Politechnika Wrocławska

Wydział Elektroniki, Fotoniki i Mikrosystemów

KIERUNEK: Automatyka i Robotyka (AIR)

PRACA DYPLOMOWA INŻYNIERSKA

TYTUŁ PRACY:

Symulator sterowania orientacją sześcionożnego robota kroczącego

AUTOR:

Maksymilian Tulewicz

PROMOTOR:

Dr inż. Robert Muszyński,
Katedra Cybernetyki i Robotyki

WROCŁAW 2025

Streszczenie

Przedmiotem pracy jest projekt sześcionożnej platformy kroczącej, realizującej dwa systemy obliczania kinematyki odwrotnej, dwa mechanizmy sterowania orientacją oraz jeden mechanizmu chodu (trójpodporowy).

W ramach pracy za pomocą środowisk CAD (Autodesk Inventor/Fusion360) zaprojektowany został trójwymiarowy model robota, który następnie został przekonwertowany i zintegrowany ze środowiskiem symulacyjnym Gazebo oraz systemem ROS 2. Warstwa sterowania zaimplementowana została w języku C++ i obejmuje dwa algorytmy liczące kinematykę odwrotną – geometryczną oraz jakobianową (z wykorzystaniem metody transpozycji jakobianu). Ponadto opracowane zostały dwa sterowniki orientacji korpusu – poprzez metodę geometryczną oraz metodę liczenia macierzy jednorodnej. Zaimplementowano również generator trajektorii chodu.

Przeprowadzenie badań symulacyjnych pozwoliło na stwierdzenie dokładności zaimplementowanych rozwiązań oraz ocenę ich wydajności. Wyniki wskazują na wyższą precyzję metody geometrycznej obliczania kinematyki odwrotnej oraz znacznie krótszy czas obliczeń w porównaniu z metodą jakobianową. W przypadku badań nad sterowaniem trajektorii udowodniono, że obydwa algorytmy realizują precyzyjnie to samo zadanie ze zbliżoną dokładnością i korelacją na poziomie powyżej $0,999$.

Kluczowe słowa: robot kroczący, heksapod, kinematyka odwrotna, sterowanie orientacją, symulacja, ROS 2, Gazebo

Abstract

The subject of this thesis is the design of a six-legged hexapod walking platform, implementing two inverse kinematics calculation systems, two orientation control mechanisms, and one gait mechanism (tripod).

In the course of the project, a three-dimensional model of the robot was devised utilising CAD environments (Autodesk Inventor/Fusion360). This model was subsequently converted and integrated with the Gazebo simulation environment and the ROS 2 system. The control layer was implemented in the programming language C++ and incorporates two inverse kinematic algorithms: geometric and Jacobian (utilising the Jacobian transpose method). Moreover, two body orientation controllers were developed using the geometric method and the homogeneous transformation matrix method. In addition, a gait trajectory generator was implemented.

The implementation of simulation tests permitted the verification of the accuracy of the solutions that had been implemented, as well as the assessment of their performance. The findings suggest that the geometric method for calculating inverse kinematics is more precise and significantly faster than the jacobian method. In the context of trajectory control research, it has been demonstrated that both algorithms execute the task with equivalent precision, exhibiting a correlation level that consistently exceeds 0.999.

Keywords: walking robot, hexapod, inverse kinematics, orientation control, simulation, ROS 2, Gazebo

Pracę tę dedykuję wszystkim, którzy wierzyli w jej powstanie, mojej rodzinie oraz przyjaciołom. Szczególne podziękowania kieruję do Pana dr. inż. Roberta Muszyńskiego za jego czas, zaangażowanie oraz wyrozumiałość.

Spis treści

Rozdział 1

Wstęp

Marzenie o stworzeniu maszyny zdolnej do samodzielnego poruszania się towarzyszy ludzkości od wieków. Od mitologicznych automatów Hefajstosa, przez mechaniczne zabawki renesansowych rzemieślników, po współczesne laboratoria robotyki. Fascynacja konstrukcjami naśladującymi sposób poruszania się organizmów żywych pozostaje niezmienna. Ogromny rozwój technologii w minionym i obecnym wieku spowodował, że projekty niegdyś będące w swerze fantazji są na wyciągnięcie ręki dla każdego gotowego poświęcić im czas.

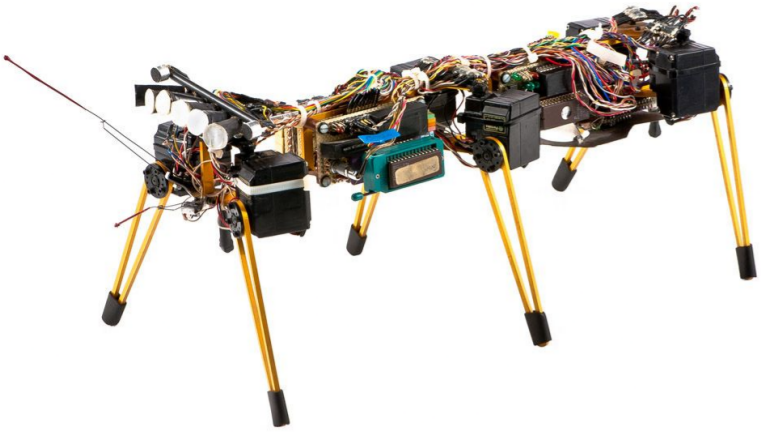
Wśród różnych architektur robotów kroczących szczególne miejsce zajmują konstrukcje sześcionożne (heksapody). Ich popularność podyktowana jest kompromisem pomiędzy stabilnością ruchu a złożonością sterowania tych platform. W przeciwieństwie do robotów dwunożnych, wymagających zaawansowanego balansowania dynamicznego, heksapody ze względu na dużą ilość odnóży są naturalnie stabilne. Jednocześnie sześć nóg, każda o trzech stopniach swobody stanowi interesujące wyzwanie z punktu widzenia teorii sterowania. Inspiracją dla tego typu konstrukcji są owady, które przez setki milionów lat ewolucji wykształciły niezwykle efektywne mechanizmy lokomocji. Przełomową realizacją tej koncepcji w robotyce był Genghis – sześcionożny robot zaprezentowany w 1989 roku przez Rodneya Brooksa z MIT (rysunek 1.1). Konstrukcja ta, wykorzystująca architekturę subsumpcyjną, udowodniła że złożone zachowania lokomocyjne mogą wynikać z interakcji prostych, reaktywnych modułów sterowania.

Od czasu Genghisa heksapody znalazły zastosowanie we wszelakich misjach poszukiwawczo-ratunkowych, inspekcji infrastruktury oraz eksploracji terenów niedostępnych dla pojazdów kołowych. Niezależnie od zastosowania, kluczowym aspektem ich funkcjonowania – wykraczającym poza samą lokomocję – jest kontrola orientacji korpusu w przestrzeni. Precyzyjne sterowanie kątami przechyłu, pochylenia i odchylenia umożliwia zarówno adaptację do profilu terenu, jak i stabilizację sensorów zamontowanych na platformie.

Niniejsza praca podejmuje właśnie to zagadnienie, czerpiąc między innymi z doświadczeń opisanych w pracach [Win24] oraz [Gie12], poświęconych konstrukcji i sterowaniu robotów tego typu.

1.1 Cel i zakres pracy

Celem pracy jest zaprojektowanie, implementacja i testy systemu sterowania sześcionożnego robota kroczącego w środowisku symulacyjnym. Szczególny nacisk położono na



Rysunek 1.1: Zdjęcie robota kroczącego Genghis [Rob25]

analizę i projekt algorytmów sterujących orientacją korpusu robota w przestrzeni. Realizacja celu obejmowała:

- zaprojektowanie modelu robota oraz środowiska symulacji,
- implementację modelu matematycznego kinematyki odwrotnej,
- opracowanie algorytmów sterowania orientacją,
- implementację generatora chodu.

W ramach pracy opisano i zaimplementowano dwie metody obliczania kinemtyki odwrotnej – analityczną opartą na analizie geometrycznej [Zie03] oraz numeryczną korzystającą z wyliczenia jakobianu [Bus09]. Ponadto zrealizowano dwa podejścia sterowania orientacją korpusu – intuicyjną metodę wyprowadzeń geometrycznych [Zie03] oraz klasyczną metodę macierzy jednorodnej w konwencji obrotów o kąty Eulera [Cra05].

W ramach realizacji zadania chodu, zrealizowano jeden tryb poruszania – chód trójpodporowy (ang. tripod gait). Pozwala on na zachowanie stabilności ruchu poprzez utrzymanie kontaktu z podłożem w trzech punktach w każdej fazie korku.

1.2 Metodyka badań

Praca ma charakter testowy. Badania zostały przeprowadzone w środowisku symulacyjnym Gazebo z wykorzystaniem architektury systemu ROS 2. W pierwszej kolejności przeprowadzono badania poprawności i wydajności algorytmów sterowania orientacją korpusu w statyce. Niezależnie od tego zweryfikowano poprawność generowania trajektorii chodu. Takie podejście pozwoliło na precyzyjną izolację błędów i dokładną analizę wpływu poszczególnych metod matematycznych na zachowanie robota.

1.3 Struktura pracy

Pracę podzielono na siedem rozdziałów, których treść po krótce prezentuje się następująco:

- Rozdział 2 – prezentacja modelu symulacyjnego oraz środowiska badawczego. Opisano w nim narzędzia, które zostały wykorzystane w pracy.
- Rozdział 3 – opis matematycznych podstaw do budowy algorytmów obliczania kinematyki odwrotnej w ujęciu geometrycznym oraz jakobianowym.
- Rozdział 4 – wyjaśnienie algorytmów sterowania robotem. W rozdziale omówiono dwie metody sterowania orientacją robota oraz generator trajektorii chodu trójpodporowego.
- Rozdział 5 – zawiera opis implementacji systemu w języku C++. Przedstawia architekturę oprogramowania bazującą na systemach komunikacyjnych ROS 2. Opisuje strukturę klas sterowników kinematyki oraz orientacji.
- Rozdział 6 – prezentacja i analiza zebranych danych badawczych z symulacji.
- Rozdział 7 – stanowi podsumowanie pracy, są w nim wyciągnięte wnioski płynące z procesu konstruowania symulacji oraz analizy zaimplementowanych modeli.

Rozdział 2

Model symulacyjny

W rozdziale przedstawiono sposób budowy i przygotowanie modelu symulacyjnego robota oraz opisano wykorzystane do tego narzędzia i oprogramowanie. Zademonstrowano proces modelowania obiektu, przekształcenie go do formy zrozumiałej dla środowiska symulacyjnego i dostosowanie parametrów fizycznych symulacji.

Kompletny model symulacyjny robota (zobacz rysunek 2.1) został opracowany z wykorzystaniem oprogramowania Autodesk Inventor oraz Fusion 360. Składa się z następujących części:

- korpusu *base_link*,
- 6 bioder *hip_link_#*, gdzie # jest oznaczeniem numeru konkretnego odnóża,
- 6 ud *thigh_link_#*,
- 6 goleni nazwanych *shin_link_#*.

Ponadto w modelu zostały zdefiniowane przeguby nazwane następująco:

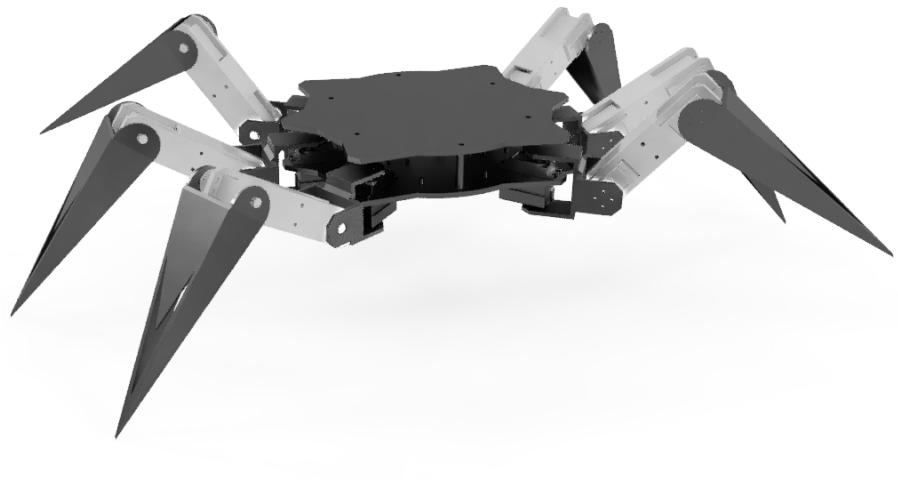
- 6 przegubów biodrowych – *hip_joint_#*,
- 6 przegubów kolan – *knee_joint_#*,
- 6 przegubów kostek – *ankle_joint_#*.

Zachowanie spójnego nazewnictwa bardzo ułatwia późniejsze prace w symulacji oraz edycję plików konfiguracyjnych modelu.

2.1 Gazebo

Gazebo jest zaawansowanym środowiskiem symulacyjnym pozwalającym na badania wszelakich systemów pod kątem zachowania w świecie rzeczywistym [KH04]. Posiada on silnik fizyczny potrafiący symulować grawitację, tarcia, kolizje, czy też siły oddziałujące na układ. Środowisko zostało zaprojektowane z myślą o robotyce, co za tym idzie, możliwe jest w nim również symulowanie danych z sensorów. Pozwala ono również na implementację własnych modeli. W przypadku niniejszej pracy takowy model został utworzony przy pomocy oprogramowania Inventor [Aut25b] oraz Fusion 360 [Aut25a].

Największą siłą Gazebo jest jego integracja z systemem ROS. Symulator działa jako niezależny serwer, komunikujący się za pomocą środków dostarczanych przez wspomniany system.



Rysunek 2.1: Model kompletnego robota

2.2 ROS2

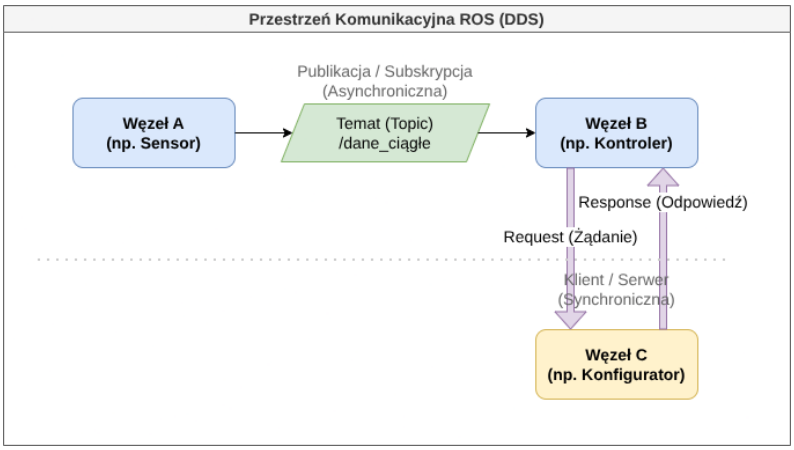
Robot Operating System 2 (ROS 2) jest otwartym środowiskiem pośredniczącym (ang. *middleware*), ułatwiającym budowę robotów [Ope25a, MFG+22]. Pozwala on na łatwą konstrukcję pakietów, mogących w czasie rzeczywistym komunikować ze sobą niezależne od siebie procesy, składające się na zintegrowany system. Począwszy od różnego rodzaju pojedynczych czujników w obrębie jednego systemu a kończąc na złożonych systemach występujących w robotyce rojowej.

Działanie w obrębie środowiska bazuje na tworzeniu paczek zawierających *węzły (ang. node)* będące procesami mogącymi komunikować się poprzez system *tematów (ang. topic)*. Co istotne, w ROS 2 wiele węzłów może być uruchomionych w ramach jednego procesu systemu operacyjnego, co skutkuje wysoką wydajnością, nawet przy jednoczesnym współdziałaniu wielu węzłów. Poszczególne węzły mogą komunikować się ze sobą, publikując bądź subskrybując dane przepływające przez dotyczące ich tematy. Komunikacja ta została zilustrowana na rysunku 2.2. Ilustruje on przepływ danych pomiędzy trzema węzłami, z których jeden posiada możliwość publikacji, a dwa pozostałe subskrybują temat i wykonują polecenia. ROS 2 zapewnia zestaw *wtyczek (ang. plugin)*, gotowych do uruchomienia w obrębie węzła.

2.3 Autodesk Inventor oraz Fusion 360

Każda symulacja fizyczna wymaga obiektu do symulacji. *Inventor* [Aut25b], jest zaawansowanym oprogramowaniem do parametrycznego modelowania 3D. Pozwala on na proste szkicowanie, edytowanie oraz eksport modeli 3D.

Środowisko *Fusion 360* [Aut25a] jest zintegrowaną platformą chmurową, która łączy w sobie CAD (Computer Aided Design), CAM (Computer Aided Manufacturing) oraz CAE (Computer Aided Engineering). Pozwala zarówno na modelowanie, wspomaganie procesów produkcji oraz analizę obiektów pod kątem wytrzymałości czy właściwości termicznych. Jego szczególną zaletą jest możliwość pisania wtyczek w formie skryptów.



Rysunek 2.2: Diagram komunikacji węzłów ROS

(a) elementy składowe (b) po złożeniu

Rysunek 2.3: Goleń robota

Modelowanie w oprogramowaniu CAD, jakim jest Inventor, jest procesem tworzenia cyfrowej reprezentacji fizycznego obiektu. Jego podstawą jest parametryczna definicja wymiarów elementu oraz zorientowanie go w przestrzeni. Proces rozpoczyna się od narysowania dwuwymiarowego szkicu na wybranej płaszczyźnie. Po ustaleniu wymiarów, parametrów i wiązań wewnątrz rysunku, szkic ten jest przekształcany w obiekt trójwymiarowy za pomocą operacji wyciągnięcia prostego (ang. extrude) bądź obrotu (ang. revolve). Przykładowy efekt wyżej opisanego procesu można zobaczyć na rysunku 2.3a. Elementy następnie zostały wyeksportowane do formatu *ipt* [Aut24], co pozwoli na dokonanie ich złożenia. Złożenie całego modelu zostało wykonane w środowisku Fusion 360. Na tym etapie wcześniej zamodelowane elementy składowe obiektów, jak na przykład front golenia, są łączone z bocznymi panelami omawianej części. Przykład wykonania złożenia golenia w sposób nieruchomy (ang. rigid joint) pokazano na rysunku 2.3b. Tak skonstruowana część jest następnie zapisywana w formacie złożeniowym *iam* [Aut24].

Zaprojektowane w powyższy sposób komponenty są ponownie formatowane. Wtyczka do oprogramowania Fusion 360 wymaga, aby komponenty systemu nie były same w sobie złożeniami. Co za tym idzie, przeprowadzone zostało *ujednolicenie (eng. shrinkwrap)* części. Proces ten skutkuje utworzeniem *brył* [Aut25c] i został przykładowo zilustrowany na nagraniu [Hac22].

Po formatowaniu i ujednolicaniu elementów składowych robota jest możliwe utworzenie ostatecznego ich złożenia. Do tego celu zostały wykorzystane złożenia obrotowe (ang. revolute joint). Operacja została zrealizowane w konkretnej kolejności dla uzyskania hierarchii komponentów w późniejszym opisie URDF. Począwszy od korpusu zamocowywano kolejno biodra, uda a na sam koniec golenie. Osie obrotu zostały umieszczone tak, aby ruch komponentu następował względem lokalnej osi Z. Skutkiem takiego zabiegu było utworzenie kompletnego modelu 3D robota przedstawionego na rysunku 2.1.

2.3.1 Wtyczka fusion2urdf

Symulator Gazebo do działania wymaga konkretnego formatu modelu. Jednym z takich formatów jest *URDF* (Unified Robot Description Format) [Wik25]. Uzyskanie wymaganych plików jest możliwe dzięki wykorzystaniu rozszerzenia do oprogramowania Fusion 360, które umożliwia przetłumaczenie złożeń i ustalonych osi obrotu na pliki opisujące pozycje i relacje w symulacji [Dhe22]. Wtyczka analizuje model – każde ogniwo (ang. link) oraz przegub (ang. joint) a następnie przypisuje im pozycję i orientację w odpowiednim formacie danych, który jest zrozumiały dla symulatora.

Sam proces wymaga nazwania członu będącego korpusem robota jako *base_link*. Wobec tego członu orientowane są kolejne komponenty*. Wtyczka *fusion2urdf* nie zawsze właściwie interpretuje parametry fizyczne modelu, dlatego należało wprowadzić zostały poprawki w pliku `hex_urdf_ros2_package.xacro`. Skorygowano tam masy oraz tensory bezwładności, które bezpośrednio wpływają na odpowiedź robota na sterowanie – zbyt małe momenty bezwładności powodują niestabilne ruchy robota. W pliku konfiguracyjnym `hex_urdf_ros2_package.gazebo` dostosowano współczynniki tarcia między stopami a podłożem oraz parametry reakcji podłoża: `<kp>` (sztywność) i `<kd>` (tłumienie). Dodatkowo przygotowano świat `hexapod_world.world` w którym zdefiniowano właściwości podłoża (w tym współczynniki tarcia) oraz domyślne wartości `<kp>` i `<kd>`. W ramach tego pliku jawnie wskazano też silnik dynamiki `ode` (domyślny w Gazebo), co ułatwia późniejszą zmianę solvera oraz strojenie parametrów.

*Należy pamiętać, by usunąć połączenie elementów z historią ich edycji tworzoną automatycznie przez środowisko Fusion, aby nie wpływała ona na procesy skryptu interpretującego.

Rozdział 3

Model matematyczny

Realizacja chodu robota opiera się na rozwiązywaniu dla każdej z nóg zadania kinematyki odwrotnej w czasie rzeczywistym. Proces ten pozwala na zaplanowanie trajektorii ich ruchu. W rozdziale opisane zostały podstawy matematyczne wspomnianego procesu oraz konwencje nazewnictwa i oznaczeń wykorzystywane w pracy.

3.1 Układ współrzędnych i konwencje

Robot heksapod składa się z korpusu oraz sześciu identycznych nóg, z których każda stanowi manipulator trójczłonowy o przegubach obrotowych. Przyjęto następujące oznaczenia:

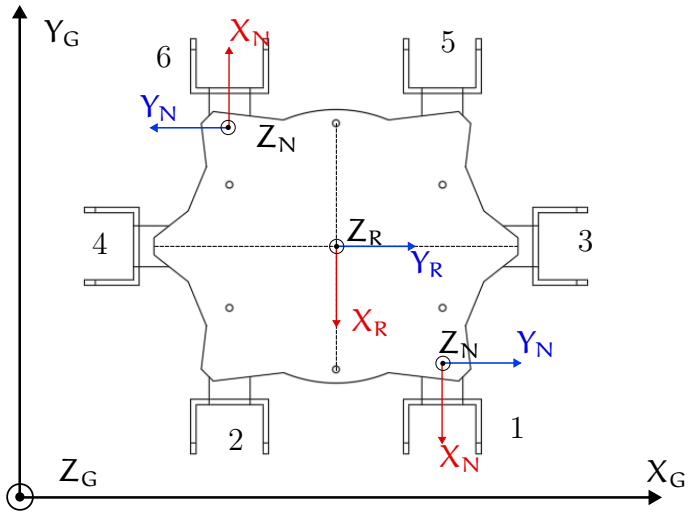
- l_1, l_2, l_3 – długości kolejnych członów (kolejno: biodro, udo, goleń),
- q_1, q_2, q_3 – kąty w przegubach nogi (biodro, kolano, kostka),
- $X_R Y_R Z_R$ – układ współrzędnych znajdujący się w geometrycznym środku korpusu robota, zorientowany osią X do przodu robota oraz osią Z w górę robota,
- $X_{N_i} Y_{N_i} Z_{N_i}$ – układy współrzędnych umieszczone w przegubach mocujących odnóża robota do korpusu,
- $X_G Y_G Z_G$ – globalny układ współrzędnych.

Wyżej opisane układy współrzędnych można zaobserwować na rysunku 3.1, przedstawiającym rzut korpusu i bioder robota na płaszczyznę $OX_G Y_G$. Transformacja między układem robota a lokalnym układem i-tej nogi dana jest wzorem

$$A_R^{N_i} = \mathrm{Trans}\left(X, x_0^i\right), \mathrm{Trans}\left(Y, y_0^i\right), \mathrm{Rot}\left(Z, \alpha_0^i\right), \tag{3.1}$$

gdzie x_0^i i y_0^i – współrzędne punktu mocowania i-tej nogi w układzie robota, α_0^i – kąt obrotu układu lokalnego względem osi Z. W implementacji przyjęto następującą konwencję orientacji:

- dla nóg 1,3,5 (lewa strona): $\alpha_0^i = 0$,
- dla nóg 2,4,6 (prawa strona): $\alpha_0^i = \pi$,



Rysunek 3.1: Rzut korpusu i bioder robota na płaszczyznę $X_G Z_G$

Po wyliczeniu transformacji (3.1) otrzymujemy następującą macierz jednorodną

$$T_0^i = \begin{bmatrix} c^i & 0 & 0 & x_0^i \\ 0 & c^i & 0 & y_0^i \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{3.2}$$

gdzie $c = \cos(\alpha_0^i)$ określa kierunek osi lokalnego układu współrzędnych – dla nóg prawych (2,4,6) osie X i Y są odwrócone względem układu nadrzędnego, co upraszcza algorytmy sterowania symetrycznymi nogami.

3.2 Kinematyka w ujęciu geometrycznym

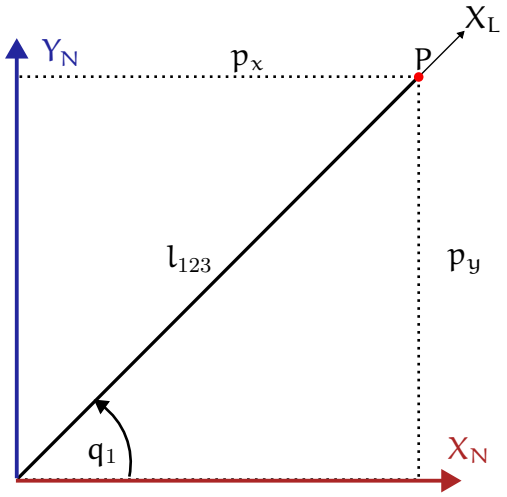
Model kinematyki prostej nogi jest podstawą do rozwiązania zadania kinematyki odwrotnej, niezbędnego do obliczenia kątów w przegubach nogi dla zadanego położenia końcówki (p_x^i, p_y^i, p_z^i). Odnóże robota, w przyjętej reprezentacji, jest modelem manipulatora trójrotacyjnego, co oznacza, że składa się z trzech przegubów obrotowych.

3.2.1 Kinematyka prosta

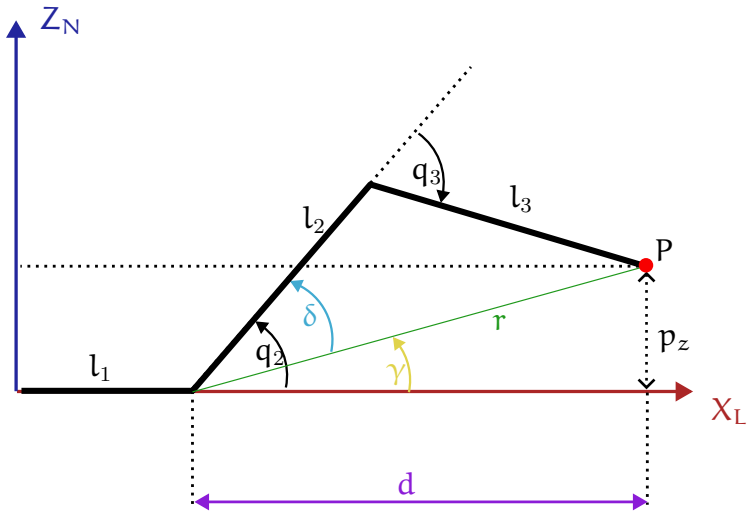
Kinematyka prosta definiuje położenie efektora (w reprezentowanym przypadku końca nogi robota) w lokalnym układzie współrzędnych nogi jako zestaw funkcji zależnych od kątów przegubów (q_1, q_2, q_3) oraz długości członów (l_1, l_2, l_3). Model kinematyki nogi można uzyskać analizując jej geometrię co zazwyczaj uzyskujemy korzystając z algorytmu *Denawida-Hartenberga* [DH55]. Z tejże analizy otrzymujemy następujące zależności opisujące położenie końca nogi w układzie współrzędnych jej mocowania $X_N Y_N Z_N$

$$\begin{cases} p_x = c_1(c_{23}\, l_3 + c_2\, l_2 + l_1) \\ p_y = s_1(c_{23}\, l_3 + c_2\, l_2 + l_1), \\ p_z = (s_{23}\, l_3 + s_2\, l_2) \end{cases} \tag{3.3}$$

gdzie $c_i = \cos(q_i)$, $s_i = \sin(q_i)$, $c_{ij} = \cos(q_i + q_j)$, $s_{ij} = \sin(q_i + q_j)$.



Rysunek 3.2: Rzut nogi na płaszczyznę $X_N Y_N$



Rysunek 3.3: Rzut nogi na płaszczyznę $X_L Z_N$

3.2.2 Zadanie kinematyki odwrotnej

Po wyliczeniu kinematyki prostej kolejnym kluczowym elementem jest obliczenie kinematyki odwrotnej co w niektórych przypadkach można zrobić również metodą geometryczną. Polega ona na wyznaczeniu kątów przegubów (q_1, q_2, q_3) na podstawie pozycji końcówki nogi (p_x, p_y, p_z), w lokalnym układzie współrzędnych nogi.

Dla kąta q_1 czyli przegubu obrotowego względem osi Z_N, jest on wyznaczany poprzez zrzutowanie nogi na płaszczyznę $X_N Y_N$ (co przedstawiono na rysunku 3.2) i użycie funkcji $\mathtt{atan2}$

$$q_1 = \mathtt{atan2}(p_y, p_x). \tag{3.4}$$

Kąty q_2 oraz q_3 (kolano i kostka) są liczone poprzez redukcję przestrzeni do płaszczyzny $OX_L Z_N$ dla $q_1 = 0$ tak jak pokazano na rysunku 3.3. Na płaszczyźnie, długość rzutu nogi na oś X_L opisana jest jako d i obliczana jest na podstawie twierdzenia Pitagorasa

$$d = \sqrt{p_x^2 + p_y^2} - l_1. \tag{3.5}$$

Wykorzystując właściwości geometryczne trójkąta zbudowanego z członów l_1, l_2 oraz wyliczonego odcinka $\sqrt{d^2 + p_z^2}$, na podstawie prawa cosinusów obliczany jest kąt q_3

$$\cos(q_3) = \frac{d^2 + p_z^2 - l_1^2 - l_2^2}{2 l_2 l_3}, \tag{3.6}$$

gdzie $q_3 = \operatorname{atan2}(s_3, c_3)$ oraz $s = \pm\sqrt{1 - c_3^2}$. Kąt q_2 wyznaczany jest na podstawie różnicy

$$q_2 = \gamma - \delta, \tag{3.7}$$

co daje zależność

$$q_2 = \operatorname{atan2}(p_z, d) - \operatorname{atan2}(l_3 s_3, l_2 + l_3 c_3). \tag{3.8}$$

3.3 Kinematyka w ujęciu jakobianowym

Alternatywnym do opisanego w podrozdziale 3.2 sposobem opisu kinematyki jest podejście jakobianowe [Bus09]. Kinematyka różniczkowa opisuje relację między prędkościami obrotowymi w przegubach $\dot{q}$ a prędkością efektora $\dot{p}$ (końca odnóża). Kluczowym elementem takiego opisu jest macierz Jacobiego oznaczona jako $J(q)$. Równanie podstawowe przyjmuje postać

$$\dot{p} = J(q)\dot{q}, \tag{3.9}$$

gdzie $\dot{p} = (\dot{p}_x, \dot{p}_y, \dot{p}_z)^T$ to wektor prędkości liniowej końcówki nogi a $\dot{q} = (\dot{q}_1, \dot{q}_2, \dot{q}_3)^T$ to wektor prędkości kątowych w przegubach.

3.3.1 Kinematyka prosta

Kinematyka prosta dla rozpatrywanego manipulatora trójczłonowego reprezentowana jest przez jakobian analityczny $J(q)$ będący macierzą o wymiarach 3×3. Elementy tejże macierzy wyznaczane są za pomocą pochodnych cząstkowych obliczanych na podstawie równań kinematyki prostej (3.3) w postaci

$$J(q) = \begin{bmatrix} \frac{\partial p_x}{\partial q_1} & \frac{\partial p_x}{\partial q_2} & \frac{\partial p_x}{\partial q_3} \\ \frac{\partial p_y}{\partial q_1} & \frac{\partial p_y}{\partial q_2} & \frac{\partial p_y}{\partial q_3} \\ \frac{\partial p_z}{\partial q_1} & \frac{\partial p_z}{\partial q_2} & \frac{\partial p_z}{\partial q_3} \end{bmatrix}. \tag{3.10}$$

Obliczając powyższe pochodne cząstkowe otrzymujemy

$$J(q) = \begin{bmatrix} -s_1(c_{23} l_3 + c_2 l_2 + l_1) & -c_1(s_{23} l_3 + s_2 l_2) & -c_1 s_{23} l_3 \\ c_1(c_{23} l_3 + c_2 l_2 + l_1) & -s_1(s_{23} l_3 + s_2 l_2) & -s_1 s_{23} l_3 \\ 0 & -(c_{23} l_3 + c_2 l_2) & -c_{23} l_3 \end{bmatrix}. \tag{3.11}$$

Jakobian ten można interpretować jako liniową aproksymację nieliniowego odwzorowania kinematyki prostej w ujęciu geometrycznym. Kolejne kolumny macierzy $J(q)$ reprezentują wpływ kolejnych przegubów na prędkość efektora.

3.3.2 Kinematyka odwrotna

Zadanie kinematyki odwrotnej w ujęciu różniczkowym polega na wyznaczeniu prędkości kątowych w przegubach $\dot{q}$ na podstawie zadanej prędkości końcówki $\dot{p}$ [SHV20]. Wykorzystane zostało przybliżenie, w którym dla niewielkich przemieszczeń zmiana pozycji Δp traktowana jest jako wynik ruchu z chwilową prędkością w krótkim czasie Δt.

$$\Delta p \approx J(q)\Delta q \tag{3.12}$$

Dla kwadratowej macierzy $J(q)$ rozwiązanie idealne można otrzymać dzięki obliczeniu jej odwrotności z równania (3.9)

$$\dot{q} = J^{-1}(q)\dot{p}. \tag{3.13}$$

Metoda ta, ma jednak wadę w postaci wyznacznika macierzy dążącego do zera w konfiguracjach osobliwych. Powoduje to, że wartości obliczonych prędkości $\dot{q}$ dążą do nieskończoności, czego skutkiem jest niestabilne zachowanie robota. Dzielenia przez zerowy wyznacznik macierzy można uniknąć dzięki zastosowaniu *transpozycji jakobianu* [Bus09]. W przeciwieństwie do metody geometrycznej, która oblicza kąty bezpośrednio ze wzorów analitycznych, metoda jakobianowa wykorzystuje proces iteracyjny do stopniowego redukowania błędu pozycjonowania. Dla zadanej pozycji docelowej końcówki $p_d = (p_x, p_y, p_z)^T$ w układzie nogi, szukamy kątów $q = (q_1, q_2, q_3)^T$, które minimalizują błąd

$$e = p_d - p, \tag{3.14}$$

gdzie p obliczane jest z kinematyki prostej (3.3) dla aktualnych wartości kątów q.

Algorytm realizuje metodę najszybszego spadku (Gradient Descent). Iloczyn transponowanego jakobianu i wektora błędu wskazuje kierunek w przestrzeni złączy, który najszybciej redukuje błąd pozycji [Bus09]

$$q^{k+1} = q^k + \alpha J^T(q^k)e^k, \tag{3.15}$$

gdzie $q^{(k)}$ – konfiguracja przegubów w k-tej iteracji, α – współczynnik uczenia (krok algorytmu) natomiast e^k – wektor błędu pozycji z równania (3.14). Transpozycja macierzy oznacza zamianę wierszy z kolumnami

$$(J^T)_{ij} = J_{ji}. \tag{3.16}$$

Po rozpisaniu równania (3.15) na składowe dla trzech przegubów, otrzymujemy wzory

$$\begin{aligned} \Delta q_1 &= \alpha(J_{11}\, e_x + J_{21}\, e_y + J_{31}\, e_z), \\ \Delta q_2 &= \alpha(J_{12}\, e_x + J_{22}\, e_y + J_{32}\, e_z), \\ \Delta q_3 &= \alpha(J_{13}\, e_x + J_{23}\, e_y + J_{33}\, e_z). \end{aligned} \tag{3.17}$$

W powyższym wzorze składowe błędu mnożone są przez elementy odpowiednich kolumn oryginalnej macierzy, które odpowiadają wierszom macierzy transponowanej. Po obliczeniu przyrostów Δq_i z równania (3.17), aktualizowane są kąty przegubów. Iteracja kontynuowana jest do momentu spełnienia warunku zbieżności

$$\|e\| = \sqrt{\langle e, e\rangle} < \varepsilon_{tol}, \tag{3.18}$$

gdzie ε_{tol} jest tolerancją błędu. Jeśli rozwiązanie nie zostanie znalezione w określonej liczbie iteracji, algorytm przerywa działanie, zwracając ostatnią najlepszą konfigurację.

Rozdział 4

Sterowanie robota ze zmianą orientacji

Sterowanie robotem kroczącym nie ogranicza się jedynie do wykonania kroku. Jego istotnym elementem jest również kontrola orientacji i pozycji samego korpusu względem podłoża, przy zachowaniu stałego punktu podparcia nóg. Jest to zadanie logicznie odwrotne do kontroli manipulatora. Polega ono na przemieszczeniu korpusu, którego środek pozostaje na stałej wysokości, względem nieruchomych końców odnóży. Z punktu widzenia kinematyki realizowany układ wykazuje bezpośrednią analogię do działania równoległego manipulatora typu platforma Stewarta [Ste65]. W pracy wybrano i zaimplementowano dwa podejścia do sterowania:

1. Metoda geometryczna [Zie03] – oparta na rozłożeniu trójwymiarowego obrotu na trzy niezależne rzuty na płaszczyzny układu współrzędnych. Jest to intuicyjna metoda, która wymaga niewielkich nakładów obliczeniowych.

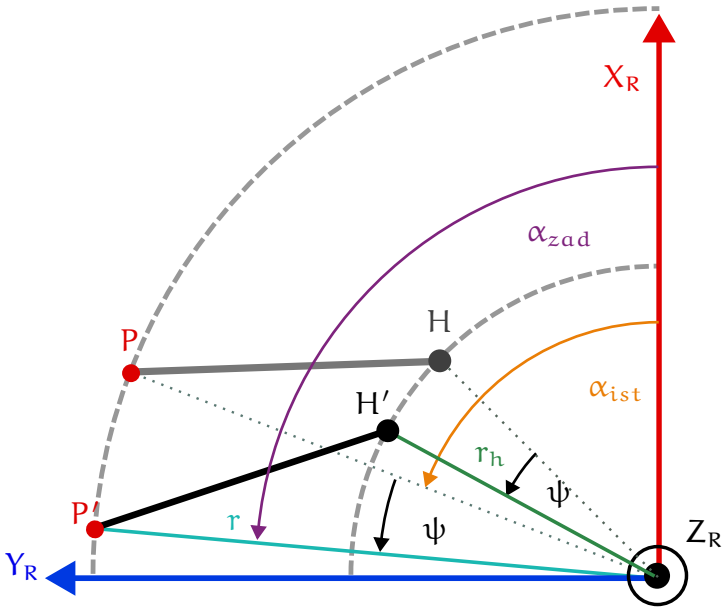
2. Metoda macierzowa [Cra05, SHV20] – wykorzystująca jednorodne macierze transformacji. Jest to standardowe podejście w robotyce pozwalające na zwięzły zapis matematyczny złożonych ruchów.

Układ sterowania kolejno najpierw oblicza zadaną orientację robota, następnie na podstawie nowych pozycji buduje trajektorię chodu. Gotowe zadane pozycje są następnie przekazywane do układu liczenia kinematyki odwrotnej zwracającego nastawy kątów realizujące umiejscowienie końca odnóża w zadanej pozycji. W rozdziale opisano podstawy matematyczne obu wyżej wspomnianych metod sterowania orientacją oraz sposób generowania trajektorii chodu.

4.1 Metoda geometryczna

Podejście geometryczne polega na analizowaniu zmian w położeniu mocowań nóg względem końców odnóży na skutek obrotu korpusu [Zie03]. Operacje w przestrzeni trójwymiarowej zostały rozbite na trzy niezależne transformacje płaskie. Odpowiadają one kątom Eulera – Roll (ϕ), Pitch (θ) oraz Yaw (ψ) [Cra05]. Metoda wykorzystuje analizę trójkątów powstałych w wyniku pochylenia korpusu oraz ich podstawowych zależności trygonometrycznych. Dla ułatwienia obliczeń i poprawienia czytelności zapisu poniżej wprowadzono parametry pomocnicze:

- l_b – połowa długości korpusu,



Rysunek 4.1: Transformacja Yaw – geometria w płaszczyźnie $X_R Y_R$, gdzie H to pozycja biodra przed obrotem, natomiast H′ to pozycja biodra po obrocie

- w_b – połowa szerokości korpusu,
- z_h – pionowe przesunięcie biodra względem centrum korpusu.

4.1.1 Obrót w płaszczyźnie XY – transformacja Yaw

Obrót wokół osi pionowej Z_R jest realizowany poprzez zmiany współrzędnych końca nogi w płaszczyźnie $X_R Y_R$. Koniec odnóża pozostaje w miejscu, natomiast korpus się obraca. Z punktu widzenia lokalnego układu nogi, koniec efektora przemieszcza się po łuku. Opisane niżej transformacje zobrazowane są na rysunku 4.1.

Konieczne jest przejście z lokalnego układu nogi $X_N Y_N Z_N$ do układu robota $X_R Y_R Z_R$. Realizowane jest ono za pomocą następujących równań

$$\begin{pmatrix} x_w \\ y_w \end{pmatrix} = \begin{pmatrix} x_l + x_0^i \\ y_l + y_0^i \end{pmatrix} \tag{4.1}$$

gdzie (x_0^i, y_0^i) to pozycja biodra i-tej nogi przed obrotem. Realizację ruchu po łuku najłatwiej jest opisać w przestrzeni współrzędnych biegunowych, możliwe dzięki zastosowaniu następujących równań

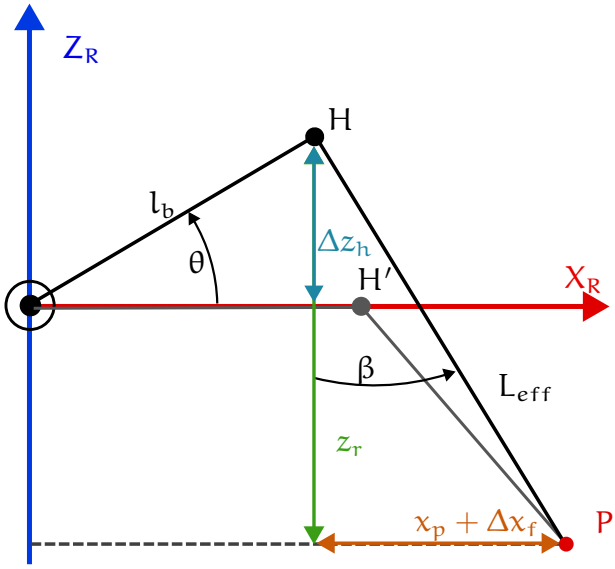
$$\alpha_{ist} = \operatorname{atan2}(y_w, x_w) \tag{4.2}$$

$$r = \sqrt{x_w^2 + y_w^2}, \tag{4.3}$$

gdzie α_{ist} jest kątem pomiędzy osią X_R a wektorem poprowadzonym od środka układu robota $X_R Y_R Z_R$ położonym w centrum korpusu do wyjściowego położenia biodra. Nowa pozycja końca nogi obliczana jest poprzez dodanie zadanego kąta ψ

$$\alpha_{zad} = \alpha_{ist} + \psi, \tag{4.4}$$

$$\begin{pmatrix} x'_w \\ y'_w \end{pmatrix} = \begin{pmatrix} r\cos(\alpha_{zad}) \\ r\sin(\alpha_{zad}) \end{pmatrix}. \tag{4.5}$$



Rysunek 4.2: Transformacja Pitch – geometria w płaszczyźnie $Z_R X_R$, gdzie H to pozycja biodra przed obrotem, natomiast H′ to pozycja biodra po obrocie

Należy również uwzględnić zmianę pozycji mocowania nogi. Biodro przemieszcza się po okręgu o promieniu

$$r_h = \sqrt{(x_0^i)^2 + (y_0^i)^2}, \tag{4.6}$$

zatem nowa pozycja obliczana jest na podstawie równań analogicznych do obliczania pozycji końca nogi

$$\begin{pmatrix} x_0'^i \\ y_0'^i \end{pmatrix} = \begin{pmatrix} r_h \cos(\alpha_h + \psi) \\ r_h \sin(\alpha_h + \psi) \end{pmatrix}. \tag{4.7}$$

By powrócić do lokalnego układu nogi należy odwrócić zależność z równania (4.1)

$$\begin{pmatrix} x'_{leg} \\ y'_{leg} \end{pmatrix} = \begin{pmatrix} x'_w - x_0'^i \\ y'_w - y_0'^i \end{pmatrix}. \tag{4.8}$$

4.1.2 Obrót w płaszczyźnie XZ – transformacja Pitch

Transformacja Pitch odbywa się w płaszczyźnie $X_R Z_R$. Jej efektem jest zmiana wysięgu nóg przednich i tylnych, wpływających na pochylenie robota do przodu lub tyłu oraz zmianę wysokości początku lokalnych układów współrzędnych poszczególnych nóg. Opisane niżej transformacje zobrazowane są na rysunku 4.2. Zachowanie uniwersalności obliczeń dla nóg wymaga wprowadzenia znaku S_p, który wynosi -1 dla nóg przednich i $+1$ dla nóg tylnych. Wysokość nóg środkowych (3 i 4) pozostaje stała, ponieważ nie chcemy zmienić wysokości środka korpusu. Obliczenia zaczynamy od zmiany wysokości biodra Δz_h. Zmiana ta wynika z pochylenia korpusu o kąt θ i wynosi

$$\Delta z_h = l_b S_p \sin(\theta). \tag{4.9}$$

Zmiana wysokości końca odnóż obliczana jest na podstawie wzoru

$$z_r = z_p - \Delta z_h, \tag{4.10}$$

gdzie z_p jest pierwotną wysokością biodra. Długość korpusu jest rzutowana na oś $X_R Z_R$, czego skutkiem jest pozorne przesunięcie biodra w osi X. Przesunięcie to jest kompensowane za pomocą zmiany pozycji końca odnóża

$$\Delta x_f = l_b S_p (1 - \cos(\theta)). \tag{4.11}$$

Kluczowym etapem obliczenia nowej pozycji końca nogi jest przejście do biegunowego układu współrzędnych, którego początek znajduje się w miejscu zamocowania biodra. W tym celu wyznaczany jest pomocniczy kąt β oraz długość efektywna rzutu nogi L_{eff}, obliczona z własności trójkąta prostokątnego o przyprostokątnych $(x_p + \Delta x_f)$ oraz z_r,

$$\beta = atan2(x_p + \Delta x_f, z_r), \tag{4.12}$$

$$L_{eff} = \frac{z_r}{\cos(\beta)}, \tag{4.13}$$

gdzie x_p to pierwotna odległość końca nogi do biodra, Δx_f jest zmianą odległości między końcem odnóża a biodrem, natomiast z_r jest odległością między biodrem a końcem nogi w osi Y. Docelowy kąt nachylenia nogi β_{cal} jest sumą kąta pierwotnego i zadanego kąta pochylenia korpusu

$$\beta_{cal} = \beta + (\theta S_p). \tag{4.14}$$

Końcowe współrzędne końca nogi po obrocie otrzymujemy poprzez powrót do kartezjańskiego układu współrzędnych

$$\begin{pmatrix} z' \\ x' \end{pmatrix} = \begin{pmatrix} L_{eff}\cos(\beta_{cal}) \\ L_{eff}\sin(\beta_{cal}) \end{pmatrix}. \tag{4.15}$$

Ostatnim krokiem jest dopasowanie końcowej wartości x' do lokalnego układu współrzędnych nogi. Jak możemy zobaczyć na rysunku 3.1, układy nóg tylnych i przednich są obrócone wobec siebie o 180°. Co za tym idzie, aby otrzymać prawidłową wartość zmiennej x' należy odwrócić znak współrzędnej poziomej poprzez wymnożenie jej przez (-1).

4.1.3 Obrót w płaszczyźnie YZ – transformacja Roll

Transformacja Roll – przechył boczny odbywa się w płaszczyźnie $Y_R Z_R$. Algorytmy obliczeń są analogiczne do transformacji Pitch. Zmianie ulega płaszczyzna rzutowania oraz parametry konstrukcyjne. Operacje są przeprowadzane na współrzędnych (y, z) a wymiarem jest w_b. Zamiast rozróżniać nogi na przednie i tylne stosujemy podział na prawe i lewe, ponownie wprowadzając parametr zmiany znaku S_r wynoszący $+1$ dla nóg lewych (1, 2, 3) oraz (-1) dla nóg prawych (2, 4, 6). Równania przyjmują postać tożsamą z (4.9)–(4.15). Zastępujemy kąt θ kątem ϕ. Wynikiem przeprowadzonej transformacji są współrzędne (y', z') w układzie nogi, a ich ostateczne wyprowadzenie wygląda następująco

$$\begin{pmatrix} z' \\ y' \end{pmatrix} = \begin{pmatrix} L_{eff}\cos(\gamma_{cal}) \\ L_{eff}\sin(\gamma_{cal}) \end{pmatrix}, \tag{4.16}$$

gdzie kąt γ_{cal} jest kątem pomocniczym i pełni rolę analogiczną do kąta β_{cal}. Podobnie jak w przypadku transformacji Pitch, konieczne jest dopasowanie wyniku do lokalnych układów współrzędnych, mnożąc wynikową pozycję y' dla nóg prawych (2, 4, 6) przez (-1).

4.1.4 Kolejność transformacji

Poprawne wyliczenie transformacji wymaga zastosowania następującej kolejności:

1. **Obrót w płaszczyźnie** XY (równania 4.8),
2. **Obrót w płaszczyźnie** XZ (równania 4.15),
3. **Obrót w płaszczyźnie** YZ (równania 4.16).

Kolejność ta dyktowana jest faktem, że powyższe operacje nie są przemienne. Oznacza to, że zmieniając kolejność obliczeń otrzymamy różne pozycje końcowe dla odnóży.

4.2 Metoda macierzy jednorodnej

Alternatywnym podejściem obliczania orientacji robota jest wykorzystanie macierzy jednorodnych (ang. *homogeneous transformation matrices*), łączących rotację i translację w jednej operacji [Cra05, SHV20]. Przewagą stosowania omawianej metody jest możliwość przeprowadzenia zmiany zarówno orientacji jak i translacji korpusu w jednej operacji. Polega ona na wyznaczeniu macierzy, które odpowiadają kolejno obrotom o kąty Eulera Roll (ϕ) wokół osi X_R, Pitch (θ) w osi Y_R oraz Yaw (ψ) w osi Z_R. Elementem realizującym operację translacji korpusu jest wektor $(t_x, t_y, t_z)^T$ pozwalającym na przesunięcia we wcześniej wspomnianych osiach.

Do poprawnej realizacji algorytmu konieczne jest zdefiniowanie nowego wirtualnego układu odniesienia (ang. *reference frame*), który pokrywa się z układem robota. Wszystkie operacje na korpusie robota zostają przeprowadzone względem tego właśnie układu. W procesie transformacji przyjęto następujące oznaczenia:

- P_N – pozycja końcówki nogi w układzie odniesienia,
- P_B – stała pozycja mocowania biodra zdefiniowana w lokalnym układzie korpusu,
- P_{Bd} – pozycja mocowania biodra w układzie odniesienia po wykonaniu transformacji.

Celem operacji jest wyznaczenie zmian w pozycjach końców poszczególnych końców odnóży, wynikających z transformacji przez macierz jednorodną $T \in R^{4\times4}$ mającą postać

$$T = \begin{bmatrix} R & t \\ 0_{1\times3} & 1 \end{bmatrix}, \tag{4.17}$$

gdzie R reprezentuje rotację, wektor $t = (t_x, t_y, t_z)^T$ reprezentuje translację, a dolny wiersz $(0_{1\times3}1)$ jest stały i służy do zachowania spójności wymiarowej macierzy. Punkt w przestrzeni jest reprezentowany jako wektor

$$p = (x, y, z, 1)^T, \tag{4.18}$$

natomiast macierze rotacji wokół poszczególnych osi kartezjańskich w formie jednorodnej rozszerzonej do wymiaru 4×4 definiowane są jako:

- macierz rotacji wokół osi X (Roll, ϕ)

$$R_x(\phi) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi & 0 \\ 0 & \sin\phi & \cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{4.19}$$

- macierz rotacji wokół osi Y (Pitch, θ)

$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{4.20}$$

- macierz rotacji wokół osi Z (Yaw, ψ)

$$R_z(\psi) = \begin{bmatrix} \cos\psi & -\sin\psi & 0 & 0 \\ \sin\psi & \cos\psi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{4.21}$$

Mnożenie przez macierz 4×4, wymaga aby każdy punkt w przestrzeni trójwymiarowej reprezentowany był jako wektor rozszerzony

$$p = (x, y, z, 1)^T. \tag{4.22}$$

Finalna macierz rotacji R rozszerzona o wektor t tworzy macierz transformacji T (wcześniej opisaną równaniem (4.17)). Macierz ta pozwala na podstawie pozycji biodra P_B w układzie robota wyliczyć nową pozycję biodra P_{Bd} w układzie odniesienia

$$P_{Bd} = TP_B. \tag{4.23}$$

Ostatecznym celem operacji jest uzyskanie wektora sterującego dla nogi (P_d). Łączy on przemieszczone biodro z ustalonym punktem podparcia końca nogi. Jest on wyznaczany jako różnica pozycji w układzie odniesienia

$$P_d = P_N - P_{Bd}. \tag{4.24}$$

Otrzymane składowe wektora P_d stanowią dane wejściowe kinematyki odwrotnej.

4.3 Algorytm chodu

Podstawowym zadaniem układu sterowania robota jest zadanie chodu. Jego głównym celem jest wyznaczenie chwilowych pozycji zadanych dla końca każdej z nóg w czasie rzeczywistym w celu płynnego przesuwania platformy robota. W pracy zastosowano jeden algorytm bazujący na chodzie trójpodporowym (ang. *tripod gait*) [Zie03] – w każdym momencie trzy nogi są w fazie podporowej (*retrakcji*), a pozostałe trzy w fazie przenoszenia (*protrakcji*), przy zastosowaniu następującego podziału:

- grupa 1: nogi $\{1, 4, 5\}$ (lewy przód, prawy środek, lewy tył),
- grupa 2: nogi $\{2, 3, 6\}$ (prawy przód, lewy środek, prawy tył).

Trajektoria pojedynczej nogi opisana jest parametrycznie przez fazę $\varphi \in [0, 1)$ cyklu chodu. Cykl chodu składa się z dwóch naprzemiennych faz:

1. fazy przenoszenia (protrakcji) – nogi z aktywnej grupy są unoszone i przemieszczane w kierunku ruchu robota,
2. fazy podparcia (retrakcji) – nogi z drugiej grupy pozostają na podłożu i przesuwają się względem korpusu w zwrocie przeciwnym do ruchu, generując tym samym siłę napędową przemieszczającą robota do przodu.

4.3.1 Matematyczny opis algorytmu chodu

Trajektoria końca nogi jest generowana w układzie odniesienia robota $\mathsf{X_R Y_R Z_R}$. Tworzona jest ona na podstawie parametrów wejściowych, którymi są długość kroku L oraz jego wysokość H. W fazie protrakcji ruch końca nogi opisywany jest równaniem

$$\begin{cases} x_{sw}(s) = -\frac{L}{2} + Ls \\ z_{sw}(s) = z_{zad} + H\sin(\pi s) \end{cases}, \tag{4.25}$$

gdzie s to lokalny postęp fazy przenoszenia ($0 \leqslant s \leqslant 1$). Ruch w osi X jest liniowy, natomiast wznoszenie w osi Z realizowane jest po funkcji sinusoidalnej. W fazie podparcia koniec nogi przesuwa się do tyłu na stałej wysokości w osi Z względem korpusu, co opisane jest równaniem

$$\begin{cases} x_{st}(s) = \frac{L}{2} - Ls \\ z_{st}(s) = z_{zad} \end{cases}, \tag{4.26}$$

gdzie x_{st} oraz z_{st} są chwilowymi stanami współrzędnych, natomiast z_{zad} jest zadaną wysokością, na której znajduje się korpus.

4.3.2 Transformacja do lokalnych układów współrzędnych

Zaplanowana powyżej trajektoria jest następnie formatowana do lokalnego układu współrzędnych nogi, poprzez dodanie właściwych przesunięć $(x_{0_i} y_{0_i})$ oraz uwzględnienie rotacji α_i

$$\begin{bmatrix} \Delta x_{l_i} \\ \Delta y_{l_i} \end{bmatrix} = \begin{bmatrix} \cos(\alpha_i) & \sin(\alpha_i) \\ -\sin(\alpha_i) & \cos(\alpha_i) \end{bmatrix} \begin{bmatrix} \Delta x_{0_i} \\ \Delta y_{0_i} \end{bmatrix}. \tag{4.27}$$

W przypadku rozważanego robota, obroty względem układu robota przyjmują wartości 0 dla nóg przednich (1,2), $\pm\pi/2$ dla nóg środowych (3,4) oraz π dla nóg tylnych (5,6).

Rozdział 5

Implementacje modeli

W rozdziale opisano implementację modelu robota w systemie symulacji. Na wstępie zaprezentowano architekturę oprogramowania wraz z przedstawieniem połączeń między węzłami w środowisku ROS 2 oraz Gazebo. Opisano działanie kolejnych węzłów realizujących obliczanie kinematyki odwrotnej oraz sterowników zadających zmiany w orientacji korpusu oraz trajektorię chodu. Implementacja została napisana w języku C++ [Str13].

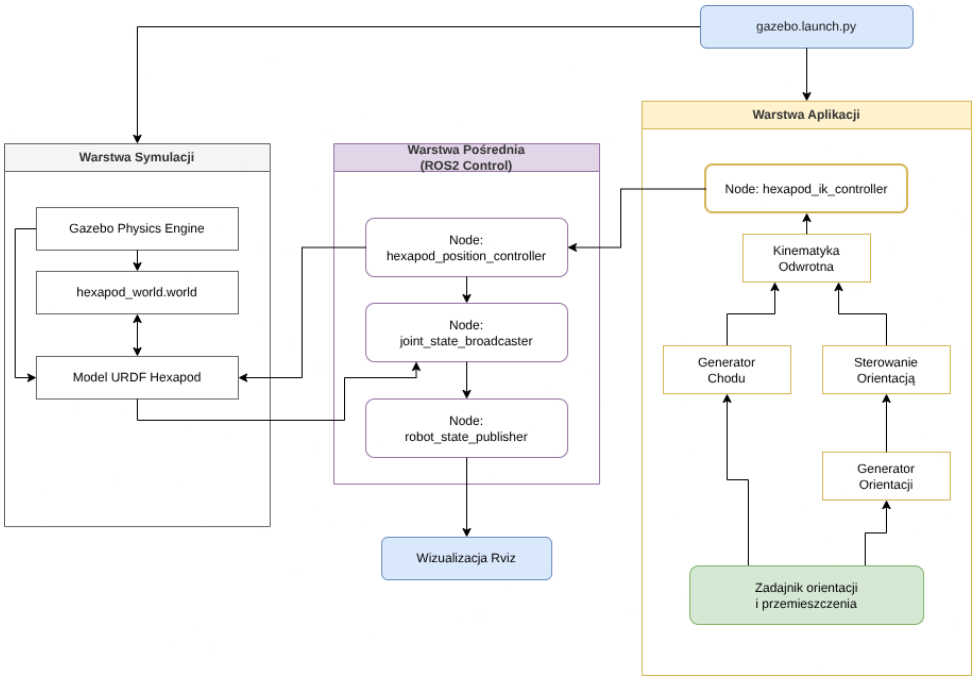
5.1 Architektura

System jest zaprojektowany w oparciu o rozproszoną strukturę środowiska ROS 2 [MFG+22]. Rysunek 5.1 przedstawia diagram przepływu danych między poszczególnymi warstwami logicznymi. Architektura dzieli się na cztery główne komponenty. Warstwa aplikacji zawiera logikę działania robota. W jej skład wchodzą mechanizmy obliczeniowe kinematyki odwrotnej, zadajniki orientacji korpusu, generator trajektorii chodu oraz pomocnicze programy zadające trajektorie testowe. Warstwa pośrednia zawiera mechanizmy komunikacji pomiędzy warstwą aplikacji a symulacją oraz obsługę strumieni zapisujących dane pomiarowe do plików *csv*. Warstwa symulacji obejmuje systemy uruchamiające symulację oraz właściwą interpretację danych przesyłanych przez warstwę pośrednią.

Kluczowym elementem konstrukcji systemu jest klasa `HexapodControllerBase`, która ustala standardową komunikację. Zawiera podstawowe mechanizmy przesyłu danych na odpowiednie tematy ROS 2, kinematykę prostą oraz parametry robota. Dzięki zastosowaniu polimorfizmu parametry te mogą być wydzielone i stosowane do wszystkich dziedziczących klas, które korzystają z tych samych mechanizmów.

5.2 Implementacja algorytmów kinematyki odwrotnej

Zadaniem modułu kinematyki odwrotnej jest transformacja zadanej pozycji końca odnóża w kartezjańskim układzie współrzędnych na zestaw kątów w przegubach. Poniżej przedstawiono implementację dwóch algorytmów służących temu celowi – w konwencji obliczeń geometrycznych (podrozdział 5.2.1) oraz poprzez wyliczenie jakobianu analitycznego (podrozdział 5.2.2). Obydwie implementacje korzystają z klasy bazowej `hexapod_conttroller_base` zawierającej mechanizmy wspólne dla obu podejść liczenia kinematyki odwrotnej.



Rysunek 5.1: Diagram komunikacji węzłów ROS

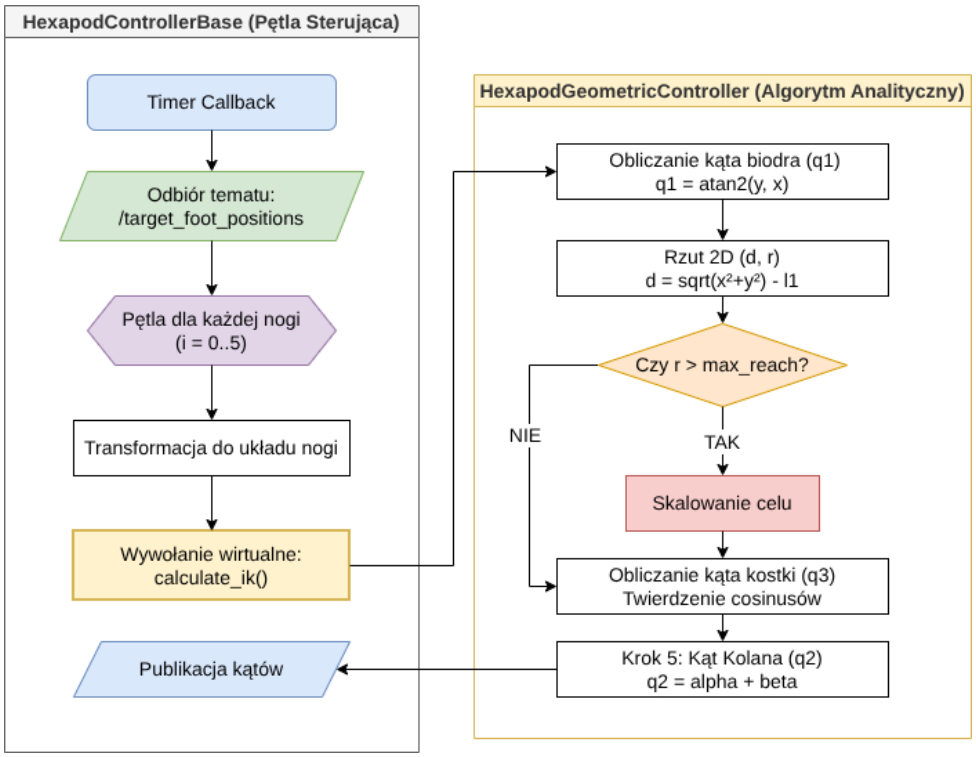
5.2.1 Algorytm geometryczny

Implementacja algorytmu bazującego na analizie geometrycznej znajduje się w klasie `HexapodGeometricController`, której schemat blokowy przedstawiony jest na rysunku 5.2. Jest ona implementacją wyprowadzeń matematycznych zawartych w podrozdziale 3.2. Funkcja licząca transformacje jest zdefiniowana w kodzie jako `calculate_leg_ik`. Przyjmuje ona jako argument współrzędne celu $p_d = (p_x, p_y, p_z)^T$ oraz numer nogi, dla której liczona jest kinematyka odwrotna. Zwraca natomiast nastawy kątów w przegubach $q = (q_1, q_2, q_3)^T$. Metodologia obliczeń ma następującą kolejność:

1. wyznaczenie kąta biodra (q_1) sterującego pozycją biodra poprzez zastosowanie funkcji `atan2`,
2. redukcja do problemu płaskiego – punkt w trzech wymiarach rzucany jest na płaszczyznę XY,
3. weryfikacja zasięgu – przed wykonaniem kolejnych obliczeń wykonywana jest sprawdzenie czy punkt docelowy znajduje się w zasięgu ramienia ($r \leqslant l_2 + l_3$), co zapobiega błędom numerycznym i nieciągłościom ruchu,
4. wyznaczenie kątów kolana i kostki – z twierdzenia cosinusów wyznaczany jest kąt q_3 a następnie za pomocą sumy kątów wyznaczanych w równaniu 3.8, wyznaczany jest kąt q_2.

5.2.2 Algorytm jakobianowy

Podejście alternatywne, zaimplementowano w klasie `HexapodJacobianController`. Wykorzystuje ono iteracyjną metodę liczenia jakobianu analitycznego, której wyprowadzenie



Rysunek 5.2: Diagram implementacji liczenia kinematyki odwrotnej w konwencji analizy geometrycznej

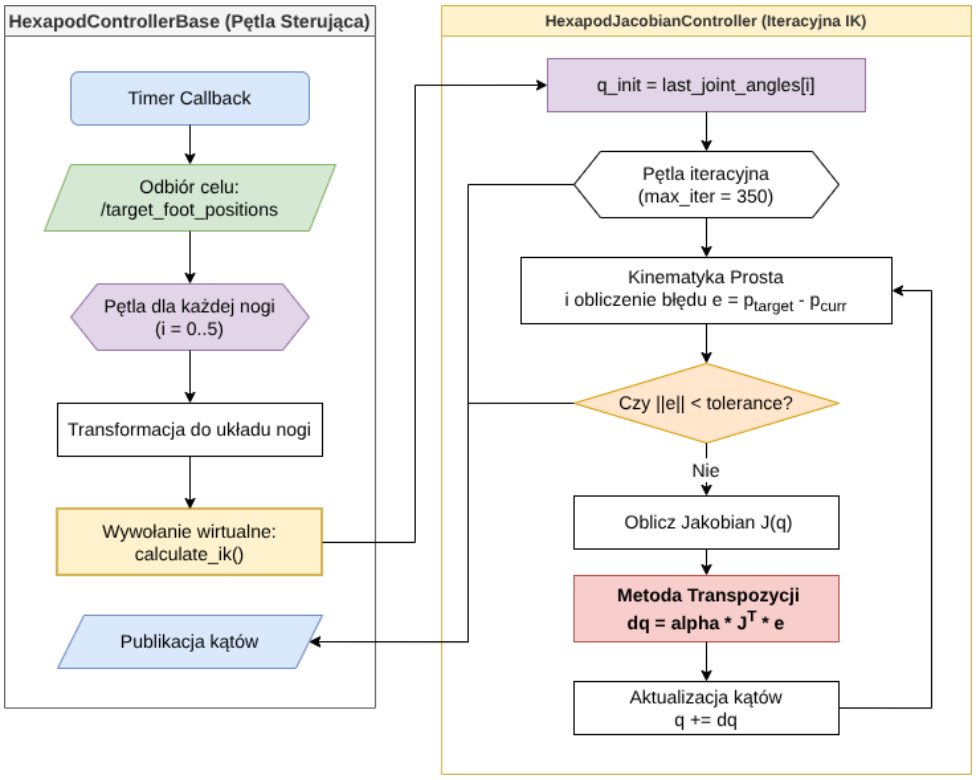
matematyczne zostało wyjaśnione w podrozdziale 3.3. W implementacji zastosowano metodę transpozycji jakobianu, która jest niewymagająca obliczeniowo. Schemat blokowy tej metody przedstawiony jest na rysunku 5.3.

Kolejność operacji w głównej pętli wykonawczej prezentuje się następująco:

1. inicjalizacja – jako punkt początkowy przyjmowana jest pozycja robota z ostatniego wykonanego w symulacji kroku,
2. wyznaczenie błędu (e) – polega na wyznaczeniu różnicy pomiędzy pozycją zadaną a aktualną,
3. obliczenie jakobianu analitycznego,
4. aktualizacja kątów – poszukiwanie nowych nastaw dla przegubów realizowane jest zgodnie ze wzorem (3.15), gdzie ustalony współczynnik uczenia α jest równy 0.5 co pozwala zachować balans pomiędzy dokładnością i prędkością obliczeń,
5. warunek stopu – pętla przerywana jest, w momencie gdy norma błędu spadnie poniżej założonego parametru tolerancji (tutaj $\varepsilon_{tol} = 0.001$) lub po przekroczeniu maksymalnej liczby iteracji (tutaj $N_{max} = 350$).

5.3 Implementacja sterowania orientacją

Aby przechylić korpus o zadany kąt należy obliczyć właściwe pozycje końca odnóży co w niniejszej pracy realizowane jest dwojako. W sposób geometryczny opisany w podroz-



Rysunek 5.3: Diagram implementacji liczenia kinematyki odwrotnej w konwencji liczeneia jakobianu analitycznego

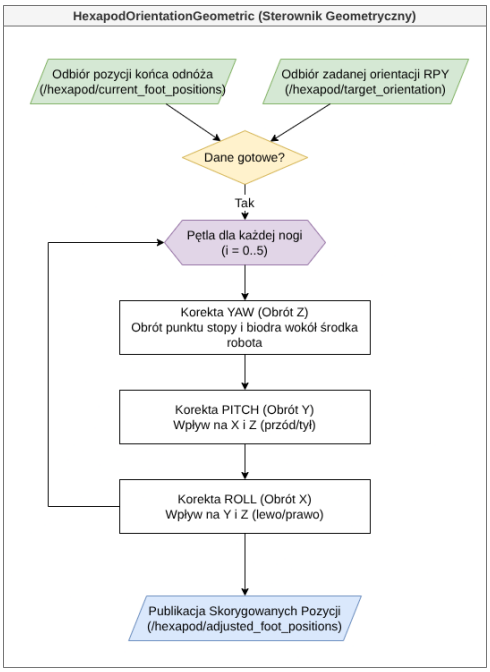
dziale 5.3.1 oraz metodą macierzy jednorodnej przedstawioną w podrozdziale 5.3.2. Moduł sterowania orientacją nie zwraca bezpośrednich nastaw dla przegubów lecz zadaną pozycję końca nogi, stanowiącą dane wejściowe do jednego z omówionych wyżej algorytmów obliczania kinematyki odwrotnej.

5.3.1 Metoda geometryczna

Klasa `HexapodOrientationGeometric` stosuje rozkład problemu orientacji na trzy płaszczyzny obliczeń. Dlatego też implementacja tego algorytmu działa sekwencyjnie, obliczając kolejno obroty w osi Z – Yaw, osi Y – Pitch a na koniec osi X – Roll. Przeprowadzona zostaje również korekcja wysokości w osi Z. Matematyczne podstawy do przeprowadzenia kolejnych operacji obrotu przedstawione zostały w podrozdziale 4.1 natomiast schemat blokowy implementacji można zobaczyć na rysunku 5.4. Działanie algorytmu sprowadza się do sprawdzenia czy dane są prawidłowo przesyłane, uruchomienia pętli sterowania dla każdej z kolejnych nóg i wykonania obliczeń w wyżej przestawionej kolejności. Na koniec dane są zwracane poprzez publikację na zdefiniowany temat.

5.3.2 Metoda macierzy jednorodnej

Alternatywna implementacja wykorzystująca macierze jednorodne zawarta jest w klasie `HexapodOrientationMatrix`. Daziała bazując na macierzach rotacji R_x, R_y, R_z, które zostały szczegółowo opisane w podrozdziale 4.2. W kodzie zdefiniowano struktury `Matrix4x4` oraz funkcje pomocnicze do ich mnożenia i transformacji punktów. Pozwa-



Rysunek 5.4: Diagram implementacji sterownika geometrycznego orientacji

la to na realizację rotacji i translacji w sześciu stopniach swobody jednocześnie. Schemat blokowy obrazujący działanie klasy można zobaczyć rysunku 5.5. Implementacja omawianej klasy działa poprzez odebranie i sprawdzenie poprawności wymaganych danych wejściowych, uruchomienie pętli obliczającej nowe pozycje dla każdej z nóg po których to obliczeniach nowa pozycja publikowana jest na temat przekazujący informacje do mechanizmów obliczeniowych kinematyki odwrotnej

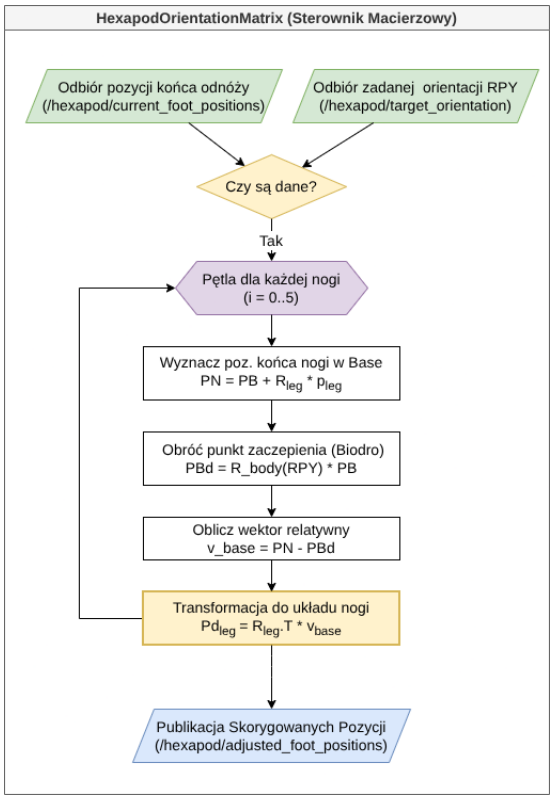
5.4 Generator trajektorii chodu

Implementacja sterowania trajektorią chodu jest realizowana na bazie klasy programu `hexapod_gait_controller`. Opisana jest ona na rysunku 5.6. Logika sterowania zbudowana jest na podstawie prostej maszyny stanów, ustalającej dwie fazy – TRANSITIONING będącej fazom inicjalizacji uruchamianą po wywołaniu klasy sterującej chodem. W tej fazie robot ustawiany jest w stabilnym pozycji stojącej, następnie odczekuje 3 sekundy by następnie przejść do fazy WALKING realizującej matematyczny model chodu w następujących krokach:

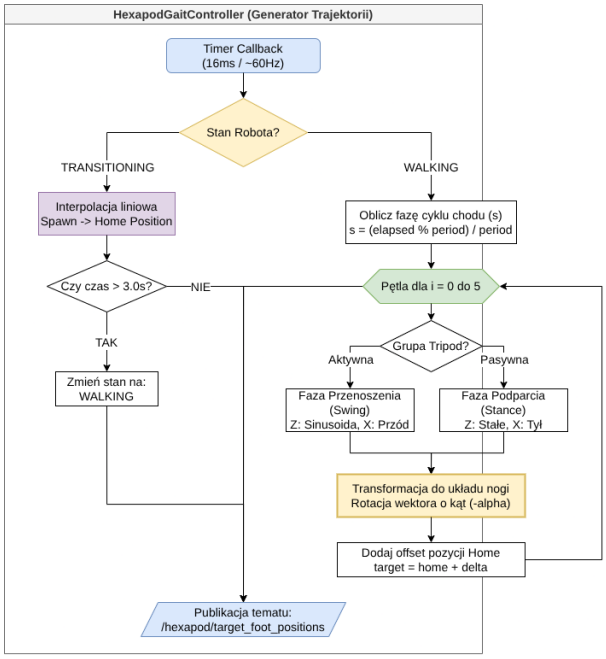
1. wyznaczenie fazy – postęp cyklu $s \in [0, 1]$ ustalany jest na podstawie czasu systemowego,

2. generacja profilu ruchu – w zależności od realizowanej fazy (protrakcji lub retrakcji), wyznaczane jest przesunięcie w układzie robota – kształt pół-sinusoidy lub prostej,

3. transformacja współrzędnych – konieczne jest przetransformowanie trajektorii do lokalnego układu współrzędnych iterowanej nogi $X_L Y_L Z_L$ za pomocą macierzy obrotu o kąt $-\alpha_i$.



Rysunek 5.5: Diagram implementacji sterownika macierzowego orientacji



Rysunek 5.6: Diagram aktywności generatora chodu. Widoczny podział na fazę inicjalizacji oraz cykliczną pętlę generowania kroków dla poszczególnych nóg

Obliczone pozycje końców nóg przesyłane są przez mechanizmy ROS 2 na temat publikacji pozycji `/hexapod/target_foot_positions` i przekazywane do klasy obliczania kinematyki odwrotnej.

5.5 Interfejs wizualizacji danych

Weryfikacja poprawności algorytmów obliczeń oraz zadanych trajektorii sterowania możliwa jest dzięki implementacji mechanizmu wizualizacji w środowisku RViz [Ope25b]. Narzędzie potrafi przyjmować punkty publikowane na tematach komunikacji między poszczególnymi węzłami, dzięki czemu może wizualizować zadane trajektorie, oraz mierzone pozycje końca nogi w czasie rzeczywistym. Rviz wizualizuje zadane dane za pomocą rysowanie punktów w przestrzeni roboczej, co pozwala na intuicyjne i szybkie bieżące weryfikacje na etapie budowania kodu.

5.6 Proces uruchomieniowy

Proces uruchomieniowy przebiega w sposób sekwencyjny. Kolejne elementy systemu wymagają do działania uruchomienia innych, które będą przyjmować ich polecenia. Pierwszym etapem jest konfiguracja środowiska symulacji. Konieczne jest załadowanie zmiennych środowiskowych, które zawierają ścieżki do plików konfiguracyjnych robota, jak na przykład wykorzystywane pakiety komunikacyjne. Jest to realizowane za pomocą skryptu `source install/setup.bash`.

5.6.1 Sekwencja startowa

Punktem wejścia symulacji jest skrypt startowy uruchamiający środowisko Gazebo. Wywołuje on instrukcje, które w efekcie inicjują świat symulacji opisany w pliku inicjalizacji `hexapod_world.world` w którym następnie umieszczają model robota za pomocą instrukcji `spawn_entity` wraz z opisem jego stanu `robot_state_publisher`. Następnie inicjalizowane są mechanizmy kontroli przegubów `joint_state_broadcaster` oraz sterowników pozycji `hexapod_position_controller` Uruchomiona warstwa symulacyjna gotowa jest na przyjęcie danych z węzłów logiki sterującej. Dane generowane są poprzez uruchomienie węzłów obliczeń kinematyki `hexapod_geometric_controller` lub alternatywnie `hexapod_jacobian_controller`. Te natomiast otrzymują dane z zadajników trajektorii bądź orientacji, na przykład `hexapod_orientation_matrix`.

Rozdział 6

Badania symulacyjne

Celem przeprowadzonych testów jest zweryfikowanie poprawności implementacji metod liczenia kinematyki odwrotnej oraz algorytmów sterowania orientacją robota w symulacji Gazebo. Na podstawie wyników badań przeprowadzono analizę zbieżności trajektorii zadanych i faktycznie obserwowanych w symulowanym robocie. Zmierzono również efektywności czasowe opisanych w pracy algorytmów i porównano je ze sobą.

Na początek przeprowadzono testy działania mechanizmów kinematyki odwrotnej, zadając obu algorytmom to samo zadanie trajektorii a następnie porównując otrzymane ślady oraz sygnatury czasowe. Testy orientacji przeprowadzono poprzez zadanie identycznych zmian w orientacji korpusu robota obliczanych w sposób geometryczny oraz za pomocą macierzy Eulera, żeby następnie porównać zbieżność wygenerowanych śladów oraz przeanalizować przesunięcie punktu wzdłuż osi pomiaru zależącej od wykonywanego testu. Na koniec przeprowadzono badanie zbieżności zadanej trajektorii chodu z śladem generowanym przez końce odnóży w uruchomionej symulacji.

Badania zostały przeprowadzone w środowisku symulacyjnym Gazebo, z wykorzystaniem architektury ROS 2. W celu eliminacji wpływów podłoża na analizę dokładności algorytmów, robot został umieszczony na przytwierdzonym do podłoża oraz spodu korpusu robota piedestale, mającym postać walca o średnicy $5\mathrm{cm}$ oraz wysokości $30\mathrm{cm}$.

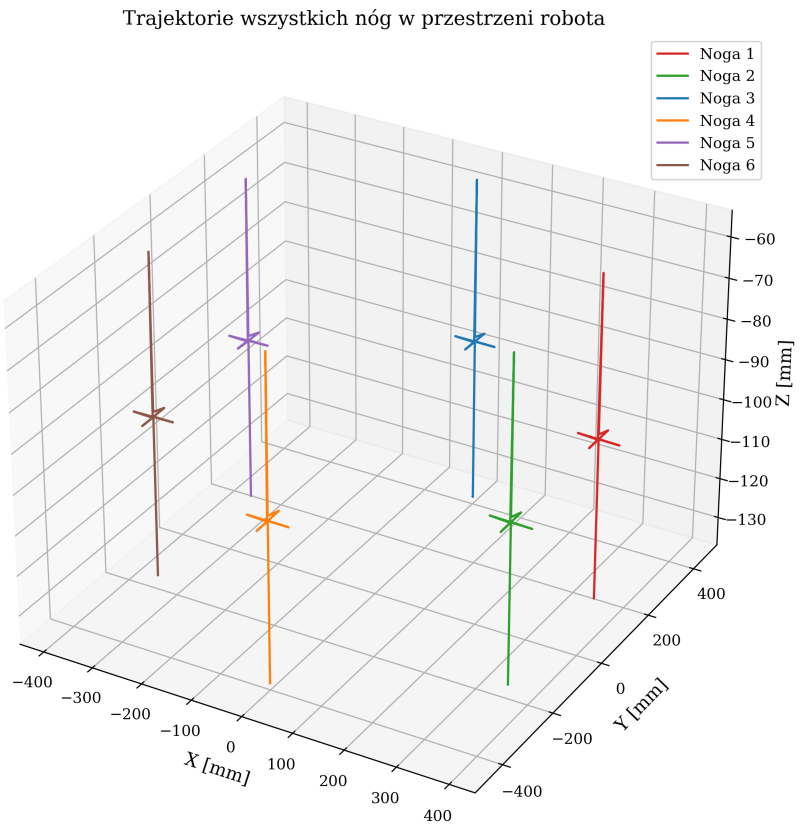
6.1 Test dokładności kinematyki odwrotnej

Scenariusz metodologii testowej zakładał sekwencję przesunięć liniowych końca nogi w zadanych osiach układu współrzędnych robota ($X_R Y_R Z_R$). Dla każdej nogi zostały zadane następujące transformacje:

- ruch wzdłuż osi X_N (przód-tył) na odległość$\pm$40 mm,
- powrót do pozycji wyjściowej,
- translację wzdłuż osi Y_N (lewo-prawo) o odległość $\pm$40 mm,
- stabilizację końca odnóża na zadanej wysokości Z_N.

Dane mierzono z częstotliwością $50\mathrm{Hz}$ biorąc pod uwagę dwie wielkości wektorowe:

1. pozycję zadaną (P_z) – trajektorię zadawana kinematyce odwrotnej,



Rysunek 6.1: Wizualizacja trajektorii wszystkich kończyn robota w przestrzeni 3D

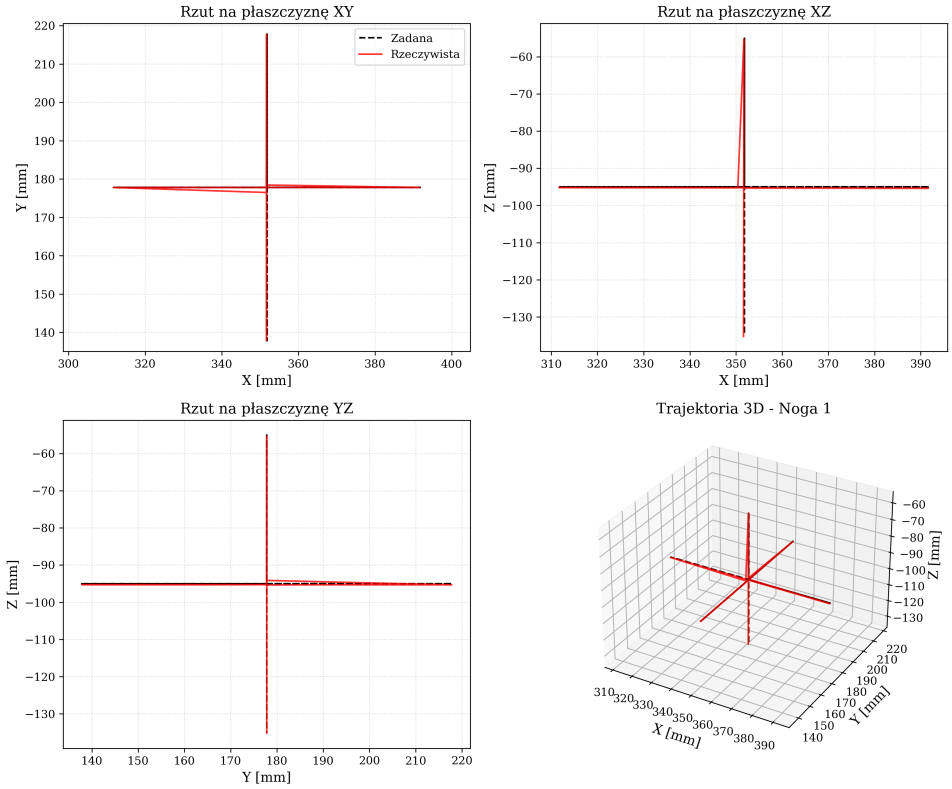
2. pozycję rzeczywistą (P_r) – będącą położeniem końcówki odnóża odczytywaną z tematu symulatora Gazebo.

Wstępna weryfikacja poprawności generacji pozycji, realizowana była wizualnie za pomocą reprezentacji wszystkich trajektorii zadanych oraz mierzonych w układzie robota ($X_R Y_R Z_R$). Wynik przedstawiono na rysunku 6.1. Przedstawiony na wykresie pomiar potwierdza prawidłową implementację geometrii robota oraz właściwe umiejscowienie osi obrotu bioder względem korpusu.

6.1.1 Wyniki dla metody geometrycznej

Szczegółową analizę pomiaru wykonanego podczas realizacji zadania opisanego w podrozdziale 6.1 przedstawiono na przykładzie nogi numer 1 (lewa przednia). Na rysunku 6.2 pokazano rzuty śladu ruchu nogi na główne płaszczyzny układu współrzędnych robota. Analizując wykresy można sformułować następujące konkluzje:

- W płaszczyźnie $X_R Y_R$ ruch odbywa się precyzyjnie wzdłuż zadanych osi. Nie zostały zaobserwowane sprzężenia co oznacza, że ruch w osi X_R odbywa się niezależnie od ruchu w osi Y_R. Kształt trajektorii zadanej jest wiernie odwzorowywany przez faktyczną pozycję końca nogi.
- W stanach ustalonych – po osiągnięciu zadanej pozycji – koniec nogi utrzymuje zadaną mu pozycję bez występowania widocznych oscylacji.



Rysunek 6.2: Ślady ruchu dla metody geometrycznej (Noga 1)

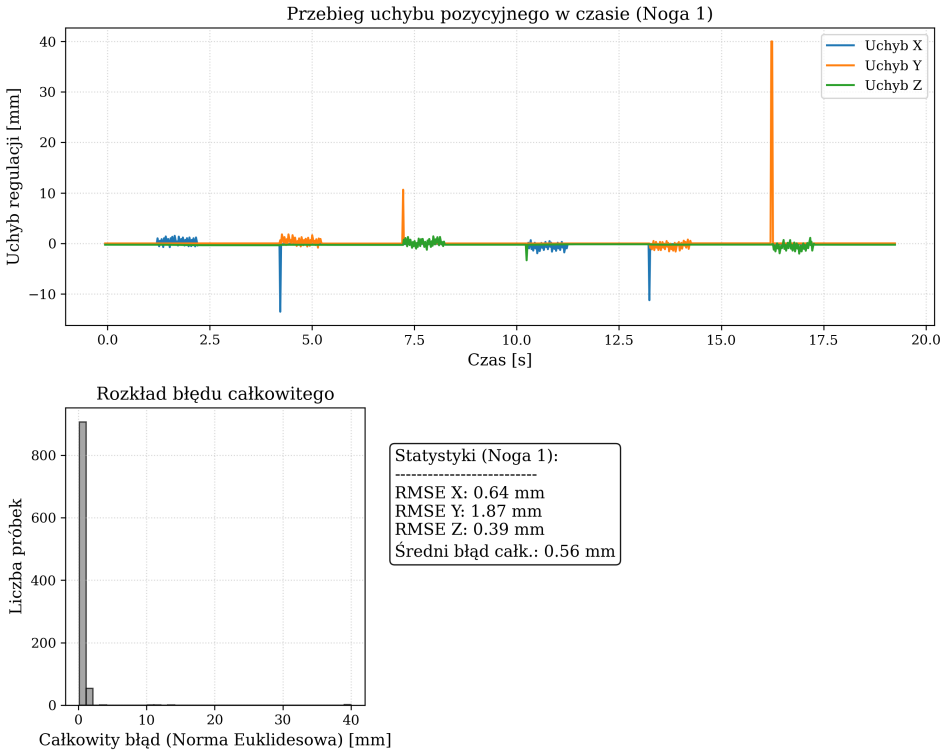
Ilościowa ocena jakości sterowania została zrealizowana poprzez analizę uchybu regulacji, który zdefiniowany jest jako różnica między pozycją zadaną a rzeczywistą. Wyniki tej analizy przedstawiono na rysunku 6.3. Wykresy obrazują przebieg uchybu w czasie oraz jego rozkład statystyczny. RMSE [BF11] – średni błąd kwadratowy dla osi X_R wynosi jedyne 0.64 mm, natomiast dla osi Y_R około 1.87 mm. Zakładając testową amplitudę ruchu o wartości 40 mm, błąd względny oscyluje w zakresie 1.5%–4%. Największe wartości uchybu obserwowane są w momentach dynamicznej zmiany prędkości takich jak początek i koniec ruchu.

6.1.2 Wyniki dla metody jakobianowej

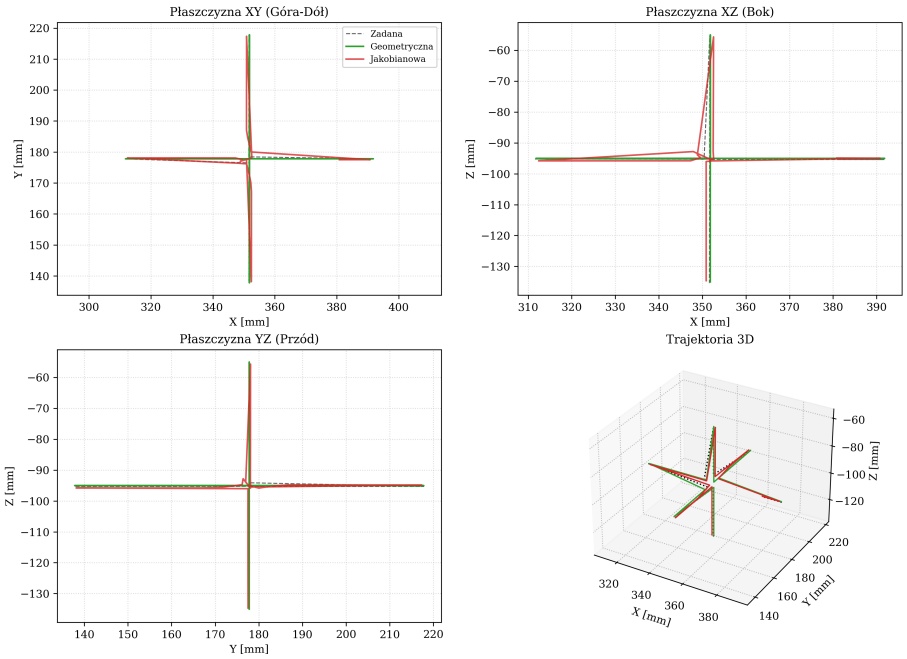
W sposób analogiczny do przypadku metody geometrycznej badania zostały przeprowadzone dla metody jakobianowej, będącej alternatywną metodą liczenia kinematyki odwrotnej. Ślady ruchu zaobserwowane w obu przypadkach zostały porównawczo przedstawione na rysunku 6.4. Kolorem zielonym oznaczono przebieg śladu końca nogi w metodzie jakobianowej, kolorem czerwonym ślad w metodzie geometrycznej, natomiast linią kreskowaną trajektorię zadaną obu tym algorytmom.

Analiza porównawcza wyników dla metody jakobianowej wykazała:

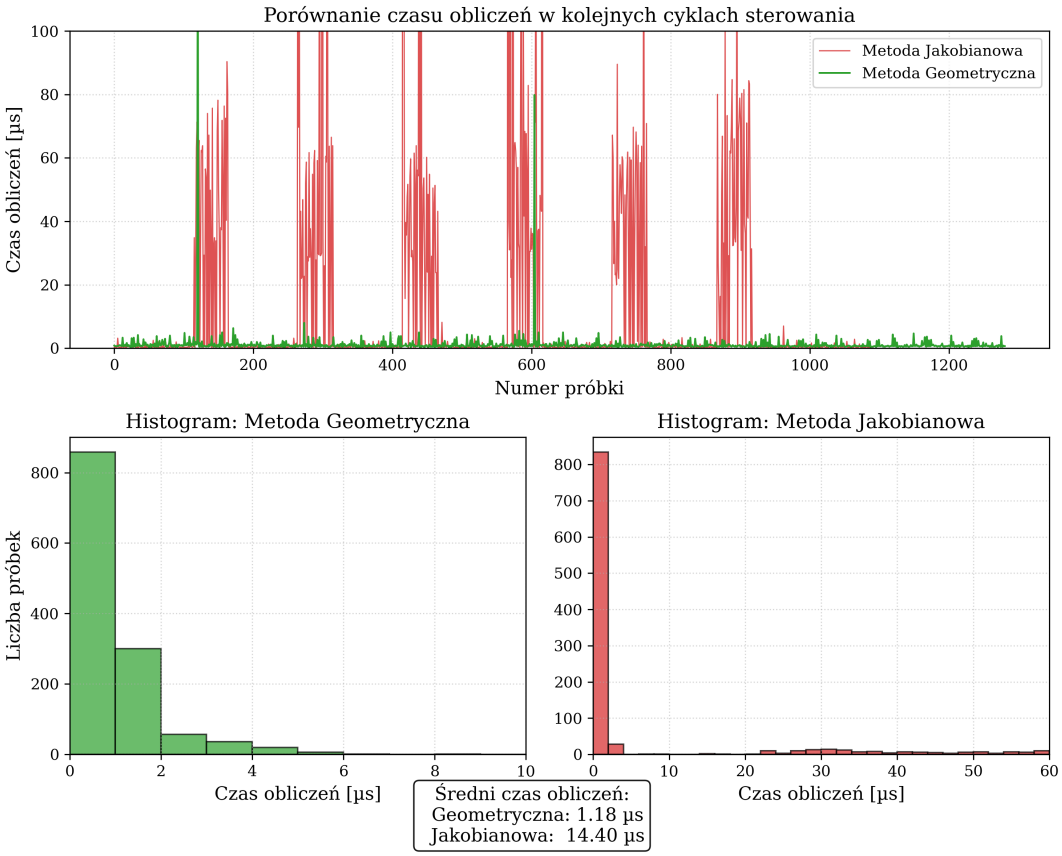
- Większy uchyb śledzenia niż w metodzie geometrycznej – średni błąd RMSE w osi X_R wzrósł do poziomu 1.30 mm (0.64 mm w metodzie geometrycznej).
- Dynamiczną charakterystykę algorytmu. Metoda jakobianu analitycznego wykazuje tendencje łagodzenia ostrych zmian kierunku ruchu. Jest to podyktowane podsta-



Rysunek 6.3: Błędy metody geometrycznej



Rysunek 6.4: Ślady ruchu w metodzie geometrycznej i jakobianowej przedstawione na jednym wykresie



Rysunek 6.5: Rozkład czasu obliczeń kinematyki odwrotnej

wową własnością algorytmu, który steruje prędkościami w układzie, co za tym idzie wprowadza opóźnienie w dojściu do zadanej mu pozycji.

- Mniejszą precyzję w realizowaniu zadania kinematyki odwrotnej, co w ogólnym rozrachunku nie wpływa na stabilność systemu.

6.2 Porównanie wydajności obliczeniowej

Drugim aspektem badań w metodach obliczania kinematyki odwrotnej był pomiar efektywności obliczeniowej, będącej istotnym kryterium w systemach czasu rzeczywistego. Na rysunku 6.5 przedstawiono porównawczą charakterystykę czasową dla obydwóch opisywanych algorytmów. Górny wykres przedstawia przebieg czasu obliczeń kinematyki odwrotnej dla kolejnych próbek symulacji. Dolne histogramy przedstawiają rozkłady statystyczne obrazowanych wyżej czasów.

Analiza zgromadzonych danych pozwala na sformułowanie następujących wniosków dotyczących charakterystyki obu metod:

- Metoda geometryczna wykazuje się deterministycznym czasem obliczeń oraz ich wysoką stabilnością.
- Średni czas potrzebny na wykonanie jednej instancji obliczenia kinematyki odwrotnej w omawianym ujęciu wynosi 1.18µs, przy czym rozrzut w wartościach jest po-

mijalnie mały. Jest to efektem analitycznego podejścia do przeprowadzanych w algorytmie obliczeń.

- Niezależnie od danych wejściowych, procesor zawsze wykonuje te same operacje matematyczne w takiej samej skończonej liczbie. Jest to gwarantem złożoności obliczeniowej $O(1)$ [CLRS09], niezwykle pożądanej w systemach czasu rzeczywistego.
- Metoda jakobianowa ma znacznie wyższy średni czas wykonania – ok. 14.4µs oraz znacznie większą wariancję w wynikach. Z uwagi na iteracyjną naturę algorytmu jej czas wykonania jest proporcjonalny do liczby koniecznych wywołań algorytmu przed osiągnięciem celu z żądaną tolerancją.
- Szczególną uwagę zwracają okresowe skoki czasu obliczeń metody jakobianowej. Podyktowane są one gwałtownymi zmianami prędkości, których efektem jest większa konieczna ilość iteracji algorytmu co za tym idzie dłuższy czas obliczeń.

Z analizy danych zebranych z działania obu algorytmów można wywnioskować, że algorytm geometryczny jest znacznie bardziej efektywny obliczeniowo oraz oferuje większą precyzję obliczeń. W kontraście do niego algorytm jakobianowy jest znacznie prostszy w rozwoju oraz nie ulega tak ogromnym komplikacjom w systemach o większej ilości przegubów.

6.3 Test poprawności transformacji orientacji

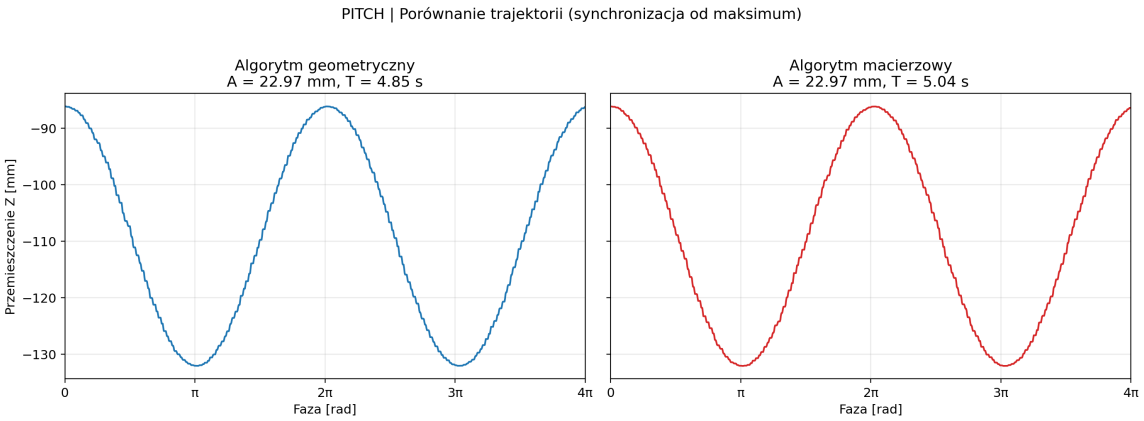
Celem testów transformacji orientacji było zweryfikowanie poprawności implementacji algorytmów sterowania przechyłem korpusu robota. Badania zostały przeprowadzone poprzez zadanie obu algorytmom cyklicznej zmiany kąta orientacji θ o wartości $0,3[\mathrm{rad}]$ (około 17.2°) w pojedynczej osi obrotu. Następnie został przeprowadzony test złożenia zmiany orientacji korpusu w kilku osiach jednocześnie. Wszystkie testy korzystały z geometrycznego podejścia do liczenia kinematyki odwrotnej.

Sygnałem wejściowym testu jest realizowane dzięki oscylacji sinusoidalnej pomiędzy skrajnymi wartościami zadanego kąta. Pomiary wykonane zostały niezależnie od siebie dla każdego z podejść oraz realizowane były zewnętrznym skryptem. Skutkowało to różnymi momentami rozpoczęcia względem fazy oscylacji, w której znajdował się robot. Z uwagi na cykliczny charakter testu, przesunięcie fazowe ma znikomy wpływ na ocenę poprawności zastosowanych algorytmów. Do właściwej analizy wykorzystane zostały pomiary amplitudy, śladu pozycjonowania końca odnóża oraz okresu mierzonego sygnału.

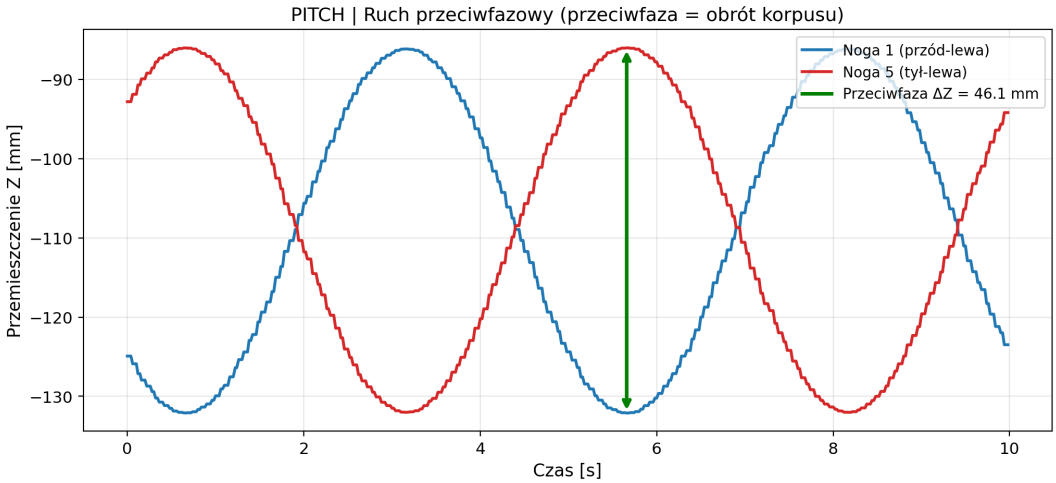
6.3.1 Test transformacji Pitch

Test przechyłu pitch polegał na zadaniu okresowej zmiany kąta obrotu korpusu względem osi Y_R. Oczekiwany wynik to przemieszczenie się końców odnóży w płaszczyźnie OX_RZ_R, gdzie nogi tylne (5,6) i przednie (1,2) poruszają się w kierunku pionowym, jednakże w przeciwnych fazach. Nogi środkowe (3,4) powinny pozostawać na stałej wysokości.

Rysunek 6.6 przedstawia porównanie przebiegów zmierzonego przemieszczenia końca nogi 1 w osi Z dla obu algorytmów – geometrycznego oraz macierzowego. Wykresy zostały znormalizowane względem fazy ruchu (oś pozioma w radianach), co pozwala na bezpośrednie porównanie kształtu trajektorii niezależnie od momentu rozpoczęcia pomiaru.



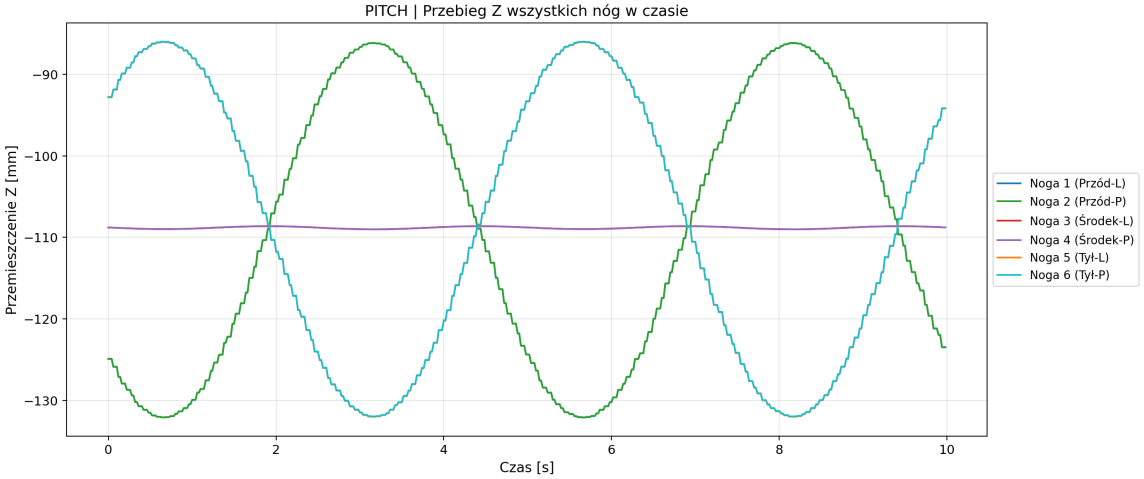
Rysunek 6.6: Porównanie trajektorii dla testu pitch – algorytm geometryczny (lewy panel) i macierzowy (prawy panel). Znormalizowano względem fazy ruchu



Rysunek 6.7: Ruch przeciwfazowy nóg przedniej i tylnej podczas testu pitch. Strzałka wskazuje maksymalną różnicę położeń $\Delta Z = 46{,}1$ mm

Z analizy wykresów można wywnioskować całkowitą zgodność zmierzonego śladu ruchu dla obu algorytmów. W obydwóch przypadkach zmierzona amplituda pomiaru przemieszczenia wynosi $A = 22{,}97$ mm, natomiast okres oscylacji wynosi $T = 4{,}99$ s co jest zgodne z danymi podanymi w inicjalizacji testu. Obliczony współczynnik korelacji po przeprowadzenia kompensacji fazy przekracza wartość $0,999$ co jednoznacznie wskazuje na zgodność obydwóch algorytmów. Cechą ruchu Pitch, którą możemy zaobserwować na rysunku 6.7 jest przeciwfazowy charakter ruchu dla nóg przednich (tutaj 1) względem tylnych (tutaj 5). Rysunek przedstawia przebieg przemieszczenia wybranych nóg w osi Z w czasie.

Zaznaczona na rysunku przeciwfaza jest bezpośrednią konsekwencją obrotu korpusu względem osi Y_R. Gdy przód robota jest obniżany, jednocześnie podnoszony jest jego tył o wartość mającą skompensować ruch po przeciwległej stronie. W rezultacie maksymalna zmierzona różnica w położeniu $\Delta Z = 46{,}1$ mm równa jest co do wartości podwojonej amplitudzie ruchu w pojedynczej nodze. Jest to oczekiwany skutek w symetrycznym rozmieszczeniu odnóży robota.



Rysunek 6.8: Przemieszczenie w osi Z wszystkich nóg podczas testu pitch

Rysunek 6.8 przedstawia przemieszczenie wszystkich nóg w osi Z w czasie. Wykres ukazuje oczekiwane dla zadanej mu rotacji wyniki:

- Nogi przednie (1, 2) oraz tylne (5, 6) wykonują ruch o pełnej amplitudzie działając w przeciwfazie w przeciwległych sobie parach – przód-tył.
- Nogi środkowe (3, 4) pozostają praktycznie nieruchome – początki lokalnych układów obu tych nóg znajdują się na osi obrotu Y w zadanym zadaniu, co skutkuje brakiem zmian w wysokości położenia.
- Nogi po tej samej stronie korpusu (lewa-prawa) mają zgodną fazę ruchu, co potwierdza, że obrót odbywa się wyłącznie wokół osi Y.

Powyższe obserwacje potwierdzają poprawność implementacji transformacji pitch w obu algorytmach.

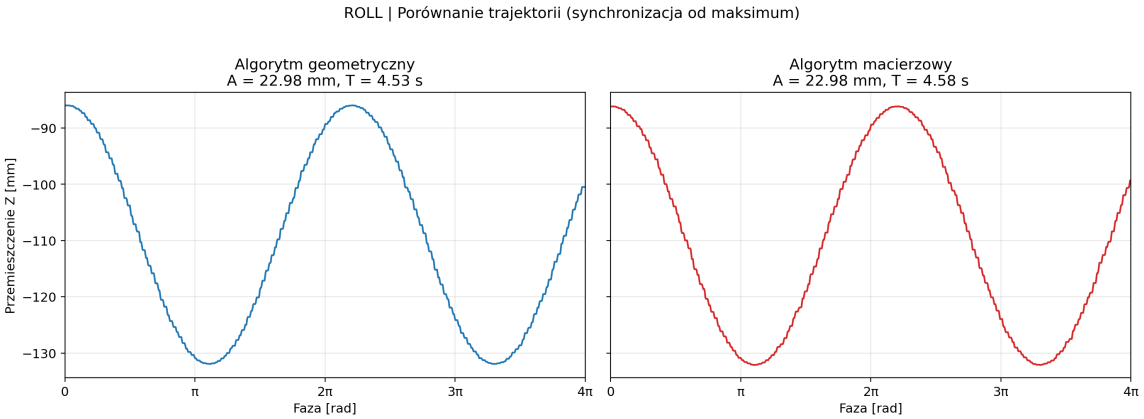
6.3.2 Test transformacji Roll

Test przechyłu roll polegał na zadaniu okresowej zmiany kąta obrotu korpusu względem osi X_R. Oczekiwany wynik to przemieszczenie się końców odnóży w płaszczyźnie OY_RZ_R, gdzie nogi lewe (1,3,5) i prawe (2,4,6) poruszają się w przeciwnych fazach. Na rysunku 6.9 przedstawiono porównanie zmierzonych śladów końca odnóża dla obu algorytmów, znormalizowane względem fazy ruchu.

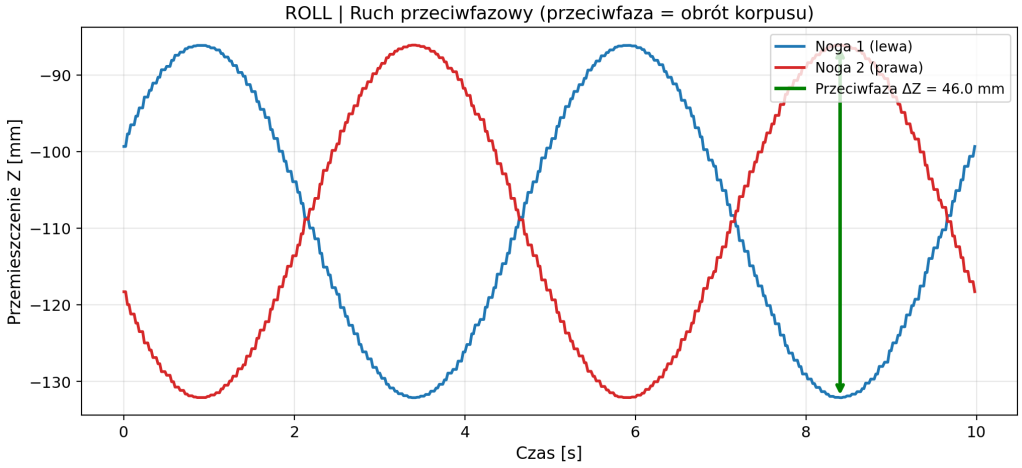
Analogicznie jak w przypadku testu pitch, oba algorytmy generują praktycznie identyczne trajektorie. Zmierzone amplitudy wynoszą $A_{geo} = 23{,}06$ mm oraz $A_{mat} = 22{,}98$ mm (różnica 0,3%), a okresy oscylacji są równe $T \approx 5{,}0$ s.

Rysunek 6.10 przedstawia różnicę w fazach w ruchu nóg lewych względem prawych. Przeciwfaza między nogami lewą (noga 1) a prawą (noga 2) jest charakterystyczna dla obrotu wokół osi X_R: przechylenie korpusu w lewo powoduje opuszczenie lewej strony i jednoczesne uniesienie prawej. Zmierzona różnica położeń $\Delta Z = 46{,}0$ mm jest zgodna z wynikami testu pitch, co potwierdza symetrię konstrukcji robota.

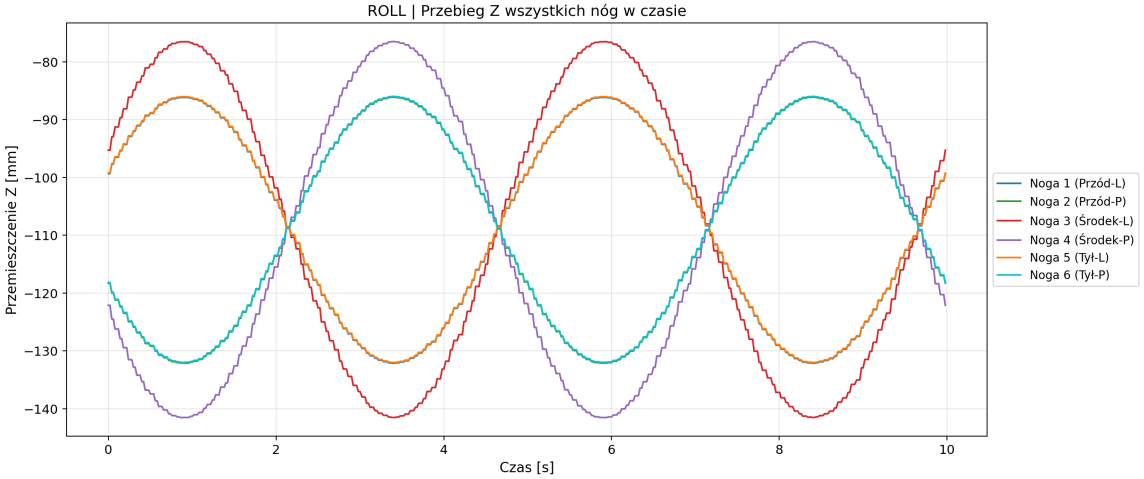
Rysunek 6.11 przedstawia przebiegi wszystkich nóg podczas testu roll. Analiza wykresu pozwala na sformułowanie następujących obserwacji:



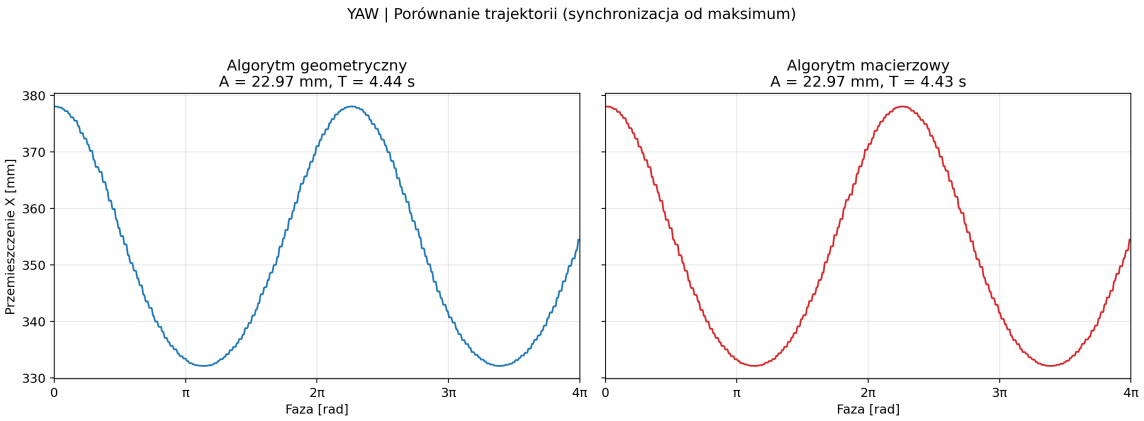
Rysunek 6.9: Porównanie trajektorii dla testu roll – algorytm geometryczny (lewy panel) i macierzowy (prawy panel)



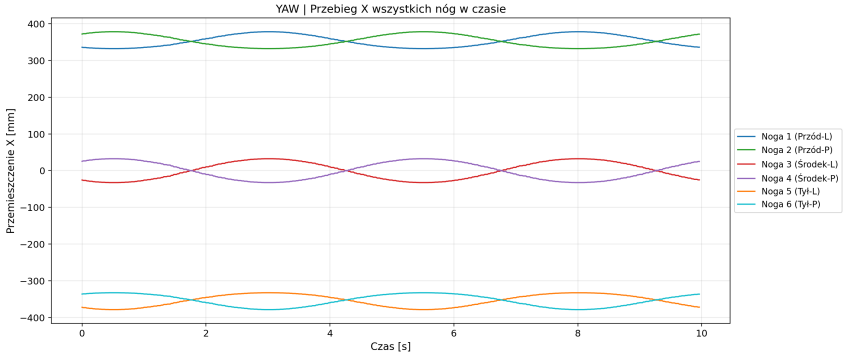
Rysunek 6.10: Ruch przeciwfazowy nóg lewej i prawej podczas testu roll. Strzałka wskazuje maksymalną różnicę położeń $\Delta Z = 46{,}0$ mm



Rysunek 6.11: Przemieszczenie w osi Z wszystkich nóg podczas testu roll



Rysunek 6.12: Porównanie trajektorii w osi X dla testu yaw – algorytm geometryczny i macierzowy

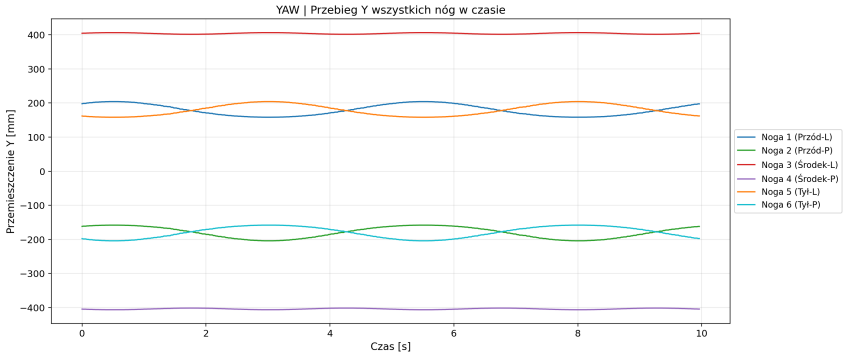


Rysunek 6.13: Przemieszczenie w osi X wszystkich nóg podczas testu yaw

- Każda z nóg wykonuje ruch – inaczej niż w obrocie pitch w roll ruch odbywa się we wszystkich nogach, ponieważ żaden z początków układów współrzędnych nogi nie leży na osi obrotu.
- Największe przemieszczenia można zaobserwować w nogach 3 i 4, mających najbardziej oddalone początki układów współrzędnych od osi obrotu.
- Nogi będące po tej samej stronie korpusu (1,3,5) i prawe (2,4,6) poruszają się w tej samej fazie.
- Nogi będące po przeciwnych stronach są w dokładnej przeciwfazie.

6.3.3 Test transformacji Yaw

Test przechyłu yaw polegał na zadaniu okresowej zmiany kąta obrotu korpusu względem osi Z_R. W odróżnieniu od pitch i roll, obrót końców odnóży obserwowany jest w płaszczyźnie $X_R Y_R$, nie zmieniając ich wysokości. Na rysunku 6.12 przedstawiono porównanie przebiegów przemieszczenia zmierzonego w osi X dla obydwóch algorytmów. Ponownie po przeprowadzeniu normalizacji fazy można zaobserwować pełną zgodność w amplitudzie $A = 22{,}97$ mm oraz okresie $T \approx 5{,}0$ s, które są identyczne dla obu metod. Na rysunku 6.13 przedstawiono przebiegi przemieszczenia w osi X dla wszystkich nóg. Na ry-



Rysunek 6.14: Przemieszczenie w osi Y wszystkich nóg podczas testu yaw

sunku 6.14 przedstawiono przebiegi przemieszczenia w osi Y dla wszystkich nóg. Z powyżej wspomnianych wykresów możemy wysnuć następujące wnioski:

- Nogi znajdujące się po przeciwnych stronach korpusu – lewe (1,3,5) i prawe (2,4,6) poruszają się w przeciwfazie w ruchu względem osi X.
- Nogi znajdujące się po przeciwnych stronach korpusu przód(1, 2) i tył (5, 6) poruszają się w przeciwfazie w ruchu względem osi Y.
- Nogi środkowe (3, 4) wykazują najmniejszą amplitudę w przemieszczeniu względem osi X, ponieważ początki ich lokalnych układów współrzędnych leżą blisko osi X_R korpusu.

Obserwacje te potwierdzają poprawność implementacji transformacji yaw oraz prawidłowe zdefiniowanie lokalnych układów współrzędnych poszczególnych nóg.

6.3.4 Test złożenia rotacji we wszystkich osiach

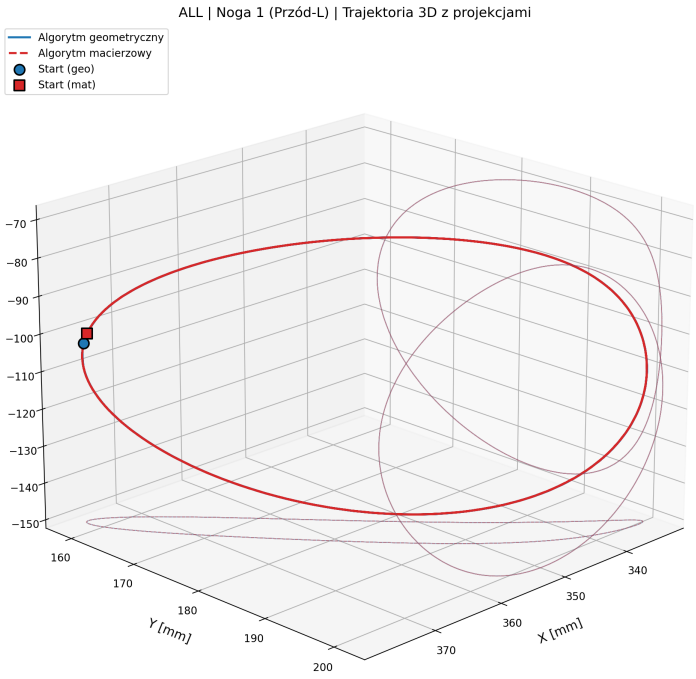
Weryfikacja poprawności implementacji złożenia transformacji orientacji została przeprowadzona przez zadanie trajektorii o sinusoidalnym charakterze ze zgodną fazą we wszystkich trzech osiach obrotu. Pozwoliło to na sprawdzenie zgodności kolejności przeprowadzania kolejnych rotacji w obydwóch algorytmach.

Rysunek 6.15 przedstawia trójwymiarowy pomiar śladu końca nogi 1 zarejestrowany podczas testu złożonego. Obydwa wygenerowane ślady trajektorii pokrywają się niemal idealnie co prowadzi do następujących konkluzji:

- Obydwa algorytmy wykonują operacje obrotu w zgodnej kolejności.
- Transformacje są wykonywane względem tych samych osi i punktów odniesienia.
- Matematyczna równoważność obydwóch implementacji jest zgodna dla złożonych zmian orientacji korpusu.

6.3.5 Podsumowanie testów orientacji

Przeprowadzone testy transformacji orientacji pozwalają na sformułowanie następujących wniosków:



Rysunek 6.15: Ślad końca nogi podczas jednoczesnej zmiany orientacji we wszystkich osiach. Na ścianach pokazano projekcje śladu na poszczególne płaszczyzny

1. Implementacje algorytmu geometrycznego oraz macierzowego generują niemalże identyczne ślady ruchu. Różnice występujące w zmierzonych amplitudach nie przekraczają 0,3%, natomiast współczynniki korelacji przebiegów wynoszą ponad 0,999.
2. Zaobserwowane prawidłowości w występowaniu przeciwfazy trajektorii są zgodne z teoretycznymi oczekiwaniami i wskazują na poprawność implementacji.
3. Analiza testu złożonej zmiany orientacji wykazała zgodność kolejności przeprowadzania rotacji pomiędzy obydwoma algorytmami.
4. Zmierzone amplitudy przemieszczenia (około 23 mm) są zgodne z wartościami teoretycznymi dla zadanego kąta $\theta = 0{,}3$ rad i znanych odległości bioder od osi obrotu.

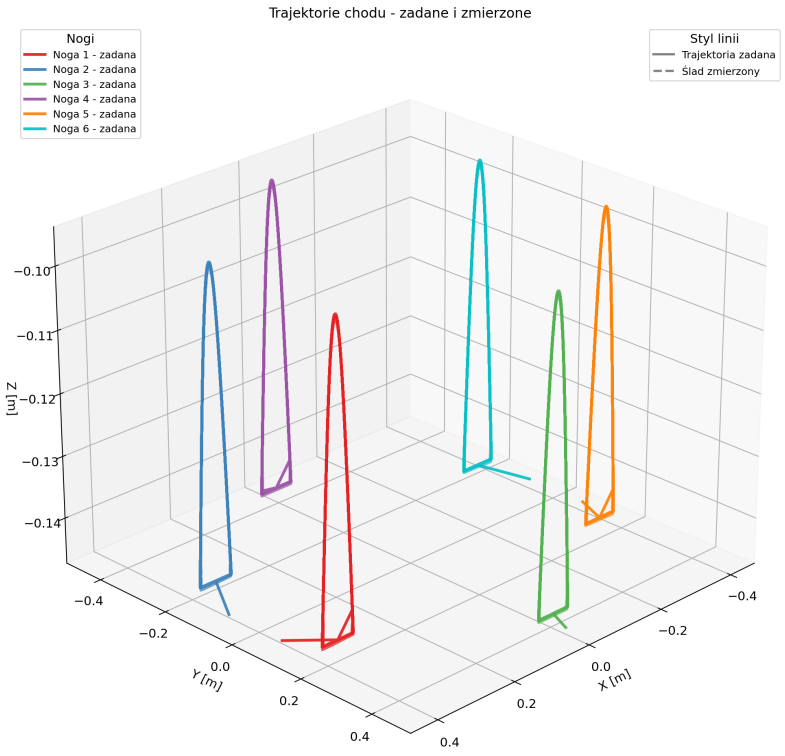
6.4 Test trajektorii chodu

Ostatnim elementem przeprowadzonych badań był test generatora trajektorii chodu. Calem testu było sprawdzenie zgodności implementacji modelu pod względem poprawności generacji trajektorii chodu trójpodporowego (tripod gait) oraz jej pokrycia ze zmierzonym śladem końca nogi.

6.4.1 Konfiguracja testu

Test przeprowadzono z wykorzystaniem następujących parametrów chodu:

- okres cyklu chodu: $T = 1{,}5$ s,



Rysunek 6.16: Trajektorie zadane dla wszystkich nóg podczas testu chodu w układzie współrzędnych robota

- długość kroku: $L = 80$ mm,
- wysokość unoszenia nogi: $H = 50$ mm,
- kierunek ruchu: do przodu (dodatni kierunek osi X_R).

Zgodnie z opisem algorytmu przedstawionym w podrozdziale 4.3, nogi robota podzielono na dwie grupy:

- grupa 1: nogi $\{1, 4, 5\}$ (lewa przednia, prawa środkowa, lewa tylna),
- grupa 2: nogi $\{2, 3, 6\}$ (prawa przednia, lewa środkowa, prawa tylna).

W każdym momencie cyklu, jedna z grup znajduje się w fazie przenoszenia (protrakcji), podczas gdy druga realizuję fazę podpardzia (retrakcji). Każda z faz trwa połowę pełnego okresu. Obie grupy działają ze sobą w przeciwfazie.

6.4.2 Analiza trajektorii

Rysunek 6.16 przedstawia trójwymiarową wizualizację zadanych trajektorii oraz zmierzonych śladów pozycji końca odnóża dla wszystkich sześciu nóg robota w układzie współrzędnych korpusu. Analiza wykresu pozwala na sformułowanie następujących obserwacji:

- Zmierzony ruch pokrywa się z zadaną trajektorią i ma właściwy kształt zamkniętej pętli w płaszczyźnie OXZ.

- Wysokość unoszenia nogi w fazie protrakcji odpowiada zadanej wartości $\mathsf{H} = 50$ mm, a profil wznoszenia ma kształt sinusoidy zgodnie z równaniem (4.25).
- Zmierzona długość kroku w kierunku X wynosi $\mathsf{L} = 80$ mm, co jest zgodne z zadaną wartością parametru.

6.4.3 Podsumowanie testu chodu

Przeprowadzony test potwierdził poprawność implementacji generatora trajektorii chodu. Zaimplementowany algorytm:

1. prawidłowo generuje trajektorie o zadanych parametrach (długość kroku, wysokość unoszenia, okres),
2. właściwie transformuje trajektorie do lokalnych układów współrzędnych poszczególnych nóg,
3. zachowuje poprawną synchronizację trójnożnych grup, zapewniając stabilność podparcia w trzech punktach,
4. realizuje płynny, cykliczny ruch umożliwiający przemieszczanie się robota do przodu.

Rozdział 7

Podsumowanie

Głównym celem pracy było zaprojektowanie symulacji implementującej systemy sterowania robotem kroczącym z naciskiem na algorytmy zmiany orientacji. Cel został zrealizowany poprzez zaprojektowanie kompletnego modelu trójwymiarowego robota i umieszczenie go w środowisku Gazebo za pomocą mechanizmów ROS 2.

7.1 Realizacja prac projektowych i implementacyjnych

W ramach pracy zrealizowano wszystkie założone etapy procesu inżynierskiego:

- Modelowanie – dzięki użyciu środowisk CAD, został zaprojektowany model robota, przekształcony do opisu URDF. Konfiguracja parametrów fizycznych ukazała jednak pewne ograniczenia w modelu. Mimo poprawnego ustalenia masy oraz dostosowanie inercji komponentów robota, parametry tarcia oraz reakcji podłoża wymagają dalszej konfiguracji. Taki stan symulacji pozwala na wierne odwzorowania ruchów robota, jednak jego interakcja ze środowiskiem pozostawia pewne niezgodności.
- Implementacja matematyki sterowania – metody matematyczne zostały zaimplementowane poprawnie, zarówno analityczne jak i numeryczne, czego skutkiem jest właściwe ustawianie pozycji robota.
- Architektura systemu – Zastosowanie architektury bazującej na środowisku ROS 2 pozwoliło na modularyzację systemu i wydzielenie poszczególnych funkcjonalności do niezależnych systemów. Takie podejście pozwala na łatwą wymianę, bądź poprawę komponentów bez wpływu na pozostałe mechanizmy symulacji.

7.2 Wnioski z analizy modeli matematycznych

Na etapie implementacji i weryfikacji narzędzi matematycznych sformułowano następujące wnioski dotyczące zastosowanych metod.

7.2.1 Kinematyka odwrotna

Podejście geometryczne pozwala na intuicyjne zrozumienie przekształcania zadanych pozycji robota na ustawienia kątów w jego przegubach. Jest to rozwiązanie stabilne, dające

stabilny i deterministyczny wynik. Jawna postać wzorów powoduje stały, przewidywalny i krótki czas obliczeń, będący dużym atutem w systemach działających w czasie rzeczywistym.

Metoda jakobianowa – bardziej ogólna i dająca się łatwiej adaptować w przypadku zmian struktury kinematycznej nogi, wymaga doboru odpowiednich nastaw dla uzyskania pożądanej dokładności danych zwrotnych w rozsądnym czasie. Niewłaściwy dobór wspomnianych parametrów może skutkować wynikami obarczonymi błędem lub długim czasem obliczeń.

7.2.2 Sterowanie orientacją

Implementacja metody geometrycznej sterowania orientacją korpusu intuicyjnie przedstawia rozłożenie skomplikowanych obliczeń na obroty w poszczególnych płaszczyznach. To co jest jednocześnie jej dużą siłą w kontekście zrozumienia mechanizmu prowadzi do większych trudności implementacyjnych w szczególności przy jednoczesnym złożeniu obrotów w kilku osiach. Metoda macierzy jednorodnych jako standard w robotyce jest łatwiejsza do implementacji, jednak wymaga posiadania wiedzy z zakresu szerszego aparatu matematycznego. Zastosowanie macierzy rotacji rozszerzonej o wektor translacji pozwala na złożenie ruchu o sześciu stopniach swobody w jedną operację obliczeniową. Jest to rozwiązanie bardziej eleganckie i upraszcza strukturę kodu.

7.3 Wnioski z badań symulacyjnych

Przeprowadzone testy pozwoliły na analizę ilościową zaimplementowanych algorytmów pod kątem zgodności trajektorii z wykonywanym ruchem oraz ich efektywności obliczeniowej. Wyniki badań prowadzą do sformułowania następujących wniosków.

7.3.1 Dokładność kinematyki odwrotnej

Analiza uchybu regulacji wykazała istotne różnice między obiema metodami obliczania kinematyki odwrotnej:

- Metoda geometryczna zapewnia wysoką precyzję pozycjonowania. Zmierzony średni błąd kwadratowy (RMSE) wyniósł $0,64$ mm dla osi X_R oraz $1,87$ mm dla osi Y_R, co przy amplitudzie ruchu testowego 40 mm odpowiada błędowi względnemu w zakresie 1,5%–4,5%.

- Metoda jakobianowa wykazała większy uchyb śledzenia trajektorii z RMSE na poziomie 1,30 mm dla osi X_R (wzrost o około 100% względem metody geometrycznej). Charakterystyczne dla tej metody jest łagodzenie ostrych zmian kierunku ruchu, wynikające z iteracyjnego sterowania prędkościami w przestrzeni przegubów.

- Największe wartości uchybu w obu metodach obserwowano w momentach dynamicznej zmiany prędkości (początek i koniec ruchu), co jest typowe dla systemów sterowania pozycją.

7.3.2 Wydajność obliczeniowa

Badania czasowe jednoznacznie wskazują na przewagę metody geometrycznej:

- Metoda geometryczna charakteryzuje się deterministycznym czasem wykonania wynoszącym średnio 1,18 µs przy pomijalnie małej wariancji. Wynika to ze stałej złożoności obliczeniowej algorytmu $O(1)$ – niezależnie od danych wejściowych procesor wykonuje identyczną, skończoną liczbę operacji matematycznych.
- Metoda jakobianowa jest algorytmem iteracyjnym, którego czas wykonania zależy od liczby iteracji potrzebnych do osiągnięcia zadanej tolerancji. Zmierzony średni czas wyniósł 14,40 µs, co stanowi wartość ponad 12-krotnie większą niż w przypadku metody geometrycznej.
- Zaobserwowano okresowe skoki czasu obliczeń metody jakobianowej (do 100 µs), korelujące z gwałtownymi zmianami w trajektorii zadanej. Jest to bezpośrednia konsekwencja zwiększonej liczby iteracji potrzebnych do zbieżności w takich punktach.

7.3.3 Sterowanie orientacją

Testy porównawcze metod sterowania orientacją korpusu (geometrycznej i macierzowej) wykazały ich pełną równoważność funkcjonalną:

- Zmierzone ślady ruchu końców odnóży dla obu algorytmów pokrywają się z wysoką dokładnością. Po przeprowadzeniu korekcji przesunięcia fazowego, współczynnik korelacji Pearsona przekracza wartość 0,999 dla wszystkich testowanych transformacji (Roll, Pitch, Yaw).
- Zmierzone amplitudy przemieszczenia końców nóg wynoszą około 23 mm, co jest zgodne z wartościami teoretycznymi obliczonymi dla zadanego kąta $\theta = 0{,}3$ rad ($\approx$17,2°) i znanych odległości bioder od osi obrotu.
- Obserwowane relacje przeciwfazowe między nogami (np. przednie i tylne w teście Pitch, lewe i prawe w teście Roll) są zgodne z oczekiwaniami teoretycznymi i potwierdzają poprawność zdefiniowania układów współrzędnych.
- Test złożonej transformacji (jednoczesna zmiana Roll, Pitch i Yaw) potwierdził zgodność kolejności przeprowadzania rotacji w obu algorytmach.

7.3.4 Generator trajektorii chodu

Weryfikacja generatora chodu trójpodporowego potwierdziła poprawność implementacji:

- Zmierzone parametry trajektorii (długość kroku $L = 80$ mm, wysokość unoszenia $H = 50$ mm) odpowiadają wartościom zadanym.
- Profil wznoszenia nogi ma kształt sinusoidy zgodny z równaniem matematycznym modelu.

- Synchronizacja trójnożnych grup zapewnia stabilność podparcia w trzech punktach przez cały cykl chodu.

Z przeprowadzonych badań wynika, że metoda geometryczna jest preferowanym rozwiązaniem dla systemów działających w czasie rzeczywistym ze względu na:

1. wyższą precyzję (RMSE mniejszy o około 50%),
2. deterministyczny i krótszy czas obliczeń (ponad 12-krotnie),
3. stałą złożoność obliczeniową $\mathsf{O}(1)$.

Metoda jakobianowa pozostaje wartościową alternatywą w przypadkach wymagających elastyczności struktury kinematycznej lub gdy priorytetem jest płynność ruchu kosztem precyzji pozycjonowania. Obie metody sterowania orientacją (geometryczna i macierzowa) są funkcjonalnie równoważne i mogą być stosowane zamiennie.

7.4 Możliwości rozwoju projektu

System symulacji wymaga dostosowania parametrów z zakresu tarcia oraz interakcji robota z podłożem. Wprowadzenie tych zmian otwiera drogę do rozwoju projektu w aspekcie badania różnych algorytmów sterowania pod kątem zmian w środowisku robota, obserwacji jak robot radzi sobie z przeszkodami, czy nawierzchniami o różnych współczynnikach tarcia.

Opracowany symulator stanowi bazę do implementacji sterowania w rzeczywistym robocie kroczącym. Jest kompletnym zestawem narzędzi potrzebnych do sterowania robotem typu hexapod. Posiada możliwości dostosowania parametrów do robotów o różnych charakterystykach fizycznych o ile opiera się na podobnej strukturze odnóży. Zastosowanie sterowania momentem siły w silnikach fizycznego modelu mogłoby pozwolić na rozbudowę o algorytmy adaptacji ustawienia robota do terenu po którym się porusza

Ze względu na swoją modularność, system pozwala na łatwe dodanie innych algorytmów chodu takich jak chód dwupodporowy czy bardzo stabilny algorytm wave gait zakładający ruch tylko jednej nogi na raz.

Literatura

[Aut24] Inc. Autodesk. IPT, IAM – formaty plików autodesk. [Dokumentacja online], 2024. https://help.autodesk.com/view/3DSMAX/2024/ENU/?guid=GUID-3DA51A42-AB12-48D6-B6D5-A82B481CD686.

[Aut25a] Inc. Autodesk. Autodesk Fusion 360. [Oprogramowanie komputerowe], 2025. https://www.autodesk.com/products/fusion-360/.

[Aut25b] Inc. Autodesk. Autodesk Inventor. [Oprogramowanie komputerowe], 2025. https://www.autodesk.com/products/inventor/.

[Aut25c] Inc. Autodesk. Bryła – definicja i opis. [Dokumentacja online], 2025. https://help.autodesk.com/view/fusion360/PLK/?guid=GUID-C1AB4941-D7AD-4D27-A035-2FA9208635B6.

[BF11] Richard L. Burden, J. Douglas Faires, *Numerical Analysis*. Cengage Learning, Boston, wydanie 9, 2011.

[Bus09] Samuel R. Buss. Introduction to Inverse Kinematics with Jacobian Transpose, Pseudoinverse and Damped Least Squares Methods. University of California, San Diego, 2009. https://www.cs.cmu.edu/~15464-s13/lectures/lecture6/iksurvey.pdf.

[CLRS09] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, *Introduction to Algorithms*. MIT press, Cambridge, MA, wydanie 3, 2009.

[Cra05] John J. Craig, *Introduction to Robotics: Mechanics and Control*. Pearson Prentice Hall, wydanie 3, 2005.

[DH55] Jacques Denavit, Richard S. Hartenberg, A kinematic notation for lower-pair mechanisms based on matrices. *Journal of Applied Mechanics*, 22:215–221, 1955.

[Dhe22] K. Dheenadhayalan. fusion2urdf-ros2: Fusion360 addin to export URDF file for ROS2. [Repozytorium kodu, online], 2022. https://github.com/dheena2k2/fusion2urdf-ros2.

[Gie12] Rafał Gierczak, *Laboratoryjny robot kroczący*. Praca magisterska, Politechnika Wrocławska, Wrocław, 2012. https://kcir.pwr.edu.pl/~mucha/Pracki/Rafal_Gierczak_praca_magisterska.pdf.

[Hac22] John Hackney. Creating a dumb solid of an assembly in fusion 360. [Nagranie wideo, online], 2022. https://www.youtube.com/watch?v=H_4UJVlztYo.

[KH04] Nathan Koenig, Andrew Howard, Design and Use Paradigms for Gazebo, An Open-Source Multi-Robot Simulator. W: *Proceedings 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, wolumen 3, strony 2149–2154, 2004.

[MFG+22] Steven Macenski, Tully Foote, Brian Gerkey, Chris Lalancette, William Woodall, Robot Operating System 2: Design, architecture, and uses in the wild. *Science Robotics*, 7(66):eabm6074, 2022.

[Ope25a] Open Robotics. Robot operating system (ROS). [Strona internetowa, online], 2025. https://www.ros.org/.

[Ope25b] Open Robotics. RViz2: ROS 2 3D Robot Visualizer. [Dokumentacja online], 2025. https://docs.ros.org/en/humble/Tutorials/Intermediate/RViz/RViz-User-Guide/RViz-User-Guide.html.

[Rob25] RobotsGuide. Genghis. [Strona internetowa, online], 2025. https://robotsguide.com/robots/genghis.

[SHV20] Mark W. Spong, Seth Hutchinson, M. Vidyasagar, *Robot Modeling and Control*. Wiley, wydanie 2, 2020.

[Ste65] D. Stewart, A platform with six degrees of freedom. *Proceedings of the Institution of Mechanical Engineers*, 180(1):371–386, 1965.

[Str13] Bjarne Stroustrup, *The C++ Programming Language*. Addison-Wesley, Boston, wydanie 4, 2013.

[Wik25] ROS Wiki. URDF: Unified Robot Description Format. [Dokumentacja online], 2025. https://wiki.ros.org/urdf.

[Win24] Kamil Winnicki, *Sterowanie chodem sześcionożnego robota kroczącego*. Praca inżynierska, Politechnika Wrocławska, Wrocław, 2024. https://kcir.pwr.edu.pl/~mucha/Pracki/Kamil_Winnicki_praca_inzynierska.pdf.

[Zie03] Teresa Zielińska, *Maszyny kroczące: podstawy, projektowanie, sterowanie i biologiczne wzorce*. Wydawnictwo Naukowe PWN, Warszawa, 2003.

Spis rysunków