ZAKŁAD URZĄDZEŃ MECHATRONICZNYCH
INSTYTUT TECHNOLOGII MECHANICZNEJ
WYDZIAŁ INŻYNIERII MECHANICZNEJ
POLITECHNIKA POZNAŃSKA

ROZPRAWA DOKTORSKA

Zastosowanie metod uczenia ze wzmocnieniem do sterowania robotem przemysłowym współpracującym z człowiekiem

Tymoteusz LINDNER

Promotor:
prof. dr hab. inż. Andrzej Milecki

Poznań, 2022

Streszczenie

ZASTOSOWANIE METOD UCZENIA ZE WZMOCNIENIEM DO STEROWANIA ROBOTEM PRZEMYSŁOWYM WSPÓŁPRACUJĄCYM Z CZŁOWIEKIEM

W pracy podjęto problematykę dotyczącą zastosowania metod sztucznej inteligencji do sterowania robotem przemysłowym, pracującym we wspólnej strefie roboczej z człowiekiem. Pierwszym celem pracy było opracowanie takiego systemu sterowania robotem, aby wykrywał i omijał on przeszkody niespodziewanie pojawiające się na jego zadanej trajektorii ruchu. Zaproponowane rozwiązanie pozwoliło na jego zastosowanie do zapewnienia bezpiecznej współpracy człowieka i robota przemysłowego, we współdzielonej strefie roboczej. Dzięki temu, możliwy był jednoczesny dostęp człowieka i robota do tej samej strefy. Do opracowania systemu sterowania zastosowano algorytmy uczenia ze wzmocnieniem, a do wykrywania przeszkód w polu roboczym robota zaprojektowano i zbudowano dedykowaną głowicę z laserowymi czujnikami odległości, zamontowaną na kiści manipulatora.

W pierwszych rozdziałach pracy przedstawiono różne rozwiązania ilustrujące współpracę robotów przemysłowych z człowiekiem. Opisano zastosowania w robotyce algorytmów bazujących na sztucznej inteligencji i uczeniu ze wzmocnieniem (*Reinforcement Learning*, RL). Omówiono różne metody wykrywania przeszkód, a następnie przedstawiono matematyczne podstawy stosowanych w badaniach algorytmów uczenia ze wzmocnieniem. Opisano środowisko pracy oraz stanowisko badawcze, w którym były przeprowadzane testy i badania symulacyjne.

Zasadniczą częścią pracy były badania symulacyjne i doświadczalne przeprowadzone w laboratorium z zastosowaniem rzeczywistego robota. Wykonano badania różnych algorytmów uczenia ze wzmocnieniem, w zadaniu pozycjonowania punktu środkowego narzędzia (*tool center point*, TCP) robota. Na podstawie uzyskanych wyników wytypowano algorytm, który zastosowano do opracowania systemu składającego się z dwóch, tak zwanych agentów pracujących równolegle i sterujących robotem. Jeden z nich był odpowiedzialny za omijanie przeszkód, a drugi za podążanie po zadanej trajektorii ruchu. Zaprojektowany system zoptymalizowano i zastosowano do sterowania robotem przemysłowym pracującym jednocześnie w tej samej strefie roboczej z człowiekiem.

Wykonane badania, potwierdziły możliwość pracy człowieka i robota przemysłowego we wspólnej strefie roboczej. Wykazano, że opracowany system sterujący, bazujący na metodach uczenia ze wzmocnieniem sterował robotem w taki sposób, że bezkolizyjnie omijał rożne przeszkody zlokalizowane w dowolnym miejscu w przestrzeni roboczej, w tym człowieka. Tym samym wykazano, że zastosowanie metod uczenia ze wzmocnieniem bazujących na sztucznej inteligencji, pozwala na bezpieczne sterowanie robotem współpracującym z człowiekiem.

Abstract

APPLICATION OF REINFORCEMENT LEARNING ALGORITHMS TO CONTROL INDUSTRIAL ROBOT COOPERATING WITH HUMAN

This thesis presents the issues related to the application of artificial intelligence methods to control an industrial robot operating in a shared workspace with a human being. The first goal of the thesis was to develop a robot control system that would detect and avoid obstacles, that unexpectedly appears on its given motion trajectory. The proposed solution allowed for its use to ensure safe cooperation between human and an industrial robot in a shared workspace. As a result, simultaneous human and robot access to the same space was possible. To develop the control system, Reinforcement Learning (RL) algorithms were used, and to detect obstacles in the workspace of the robot, a head with laser distance sensors was designed and built, mounted on the manipulator wrist.

In the first chapters of the thesis, various solutions illustrating the cooperation of industrial robots with humans were presented. The application of algorithms based on artificial intelligence and RL methods in robotics was described. Various methods of obstacle detection were discussed, and then the mathematical foundations of the RL algorithms used in the research were presented. The environment and the test stand in which the tests and simulation experiments were carried out were described.

The essential part of the thesis was experimental research carried out in a laboratory with the application of a real robot. Research on various RL algorithms was carried out in the task of positioning the tool center point (TCP) of the robot. Based on the obtained results, an algorithm was selected and used to develop a system consisting of two agents working in parallel and controlling the robot. One of them was responsible for avoiding obstacles, and the other one was responsible for moving the TCP of the robot on a given trajectory of motion. The designed system was optimized and used to control an industrial robot operating simultaneously in the same workspace with a human being.

The tests carried out confirmed the possibility of human and industrial robot cooperating in a shared workspace. It was shown that the developed control system, based on the RL methods, controlled the robot in such a way that it avoided collision-free various obstacles located anywhere in the workspace, including humans. Thus, it has been shown that the use of RL methods based on artificial intelligence allows for safe control of a robot cooperating with a human.

SPIS TREŚCI

SPIS SYMBOLI I SKRÓTÓW

Symbole matematyczne i funkcje

$a\|b$	–	prawdopodobieństwo warunkowe; prawdopodobieństwo wystąpienia zdarzenia a pod warunkiem zajścia zdarzenia b
$\overrightarrow{a_{xyz}b_{xyz}}$	–	wektor między punktami a_{xyz} i b_{xyz}
$\mathrm{clip(a, b, c)}$	–	funkcja ograniczająca wartość a w zakresie od b do c
$\mathbb{E}$	–	wartość oczekiwana
$max\{\}$, $min\{\}$	–	funkcje: wartość maksymalna, minimalna
$\|\mathrm{A}\|$	–	moduł wektora A; długość wektora
$\nabla_\theta J$	–	gradient funkcji J względem θ
$\leftarrow$	–	operator przypisania

Symbole

a_t, a_{t+1}	–	akcja w kroku czasowym t oraz $t+1$
$a_{xyz}, (a_x, a_y, a_z)$	–	aktualna pozycja TCP robota
ccw	–	przeciwnie do kierunku ruchu wskazówek zegara
cw	–	zgodnie z kierunkiem ruchu wskazówek zegara
B	–	porcja danych pobrana z bufora
D	–	bufor z danymi historycznymi (doświadczeniem)
d_{obs}	–	odległość TCP robota od przeszkody w przestrzeni 3D
d_t	–	sygnał ukończenia epizodu w kroku czasowym t
d_{th}	–	odległość progowa
e	–	błąd pozycjonowania TCP robota w zadanym punkcie g_{xyz}
G	–	skumulowana przyszła nagroda, zwrot
$g_{xyz}, (g_x, g_y, g_z)$	–	zadana pozycja TCP robota
i	–	i-ty element
Q	–	funkcja użyteczności stan-akcja
$Q(s, a\|\theta^Q)$	–	sieć neuronowa aproksymująca funkcję Q sparametryzowaną przez parametry θ^Q, której wejściem jest stan s i akcja a
R	–	funkcja nagrody
r_t	–	nagroda otrzymana w kroku czasowym t
S, A	–	zbiory: stanów, akcji
$(s_i, a_i, r_i, s_{i+1}, d_i)$	–	krotka z doświadczeniem pobierana z bufora
$(s_t, a_t, r_t, s_{t+1}, d_t)$	–	krotka z doświadczeniem zapisywana do bufora
s_t, s_{t+1}	–	stan w kroku czasowym t oraz $t+1$
$s_{xyz}, (s_x, s_y, s_z)$	–	pozycja startowa TCP robota

t	–	dyskretny krok czasowy
V	–	funkcja użyteczności stanu
$v_{xyz}, (v_x, v_y, v_z)$	–	prędkość liniowa TCP robota
α	–	współczynnik uczenia
γ	–	współczynnik dyskontowy
θ^x	–	wagi sieci neuronowej x
θ_n	–	kąt obrotu przegubu n ramienia robota
$\mu(s)$	–	strategia (polityka) deterministyczna
$\mu(s\|\theta^\mu)$	–	sieć neuronowa aproksymująca strategię μ sparametryzowaną przez parametry θ^μ, której wejściem jest stan s
$\pi(a\|s)$	–	strategia (polityka) stochastyczna

Skróty

CHOMP	–	*Covariant Hamiltonian Optimization for Motion Planning*
CNN	–	Konwolucyjne sieci neuronowe (*Convolutional Neural Network*)
DDPG	–	*Deep Deterministic Policy Gradient*
HER	–	*Hindsight Experience Replay*
HRC	–	*Human-robot collaboration*
HRI	–	*Human-robot interaction*
ML	–	uczenie maszynowe (*Machine Learning*)
MSBE	–	błąd średniokwadratowy Bellmana (*Mean Squared Bellman Error*)
MSE	–	błąd średniokwadratowy (*Mean Squared Error*)
PPO	–	*Proximal Policy Optimization*
PRM	–	*Probabilistic Roadmap*
RL	–	uczenie ze wzmocnieniem (*Reinforcement Learning*)
RMS	–	średnia kwadratowa (*Root Mean Square*)
RRT	–	*Rapidly Exploring Random Tree*
SAC	–	*Soft Actor-Critic*
SI	–	sztuczna inteligencja (*Artificial Intelligence*)
TCP	–	punkt środkowy narzędzia (*tool center point*)
TD3	–	*Twin Delayed Deep Deterministic Policy Gradient*
TRPO	–	*Trust Region Policy Optimization*

Uwagi ogólne

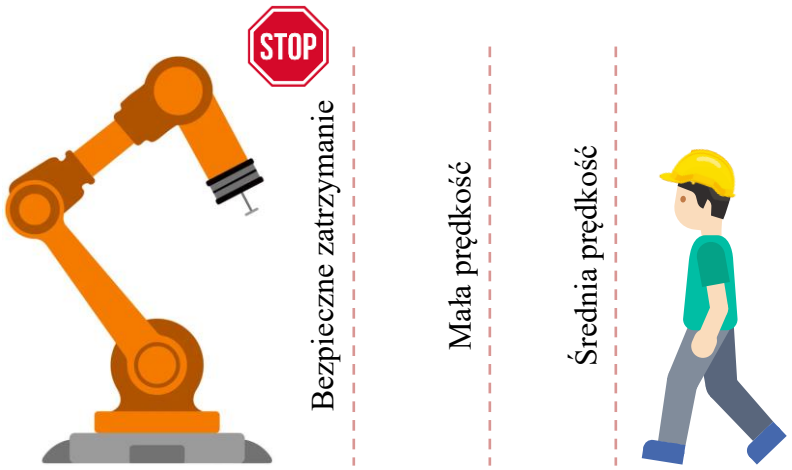
1. W pracy jako separator dziesiętny zastosowano znak kropki, na przykład 1.5m.
2. Rysunki i tabele są domyślnie opracowaniem własnym.

3. Zasoby zaczerpnięte z prac innych Autorów oznaczono przez odwołanie do pozycji literaturowych.
4. Angielskie odpowiedniki polskich pojęć lub nazwy własne na przykład algorytmów, zostały zaznaczone w tekście *pochyloną czcionką.*

1 Wstęp

Postęp techniczny polega między innymi na zwiększaniu poziomu automatyzacji, która dąży do ograniczenia lub całkowitego wyeliminowania udziału człowieka w czynnościach związanych ze sterowaniem różnorodnymi obiektami technicznymi. Obecnie w zakładach przemysłowych coraz częściej stosuje się rożnego rodzaju manipulatory i roboty przemysłowe, które zastępują pracę człowieka. Cechują się one między innymi dużą szybkością wykonywanych ruchów, dobrą powtarzalnością i precyzją działania. Jednak brak im niektórych możliwości, które posiada człowiek to znaczy zdolności do percepcji, rozpoznawania złożonych obiektów i intencji oraz umiejętności podejmowania decyzji dotyczących sterowania w zależności od zmian w otoczeniu.

W przypadku niemożliwości całkowitego wyeliminowania człowieka z procesu produkcyjnego, stosowanym rozwiązaniem jest współpraca człowieka i robota na jednym stanowisku pracy. W takich sytuacjach robot i człowiek kooperują ze sobą i wykonują zadania w tej samej strefie roboczej. W normie ISO 10218-1 [1], dotyczącej wymagań bezpieczeństwa dla robotów przemysłowych, opisano różne modele współpracy człowiek-robot. Norma ta zawiera wymagania dotyczące budowy stanowisk ze wspólną przestrzenią roboczą z człowiekiem. W punkcie 3.4 definiuje ona współpracę operacyjną (*collaborative operation*) pomiędzy człowiekiem, a robotem jako stan, w którym zaprojektowane do tego celu roboty pracują w bezpośredniej kooperacji z człowiekiem, w określonym obszarze roboczym. Definicja wspólnej przestrzeni roboczej (*collaborative worskpace*) podana w punkcie 3.5 jest określana jako przestrzeń, w której robot i człowiek mogą wykonywać zadania jednocześnie podczas operacji produkcyjnych. Według normy [1], w przypadku zastosowania robota do współpracy z człowiekiem, powinien on mieć możliwość poruszania TCP z prędkością mniejszą niż $250\frac{mm}{s}$. Zastosowania, w których człowiek współpracuje z robotem, w terminologii angielskojęzycznej nazywane są *human-robot collaboration* (HRC).



Rys. 1. Metoda monitorowania prędkości i separacji (Speed and Separation Monitoring)

Jeśli robot i człowiek mają współpracować w tej samej przestrzeni, to konieczne jest zapewnienie bezpieczeństwa człowiekowi. Zwykle robot porusza się po zadanej trajektorii ruchu i wykonuje określone zadania, a w przypadku zaistnienia ryzyka kolizji z człowiekiem

albo innym obiektem, musi zareagować w odpowiedni sposób, na przykład omijając przeszkodę. W niniejszej pracy oparto się na metodzie współpracy, która w specyfikacji technicznej [2], dotyczącej robotów współpracujących, oznaczona jest numerem 3 i nazwana jest monitorowaniem prędkości i separacji (*Speed and Separation Monitoring*, SSM). Została ona przedstawiona na Rys. 1. W opisanej w specyfikacji metodzie, człowiek ma dostęp do przestrzeni współpracy podczas pracy robota. Wymagane jest jednak zachowanie odpowiedniej odległości od robota, dla zapewnienia bezpieczeństwa człowieka, a maksymalna prędkość robota jest ustalana w zależności od tej odległości. Jeśli jest ona zbyt mała, to następuje zatrzymanie robota.

W badanym w niniejszej pracy rozwiązaniu, robot oraz człowiek będą pracować w jednej i tej samej strefie roboczej. Prędkość robota ma być ograniczona z uwagi na bezpieczeństwo. Dodatkowo, algorytm sterujący pracą robota ma pozwalać na omijanie przeszkód, które znajdą się w strefie roboczej robota. Dzięki takiemu rozwiązaniu, robot i człowiek będą mogli nieprzerwanie i jednocześnie pracować w tej samej strefie. W przypadku, gdy na zadanej trajektorii ruchu robota znajdzie się człowiek, to zostanie on bezkolizyjnie ominięty, co zapewni wymagane bezpieczeństwo. Dodatkowo, jeśli człowiek będzie w zbyt bliskiej odległości od TCP robota, to robot zostanie zatrzymany. Po odsunięciu człowieka, robot będzie kontynuował swoją pracę.

Obecnie, roboty są najczęściej programowane przez programistę, który wprowadza instrukcje opisujące wykonywanie określonej sekwencji ruchów. Wszystkie możliwe zmiany w środowisku produkcyjnym, które otacza robota muszą być przewidziane w napisanym programie. Do poprawnej i bezkolizyjnej pracy robotów wymagana jest wiedza o samym procesie i dynamice zmian w otoczeniu. Z tego względu programowanie robotów jest trudne. W książce [3], opisano problemy planowania ruchów robota, w celu osiągnięcia zadanych pozycji i wykonania zadań w zależności od rozmieszczenia elementów w polu roboczym. Obecnie stosuje się dwa główne tryby programowania robotów. W trybie *on-line,* programista ręcznie ustawia TCP manipulatora w zadanych punktach (pozycjach) i zapisuje ich współrzędne. W metodzie *off-line* program pisany jest przez programistę za pomocą komputera z wykorzystaniem dedykowanego oprogramowania, a następnie zwykle jest testowany w środowisku symulacyjnym. Po sprawdzeniu poprawności działania jest on przesyłany i uruchamiany na sterowniku robota. W przypadku programowania prostych zadań należy określić w programie zmienne zawierające wiele szczegółów, stałych i parametrów. W większości przypadków, stanowiska zrobotyzowane są ogrodzone bądź otoczone tak zwanymi barierami świetlnymi, które zatrzymują robota w przypadku wkroczenia człowieka do strefy chronionej. W praktyce robot musi pracować w zmiennym środowisku, w którym może wystąpić kolizja z obiektami znajdującymi się w jego polu roboczym, na przykład z ramionami innych robotów. Obiekty te mogą być też elementami używanymi w operacjach montażowych. W takich przypadkach, zadania robota zwykle koncentrują się na jego pozycjonowaniu, chwytaniu elementów i planowaniu płynnych ruchów. W każdym przypadku robot powinien omijać przeszkody. Jeśli pozycja przeszkód w polu roboczym robota jest znana i niezmienna to programista uwzględnia to w kodzie programu, programując bezkolizyjne ich ominięcie przez robota. W przypadku, gdy w polu roboczym mogą pojawić się przeszkody w niezdefiniowanym miejscu, to robot musi być wyposażony w system ich wykrywania, a program sterujący powinien zmienić aktualną trajektorię ruchu na

bezkolizyjną i następnie osiągnąć pozycję zadaną. Przegląd klasycznego planowania i programowania ruchu robotów przedstawiono między innymi w pracach [3], [4].

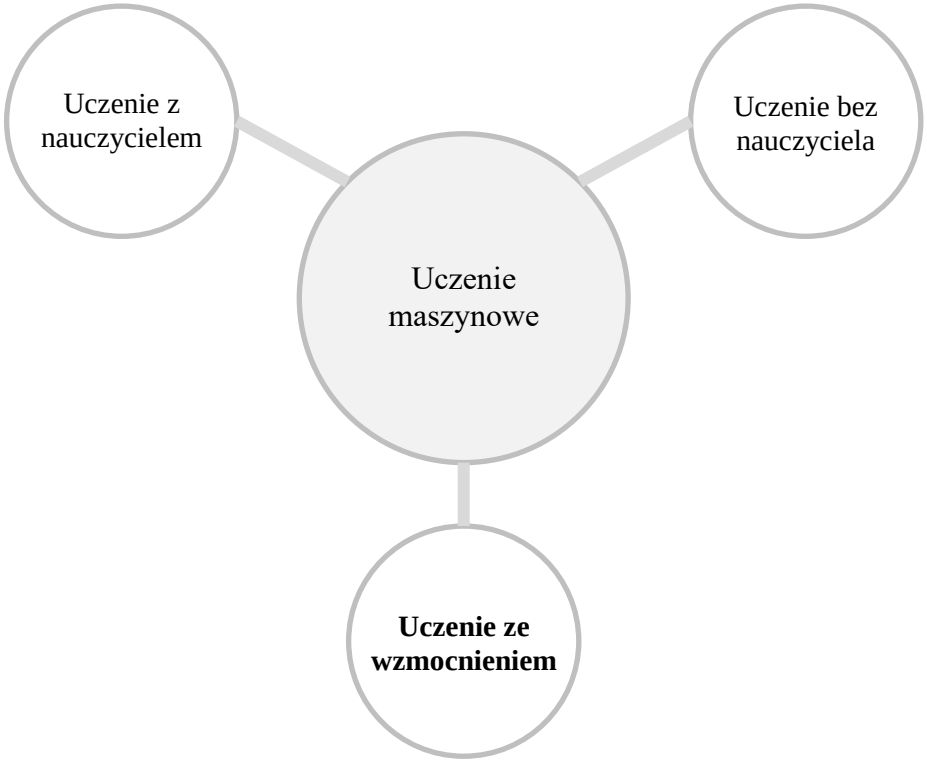
Zdaniem autora niniejszej pracy opracowanie systemów sterowania dających robotom możliwość podejmowania autonomicznych decyzji dotyczących trajektorii ich ruchu, w zależności od zmian w otoczeniu, wydaje się być naturalnym kierunkiem rozwoju metod programowania i sterowania robotami. Nauczenie robota wykonywania określonych czynności, zamiast programowania go za pomocą kolejnych instrukcji jest wyzwaniem, którego rozwiązanie może przyczynić się do znaczącego postępu w programowaniu i zastosowaniu robotów. Takie podejście jest bardzo podobne do nauki pracownika wykonywania określonych czynności albo tylko do wydawania mu określonych poleceń. Możliwość opracowania systemu autonomicznego sterowania robotami daje zastosowanie różnych technik sztucznej inteligencji (SI, *Artificial Intelligence*). Tego typu próby są już od pewnego czasu z powodzeniem wdrażane w różnych dziedzinach techniki. SI w połączeniu z systemami wizyjnymi jest przydatnym narzędziem w operacjach montażu lub pakowania, wykonywanymi przez na przykład manipulatory przemysłowe. Roboty humanoidalne stosowane do obsługi klienta, wykorzystują różne algorytmy związane z przetwarzaniem języka naturalnego do komunikacji z ludźmi. Natomiast bezzałogowe statki powietrzne wykorzystują algorytmy SI do rozpoznawania różnego rodzaju obiektów.

W celu zapewnienia bezpiecznej współpracy człowieka z robotem w tej samej przestrzeni roboczej, w niniejszej pracy podjęto badania nad opracowaniem systemu, pozwalającego na omijanie przeszkód na zadanej trajektorii ruchu robota. Głównym problemem badawczym było zaproponowanie odpowiednich do tego zadania algorytmów. Na podstawie analizy literatury przyjęto, że budowany system będzie oparty o algorytmy uczenia ze wzmocnieniem (*Reinforcement Learning*), które można sklasyfikować jako algorytmy należące do SI. Podstawy teoretyczne związane z tymi algorytmami zostały opisane w książce autorstwa Suttona i Burto [5].

Techniki SI obejmują między innymi takie dziedziny nauki jak uczenie maszynowe (*Machine Learning*) oraz uczenie głębokie (*Deep Learning*). Zagadnienia związane z różnymi technikami SI zostały opisane w książce [6]. Samo uczenie maszynowe można zdefiniować jako dziedzinę nauki o algorytmach, które automatycznie poprawiają swoją skuteczność dzięki zdobywanemu doświadczeniu, poprzez dostarczanie im określonych danych. W przypadku tych algorytmów to człowiek jest odpowiedzialny za wykonanie ekstrakcji cech z danych, a model na przykład za klasyfikację. Uczenie głębokie jest rozwinięciem metod uczenia maszynowego. Bazuje ono na wykorzystaniu głębokich sieci neuronowych (*deep neural network*). Algorytm uczenia sam wykonuje ekstrakcję cech z danych i następnie na przykład klasyfikację. Techniki uczenia głębokiego z powodzeniem są stosowane między innymi do rozpoznawania twarzy, różnych obiektów na obrazach, przetwarzania języka naturalnego, czy też do sterowania samochodami autonomicznymi.

Uczenie ze wzmocnieniem jest jedną z trzech gałęzi uczenia maszynowego co zostało przedstawione na Rys. 2. Różnego rodzaju techniki SI takie jak algorytmy uczenia ze wzmocnieniem lub konwolucyjne sieci neuronowe (*Convolutional Neural Network*, CNN) są znane już od około dwudziestu lat, jednak dopiero w ostatnich latach nastąpił nagły wzrost ich rozwoju oraz zastosowania w różnych gałęziach przemysłu. Można wyróżnić trzy główne przyczyny, które spowodowały ten rozwój. Pierwszą z nich jest komputeryzacja, która

spowodowała generację większej ilości danych przez ludzi oraz urządzenia, zwiększony do nich dostęp oraz możliwość łatwiejszego ich przechowywania i manipulowania nimi. Kolejnym ważnym aspektem rozwoju SI jest zwiększenie mocy obliczeniowej komputerów, w tym opracowanie szybkich kart graficznych i wykorzystanie ich do obliczeń numerycznych. Firma NVIDIA opracowała technologię CUDA [7], która umożliwia wykorzystanie procesorów graficznych do obliczeń równoległych (*parallel computing*). Takie podejście doprowadziło do znacznego przyspieszenia obliczeń w porównaniu do sekwencyjnych procesorów ogólnego zastosowania. Ostatnim czynnikiem przyspieszającym rozwój SI jest opracowanie i rozwój bibliotek programistycznych, przeznaczonych do budowy modeli SI takich jak *Tensorflow* [8], *Keras* [9], czy *PyTorch* [10]. Przełożyło się to na udoskonalanie znanych algorytmów, tworzenie nowych modeli oraz szybkość i prostotę w pracy z algorytmami SI.



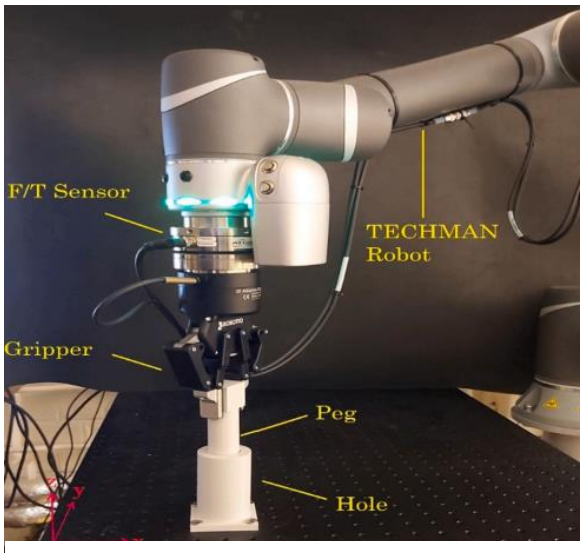
Rys. 2. Podział metod uczenia maszynowego

Zaletą stosowania technik SI jest zapewnienie generalizacji rozwiązywanego problemu. Techniki te znajdują obecnie obszerne zastosowanie w rozwiazywaniu problemów, które dla ludzi są relatywnie proste, takie jak na przykład rozpoznawanie obiektów na zdjęciach czy podejmowanie decyzji w zależności od zaistniałej sytuacji. W przypadku robotów, algorytmy SI mogą zapewnić pewien stopień ich autonomiczności w podejmowaniu decyzji w zmieniającym się środowisku. Dodatkowo, maszyny lub systemy wyposażone w algorytmy SI mogą działać znacznie szybciej, skuteczniej i czasami nawet precyzyjniej od człowieka.

2 Przegląd literatury

2.1 Współpraca robota z człowiekiem

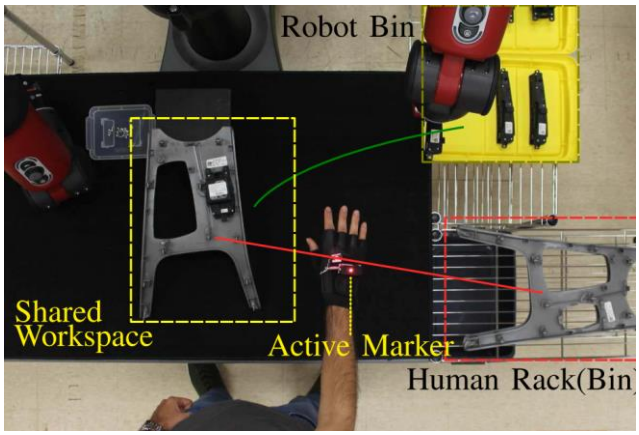
Pojawienie się w ostatnich latach lekkich, elastycznych i przyjaznych człowiekowi robotów z wbudowanymi funkcjami bezpieczeństwa, przyczyniło się do rozwoju zastosowań, w których robot przemysłowy współpracuje z człowiekiem. Te zmiany poprawiają interakcję człowiek-robot do tego stopnia, że niekiedy roboty są postrzegane jako istoty społeczne, z którymi ludzie wchodzą w interakcje [11]. Przegląd stanu wiedzy dotyczący interakcji robotów z ludźmi przedstawiono w [12]. Zamieszczono tam zarys postępów w dziedzinie poprawy bezpieczeństwa ludzi, poruszając tematy związane z analizą możliwych urazów i normami bezpieczeństwa. Następnie zaprezentowano podstawy projektowania robotów bezpiecznych dla człowieka. Omówiono również techniki planowania ruchu robota, zapewniające bezpieczeństwo człowieka. Praca [13] zawiera podsumowanie kryteriów projektowania bezpieczniejszych robotów i omawia strategie zapewniające bezpieczne wykonywanie przez nie zadań, w obecności ludzi. Unikanie kolizji jest podstawowym wymogiem dla bezpiecznej interakcji robota i człowieka. Temat ten został szeroko przebadany w robotyce oraz opisany w opracowaniu [14].



Rys. 3. Platforma eksperymentalna do demontażu walca z otworu [15]

W artykule [15] opisano opracowane rozwiązanie do współpracy robota z człowiekiem w procesie demontażu. Autorzy zaproponowali model matematyczny procesu technologicznego oraz metody genetyczne do optymalizacji jego parametrów. Model był testowany na zadaniu, które polegało na demontażu cylindra z otworu (Rys. 3). Współpraca robota z człowiekiem w zadaniu, jakim był demontaż pompy paliwa, polegała na określeniu w czasie, ich dostępu do stanowiska pracy. Opracowany algorytm został przedstawiony w formie sekwencyjnego wykresu Ganta.

Autorzy artykułu [16] opracowali sposób postępowania (sterowania) przy interakcji robota z człowiekiem w procesie montażu. Algorytm przydzielał zadania i planował proces montażu. Składał się on z dwuwarstwowego tak zwanego planera ruchów. Jedna warstwa generowała skoordynowaną sekwencję ruchów dla zespołu człowiek-robot na podstawie wielosystemowego modelu środowiska. Druga warstwa składała się z planera tworzącego maszynę stanu, która była odpowiedzialna tylko za ruchy przegubów robota. Tak zaprojektowany system, według Autorów, był w stanie działać poprawnie w trudnych do przewidzenia sytuacjach w trakcie kooperacji robota i człowieka. W pracy [17] przedstawiono rozwiązanie, w którym człowiek i robot, określani jako agenci, współpracowali w tej samej celi montażowej i wykonywali zadania takie jak podnoszenie, przenoszenie i puszczanie. Były one przypisywane sekwencyjne do człowieka albo do robota na podstawie ich

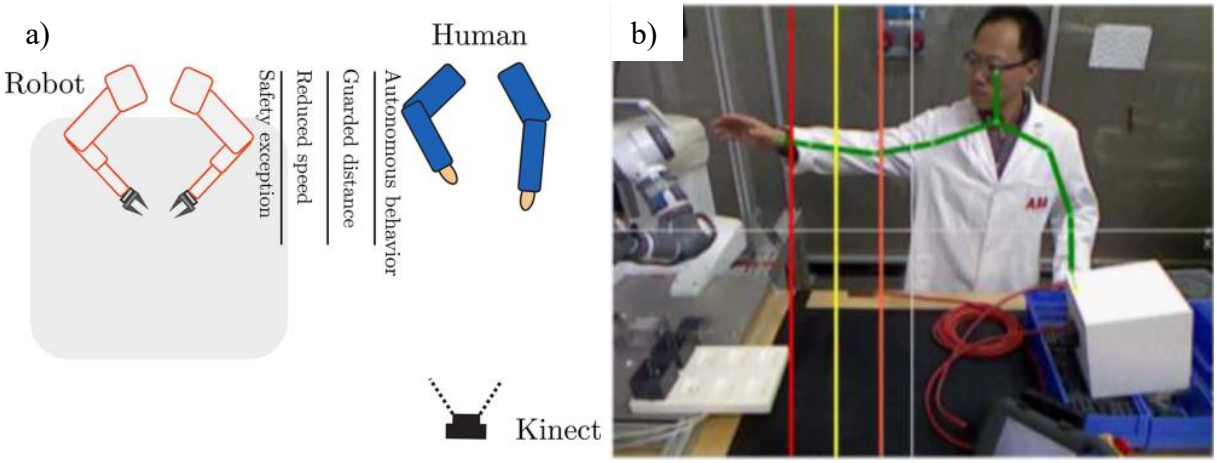


Rys. 4. Stanowisko pracy do montażu elementów samochodu osobowego [18]

dostępności oraz czasu operacji, potrzebnego każdemu agentowi do wykonania zadania. W ramach tego rozwiązania człowiek i robot nie wykonywali prac jednocześnie, a ich interakcja była ograniczona ze względu na bezpieczeństwo.

W artykule [18] zaproponowano rozwiązanie, w którym robot współpracował z człowiekiem w zadaniu montażu części samochodów osobowych w określonej strefie pokazanej na Rys. 4. Człowiek wyposażony był w rękawiczki z markerami, które były wykrywane przez dwa systemy wizyjne. Dzięki takiemu rozwiązaniu, algorytm sterujący robotem „znał" położenie rąk człowieka i mógł bezkolizyjnie sterować pracą robota. W zaproponowanym rozwiązaniu, we wspólnej strefie, w określonej chwili czasowej mógł znajdować się tylko człowiek albo robot. Dodatkowo, Autorzy pracy zaimplementowali emocje robota takie jak na przykład niepewność, przedstawiane na ekranie w formie zmieniającego się obrazu twarzy. Zdaniem Autorów przełożyło się to na zwiększenie zaufania człowieka do robota, co przyspieszyło jego pracę.

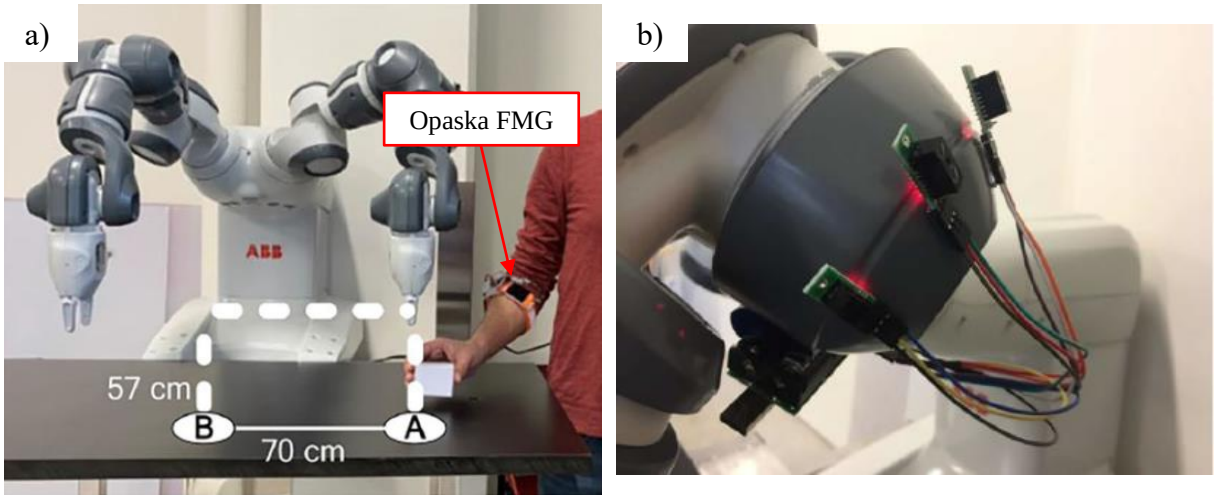
Autorzy prac [19] oraz [20] zaproponowali stanowisko do współpracy człowieka z robotem firmy ABB o dwóch ramionach. Przestawione na Rys. 5a stanowisko pracy było podzielone na strefy. Prędkość robota była zależna od jego odległości od człowieka. Pozycja człowieka była wyznaczana na podstawie danych z kamery RGBD – Microsoft Kinect. Jeśli człowiek znalazł się w niebezpiecznej, czerwonej strefie, to algorytm sterujący pracą robota zatrzymywał go, żeby zapewnić bezpieczeństwo człowiekowi. Przykład przekroczenia niebezpiecznej strefy przez człowieka został zilustrowany na Rys. 5b.



Rys. 5. Stanowisko do współpracy robota z człowiekiem: a) – schemat z podziałem na odrębne strefy, b) – obraz z kamery pokazujący przekroczenie przez człowieka niebezpiecznej strefy [19], [20]

Kolejny przykład zastosowania robota z dwoma ramionami do współpracy z człowiekiem opisano w artykule [21]. Autorzy pracy zaproponowali system, składający się z podwójnego

manipulatora do asystowania i współpracy z człowiekiem w trakcie montażu zawieszenia samochodów osobowych. Zadanie robota polegało na wykonywaniu operacji manipulacyjnych, na przykład przykręcania śrub. Bezpieczeństwo człowieka było zapewnione dzięki pracy sekwencyjnej, polegające na tym, że w strefie współdzielonej mógł pracować robot albo człowiek. W artykule [22] opisano autonomicznego robota mobilnego, z zamontowanym 4-osiowym ramieniem. Robot ten miał za zadanie wykrywać i sortować puszki ze względu na ich rozmiar, różne defekty i kolory. W trakcie tego zadania współpracował on z operatorem, którego wykrywał z wykorzystaniem systemu wizyjnego. W pracy [23] opisano współpracę dwóch robotów w środowisku symulacyjnym. Sterowanie tą współpracą posłużyło do sprawdzenia potencjalnego zastosowania opracowanego systemu do współpracy robota z człowiekiem. Autorzy zastosowali dwa roboty typu Turtlebot z ramieniem Phantom X. Jeden robot był nauczycielem, a drugi obserwatorem, który uczył się obserwując sposób wykonywania zadania, jakim było sortowanie dojrzałych i niedojrzałych owoców (kolorowych piłek). Po określonym czasie robot-obserwator zaczynał współpracować z pierwszym robotem w jednej strefie roboczej. Dwa współpracujące roboty wykonały zadanie sortowania dwa razy szybciej w porównaniu do jednego robota.



Rys. 6. System do współpracy człowieka z robotem dwuramiennym: a) – przenoszenie obiektu z punktu A do B, b) – czujniki zamontowane na przegubie robota służące do wykrywania obiektów w odległości 5cm [24]

Nowatorskie rozwiązanie w zakresie współpracy robota i człowieka zostało opisane w [24]. Autorzy opracowali opaskę FMG (*force myography*), która była zamocowana na przedramieniu człowieka (Rys. 6a). Odczytywała ona sygnały z mięśni, które były wykorzystywane do rozpoznawania intencji ruchów człowieka, za pomocą systemu z rekurencyjnymi sieciami neuronowymi. Zastosowanie takiej opaski miało na celu zmniejszenie ryzyka kolizji pomiędzy człowiekiem a robotem. Dodatkowo, Autorzy zastosowali trzy czujniki odległości przedstawione na Rys. 6b, które monitorowały strefę w odległości 5cm od ramienia robota. Przetestowane zostały różne scenariusze ewaluujące system w momencie, kiedy człowiek przenosił obiekt lub dotykał robota. Zaproponowany algorytm do rozpoznawania ruchów człowieka osiągnął wynik parametru F1 na poziomie 0.9. Parametr ten został opisany w publikacji [25]. Inny przykład wykrywania intencji człowieka współpracującego z robotem opisano w pracy [26]. Autorzy wykorzystali kamery do

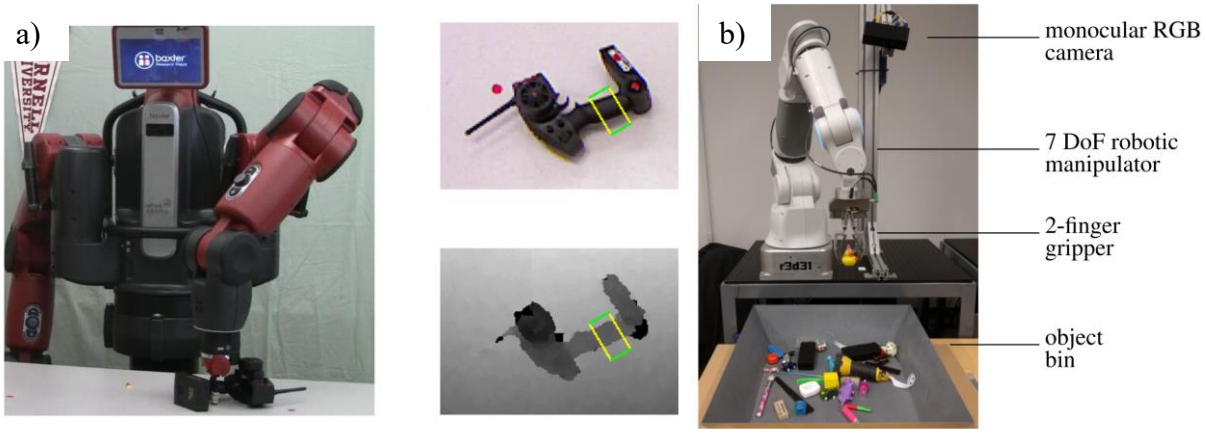
zbierania danych dotyczących pozycji człowieka. Przedstawiony algorytm sterujący pracą robota, rozpoznawał jakie zadanie wykonuje człowiek i rozpoczynał swoje zadanie mające taki sam cel. Dodatkowo, algorytm był w stanie wykrywać określone wzorce ruchu ręki i gesty operatora.

2.2 Algorytmy sztucznej inteligencji w sterowaniu robotami

Już w latach 90 poprzedniego wieku w literaturze opisywano wiele prób wykorzystania uczenia maszynowego w robotach. Rożne podejścia do sterowania robotami z zastosowaniem algorytmów sztucznej inteligencji zostały opisane w pracy [27]. Przedstawiono i porównano w niej metody uczenia robotów: uczenie indukcyjne (*inductive learning*), uczenie oparte na demonstracji, uczenie ze wzmocnieniem oraz uczenie oparte o algorytmy ewolucyjne. Jednym z przykładów jest technika uczenia nieparametrycznego, wykorzystywana do uczenia robota żonglowania, opisana przez Schaala i Atkesona w pracy [28]. W kolejnych badaniach ci sami Autorzy wykorzystali dane z demonstracji do nauki ramienia robota ruchu huśtania się wahadła [29]. Doszli do wniosku, że taka metoda jest odpowiednia do zastosowań takich jak zgrzewanie punktowe, gdzie robot pracuje tylko jako powtarzalny automat, ale nie może być odpowiednia do bardziej skomplikowanych zadań na przykład sterowania ramieniem robota, do którego TCP zostało przymocowane wahadło. W artykule [30] omówiono najważniejsze metody uczenia robotów, takie jak: bezpośrednie i pośrednie (oparte na modelu, nieoparte na modelu), uczenie imitacyjne, uczenie ze wzmocnieniem, uczenie z nauczycielem.

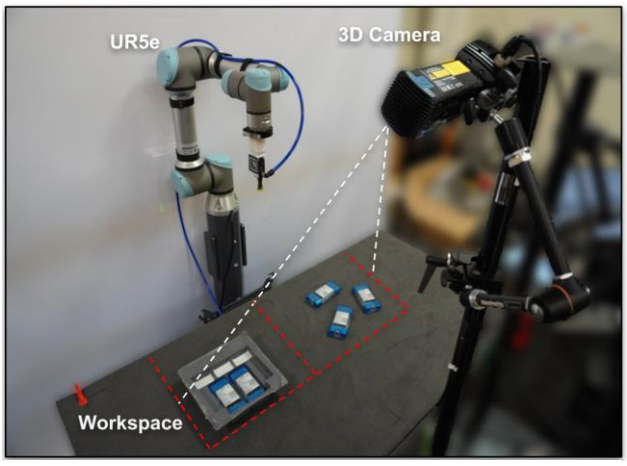
W ostatnich kilku latach znacznie rozwinięto różne architektury sieci neuronowych. Do najważniejszych można zaliczyć sieci konwolucyjne takie jak modele incepcyjne [31] i rezydualne [32], które pozwoliły na klasyfikacje obiektów na podstawie danych ze zdjęć. W dziedzinie wizji maszynowej powstały takie algorytmy jak na przykład *YOLO* [33]–[35], przeznaczone do detekcji obiektów oraz różne algorytmy do semantycznej segmentacji [36]. W przypadku przetwarzania języka naturalnego duży postęp osiągnęła firma Microsoft, projektując głęboko uczony model językowy *Turing-NLG* [37] oraz firma OpenAI budując model *GPT-3* [38]. W dziedzinie strategii trenowania sieci nastąpił również duży postęp, polegający na opracowaniu różnych algorytmów do optymalizacji parametrów, na przykład *Adam* [39] oraz usprawniających naukę, takich jak normalizacja porcji danych (*Batch Normalization*) [40] albo *Dropout* [41].

Ze względu na wzrost zainteresowań w zastosowaniu systemów wizyjnych, w ostatnich latach klasyczne metody sterowania łączy się z algorytmami uczenia maszynowego w obszarze wizji maszynowej. Konwolucyjne sieci neuronowe są wykorzystywane do oceny stanu, w jakim znajduje się robot lub wykrycia pozycji i orientacji obiektu, który ma być chwycony przez robota. W artykule [42] dotyczącym wykorzystania uczenia głębokiego do chwytania obiektów codziennego użytku przez robota, opisano podejście oparte o głębokie konwolucyjne sieci neuronowe. Do wykrywania obiektów została zastosowana kamera RGBD. Autorzy wykorzystali sieci neuronowe jako estymator położenia i orientacji obiektów, który miał za zadanie być ekstraktorem cech. Na podstawie tych cech była określana odpowiednia metoda i miejsce chwytu obiektu przez robota (Rys. 7a). System został zaimplementowany na dwóch różnych robotach: Baxter oraz PR2. Uzyskano współczynniki sukcesu chwytania obiektów odpowiednio 84% i 89%.



Rys. 7. a) – Robot Baxter w trakcie chwytania joysticka oraz obraz z systemu wizyjnego z zaznaczonym miejscem chwytu za pomocą prostokąta [42]. b) – Stanowisko z robotem i kamerą do zbierania danych i nauki chwytania przedmiotów [43]

Do trenowania algorytmów sztucznej inteligencji, bazujących na sieciach głęboko uczonych wymagana jest duża liczba danych treningowych. Autorzy pracy [43] w podobnym zastosowaniu jak opisano w artykule [42], wykorzystali konwolucyjne sieci neuronowe do nauki koordynacji ręka-oko (kamera) robota w chwytaniu różnych przedmiotów. Do badań wykorzystano 14 robotów i kamer z jednym obiektywem. Autorzy utworzyli zbiór danych z 800 tysiącami prób chwytania obiektów. Jedno ze stanowisk z robotem i kamerą zostało przedstawione na Rys. 7b. Badania eksperymentalne wykazały, że zaproponowana metoda zapewniała skuteczne sterowanie chwytaniem w czasie rzeczywistym. Robot z powodzeniem był w stanie uchwycić nowe obiekty i korygować błędy przez obserwacje i dostarczanie nowych danych. W zależności od rozmiarów zbioru treningowego, uzyskano 90% współczynnik sukcesu.



Rys. 8. Stanowisko badawcze dla algorytmu Form2Fit [44]

Ciekawym przykładem zastosowania uczenia głębokiego jest algorytm *Form2Fit* przedstawiony w pracy [44], przeznaczony do montażu elementów, na przykład układania dezodorantów w pudełku. Autorzy algorytmu wykorzystali trzy moduły sieci neuronowych. Pierwszy moduł był odpowiedzialny za chwycenie przedmiotu (przyssawką), drugi za odpowiednie ułożenie elementu w miejscu docelowym, a ostatni za pozycję i orientację układanego obiektu. Stanowisko badawcze zostało przestawione na Rys. 8. Opracowany system był w stanie montować elementy z określoną kolejnością oraz o takich kształtach, na których nie był trenowany, na przykład puzzle. Dla znanych obiektów Autorzy uzyskali współczynnik sukcesu równy 94%, natomiast dla nowych konfiguracji i przedmiotów ten współczynnik wynosił 86%.

2.3 Zastosowanie algorytmów uczenia ze wzmocnieniem w robotyce

Metody oparte o naukę na podstawie prób i błędów i bazujące na sterowaniu optymalnym zostały opracowane w latach 50-tych XX wieku. Ich podwaliny teoretyczne stworzył między innymi Richard Bellman. Badania nad rozwojem i zastosowaniem metod uczenia ze wzmocnieniem pokazały, że są one skuteczne w rozwiązywaniu różnych, złożonych problemów sterowania. Przykładem są wyniki badań opisane przez Maesa i Brooksa [45]. Zaproponowali oni proces nauki poruszania się sześcionożnego robota, oparty na pozytywnych i negatywnych nagrodach (sprzężeniach zwrotnych) wyznaczanych na podstawie sygnałów z otoczenia. W pracy [46], opisano zastosowanie technik uczenia ze wzmocnieniem, do nauki pchania pudełka przez robota mobilnego. Robot nauczył się wykonywać lepiej to zadanie, wykorzystując optymalizowaną przez siebie strategię, w porównaniu ze strategią zaprogramowaną „ręcznie” przez człowieka.

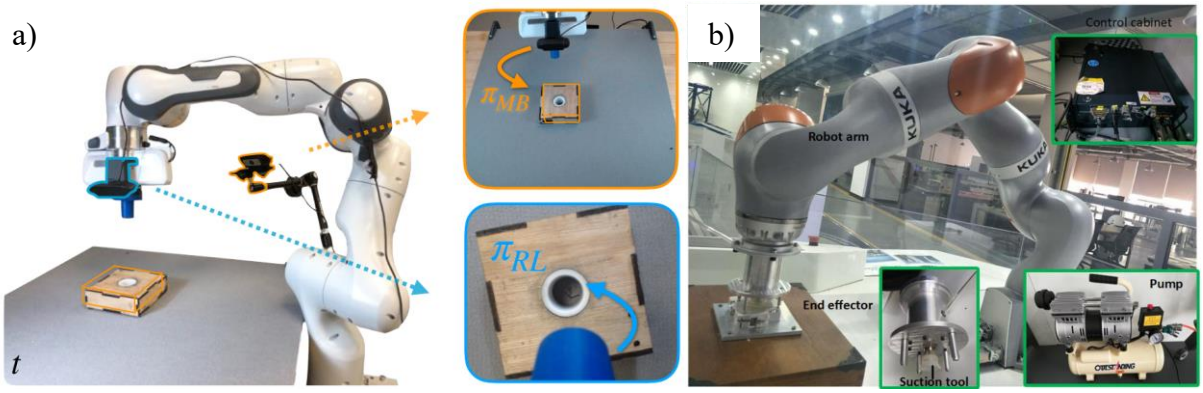
Kober [47] dokonał przeglądu stanu techniki na rok 2013 w zakresie zastosowania technik uczenia ze wzmocnieniem w robotyce. W pracy tej zwrócono uwagę na wyzwania w zastosowaniach tych technik oraz uzyskane sukcesy. Omówiono w niej również zasady wyboru między metodami opartymi na modelu oraz nieopartymi na modelu. Opisano także metody bazujące na strategii oraz na nauce funkcji użyteczności. Przedstawiono również ogromny potencjał metod uczenia ze wzmocnieniem w przyszłych badaniach. W podsumowaniu stwierdzono, że wykorzystanie algorytmów uczenia ze wzmocnieniem w programowaniu robotów nie jest łatwe i wymaga od programisty znajomości zaawansowanej wiedzy i umiejętności. Plappert w artykule [48] zwrócił uwagę na trudności w wykorzystaniu algorytmów uczenia ze wzmocnieniem w robotyce, w przypadku zastosowania ich do wykonywania zróżnicowanych czynności, jedna po drugiej (*Multi-Goal Reinforcement Learning*). W opracowaniu [49] przedstawiono przegląd ostatnich osiągnieć w dziedzinie głębokiego uczenia ze wzmocnieniem (*Deep Reinforcement Leraning*). Omówiono teoretyczne podstawy uczenia maszynowego, uczenia głębokiego oraz uczenia ze wzmocnieniem. Przedstawiono podstawowe informacje o tym jak działają algorytmy uczenia ze wzmocnieniem. Opisano również szereg ich zastosowań w dziedzinie robotyki, gier komputerowych, przetwarzania języka naturalnego, wizji maszynowej, zarządzania, finansów, edukacji, medycyny, przemysłu 4.0 oraz transportu.

W literaturze przedstawiono wiele przykładów zastosowania algorytmów uczenia ze wzmocnieniem, ale głównie w badaniach symulacyjnych. Używano je do sterownia uproszczonymi modelami robotów na przykład robota humanoidalnego, dwu oraz czteronożnego, modelu helikoptera, dżdżownicy i innych [50]–[54]. Algorytmy uczenia ze wzmocnieniem stosowano do nauki umiejętności związanych z poruszaniem się modeli robotów kroczących i do pokonywania nierówności terenu [55]–[57]. Niektóre z tych zadań są jednak trudne do adaptacji w rzeczywistym świecie. Wykonane badania pokazały, że nauczenie modelu robota kroczącego chodzenia i biegania, nie może być łatwo i bezpośrednio zaimplementowane i zaadaptowane na rzeczywistym odpowiedniku. Podstawową trudność stanowi wprowadzenie do układu zmieniających się w czasie parametrów takich jak masa, tłumienie, tarcie itd. Problemy występują również w przypadku modelowania w środowisku symulacyjnym danych z zastosowanych kamer i systemów wizyjnych. Do rozwiązania tych problemów w literaturze zaproponowano zastosowanie różnych technik jak na przykład

Domain Randomization [58], [59], *Domain Adaptation* [60], [61] oraz zastosowanie sieci adaptacyjnych (*Randomized-to-Canonical Adaptation Networks*) [62]. Techniki te znacząco przyczyniły się do zwiększenia skuteczności implementacji różnych technik sztucznej inteligencji w rzeczywistych rozwiązaniach.

W publikacji [63] przedstawiono wyniki badań mających na celu ewaluację różnych algorytmów uczenia ze wzmocnieniem, zastosowanych do sterowania rzeczywistymi obiektami. Testy wykonano na trzech typach urządzeń: manipulator UR5, napęd Dynamixel MX-64AT oraz robot mobilny iRobot Create2. W zależności od rodzaju urządzenia, zadaniem było osiągnięcie zadanego położenia, śledzenie zadanego punktu albo ruch z określonymi założeniami lub dokowanie. Badaniom poddano cztery algorytmy uczenia ze wzmocnieniem. Testy wykazały, że osiągnęły one zadowalające rezultaty.

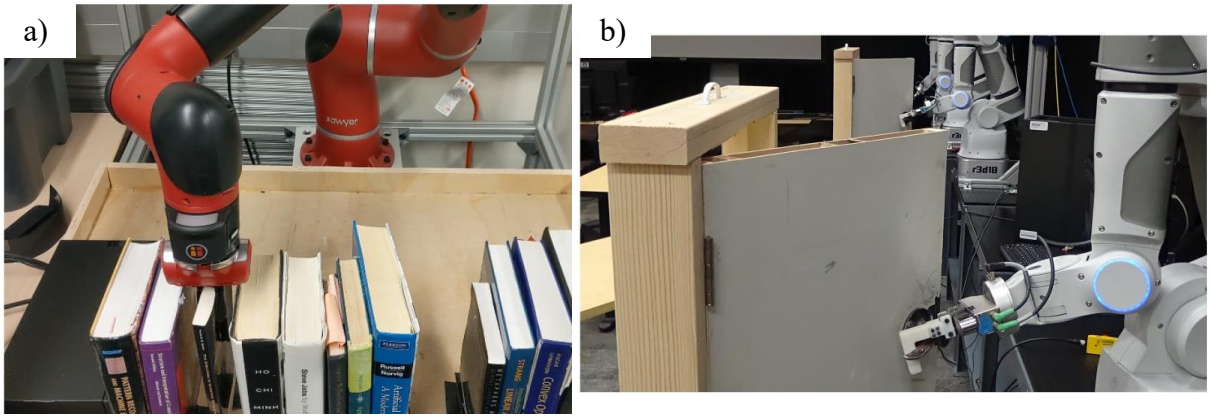
Jednym z przykładów zastosowań algorytmów uczenia ze wzmocnieniem jest montaż elementów. W artykule [64] zaproponowano algorytm do montażu cylindra w prostopadłościanie z otworem. Dane o położeniu elementu docelowego były dostarczane z wykorzystaniem dwóch kamer, jednej umieszczonej na robocie, drugiej na statywie (Rys. 9a). Po nauczeniu, algorytm poprawnie umieszczał cylinder w otworze dla 93% przypadków. Inny przykład zastosowania algorytmów uczenia ze wzmocnieniem w montażu opisano w pracy [65]. Zaprojektowano stanowisko do montażu bezpieczników elektronicznych przedstawione na Rys. 9b. Algorytm uczenia ze wzmocnieniem miał 94% skuteczności montażu bezpiecznika. Dla porównania, metoda bazująca na technice *Extreme Learning Machine* (ELM) miała tylko 56% skuteczność.



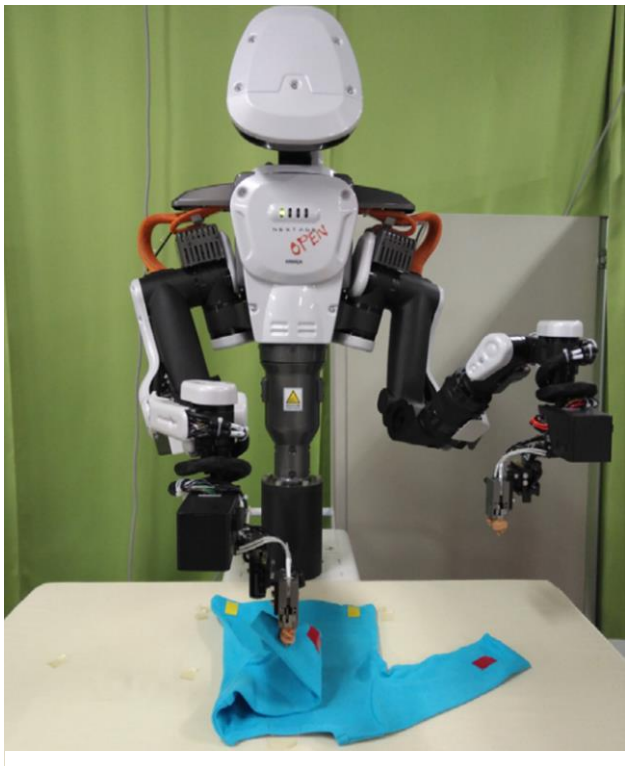
Rys. 9. a) – Stanowisko do montażu cylindra w prostopadłościanie z otworem z dwoma kamerami zaznaczonymi kolorem niebieskim i pomarańczowym [64]. b) – Stanowisko do montażu bezpieczników elektronicznych [65]

Kolejnym przykładem zastosowania algorytmów uczenia ze wzmocnieniem jest sterowanie robotem wykonującym domowe czynności takie jak: zamykanie drzwi, układanie książek na półce itp. W artykule [66] zaproponowano zmodyfikowaną metodykę opartą o algorytmy uczenia ze wzmocnieniem, ale bez funkcji nagrody. Człowiek oceniał, czy zadanie jakim było chwycenie kubka, odłożenie książki na półkę (Rys. 10a) albo otwarcie drzwi zostało wykonane poprawnie. Zaproponowane podejście skutkowało 100% współczynnikiem sukcesu. Podobne badania dotyczące zadań typu otwieranie drzwi (Rys. 10b) lub *pick-and-place* przedstawiono w publikacji [67]. Autorzy do trenowania algorytmu wykorzystali tak zwane podejście asynchroniczne, trenując kilka instancji algorytmów jednocześnie. Pozwoliło

to na przyspieszenie treningu i osiągnięcia zadowalających rezultatów. Uzyskano współczynnik sukcesu, w zależności od zadania, w zakresie od 85% do 95%. Należy podkreślić, że algorytmy uczyły się bez danych demonstracyjnych, tylko na podstawie zdobywanego doświadczenia. Podobne podejście dotyczące wykorzystania metod asynchronicznych do trenowania algorytmów uczenia ze wzmocnieniem zostało opisane w artykule [68]. Przedstawiono w nim badania porównawcze dla różnych zadań w domenie symulacyjnej.



Rys. 10. a) – Robot układający książki na półce [66]. b) – Robot otwierający drzwi [67]



Rys. 11. Robot składający T-shirt. [69]

Inne praktyczne zastosowanie algorytmów uczenia ze wzmocnieniem do wykonywania prac domowych przez roboty przedstawiono w pracy [69]. Zastosowano sztuczną inteligencję do sterowania robotem w zadaniu składania rzeczy. Testy wykonano dla koszulki (Rys. 11) oraz chusteczki. Do estymacji położenia składanego obiektu oraz do ekstrakcji cech wymaganych do podjęcia określonej akcji, zastosowano konwolucyjne sieci neuronowe. Zadanie zostało uproszczone do dyskretnych punktów w przestrzeni, do których chwytak robota miał się przemieścić i wykonać odpowiednią akcję chwycenia lub puszczenia. Autorzy stwierdzili, że zaproponowana metoda cechowała się lepszą skutecznością niż wszystkie inne opracowane do tej pory. W pracy [70] testowano algorytm bazujący na technikach uczenia ze wzmocnieniem i wizji maszynowej, przeznaczony do wykonywania czterech zadań przez ramię robota (Rys. 12). Było to: wieszanie wieszaka do ubrań na stojaku, dopasowanie klocka o określonym

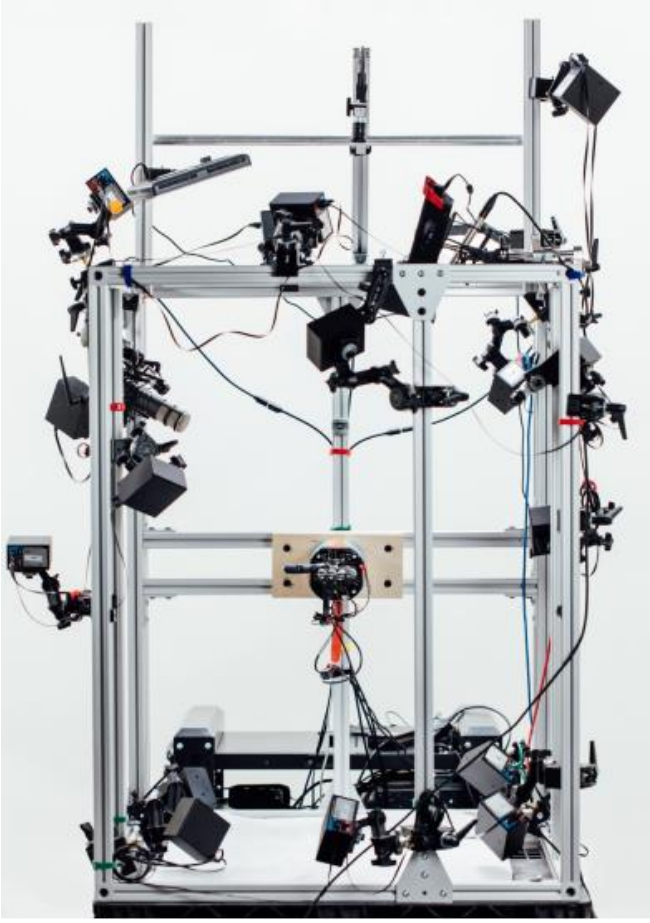
kształcie do otworu, wyciągnięcie gwoździa młotkiem i zakręcenie butelki. Dla tych zadań uzyskano następujące współczynniki sukcesu odpowiednio: 100%, 87.5%, 78.3% i 62.5%.

Rys. 12. Zadania wykonywane przez robota przedstawione w publikacji [70]. Od lewej: wieszak, dopasowanie klocka, młotek, zakręcenie butelki [70]

Zadania typu *push* i *pick-and-place*, w których do sterowania robotami wykorzystano algorytmy uczenia ze wzmocnieniem zostały zaprezentowane w publikacjach [71], [72] i [73]. W pracy [71] opisano badania symulacyjne, w których do przenoszenia określonych przedmiotów wykorzystano model robota Kuka. Autorzy przetestowali i porównali dwa różne algorytmy uczenia ze wzmocnieniem – *Deep Deterministic Policy Gradient* (DDPG) oraz *Proximal Policy Optimization* (PPO). Dla środowiska bez przeszkody w polu roboczym robota uzyskano współczynniki sukcesu równe odpowiednio 48% i 75%. Dla środowiska z przeszkodą w postaci ściany, co utrudniało proces chwytania, uzyskano współczynniki równe 80% i 18%. W artykule [72] przedstawiono wyniki badań doświadczalnych, w których zastosowano robota UR5 do chwytania różnych obiektów w nieuporządkowanym i zaśmieconym środowisku. Zaprojektowany algorytm bazujący na głębokim uczeniu ze wzmocnieniem oceniał, czy robot będzie w stanie chwycić określony obiekt. Jeśli nie było to możliwe, to robot zmieniał ułożenie elementów poprzez ich popchnięcie. Do systemu oceny możliwości chwytu zastosowano kamerę RGBD, z której obraz był przetwarzany przez konwolucyjne sieci neuronowe. Uzyskano współczynnik sukcesu równy 71%. W publikacji [73] zaproponowano system do chwytania obiektów o różnorodnych kształtach i rozmiarach, który uzyskał 96% skuteczności. Jest to usprawnienie systemu przedstawionego w pracy [43], wykorzystujące do sterowania pracą robota oprócz konwolucyjnych sieci neuronowych, także algorytmy uczenia ze wzmocnieniem.

Przykład zastosowania algorytmów uczenia ze wzmocnieniem do pozycjonowania robota, został przedstawiony w publikacji [74]. Autorzy zaprojektowali algorytm uczenia oparty na kooperatywnym systemie wieloagentowym, sterującym 6-osiowym robotem antropomorficznym firmy Baxter. Algorytm został opracowany w środowisku symulacyjnym, w którym robot wykonywał ruchy na planarnej, dyskretnej przestrzeni. Średnia dokładność pozycjonowania w warunkach symulacyjnych wynosiła 5.6mm. Autorzy stwierdzili, że algorytm był powtarzalny. W warunkach laboratoryjnych średni błąd pozycjonowania wynosił 8mm. Kolejnym przykładem zastosowania głębokich technik uczenia ze wzmocnieniem do sterowania i planowania ruchu ramienia robota podano w pracy [75]. Metody uczenia ze wzmocnieniem są również stosowane do nauki kinematyki odwrotnej manipulatorów. Przykłady takich zastosowań zostały opisane w pracach [76], [77], [78]

Ciekawym zastosowaniem metod uczenia ze wzmocnieniem jest opisany w [79] algorytm, który sterował ramieniem robota, podczas chwytania określonych obiektów i rzucania nimi tak, żeby trafić do odpowiedniej szuflady. Algorytm był w stanie ocenić, w jaki sposób chwycić obiekt o określonym kształcie oraz w jaki sposób wykonać manewr rzutu, czyli dobrać prędkość ruchu robota oraz moment puszczenia przedmiotu.



Rys. 13. Stanowisko z 19-toma kamerami służące do nauki bionicznej ręki zręcznego manipulowania różnymi obiektami [80]

Techniki uczenia ze wzmocnieniem nie są tylko stosowane do sterowania ramionami robotów, ale również innymi obiektami. Naukowcy z firmy OpenAI wykorzystali je do nauki manipulowania obiektami przez bioniczną rękę o 24 stopniach swobody [80]. Wstępne badania oraz trenowanie algorytmu było wykonywane w symulacji. Algorytm sterujący pracą ręki uczył się manipulować między innymi sześcienną kostką, długopisem czy jajkiem. Wyzwaniem okazało się zaimplementowanie wytrenowanego w domenie symulacyjnej algorytmu na rzeczywistym obiekcie. W tym celu wykonano stanowisko składające się łącznie z 19 kamer przedstawione na Rys. 13. Zastosowano techniki *Domain Randomization* [58], [59]. Do estymacji położenia manipulowanego obiektu wykorzystano konwolucyjne oraz rekurencyjne sieci neuronowe. Błąd obrotu dla sześciennej kostki wynosił około 3°. Inne przykłady, w których zastosowano techniki uczenia ze wzmocnieniem do sterowania ruchami bionicznej ręki opisano w [81]–[84].

Artykuły [85] i [86] dotyczą sterowania czworonożnym robotem z zastosowaniem uczenia ze wzmocnieniem. Przedstawiono w nich koncepcję zespołu algorytmów, które pozwoliły robotowi na wykonywanie czynności związanych z przemieszczaniem i poruszaniem się po nierównym terenie. Sterowany tym algorytmem robot potrafił omijać przeszkody, a po wywróceniu się robota, potrafił także tak sterować napędami, aby robot wstał. Kolejnym przykładem zastosowania metod uczenia ze wzmocnieniem do sterowania i nauki sprawności ruchowej są badania opisane w pracach [87], [88]. Autorzy zaproponowali rozwiązanie, w którym model dinozaura T-Rexa w domenie symulacyjnej uczył się chodzić, a następnie dryblować piłką. Inny ciekawy przykład sterowania modelem robota humanoidalnego podano w artykule [89]. Robot miał za zadanie nauczyć się wykonywać różne figury akrobatyczne, oglądając filmy z serwisu YouTube. Badania wykazały, że w

zależności od figury akrobatycznej, algorytm osiągnął współczynnik sukcesu w zakresie od 70% do 80%.

Algorytmy uczenia ze wzmocnieniem są również stosowane do sterowania robotami mobilnymi. W artykule [90] opisano rozwiązanie, w którym bezpośrednio agent algorytmu *Q-learning* sterował ruchem robota mobilnego. Celem robota było przemieszczenie się do punktu docelowego, przy jednoczesnym omijaniu przeszkód. Nagroda o wartości +1 była stosowana, jeśli robot osiągnął cel. Za zderzenie z przeszkodą, przyznawana była nagroda (kara) o wartości -1. We wszystkich innych przypadkach nagroda wynosiła 0. Robot był w stanie osiągnąć pozycję zadaną w 95% wszystkich testów. Kolejne przykłady zastosowania technik uczenia ze wzmocnieniem podano w artykułach [91] i [92], w których badano możliwości nawigacji robotami mobilnymi w labiryncie. Inne przykłady zastosowań metod uczenia ze wzmocnieniem do projektowania planerów omijających przeszkody przez roboty mobilne przedstawiono w pracach [93] i [94].

Rys. 14. Agenci grający w chowanego [openai.com]

Bardzo interesujące zastosowanie algorytmów uczenia ze wzmocnieniem do kontroli agentów, rywalizujących w prostej grze w chowanego, przedstawiono w pracy [95]. Do wytrenowania algorytmów zaproponowano zdywersyfikowane środowisko oparte na rywalizacji i kooperacji. W procesie uczenia wykorzystano tylko funkcję nagrody, opartą na widoczności i rywalizacji. Wyniki pokazały, że agenci nauczyli się wielu nowych umiejętności i strategii, co zaskoczyło nawet samych twórców. W jednym ze środowisk, trenowani agenci nauczyli się współpracować ze sobą, żeby je zmieniać i dopasować do swoich potrzeb. Dwóch agentów, którzy się chowali wypracowali strategię, w której, żeby się ukryć przesuwali wspólnie elementy otoczenia w postaci prostopadłościanów w celu zastawienia drzwi do pokoju, w którym się ukrywali. Dzięki temu agenci, którzy szukali nie mogli wejść do pomieszczenia. W odpowiedzi na takie zachowanie agenci, którzy szukali nauczyli się przystawiać rampę do ściany, po której mogli wejść do zamkniętego pokoju (Rys. 14). Na rysunku widać, jak agenci, którzy się ukrywają (niebiescy) zastawili wejścia do pokoju, w którym się chowają. Agenci, którzy szukają (czerwoni) przestawiają rampę pod ścianę, żeby wejść do zamkniętego pomieszczenia. W kolejnych etapach nauki agenci, którzy się ukrywali nauczyli się chować te rampy, to znaczy przesuwali je z obszaru środowiska do pokoju, w którym zastawiali drzwi i się chowali. Dzięki temu, agenci szukający nie mogli wejść po ramach.

Możliwości algorytmów uczenia ze wzmocnieniem były również testowane w grach komputerowych takich jak Atari [96], multiplayer typu Dota 2 (algorytm *OpenAI Five*) [97] lub Starcraft (algorytm *AlphaStar*) [98]. W przypadku dwóch ostatnich gier, zaprojektowane

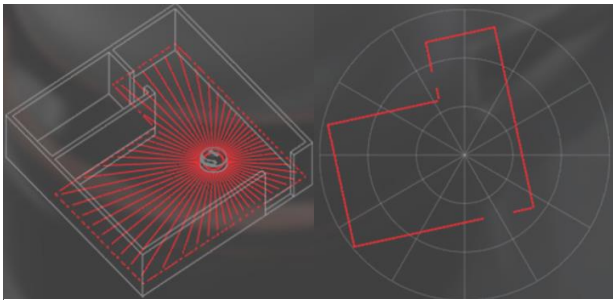
algorytmy były w stanie pokonać światowych mistrzów. Żeby utrzymać rywalizację pomiędzy komputerem, a człowiekiem na sprawiedliwym poziomie, algorytmy miały ograniczoną liczbę kliknięć myszy na sekundę, do takiej jaką mieli najlepsi gracze. Algorytmy uczenia ze wzmocnieniem były również stosowane z powodzeniem w różnych grach logicznych, takich jak szachy, Shogi [99] czy Go [100] i również ogrywały mistrzów świata.

W trakcie stosowania algorytmów uczenia ze wzmocnieniem w robotyce, bezpieczna eksploracja środowiska jest bardzo ważną kwestią, co zostało podkreślone w [101]. Perkins i Barto [102] zaproponowali sposób projektowania środowisk do trenowania agentów, który zapewnia bezpieczeństwo. Autorzy stwierdzili, że opisane metody mają wiele korzyści i nie spowalniają procesu uczenia. W pracy [103] rozważono problemy związane z formułowaniem modelu środowiska i agenta w sterowaniu robotami. Przedstawiono różne możliwe sposoby uczenia się modeli w robotyce. Rozpoznano problemy, które implikują te podejścia. Zaproponowano również przyszłe kierunki rozwoju algorytmów uczenia się robotów w czasie rzeczywistym.

2.4 Omijanie przeszkód na zadanej trajektorii ruchu

W celu zapewnienia bezpiecznej pracy robota i człowieka w tym samym obszarze pracy, robot musi często omijać bezkolizyjnie przeszkody, w tym także człowieka oraz wykonywać określone zadania, takie jak na przykład poruszanie się po zadanej trajektorii ruchu.

Zadania nawigacyjne planowania trasy i omijania przeszkód, powinny współpracować ze sobą w celu jak najszybszego osiągnięcia pozycji zadanej i uniknięcia kolizji z przeszkodami podczas ruchu. W rzeczywistym środowisku, wewnątrz budynków znajdują się statyczne przeszkody, takie jak stoły, krzesła oraz dynamiczne przeszkody, takie jak na przykład ludzie. W niektórych przypadkach, prędkość ruchu lub kształty tych obiektów mogą zmieniać się w nieprzewidywalny sposób. Planowanie trajektorii ruchu, manipulowanie obiektami i omijanie przeszkód, to jedno z najważniejszych wyzwań stojących przed każdym wdrożeniem robotów w rzeczywistych środowiskach.



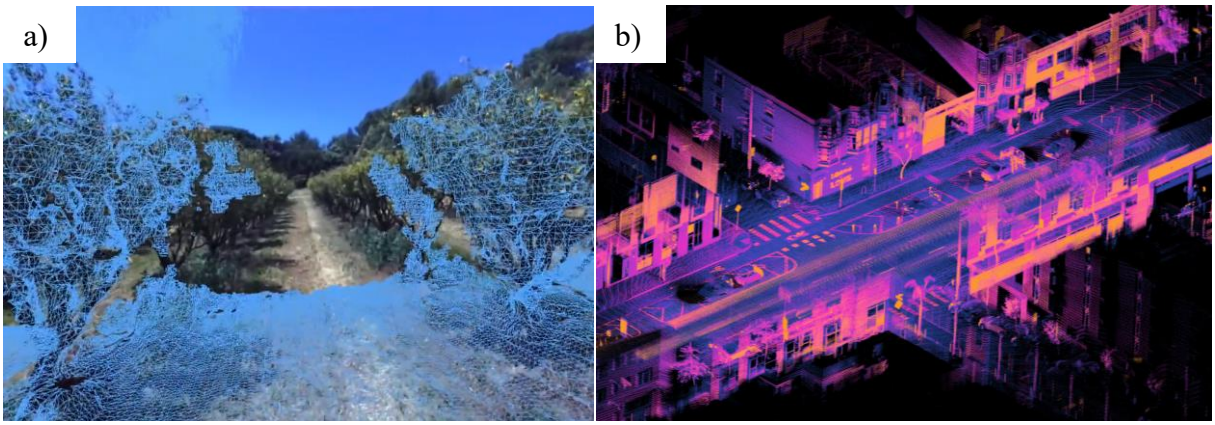
Rys. 15. Wizualizacja danych z czujnika RPLIDAR A2 [slamtec.com]

Rozpoznanie przeszkody wymaga zastosowania odpowiednich czujników w ramieniu robota. Obecnie do śledzenia otoczenia robota i wykrywania przeszkód często stosuje się systemy wizyjne. Stosuje się też czujniki zbliżeniowe, ultradźwiękowe albo optyczne, które potrafią wykryć przeszkody w bliskim położeniu robota.

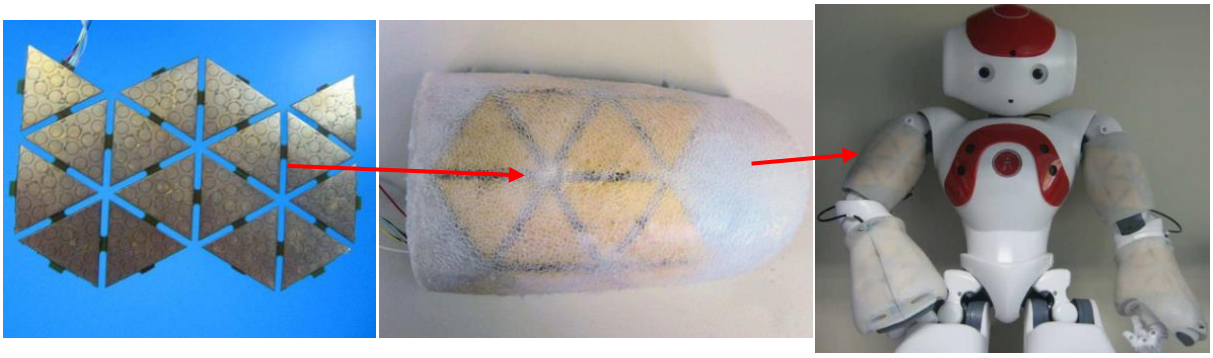
Już w 1985r. Moravec i Elfes [104] zaproponowali użycie wielu szerokokątnych czujników do mapowania otoczenia robota mobilnego. Tworzona mapa była sukcesywnie aktualizowana przez akumulację pomiarów z czujników. Dzięki temu robot był w stanie wykrywać kształt elementów w otaczającym go środowisku, w tym przeszkód. W robotach mobilnych do wykrywania przeszkód w płaszczyźnie dwuwymiarowej stosuje się obecnie czujniki LIDAR. Są one relatywnie tanie i pozwalają na wykrywanie przeszkód w zakresie 360°. Można je znaleźć w małych robotach

przemieszczających się w pomieszczeniach, na przykład w odkurzaczach domowych. Przykładowa wizualizacja danych i kształt pomieszczenia zarejestrowany przez RPLIDAR A2 został przedstawiony na Rys. 15.

Ostatnio na rynku pojawiły się zaawansowane systemy obrazowania, takie jak kamery RGBD i trójwymiarowe skanery laserowe (LIDAR 3D). Wymagają one zastosowania skomplikowanych technik przetwarzania sygnałów oraz wydajnych komputerów, ale mogą dostarczyć wielu dokładnych informacji o otaczającym robota środowisku. Kamery RGBD i skanery laserowe mogą reprezentować otaczające środowisko w postaci chmury punktów (*point cloud*), dostarczając dodatkowo informacji o tak zwanej głębi. Innym przykładem przedstawienia otoczenia w trójwymiarowej formie jest mapowanie przestrzenne (*spatial mapping*), zaprezentowane na Rys. 16a. W taką funkcjonalność wyposażone są na przykład kamery RGBD ZED i ZED2. Skanery laserowe są powszechnie wykorzystywane w samochodach autonomicznych, między innymi z uwagi na ich duży zasięg (250m), rozdzielczość i możliwość próbkowania danych w czasie rzeczywistym. W zależności od producenta skanery te mogą skanować otoczenie w zakresie horyzontalnym od 90° do 360° i wertykalnym od 20° do 90°. Przykład chmury punktów wykonanej przez LIDAR 3D Ouster OS1, który był zamontowany na samochodzie autonomicznym został pokazany na Rys. 16b.



Rys. 16. a) – Przykład mapowania przestrzennego wykonanego kamerą RGBD ZED2 [stereolabs.com]. b) – Mapa ulicy przedstawiona jako chmura punktów wykonana czujnikiem LIDAR 3D Ouster OS1 [ouster.com]

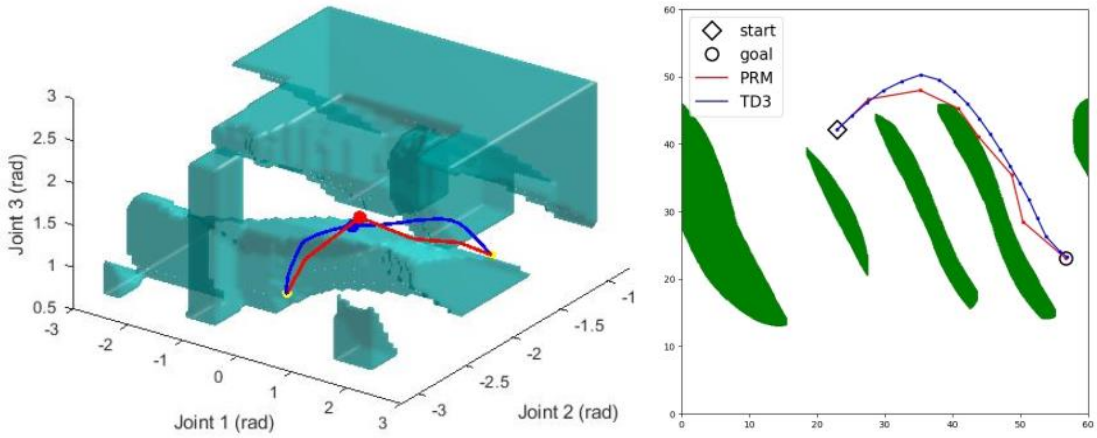


Rys. 17. Sztuczna skóra oparta o czujniki pojemnościowe zamocowana do ramion robota humanoidalnego [106]

Kolejnym rozwiązaniem przeznaczonym do wykrywania przeszkód, może być zastosowanie sztucznej skóry, która potrafi wykryć obiekty za pomocą dotyku. Lumelsky w pracy [105] jako pierwszy, zastosował wrażliwą skórę pokrywającą ramię manipulatora w celu wykrycia kontaktu, czyli dotknięcia przedmiotu. Idea sztucznej skóry opartej o czujniki pojemnościowe została przedstawiona w publikacji [106]. Autorzy opracowali własną koncepcję zespołu elastycznych czujników, którymi pokryli ramiona robota humanoidalnego przedstawionego na Rys. 17. Takie czujniki są w stanie wykrywać obiekty w odległości do kilku centymetrów.

Do wykrywania przeszkód stosuje się również czujniki ultradźwiękowe, które są w stanie wykrywać także obiekty transparentne, na przykład szyby. Są one dobrym uzupełnieniem czujników optycznych, takich jak LIDAR, które nie są w stanie poprawnie wykrywać przezroczystych przeszkód. Koncepcję układu czujników składającego się z czujników ultradźwiękowych opisano w pracy [107].

Petric w pracy [108] przedstawił szeroki przegląd różnych strategii omijania przeszkód przez roboty przemysłowe. Stwierdzono w niej, że podstawową strategią jest rozważenie problemu na poziomie kinematycznym, która polega na zidentyfikowaniu punktu na ramieniu robota, który znajduje się w pobliżu przeszkód i wyznaczeniu zmiany prędkości tego punktu tak, aby manipulator odsunął się od przeszkody. Zaproponowano dodatkowo zmianę prędkości w stosunku do odległości do przeszkody. W kolejnej części pracy, omówiono problemy związane z omijaniem przeszkód przez roboty z wieloma ramionami oraz roboty humanoidalne.



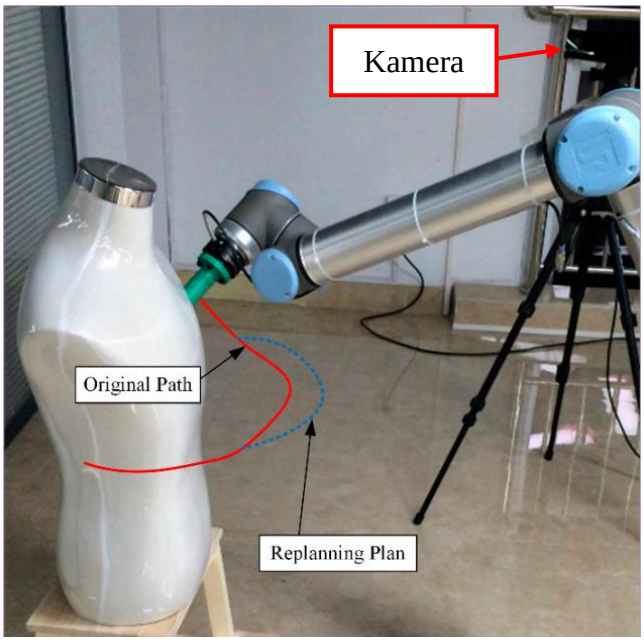
Rys. 18. Przykłady trajektorii ruchu po jakiej poruszał się manipulator sterowany algorytmem TD3 i PRM [109]

Z uwagi na rozwój sztucznej inteligencji, w ostatnich latach pojawiły się rozwiązania oparte o algorytmy uczenia ze wzmocnieniem, wykorzystywane do omijania przeszkód. W artykule [109] opisano zastosowanie takiego algorytmu do omijania przeszkód przez 3-osiowe ramię robota. Autorzy zastosowali algorytm uczenia ze wzmocnieniem *Twin Delayed Deep Deterministic Policy Gradient* (TD3) oraz porównali jego działanie z algorytmem *Probabilistic Roadmap* (PRM). Uzyskano 90% współczynnik sukcesu dla algorytmu uczenia ze wzmocnieniem i o 3% krótszą trajektorię ruchu w porównaniu do algorytmu PRM. Porównanie uzyskanych trajektorii przedstawiono na Rys. 18. Jednak algorytm został zaimplementowany tylko w symulacji. Ci sami Autorzy w pracy [110] przetestowali

zaproponowany system w warunkach laboratoryjnych na rzeczywistym robocie. Dodatkowo zaimplementowali algorytmy uczenia ze wzmocnieniem, takie jak *Soft Actor Critic* (SAC) oraz *Hindsight Experience Replay* (HER). Dzięki różnym kombinacjom algorytmów uzyskano do 10% krótszą trajektorię w porównaniu do algorytmu PRM.

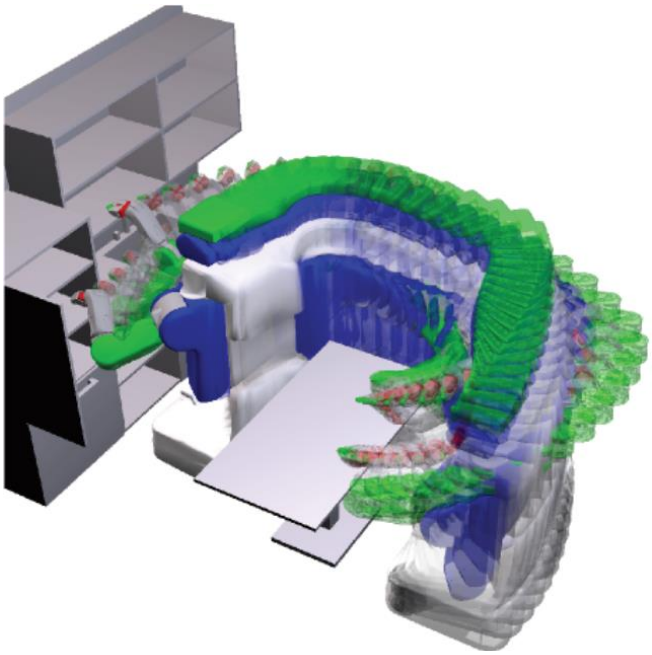
Wen i inni [111] zaproponowali wykorzystanie algorytmu *Deep Deterministic Policy Gradient* (DDPG) do planowania trajektorii ruchu modelu symulacyjnego ramienia robota Nao. Zastosowali funkcję nagrody, zależną od pozycji zadanej i od aktualnej pozycji manipulatora. Tak wytrenowany agent sterujący był w stanie pozycjonować TCP robota w zadanym punkcie, ale nie był w stanie omijać bezkolizyjnie przeszkód. W celu usprawnienia procedury uczenia, Autorzy sformułowali funkcję nagrody zależną również od odległość manipulatora do przeszkody. Wyniki badań symulacyjnych potwierdziły, że zmodyfikowany algorytm pozwalał robotowi z powodzeniem omijać przeszkody. Podobne podejście zastosowali Autorzy artykułu [112], w którym również zastosowano algorytm uczenia ze wzmocnieniem o nazwie *Q-learning* do planowania ścieżki robota i omijania przeszkód. Badania zostały wykonane tylko w domenie symulacyjnej.

Algorytmy uczenia ze wzmocnieniem są również stosowane do planowania trajektorii ruchu robotów mobilnych w celu omijania przeszkód. W pracy [113] wykorzystano i porównano algorytmy DDPG oraz *Proximal Policy Optimization* (PPO) do sterowania mobilnym robotem TurtleBot Burger. Zaproponowano planer oraz technikę kształtowania funkcji nagrody (*reward shaping*), które zostały przetestowane w warunkach symulacyjnych. Z zaprezentowanych wyników badań można wywnioskować, że algorytm PPO z techniką kształtowania nagrody osiągnął najlepszą wydajność.



Rys. 19. Przykład trajektorii ruchu, po której poruszał się manipulator [114]

Do planowania trajektorii ruchu bardzo często stosowanym algorytmem w różnych aplikacjach, w dziedzinie robotyki jest *Rapidly Exploring Random Tree* (RRT). W pracy [114] zaproponowano zmodyfikowaną wersję tego algorytmu o nazwie *Smoothly-RRT* (S-RRT), do planowania trajektorii ruchu z funkcją autonomicznego omijania przeszkód, dla manipulatora przedstawionego na Rys. 19. Do wykrywania przeszkód zastosowano kamerę RGBD Kinect V2 zamontowaną na statywie. Zaproponowany system był w stanie omijać przeszkody zarówno statyczne, jak i pojawiające się dynamicznie w strefie roboczej robota. Algorytm RRT wykonał poprawnie manewr omijania przeszkody 12 razy na 20 prób, a algorytm S-RTT 20 razy na 20 prób.



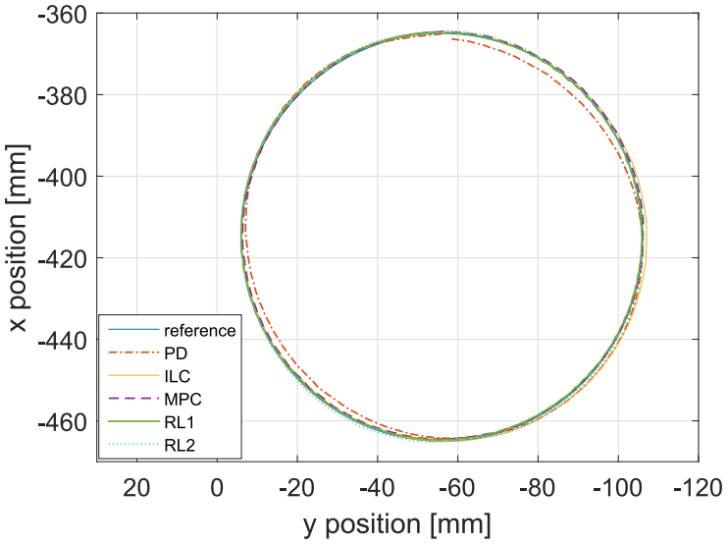
Rys. 20. Zaplanowana trajektoria ruchu przez algorytm TrajOpt dla robota PR2 [116]

Nowatorskimi metodami do planowania trajektorii ruchu, w tym omijania przeszkód, są algorytmy takie jak *Covariant Hamiltonian Optimization for Motion Planning* (CHOMP) [115] oraz *TrajOpt* [116]. Algorytmy te są przeznaczone do sterowania robotami z dużą liczbą stopni swobody. *TrajOpt* z powodzeniem był wykorzystywany do sterowania w warunkach symulacyjnych takimi robotami jak PR2 (18 stopni swobody), czy Atlas (34 stopnie swobody). Wyniki badań pokazały, że algorytm *TrajOpt* cechował się największym współczynnikiem sukcesu oraz niskim czasem przetwarzania w planowaniu trajektorii ruchu, w porównaniu do takich algorytmów jak RRT i CHOMP. W zależności od liczby zaangażowanych stopni swobody robota PR2 dla zadania, jakim było odłożenie przedmiotu na półkę (Rys. 20), współczynnik sukcesu dla *TrajOpt* wahał się w przedziale od 88% do 99%.

Ciekawym rozwiązaniem zaprezentowanym w pracy [117] było zastosowanie interakcji dotykowej w ramieniu robota do omijania przeszkód. Optyczne czujniki momentu obrotowego zostały rozmieszczone na każdym przegubie robota. Dzięki nim, system sterowania był w stanie płynnie sterować robotem tak, aby podążał on za obrysem obiektu, aż do dotarcia do pozycji zadanej.

Większość przedstawionych w literaturze metod do omijania przeszkód wymaga posiadania modelu robota i jego otoczenia, a także modelu geometrycznego przeszkód i miejsc ich usytuowania. Model ten zwykle musi zostać opracowany przez programistę i zaimplementowany w algorytmie sterowania robota. Ponadto, stosowane obecnie metody programowania robotów, w których kod programu dokładnie opisuje trajektorię ruchu, mają istotne ograniczenia, dotyczące w szczególności braku odpowiedniego sterowania robotem w przypadku dynamicznych zmian otoczenia (na przykład zmiany położenia przeszkód), których programista nie przewidział. Roboty przyszłości powinny mieć zupełnie nowe możliwości programowania, umożliwiające na przykład autonomiczną pracę. Dlatego muszą być w stanie samodzielnie nauczyć się dostosowywać do nowych lub tylko częściowo znanych środowisk. Tego typu wymagania mogą być spełnione dzięki zastosowaniu metod sztucznej inteligencji.

Istotną funkcjonalnością jaką musi mieć system sterujący pracą robota przeznaczonego do współpracy z człowiekiem oprócz omijania przeszkód, jest poruszanie się po zadanej trajektorii ruchu. W literaturze można znaleźć przykłady wykorzystania algorytmów uczenia ze wzmocnieniem do podążania po zadanej trajektorii ruchu przez roboty.



Rys. 21. Przykład podążania po dwuwymiarowej trajektorii zadanej o kształcie okręgu przez różne algorytmy [118]

W artykule [118] przedstawiono system oparty o algorytm uczenia ze wzmocnieniem do kompensacji błędu miedzy trajektorią zadaną, a trajektorią ruchu robota UR5. Działanie tego rozwiązania zostało porównane z algorytmami bazującymi na sterowaniu predykcyjnym (*model predictive control*, MPC), sterowaniu z uczeniem iteracyjnym (*iterative learning control*, ILC) oraz regulatorem PD. Autorzy wykonali testy dla trajektorii jedno oraz dwuwymiarowych. Dla trajektorii o kształcie okręgu przedstawionej na Rys. 21, rozwiązanie oparte o algorytmy uczenia ze wzmocnieniem cechowało się najmniejszym błędem RMS wynoszącym ok. 0.5mm dla osi *X* i 0.25mm dla osi *Y*. Podobną metodykę badań zastosowano w publikacji [119]. Autorzy zaproponowali ulepszoną wersję algorytmu *Q-learning* dla dyskretnej przestrzeni akcji i obserwacji. Algorytm przetestowano dla dwuwymiarowej zadanej trajektorii oraz porównano jego działanie z algorytmem *NI-like grid mode* (NIGM).

Inny przykład zastosowania algorytmów uczenia ze wzmocnieniem do podążania po zadanej trajektorii opisano w pracy [120]. W pierwszym etapie zaproponowano algorytm, który miał osiągnąć punkt zadany w przestrzeni trójwymiarowej. W drugim etapie dodano zadaną trajektorię, po której robot miał się przemieszczać, żeby osiągnąć punkt zadany. Współczynnik sukcesu osiągnięcia tego punktu wynosił 94%. Algorytm został przetestowany w warunkach laboratoryjnych na 6-osiowym robocie Mitsubishi MELFA RV-FR.

2.5 Podsumowanie przeglądu literatury

W trakcie przeglądu literatury przeanalizowano ponad 120 prac anglojęzycznych, dotyczących współpracy robota przemysłowego z człowiekiem oraz zastosowania metod sztucznej inteligencji, w tym algorytmów uczenia ze wzmocnieniem w robotyce. Prawie 100 publikacji zostało opublikowanych w latach 2016-2021. Na podstawie wykonanego przeglądu literatury można stwierdzić, że nie napotkano na żadne rozwiązanie oparte o algorytmy uczenia ze wzmocnieniem, które pozwalałoby na współpracę robota przemysłowego z człowiekiem w tej samej, wspólnej strefie roboczej. Rozwiązania HRI prezentowane w publikacjach naukowych bazują na sekwencyjnym (przemiennym) dostępie człowieka i robota do strefy roboczej, realizowanym dzięki zastosowaniu układu do wykrywania obecności człowieka we wspólnej strefie.

Przegląd literatury pokazał także, że brak jest publikacji dotyczących omijania przeszkód na zadanej trajektorii ruchu przez ramiona robotów sterowane z zastosowaniem algorytmów uczenia ze wzmocnieniem. Z opublikowanych prac wynika, że algorytmy te były stosowane

tylko do omijania przeszkód lub tylko do podążania po zadanej trajektorii przez manipulator. W przypadku publikacji dotyczących podążania po zadanej trajektorii, badania były wykonywane tylko dla trajektorii jedno i dwuwymiarowych (wymiary geometryczne trajektorii nie zmieniały się w osi Z, płaszczyzna trajektorii równoległa do płaszczyzny XY). Brak jest także publikacji prezentujących możliwości algorytmów uczenia ze wzmocnieniem dla trajektorii trójwymiarowych (wymiary geometryczne trajektorii zmieniały się w osi Z). Z przeglądu literatury wynika też, że w przypadku prac badawczych z zakresu omijania przeszkód, większość z nich realizowana była tylko w środowisku symulacyjnym. Wyraźnie widać lukę aplikacyjną tego typu systemów na rzeczywistych obiektach.

W związku z tym, w niniejszej pracy podjęto badania nad opracowaniem systemu sterowania robotem, pozwalającego na jednoczesną pracę człowieka i robota we wspólnej strefie roboczej zakładając, że w przypadku wykrycia człowieka na zadanej trajektorii, robot będzie go omijał.

3 Problem badawczy

Wykorzystanie robotów do wykonywania różnych zadań na liniach montażowych musi być poprzedzone ich „ręcznym” zaprogramowaniem przez programistę, co jest zwykle dość trudne i wymaga wiedzy zarówno o programowaniu, jak i o całym procesie i środowisku produkcyjnym, w którym robot będzie zastosowany. Programista musi przewidzieć i zaprogramować dokładnie kolejne kroki (przemieszczenia), jakie będzie wykonywał robot. Jeśli w jego strefie roboczej, a dokładnie na jego ścieżce ruchu będą znajdowały się przeszkody, to programista musi go tak zaprogramować, aby bezkolizyjnie je omijał. Takie rozwiązanie ma wady, ponieważ zmiana położenia przeszkody lub pojawienie się nowej spowoduje, że napisany przez programistę program może okazać się bezużyteczny. Omijanie przeszkód przez ramię robota jest bardzo ważne w przypadku zastosowań HRI i HRC, w których robot i człowiek mają współpracować we wspólnej strefie roboczej. W przedstawionych w rozdziale 2 rozwiązaniach, robot i człowiek nie mają jednoczesnego dostępu do współdzielonej strefy roboczej, w której mają wykonać określone czynności. Jeśli człowiek wykonuje określone operacje we wspólnej strefie, to robot oczekuje na ich zakończenie i dopiero wtedy realizuje swoje zadania. Dlatego implementacja funkcjonalności, która pozwala na jednoczesną pracę robota i człowieka we wspólnej strefie roboczej, daje możliwości wyeliminowania sekwencyjnego do niej dostępu oraz skrócenia czasu wykonywania wspólnego zadania.

Zastosowanie algorytmów, które pozwalają na autonomiczną pracę i podejmowanie decyzji przez robota może zapewnić, że niezależnie od stanu środowiska, robot będzie w stanie samodzielnie omijać przeszkody, które znajdą się w jego polu roboczym. W niniejszej pracy podjęto badania nad opracowaniem autonomicznego systemu do sterowania 6-osiowym manipulatorem antropomorficznym. Założono, że w systemie tym zostaną wykorzystane gradientowe algorytmy uczenia ze wzmocnieniem (*Policy Gradient Reinforcement Learning algorithms*). Wykorzystanie takich algorytmów jest obecnie przedmiotem wielu badań, prowadzonych w różnych jednostkach naukowych na świecie, na przykład w firmach Google, OpenAI, czy na uniwersytecie technicznym w Zurychu. Ich wyniki pokazują, że zastosowania algorytmów uczenia ze wzmocnieniem otwierają możliwości autonomicznej pracy robotów. Dzięki ich zastosowaniu, program sterujący robotem będzie mógł dostosować się do zmian występujących w otaczającym go środowisku, a co za tym idzie będzie umożliwiał bezkolizyjną współpracę z człowiekiem. Założono, że do wykrywania przeszkód w polu roboczym robota, zostanie zaprojektowana i zbudowana dedykowana głowica z laserowymi czujnikami odległości, która będzie zamontowana na kiści manipulatora. Tak zaprojektowany system będzie testowany na stanowisku, w którym robot będzie pracował jednocześnie w tej samej strefie roboczej z człowiekiem.

Z przeprowadzonego przeglądu literatury wynika, że dotychczas nie opracowano systemu, który wykorzystuje gradientowe algorytmy uczenia ze wzmocnieniem, do sterowania robotem współpracującym z człowiekiem w jednej strefie roboczej. W związku z tym uznano za celowe podjęcie badań nad opracowaniem takiego właśnie systemu.

4 Cele i teza pracy

Na podstawie przeglądu literatury zaprezentowanego w rozdziale 2 oraz problemu badawczego przedstawionego w rozdziale 3 można podać, że głównym i zasadniczym celem pracy jest **opracowanie systemu opartego o gradientowe algorytmy uczenia ze wzmocnieniem, który pozwala na bezpieczną współpracę we wspólnej strefie roboczej człowieka i robota przemysłowego.** Celem pośrednim, który zamierza się osiągnąć, żeby zrealizować cel główny, jest opracowanie systemu, który pozwoli na omijanie przeszkód na zadanej trajektorii ruchu przez manipulator. Takie rozwiązanie pozwoli na bezpieczną współpracę człowieka i robota przemysłowego, we współdzielonej strefie roboczej oraz przyspieszy pracę z uwagi na możliwość jednoczesnego dostępu człowieka i robota do tej samej strefy.

Założono osiągnięcie następujących celów cząstkowych:

1. Opracowanie środowiska symulacyjnego oraz modelu kinematyki manipulatora antropomorficznego stosowanego w badaniach doświadczalnych.
2. Opracowanie modelu sterowania robotem w środowisku symulacyjnym z wykorzystaniem kinematyki odwrotnej uwzględniającego parametry mechaniczne i dynamiczne robota oraz opracowanie algorytmów sterujących rzeczywistym robotem.
3. Opracowanie systemu wykrywania przeszkód.
4. Dostosowanie i implementacja wybranych algorytmów uczenia ze wzmocnieniem i ich badania w pozycjonowaniu TCP rzeczywistego robota.
5. Opracowanie systemu sterującego pracą robota przemysłowego, pozwalającego na omijanie przeszkód znajdujących się w polu roboczym.
6. Adaptacja wytrenowanych w środowisku symulacyjnym algorytmów uczenia ze wzmocnieniem do omijania przeszkód na zadanej trajektorii ruchu.
7. Opracowanie systemu sterującego pracą robota przemysłowego pozwalającego na jednoczesną pracę człowieka i robota we wspólnej strefie roboczej.
8. Badania doświadczalne weryfikujące prawdziwość tezy pracy.

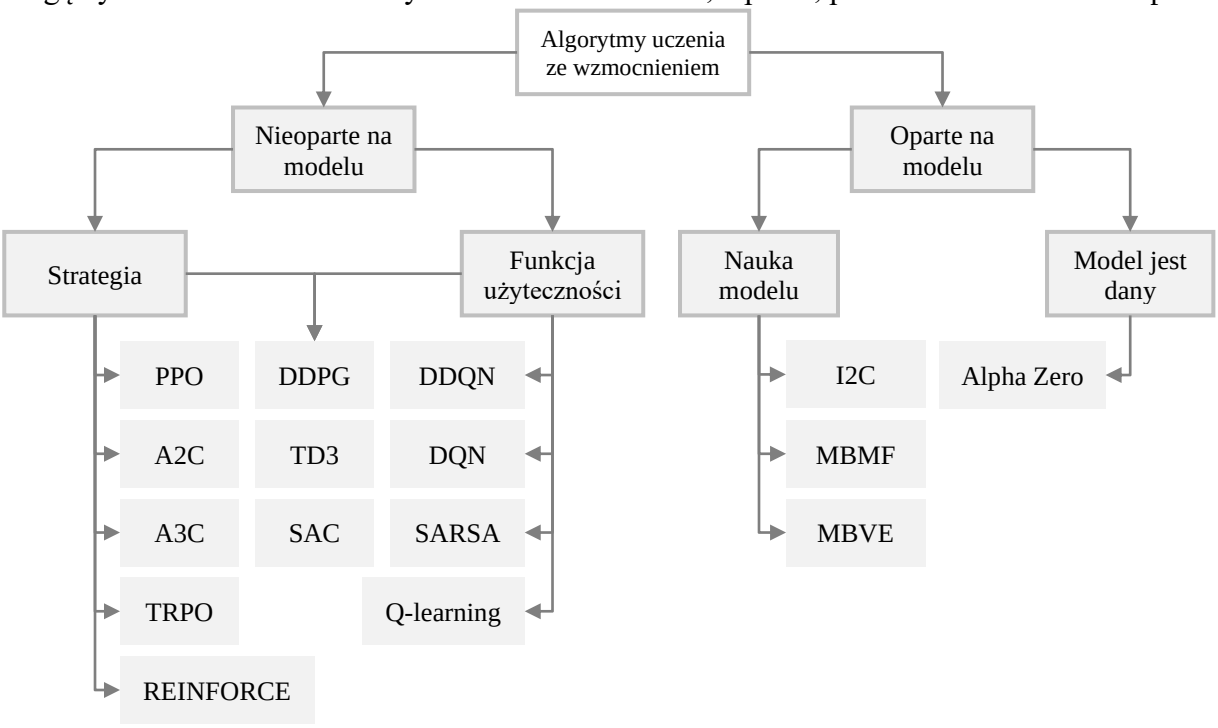
W związku z przedstawionymi celami pracy sformułowano następującą tezę pracy:

Gradientowe algorytmy uczenia ze wzmocnieniem mogą być zastosowane do opracowania bezpiecznego systemu sterowania robotem przemysłowym, współpracującym z człowiekiem.

5 Algorytmy uczenia ze wzmocnieniem

5.1 Wprowadzenie

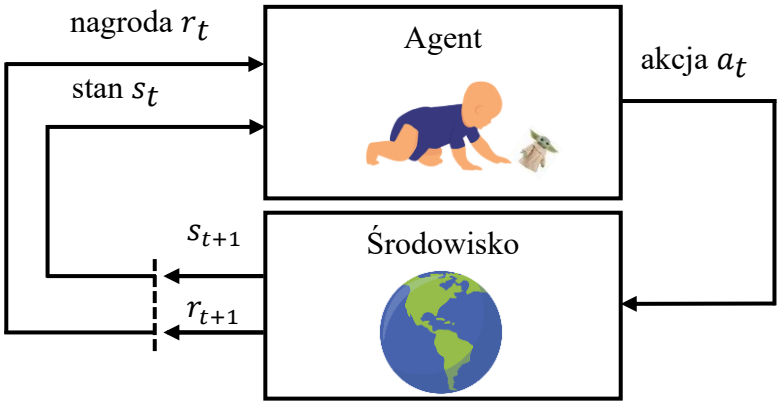
Uczenie ze wzmocnieniem jest metodą, którą można zastosować do trenowania różnych systemów, w zakresie optymalnego podejmowania decyzji [5]. Zadania związane ze sterowaniem robotem można traktować jako sekwencyjne procesy decyzyjne, a w algorytmach uczenia ze wzmocnieniem, dla każdej iteracji wymagana jest decyzja, którą trzeba podjąć. Tego typu algorytmy można podzielić na bazujące na modelu środowiska (*Model-Based RL*) oraz takie, które nie bazują na modelu środowiska (*Model-Free RL*). Podział algorytmów uczenia ze wzmocnieniem został przedstawiony na Rys. 22. Przykładem algorytmu, który bazuje na modelu środowiska jest algorytm do gry w szachy, Shogi i Go [99], opracowany przez firmę DeepMind. Algorytmy, które nie bazują na modelu środowiska mogą być zastosowane w robotyce do nauki chodzenia, łapania, przenoszenia obiektów itp.



Rys. 22. Podział algorytmów uczenia ze wzmocnieniem z przykładami [openai.com]

W algorytmach uczenia ze wzmocnieniem występuje tak zwany agent, który podejmuje akcje (działania) w środowisku, które go otacza. Na każdym etapie procesu sterowania agent wykonuje akcję $a \in A$ na przykład przesunięcie TCP robota do określonego punktu, które może być zmianą położenia w metrach w przestrzeni trójwymiarowej $(\Delta x, \Delta y, \Delta z)$. Po wykonaniu akcji agent otrzymuje obserwacje $o \in O$, która może być pełnym lub niepełnym opisem stanu $s \in S$, w którym się znajduje. Przykładem obserwacji może być aktualna pozycja TCP robota w metrach podana w odniesieniu do początku układu współrzędnych manipulatora, prędkość liniowa TCP robota w $\frac{m}{s}$, zadana pozycja TCP w metrach, pozycje i kształt przeszkód zarejestrowanych przez czujniki itd. W konwencji oznaczeń przyjmuje się, że niezależnie od tego czy obserwacje są pełnym lub niepełnym opisem stanu agenta, oznacza

się je jako s. Obserwacje i akcje mogą być zarówno w przestrzeni dyskretnej jak i ciągłej (*discrete / continuous space*). W efekcie podjętych działań, agent otrzymuje nagrodę $r \in R$ (liczbę), wskazującą jak „dobra" była wykonana akcja. Nagrodą może być różnica pomiędzy aktualną, a zadaną pozycją TCP robota podana w metrach, czyli błąd pozycjonowania. Im mniejszy błąd, tym większa nagroda. Celem agenta jest znalezienie optymalnej strategii (*policy*) generowania akcji i zmaksymalizowanie uzyskanej, skumulowanej przyszłej nagrody G, zwanej zwrotem (*return*), a tym samym zwiększenie przyszłej nagrody agenta. Agent aktualizuje (usprawnia) swoją politykę (strategię) poprzez interakcję ze środowiskiem metodą prób, za które otrzymuje nagrody – oceny. Strategia agenta to tak naprawdę mózg agenta. Można wyróżnić strategię stochastyczną $\pi(a|s)$ lub deterministyczną $\mu(s)$. Wykonywane czynności, a także uzyskane obserwacje i nagrody określają doświadczenie agenta. Zbiór sekwencji stan-akcja $(s_0, a_0, s_1, a_1, \dots)$ nazywany jest trajektorią τ. Wykonywanie kolejnych akcji umożliwia agentowi samodzielne odkrywanie (uczenie się) coraz lepszych działań, aby osiągnąć pożądany cel. Uproszczoną pętlę interakcji agent-środowisko przedstawiono na Rys. 23, na którym jako agent występuje dziecko, które uczy się chwytania zabawki, stosując metodę prób i błędów oraz autoocenę każdej podjętej akcji, wykorzystywaną do jej poprawienia. Próby definiowane są przypadkowo, na przykład w lewo z prawdopodobieństwem 0.2, do przodu 0.6 i w prawo 0.2, bądź stosując jakąś zdeterminowaną zasadę, na przykład zawsze do przodu. Po wykonaniu akcji (ruch ręką) dziecko ocenia skutek i poprawia swoją strategię, a następnie wykonuje ponowną próbę. Chwytanie zabawki jest stale modyfikowane i w końcu dziecko potrafi wykonywać to zadanie bezbłędnie.



Rys. 23. Pętla interakcji agent-środowisko dla algorytmów uczenia ze wzmocnieniem

Algorytmy uczenia ze wzmocnieniem wykorzystują dwie funkcje: użyteczności stanu V (*state value function*) oraz użyteczności stan-akcja Q (*state-action value function*). Wartości tych funkcji odzwierciedlają oczekiwany przyszły zwrot G_t, czyli skumulowaną przyszłą nagrodę, zaczynając w stanie s_t lub w parze stan-akcja s_t, a_t i postępując zgodnie z określoną strategią. Wartość zwrotu G_t można wyznaczyć ze wzoru:

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{1}$$

gdzie γ to współczynnik dyskontowy, wskazujący jaki wpływ na wartość zwrotu G_t mają przyszłe nagrody. Wartość tego współczynnika mieści się w zakresach od 0 do 1 i jest

zazwyczaj zbliżona do 1. W przypadku sterowania robotem oczekiwany zwrot i wartość funkcji nagrody może określać odległość w metrach TCP robota od punktu zadanego. Im mniejsza odległość (błąd), tym większa nagroda i oczekiwany przyszły zwrot. Funkcja użyteczności V dana jest wzorem:

$$V^{\pi}(s) = \mathbb{E}_{\pi}[G_t | s_t = s] \tag{2}$$

Daje ona oczekiwany zwrot G_t w chwili t, zaczynając w stanie s_t równym s i zawsze postępując zgodnie ze strategią π. Funkcja użyteczności stan-akcja Q dana jest wzorem:

$$Q^{\pi}(s, a) = \mathbb{E}_{\pi}[G_t | s_t = s, a_t = a] \tag{3}$$

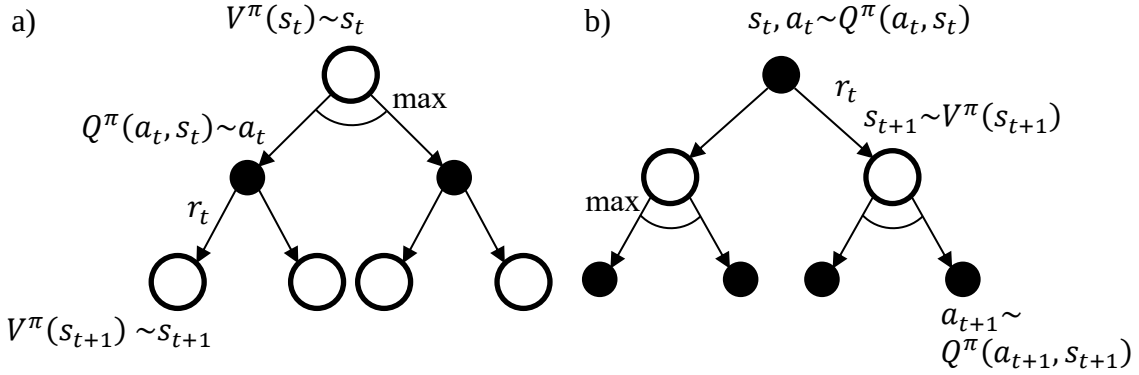
Daje ona oczekiwany zwrot G_t, zaczynając w stanie s_t równym s, podejmując akcję a_t równą a i następnie zawsze postępując zgodnie ze strategią π. Dla jednej i drugiej funkcji można sformułować optymalną funkcję użyteczności stanu V^* oraz optymalną funkcję użyteczności stan-akcja Q^*. Są one dane wzorami:

$$V^*(s) = \max_{\pi}\{V^{\pi}(s)\} \qquad \forall s \epsilon S \tag{4}$$

$$Q^*(s, a) = \max_{\pi}\{Q^{\pi}(s, a)\} \qquad \forall s \epsilon S, a \epsilon A \tag{5}$$

Funkcje te pozwalają wyznaczyć maksymalny oczekiwany zwrot G_t postępując zgodnie z optymalną strategią π.

Funkcje V i Q są ze sobą ściśle powiązane, co jest reprezentowane przez równanie Bellmana (*Bellman Equation*) [5]. Wizualizacja w formie diagramu równań dla funkcji V i Q została przedstawiona na Rys. 24. Każdy otwarty okrąg reprezentuje stan, a każdy pełny reprezentuje parę stan-akcja. Na rysunku 24a przedstawiony jest diagram reprezentujący podejmowanie akcji na podstawie funkcji V. Aby znaleźć akcję, która maksymalizuje funkcję V trzeba wykonywać tak zwane wyszukiwanie z wyprzedzeniem. Algorytm korzystający z tej funkcji, żeby podjąć najlepszą akcję, musi najpierw wykonać każdą dostępną akcję, przejść do stanu s_{t+1} i dopiero może ocenić jaka jest wartość funkcji V, dla każdej z dostępnych akcji. Ostatecznie algorytm wybiera akcję, która ma największą wartość funkcji V w stanie s_{t+1} i prowadzi do największej skumulowanej nagrody G. Dla funkcji Q (Rys. 24b) nie ma



Rys. 24. Reprezentacja równania Bellmana dla funkcji: a) – V, b) – Q

potrzeby wykonywania wyszukiwania z wyprzedzeniem, ponieważ ta funkcja już zawiera informację o tym, która akcja prowadzi do stanu z największą wartością funkcji i skumulowanej nagrody G.

5.2 Zastosowane algorytmy

W niniejszej pracy zastosowano gradientowe algorytmy uczenia ze wzmocnieniem (*Policy Gradient Reinforcement Learning*), z uwagi na ich możliwość zastosowania dla ciągłej przestrzeni akcji i obserwacji. Bazują one na bezpośredniej nauce strategii π, która jest funkcją zależną od parametrów θ^π. Funkcja strategii π może być aproksymowana przez sieci neuronowe, wtedy parametry θ^π są wagami sieci. Algorytmy gradientowe wykorzystują funkcję celu $J(\theta^\pi)$, dla której poszukiwane są takie wartości θ^π, które maksymalizują jej wartość. Do optymalizacji parametrów θ^π można wykorzystać np. metodę *gradient ascent* (szukanie lokalnego maksimum funkcji) daną wzorem:

$$\theta^\pi \leftarrow \theta^\pi + \alpha_\pi \nabla_{\theta^\pi} J(\theta^\pi) \tag{6}$$

gdzie α_π jest współczynnikiem uczenia, a gradient funkcji $J(\theta^\pi)$ obliczany jest według wzoru:

$$\nabla_{\theta^\pi} J(\theta^\pi) = \mathbb{E}_\pi[\nabla_{\theta^\pi} \log \pi(a|s, \theta^\pi) Q^\pi(s, a)] \tag{7}$$

Wzór (7) nazywany jest *Policy Gradient Theorem* i stanowi podstawę teoretyczną dla różnych gradientowych algorytmów uczenia ze wzmocnieniem.

W badaniach zastosowano najnowsze, najbardziej znane oraz najefektywniejsze algorytmy dla ciągłej przestrzeni akcji i obserwacji (*continuous action, observation space)*, które zostały opracowane na przestrzeni kilku ostatnich lat. Algorytmami, które były stosowane w trakcie badań są:

- *Deep Deterministic Policy Gradient* (DDPG) [51],
- *Twin Delayed Deep Deterministic Policy Gradient* (TD3) [121],
- *Soft Actor Critic* (SAC) [122], [123],
- *Hindsight Experience Replay* (HER) [124].

Algorytmy te zostały zastosowane w sześciu wariantach: DDPG, TD3, SAC, DDPG+HER, TD3+HER, SAC+HER. We wczesnej fazie badań testowane były również takie algorytmy jak *Proximal Policy Optimization* (PPO) [125] i *Trust Region Policy Optimization* (TRPO) [126], ale nie dały one zadowalających rezultatów. Do własnej implementacji algorytmów wykorzystano język programowania Python oraz biblitoteki *Tensorflow* [8] i *Stable Baselines* [127]. Poniższe podrozdziały opisują algorytmy stosowane w trakcie badań. Równania i oznaczenia zostały zunifikowane, z tego względu mogą się one różnić od formy, w jakiej zostały zaprezentowane w oryginalnych artykułach. Hiperparametry poszczególnych algorytmów zostały przedstawione w załączniku w formie tabel.

5.2.1 Deep Deterministic Policy Gradient

Deep Deterministic Policy Gradient (DDPG) to algorytm niebazujący na modelu środowiska, który uczy się funkcji Q oraz deterministycznej polityki μ. Agent nie optymalizuje na bieżąco swojej polityki i funkcji Q w trakcie interakcji ze środowiskiem,

tylko zapisuje zbierane doświadczenie do bufora D. W literaturze tego typu algorytm klasyfikowany jest jako *off-policy*. DDPG wykorzystuje równanie Bellmana do nauki funkcji Q (zamodelowanej przez sieć neuronową) i następnie stosuje funkcję Q do nauki strategii μ (również zamodelowanej przez sieć neuronową). Algorytm DDPG łącznie wykorzystuje cztery sieci neuronowe: sieć $Q(s,a|\theta^Q)$, sieć $\mu(s|\theta^\mu)$, docelową sieć (*target network*) $Q'(s,a|\theta^{Q'})$ oraz docelową sieć dla polityki $\mu'(s|\theta^{\mu'})$. Sieci neuronowe są scharakteryzowane parametrami θ^x z indeksem górnym wskazującym, której sieci dotyczą dane parametry, na przykład parametry θ^Q dotyczą funkcji Q. Symbol $Q(s,a|\theta^Q)$ oznacza sieć neuronową aproksymującą funkcję Q sparametryzowaną przez parametry θ^Q, której wejściem jest stan s i akcja a.

W trakcie poszczególnych epizodów nauki, DDPG wykorzystuje bufor (*replay buffer*) D o określonym rozmiarze do zapisywania doświadczenia w formie krotek $(s_t, a_t, r_t, s_{t+1}, d_t)$, gdzie s_t to stan w kroku czasowym t, a_t to akcja w kroku czasowym t, r_t to nagroda w kroku czasowym t, s_{t+1} to stan w kroku czasowymi $t+1$, d_t to sygnał wskazujący, czy stan s_{t+1} jest stanem kończącym epizod (*terminal state*). W trakcie nauki i optymalizacji parametrów sieci neuronowych, dane są pobierane w formie losowych porcji (*minibatch*) B jako krotki $(s_i, a_i, r_i, s_{i+1}, d_i)$. Indeks dolny i oznacza, że dane zostały pobrane z bufora jako przeszłe doświadczenie. Schemat blokowy algorytmu zawierający budowę modelu sieci neuronowych, przepływ danych oraz aktualizacje parametrów został przedstawiony na Rys. 25. Agenta można podzielić na dwa moduły w postaci aktora (*actor*) definiowanego funkcją Q, oraz krytyka (*critic*) definiowanego strategią μ. Te moduły zawierają po dwie sieci neuronowe, główną i docelową, oznaczone kolorem niebieskim i zielonym. Sieci neuronowe trenowane są poprzez minimalizację funkcji kosztu L daną wzorem (9), w przypadku funkcji Q, oraz maksymalizację funkcji kosztu J daną wzorem (10), w przypadku strategii μ.

Cel (*target*) y_i dla sieci neuronowej $Q(s,a|\theta^Q)$, który jest oparty na równaniu Bellmana, obliczany jest według wzoru:

$$y_i(r_i, s_{i+1}, d_i) = r_i + \gamma(1-d_i)Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'}) \tag{8}$$

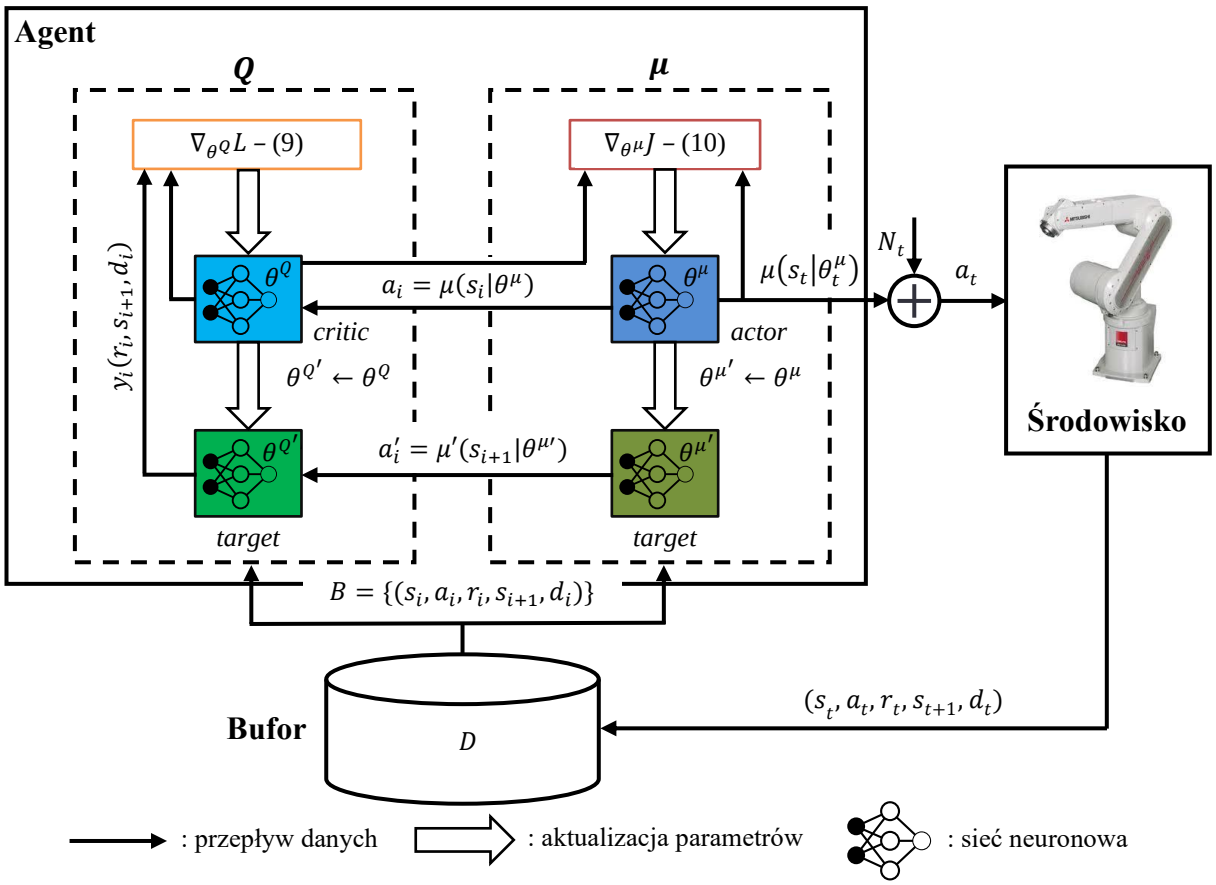
gdzie γ to współczynnik dyskontowy wskazujący jaki wpływ na wartość funkcji Q mają przyszłe nagrody. Wartość tego współczynnika mieści się w zakresach od 0 do 1. Do obliczenia celu brana jest pod uwagę docelowa sieć $Q'(s,a|\theta^{Q'})$.

Sieć neuronowa $Q(s,a|\theta^Q)$ jest trenowana z wykorzystaniem funkcji kosztu biorącej pod uwagę błąd średniokwadratowy Bellmana (*Mean Squared Bellman Error loss function*) [5] pomiędzy celem y_i, a wartością z głównej sieci aproksymującej funkcję Q według wzoru:

$$\nabla_{\theta^Q} L = \nabla_{\theta^Q} \frac{1}{|B|} \sum [y_i(r_i, s_{i+1}, d_i) - Q(s_i, a_i|\theta^Q)]^2 \tag{9}$$

Polityka μ musi zwrócić akcję a, która maksymalizuje funkcję Q. Jest ona optymalizowana według równania:

$$\nabla_{\theta^\mu} J = \nabla_{\theta^\mu} \frac{1}{|B|} \sum [Q(s_i, \mu(s_i|\theta^\mu)|\theta^Q)] \tag{10}$$



Rys. 25. Schemat blokowy algorytmu Deep Deterministic Policy Gradient

Wartości parametrów dla sieci głównych i docelowych są kopiowane co określoną liczbę kroków (*steps*) algorytmu z wykorzystaniem uśredniania Polyaka (*Polyak averaging*) według wzorów:

$$\theta^{Q'} \leftarrow \rho\theta^{Q'} + (1-\rho)\theta^{Q}$$
$$\theta^{\mu'} \leftarrow \rho\theta^{\mu'} + (1-\rho)\,\theta^{\mu} \tag{11}$$
$$\text{gdzie } \rho \in <0{,}1>$$

W celu zapewnienia odpowiedniej eksploracji agentowi z deterministyczną polityką, algorytm DDPG wykorzystuje szum, który jest dodawany do każdej akcji. Autorzy algorytmu w artykule [51] zaproponowali zastosowanie szumu Ornsteina-Uhlenbecka [128], który jest skorelowany czasowo. Akcja a_t obliczana jest według wzoru:

$$a_t = \mu(s_t|\theta_t^{\mu}) + N_t \tag{12}$$

gdzie N_t to wartość szumu w kroku czasowym t.

Przedstawiony w tej sekcji algorytm został pokazany w formie pseudokodu jako Algorytm 1. Kolorami zostały zaznaczone sieci neuronowe oraz funkcje kosztu. Algorytm trenowany jest iteracyjnie przez liczbę kroków T, określoną przez programistę. Z uwagi na to, że algorytm DDPG jest algorytmem typu *off-policy*, nie wykonuje on aktualizacji parametrów

sieci neuronowych po każdym kroku i wykonanej akcji w środowisku. Parametry sieci są aktualizowane co określoną liczbę kroków lub epizodów, a dane do aktualizacji parametrów pobierane są z bufora D.

Algorytm 1. Deep Deterministic Policy Gradient (DDPG)

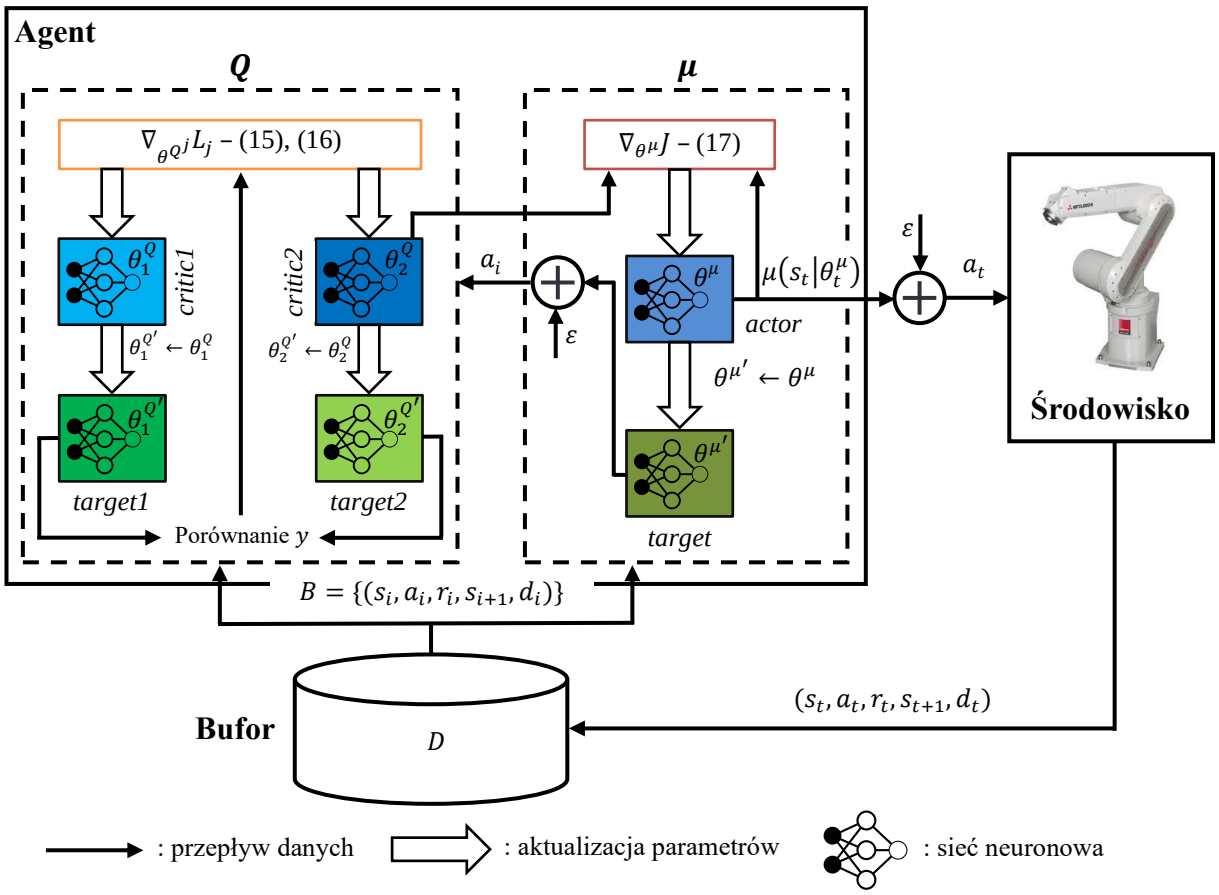
Inicjalizacja parametrów sieci neuronowych θ^{μ}, θ^{Q} oraz $\theta^{Q'} \leftarrow \theta^{Q}, \theta^{\mu'} \leftarrow \theta^{\mu}$
Inicjalizacja bufora D
for $t = t \ldots T$: **do**
  obserwuj stan s_t, wybierz akcję $a_t = \mu(s_t|\theta_t^{\mu}) + N_t$ (12)
  wykonaj akcję a_t, otrzymaj nagrodę r_t, obserwuj stan s_{t+1} oraz sygnał d_t
  przechowaj w buforze D krotkę $(s_t, a_t, r_t, s_{t+1}, d_t)$
  if aktualizuj parametry sieci **then**
    pobierz losową partię danych $B = \{(s_i, a_i, r_i, s_{i+1}, d_i)\}$ z bufora D
    oblicz cel y
      $y_i(r_i, s_{i+1}, d_i) = r_i + \gamma(1 - d_i)Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$ (8)
    aktualizuj parametry sieci neuronowej z parametrami θ^{Q} z wykorzystaniem np. *Gradient Descent*
      $\nabla_{\theta^Q} L = \nabla_{\theta^Q} \frac{1}{|B|} \sum [y_i(r_i, s_{i+1}, d_i) - Q(s_i, a_i|\theta^Q)]^2$ $\theta^Q \leftarrow \theta^Q - \alpha_Q \nabla_{\theta^Q} L$ (9)
    aktualizuj parametry sieci neuronowej z parametrami θ^{μ} z wykorzystaniem np. *Gradient Ascent*
      $\nabla_{\theta^\mu} J = \nabla_{\theta^\mu} \frac{1}{|B|} \sum [Q(s_i, \mu(s_i|\theta^{\mu})|\theta^Q)]$ $\theta^{\mu} \leftarrow \theta^{\mu} + \alpha_{\mu} \nabla_{\theta^\mu} J$ (10)
    aktualizuj parametry sieci docelowych (*target network*)
      $\theta^{Q'} \leftarrow \rho\theta^{Q'} + (1 - \rho)\theta^{Q}$
      $\theta^{\mu'} \leftarrow \rho\theta^{\mu'} + (1 - \rho)\theta^{\mu}$ (11)

5.2.2 Twin Delayed Deep Deterministic Policy Gradient

Twin Delayed Deep Deterministic Policy Gradient (TD3) jest algorytmem, który wprowadził szereg ulepszeń do algorytmu DDPG. Tak jak w przypadku poprzednika, TD3 jest również algorytmem typu *off-policy*. Bazuje on jednak na sześciu sieciach neuronowych: dwóch sieciach aproksymujących funkcję Q tj. $Q_1(s, a|\theta^{Q_1})$, $Q_2(s, a|\theta^{Q_2})$, sieci deterministycznej polityki $\mu(s|\theta^{\mu})$ oraz trzech sieciach docelowych $Q_1'(s, a|\theta^{Q_1'})$, $Q_2'(s, a|\theta^{Q_2'})$, $\mu'(s|\theta^{\mu'})$. Obydwie sieci Q są trenowane z wykorzystaniem minimalizacji błędu średniokwadratowego Bellmana, który jest różnicą kwadratu pomiędzy celem y i aproksymacją funkcji Q. Proces uczenia przebiega podobnie jak w przypadku DDPG i jednej sieci. TD3 również wykorzystuje bufor D, w którym przechowywane są krotki z doświadczeniem. Dane z bufora pobierane są w losowych porcjach B w momencie optymalizacji parametrów sieci neuronowych. Schemat blokowy algorytmu zawierający budowę modelu, przepływ danych oraz aktualizacje parametrów został przedstawiony na Rys. 26.

Pierwsze ulepszenie, które zostało wprowadzone przez Autorów TD3 polega na „wygładzeniu" wartości z docelowej sieci polityki μ'. Algorytm dodaje szum do akcji, aby utrudnić strategii μ wykorzystanie błędów aproksymacji funkcji Q. Akcje wykorzystywane do obliczenia celu y, bazują na wyjściu z sieci docelowej $\mu'(s|\theta^{\mu'})$ oraz ograniczonej wartości ϵ w zakresie $-c$ oraz $+c$, pobieranej z szumu N z zakresu 0 i σ (zazwyczaj nieskorelowany szum Gaussa ze średnią równą 0). Dodanie szumu zapewnia eksplorację środowiska. Same akcje do obliczenia celu y są ograniczone w zakresie $a_{Low} \leq a \leq a_{High}$ i obliczane ze wzoru:

$$a_{i+1}(s_{i+1}) = \text{clip}\left(\mu'\left(s_{i+1}\middle|\theta_t^{\mu'}\right) + \text{clip}(\epsilon, -c, c), a_{Low}, a_{High}\right) \qquad \epsilon \sim N(0, \sigma) \qquad (13)$$



Rys. 26. Schemat blokowy algorytmu Twin Delayed Deep Deterministic Policy Gradient

Drugie ulepszenie polegało na zastosowaniu dwóch sieci aproksymujących dwie konkurencyjne funkcje Q. Do obliczenia celu y, brana jest pod uwagę ta sieć aproksymująca funkcję Q, która zwróci mniejszą wartość. Cel y obliczany jest ze wzoru:

$$y_i(r_i, s_{i+1}, d_i) = r_i + \gamma(1 - d_i) \min_{j=1,2} Q'_j(s_{i+1}, a_{i+1}(s_{i+1})|\theta^{Q^{j\prime}}) \tag{14}$$

Jedna i druga sieć Q trenowana jest z zastosowaniem minimalizacji funkcji kosztu MSBE:

$$\nabla_{\theta^{Q^1}} L_1 = \nabla_{\theta^{Q^1}} \frac{1}{|B|} \sum [y_i(r_i, s_{i+1}, d_i) - Q_1(s_i, a_i|\theta^{Q^1})]^2 \tag{15}$$

$$\nabla_{\theta^{Q^2}} L_2 = \nabla_{\theta^{Q^2}} \frac{1}{|B|} \sum [y_i(r_i, s_{i+1}, d_i) - Q_2(s_i, a_i|\theta^{Q^2})]^2 \tag{16}$$

Strategia μ trenowana jest z wykorzystaniem maksymalizacji wartości pierwszej sieci neuronowej $Q_1(s, a|\theta^{Q^1})$ aproksymującej funkcję Q według wzoru:

$$\nabla_{\theta^\mu} J = \nabla_{\theta^\mu} \frac{1}{|B|} \sum [Q_1(s_i, \mu(s_i|\theta^\mu)|\theta^{Q^1})] \tag{17}$$

Ostatnim, trzecim ulepszeniem jest opóźniona w czasie aktualizacja parametrów sieci neuronowych. TD3 aktualizuje parametry sieci neuronowych strategii μ znacznie rzadziej niż sieci Q. Autorzy algorytmu, zalecają aktualizację sieci strategii μ raz na dwie aktualizacje sieci Q. Parametry aktualizowane są z wykorzystaniem uśredniania Polyaka:

$$\theta^{Q^{j\prime}} \leftarrow \rho\theta^{Q^{j\prime}} + (1-\rho)\theta^{Q^{j}} \quad \text{dla } j = 1{,}2$$
$$\theta^{\mu\prime} \leftarrow \rho\theta^{\mu\prime} + (1-\rho)\,\theta^{\mu} \tag{18}$$
$$\text{gdzie } \rho \in < 0{,}1 >$$

TD3 został podsumowany w formie pseudokodu jako Algorytm 2.

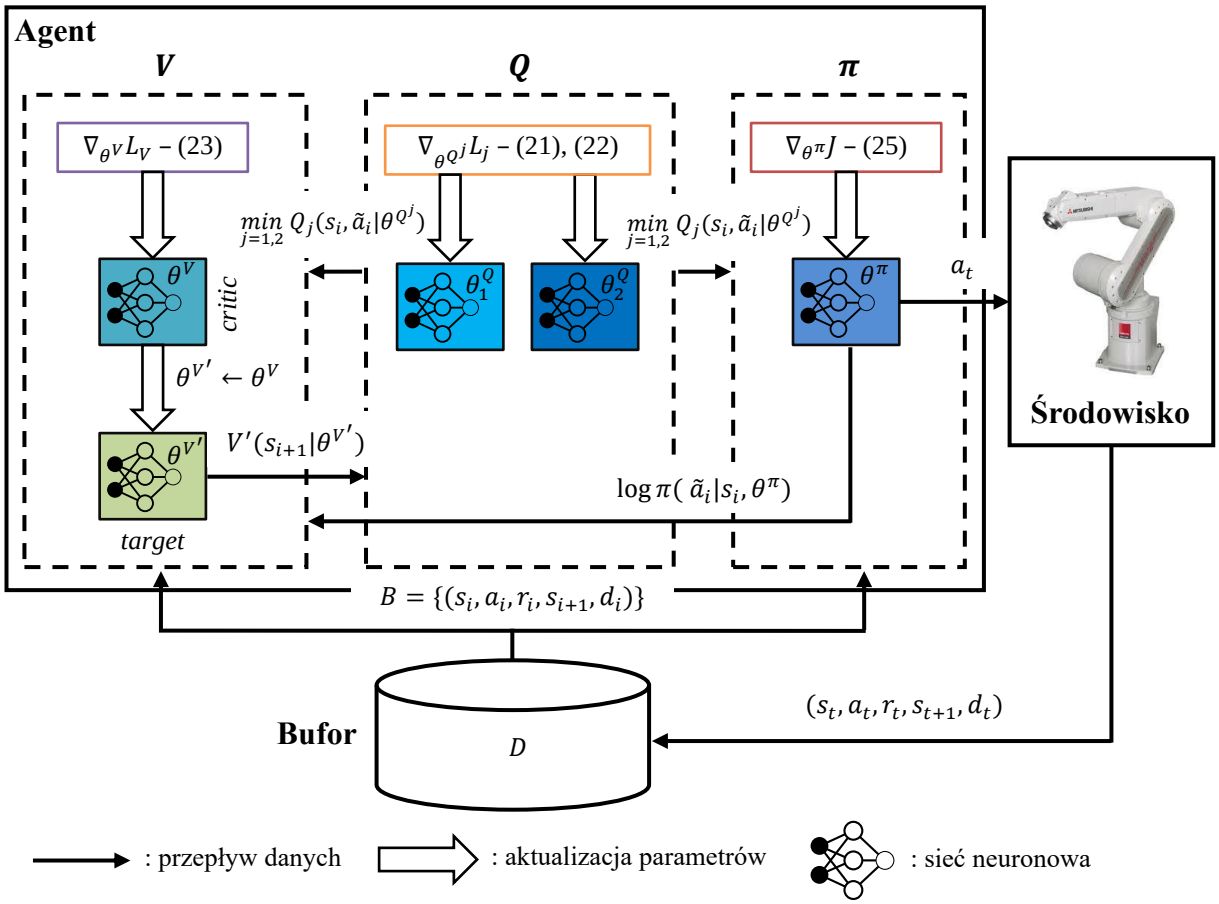
Algorytm 2. Twin Delayed Deep Deterministic Policy Gradient (TD3)

Inicjalizacja parametrów sieci neuronowych $\theta^{\mu}, \theta^{Q^1}, \theta^{Q^2}$ oraz $\theta^{Q^{1\prime}} \leftarrow \theta^{Q^1}$, $\theta^{Q^{2\prime}} \leftarrow \theta^{Q^2}\theta^{\mu\prime} \leftarrow \theta^{\mu}$
Inicjalizacja bufora D
for $t = t$ ... T: **do**
  obserwuj stan s_t, wybierz akcję $a_t(s_t) = \text{clip}\left(\mu\left(s_t|\theta_t^{\mu}\right) + N_t, a_{Low}, a_{High}\right)$
  wykonaj akcję a_t, otrzymaj nagrodę r_t, obserwuj stan s_{t+1} oraz sygnał d_t
  przechowaj w buforze D krotkę $(s_t, a_t, r_t, s_{t+1}, d_t)$
  if aktualizuj parametry sieci **then**
    pobierz losową partię danych $B = \{(s_i, a_i, r_i, s_{i+1}, d_i)\}$ z bufora D
    oblicz akcję dla celu y
      $a_{i+1}(s_{i+1}) = \text{clip}\left(\mu'\left(s_{i+1}\middle|\theta_t^{\mu\prime}\right) + \text{clip}(\epsilon, -c, c), a_{Low}, a_{High}\right)$ $\quad \epsilon \sim N(0,\sigma)$ (13)
    oblicz cel y
      $y_i(r_i, s_{i+1}, d_i) = r_i + \gamma(1-d_i)\min_{j=1,2} Q_j'(s_{i+1}, a_{i+1}(s_{i+1})|\theta^{Q^{j\prime}})$ (14)
    aktualizuj parametry sieci neuronowych z parametrami $\theta^{Q^1}, \theta^{Q^2}$ dla j=1, 2
      $\nabla_{\theta^{Q^j}} L_j = \nabla_{\theta^{Q^j}} \frac{1}{|B|}\sum[y_i(r_i, s_{i+1}, d_i) - Q_j(s_i, a_i|\theta^{Q^j})]^2$ $\quad \theta^{Q^j} \leftarrow \theta^{Q^j} - \alpha_Q \nabla_{\theta^Q} L_j$ (15), (16)
    if czas na aktualizację strategii **then**
      aktualizuj parametry sieci neuronowej z parametrami θ^{μ}
        $\nabla_{\theta^{\mu}} J = \nabla_{\theta^{\mu}} \frac{1}{|B|}\sum[Q_1\left(s_i, \mu(s_i|\theta^{\mu})\middle|\theta^{Q^1}\right)]$ $\quad \theta^{\mu} \leftarrow \theta^{\mu} + \alpha_{\mu}\nabla_{\theta^{\mu}} J$ (17)
      aktualizuj parametry sieci docelowych (*target network*)
        $\theta^{Q^{j\prime}} \leftarrow \rho\theta^{Q^{j\prime}} + (1-\rho)\theta^{Q^j}$ $\text{dla } j = 1{,}2$
        $\theta^{\mu\prime} \leftarrow \rho\theta^{\mu\prime} + (1-\rho)\theta^{\mu}$ (18)

5.2.3 Soft Actor-Critic

Soft Actor-Critic (SAC) jest algorytmem typu *off-policy* ze stochastyczną polityką π. Algorytm ten ma zmodyfikowaną funkcję celu w porównaniu do DDPG. Zamiast maksymalizować tylko nagrodę, SAC bierze pod uwagę również entropię polityki. Strategia jest trenowana w taki sposób, żeby zbalansować kompromis pomiędzy skumulowaną przyszłą nagrodą, a entropią. Zwiększenie entropii powoduje zwiększenie eksploracji środowiska, ale również zapobiega przedwczesnemu utknięciu polityki w złym lokalnym optimum. SAC wykorzystuje sieć neuronową do aproksymacji polityki $\pi(\cdot|s, \theta^{\pi})$, dwie sieci Q: $Q_1(s, a|\theta^{Q^1})$, $Q_2(s, a|\theta^{Q^2})$, jedną sieć $V(s|\theta^{V})$ oraz jedną sieć docelową $V'(s|\theta^{V'})$. SAC wykorzystuje bufor D, w którym przechowywane są krotki $(s_t, a_t, r_t, s_{t+1}, d_t)$ z doświadczeniem. Dane z bufora pobierane są w losowych porcjach B w momencie optymalizacji parametrów sieci neuronowych. Schemat blokowy algorytmu zawierający

budowę modelu, przepływ danych oraz aktualizacje parametrów został przedstawiony na Rys. 27.



Rys. 27. Schemat blokowy algorytmu Soft Actor Critic

Cele do trenowania sieci neuronowych związanych z funkcjami $Q - y^Q$ i $V - y^V$ obliczane są według wzorów:

$$y_i^Q(r_i, s_{i+1}, d_i) = r_i + \gamma(1 - d_i)V'(s_{i+1}|\theta^{V'}) \tag{19}$$

$$y_i^V(s_i) = \min_{j=1,2} Q_j(s_i, \tilde{a}_i|\theta^{Q_j}) - \alpha_T \log \pi(\tilde{a}_i|s_i, \theta^\pi) \qquad \tilde{a} \sim \pi(\cdot\,|s, \theta^\pi) \tag{20}$$

gdzie α_T oznacza współczynnik kompromisu pomiędzy eksploracją i eksploitacją, dla wyższego współczynnika jest większa eksploracja środowiska. Notacja $\tilde{a}$ oznacza, że akcja jest pobierana z polityki $\pi(\cdot\,|s, \theta^\pi)$, a nie tak jak w przypadku stanu s lub nagrody r z bufora D.

Sieci neuronowe Q są trenowane z wykorzystaniem błędu średniokwadratowego Bellmana, jedna i druga sieć używa tego samego celu y^Q. Do obliczenia funkcji kosztu stosuje się następujące formuły:

$$\nabla_{\theta^{Q^1}} L_1 = \nabla_{\theta^{Q^1}} \frac{1}{|B|} \sum [y_i^Q(r_i, s_{i+1}, d_i) - Q_1(s_i, a_i|\theta^{Q^1})]^2 \tag{21}$$

$$\nabla_{\theta^{Q^2}} L_2 = \nabla_{\theta^{Q^2}} \frac{1}{|B|} \sum [y_i^Q(r_i, s_{i+1}, d_i) - Q_2(s_i, a_i|\theta^{Q^2})]^2 \tag{22}$$

Funkcja użyteczności V jest trenowana z wykorzystaniem zależności między funkcjami Q i V. Do optymalizacji wartości funkcji V wykorzystywany jest błąd średniokwadratowy, który bierze pod uwagę sieć neuronową Q, która zwróciła mniejszą wartość. Funkcja kosztu dla funkcji V dana jest wzorem:

$$\nabla_{\theta^V} L_V = \nabla_{\theta^V} \frac{1}{|B|} \sum [y_i^V(s_i) - V(s_i|\theta^V)]^2 \tag{23}$$

Stochastyczna polityka π została zrealizowana poprzez wykorzystanie deterministycznej polityki μ, funkcji ograniczającej oraz szumu N (Autorzy algorytmu rekomendowali szum Gaussa). Akcje obliczane są z następującego wzoru:

$$\tilde{a}(s, \xi|\theta^\pi) = \tanh(\mu(s|\theta^\pi) + \xi) \quad \xi \sim N(0, I) \tag{24}$$

gdzie ξ jest wartością próbkowaną z szumu N. Polityka jest optymalizowana poprzez maksymalizację funkcji celu danej wzorem:

$$\nabla_{\theta^\pi} J = \nabla_{\theta^\pi} \frac{1}{|B|} \sum [\min_{j=1,2} Q_j(s_i, \tilde{a}_i(s_i, \xi_i|\theta^\pi)|\theta^{Q^j}) - \alpha_T \log \pi(\tilde{a}_i(s_i, \xi_i|\theta^\pi)|s_i, \theta^\pi)] \tag{25}$$

Parametry docelowej sieci neuronowej V' są aktualizowane tak jak w przypadku algorytmów DDPG i TD3 z wykorzystaniem uśredniania Polyaka:

$$\theta^{V'} \leftarrow \rho\theta^{V'} + (1 - \rho)\theta^V \tag{26}$$

SAC został podsumowany w formie pseudokodu jako Algorytm 3.

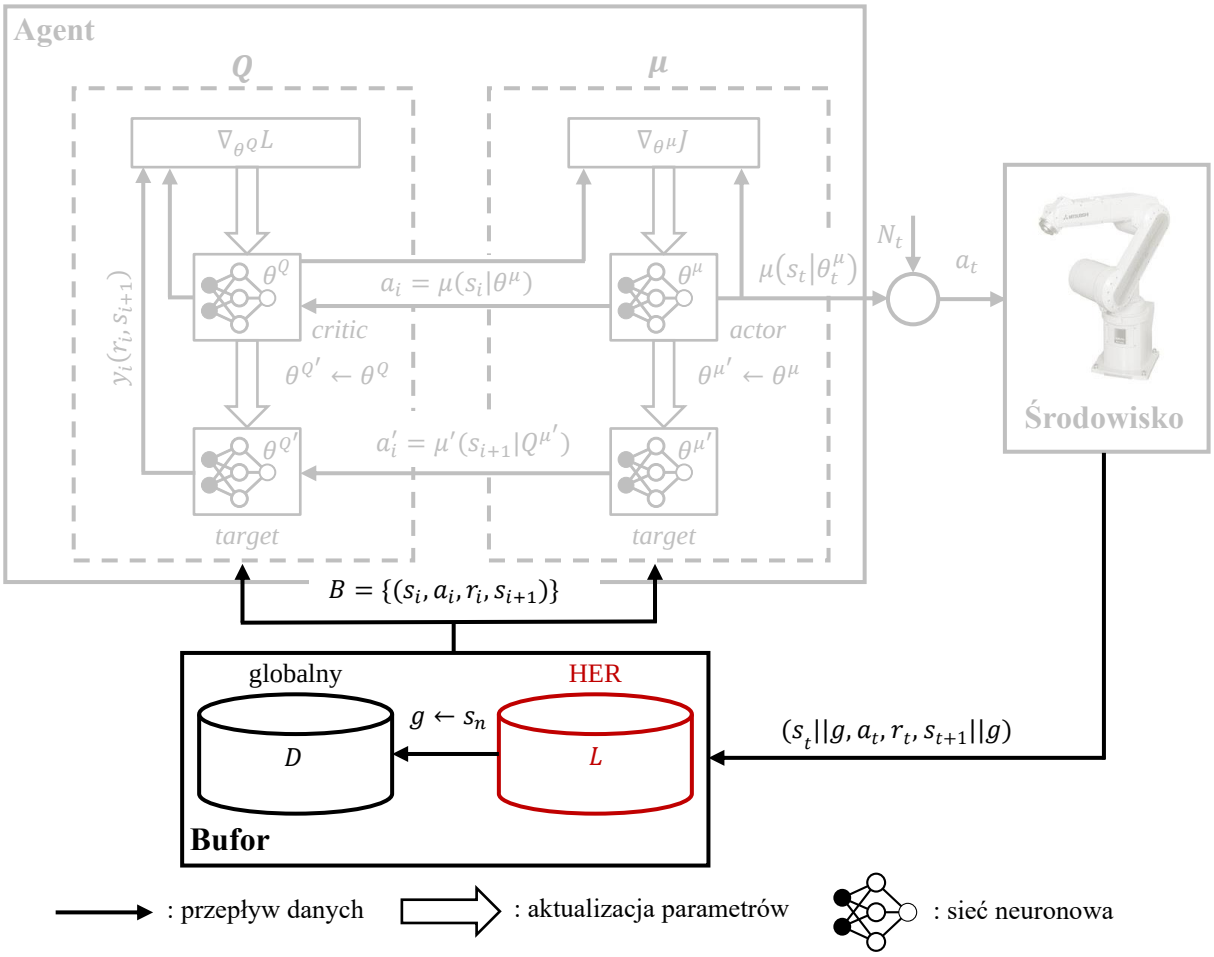
Algorytm 3. Soft Actor Critic (SAC)

Inicjalizacja parametrów sieci neuronowych $\theta^\pi, \theta^{Q^1}, \theta^{Q^2}, \theta^V$ oraz $\theta^{V'} \leftarrow \theta^V, \theta^{\pi'} \leftarrow \theta^\pi$
Inicjalizacja bufora D
for $t = t$ ... T: **do**
  obserwuj stan s_t, wybierz akcję $a_t \sim \pi(\cdot|s_t, \theta^\pi)$
  wykonaj akcję a_t, otrzymaj nagrodę r_t, obserwuj stan s_{t+1} oraz sygnał d_t
  przechowaj w buforze D krotkę $(s_t, a_t, r_t, s_{t+1}, d_t)$
  if aktualizuj parametry sieci **then**
    pobierz losową partię danych $B = \{(s_i, a_i, r_i, s_{i+1}, d_i)\}$ z bufora D
    oblicz y^Q i y^V
      $y_i^Q(r_i, s_{i+1}, d_i) = r_i + \gamma(1 - d_i)V'(s_{i+1}|\theta^{V'})$ (19)
      $y_i^V(s_i) = \min_{j=1,2} Q_j(s_i, \tilde{a}_i|\theta^{Q^j}) - \alpha_T \log \pi(\tilde{a}_i|s_i, \theta^\pi)$ $\tilde{a} \sim \pi(\cdot|s, \theta^\pi)$ (20)
    aktualizuj parametry sieci neuronowych z parametrami $\theta^{Q^1}, \theta^{Q^2}$ dla j=1, 2
      $\nabla_{\theta^{Q^j}} L_j = \nabla_{\theta^{Q^j}} \frac{1}{|B|} \sum [y_i^Q(r_i, s_{i+1}, d_i) - Q_j(s_i, a_i|\theta^j)]^2$ $\theta^{Q^j} \leftarrow \theta^{Q^j} - \alpha_Q \nabla_{\theta^Q} L_j$ (21), (22)
    aktualizuj parametry sieci neuronowej z parametrami θ^V
      $\nabla_{\theta^V} L_V = \nabla_{\theta^V} \frac{1}{|B|} \sum [y_i^V(s_i) - V(s_i|\theta^V)]^2$ $\theta^V \leftarrow \theta^V - \alpha_V \nabla_{\theta^V} L_V$ (23)
    aktualizuj parametry sieci neuronowej z parametrami θ^π
      $\nabla_{\theta^\pi} J = \nabla_{\theta^\pi} \frac{1}{|B|} \sum [\min_{j=1,2} Q_j(s_i, \tilde{a}_i(s_i, \xi_i|\theta^\pi)|\theta^{Q^j}) - \alpha_T \log \pi(\tilde{a}_i(s_i, \xi_i|\theta^\pi)|s_i, \theta^\pi)]$ (25)
      $\theta^\pi \leftarrow \theta^\pi + \alpha_\pi \nabla_{\theta^\pi} J$

19: aktualizuj parametry sieci docelowych (*target network*)
20: $\theta^{V'} \leftarrow \rho\theta^{V'} + (1-\rho)\theta^{V}$ (26)

5.2.4 Hindsight Experience Replay

Hindsight Experience Replay (HER) jest tak zwanym buforem przeszłego doświadczenia i dzięki swojej efektywności umożliwia naukę z nagród, które są binarne (0 lub 1). Ponieważ HER jest dodatkowym buforem, może on być używany w kombinacji z algorytmami typu *off-policy*. Schemat blokowy HER w kombinacji z DDPG (zaznaczonym kolorem szarym) został przedstawiony na Rys. 28. Dodatkowy bufor utworzony do algorytmu HER został oznaczony kolorem czerwonym. Bufory D oraz L na Rys. 28 zostały przedstawione jako dwa odrębne bufory, żeby uwypuklić algorytm HER na schemacie. W rzeczywistość dane z jednego i drugiego bufora są przechowywane razem.



Rys. 28. Schemat blokowy algorytmu Deep Deterministic Policy Gradient zastosowanego w kombinacji z algorytmem Hindsight Experience Replay

Autorzy algorytmu HER proponują następujące rozwiązanie. Zakładając, że agent wykonuje epizod, w którym zaczyna w stanie początkowym s_0, jego celem jest stan docelowy g, ale nie udaje mu się to i kończy w stanie s' na końcu epizodu. Trajektorie z przebiegu epizodu zapisywane są do standardowego bufora D w postaci:

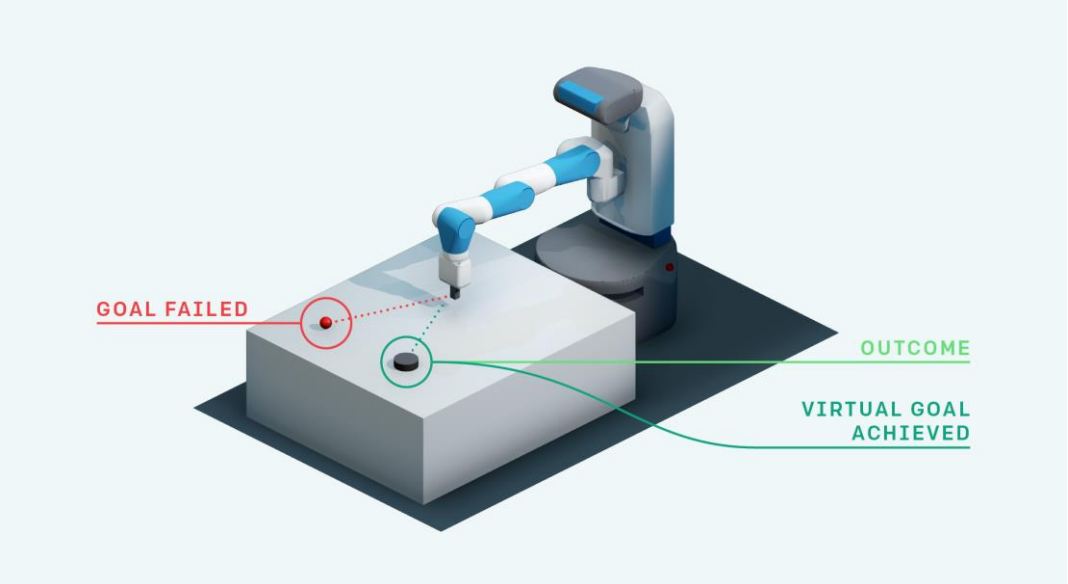
$$\{(s_0||g, a_0, r_0, s_1||g), (s_1||g, a_1, r_1, s_2||g), \dots, (s_n||g, a_n, r_n, s'||g)\} \quad (27)$$

gdzie || oznacza konkatenację, $s_0, s_1 \dots s_n$– stany, $a_0, a_1 \dots a_n$– akcje, $r_0, r_1 \dots r_n$– nagrody.

Idea HER pochodzi od ludzkiego zachowania i nauki rozwiązywania problemów, w którym ludzie czasami w osiągnięciu celu odnoszą porażkę. W takich przypadkach ludzie potrafią rozpoznać co zrobili źle i na bazie błędów oraz nabytego doświadczenia poprawić swoje działania, żeby osiągnąć cel. Głównym założeniem HER jest wyobrażenie, że cel agenta był tak naprawdę w stanie s', dzięki temu może on otrzymać pozytywną nagrodę r'. Dodatkowo w buforze L przechowywana jest następująca trajektoria:

$$\{(s_0||s', a_0, r'_0, s_1||s'), (s_1||s', a_1, r'_1, s_2||s'), \dots, (s_n||s', a_n, r'_n, s'||s')\} \tag{28}$$

Jest to wyobrażona trajektoria, która powstała na bazie nieudanego epizodu, który zakończył się porażką. Dzięki zapisywaniu w buforze takiej trajektorii, agent może nauczyć się jak osiągnąć cel, podejmując akcje nawet z wykorzystaniem błędnej strategii.



Rys. 29. Przykład zastosowania algorytmu HER, w którym robot popycha krążek do pozycji zadanej [openai.com]

Autorzy algorytmu HER zaprezentowali jego działanie na kilku przykładach. Jednym z nich jest zadanie, w którym ramię robota ma popchnąć krążek do określonej, zadanej pozycji na stole. Na rysunku 29 oznaczono czerwony punkt (*goal failed*), który jest pozycją zadaną krążka oraz pozycję końcową (*outcome*), do której został przesunięty w wyniku akcji podjętych przez agenta. Przedstawiony przykład epizodu jest nieudany i w przypadku zastosowania na przykład tylko algorytmu DDPG, agent otrzymywałby tylko negatywne nagrody. Dzięki zastosowaniu algorytmu HER, nieudana trajektoria i jej końcowy wyniki, czyli osiągnięta pozycja krążka (*virtual goal achieved*), zostaje wykorzystana do utworzenia wyobrażonej trajektorii, w której agent otrzymuje pozytywne nagrody, tak jakby rzeczywiście krążek osiągnął pozycję zadaną. Taka metodologia pozwala na stopniową optymalizację polityki agenta nawet na bazie nieudanych epizodów oraz na znacznie szybszą naukę. Dodatkowo, według Autorów algorytmu HER, daje on bardzo dobre rezultaty w przypadku stosowania binarnych nagród typu *sparse*, czyli na przykład 0 i 1 lub -1 i 0. Działanie algorytmu HER zostało podsumowane w formie pseudokodu jako Algorytm 4.

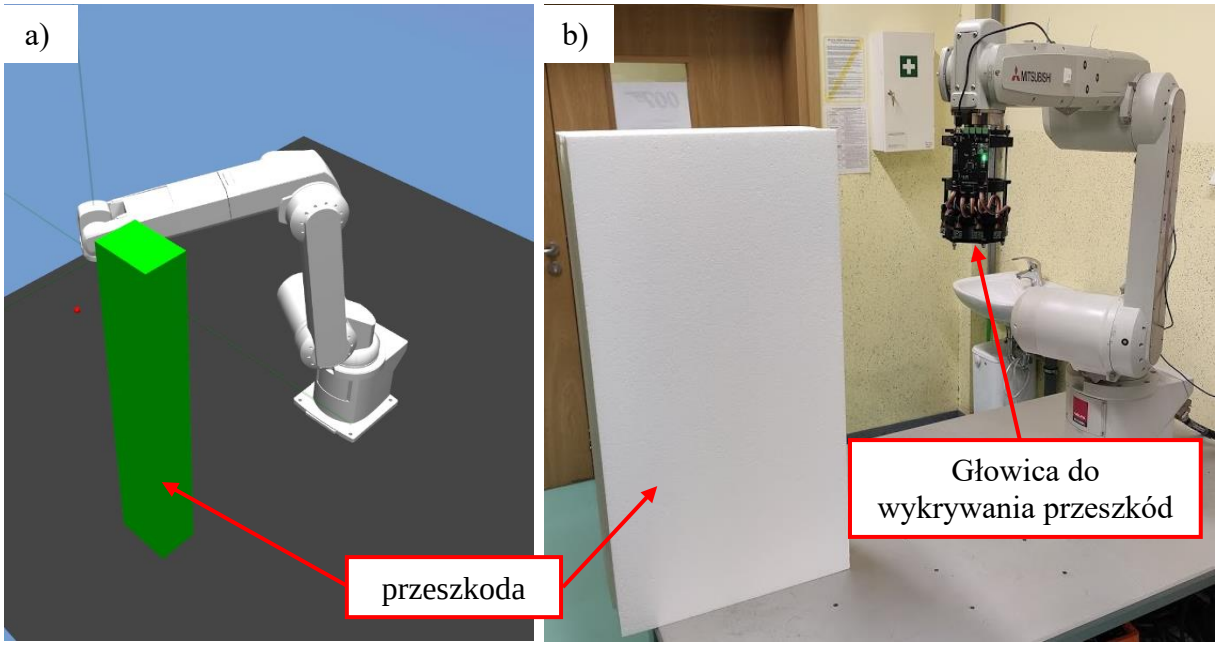
Algorytm 4. Hindsight Experience Replay (HER)

Mając:
• algorytm A uczenia ze wzmocnieniem typu *off-policy* — np. DDPG
• sposób S pobierania celów z bufora — np. $S(s_t, \dots, s_T) = m(s_T)$
• funkcję nagrody R — np. $R(s, a, g)$
Inicjalizacja algorytmu A — np. parametry sieci neuronowych
Inicjalizacja bufora D
for epizod = 1, M **do**
  próbkuj cel g i stan początkowy s_0
  for t = 0, T - 1 **do**
    wybierz akcję a_t z wykorzystaniem strategii z algorytmu A:
      $a_t = \mu(s_t || g)$ — || oznacza konkatenację
    wykonaj akcję a_t oraz obserwuj kolejny stan s_{t+1}
  for t = 0, T - 1 **do**
    $r_t := R(s_t, a_t, g)$
    zapisz krotkę $(s_t || g, a_t, r_t, s_{t+1} || g)$ w buforze D — standardowy bufor z doświadczeniem
    próbkuj zestaw dodatkowych celów $G := S$ — (obecny epizod)
    for $g' \in g_{ns}$ **do**
      $r' := R(s_t, a_t, g')$
      zapisz krotkę $(s_t || g', a_t, r', s_{t+1} || g')$ w buforze L — **HER**
  for t = 1, n **do**
    pobierz losową porcję danych B z bufora D oraz L
    wykonaj optymalizację algorytmu A z wykorzystaniem porcji danych B

6 Stanowisko badawcze

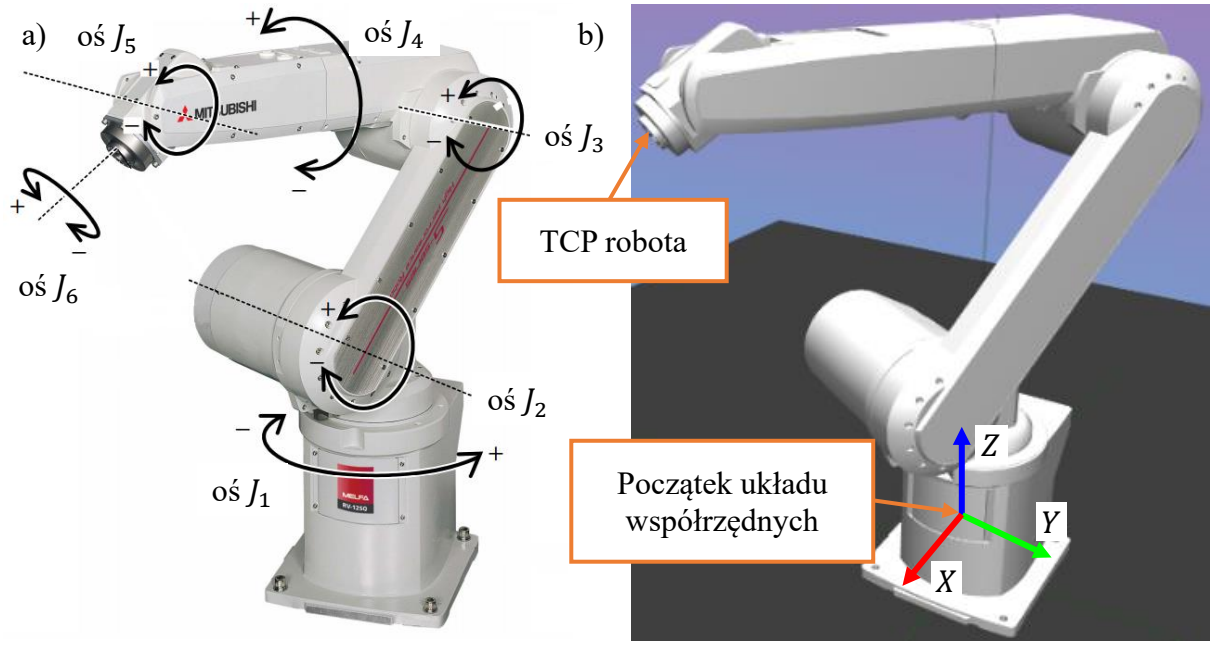
6.1 Robot przemysłowy Mitsubishi RV-12SDL-12S

Do badań symulacyjnych oraz do trenowania algorytmów został opracowany model robota w środowisku symulacyjnym *MuJoCo* (*Multi-Joint dynamics with Contact*) [129] przedstawiony na Rys. 30a. Środowisko to tak zwany silnik fizyki (*physic engine*). Określenie to należy rozumieć jako narzędzie do zamiany parametrów liczbowych na wielkości fizyczne (przesunięcie, prędkość), które mogą być wizualizowane w środowisku wirtualnym. Silnik fizyki to biblioteka programistyczna oraz program do wizualizacji służący do symulacji układów fizycznych. Ma on na celu ułatwienie modelowania i badań w obszarze robotyki, biomechaniki, grafiki i animacji. Do badań doświadczalnych wykorzystano 6-osiowego robota przemysłowego Mitsubishi RV-12SDL-12S z głowicą do wykrywania przeszkód, przedstawionego na Rys. 30b. Widoczna jest tam także przeszkoda usytuowana w polu roboczym.

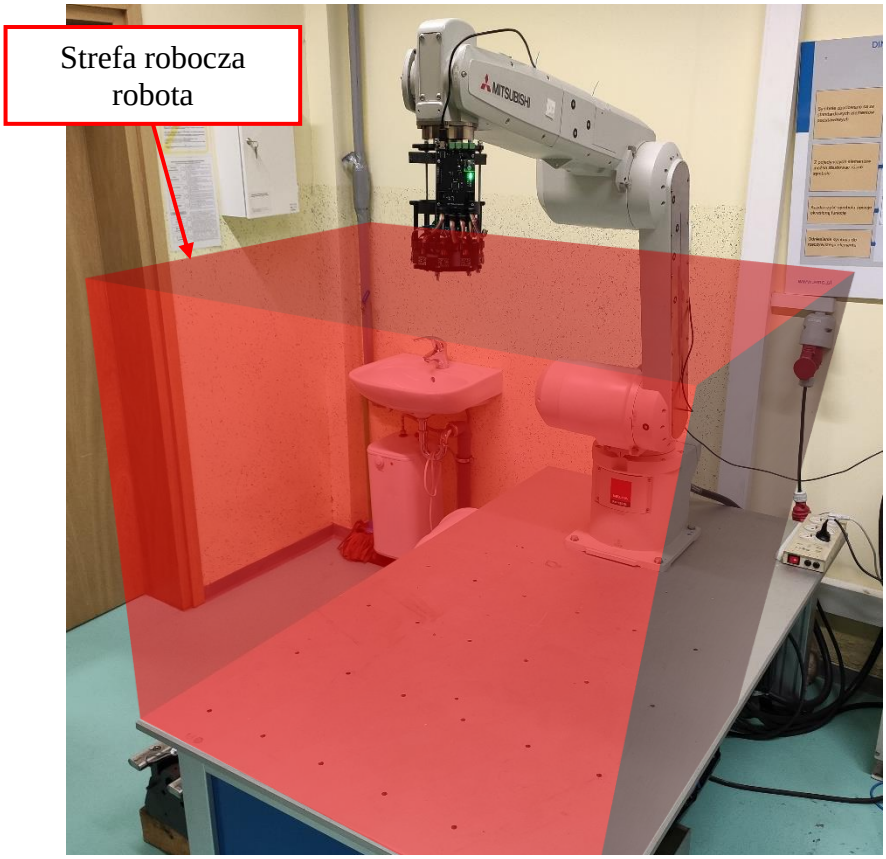


Rys. 30. a) – Model robota w środowisku symulacyjnym MuJoCo. b) – Robot Mitsubishi RV-12SDL-12S z zamontowaną głowicą do wykrywania przeszkód

Wszyscy agenci generujący akcje byli najpierw trenowani do wykonywania różnych zadań z wykorzystaniem środowiska symulacyjnego. Pozwoliło to na zarówno szybsze, jak i bezpieczniejsze ich trenowanie. W celu weryfikacji jakości działania wytrenowanych agentów w środowisku symulacyjnym zbudowano model robota w środowisku *MuJoCo*. Uwzględniono w nim masy oraz momenty bezwładności poszczególnych elementów składowych robota. Dodatkowo do sterowania przemieszczeniem TCP robota w środowisku symulacyjnym zaimplementowano jego kinematykę odwrotną na bazie modelu matematycznego i struktury kinematycznej, która została opisana w rozdziale 6.2. **Opracowanie modelu robota oraz modelu systemu sterowania oznacza osiągnięcie celów cząstkowych numer 1 i 2.**



Rys. 31. a) – Struktura przegubów robota Mitsubishi RV-12SDL-12S. b) – TCP robota oraz początek układu współrzędnych



Rys. 32. Strefa robocza robota Mitsubishi RV-12SDL-12S wykorzystywana w badaniach symulacyjnych i doświadczalnych

Powtarzalność pozycjonowania robota Mitsubishi według producenta wynosi ±0.05 mm. Struktura przegubów robota została przedstawiona na Rys. 31a. Zakres przemieszczeń

przegubów to: [−170°, +170°], [−100°, +130°], [−130°, +160°], [−160°, +160°], [−120°, +120°] oraz [−360°, +360°]. Robot wyposażony jest w enkodery absolutne, które służą jako elementy do pomiaru położenia. Manipulator komunikuje się z komputerem z wykorzystaniem protokołu TCP/IP (*Transmission Control Protocol/Internet Protocol*). TCP manipulatora oraz początek układu współrzędnych robota zostały przedstawione na Rys. 31b.

Do trenowania algorytmów oraz przeprowadzania badań symulacyjnych został zastosowany komputer PC o następującej konfiguracji: CPU − Intel Core i7-8700K, GPU − GeForce RTX 2080Ti, OS − Ubuntu 18.04. Do sterowania robotem w badaniach doświadczalnych, przeprowadzonych w laboratorium został zastosowany laptop Thinkpad E490 z systemem operacyjnym Ubuntu 18.04.

Strefa robocza robota Mitsubishi RV-12SDL-12S wykorzystywana w badaniach symulacyjnych i doświadczalnych została zaznaczona na Rys. 32. Z uwagi na ograniczenia związane ze sposobem montażu robota w laboratorium, miała ona następujące wymiary: oś X: od 0.6m do 1.05m, oś Y: od −0.5m do 0.5m, oś Z: od 0m do 1.1m, względem początku układu współrzędnych (Rys. 31b).

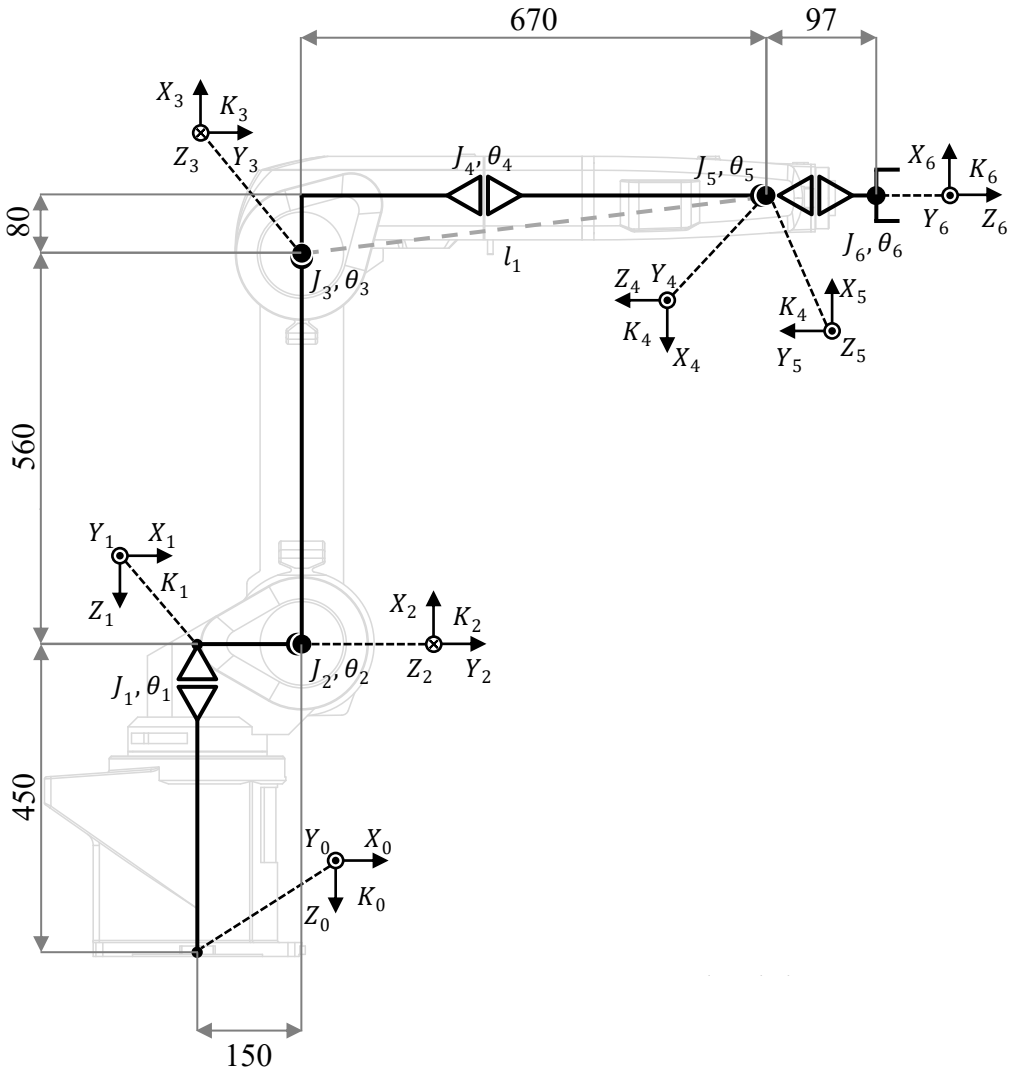
6.2 Kinematyka odwrotna 6-osiowego robota przemysłowego

Do opisu robota zastosowano notację Denavita-Hartenberga (D-H) [130]. W celu implementacji kinematyki odwrotnej utworzono tabelę parametrów D-H (Tabela 1) robota Mitsubishi RV-12SDL-12S, opracowaną na podstawie jego wymiarów i struktury kinematycznej przedstawionej na Rys. 33 [131].

Tabela 1. Tabela parametrów D-H robota Mitsubishi RV-12SDL-12S

$\boldsymbol{i}$	$\boldsymbol{a}$ [mm]	$\boldsymbol{\alpha}$ [rad]	$\boldsymbol{d}$ [mm]	$\boldsymbol{\theta_n}$ [rad]
0	150	0	-	-
1	0	$\frac{\pi}{2}$	−450	θ_1
2	560	0	0	$-\frac{\pi}{2}+\theta_2$
3	80	$\frac{\pi}{2}$	0	θ_3
4	0	$-\frac{\pi}{2}$	−670	$\pi+\theta_4$
5	0	$\frac{\pi}{2}$	0	$\pi+\theta_5$
6	-	-	97	θ_6

Kąt obrotu θ_1 dla pierwszego przegubu można obliczyć poprzez wykorzystanie rzutu wektora $\overrightarrow{{}^{0}_{4}P_{K_0}}$ łączącego punkt K_0 i K_4 na płaszczyznę XY. Macierz transformacji ${}^{0}_{G}T$ w odniesieniu do początku układu współrzędnych robota dana jest wzorem:



Rys. 33. Struktura kinematyczna oraz wymiary robota Mitsubishi RV-12SDL-12S

$$ {}^0_G T = \begin{bmatrix} {}^0_G T_{11} & {}^0_G T_{12} & {}^0_G T_{13} & {}^0_G T_{14} \\ {}^0_G T_{21} & {}^0_G T_{22} & {}^0_G T_{23} & {}^0_G T_{24} \\ {}^0_G T_{31} & {}^0_G T_{32} & {}^0_G T_{33} & {}^0_G T_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix} \Rightarrow \overrightarrow{{}^0_6 N_{K_0}} = \begin{bmatrix} {}^0_G T_{13} \\ {}^0_G T_{23} \\ {}^0_G T_{33} \end{bmatrix} \tag{29} $$

$$ \begin{cases} \overrightarrow{{}^0_6 P_{K_0}} = d_6 \times \overrightarrow{{}^0_6 N_{K_0}} \\ \overrightarrow{{}^0_6 P_{K_0}} = \begin{bmatrix} {}^0_G T_{14} \\ {}^0_G T_{24} \\ {}^0_G T_{34} \end{bmatrix} \end{cases} \Rightarrow \overrightarrow{{}^0_4 P_{K_0}} = \overrightarrow{{}^0_6 P_{K_0}} - \overrightarrow{{}^4_6 P_{K_0}} = \begin{bmatrix} {}^0_G T_{14} - d_6 {}^0_G T_{13} \\ {}^0_G T_{24} - d_6 {}^0_G T_{23} \\ {}^0_G T_{34} - d_6 {}^0_G T_{33} \end{bmatrix} \tag{30} $$

Z przedstawianych wzorów można wyprowadzić wzór na kąt obrotu θ_1 dla pierwszego przegubu:

$$\theta_1 = \begin{cases} atan2({}^0_G T_{24} - d_6 {}^0_G T_{23}, {}^0_G T_{14} - d_6 {}^0_G T_{13}) \\ atan2({}^0_G T_{24} - d_6 {}^0_G T_{23}, {}^0_G T_{14} - d_6 {}^0_G T_{13}) + \pi \end{cases} \tag{31}$$

Równanie na kąt obrotu θ_3 dla trzeciego przegubu robota jest następujące:

$$\theta_3 = \begin{cases} \pi - \Phi - \varepsilon \\ \pi + \Phi - \varepsilon \end{cases} \tag{32}$$

gdzie współczynniki Φ oraz ε dane są wzorami:

$$\Phi = asin\left(\frac{\left(l_1^2 - a_2^2 + \left|\overrightarrow{{}^2_4 P_{K_0}}\right|^2\right)}{2\left|\overrightarrow{{}^2_4 P_{K_0}}\right| l_1}\right) + asin\left(\frac{\left(\left|\overrightarrow{{}^2_4 P_{K_0}}\right| - \frac{l_1^2 - a_2^2 + \left|\overrightarrow{{}^2_4 P_{K_0}}\right|^2}{2\left|\overrightarrow{{}^2_4 P_{K_0}}\right|}\right)}{a_2}\right) \tag{33}$$

$$\varepsilon = \mathrm{a}tan2(-d_4, a_3) \tag{34}$$

Kąt obrotu θ_2 dla drugiego przegubu można obliczyć z wykorzystaniem zależności:

$$\theta_2 = \begin{cases} \frac{\pi}{2} - (|\beta_1| + \beta_2) \\ \frac{\pi}{2} + (|\beta_1| - \beta_2) \end{cases} \tag{35}$$

gdzie współczynniki β_1 oraz β_2 dane są wzorami:

$$\beta_1 = \mathrm{a}tan2\left(\overrightarrow{{}^2_4 P_{K_{2x}}}, \overrightarrow{{}^2_4 P_{K_{2y}}}\right) \tag{36}$$

$$\beta_2 = \mathrm{a}sin\left(\frac{\left(l_1^2 + a_2^2 - \left|\overrightarrow{{}^2_4 P_{K_0}}\right|^2\right)}{2 l_1 a_2}\right) + \mathrm{a}sin\left(\frac{\left(l_1 - \frac{l_1^2 + a_2^2 - \left|\overrightarrow{{}^2_4 P_{K_0}}\right|^2}{2 l_1}\right)}{\left|\overrightarrow{{}^2_4 P_{K_0}}\right|}\right) \tag{37}$$

Kąt obrotu θ_5 dla piątego przegubu można obliczyć z wykorzystaniem wzoru:

$$\theta_5 = \pi - \mathrm{a}cos\left(\overrightarrow{{}^0_4 N_{K_0}} \cdot \overrightarrow{{}^0_6 N_{K_0}}\right) \tag{38}$$

Do obliczenia kątów obrotu w przegubach numer cztery i sześć θ_4 i θ_6 została zastosowana macierz rotacji ${}^4_6 R$:

$$ {}^{4}_{6}R = \begin{bmatrix} -c_4c_5c_6 - s_4s_6 & c_4c_5s_6 - s_4c_6 & -c_4s_5 \\ -s_4c_5c_6 + c_4s_6 & s_4c_5s_6 + c_4c_6 & -s_4s_5 \\ s_5c_6 & -s_5s_6 & -c_5 \end{bmatrix} \tag{39} $$

gdzie c_4 to skrócony zapis $\cos(\theta_4)$, a s_4 to $\sin(\theta_4)$. Macierz ${}^{4}_{6}R$ może być obliczona według wzoru:

$$ {}^{4}_{6}R = \begin{bmatrix} {}^{4}_{6}R_{11} & {}^{4}_{6}R_{12} & {}^{4}_{6}R_{13} \\ {}^{4}_{6}R_{21} & {}^{4}_{6}R_{22} & {}^{4}_{6}R_{23} \\ {}^{4}_{6}R_{31} & {}^{4}_{6}R_{32} & {}^{4}_{6}R_{33} \end{bmatrix} \tag{40} $$

Z przedstawianych wzorów można wyprowadzić kąt obrotu θ_4 i θ_6:

$$ \theta_4 = atan2(-{}^{4}_{6}R_{23}, -{}^{4}_{6}R_{13}) \tag{41} $$

$$ \theta_6 = atan2(-{}^{4}_{6}R_{32}, -{}^{4}_{6}R_{31}) \tag{42} $$

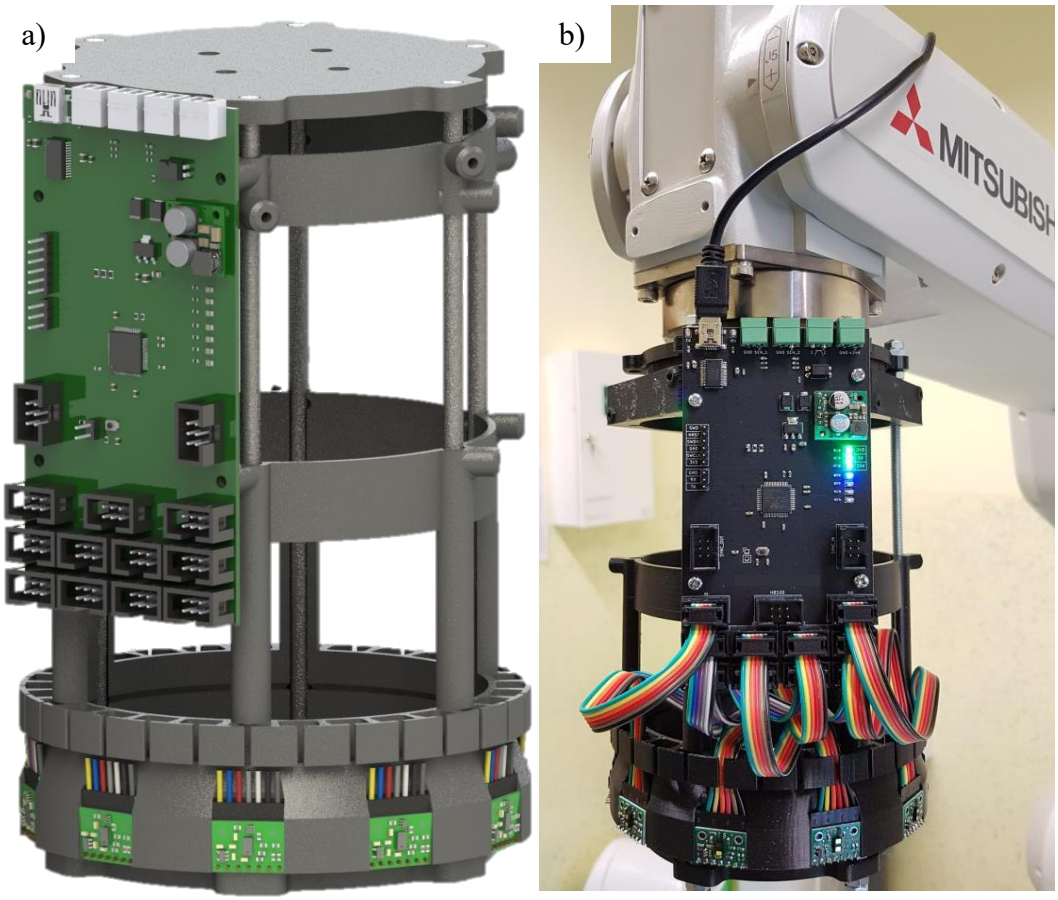
Przedstawione w tym rozdziale równania zostały zaimplementowane w programie, tworząc model kinematyki odwrotnej robota Mitsubishi, którego wejściem była zadana pozycja TCP robota, na Rys. 33 oznaczona jako punkt K_6 (współrzędne X_6, Y_6, Z_6). W trakcie badań orientacja TCP robota była stała. Wyjściem z modelu były kąty obrotu poszczególnych przegubów od θ_1 do θ_6.

6.3 Głowica do wykrywania przeszkód

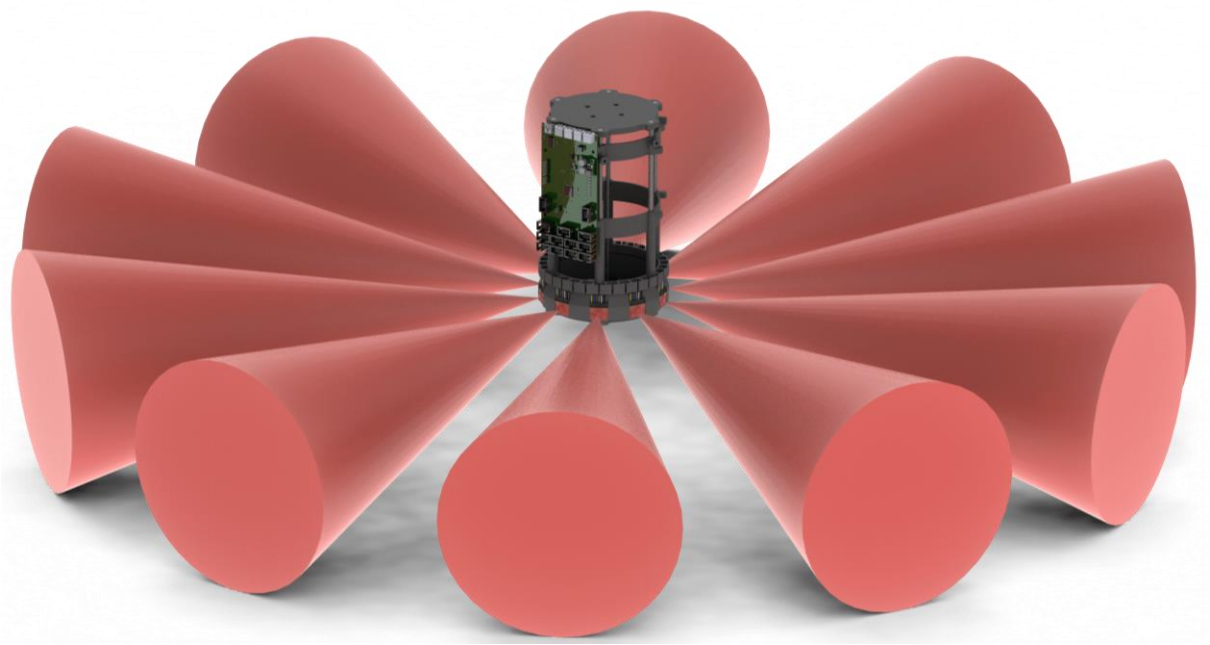
W trakcie trenowania agentów oraz badań symulacyjnych przeszkoda miała kształt prostopadłościanu. W badaniach wyznaczano odległość d_{obs} jako odległość TCP robota do najbliższego punktu na powierzchni prostopadłościanu. Jej wyznaczenie w środowisku symulacyjnym było proste, ponieważ położenie oraz rozmiar przeszkody, a także położenie TCP robota były znane.

W badaniach doświadczalnych, przeszkoda była ustawiana w przypadkowych miejscach i wyznaczenie odległości d_{obs} względem układu współrzędnych robota wymagało zastosowania urządzenia pomiarowego. W trakcie badań zmieniano także jej kształt. Dlatego zaprojektowano i zbudowano dedykowaną głowicę z czujnikami optycznymi (laserowymi), która była przymocowana do kiści robota (Rys. 30b i Rys. 32). Zaprojektowaną głowicę pokazano na Rys. 34. Do wykrywania przeszkód zastosowano 10 czujników laserowych VL53L1 typu *Time-of-Flight*. Prototyp głowicy został wykonany z wykorzystaniem technologii druku 3D. Poszczególne elementy zostały wykonane z materiału PLA, do których zostały przymocowane czujniki oraz płytka z elektroniką. Całość została zamontowana na gwintowanych stalowych prętach. Widok rozstrzelony głowicy razem z zaznaczonymi poszczególnymi elementami został przedstawiony na Rys. 36.

Czujniki VL53L1 mierzą czas, jaki upływa od wyemitowania światła od czujnika do obiektu znajdującego się przed nim i z powrotem. Charakteryzują się one małym błędem wynoszącym maksymalnie $\pm$2.5%. Każdy z nich jest wyposażony w laser typu *Vertical-Cavity Surface-Emitting Laser* (VCSEL), emitujący światło podczerwone o długości fali 940nm oraz fotodiodę lawinową SPAD (*Single Photon Avalanche Detector*), przeznaczoną do



Rys. 34. Zaprojektowana głowica do wykrywania przeszkód: a) – model CAD, b) – zamontowana na kiści robota

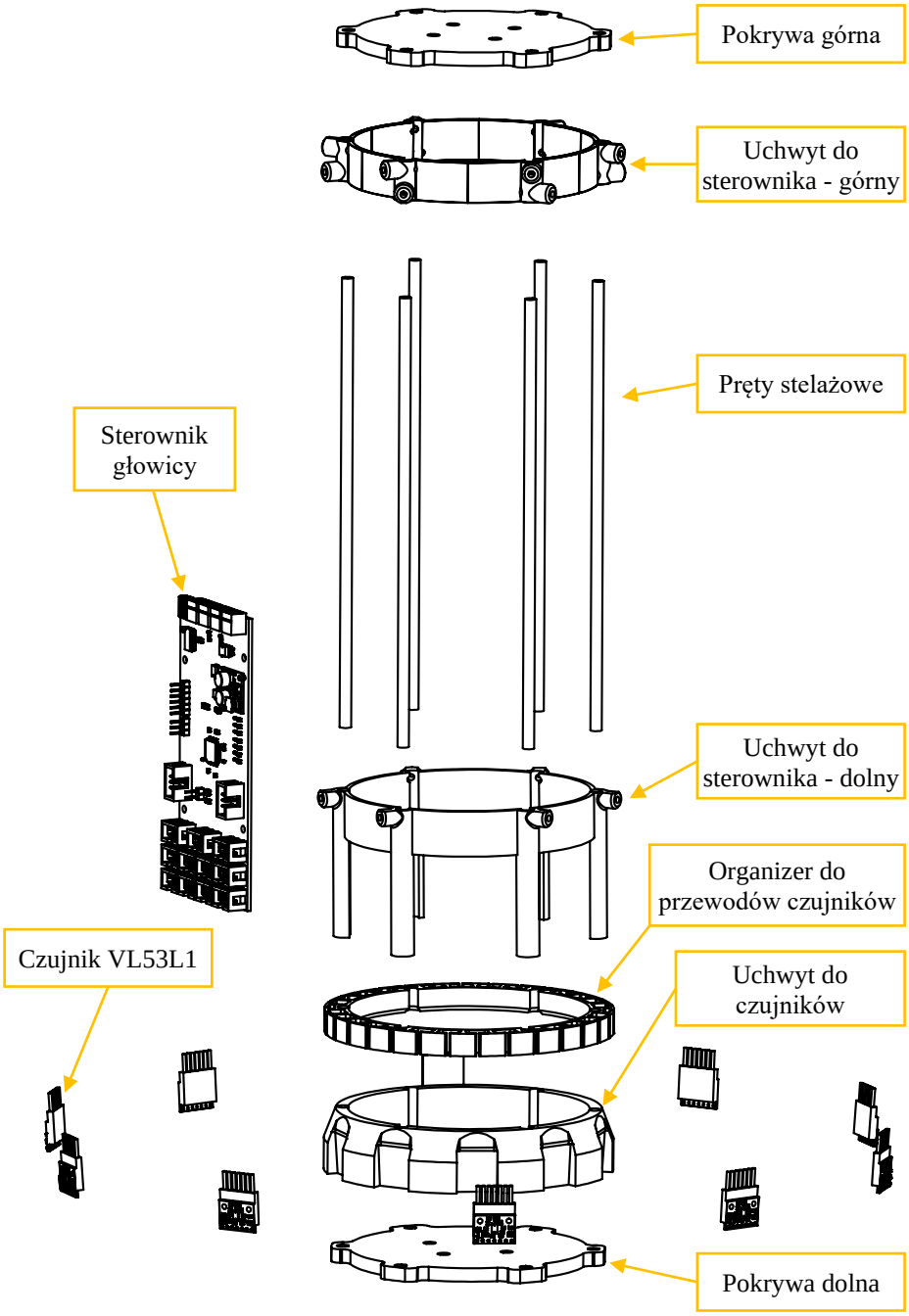


Rys. 35. Wizualizacja pola widzenia każdego z czujników VL53L1 w głowicy

wykrywania odbitego światła. Pole widzenia (*field of view*) każdego czujnika optycznego to stożek o kącie rozwarcia 27°. Czujniki umieszczono na obwodzie głowicy. Pole widzenia dla

każdego z czujników zostało zwizualizowane na Rys. 35. Mimo że nie pokrywa ono całej przestrzeni przed głowicą, to jest ona w stanie prawidłowo wykrywać przeszkody o wymiarach większych niż 40mm w odległości 10mm od głowicy oraz większych niż 85mm w odległości 300mm od głowicy. Zakres wykrywania przeszkód przez głowicę został ograniczony do 1.5m.

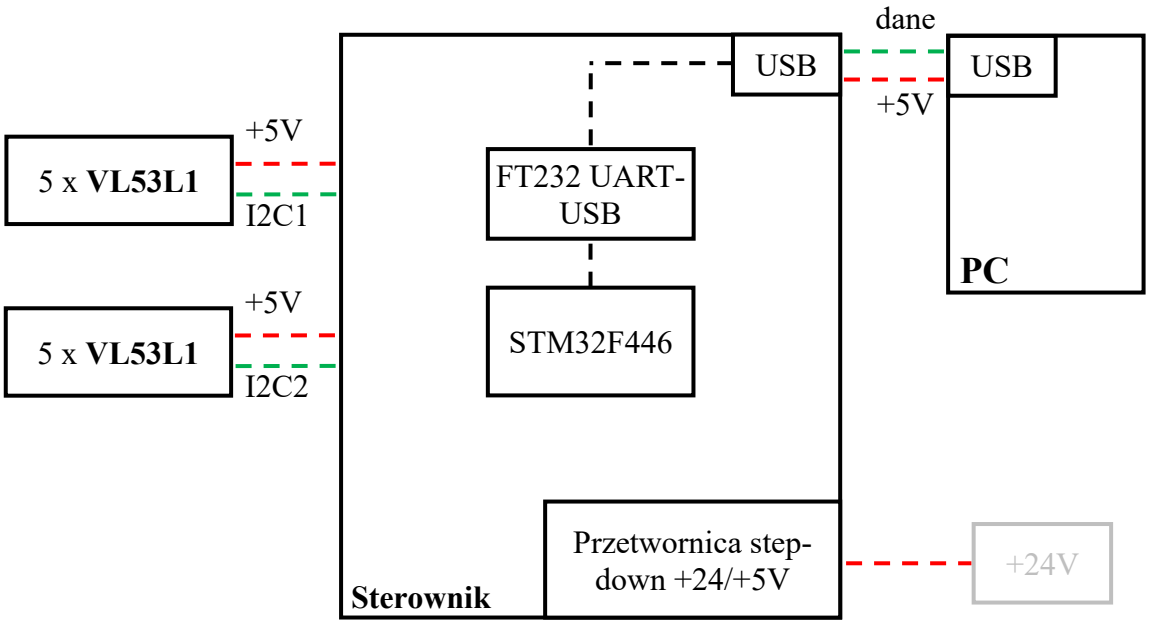
W celu zbierania danych, wyznaczania odległości i wstępnej analizy danych zaprojektowano i wykonano sterownik głowicy z mikrokontrolerem STM32F446 firmy STMicroelectronics. Komunikował się on z komputerem sterującym przez interfejs szeregowy UART-USB. Uproszczony schemat elektroniczny sterownika został



Rys. 36. Widok rozstrzelony głowicy do wykrywania przeszkód

przedstawiony na Rys. 37. Zebrane dane były przesyłane do komputera, który następnie dokonywał ich analizy w celu określenia najkrótszej odległości między głowicą, a przeszkodą i sterował robotem. Do wyznaczenia odległości d_{obs} była wybierana najmniejsza odległość z jednego z czujników.

Czujniki komunikowały się z mikrokontrolerem z wykorzystaniem interfejsu I2C. Dane z nich pobierane były w sposób zdarzeniowy. Do każdego czujnika wykorzystywany był osobny pin mikrokontrolera. Stan niski na pinie oznaczał gotowość do odbioru danych z czujnika. Z uwagi na ilość danych i maksymalną dostępną prędkość komunikacji I2C w mikrokontrolerze STM32F446 wynoszącą 400kHz, czujniki VL53L1 zostały podzielone na dwie oddzielne szyny komunikacji. Do każdej szyny podłączone było po 5 czujników. Zapewniało to częstotliwość pobierania danych z każdego z czujników na poziomie 15Hz.

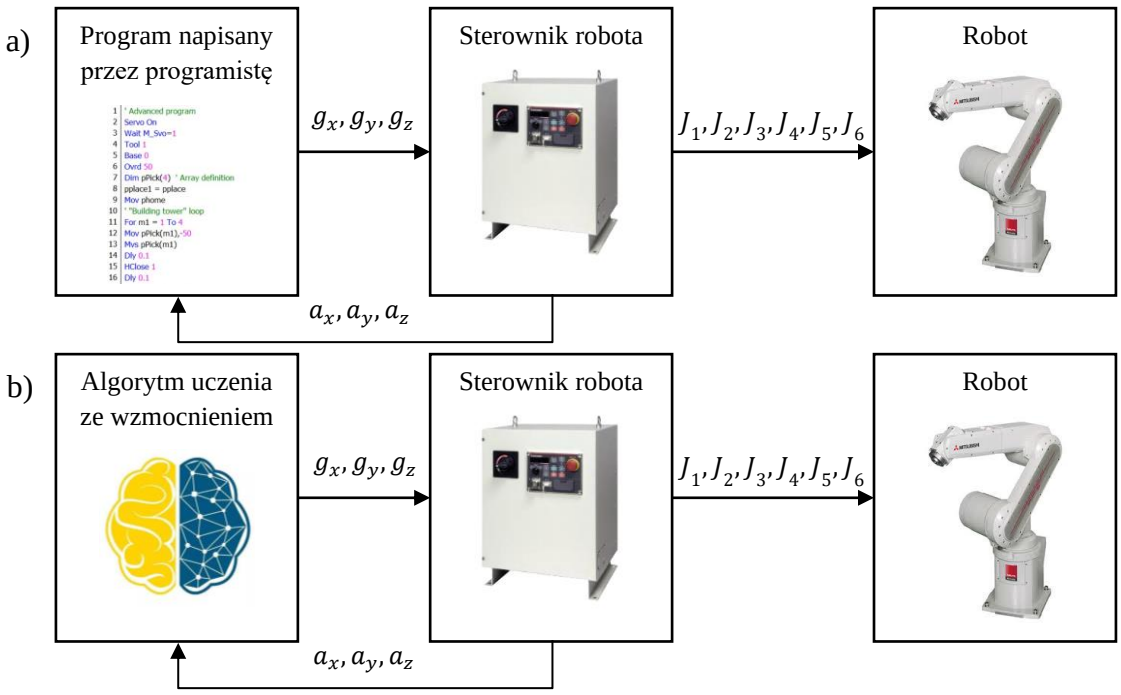


Rys. 37. Uproszczony schemat elektroniczny sterownika głowicy do wykrywania przeszkód

Zbudowanie głowicy do wykrywania przeszkód oraz opracowanie algorytmu pobierającego z niej dane i analizującego je oznacza osiągnięcie celu cząstkowego numer 3.

6.4 Architektura systemu sterowania

Uproszczony schemat blokowy stosowanego powszechnie układu sterowania pracą robota pokazano na Rys. 38a. Sterownik, na podstawie napisanego programu przez programistę przesyła do sterownika robota współrzędne pozycji zadanej (g_x, g_y, g_z) TCP manipulatora (na rysunku pominięto zadaną orientację (g_a, g_b, g_c) TCP manipulatora). Na tej podstawie sterownik robota steruje poszczególnymi napędami obracając ramiona w przegubach $J_1, J_2, J_3, J_4, J_5, J_6$ o odpowiednie kąty, tak aby TCP robota przemieścił się do zadanej pozycji. Do napisanego programu mogą być przesyłane dane w formie sprzężenia zwrotnego ze sterownika robota, na przykład aktualna pozycja TCP robota (a_x, a_y, a_z).



Rys. 38. Uproszczony schemat blokowy sterowania pracą robota: a) – metoda standardowa, b) – algorytmy uczenia ze wzmocnieniem

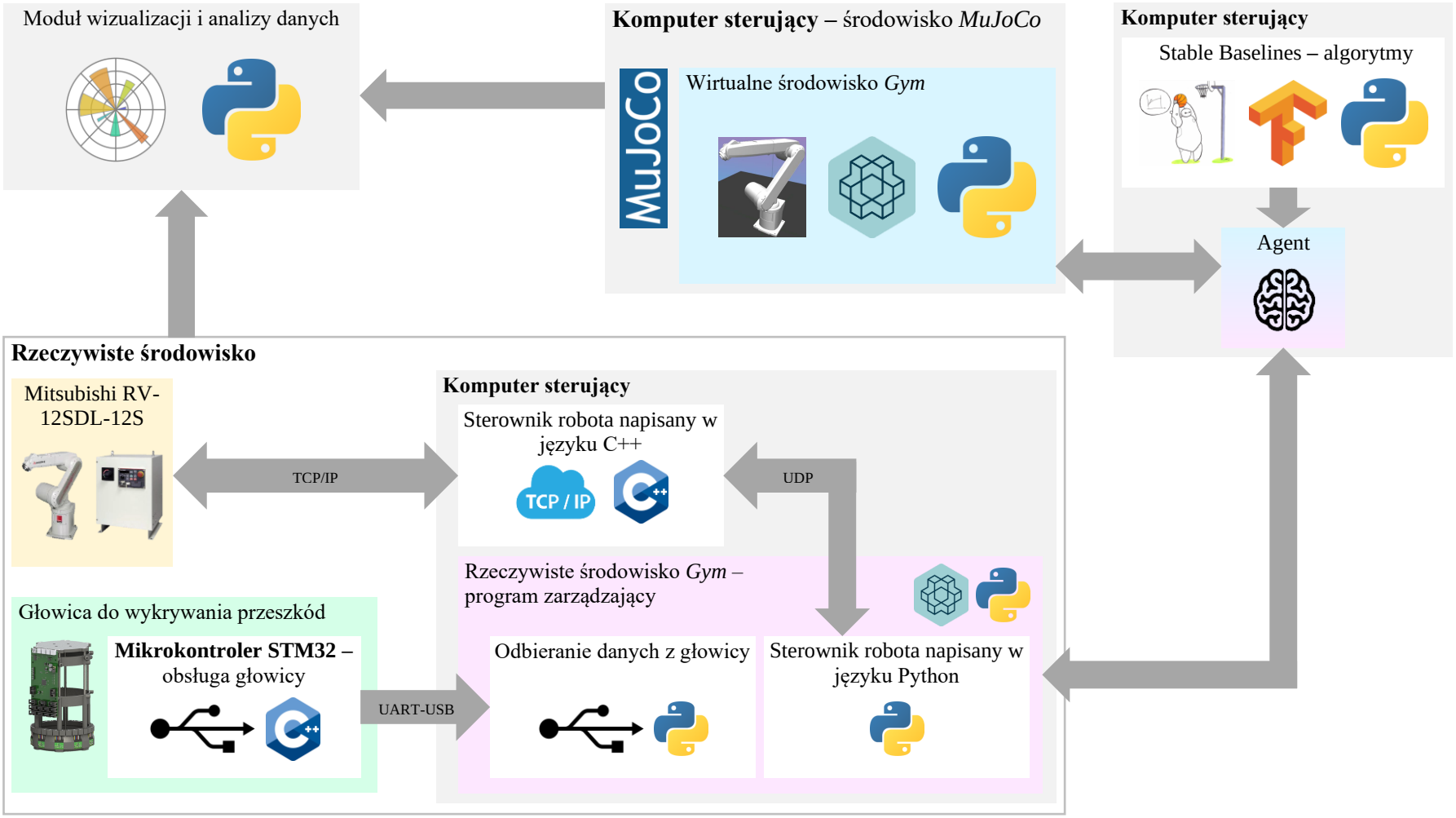
W rozwiązaniu badanym w niniejszej pracy, agent wytrenowany z wykorzystaniem algorytmów uczenia ze wzmocnieniem, zastąpił napisany przez programistę program do sterowania rzeczywistym robotem przemysłowym. Tak jak pokazano na Rys. 38b, akcja agenta to zadane współrzędne TCP robota (g_x, g_y, g_z). Obserwacją będzie w tym przypadku aktualna pozycja TCP robota (a_x, a_y, a_z).

Do przeprowadzenia badań symulacyjnych i doświadczalnych napisano dedykowane oprogramowanie podzielone na moduły, które zostały przedstawione na Rys. 39. Moduły związane ze środowiskiem dotyczące sterownia modelem robota lub rzeczywistym robotem, zostały napisane zgodnie z interfejsem programowania aplikacji (API) – *Gym* [132], zaproponowanym przez firmę OpenAI. *Gym* definiuje pewien zbiór reguł związany z nazwami funkcji oraz strukturą klas.

Oprogramowanie składa się z następujących modułów:

- Stable Baselines – biblioteka służąca do implementacji wykorzystywanych algorytmów uczenia ze wzmocnieniem. Jej elementy wykorzystywane są do utworzenia obiektu agenta.
- Wirtualne środowisko *Gym* – model robota zaimplementowany w *MuJoCo*; zawiera funkcje: obliczające kinematykę odwrotną robota, sterujące TCP manipulatora, pobierające dane o stanie robota, zarządzające procesem trenowania agenta. Moduł opracowano zgodnie z API *Gym*.
- Sterownik robota napisany w języku C++ – program uruchomiony na komputerze sterującym pozwalający między innymi na komunikację z robotem Mitsubishi z wykorzystaniem protokołu TCP/IP. Program przesyła komendy w postaci zadanych pozycji oraz odczytuje aktualne pozycje TCP i kąty przegubów robota.

- Sterownik robota napisany w języku Python – program, który jest adapterem (*wrapper*), zawiera funkcje wywołujące dla programu sterownika robota, napisanego w języku C++. Moduł jest niezbędny, bo całość oprogramowania związana z algorytmami jest napisana w języku Python.
- Rzeczywiste środowisko *Gym* – funkcje sterujące pozycją TCP manipulatora w przestrzeni trójwymiarowej, pobierające dane o stanie robota, analizujące dane z głowicy do wykrywania przeszkód, zarządzające procesem trenowania agenta. Moduł opracowano zgodnie z API *Gym*.
- Odbieranie danych z głowicy – program uruchomiony na komputerze sterującym, odbierający dane z głowicy do wykrywania przeszkód z wykorzystaniem standardu USB.
- Głowica do wykrywania przeszkód – program napisany na mikrokontroler STM32F446 służący do odbierania danych z czujników VL53L1 oraz do wstępnej ich analizy oraz wysłania z wykorzystaniem standardu USB do komputera.
- Wizualizacja i analiza danych – program służący do tworzenia wykresów 2D, 3D, oraz analizy danych, na przykład obliczania błędów pozycjonowania itd.



Rys. 39. Architektura modułowa sprzętu i oprogramowania sterującego

7 Badania algorytmu pozycjonowania robota

7.1 Opis metodologii badań

W praktyce przemysłowej, pozycjonowanie TCP robotów jest jednym z częściej wykonywanych zadań. Aby je zrealizować konieczne jest wpisanie odpowiedniego zestawu poleceń do programu, z podaniem współrzędnych położenia końcowego. W ramach niniejszej pracy podjęto badania nad zastosowaniem algorytmów uczenia ze wzmocnieniem do pozycjonowania, czyli sterowania przemieszczeniem TCP robota do zadanego punktu. Zadaniem badanych algorytmów było nauczenie się, jak osiągnąć zadaną pozycję TCP robota w przestrzeni trójwymiarowej. W trakcie badań robot nie był obciążony. W pierwszym etapie prac, w obszarze roboczym nie było przeszkód, które robot musiałby omijać. Do badań wybrano sześć algorytmów uczenia ze wzmocnieniem: DDPG, TD3, SAC, DDPG+HER, TD3+HER i SAC+HER. W celu wytrenowania agentów w środowisku symulacyjnym zastosowano trzy funkcje nagród: *sparse, dense* oraz *dense trajectory*. W każdym epizodzie treningowym, pozycja startowa robota (s_x, s_y, s_z) była taka sama, a pozycja zadana była losowo generowana z wykorzystaniem dystrybucji jednolitej (*uniform distribution*) z przestrzeni roboczej robota, jako współrzędne celu (g_x, g_y, g_z). Dla każdego epizodu maksymalna liczba kroków wynosiła 50. Epizod kończył się po wykonaniu tej liczby kroków albo gdy TCP robota znajdował się w określonej odległości mniejszej niż d_{th} od pozycji zadanej, która była progiem wskazującym, że manipulator przemieścił się w przestrzeni trójwymiarowej do pozycji zadanej z akceptowalną tolerancją, czyli uzyskał sukces. Algorytmy DDPG, SAC, TD3 były trenowane dla $1{\cdot}10^6$ kroków uczenia, a algorytmy połączone z HER dla $5{\cdot}10^4$ kroków. Czas treningu na opisanym w rozdziale 6.1 komputerze wynosił dla: DDPG, SAC, TD3 około 4h, a dla HER około 15min. W celu optymalizacji wartości hiperparametrów algorytmów wykorzystano bibliotekę Optuna [133]. Hiperparametrami są liczby, które nie są optymalizowane przez algorytm w trakcie nauki, ale są podawane przez programistę. Jeden cykl takiej optymalizacji dla każdego algorytmu trwał około 80h (DDPG, SAC, TD3) i 16h (HER). Optymalizacja hiperparametrów była wykonywana kilka razy, dla każdego algorytmu, ponieważ testowano różne strategie, które nie zawsze dawały oczekiwane rezultaty.

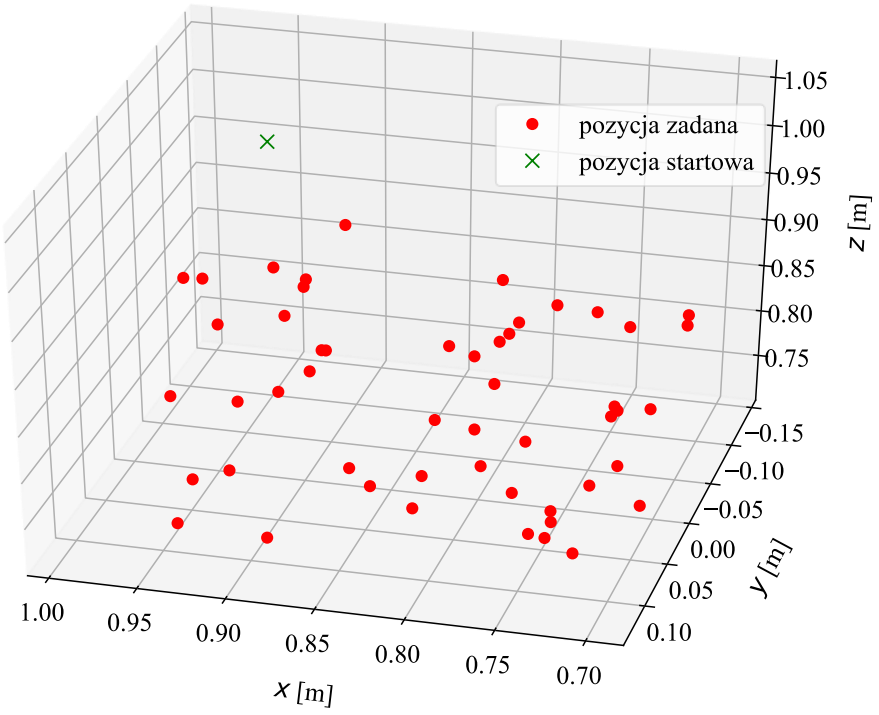
W trakcie badań, dla każdego algorytmu przetestowano trzy odległości (progi) d_{th}: ±50mm, ±5mm i ±0.5mm. Jeden z trzech algorytmów wytrenowany z określonym progiem d_{th} (±50mm, ±5mm, ±0.5mm), który osiągnął najmniejszą finalną różnicę pomiędzy pozycją TCP manipulatora, a pozycją zadaną był brany pod uwagę w dalszej analizie. Niektóre algorytmy w kombinacji z różnymi odległościami d_{th} i funkcjami nagród nie były w ogóle w stanie wykonać zadania i poruszać TCP manipulatora w kierunku pozycji zadanej. Takie kombinacje algorytmów były odrzucane.

Obserwacjami agenta tworzącymi stan s w postaci 9-elementowego wektora były:

- aktualna pozycja TCP manipulatora (a_x, a_y, a_z) [m],
- prędkość liniowa TCP manipulatora (v_x, v_y, v_z) $[\frac{m}{s}]$,
- zadana pozycja (g_x, g_y, g_z) [m].

W każdym kroku sterowania, każdy algorytm generował akcję a, która była zmianą położenia TCP robota w metrach w przestrzeni trójwymiarowej $(\Delta x, \Delta y, \Delta z)$.

W celu porównania i ewaluacji wyników w zakresie dokładności pozycjonowania TCP robota w zadanym punkcie w przestrzeni trójwymiarowej, wylosowano 50 punktów z wykorzystaniem dystrybucji jednolitej, które zastosowano jako współrzędne pozycji zadanej (g_x, g_y, g_z). W epizodach ewaluacyjnych wytrenowani agenci sterujący modelem robota i robotem rzeczywistym, zostali porównani dla tych samych 50 punktów. Wszystkie wylosowane pozycje punktów oraz pozycję startową robota pokazano graficznie na Rys. 40.



Rys. 40. Pozycja startowa oraz pozycje zadane testowane w trakcie ewaluacji sześciu kombinacji algorytmów dla pozycjonowania TCP robota

W rozdziałach poniżej (7.2 i 7.3) przedstawiono wyniki dla wszystkich 50 punktów. W trakcie ewaluacji w badaniach symulacyjnych i doświadczalnych wzięto pod uwagę kształt trajektorii po jakiej poruszał się TCP robota oraz dokładność pozycjonowania TCP robota. Porównywane parametry to odległość (średni błąd e) TCP robota od pozycji zadanej, odchylenie standardowe σ_e błędu e, średnia odległość d_{tr}, która jest odległością pomiędzy trajektorią, po której przemieszczał się TCP robota, a linią prostą wyznaczoną na podstawie pozycji startowej (s_x, s_y, s_z) i zadanej (g_x, g_y, g_z), odchylenie standardowe σ_{dtr} odległości d_{tr}. Dla każdego punktu, błąd pozycjonowania e obliczany był według wzoru:

$$e = \sqrt{\Delta x_T^2 + \Delta y_T^2 + \Delta z_T^2} \tag{43}$$

gdzie $\Delta x_T, \Delta y_T, \Delta z_T$ są różnicami między współrzędnymi położenia TCP robota w ostatnim kroku danego epizodu, a pozycją zadaną. Na wykresach w rozdziale 7 przedstawiono wyniki dla dwóch, wybranych punktów zadanych g_1 i g_2: g_{1x} = 0.99086m, g_{1y} = −0.10163m, g_{1z} =

0.82122m i g_{2x} = 0.71792m, g_{2y} = 0.10862m, g_{2z} = 0.75629m. Pozycja startowa miała następujące współrzędne s_{1x} = 0.91m, s_{1y} = 0.0m, s_{1z} = 1.06m.

7.1.1 Funkcje nagród

Przetestowano trzy funkcje nagrody: *sparse, dense* oraz *dense trajectory*. Pierwsza z nich generowała sygnał binarny, to znaczy agent otrzymywał nagrodę o wartości 0, jeśli różnica pomiędzy pozycją TCP robota, a pozycją zadaną była mniejsza od odległości d_{th} (±50 mm, ±5mm, ±0.5mm). W każdym innym przypadku agent otrzymywał nagrodę równą –1. Funkcja nagrody typu *sparse* obliczana była według wzoru:

$$R_s = \begin{cases} 0, & \text{jeśli } d(a_{xyz}, g_{xyz}) < d_{th} \\ -1, & \text{jeśli } d(a_{xyz}, g_{xyz}) > d_{th} \end{cases} \tag{44}$$

gdzie $d(a_{xyz}, g_{xyz})$ to odległość euklidesowa pomiędzy aktualną pozycją TCP robota (a_x, a_y, a_z), a pozycją zadaną (g_x, g_y, g_z).

Funkcja *dense* była ciągła i brała pod uwagę odległość euklidesową pomiędzy aktualną pozycją TCP robota (a_x, a_y, a_z), a pozycją zadaną (g_x, g_y, g_z). Była ona obliczana według wzoru:

$$R_d = -\sqrt{(a_x - g_x)^2 + (a_y - g_y)^2 + (a_z - g_z)^2} \tag{45}$$

Trzecia funkcja nagrody jest rozszerzeniem drugiej. Dodatkowo brała ona pod uwagę trajektorię, po jakiej poruszał się TCP robota. Nagroda typu *dense trajectory* obliczana była z wykorzystaniem równania:

$$R_{dtr} = -\sqrt{(a_x - g_x)^2 + \left(a_y - g_y\right)^2 + (a_z - g_z)^2} - d_{tr} \cdot w_{dtr} \tag{46}$$

gdzie parametr d_{tr} w danym kroku czasowym, to odległość pomiędzy punktem leżącym na trajektorii, po której przemieszcza się TCP robota, a trajektorią wyznaczoną na podstawie pozycji startowej (s_x, s_y, s_z) i zadanej (g_x, g_y, g_z). Odległość d_{tr} obliczana jest według wzoru:

$$d_{tr} = \frac{\left|\overrightarrow{s_{xyz}g_{xyz}} \times \overrightarrow{s_{xyz}a_{xyz}}\right|}{\left|\overrightarrow{s_{xyz}g_{xyz}}\right|} \tag{47}$$

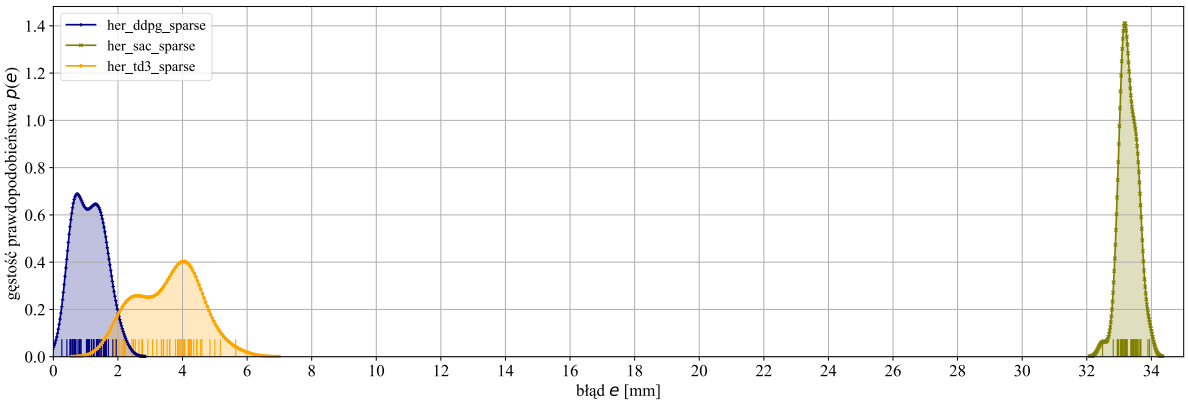
gdzie $\overrightarrow{s_{xyz}g_{xyz}}$ to wektor pomiędzy puntem początkowym s_{xyz}, a pozycją zadaną g_{xyz}, $\overrightarrow{s_{xyz}a_{xyz}}$ to wektor pomiędzy puntem początkowym s_{xyz}, a aktualną pozycją a_{xyz}. Parametr w_{dtr} jest współczynnikiem wagi, który opisuje jak bardzo odległość d_{tr} brana jest pod uwagę w trakcie obliczania nagrody R_{dtr}.

7.2 Wyniki badań symulacyjnych

Poniżej przedstawiono wyniki badań symulacyjnych badań pozycjonowania. W tabelach przedstawionych w tym rozdziale przedstawiono wyniki scharakteryzowane przez parametry

przedstawione w rozdziale 7.1. Pogrubioną czcionką oznaczono najlepsze rezultaty. W przypadku zastosowania nagrody typu *sparse* (Tabela 2), najmniejszy błąd e został osiągnięty przez kombinację algorytmów DDPG i HER. Algorytm został wytrenowany z progiem d_{th} równym 0.5mm, a uzyskany średni błąd e wyniósł 1.12mm. DDPG w połączeniu z HER osiągnął odległość d_{tr} równą 5.92 mm, co jest również najmniejszą wartością spośród wszystkich trenowanych algorytmów z nagrodą *sparse*. Na rysunku 41, w celach wizualizacyjnych została przedstawiona funkcja gęstości prawdopodobieństwa (*probability density function*, PDF) dla poszczególnych błędów e, uzyskanych w 50 epizodach ewaluacyjnych. Funkcja została oszacowana z wykorzystaniem jądrowego estymatora gęstości (*kernel density estimation*, KDE) [134]. Dodatkowo, na wykresie dodano wykres dywanowy, w którym każda pionowa linia (*spike*) reprezentuje jeden z 50 błędów e uzyskanych w epizodach ewaluacyjnych, dzięki temu zilustrowano dokładnie wartości błędów na osi liczbowej.

Badania wykazały, że algorytmy, które nie były w kombinacji z algorytmem HER, nie były w stanie nauczyć się, jak wykonać zadanie z żadnym z trzech progów odległości d_{th}. Agenci wytrenowani z wykorzystaniem algorytmów DDPG, SAC, TD3 nie sterowały robotem tak, aby TCP robota poruszał się w kierunku pozycji zadanej. W przypadku zastosowania nagrody typu *sparse*, która daje agentowi najmniej informacji o tym jaka była jakość akcji, widać zaletę zastosowania połączenia algorytmów DDPG, TD3 i SAC z HER. Zadanie pozycjonowania TCP robota okazało się zbyt trudne do nauczenia dla tych trzech algorytmów bez HER.



Rys. 41. Funkcja gęstości prawdopodobieństwa oraz wykres dywanowy dla badań symulacyjnych i nagrody typu sparse

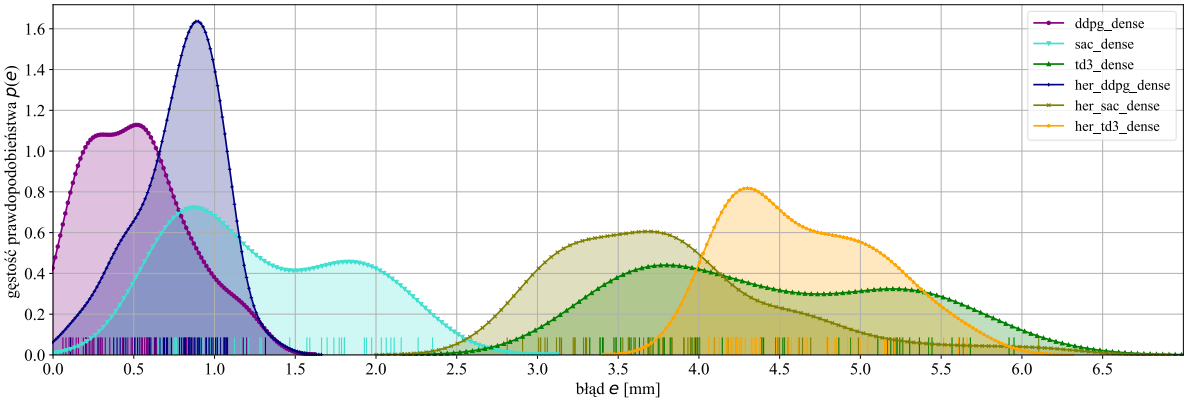
Tabela 2. Wyniki badań symulacyjnych pozycjonowania TCP robota w zadanym punkcie i nagrody typu sparse.

Algorytm	**śr. $e \pm \sigma_e$ [mm]**	**śr. $d_{tr} \pm \sigma_{dtr}$ [mm]**
HER+DDPG d_{th}= 0.5mm	**1.12±0.46**	**5.92±14.35**
HER+SAC d_{th}= 50mm	33.29±0.27	20.98±10.32

HER+TD3 d_{th}= 5mm	3.56±0.92	9.06±21.47
Algorytmy DDPG, SAC, TD3 nie były w stanie nauczyć się jak wykonać zadanie.		

Tabela 3. Wyniki badań symulacyjnych pozycjonowania TCP robota w zadanym punkcie i nagrody typu dense.

Algorytm	**śr. $e \pm \sigma_e$ [mm]**	**śr. $d_{tr} \pm \sigma_{dtr}$ [mm]**
DDPG d_{th}= 0.5mm	**0.51±0.30**	4.90±12.67
SAC d_{th}= 0.5mm	1.30±0.53	5.27±12.87
TD3 d_{th}= 0.5mm	4.43±0.79	6.90±12.63
HER+DDPG d_{th}= 0.5mm	0.78±0.25	**4.80±12.03**
HER+SAC d_{th}= 0.5mm	3.81±0.70	5.71±8.51
HER+TD3 d_{th}= 0.5mm	4.66±0.45	7.92±12.23

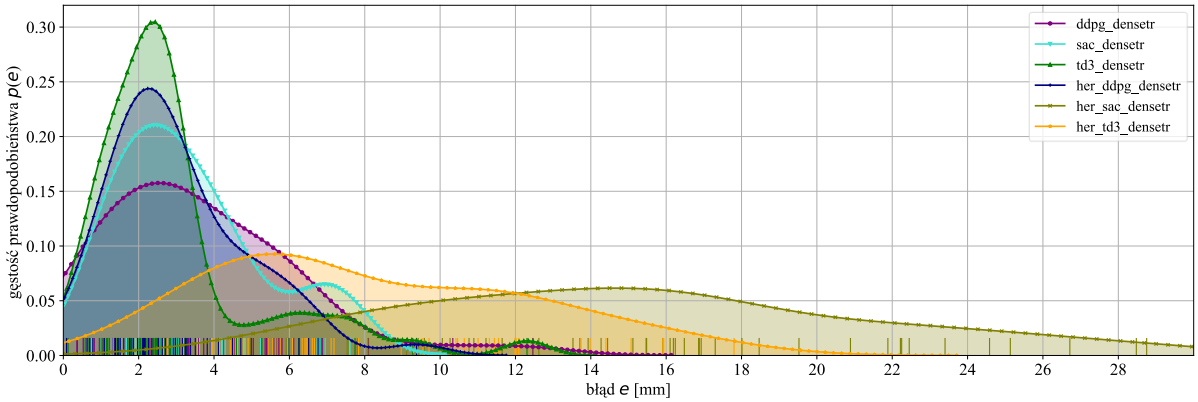


Rys. 42. Funkcja gęstości prawdopodobieństwa oraz wykres dywanowy dla badań symulacyjnych i nagrody typu dense

Najmniejszy błąd e dla nagrody typu *dense* uzyskano, dla algorytmu DDPG (Tabela 3). Błąd ten był równy 0.51mm. Najmniejszą odległość d_{tr} uzyskano dla kombinacji HER i DDPG. Dla tego połączenia błąd e był nieznacznie większy i wynosił 0.78mm. Oba algorytmy pozwoliły na osiągnięcie podobnych wyników. W przypadku zastosowania nagrody typu *dense*, wszystkie badane algorytmy pozycjonowały TCP robota z błędem e mniejszym niż 5mm. We wszystkich przypadkach średnia odległość d_{tr} była bardzo zbliżona, to znaczy w zakresie od 5mm do 8mm. Na rysunku 42 została przedstawiona funkcja gęstości prawdopodobieństwa i wykres dywanowy w przypadku zastosowania nagrody typu *dense*.

Tabela 4. Wyniki badań symulacyjnych pozycjonowania TCP robota w zadanym punkcie i nagrody typu dense trajectory.

Algorytm	**śr. $e \pm \sigma_e$ [mm]**	**śr. $d_{tr} \pm \sigma_{dtr}$ [mm]**
DDPG d_{th}= 0.5mm, w_{dtr} = 10	3.61±2.50	**0.79±0.63**
SAC d_{th}= 0.5mm, w_{dtr} = 10	3.38±1.94	2.03±1.53
TD3 d_{th}= 0.5mm, w_{dtr} = 10	**3.00±2.35**	2.15±1.48
HER+DDPG d_{th}= 0.5mm, w_{dtr} = 20	3.07±1.84	1.37±0.84
HER+SAC d_{th}= 0.5mm, w_{dtr} = 10	15.33±5.94	6.99±3.05
HER+TD3 d_{th}= 0.5mm w_{dtr}, = 10	8.22±4.03	4.86±2.58

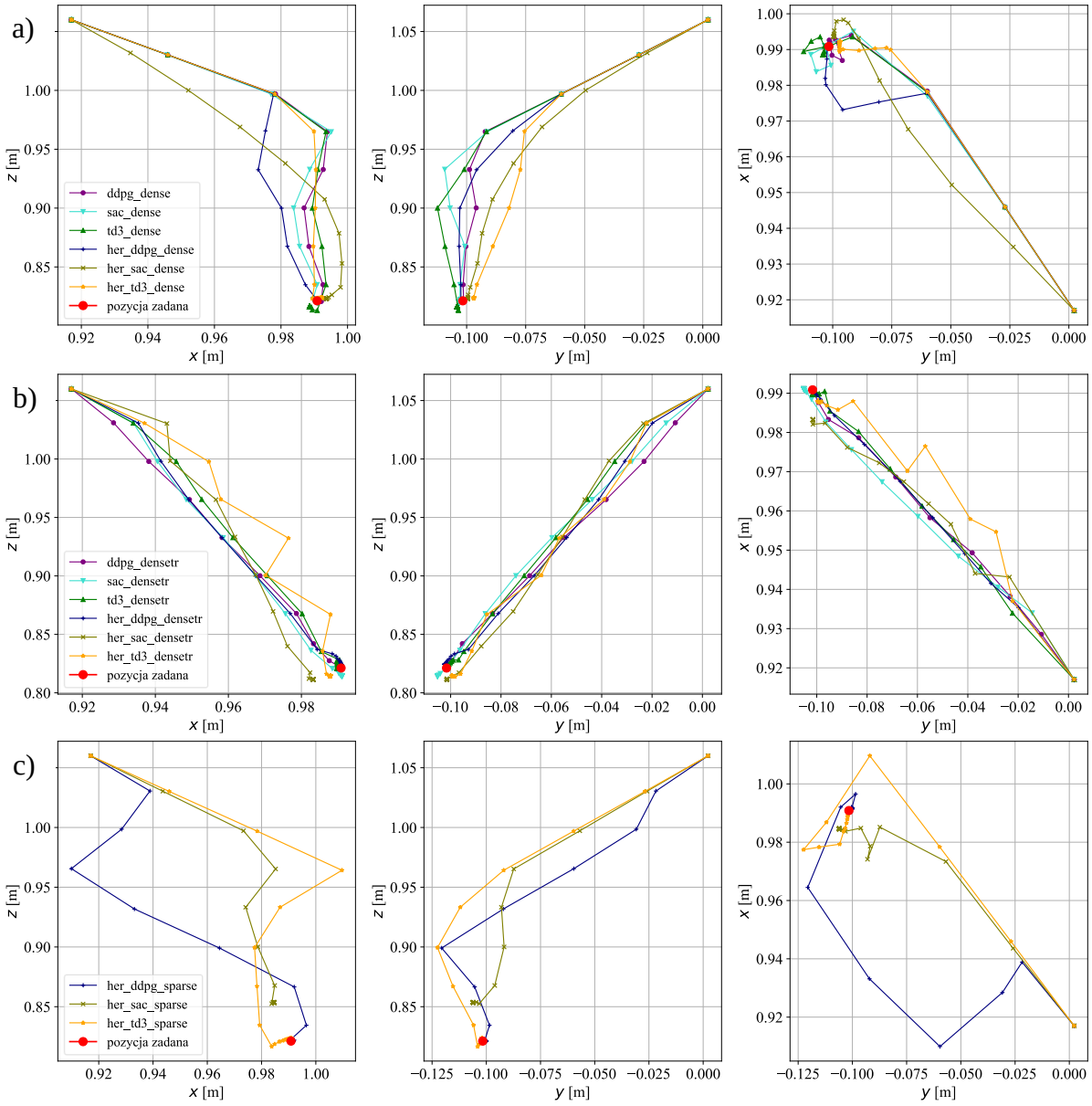


Rys. 43. Funkcja gęstości prawdopodobieństwa oraz wykres dywanowy dla badań symulacyjnych i nagrody typu dense trajectory

W przypadku zastosowania nagrody typu *dense trajectory* (Tabela 4) najmniejszy błąd e wynosił 3.0mm dla algorytmu TD3. Jednak biorąc pod uwagę również odchylenie standardowe σ_e, lepszy okazał się algorytm HER+DDPG. Różnica między błędami e tych dwóch algorytmów wynosiła tylko 0.07 mm, a odchylenie standardowe σ_e dla TD3 wynosiło 2.35mm, natomiast dla HER+DDPG 1.84mm. Najmniejszą odległość d_{tr} równą 0.79mm otrzymano przy zastosowaniu algorytmu DDPG. Algorytm HER+DDPG uzyskał drugi najlepszy rezultat. Wyniki w przypadku zastosowania nagrody typu *dense trajectory* uzyskano dla progu d_{th} równego 0.5 mm i współczynnika w_{dtr} równego 10, z wyjątkiem algorytmu HER+DDPG, w którym zastosowano w_{dtr} równy 20. Jak wykazały badania, pozwoliło to zmniejszyć średni błąd d_{tr}. Ta sama zmiana, czyli zwiększenie współczynnika w_{dtr} do 20 dla innych, testowanych algorytmów spowodowała, że algorytmy te miały dużo większy błąd e. Na rysunku 43 została przedstawiona funkcja gęstości prawdopodobieństwa i wykres dywanowy w przypadku zastosowania nagrody typu *dense trajectory*.

Na wykresach przedstawionych na Rys. 44 i Rys. 45 zobrazowane zostały trajektorie, po jakich poruszał się TCP robota, dla trzech różnych funkcji nagrody i punktów zadanych g_1 i

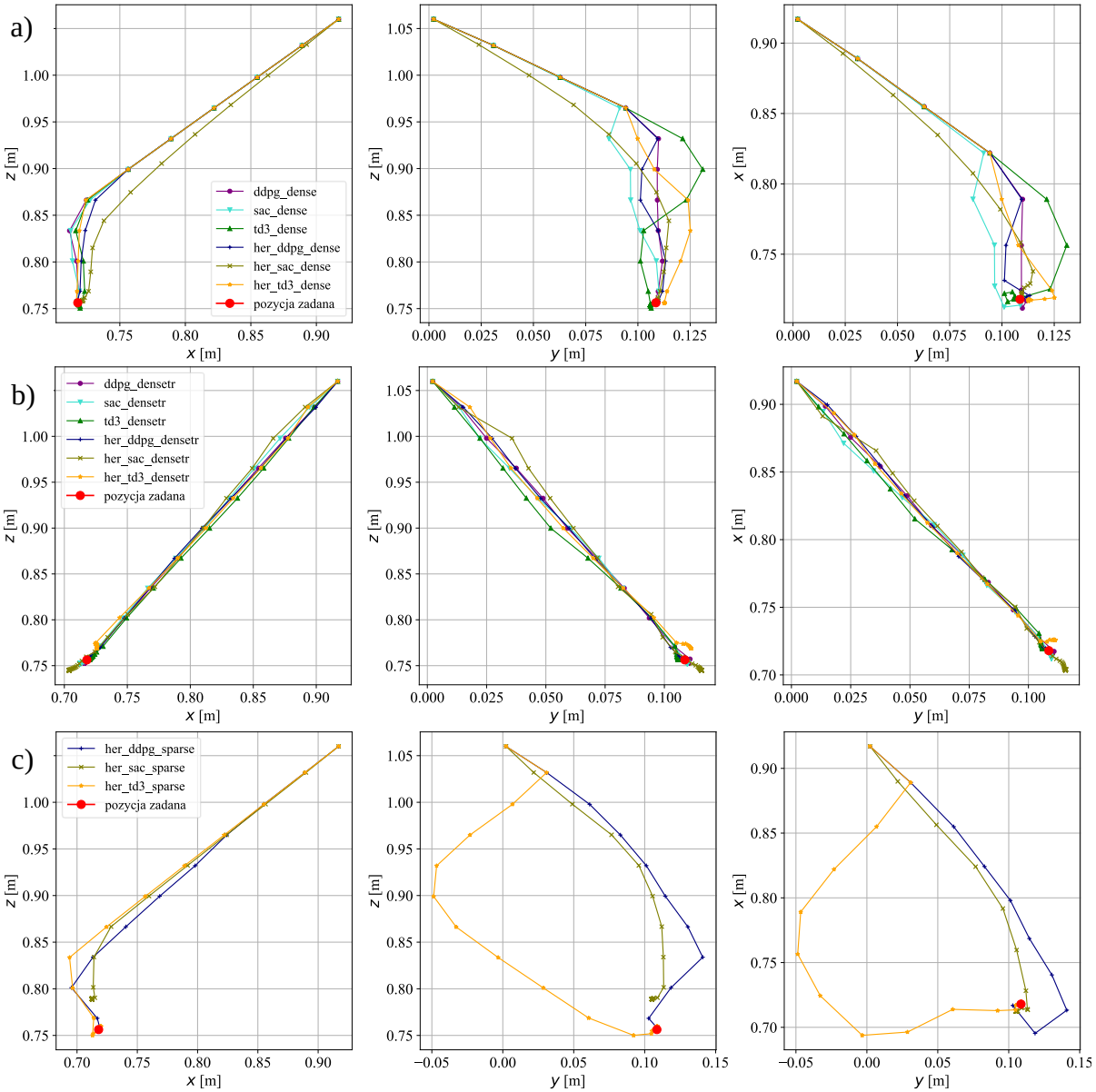
g_2, zaznaczonych na czerwono. Trzy wykresy umieszczone poziomo obok siebie przedstawiają trajektorie ruchu w trzech płaszczyznach, to jest ZX, ZY i XY, uzyskane z wykorzystaniem różnych algorytmów i tego samego rodzaju nagrody. Dla przykładu etykieta *ddpg_dense* (Rys. 44a) oznacza zastosowanie algorytmu DDPG i nagrody typu *dense*.



Rys. 44. Trajektorie po jakich poruszał się TCP manipulatora w badaniach symulacyjnych dla punktu g_1 i funkcji nagrody: a) – dense, b) – dense trajectory, c) – sparse

Biorąc pod uwagę tylko zarejestrowane trajektorie ruchu, można stwierdzić, że najlepsze uzyskano przy zastosowaniu nagrody typu *dense trajectory* (Rys. 44b i Rys. 45b). Dla tego typu nagrody, kształt trajektorii był w zasadzie dla wszystkich przebiegów zbliżony do linii prostej, która jest optymalną trajektorią. Najbardziej oddalony od linii prostej był przebieg uzyskany dla algorytmu HER+TD3 (żółta krzywa, Rys. 44b). Trajektoria ruchu TCP robota sterowana przez agenta wytrenowanego z wykorzystaniem tego algorytmu, charakteryzuje się największymi odchyleniami na boki. Najbardziej oddalone od prostej kształty trajektorii,

uzyskano w przypadku zastosowania nagrody typu *sparse* (Rys. 44c i Rys. 45c). Wszystkie algorytmy trenowane z tą nagrodą miały tendencję do podejmowania znacznie większych akcji, to znaczy większych przemieszczeń (skoków) w poszczególnych krokach, w porównaniu do przebiegów uzyskanych przy zastosowaniu dwóch pozostałych funkcji nagrody. Z tego powodu algorytmy z nagrodą typu *sparse* osiągały zadaną pozycją, po wykonaniu mniejszej liczby kroków, jednak w końcowej fazie ruchu występowały niepożądane gwałtowne skoki. Dodatkowo, trajektorie po jakich poruszał się TCP manipulatora były bardziej zakrzywione (oddalone od linii prostej) w porównaniu do trajektorii uzyskanych przez algorytmy z nagrodą typu *dense* i *dense trajectory*. Jest to szczególnie widoczne w przypadku algorytmu SAC połączonego z HER, dla którego średnia odległość d_{tr} wynosiła 20.98mm.



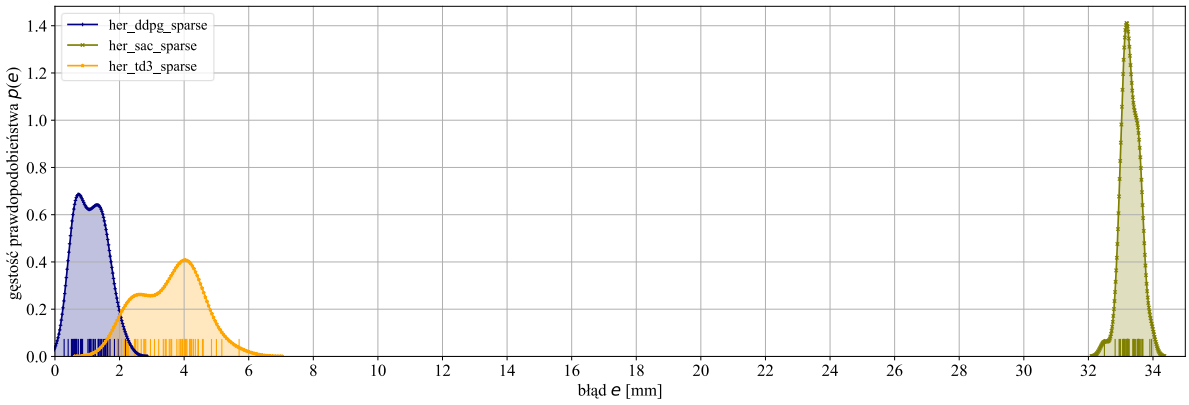
Rys. 45. Trajektorie po jakich poruszał się TCP manipulatora w badaniach symulacyjnych dla punktu g_2 i funkcji nagrody: a) – dense, b) – dense trajectory, c) – sparse

7.3 Wyniki badań doświadczalnych

Te same testy jak w przypadku badań symulacyjnych, wykonano również w warunkach laboratoryjnych z wykorzystaniem robota Mitsubishi przedstawionego na Rys. 30 i opisanego w rozdziale 6.1. Dane zamieszczone w tabelach poniżej porównano z danymi uzyskanymi w rozdziale 7.2 dotyczącym badań symulacyjnych. Porównując tabelę 2 i 5 można stwierdzić, że najlepsze wyniki uzyskano stosując algorytm HER+DDPG. Wyniki dla średniego błędu e i odległości d_{tr} uzyskane w badaniach doświadczalnych są bardzo zbliżone do wyników uzyskanych z symulacji, dla wszystkich trzech algorytmów. Na rysunku 46 została pokazana funkcja gęstości prawdopodobieństwa i wykres dywanowy dla nagrody *sparse*. Porównując go z wykresem przedstawionym na Rys. 41 uzyskanym w trakcie badań symulacyjnych, można zauważyć dużą zbieżność wyników.

Tabela 5. Wyniki badań doświadczalnych pozycjonowania TCP robota w zadanym punkcie i nagrody typu sparse.

Algorytm	**śr. $e \pm \sigma_e$ [mm]**	**śr. $d_{tr} \pm \sigma_{dtr}$ [mm]**
HER+DDPG d_{th}= 0.5mm	**1.11±0.46**	**6.03±14.38**
HER+SAC d_{th}= 50mm	33.30±0.27	21.05±10.30
HER+TD3 d_{th}= 5mm	3.57±0.91	9.07±20.89
Algorytmy DDPG, SAC, TD3 nie były w stanie nauczyć się jak wykonać zadanie.		

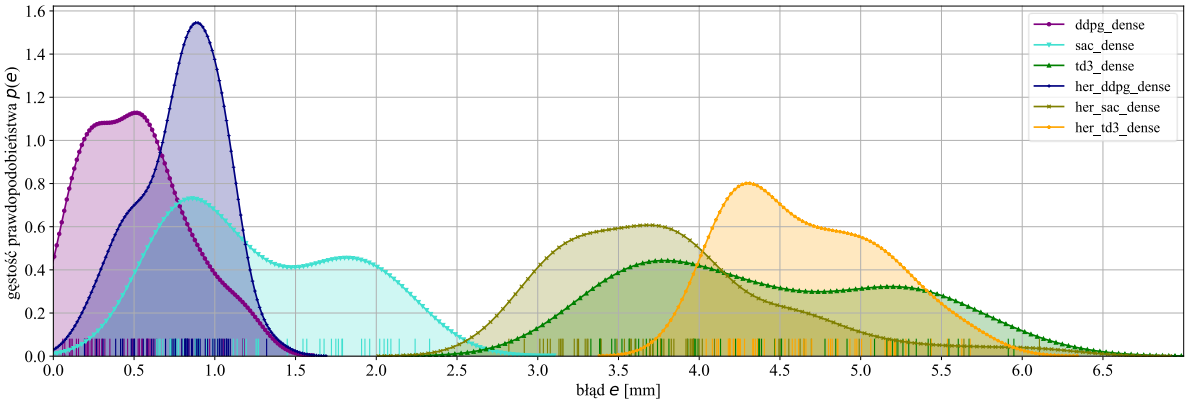


Rys. 46. Funkcja gęstości prawdopodobieństwa oraz wykres dywanowy dla badań doświadczalnych i nagrody typu sparse

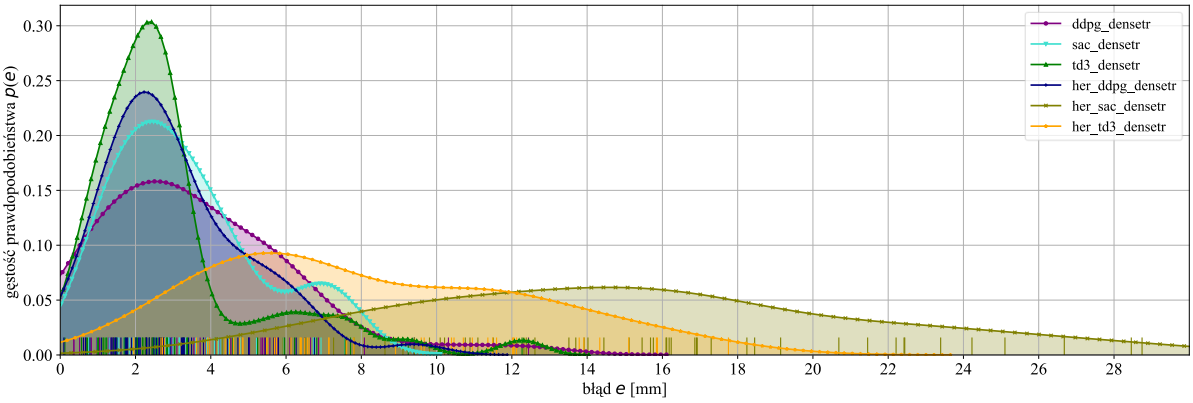
Tabela 6. Wyniki badań doświadczalnych pozycjonowania TCP robota w zadanym punkcie i nagrody typu dense

Algorytm	**śr. $e \pm \sigma_e$ [mm]**	**śr. $d_{tr} \pm \sigma_{dtr}$ [mm]**
DDPG d_{th}= 0.5mm	**0.51±0.30**	4.97±12.69

SAC d_{th}= 0.5mm	1.28±0.53	5.33±12.92
TD3 d_{th}= 0.5mm	4.42±0.79	6.89±12.63
HER+DDPG d_{th}= 0.5mm	0.79±0.25	**4.86±12.02**
HER+SAC d_{th}= 0.5mm	3.81±0.70	5.77±8.57
HER+TD3 d_{th}= 0.5mm	4.67±0.45	8.02±12.26



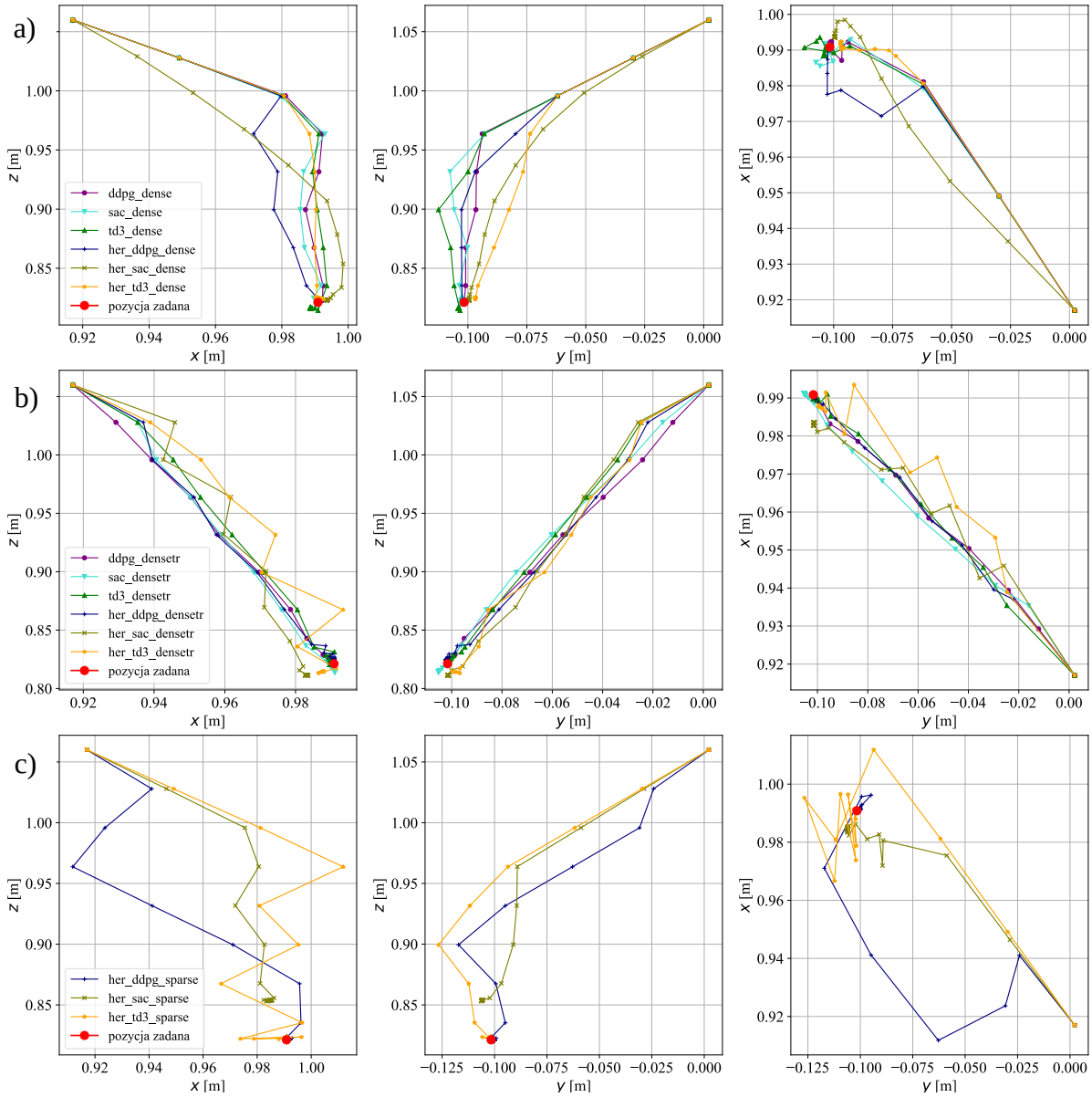
Rys. 47. Funkcja gęstości prawdopodobieństwa oraz wykres dywanowy dla badań doświadczalnych i nagrody typu dense



Rys. 48. Funkcja gęstości prawdopodobieństwa oraz wykres dywanowy dla badań doświadczalnych i nagrody typu dense trajectory

Taką samą zgodność pomiędzy wynikami symulacyjnymi i doświadczalnymi można zauważyć porównując dane z tabel 3 i 6, uzyskane z wykorzystaniem funkcji nagrody typu *dense*. Jeśli chodzi o błąd e to najlepszy wynik uzyskano stosując algorytm DDPG. W przypadku odległości d_{tr} najlepszy okazał się algorytm HER+DDPG. Na rysunku 47 została przedstawiona funkcja gęstości prawdopodobieństwa i wykres dywanowy dla nagrody *dense*. Wykres jest również zgodny z wykresem przedstawionym na Rys. 42.

Tak samo jak w przypadku nagród typu *sparse* i *dense*, wyniki badań doświadczalnych dla nagrody *dense trajectory* (Tabela 7) są bardzo zbliżone do wyników badań symulacyjnych (Tabela 4). Najlepsze rezultaty i najmniejszy błąd e otrzymano dla TD3, a dla odległości d_{tr},



Rys. 49. Trajektorie po jakich poruszał się TCP manipulatora w badaniach doświadczalnych dla punktu g_1 i funkcji nagrody: a) – dense, b) – dense trajectory, c) – sparse

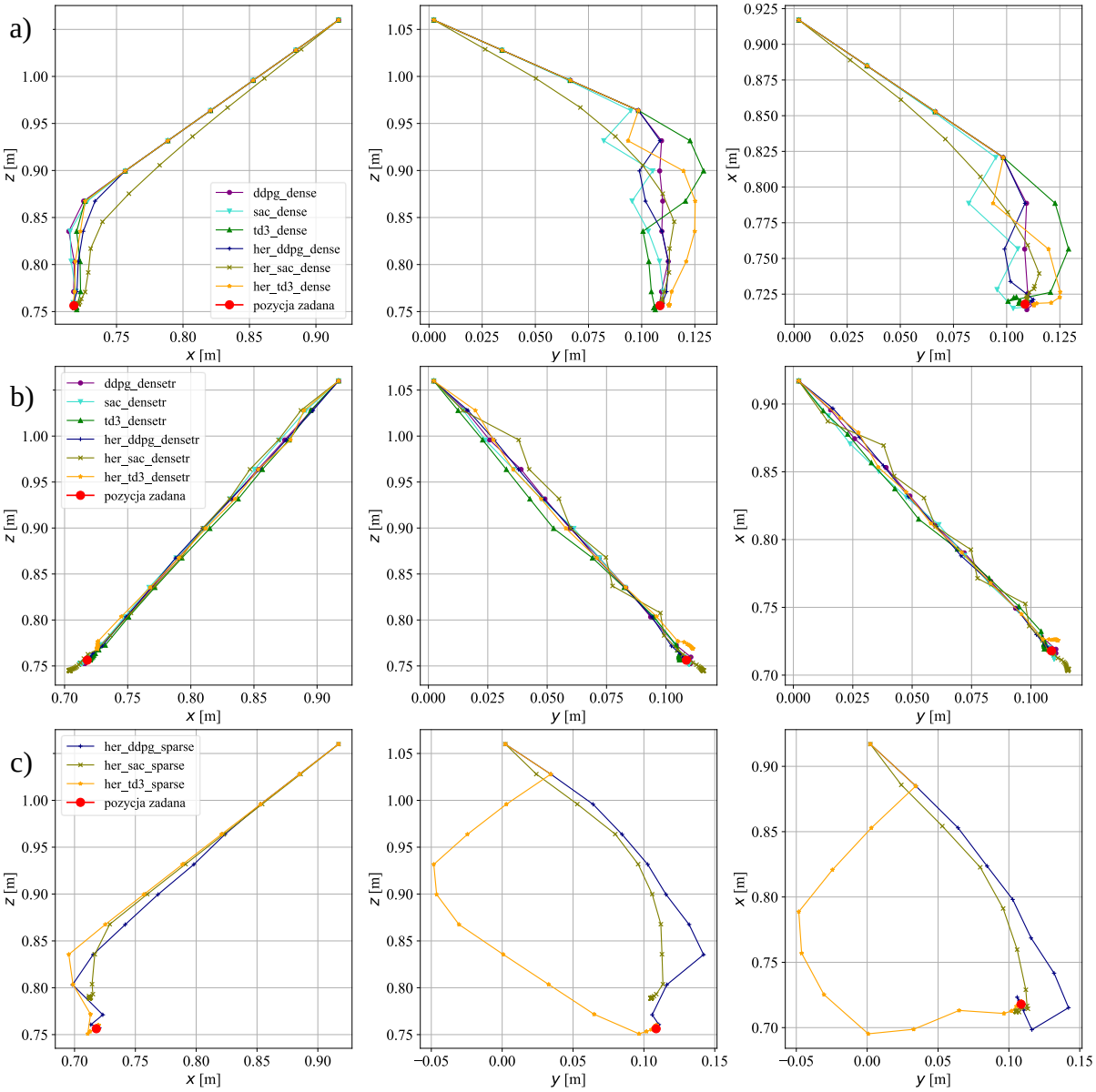
dla algorytmu DDPG. Na rysunku 48 przedstawiono funkcję gęstości prawdopodobieństwa i wykres dywanowy dla nagrody *dense trajectory*. Porównując ten wykres z przedstawionym na Rys. 43 można również stwierdzić dużą zbieżność wyników.

Na rysunkach 49 i 50 przedstawiono trajektorie, po jakich poruszał się TCP robota w trakcie badań doświadczalnych dla zadanych punktów g_1 i g_2.

Tabela 7. Wyniki badań doświadczalnych pozycjonowania TCP robota w zadanym punkcie i nagrody typu dense trajectory

Algorytm	**śr. $e \pm \sigma_e$ [mm]**	**śr. $d_{tr} \pm \sigma_{dtr}$ [mm]**
DDPG d_{th}= 0.5mm, w_{dtr} = 10	3.60±2.50	**0.81±0.66**

SAC d_{th}= 0.5mm, w_{dtr} = 10	3.37±1.93	2.02±1.59
TD3 d_{th}= 0.5mm, w_{dtr} = 10	**3.00±2.34**	2.17±1.57
HER+DDPG d_{th}= 0.5mm, w_{dtr} = 20	3.07±1.85	1.40±0.90
HER+SAC d_{th}= 0.5mm, w_{dtr} = 10	15.28±5.94	7.09±3.14
HER+TD3 d_{th}= 0.5mm w_{dtr}, = 10	8.22±4.02	4.94±2.65

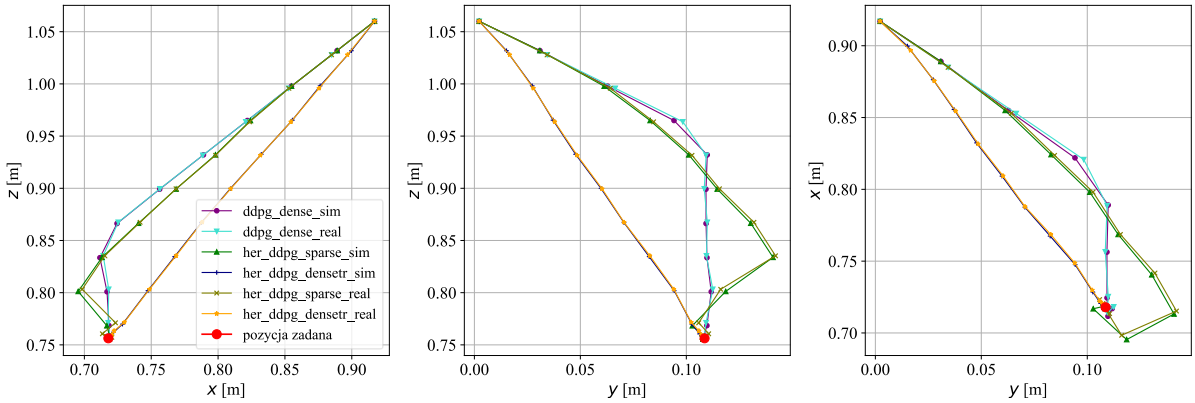


Rys. 50. Trajektorie po jakich poruszał się TCP manipulatora w badaniach doświadczalnych dla punktu g_2 i funkcji nagrody: a) – dense, b) – dense trajectory, c) – sparse

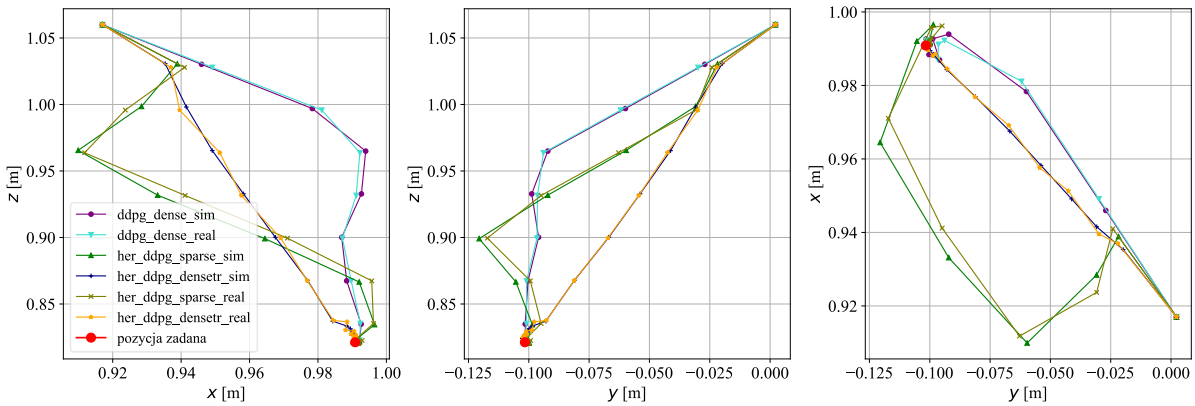
7.4 Podsumowanie

Przeprowadzone badania symulacyjne i doświadczalne pokazały, że możliwe jest zastosowanie algorytmów uczenia ze wzmocnieniem do wykonywania operacji przemieszczenia TCP robota z wybranego punktu do określonego punktu w przestrzeni roboczej. Dzięki temu robot uzyskał pewną samodzielność, charakteryzującą człowieka. Polega ona na tym, że wystarczy wskazać współrzędne punktu, a robot przemieści TCP do tego punktu. Może on być wskazany przez dowolne urządzenie, które wygeneruje współrzędne punktu. **Wykonane testy wybranych algorytmów uczenia ze wzmocnieniem oraz badania dokładności pozycjonowania TCP robota w zadanym punkcie, przedstawione w rozdziale 7 potwierdzają osiągniecie celu cząstkowego numer 4.**

W celu porównania badań trajektorii ruchu TCP robota dla badań symulacyjnych i doświadczalnych dla punktów g_1 i g_2, na Rys. 51 i Rys. 52 zostały przedstawione zarejestrowane krzywe. Na wykresach pokazano trajektorie dla trzech testowanych funkcji nagród, ale tylko dla tych algorytmów, które miały najmniejszy błąd e. Wyraźnie widać, że dla poszczególnych algorytmów i funkcji nagród, uzyskane w wyniku badań symulacyjnych i doświadczalnych trajektorie w zasadzie się pokrywają.



Rys. 51. Porównanie trajektorii dla algorytmów z najmniejszym błędem oraz trzech testowanych funkcji nagród dla badań symulacyjnych i doświadczalnych dla punktu g_1



Rys. 52. Porównanie trajektorii dla algorytmów z najmniejszym błędem oraz trzech testowanych funkcji nagród dla badań symulacyjnych i doświadczalnych dla punktu g_2

Badania wykazały, że najmniejszy błąd e osiągnęły następując algorytmy (w nawiasie wynik dla badań doświadczalnych):

- nagroda *sparse*, HER+DDPG (śr. błąd e=1.11mm),
- nagroda *dense*, DDPG (śr. błąd e=0.51mm) oraz HER+DDPG (śr. błąd e =0.79mm),
- nagroda *dense trajectory*, TD3 (śr. błąd e= 3.00mm) oraz HER+DDPG (śr. błąd e=3.07mm).

Dla odległości d_{tr} najlepsze wyniki osiągneły algorytmy:

- nagroda *sparse*, HER+DDPG (śr. odległość d_{tr}=6.03mm),
- nagroda *dense*, HER+DDPG (śr. odległość d_{tr}=4.86mm) oraz DDPG (śr. odległość d_{tr}=4.97mm),
- nagroda *dense trajectory*, DDPG (śr. odległość d_{tr}= 0.81mm) oraz HER+DDPG (śr. odległość d_{tr}=1.4mm).

Badania wykazały, że wyniki uzyskane w trakcie badań symulacyjnych i doświadczalnych są do siebie bardzo zbliżone. Potwierdza to możliwość implementacji wytrenowanego agenta w środowisku symulacyjnym na rzeczywistym robocie. Dla sześciu testowanych wariantów algorytmów i trzech funkcji nagród można stwierdzić, że najlepszą dokładność pozycjonowania i zarazem najmniejszy średni błąd e uzyskano przy zastosowaniu nagrody typu *dense* i algorytmów DDPG oraz DDPG+HER. Najmniejszą średnią odległość d_{tr} osiągnięto przy wykorzystaniu nagrody typu *dense trajectory* oraz tych samych algorytmów, co w przypadku nagrody *dense*. Na podstawie uzyskanych wyników badań można stwierdzić, że algorytmy z tą nagrodą uzyskały mniejszą odległość d_{tr} i większy błąd e, ponieważ ta nagroda została zaprojektowana w celu generowania trajektorii ruchu zbliżonej do linii prostej.

W przypadku zastosowania nagrody typu *sparse* niektóre algorytmy nie były w stanie wykonać zadania i sterować robotem w taki sposób, żeby osiągnął on zadany punkt w przestrzeni trójwymiarowej. Algorytmy uczenia ze wzmocnieniem są bardzo skomplikowane matematycznie, a fakt, że wykorzystują sztuczne sieci neuronowe, jeszcze bardziej utrudnia ocenę ich możliwości. Dlatego trudno jasno wskazać, dlaczego niektóre z nich nie są w stanie nauczyć się i wykonać określonych zadań, na przykład sterowania robotami. Algorytm, który dawał najlepsze rezultaty i przewyższał inne w jednym konkretnym zadaniu, takim jak pozycjonowanie ramienia robota, nie musi być najlepszy w innych zadaniach. Na świecie nadal trwają badania nad tymi algorytmami. Mają one na celu odkrycie mocnych i słabych stron każdego z nich w określonych zadaniach. Dopiero przeprowadzenie symulacji, a zwłaszcza testów eksperymentalnych, może potwierdzić, że dany algorytm może rzeczywiście rozwiązać dany problem.

Najlepsze rezultaty uzyskano dla algorytmu DDPG oraz kombinacji DDPG i HER. Z tego względu do dalszych badań dotyczących omijania przeszkód na zadanej trajektorii ruchu wytypowano kombinację algorytmów DDPG i HER. Połączenie tych dwóch algorytmów pozwoliło na uzyskanie jednego z najmniejszych błędów e ze wszystkich testowanych algorytmów. Dodatkowo różne algorytmy w połączeniu z HER wymagają znacznie mniejszego czasu treningu, co przekłada się na szybkość testów, badań i uruchomienia.

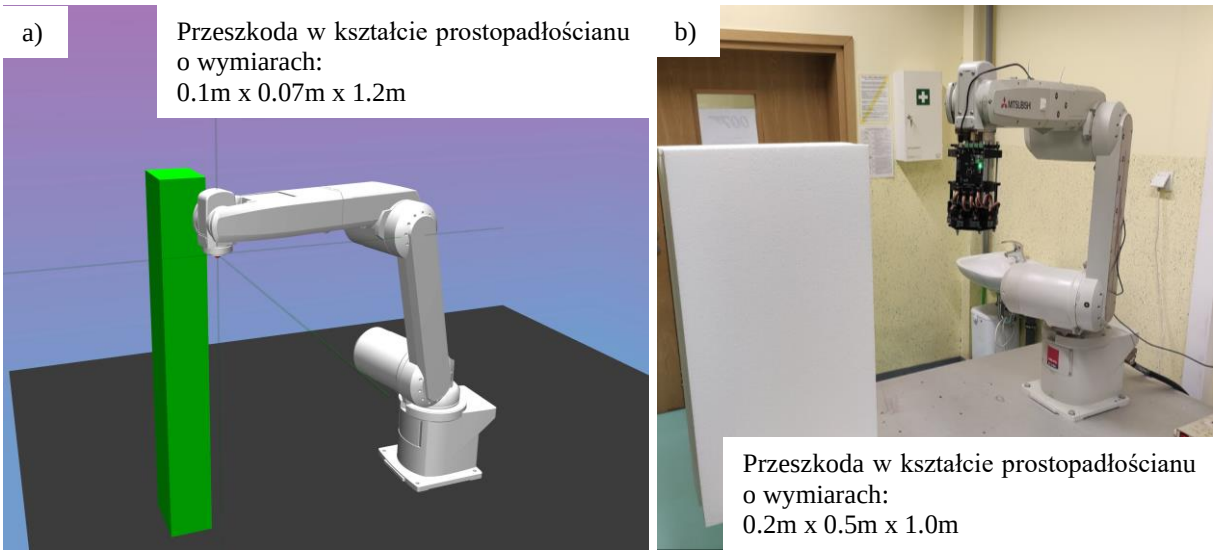
8 Badania algorytmu omijania przeszkód

8.1 Opis metodologii badań

W trakcie kolejnych badań zadanie agenta zostało rozszerzone tak, że miał on nauczyć się jak poruszać TCP manipulatora, żeby osiągnąć punkt zadany, ale dodatkowo miał być zdolny do omijania przeszkód, które mogły pojawić się w polu roboczym robota. W przypadku, gdy przeszkody nie było w polu roboczym, agent miał osiągnąć punkt zadany najszybciej jak to możliwe, czyli poruszając TCP robota po linii prostej wyznaczonej pomiędzy punktem startowym i zadanym. Do tego zadania została wytypowana kombinacja algorytmów DDPG i HER. Agenci byli trenowani przez $5 \cdot 10^4$ kroków w środowisku symulacyjnym. W celu optymalizacji wartości hiperparametrów algorytmów wykorzystano bibliotekę Optuna [133].

Proces treningowy był podzielony na epizody o maksymalnej liczbie kroków równej 50. Pozycja przeszkody była generowana z wykorzystaniem dystrybucji jednolitej z zakresu: oś X: od 0.85m do 1.05m, oś Y: od -0.1m do 0.1m, oś Z: $const$. Pozycje początkowe i zadane TCP robota były generowane z dystrybucji jednolitej z zakresu: oś X: od 0.6m do 1.05m, oś Y: od -0.45m do 0.45m, oś Z: od 0.6 do 0.9m. Jeśli w polu roboczym robota znajdowała się przeszkoda to pozycja początkowa i zadana nie mogły znajdować się wewnątrz przeszkody. Dodatkowo, punkt zadany był generowany w odległości co najmniej 0.115m od przeszkody. Odległość ta jest związana z funkcją nagrody opisaną w rozdziale 8.1.1.

W trakcie procesu treningowego oraz badań symulacyjnych, została zastosowana przeszkoda w kształcie prostopadłościanu o wymiarach 0.1m x 0.07m x 1.2m. W przypadku badań doświadczalnych zastosowano przeszkodę wykonaną z polistyrenu ekspandowanego (pot. styropianu) o kształcie prostopadłościanu i wymiarach 0.2m x 0.5m x 1.0m. Zastosowanie przeszkód o różnych rozmiarach było celowe i dało możliwość oceny czy agent jest w stanie omijać różne przeszkody (wyniki badań dla przeszkód o innych rozmiarach przedstawiono w rozdziale 9). Model robota w środowisku symulacyjnym oraz rzeczywisty robot razem z przeszkodami zostały pokazane odpowiednio na Rys. 53a i Rys. 53b. Z powodu



Rys. 53. Widok robota ze znajdującą się w polu roboczym przeszkodą w trakcie badań: a) – symulacyjnych, b) – doświadczalnych

ograniczeń zaprojektowanej głowicy w wykrywaniu przeszkód, ich wysokość, czyli jej rozmiar w osi Z był większy niż maksymalne położenie TCP manipulatora dla pozycji początkowej lub zadanej. Głowica była w stanie wykrywać przeszkody tylko w ograniczonej przestrzeni, w polu widzenia głowicy przedstawionym na Rys. 35.

W celu zapewnienia, że agent ominie przeszkodę znajdującą się w polu roboczym i następnie jak najszybciej osiągnie pozycję zadaną, zaproponowano własną technikę uczenia. Każdy epizod w trakcie treningu różnił się od siebie, ponieważ przeszkoda pojawiała się z określonym prawdopodobieństwem w polu roboczym i w różnym miejscu. Zastosowana technika uczenia została podsumowana w formie pseudokodu jako Algorytm 5.

Algorytm 5. Technika uczenia agenta do omijania przeszkód

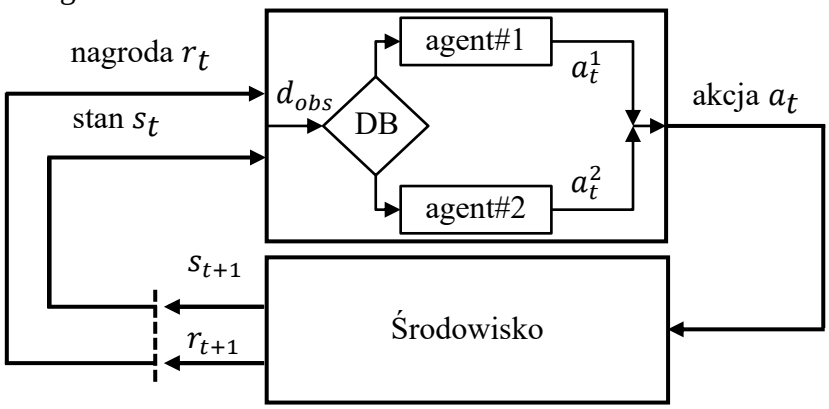
```
if random_uniform(0, 1) > OBSTACLE_APPEAR_PROBABILITY
  ukryj przeszkodę
  generuj pozycję zadaną
else
  generuj pozycję przeszkody
  generuj pozycję zadaną
  #nie wewnątrz przeszkody oraz strefy zabronionej blisko przeszkody
  if random_uniform(0, 1) < GOAL_POS_THE_SAME_AS_TCP_START_POS_PROBABILITY
    generuj pozycję startową po tej samej stronie przeszkody co pozycja zadana
    #współrzędne y mają te same znaki
  else
    generuj pozycję startową po przeciwnej stronie przeszkody co pozycja zadana
    #współrzędne y mają przeciwne znaki
```

W technice uczenia stosowano dwa główne współczynniki. Pierwszym z nich było prawdopodobieństwo występowania przeszkody p_{obsa} (`OBSTACLE_APPEAR_PROBABILITY`) w polu roboczym robota, które zostało ustawione na wartość 0.9. Drugi współczynnik to prawdopodobieństwo p_{gptp}, że pozycja początkowa TCP robota znajdowała się po tej samej stronie przeszkody co pozycja zadana (`GOAL_POS_THE_SAME_AS_TCP_START_POS_PROBABILITY`). Oznaczało to, że znaki współrzędnej dla osi Y pozycji początkowej i pozycji zadanej były takie same. Wartość tego współczynnika została ustawiona na 0.1.

Pierwszy współczynnik zapewniał taką generalizację środowiska treningowego, że minimalizował wpływ obecności przeszkody w polu roboczym na kształt trajektorii po jakiej poruszał się TCP robota. Prawdopodobieństwo p_{obsa} pozwalało na uniknięcie nadmiernego dopasowania się algorytmu do danych treningowych (*overfitting*). Badania wykazały, że jeśli współczynnik ten był zbyt mały, to agent nie był w stanie poprawnie wykonać manewru omijania przeszkody w każdym epizodzie. Można wysnuć wniosek, że w tej sytuacji agent zdobył zbyt małe doświadczenie w omijaniu przeszkód. Stwierdzono, że liczba kroków uczenia nie wpływała na zwiększenie liczby poprawnie wykonanych manewrów omijania przeszkód. Jeśli prawdopodobieństwo występowania przeszkody p_{obsa} było zbyt duże lub nawet równe 1 (we wszystkich epizodach treningowych pojawiała się przeszkoda), to w trakcie ewaluacji, TCP robota poruszał się po zakrzywionej trajektorii spowodowanej nieistniejącą przeszkodą w polu roboczym. Manipulacja wartością tego współczynnika nie dała możliwości całkowitego wyeliminowania zakrzywienia trajektorii.

Drugi współczynnik p_{gptp} został wprowadzony po to, aby agent wykonał swój główny cel, jakim było osiągniecie zadanej pozycji w przestrzeni trójwymiarowej. Przeprowadzone testy wykazały, że jeśli w trakcie uczenia prawdopodobieństwo to było za małe (mniejsze niż 0.1), to agent nie był w stanie osiągnąć pozycji docelowej. W takich przypadkach TCP robota nie poruszał się w kierunku celu, ponieważ będąc zbyt blisko przeszkody agent otrzymywał dużą karę, która uniemożliwiała mu eksplorację strefy roboczej i znalezienie celu, a co za tym idzie otrzymanie większej nagrody. Z kolei, gdy prawdopodobieństwo p_{gptp} było zbyt duże, agent nie mógł nauczyć się jak omijać przeszkody.

Testy wykazały, że agent wytrenowany do omijania przeszkód i jak najszybszego osiągnięcia pozycji zadanej ewaluowany w środowisku, w którym w polu roboczym nie było przeszkody, nie osiągnął celu tak szybko, jak agent, który został wytrenowany tylko po to, aby jak najszybciej osiągnąć pozycję zadaną. W przypadku pierwszego agenta występowało zakrzywienie trajektorii, po której poruszał się TCP robota. W celu rozwiązania tego problemu, zaproponowana została architektura składająca się z dwóch, działających równolegle agentów. Schemat takiego systemu przedstawiono na Rys. 54. Agent, który został wytrenowany do omijaniu przeszkód nazwano agentem#1, a agent wytrenowany do jak najszybszego osiągnięcia pozycji zadanej poruszając się po trajektorii zbliżonej do linii prostej nazywano agentem#2.



Rys. 54. Schemat systemu wieloagentowego zaprojekowanego do zadania omijania przeszkody i pozycjonowania TCP robota w zadanym punkcie

Obserwacjami agenta#1 i agenta#2 tworzącymi stan s w postaci 11-elementowego wektora były:

- aktualna pozycja TCP manipulatora (a_x, a_y, a_z) [m],
- prędkość liniowa TCP manipulatora (v_x, v_y, v_z) $[\frac{m}{s}]$,
- zadana pozycja (g_x, g_y, g_z) [m],
- odległość d_{obs} TCP robota od przeszkody [m],
- binarna informacja c, wskazująca czy robot dotknął przeszkody [prawda/fałsz].

Ostatnia obserwacja – c, była uwzględniona tylko w trakcie badań symulacyjnych, ponieważ bardzo trudno stwierdzić, że rzeczywisty robot dotknął przeszkodę w danym kroku czasowym czy też nie dotknął. Informacja ta w trakcie treningu pomagała agentowi nauczyć się kinematyki zastosowanego w badaniach robota. Akcją a agentów, była zmiana położenia TCP robota w metrach w przestrzeni trójwymiarowej $(\Delta x, \Delta y, \Delta z)$. Agenci działali

równolegle generując akcje a_t^1 i a_t^2, a dodatkowy blok decyzyjny (DB) decydował, który z nich powinien wykonać akcję w środowisku i przemieścić TCP robota. Blok DB to po prostu instrukcja warunkowa `if`, która na podstawie odległości między TCP robota, a przeszkodą decydowała, który agent powinien podjąć działanie. W przeprowadzonych badaniach, próg odległości d_{thobs} został ustawiony na wartość 0.25m. Oznacza to, że jeśli odległość między TCP robota, a przeszkodą była mniejsza niż 0.25m, to działanie podejmował agent#1, natomiast jeśli wartość ta była większa, to agent#2.

W trakcie ewaluacji poprawności działania systemu wieloagentowego wykonano badania symulacyjne i doświadczalne oraz porównano wyniki dla 50 wygenerowanych losowo par punktów startowego i zadanego. W przypadku badań symulacyjnych wygenerowano 50 losowych pozycji przeszkody. Pozycja przeszkody w przypadku badań doświadczalnych była ustalana ręcznie przez operatora. Dzięki temu w każdym epizodzie ewaluacyjnym była inna i nieznana. W ten sposób sprawdzono, czy system działał poprawnie w przypadku przeszkody usytuowanej w dowolnym, nieznanym wcześniej położeniu. W trakcie badań doświadczalnych ze względów bezpieczeństwa, maksymalna prędkość manipulatora została ograniczona do wartości $35\frac{mm}{s}$.

W tabelach w rozdziałach 8.2 i 8.3 przedstawiono odpowiednio wyniki badań symulacyjnych i doświadczalnych. Zamieszczono w nich następujące parametry: odległość (średni błąd e) TCP robota od pozycji zadanej, maksymalny błąd e, odchylenie standardowe σ_e błędu e oraz średnie błędy e_x, e_y, e_z dla poszczególnych osi. Błąd e tak jak w przypadku badań przedstawionych w rozdziale 7 był obliczany według wzoru:

$$e = \sqrt{\Delta x_T^2 + \Delta y_T^2 + \Delta z_T^2} = \sqrt{e_x^2 + e_y^2 + e_z^2} \tag{48}$$

gdzie $\Delta x_T, \Delta y_T, \Delta z_T$ są różnicami między współrzędnymi położenia TCP robota w ostatnim kroku danego epizodu, a pozycją zadaną.

8.1.1 Funkcje nagród

Agent#1 trenowany do omijania przeszkód, był szkolony z wykorzystaniem funkcji nagrody, która uwzględniała aktualną pozycję TCP robota (a_x, a_y, a_z) oraz pozycję zadaną (g_x, g_y, g_z). Nagroda była obliczana na podstawie odległości euklidesowej między tymi dwoma punktami w przestrzeni trójwymiarowej. Dodatkowym czynnikiem, który uwzględniał pozycję przeszkody w stosunku do robota był parametr r_{obs}. Wyznaczano go na podstawie logarytmicznej odległości d_{obs} TCP robota od przeszkody pomnożonej przez współczynnik wagi w_{obs}, który wynosił 0.5. Aby uniknąć problemów obliczeniowych w momencie, kiedy odległość d_{obs} była równa 0, do argumentu funkcji logarytmicznej dodano małą liczbę. Współczynnik r_{obs} był równy 0, jeśli odległość d_{obs} była większa niż ustalony próg d_{throbs}, który był równy 0.115m. Sumaryczna nagroda R_{dobs} była obliczana ze wzoru:

$$R_{dobs} = -\sqrt{(a_x - g_x)^2 + \left(a_y - g_y\right)^2 + (a_z - g_z)^2} + \tag{49}$$

$$+\begin{cases} \text{jeśli } d_{obs} > d_{throbs} & r_{obs} = 0 \\ \text{jeśli } d_{obs} \leq d_{throbs} & r_{obs} = \text{clip}(\log(d_{obs} + 0.00001) \cdot w_{obs}, -\infty, 0) \end{cases}$$

We wzorze nie zostały uwzględnione dwa przypadki. Pierwszym z nich była sytuacja, w której robot w danym kroku czasowym dotknął przeszkody, czyli obserwacja c z wektora stanu s była równa 1 (prawda). W takim przypadku agent otrzymywał dużą karę o wartości -10. Drugi przypadek to sytuacja, w której różnica między TCP robota, a pozycją zadaną była mniejsza niż 1mm. W takiej sytuacji agent otrzymywał bonus do nagrody o wartości +3. Agent#2 był trenowany z wykorzystaniem funkcji nagrody R_{dtr} danej wzorem (46) i przedstawionej w rozdziale 7.1.1. Współczynnik w_{dtr} miał wartość 20.

Połączenie funkcji nagrody R_{dtr} i R_{dobs} w jeden wzór oraz wykorzystanie tylko jednego agenta, nie dało tak dobrych rezultatów, jak wykorzystanie systemu z dwoma agentami z różnymi funkcjami nagród, pracującymi równolegle.

8.1.2 Modyfikacja algorytmu

W celu adaptacji przedstawionych w rozdziale 5.2 algorytmów do zadania, jakim było omijanie przeszkód, należało dokonać pewnych zmian w ich funkcjonowaniu. Zmianie uległ algorytm HER (Algorytm 4), a dokładnie sposób przechowywania danych w standardowym buforze D oraz buforze L tworzonym dla dodatkowych celów. Dodano następujące informacje, które były wpisywane do buforów: odległość d_{obs} oraz binarna informacja c. Dane te były wykorzystywane przy obliczaniu funkcji nagrody i optymalizacji parametrów algorytmu DDPG. Algorytm 6 pokazuje dokonane przez autora niniejszej pracy zmiany w algorytmie HER, które zaznaczono kolorem zielonym.

Algorytm 6. Modyfikacja algorytmu HER (Algorytm 4)

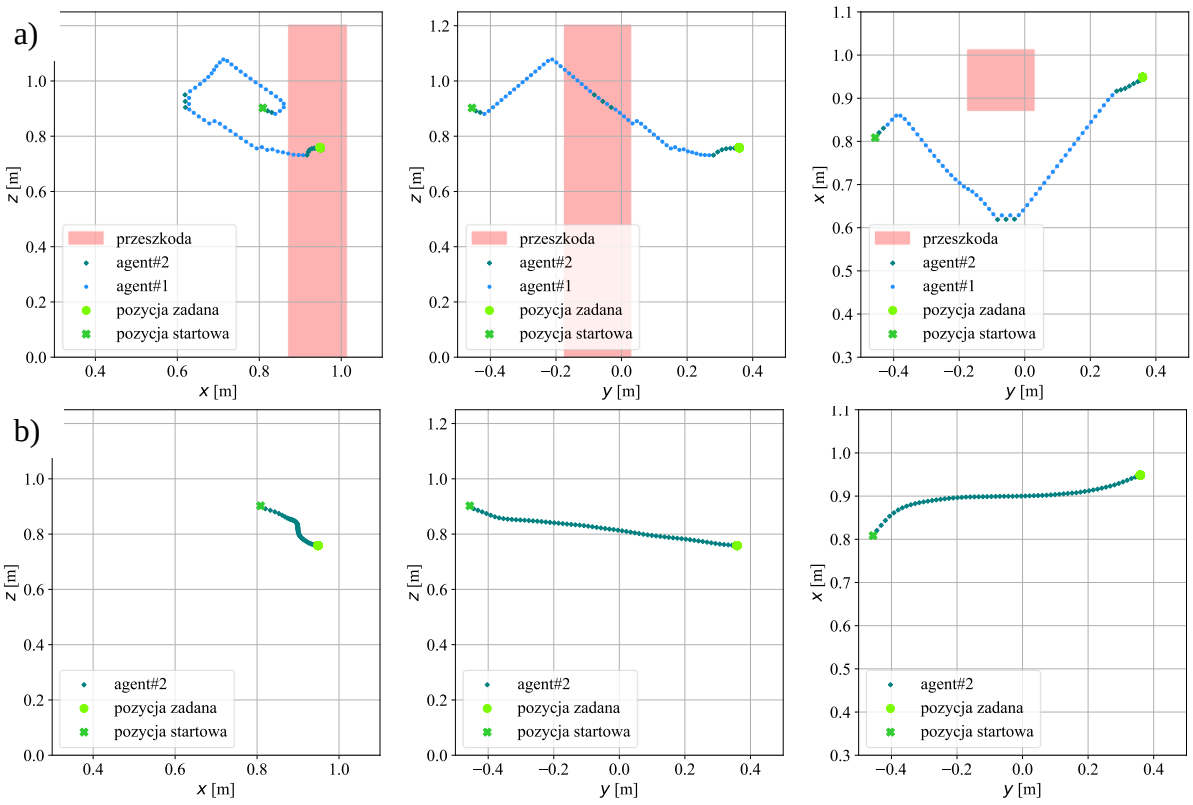
13:	**for** $t = 0, T - 1$ **do**	
14:	$r_t \coloneqq R(s_t, a_t, g)$	
15:	zapisz krotkę $(s_t \| \| g, a_t, r_t, s_{t+1} \| \| g, d_{obs} \| \| c)$ w buforze D	standardowy bufor z doświadczeniem
16:	próbkuj zestaw dodatkowych celów $G \coloneqq S$	(obecny epizod)
17:	**for** $g' \in G$ **do**	
18:	$r' \coloneqq R(s_t, a_t, g')$	
19:	zapisz krotkę $(s_t \| \| g', a_t, r', s_{t+1} \| \| g', d_{obs} \| \| c)$ w buforze L	HER

8.2 Wyniki badań symulacyjnych

Tabela 8 przedstawia wyniki badań symulacyjnych dla zadania, jakim było omijanie przeszkód znajdujących się w polu roboczym i pozycjonowanie TCP robota w zadanym punkcie. Średni błąd e wyniósł 3.84mm. Odchylenie standardowe σ_e dla błędu e wyniosło 1.83 mm. Maksymalny błąd e, który wystąpił podczas testowania, wynosił ponad 8mm i był to tylko jeden przypadek. Wyniki oscylowały głównie wokół średniego błędu e. Dla poszczególnych osi największy błąd wystąpił dla osi Y, który wyniósł prawie 2.6mm. Błędy dla pozostałych dwóch osi były poniżej 2mm.

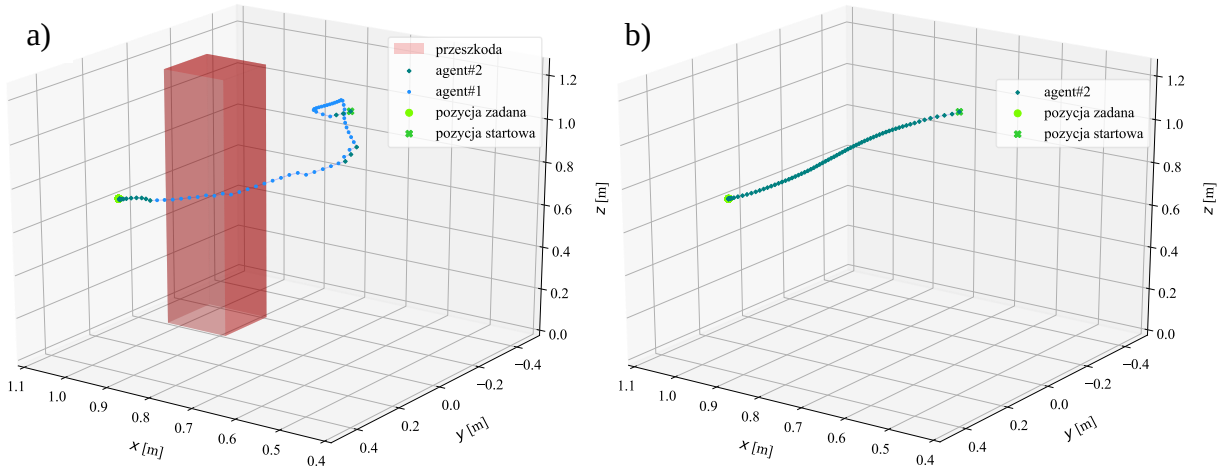
Tabela 8. Wyniki badań symulacyjnych dla zadania pozycjonowania TCP robota w zadanym punkcie oraz omijania przeszkody

śr. $e \pm \sigma_e$ [mm]	maks. e [mm]	śr. $e_x \pm \sigma_{e_x}$[mm]	śr. $e_y \pm \sigma_{e_y}$[mm]	śr. $e_z \pm \sigma_{e_z}$[mm]
3.84±1.83	8.39	1.67±1.32	2.58±1.73	1.43±1.12

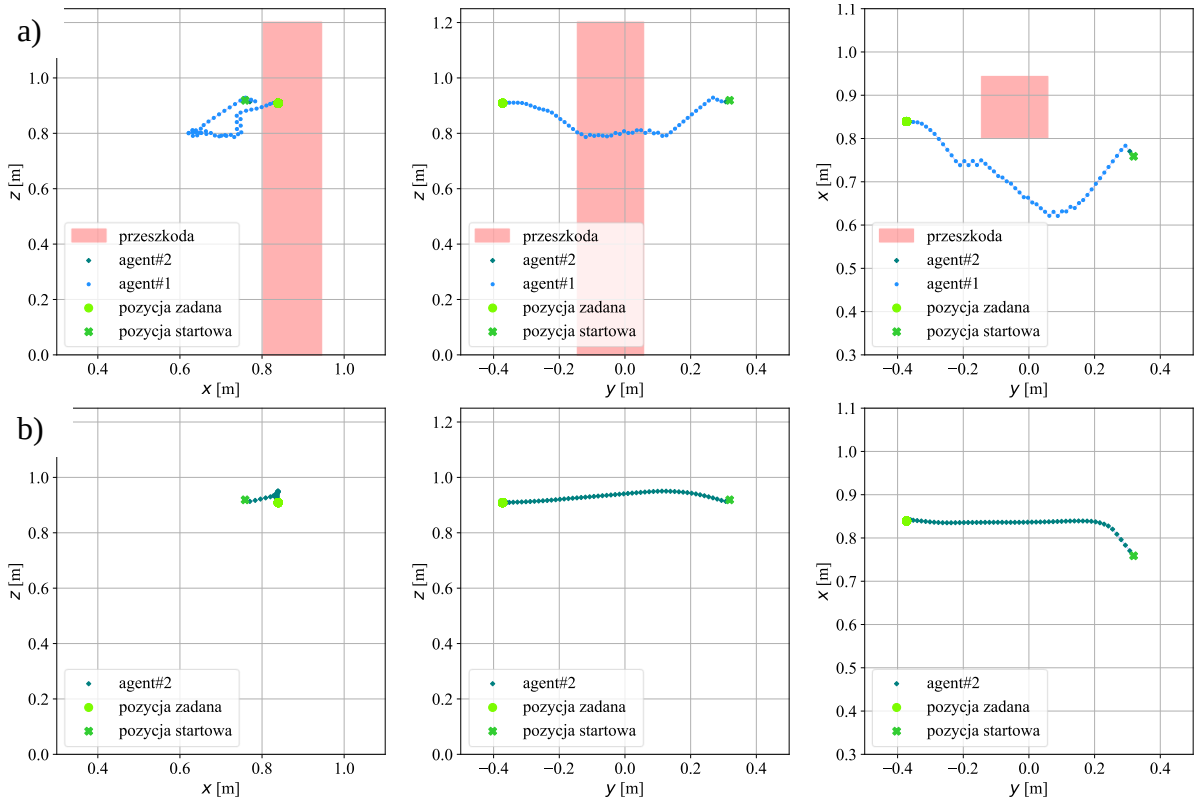


Rys. 55. Trajektorie po jakich poruszał się TCP manipulatora w badaniach symulacyjnych dla zadania omijania przeszkody i pozycjonowania TCP robota w punkcie zadanym g_3; wykres 2D; a) – przeszkoda znajdowała się w polu roboczym, b) – brak przeszkody w polu roboczym

Na rysunkach od 55 do 58 przedstawiono wykresy 2D i 3D obrazujące trajektorie ruchu TCP robota zarejestrowane podczas testów symulacyjnych. Na każdym rysunku znajdują się dwie grupy przebiegów: z przeszkodą w obszarze roboczym robota – a) i bez przeszkody – b). Różnią się one obecnością przeszkód, ale zostały wykonane dla tej samej pozycji początkowej TCP robota i tej samej pozycji zadanej. Akcje były generowane przez dwóch różnych agentów pracujących równolegle, dlatego trajektorie mają dwa kolory. Punkty zaznaczone na jasnoniebiesko, to akcje podjęte przez agenta#1 (etykieta *agent#1*), punkty zaznaczone na ciemnoniebiesko, to działania podjęte przez agenta#2 (etykieta *agent#2*). Trajektorie przedstawiają przypadki, w których pozycja startowa robota znajduje się po obu stronach przeszkody w osi Y. Oznacza to, że współrzędna y pozycji początkowej raz miała znak dodatni, a raz ujemny. Punkty zadane miały następujące współrzędne: g_{3x} = 0.94874m, g_{3y} = 0.35906m, g_{3z} = 0.75867m, g_{4x} = 0.83933m, g_{4y} = –0.37261m, g_{4z} = 0.90889m. W każdym epizodzie ewaluacyjnym przeszkoda pojawiała się w polu roboczym po kilku krokach

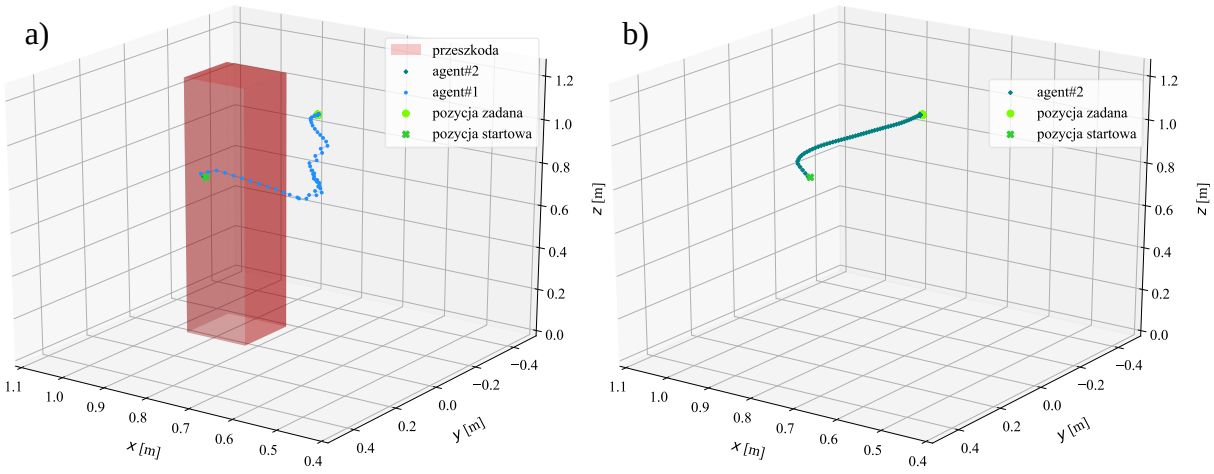


Rys. 56. Trajektorie po jakich poruszał się TCP manipulatora w badaniach symulacyjnych dla zadania omijania przeszkody i pozycjonowania TCP robota w punkcie zadanym g_3; wykres 3D; a) – przeszkoda znajdowała się w polu roboczym, b) – brak przeszkody w polu roboczym



Rys. 57. Trajektorie po jakich poruszał się TCP manipulatora w badaniach symulacyjnych dla zadania omijania przeszkody i pozycjonowania TCP robota w punkcie zadanym g_4; wykres 2D; a) – przeszkoda znajdowała się w polu roboczym, b) – brak przeszkody w polu roboczym

wykonanych przez agenta. Miało to zasymulować nagłe pojawienie się przeszkody w polu roboczym i przetestować reakcję algorytmu na taką zmianę w środowisku.



Rys. 58. Trajektorie po jakich poruszał się TCP manipulatora w badaniach symulacyjnych dla zadania omijania przeszkody i pozycjonowania TCP robota w punkcie zadanym g_4; wykres 3D; a) – przeszkoda znajdowała się w polu roboczym, b) – brak przeszkody w polu roboczym

Trajektorie przedstawione na Rys. 55 – 2D i Rys. 56 – 3D zostały wykonane dla tego samego punktu zadanego g_3. Można zauważyć, że jeśli przeszkoda znajdowała się w polu roboczym (Rys. 55a, Rys. 56a) to na początku ruchu akcje wykonywał agent#2. Po wykryciu przeszkody system przełącza agentów, a akcje wykonywał agent#1. Po kilku krokach podejmował on akcje w celu jej ominięcia. Na przebiegu widać, że w momencie, kiedy TCP robota poruszał się wzdłuż przeszkody, to niektóre akcje podejmował także agent#2. Działo się tak dlatego, że odległość d_{obs} między TCP robota, a przeszkodą, przekraczała wartość progową d_{thobs}, która wynosiła 0.25m. Po wykonaniu manewru ominięcia przeszkody, agent#1 zaczynał poruszać TCP robota w kierunku pozycji zadanej. Ostateczne pozycjonowanie TCP robota w punkcie zadanym zostało wykonane przez agenta#2. Analizując ruch agenta#1 w osi Z, widać, że nie poruszał on TCP po trajektorii zbliżonej do linii prostej. Można zauważyć ruch w górę w osi Z, mimo że punkt zadany znajdował się poniżej aktualnego położenia TCP robota. W przypadku, gdy w obszarze roboczym nie było przeszkody (Rys. 55b, Rys. 56b), to zadaniem agenta#2 było jak najszybsze przemieszczenie TCP robota w kierunku zadanej pozycji. Zarejestrowane trajektorie pokazują pewne odchylenia od linii prostej, zwłaszcza w osi X.

Przebiegi przedstawione na Rys. 57 – 2D i Rys. 58 – 3D zostały zarejestrowane dla tego samego punktu zadanego g_4. Podobnie jak w przypadku trajektorii dla punktu zadanego g_3, pierwsze akcje podejmował agent#2, po wykryciu przeszkody kontrolę przejmował agent#1 (Rys. 57a, Rys. 58a). Rysunki 57b i 58b przedstawiają sytuację, gdy w polu roboczym nie było przeszkody. Można zauważyć, że agent nie poruszał TCP robota idealnie po trajektorii zbliżonej do linii prostej, największe odchylenie od prostej można zobaczyć ponownie w osi X.

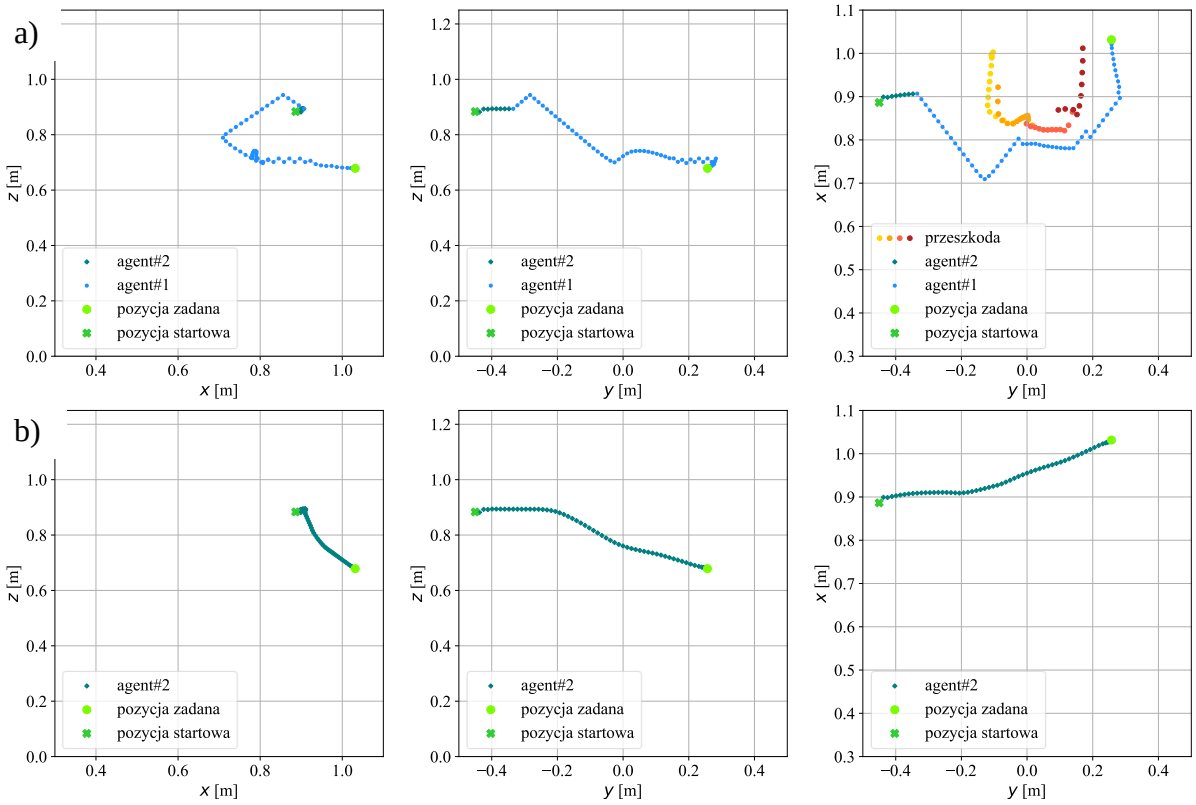
8.3 Wyniki badań doświadczalnych

Tabela 9 przedstawia wyniki badań doświadczalnych dla zadania, jakim było omijanie przeszkód znajdujących się w polu roboczym (Rys. 53b) i pozycjonowanie TCP robota w zadanym punkcie. Dane przedstawione w tej tabeli można porównać z wynikami uzyskanymi

w trakcie badań symulacyjnych, podanymi w Tabeli 8. Uzyskane wyniki są do siebie bardzo zbliżone. Dla badań doświadczalnych średni błąd e wyniósł 3.91mm, czyli różnił się o 0.07mm w porównaniu z wynikami uzyskanymi w testach symulacyjnych. Odchylenie standardowe σ_e dla błędu e było większe niż w przypadku badań symulacyjnych i wynosiło 2.51mm. Maksymalny błąd e również przekraczał 8mm. Dla poszczególnych osi największy błąd wystąpił dla osi Y, wyniósł on 2.69mm. Błędy dla pozostałych dwóch osi były poniżej 2mm.

Tabela 9. Wyniki badań doświadczalnych dla zadania pozycjonowania TCP robota w zadanym punkcie oraz omijania przeszkody

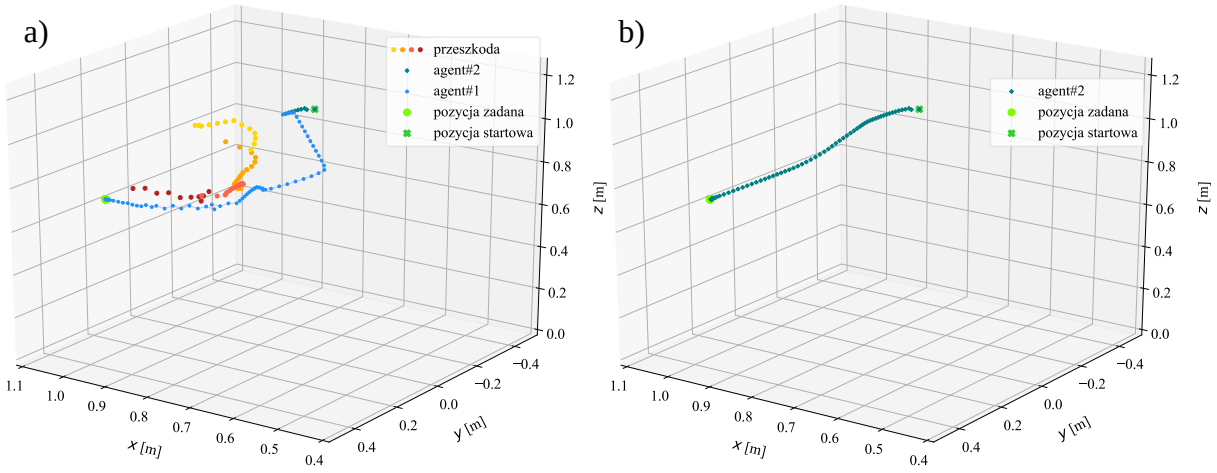
śr. $e \pm \sigma_e$ [mm]	maks. e [mm]	śr. $e_x \pm \sigma_{e_x}$[mm]	śr. $e_y \pm \sigma_{e_y}$[mm]	śr. $e_z \pm \sigma_{e_z}$[mm]
3.91±2.51	8.74	1.93±1.41	2.69±2.20	1.48±1.25



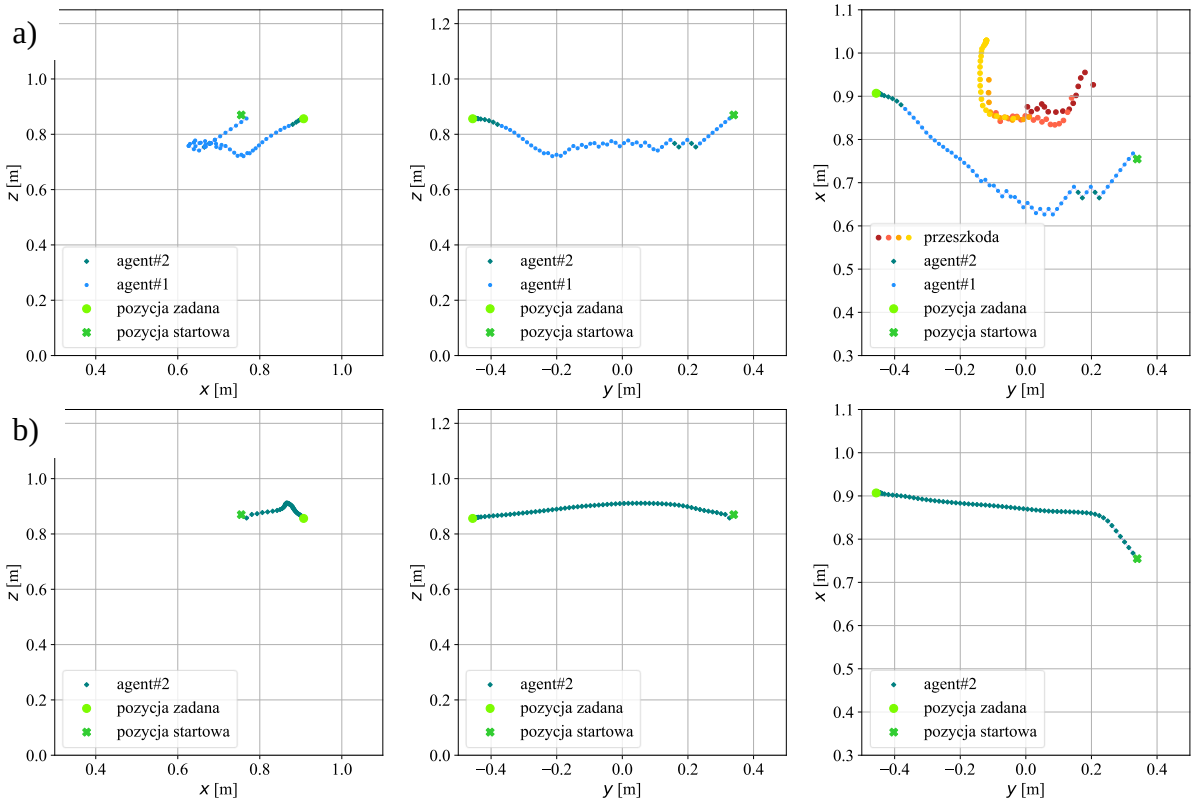
Rys. 59. Trajektorie po jakich poruszał się TCP manipulatora w badaniach doświadczalnych dla zadania omijania przeszkody i pozycjonowania TCP robota w punkcie zadanym g_5; wykres 2D; a) – przeszkoda znajdowała się w polu roboczym, b) – brak przeszkody w polu roboczym

Na rysunkach od 59 do 62 przedstawiono wykresy 2D i 3D obrazujące trajektorie ruchu TCP robota, uzyskane w trakcie testów w warunkach laboratoryjnych. Kolory punktów na trajektorii dla agenta#1 i agenta#2 są takie same jak na przebiegach pokazanych w rozdziale 8.2, dotyczącym badań symulacyjnych. Zaprojektowana głowica nie wykrywała przeszkody we wszystkich trzech płaszczyznach, dlatego też na zaprezentowanych wykresach na Rys. 59a

i Rys. 61a, przeszkoda została pokazana tylko na płaszczyźnie *XY*. Na rysunkach przeszkoda jest zaznaczona w formie kolorowych punktów, obrazujących dane o odległości z poszczególnych czujników znajdujących się na głowicy. Każdy czujnik został oznaczony



Rys. 60. Trajektorie po jakich poruszał się TCP manipulatora w badaniach doświadczalnych dla zadania omijania przeszkody i pozycjonowania TCP robota w punkcie zadanym g_5; wykres 3D; a) – przeszkoda znajdowała się w polu roboczym, b) – brak przeszkody w polu roboczym

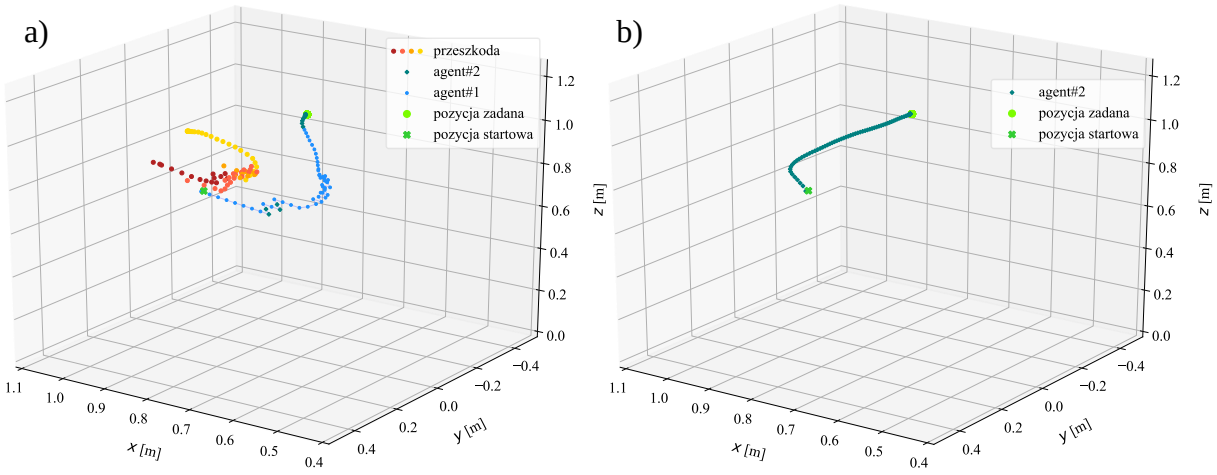


Rys. 61. Trajektorie po jakich poruszał się TCP manipulatora w badaniach doświadczalnych dla zadania omijania przeszkody i pozycjonowania TCP robota w punkcie zadanym g_6; wykres 2D; a) – przeszkoda znajdowała się w polu roboczym, b) – brak przeszkody w polu roboczym

innym kolorem, dzięki czemu widać kształt przeszkody rejestrowany w różnych krokach czasowych w danym epizodzie. Tak jak w przypadku badań symulacyjnych, przeszkoda była umieszczana w polu roboczym, żeby ocenić poprawność działania algorytmu. Punkty zadane miały następujące współrzędne: g_{5x} = 1.03153m, g_{5y} = 0.25699m, g_{5z} = 0.67847m, g_{6x} = 0.90690m, g_{6y} = −0.45613m, g_{6z} = 0.85618m.

Trajektorie przedstawione na Rys. 59 – 2D i Rys. 60 – 3D zostały wykonane dla tego samego punktu zadanego g_5. Sposób działania agentów był bardzo zbliżony do tego, jaki zarejestrowano w trakcie badań symulacyjnych. Jak widać na Rys. 59a i Rys. 60a, na początku epizodu akcje wykonywał agent#2, ponieważ przeszkoda została wstawiona w pole robocze robota w trakcie trwania epizodu. Agent#1 poruszał TCP robota wzdłuż przeszkody i po jej całkowitym ominięciu przemieszczał go do punktu zadanego. Agent#1 wykonywał wszystkie akcje do końca epizodu, ponieważ punkt zadany znajdował się zbyt blisko przeszkody. Gdy przeszkoda nie znajdowała się w polu roboczym (Rys. 59b i Rys. 60b), to na przebiegach widać, że agent#2 zarówno dla osi X, jak i Z miał duże odchylenie od trajektorii zbliżonej do linii prostej.

Na przebiegach zaprezentowanych na Rys. 61 – 2D i Rys. 62 – 3D pokazano trajektorie ruchu uzyskane dla punktu g_6. Analizując je w momencie, gdy przeszkoda znajdowała się w polu roboczym (Rys. 61a, Rys. 62a), można zauważyć, że podczas manewru jej omijania pojawiły się nieregularne ruchy w kształcie trójkątów. Po poprawnym ominięciu przeszkody, agent#1 przemieszczał TCP robota w kierunku punktu zadanego. W przypadku, gdy w polu roboczym nie było przeszkody (Rys. 61b, Rys. 62b), to tak jak poprzednio, wystąpiło odchylenie od linii prostej w osi X.



Rys. 62. Trajektorie po jakich poruszał się TCP manipulatora w badaniach doświadczalnych dla zadania omijania przeszkody i pozycjonowania TCP robota w punkcie zadanym g_6; wykres 3D; a) – przeszkoda znajdowała się w polu roboczym, b) – brak przeszkody w polu roboczym

8.4 Podsumowanie

Zastosowana w badaniach kombinacja algorytmów HER i DDPG oraz architektura z dwoma agentami pracującymi równolegle, umożliwiła zbudowanie systemu do omijania przeszkód przez ramię robota oraz do pozycjonowania jego TCP w zadanym punkcie w

przestrzeni trójwymiarowej. Potwierdziły to zarówno badania doświadczalne oraz symulacyjne. Średni błąd pozycjonowania e w przypadku testów doświadczalnych wyniósł 3.91mm, maksymalny błąd e to około 8mm. Średni błąd e pozycjonowania TCP robota w zadanym punkcie, uzyskany w trakcie badań dotyczących omijania przeszkód dla kombinacji algorytmów HER i DDPG był większy niż ten przedstawiony w rozdziale 7.3 (średni błąd e=3.07mm). Niektóre punkty zadane znajdowały się na tyle blisko przeszkody, że agent#1 odpowiedzialny za omijanie przeszkód wykonywał wszystkie akcje do końca trwania epizodu i pozycjonował TCP robota w zadanym punkcie. W takich przypadkach występował nieznacznie większy błąd pozycjonowania e TCP robota, w porównaniu do sytuacji, gdy akcje wykonywał agent#2. Z tego też względu średni błąd e dla wszystkich 50 epizodów ewaluacyjnych wynosił 3.91mm.

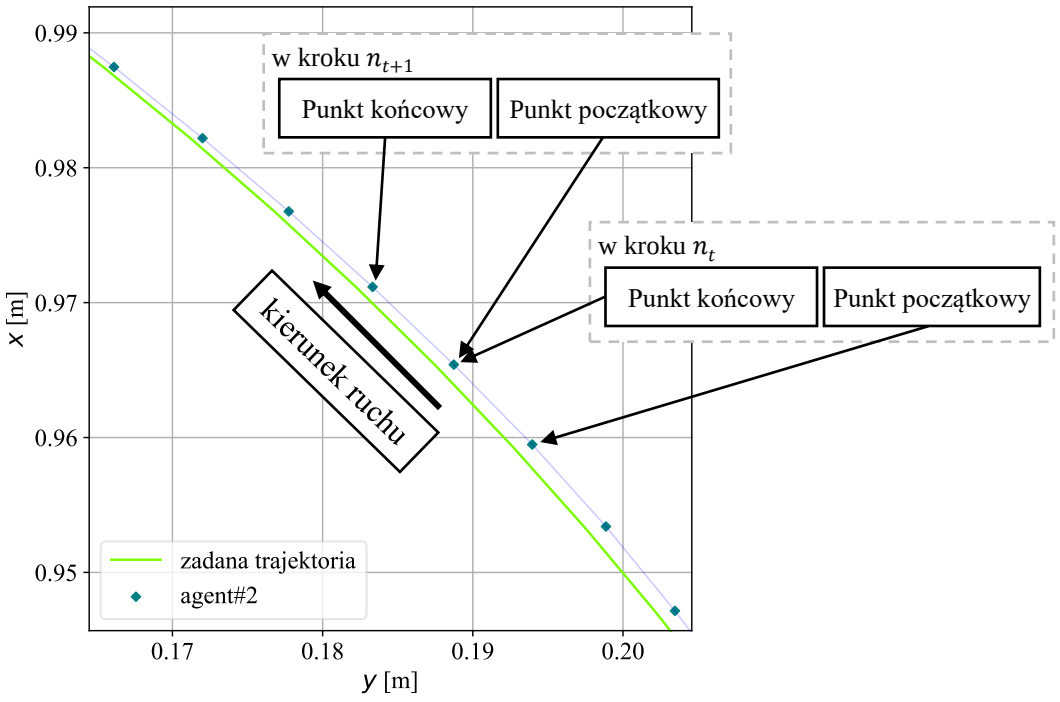
Dzięki wykorzystaniu algorytmów ze wzmocnieniem HER oraz DDPG robot był zdolny do podejmowania w pewnym stopniu autonomicznych decyzji dotyczących omijania przeszkód. Robot sterowany przez zaproponowany przez autora niniejszej pracy system, bezkolizyjnie omijał przeszkody dla wszystkich 50 epizodów ewaluacyjnych, które posłużyły do analizy wyników. Łącznie w trakcie badań doświadczalnych przeprowadzonych w laboratorium robot wykonał ponad 120 bezkolizyjnych manewrów omijania przeszkód znajdujących się w jego polu roboczym. **Opracowanie i przetestowanie systemu sterującego pracą robota przemysłowego, pozwalającego na omijanie przeszkód znajdujących się w polu roboczym, potwierdziło osiągniecie celu cząstkowego numer 5.**

9 Badania algorytmu omijania przeszkód na zadanej trajektorii ruchu

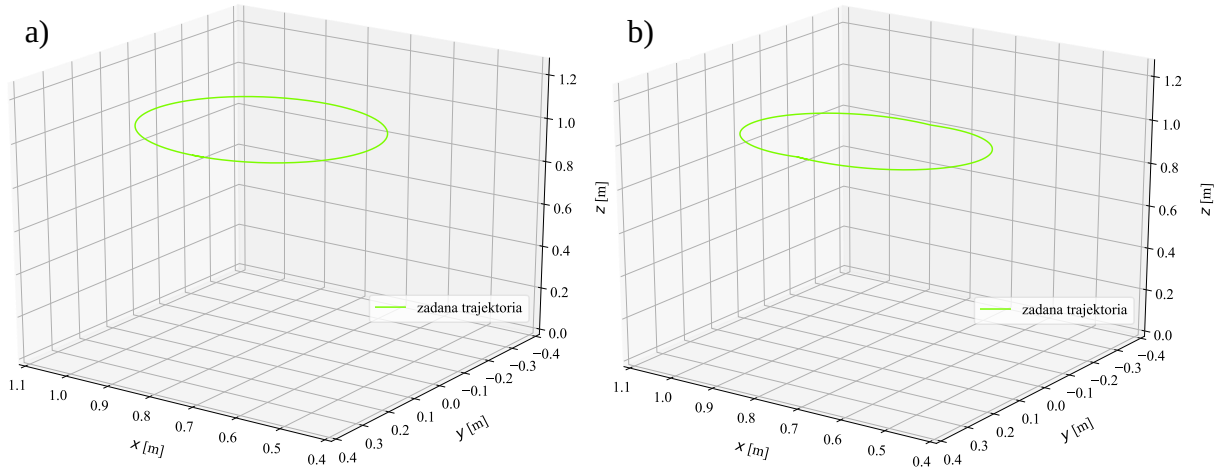
9.1 Opis metodologii badań

Badania i testy wykonane w tym rozdziale zostały podzielone na dwie części, które są związane z dwoma zadaniami, jakie miał do wykonania system sterujący pracą robota. Pierwsza część badań dotyczyła przemieszczania TCP robota po zadanej trajektorii, stosując do sterowania algorytm uczenia ze wzmocnieniem. W ocenie wyników brano pod uwagę uchyb pomiędzy trajektorią zadaną, a trajektorią po jakiej poruszał się TCP robota. Uchyb wyznaczany był na podstawie odległości euklidesowej, pomiędzy pozycją TCP robota, a punktem leżącym na zadanej trajektorii ruchu. Druga część badań była związana z omijaniem przeszkód na zadanej trajektorii ruchu. Testowano różne przypadki związane z kształtem przeszkody, jej położeniem, liczbą przeszkód oraz kształtem zadanej trajektorii ruchu. Konkretne scenariusze zostały przedstawione w rozdziałach związanych z badaniami. Oceniono, czy algorytm był w stanie bezkolizyjnie ominąć przeszkodę, a następnie po jej ominięciu, dalej podążać po zadanej trajektorii ruchu.

Do badań wykonywanych w tym rozdziale nie przeprowadzono dodatkowego uczenia agentów. Zastosowano wytrenowanych agentów oraz system wieloagentowy opisany w rozdziale 8.1 i przedstawiony na Rys. 54. W trakcie badań dotyczących omijania przeszkód na zadanej trajektorii ruchu, agent#1 był odpowiedzialny za omijanie przeszkód, natomiast agent#2 za sterowanie ruchem po zadanej trajektorii. Ponieważ algorytmy uczenia ze wzmocnieniem pracują w trybie dyskretnym, czyli w krokach czasowych, to zadana trajektoria została podzielona na odcinki, tak jak to pokazano na Rys. 63.



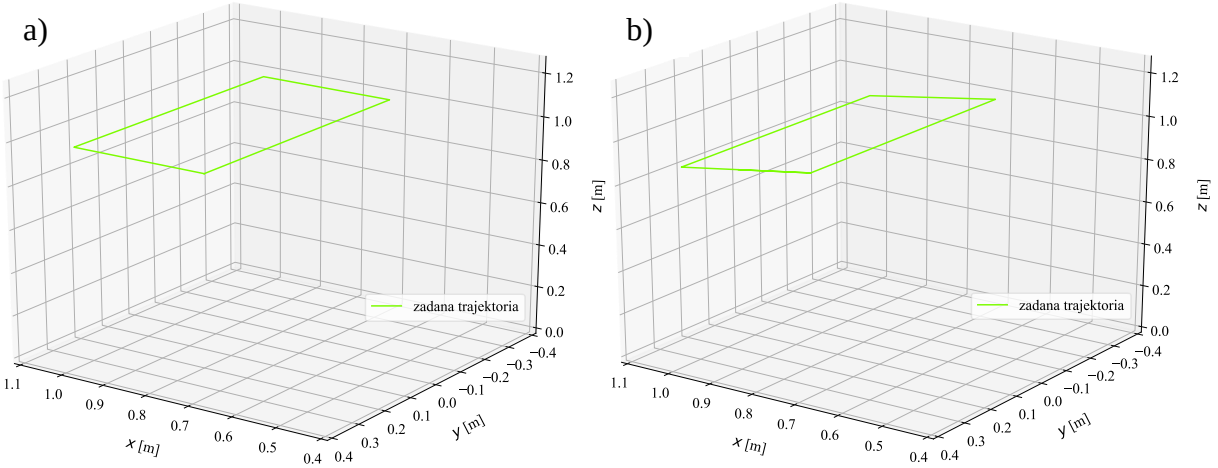
Rys. 63. Przykład dyskretyzacji zadanej trajektorii ruchu



r=0.25m
x_0=0.8m
y_0=0m
z = 0.99298m

r=0.25m
x_0=0.8m
y_0=0m
z_{min} = 0.9m
z_{max} = 0.99298m

Rys. 64. Zadana trajektoria – okrąg: a) – dwuwymiarowa, b) – trójwymiarowa



x_{min}=0.7m
x_{max}=1.0m
y_{min}= −0.35m
y_{max}=0.35m
z = 0.99298m

x_{min}=0.7m
x_{max}=1.0m
y_{min}=−0.35m
y_{max}=0.35m
z_{min} = 0.9m
z_{max} = 0.99298m

Rys. 65. Zadana trajektoria – prostokąt: a) – dwuwymiarowa, b) – trójwymiarowa

W każdym kroku można było wyróżnić dwa punkty – początkowy oraz końcowy (zadany). Punkt, który w kroku czasowym n_t był punktem końcowym, w kroku czasowym n_{t+1} był punktem początkowym. Odległości pomiędzy punktami na zdyskretyzowanej, zadanej trajektorii w zależności od jej rodzaju wynosiły od 8 do 10mm. Dzięki zastosowaniu dyskretyzacji, zadanie podążania po zadanej trajektorii ruchu, zostało uproszczone do szeregowo wykonywanych zadań pozycjonowania TCP robota w kolejnych zadanych

punktach. Kolejny punkt końcowy był brany pod uwagę w momencie, jeśli odległość euklidesowa w przestrzeni trójwymiarowej pomiędzy punktem zadanym, a aktualną pozycją była mniejsza niż odległość progowa d_{th} równa 5mm. Wartość ta została dobrana z uwagi na błąd agenta#2 wytrenowanego do pozycjonowania TCP w zadanym punkcie, który został zastosowany do podążania po zadanej trajektorii ruchu. W przypadku badań przedstawionych w rozdziale 8, średni błąd e dla agenta#2 wynosił około 3mm, w większości przypadków było to jednak od 1.5mm do 2mm. Odległość progowa d_{th} została ustawiona na taką wartość, żeby uniknąć przypadków, w których agent próbuje przez zbyt dużą liczbę kroków pozycjonować TCP robota w tym samym zadanym punkcie. Liczba maksymalnych kroków pomiędzy punktem początkowym i końcowym została ustawiona na wartość 5.

Badano ruch robota po czterech różnych trajektoriach, przy czym badano trajektorie, których wymiary geometryczne nie zmieniały się w osi Z (płaszczyzna trajektorii równoległa do płaszczyzny XY) oraz gdy zmieniały się w tej osi. Dla uproszczenia nazewnictwa trajektorie nazwano odpowiednio jako dwumiarowe oraz trójwymiarowe. Dwie miały kształt okręgu (Rys. 64) i dwie kształt prostokąta (Rys. 65). W trakcie badań doświadczalnych oprócz przeszkody przedstawionej na Rys. 53b, stosowano również przeszkodę w kształcie walca pokazaną na Rys. 66. Testowane trajektorie zostały wygenerowane przed rozpoczęciem ruchu robota, a punkty końcowe i początkowe były z niej pobierane w trakcie ruchu.

Przeszkoda w kształcie walca o wymiarach:
0.24m x 1.05m

Rys. 66. Robot Mitsubishi z przeszkodą w kształcie walca w polu roboczym

W tabelach w rozdziałach 9.2.1 i 9.3.1 przedstawiono wyniki dla czterech testowanych trajektorii. Porównywane parametry to całkowity uchyb (sumaryczny błąd) e_{tr}, czyli różnica pomiędzy trajektorią zadaną, a trajektorią ruchu TCP robota, średni błąd e_{tr} (całkowity błąd e_{tr} podzielony przez liczbę dyskretnych punktów na trajektorii zadanej), maksymalny błąd e_{tr}, który wystąpił dla jednego z dyskretnych punktów na trajektorii zadanej, średnie błędy e_{tr} dla poszczególnych osi. Tak jak w przypadku badań doświadczalnych przedstawionych w

rozdziale 8.3, tak i dla badań przedstawionych w rozdziale 9.3, maksymalna prędkość robota została ograniczona do $35\frac{mm}{s}$.

9.2 Wyniki badań symulacyjnych

9.2.1 Ruch po zadanej trajektorii

Tabela 10 zawiera wyniki uzyskane w trakcie badań symulacyjnych dla ruchu po trajektorii w kształcie okręgu. Najmniejszy sumaryczny błąd e_{tr} uzyskano dla trajektorii trójwymiarowej ($z = var$) oraz kiedy TCP robota poruszał się przeciwnie do kierunku ruchu wskazówek zegara (ccw). Średni błąd e_{tr} dla tego przypadku wynosił 1.27mm. Dla wszystkich czterech testowych przypadków, średni błąd nie był większy niż 2.5mm, natomiast błąd maksymalny nie był większy niż 5mm. Największe błędy występowały, kiedy TCP robota poruszał się zgodnie z kierunkiem ruchu wskazówek zegara po zadanej trajektorii (cw). Dla tych przypadków średni błąd wynosił ponad 2mm. Dla poszczególnych osi największe błędy występowały w osi X, ale nie przekraczały one jednak 1.5mm.

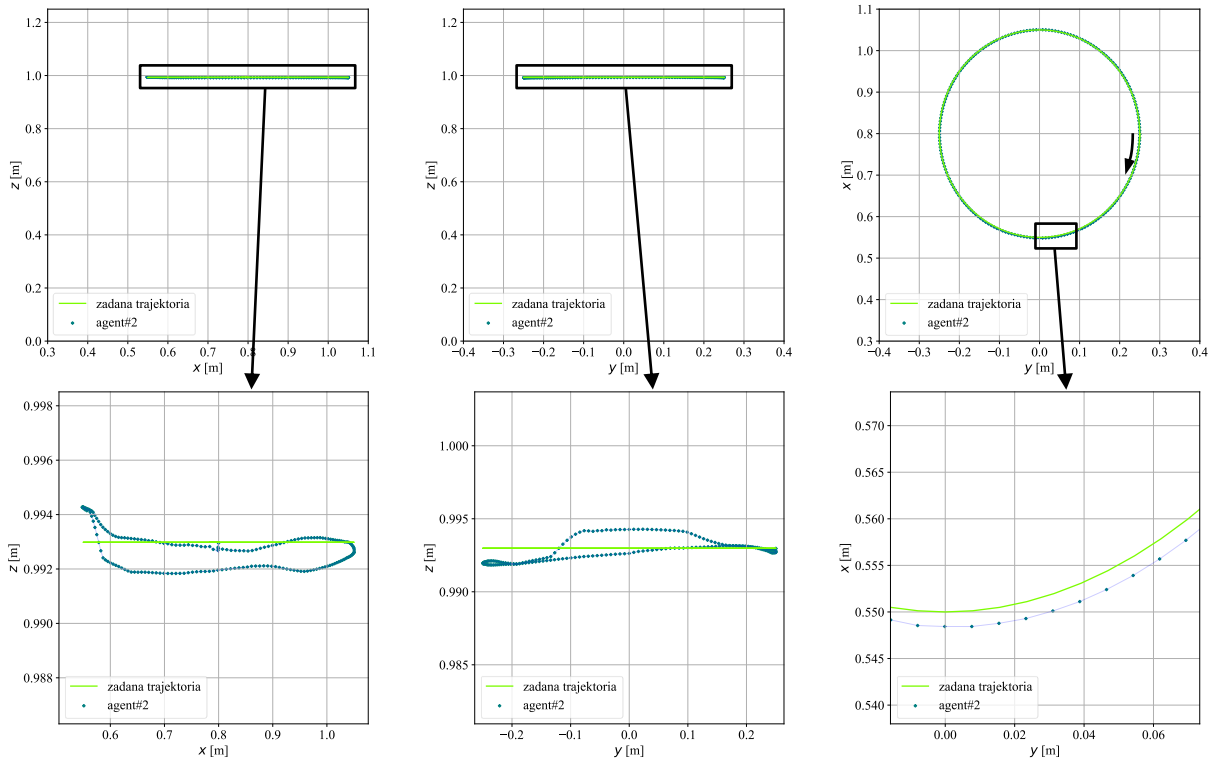
Tabela 10. Wyniki badań symulacyjnych ruchu po trajektorii w kształcie okręgu

z=$const$/ var	**kierunek**	**$\sum e_{tr}$ [mm]**	**śr. e_{tr} ± $\sigma_{e_{tr}}$ [mm]**	**maks. e_{tr} [mm]**	**śr. e_{trx} ± $\sigma_{e_{trx}}$[mm]**	**śr. e_{try} ± $\sigma_{e_{try}}$[mm]**	**śr. e_{trz} ± $\sigma_{e_{trz}}$[mm]**
$const$	*ccw*	322.11	1.57±0.75	2.75	1.26±0.81	0.38±0.27	0.61±0.44
	cw	503.11	2.36±1.28	4.97	1.41±1.12	1.55±0.95	0.67±0.49
var	*ccw*	259.89	**1.27±0.64**	2.52	0.98±0.68	0.40±0.31	0.44±0.36
	cw	476.29	2.32±1.13	4.70	1.47±1.19	1.39±0.90	0.50±0.30

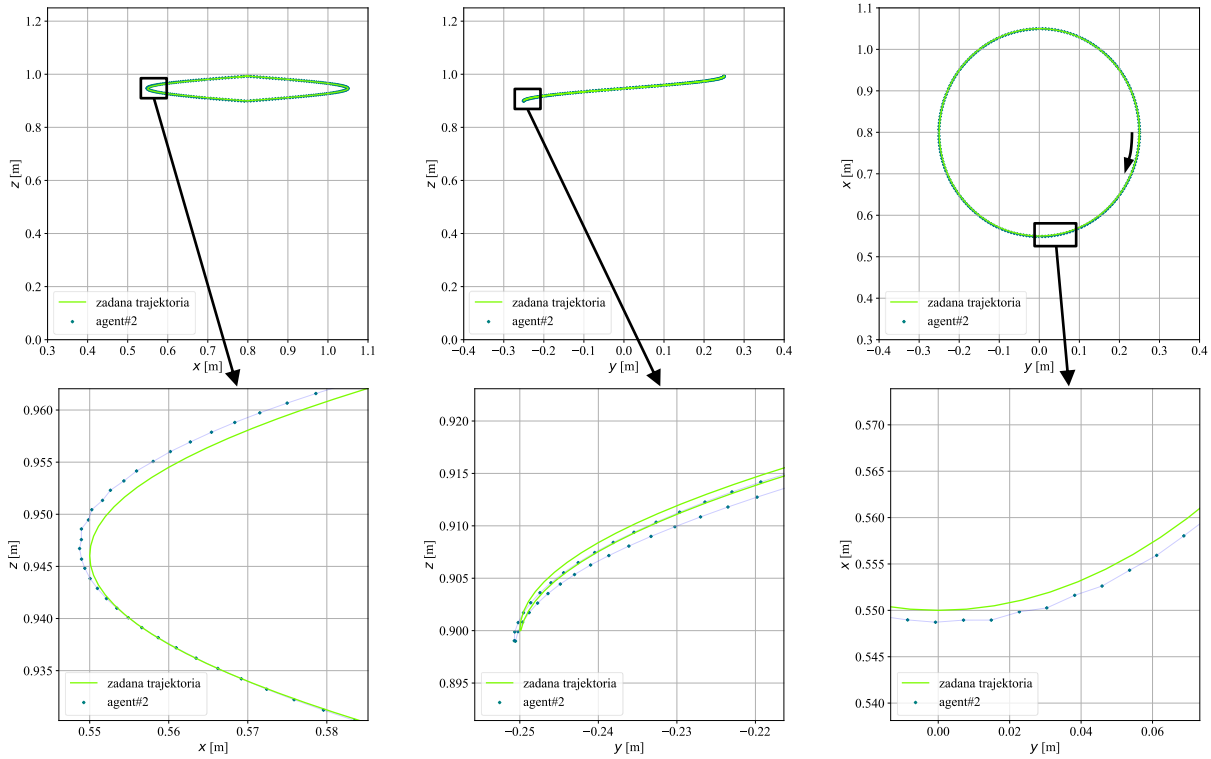
W Tabeli 11 przedstawiono wyniki badań przy poruszaniu się TCP robota po trajektorii zadanej w kształcie prostokąta. Tak jak w przypadku trajektorii zadanej w kształcie okręgu, najmniejszy błąd osiągnięto dla trajektorii trójwymiarowej ($z = var$), kiedy TCP robota poruszał się przeciwnie do ruchu wskazówek zegara. Średni błąd e_{tr} wynosił 1.15mm, a maksymalny 3.55mm. Dla testowanych trajektorii błąd e_{tr} nie przekraczał 2mm. Największy średni błąd dla poszczególnych osi wystąpił dla osi Y i trajektorii dwuwymiarowej, kiedy TCP robota poruszał się zgodnie z kierunkiem ruchu wskazówek zegara. W większości przypadków średnie błędy dla poszczególnych osi były mniejsze niż 1mm.

Tabela 11. Wyniki badań symulacyjnych ruchu po trajektorii w kształcie prostokąta

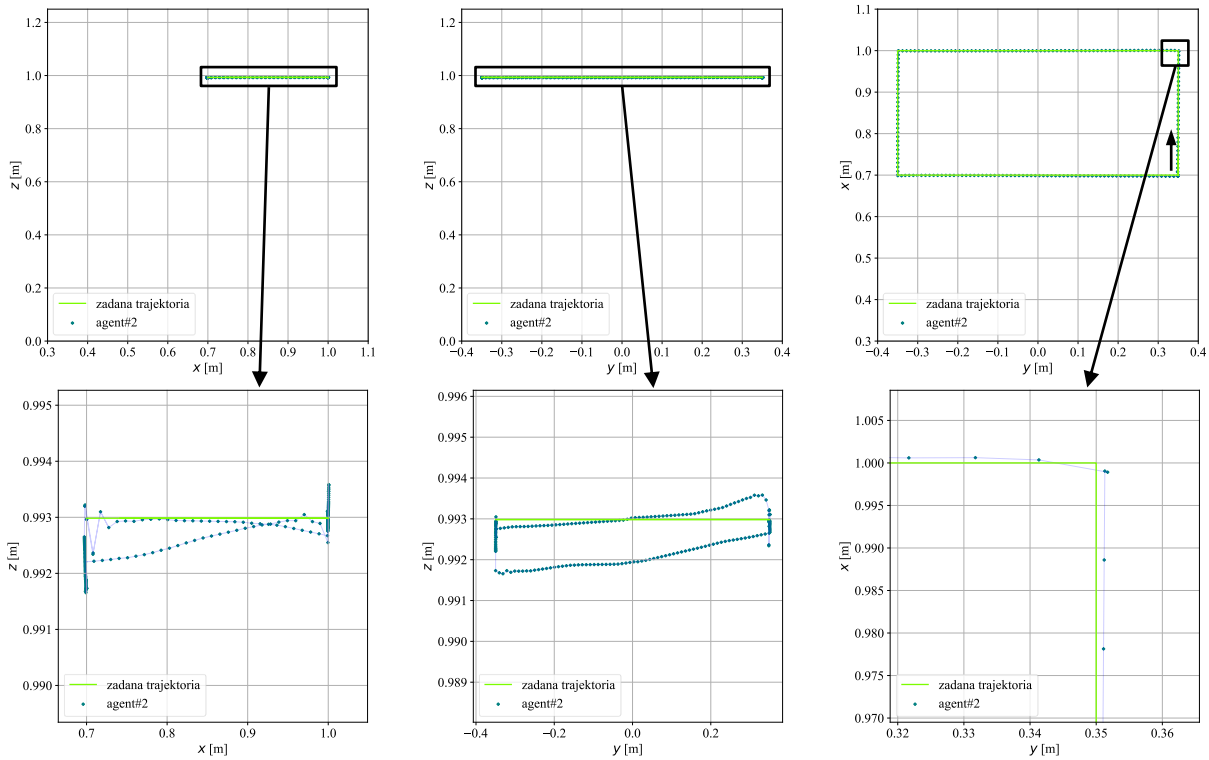
z=$const$/ var	**kierunek**	**$\sum e_{tr}$ [mm]**	**śr. e_{tr} ± $\sigma_{e_{tr}}$ [mm]**	**maks. e_{tr} [mm]**	**śr. e_{trx} ± $\sigma_{e_{trx}}$[mm]**	**śr. e_{try} ± $\sigma_{e_{try}}$[mm]**	**śr. e_{trz} ± $\sigma_{e_{trz}}$[mm]**
$const$	*ccw*	281.44	1.38±0.64	3.55	0.77±0.63	0.78±0.55	0.50±0.42
	cw	412.72	2.02±0.60	3.62	1.08±0.85	1.34±0.61	0.46±0.41
var	*ccw*	234.44	**1.15±0.61**	3.55	0.86±0.51	0.46±0.38	0.40±0.39
	cw	314.37	1.67±0.71	3.21	0.96±0.84	0.99±0.55	0.49±0.41



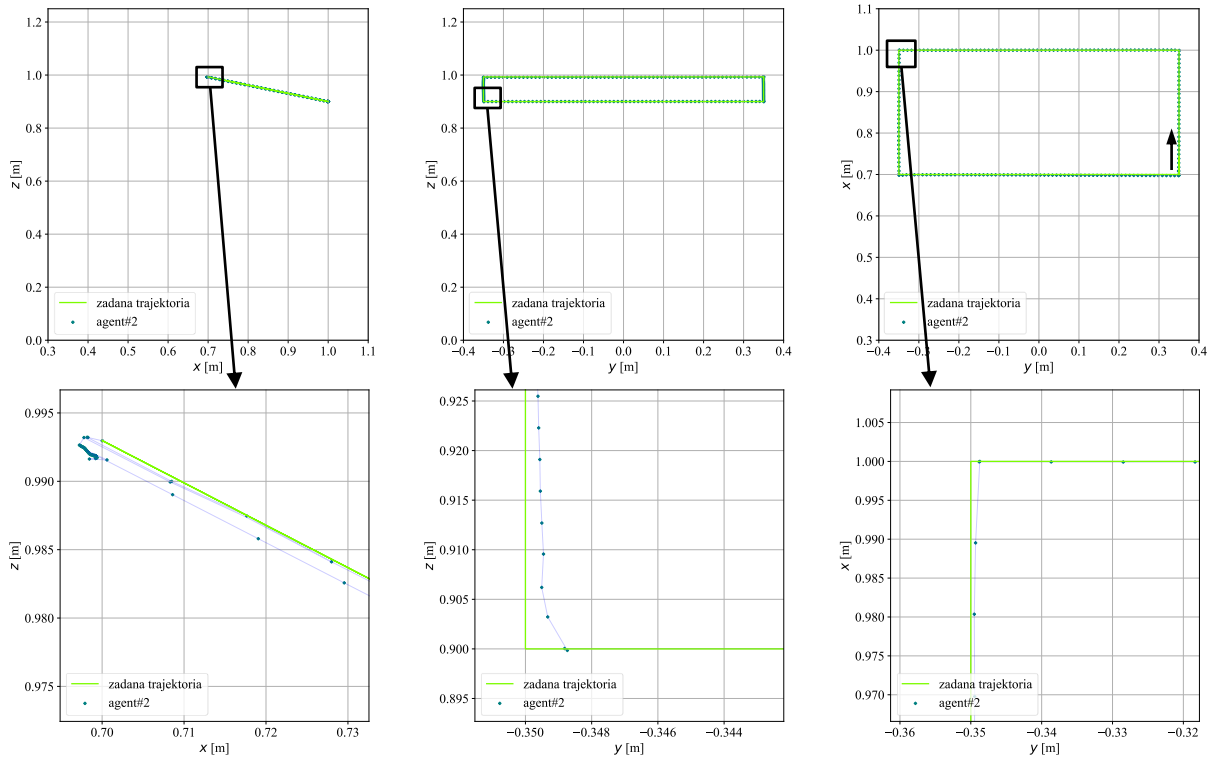
Rys. 67. Trajektoria po jakiej poruszał się TCP robota w badaniach symulacyjnych dla dwuwymiarowej (z = const) zadanej trajektorii w kształcie okręgu. Ruch TCP robota zgodnie z kierunkiem ruchu wskazówek zegara



Rys. 68. Trajektoria po jakiej poruszał się TCP robota w badaniach symulacyjnych dla trójwymiarowej (z = var) zadanej trajektorii w kształcie okręgu. Ruch TCP robota zgodnie z kierunkiem ruchu wskazówek zegara



Rys. 69. Trajektoria po jakiej poruszał się TCP robota w badaniach symulacyjnych dla dwuwymiarowej ($z = const$) zadanej trajektorii w kształcie prostokąta. Ruch TCP robota przeciwnie do kierunku ruchu wskazówek zegara

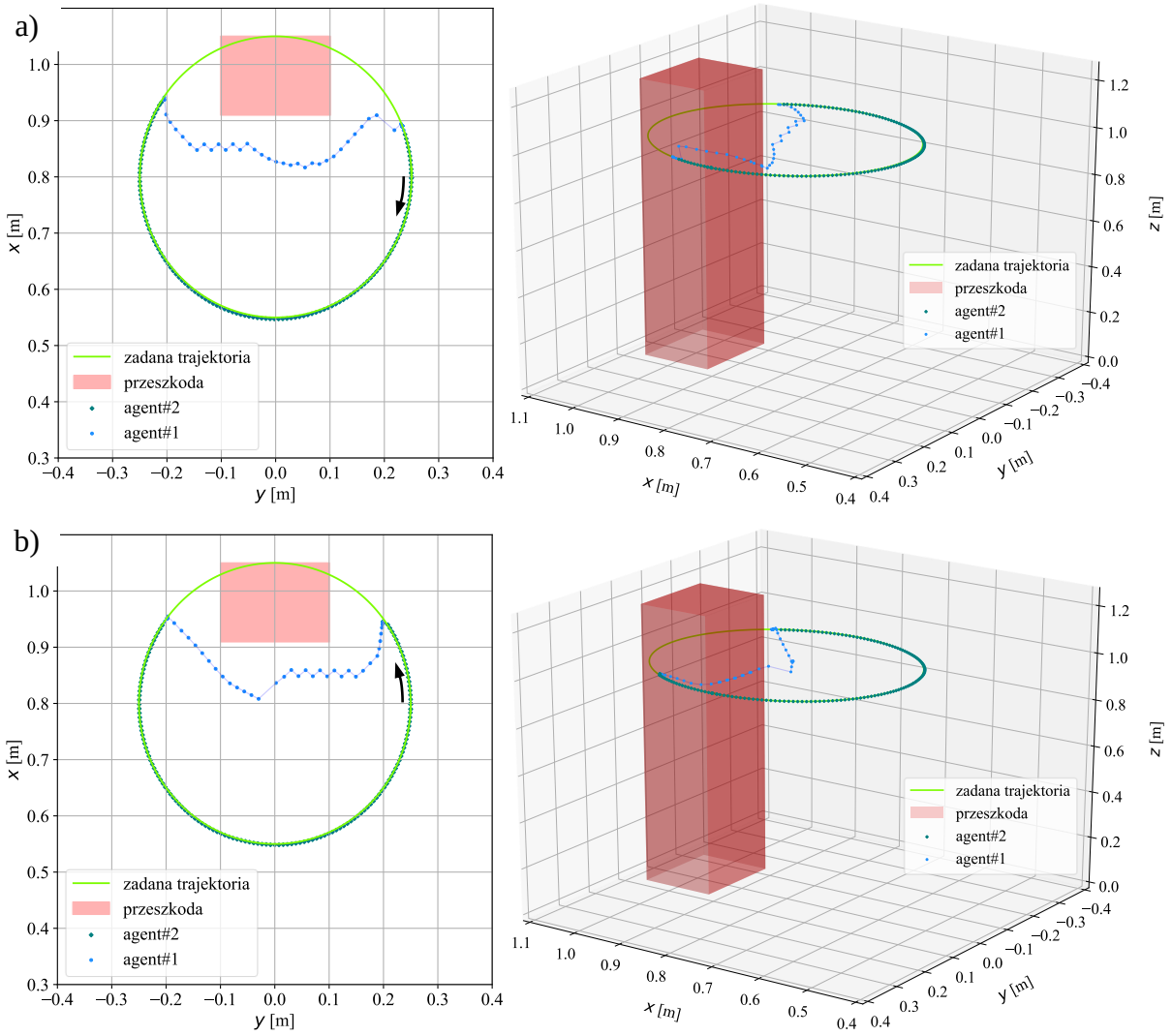


Rys. 70. Trajektoria po jakiej poruszał się TCP robota w badaniach symulacyjnych dla trójwymiarowej ($z = var$) zadanej trajektorii w kształcie prostokąta. Ruch TCP robota przeciwnie do kierunku ruchu wskazówek zegara

Na rysunkach od 67 do 70 przedstawiono przykłady czterech różnych trajektorii, po jakich poruszał się TCP robota. Dla dwuwymiarowej trajektorii w kształcie okręgu (Rys. 67) można zauważyć, że TCP robota poruszał się bez nieregularnych ruchów. W przypadku trajektorii trójwymiarowej (Rys. 68) największe błędy i różnice pomiędzy trajektorią, po której poruszał się TCP robota, a zadaną występowały w miejscach zmiany kierunku ruchu TCP robota, zwłaszcza na płaszczyźnie ZX i ZY. Dla trajektorii zadanej o kształcie prostokąta, zarówno dla dwuwymiarowej (Rys. 69) jak i trójwymiarowej (Rys. 70) widać, że największy uchyb występuje w punktach zmiany kierunku ruchu, to jest na narożnikach. Można to zaobserwować w szczególności na płaszczyźnie XY.

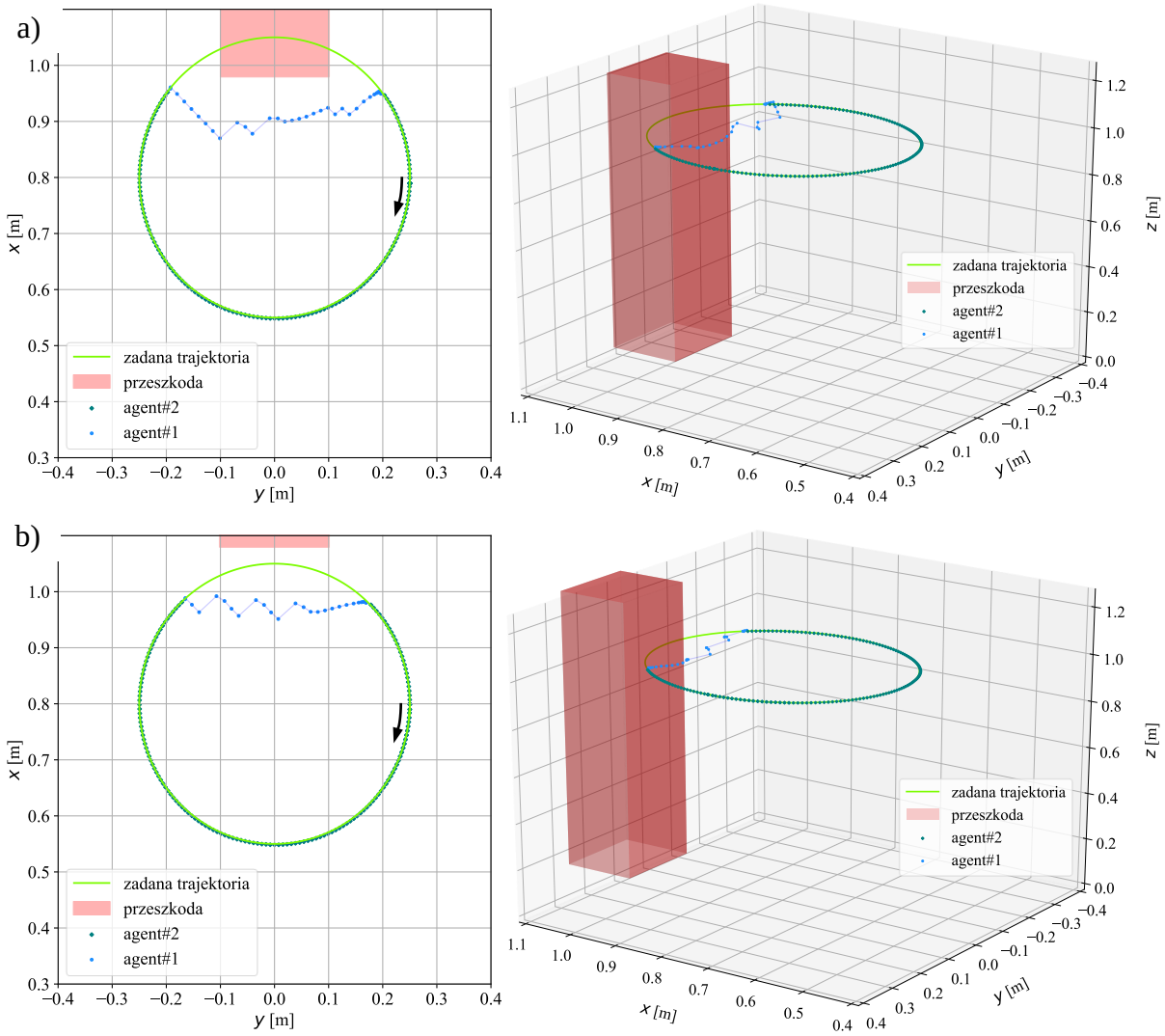
9.2.2 Omijanie przeszkód na zadanej trajektorii ruchu

W trakcie opisanych niżej badań symulacyjnych skupiono się na wstępnej ewaluacji poprawności algorytmu pod kątem omijania przeszkód na zadanej, dwuwymiarowej



Rys. 71. Trajektoria po jakiej poruszał się TCP robota w badaniach symulacyjnych, gdy przeszkoda znajdowała się w polu roboczym. Trajektoria zadana okrąg $z = const$. Ruch TCP robota: a) – zgodnie z kierunkiem, b) – przeciwnie do kierunku ruchu wskazówek zegara

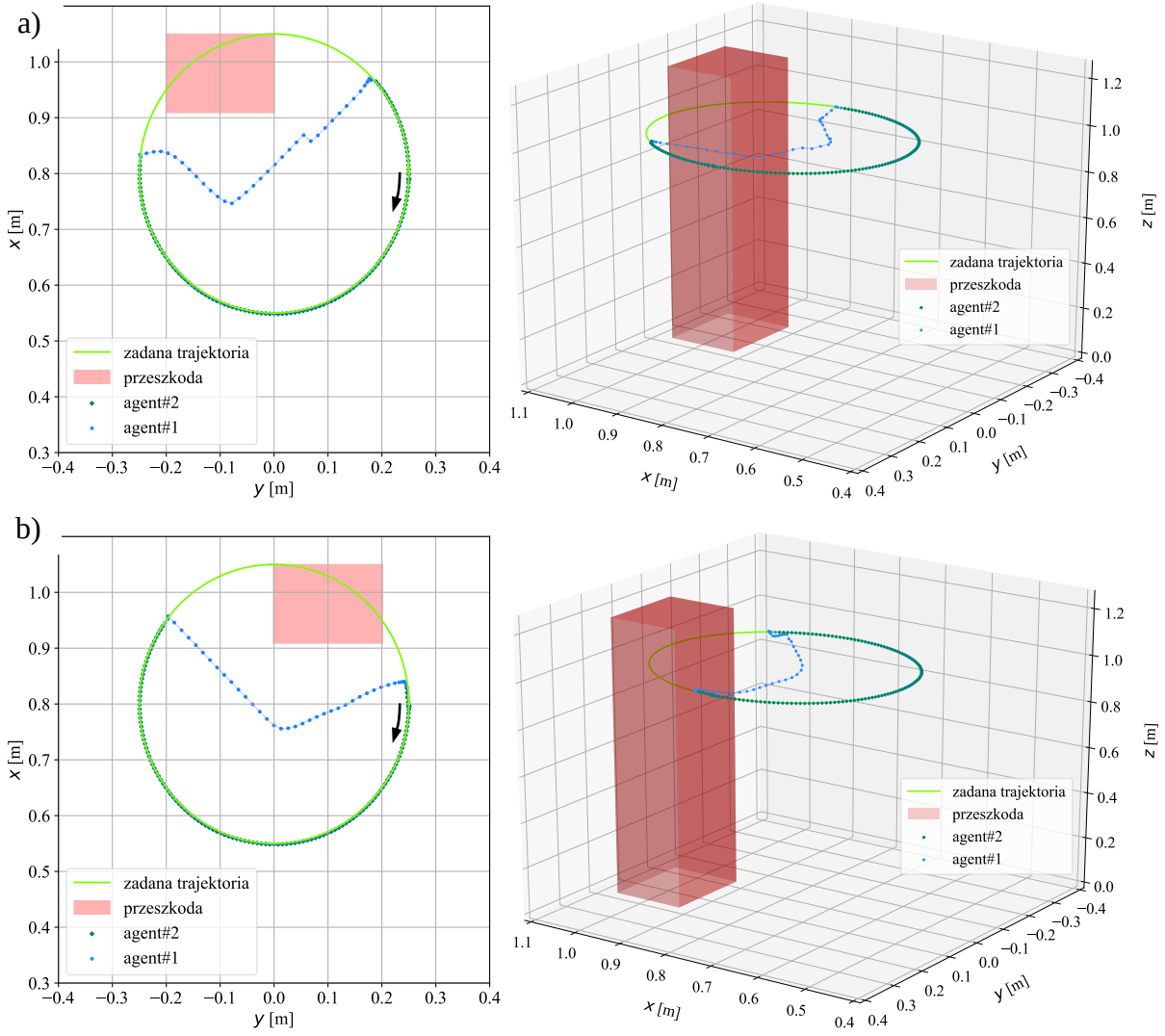
trajektorii w kształcie okręgu i prostokąta. Przeszkoda była umieszczana w różnych miejscach w polu roboczym. Testowano zachowanie się algorytmu i poruszanie się robota po trajektorii zgodnie i przeciwnie do kierunku ruchu wskazówek zegara. Badania symulacyjne miały na celu przetestowanie, czy wytrenowany system osiągnął dostateczny stopień generalizacji problemu, jakim jest omijanie przeszkód na zadanej trajektorii. Testy symulacyjne dostarczyły poglądowych informacji, czy zaimplementowany w warunkach laboratoryjnych system będzie działał poprawnie.



Rys. 72. Trajektoria po jakiej poruszał się TCP robota w badaniach symulacyjnych, gdy przeszkoda znajdowała się w polu roboczym. Trajektoria zadana okrąg $z = const$. Ruch TCP robota zgodnie z kierunkiem ruchu wskazówek zegara. Pozycja przeszkody: a) – x=1.05m, y=0m; b) – x=1.15m, y=0m

Przebiegi przedstawione na Rys. 71 pokazują trajektorie, po których poruszał się TCP robota dla trajektorii zadanej o kształcie okręgu, kiedy w polu roboczym znajdowała się przeszkoda w kształcie prostopadłościanu, usytuowana w pozycji x=0.98m, y=0m. Jak widać zarówno dla ruchu zgodnie jak i przeciwnie do kierunku ruchu wskazówek zegara, algorytm poprawnie dokonywał manewru ominięcia przeszkody. W kształcie trajektorii ruchu robota można zauważyć podobieństwa do kształtów przebiegów, które zostały przedstawione w

rozdziale 8.2. W przypadku, gdy przeszkoda nie była wykrywana, agent#2 przemieszczał TCP robota po zadanej trajektorii. W momencie, gdy umieszczona w polu roboczym przeszkoda została wykryta, agent#1 sterował TCP robota tak, aby ją bezkolizyjnie ominąć.

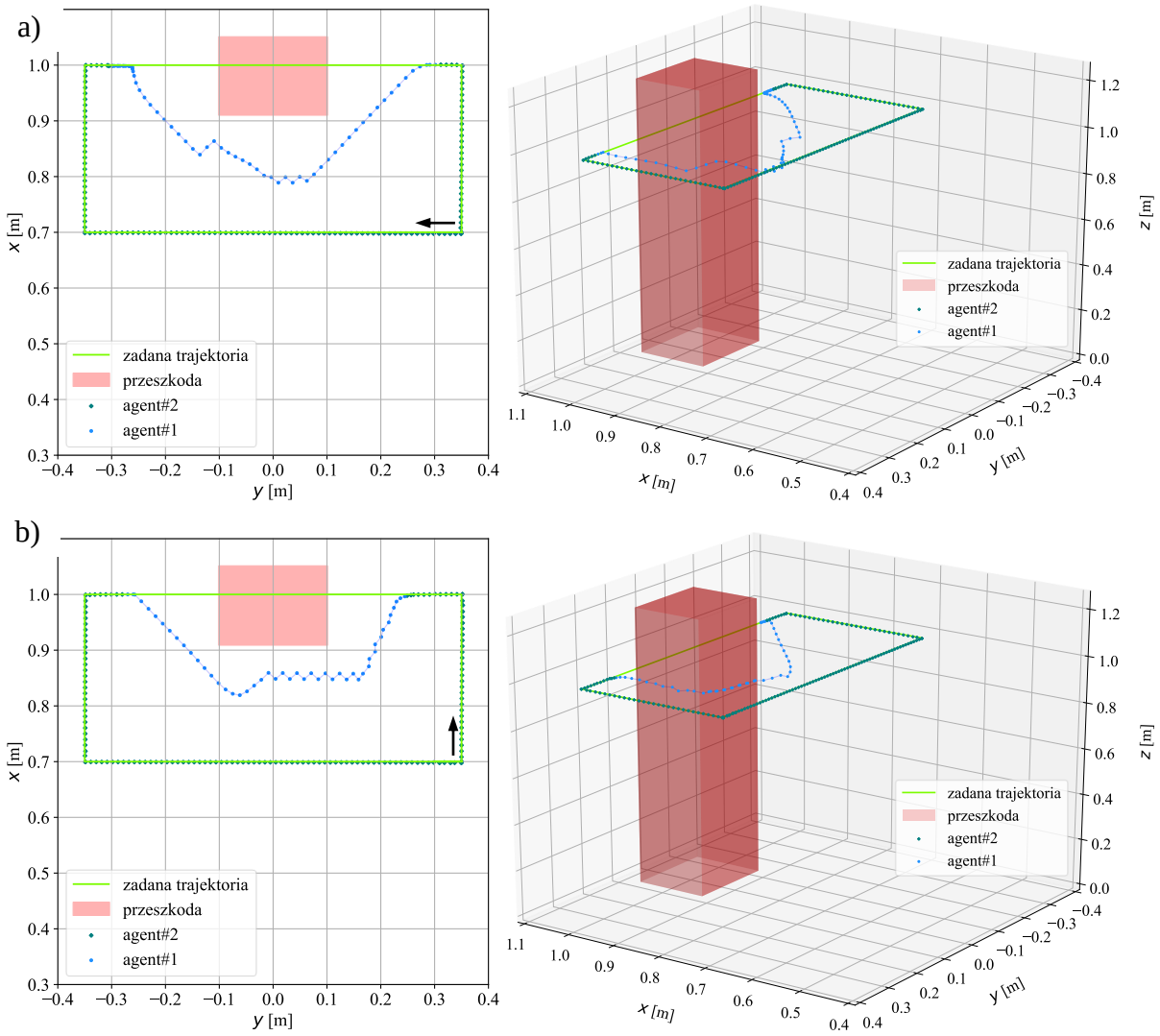


Rys. 73. Trajektoria po jakiej poruszał się TCP robota w badaniach symulacyjnych, gdy przeszkoda znajdowała się w polu roboczym. Trajektoria zadana okrąg $z = const$. Ruch TCP robota zgodnie z kierunkiem ruchu wskazówek zegara. Pozycja przeszkody: a) – x=0.98m, y=–0.1m; b) – x=0.98m, y=0.1m

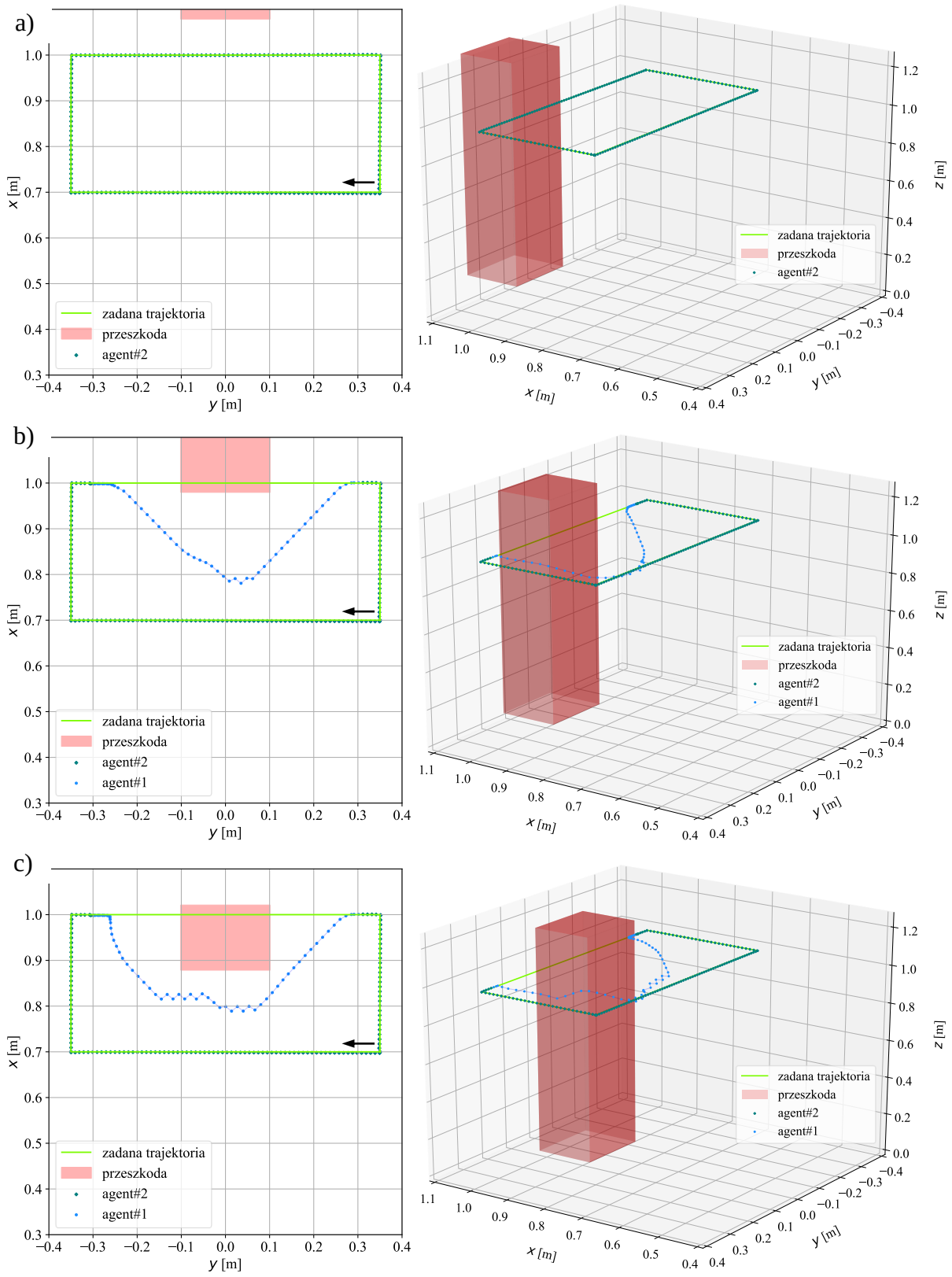
Na rysunkach 72 i 73 zamieszczono zarejestrowane trajektorie ruchu robota dla trajektorii zadanej o kształcie okręgu, dla różnych pozycji przeszkody w strefie roboczej. Na dwóch wykresach przedstawionych na Rys. 72 pozycja przeszkody była zmieniana w osi X w porównaniu do pozycji z Rys. 71. Dla pozycji przeszkody x=1.05m, y=0m można zaobserwować podobne zachowanie się systemu i trajektorie ruchu, jak dla pozycji przeszkody przedstawionej na Rys. 71. Dla pozycji przeszkody x=1.15m, y=0m (Rys. 72b) zadana trajektoria nie przecinała obrysu przeszkody, ale można zauważyć, że niektóre akcje wykonywał agent#1, odpowiedzialny za omijanie przeszkód. Pomimo tego, że trajektoria zadana dla TCP robota nie przecinała obrysu przeszkody, to cała kiść robota mogła już w nią uderzyć. Z tego powodu, gdy TCP manipulatora znajdował się w bliskiej odległości od

przeszkody (około 0.1m), system sterujący przechodził od poruszania się po zadanej trajektorii do wykonywania akcji omijania w celu dalszego bezkolizyjnego ruchu.

Trajektorie pokazane na Rys. 73 zarejestrowano, gdy pozycja przeszkody została zmieniona w osi Y w porównaniu do pozycji przedstawionej na Rys. 71. Przetestowano dwie pozycje, pierwsza o współrzędnych x=0.98m, y=−0.1m (Rys. 73a), a druga x=0.98m, y=0.1m (Rys. 73b). Jak widać na przedstawionych wykresach, algorytm poprawnie omijał przeszkody umieszczane w polu roboczym robota. W przypadku, gdy przeszkoda znajdowała się skrajnie z lewej (y=−0.1m) albo prawej (y=0.1m) strony można zauważyć, że algorytm znacznie wcześniej zaczynał manewr jej omijania w porównaniu z działaniami wykonywanymi, gdy przeszkody znajdowały się na środku (y=0m) obszaru roboczego. W takich sytuacjach agent#1 po dokonaniu manewru omijania przeszkody wykazywał „tendencję” do tego, aby nie przemieszczać TCP robota w rejony bardzo blisko przeszkody, tylko w kierunku najbliższego punktu znajdującego się na trajektorii zadanej.



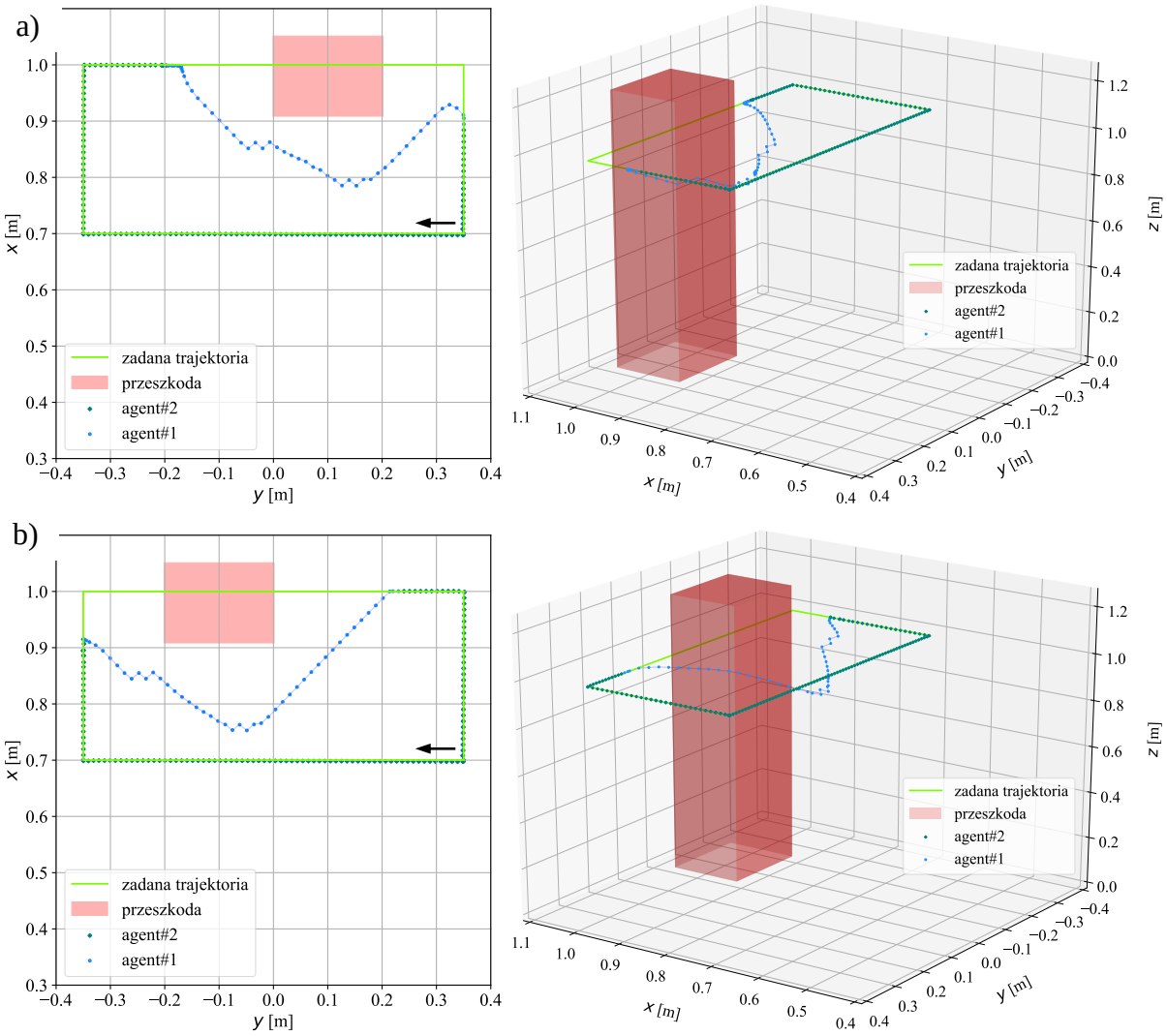
Rys. 74. Trajektoria po jakiej poruszał się TCP robota w badaniach symulacyjnych, gdy przeszkoda znajdowała się w polu roboczym. Trajektoria zadana prostokąt $z = const$. Ruch TCP robota: a) – zgodnie z kierunkiem, b) – przeciwnie do kierunku ruchu wskazówek zegara



Rys. 75. Trajektoria po jakiej poruszał się TCP robota w badaniach symulacyjnych, gdy przeszkoda znajdowała się w polu roboczym. Trajektoria zadana prostokąt z = const. Ruch TCP robota zgodnie z kierunkiem ruchu wskazówek zegara. Pozycja przeszkody: a) – x=1.15 m, y=0m; b) – x=1.05m, y=0m; c) – x=0.95m, y=0m

Przebieg przedstawiony na Rys. 74 ilustruje trajektorie w kształcie prostokąta, po których poruszał się TCP robota, gdy w polu roboczym znajdowała się przeszkoda w kształcie prostopadłościanu w pozycji x=0.98m, y=0m. Trajektorie przedstawiono dla ruchu zgodnym z kierunkiem ruchu wskazówek zegara oraz przeciwnym. Podobnie jak w przypadku trajektorii zadanej o kształcie okręgu, algorytm bezkolizyjnie omijał przeszkody.

Trzy trajektorie przedstawione na Rys. 75 przedstawiają sytuację, w której pozycja przeszkody była zmieniana w osi X w odniesieniu do pozycji przeszkody przedstawionej na Rys. 74. W porównaniu do dwóch testowanych pozycji przeszkody dla trajektorii w kształcie okręgu przetestowano także działanie algorytmu, gdy umieszczono przeszkodę w pozycji x=0.95m, y=0m (Rys. 75c). Zarówno dla tej pozycji jak i pozycji x=1.05m, y=0m (Rys. 75b) trajektoria ruchu jest podobna do tej przedstawionej na Rys. 74. W przypadku pozycji przeszkody x=1.15m, y=0m (Rys. 75a) trajektoria zadana nie przecina obrysu przeszkody. Jednak TCP robota znajdował się na tyle daleko od przeszkody, że dla całego ruchu, agent



Rys. 76. Trajektoria po jakiej poruszał się TCP robota w badaniach symulacyjnych, gdy przeszkoda znajdowała się w polu roboczym. Trajektoria zadana prostokąt $z = const$. Ruch TCP robota zgodnie z kierunkiem ruchu wskazówek zegara. Pozycja przeszkody: a) – x=0.98m, y=0.1m; b) – x=0.98m, y=–0.1m

podążał po zadanej trajektorii w kształcie prostokąta. Nie wystąpiła taka sytuacja jak dla trajektorii o kształcie okręgu, przedstawionej na Rys. 72b.

Na Rys. 76 pokazano trajektorie, analogiczne do tych na Rys. 73. Przetestowano zachowanie algorytmu dla dwóch pozycji przeszkody x=0.98m, y=0.1m (Rys. 76a) i x=0.98m, y=−0.1m (Rys. 76b). Algorytm poprawnie wykonywał manewr omijania przeszkód znajdujących się w polu roboczym robota.

9.3 Wyniki badań doświadczalnych

9.3.1 Ruch po zadanej trajektorii

W tabelach 12 i 13 przedstawiono wyniki badań doświadczalnych dla sterowania robotem podczas ruchu po zadanej trajektorii o kształcie okręgu albo prostokąta. Dane z tabel można porównać z wynikami badań symulacyjnych zaprezentowanych odpowiednio w tabeli 10 i 11. Zarówno dla trajektorii zadanej w kształcie okręgu jak i prostokąta, wyniki uzyskane w trakcie badań doświadczalnych są bardzo zbliżone do wyników uzyskanych w trakcie badań symulacyjnych.

Tabela 12. Wyniki badań doświadczalnych ruchu po trajektorii w kształcie okręgu

z=$const$/ var	**kierunek**	**$\sum e_{tr}$ [mm]**	**śr. e_{tr} ± $\sigma_{e_{tr}}$ [mm]**	**maks. e_{tr} [mm]**	**śr. e_{trx} ± $\sigma_{e_{trx}}$[mm]**	**śr. e_{try} ± $\sigma_{e_{try}}$[mm]**	**śr. e_{trz} ± $\sigma_{e_{trz}}$[mm]**
$const$	*ccw*	380.65	1.65±1.08	5.51	1.39±1.10	0.43±0.33	0.54±0.41
	cw	503.34	2.44±1.19	4.89	1.50±1.25	1.51±0.97	0.48±0.31
var	*ccw*	287.68	**1.40±1.27**	5.99	1.15±1.25	0.40±0.44	0.46±0.36
	cw	511.08	2.53±1.40	5.59	1.41±1.17	1.75±1.10	0.67±0.53

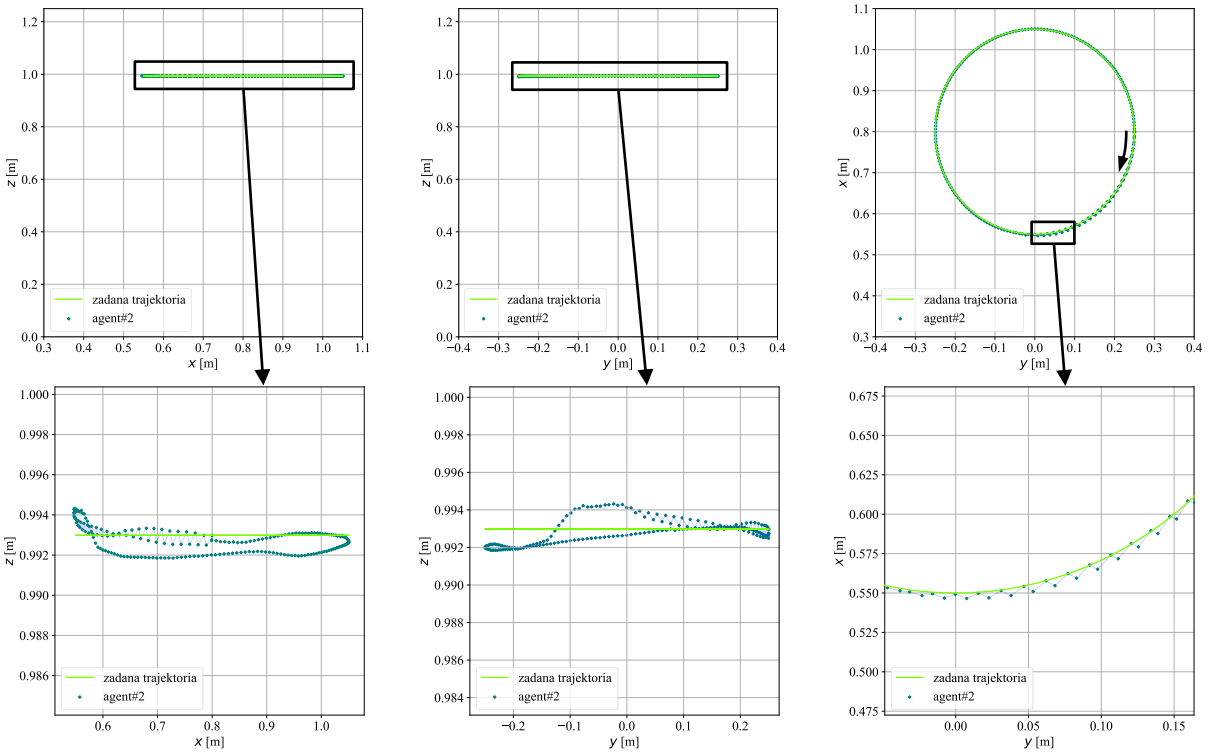
Najmniejszy średni błąd e_{tr} w przypadku trajektorii w kształcie okręgu uzyskano dla trajektorii trójwymiarowej, kiedy TCP robota poruszał się przeciwnie do kierunku ruchu wskazówek zegara. Dla tej trajektorii zarejestrowano też największy maksymalny błąd e_{tr}, który wynosił prawie 6mm (dla badań symulacyjnych był równy 2.52mm). Największe średnie błędy e_{tr} występowały, gdy TCP robota poruszał się zgodnie z kierunkiem ruchu wskazówek zegara.

Tabela 13. Wyniki badań doświadczalnych ruchu po trajektorii w kształcie prostokąta

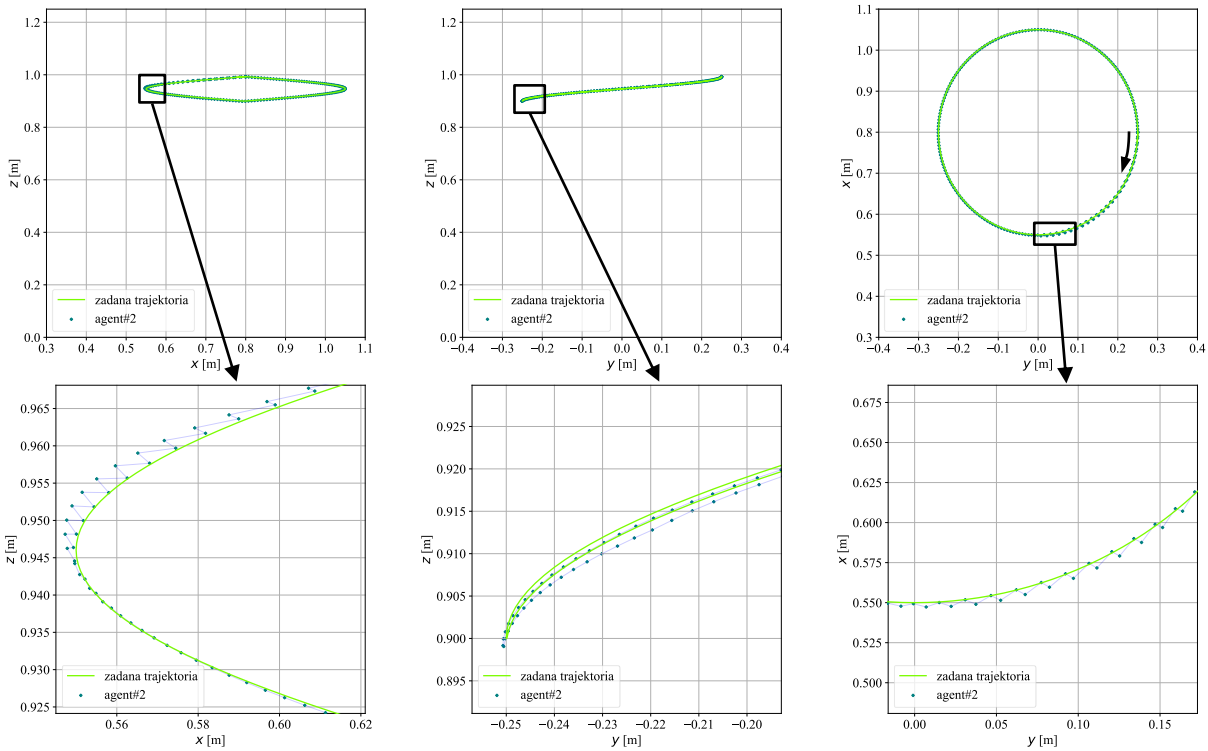
z=$const$/ var	**kierunek**	**$\sum e_{tr}$ [mm]**	**śr. e_{tr} ± $\sigma_{e_{tr}}$ [mm]**	**maks. e_{tr} [mm]**	**śr. e_{trx} ± $\sigma_{e_{trx}}$[mm]**	**śr. e_{try} ± $\sigma_{e_{try}}$[mm]**	**śr. e_{trz} ± $\sigma_{e_{trz}}$[mm]**
$const$	*ccw*	327.56	1.53±0.76	4.13	1.21±1.43	6.91±4.15	0.59±0.32
	cw	456.35	2.31±1.34	4.57	1.26±1.66	6.14±4.31	0.48±0.65
var	*ccw*	252.41	**1.24±0.95**	4.26	1.99±1.72	6.05±4.26	0.41±0.44
	cw	303.70	1.47±0.86	4.44	1.12±1.60	6.82±4.67	1.60±1.09

W przypadku trajektorii zadanej o kształcie prostokąta, najmniejszy średni błąd e_{tr} wystąpił, tak samo jak w przypadku badań symulacyjnych dla trajektorii trójwymiarowej, kiedy TCP robota poruszał się przeciwnie do kierunku ruchu wskazówek zegara. Średni błąd e_{tr} wynosił 1.24mm, przy maksymalnym błędzie równym 4.26mm. Największy średni błąd e_{tr} zarejestrowano dla trajektorii dwuwymiarowej i ruchu TCP zgodnie z kierunkiem ruchu wskazówek zegara. Maksymalne błędy dla badań doświadczalnych mieściły się w zakresie od 4mm do 4.5mm. Oznacza to średnio o 1mm gorszy wynik w porównaniu z wynikami uzyskanymi w badaniach symulacyjnych. Największy błąd wystąpił dla osi Y i wahał się on w granicach od 6mm do prawie 7mm.

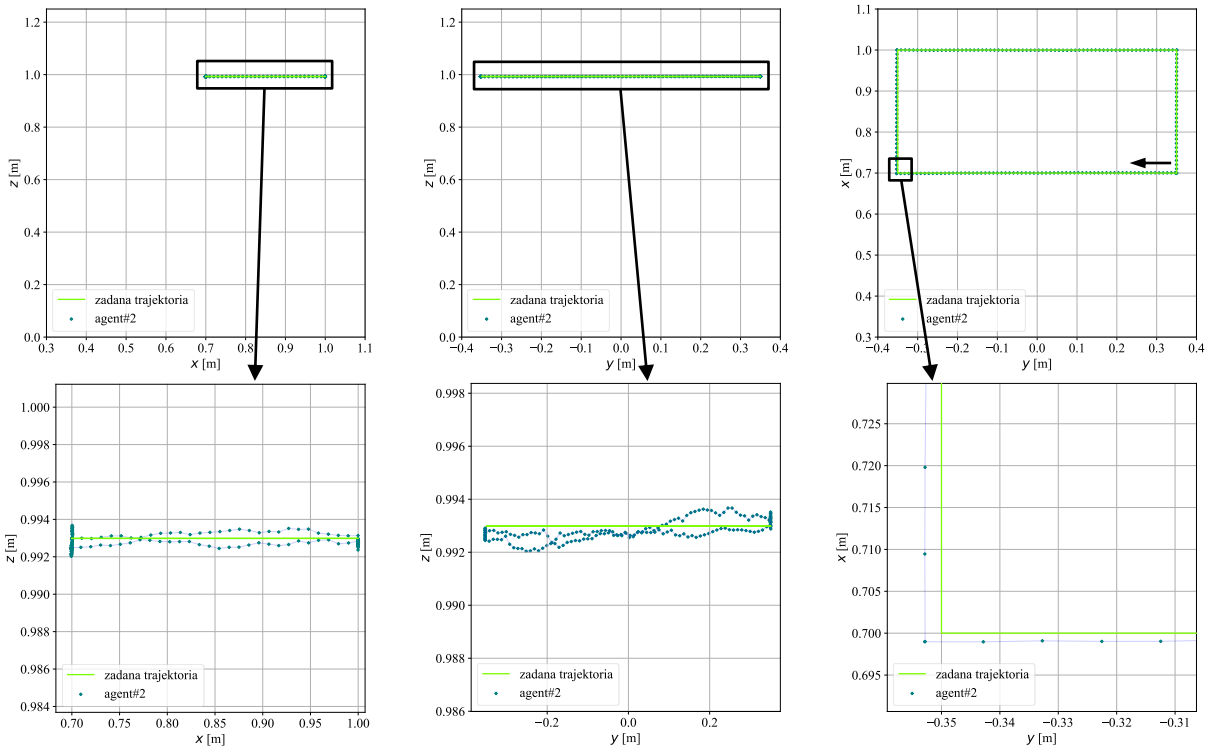
Na rysunkach od 77 do 80 przedstawiono trajektorie ruchu, po których poruszał się TCP robota dla czterech testowanych trajektorii. Uzyskane wyniki są bardzo zbliżone do zarejestrowanych w trakcie badań symulacyjnych, zaprezentowanych w rozdziale 9.2.1. Dla dwuwymiarowej (Rys. 77) oraz trójwymiarowej (Rys. 78) trajektorii o kształcie okręgu można zauważyć, że występują nieregularne ruchy po trójkącie w płaszczyźnie XY oraz ZX. Wartość międzyszczytowa tych ruchów waha się od 2mm do 8mm. W przypadku trajektorii zadanej o kształcie prostokąta, zarówno dla trajektorii dwuwymiarowej (Rys. 79), jak i trójwymiarowej (Rys. 80), największe odchylenia od zadanej trajektorii występują w narożnikach prostokąta, gdy TCP robota zmieniał kierunek ruchu.



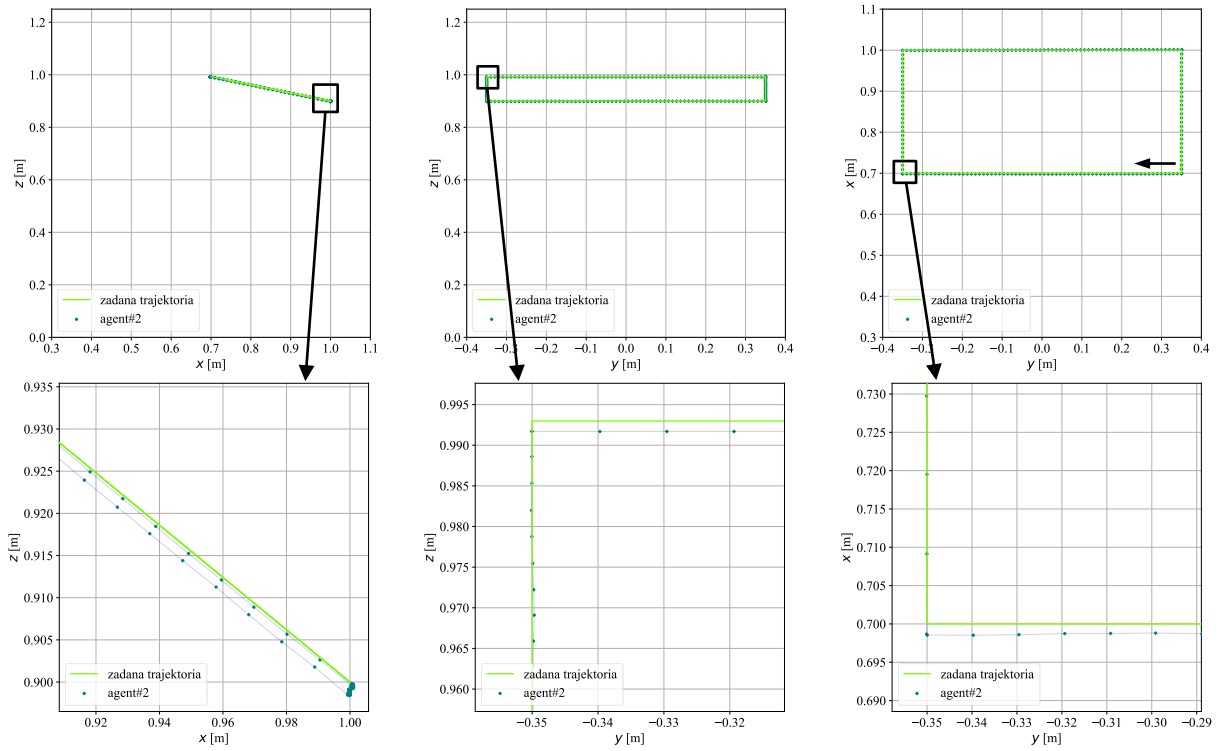
Rys. 77. Trajektoria po jakiej poruszał się TCP robota w badaniach doświadczalnych dla dwuwymiarowej ($z = const$) zadanej trajektorii w kształcie okręgu. Ruch TCP robota zgodnie z kierunkiem ruchu wskazówek zegara



Rys. 78. Trajektoria po jakiej poruszał się TCP robota w badaniach doświadczalnych dla trójwymiarowej ($z = var$) zadanej trajektorii w kształcie okręgu. Ruch TCP robota zgodnie z kierunkiem ruchu wskazówek zegara



Rys. 79. Trajektoria po jakiej poruszał się TCP robota w badaniach doświadczalnych dla dwuwymiarowej ($z = const$) zadanej trajektorii w kształcie prostokąta. Ruch TCP robota zgodnie z kierunkiem ruchu wskazówek zegara



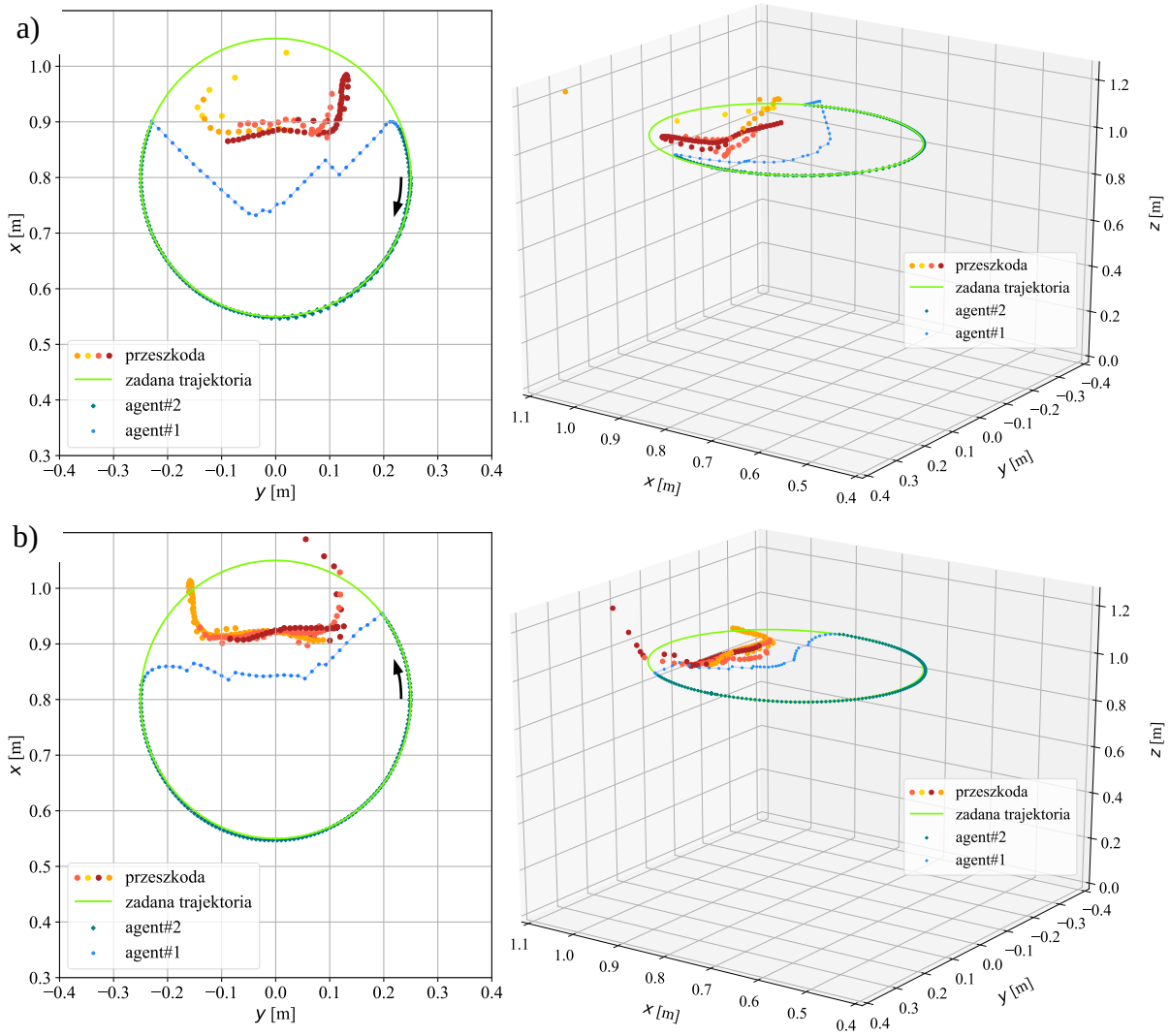
Rys. 80. Trajektoria po jakiej poruszał się TCP robota w badaniach doświadczalnych dla trójwymiarowej (z = var) zadanej trajektorii w kształcie prostokąta. Ruch TCP robota zgodnie z kierunkiem ruchu wskazówek zegara

9.3.2 Omijanie przeszkód na zadanej trajektorii ruchu

Badania doświadczalne były uzupełnieniem i rozszerzeniem badań symulacyjnych, mającym na celu potwierdzenie poprawności działania algorytmu i możliwość omijania przeszkód na zadanej trajektorii ruchu. W trakcie badań doświadczalnych przetestowano przeszkody o różnym kształcie oraz rozmieszone w różnych miejscach. Kombinacje wykonanych badań (przeszkoda – zadana trajektoria) zostały przedstawione w tabeli 14. Przetestowano działanie algorytmu w przypadku zastosowania przeszkody w kształcie walca przedstawionej na Rys. 66. Następnie sprawdzono działanie algorytmu, gdy w polu roboczym umieszczono przeszkody w kształcie prostopadłościanu o różnych rozmiarach (w tabeli 14 oznaczone jako prostopadłościan 1 i 2). Pierwsza z nich była użyta w testach prowadzonych w trakcie badań zaprezentowanych w rozdziale 8.3 i miała rozmiar 0.2m x 0.5m x 1.0m. Druga przeszkoda w kształcie prostopadłościanu była mniejsza i miała rozmiar 0.1m x 0.5m x 1.0m. Przetestowane zostało działania algorytmu, gdy w strefie roboczej robota umieszczono dwie przeszkody jedna po drugiej. Jedna z nich miała kształt prostopadłościanu, a druga walca. Finalnie przetestowano zachowanie się algorytmu w przypadku pojawienia się w polu roboczym przeszkody o nieregularnym kształcie, jakim był człowiek.

W trakcie badań doświadczalnych wykonanych w warunkach laboratoryjnych przeszkoda była umieszczana w polu roboczym ręcznie. Jej dokładna pozycja w stosunku do początku układu współrzędnych robota nie była znana. Takie umiejscowienie przeszkody skutkowało brakiem powtarzalności wyników oraz utrudnionym porównaniem ich z badaniami

symulacyjnymi. Jednak dało to możliwość oceny zachowania się systemu w różnych scenariuszach oraz uwiarygodniło generalizację problemu jakim jest omijanie przeszkód.



Rys. 81. Trajektoria po jakiej poruszał się TCP robota w badaniach doświadczalnych. Przeszkoda w kształcie prostopadłościanu. Trajektoria zadana okrąg $z = const$. Ruch TCP robota: a) – zgodnie z kierunkiem, b) – przeciwnie do kierunku ruchu wskazówek zegara

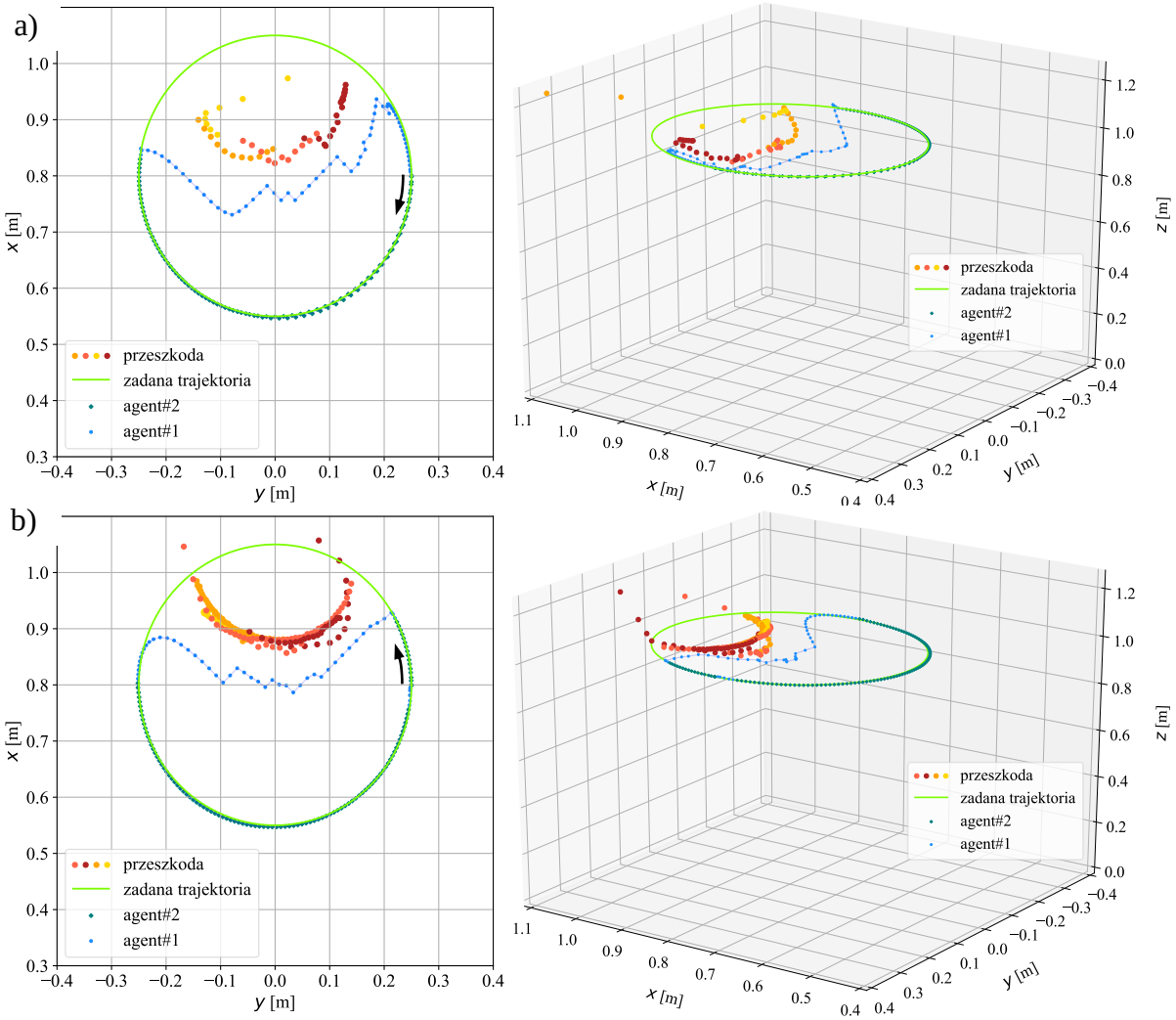
Tabela 14. Kombinacje badań doświadczalnych wykonanych w warunkach laboratoryjnych dla omijania przeszkód na zadanej trajektorii ruchu

zadana trajektoria	**przeszkoda**	**rozmiar**
okrąg 2D, 3D	prostopadłościan 1	0.2m x 0.5m x 1.0m
okrąg 2D, 3D	walec	0.24m x 1.05m
prostokąt 2D, 3D	prostopadłościan 1	0.2m x 0.5m x 1.0m
prostokąt 2D, 3D	walec	0.24m x 1.05m

prostokąt 2D	prostopadłościan 2	0.1m x 0.5m x 1.0m
prostokąt 2D	prostopadłościan 1 + walec	0.2m x 0.5m x 1.0m; 0.24m x 1.05m
prostokąt 2D	człowiek	–

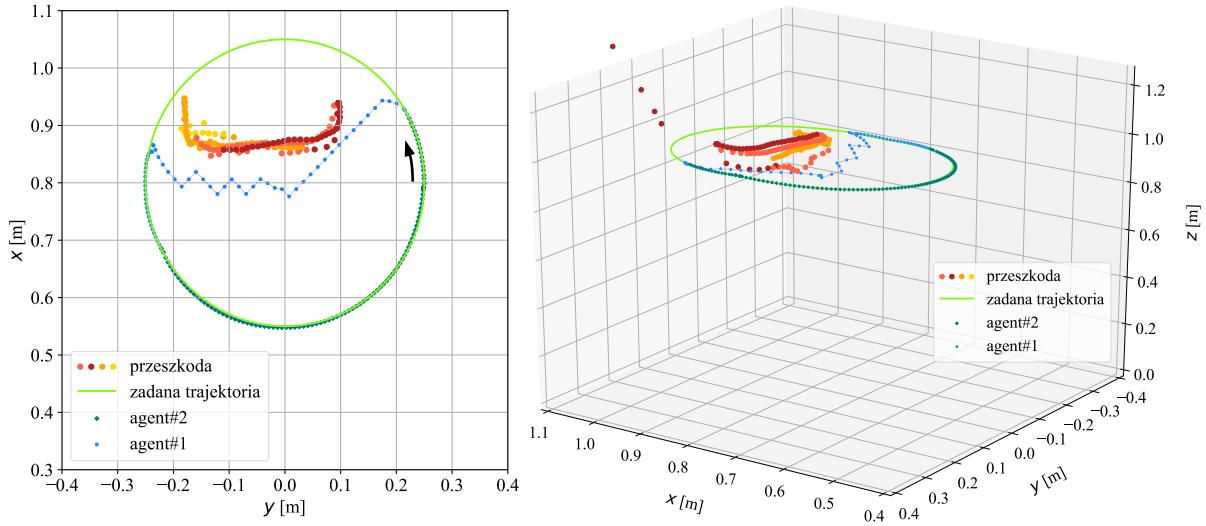
Na wykresach przedstawionych w tym rozdziale przeszkoda jest zaznaczona w formie kolorowych punktów obrazujących dane o odległości z poszczególnych czujników znajdujących się na głowicy. Każdy czujnik został oznaczony innym kolorem, dzięki temu widać zarys kształtu przeszkody rejestrowany w różnych krokach czasowych w danym epizodzie. W przypadku wykresów dwuwymiarowych przedstawiono tylko płaszczyznę *XY*.

Na rysunkach 81 i 83 pokazano trajektorie ruchu robota dla zadanych trajektorii odpowiednio dwuwymiarowej i trójwymiarowej o kształcie okręgu. Przeszkoda, która znajdowała się w polu roboczym miała kształt prostopadłościanu. Wykresy przedstawione na Rys. 82 i Rys. 84 ilustrują trajektorie, po jakich poruszał się TCP robota dla zadanej

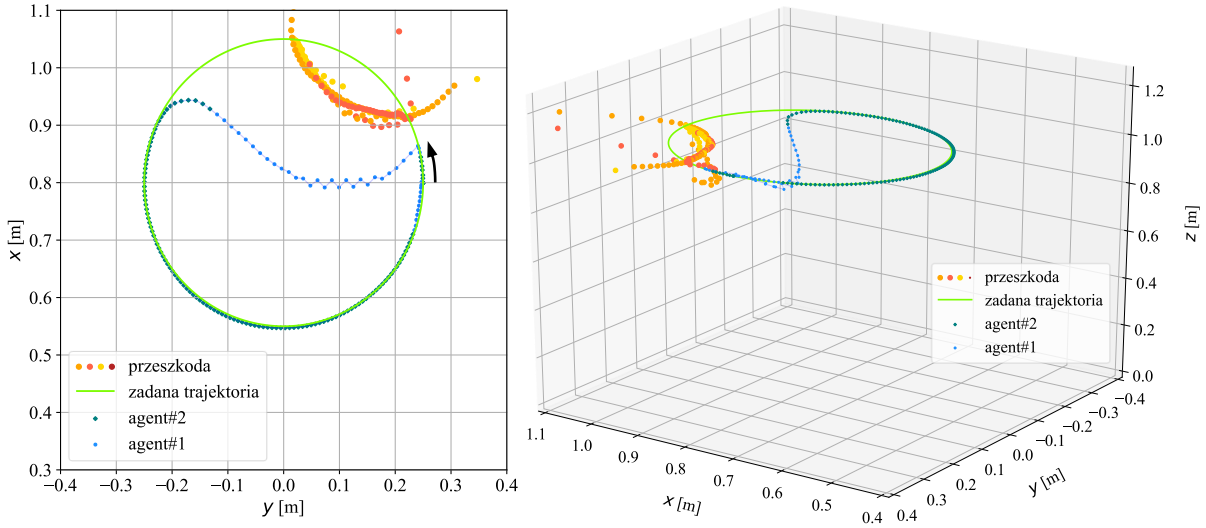


Rys. 82. Trajektoria po jakiej poruszał się TCP robota w badaniach doświadczalnych. Przeszkoda w kształcie walca. Trajektoria zadana okrąg $z = const$. Ruch TCP robota: a) – zgodnie z kierunkiem, b) – przeciwnie do kierunku ruchu wskazówek zegara

trajektorii dwuwymiarowej o kształcie okręgu, gdy w polu roboczym znajdowała się przeszkoda w kształcie walca. Rysunek 85 przedstawia trajektorię ruchu dla trójwymiarowej trajektorii zadanej, w kształcie walca. Przedstawione wyniki pokazują, że system omijał przeszkodę o różnym kształcie zarówno dla trajektorii dwu, jak i trójwymiarowej.



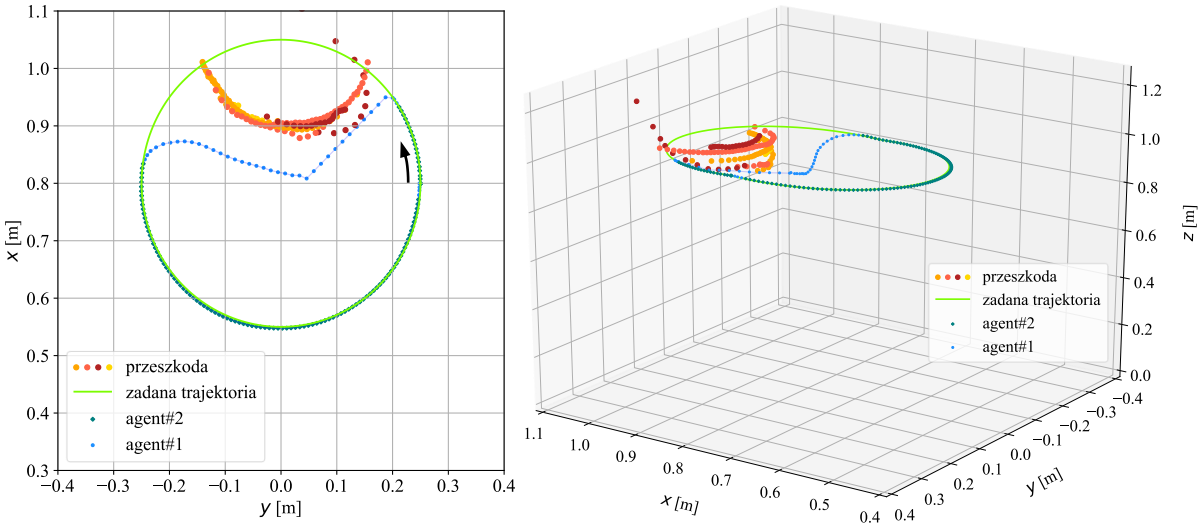
Rys. 83. Trajektoria po jakiej poruszał się TCP robota w badaniach doświadczalnych. Przeszkoda w kształcie prostopadłościanu. Trajektoria zadana okrąg $z = var$. Ruch TCP robota przeciwnie do kierunku ruchu wskazówek zegara



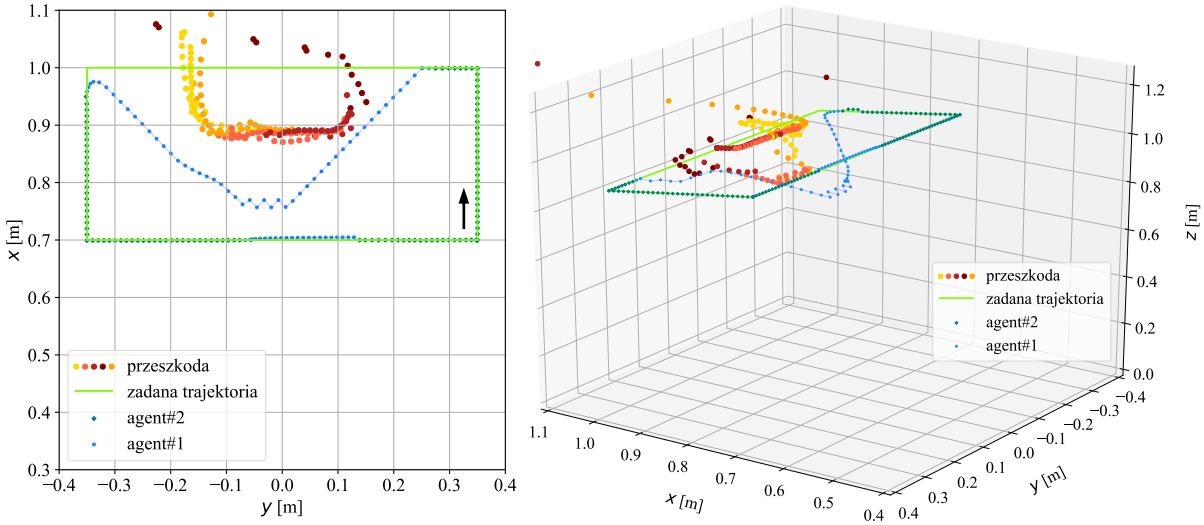
Rys. 84. Trajektoria po jakiej poruszał się TCP robota w badaniach doświadczalnych. Przeszkoda w kształcie walca. Trajektoria zadana okrąg $z = const$. Ruch TCP robota przeciwnie do kierunku ruchu wskazówek zegara. Pozycja przeszkody $y>0m$

Na rysunkach 86, 87, 88 przedstawiono trajektorie ruchu TCP robota dla trajektorii zadanej o kształcie prostokąta, gdy przeszkoda umieszczana w polu roboczym robota miała kształt prostopadłościanu o różnych wymiarach. Prostopadłościan o mniejszym wymiarze (w tabeli 14 oznaczony jako prostopadłościan 2) umieszczono w polu roboczym pod pewnym kątem do osi X, co zostało pokazane na Rys. 88. Jak widać, algorytm bez problemu omijał

zarówno prostopadłościan o większych rozmiarach, jak i mniejszych na trajektorii dwuwymiarowej i trójwymiarowej.

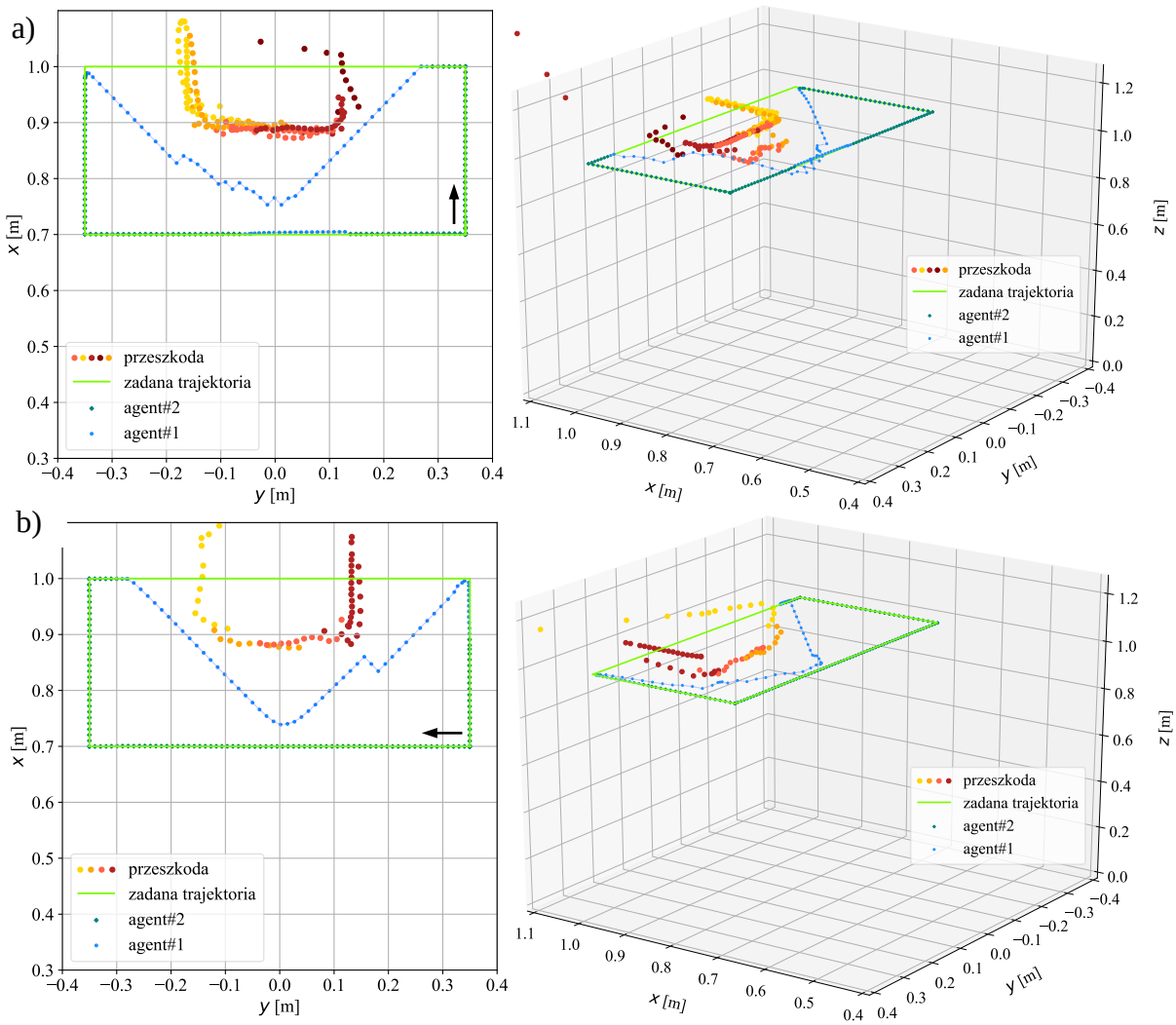


Rys. 85. Trajektoria po jakiej poruszał się TCP robota w badaniach doświadczalnych. Przeszkoda w kształcie walca. Trajektoria zadana okrąg $z = var$. Ruch TCP robota przeciwnie do kierunku ruchu wskazówek zegara

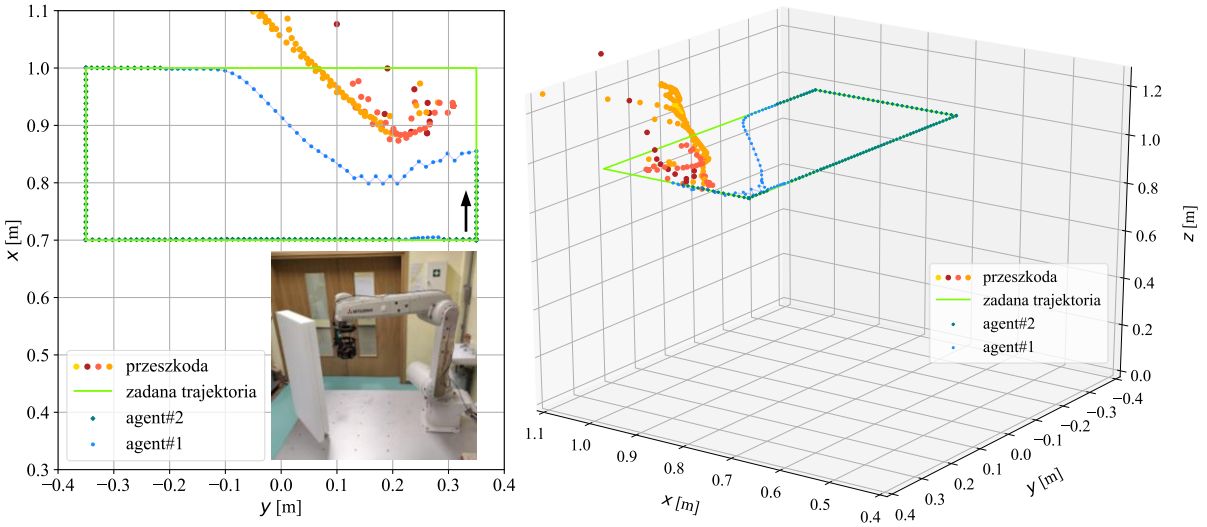


Rys. 86. Trajektoria po jakiej poruszał się TCP robota w badaniach doświadczalnych. Przeszkoda w kształcie prostopadłościanu. Trajektoria zadana prostokąt $z = var$. Ruch TCP robota przeciwnie do kierunku ruchu wskazówek zegara

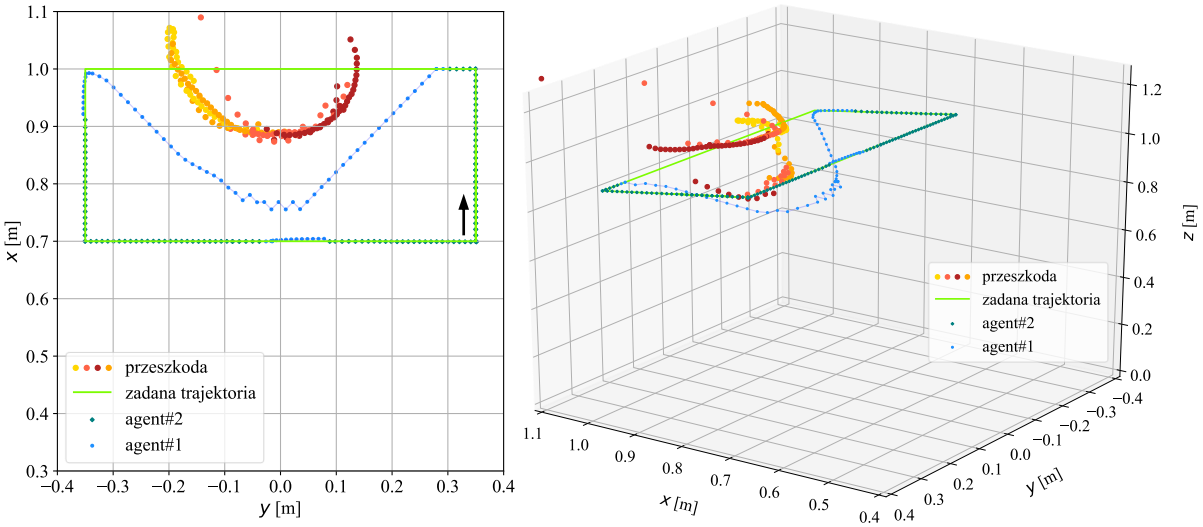
Trajektorie ruchu w przypadku, gdy w polu roboczym została umieszczona przeszkoda w kształcie walca, a TCP robota poruszał się w kierunku *ccw* oraz *cw* po zadanej trajektorii dwu i trójwymiarowej o kształcie prostokąta, zostały przedstawione na rysunkach 89, 90 i 91. Pokazano wyniki dla różnych ustawień przeszkody o kształcie walca. Tak jak w przypadku trajektorii zadanej o kształcie okręgu, algorytm bezkolizyjnie omijał przeszkody, gdy TCP robota poruszał się po zadanej trajektorii.



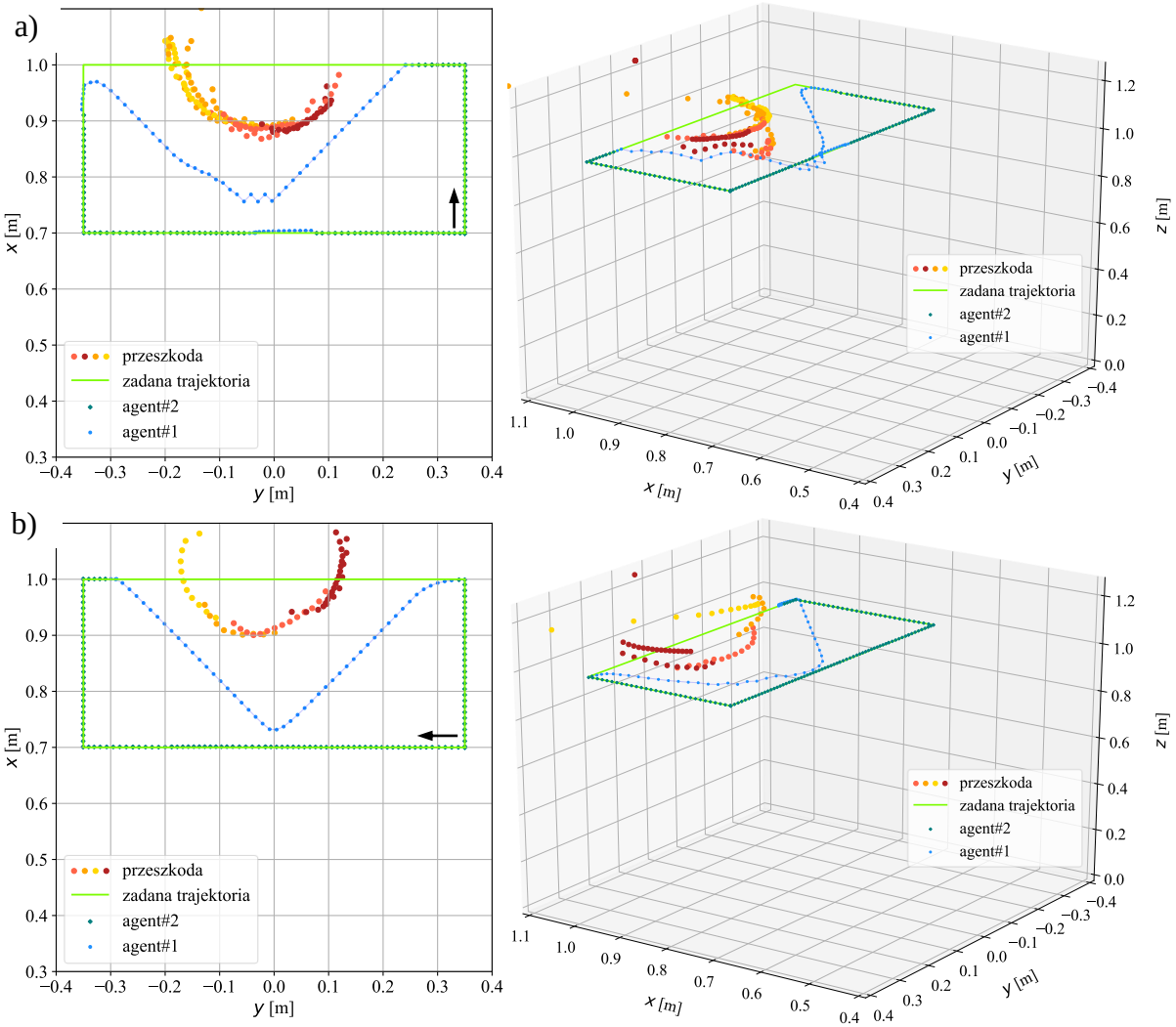
Rys. 87. Trajektoria po jakiej poruszał się TCP robota w badaniach doświadczalnych. Przeszkoda w kształcie prostopadłościanu. Trajektoria zadana prostokąt $z = const$. Ruch TCP robota: a) – przeciwnie do kierunku, b) – zgodnie z kierunkiem ruchu wskazówek zegara



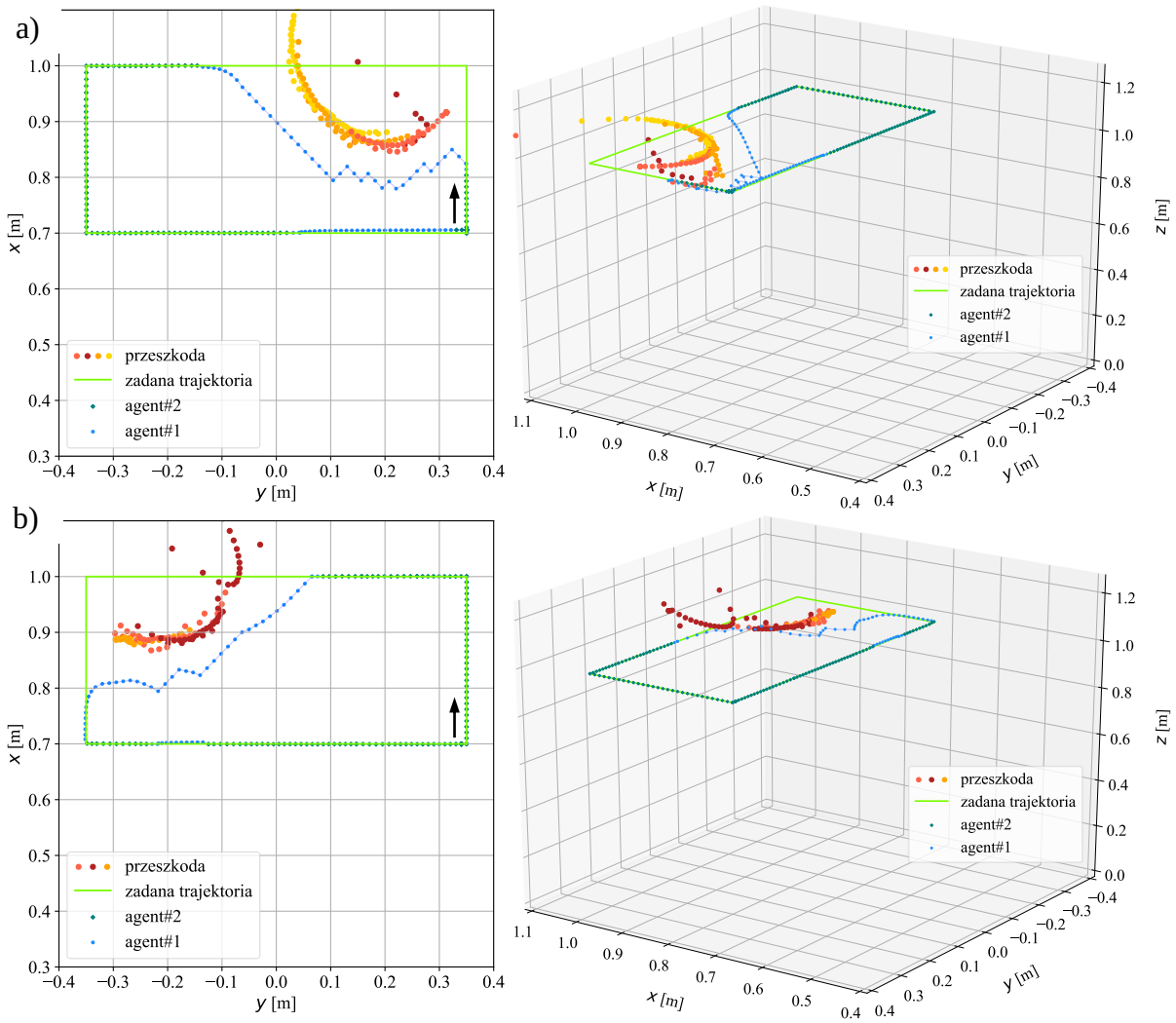
Rys. 88. Trajektoria po jakiej poruszał się TCP robota w badaniach doświadczalnych. Przeszkoda w kształcie prostopadłościanu o mniejszym rozmiarze. Trajektoria zadana prostokąt $z = const$. Ruch TCP robota przeciwnie do kierunku ruchu wskazówek zegara



Rys. 89. Trajektoria po jakiej poruszał się TCP robota w badaniach doświadczalnych. Przeszkoda w kształcie walca. Trajektoria zadana prostokąt z = var. Ruch TCP robota przeciwnie do kierunku ruchu wskazówek zegara

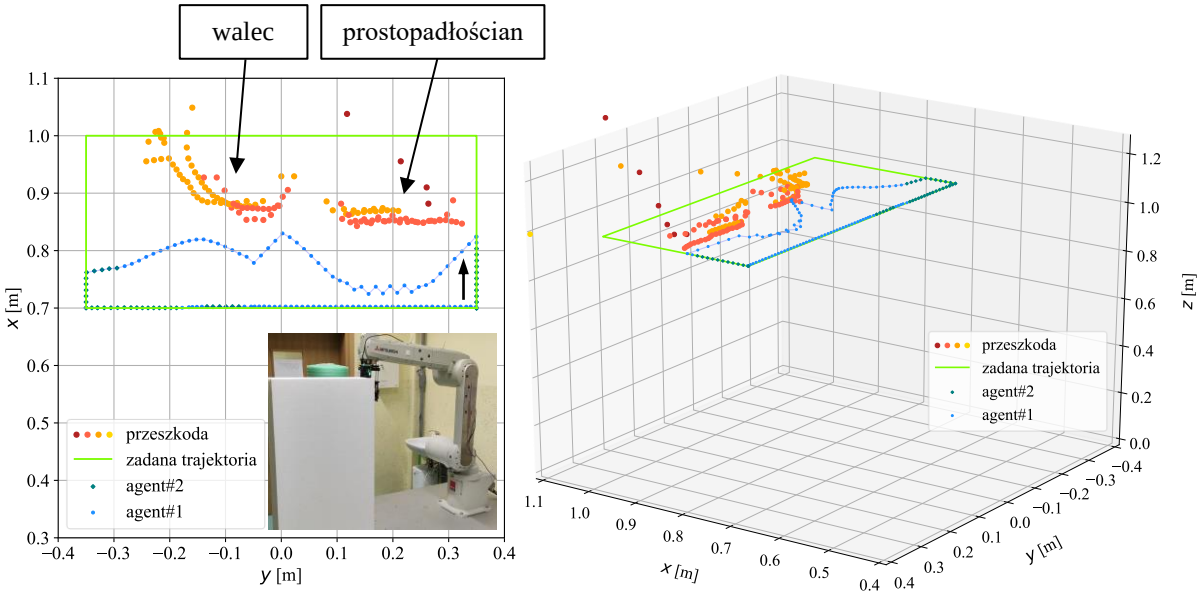


Rys. 90. Trajektoria po jakiej poruszał się TCP robota w badaniach doświadczalnych. Przeszkoda w kształcie walca. Trajektoria zadana prostokąt z = const. Ruch TCP robota: a) – przeciwnie do kierunku, b) – zgodnie z kierunkiem ruchu wskazówek zegara



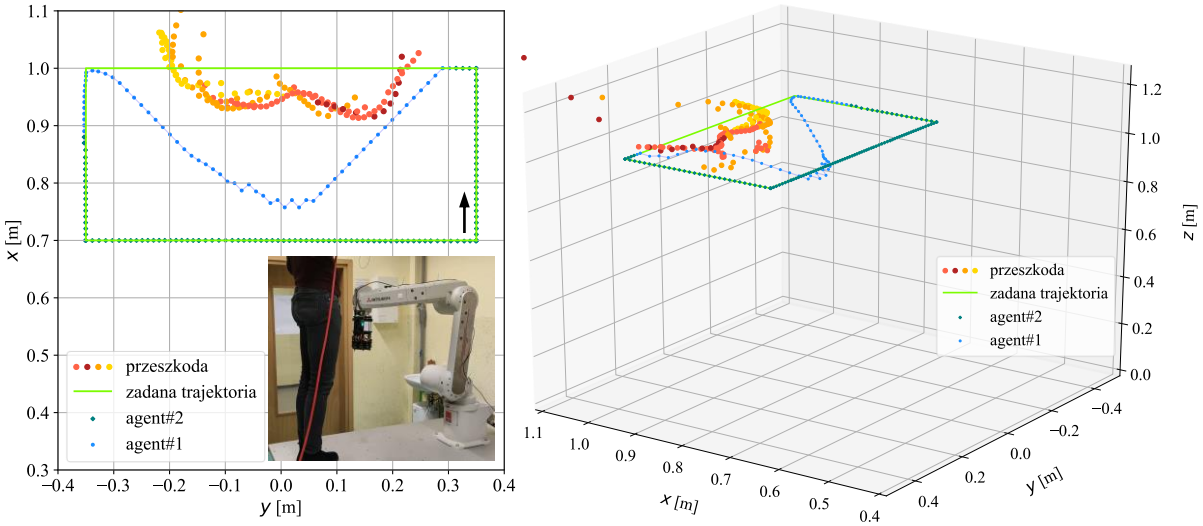
Rys. 91. Trajektoria po jakiej poruszał się TCP robota w badaniach doświadczalnych. Przeszkoda w kształcie walca. Trajektoria zadana prostokąt $z = const$. Położenie przeszkody współrzędna: a) – $y>0m$, b) – $y<0m$. Ruch TCP robota przeciwnie do kierunku ruchu wskazówek zegara

Na rysunku 92 przedstawiono zdjęcie i zarejestrowane przebiegi badań, gdy w polu roboczym umieszczono przeszkodę w kształcie prostopadłościanu, a następnie przeszkodę w kształcie walca. Na początku epizodu TCP robota poruszał się po zadanej trajektorii o kształcie prostokąta. Po kilku krokach, operator umieścił w polu roboczym pierwszą przeszkodę w kształcie prostopadłościanu, którą algorytm bezkolizyjnie ominął. Po manewrze omijania przeszkody w kształcie prostopadłościanu, gdy TCP robota przemieszczał się w kierunku zadanej trajektorii, w polu roboczym została umieszczona druga przeszkoda w kształcie walca. Ta przeszkoda została również bezkolizyjnie ominięta. Można zauważyć, że dla dwóch przeszkód umieszczonych w polu roboczym i testowanej trajektorii, robot sterowany przez wytrenowanego agenta po ominięciu drugiej przeszkody, zmierzał w kierunku prostopadłym do zadanej trajektorii. W innych testowanych przypadkach, gdy w polu roboczym była jedna przeszkoda, to trajektoria ruchu w momencie wykonywania manewru jej omijania miała kształt zbliżony do trapezu, na przykład na Rys. 90.



Rys. 92. Trajektoria po jakiej poruszał się TCP robota w badaniach doświadczalnych. Dwie przeszkody: prostopadłościan i walec. Trajektoria zadana prostokąt $z = const$. Ruch TCP robota przeciwnie do kierunku ruchu wskazówek zegara

Na rysunku 93 przedstawiono trajektorię ruchu, gdy w polu roboczym robota znajdował się człowiek. Taka przeszkoda ma nieregularny kształt, inny od testowanych wcześniej. Jej zarys zarejestrowany przez czujniki w głowicy pokazano na płaszczyźnie *XY*. W takim przypadku algorytm nie miał żadnego problemu z bezkolizyjnym ominięciem osoby, która weszła w strefę roboczą robota. Sama trajektoria ruchu TCP robota dla takiej przeszkody jest bardzo podobna i zbliżona do trapezu, tak jak w przypadku przeszkód o kształcie prostopadłościanu i walca. Można stwierdzić, że algorytm był w stanie omijać przeszkody o różnych nieregularnych kształtach.



Rys. 93. Trajektoria po jakiej poruszał się TCP robota w badaniach doświadczalnych. Przeszkoda – człowiek. Trajektoria zadana prostokąt $z = const$. Ruch TCP robota przeciwnie do kierunku ruchu wskazówek zegara

9.4 Podsumowanie

Na podstawie zaprezentowanych w rozdziale 9 wyników badań można stwierdzić, że zaproponowany system, który wykorzystuje algorytmy uczenia ze wzmocnieniem, do omijania przeszkód na zadanej trajektorii, działał poprawnie i omijał bezkolizyjnie przeszkody oraz podążał po różnych zadanych trajektoriach ruchu. W przypadku samego poruszania się po trajektorii średni uchyb, czyli różnica pomiędzy trajektorią ruchu TCP robota, a trajektorią zadaną wahał się w zakresie od około 1.5mm do 2.5mm, w zależności od kształtu trajektorii i kierunku ruchu robota. Maksymalne błędy nie przekraczały 6mm. W przypadku implementacji algorytmu w warunkach laboratoryjnych pojawiły się nieregularne ruchy po trójkącie, jednak ich maksymalna wartość międzyszczytowa nie była duża w porównaniu z odchyłką od zadanej trajektorii i wynosiła maksymalnie 8mm. **Zastosowanie wytrenowanych w środowisku symulacyjnym algorytmów uczenia ze wzmocnieniem oraz ich adaptacja w trakcie badań doświadczalnych w zadaniu, jakim było omijanie przeszkód na zadanej trajektorii ruchu pozwoliło na osiągniecie celu cząstkowego numer 6.**

W testach wykonanych w celu sprawdzenia możliwości algorytmu w zadaniu jakim było omijanie przeszkód na zadanej trajektorii, przeprowadzono szereg badań zarówno dla przeszkód o różnym kształcie, jak i dla rozmieszczenia ich w różnej pozycji. Na podstawie zarejestrowanych wyników badań można stwierdzić, że zaproponowany algorytm omijał bezkolizyjnie przeszkody. System nie był w stanie ominąć przeszkody tylko w sytuacji, jeśli była ona umieszczona w polu roboczym, zbyt blisko TCP robota i głowicy do wykrywania przeszkód, tj. około 0.05m. W takiej sytuacji agent sterujący robotem zatrzymywał jego pracę lub TCP robota wykonywał nieregularne ruchy w postaci skoków. Agent próbował ominąć przeszkodę wykonując określoną akcję, ale okazywało się, że jest zbyt blisko przeszkody. Dlatego jego kolejną akcją było odsunięcie się od przeszkody. Taki ruch powtarzał się, aż do momentu zakończenia epizodu lub usunięcia przeszkody ze strefy roboczej robota. Łącznie w trakcie badań doświadczalnych przeprowadzonych w laboratorium robot wykonał prawie 100 bezkolizyjnych manewrów omijania przeszkód na zadanej trajektorii jego ruchu.

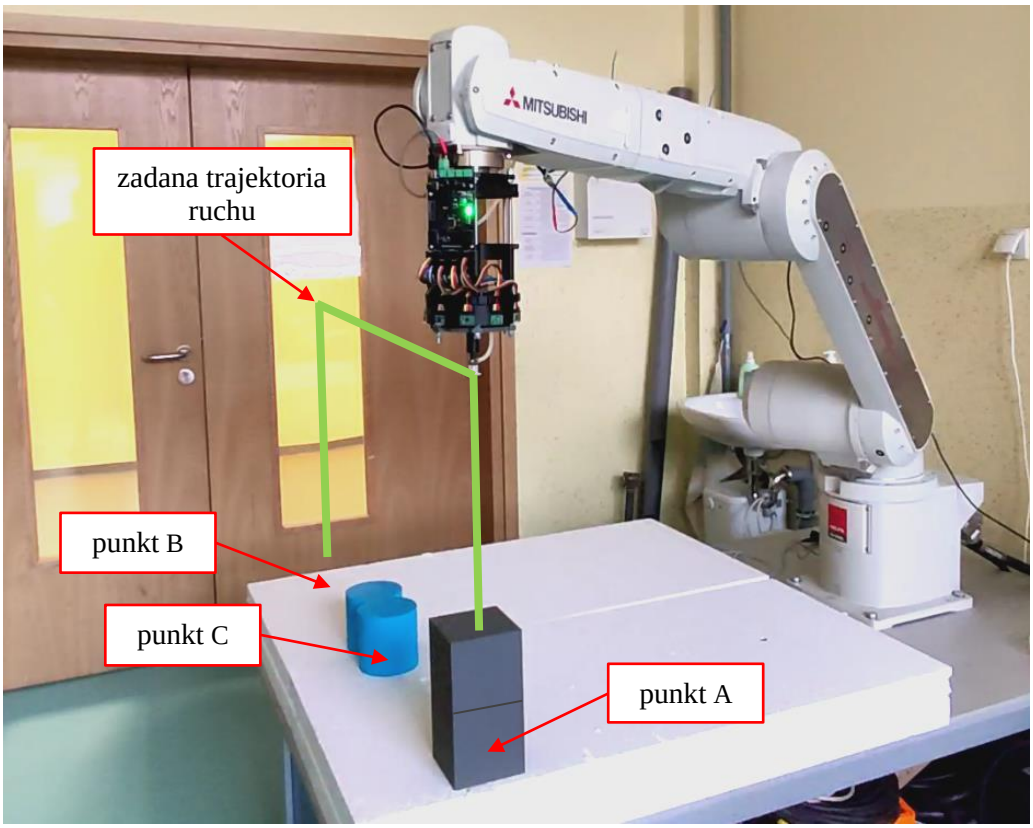
10 Współpraca robota przemysłowego z człowiekiem

10.1 Opis stanowiska

Rys. 94. Przyssawka przymocowana do głowicy wykrywającej przeszkody

Zaprojektowany i przetestowany w laboratorium, system do omijania przeszkód na zadanej trajektorii ruchu, został zastosowany do sterowania robotem przemysłowym współpracującym z człowiekiem. Do tego celu zaprojektowano stanowisko laboratoryjne, w którym robot pracował w tej samej strefie roboczej z człowiekiem, wykonując zadanie *pick-and-place*. Do głowicy wykrywającej przeszkody została zamontowana przyssawka podciśnieniowa przedstawiona na Rys. 94. Była ona uruchamiana ze sterownika głowicy.

Zbudowane stanowisko do współpracy robota z człowiekiem zostało przedstawione na Rys. 95. Zaznaczono na nim trzy punkty A, B, C symbolizujące stanowiska, w których operuje człowiek i robot oraz prostopadłościenne i walcowe elementy w dwóch różnych kolorach. W postawionym zadaniu, prostopadłościany i walce miały być przenoszone i



Rys. 95. Stanowisko do współpracy człowieka z robotem przemysłowym

ustawiane przez robota i człowieka. Zadaniem robota było przeniesienie dwóch szarych prostopadłościanów z punktu A do B. Zaplanowano, że będzie się on poruszać po zadanej trajektorii, narysowanej na Rys. 95 kolorem zielonym. Zadaniem człowieka było manipulowanie niebieskim walcami umiejscowionymi w punkcie C. Robot, żeby przemieścić się z punktu A do B lub B do A musi bezkolizyjnie ominąć człowieka, który może wykonywać swoje zadanie w okolicy punktu C. W przypadku, gdy człowiek odszedł od swojego stanowiska, robot powinien poruszać się po zadanej trajektorii.

Do poprawnego sterowania robotem na stanowisku, w którym miał on współpracować z człowiekiem został zastosowany system wieloagentowy przedstawiony w rozdziale 8.1 na Rys. 54, który był stosowany w badaniach opisanych w rozdziałach 8 i 9. Za omijanie przeszkód odpowiedzialny był agent#1, a za podążanie po zadanej trajektorii ruchu agent#2. Zadana trajektoria (Rys. 95), po której poruszał się wytrenowany agent składała się z trzech odcinków. Sam agent nie był odpowiedzialny za dokładne pozycjonowanie TCP robota nad prostopadłościennymi elementami oraz za ich chwycenie lub puszczenie. Do tych zadań został użyty dedykowany kontroler robota. Gdy TCP robota był blisko pozycji chwycenia lub puszczenia, wysyłana była komenda z komputera sterującego do kontrolera robota ze współrzędnymi pozycji, do której robot miał się przemieścić i chwycić albo puścić prostopadłościenny element. Dojazd do pozycji chwycenia lub puszczenia obiektu był wykonywany z dokładnością, jaką zapewnia producent robota. Współrzędne punktów, w jakich robot miał chwytać lub puszczać prostopadłościan były zapisane w programie sterującym jego pracą.

Przedstawione w rozdziale 10 badania zostały przeprowadzone, gdy przeszkodą w polu roboczym był człowiek. Dlatego przed ich rozpoczęciem zastosowano należyte środki bezpieczeństwa oraz wykonano odpowiednie testy. W celu oszacowania poziomu niezawodności zaproponowanego systemu sterowania w trakcie badań przedstawionych w rozdziałach 8 i 9 przeprowadzono ponad 200 udanych i bezkolizyjnych manewrów omijania różnych przeszkód znajdujących się w polu roboczym robota. W trakcie testów zaprojektowanego systemu przedstawionych w rozdziale 10, prędkość robota przemysłowego została ograniczona do wartości $25\frac{mm}{s}$ w:

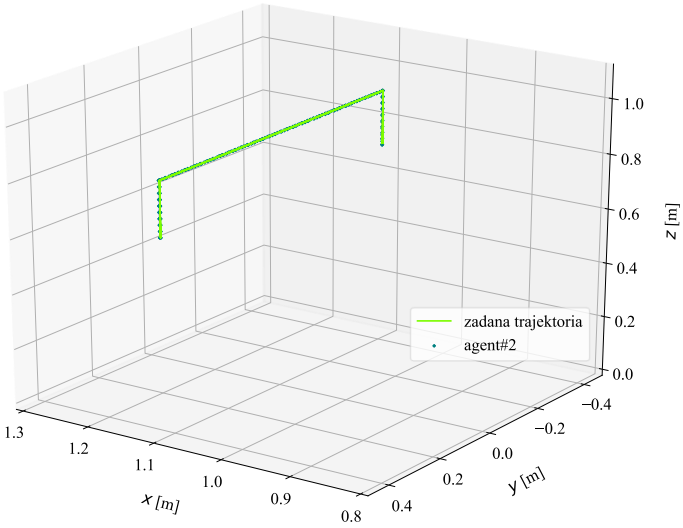
- kontrolerze robota,
- programie odbierającym dane z komputera sterującego w kontrolerze robota,
- programie sterującym i komunikującym się z robotem w komputerze sterującym,
- algorytmie sterującym i generującym akcje.

Dodatkowo, drugi operator nieustannie kontrolował pracę robota i przez cały czas trwania ruchu trzymał rękę na przycisku bezpieczeństwa.

10.2 Wyniki badań doświadczalnych

Poniżej opisano wykonane badania doświadczalne algorytmu sterowania robotem w trakcie przenoszenia elementów w strefie, w której mógł równocześnie pracować człowiek. Przedstawiono trzy przykładowe sekwencje ruchu robota, który przekładał dwa prostopadłościenne elementy z punktu A do punktu B. Na wykresach i zdjęciach zamieszczonych poniżej, zostało przedstawione działanie systemu w tych trzech przypadkach:

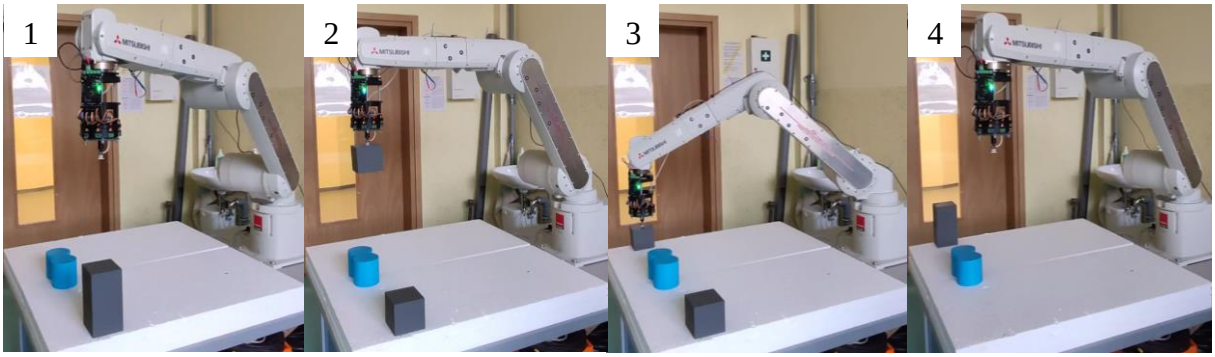
- brak człowieka w strefie roboczej robota,



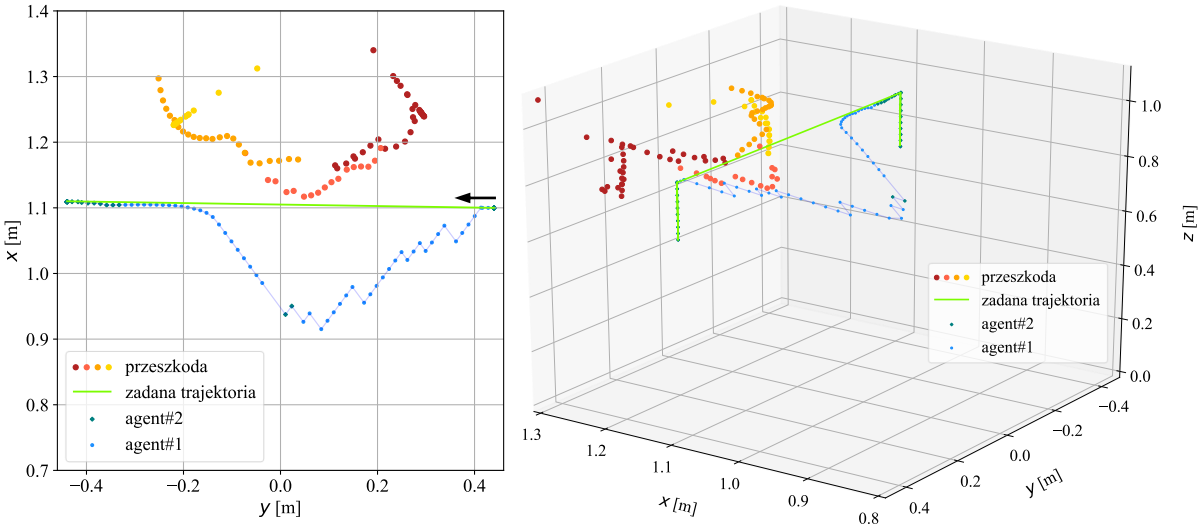
Rys. 96. Trajektoria ruchu TCP robota w przypadku, gdy w polu roboczym nie było osoby współpracującej z robotem. Pierwsza sekwencja

- człowiek w strefie roboczej robota,
- człowiek w bardzo bliskiej odległości od robota.

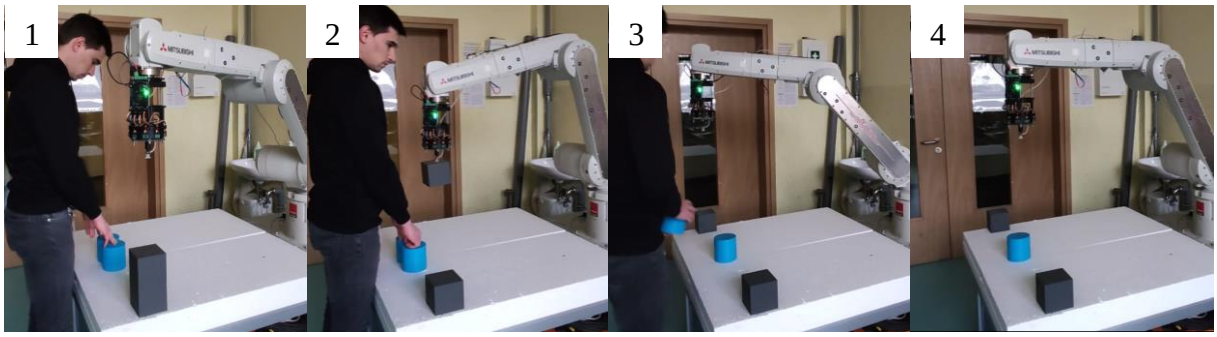
Dla pierwszej sekwencji ruchów, w której człowieka nie było w strefie roboczej, robot mógł bez przeszkód poruszać się po zadanej trajektorii ruchu i przenosić prostopadłościany z punktu A do punktu B. Przykład trajektorii ruchu, po której robot przeniósł obydwa elementy pokazano na Rys. 96. Zdjęcia robota w trakcie ruchu, gdy w strefie roboczej nie było człowieka przedstawiono na Rys. 97.



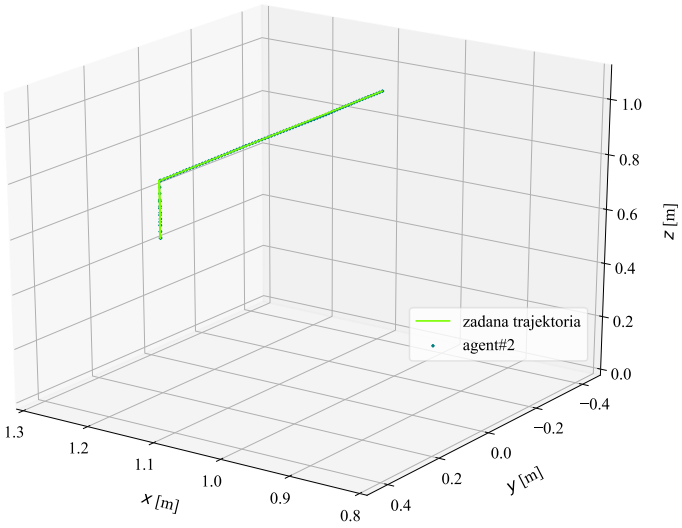
Rys. 97. Zdjęcia robota w trakcie przenoszenia przedmiotów, gdy w polu roboczym nie było człowieka



Rys. 98. Trajektoria ruchu TCP robota w przypadku, gdy w polu roboczym był człowiek. Ruch z punktu A do B. Pierwsze przejście dla drugiej sekwencji

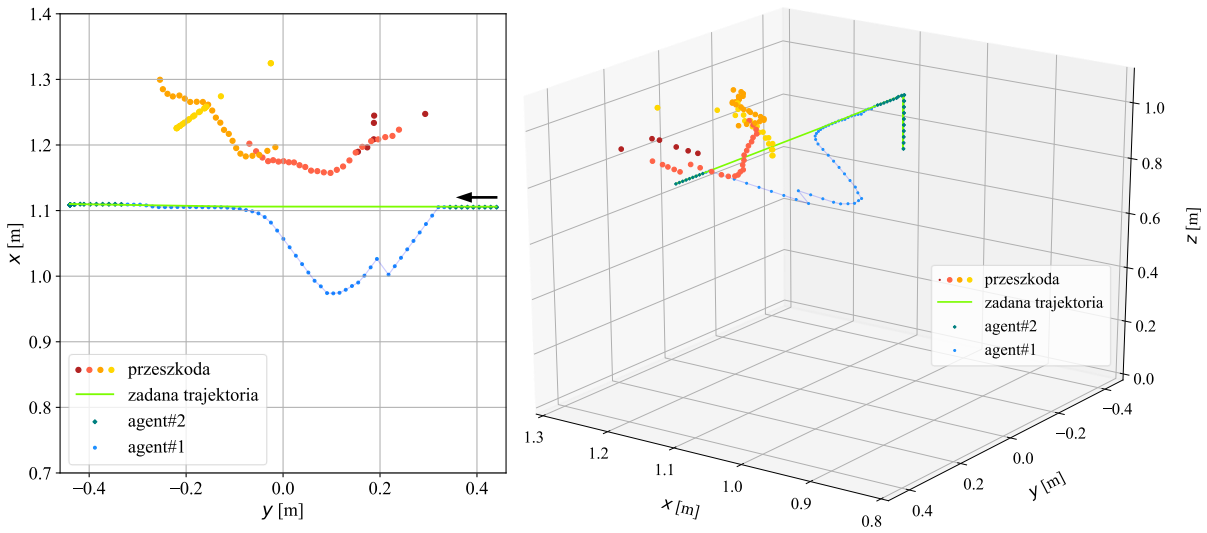


Rys. 99. Wizualizacja ruchu robota w przypadku, gdy w polu roboczym był człowiek. Ruch z punktu A do B. Pierwsze i drugie przejście dla drugiej sekwencji



Rys. 100. Trajektoria ruchu TCP robota w przypadku, gdy w polu roboczym nie było człowieka. Ruch z punktu B do A. Drugie przejście dla drugiej sekwencji

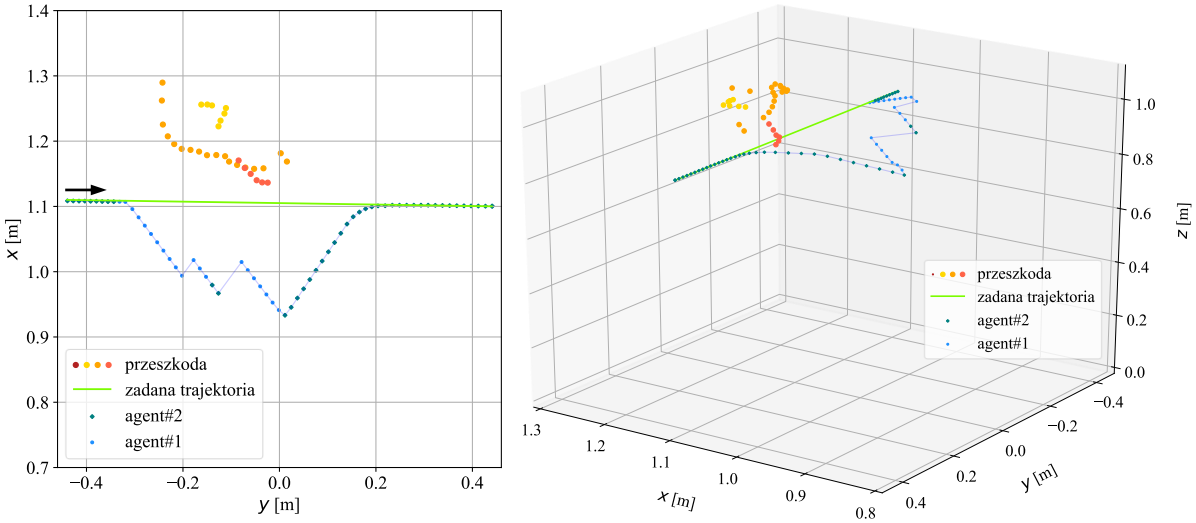
Na przebiegach przedstawionych na rysunkach 98, 100, 101, 102 pokazano trajektorie, po których poruszał się TCP robota dla drugiej przykładowej sekwencji. Ta sekwencja ruchu została podzielona na cztery przejścia. Pierwsze przejście składało się z trzech operacji: chwycenie pierwszego prostopadłościanu w punkcie A, przemieszczenie TCP robota do punktu B, puszczenie pierwszego elementu w punkcie B. Drugie przejście to przemieszczenie TCP do punktu A oraz chwycenie drugiego prostopadłościanu. Trzecie przejście to przemieszczenie TCP do punktu B oraz puszczenie drugiego



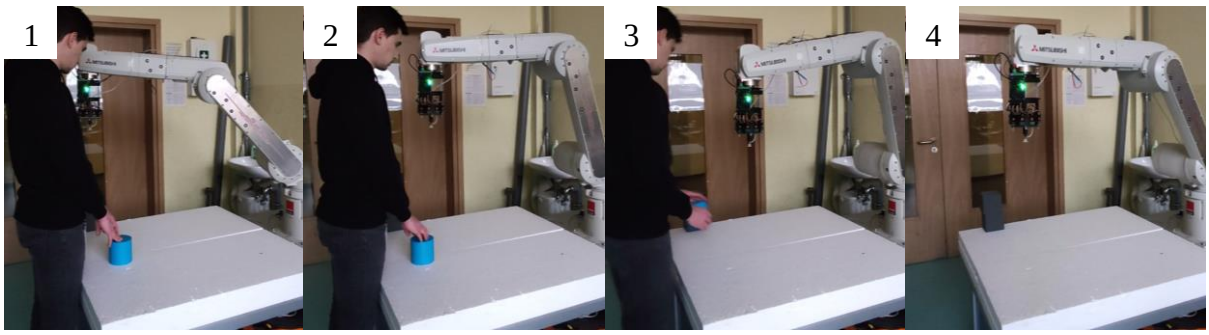
Rys. 101. Trajektoria ruchu TCP robota w przypadku, gdy w polu roboczym był człowiek. Ruch z punktu A do B. Trzecie przejście dla drugiej sekwencji

obiektu. Ostatnie przejście to powrót TCP robota do punktu A.

Na rysunku 98 przedstawiono trajektorię dla pierwszego przejścia i przeniesienia pierwszego prostopadłościanu z punktu A do B. W trakcie tego ruchu w polu roboczym obecny był człowiek, stojący cały czas przy stanowisku w punkcie C do momentu osiągnięcia punktu B przez robota. Zdjęcia pokazujące pozycje robota w czterech chwilach czasowych pokazano na Rys. 99. Na trzecim zdjęciu (Rys. 99) widać, że człowiek odchodzi od punktu C w momencie, gdy TCP robota znajduje się w punkcie B. W trakcie ruchu powrotnego robota do punktu A, czyli podczas drugiego przejścia, w polu roboczym nie było osoby współpracującej z robotem, co zostało przedstawione na ostatnim zdjęciu na Rys. 99. Trajektoria ruchu dla tego przejścia została przedstawiona na Rys. 100.



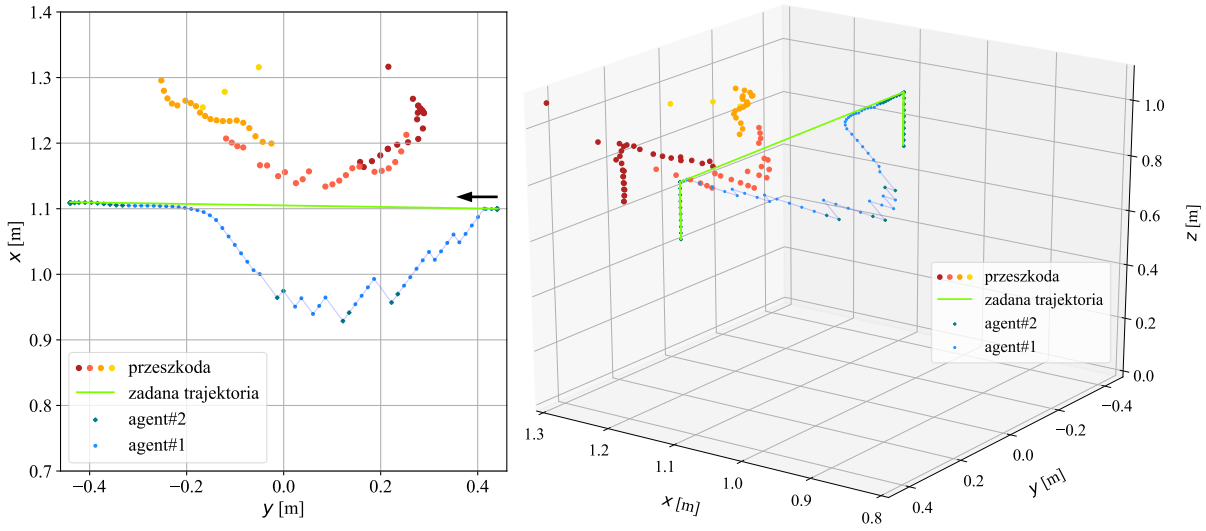
Rys. 102. Trajektoria ruchu TCP robota w przypadku, gdy w polu roboczym był człowiek, który odszedł od stanowiska w trakcie manewru omijania. Ruch z punktu B do A. Czwarte przejście dla drugiej sekwencji



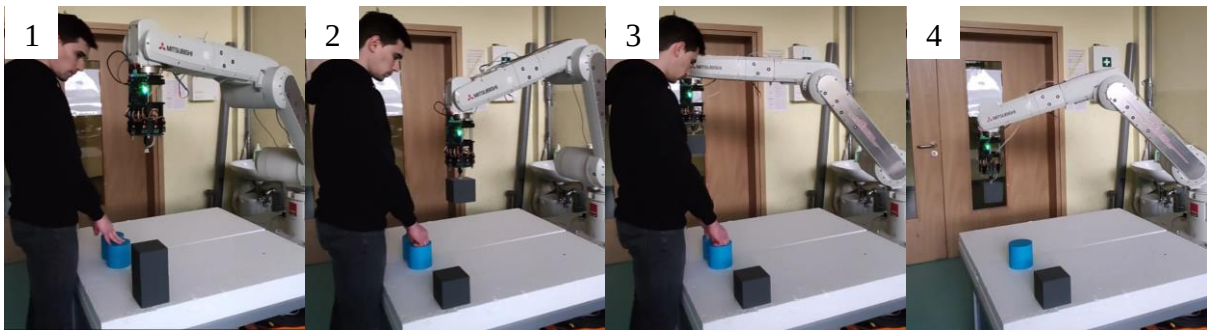
Rys. 103. Wizualizacja ruchu robota w przypadku, gdy w polu roboczym był człowiek, który odszedł od stanowiska w trakcie manewru omijania. Ruch z punktu B do A. Czwarte przejście dla drugiej sekwencji

W przypadku trzeciego przejścia robota człowiek wrócił do swojego stanowiska w punkcie C. Trajektoria, po której poruszał się TCP robota została przedstawiona na Rys. 101. Przejście trzecie jest bardzo podobne do przejścia pierwszego, w którym w trakcie całego ruchu robota z punktu A do B, w polu roboczym był człowiek. Człowiek stał przy stanowisku w punkcie C aż do czwartego przejścia. W trakcie tego przejścia robot przemieszczał się z punktu B do A, a człowiek odszedł od stanowiska C w trakcie manewru omijania go przez

robota. W tym przypadku robot wrócił do podążania po zadanej trajektorii ruchu i poruszał się w kierunku punktu A. Trajektoria ruchu TCP robota dla czwartego przejścia została przedstawiona na Rys. 102. Widać, że w pierwszej fazie ruchu robota, nie wykryto przeszkody i aktywny był agent#2. W chwili wykrycia człowieka przez głowicę jako przeszkody, co zaznaczono za pomocą kropek na rysunku, robotem zaczął sterować agent#1, który rozpoczął proces omijania. W jego trakcie człowiek odszedł od stanowiska, co zostało wykryte i w rezultacie do punktu A robotem sterował agent#2. Wizualizacja ruchu robota została przedstawiona na zdjęciach (Rys. 103).



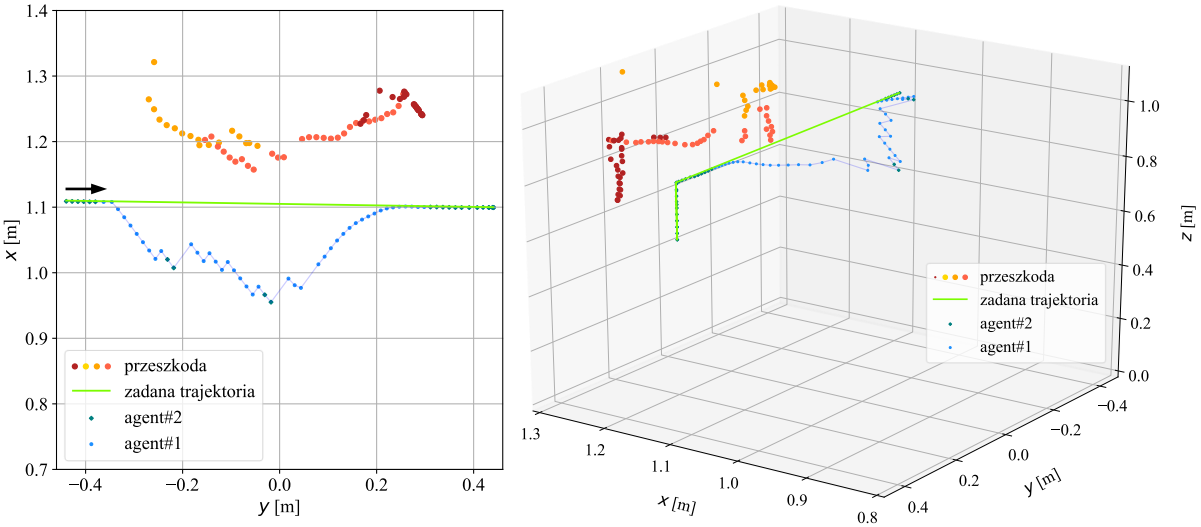
Rys. 104. Trajektoria ruchu TCP robota w przypadku, gdy w polu roboczym był człowiek. Ruch z punktu A do B. Pierwsze przejście dla trzeciej sekwencji



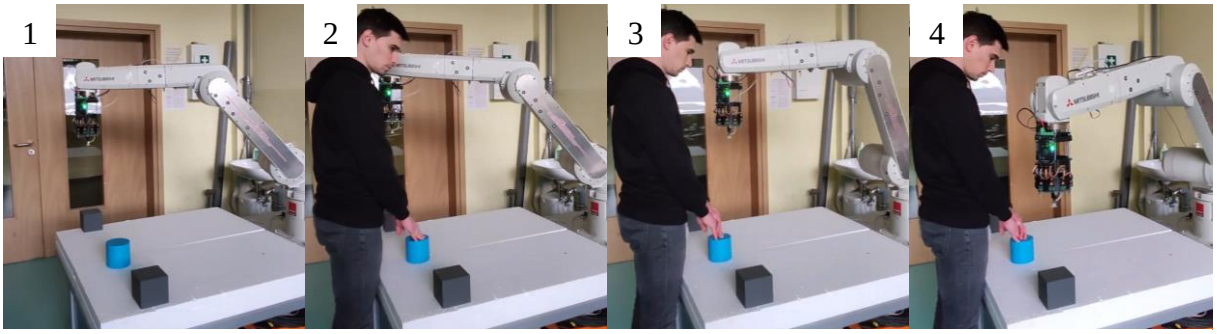
Rys. 105. Wizualizacja ruchu robota w przypadku, gdy w polu roboczym był człowiek. Ruch robota z punktu A do B. Pierwsze przejście dla trzeciej sekwencji.

Na przebiegach zaprezentowanych na rysunkach 104, 106, 108 oraz 110 przedstawiono trajektorie, po których poruszał się TCP robota w trakcie trzeciej przykładowej sekwencji. Tak jak w przypadku sekwencji drugiej, występują takie same cztery przejścia, które zostały przedstawione na czterech rysunkach. Trajektoria przedstawiona na Rys. 104 ilustruje podobną sytuację jak w pierwszym przejściu z sekwencji drugiej przedstawionej na Rys. 98. Osoba współpracująca z robotem znajduje się w polu roboczym robota, który przenosi pierwszy prostopadłościan z punktu A do B. W momencie, kiedy robot znajduje się w punkcie B, człowiek odszedł od stanowiska C. W drugim przejściu przedstawionym na Rys. 106, robot przemieszczał się z punktu B do A, a człowiek wrócił do stanowiska C. Robot

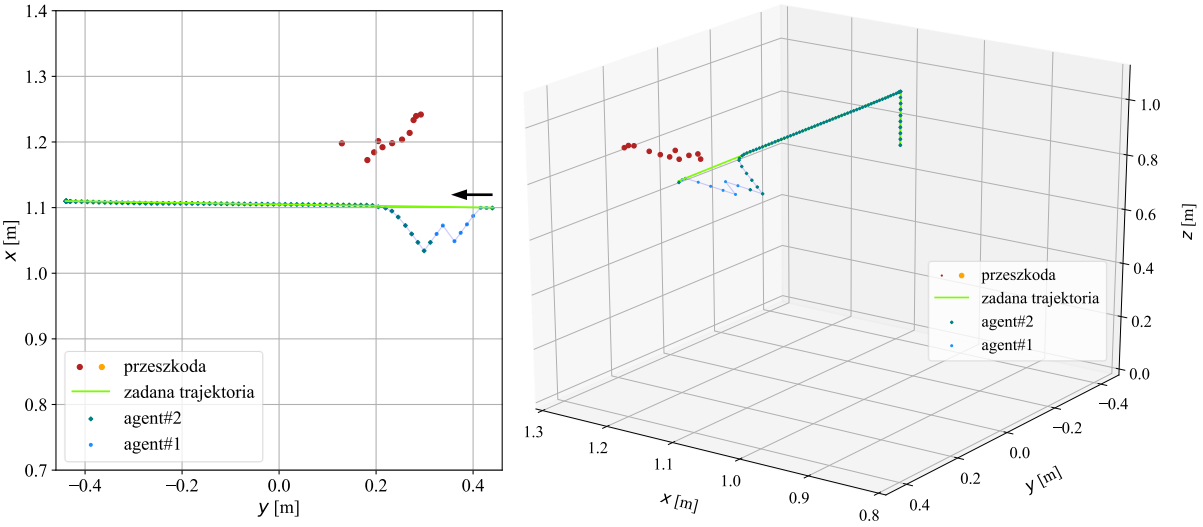
wykonał manewr omijania osoby i pobrał kolejny element. Wizualizacje tych dwóch przejść zostały przedstawione kolejno na Rys. 105 oraz Rys. 107.



Rys. 106. Trajektoria ruchu TCP robota w przypadku, gdy w polu roboczym był człowiek. Ruch z punktu B do A. Drugie przejście dla trzeciej sekwencji



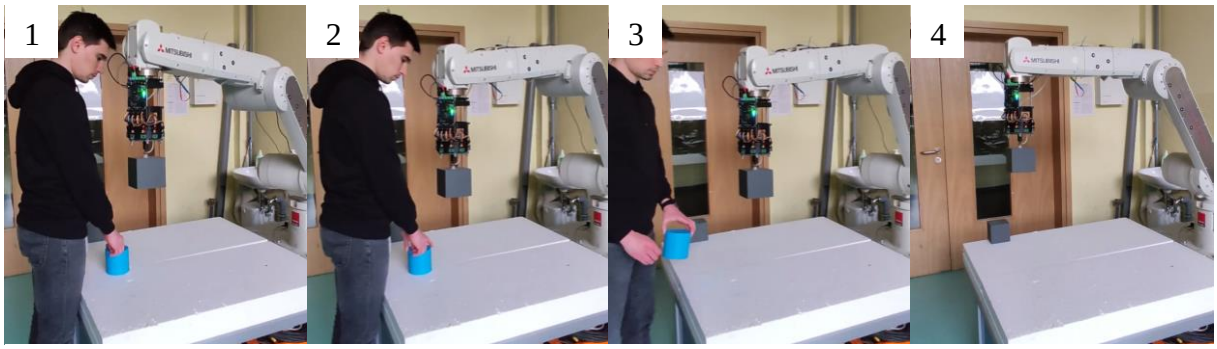
Rys. 107. Wizualizacja ruchu robota w przypadku, gdy w polu roboczym był człowiek. Ruch robota z punktu B do A. Drugie przejście dla trzeciej sekwencji.



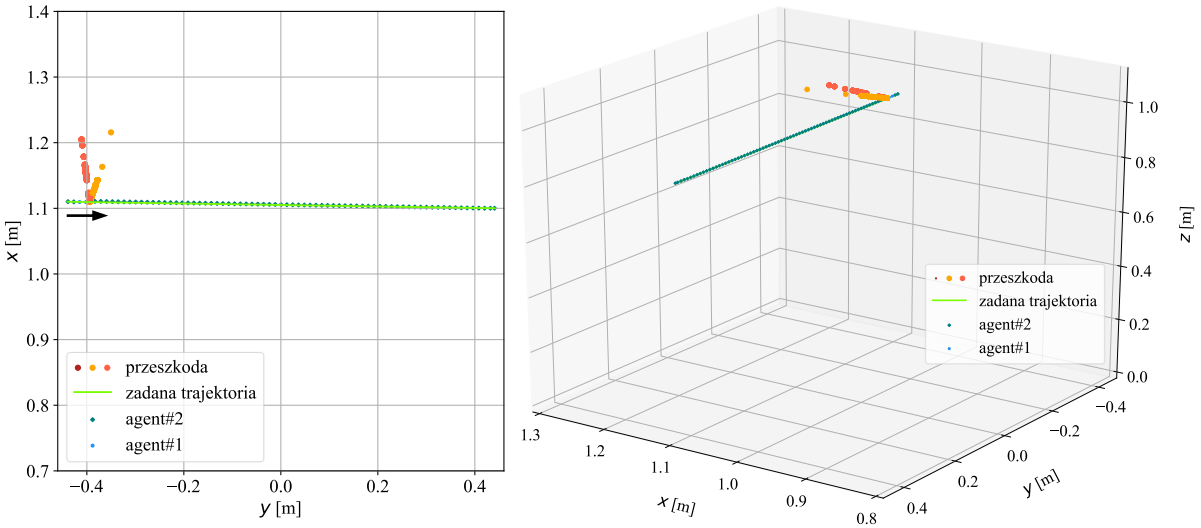
Rys. 108. Trajektoria ruchu TCP robota w przypadku, gdy w polu roboczym był człowiek, który odszedł od stanowiska w trakcie manewru omijania. Ruch z punktu A do B. Trzecie przejście dla trzeciej sekwencji

W trzecim przejściu robot przenosił drugi prostopadłościan z punktu A do B. Człowiek był obecny przy stanowisku C tylko w początkowej fazie wykonywania manewru omijania przez robota. Po kilku krokach wykonanych przez agenta#1, człowiek odszedł od stanowiska w punkcie C, a robot wrócił do podążania po zadanej trajektorii ruchu i przemieszcza się po linii prostej w kierunku punktu B. Zarejestrowana trajektoria ruchu z tego przejścia została przedstawiona na Rys. 108.

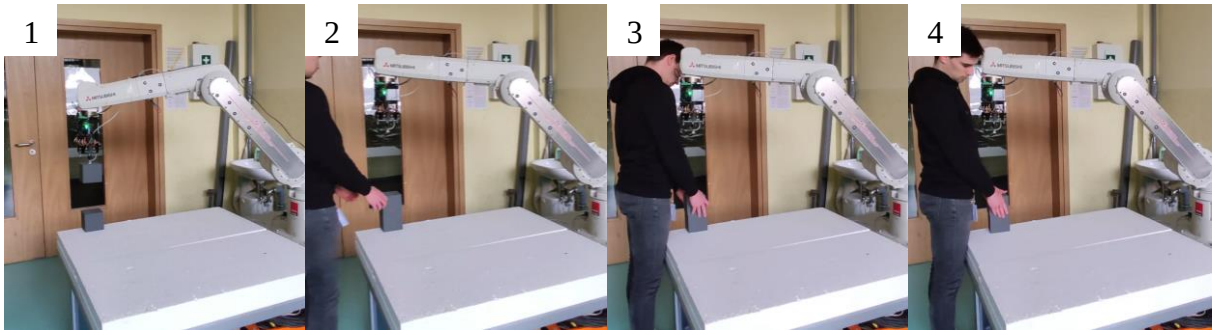
Ostatnie przejście trzeciej sekwencji zostało przedstawione na Rys. 110. Ilustruje ono ruch powrotny robota z punktu B do A. W trakcie tego ruchu człowiek poszedł do punktu B, w którym robot puścił ostatni prostopadłościan. Następnie, człowiek chwycił prostopadłościenne elementy, znajdując się w odległości kilku centymetrów od robota. W takim przypadku algorytm sterujący pracą robota zatrzymał go, aby zapewnić bezpieczeństwo człowiekowi. Bezpieczna odległość, po przekroczeniu, której następuje zatrzymanie robota została ustawiona na wartość 5cm. Robot kontynuuje ruch dopiero wtedy, gdy człowiek oddali się dostatecznie daleko. Zdjęcia ilustrujące tę sekwencję, zostały przedstawione na Rys. 111.



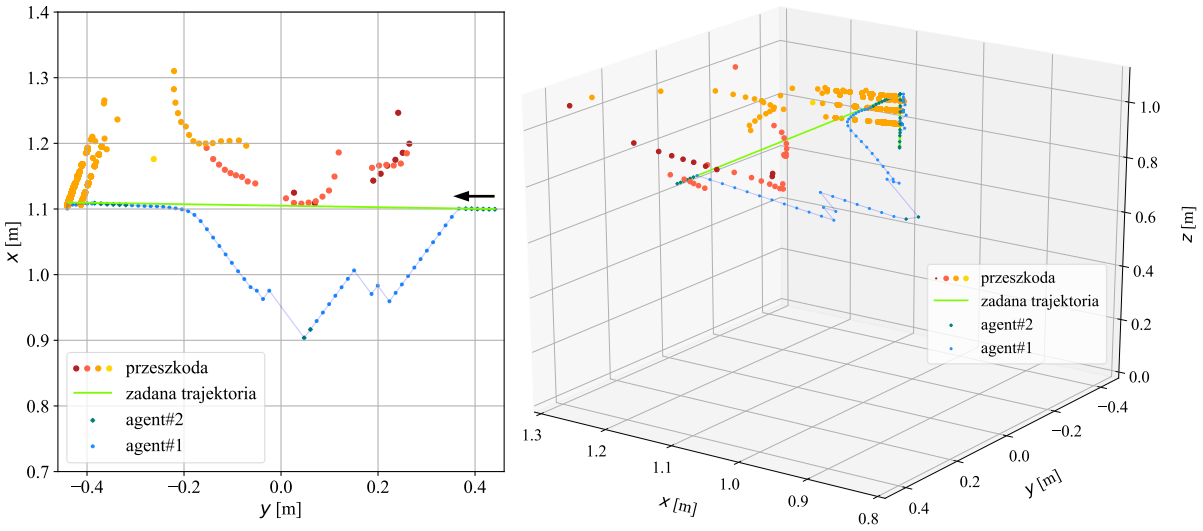
Rys. 109. Wizualizacja ruchu robota w przypadku, gdy w polu roboczym był człowiek, który odszedł od stanowiska w trakcie manewru omijania. Ruch robota z punktu A do B. Trzecie przejście dla trzeciej sekwencji.



Rys. 110. Trajektoria ruchu TCP robota w przypadku, gdy człowiek podchodzi na odległość kilku centymetrów od robota. Ruch z punktu B do A. Czwarte przejście dla trzeciej sekwencji

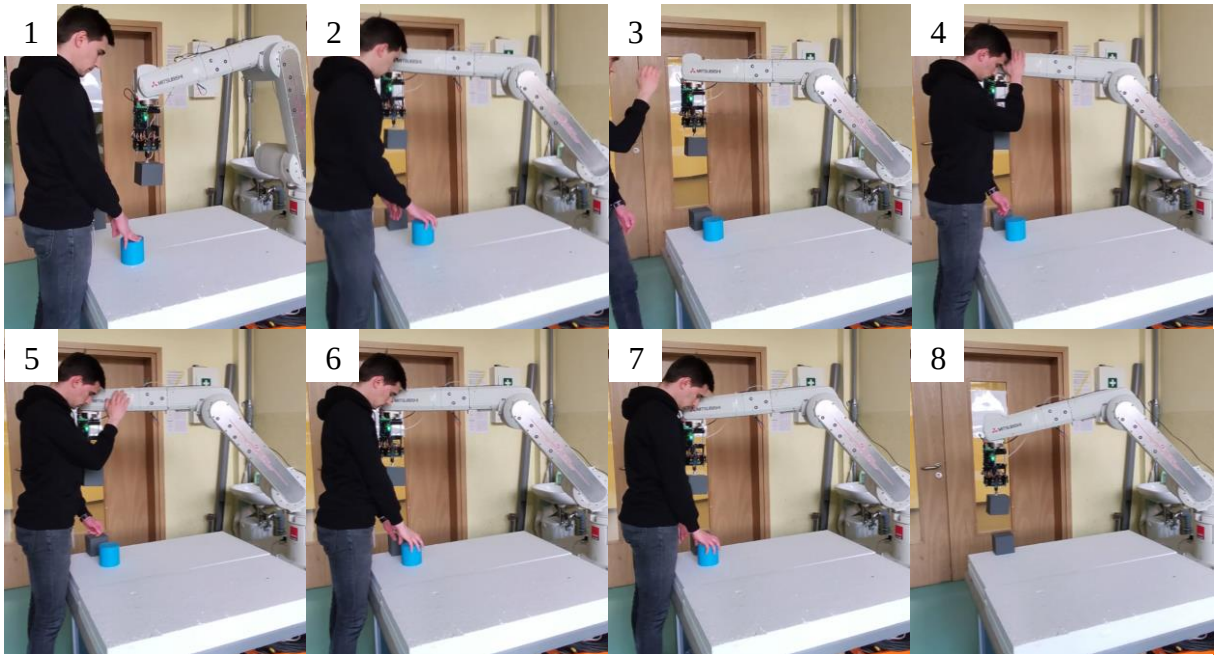


Rys. 111. Wizualizacja ruchu robota w przypadku, gdy człowiek podchodzi na odległość kilku centymetrów od robota. Ruch robota z punktu B do A. Czwarte przejście dla trzeciej sekwencji.



Rys. 112. Trajektoria ruchu TCP robota w przypadku, gdy człowiek kilkakrotnie podchodzi na odległość kilku centymetrów od robota. Ruch z punktu A do B. Trzecie przejście dla czwartej sekwencji

Dodatkowy wykres przedstawiający zachowanie robota w momencie, kiedy w jego bardzo bliskim otoczeniu, to znaczy kilku centymetrów, znajdowała się przeszkoda, czyli w tym przypadku człowiek, przedstawiono na Rys. 112. Jest to trzecie przejście z czwartej, dodatkowej sekwencji, w której robot przenosił drugi prostopadłościan z punktu A do B, a człowiek znajdował się w punkcie C. W momencie, gdy robot znajdował się w pobliżu punktu B, człowiek poszedł do robota na odległość kilku centymetrów, co spowodowało zatrzymanie jego pracy. Następnie, osoba kilkakrotnie podeszła i następnie odeszła od robota. Za każdy razem, gdy człowiek odszedł, robot kontynuował pracę, a gdy podszedł ponownie, to zatrzymywał się. Na rysunku 112 (wykres 3D) widać, że w momencie, kiedy TCP robota znajduje się w pobliżu punktu B, to wykryta przeszkoda w formie pomarańczowych kropek zmienia położenie w osi Z. Pokazuje to, że robot przemieszcza się w dół, wzdłuż osi Z w momencie, kiedy człowieka nie ma w pobliżu. Kiedy człowiek się pojawia, to jest wykrywany, ale za każdym podejściem na innej wysokości. Wizualizacja ruchu robota za pomocą zdjęć, gdy człowiek kilkakrotnie podchodzi na odległość kilku centymetrów od robota została przedstawiona na Rys. 113.



Rys. 113. Wizualizacja ruchu robota w przypadku, gdy człowiek kilkakrotnie podchodzi na odległość kilku centymetrów od robota. Ruch robota z punktu A do B. Trzecie przejście dla czwartek sekwencji.

Przeprowadzone badania oraz testy przedstawione w rozdziale 10 **potwierdzają osiągniecie celu cząstkowego numer 7, który zakładał opracowanie systemu sterującego pracą robota przemysłowego, pozwalającego na jednoczesną pracę człowieka i robota we wspólnej strefie roboczej.**

11 Podsumowanie

W pracy podjęto problematykę sterowania robotem przemysłowym w warunkach jego pracy w bezpośredniej bliskości z człowiekiem. Zaproponowano zastosowanie gradientowych algorytmów uczenia ze wzmocnieniem do sterowania robotem przemysłowym mogącym omijać przeszkody na zadanej trajektorii ruchu. Algorytmy te, bazowały na zastosowaniu sztucznych sieci neuronowych i metod sztucznej inteligencji. Należy zaznaczyć, że zastosowanie gradientowych algorytmów uczenia ze wzmocnieniem do sterowania robotem współpracującym z człowiekiem oraz omijającym bezkolizyjnie przeszkody na zadanej trajektorii ruchu, nie było dotychczas przedmiotem jakichkolwiek opublikowanych badań. Nowością jest również przetestowanie użyteczności takich algorytmów jak DDPG, TD3, SAC, HER, PPO, TRPO do pozycjonowania TCP robota w zadanym punkcie. Istotnym wkładem jest również zaproponowanie systemu sterowania składającego się z dwóch agentów pracujących równolegle oraz modyfikacja algorytmów w celu ich adaptacji do omijania przeszkód na zadanej trajektorii ruchu. Do wykrywania przeszkód w polu roboczym robota zaprojektowano i zbudowano dedykowaną głowicę z laserowymi czujnikami odległości zamontowaną na kiści manipulatora. Wykonane badania potwierdziły, że opracowany system sterownia robotem przemysłowym umożliwia jego współpracę w tej samej strefie roboczej z człowiekiem.

Głównymi osiągnięciami pracy są:

- zbudowanie i zweryfikowanie działania, służącej do wykrywania przeszkód głowicy z laserowymi czujnikami odległości, montowanej na kiści robota,
- opracowanie sterowania robotem w środowisku symulacyjnym uwzględniającego parametry mechaniczne i dynamiczne robota,
- badania i porównanie dokładności pozycjonowania TCP robota, sterowanego przez wybrane algorytmy uczenia ze wzmocnieniem,
- opracowanie systemu bazującego na algorytmach uczenia ze wzmocnieniem, służącego do omijania przeszkód na zadanej trajektorii ruchu robota przemysłowego,
- opracowanie systemu bazującego na algorytmach uczenia ze wzmocnieniem do bezpiecznej, ciągłej i wspólnej współpracy robota przemysłowego i człowieka w tej samej strefie roboczej.

Opracowane stanowisko laboratoryjne umożliwiło zaprezentowanie i zbadanie możliwości, jakie mogą dać algorytmy uczenia ze wzmocnieniem w zastosowaniu do sterowania robotem współpracującym z człowiekiem i potencjalnej aplikacji w warunkach przemysłowych. W warunkach laboratoryjnych nie było możliwości przetestowania systemu w trakcie, na przykład wiercenia i montażu elementów urządzeń we współpracy z człowiekiem, dlatego zdecydowano się na zastąpienie tych operacji symbolami w postaci kolorowych brył prostopadłościanów i walców. Operacje, które robot wykonywał w punkcie A (w warunkach laboratoryjnych podniesienie szarego prostopadłościanu) mogą zastępować wykonywanie wiercenia i następnie przeniesienia elementu do punktu B w celu montażu. Człowiek wykonywał swoje operacje montażowe w punkcie C i w niektórych momentach podchodził do punktu B (w warunkach laboratoryjnych pobierał szare prostopadłościany), co

może zastępować współpracę w montażu elementów, które były obrabiane przez robota w punkcie A.

Na podstawie badań różnych algorytmów uczenia ze wzmocnieniem, które polegały na pozycjonowaniu TCP robota w zadanym punkcie, za najlepszą uznano kombinację algorytmów HER i DDPG. Połączenie tych dwóch algorytmów pozwoliło na uzyskanie jednego z najmniejszych średnich błędów e ze wszystkich testowanych algorytmów. Przy zastosowaniu funkcji nagrody, która była proporcjonalna do odległości TCP robota od punktu zadanego, średni błąd e wynosił poniżej 1mm.

Kombinacje tych dwóch algorytmów zastosowano także w systemie do omijania przeszkód na zadanej trajektorii ruchu manipulatora. Zaproponowano i opracowano system sterowania, składający się z dwóch agentów pracujących równolegle. Jeden agent był odpowiedzialny za omijanie przeszkód, drugi za podążanie po zadanej trajektorii ruchu. W przypadku podążania, średni błąd e_{tr}, czyli różnica pomiędzy trajektorią ruchu TCP robota, a trajektorią zadaną, wahał się w zakresie od około 1.5mm do 2.5mm. Biorąc pod uwagę fakt, że robot był sterowany algorytmem, który uczy się i działa podobnie do człowieka można zauważyć, że błędy te są porównywalne z błędami, jakie mógłby popełnić człowiek wykonując takie samo zadanie jak algorytm. Dlatego, zdaniem autora niniejszej pracy, uzyskane dokładności i błędy są całkowicie akceptowalne na obecnym stanie rozwoju technik uczenia ze wzmocnieniem. Trzeba też zauważyć, że kształty trajektorii robota w trakcie omijania przeszkód są nieco chaotyczne i zauważalne są nieregularne ruchy po trójkącie. Eliminacja takich zachować wymagać będzie dalszych prac.

Na podstawie wyników przeprowadzonych testów można stwierdzić, że zaproponowany system był w stanie omijać bezkolizyjnie przeszkody, które znajdowały się w strefie roboczej robota. Algorytm sterujący robotem nie był w stanie ominąć tylko takich przeszkód, które pojawiły się w odległości kilku centymetrów od głowicy wykrywającej przeszkody. W takim przypadku system sterujący robotem zatrzymywał jego pracę.

Implementacja opracowanego w niniejszej pracy systemu do omijania przeszkód na zadanej trajektorii ruchu na rzeczywistym robocie przemysłowym, potwierdziła możliwości jednoczesnej jego pracy w tej samej strefie roboczej z człowiekiem. Pozwoliło to na zrezygnowanie z sekwencyjnego dostępu do wspólnej strefy przez robota i człowieka, co przyspieszyło wykonywanie przez nich zadań. Dzięki wykorzystaniu algorytmów uczenia ze wzmocnieniem robot mógł podejmować w pewnym stopniu autonomiczne decyzje w środowisku, w którym pojawiała się przeszkoda w jego polu roboczym. **Wykonane badania potwierdziły tezę pracy, która brzmiała: *gradientowe algorytmy uczenia ze wzmocnieniem mogą być zastosowane do opracowania bezpiecznego systemu sterowania robotem przemysłowym, współpracującym z człowiekiem.***

Przedstawiona w niniejszej pracy problematyka nie wyczerpuje w całości zagadnienia dotyczącego współpracy robota przemysłowego z człowiekiem. Istnieje zatem potrzeba prowadzenia dalszych prac obejmujących między innymi:

1. poprawę metod wykrywania przeszkód. Zaprojektowana głowica przymocowana do kiści robota wykrywa przeszkody tylko w pewnej określonej przestrzeni wokół niej. Powoduje to pewne ograniczenia w sterowaniu samym robotem oraz wykrywaniem przeszkód, które nie znajdują się w bezpośredniej bliskości TCP robota.

2. Zastosowanie kamer, które umożliwią wykrywanie przeszkód w przestrzeni trójwymiarowej wokół robota. Wymagają one zastosowania wydajnych komputerów wyposażonych w procesory graficzne z rdzeniami CUDA. Zastosowanie systemów wizyjnych i algorytmów uczenia ze wzmocnieniem do współpracy robota o elastycznych ramionach z człowiekiem.
3. Zastosowanie sztucznej skóry opartej na przykład o czujniki pojemnościowe. Taka skóra mogłaby być przymocowana do całego ramienia, co dałoby informacje o wszystkich przeszkodach wokół manipulatora. W tego typu rozwiązaniu wyzwaniem byłoby opracowanie algorytmu interpretującego dane ze sztucznej skóry i odpowiednio sterującego pracą robota, żeby omijał bezkolizyjnie przeszkody.
4. Eliminacje nieregularnych skoków po trójkącie w trakcie ruchu robota wynikających z zastosowania algorytmu uczenia ze wzmocnieniem. W celu rozwiązania tego problemu można zastosować dodatkowy algorytm, który wygładzałby trajektorię ruchu robota.
5. Badania nad opracowaniem szybkiego i wydajnego sterownika umożliwiającego bezpieczną współpracę robota z człowiekiem przy większych prędkościach ruchu robota, przekraczających $35\frac{mm}{s}$.

Załączniki

Wybrane hiperparametry algorytmów stosowanych w badaniach

Tabela 15. Hiperparametry algorytmu DDPG trenowanego do pozycjonowania TCP robota w zadanym punkcie (rozdział 7)

Nazwa	Symbol/objaśnienie	Wartość
`action_noise`	N_t	szum o rozkładzie normalnym ($n_\mu = 0,\ \sigma = 0.147013$)
`gamma`	γ	0.95
`actor_lr`	α_μ	0.00024433
`critic_lr`	α_Q	0.00024433
`batch_size`	B	256
`buffer_size`	D	50000
`rho`	ρ	0.999
sieci neuronowe aproksymujące politykę μ	liczba neuronów: funkcje aktywacji:	[64, 64] [ReLu, ReLu]
sieci neuronowe aproksymujące funkcję Q	liczba neuronów: funkcje aktywacji:	[64, 64] [ReLu, ReLu]

Tabela 16. Hiperparametry algorytmu TD3 trenowanego do pozycjonowania TCP robota w zadanym punkcie (rozdział 7)

Nazwa	Symbol/objaśnienie	Wartość
`action_noise`	N_t	szum o rozkładzie normalnym ($n_\mu = 0,\ \sigma = 0.847088$)
`gamma`	γ	0.9
`actor_lr`	α_μ	0.00562769
`critic_lr`	α_Q	0.00562769
`batch_size`	B	256
`buffer_size`	D	100000
`policy_delay`	opóźniona aktualizacja sieci polityki	2
`target_policy_noise`	ϵ	szum o rozkładzie normalnym ($\mu = 0,\ \sigma = 0.2$)
`target_policy_clip`	$-c, c$	$-0.5, 0.5$
`rho`	ρ	0.999
sieci neuronowe aproksymujące	liczba neuronów:	[64, 64]

politykę μ	funkcje aktywacji:	[ReLu, ReLu]
sieci neuronowe aproksymujące funkcję Q	liczba neuronów: funkcje aktywacji:	[64, 64] [ReLu, ReLu]

Tabela 17. Hiperparametry algorytmu SAC trenowanego do pozycjonowania TCP robota w zadanym punkcie (rozdział 7)

Nazwa	Symbol/objaśnienie	Wartość
`gamma`	γ	0.9
`actor_lr`	α_π	0.00099109
`critic_lr`	α_Q,α_V	0.00099109
`batch_size`	B	128
`buffer_size`	D	50000
`ent_coeff`	α_T	0.0001
`rho`	ρ	0.995
sieć neuronowa aproksymująca politykę π	liczba neuronów: funkcje aktywacji:	[64, 64] [ReLu, ReLu]
sieci neuronowe aproksymujące funkcję Q	liczba neuronów: funkcje aktywacji:	[64, 64] [ReLu, ReLu]
sieci neuronowe aproksymujące funkcję V	liczba neuronów: funkcje aktywacji:	[64, 64] [ReLu, ReLu]

Tabela 18. Hiperparametry algorytmu HER + DDPG trenowanego do pozycjonowania TCP robota w zadanym punkcie (rozdział 7)

Nazwa	Symbol/objaśnienie	Wartość
`action_noise`	N_t	szum Ornsteina-Uhlenbecka ($n_\mu = 0,\ \sigma = 0.522048$)
`gamma`	γ	0.9
`actor_lr`	α_μ	0.00116661
`critic_lr`	α_Q	0.00116661
`batch_size`	B	256
`buffer_size`	D	100000
`rho`	ρ	0.999
sieci neuronowe aproksymujące politykę μ	liczba neuronów: funkcje aktywacji:	[256, 256, 256] [ReLu, ReLu, ReLu]
sieci neuronowe aproksymujące funkcję Q	liczba neuronów: funkcje aktywacji:	[256, 256, 256] [ReLu, ReLu, ReLu]
`goal_selection_strategy`	$m(s_T)$	`FUTURE`
`n_sampled_goal`	g_{ns}	8

Tabela 19. Hiperparametry algorytmu HER + TD3 trenowanego do pozycjonowania TCP robota w zadanym punkcie (rozdział 7)

Nazwa	Symbol/objaśnienie	Wartość
`action_noise`	N_t	szum Ornsteina-Uhlenbecka ($n_\mu = 0,\ \sigma = 0.107301$)
`gamma`	γ	0.9
`actor_lr`	α_μ	0.00022318
`critic_lr`	α_Q	0.00022318
`batch_size`	B	256
`buffer_size`	D	100000
`policy_delay`	opóźniona aktualizacja sieci polityki	2
`target_policy_noise`	ϵ	szum o rozkładzie normalnym ($\mu = 0,\ \sigma = 0.2$)
`target_policy_clip`	$-c, c$	$-0.5, 0.5$
`rho`	ρ	0.995
sieci neuronowe aproksymujące politykę μ	liczba neuronów: funkcje aktywacji:	[256, 256, 256] [ReLu, ReLu, ReLu]
sieci neuronowe aproksymujące funkcję Q	liczba neuronów: funkcje aktywacji:	[256, 256, 256] [ReLu, ReLu, ReLu]
`goal_selection_strategy`	$m(s_T)$	`FUTURE`
`n_sampled_goal`	g_{ns}	6

Tabela 20. Hiperparametry algorytmu HER + SAC trenowanego do pozycjonowania TCP robota w zadanym punkcie (rozdział 7)

Nazwa	Symbol/objaśnienie	Wartość
`gamma`	γ	0.95
`actor_lr`	α_π	0.00456012
`critic_lr`	α_Q,α_V	0.00456012
`batch_size`	B	128
`buffer_size`	D	100000
`ent_coeff`	α_T	0.001
`rho`	ρ	0.995
sieć neuronowa aproksymująca politykę π	liczba neuronów: funkcje aktywacji:	[256, 256, 256] [ReLu, ReLu, ReLu]
sieci neuronowe aproksymujące funkcję Q	liczba neuronów: funkcje aktywacji:	[256, 256, 256] [ReLu, ReLu, ReLu]

sieci neuronowe aproksymujące funkcję V	liczba neuronów: funkcje aktywacji:	[256, 256, 256] [ReLu, ReLu, ReLu]
`goal_selection_strategy`	$m(s_T)$	`FUTURE`
`n_sampled_goal`	g_{ns}	6

Tabela 21. Hiperparametry algorytmu HER + DDPG trenowanego do omijania przeszkód (rozdział 8)

Nazwa	**Symbol/objaśnienie**	**Wartość**
`action_noise`	N_t	szum Ornsteina-Uhlenbecka ($n_\mu = 0,\ \sigma = 0.434781$)
`gamma`	γ	0.95
`actor_lr`	α_μ	0.00037169
`critic_lr`	α_Q	0.00037169
`batch_size`	B	256
`buffer_size`	D	100000
`rho`	ρ	0.999
sieci neuronowe aproksymujące politykę μ	liczba neuronów: funkcje aktywacji:	[256, 256, 256] [ReLu, ReLu, ReLu]
sieci neuronowe aproksymujące funkcję Q	liczba neuronów: funkcje aktywacji:	[256, 256, 256] [ReLu, ReLu, ReLu]
`goal_selection_strategy`	$m(s_T)$	`FUTURE`
`n_sampled_goal`	g_{ns}	8

Bibliografia

[1] „ISO 10218-1:2011, Robots and robotic devices — Safety requirements for industrial robots — Part 1: Robots”, *ISO*, 2011. https://www.iso.org/standard/51330.html.

[2] „ISO/TS 15066:2016, Robots and robotic devices — Collaborative robots”, *ISO*, 2016. https://www.iso.org/standard/62996.html.

[3] J.-C. Latombe, *Robot Motion Planning. Introduction and Overview*, t. 124. The Springer International Series in Engineering and Computer Science, 1991.

[4] A. Koubaa, H. Bennaceur, I. Chaari, S. Trigui, A. Ammar, M.-F. Sriti, M. Alajlan, O. Cheikhrouhou, i Y. Javed, *Robot Path Planning and Cooperation: Foundations, Algorithms and Experimentations*. Springer International Publishing, 2018.

[5] R. S. Sutton i A. G. Barto, *Reinforcement Learning: An Introduction*. The MIT Press, 2018.

[6] I. Goodfellow, Y. Bengio, i A. Courville, *Deep Learning*. MIT Press, 2016.

[7] NVIDIA, P. Vingelmann, i F. H. P. Fitzek, „CUDA - Compute Unified Device Architecture”, 2020. https://developer.nvidia.com/cuda-toolkit.

[8] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mane, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viegas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, i X. Zheng, „TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems”, *arXiv:1603.04467 [cs]*, 2016.

[9] F. Chollet, „Keras”, 2015. https://keras.io.

[10] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, i S. Chintala, „PyTorch: An Imperative Style, High-Performance Deep Learning Library”, *arXiv:1912.01703 [cs, stat]*, 2019.

[11] C. Breazeal, „Social interactions in HRI: the robot view”, *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, t. 34, nr 2, s. 181–186, 2004.

[12] S. Haddadin i E. Croft, „Physical Human–Robot Interaction”, *Springer Handbook of Robotics*, Cham, s. 1835–1874, 2016.

[13] A. Bicchi, M. A. Peshkin, i J. E. Colgate, „Safety for Physical Human–Robot Interaction”, *Springer Handbook of Robotics*, Berlin, Heidelberg, s. 1335–1348, 2008.

[14] A. De Santis, B. Siciliano, A. De Luca, i A. Bicchi, „An atlas of physical human–robot interaction”, *Mechanism and Machine Theory*, t. 43, nr 3, s. 253–270, 2008.

[15] S. Parsa i M. Saadat, „Human-robot collaboration disassembly planning for end-of-life product disassembly process”, *Robotics and Computer-Integrated Manufacturing*, t. 71, s. 102–117, 2021.

[16] L. Johannsmeier i S. Haddadin, „A Hierarchical Human-Robot Interaction-Planning Framework for Task Allocation in Collaborative Industrial Assembly Processes”, *IEEE Robot. Autom. Lett.*, t. 2, nr 1, s. 41–48, 2017.

[17] P. Tsarouchi, A.-S. Matthaiakis, S. Makris, i G. Chryssolouris, „On a human-robot collaboration in an assembly cell”, *International Journal of Computer Integrated Manufacturing*, t. 30, nr 6, s. 580–589, 2017.

[18] B. Sadrfaridpour i Y. Wang, „Collaborative Assembly in Hybrid Manufacturing Cells: An Integrated Framework for Human–Robot Interaction”, *IEEE Trans. Automat. Sci. Eng.*, t. 15, nr 3, s. 1178–1192, 2018.

[19] A. M. Zanchettin, N. M. Ceriani, P. Rocco, H. Ding, i B. Matthias, „Safety in human-robot collaborative manufacturing environments: Metrics and control", *IEEE Trans. Automat. Sci. Eng.*, t. 13, nr 2, s. 882–893, 2016.
[20] H. Ding, M. Schipper, i B. Matthias, „Collaborative behavior design of industrial robots for multiple human-robot collaboration", *IEEE ISR 2013*, Seoul, Korea (South), s. 1–6, 2013.
[21] N. Kousi, G. Michalos, S. Aivaliotis, i S. Makris, „An outlook on future assembly systems introducing robotic mobile dual arm workers", *Procedia CIRP*, t. 72, s. 33–38, 2018.
[22] I. Garcia, F. Goncalves, T. Ribeiro, P. Fernandes, C. Rocha, R. Boucinha, G. Lopes, i A. F. Ribeiro, „Autonomous 4DOF Robotic Manipulator Prototype for Industrial Environment and Human Cooperation", *2019 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, Porto, s. 1–6, 2019.
[23] M. Trivedi i P. Doshi, „Inverse Learning of Robot Behavior for Collaborative Planning", zaprezentowano na 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems, Madrid, 2018.
[24] M. Anvaripour, M. Khoshnam, C. Menon, i M. Saif, „FMG- and RNN-Based Estimation of Motor Intention of Upper-Limb Motion in Human-Robot Collaboration", *Front. Robot. AI*, t. 7, 2020.
[25] M. Sokolova, N. Japkowicz, i S. Szpakowicz, „Beyond Accuracy, F-Score and ROC: A Family of Discriminant Measures for Performance Evaluation", *AI 2006: Advances in Artificial Intelligence*, Berlin, Heidelberg, t. 4304, s. 1015–1021, 2006.
[26] M. Awais i D. Henrich, „Human-robot collaboration by intention recognition using probabilistic state machines", *19th International Workshop on Robotics in Alpe-Adria-Danube Region (RAAD 2010)*, Hungary, s. 75–80, 2010.
[27] S. Mahadevan, „Machine Learning for Robots: A Comparison of Different Paradigms", zaprezentowano na IEEE/RSJ International Conference on Intelligent Robots and Systems, Osaka, 1996.
[28] S. Schaal i C. G. Atkeson, „Robot learning by nonparametric regression", *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'94)*, t. 1, s. 478–485, 1994.
[29] C. G. Atkeson i S. Schaal, „Learning tasks from a single demonstration", *Proceedings of International Conference on Robotics and Automation*, t. 2, s. 1706–1712, 1997.
[30] S. Schaal i C. G. Atkeson, „Learning Control in Robotics", *IEEE Robotics Automation Magazine*, t. 17, nr 2, s. 20–29, 2010.
[31] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, i A. Rabinovich, „Going Deeper with Convolutions", *arXiv:1409.4842 [cs]*, 2014.
[32] K. He, X. Zhang, S. Ren, i J. Sun, „Deep Residual Learning for Image Recognition", *arXiv:1512.03385 [cs]*, 2015.
[33] J. Redmon, S. Divvala, R. Girshick, i A. Farhadi, „You Only Look Once: Unified, Real-Time Object Detection", *arXiv:1506.02640 [cs]*, 2016.
[34] J. Redmon i A. Farhadi, „YOLOv3: An Incremental Improvement", *arXiv:1804.02767 [cs]*, 2018.
[35] X. Long, K. Deng, G. Wang, Y. Zhang, Q. Dang, Y. Gao, H. Shen, J. Ren, S. Han, E. Ding, i S. Wen, „PP-YOLO: An Effective and Efficient Implementation of Object Detector", *arXiv:2007.12099 [cs]*, 2020.
[36] S. Minaee, Y. Boykov, F. Porikli, A. Plaza, N. Kehtarnavaz, i D. Terzopoulos, „Image Segmentation Using Deep Learning: A Survey", *arXiv:2001.05566 [cs]*, 2020.

[37] Microsoft, „Turing-NLG: A 17-billion-parameter language model by Microsoft", 2020. https://www.microsoft.com/en-us/research/blog/turing-nlg-a-17-billion-parameter-language-model-by-microsoft/.
[38] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, i D. Amodei, „Language Models are Few-Shot Learners", *arXiv:2005.14165 [cs]*, 2020.
[39] D. P. Kingma i J. Ba, „Adam: A Method for Stochastic Optimization", *arXiv:1412.6980 [cs]*, 2017.
[40] S. Ioffe i C. Szegedy, „Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift", *arXiv:1502.03167 [cs]*, 2015.
[41] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, i R. Salakhutdinov, „Dropout: A Simple Way to Prevent Neural Networks from Overfitting", *Journal of Machine Learning Research*, t. 15, s. 1929–1958, 2014.
[42] I. Lenz, H. Lee, i A. Saxena, „Deep Learning for Detecting Robotic Grasps", *arXiv:1301.3592 [cs]*, 2014.
[43] S. Levine, P. Pastor, A. Krizhevsky, i D. Quillen, „Learning Hand-Eye Coordination for Robotic Grasping with Deep Learning and Large-Scale Data Collection", *arXiv:1603.02199 [cs]*, 2016.
[44] K. Zakka, A. Zeng, J. Lee, i S. Song, „Form2Fit: Learning Shape Priors for Generalizable Assembly from Disassembly", *arXiv:1910.13675 [cs]*, 2020.
[45] P. Maes i R. A. Brooks, „Learning to Coordinate Behaviors", *Eighth National Conference on Artificial Intelligence*, s. 7, 1990.
[46] S. Mahadevan i J. Connell, „Automatic programming of behavior-based robots using reinforcement learning", *Artificial Intelligence*, t. 55, nr 2, s. 311–365, 1992.
[47] J. Kober, J. A. Bagnell, i J. Peters, „Reinforcement learning in robotics: A survey", *The International Journal of Robotics Research*, t. 32, nr 11, s. 1238–1274, 2013.
[48] M. Plappert, M. Andrychowicz, A. Ray, B. McGrew, B. Baker, G. Powell, J. Schneider, J. Tobin, M. Chociej, P. Welinder, V. Kumar, i W. Zaremba, „Multi-Goal Reinforcement Learning: Challenging Robotics Environments and Request for Research", *arXiv:1802.09464 [cs]*, 2018.
[49] Y. Li, „Deep Reinforcement Learning: An Overview", *arXiv:1701.07274 [cs]*, 2018.
[50] T. Y. Chun, J. B. Park, i Y. H. Choi, „Reinforcement Q-learning based on Multirate Generalized Policy Iteration and Its Application to a 2-DOF Helicopter", *International Journal of Control, Automation and Systems*, t. 16, nr 1, s. 377–386, 2018.
[51] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, i D. Wierstra, „Continuous control with deep reinforcement learning", *arXiv:1509.02971 [cs, stat]*, 2015.
[52] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, i D. Hassabis, „Human-level control through deep reinforcement learning", *Nature*, t. 518, nr 7540, s. 529–533, 2015.
[53] Y. Duan, X. Chen, R. Houthooft, J. Schulman, i P. Abbeel, „Benchmarking Deep Reinforcement Learning for Continuous Control", *arXiv:1604.06778 [cs]*, 2016.
[54] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, i D. Meger, „Deep Reinforcement Learning that Matters", *arXiv:1709.06560 [cs, stat]*, 2019.

[55] N. Heess, D. TB, S. Sriram, J. Lemmon, J. Merel, G. Wayne, Y. Tassa, T. Erez, Z. Wang, S. M. A. Eslami, M. Riedmiller, i D. Silver, „Emergence of Locomotion Behaviours in Rich Environments”, *arXiv:1707.02286 [cs]*, 2017.
[56] L. Liu i J. Hodgins, „Learning to Schedule Control Fragments for Physics-Based Characters Using Deep Q-Learning”, *ACM Transactions on Graphics*, t. 36, nr 3, s. 1–14, 2017.
[57] X. B. Peng, G. Berseth, i M. van de Panne, „Terrain-adaptive locomotion skills using deep reinforcement learning”, *ACM Transactions on Graphics*, t. 35, nr 4, s. 1–12, 2016.
[58] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, i P. Abbeel, „Domain randomization for transferring deep neural networks from simulation to the real world”, *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, s. 23–30, 2017.
[59] X. B. Peng, M. Andrychowicz, W. Zaremba, i P. Abbeel, „Sim-to-Real Transfer of Robotic Control with Dynamics Randomization”, *2018 IEEE International Conference on Robotics and Automation (ICRA)*, s. 1–8, 2018.
[60] L. Duan, D. Xu, i I. Tsang, „Learning with Augmented Features for Heterogeneous Domain Adaptation”, *arXiv:1206.4660 [cs]*, 2012.
[61] W. Li, L. Duan, D. Xu, i I. W. Tsang, „Learning With Augmented Features for Supervised and Semi-Supervised Heterogeneous Domain Adaptation”, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, t. 36, nr 6, s. 1134–1148, 2014.
[62] S. James, P. Wohlhart, M. Kalakrishnan, D. Kalashnikov, A. Irpan, J. Ibarz, S. Levine, R. Hadsell, i K. Bousmalis, „Sim-to-Real via Sim-to-Sim: Data-efficient Robotic Grasping via Randomized-to-Canonical Adaptation Networks”, *arXiv:1812.07252 [cs]*, 2018.
[63] A. R. Mahmood, D. Korenkevych, G. Vasan, W. Ma, i J. Bergstra, „Benchmarking Reinforcement Learning Algorithms on Real-World Robots”, *arXiv:1809.07731 [cs, stat]*, 2018.
[64] M. A. Lee, C. Florensa, J. Tremblay, N. Ratliff, A. Garg, F. Ramos, i D. Fox, „Guided Uncertainty-Aware Policy Optimization: Combining Learning and Model-Based Strategies for Sample-Efficient Policy Learning”, *2020 IEEE International Conference on Robotics and Automation (ICRA)*, s. 7, 2020.
[65] F. Li, Q. Jiang, W. Quan, R. Song, i Y. Li, „Manipulation Skill Acquisition for Robotic Assembly using Deep Reinforcement Learning”, *2019 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)*, Hong Kong, China, s. 13–18, 2019.
[66] A. Singh, L. Yang, K. Hartikainen, C. Finn, i S. Levine, „End-to-End Robotic Reinforcement Learning without Reward Engineering”, *arXiv:1904.07854 [cs, stat]*, 2019.
[67] S. Gu, E. Holly, T. Lillicrap, i S. Levine, „Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates”, s. 3389–3396, 2017.
[68] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, i K. Kavukcuoglu, „Asynchronous Methods for Deep Reinforcement Learning”, *arXiv:1602.01783 [cs]*, 2016.
[69] Y. Tsurumine, Y. Cui, E. Uchibe, i T. Matsubara, „Deep reinforcement learning with smooth policy update: Application to robotic cloth manipulation”, *Robotics and Autonomous Systems*, t. 112, s. 72–83, 2019.
[70] S. Levine, C. Finn, T. Darrell, i P. Abbeel, „End-to-End Training of Deep Visuomotor Policies”, *arXiv:1504.00702 [cs]*, 2016.

[71] A. Iriondo, E. Lazkano, L. Susperregi, J. Urain, A. Fernandez, i J. Molina, „Pick and Place Operations in Logistics Using a Mobile Manipulator Controlled with Deep Reinforcement Learning", *Applied Sciences*, t. 9, nr 2, s. 348, 2019.
[72] Y. Deng, X. Guo, Y. Wei, K. Lu, B. Fang, D. Guo, H. Liu, i F. Sun, „Deep Reinforcement Learning for Robotic Pushing and Picking in Cluttered Environment", *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, s. 619–626, 2019.
[73] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, i S. Levine, „QT-Opt: Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation", *arXiv:1806.10293 [cs, stat]*, 2018.
[74] Y. Ansari, E. Falotico, Y. Mollard, B. Busch, M. Cianchetti, i C. Laschi, „A Multiagent Reinforcement Learning approach for inverse kinematics of high dimensional manipulators with precision positioning", *2016 6th IEEE International Conference on Biomedical Robotics and Biomechatronics (BioRob)*, s. 457–463, 2016.
[75] M. Tang, X. Yue, Z. Zuo, X. Huang, Y. Liu, i N. Qi, „Coordinated Motion Planning of Dual-arm Space Robot with Deep Reinforcement Learning", *2019 IEEE International Conference on Unmanned Systems (ICUS)*, Beijing, s. 469–473, 2019.
[76] M. Duguleana, F. G. Barbuceanu, A. Teirelbar, i G. Mogan, „Obstacle avoidance of redundant manipulators using neural networks based reinforcement learning", *Robotics and Computer-Integrated Manufacturing*, t. 28, nr 2, s. 132–146, 2012.
[77] P. Singamaneni, P. Dewangan, A. Sarkar, i M. K. Krishna, „Learning Multi-Goal Inverse Kinematics in Humanoid Robot", *ISR 2018; 50th International Symposium on Robotics*, s. 1–6, 2018.
[78] Z. Guo, J. Huang, W. Ren, i C. Wang, „A Reinforcement Learning Approach for Inverse Kinematics of Arm Robot", *Proceedings of the 2019 4th International Conference on Robotics, Control and Automation - ICRCA 2019*, Guangzhou, China, s. 95–99, 2019.
[79] A. Zeng, S. Song, J. Lee, A. Rodriguez, i T. Funkhouser, „TossingBot: Learning to Throw Arbitrary Objects with Residual Physics", *arXiv:1903.11239 [cs.RO]*, 2019.
[80] OpenAI, M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, J. Schneider, S. Sidor, J. Tobin, P. Welinder, L. Weng, i W. Zaremba, „Learning Dexterous In-Hand Manipulation", *arXiv:1808.00177 [cs, stat]*, 2019.
[81] V. Kumar, A. Gupta, E. Todorov, i S. Levine, „Learning Dexterous Manipulation Policies from Experience and Imitation", *arXiv:1611.05095 [cs]*, 2016.
[82] V. Kumar, E. Todorov, i S. Levine, „Optimal control with learned local models: Application to dexterous manipulation", *2016 IEEE International Conference on Robotics and Automation (ICRA)*, Stockholm, Sweden, s. 378–383, 2016.
[83] M. Liu, L. Hao, W. Zhang, Y. Chen, i J. Chen, „Reinforcement Learning Control of a Shape Memory Alloy-based Bionic Robotic Hand", *2019 IEEE 9th Annual International Conference on CYBER Technology in Automation, Control, and Intelligent Systems (CYBER)*, s. 969–973, 2019.
[84] E. V. Añazco, P. R. Lopez, H. Park, N. Park, J. Oh, S. Lee, K. Byun, i T.-S. Kim, „Human-like Object Grasping and Relocation for an Anthropomorphic Robotic Hand with Natural Hand Pose Priors in Deep Reinforcement Learning", *2019 2nd International Conference on Robot Systems and Applications*, New York, s. 46–50, 2019.
[85] J. Lee, J. Hwangbo, i M. Hutter, „Robust Recovery Controller for a Quadrupedal Robot using Deep Reinforcement Learning", *arXiv:1901.07517 [cs]*, 2019.

[86] J. Hwangbo, J. Lee, A. Dosovitskiy, D. Bellicoso, V. Tsounis, V. Koltun, i M. Hutter, „Learning agile and dynamic motor skills for legged robots”, *Sci. Robot.*, t. 4, nr 26, s. 1–20, 2019.
[87] X. B. Peng, M. Chang, G. Zhang, P. Abbeel, i S. Levine, „MCP: Learning Composable Hierarchical Control with Multiplicative Compositional Policies”, *arXiv:1905.09808 [cs, stat]*, 2019.
[88] X. B. Peng, G. Berseth, K. Yin, i M. Van De Panne, „DeepLoco: dynamic locomotion skills using hierarchical deep reinforcement learning”, *ACM Transactions on Graphics*, t. 36, nr 4, s. 1–13, 2017.
[89] X. B. Peng, A. Kanazawa, J. Malik, P. Abbeel, i S. Levine, „SFV: Reinforcement Learning of Physical Skills from Videos”, *arXiv:1810.03599 [cs]*, 2018.
[90] W. D. Smart i L. Pack Kaelbling, „Effective reinforcement learning for mobile robots”, *IEEE International Conference on Robotics and Automation (Cat. No.02CH37292)*, Washington, DC, USA, t. 4, s. 3404–3410, 2002.
[91] P. Mirowski, R. Pascanu, F. Viola, H. Soyer, A. J. Ballard, A. Banino, M. Denil, R. Goroshin, L. Sifre, K. Kavukcuoglu, D. Kumaran, i R. Hadsell, „Learning to Navigate in Complex Environments”, *arXiv:1611.03673 [cs]*, 2017.
[92] H. Bae, G. Kim, J. Kim, D. Qian, i S. Lee, „Multi-Robot Path Planning Method Using Reinforcement Learning”, *Applied Sciences*, t. 9, nr 15, s. 1–16, 2019.
[93] R.-J. Wai, C.-M. Liu, i Y.-W. Lin, „Design of switching path-planning control for obstacle avoidance of mobile robot”, *Journal of the Franklin Institute*, t. 348, nr 4, s. 718–737, 2011.
[94] M. Duguleana i G. Mogan, „Neural networks based reinforcement learning for mobile robots obstacle avoidance”, *Expert Systems with Applications*, t. 62, s. 104–115, 2016.
[95] B. Baker, I. Kanitscheider, T. Markov, Y. Wu, G. Powell, B. McGrew, i I. Mordatch, „Emergent Tool Use From Multi-Agent Autocurricula”, *arXiv:1909.07528 [cs, stat]*, 2019.
[96] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, i M. Riedmiller, „Playing Atari with Deep Reinforcement Learning”, *arXiv:1312.5602 [cs]*, 2013.
[97] OpenAI, C. Berner, G. Brockman, B. Chan, V. Cheung, P. Dębiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, R. Józefowicz, S. Gray, C. Olsson, J. Pachocki, M. Petrov, H. P. d O. Pinto, J. Raiman, T. Salimans, J. Schlatter, J. Schneider, S. Sidor, I. Sutskever, J. Tang, F. Wolski, i S. Zhang, „Dota 2 with Large Scale Deep Reinforcement Learning”, *arXiv:1912.06680 [cs, stat]*, 2019.
[98] K. Arulkumaran, A. Cully, i J. Togelius, „AlphaStar: An Evolutionary Computation Perspective”, *arXiv:1902.01724 [cs]*, 2019.
[99] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, i D. Hassabis, „Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm”, *arXiv:1712.01815 [cs]*, 2017.
[100] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, i D. Hassabis, „Mastering the game of Go with deep neural networks and tree search”, *Nature*, t. 529, nr 7587, Art. nr 7587, 2016.
[101] T. M. Moldovan i P. Abbeel, „Safe Exploration in Markov Decision Processes”, *arXiv:1205.4810 [cs]*, 2012.
[102] T. J. Perkins i A. G. Barto, „Lyapunov Design for Safe Reinforcement Learning”, *Journal of Machine Learning Research 3*, s. 803–832, 2002.

[103] D. Nguyen-Tuong i J. Peters, „Model learning for robot control: a survey", *Cogn Process*, t. 12, nr 4, s. 319–340, 2011.
[104] H. Moravec i A. Elfes, „High resolution maps from wide angle sonar", *1985 IEEE International Conference on Robotics and Automation Proceedings*, t. 2, s. 116–121, 1985.
[105] V. J. Lumelsky, M. S. Shur, i S. Wagner, „Sensitive skin", *IEEE Sensors Journal*, t. 1, nr 1, s. 41–51, 2001.
[106] P. Maiolino, A. Ascia, M. Maggiali, L. Natale, G. Cannata, i G. Mett, „Large Scale Capacitive Skin for Robots", *Smart Actuation and Sensing Systems - Recent Advances and Future Challenges*, t. 8, s. 1–18, 2012.
[107] A. M. Sabatini, V. Genovese, E. Guglielmelli, A. Mantuano, G. Ratti, i P. Dario, „A low-cost, composite sensor array combining ultrasonic and infrared proximity sensors", *Proceedings 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems. Human Robot Interaction and Cooperative Robots*, t. 3, s. 120–126 t.3, 1995.
[108] T. Petrič, A. Gams, N. Likar, i L. Žlajpah, „Obstacle Avoidance with Industrial Robots", *Motion and Operation Planning of Robotic Systems: Background and Practical Approaches*, Cham, s. 113–145, 2015.
[109] M. Kim, D.-K. Han, J.-H. Park, i J.-S. Kim, „Motion Planning of Robot Manipulators for a Smoother Path Using a Twin Delayed Deep Deterministic Policy Gradient with Hindsight Experience Replay", *Applied Sciences*, t. 10, nr 2, s. 575, 2020.
[110] E. Prianto, M. Kim, J.-H. Park, J.-H. Bae, i J.-S. Kim, „Path Planning for Multi-Arm Manipulators Using Deep Reinforcement Learning: Soft Actor–Critic with Hindsight Experience Replay", *Sensors*, t. 20, nr 20, s. 1–22, 2020.
[111] S. Wen, J. Chen, S. Wang, H. Zhang, i X. Hu, „Path Planning of Humanoid Arm Based on Deep Deterministic Policy Gradient", *2018 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, Kuala Lumpur, Malaysia, s. 1755–1760, 2018.
[112] M. Ji, L. Zhang, i S. Wang, „A Path Planning Approach Based on Q-learning for Robot Arm", *2019 3rd International Conference on Robotics and Automation Sciences (ICRAS)*, Wuhan, China, s. 15–19, 2019.
[113] D. Zhang i C. P. Bailey, „Obstacle Avoidance and Navigation Utilizing Reinforcement Learning with Reward Shaping", *arXiv:2003.12863 [cs, eess]*, 2020.
[114] K. Wei i B. Ren, „A Method on Dynamic Path Planning for Robotic Manipulator Autonomous Obstacle Avoidance Based on an Improved RRT Algorithm", *Sensors*, t. 18, nr 2, Art. nr 2, 2018.
[115] N. Ratliff, M. Zucker, J. A. Bagnell, i S. Srinivasa, „CHOMP: Gradient optimization techniques for efficient motion planning", *2009 IEEE International Conference on Robotics and Automation*, Kobe, s. 489–494, 2009.
[116] J. Schulman, Y. Duan, J. Ho, A. Lee, I. Awwal, H. Bradlow, J. Pan, S. Patil, K. Goldberg, i P. Abbeel, „Motion planning with sequential convex optimization and convex collision checking", *The International Journal of Robotics Research*, t. 33, nr 9, s. 1251–1270, 2014.
[117] D. Tsetserukou, N. Kawakami, i S. Tachi, „Obstacle avoidance control of humanoid robot arm through tactile interaction", *Humanoids 2008 - 8th IEEE-RAS International Conference on Humanoid Robots*, s. 379–384, 2008.
[118] Y. P. Pane, S. P. Nageshrao, J. Kober, i R. Babuška, „Reinforcement learning based compensation methods for robot manipulators", *Engineering Applications of Artificial Intelligence*, t. 78, s. 236–247, 2019.
[119] J. Xiao, L. Li, Y. Zou, i T. Zhang, „Reinforcement Learning for Robotic Time-optimal Path Tracking Using Prior Knowledge", *arXiv:1907.00388 [cs, eess]*, 2019.

[120] K. Ota, D. K. Jha, T. Oiki, M. Miura, T. Nammoto, D. Nikovski, i T. Mariyama, „Trajectory Optimization for Unknown Constrained Systems using Reinforcement Learning”, *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, s. 3487–3494, 2019.
[121] S. Fujimoto, H. van Hoof, i D. Meger, „Addressing Function Approximation Error in Actor-Critic Methods”, *arXiv:1802.09477 [cs, stat]*, 2018.
[122] T. Haarnoja, A. Zhou, P. Abbeel, i S. Levine, „Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor”, *arXiv:1801.01290 [cs, stat]*, 2018.
[123] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, i S. Levine, „Soft Actor-Critic Algorithms and Applications”, *arXiv:1812.05905 [cs, stat]*, 2018.
[124] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel, i W. Zaremba, „Hindsight Experience Replay”, *arXiv:1707.01495 [cs]*, 2017.
[125] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, i O. Klimov, „Proximal Policy Optimization Algorithms”, *arXiv:1707.06347 [cs]*, 2017.
[126] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, i P. Abbeel, „Trust Region Policy Optimization”, *arXiv:1502.05477 [cs]*, 2015.
[127] A. Hill, A. Raffin, M. Ernestus, A. Gleave, A. Kanervisto, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, i Y. Wu, „Stable Baselines”, *GitHub repository*, 2018. https://github.com/hill-a/stable-baselines.
[128] G. E. Uhlenbeck i L. S. Ornstein, „On the Theory of the Brownian Motion”, *Phys. Rev.*, t. 36, nr 5, s. 823–841, 1930.
[129] E. Todorov, T. Erez, i Y. Tassa, „MuJoCo: A physics engine for model-based control”, *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, s. 5026–5033, 2012.
[130] J. Denavit i R. S. Hartenberg, „A Kinematic Notation for Lower-Pair Mechanisms Based on Matrices”, *Journal of Applied Mechanics*, t. 22, nr 2, s. 215–221, 1955.
[131] A. Khatamian, „Solving Kinematics Problems of a 6-DOF Robot Manipulator”, *Int’l Conf. Scientific Computing*, s. 6, 2015.
[132] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, i W. Zaremba, „OpenAI Gym”, *arXiv:1606.01540 [cs]*, 2016.
[133] T. Akiba, S. Sano, T. Yanase, T. Ohta, i M. Koyama, „Optuna: A Next-generation Hyperparameter Optimization Framework”, *arXiv:1907.10902 [cs, stat]*, 2019.
[134] JooSeuk Kim i C. Scott, „Robust kernel density estimation”, *2008 IEEE International Conference on Acoustics, Speech and Signal Processing*, s. 3381–3384, 2008.