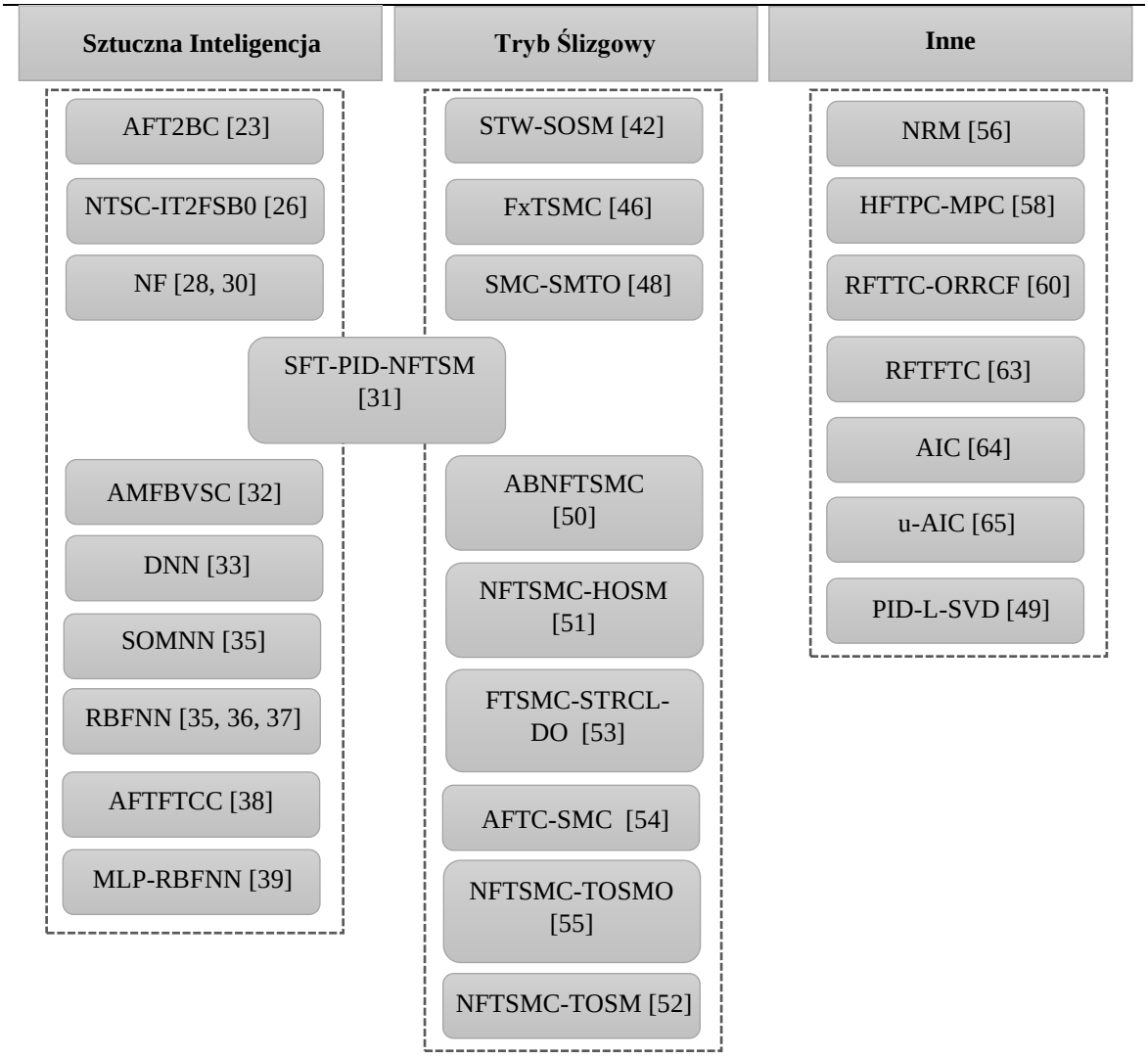
W pracach [68, 69] zaprezentowano nowe rozwiązanie w zakresie sterowania odpornego na błędy, polegające na zakłócaniu toru komunikacyjnego, przez który przesyłane są informacje, pomiędzy elementami wykonawczymi robota a sterownikiem. w pracy zaproponowano rekonstrukcję danych uszkodzonych podczas transmisji z wykorzystaniem kodu splotowego [70]. Transmisja została przeprowadzona z wykorzystaniem protokołu Controller Area Network (CAN) [71].

W pracy [72] przedstawiono aż 4 techniki postępowania w przypadku wystąpienia kontrolowanych oscylacji i uszkodzeń osi. Techniki te nazwano: sinh-cosh, kompensacja neuronowa, PID gain scheduling oraz sinh-cosh gain scheduling. Wszystkie one zostały przetestowane na manipulatorze, w którym awaria została wykryta poprzez pomiar prądów pobieranych przez napędy osi robota. w momencie awarii osi prąd chwilowo spadał prawie do zera. Po wykryciu tego, oś była traktowana jako pasywna, czyli nie biorąca udziału w sterowaniu ruchu. Porównano ze sobą pracę wszystkich 4. metod. Do tego zastosowano porównanie błędu śledzenia trajektorii. Najlepsze wyniki uzyskano dla kompensacji neuronowej, natomiast najgorsze wyniki wykazała metoda szeregowania wzmocnienia sinh-cosh.

2.5 Podsumowanie przeglądu literatury

Przegląd literatury wykonano stosując głównie wyszukiwarkę internetową Google Scholar. Jako słowa kluczowe stosowano: “FTC in manipulator control”, “neural network fault tolerant controller for manipulator”, “artificial intelligence FTC for manipulator”. Znaleziono 55130 pozycje literaturowe. Zapoznano się i opisano publikacje pochodzące głównie z ostatnich pięciu lat. w celu zobrazowania rezultatów przeglądu, znalezione metody sterowania typu FTC, zestawiono na rysunku 12. Wiodącymi w nich metodami i rozwiązaniami są te bazujące na sztucznej inteligencji (ang. Artificial Intelligence – AI) oraz sterowaniu ślizgowym (ang. Sliding Mode Control – SMC). Opracowano również system sterowania odpornego na błędy, który jest połączeniem AI i SMC, a mianowicie SFT-PID-NFTSM [25]. Inne prezentowane metody oparte są na regulatorze PID [49], NRM [56], MPC [58], AIC [64, 64] lub na propozycjach, które zajmują się zakłóceniami występującymi w protokole komunikacyjnym CAN [68, 69].

Wykonany przegląd literatury pokazał, że dotychczas nikt nie opracował algorytmu, który pozwoliłby na sterowanie manipulatorem wykonującym zadanie „pick and place” w przypadku awarii osi, z jednoczesnym omijaniem przeszkód, znajdujących się w przestrzeni roboczej robota. z tego względu uznano za celowe podjęcie tej tematyki w niniejszej pracy.



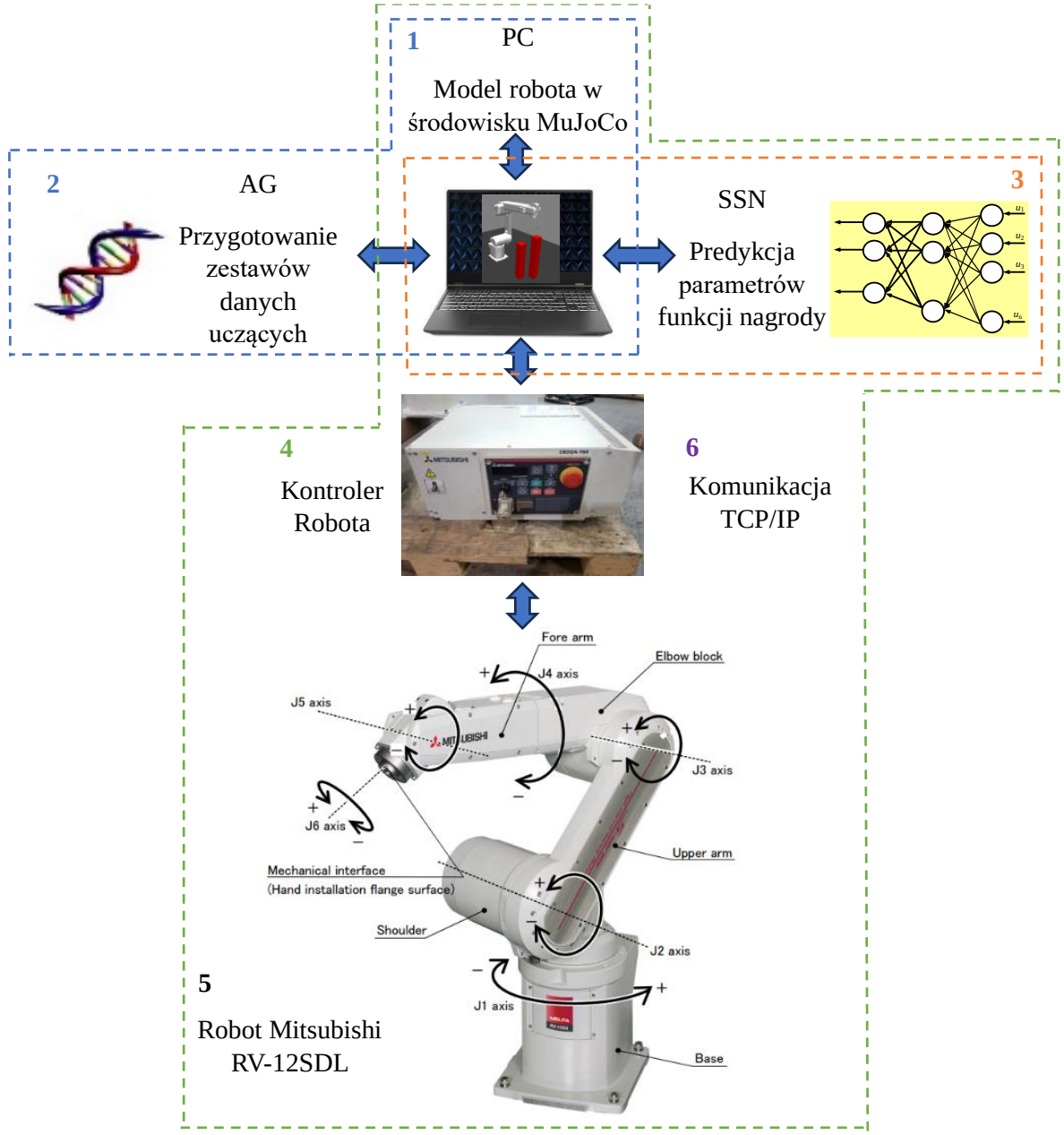
Rysunek 12. Zestawienie stosowanych w manipulatorach metod sterowania odpornego na uszkodzenia

Autorzy prac omawianych w zamieszonym wyżej przeglądzie, zajmowali się sterowaniem odpornym na awarie zarówno jednej, jak i kilku osi. Ponadto poruszali aspekty tolerowania błędnych odczytów czujników położenia oraz zaproponowali algorytmy postępowania, w przypadku zakłóceń występujących na linii komunikacyjnej, pomiędzy sterownikiem a elementami wykonawczymi robota. z przeglądu literatury wynika, że tematyka FTCS w zastosowaniu do manipulatorów, jest obecnie mocno rozwinięta. Zastosowanie systemów FTCS, pozwala na ciągłe utrzymanie produkcji, poprzez uniknięcie przestojów spowodowanych awarią poszczególnych elementów ramienia robota. Nie znaleziono jednak żadnej pracy, w której opisano algorytm mogący realizować zadania FTC i jednocześnie reagować na otaczające go środowisko, a w szczególności omijać przeszkody, znajdujące się w polu pracy robota. w związku z tym, w niniejszej pracy postanowiono rozwiązać ten problem.

3 Opis zaproponowanego rozwiązania oraz cele i teza pracy

3.1 Ogólny schemat działania systemu

W celu nisko kosztowego, bezpiecznego i wygodnego opracowania algorytmu sterowania robotem, zbudowano środowisko symulacyjne, aby przeprowadzić w nim pierwsze badania. Zbudowano również stanowisko badawcze, które graficzne przedstawiono na rys. 13.



Rysunek 13. Ogólny schemat systemu do badań sterowania robotem metodą FTC; moduły wykorzystywane w sterowaniu: 1 - komputer PC, 2 - algorytm genetyczny, 3 - sztuczna sieć neuronowa (SSN), 4 - sterownik robota, 5 - robot przemysłowy, 6 – moduł komunikacji TCP/IP

Przedstawia on, wykorzystane w badaniach różne moduły, użyte w poszczególnych procesach badawczych i kontrolnych, wykonanych w ramach niniejszej pracy. Modułami sprzętowymi są: komputer osobisty klasy PC (1), sterownik robota (4) oraz sześcioosiowy robot przemysłowy Mitsubishi RV-12SDL (5). Parametry komputera PC były następujące: procesor Intel Core i7-9750H, 32 GB RAM oraz karta graficzna NVIDIA GeForce GTX 1660Ti. Zaimplementowano na nim moduły oprogramowania do sterowania i symulacji. Zaproponowano teoretyczny model robota, który został zaimplementowany na komputerze PC z wykorzystaniem tzw., „silnika fizyki" w programie MuJoCo [73]. z jego wykorzystaniem opracowano trójwymiarowy, graficzny model symulacyjny robota. Umożliwił on dokładną i szybką symulację ruchu wszystkich osi robota oraz punktu środkowego narzędzia TCP. Do zadań sterowania zastosowano następujące moduły oprogramowania: sztuczną sieć neuronową – do generowania parametrów funkcji nagrody (3), algorytm genetyczny – do generowania danych uczących SSN (4) oraz moduł do komunikacji TCP/IP (6) pomiędzy komputerem a sterownikiem rzeczywistego robota.

W niniejszej pracy podjęto problem opracowania odpornego na błędy sterowania, sześcioosiowym robotem przegubowym, w przypadku awarii jednej lub dwóch osi. w przedstawionych poniżej rozważaniach, tj. w zaproponowanym algorytmie sterowania przyjęto następujące założenia:

- robot pracuje w trybie „od punktu do punktu" (ang. „point to point" – PTP), a jego punktem początkowym jest P_s (x_s, y_s, z_s), który jest również punktem zakończenia ruchu. Punktami docelowymi są: P_{t1} (x_{t1}, y_{t1}, z_{t1}) i P_{t2} (x_{t2}, y_{t2}, z_{t2}),
- robot jest testowany w zadaniu „złap i przenieś" (ang. „pick-and-place") z wycofaniem robota do początkowej pozycji,
- współrzędne punktu startowego i punktów docelowych są znane i podawane do kontrolera na początku każdego zadania,
- w przestrzeni roboczej robota, również na jego drodze ruchu, może znajdować się jedna lub dwie nieruchome przeszkody,
- przeszkody mają kształt prostopadłościanów, o podstawie kwadratu i o znanych wymiarach, tj. długości boku 100 mm i wysokościach 600 mm i 700 mm,
- jeśli przeszkoda albo dwie przeszkody są obecne (wykryte), to ich lokalizacja na podłożu tj. płaszczyźnie X-Y w przestrzeni roboczej robota jest znana, przed rozpoczęciem działania algorytmu i ruchu robota,

- uszkodzenia osi mogą wystąpić w dowolnej chwili, także przed rozpoczęciem i w trakcie ruchu robota,
- może dojść do uszkodzenia maksymalnie dwóch osi robota, a uszkodzone osie są blokowane w pozycji, w której wykryto uszkodzenie; współrzędne tych pozycji są zapamiętywane i nie są zmieniane podczas ruchu,
- algorytm sterowania działa krokowo, generując w każdym kroku nowe współrzędne punktu na trajektorii, prowadzącej do punktu docelowego, przy jednoczesnym omijaniu przeszkód,
- w każdym kroku, każda z aktywnych osi robota może wykonać krok kątowy tj. obrót o stały kąt w lewo albo w prawo,
- dla każdej osi aktywnej (sprawnej) możliwe jest wykonanie przemieszczenia obrotowego: szybkiego (normalnego) o kąt α_s oraz wolnego o kąt β_s, który jest wykonywany podczas dokładnego pozycjonowania, czyli zbliżania się do punktu docelowego,
- wartości tych kątów są ustawiane dla każdej osi, przez użytkownika na początku pracy algorytmu i wynoszą: dla szybkiego przesunięcia α_s = 0,5° lub 1°, a dla wolnego β_s = 0,05 lub 0,2°; w ogólności wartości te mogą być inne – dostosowane do sytuacji, robota lub wykonywanego zadania,
- szybkie albo wolne przemieszczenia obrotowe wszystkich aktywnych osi, określają współrzędne przemieszczenia punktu środkowego narzędzia robota (TCP) w każdym kroku,
- w trakcie wyznaczania kolejnego kroku, używany jest model symulacyjny robota do określania wszystkich możliwych przemieszczeń kątowych robota, spośród których wybierany jest najlepszy ruch, z punktu widzenia funkcji nagrody,
- w algorytmie rozważane są wszystkie możliwe przemieszczenia obrotowe wszystkich aktywnych osi i dla każdego możliwego do wykonania w danym kroku przemieszczenia, obliczana jest wartość nagrody,
- zbiór przemieszczeń wszystkich aktywnych osi, które otrzyma najlepszą nagrodę jest wybierany do wykonania rzeczywistego ruchu;
- wybrane jako najlepsze w danym kroku przemieszczenia kątowe każdej osi aktywnej, tj. kąty ruchu do wykonania są wysyłane do sterownika robota,
- prędkość robota pomiędzy punktami P_s i P_{t1}, P_{t1} i P_{t2} oraz pomiędzy punktami P_{t2} i P_s jest stała.

W ramach niniejszej pracy podjęto badania algorytm sterowania robotami przegubowymi, które miały wszystkie osie obrotowe. Za **cel algorytmu postawiono znalezienie najkrótszej drogi do punktu końcowego, omijając przeszkody, przy wykorzystaniu dostępnych (nie uszkodzonych) osi.** Każdorazowo przed rozpoczęciem ruchu robota, określana (wykrywana) jest obecność jednej lub dwóch przeszkód i jeśli są one obecne, to ustalane (w badaniach, wprowadzane przez operatora) jest ich położenie. w badaniach zarówno chwila czasowa wystąpienia awarii, jak i numer uszkodzonej osi są generowane losowo. Po rozpoznaniu przez system sterowania, że uszkodzeniu uległa któraś z osi, uruchamiany zostaje algorytm FTC, który rozpoczyna poszukiwanie nowej trajektorii ruchu, w celu generowania kolejnych, krokowych przemieszczeń osi robota, prowadzących jego tzw. końcówkę tj. jego TCP do punktu końcowego. w algorytmie przyjęto, że uszkodzone osie robota są zatrzymane. Wstępnie zaproponowany algorytm działa w następujących krokach:

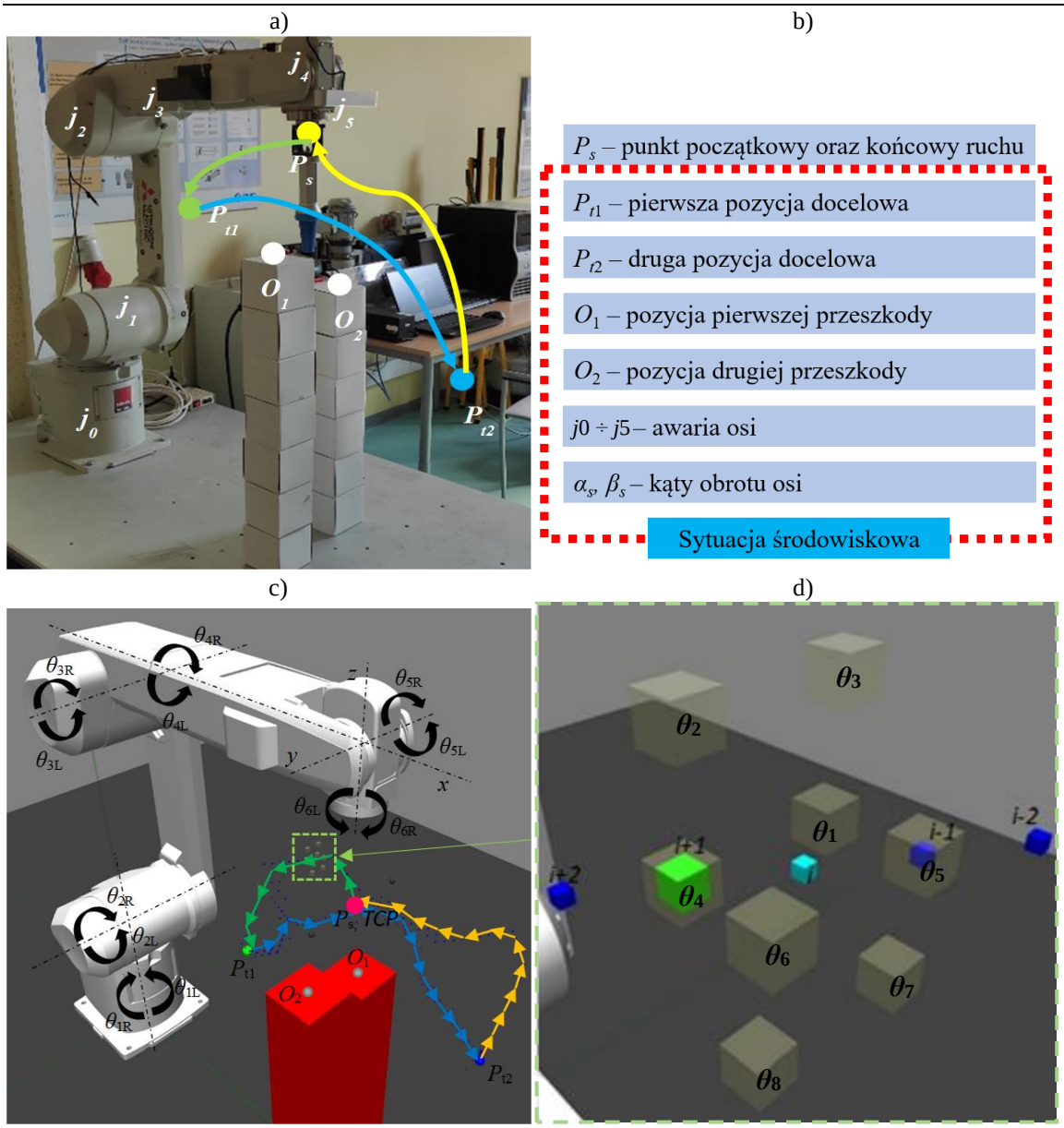
1. Sprawdzenie które osie są sprawne a które nie i odpowiednie do tego, ustawienie wartości parametrów od j_0 do j_5.
2. Wygenerowanie zbioru $\boldsymbol{\theta_i}$ możliwych do wykonania przemieszczeń kątowych w kroku *i*-tym dla wszystkich aktywnych *n* osi robota. Zbiór ten zawiera wszystkie możliwe kombinacje przemieszczeń kątowych osi aktywnych i pozycji TCP, które robot może wykonać ze swojej aktualnej pozycji, w danym kroku. Definicja zbioru $\boldsymbol{\theta_i}$ jest następująca:

$$\boldsymbol{\theta_i} = \{(\theta_{1Li}, \theta_{2Li}, \ldots \theta_{nLi}); \ldots; (\theta_{1Ri}, \theta_{2Ri}, \ldots \theta_{nRi})\}, \tag{27}$$

   gdzie: θ_{jL} oraz θ_{jR} są przesunięciami kątowymi *j*-tej osi, odpowiednio w kierunku lewym i prawym (rys. 14). Przykładowo element zbioru (θ_{1L},θ_{2L},θ_{3R}) oznacza: obrót pierwszej osi w lewo o kąt θ_{1L}, drugiej w lewo o kąt θ_{2L} i trzeciej osi w prawo o kąt θ_{3R}.

3. Wykorzystanie modelu symulacyjnego robota do obliczenia pozycji wszystkich ramion robota i jego TCP, po wykonaniu przemieszczeń podanych w zbiorze $\boldsymbol{\theta_i}$.
4. Obliczanie wartości nagród dla każdego przemieszczenia określonego w zbiorze $\boldsymbol{\theta_i}$ i wybór najlepszego ruchu. Przesłanie wybranej pozycji do robota.
5. Zatrzymanie, jeśli pozycja docelowa została osiągnięta w przeciwnym razie przejście do punktu 1 algorytmu.

Na rysunku 14a pokazano zdjęcie rzeczywistego robota na stanowisku badawczym. Zaznaczono punkty na trajektorii, oznaczenia poszczególnych ramion robota oraz przeszkód. Na rysunku 14c przedstawiono model robota z zaznaczonymi kątami obrotu poszczególnych osi. Na rysunku 14d pokazano graficzne położenie poszczególnych punktów w przestrzeni, uwzględnianych przez algorytm. Kolorem jasnoniebieskim oznaczono położenie TCP robota w kroku i. Kolejne, poprzednie przemieszczenia TCP robota oznaczono kolorem ciemnoniebieskim i są to: i – 2, i – 1. Prostopadłościany szaro-żółte oznaczone przez θ_1 do θ_8, oznaczają wszystkie punkty, do których można przesunąć robota, w następnym kroku. Dla trzech osi aktywnych możliwe jest przesunięcie robota do 2^3, czyli 8-miu punktów. Wśród nich jest także punkt θ_5, w którym TCP był w kroku i – 1. Kolorem jasnozielonym oznaczono punkt θ_4, który algorytm wybrał jako najlepszy na podstawie wartości nagrody, do którego przemieści się TCP robota w kroku i + 1. Strzałki na rys. 14c pokazują poszczególne przemieszczenia (kroki) TCP dla aktywnych trzech pierwszych osi robota.



Rysunek 14. Ilustracja sterowania robotem trzy-osiowym: a) widok robota na stanowisku badawczym, z zaznaczeniem ramion robota, punktów na trajektorii oraz trajektorii ruchu; b) oznaczenia punktów i osi; c) model robota z zaznaczoną trajektorią ruchu oraz oznaczeniami kątów obrotu dla poszczególnych osi; d) powiększenie obszaru pokazującego możliwe do wykonania przemieszczenia TCP w wybranym kroku robota dla aktywnych trzech osi

3.2 Cele i teza pracy

Na podstawie przeglądu literatury przedstawionego w rozdziale 2 można stwierdzić, że brak jest skutecznych algorytmów sterowania robotem, w przypadku uszkodzenia jednej albo dwóch osi, które są w stanie omijać do dwóch przeszkód znajdujących się na jego trajektorii ruchu. Zasadniczym celem pracy jest **opracowanie algorytmu sterowania opartego o model symulacyjny robota, algorytm genetyczny, funkcję nagrody oraz sztuczne sieci neuronowe, który pozwala na pracę robota, mimo awarii jego jednej albo dwóch osi, z jednoczesnym omijaniem przeszkód**.

W celu zrealizowania tego głównego celu badawczego pracy, postanowiono wyznaczyć i uzyskać najpierw cele cząstkowe. Założono następujące cele cząstkowe:

1. Opracowanie środowiska symulacyjnego w postaci modeli: kinematycznego oraz momentowego robota wraz z uwzględnieniem przeszkód, występujących w jego przestrzeni roboczej.
2. Opracowanie algorytmu pozwalającego na sterowanie robotem pomimo awarii jednej albo dwóch osi, z jednoczesnym omijaniem przeszkód.
3. Opracowanie i badania algorytmu genetycznego do optymalizacji parametrów funkcji nagrody, wykorzystywanej do oceny możliwych do wykonania akcji, w danym kroku symulacji.
4. Opracowanie i nauczenie (za pomocą zbioru parametrów quasi-optymalnych, wygenerowanych przez algorytm genetyczny) sztucznej sieci neuronowej, do estymacji parametrów funkcji nagrody dla aktualnej sytuacji środowiskowej.
5. Badania symulacyjne oraz doświadczalne, weryfikujące pracę poszczególnych modułów rozwiązania.

Przyjęto, że badania przeprowadzone w każdym kroku, pozwolą na uzyskanie zasadniczego celu badawczego pracy. Na podstawie przedstawionych celów pracy, sformułowano następującą tezę pracy:

algorytm działający w oparciu o model symulacyjny robota, algorytm genetyczny, sztuczną sieć neuronową oraz funkcję nagrody, pozwala na sterowanie robotem przemysłowym, które jest odporne na awarię jego do dwóch osi oraz zapewnia omijanie przeszkód.

4 Algorytm odporny na uszkodzenie osi robota

4.1 Wprowadzenie

W celu zwiększenia niezawodności pracy robota opracowano algorytm odporny na wystąpienie awarii jednej albo dwóch dowolnych osi. Założono, że po wykryciu awarii osi, układ sterowania będzie sterować robotem w taki sposób, aby wykorzystując tylko sprawne osie, przesunąć TCP robota do punktu docelowego, poruszając się po nowej trajektorii ruchu, dostępnej przy braku jednej albo dwóch osi. Opracowany algorytm powinien również omijać maksymalnie dwie stacjonarne przeszkody, występujące w przestrzeni roboczej, w tym na trajektorii ruchu. Przyjęto, że będzie on działał krokowo w trybie on-line i w każdym kroku wyznaczy najpierw wszystkie możliwe ruchy robota, spośród których, do wykonania wybierze najlepszy. Do tego wyboru będzie stosowana funkcja nagrody.

Zaproponowany system sterowania składa się z trzech głównych bloków: modelu symulacyjnego robota, algorytmu genetycznego oraz sztucznej sieci neuronowej. Wykorzystuje on funkcję nagrody, do wyboru najlepszej akcji tj. ruchu poszczególnych osi robota. Może on funkcjonować w dwóch trybach:

1. Wykonywane zadanie jest znane.

   Ten tryb pracy jest używany, jeśli robot wykonuje przemieszczenie, o parametrach określonych przez „sytuację środowiskową", która była wykorzystywana wcześniej przez algorytm genetyczny, który na etapie wstępnym wygenerował dla niej optymalne wartości parametrów funkcji nagrody i zapisał je w tabeli. w tym przypadku, w każdym kroku sterowania robotem algorytm oblicza wartość nagrody za pomocą funkcji, której parametry zostały wcześniej stabelaryzowane, za pomocą algorytmu genetycznego.

2. Wykonywane zadanie nie jest znane.

   W drugim trybie algorytm wykorzystuje SSN do generowania parametrów funkcji nagrody, która była nauczona wcześniej z użyciem zbioru danych uczących, przygotowanych przez AG. Ten tryb pracy jest stosowany, gdy aktualna „sytuacja środowiskowa" jest inna od wykorzystywanych przez algorytm genetyczny, do wyznaczenia optymalnych wartości parametrów funkcji nagrody. Sztuczna sieć neuronowa działa wtedy jako układ aproksymujący parametry funkcji nagrody.

W obu przypadkach algorytm bazuje na modelu robota, zaimplementowanym w środowisku symulacyjnym MuJoCo oraz na funkcji nagrody, zgodnie z którą wybierana jest akcja optymalna, w danym kroku symulacji. Pierwszy tryb pracy był wykorzystywany tylko do przygotowania algorytmu i do sprawdzania, czy wyznaczone przez algorytm genetyczny wartości parametrów były poprawne i czy pozwalały na wygenerowanie optymalnych trajektorii, dla wszystkich przygotowanych wstępnie, treningowych „sytuacji środowiskowych". w rzeczywistych zastosowaniach, stawiane jest wymaganie, aby opracowany i zastosowany system sterowania typu FTC, działał poprawnie dla wszystkich możliwych sytuacji, a nie tylko dla tych, których się wcześniej uczył. Dlatego głównym celem pracy było opracowanie algorytmu, generującego trajektorię dla w zasadzie dowolnego, realizowalnego zadania, co odpowiada drugiemu rodzajowi pracy.

4.2 Funkcja nagrody

Pierwsze prace nad algorytmem dotyczyły sformułowania wspomnianej wyżej funkcji nagrody, która służyła do oceny zachowania robota, w zależności od aktualnych zmiennych dotyczących robota i jego otoczenia. Na początku wykorzystano następujące proste równanie dla *i*-tego kroku:

$$r(i) = d_g(i) - d_e(i), \tag{28}$$

gdzie: i - numer kroku, $r(i)$ - wartość nagrody w i-tym kroku, $d_g(i)$ - odległość TCP do punktu docelowego, $d_e(i)$ - suma odległości poszczególnych ramion robota od jednej lub dwóch przeszkód.

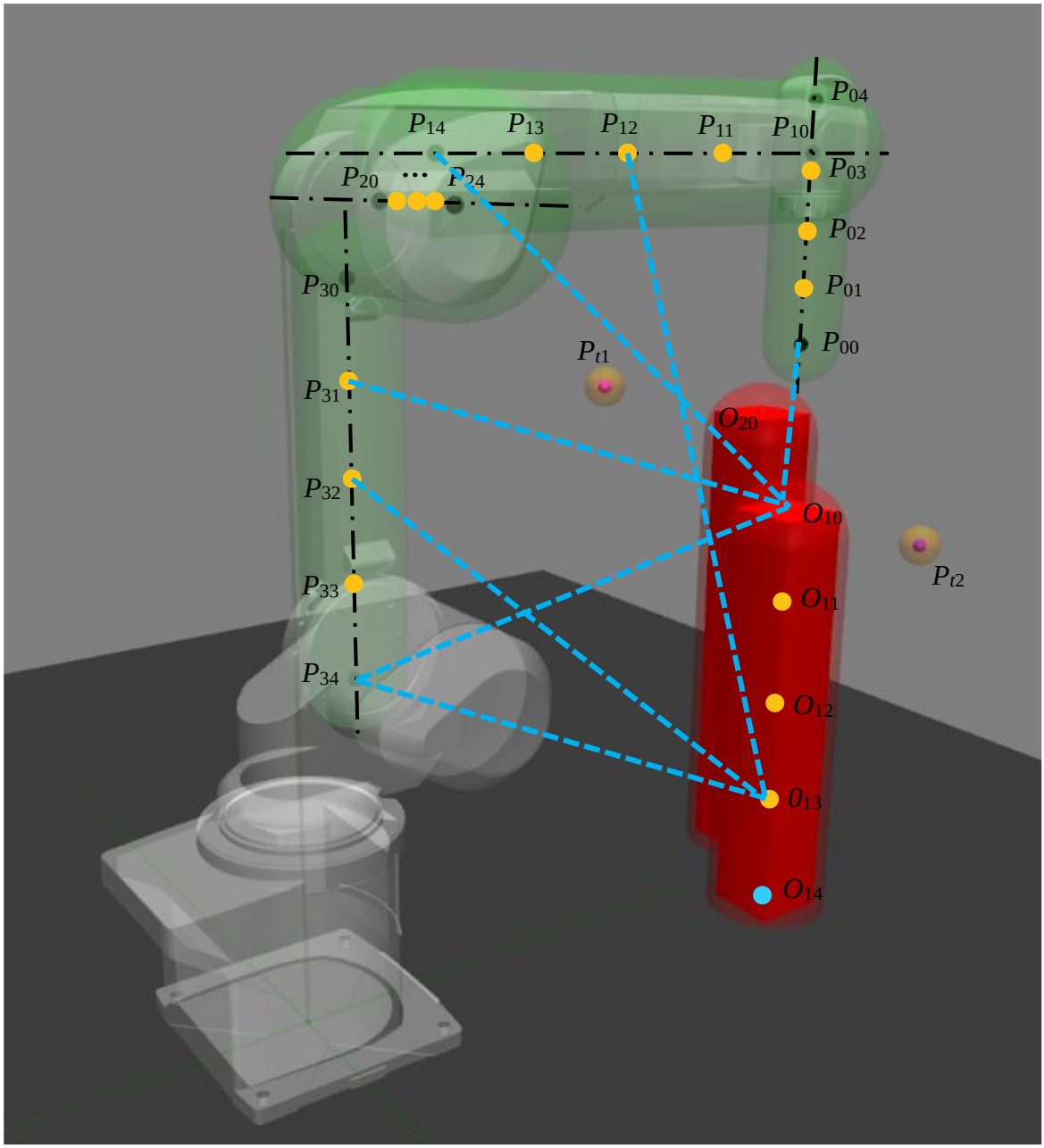
Zgodnie z funkcją (28) im mniejsza była wartość nagrody r, tym korzystniejszy, z punktu widzenia algorytmu, był oceniany ruch TCP robota. w związku z tym algorytm powinien wybierać takie przemieszczenie w danym kroku, aby dążyć do jej zmniejszania. Sposób wyznaczania odległości poszczególnych ramion robota od przeszkód d_e przedstawiono na rysunku 15. Linie przerywane koloru czarnego, oznaczają osie symetrii poszczególnych ramion robota. Na każdej z tych osi zaznaczono pięć punktów, umieszczonych w tej samej odległości od siebie, np. dla osi numer 0 były to: P_{00}, P_{01}, P_{02}, P_{03}, P_{04}. Podobnie oznaczono punkty na pozostałych trzech ramionach robota oraz na przeszkodach. Także na każdej z nich określono pięć punktów ponumerowanych od 0 do 4. Suma odległości rozważanych ramion robota od dwóch przeszkód w danym kroku i, jest wyrażona następująco:

$$d_e(i) = \sum_{a=0}^{3} \sum_{m=0}^{4} \sum_{o=0}^{1} \sum_{n=0}^{4} d(P_{am}, O_{on}) \tag{29}$$

gdzie: a = 0...3 jest liczbą ramion mogących wywołać kolizję, *m* = 0...4 jest liczbą punktów na każdym ramieniu, o (1 albo 2) jest liczbą przeszkód, *n* (od 0 do 4) jest liczbą punktów na osiach przeszkód, a $d(P_{am}, O_{on})$ jest odległością między punktem P_{am} na ramieniu robota a punktem O_{on} na przeszkodzie, zdefiniowaną w następujący sposób:

$$d(P_{am}, O_{on}) = \sqrt{((xP_{am} - xO_{on})^2 + (yP_{am} - yO_{on})^2 + (zP_{am} - zO_{on})^2)} \quad (30)$$

gdzie: P_{am} - punkt *m*-ty na osi numer *a*; O_{on} - punkt *n*-ty na przeszkodzie numer *o*; xP_{am}, yP_{am}, zP_{am} – kolejne współrzędne punktu $P_{am\ w}$ przestrzeni; xO_{on}, yO_{on}, zO_{on} – kolejne współrzędne punktu $O_{on\ w}$ przestrzeni.



Rysunek 15. Punkty charakterystyczne opisujące położenie przeszkód i ramion robota; kolorem niebieskim oznaczono kilka wybranych odległości

Badania wykazały, że zastosowanie równania (28) prowadziło niekiedy do wyznaczania trajektorii ruchu TCP robota, które nie kierowały go do zdefiniowanego punktu docelowego. w zależności od przyjętych danych początkowych, robot dotykał przeszkodę, oddalał się od niej nawet na ok. 0,5 m a następnie wykonywał przypadkowe przemieszczenia w różnych kierunkach i ostatecznie po wielu próbach w ogóle nie docierał do punktu docelowego. Algorytm działający z tą funkcją nagrody sterował robotem, tak że ten wykonywał ruchy , polegające na naprzemiennym zbliżaniu się i oddalaniu od przeszkody. Po wykonaniu szeregu nieudanych prób dojścia do celu przez robota, zaproponowano następującą funkcję nagrody:

$$r(i) = d_g(i) - \left(\frac{d_e(i)}{(p_m + (i \cdot p_t))}\right) \tag{31}$$

gdzie: p_m, p_t – parametry.

Modyfikacja dotyczyła tylko sumy odległości poszczególnych ramion robota od przeszkód. Przyjęto że wartości parametrów p_m i p_t będą mieściły się w przedziale od 0 do 1 i będą wyznaczane oddzielnie przed każdym nowym zadaniem, tj. dla każdej sytuacji środowiskowej. w założeniu, parametry te miały dopasowywać funkcję nagrody do wykonywanego zadania i tym samym zapobiegać nieprawidłowym sytuacjom, obserwowanym w eksperymentach z wykorzystaniem równania (28). Parametr p_m został użyty w celu zwiększenia znaczenia odległości ramion robota od przeszkód, w aktualnie ocenianym kroku. z kolei występujący we wzorze (31) parametr p_t, mnożony przez wartość równą liczbie wykonanych kroków i, powodował że im większa była ich liczba, tym mniejsza była wartość odjemnika w równaniu (31), a więc znaczenie odległości robota od przeszkód $d_e(i)$ malało, w trakcie wykonywania ruchu tj. w miarę wzrostu liczby wykonanych kroków. w początkowej fazie ruchu, znaczenie parametru p_t jest niewielkie i wartość funkcji nagrody zależy głównie od odległości robota od przeszkody, co pozwala na to aby TCP robota oddalał się od celu, co ułatwia albo wręcz umożliwia omijanie przeszkód. w miarę postępu ruchu robota, wartość funkcji nagrody coraz mniej zależy od odległości robota od przeszkody, a coraz bardziej zależy od odległości robota od celu, co powoduje zbliżanie się TCP do punktu docelowego.

4.3 Zastosowanie algorytmu genetycznego

W zaproponowanej funkcji nagrody występują dwa parametry p_m oraz p_t, których wartości nie są z góry znane. Powinny być one dobrane tak, aby algorytm działał poprawnie i w miarę możności optymalnie, czyli doprowadzał TCP robota do celu omijając

przeszkody. Dodatkowo, algorytm powinien realizować postawione zadanie, wykonując jak najmniejszą liczbę kroków albo zużywając jak najmniej energii. Najpierw postanowiono rozwiązać problem doboru wartości parametrów p_m oraz p_t występujących w funkcji nagrody, poprzez wprowadzenie ich przez operatora. Wartości tych parametrów dobierane były na podstawie indywidualnych obserwacji, w taki sposób, aby jak najbardziej skrócić trajektorię ruchu TCP robota. Wymagało to wielokrotnego wpisywania parametrów do komputera sterującego, sprawdzania wyników, zmiany tych parametrów, ponownego sprawdzenia itd. w związku z tym w wielu przypadkach należało proces ten powtarzać kilkanaście lub nawet kilkadziesiąt razy, a mimo tego nie było gwarancji, iż wpisane parametry , są bliskie optymalnym. Dlatego podjęto prace nad opracowaniem metody albo specjalnego algorytmu do wyznaczenia wartości tych parametrów. Po wykonaniu rozpoznania wstępnego, zaproponowano wykorzystanie algorytmu genetycznego (AG), którego zadaniem było znalezienie takich wartości parametrów funkcji nagrody, które zapewniły by ruch TCP robota po najkrótszej ścieżce do punktu docelowego, dla zadanych danych początkowych, zwanych dalej "sytuacją środowiskową". Każda z takich "sytuacji środowiskowych" jest opisana zbiorem:

$$\boldsymbol{S_i} = \{P_s, P_{t1}, P_{t2}, O_1, O_2, \alpha_s, \beta_s, j_{1i}, j_{2i}, j_{3i}, j_{4i}, j_{5i}\}, \tag{32}$$

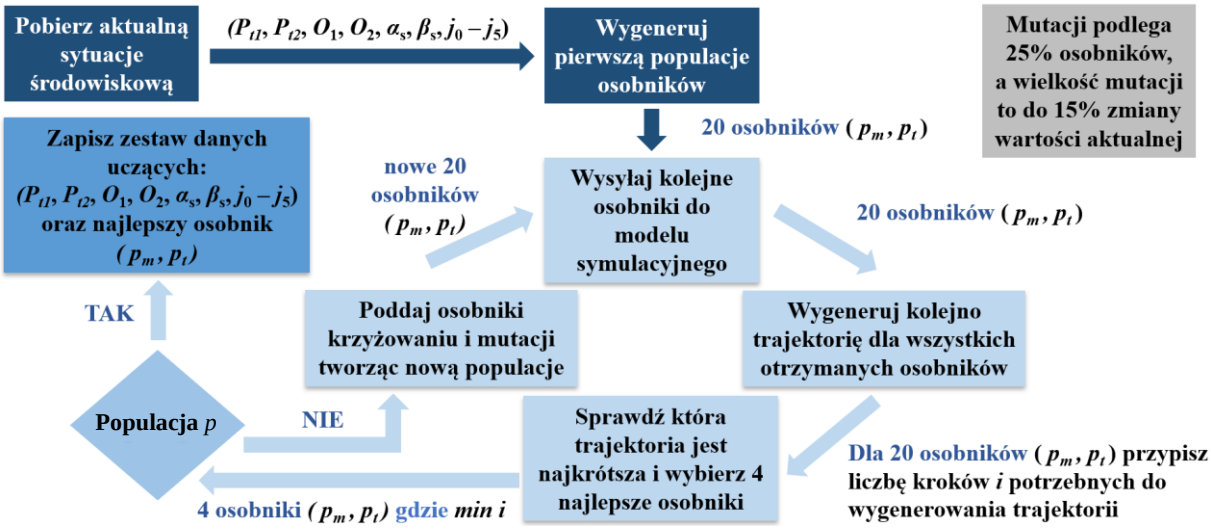
gdzie: P_s, P_{t1}, P_{t2}, to współrzędne odpowiednio: punktu startowego, punktu początkowego (chwyć obiekt) i końcowego (puść), współrzędne wierzchołków przeszkód O_1 i O_2, α_s i β_s to kąty obrotu osi odpowiednio dla ruchu szybkiego i wolnego oraz j_1 do j_5 to stany poszczególnych osi, określające czy oś jest uszkodzona (stan 0) czy nie (stan 1). Założono, że przeszkody będą znajdowały się w odległości większej niż 100 mm od punktu startowego i końcowego. w przypadku robota 6-osiowego, dla którego ponumerowano osie od 0 do 5, można wyróżnić następujące jego stany, dotyczące sprawności osi:

- wszystkie osie są sprawne (1 możliwość),
- jedna z osi jest uszkodzona (6 możliwości),
- dwie osie są uszkodzone (15 możliwości).

Daje to w sumie 22. konfiguracje stanów robota, w tym 21 różnych konfiguracji z jedną bądź dwiema awariami osi. Celem działania AG było znalezienie optymalnych wartości parametrów p_m i p_t funkcji nagrody, dla określonej (ograniczonej) liczby różnych „sytuacji środowiskowych”. w wykonanych badaniach AG, ich liczbę przyjęto jako iloczyn 22. możliwych konfiguracji (stanów) kinematycznych robota oraz 20. różnych położeń

przeszkód i punktów docelowych. Zgodnie z tym, na początku działania procedury algorytmu genetycznego, wygenerowano 440 różnych "sytuacji środowiskowych", $\boldsymbol{S_i}$. w trakcie działania, algorytm genetyczny wykorzystywał po kolei te zadane „sytuacje środowiskowe", do wyznaczania dla nich, optymalnych wartości parametrów p_m i p_t funkcji nagrody. Algorytm genetyczny został zaimplementowany w języku *Python*. Jego sposób działania można opisać następująco:

1. Jeśli jest, to pobierz sytuację środowiskową opisaną zbiorem $\boldsymbol{S_i}$, jeśli nie ma to ZAKOŃCZ.
2. Utwórz populację początkową chromosomów , to jest wygeneruj 20 chromosomów składających się z dwóch losowo wygenerowanych liczb (z zakresu od 0,1 do 1,0), reprezentujących odpowiednio p_m i p_t.
3. Przesyłaj po kolei chromosomy zawierające p_m, p_t do algorytmu sterującego opisanego w rozdziale 3.1 i wygeneruj dla nich trajektorie (łącznie 20) zgodnie z równaniami (29-31).
4. Oceń efektywność każdej pary parametrów na podstawie liczby kroków, które model symulacyjny robota musiał wykonać, aby przemieścić TCP do celu, czyli im krótsza była trajektoria, tym lepiej.
5. Jeśli wykonano p populacji to zapisz najlepszą parę p_m, p_t i sytuację środowiskową oraz przejdź do punktu 1, w przeciwnym razie przejdź do punktu 6.
6. Wykonaj selekcję, krzyżowanie i mutację, tworząc nową populację chromosomów i przejdź do punktu 3.
7. Przejdź do punktu 1.



Rysunek 16. Proces generowania wartości parametrów funkcji nagrody przez algorytm genetyczny

W opisanym wyżej graficzne procesie wyznaczania wartości optymalnych dla parametrów funkcji nagrody przedstawiono na rysunku 16. w trakcie przeprowadzonych prac, algorytm genetyczny przeanalizował wszystkie 440 "sytuacje środowiskowe", poszukując wartości parametrów p_m i p_t, zapewniających najlepszą tj. najkrótszą trajektorię ruchu. Wykonanie wszystkich obliczeń, łącznie z symulacjami dla jednej populacji, zajmowało ponad 30 minut. Trwało to tak długo, ponieważ rozważano robota 6-osiowego i aby wyznaczyć każdy jego krok, jak wynika ze wzoru (27), wykonano obliczenia dla 64 możliwych ruchów wszystkich osi. Obliczenia należało wykonać dla 440 sytuacji, co zajęło prawie 250 godzin (ponad 10 dni), dla jednej populacji. Dlatego zdecydowano, aby obliczenia wykonać tylko dla trzech populacji co zajęło aż 750 godzin nieprzerwanej pracy AG. Dlatego przyjęto, że $p = 3$, a uzyskane rozwiązania uznano za prawie-optymalne (semi-optymalne). Przeprowadzone badania wykazały, że nawet tak mała liczba cykli, wystarczyła do uzyskania zadowalających wyników w 293. przypadkach z 440. rozpatrywanych. Dla 293. przypadków, uzyskano parametry p_m i p_t pozwalające na wygenerowanie trajektorii bezkolizyjnej. w 134. sytuacjach robot z uszkodzoną jedną lub dwiema osiami, nie mógł w ogóle osiągnąć pozycji docelowej, ponieważ znajdowała się ona w tzw. obszarze kolizyjnym (to obszar, który powoduje kolizję któregoś z członów robota z przeszkodą) lub poza obszarem roboczym robota z uszkodzonymi osiami. Pozostałe 13 błędnych sytuacji wynikało z tego, że AG nie znalazł rozwiązania, mimo że punkt docelowy znajdował się w jego obszarze roboczym. Mogło to być spowodowane wykonaniem przez AG zbyt małej liczby populacji. Do oceny poszczególnych osobników zbudowanych z chromosomów zawierających p_m i p_t, sprawdzano liczbę kroków, które wykonał robot sterowany przez algorytm, wykorzystujący te chromosomy. w algorytmie wybierano, stosując metodę rankingową, cztery najlepsze osobniki tj. chromosomy. Następnie łączono je losowo w dwadzieścia par, które poddawano procesowi krzyżowania. Na wygenerowanych nowych osobnikach przeprowadzono proces mutacji, której prawdopodobieństwo wynosiło 0,25, co oznaczało, że aż 5 z 20 osobników wygenerowanych w procesie krzyżowania ulegało mutacji. Mutacja wprowadzała losowe zmiany parametrów o ±15% ich aktualnej wartości. Powstałe chromosomy tworzyły nową populację, na której wykonywana była kolejna sekwencja algorytmu genetycznego. Po obliczeniu trzech populacji wybierano najlepszego osobnika zawierającego parametry p_m i p_t funkcji nagrody. Najpierw algorytm genetyczny został wykorzystany do budowy tablicy, która posłużyła do sterowania, a dokładniej do wyboru, dla aktualnej sytuacji środowiskowej, najlepszych parametrów

funkcji nagrody. Była to, wymieniona wyżej, praca w trybie 1 systemu sterowania robotem, w którym algorytm wybierał parametry funkcji nagrody p_m i p_t na podstawie określonej sytuacji środowiskowej, z danych stabelaryzowanych. Później dane wygenerowane przez AG posłużyły do nauki sztucznej sieci neuronowej.

4.4 Opis procesu sterowania

Parametry opisujące sytuację środowiskową zestawiono w tabeli I. Stałe wykorzystywane podczas procesu sterowania są następujące: pozycje przeszkód O_1, O_2, punkty docelowe P_{t1}, P_{t2} oraz kąty, o które mogą obracać się wszystkie osie aktywne w jednym kroku, dla ruchu szybkiego to α_s lub wolnego to β_s. Te dwa parametry (kąty) algorytmu były wcześniej ustalane przez operatora. w wykonanych badaniach, wolny ruch tj. pozycjonowanie (obracanie o kąt β_s), rozpoczynał się najczęściej w odległości mniejszej niż 20 mm od punktu docelowego. Dzięki temu ruch TCP robota w jego końcowej fazie był płynniejszy, bezpieczniejszy i bardziej precyzyjny. Awaria osi mogła być zgłaszana w każdym momencie ruchu robota, tzn. parametry określające aktywność osi $j0 \div j5$, mogły być zmieniane w trakcie ruchu, gdzie zmiana następowała po wykryciu awarii. Po zakończeniu działania AG, do każdej "sytuacji środowiskowej" dodawano znalezione przez AG, optymalne parametry p_m i p_t, tworząc tym samym tabelę, która była wykorzystywana w trakcie działania systemu sterowania odpornego na błędy FTC, pracującego w trybie 1, jak podano na str. 43.

TABELA I
PARAMETRY ŚRODOWISKOWE WYKORZYSTYWANE DO OSZACOWANIA PARAMETRÓW p_m i p_t przez ALGORYTM GENETYCZNY

Symbol	Opis
O_{1x}, O_{1y}, O_{1z}	przeszkoda pierwsza, pozycje x, y, z [m]
O_{2x}, O_{2y}, O_{2z}	przeszkoda druga, pozycje x, y, z [m]
P_{t1x}, P_{t1y}, P_{t1z}	pierwszy punkt docelowy, pozycje x, y, z [m]
P_{t2x}, P_{t2y}, P_{t2z}	drugi punkt docelowy, pozycje x, y, z [m]
α_s	wielkość kroku szybkiego dla poszczególnych osi [°]
β_s	wielkość kroku wolnego dla poszczególnych osi [°]
$j0$, $j1$, $j2$, $j3$, $j4$, $j5$	stan osi, 0 jeśli oś uszkodzona, 1 jeśli oś aktywna

Opisany w rozdziale 3.1 algorytm pracował w trybie krokowym i w wyniku każdego kroku, generował najlepsze z punktu widzenia funkcji nagrody współrzędne ruchu robota. Najlepsze oznacza tutaj, że robot zbliżał się do celu poruszając się po najkrótszej trajektorii, unikając kolizji z przeszkodą. Przy wyznaczaniu trajektorii, brano pod uwagę tylko takie

działania, które znajdowały się w zbiorze wszystkich dopuszczalnych przemieszczeń, jakie robot mógł wykonać w danym kroku.

Algorytm 1: Proces sterowania.

```
1   Inicjalizacja tablicy akcji Θ
2   Inicjalizacja tablicy do kopiowania najlepszych akcji
3   for krok in range k_max
5     for akcja Θ_i in range rozmiar Θ
6       Wykonaj akcje: Θ_i
7       Pobierz pozycje rys.15 P_00 -P_34, TCP)
8       Oblicz d_g(i) dystans TCP kolejno od: P_t1 -> P_t2 -> P_s
9       Oblicz d_e(i) dystans P_00 -P_34 do O_1 i O_2
10        Sprawdź czy wystąpiła kolizja
11        if wystąpiła kolizja
12          Akcja niepoprawna wykonaj kolejną akcję
13        Pobierz p_m i p_t wygenerowane przez AG dla sytuacji środow.
14        Oblicz funkcje nagrody: r(i) = d_g(i) − (d_e(i) / (p_m + (i·p_t)))
15        if r(i) < r(1) do r(i-1)
16          Zapisz akcje dla kroku: θ_step = θ_step-1 + Θi
17        Ustaw pozycje robota dla kroku poprzedniego: θ_step-1
18        if wszystkie akcje zostały sprawdzone
19          Wykonaj akcje z tablicy kopii najlepszych akcji: θ_step
20          Sprawdź czy wszystkie osie są sprawne: θ_step = θ_real_robot
21          if osie są sprawne: θ_step = θ_real_robot
22            Idź do kolejnego kroku
23        else
24            Powtórz process ze zmienioną kinematyką
```

Główny moduł sterujący przedstawiono w Algorytmie 1, jako program komputerowy. Przed rozpoczęciem działania algorytmu, operator wprowadzał wartości kroku α_s i β_s, czyli wyrażane w stopniach przemieszczenia kątowe osi robota, dla ruchu szybkiego i wolnego, tj. pozycjonowania. w algorytmie występował także parametr k_{max}, oznaczający maksymalną, ustaloną przed rozpoczęciem działania liczbę kroków, jakie może wykonać program aby, robot osiągnął pozycję zadaną. Przekroczenie tej liczby oznaczało, że robot w ogóle nie mógł uzyskać celu. Po przekroczeniu założonej maksymalnej liczby kroków, system generował informacje o błędzie. Wartość tego parametru została eksperymentalnie ustawiona na maksymalną liczbę równą 500 kroków, która powinna pozwolić na osiągnięcie dowolnej pozycji końcowej ruchu.

4.5 Zastosowanie sztucznej sieci neuronowej

Po przebadaniu algorytmu genetycznego i sprawdzeniu, że poprawia on istotnie jakość generowanej trajektorii, biorąc pod uwagę kryterium najkrótszej, możliwej ścieżki TCP, postanowiono rozwinąć algorytm o możliwość sterowania robotem w obrębie zakładanej strefy roboczej, także w przypadku dowolnej, innej niż zastosowana w AG „sytuacji

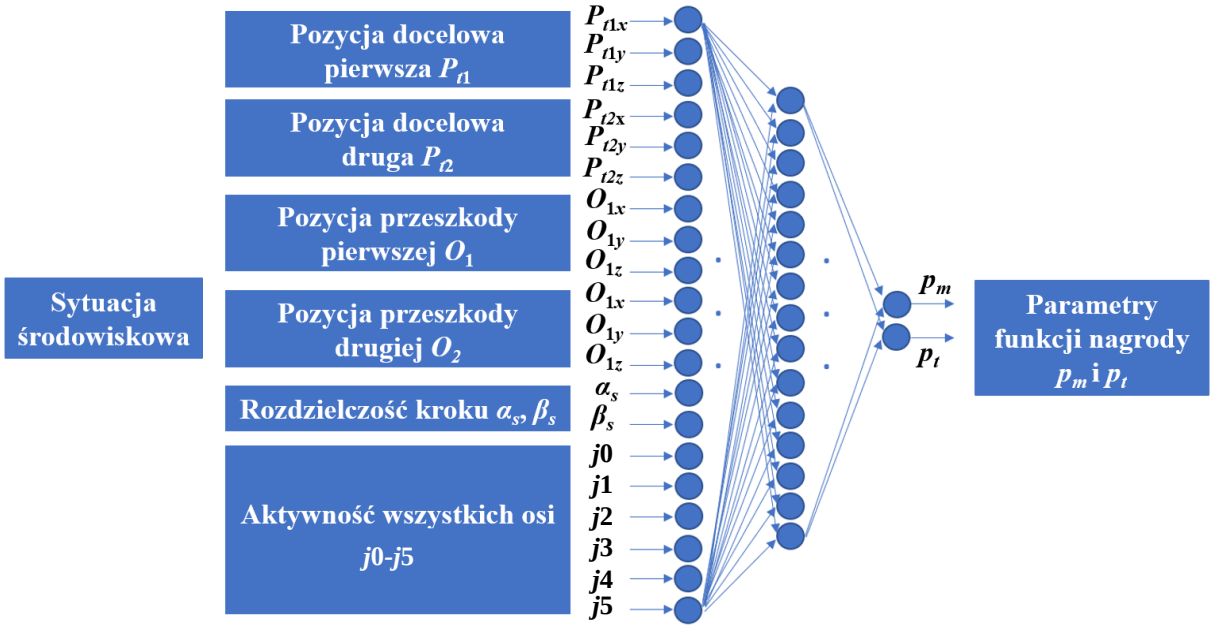
środowiskowej", opisanej w Tabeli I. We wcześniejszych badaniach algorytm funkcjonował jedynie dla danych, wcześniej wygenerowanych przez AG. Aby możliwe było oszacowanie wartości p_m oraz p_t także dla „sytuacji środowiskowych" innych niż te, które były stosowane przez algorytm genetyczny, do wyznaczania quasi-optymalnych parametrów, postanowiono do algorytmu sterowania robotem FTC, zaimplementować sztuczną sieć neuronową. Zadaniem takiej sieci było wyznaczanie wartości parametrów p_m oraz p_t dla dowolnej, aktualnej „sytuacji środowiskowej". w tym celu, dane wygenerowane przez AG w postaci tabeli posłużyły do uczenia SSN.

Przyjęto założenie, że zastosowanie sztucznej sieci neuronowej do wyznaczania parametrów funkcji nagrody, pozwoli na działanie algorytmu w sytuacji, gdy dana „sytuacja środowiskowa" nie była wcześniej wykorzystana przez algorytm genetyczny do wygenerowania i zapisania w tabeli, semi-optymalnych wartości parametrów p_m oraz p_t. w tym przypadku wykorzystywano właściwości SSN, do uczenia się na określonym zestawie danych wejściowych i wyjściowych, generowania sygnałów także dla sytuacji, które nie występowały w danych uczących. Dzięki zastosowaniu i nauczeniu SSN możliwe było sterowanie typu FTC, także dla sytuacji, które nie występowały w trakcie badań, wykonywanych z użyciem AG i dla których algorytm ten nie wygenerował parametrów semi-optymalnych. Zastosowanie sztucznej sieci neuronowej umożliwiło, działanie algorytmu w trybie pracy 2. Należy zauważyć, że gdy dane wejściowe są inne niż w zbiorze uczącym, to sieć neuronowa jest w stanie tylko oszacować albo aproksymować dane wyjściowe, zgodnie z wagami neuronów, wyznaczonymi w trakcie uczenia. Tym samym, zastosowanie takiej sieci ma przewagę nad zastosowaniem samych tablic, w których sytuacje środowiskowe (wejścia) i dane w tablicy (wyjścia) są ze sobą sztywno powiązane.

W zaproponowanym rozwiązaniu sztuczna sieć neuronowa służyła do wyznaczania parametrów funkcji nagrody na podstawie aktualnej sytuacji środowiskowej, tj. położenia przeszkód O_1, O_2, położenia pierwszego i drugiego punktu docelowego Pt_1, Pt_2, kąta ruchu szybkiego α_s, kąta ruchu wolnego β_s oraz aktualnej kinematyki robota, wynikającej z uszkodzenia jednej albo dwóch osi: $j0$, $j1$, $j2$, $j3$, $j4$ i $j5$. Sieć neuronowa w pierwszym kroku była uczona z wykorzystaniem przygotowanych przez AG tablic. Nauczona, brała następnie aktywny udział w procesie sterowania, poprzez generowanie parametrów funkcji nagrody. Parametry te podstawione do owej funkcji, determinowały w jaki sposób oceniane były poszczególne akcje z przestrzeni akcji $\boldsymbol{\theta}$ opisanej wzorem (27).

W Tabeli i przedstawiono parametry, które były brane pod uwagę podczas procesu sterowania robotem. Parametry te były także wykorzystywane jako wejścia sztucznej sieci neuronowej. Parametrami wyjściowymi sieci były dwa parametry p_m i p_t funkcji nagrody. Zaproponowana w omawianym tutaj rozwiązaniu, sieć neuronowa to perceptron wielowarstwowy. Liczba neuronów wejściowych odpowiadała liczbie parametrów opisujących aktualną „sytuację środowiskową" i dlatego była równa 20-stu neuronom. Sieć posiadała jedną warstwę ukrytą o liczbie neuronów równej 14 neuronów. Taka właśnie liczba neuronów warstwy ukrytej, została wyznaczona w trakcie prób wykonanych zgodnie z eksperymentami przedstawionymi w publikacji [74]. w niniejszej pracy liczba neuronów warstwy ukrytej została wstępnie oszacowana na podstawie liczby neuronów wejściowych perceptronu wielowarstwowego. w rezultacie przeprowadzonych badaniach stwierdzono, że użycie tylko 14-tu neuronów w warstwie ukrytej, wystarczyło do rozwiązywania postawionych sztucznej sieci neuronowej zadań. Zastosowana SSN, była w stanie nauczyć się, a następnie poprawnie wygenerować wartości parametrów funkcji nagrody.

Wyjściem sieci były dwa neurony, generujące na wyjściu liczby odpowiadające wartości parametrów p_m i p_t funkcji nagrody. Architektura sieci została przedstawiona na rysunku 17. Jako funkcje aktywacji zastosowano w niej funkcje ReLU [75] dla neuronów we wszystkich trzech warstwach. Sieć była uczona metodą propagacji wstecznej błędu [76].



Rysunek 17. Architektura sztucznej sieci neuronowej

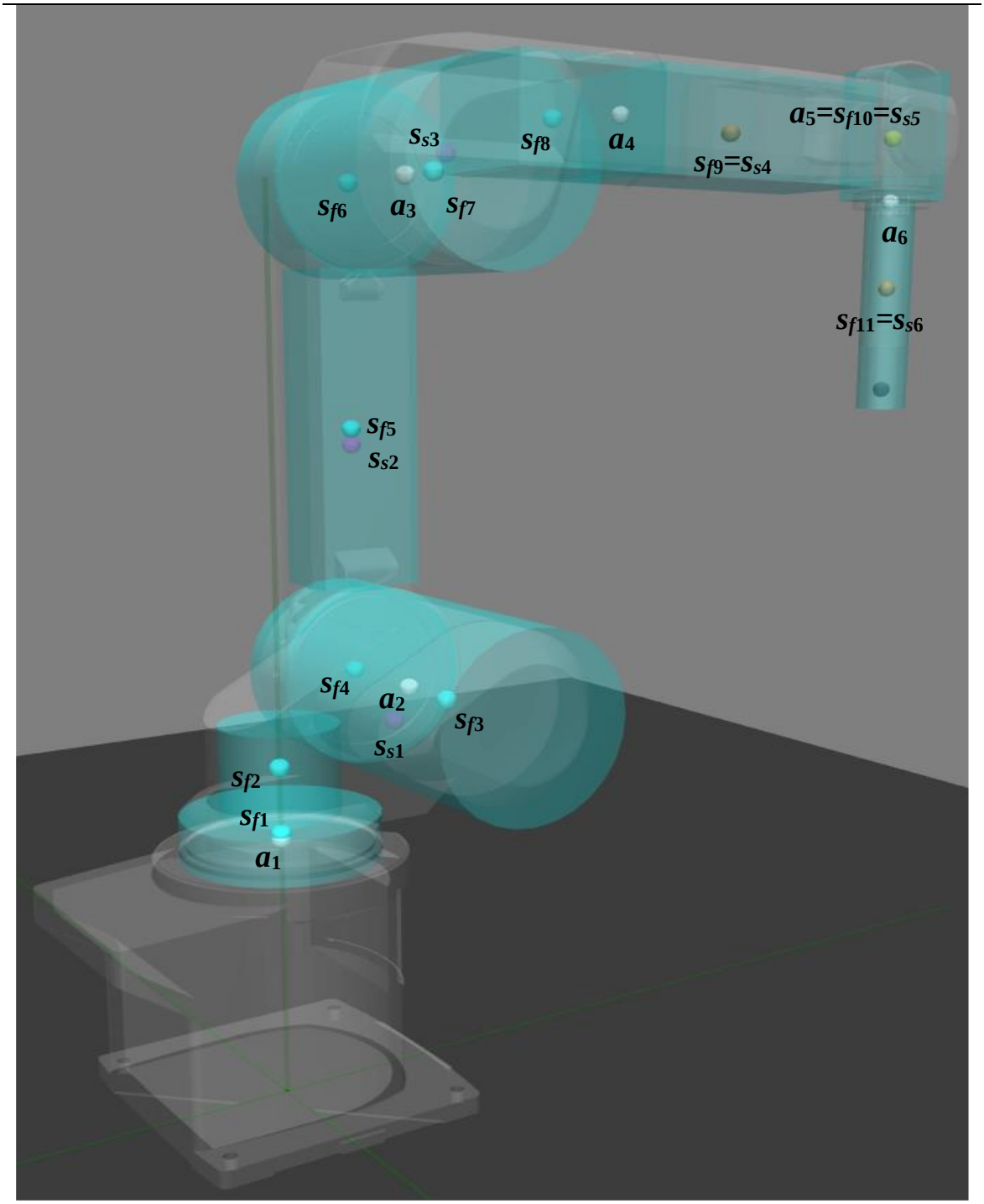
4.6 Model robota do badań zużycia energii

W kolejnych badaniach sprawdzano, czy zastosowana metoda sterowania typu FTC, pozwoli również na generowanie takiej trajektorii ruchu TCP, aby zużycie energii przez robota było minimalne. w tym celu przeprowadzono trzeci test, w którym wykorzystano kryterium jak najmniejszego zużycia energii. w tym przypadku badania przeprowadzono również dla maksymalnie dwóch uszkodzonych osi. Badanie miało na celu udowodnienie poprawności i skuteczności działania algorytmu w momencie uszkodzenia dwóch osi dla innego od omówionego wyżej, kryterium optymalizującego trajektorię ruchu. Podobnie jak w poprzednich przypadkach, do badania wykorzystano 220 „sytuacji środowiskowych". Modelem wykorzystywanym do określania zużycia energii był tzw. model momentów. Na rysunku 18 zaprezentowano taki model robota, wykonany w tzw. silniku fizycznym MuJoCo.

TABELA II

PARAMETRY POBIERANE Z MUJOCO DO MODUŁU OBLICZAJĄCEGO ZAPOTRZEBOWANIE ENERGETYCZNE NA PODSTAWIE MOMENTU

Parametry	**Opis**
a_1, a_2, a_3, a_4, a_5, a_6	pozycja *x*, *y*, *z* dla osi 1, 2, 3, 4, 5, 6
s_{s1}, s_{s2}, s_{s3}, s_{s4}, s_{s5}, s_{s6}	pozycja *x*, *y*, *z* dla centrum masy segmentu 1, 2, 3, 4, 5, 6
s_{f1}, s_{f2}, s_{f3}, s_{f4}, s_{f5}, s_{f6}, s_{f7}, s_{f8}, s_{f9}, s_{f10}, s_{f11}	pozycja *x*, *y*, *z* dla centrum masy figury 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11



Rysunek 18. Uproszczenie konstrukcji poszczególnych członów robota do podstawowych figur geometrycznych, zaznaczonych na rysunku kolorem niebieskim

W Tabeli II zamieszczono opisy oznaczeń użytych w zbudowanym modelu momentów. Zgodnie z danymi z Tabeli II oraz rysunkiem 18 oznaczono położenia osi robota, położenia środków mas poszczególnych segmentów robota oraz położenia środków mas poszczególnych figur będących uproszczeniem geometrii robota. Model ten wykorzystuje sumę wszystkich momentów obrotowych, generowanych przez poszczególne

serwomechanizmy oraz prędkości obrotowe, do wyznaczenia wykonanej pracy. w modelu założono, że moment siły generowany przez silnik elektryczny jest równoważony przez sumę: momentu siły wynikającego z jego masy, momentu bezwładności zredukowanego na wał silnika oraz momentu tarcia i ewentualnie momentu oporu ruchu, spowodowanego przez obciążenie zewnętrzne. w prowadzonych badaniach, ten ostatni składnik nie występował tzn. był równy 0, ze względu na to, że robot nie był obciążony. Dodatkowo przyjmowano, że struktura robota była jednorodna. Moment siły rozumiany był jako iloczyn wektorowy ramienia o długości *r*, pomiędzy punktem przyłożenia siły *F* a środkiem osi obrotu. Zależność na moment obrotowy opisana jest wzorem:

$$\overrightarrow{M_f(t)} = \vec{r} \times \overrightarrow{F(t)} \tag{33}$$

gdzie: $\overrightarrow{M_f}$ – wektor momentu siły obliczony przez model [Nm], r – wektor punktu przyłożenia siły [m], $\vec{F}$ – wektor siły [N].

Zgodnie z definicją iloczynu wektorowego, wartość wektora momentu siły obliczana jest ze wzoru:

$$M_f(t) = rF(t)sin\sigma(t) \tag{34}$$

gdzie: M_f – moment siły obliczony przez model [Nm], r – ramię przyłożenia siły [m], $F = mg$ – siła ciężkości ramienia robota [N], σ – kąt pomiędzy ramieniem działania siły a prostą poziomą.

Do obliczenia momentu siły model wykorzystano położenie osi a_j, gdzie j jest numerem osi, oraz położenie środków masy każdego segmentu s_{sk}, gdzie k jest liczbą kolejnych segmentów, na które podzielono model robota. Jak widać na rys. 18, kształty ramion robota zostały przybliżone za pomocą figur geometrycznych, w postaci walców i prostopadłościanów. Za pomocą programu CAD Inventor wyznaczono środki masy każdego z segmentów (ramion). Znając masę całego robota oraz jego wymiary geometryczne, w tym programie wyznaczono zarówno środki masy poszczególnych segmentów, jak i ich masy. Wartości te zostały wstawione do modelu w silniku fizyki MuJoCo. Znając położenia osi, położenia środków mas ramion, wartości kątów pomiędzy wektorami r i F oraz masy poszczególnych segmentów, algorytm obliczał momenty sił w poszczególnych osiach, zgodnie z równaniem (34).

Uproszczenie geometrii robota pozwoliło, stosunkowo łatwo wyprowadzić wzory na momenty bezwładności figur względem danej osi. Moment bezwładności ciała

składającego się z *n* punktów materialnych jest sumą momentów bezwładności wszystkich tych punktów względem wybranej osi obrotu. Dla ciał o ciągłym rozkładzie masy, we wzorze na moment bezwładności występuje całkowanie. Gdy ciało zostanie podzielone na nieskończenie małe elementy o masach *dm* i *r* będzie oznaczać odległość każdego takiego elementu od osi obrotu, to wtedy moment bezwładności dany jest wzorem:

$$I = \int_M r^2 \ dm = \int_V \rho r^2 \ dV \tag{35}$$

gdzie całkowanie przebiega przez masę *m* ciała lub przez jego objętość *V*.

Dla ciał, które można podzielić na małe elementy składowe, całkowanie występujące we wzorze na moment bezwładności, zastępuje się sumowaniem iloczynów mas i ich odległości od osi obrotu, co wyraża równanie:

$$I = \sum_{j-1}^{n} m_j \, r_j^2 \tag{36}$$

Do obliczenia momentów bezwładności dla danej osi napędowej robota, model wykorzystuje położenia osi a_j, gdzie j jest numerem osi, oraz położenie środków mas poszczególnych figur s_{fw}, gdzie w jest numerem kolejnej figury. Przez figurę rozumie się poszczególne elementy robota, które dla uproszczenia obliczeń zostały przybliżone do kształtu podstawowych przestrzennych figur geometrycznych.

Do obliczenia momentu tarcia w łożyskach wykorzystano wzór na moment tarcia w funkcji obciążenia [77]:

$$M_T(t) = f F_z(t) d_m \tag{37}$$

gdzie M_T to moment tarcia jako funkcja obciążenia [Nm], f współczynnik zależny od rodzaju i wielkości łożyska oraz współczynnika dopuszczalnego obciążenia statycznego, F_z to obciążenie zastępcze (równoważne), które wyznaczano jako siła nacisku działająca prostopadle do osi obrotu na dane łożysko [N], d_m to średnica podziałowa łożyska [m].

Energia całkowita jaką zużywał robot, równa była sumie wykonanych prac składowych, obliczanych we wszystkich osiach na podstawie momentów sił. Do obliczenia mocy posłużono się równaniem:

$$P_f(t) = \ M_f(t) \cdot \omega(t) \tag{38}$$

gdzie: P_f – moc wynikająca z momentu siły [W], M_f – moment siły obliczony przez model [Nm], ω – prędkość kątowa [rad/s]

Do obliczenia mocy związanej z momentem wynikającym z tarcia posłużono się równaniem:

$$P_T(t) = M_T(t) \cdot \omega(\mathrm{t}) \tag{39}$$

gdzie: P_T – moc wynikająca z momentu bezwładności [W], M_T – moment wynikający z tarcia obliczony przez model z równania (37) [Nm], $\omega = \frac{d\theta}{dt}$

Do obliczenia mocy związanej z momentem bezwładności posłużono się w ogólności wzorem:

$$P_I(t) = M_I(t) \cdot \omega(t) = I \cdot \varepsilon(t) \cdot \omega(t) \tag{40}$$

gdzie: P_I – moc wynikająca z momentu bezwładności [W], M_I – moment obrotowy do przyspieszenia, I – moment bezwładności obliczony przez model [kg $\cdot$ m^2], ε – przyspieszenie kątowe osi [rad/s^2].

Równania (38) – (40) stosowano dla każdej osi indywidualnie, a następnie sumowano ich wyniki w celu obliczenia mocy całkowitej, np. dla osi a_0:

$$P_{Ca_0}(t) = P_{fa_0}(t) + P_{Ta_0}(t) + P_{Ia_0}(t) \tag{41}$$

gdzie: P_C – moc całkowita [W], P_f – moc wynikająca z momentu siły [W], P_T – moc wynikająca z momentu tarcia [W], P_I – moc wynikająca z momentu bezwładności [W]

Algorytm uwzględniał również sprawność napędów manipulatora [78]. Wprowadzenie zagadnienia sprawności do modelu momentowego robota wykonano w celu porównania wyników pomiaru mocy na robocie rzeczywistym z mocą uzyskaną za pomocą modelu. Aby wyznaczyć zużycie energii tj. wykonaną pracę, chwilowe wartości mocy mnożono przez czas.

$$E_c = P_c \cdot t \tag{42}$$

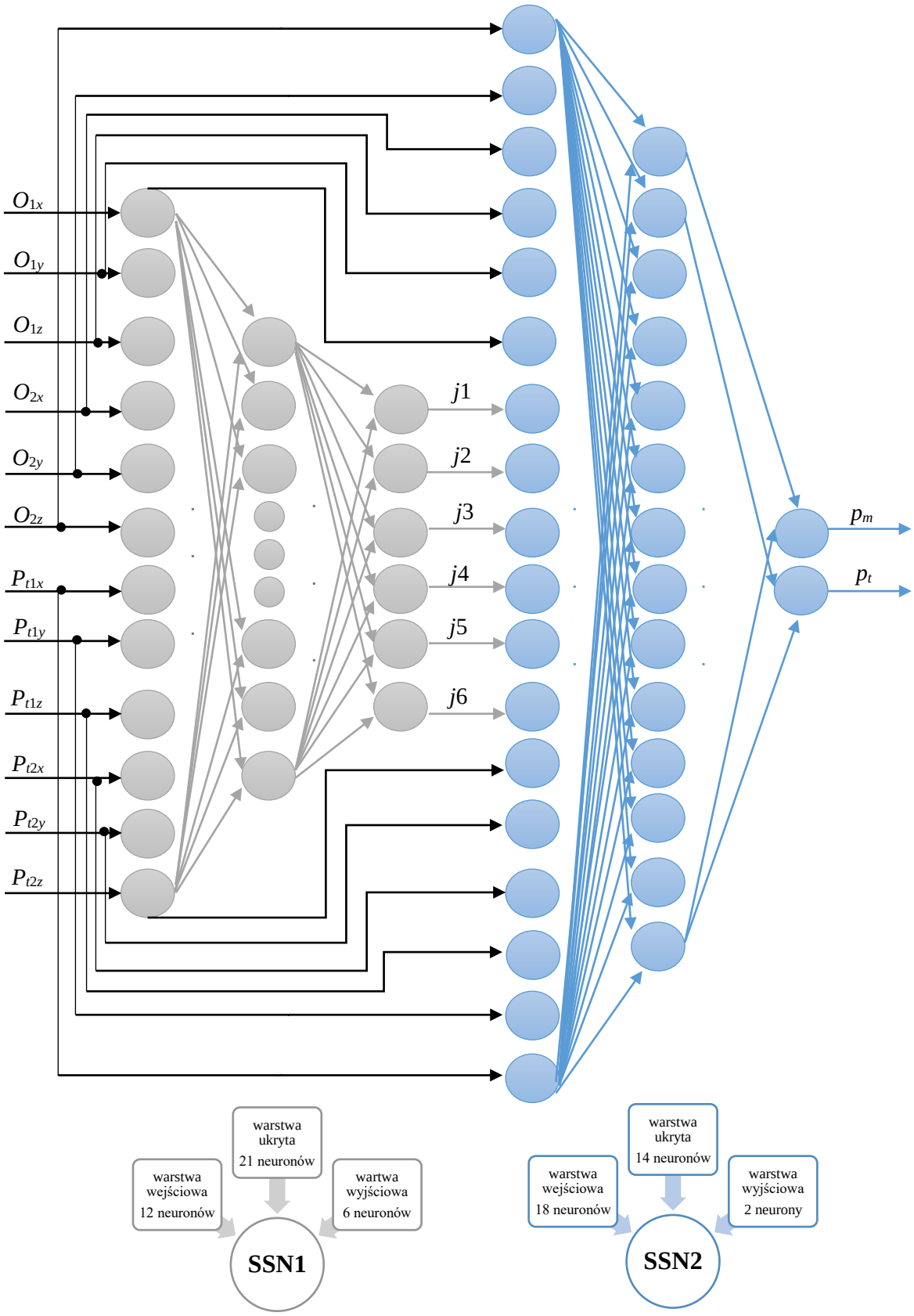
gdzie: E_c – energia jaką zużył robot [Ws], P_c – moc całkowita obliczona dla wszystkich osi, t – czas wykonywania danej trajektorii.

4.7 Algorytm minimalizujący zużycie energii

W ramach niniejszej pracy, wykonano także badania dotyczące opracowania algorytmu odpornego na błędy w postaci uszkodzenia jednej bądź dwóch dowolnych osi robota, wykorzystującego AG, SSN oraz funkcję nagrody do generowania takiej trajektorii robota, aby zużyta moc byłą minimalna. Założono, że podobnie jak poprzednio, algorytm powinien zapewniać omijanie do dwóch stałych przeszkód, zlokalizowanych w przestrzeni pracy robota. Opisane niżej badania wykonano na danych stabelaryzowanych wcześniej przez AG, a następnie wykorzystanych do nauczenia SSN. w przypadku kryterium minimalnego

zużycia energii zaprojektowano nową SSN, zbudowaną z dwóch sieci. Pierwsza z nich miała za zadanie, na podstawie aktualnej sytuacji środowiskowej, określić która konfiguracja kinematyczna pozwoli na wykonanie zadania, przy nakładzie najmniejszej ilości energii. Druga zastosowana sieć, była analogiczna do sieci wykorzystywanej w badaniu z użyciem kryterium najkrótszej możliwej trajektorii z tą różnicą, że w tym badaniu kąty obrotu osi α_s i β_s były stałe i wynosiły 1 stopień dla ruchu szybkiego oraz 0,2 stopnia dla ruchu wolnego. Architekturę zastosowanej sztucznej sieci neuronowej przedstawiono na rysunku 19.

W przypadku wykorzystania kryterium najmniejszego zużycia energii przez system sterowania, w algorytmie optymalizującym generowaną trajektorię zmieniono model symulacyjny na model momentowy oraz rozszerzono go o dodatkową sieć neuronową. Algorytm genetyczny działał w dokładnie taki sam sposób jak poprzednio, a jedyną różnicą było wybieranie najlepszego osobnika na podstawie kryterium zużycia energii a nie jak wcześniej, na podstawie najmniejszej liczby kroków. Działanie algorytmu opierało się na tej samej funkcji nagrody, której parametry p_m i p_t były ustalane przez AG tak, aby funkcja nagrody pozwalała na uzyskanie minimalnego zużycia energii dla danej sytuacji środowiskowej, w celu przemieszczenia TCP robota do punktu docelowego.



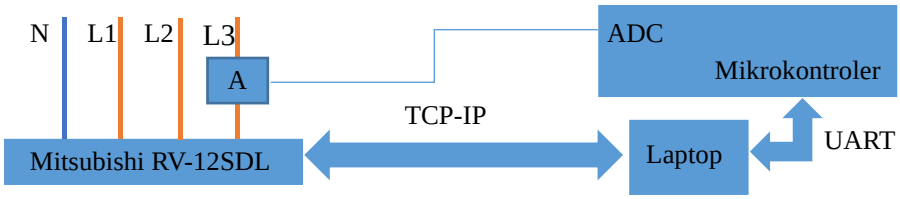
Rysunek 19. Architektura SSN do szacowania optymalnego, pod względem energetycznym łańcucha kinematycznego robota oraz do estymacji parametrów funkcji nagrody dla tej konfiguracji

W procesie generowania danych uczących przez AG dla danej sytuacji środowiskowej, wyodrębniano dodatkowo poszukiwanie odpowiedzi na pytanie, jaka konfiguracja kinematyczna, dla danych położeń przeszkód oraz punktów docelowych, pozwala na zużycie najmniejszej ilości energii. Dane te były potrzebne do nauki sztucznej sieci neuronowej nr 1, oznaczanej dalej jako SSN1 (patrz rysunek 19, kolor szary), która na podstawie pozyskanej wiedzy oraz pozycji przeszkód i punktów docelowych, wyznaczała optymalną energetycznie konfigurację kinematyczną robota. Jak wspomniano wyżej, dla kryterium energetycznego rozwiązano 220 „sytuacji środowiskowych", co oznacza 10 różnych pozycji przeszkód i punktów docelowych w 22. różnych konfiguracjach kinematycznych robota. Do wyodrębnienia danych do nauki SSN1, należało znaleźć trajektorie dla wszystkich 22. konfiguracji kinematycznych robota dla danych pozycji przeszkód i punktów docelowych, a następnie sprawdzić, która konfiguracja aktywnych osi, pozwala na zużycie najmniejszej ilości energii przez robota. Drugi zestaw danych uczących SSN2 był taki sam jak w przypadku kryterium najkrótszej możliwej trajektorii, z tym że sieć nie uwzględniała wartości kroku normalnego α_s oraz dokładnego β_s. Sztuczna sieć neuronowa numer 2 służyła do wyznaczania parametrów funkcji nagrody, jak to miało miejsce w przypadku SSN wykorzystywanej w algorytmie stosującym kryterium najkrótszej możliwej trajektorii. Zarówno SSN1 jak i SSN2 uczone były metodą propagacji wstecznej błędu. Nie uwzględnianie w algorytmie możliwości zmian parametrów α_s oraz β_s, spowodowane było tym, iż zmiana kroku a tym samym i prędkości ruchu, wpływa na wartość pracy wykonywanej w trakcie ruchu. w związku z tym, w badaniu algorytmu z kryterium minimum energii postanowiono, aby tak prędkości jak i przyspieszenia kątowe, w poszczególnych osiach były takie same, dla wszystkich konfiguracji kinematycznych. w przypadku takiego podejścia, można sprawdzić, czy w momencie uszkodzenia/rozłączenia jednej lub dwóch osi, algorytm potrafi dobrać konfigurację kinematyczną robota oraz trajektorię w taki sposób, aby zaoszczędzić energię.

Zaproponowane rozwiązanie oparte o kryterium zużycia energii jest również rozwiązaniem, które pozytywnie rozwiązuje sterowanie robotem po napotkaniu w trakcie ruchu, problemu w postaci uszkodzenia osi. Działa jednak tylko w przypadku gdy w wyniku działania SSN1 zostanie wybrana konfiguracja, w której: aktywne są wszystkie osie lub jedna oś jest nieaktywna. Wtedy w momencie awarii kolejnej osi, kinematyka robota może zostać zmieniona. w momencie gdy SSN1 rozłączy dwie osie a uszkodzeniu ulegnie trzecia oś, to działanie algorytmu jest niemożliwe, ponieważ w trakcie działania

AG, rozważano awarie maksymalnie dwóch osi robota. Sztuczna sieć neuronowa nr 2 działa dokładnie w taki sam sposób, jak w rozwiązaniu gdzie stosowano kryterium najkrótszej możliwej trajektorii.

Algorytm do sterowania typu FTC, minimalizujący zużycie energii przez robota przetestowano najpierw w warunkach symulacyjnych a następnie eksperymentalnie na robocie Mitsubishi RV-12SDL. Do pomiaru zużycia energii przez robota rzeczywistego wykonano układ przedstawiony na rysunku 20.



Rysunek 20. Schemat pomiarowy zużycia energii przez robota

Robot Mitsubishi RV-12SDL zasilany był napięciem trójfazowym 3 x 400VAC. Sprawdzono doświadczalnie, że układy napędowe robota stanowiły trójfazowe obciążenie symetryczne. Oznacza to, że w każdej fazie płynął prąd o tym samym natężeniu skutecznym. Dlatego do zmierzenia poboru mocy wystarczył pomiar natężenia prądu tylko w jednej z faz. w pomiarach wykorzystano amperomierz (A), mierzący chwilowe natężenie prądu skutecznego [79]. Pomiar realizowano w sposób opisany poniżej. Ruch robota rozpoczynał się w momencie, gdy z laptopa z pomocą komunikacji TCP-IP został wysłany pierwszy zestaw pozycji poszczególnych osi robota. w tym samym momencie oprogramowanie na laptopie wysyłało sygnał za pomocą komunikacji UART do mikrokontrolera 8 bitowego, o tym aby rozpocząć pomiar natężenia prądu płynącego w przewodzie fazowym. Mikrokontroler przeliczał sygnał analogowy na wartość cyfrową natężenia prądu fazowego i wysyłał tę informację do oprogramowania na laptopie. Natężenie prądu było odczytywane z przetwornika ADC przez mikrokontroler co 2,5 ms. Wyniki zapisywane były do pliku tekstowego, aby w późniejszym czasie możliwa była obróbka danych. Moc jaka była zużyta do wykonania zadanego ruchu obliczano ze wzoru:

$$P_r(k) = 3 \cdot U \cdot I_f(k) \cdot \cos(\varphi) \tag{43}$$

gdzie: k – krok, P_r – moc obliczona na podstawie pomiarów prądu fazy [W], $U = 400\text{VAC}$ – napięcie międzyfazowe [V], I_f – skuteczny prąd fazy [A], $\cos(\varphi)$ – 0,7 - współczynnik mocy, którego wartość przyjęto na podstawie danych z dokumentacji napędów stosowanych w Mitsubishi RV-12SDL [78].

5 Badania symulacyjne oraz doświadczalne

Do przeprowadzenia badań oraz do zbierania i prezentacji wyników, zaprojektowano szereg algorytmów napisanych w języku *Python*. Dzieląc cały system oprogramowania sterującego na moduły, można wyróżnić moduły:

- komunikacyjny (około 1000 linii kodu),

- obsługi robota w symulacji (około 2000 linii kodu),

- obsługi robota rzeczywistego (około 1000 linii kodu),

- algorytmu genetycznego (około 500 linii kodu),

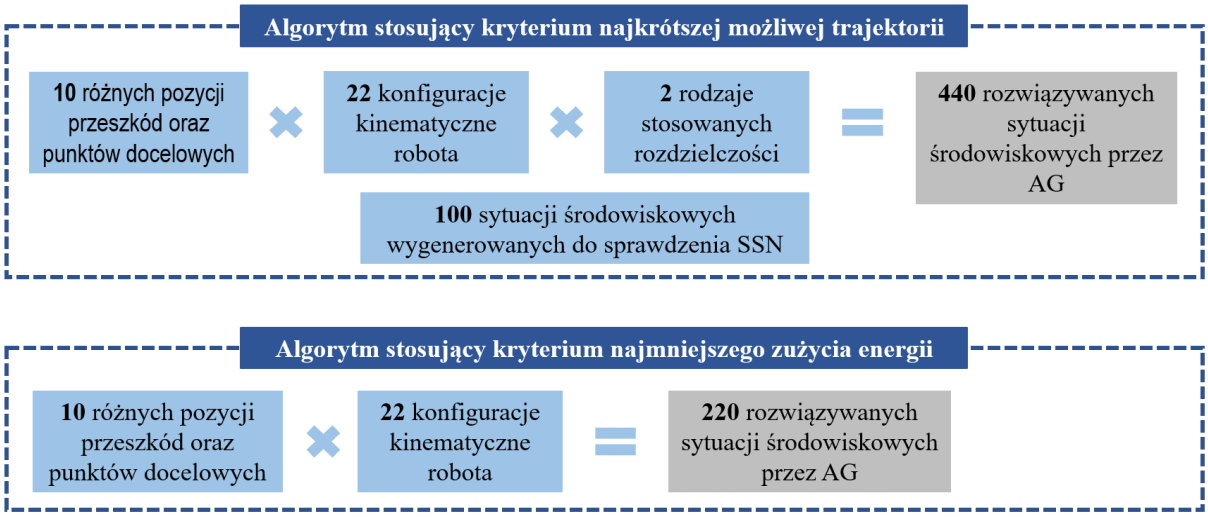
- sztucznej sieci neuronowej (około 500 linii kodu),

- obliczania modelu momentowego robota (około 1000 linii kodu),

- sprawdzający strefę roboczą robota (około 500 linii kodu),

- zapisywania i obróbki danych (około 1000 linii kodu).

Razem było to około 7500 linii kodu, co świadczy o znacznej złożoności opracowanego programu a tym samym algorytmu i systemu sterowania typu FTC.



Rysunek 21. Schemat blokowy pokazujący liczby rozwiązanych podczas badań sytuacji środowiskowych

5.1 Badanie algorytmu do określania strefy roboczej robota

Do określenia jakości działania zaproponowanego algorytmu, konieczne okazało się zaprojektowanie metody pozwalającej na wyznaczenie obszaru roboczego robota, dla danej

konfiguracji łańcucha kinematycznego. Uszkodzenie i zablokowanie którejś osi, zmniejsza bowiem strefę roboczą robota, dlatego wyznaczenie obszaru roboczego robota z uszkodzoną osią, pozwala określić czy aktualny punkt docelowy znajduje się w tym obszarze czy też nie, a tym samym stwierdzić, czy zaproponowany algorytm może w ogóle doprowadzić TCP robota do punktu docelowego. Jeśli nie, to jest oczywistym, że robot z uszkodzonymi osiami nie może dotrzeć do tego punktu. Jeśli punkt końcowy znajduje się w przestrzeni roboczej robota z uszkodzonymi osiami, a algorytm nie był w stanie doprowadzić jego TCP do tego punktu, to może to oznaczać, że algorytm został źle zaprojektowany lub niedostatecznie nauczony.

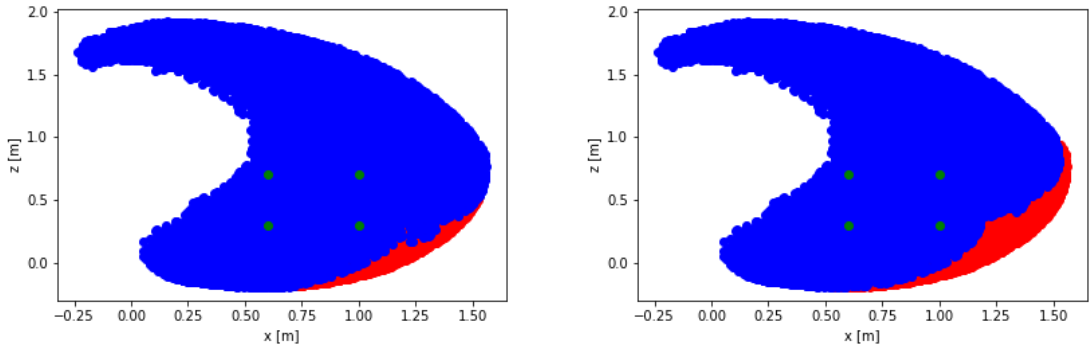
Do wyznaczenia obszaru roboczego robota posłużono się jego modelem symulacyjnym, wykonanym w silniku fizycznym MuJoCo. Model ten został zaprezentowany na rysunku 15. Zadaniem algorytmu do wyznaczania obszaru roboczego było ustawienie na stałe położenia kątowego (z momentu awarii) zablokowanych osi i wykorzystując model robota, zmienianie krokowo wszystkich położeń kątowych pozostałych tzn. aktywnych osi oraz wyznaczanie za każdym razem położenia TCP robota. Te prace wykonano dla wszystkich rozwiązanych sytuacji środowiskowych. Algorytm zapisywał pobrane z modelu symulacyjnego robota pozycje *x*, *y*, oraz z TCP robota. Zbiór tych zapisanych współrzędnych określał przestrzeń roboczą robota w danej "sytuacji środowiskowej" $\boldsymbol{S}$, dla każdej z możliwych konfiguracji robota. Dodatkowo sprawdzał czy dla każdego znalezionego położenia TCP, nie wystąpiła kolizja z przeszkodą znajdującą się w strefie roboczej robota. Do wyznaczania kolizji posłużono się opisem robota oraz przeszkód, przedstawionych na rys. 15. Gdy kolizja miała miejsce, to algorytm zapisywał to położenie *x, y,* z w osobnym pliku, aby możliwe było graficzne przedstawienie obszarów, dla jakich położeń robot nie jest wstanie przemieścić się bezkolizyjnie. w wyniku działania oprogramowania do sprawdzania strefy roboczej robota oraz sprawdzania ewentualnej kolizji, możliwa była ocena jakości algorytmu do wyznaczania bezkolizyjnej ścieżki z punktu startowego do końcowego. Algorytm 2 przedstawia program do generowania punktów TCP stanowiących przestrzeń roboczą manipulatora. Parametr α_w oznacza wartość kroku, czyli zmiany położeń kątowych poszczególnych osi robota. Poniżej zamieszczono Algorytm 2 generujący przestrzeń roboczą robota dla 6-ciu osi.

Algorytm 2: Proces sprawdzania przestrzeni roboczej manipulatora.

```
for j0 in range (-70°, 71°, αw):
    for j1 in range (-40°, 101°, αw):
        for j2 in range (-100°, 101°, αw):
            for j3 in range (-270°, 271°, αw):
                for j4 in range (-310°, 31°, αw):
                    for j5 in range (0°, 360°, αw):
                    skaluj położenie kątowe na radiany
                    ustaw ramie robota w pozycjach( j0, j1, j2, j3, j4, j5 )
                    sprawdź czy wystąpiła kolizja
                    if kolizja == prawda:
                        zapisz położenie TCP jako kolizyjne
                    else:
                        zapisz położenie TCP jako bezkolizyjne
```

Na rysunkach od 22 do 25 przedstawiono wyznaczone, przykładowe obszary robocze dla różnych uszkodzeń (zablokowanie) osi manipulatora. Obszary te zostały wyznaczone dla pozycji przeszkód: O_1(0,8; -0,075; 0,6), O_2(0,8; 0,075; 0,7). Kolorem niebieskim zaznaczono obszar do którego TCP robota ma możliwość dotrzeć w sposób bezkolizyjny. Kolorem czerwonym oznaczono obszar do którego TCP robota może dotrzeć, ale zawsze wystąpi kolizja. Dla przekroju na rysunku 22b), gdzie y>0, widać wyższą strefę kolizyjną, co wynika z faktu iż przeszkoda druga dla której y>0 jest przeszkodą wyższą.

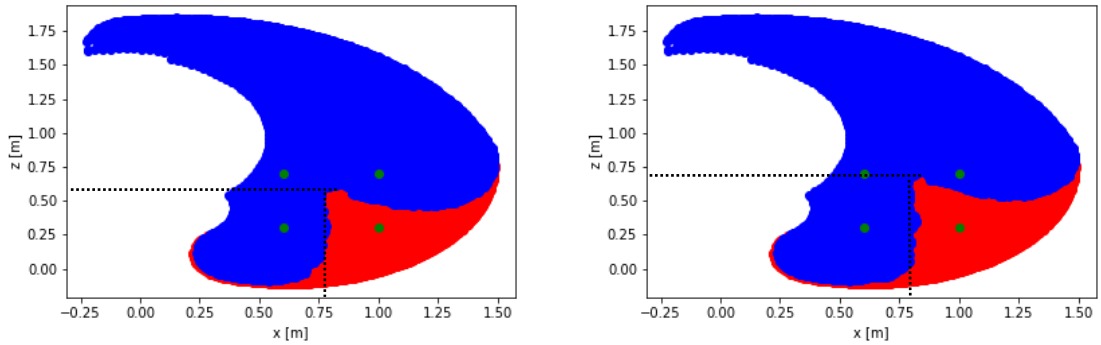
a) b)



Rysunek 22. Widok przekroju strefy roboczej manipulatora dla aktywnych wszystkich osi. a) dla y ∈ (-0,2028, -0,1972) m, b) dla y ∈ (0,1972, 0,2028) m. Zielone punkty oznaczają pozycje do których przemieszczało się TCP robota, na niebiesko strefa bezkolizyjna, na czerwono strefa kolizyjna, α_w = 10°

Na rysunku 22 widoczne są powierzchnie x-z obszaru roboczego robota, gdy sprawne są jego wszystkie osie. Rysunek ten pokazuje widok przekroju strefy roboczej dla określonego y. Dla rysunku 22a jest to przekrój dla wartości osi y z zakresu od –0,2028 metra do -0,1972 metra. z wykresów na rysunku 22 widać, że wszystkie 4 punkty docelowe są możliwe do osiągnięcia przez TCP robota, w sposób bezkolizyjny.

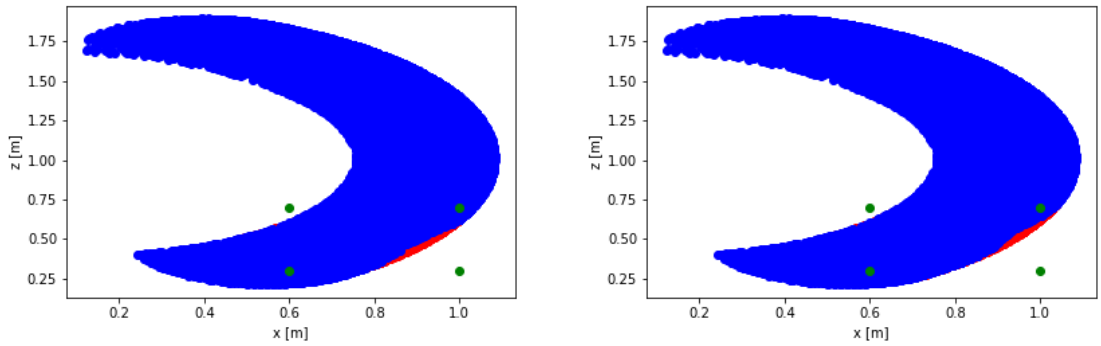
a) b)



Rysunek 23. Widok przekroju strefy roboczej manipulatora dla uszkodzonej osi *j*0. a) dla y ∈ (-0,2028, -0,1972) m, b) dla y ∈ (0,1972, 0,2028) m. Zielone punkty oznaczają pozycje do których przemieszczało się TCP robota, na niebiesko strefa bezkolizyjna, na czerwono strefa kolizyjna, $\alpha_w = 5°$

Na rysunku 23 pokazano powierzchnie strefy roboczej w przypadku uszkodzenia osi *j*0 robota. Można zauważyć, że dwa punkty docelowe (oznaczone na zielono) są niemożliwe do osiągnięcia przez TCP robota w sposób bezkolizyjny, ponieważ znajdują się w oznaczonej kolorem czerwonym strefie kolizyjnej. Za pomocą czarnych linii kropkowanych zaznaczono widoczne początki zarysu strefy kolizyjnej wynikające z rozmieszczenia przeszkód O_1 i O_2. Podobnie zrobiono na rysunku 25. Informacja dotycząca dostępnych stref jest konieczna podczas testowania i oceny zaproponowanego rozwiązania, do określenia poprawności jego działania.

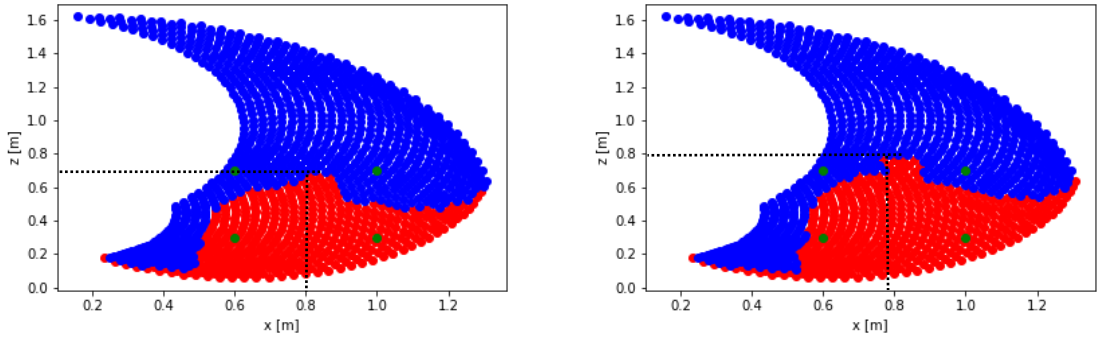
a) b)



Rysunek 24. Widok przekroju strefy roboczej manipulatora dla uszkodzonej osi *j*1. a) dla y ∈ (-0,2028, -0,1972) m, b) dla y ∈ (0,1972, 0,2028) m. Zielone punkty oznaczają pozycje do których przemieszczało się TCP robota, na niebiesko strefa bezkolizyjna, na czerwono strefa kolizyjna, $\alpha_w = 5°$

Rysunek 24 przedstawia przekroje strefy roboczej przy wystąpieniu awarii osi *j*1. Po uszkodzeniu tej osi osiągnięcie aż czterech położeń docelowych przez TCP robota jest niemożliwe. Położenia te znajdują się poza strefą roboczą robota.

a) b)



Rysunek 25. Widok przekroju strefy roboczej manipulatora dla uszkodzonej osi *j*0 i *j*4. a) dla y ∈ (-0,2028, -0,1972) m, b) dla y ∈ (0,1972, 0,2028) m. Zielone punkty oznaczają pozycje do których przemieszczało się TCP robota, na niebiesko strefa bezkolizyjna, na czerwono strefa kolizyjna, $\alpha_w = 5°$

Rysunek 25 przedstawia sytuacje, w której uszkodzeniu uległy dwie osie *j*0 oraz *j*4. Po uszkodzeniu tych osi, cztery pozycje są możliwe do osiągnięcia w sposób bezkolizyjny, a cztery pozostałe znajdują się w strefie roboczej, ale dotarcie do tych położeń było by możliwe jedynie po usunięciu przeszkód z obszaru roboczego. Punkty docelowe na wszystkich przekrojach znajdowały się w położeniu tj. płaszczyźnie o współrzędnej *y* równej 0,2 metra. Zakres przekroju ustawiono na plus minus 2,8 mm. Dobrano go na podstawie oceny wizualnej kilkunastu stref roboczych robota z uszkodzonymi dwoma osiami. w przypadku gdy przekrój strefy robocze wykonano dla wartości *y* wynoszącej 0,2 metra, to w wyznaczonej strefie znajdowało się tylko około od kilkunastu do kilkudziesięciu punktów i trudno było oszacować, czy punkt docelowy znajduje się w strefie roboczej czy w strefie kolizyjnej. Wartość przedziału wartości *y* zwiększano co 0,0002 m.

Zaprojektowany algorytm, przedstawiony w tym podrozdziale służył jedynie do oceny poprawności działania proponowanego systemu sterowania, tzn. do stwierdzenia, czy to że algorytm nie doprowadził TCP robota do punktu docelowego wynikało z tego, że ten punkt znajdował się poza strefą roboczą uszkodzonego robota, czy też dlatego, że algorytm nie był w stanie tego zrobić z innych powodów, np. jego niedoskonałości. Nie był on stosowany on-line do oceny tego, czy ruch jest w ogóle możliwy do wykonania, ponieważ musiał on wykonywać obliczenia i symulacje dla bardzo dużej liczby pozycji TCP robota i

w przypadku gdy wszystkie osie były sprawne a parametr α_w był równy 10°, to , liczba użytych w Algorytmie 2, koniecznych ustawień pozycji osi do sprawdzenia obszaru roboczego wynosi:

$$k = j_{5s} \cdot j_{4s} \cdot j_{3s} \cdot j_{2s} \cdot j_{1s} \cdot j_{0s} \tag{44}$$

gdzie: j_{5s} – liczba możliwych pozycji dla osi 5, j_{4s} – liczba możliwych pozycji dla osi 4, j_{3s} – liczba możliwych pozycji dla osi 3, j_{2s} – liczba możliwych pozycji dla osi 2, j_{1s} – liczba możliwych pozycji dla osi 1, j_{0s} – liczba możliwych pozycji dla osi 0.

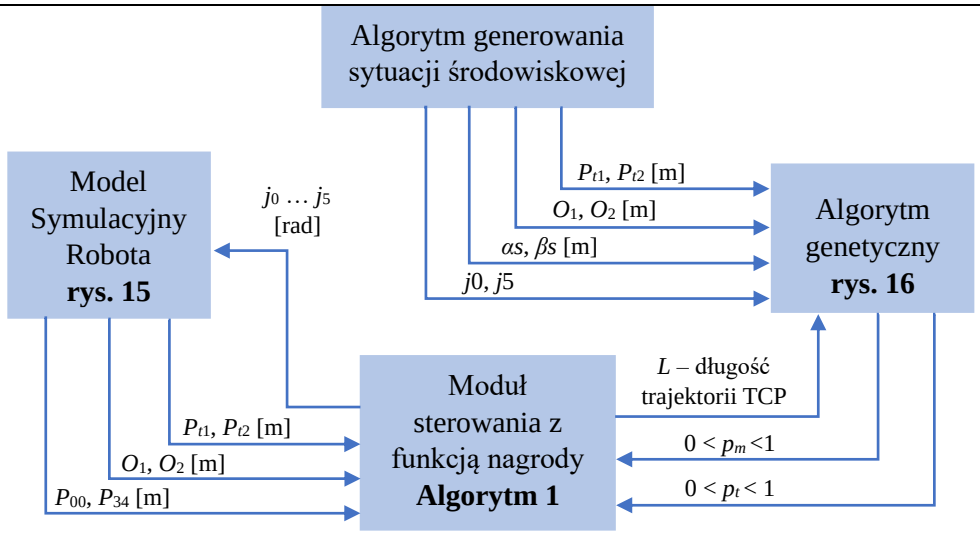
Gdy któraś z osi jest niesprawna, to liczba możliwych pozycji dla tej osi wynosi 1. Dla wszystkich aktywnych osi, gdy rozdzielczość (krok) przemieszczenia osi wynosi $\alpha_w = 10°$, liczba położeń TCP jaką wygeneruje symulacja na podstawie Algorytmu 2 wynosi aż 336 538 125. Wysłanie jednego zestawu położeń osi oraz odesłanie obliczonego przez MuJoCo położenia TCP manipulatora, zajmowało około 15 mikrosekund. Stąd czas potrzebny na obliczenie wszystkich wymaganych kombinacji wynosił by około 84 minuty dla jednej sytuacji środowiskowej. Tak długi okres czasu, konieczny do sprawdzenia czy w danej konfiguracji i sytuacji środowiskowej, robot może osiągnąć pozycję zadaną jest stanowczo zbyt długi, aby mógł być zastosowany w sterowaniu on-line.

5.2 Optymalizacja parametrów funkcji nagrody przez algorytm genetyczny

Jak wspomniano wyżej, do modelowania robota wybrany został silnik fizyki MuJoCo [72]. Silnik ten posiada interaktywną wizualizację 3D zbudowaną w oparciu o bibliotekę OpenGL, dzięki czemu możliwe jest przetestowanie działania algorytmu, przed zastosowaniem go do rzeczywistego robota. MuJoCo spełnia podstawowe wymagania, jakie zostały postawione przed środowiskiem, w którym można przeprowadzić testy proponowanych algorytmów sterowania. Punkty charakterystyczne wskazane na rys. 15, zostały wykorzystane w algorytmie sterowania, między innymi do obliczenia odległości ramion robota od przeszkód. Punkt P_{00} oznaczał aktualny punkt TCP. Punkt P_s oznaczał punkt startowy, punkt P_{t1} definiował pierwszą pozycję, do której TCP robota musiał się przemieścić. Punkt P_{t2} definiował cel, którym była druga zdefiniowana pozycja, do której TCP robota musiał się przemieścić. Wokół punktów P_{t1} i P_{t2}, na rysunku 15 występują kolorowe beżowe koła, obrazujące kule w przestrzeni 3D. Definiują one obszary tzw. dokładnego pozycjonowania tj. sfery o promieniu 20 mm, gdzie P_{t1} i P_{t2} są ich punktami środkowymi. w tych strefach, w celu uzyskania dokładnego pozycjonowania. TCP robota

poruszał się ze zmniejszonym krokiem β_s a tym samym z małą prędkością. w celu zwiększenia szybkości działania algorytmu do omijania przeszkód, do obliczania odległości poszczególnych ramion robota od przeszkód, uproszczono opis jego ramion do postaci przedstawionej na rys. 15, w której ramiona otoczone są przezroczystymi zielonymi „rurkami" z półkulami na końcach. Algorytm obliczał odległość $d_g(i)$ pomiędzy TCP a celem oraz odległość $d_e(i)$ pomiędzy punktami na ramionach robota a punktami na przeszkodach, korzystając ze wzorów (29) i (30). w celu wykrycia kolizji któregoś z ramion robota z przeszkodami, wybierano najmniejszą wartość spośród wszystkich wyznaczonych wartości $d(P_{am},O_{on})$. Jeśli była ona mniejsza od sumy promieni sfer ramienia i przeszkody system uznawał, że ramię może uderzyć w przeszkodę i zmieniał punkt na trajektorii ruchu na inny, aby nie nastąpiła kolizja i zatrzymanie robota. Średnica tych sztucznych rur została określona w taki sposób, aby poszczególne ramiona robota i przeszkody znajdowały się w całości wewnątrz takiej rury, co gwarantowało, że robot nie przemieści się zbyt blisko przeszkody. Rury te zostały przedstawione na rys. 15. Za pomocą przezroczystych, zielonych rur oznaczono strefę zajmowaną w przestrzeni przez robota. Strefy zajmowane przez przeszkody zaznaczono i opisano w algorytmie za pomocą przezroczystych czerwonych rur.

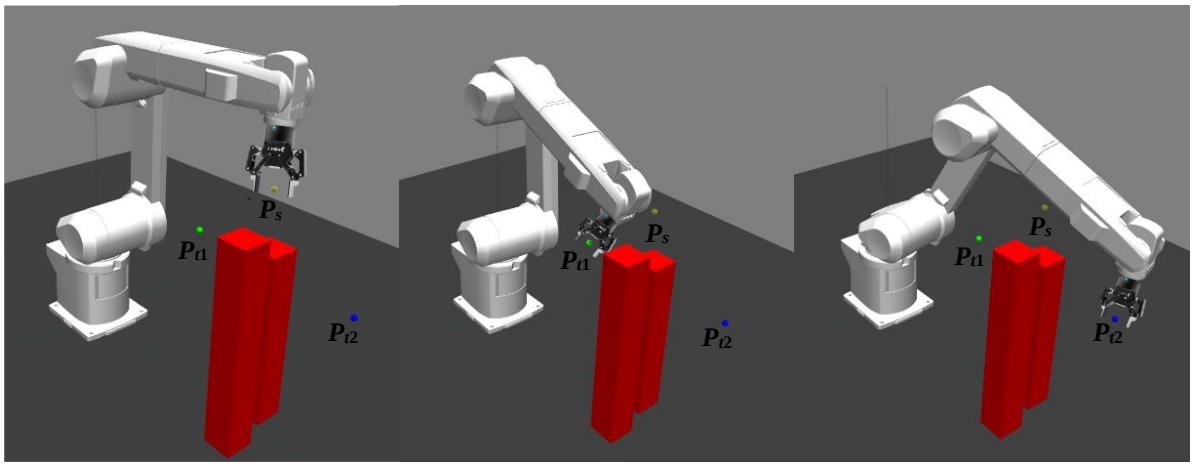
Jak napisano wyżej, ze względu na długi czas działania algorytmu genetycznego, proces optymalizacyjnych ograniczono do trzech epok. Do przeprowadzenia pierwszych badań algorytmu genetycznego zaprojektowano podsystem, którego schemat blokowy został zaprezentowany na rysunku 26. Jego zadaniem było wyznaczenie najlepszych osobników, zawierających parametry p_m oraz p_t funkcji nagrody. Wyniki działania algorytmu sterowania robotem, w postaci długości trajektorii, zapisywane były w pliku tekstowym. w pliku tym zapisywana były: „sytuacja środowiskowa", 60 osobników (3 populacje po 20 osobników), które opisywały parametry funkcji nagrody oraz długość trajektorii dla każdego osobnika. Dzięki temu po sprawdzeniu trzech populacji AG, można było sprawdzić które parametry p_m oraz p_t na przestrzeni populacji dały najlepsze wyniki, czyli najkrótszą trajektorię ruchu.



Rysunek 26. Schemat układu optymalizującego parametry funkcji nagrody

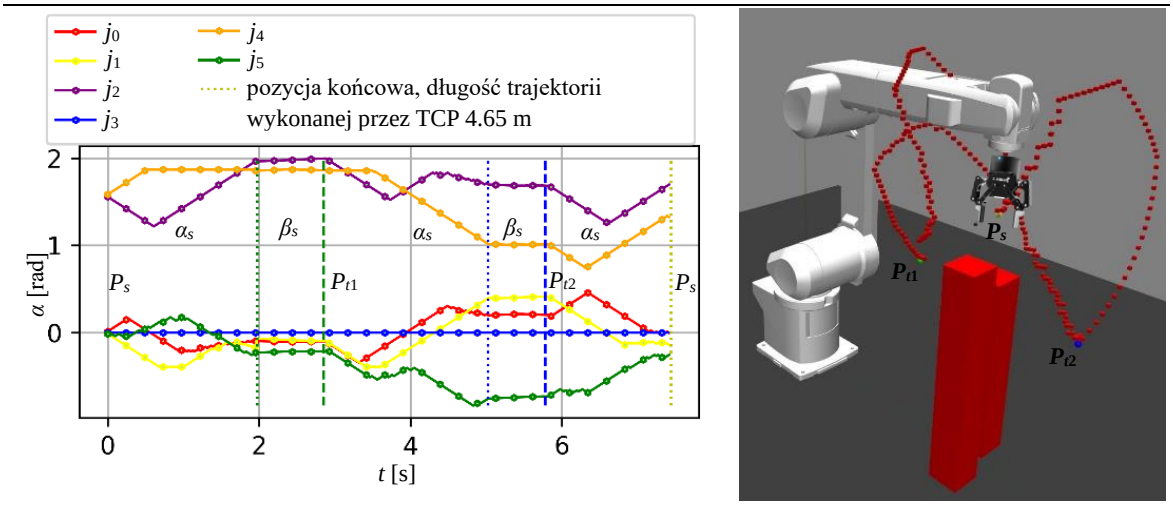
Podczas pierwszych badań symulacyjnych sprawdzono, czy i w jakim stopniu AG wyznaczał lepsze wartości parametrów p_m i p_t, po każdej wykonanej epoce. Wyniki tych badań przedstawiono na rysunkach od 27 do 30. Uzyskano je dla sytuacji środowiskowej, w której oś j_3 robota była uszkodzona. Na rysunku 27 przedstawiono 3 obrazy robota, którego TCP znajdowało się odpowiednio w punktach P_s, P_{t1} i P_{t2}. w momencie gdy TCP robota znajdowało się w punkcie startowym P_s, jego osie były ustawione w następująco: $j_0 = j_1 = j_3 = j_5 = 0°$, $j_2 = 90°$, $j_4 = 90°$. Na rysunkach od 28 do 30 pokazano jak wyglądały przebiegi zmian położenia poszczególnych osi i jakie wyniki tzn. trajektorie, oznaczone za pomocą krzywych kropkowych, uzyskiwał algorytm genetyczny w kolejnych populacjach. Przesunięcia kątowe poszczególnych osi w czasie pokazano na rysunkach 28a, 29a i 30a. z kolei na rysunkach 28b, 29b i 30b przedstawiono trajektorie ruchu TCP robota dla tych samych punktów: P_s, P_{t1} i P_{t2}. Trajektorie te zostały wyznaczone z wykorzystaniem parametrów p_m i p_t, wygenerowanych przez AG kolejno po pierwszej, drugiej i trzeciej populacji. Dla najlepszego osobnika z pierwszej populacji, długość trajektorii wykonanej przez TCP robota wynosiła około 4,65 m. Do wygenerowania takiej trajektorii robot musiał wykonać 305 kroków. Parametry funkcji nagrody znalezione przez AG po wygenerowaniu pierwszej populacji, wynosiły $p_m = 0{,}219$ oraz $p_t = 0{,}019$. Osobniki z drugiej populacji AG wykorzystane w algorytmie umożliwiły wygenerowanie trajektorii ruchu TCP robota o długości ok. 2,97 m, która składała się z 191 kroków. Parametry funkcji nagrody znalezione dla tej populacji to $p_m = 0{,}39$, $p_t = 0{,}037$. Osobnik (chromosom) wyznaczony w trzeciej populacji, pozwolił na uzyskanie najkrótszej trajektorii, tj. o długości ok. 2,14 m, której dotarcie do punktu docelowego zajęło 144 kroki. Parametry funkcji nagrody

znalezione dla tej populacji to: p_m = 0,715, p_t =0,090. Porównując parametry funkcji uzyskane przez AG można zauważyć, że dla kolejnych populacji, wartości tych parametrów rosły i tym samym skracały trajektorię i liczbę wykonanych kroków. w praktyce oznaczało to, że wraz ze wzrostem liczby wykonanych epok, przy obliczaniu nagrody malało znaczenie odległości robota od przeszkód, a rosło znaczenie odległości TCP od punktu docelowego. Powodowało to, że w końcowym etapie ruchu, robot poruszał się bardziej bezpośrednio tj, po prostej do punktu docelowego. Jednak takie zmiany parametrów jak opisane powyżej, występowały tylko w niektórych przypadkach. Wiele było również sytuacji, w których już w pierwszej populacji znaleziono osobnika, który pozostał najlepszy do końca procesu optymalizacji wykonywanego przez AG.



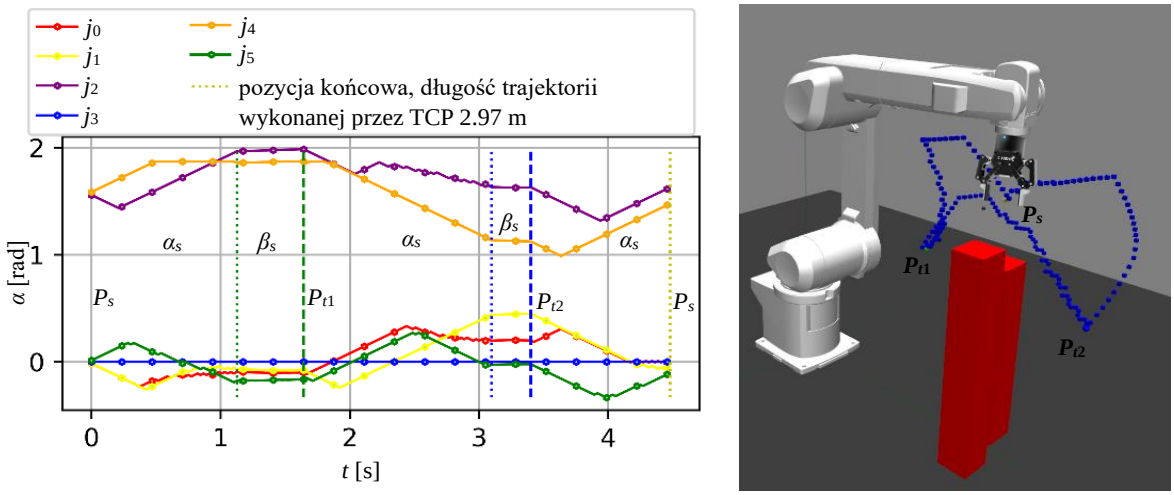
a) b) c)

Rysunek 27. Przedstawienie a) pozycji poszczególnych osi robota w pozycji startowej P_s, b) pozycji poszczególnych osi robota w pierwszej pozycji docelowej P_{t1}, c) pozycji poszczególnych osi robota w drugiej pozycji docelowej P_{t2}



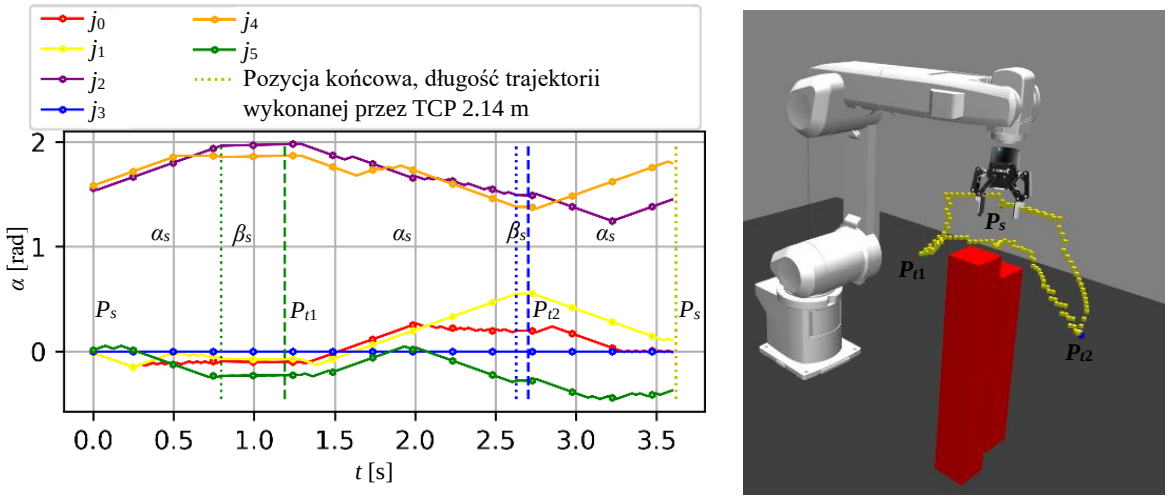
a) b)

Rysunek 28. Przedstawienie działania algorytmu tj. trajektorii dla najlepszego osobnika populacji pierwszej AG: a) przedstawienie położeń kątowych poszczególnych osi w czasie, b) przedstawienie wygenerowanej przez TCP trajektorii



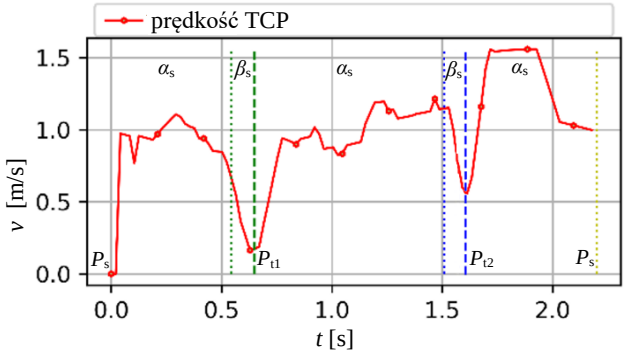
a) b)

Rysunek 29. Przedstawienie działania algorytmu dla najlepszego osobnika populacji drugiej AG. a) Przedstawienie położeń kątowych poszczególnych osi w czasie. b) Przedstawienie wygenerowanej przez TCP trajektorii



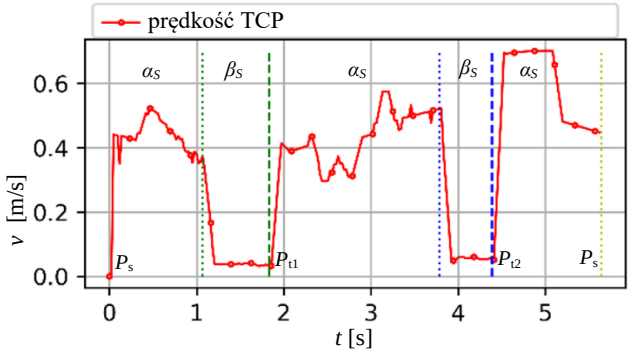
Rysunek 30. Przedstawienie działania algorytmu dla najlepszego osobnika populacji trzeciej AG. a) przedstawienie położeń kątowych poszczególnych osi w czasie. b) przedstawienie wygenerowanej przez TCP trajektorii

W kolejnym kroku, sprawdzono możliwości algorytmu pod względem szybkości ruchu TCP robota oraz dokładności jego pozycjonowania. Na rysunku 31 przedstawiono zmiany prędkości TCP robota podczas ruchu z punktu P_s do P_{t1}, następnie do P_{t2} i z powrotem do P_s. Wynik ten uzyskano, gdy kąt obrotu dla kroku szybkiego α_s był równy ±1,0°stopień, natomiast kąt dla kroku dokładnego β_s wynosił ±0,2 stopnia. Pozwoliło to na osiągnięcie średniej prędkości v wynoszącej 1 m/s dla kroku szybkiego, czyli gdy TCP robota znajdował się w odległości większej niż 20 mm od pozycji docelowej. Natomiast, gdy odległość ta była mniejsza niż 20mm, sterownik robota przechodził na sterowanie precyzyjne tj. na ruch wolny, co w przypadku zaproponowanego algorytmu skutkowało prędkością ruchu v wynoszącą 0,4 m/. Dokładność pozycjonowania do punktu docelowego była lepsza niż ±2 mm.



Rysunek 31. Zmiany prędkości dla niższej rozdzielczości tj. kroku szybkiego α_s = 1,0 stopnia i wysokiej rozdzielczości tj. dokładnego pozycjonowania β_s =0,2 stopnia

Gdy α_s ustawiono na ±0,5°, a β_s na ±0,05°, to prędkość ruchu v spadła do wartości odpowiednio około 0,5 m/s i 0,1 m/s, co widać na rysunku 32. Dokładność pozycjonowania, jaką uzyskano przy tych parametrach kroku, była lepsza niż ±0,5 mm. Takie prędkości ruchu są w pełni akceptowalne w większości zadań stawianych manipulatorom. w niektórych przypadkach, problemem może być dokładność pozycjonowania TCP, na którą wpływa parametr β_s. Przy ustawieniu dużej rozdzielczości ruchu w fazie precyzyjnego sterowania, czyli w powolnym zbliżaniu się do zadanej pozycji, dokładności pozycjonowania TCP mieściły się w zakresie od ±0,3 mm do ±0,5 mm.

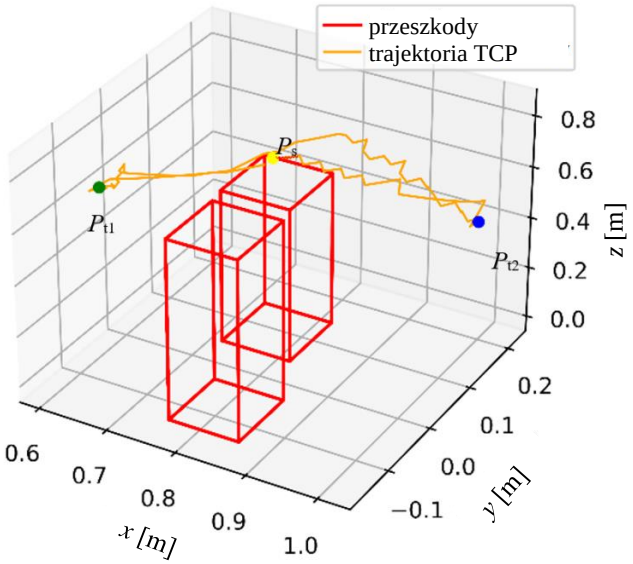


Rysunek 32. Zmiany prędkości dla α_s = 0,5 stopnia i dla β_s = 0,05 stopnia

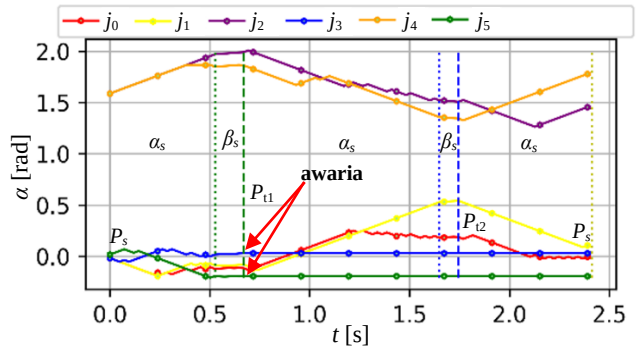
W praktycznych zastosowaniach w przypadku precyzyjnych zadań manipulacyjnych, taka dokładność pozycjonowania może być niewystarczająca. Możliwe jest ustawienie większej rozdzielczości, czyli mniejszego minimalnego kroku jak β_s na ±0,02°, ale wpłynie to na znaczące zmniejszenie prędkości ruchu przy dochodzeniu do punktu docelowego. Ponadto, ustawienie większej rozdzielczości ruchu, zwiększy wymagania dotyczące dokładności modelu symulacyjnego robota, co z kolei wydłuży czas reakcji algorytmu i zmniejszy prędkość robota rzeczywistego. Tym samym wydłużeniu ulegnie również czas detekcji uszkodzeń osi itd.

W ramach badań wykonano również takie same testy ale przy wystąpieniu awarii więcej niż jednej osi w czasie ruchu. Po wykryciu awarii osi blokowano ją, a parametry p_m i p_t funkcji nagrody pobierano z tabeli wygenerowanej przez AG. Zarejestrowane wyniki dla takich sytuacji przedstawiono na rysunkach 33, 34 i 35. Na rysunku 33 kolorem pomarańczowym przedstawiono trajektorię ruchu TCP robota. Widać, że w momencie gdy TCP przemieszcza się do pozycji docelowej P_{t1} ruch jest wykonywany prawie po linii prostej. w przypadku ruchu do pozycji P_{t2} oraz od P_{t2} do P_s, ruch jest zygzakowaty. Wynika to z faktu, iż w momencie gdy TCP osiągnęło pozycje P_{t1}, awarii uległy osie j_3 oraz j_5, co

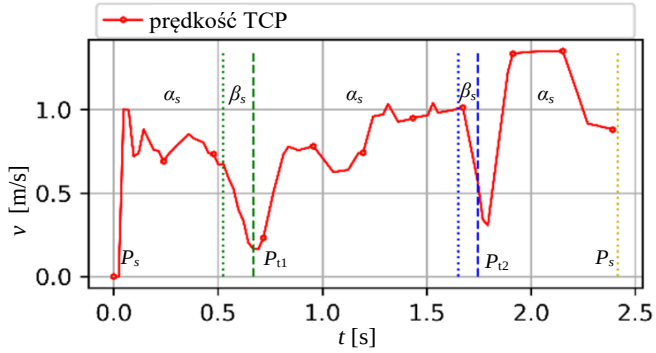
na rysunku 34 zaznaczono czerwonymi strzałkami. Rysunek 35 przedstawia zmiany prędkości ruchu w trakcie przemieszczania TCP.



Rysunek 33. Wygenerowana trajektoria dla awarii osi 3 i 5



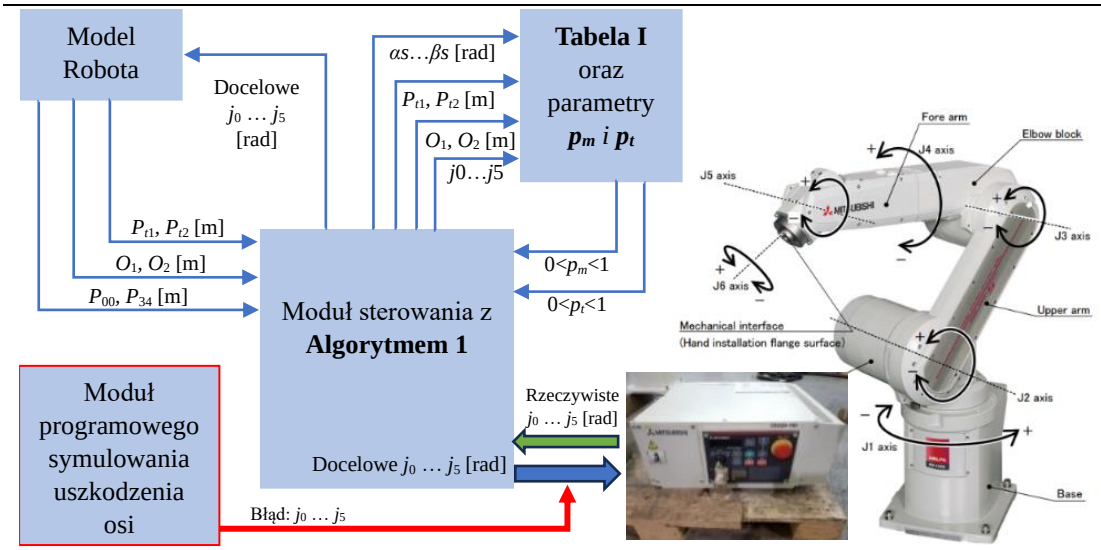
Rysunek 34. Zmiany położenia kątowego poszczególnych osi w czasie (ruch zgrubny i dokładny)



Rysunek 35. Średnie wartości prędkości i błąd pozycji dla dolnego zakresu rozdzielczości 1 stopnia, 0,2 stopnia

Podobne do przedstawionych wyżej testów symulacyjnych, przeprowadzono również na rzeczywistym robocie przemysłowym. Robot ten nie był przystosowany do pracy

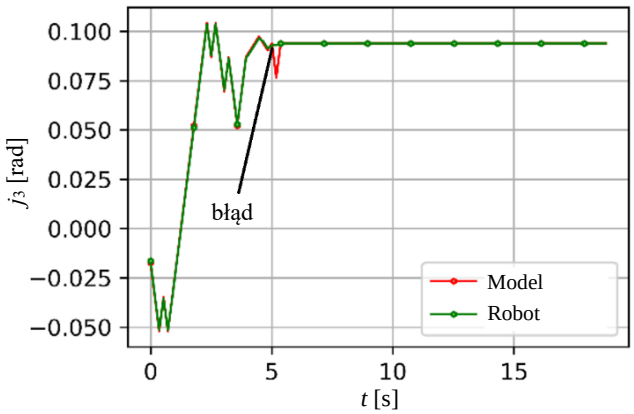
w przypadku wystąpienia awarii osi i po wykryciu awarii, sterownik robota sygnalizował błąd i dalsza praca robota nie była możliwa bez usunięcia awarii. Taka sytuacja miała miejsce jeśli np. odłączono enkoder w którejś z osi lub odcięto zasilanie silnika. Dlatego, w trakcie badań rzeczywistego robota, uszkodzenie osi było sztucznie wywoływane za pomocą opracowanego przez autora oprogramowania, które blokowało w zadanym przez osobę wykonującą badania momencie, wybraną oś robota. w celu zasymulowania uszkodzenia wybranej osi, opracowano moduł programistyczny o nazwie "Programowe symulowanie uszkodzenia osi", który przejmował kontrolę nad wysyłaniem aktualnej nowej pozycji kątowej danej osi, lub nie symulując tym samym jej uszkodzenie. Schemat blokowy układu sterowania przedstawiono na rysunku 36. Badania na rzeczywistym robocie przeprowadzono w podobny sposób jak badania symulacyjne. w przypadku badań symulacyjnych algorytm wysyłał w każdym kroku pozycje zadane tylko do modelu symulacyjnego a w badaniach rzeczywistego robota, pozycje docelowe każdej osi były wysyłane zarówno do modelu robota w środowisku MuJoCo, jak i do robota rzeczywistego. Identyfikacja awarii dowolnej osi przez algorytm odbywała się poprzez porównanie położenia kątowego każdej osi robota, uzyskiwanego w symulacji z położeniem kątowym wszystkich osi, odczytanym z enkoderów zamontowanych w przegubach robota rzeczywistego. Parametr określający dopuszczalny błąd między aktualnymi pozycjami z symulacji i z robota, dla wszystkich pojedynczych osi został ustawiony na 0,001 rad (0,057 stopnia). Takie ustawienie dokładności pozycjonowania osi oznaczało, że gdy błąd kątowy w danej osi był mniejszy od ustawionej wartości, sterownik "uznawał", że pozycja tej osi w danym kroku, została uzyskana i oś działa poprawnie. Po uzyskaniu zadanej pozycji, robot był gotowy do przejścia do kolejnego kroku, w którym algorytm wysyłał mu nowe współrzędne (przemieszczenia) dla wszystkich osi.



Rysunek 36. Schemat przedstawiający system sterowania robotem wykorzystywany podczas testów prawdziwego robota

Operator mógł określić moment wystąpienia awarii przez Moduł programowego symulowania uszkodzenia. w tym przypadku, zamiast aktualnej pozycji, moduł ten przesyłał do rzeczywistego robota stale aktualną wartość pozycji kątowej danej osi. Wysłanie tej samej pozycji osi w dwóch kolejnych krokach powodowało, że różnica położeń kątowych danych osi, była większa niż założone 0,001 radiana i algorytm wykrywał uszkodzenie. Po wykryciu awarii program nadzorujący pracę ustawiał kąt danej osi w modelu na kąt odczytany z robota rzeczywistego, a następnie blokował daną oś w modelu robota. Kolejne kroki algorytmu, czyli sterowanie robotem, odbywały się z wyłączeniem uszkodzonej osi przez opracowany w ramach niniejszej pracy, algorytm odporny na błędy.

Położenia kątowe osi j_3 robota rzeczywistego i jego modelu symulacyjnego w czasie przedstawiono na rysunku 37. Zaznaczono na nim moment wystąpienia błędu w postaci uszkodzenia osi, który nastąpił po około 5 sekundach ruchu. Na zarejestrowanych przebiegach można zauważyć wystąpienie różnicy między sygnałami położenia z modelu (przebieg czerwony) i z rzeczywistego robota (przebieg zielony).



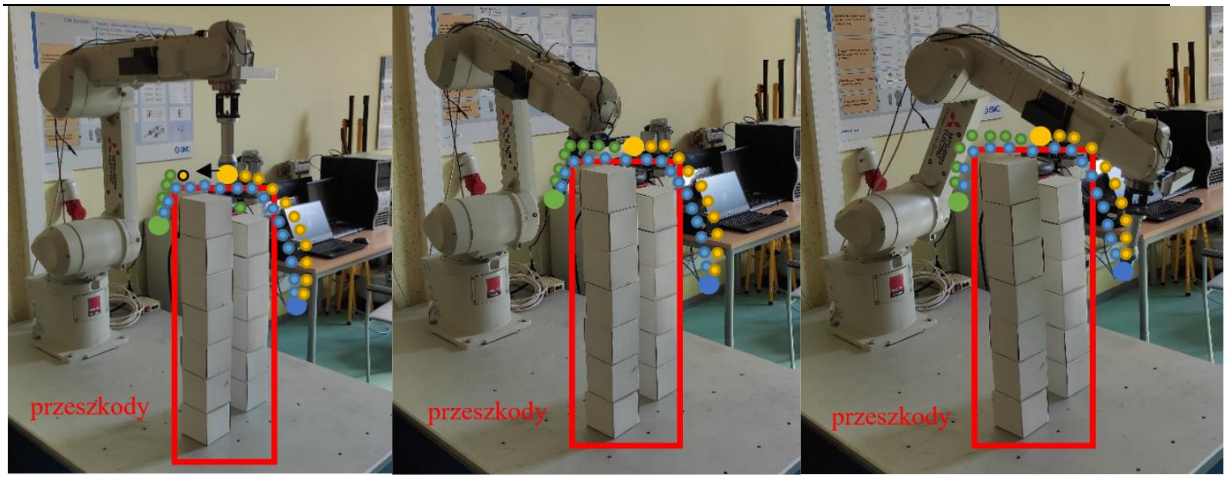
Rysunek 37. Pozycje osi robota rzeczywistego i jego modelu symulacyjnego w przypadku awarii osi j_3

Po wykryciu awarii osi zmniejszeniu ulegała przestrzeń czynności, które robot może wykonać w danym kroku. Opracowany algorytm był w stanie na testowanym sprzęcie tj. komputerze, generować kolejne akcje (ruchy, kroki) w czasie krótszym niż 20ms, co w symulacji dawało prędkości TCP robota, przekraczające 1 m/s dla szybkiego ruchu (sterowania).

Norma ISO 10218-1 mówi, że w przypadku robotów, w których nie ma dodatkowych zabezpieczeń w postaci np. systemu monitorującego pozycję człowieka, maksymalna prędkość robota współpracującego z człowiekiem może wynosić 250 mm/s. Stanowisko badawcze nie było w żaden sposób zabezpieczone, tzn. nie było np. kurtyn bezpieczeństwa, dlatego maksymalna prędkość ruchu została ograniczona do 250 mm/s.

Przy sterowaniu robotem, algorytm działał tak samo jak na modelu, z tą różnicą, że pozycje osi były również wysyłane do robota rzeczywistego, a uszkodzenie osi nie było określane przez operatora, lecz przez system zakłóceń ramki komunikacyjnej i moduł odczytywania aktualnej pozycji robota i porównywania jej z pozycją modelu robota. w badaniach przeprowadzonych i opisanych w niniejszej pracy, testowano rozwiązanie zadania podobnego do "pick and place", czyli chwytania, przenoszenia i odkładania obiektu. Nie wykonywano jednak chwytania obiektu, a jedynie pozycjonowanie TCP robota w określonych punktach tj. P_{t1} i P_{t2}.

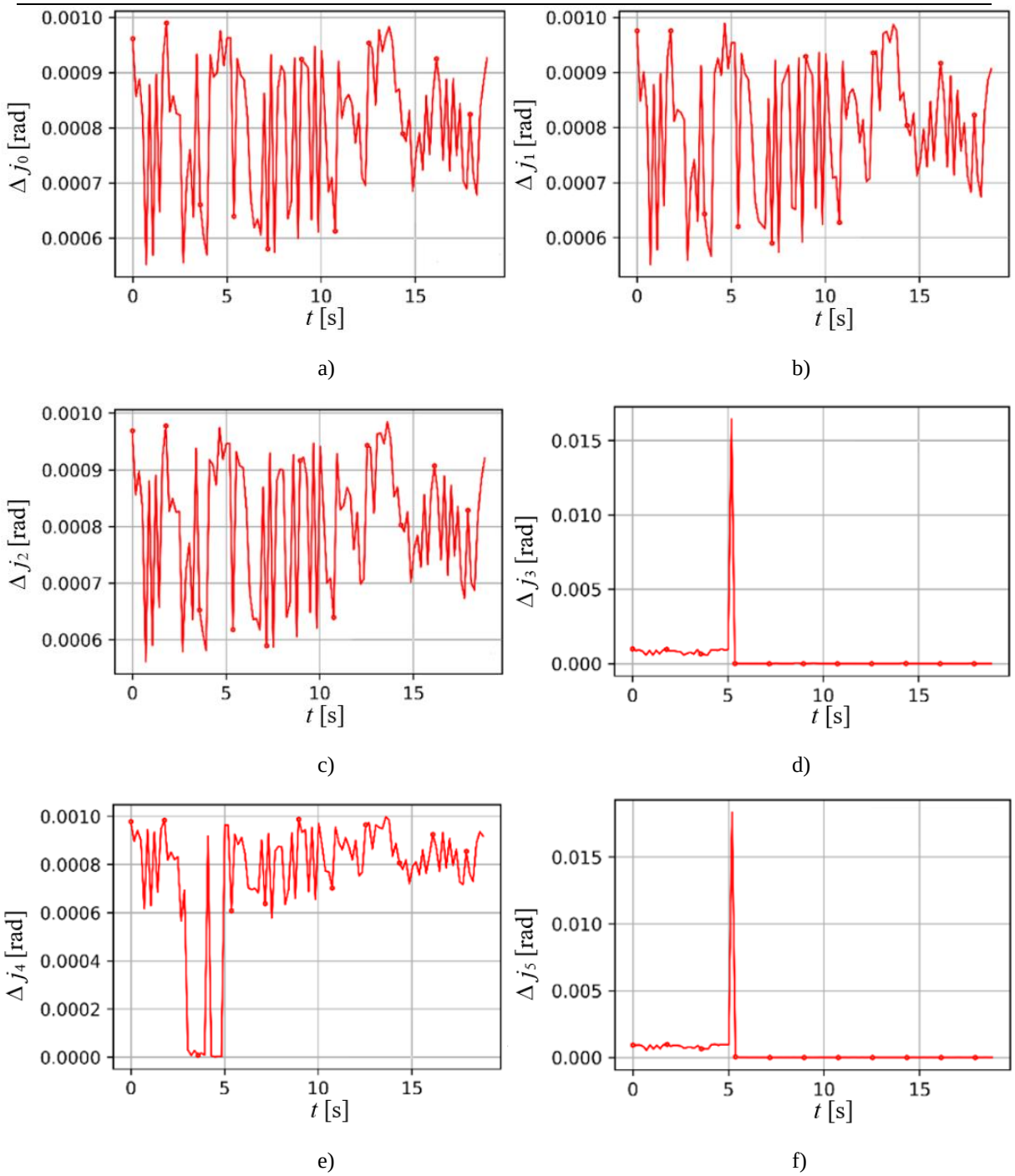
Badania wykonane w symulacji, z których przebiegi pokazano na rysunkach 33, 34, 35, zostały przeprowadzone również na robocie Mitsubishi RV-12SDL, co pokazano na rys. 38.



a) b) c)

Rysunek 38. Reprodukcja sytuacji przedstawionej na rysunkach 31, 32, 33. Rzeczywiste pozycje robota: a) pozycja startowa, b) pierwsza pozycja docelowa, c) druga pozycja docelowa

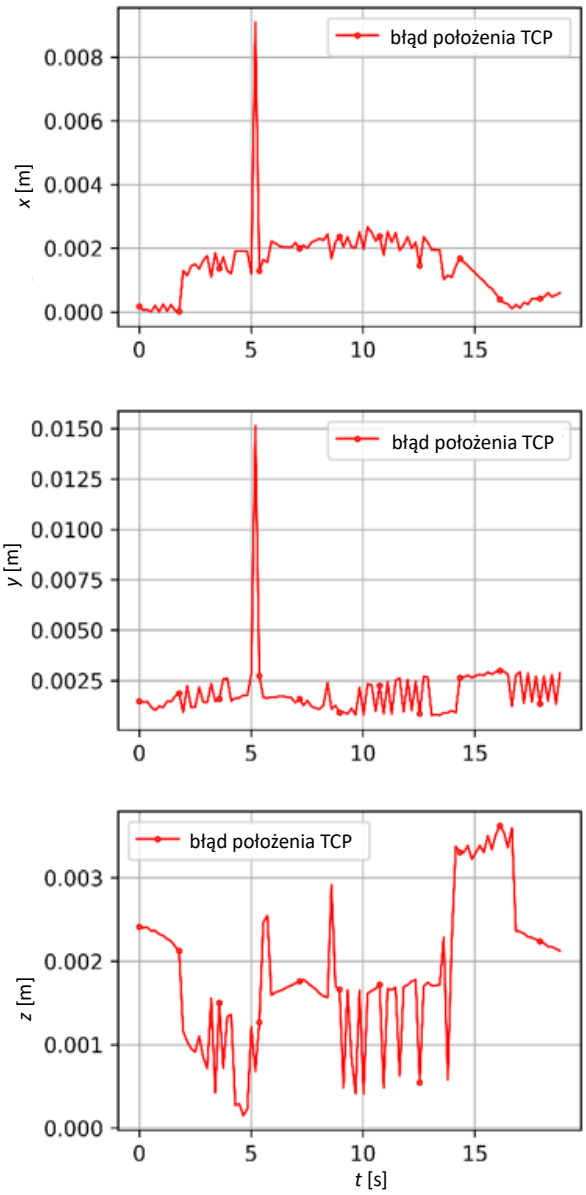
W tym przypadku uszkodzone były osie j_3 i j_5. Jak wspomniano wcześniej, zaimplementowany w sterowniku robota, algorytm wykrywania awarii osi był ustawiony na „dopuszczenie" błędu każdej osi mniejszego niż 0,0010 radiana. Gdy następowało przekroczenie tej dopuszczalnej wartości błędu, to system sterowania „uznawał", że w tej osi wystąpił błąd tj. awaria. w badaniach doświadczalnych, błąd osi był sztucznie wywoływany przez oprogramowanie, które blokowało ruch rzeczywistej, wybranej osi, a dokładnie ruch silnika. Na rysunku 39 przedstawiono przebiegi zmian błędów położenia kątowego wszystkich 6-ciu osi podczas ruchu. Dla osi, które nie uległy awarii, błąd ten nie przekraczał wartości „dopuszczalnej", ustawionej na sterowniku robota rzeczywistego. Jak to widać na rysunku 39d oraz 39f, w piątej sekundzie trwania ruchu, jednocześnie wystąpiły awarie osi numer 3 i 5. Na rysunku 40 przedstawiono błędy TCP rzeczywistego robota w osiach *x*, *y*, *z*, względem pozycji TCP robota zamodelowanego w MuJoCo. Podobnie jak poprzednim, widać, że w piątej (t = 5s) sekundzie trwania ruchu, błędy dla współrzędnych wyniosły odpowiednio *x* około 9 mm, dla *y* około 15 mm i dla z około 1 mm. Wyniki te pokazują, iż w momencie awarii wyżej wymienionych osi robota rzeczywistego jego TCP znajdowało się w odległości kilkunastu mm od położenia TCP robota w symulacji. Przed awariami, różnice położeń TCP robota i jego modelu nie przekraczały 4 mm dla współrzędnych (osi) oraz 5 mm dla odległości w przestrzeni TCP robota rzeczywistego od TCP modelu symulacyjnego robota.



Rysunek 39. Zarejestrowane w trakcie symulacji przebiegi zmian błędów położenia kątowego wszystkich osi

Opisane wyżej, przeprowadzone badania symulacyjne oraz doświadczalne, potwierdziły skuteczność zaproponowanej metody i zaimplementowanego algorytmu, w zadaniu optymalizacji parametrów funkcji nagrody, przez AG oraz możliwości pracy algorytmu na danych stabelaryzowanych, w trakcie wcześniejszych prób. w omawianym przypadku, w procesie nie stosowano SSN. Algorytm działał jedynie w przypadku wykonywania ruchu i pozycjonowania w „sytuacjach środowiskowych", rozwiązywanych wcześniej

przez AG, tzn. na etapie opracowywania programu sterującego. Program ten był w stanie generować trajektorię ruchu w taki sposób, aby możliwe było ominięcie przeszkód. Potwierdzają to wykonane badania, których wyniki pokazano na rysunkach 28, 29, 30.



Rysunek 40. Błędy położenia TCP robota rzeczywistego względem położenia TCP modelu symulacyjnego robota dla osiach: *x*, *y*, *z*

5.3 Badania algorytmu z kryterium najkrótszej możliwej trajektorii

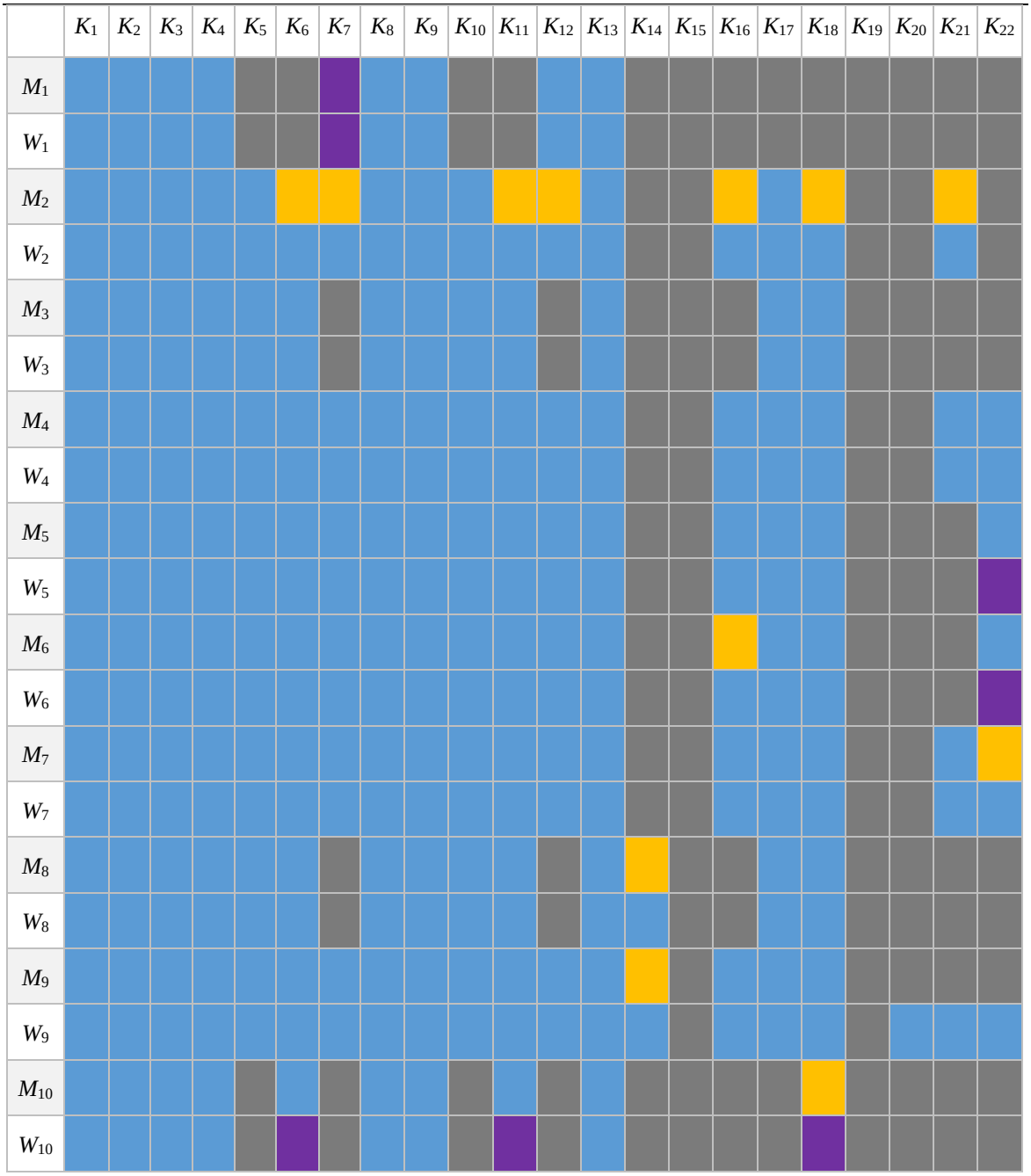
W poprzednim podrozdziale pokazano skuteczność optymalizacji przez algorytm genetyczny parametrów funkcji nagrody oraz poprawność pracy programu sterującego robotem metodą FTC, na danych stabelaryzowanych wcześniej. Jednak w praktyce algorytm ten funkcjonował poprawnie tylko w przypadku sytuacji, które zostały wcześniej

rozwiązane przez AG, a dane dotyczące sytuacji środowiskowej oraz odpowiadającym im parametrom p_m i p_t funkcji nagrody, zostały ze sobą powiązane i stabelaryzowane. Aby zapewnić poprawne działanie algorytmu dla wszystkich możliwych sytuacji środowiskowych konieczne było opracowanie dodatkowego algorytmu, który byłby w stanie generować dla nich, optymalne albo zbliżone do optymalnych (quasi-optymalne) parametry funkcji nagrody. w prezentowanym poniżej rozwiązaniu, postanowiono użyć stabelaryzowanych danych do nauki SSN pokazanej na rys. 17, która po nauczeniu będzie generowała takie quasi-optymalne parametry p_m i p_t funkcji nagrody dla dowolnej sytuacji środowiskowej. w pierwszym kroku badano działania algorytmu genetycznego i SSN w celu sprawdzenia zdolności do rozwiązania problemu znalezienia optymalnych parametrów p_m i p_t. w tabeli III, zestawiono wszystkie sytuacje, które zostały sprawdzone za pomocą algorytmu opisanego w rozdziale 5.1. Określono i opisano w niej parametry wykorzystywane w trakcie badań. Zostały one wykorzystane do zobrazowania uzyskanych wyników na rysunku 41.

TABELA III

OPIS PARAMETRÓW WYKORZYSTANYCH NA RYSUNKU 41

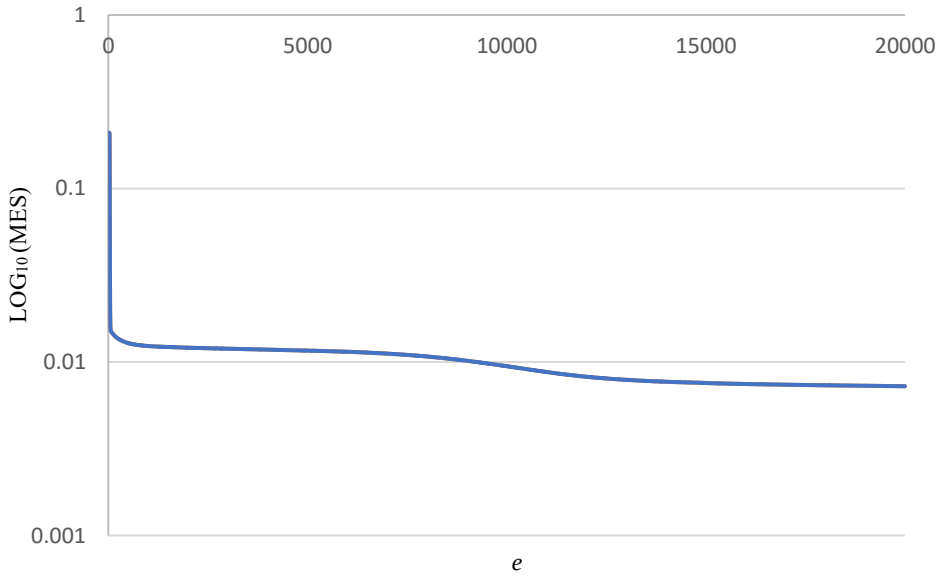
Parametr	**Opis**
K	wszystkie osie sprawne
K_2	uszkodzona oś *j5*
K_3	uszkodzona oś *j4*
K_4	uszkodzona oś *j3*
K_5	uszkodzona oś *j2*
K_6	uszkodzona oś *j1*
K_7	uszkodzona oś *j0*
K_8	uszkodzona oś *j5* i *j4*
K_9	uszkodzona oś *j5* i *j3*
K_{10}	uszkodzona oś *j5* i *j2*
K_{11}	uszkodzona oś *j5* i *j1*
K_{12}	uszkodzona oś *j5* i *j0*
K_{13}	uszkodzona oś *j4* i *j3*
K_{14}	uszkodzona oś *j4* i *j2*
K_{15}	uszkodzona oś *j4* i *j1*
K_{16}	uszkodzona oś *j4* i *j0*
K_{17}	uszkodzona oś *j3* i *j2*
K_{18}	uszkodzona oś *j3* i *j1*
K_{19}	uszkodzona oś *j3* i *j0*
K_{20}	uszkodzona oś *j2* i *j1*
K_{21}	uszkodzona oś *j2* i *j0*
K_{22}	uszkodzona oś *j1* i *j0*
M_r	dane w postaci pozycji przeszkód: O_1, O_2 i pozycji docelowych: P_{t1}, P_{t2}, dla kroku α_s gdzie: r licznik kolejnych pozycji
W_r	dane w postaci pozycji przeszkód: O_1, O_2 i pozycji docelowych: P_{t1}, P_{t2}, dla kroku β_s gdzie: r licznik kolejnych pozycji



Rysunek 41. Zobrazowanie skuteczności znalezienia rozwiązania przez algorytm genetyczny oraz sztuczną sieć neuronową

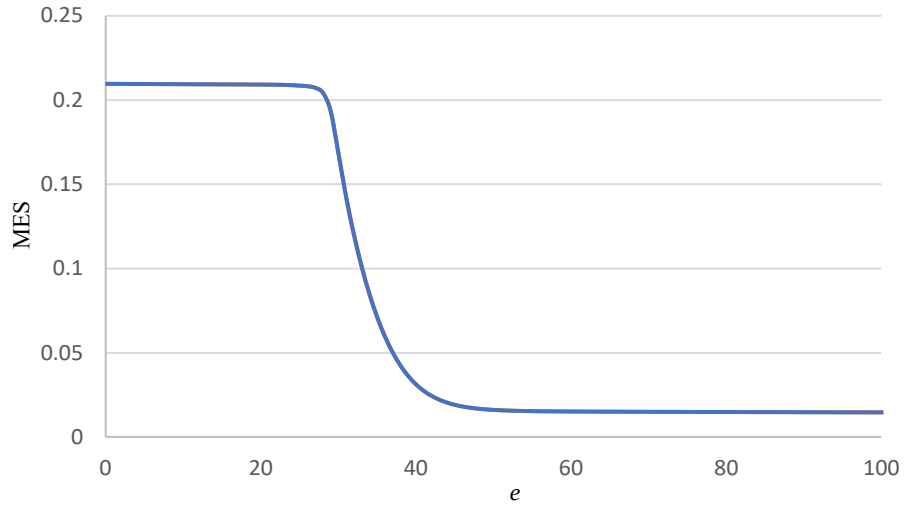
Zastosowano 4 kolory. Kolorem niebieskim oznaczono „sytuacje środowiskowe", rozwiązane przez algorytm genetyczny. Na szaro przedstawiono „sytuacje środowiskowe", które nie zostały rozwiązane przez algorytm genetyczny, dlatego że punkty docelowe nie były osiągalne przez TCP robota, to znaczy, że znajdowały się poza obszarem roboczym robota z uszkodzonymi osiami. Kolorem żółtym oznaczono „sytuacje środowiskowe", które były możliwe do osiągnięcia przez TCP robota bez kolizji, ale algorytm genetyczny nie znalazł rozwiązania. Algorytm podany w rozdziale 5.1 pozwalał na sprawdzenie, czy dana pozycja jest możliwa do osiągnięcia przez robota w sposób bezkolizyjny. Na

fioletowo zaznaczono „sytuacje środowiskowe”, dla których algorytm genetyczny znalazł rozwiązanie, ale wyuczona sztuczna sieć neuronowa, która wykorzystała wyznaczone przez AG parametry, nie była w stanie „doprowadzić” robota do punktu docelowego. Jak wynika z tabeli na rysunku 41, dla 306 „sytuacji środowiskowych” punkty docelowe znajdowały się w strefie działania uszkodzonego robota i teoretycznie, algorytm genetyczny powinien znaleźć rozwiązanie, jednak AG znalazł rozwiązanie dla 293 „sytuacji środowiskowych”. Wynik ten pozwala stwierdzić, iż skuteczność dobierania parametrów funkcji nagrody przez AG, była całkiem dobra i wyniosła 95,7%. Dane dla rozwiązanych 293. sytuacji posłużyły do nauki SSN. Proces uczenia, określany tutaj przez LOG_{10}(MES) czyli logarytm dziesiętny błędu średniokwadratowego dla sztucznej sieci neuronowej został przedstawiony na rysunku 42.



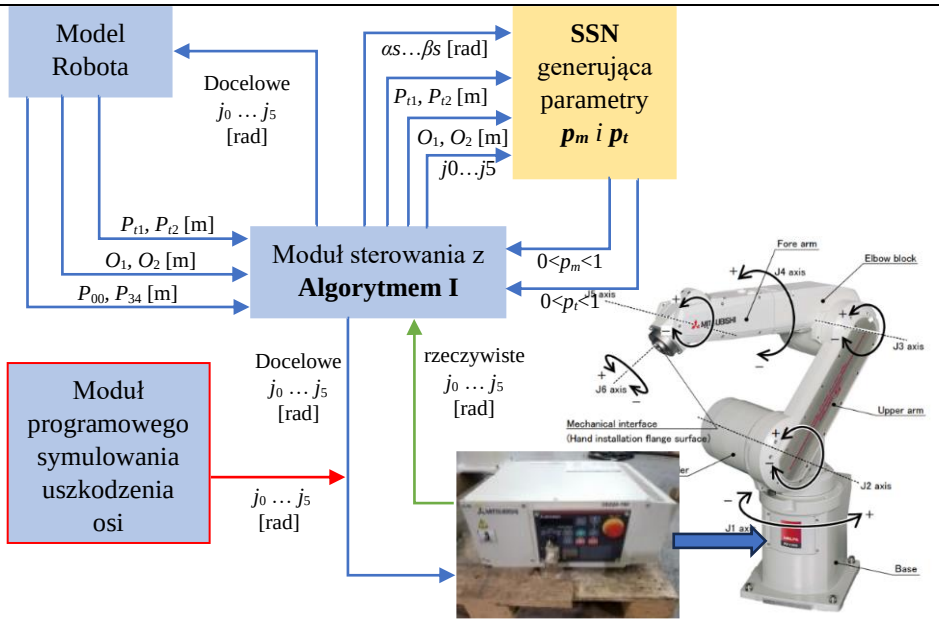
Rysunek 42. Przebieg procesu uczenia sztucznej sieci neuronowej, gdzie MES to błąd średniokwadratowy sieci

Ze względu na niewielką liczbę danych uczących sieć, to znaczy 293 „sytuacje środowiskowe”, którym odpowiadało tyle samo par parametrów p_m i p_t, jedna epoka procesu uczenia trwała około 100 ms. Dzięki temu można było ustawić ograniczenie liczby epok na 20 000. Czas nauki w tym przypadku wyniósł nieco ponad 30 minut. Na rysunku 43 przedstawiono proces uczenia dla pierwszych 100 epok. Widać, że już po 30-stu epokach sieć jest już nauczona, a po 50-tej epoce widać znaczne spowolnienie procesu uczenia. Jednak w przypadku 50-tej epoki, błąd średniokwadratowy wyniósł 0,01613, natomiast po 20000 epokach uczących, błąd ten wyniósł 0,00724.

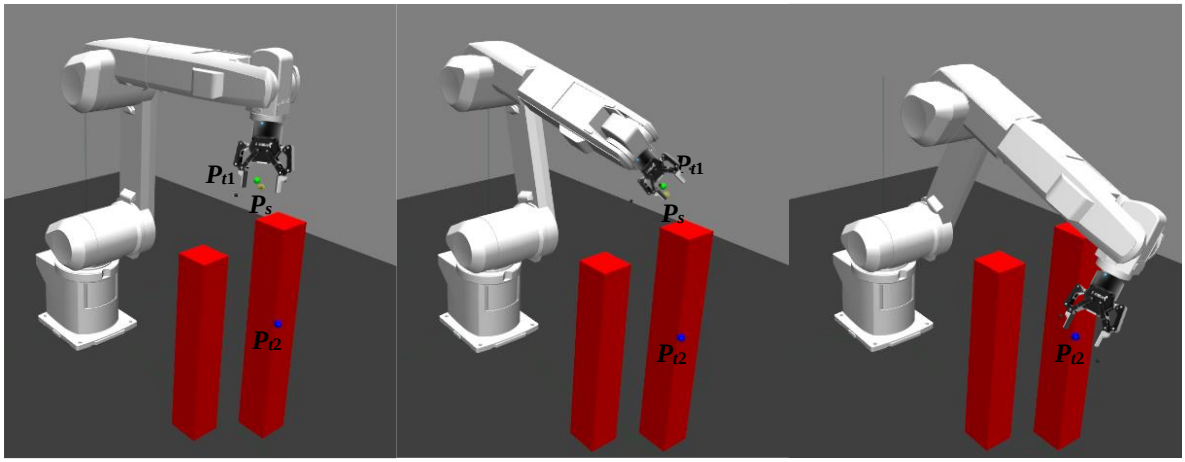


Rysunek 43. Przebieg procesu uczenia sztucznej sieci neuronowej dla pierwszych 100 epok

Osiągnięty przez algorytm uczenia sztucznej sieci neronowej błąd średniokwadratowy, pozwolił na nauczenie się generowania skutecznych wartości parametrów funkcji nagrody dla 286 z 293 „sytuacji środowiskowych", które zostały rozwiązane przez algorytm genetyczny. Skuteczność sieci wyniosła więc w tym przypadku 97,6 %. Sztuczna sieć neuronowa w proponowanym rozwiązaniu została zaprojektowana do generowania parametrów funkcji nagrody dla aktualnej „sytuacji środowiskowej", dla której AG nie wyznaczył wcześniej tych parametrów. w celu sprawdzenia, czy i w jakim zakresie nauczona SSN będzie w stanie tak sterować robotem aby jego TCP osiągnęło punkt zadany, postanowiono wygenerować 100 nowych „sytuacji środowiskowych". Badania pokazały, że dla 97 „sytuacji środowiskowych", co oznacza 97% skuteczności działania, sztuczna sieć neuronowa poradziła sobie z wygenerowaniem parametrów funkcji nagrody i robot osiągnął zadane pozycje docelowe, bez wystąpienia kolizji z przeszkodami. Na rysunkach od 45 do 49 przedstawiono przykładowe trajektorie w przypadku uszkodzenia osi $j0$, dla parametrów funkcji nagrody uzyskanych w populacji pierwszej, drugiej oraz trzeciej algorytmu genetycznego oraz wygenerowanych przez sztuczną sieć neuronową. Do przeprowadzenia tych testów posłużono się analogicznym schematem sterowania, jak w przypadku zastosowania danych stabelaryzowanych (rys. 36). Schemat blokowy algorytmu sterowania ze sztuczną siecią neuronową zaprezentowano na rysunku 44.



Rysunek 44. Schemat przedstawiający proces sterowania robotem wykorzystywany podczas testów prawdziwego robota

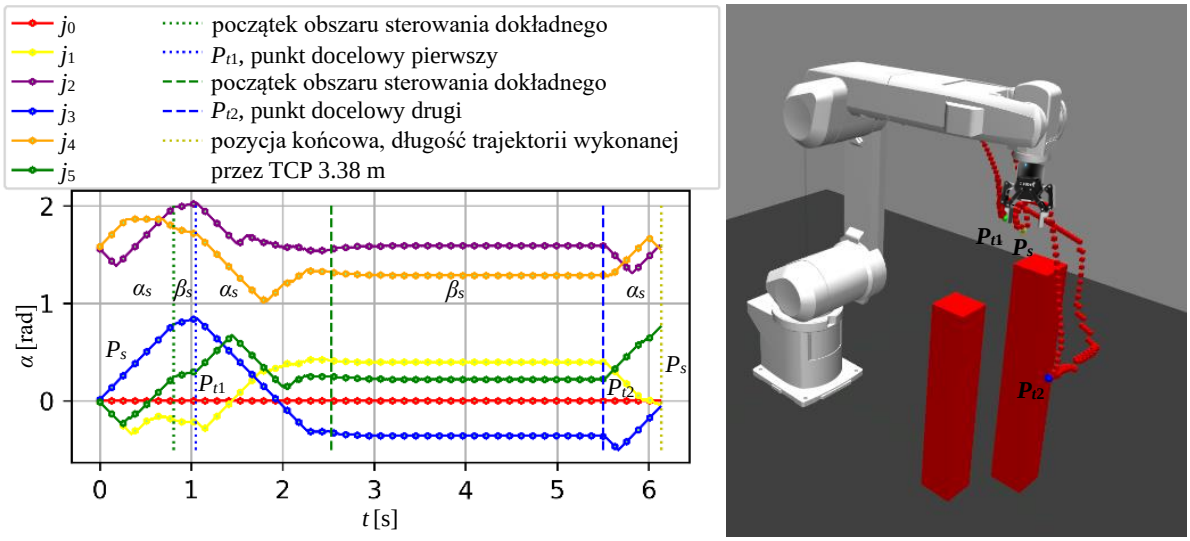


a) b) c)

Rysunek 45. Przedstawienie położenia ramion modelu: a) pozycji poszczególnych osi robota w pozycji startowej P_s b) pozycji poszczególnych osi robota w pierwszej pozycji docelowej P_{t1} c) pozycji poszczególnych osi robota w drugiej pozycji docelowej P_{t2}

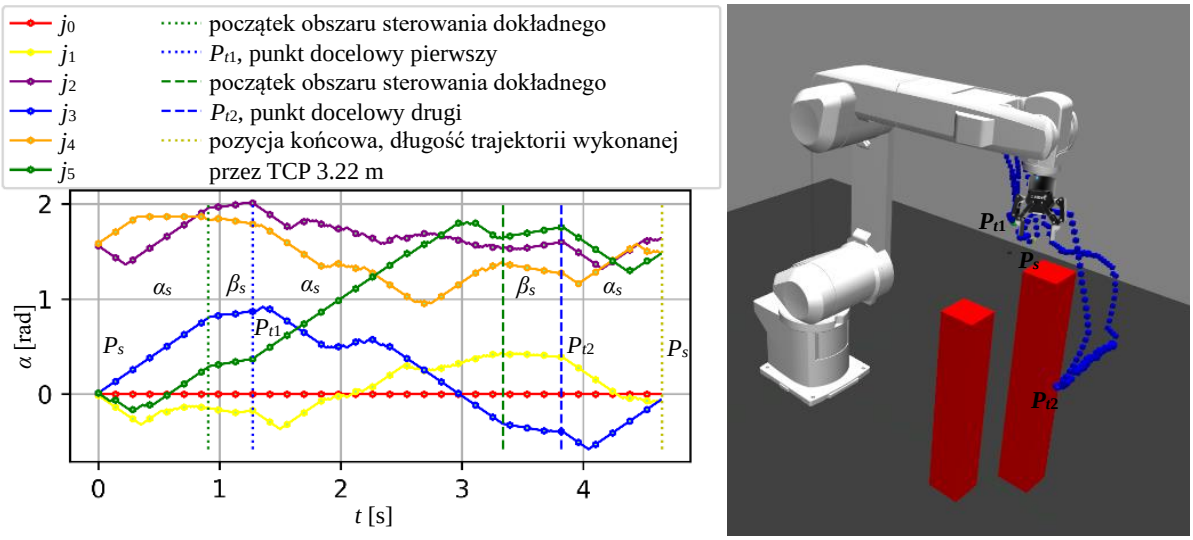
Na rysunku 45 pokazano położenia ramion robota w trzech pozycjach, gdy oś $j0$ robota była uszkodzona od początku trwania ruchu, co ma swoje odzwierciedlenie w położeniach osi. Widać, że położenie osi zerowej (w podstawie) nie zmienia się dla wszystkich pozycji. Informację o uszkodzonej osi można również zauważyć na rysunkach 46a, 47a, 48a oraz 49a. Na rysunkach 46b, 47b, 48b oraz 49b pokazano wygenerowaną trajektorię ruchu TCP robota. Jak widać, długość trajektorii dla najlepszych osobników AG w kolejnych populacjach była coraz mniejsza. Świadczy to o tym, że algorytm uzyskiwał coraz lepsze

rezultaty tj. parametry i możliwa jest dalsza optymalizacja algorytmu genetycznego w następnych populacjach.



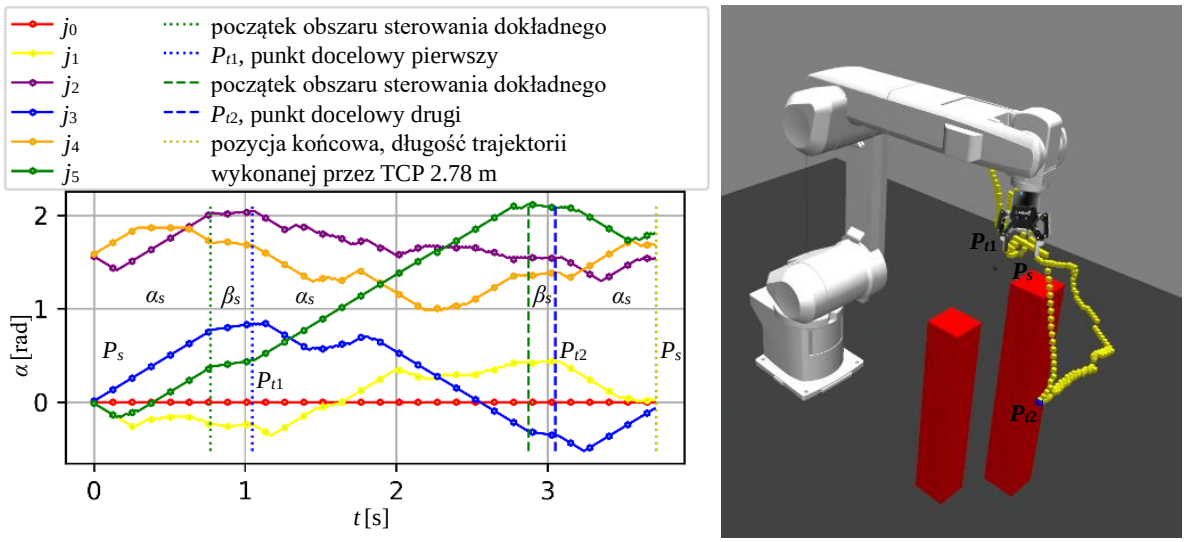
a) b)

Rysunek 46. Przedstawienie działania algorytmu dla najlepszego osobnika populacji pierwszej AG: a) zmiany w czasie położeń kątowych poszczególnych osi, b) robot i wygenerowana trajektoria ruchu TCP



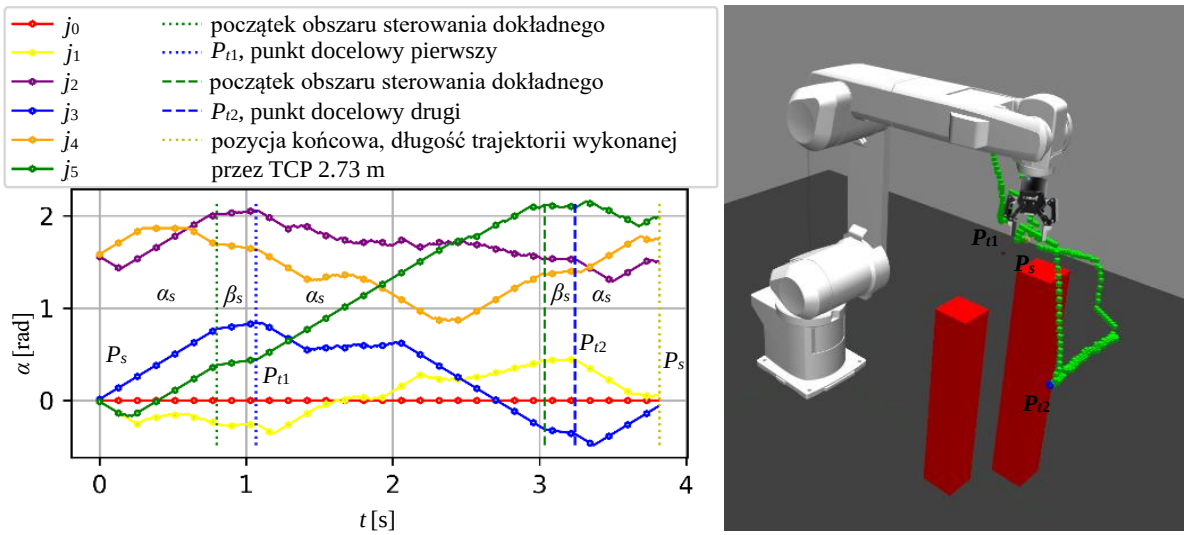
a) b)

Rysunek 47. Przedstawienie działania algorytmu dla najlepszego osobnika populacji drugiej AG: a) zmiany w czasie położeń kątowych poszczególnych osi, b) robot i wygenerowana trajektoria ruchu TCP



Rysunek 48. Przedstawienie działania algorytmu dla najlepszego osobnika populacji trzeciej AG: a) zmiany w czasie położeń kątowych poszczególnych osi, b) robot i wygenerowana trajektoria ruchu TCP

Trajektoria wygenerowana z użyciem parametrów funkcji nagrody, dobranych przez SSN, była w około 74% przypadków krótsza niż ta wygenerowana przez najlepszego osobnika populacji 3-ciej algorytmu genetycznego, który służył do nauki sztucznej sieci neuronowej. Jest to prawdopodobnie rezultatem tego, że SSN generalizowała wynik na podstawie informacji z wielu rozwiązanych sytuacji i była w stanie dobierać coraz lepsze parametry p_m i p_t.



Rysunek 49. Przedstawienie działania algorytmu dla wyuczonej SSN: a) zmiany w czasie położeń kątowych poszczególnych osi, b) robot i wygenerowana trajektoria ruchu TCP

Zaprezentowane w tym rozdziale wyniki badań, potwierdzają możliwość wykorzystania SSN do sterowania robotem, w momencie awarii osi, z jednoczesnym omijaniem przeszkód w przestrzeni roboczej robota. Badania pokazały wysoką skuteczność algorytmu genetycznego w poszukiwaniu rozwiązań optymalnych. Dodatkowo, dzięki zastosowaniu SSN, algorytm był w stanie rozwiązywać zadania, które nie były wcześniej optymalizowane przez AG.

5.4 Badanie algorytmu z kryterium minimalnego zużycia energii

Badanie poprawności działania proponowanego systemu sterowania dla kryterium energetycznego, rozpoczęto od optymalizacji parametrów funkcji nagrody przez AG, dla 220-tu „sytuacji środowiskowych", które były dokładnie takie same jak w przypadku tabeli na rysunku 41 dla kroku α_s.

	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8	M_9	M_{10}
K_1	1091 [Ws]	397 [Ws]	257 [Ws]	100 [Ws]	115 [Ws]	358 [Ws]	213 [Ws]	274 [Ws]	154 [Ws]	256 [Ws]
K_2	1128 [Ws]	344 [Ws]	239 [Ws]	107 [Ws]	589 [Ws]	275 [Ws]	175 [Ws]	264 [Ws]	163 [Ws]	258 [Ws]
K_3	2763 [Ws]	615 [Ws]	420 [Ws]	758 [Ws]	254 [Ws]	269 [Ws]	887 [Ws]	671 [Ws]	252 [Ws]	471 [Ws]
K_4	1066 [Ws]	449 [Ws]	405 [Ws]	82 [Ws]	1493 [Ws]	279 [Ws]	290 [Ws]	375 [Ws]	221 [Ws]	325 [Ws]
K_5	-	341 [Ws]	238 [Ws]	303 [Ws]	376 [Ws]	2105 [Ws]	235 [Ws]	289 [Ws]	292 [Ws]	-
K_6	-	-	980 [Ws]	440 [Ws]	-	-	597 [Ws]	1320 [Ws]	195 [Ws]	785 [Ws]
K_7	-	-	-	162 [Ws]	3032 [Ws]	-	1386 [Ws]	-	1691 [Ws]	-
K_8	2924 [Ws]	638 [Ws]	505 [Ws]	540 [Ws]	241 [Ws]	537 [Ws]	439 [Ws]	698 [Ws]	242 [Ws]	475 [Ws]
K_9	1084 [Ws]	416 [Ws]	363 [Ws]	119 [Ws]	915 [Ws]	256 [Ws]	288 [Ws]	387 [Ws]	242 [Ws]	313 [Ws]
K_{10}	-	393 [Ws]	325 [Ws]	395 [Ws]	387 [Ws]	3588 [Ws]	254 [Ws]	368 [Ws]	306 [Ws]	-

K_{11}	-	-	1012 [Ws]	425 [Ws]	-	-	2694 [Ws]	1392 [Ws]	195 [Ws]	739 [Ws]
K_{12}	-	-	-	295 [Ws]	232 [Ws]	-	1397 [Ws]	-	1723 [Ws]	-
K_{13}	-	740 [Ws]	695 [Ws]	535 [Ws]	281 [Ws]	384 [Ws]	469 [Ws]	708 [Ws]	403 [Ws]	550 [Ws]
K_{14}	-	-	-	-	-	-	-	2523 [Ws]	-	-
K_{15}	-	-	-	-	-	-	-	-	-	-
K_{16}	-	-	-	515 [Ws]	395 [Ws]	-	1728 [Ws]	-	1803 [Ws]	-
K_{17}	-	402 [Ws]	357 [Ws]	290 [Ws]	-	421 [Ws]	279 [Ws]	380 [Ws]	341 [Ws]	-
K_{18}	-	-	-	416 [Ws]	-	-	553 [Ws]	1011 [Ws]	217 [Ws]	731 [Ws]
K_{19}	-	-	-	-	-	-	-	-	-	-
K_{20}	-	-	-	-	-	-	-	-	1087 [Ws]	-
K_{21}	-	-	-	402 [Ws]	-	-	1017 [Ws]	-	1129 [Ws]	-
K_{22}	-	-	-	397 [Ws]	-	-	1681 [Ws]	-	1787 [Ws]	-

Rysunek 50. Tabela obrazująca działanie algorytmu w momencie użycia kryterium energetycznego, kolor szary – sytuacje dla których punkty docelowe znajdują się poza obszarem roboczym, kolor niebieski – sytuacje rozwiązane przez AG, kolor żółty – sytuacje które były w obszarze roboczym ale AG ich nie rozwiązał, kolor zielony – zaznaczona konfiguracja kinematyczna robota pozwalająca na użycie najmniejszej ilości energii.

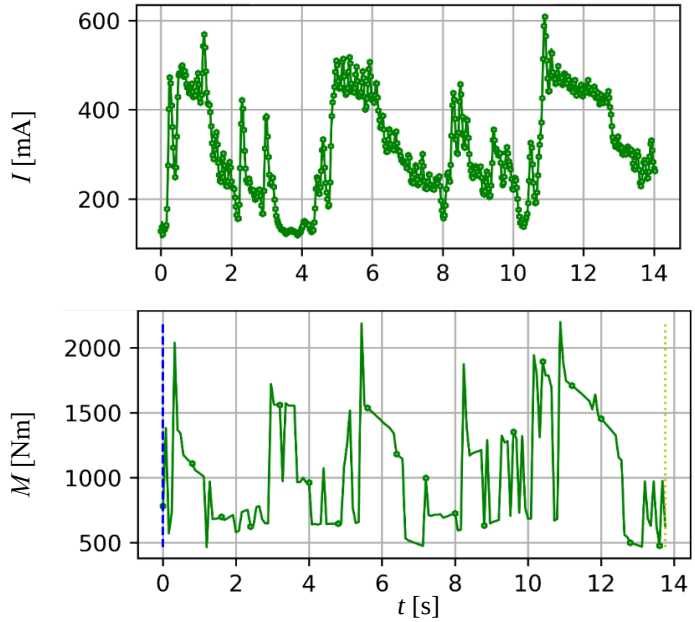
W poszczególnych komórkach tabeli, przedstawionej na rysunku 50, zapisano wartości energii, które obliczył algorytm na podstawie modelu momentowego dla poszczególnych „sytuacji środowiskowych”. Obliczona energia dotyczy najlepszego osobnika ze wszystkich trzech populacji. Na niebiesko zaznaczono sytuacje rozwiązane przez algorytm genetyczny. Na zielono zaznaczono konfiguracje kinematyczne, które pozwalają na wykonanie zadania przemieszczenia do punktów docelowych, z ominięciem przeszkód, przy zużyciu najmniejszej ilości energii. Na szaro oznaczono sytuacje, które nie były

możliwe do rozwiązania, ze względu na występowanie kolizji robota z przeszkodami. Na żółto są pokazane sytuacje, które teoretycznie powinny zostać rozwiązane przez algorytm genetyczny, który jednak „nie poradził" sobie z postawionym zadaniem. w przypadku algorytmu genetycznego, w którym jedyną modyfikacją była zmiana kryterium optymalizującego, skuteczność uzyskania punktu docelowego przez TCP robota wyniosła 87,6 %.

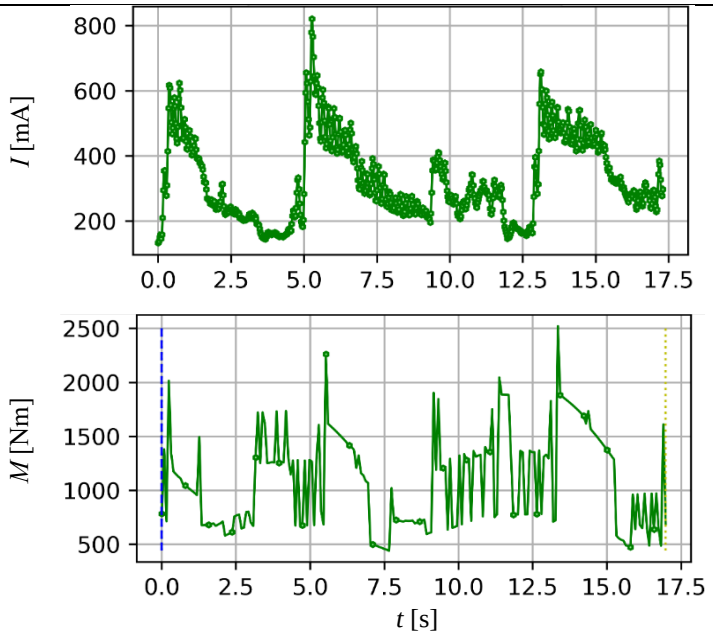
Analizując tabelę przedstawioną na rysunku 50, można zauważyć, że w 7-miu sytuacjach nieoczekiwanym skutkiem awarii osi, było zaoszczędzenie średnio około 12% energii. Wynik ten pozwala na stwierdzenie, że zaproponowane rozwiązanie ma dodatkowy potencjał w uzyskiwaniu oszczędności energii. Aby to wystarczająco dobrze potwierdzić, należało by wykonać zdecydowanie większą liczbę badań, na większym zestawie danych. Jednak takie badania będą czasochłonne, ze względu na ich bardzo duży koszt obliczeniowy. Do obliczenia jednej populacji, dla jednej „sytuacji środowiskowej", konieczna jest około jedna godzina czasu pracy systemu. Wyniki zamieszczone w tabeli na rysunku 48, prezentują 10 sytuacji środowiskowych, dla 22 konfiguracji kinematycznych robota po 3 populacje dla każdej sytuacji, co w sumie daje 660 populacji, które muszą być wykonane. Przekłada się to na czas pacy systemu rzędu 660 godzin. Do tego czasu należy doliczyć czas potrzebny na wykonanie analizy danych itp. Stąd w ramach tej rozprawy doktorskiej, postanowiono oprzeć się tylko na ograniczonych, przedstawionych tutaj wynikach, tylko po to aby pokazać, iż opracowany system sterowania umożliwia oszczędzanie energii oraz wskazać, że ma on jeszcze większy potencjał możliwości i zastosowań.

Zaprojektowana SSN1 do wyboru „quasi-optymalnej kinematyki" oraz SSN2 do wyboru parametrów funkcji nagrody, zadziałały zgodnie z oczekiwaniami dla wszystkich przypadków rozważanych przez AG. Sztuczna sieć neuronowa działała poprawnie dla „sytuacji środowiskowych" innych niż rozwiązane przez algorytm genetyczny. w proponowanym rozwiązaniu SSN1 miała za zadanie wybrać możliwie bliską optymalnej kinematykę robota, na podstawie położeń docelowych oraz położeń przeszkód. w tym przypadku dla uczenia SSN1, na podstawie algorytmu genetycznego przygotowano jedynie 10 zestawów danych uczących (oznaczone w tabeli na rysunku 50 przez zielone pola). Proponowany algorytm działający w oparciu o zużycie energii, ze względu na wysokie koszty obliczeniowe, jest możliwy do zrealizowania i głębszego przebadania jedynie na jednostkach komputerowych o bardzo dużej mocy obliczeniowej.

W celu sprawdzenia dokładności modelu momentowego, otrzymane z jego pomocą wyniki porównano z wynikami pomiarów natężenia prądu pobieranego przez robota rzeczywistym. Wykonano 50 prób, z których wynikało że średni błąd modelu momentowego, względem odczytów rzeczywistych wynosił około 5,8%. Maksymalny błąd w jednej z prób wyniósł 8,7%. Na rysunkach 51 oraz 52 pokazano przykładowe przebiegi zmian natężenia prądu i momentu dla dwóch trajektorii wykonanych przez TCP robota. Można na nich zauważyć, iż przebieg prądu w czasie, jest proporcjonalny do momentu obrotowego, obliczonego w danej chwili czasowej za pomocą zaproponowanego modelu, co potwierdza poprawność działania modelu.



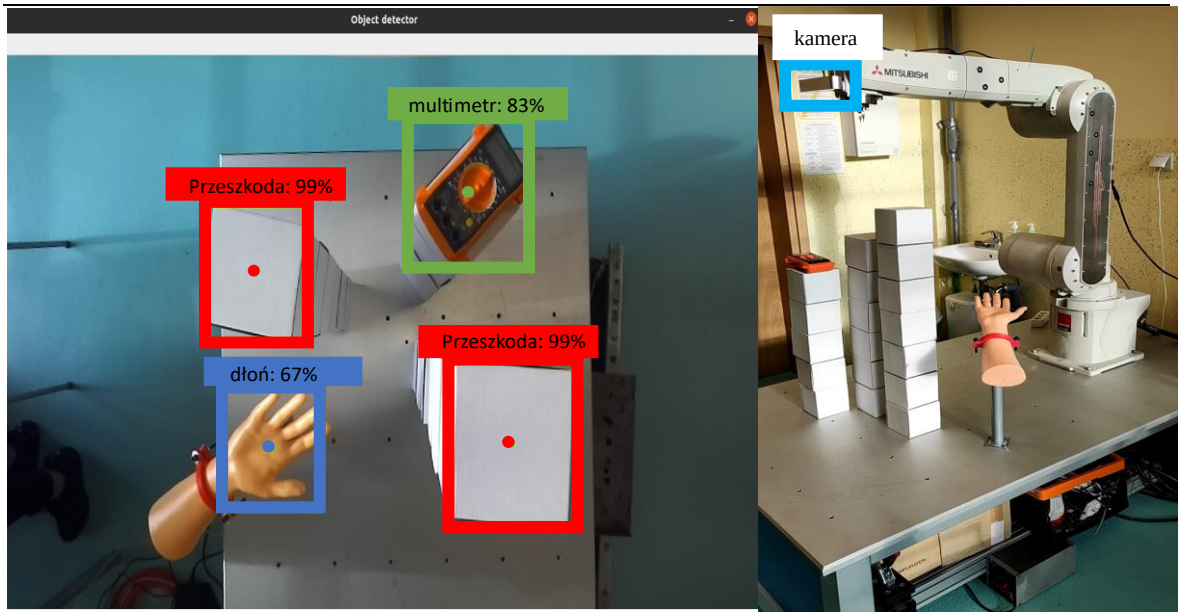
Rysunek 51. Wykres natężenia prądu oraz momentu obrotowego względem czasu dla jednakowej trajektorii



Rysunek 52. Wykres natężenia prądu oraz momentu obrotowego względem czasu dla jednakowej trajektorii

5.5 Badania zastosowania sterowania robotem metodą FTC do współpracy z człowiekiem

W pracy wykonano także badanie sprawdzające możliwość funkcjonowania opracowanego algorytmu, podczas współpracy z człowiekiem. Do tego celu postanowiono ustawić w przestrzeni roboczej dwie przeszkody oraz umieścić obiekt do chwycenia i przeniesienia go do sztucznej dłoni. Podczas pracy tego algorytmu nie chwytano obiektu, ale skupiono się tylko na wyznaczeniu: pozycji przeszkód O_1 i O_2, pozycji obiektu P_{t1} oraz pozycji dłoni P_{t2}, które następnie zaimportowano do środowiska symulacyjnego. Do rozpoznawania oraz lokalizowania obiektów, znajdujących się na stole roboczym robota, wykorzystano kamerę z detekcją głębi typu ZED1 firmy *Stereolabs* [80]. Wykorzystanie tego rodzaju kamery pozwalało na łatwe wyznaczenie pozycji określonego obiektu w przestrzeni. Rozpoznawanie obiektów w przeprowadzonym badaniu wykonano za pomocą sieci *MobileNetV2* [81]. Sieć tą wybrano za względu na to, iż wymaga ona niewielkiej liczby danych, koniecznych do nauczenia odpowiedniego klasyfikatora. Do nauczenia sieci wystarczyło jedynie po 200 przykładowych ilustracji danego obiektu, aby możliwe było ich klasyfikowanie. Na rysunku 53 przedstawiono w jaki sposób algorytm z siecią *MobileNetV2*, oznaczał obiekty w przestrzeni roboczej robota.



Rysunek 53. Rozpoznawanie obiektów w przestrzeni roboczej oraz stanowisko badawcze

W pierwszym kroku algorytm rozpoznawał obiekty i oznaczał je ramką odpowiedniego koloru. Następnie wyznaczany był środek geometryczny danych prostokątów. w kolejnym kroku, algorytm wyznaczał współrzędne x, y, z środków obiektów względem kamery. Znając pozycję kamery w przestrzeni, algorytm określał położenia obiektów w tej przestrzeni. Kolejnym krokiem było ustawienie w środowisku symulacyjnym przeszkód na odpowiednich położeniach oraz dodanie pozycji położenia multimetru oraz dłoni, stanowiących pozycje P_{t1} i P_{t2}. Po wykonaniu tych czynności, zaproponowany system FTC mógł przeprowadzić swoją pracę.

Zadanie do wykonania, przedstawione w tym rozdziale, jest jedną z możliwość wykorzystania zaproponowanego algorytmu i całego systemu sterowania odpornego na błędy.

6 Podsumowanie

W niniejszej pracy podjęto zadania rozwiązania problemu sterownia robotem przemysłowym, od momentu wystąpienia awarii jednej albo dwóch dowolnych jego osi. Do jego rozwiązania zaproponowano system sterowania, opierający swoje działanie na modelu symulacyjnym robota rzeczywistego oraz na metodach sztucznej inteligencji, sterujący przemieszczaniem się TCP robota do zadanych punktów docelowych. Dodatkowym zadanie algorytmu było jednoczesne omijanie przeszkód, znajdujących się na wyznaczanej drodze robota. Warto zauważyć, że we wcześniejszych doniesieniach literaturowych nie znaleziono rozwiązania, pozwalającego na sterowanie robotem w momencie uszkodzenia osi i zapewniającego jednoczesne omijanie przeszkód. Nowością w podejściu do rozwiązania problemu sterowania odpornego na błędy w postaci awarii osi, jest wykorzystanie algorytmu genetycznego do optymalizacji generowanej trajektorii, za pomocą doboru odpowiednich parametrów zaproponowanej funkcji nagrody. Innowację stanowi także wykorzystanie algorytmu genetycznego do przygotowania danych uczących, dla sztucznej sieci neuronowej, której zadaniem było generowania parametrów funkcji nagrody, dla dowolnej tzw. „sytuacji środowiskowej”, określającej położenia punktów: początkowego i końcowego, położenia przeszkód oraz miejsc wystąpienia awarii dowolnej jednej albo dwóch osi. Dotychczas stosowane rozwiązania w zakresie odpornego na błędy sterowania robotami, nakierowane są na dokładność pozycjonowania, rozwiązanie problemu drgań, szybkie wykrywanie oraz izolowanie usterek. w niniejszej pracy skupiono się zarówno na sterowaniu robotem z uszkodzonymi osiami, jak i na omijaniu przeszkód oraz na pracy systemu sterowania dla różnych kryteriów, to jest: jak najkrótsza albo optymalna energetycznie trajektoria ruchu. Opracowany system sterowania może być zastosowany do sterowania dowolnym manipulatorem o liczbie stopni swobody od dwóch do sześciu. System był testowany z powodzeniem na robocie o sześciu stopniach swobody i mógłby, zdaniem autora, z powodzeniem funkcjonować w przypadku awarii większej liczby osi. Jednak już w przypadku manipulatora o 6-ciu stopniach swobody, uszkodzenie kolejnej tj. trzeciej osi, powodowało znaczne ograniczenie obszaru roboczego robota i znacząco wydłużało czas pracy algorytmu. Dlatego skupiono się tylko na sterowaniu robotem typu FTC, przy awarii do dwóch jego osi.

Głównymi osiągnięciami pracy są:

- Opracowanie modelu symulacyjnego robota uwzględniającego parametry kinematyczne robota i umożliwiającego sprawdzanie algorytmu w środowisku wirtualnym. Pokazują to rezultaty opisane w rozdziale 5.1.
- Opracowanie modelu symulacyjnego robota pozwalającego na obliczenie zużycia energii w celu zastosowania kryterium energetycznego. Model przedstawiono w rozdziale 4.6, natomiast uzyskane wyniki w rozdziale 5.4.
- Opracowanie algorytmu optymalizującego trajektorię ruchu w postaci algorytmu genetycznego, dla którego wyniki przedstawiono w rozdziale 5.2, 5.3, 5.4
- Opracowanie funkcji nagrody zgodnie z którą oceniane są poszczególne kroki robota. Funkcję tą opisano w rozdziale 4.2.
- Zastosowanie sztucznej sieci neuronowej do predykcji parametrów funkcji nagrody na podstawie aktualnej sytuacji środowiskowej. Sztuczne sieci neuronowe, zastosowane do tego celu, przedstawiono w rozdziałach 4.5 oraz 4.7
- Opracowanie systemu odpornego na uszkodzenia osi robota oraz pozwalającego na omijanie znajdujących się w obszarze roboczym przeszkód, co potwierdzają uzyskane wyniki w rozdziałach 5.2 oraz 5.3.
- Wykonanie badań symulacyjnych i doświadczalnych potwierdzających działanie proponowanego rozwiązania, które przedstawiono w rozdziale 5

Przedstawione wyżej osiągnięcia pozwalają stwierdzić, iż w pracy zrealizowano wszystkie zaplanowane cele cząstkowe.

Przeprowadzone badania wskazują, że w zależności od rozdzielczości ruchu podyktowanej wartością parametrów α_s i β_s, algorytm był w stanie pozycjonować TCP robota w symulacji z dokładnością poniżej 2 mm, w przypadku gdy α_s=1,0° i β_s=0,2°. w przypadku zastosowania wyższej rozdzielczości, a mianowicie α_s=0,2° i β_s=0,05° dokładność pozycjonowania wyniosła poniżej ±0.5mm. Wartości te są akceptowalne w przypadku zadania większości zadań robota typu „pick and place” tzn. podnieś odłóż. w przypadku sterowania robotem rzeczywistym błąd zależy od dokładności modelu oraz dokładności jaką deklaruje producent robota.

Zaproponowana w badaniach sztuczna sieć neuronowa na podstawie danych, które wygenerował algorytm genetyczny, była w stanie nauczyć się generowania quasi-optymalnych parametrów funkcji nagrody. Wyuczona sieć była w stanie rozwiązać 97% „sytuacji środowiskowych”, które nie były wcześniej optymalizowane przez algorytm genetyczny.

Dzięki optymalizacji parametrów funkcji nagrody, zgodnie z kryterium energetycznym, oraz zaprojektowaniem dodatkowej sieci wybierającej optymalną energetycznie „konfiguracje kinematyczną" robota oraz w zależności od „sytuacji środowiskowej", proponowany system był w stanie zaoszczędzić około 12% energii.

Przedstawione w pracy wyniki badań potwierdzają postawioną tezę, że **algorytm działający w oparciu o model symulacyjny robota, algorytm genetyczny, sztuczną sieć neuronową oraz funkcję nagrody, pozwala na sterowanie robotem przemysłowym, które jest odporne na awarię do dwóch osi oraz zapewnia omijanie przeszkód**. Wyniki te pozwalają także stwierdzić, że zrealizowano główny cel pracy jakim było sterowanie oporne na awarię do dwóch osi, z jednoczesnym omijaniem przeszkód w przestrzeni roboczej robota.

Przestawione w niniejszej pracy badania nie wyczerpują w całości problemu sterowania robotem, w momencie uszkodzenia osi z jednoczesnym omijaniem przeszkód. Zdaniem autora niniejszej pracy, w przyszłych badania możliwe jest rozszerzenie badań, aby w bardziej znaczącym stopniu potwierdzić poprawność działania algorytmu oraz poprawić jego niezawodność, dokładność i szybkość działania. Do najważniejszych prac w tym zakresie można zaliczyć między innymi:

1. Zoptymalizowanie przez algorytm genetyczny kilku tysięcy „sytuacji środowiskowych", aby przygotować większy zestaw danych uczących, zarówno dla kryterium najkrótszej możliwej trajektorii, jak i dla kryterium najmniejszego zużycia energii. Wymaga to jednak dużej mocy obliczeniowej i dużo czasu.
2. Zwiększenie liczby osobników w populacji oraz liczby populacji algorytmu genetycznego, co skutkowało by znalezieniem optymalnego rozwiązania.
3. Wyuczenie sztucznej sieci neuronowej na większym zestawie danych uczących, aby sprawdzić czy i o ile poprawią się możliwości zaproponowanego rozwiązania, pod względem skuteczności rozwiązywania sytuacji środowiskowych.
4. Wbudować zaawansowany system detekcji i lokalizowania przeszkód oraz pozycji docelowych w celu rozszerzenia możliwości algorytmu o omijanie przeszkód ruchomych oraz o możliwość zmiany położenia punktów docelowych w trakcie ruchu.

Bibliografia

1. Grau, A., Indri, M., Bello, L. L., & Sauter, T. (2020). Robots in industry: The past, present, and future of a growing collaboration with humans. IEEE Industrial Electronics Magazine, 15(1), 50-61.
2. Ghodsian, N., Benfriha, K., Olabi, A., Gopinath, V., & Arnou, A. (2023). Mobile Manipulators in Industry 4.0: a Review of Developments for Industrial Applications. Sensors, 23(19), 8026.
3. Tenhundfeld, N. L., Barr, H. M., O'Hear, E., Atchley, A., & Cotter, J. E. (2021, September). Effects of human-likeness on robot use in high-risk environments. In Proceedings of the Human Factors and Ergonomics Society Annual Meeting (Vol. 65, No. 1, pp. 1049-1053). Sage CA: Los Angeles, CA: SAGE Publications.
4. Gentile, C., Lunghi, G., Buonocore, L. R., Cordella, F., Di Castro, M., Masi, A., & Zollo, L. (2023). Manipulation Tasks in Hazardous Environments Using a Teleoperated Robot: a Case Study at CERN. Sensors, 23(4), 1979.
5. Picard, M. (2019). AN OVERVIEW OF THE CSA RECENT ACTIVITIES IN SPACE ROBOTICS.
6. Aviziens, A. (1976). Fault-tolerant systems. IEEE transactions on computers, 100(12), 1304-1312.
7. Stengel, R. F. (1991). Intelligent failure-tolerant control. IEEE Control Systems Magazine, 11(4), 14-23.
8. Veillette, R. J., Medanic, J. V., & Perkins, W. R. (1990, December). Design of reliable control systems. In 29th IEEE Conference on Decision and Control (pp. 1131-1136). IEEE.
9. Patton, R. (1993, May). Robustness issues in fault-tolerant control. In IEE Colloquium on Fault Diagnosis and Control System Reconfiguration (pp. 1-1). IET.
10. Patton, R. J. (1997). Fault-tolerant control: The 1997 situation. IFAC Proceedings Volumes, 30(18), 1029-1051.
11. Maciejewski, A. A. (1990, May). Fault tolerant properties of kinematically redundant manipulators. In Proceedings., IEEE International Conference on Robotics and Automation (pp. 638-642). IEEE.
12. Chladek, J. T. (1990, June). Fault tolerance for space based manipulator mechanisms and control system. In First International Symposium On Measurement and Control in Robotics.
13. Tesar, D., Sreevijayan, D., & Price, C. (1990, June). Four-level fault tolerance in manipulator design for space operations. In Proceedings of the First International Symposium on Measurement and Control in Robotics. Houston, TX.
14. Wu, E., Diftler, M., Hwang, J., & Chladek, J. (1991, April). a fault tolerant joint drive system for the space shuttle remote manipulator system. In Proceedings. 1991 IEEE International Conference on Robotics and Automation (pp. 2504-2509). IEEE.

15. Tesar, D. (1993, February). On the design of fault-tolerant robotic manipulator systems. In 6th Annual Workshop on Space Operations Applications and Research (SOAR 1992), Volume 2 (p. 223).

16. Visinsky, M. L., Cavallaro, J. R., & Walker, I. D. (1993). Robotic fault tolerance: algorithms and architectures. Prentice Hall, Englewood Cliffs, NJ.

17. Ting, Y., Tosunoglu, S., & Tesar, D. (1993, May). a control structure for fault-tolerant operation of robotic manipulators. In [1993] Proceedings IEEE International Conference on Robotics and Automation (pp. 684-690). IEEE.

18. Selkäinaho, J., & Halme, A. (1987). An intelligent fault tolerant control system. IFAC Proceedings Volumes, 20(5), 69-73.

19. Hörmann, A., Hugel, T., & Meier, W. (1988). a concept for an intelligent and fault-tolerant robot system. Journal of Intelligent and Robotic Systems, 1(3), 259-286.

20. Groom, K. N., Maciejewski, A. A., & Balakrishnan, V. (1999). Real-time failure-tolerant control of kinematically redundant manipulators. IEEE transactions on robotics and automation, 15(6), 1109-1115.

21. Shin, J. H., & Lee, J. J. (1999, May). Fault detection and robust fault recovery control for robot manipulators with actuator failures. In Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No. 99CH36288C) (Vol. 2, pp. 861-866). IEEE.

22. Amin, A. A., & Hasan, K. M. (2019). a review of fault tolerant control systems: advancements and applications. Measurement, 143, 58-68.

23. Rahali, H., Zeghlache, S., & Benyettou, L. (2021). Fault Tolerant Control of Robot Manipulators Based on Adaptive Fuzzy Type-2 Backstepping in Attendance of Payload Variation. International Journal of Intelligent Engineering and Systems, 14(4), 312-325.

24. Ali, K., Mehmood, A., & Iqbal, J. (2021). Fault-tolerant scheme for robotic manipulator—Nonlinear robust back-stepping control with friction compensation. Plos one, 16(8), e0256491.

25. Tanaka, K., Hori, T., & Wang, H. O. (2003). a multiple Lyapunov function approach to stabilization of fuzzy control systems. IEEE Transactions on fuzzy systems, 11(4), 582-589.

26. Hagh, Y. S., Asl, R. M., Fekih, A., Wu, H., & Handroos, H. (2021). Active fault-tolerant control design for actuator fault mitigation in robotic manipulators. IEEE Access, 9, 47912-47929.

27. Asl, R. M., Hagh, Y. S., & Handroos, H. (2017, August). Adaptive extended Kalman filter designing based on non-singular fast terminal sliding mode control for robotic manipulators. In 2017 IEEE International Conference on Mechatronics and Automation (ICMA) (pp. 1670-1675). IEEE.

28. Dev Anand, M., Selvaraj, T., Kumanan, S., & Ajitha, T. (2010). a neuro-fuzzy-based fault detection and fault tolerance methods for industrial robotic manipulators. International Journal of Adaptive and Innovative Systems, 1(3-4), 334-371.

29. Deo, A. S. (1991). Application of optimal damped least-squares method to inverse kinematics of robotic manipulators. Rice University.

30. Dev Anand, M., Selvaraj, T., & Kumanan, S. (2012). Fault Detection and Fault Tolerance Methods for Industrial Robot Manipulators Based On Hybrid Intelligent Approach. Advances in Production Engineering & Management, 7(4).

31. Van, M., Do, X. P., & Mavrovouniotis, M. (2020). Self-tuning fuzzy PID-nonsingular fast terminal sliding mode control for robust fault tolerant control of robot manipulators. ISA transactions, 96, 60-68.

32. Piltan, F., Prosvirin, A. E., Sohaib, M., Saldivar, B., & Kim, J. M. (2020). An SVM-based neural adaptive variable structure observer for fault diagnosis and fault-tolerant control of a robot manipulator. Applied Sciences, 10(4), 1344.

33. Li, Z., Li, C., Li, S., & Cao, X. (2019). a fault-tolerant method for motion planning of industrial redundant manipulator. IEEE transactions on industrial informatics, 16(12), 7469-7478.

34. Zhang, Y., Wang, J., & Xia, Y. (2003). a dual neural network for redundancy resolution of kinematically redundant manipulators subject to joint limits and joint velocity limits. IEEE transactions on neural networks, 14(3), 658-667.

35. Eski, I., Erkaya, S., Savas, S., & Yildirim, S. (2011). Fault detection on robot manipulators using artificial neural networks. Robotics and Computer-Integrated Manufacturing, 27(1), 115-123.

36. Zhao, B., & Li, Y. (2014). Local joint information based active fault tolerant control for reconfigurable manipulator. Nonlinear dynamics, 77(3), 859-876.

37. Zhang, S., Yang, P., Kong, L., Chen, W., Fu, Q., & Peng, K. (2019). Neural networks-based fault tolerant control of a robot via fast terminal sliding mode. IEEE Transactions on Systems, Man, and Cybernetics: Systems, 51(7), 4091-4101.

38. Hwang, C. L., & Yu, W. S. (2020). Tracking and cooperative designs of robot manipulators using adaptive fixed-time fault-tolerant constraint control. IEEE Access, 8, 56415-56428.

39. Tinós, R., Terra, M. H., & Bergerman, M. (2007). a fault tolerance framework for cooperative robotic manipulators. Control Engineering Practice, 15(5), 615-625.

40. Yu, X., Feng, Y., & Man, Z. (2020). Terminal sliding mode control–An overview. IEEE Open Journal of the Industrial Electronics Society, 2, 36-52.

41. Khireddine, M. S., & Boutarfa, A. (2013, January). Fault tolerant control on robotic manipulator using sliding mode observers. In 2013 International Conference on Computer Applications Technology (ICCAT) (pp. 1-6). IEEE.

42. Van, M., Franciosa, P., & Ceglarek, D. (2016). Fault diagnosis and fault-tolerant control of uncertain robot manipulators using high-order sliding mode. Mathematical Problems in Engineering, 2016.

43. Piltan, F., Mirzaei, M., Shahriari, F., Nazari, I., & Emamzadeh, S. (2012). Design baseline computed torque controller. International Journal of Engineering, 6(3), 129-141.

44. Tosunoglu, S. (1995, November). Fault-tolerant control of mechanical systems. In Proceedings of IECON'95-21st Annual Conference on IEEE Industrial Electronics (Vol. 1, pp. 127-132). IEEE.

45. Merheb, A. R., Noura, H., & Bateman, F. (2015). Design of passive fault-tolerant controllers of a quadrotor based on sliding mode theory. International Journal of Applied Mathematics and Computer Science, 25(3).

46. Van, M., & Ceglarek, D. (2021). Robust fault tolerant control of robot manipulators with global fixed-time convergence. Journal of the Franklin Institute, 358(1), 699-722.

47. Xu, B., Zhang, L., & Ji, W. (2021). Improved non-singular fast terminal sliding mode control with disturbance observer for PMSM drives. IEEE Transactions on Transportation Electrification, 7(4), 2753-2762.

48. Chen, Y., & Guo, B. (2019). Sliding mode fault tolerant tracking control for a single-link flexible joint manipulator system. IEEE Access, 7, 83046-83057.

49. Abd Latip, S. F., Husain, A. R., Ahmad, M. N., & Mohamed, Z. (2016). Fault tolerant control for sensor fault of a single-link flexible manipulator system. Jurnal teknologi, 78(6-13).

50. Van, M., Mavrovouniotis, M., & Ge, S. S. (2018). An adaptive backstepping nonsingular fast terminal sliding mode control for robust fault tolerant control of robot manipulators. IEEE Transactions on Systems, Man, and Cybernetics: Systems, 49(7), 1448-1458.

51. Van, M., Ge, S. S., & Ren, H. (2016). Finite time fault tolerant control for robot manipulators using time delay estimation and continuous nonsingular fast terminal sliding mode control. IEEE transactions on cybernetics, 47(7), 1681-1693.

52. Nguyen, V. C., Vo, A. T., & Kang, H. J. (2021). a finite-time fault-tolerant control using non-singular fast terminal sliding mode control and third-order sliding mode observer for robotic manipulators. IEEE Access, 9, 31225-31235.

53. Vo, A. T., & Kang, H. J. (2020). a novel fault-tolerant control method for robot manipulators based on non-singular fast terminal sliding mode control and disturbance observer. IEEE Access, 8, 109388-109400.

54. Le, Q. D., & Kang, H. J. (2020). Implementation of fault-tolerant control for a robot manipulator based on synchronous sliding mode control. Applied Sciences, 10(7), 2534.

55. Nguyen, V. C., Tran, X. T., & Kang, H. J. (2022, September). a Novel High-Speed Third-Order Sliding Mode Observer for Fault-Tolerant Control Problem of Robot Manipulators. In Actuators (Vol. 11, No. 9, p. 259). MDPI.

56. Zhang, Y., Zhu, W., & Rosendo, A. (2019). QR code-based self-calibration for a fault-tolerant industrial robot arm. IEEE Access, 7, 73349-73356.

57. Ge, D. (2022, April). Kinematics modeling of redundant manipulator based on screw theory and Newton-Raphson method. In Journal of Physics: Conference Series (Vol. 2246, No. 1, p. 012068). IOP Publishing.

58. Zahaf, A., Bououden, S., Chadli, M., & Chemachema, M. (2022). Robust fault tolerant optimal predictive control of hybrid actuators with time‐varying delay for industrial robot arm. Asian Journal of Control, 24(1), 1-15.

59. Altan, A., & Hacıoğlu, R. (2020). Model predictive control of three-axis gimbal system mounted on UAV for real-time target tracking under external disturbances. Mechanical Systems and Signal Processing, 138, 106548.

60. Bu, N., Pang, J., & Deng, M. (2020). Robust fault tolerant tracking control for the multi-joint manipulator based on operator theory. Journal of the Franklin Institute, 357(5), 2696-2714.

61. Zhang, G., & Cheng, D. (2020). Adaptive fault-tolerant guaranteed performance control for euler-Lagrange systems with its application to a 2-link robotic manipulator. IEEE Access, 8, 184160-184171.

62. Ma, H., Zhou, Q., Li, H., & Lu, R. (2021). Adaptive prescribed performance control of a flexible-joint robotic manipulator with dynamic uncertainties. IEEE Transactions on Cybernetics.

63. Liu, L., Zhang, L., Wang, Y., & Hou, Y. (2021). a novel robust fixed‐time fault‐tolerant tracking control of uncertain robot manipulators. IET Control Theory & Applications, 15(2), 195-208.

64. Pezzato, C., Baioumy, M., Corbato, C. H., Hawes, N., Wisse, M., & Ferrari, R. (2020, September). Active inference for fault tolerant control of robot manipulators with sensory faults. In International Workshop on Active Inference (pp. 20-27). Springer, Cham.

65. Baioumy, M., Pezzato, C., Ferrari, R., Corbato, C. H., & Hawes, N. (2021, June). Fault-tolerant control of robot manipulators with sensory faults using unbiased active inference. In 2021 European Control Conference (ECC) (pp. 1119-1125). IEEE.

66. Brivadis, L., Andrieu, V., & Serres, U. (2019, December). Luenberger observers for discrete-time nonlinear systems. In 2019 IEEE 58th Conference on Decision and Control (CDC) (pp. 3435-3440). IEEE.

67. Epps, B. P., & Krivitzky, E. M. (2019). Singular value decomposition of noisy data: noise filtering. Experiments in Fluids, 60(8), 1-23.

68. Urrea, C., Kern, J., & López-Escobar, R. (2021). Design and implementation of a fault-tolerant system for industrial robots under hostile operating conditions. Computers & Electrical Engineering, 90, 106951.

69. Urrea, C., Kern, J., & López, R. (2020). Fault-tolerant communication system based on convolutional code for the control of manipulator robots. Control Engineering Practice, 101, 104508.

70. McEliece, R. J. (1998). The algebraic theory of convolutional codes. Handbook of coding theory.

71. Lokman, S. F., Othman, A. T., & Abu-Bakar, M. H. (2019). Intrusion detection system for automotive Controller Area Network (CAN) bus system: a review. EURASIP Journal on Wireless Communications and Networking, 2019(1), 1-17.

72. Urrea, C., Kern, J., & Álvarez, E. (2022). Design and implementation of fault-tolerant control strategies for a real underactuated manipulator robot. Complex & Intelligent Systems, 1-23.

73. Todorov, E., Erez, T., & Tassa, Y. (2012, October). Mujoco: a physics engine for model-based control. In 2012 IEEE/RSJ international conference on intelligent robots and systems (pp. 5026-5033). IEEE.

74. Białek, M., Nowak, P., & Rybarczyk, D. (2020). Application of an artificial neural network for planning the trajectory of a mobile robot. Journal of Automation, Mobile Robotics and Intelligent Systems, 13-23.

75. Banerjee, C., Mukherjee, T., & Pasiliao Jr, E. (2019, April). An empirical study on generalizations of the ReLU activation function. In Proceedings of the 2019 ACM Southeast Conference (pp. 164-167).

76. Werbos, P. J. (1990). Backpropagation through time: what it does and how to do it. Proceedings of the IEEE, 78(10), 1550-1560.

77. Adamczak, S., Domagalski, R., & Sender, E. (2011). Moment oporowy w łożyskach tocznych-metody i urządzenia badawcze. Tribologia, 19-28.

78. https://dl.mitsubishielectric.com/dl/fa/document/manual/servo/sh030041/sh030041l.pdf

79. https://pdf1.alldatasheet.com/datasheet-pdf/view/1160229/YHDC/SCT013-005.html

80. https://www.stereolabs.com/zed-2/

81. Dong, K., Zhou, C., Ruan, Y., & Li, Y. (2020, December). MobileNetV2 model for image classification. In 2020 2nd International Conference on Information Technology and Computer Application (ITCA) (pp. 476-480). IEEE.