



**Politechnika
Śląska**

PROJEKT INŻYNIERSKI

Sterowanie robotem mobilnym typu HEXAPOD

Maksymilian KISIEL

Nr albumu: 296829

Kierunek: Automatyka i Robotyka

Specjalność: TI

PROWADZĄCY PRACĘ

Dr inż. Krzysztof Jaskot

KATEDRA Automatyki i Robotyki

Wydział Automatyki, Elektroniki i Informatyki

Gliwice 2024

Tytuł pracy

Sterowanie robotem mobilnym typu HEXAPOD

Streszczenie

Celem niniejszej pracy jest zaprojektowanie i wdrożenie systemu sterowania dla robota mobilnego typu hexapod, wykorzystując platformę Raspberry Pi jako centralną jednostkę obliczeniową. Komunikacja z serwomechanizmami odbywa się za pośrednictwem Dockera, ROS2 i protokołu UART Half-Duplex. W pracy analizowane są różne rodzaje chodów w kontekście ich zastosowania w robotach z sześcioma odnóżami. Omawiane są algorytmy chodu oraz algorytm komunikacji z użytkownikiem, który operator wykorzystuje do sterowania robotem. Praca zawiera pełną instrukcję instalacji i obsługi, wraz z przykładowymi operacjami. Weryfikacja i walidacja systemu obejmują testowanie różnych funkcji robota, dla którego głównym wzorcem testowym jest wzorzec trójpodporowy. W podsumowaniu pracy dokonano analizy uzyskanych wyników w kontekście postawionych celów, wskazano obszary do dalszej pracy oraz omówiono napotkane problemy.

Słowa kluczowe

Hexapod, robotyka, sterowanie, ROS2, Raspberry Pi

Thesis title

HEXAPOD mobile robot control

Abstract

The objective of this thesis is to design and implement a control system for a hexapod mobile robot, using the Raspberry Pi platform as the central processing unit. Communication with the servos is achieved via Docker, ROS2, and the UART Half-Duplex protocol. The paper analyses various types of gaits in the context of their application in six-legged robots. It discusses gait algorithms and the user communication algorithm that the operator uses to control the robot. The task involves providing comprehensive installation and operation instructions, along with sample operations. To verify and validate the system, the various functions of the robot are tested, with the primary gait pattern being a tripod gait. The paper concludes by analysing the results obtained in the context of the set objectives, identifying areas for further work, and discussing the encountered problems.

Key words

Hexapod, robotics, control theory, ROS2, Raspberry Pi

Spis treści

1 Wstęp	1
1.1 Cel pracy	2
1.2 Zakres pracy	4
1.3 Zwięzła charakterystyka rozdziałów	5
1.4 Wkład autora	6
2 Analiza istniejących rozwiązań	7
2.1 Chód i jego dynamika	7
2.2 Rodzaje chodu	9
2.2.1 Liczba odnóży	9
2.2.2 Ułożenie odnóży	10
2.2.3 Punkty podparcia	10
2.2.4 Wzorce chodu	11
2.3 Raspberry Pi	12
2.4 Docker	12
2.5 ROS2	13
3 Wymagania i narzędzia	15
3.1 Wymagania funkcjonalne	15
3.2 Wymagania niefunkcjonalne	16
3.3 Interfejsy i narzędzia	17
3.3.1 Narzędzia	17
3.3.2 Interfejsy	19
4 Specyfikacja zewnętrzna	21
4.1 Wymagania sprzętowe	21
4.2 Wymagane oprogramowanie	22
4.3 Zdalne połaczenie - prywatna sieć	22
4.4 Pierwsze uruchomienie programu	24
4.5 Wyłączanie programu	27
4.6 Kolejne uruchomienia programu	27

4.7	Możliwe problemy	28
4.7.1	Zmiana konwertera USB2TTL spowodowała, że program przestał działać.	28
4.7.2	Wywołanie polecenia ‘colcon build’ skutkuje pojawiением się błędów i aplikacja nie działa.	28
4.8	Przykładowe działanie robota	29
4.8.1	Zmiana położenia względem osi Z w trakcie chodu	29
4.8.2	Rotacja przeciwnie do ruchu wskazówek zegara	30
5	Szczegóły techniczne	31
5.1	Synteza układu	31
5.2	Oprogramowanie	35
5.3	Architektura proponowanego rozwiązania	38
5.3.1	Aspekty sprzętowe	38
5.3.2	Aspekty programowe	38
5.3.3	Elementy łączące	39
5.4	Algorytmy	39
5.4.1	Algorytm chodu	39
5.4.2	Algorytm komunikacji	41
6	Weryfikacja i walidacja	43
6.1	Wykryte i poprawione błędy	44
7	Podsumowanie i wnioski	47
7.1	Rozbudowa funkcjonalna	48
7.2	Napotkane problemy	49
	Bibliografia	52
	Spis skrótów i symboli	55
	Lista dodatkowych plików, uzupełniających tekst pracy	57
	Spis rysunków	59

Rozdział 1

Wstęp

Współczesna robotyka mobilna, w kontekście dynamicznego rozwoju technologii, stanowi obszar o ogromnym potencjale rewolucyjnym, wprowadzając nowe możliwości oraz stawiając przed naukowcami i inżynierami wyzwania, których rozwiązywanie przyczynia się do postępu w wielu dziedzinach. Jednym z najbardziej fascynujących przykładów innowacji w tej dziedzinie są roboty mobilne typu hexapod, które są częścią grupy robotów kroczących i dzięki swojej unikalnej konstrukcji z sześcioma odnóżami zdolne są do poruszania się w różnorodnych i trudnych warunkach terenowych, w których tradycyjne podejścia oparte na kołowych podwoziach spotykają pewne ograniczenia. Sprawia to, że hexapody zyskują na popularności, ponieważ posiadają wiele zalet, do których zaliczyć można, chociażby wspomniane poruszanie się po nierównym podłożu, czy zwiększoną odporność na awarie. Odporność ta wynika z dużej ilości dostępnych odnóży, których wykorzystanie może zostać zmienione w trakcie pracy poprzez zmianę wzorca chodu, co minimalizuje wpływ uszkodzenia jednego z odnóży na funkcjonalność robota. W dziedzinie robotyki mobilnej istnieje zatem potrzeba opracowania efektywnych rozwiązań sterowania dla tego typu robotów, pozwalających na optymalne wykorzystanie ich zdolności poruszania się.

Pomimo tego, że roboty kroczące nie są obecnie szeroko wykorzystywane, to hexapody, jako roboty o zaawansowanej konstrukcji, mogłyby znaleźć zastosowanie w wielu obszarach, ponieważ są tworami niezwykle wszechstronnymi. Obecnie największym rozwojem cieszą się roboty kroczące wykorzystujące dwie lub cztery kończyny, czyli odpowiednio bipedy (w szczególności roboty humanoidalne) oraz quadropedy, które wyglądem i sposobem poruszania wzorują się na ssakach lub gadach.

Przykłady zastosowań hexapodów obejmują eksplorację nieznanych, trudno dostępnych obszarów na Ziemi, czy też terenów dotkniętych różnorodnymi katastrofami. Hexapody mogą być wykorzystywane do zbierania danych, badania środowiska i dostarczania informacji z obszarów trudno dostępnych dla człowieka. Co więcej, w sytuacjach kryzysowych, jak trzęsienia ziemi czy inne katastrofy naturalne, hexapody mogą odgrywać kluczową rolę w poszukiwaniu ofiar, zwłaszcza w miejscowościach, do których trudno byłoby dotrzeć, wykorzystując inne środki np. mobilne platformy poruszające się za pomocą kół.

Jednakże, mimo szerokiego spektrum możliwych zastosowań, skomplikowana konstrukcja, która zawiera wiele stopni swobody, oraz ich zdolność do poruszania się w różnych kierunkach i na różnych powierzchniach sprawiają, że efektywne sterowanie nimi jest wyzwaniem. Projektowanie systemu sterowania, który umożliwia skoordynowane ruchy wszystkimi sześcioma odnóżami, jest kluczowe dla efektywnego wykorzystania potencjału tych robotów w praktyce. Ponadto, istnieje potrzeba zintegrowania tych robotów z nowoczesnymi technologiami informatycznymi, aby umożliwić zdalne sterowanie, monitorowanie oraz implementację autonomicznych funkcji. Aspekt sterowania stanowi główny cel tej pracy, która skupia się na implementacji zaawansowanych algorytmów sterowania, opartych na komputerze jednopłytkowym Raspberry Pi oraz wykorzystaniu otwartego standardu ROS2 (ang. *Robot Operating System*), aby stworzyć elastyczne i rozbudowane środowisko sterowania dla hexapoda.

Projektowanie systemu sterowania dla hexapoda przy użyciu komputera jednopłytkowego SBC (ang. *Single Board Computer*), takiego jak Raspberry Pi, wpisuje się w nurt trendów związanych z wykorzystaniem otwartych standardów i narzędzi, takich jak ROS2. Dążenie do modułowego i elastycznego projektu systemu sterowania jest kluczowe, aby umożliwić rozwój i dostosowanie hexapoda do różnych zastosowań oraz ewentualne integracje z innymi systemami informatycznymi. Osiągnięcie tych celów wymaga nie tylko znajomości technologii informatycznych i ich ograniczeń, ale również głębokiego zrozumienia specyfiki robotyki mobilnej i wyzwań z nią związanych.

1.1 Cel pracy

Głównym celem niniejszej pracy jest zaprojektowanie, implementacja oraz ewaluacja kompleksowego systemu sterowania dla robota mobilnego typu hexapod, który zostanie zbudowany z wykorzystaniem komputera jednopłytkowego Raspberry Pi. Zadaniem tego systemu jest umożliwienie hexapodowi efektywnego poruszania się w zróżnicowanych warunkach terenowych oraz wykonywania zdefiniowanych zadań.

W ramach tego celu praca skoncentruje się na stworzeniu skonteneryzowanego środowiska opartego na ROS2, umożliwiającego efektywną komunikację z serwomechanizmami oraz implementację zaawansowanych algorytmów sterowania, które uwzględniają specyficzne cechy konstrukcyjne hexapoda, takie jak sześć odnóży w układzie pajaka i ich duża liczba stopni swobody. Implementacja ma obejmować również zastosowanie konteneryzacji za pomocą platformy Docker, co zapewni modułowość i elastyczność systemu.

W ramach tego ogólnego celu praca skupi się na kilku kluczowych obszarach:

1. Skonteneryzowane środowisko ROS2:

Stworzenie środowiska opartego na ROS2, które będzie skonteneryzowane, czyli stworzone w kontenerze, który działa w sposób zbliżony do wirtualnej maszyny, przy użyciu technologii Docker. To umożliwia izolację i łatwe zarządzanie zależnościami, co z kolei wspomoże w utrzymaniu systemu oraz jego ewentualnej skalowalności.

Za największy atut uznać można zastosowanie tej technologii do zwiększenia bezpieczeństwa poprzez izolację danych systemowych w systemie macierzystym (w tym wypadku Ubuntu Server) od danych aplikacji sterującej robotem. Kolejną zaletą jest powtarzalność konfiguracji, co może posłużyć do zwiększenia skali projektu, np. produkcja masowa.

2. Komunikacja z serwomechanizmami:

Opracowanie efektywnego systemu komunikacji pomiędzy komputerem jednopłytowym Raspberry Pi i serwomechanizmami Dynamixel sterującymi ruchem odnóży hexapoda. W tym kontekście praca będzie obejmować zarówno sprzętowe, jak i programowe aspekty zapewniające płynną synchronizację ruchu kończyn.

3. Algorytmy sterowania:

Implementacja zaawansowanych algorytmów sterowania, uwzględniających specyficzne cechy konstrukcyjne hexapoda, takie jak sześć odnóży i duża liczba stopni swobody. Algorytmy te będą miały na celu osiągnięcie precyzyjnego i skoordynowanego ruchu w różnych kierunkach.

4. Otwarte standardy i modułowość:

Dodatkowym atutem pracy będzie skupienie się na otwartych standardach, takich jak ROS2, co umożliwia nie tylko skutecną realizację celu, ale również ułatwia rozwój systemu przez społeczność, jeżeli zdecydowano by się na udostępnienie interfejsów i narzędzi programistycznych (API, ang. *Application Programming Interface*) w celu rozwoju robotyki mobilnej. Mogłoby to prowadzić do rozwoju robotyki zarówno amatorskiej, jak i profesjonalnej, gdyż wiele osób zyskałoby dostęp do gotowych materiałów, co mogłoby zwiększyć zainteresowanie tą dziedziną.

Ponadto wykorzystanie otwartych standardów, zwłaszcza ROS2, umożliwia integrację z istniejącymi rozwiązaniami oraz zwiększa elastyczność systemu. Dążenie do modułowego projektu pozwoli na łatwiejsze rozbudowywanie funkcjonalności oraz dostosowywanie systemu do różnych scenariuszy zastosowań.

Praktyczna implementacja algorytmów sterowania na platformie Raspberry Pi oraz ich integracja z ROS2 dostarczą wartościowego doświadczenia praktycznego dla przyszłych

badań w dziedzinie robotyki mobilnej. Otwarta natura systemu zapewni potencjał do dalszego rozwoju poprzez wkład społeczności w rozwijanie i udostępnianie dodatkowych funkcji oraz rozszerzeń do projektu, a także integracji z innymi systemami oraz dopasowanie do różnych zastosowań w dziedzinie robotyki mobilnej.

1.2 Zakres pracy

Zadaniem pracy jest kompleksowe opracowanie układu sterowania dla robota mobilnego typu hexapod, skupiając się na konkretnych aspektach implementacyjnych i funkcjonalnych. Zakres pracy obejmuje następujące obszary:

1. Implementacja skonteneryzowanego środowiska ROS2:

Stworzenie skonteneryzowanego środowiska opartego na ROS2 przy użyciu Docker, obejmującego niezbędne narzędzia, biblioteki i środowisko. To środowisko powinno być gotowe do instalacji na komputerze jednopłytkowym Raspberry Pi i przygotowane do replikacji.

2. Komunikacja z serwomechanizmami:

Implementacja protokołów komunikacyjnych umożliwiających efektywną interakcję między komputerem jednopłytkowym Raspberry Pi a serwomechanizmami hexapoda. Zagadnienia obejmują synchronizację odnóży, reagowanie na sygnały sterujące oraz obsługę ewentualnych awarii.

3. Algorytmy sterowania:

Realizacja algorytmów sterowania hexapodem uwzględniających zarówno podstawowe komendy ruchu, jak i bardziej zaawansowane strategie sterowania.

4. Integracja z komputerem jednopłytkowym Raspberry Pi:

Zapewnienie poprawnej integracji pomiędzy stworzonym środowiskiem ROS2 i komputerem jednopłytkowym. Obejmuje to konfigurację interfejsów komunikacyjnych, zapewnienie płynnej transmisji danych i synchronizację działania systemu.

5. Zastosowanie konteneryzacji w rozwoju systemu:

Wykorzystanie konteneryzacji (Docker) do zapewnienia modułowości i elastyczności systemu. Dążenie do łatwej rozbudowy funkcjonalności, integracji nowych komponentów oraz zapewnienie łatwej replikacji środowiska.

6. Testowanie i ewaluacja:

Przeprowadzenie testów funkcjonalnych układu sterowania w różnych scenariuszach.

7. Dokumentacja:

Prowadzenie szczegółowej dokumentacji technicznej obejmującej wszystkie etapy implementacji, konfiguracji oraz testów. Opracowanie raportu prezentującego osiągnięcia, wnioski oraz potencjalne kierunki rozwoju systemu.

Poprzez realizację powyższych punktów, praca ma na celu nie tylko stworzenie działającego systemu sterowania dla hexapoda, ale także dostarczenie pełnej listy procesów i decyzji wpływających na efekt końcowy projektu. Skoncentrowanie się na detalach implementacji oraz staranność w przeprowadzeniu testów pozwoli na pełne zrozumienie funkcjonalności stworzonego systemu i wprowadzenie ewentualnych usprawnień w trakcie rozwoju.

1.3 Zwięzła charakterystyka rozdziałów

Drugi rozdział skupia się na analizie istniejących rozwiązań, które są kluczowym krokiem w zrozumieniu kontekstu projektu. Zawiera ogólne wprowadzenie, a następnie szczegółowo omawia zagadnienia związane z chodem, dynamiką, rodzajami chodu oraz kluczowymi dla projektu technologiami: Raspberry Pi, Docker, i ROS2.

Rozdział trzeci definiuje wymagania funkcjonalne i niefunkcjonalne projektu oraz prezentuje interfejsy i narzędzia, a także szczegółowe informacje na ich temat.

W rozdziale czwartym przedstawiono dokumentację, czyli specyfikację zewnętrzną projektu, koncentrując się na wymaganiach sprzętowych, oprogramowaniu, a także zagadnieniach związanych z instalacją, konfiguracją, i obsługą stworzonego systemu.

Rozdział piąty omawia szczegóły techniczne, w tym genezę wykorzystywanych obliczeń i opisów matematycznych, syntezę układu, oprogramowanie, architekturę proponowanego rozwiązania, oraz algorytmy chodu i komunikacji.

Przedostatni rozdział pracy poświęcony jest weryfikacji i walidacji systemu, przedstawiając sposób testowania, błędy, które wystąpiły w trakcie projektu, oraz organizację eksperymentów.

Ostatni rozdział pracy stanowi podsumowanie i wnioski. Analizuje uzyskane wyniki w kontekście celów projektu, wskazuje kierunki dalszych prac i przedstawia konkretne pomysły rozwoju, a także podsumowuje napotkane problemy.

1.4 Wkład autora

Autor niniejszej pracy odpowiedzialny był za takie elementy jak:

1. Projektowanie architektury systemu:

Autor odpowiadał za projektowanie ogólnej architektury systemu sterowania hexapodem. Obejmowało to określenie struktury oprogramowania, interfejsów komunikacyjnych i organizacji komponentów programu.

2. Implementacja środowiska ROS2:

Autor przeprowadził konfigurację narzędzi, bibliotek i środowiska wykonawczego uwzględniając specyfikę pracy na komputerze jednoprzykowym Raspberry Pi.

3. Komunikacja z serwomechanizmami:

Wkład autora obejmuje projektowanie i implementację protokołów komunikacyjnych, strategii synchronizacji ruchu odnóży oraz procedur obsługi błędów w kontekście komunikacji między komputerem jednoprzykowym a serwomechanizmami.

4. Algorytmy sterowania:

Autor zdefiniował i zaimplementował algorytmy sterowania hexapodem, biorąc pod uwagę unikalne cechy konstrukcyjne tego typu robota. Obejmowało to zarówno podstawowe algorytmy sterowania, jak i bardziej zaawansowane strategie programowe, czy komunikacyjne.

5. Zastosowanie konteneryzacji w rozwoju systemu:

Autor był odpowiedzialny za implementację konteneryzacji (Docker), co obejmowało tworzenie kontenerów, zarządzanie zależnościami, a także dbanie o modułowość i elastyczność systemu.

6. Testowanie i wyniki:

Wkład autora w tym obszarze obejmuje planowanie i przeprowadzanie testów funkcjonalnych w różnych warunkach i pod kątem spełnienia założeń projektowych.

7. Dokumentacja i wnioski:

Wkład autora w tym obszarze obejmuje analizę wyników, wnioski płynące z przeprowadzonych badań oraz propozycje kierunków dalszego rozwoju systemu. Autor był również odpowiedzialny za stworzenie dokumentacji technicznej, która pozwala na reprodukcję i uruchamianie środowiska odpowiedzialnego za sterowanie robotem.

Rozdział 2

Analiza istniejących rozwiązań

Robotyka inspirowana biologią to dynamicznie rozwijająca się dziedzina, która czerpie z natury, aby opracować zaawansowane systemy sterowania dla robotów. Architektury sterowania oparte na tych projektach, umożliwiają robotom wykonywanie złożonych zadań, takich jak chodzenie, pływanie, czołganie się, skakanie i latacie [4]. Te różnorodne zadania są możliwe dzięki specyficzny formom ruchu, znanych jako wzorce chodu (ang. *gait*), które czasami przekraczają wydajność konwencjonalnej inżynierii.

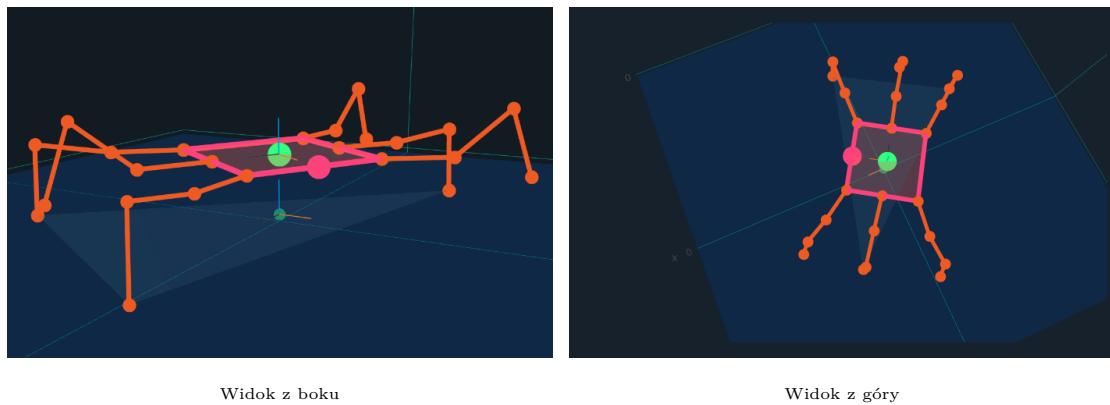
W kontekście robotów poruszających się za pomocą kończyn (w przypadku hexapodów nazywanych odnóżami), ich głównym konkurentem są tradycyjne konstrukcje robotów o podstawie kołowej, szeroko stosowane w rzeczywistych zastosowaniach, takich jak eksploracja (szczególnie łaziki planetarne) oraz pojazdy komercyjne do transportu ludzi i materiałów. Niemniej jednak roboty posiadające odnóża, mogą mieć przewagę na nierównym terenie, gdzie koła przestają być odpowiednim rozwiązaniem ze względu na trudności z utrzymaniem statycznego tarcia o podłożę. Roboty poruszające się za pomocą odnóży oferują większą stabilność i kontrolę nad ciałem, co pozwala im manewrować w bardziej trudnym terenie niż roboty kołowe. Dlatego większość zastosowań tych robotów związana jest z przemierzaniem trudnego terenu jak przemieszczanie się po obszarach zawałonych gruzem i pokonywanie przeszkód. Zastosowania te obejmują działania ratunkowe, konserwację i naprawy w trudnych do nawigacji środowiskach.

2.1 Chód i jego dynamika

W dziedzinie robotyki hexapody, czyli roboty wyposażone w sześć odnóży, zdobywają popularność dzięki swojej wszechstronności. Hexapody oferują znaczną liczbę statycznie stabilnych chodów, co stanowi kluczową przewagę w porównaniu do robotów czworonożnych, które posiadają tylko jeden taki chód, oraz robotów dwunożnych, które w ogóle pozbawione są tego rodzaju chodu [3, 8]. Pojęcie "chód" odnosi się tutaj do sposobu lokomocji, gdzie kończyny robota nawiązują kontakt z podłożem w dyskretnych punktach. W przypadku robotów kołowych kontakt z podłożem jest ciągły, co stanowi zasadniczą

różnicę.

W kontekście hexapodów istotnym terminem jest wspomniany "chód statycznie stabilny". Oznacza on rodzaj chodu, w którym robot zachowuje stabilność, nawet gdy ruch w jego kończynach zostanie wstrzymany w dowolnej chwili. Spowodowane jest to tym, że rzut środka ciężkości ciała znajduje się w środku wielokąta podparcia, którym w naszym przypadku najczęściej jest trójkąt. Wielokąt widoczny jest na rysunku 2.1. Analizując ten rysunek możemy również wyznaczyć zapas stabilności statycznej poprzez wyznaczenie najmniejszej długości odcinka poprowadzonego od rzutowanego środka ciężkości do krawędzi figury [5, 8]. Istnieje również chód "dynamicznie stabilny", czyli taki, w którym robot zachowuje stabilność tylko wtedy, gdy jego ciało znajduje się w ciągłym ruchu. Gdy tylko ruch zostanie zatrzymany, w dowolnej chwili, robot traci stabilność (środek ciężkości znajduje się poza wielokątem podparcia). W przypadku hexapodów możliwe jest przełączanie pomiędzy tymi dwoma typami chodu. Posiadanie sześciu odnóży pozwala hexapodom na szeroki zakres ruchów i prędkości, umożliwiając płynne przełączanie się między różnymi chodami i dostosowywanie się do zmiennych wymagań. Dodatkowo fakt, że ciało hexapoda jest zawsze wspierane przez co najmniej trzy przeciwnostawne odnóża (trójkąt stabilności), gwarantuje, że w normalnych warunkach pracy nigdy nie osiągnie ono statycznie niestabilnych położień.



Rysunek 2.1: Przedstawienie wielokąta podparcia na podstawie symulacji [9, 10]

Roboty mogą poruszać się chodem statycznie stabilnym, jeżeli mają co najmniej cztery kończyny, jednak w tym przypadku możliwe jest zastosowanie jedynie chodu polegającego na przenoszeniu jednej kończyny naraz. Takie podejście wiąże się z koniecznością wykonania czterech niezależnych cykli, aby pokonać odległość jednego kroku. Wykorzystanie robotów o sześciu odnóżach daje znacznie większe możliwości, ponieważ nie trzeba się już ograniczać do ruchu wyłącznie jedną kończyną. Możliwe jest zastosowanie wzorca trójpodporowego, który jest najbardziej popularnym spośród wszystkich dostępnych hexapodów. Wzorzec ten pozwala przemieszczać aż trzy odnóża jednocześnie, podczas gdy ciało stabilnie podparte jest pozostałymi odnóżami tworzącymi trójkąt. Skutkiem zastosowania takiego chodu jest pokonanie dystansu jednego kroku w dwóch cyklach.

Chód statycznie stabilny gwarantuje ciągłą stabilność robota i jest łatwiejszy w implementacji, ponieważ nie wymaga ciągłego utrzymywania równowagi. Chód dynamicznie stabilny jest za to mniej skomplikowany w kontekście złożoności konstrukcji z uwagi na mniejszą wymaganą liczbę kończyn.

2.2 Rodzaje chodu

Rozważając dalsze aspekty chodu, możemy skupić się na jego różnych rodzajach, które mogą być zależne od wielu czynników, takich jak ogólna liczba kończyn, liczba kończyn dotykających podłoża w danej chwili, ich ułożenie, czy sposób, w jaki są przenoszone [1].

2.2.1 Liczba odnóży

W przypadku ogólnej liczby odnóży możemy wyróżnić cztery istotne grupy:

- Skrajny przypadek, w którym robot posiada tylko i wyłącznie jedną kończynę - w takim przypadku robot skacze. Każda inna próba przemieszczania się sprawia, że robot traci stabilność, czyli się przewraca. Alternatywą może być czołganie się, czy też pełzanie, w którego trakcie robot wykorzystuje kończynę do odpychania się od podłoża.
- Chód na dwóch kończynach - roboty poruszają się w sposób zbliżony do człowieka.
- Chód wykorzystujący cztery kończyny - poruszanie się w sposób zbliżony do zwierząt, np. psów, czy też gadów. Ten rodzaj znacznie różni się od wymienionych poprzedników.
- Chód wykorzystujący sześć odnóży - wykorzystywany przez hexapody to rodzaj chodu, który również wyróżnia się na tle pozostałych, szczególnie biorąc pod uwagę stabilność konstrukcji (nie tylko w trakcie chodu).

Roboty posiadające więcej niż sześć odnóży nie wprowadzają już większych zmian w kwestii mechaniki ruchu.

2.2.2 Ułożenie odnóży

Podział ze względu na ułożenie odnóży obejmuje grupy takie jak:

- dwie kończyny ułożone symetrycznie po obu stronach i równolegle wzduż ciała, np. ssaki,
- cztery kończyny ułożone bocznie i równolegle wzduż ciała, np. gady i ssaki czworożne,
- sześć odnóży rozmieszczonej wokół ciała, mogą być w układzie pajaka, lub w układzie równoległym, np. owady.

W przypadku rozmieszczenia kończyn dla grupy quadropedów, ale w szczególności dla hexapodów, należy również wziąć pod uwagę istnienie możliwości, że poszczególne pary odnóży mogą mieć różną długość lub kształt. W robotyce znaczącą popularnością cieszą się jednak roboty o jednakowych odnóżach, co w szczególności uwarunkowane jest złożonością algorytmu sterowania, jak i względami ekonomicznymi. Hexapody mogą mieć, jak wspomniano powyżej, różny układ odnóży - równoległy lub pajaka. Pierwszy z nich zakłada, że kończyny ułożone są podobnie jak u ssaków czworożnych lub gadów, czyli równolegle względem siebie i pod korpusem. Drugi układ zakłada natomiast, że pierwsza i ostatnia para odnóży zostaną odchylone o taki sam kąt względem pary środkowej. Przednia para zostaje wysunięta do przodu o kąt od 30° do 40° , a tylna cofnięta symetrycznie względem przedniej.

Ta praca skupia się na sterowaniu odnóżami w układzie pajaka, który zapewnia bezkolizyjną trajektorię w trakcie wykonywania kroku i znacząco ułatwia tworzenie algorytmu sterowania. Należy zwrócić jednak uwagę na to, że po odchyleniu odnóża będą się poruszały zgodnie zadaną trajektorią, która również będzie pod kątem względem kierunku ruchu robota. Zastosowanie takiego podejścia może wiązać się z większym obciążeniem podczas chodu, ponieważ w trakcie odpychania robota, nie wszystkie końce odnóży poruszają się równolegle względem siebie. Pomimo tej drobnej niedoskonałości, generowana w ten sposób trajektoria jest jednak bardzo szybką i skuteczną metodą umożliwiającą poruszanie się hexapoda, niezależnie od wskazanego kierunku i rodzaju ruchu [5].

2.2.3 Punkty podparcia

Roboty o sześciu odnóżach swoją konstrukcją, czyli parametrami fizycznymi, czerpią inspirację z natury, w szczególności z niektórych owadów jak muchy, czy mrówki. Wzorce chodu są więc ściśle powiązane z tymi występującymi u owadów [8, 22]. Chody te charakteryzują się, wspomnianą w sekcji 2.1, stabilnością statyczną, która posiada między innymi wskaźnik zapasu stabilności. Poprawnie zrealizowany algorytm sterowania powinien zapewnić odpowiednią minimalną wartość tego parametru, aby robot się nie przewrócił. Analizując dotychczasowe informacje, można określić w łatwy sposób, że hexapod

musi być podparty w jednym momencie przez co najmniej trzy odnóża, aby zachować stabilność. Oznacza to, że w trakcie chodu możemy wykorzystywać ruch:

- jednego odnóża, podczas gdy pozostałe utrzymują konstrukcję (chód metachroniczny),
- dwóch odnóży, podczas gdy pozostałe utrzymują konstrukcję (chód czterokontaktowy lub gąsienicowy),
- trzech odnóży, podczas gdy pozostałe utrzymują konstrukcję (chód trójpodporowy),

Istnieje również możliwość łączenia ze sobą powyższych ruchów, czego przykładem jest chód czterokontaktowy, w którym to najpierw przedstawiane są po dwie kończyny, a na koniec pozostała para przedstawiana jest pojedynczo odnóżem za odnóżem.

2.2.4 Wzorce chodu

W rozważaniach dotyczących wzorców chodu dla hexapodów istnieje szereg charakterystyk, które należy wziąć pod uwagę w kontekście doboru odpowiedniego wzorca dla konkretnej aplikacji. W niniejszej sekcji przedstawione zostaną cztery główne wzorce chodu.

1. **Chód metachroniczny:** Jest to najwolniejszy z chodów hexapodów, gdzie odnóża są przedstawiane jedno za drugim, co oznacza, że odległość jednego kroku pokonywana jest po sześciu cyklach. Charakteryzuje się niskim współczynnikiem obciążenia dla każdego odnóża, wynoszącym $\frac{1}{6}$. Dodatkowo jego duże zapasy stabilności zwiększa prawdopodobieństwo utrzymania stabilności robota nawet w trudnych warunkach terenowych.
2. **Chód czterokontaktowy:** Ten wzorzec cechuje się tym, że w dwóch fazach chodu przedstawiane są jednocześnie aż dwa odnóża, podczas gdy w pozostałych dwóch fazach tylko jedno. Takie podejście znaczco zwiększa prędkość poruszania się bez znacznego pogorszenia zapasu stabilności. Współczynnik obciążenia dla tego chodu wynosi $\frac{1}{4}$.
3. **Chód gąsienicowy:** Charakteryzuje się przedstawianiem kończyn parami w kolejności: tylne, środkowe, przednie. Ten wzorzec nie jest odpowiedni do pokonywania nieregularnych przeszkód z uwagi na niewielki zapas stabilności. Współczynnik obciążenia dla tego chodu wynosi $\frac{1}{3}$.
4. **Chód trójpodporowy:** Jest najszybszym z dotychczas wymienionych wzorców oraz najbardziej popularnym. To również najszybszy możliwy chód, który jednocześnie zapewnia statyczną stabilność hexapoda. Współczynnik obciążenia dla tego chodu wynosi $\frac{1}{2}$.

W ramach niniejszej pracy skoncentrowano się na zaimplementowaniu wzorca chodu trójpodporowego, a decyzja ta została podjęta ze względu na połączenie prędkości i stabilności. Chód trójpodporowy, jako wybrany wzorzec, umożliwia efektywne poruszanie się robota, utrzymując jednocześnie wysoki poziom stabilności. Przyjęcie tego konkretnego wzorca wynika z jego zdolności do zachowania równowagi nawet w dynamicznym środowisku, co sprawia, że jest on szczególnie przydatny w kontekście robotów mobilnych. Warto jednak podkreślić, że struktura opracowanej aplikacji została zaprojektowana w sposób modułowy, co otwiera możliwość łatwej implementacji innych wzorców chodu w przyszłości. Ta elastyczność stanowi istotny element, który umożliwia dostosowanie zachowań robota do różnorodnych warunków i wymagań środowiska, zwiększać tym samym uniwersalność systemu, który może zmieniać wzorzec chodu w zależności od wymagań środowiskowych.

2.3 Raspberry Pi

W trakcie realizacji tego projektu centralną jednostką obliczeniową jest komputer jednopłytkowy (ang. *SBC, Single Board Computer*) Raspberry Pi 4B, odpowiadający za sterowanie robotem mobilnym typu hexapod. Wybór modelu Raspberry Pi 4B wynika z jego powszechniej popularności, łatwej dostępności oraz wystarczającej mocy obliczeniowej do obsługi algorytmów sterowania. Dodatkowo w porównaniu do konkurencyjnych rozwiązań, takich jak Arduino, Raspberry Pi oferuje rozszerzone możliwości funkcjonalne. Integracja Raspberry Pi z systemem sterowania robota umożliwia płynne realizowanie zadań związanych z poruszaniem się. W kolejnych sekcjach pracy dokładnie omówione zostaną aspekty związane z zastosowaniem Raspberry Pi w kontekście tego projektu, ze szczególnym uwzględnieniem jego integracji z ROS2 i Docker.

2.4 Docker

Wdrożenie Docker na Raspberry w ramach tego projektu stanowi kluczowy komponent, mający na celu utworzenie odizolowanego i przenośnego środowiska pracy dla systemu sterowania hexapoda. Docker to platforma do konteneryzacji aplikacji, co umożliwia wygodne zarządzanie zależnościami, konfiguracją oraz implementację. Użycie kontenera Docker ułatwi replikację projektu na różnych platformach sprzętowych oraz umożliwi łatwe udostępnianie i instalację projektu innym użytkownikom. W późniejszych etapach pracy zostaną dokładnie omówione kwestie związane z konfiguracją kontenerów Docker oraz ich integracją z systemem sterowania hexapoda kontrolowanych przez środowisko ROS2.

2.5 ROS2

Koncepcja integracji ROS2 (Robot Operating System 2) w ramach tego projektu stanowi kluczowy aspekt, mający na celu zapewnienie efektywnego i modułowego środowiska sterowania dla robota. ROS2 to system pośredniczący (ang. *middleware*) typu *open source*, czyli o publicznym kodzie źródłowym, przeznaczonym dla robotów. Jest to system oferujący szereg narzędzi i bibliotek ułatwiających rozwój, testowanie i kontrolę robotów w jednym centralnym miejscu. Poprzez zastosowanie ROS2, projekt ma na celu umożliwienie łatwej komunikacji między poszczególnymi komponentami systemu, co przyczyni się do modułowości i skalowalności całego rozwiązania. W kolejnych etapach pracy szczegółowo omówione zostaną aspekty implementacyjne oraz konfiguracyjne związane z wykorzystaniem ROS2.

Rozdział 3

Wymagania i narzędzia

Rozwiązanie, które zostało przygotowane, aby osiągnąć wyznaczony we wstępie cel, powinno spełniać szereg wymagań funkcjonalnych i niefunkcjonalnych, które zostaną dokładniej opisane w poszczególnych sekcjach bieżącego rozdziału, wraz z opisem wykorzystywanych narzędzi, które pozwolą te wymagania spełnić.

3.1 Wymagania funkcjonalne

Stworzone rozwiązanie powinno spełniać takie kluczowe funkcje jak:

1. Sterowanie robotem

System powinien zapewnić precyzyjne sterowanie hexapodem w wybrany sposób. Poruszanie się jest wymuszane i nadzorowane przez użytkownika za pomocą kontrolera, którym jest klawiatura. Robot powinien móc wykonywać różne rodzaje ruchu, między innymi poruszać się do przodu lub do tyłu, obracać się wokół własnej osi w kierunku wybranym przez użytkownika, a także zmieniać położenie swojego ciała w trakcie chodu, np. obniżanie lub podnoszenie. Pomoże to w zwiększeniu liczby możliwych zastosowań systemu.

2. Szybka adaptacja na zmieniające się komendy

Robot powinien szybko reagować na wydane przez użytkownika rozkazy i w jak najlepszy sposób zmieniać zadane położenie serwomechanizmu, aby robot nie stracił stabilności.

3. Niezawodna komunikacja

System powinien zapewniać niezawodną komunikację za pomocą każdego z zaimplementowanych interfejsów, które szerzej opisane zostały w rozdziale 3.3.

4. Modułowość i łatwe programowanie

Przygotowane rozwiązanie powinno umożliwiać łatwe tworzenie i implementację algorytmów i sposobów sterowania robota, a także zapewniać możliwość fizycznego rozszerzania funkcjonalności, np. poprzez dodanie w przyszłości modułu wizyjnego lub modułu serwera powiadamiania przez internet, np. MQTT.

5. Źródło zasilania zapewniające mobilność

Robot powinien być w stanie poruszać się swobodnie w wyznaczonym obszarze, który powinien być ograniczony jedynie przez zasięg sieci Wi-Fi. Aby osiągnąć ten cel, wszystkie komponenty hexapoda powinny być zasilane z baterii o wysokiej wydajności prądowej, jak np. ogniwa litowo-polimerowe Li-Po.

6. Wysokie bezpieczeństwo danych

Wykorzystywane interfejsy komunikacyjne (sekcja 3.3) powinny zapewniać możliwie bezpieczną komunikację pomiędzy modułami programu, a także pomiędzy wirtualnym środowiskiem kontenera Docker a głównym systemem operacyjnym komputera Raspberry Pi.

3.2 Wymagania niefunkcjonalne

Stworzony system powinien spełniać pewne wymogi jakościowe, aby zapewnić dokładność wykonywania funkcji robota. Do tych wymogów zaliczają się:

1. Precyzja sterowania

Hexapod powinien wykonywać ruchy z jak największą precyzją i powtarzalnością. System sterowania powinien zapewnić skoordynowane i dokładne poruszanie każdym odnóżem, co pozwoli osiągnąć docelową pozycję w przestrzeni. Jednocześnie należy zachować maksymalną szybkość ruchów silników, aby system działał w cyklach o przewidywalnej długości, czyli zachowywał deterministyczność.

2. Stabilność

Robot powinien zawsze utrzymywać równowagę, niezależnie od wykonywanej operacji i momentu, w którym zostanie ona przerwana. W przypadku chodu odnosi się to do wspomnianego w rozdziale 2.1 chodu statycznie stabilnego.

3. Wydajność

System powinien zapewniać szybkie i efektywne wykonywanie zadań. Robot musi poruszać się z odpowiednią prędkością i reagować szybko na zmianę sygnału sterującego, aby osiągnąć wysoką sprawność, czyli działać wydajnie.

4. Niezawodność

Robot mobilny powinien utrzymywać niski poziom zakłóceń i awarii, zarówno w kwestii fizycznej, sprzętowej, jak i programowej. Działania te mają za zadanie zapewnić jak największą bezawaryjność hexapoda.

5. Modułowość

Przygotowany system powinien być skonstruowany w sposób zapewniający łatwe dodawanie i modyfikowanie poszczególnych bloków programowych i sprzętowych. W przypadku wystąpienia błędów daje to również możliwość usunięcia uszkodzonych lub niekompletnych modułów z układu. Modułowość zwiększa także możliwości rozwoju i rozbudowy, gdyż nie jest wymagana całkowita restrukturyzacja kodu.

6. Skalowalność

Skalowalność odnosi się tutaj zarówno do możliwości zwiększenia złożoności i funkcji programu, czyli wykorzystanie modułowości, jak i do tworzenia kopii, czyli reprodukcji istniejącego rozwiązania. System sterowania powinien umożliwiać dostosowanie do aktualnych wymagań projektowych, a także pozwalać na rozbudowę.

7. Bezpieczeństwo

Aspekt bezpieczeństwa jest istotny, ponieważ robot nie powinien stwarzać zagrożenia dla innych, ani dla siebie samego. Wymaganie to obejmuje ochronę przed potencjalnymi zagrożeniami znajdującymi się w otoczeniu, jak i bezpieczne sterowanie robotem.

3.3 Interfejsy i narzędzia

System ten posiada wiele różnorodnych interfejsów oraz narzędzi, które są kluczowe dla efektywnej komunikacji i zarządzania hexapodem.

3.3.1 Narzędzia

1. Raspberry Pi 4B

Platforma ta jest mózgiem, centralną jednostką obliczeniową hexapoda. Komputer ten oferuje moc obliczeniową, która pozwala koordynować pracę wszystkich podsystemów i wykonywać niezbędne obliczenia do sterowania robotem. Zainstalowany system operacyjny to Ubuntu Server, który wybrany został z uwagi na niskie wymagania i niewielkie pasywne wykorzystanie zasobów. Pozwala on na instalację wszystkich wymaganych programów i bibliotek, a przy tym nie zajmuje dużo miejsca w przestrzeni dyskowej, gdyż nie posiada on interfejsu graficznego (ang.

GUI, Graphical User Interface). Do jednego z gniazd USB (ang. *Universal Serial Bus*) podpięto konwerter USB2TTL (ang. *USB to TTL, Universal Serial Bus to Transistor-Transistor Logic*), który pozwala na komunikację z serwomechanizmami. Na komputerze zainstalowano aplikację Docker, za pomocą której stworzono kontener z wirtualnym środowiskiem ROS2.

2. Docker

Docker to programistyczna platforma typu *open source*, która pozwala tworzyć, dostarczać i uruchamiać aplikacje w technologii kontenerowej. Kontenery są ideoowo zbliżone do maszyn wirtualnych, jednak nie zapewniają one wirtualizacji sprzętowej, a jedynie wirtualizację na poziomie systemu operacyjnego. Kontenery dzielą jądro systemu macierzystego (w tym wypadku Ubuntu Server) pomiędzy sobą i są nietrwałe. Uruchamia się je za pomocą obrazów, czyli zbiorów oprogramowania, które zawierają instrukcję jak dany kontener stworzyć. Obrazy są niezmienne, więc ponowne uruchamianie nie wymaga tworzenia go na nowo, jednak zmiany w instrukcji tworzenia kontenera wymagają utworzenia nowego obrazu. Kontenery można ogólnie zdefiniować jako działającą instancję obrazu Docker.

Platforma ta pozwala na szybkie i niezawodne działanie stworzonej przez nas aplikacji, niezależnie od środowiska obliczeniowego. Obrazy zawierają wszystkie niezbędne aplikacje, biblioteki, narzędzia systemowe i kod. Docker znajduje również zastosowanie w dziedzinie zarządzania przepływem pracy (narzędzia CI/CD, ang. *Continuous Integration / Continuous Delivery*), ponieważ pozwala na szybkie tworzenie, testowanie i wdrażanie aplikacji.

W tej pracy Docker, poza uruchamianiem kontenera z aplikacją sterującą, zapewnia także połączenie fizycznego konwertera podpiętego do gniazda USB z wirtualnym środowiskiem ROS2, a także mapuje do niego fragment przestrzeni dyskowej Raspberry, żeby możliwe było odizolowanie danych. Takie podejście pozwala na zwiększenie bezpieczeństwa systemu macierzystego i minimalizuje ryzyko uszkodzenia danych.

3. ROS2

ROS, czyli Robot Operating System, jest platformą programistyczną typu *open source*, która obejmuje narzędzia i biblioteki wykorzystywane do tworzenia oprogramowania robotów.

Tworzone aplikacje to tak zwane środowiska, a każde z nich składa się z modułów, które nazywane są paczkami. Paczki są elementami, które przechowują węzły (ang. *node*), czyli kod programu. Takie podejście zapewnia modułowość tworzonego przez nas oprogramowania. Paczki mają standardową strukturę, która kontrolowana jest przez wbudowane narzędzia środowiska ROS. Każdy węzeł zawarty w paczce

reprezentuje aplikację (proces), która może publikować (ang. *publisher*) lub subskrybować (ang. *subscriber*) temat (ang. *topic*) oraz udostępniać lub korzystać z usług (ang. *service*). Tematem nazywamy kanał komunikacyjny, który służy do komunikacji (wymiany danych) między węzłami. Jest to unikalna nazwa, dzięki której węzły mogą nawiązywać połączenia. Temat powstaje w momencie rozpoczęcia publikacji przez węzeł, a treść tematu nazywamy wiadomością (ang. *message*), która może mieć różne typy. Typy wiadomości natomiast nazywane są interfejsami.

4. Raspberry Pi Pico W

Ten niewielki mikrokontroler programowany jest w języku MicroPython, który został stworzony na podstawie języka Python, z myślą o mikrokontrolerach. Jest to język posiadający wiele funkcji i umożliwił wykorzystanie modułu bezprzewodowego, który umieszczony jest na płytce Raspberry. Moduł ten obsługuje możliwość pracy jako punkt dostępowy, umożliwiając zdalne połączenie z hexapodem. Ta funkcjonalność otwiera możliwości zdalnego sterowania oraz integracji z innymi interfejsami mobilnymi. Napisana aplikacja funkcjonuje również jako serwer www, który może zostać w przyszłości wykorzystany do uruchamiania aplikacji internetowej do sterowania robotem.

3.3.2 Interfejsy

Przytoczone narzędzia obsługują następujące interfejsy:

- Punkt dostępowy Wi-Fi (ang. *Access Point*) powstały z wykorzystaniem Raspberry Pi Pico W, który pozwala na zdalne połączenie się z robotem poprzez SSH i bezprzewodowe sterowanie za pomocą klawiatury.
- SSH, czyli Secure Shell, pełni kluczową funkcję w projekcie jako protokół służący do bezpiecznej wymiany danych pomiędzy użytkownikiem a mózgiem robota, czyli komputerem Raspberry. Pozwala to na zdalne zarządzanie kontenerem Docker oraz dostęp do systemu ROS2 bez konieczności fizycznego dostępu do urządzenia.
- Kontener Docker, który wymienia dane z rzeczywistym systemem operacyjnym na komputerze Raspberry, a także umożliwia wykorzystanie modułu konwertera USB2TTL przez wirtualne środowisko ROS2, aby zapewnić wymianę danych poprzez protokół UART.
- UART (Universal Asynchronous Receiver-Transmitter) to standardowy interfejs szeregowy, który pełni istotną rolę w wymianie danych pomiędzy komputerem Raspberry Pi a serwomechanizmami odpowiedzialnymi za ruch odnóży robota. Protokół ten jest szeregowy, co oznacza, że umożliwia komunikację między urządzeniami, przesyłając dane bit po bicie.

W tej pracy wykorzystano protokół UART w trybie half-duplex, co oznacza, że dane mogą być wysyłane w obu kierunkach, ale nigdy jednocześnie. Wiąże się to z tym, że nadające urządzenie nie jest w stanie w tym samym momencie odbierać danych. Zastosowanie tego trybu wynika z ograniczeń narzuconych przez producenta silników [19]. Sposób przesyłania wiadomości w tym trybie to wysłanie informacji o żądanym położeniu odnóży z komputera do serwomechanizmów oraz czekanie na informację zwrotną z aktualnym położeniem odnóży. Zaletą takiej komunikacji jest to, że wymagany jest tylko jeden przewód komunikacyjny, a silniki połączone są ze sobą w magistralę, przez co nie są wymagane osobne linie transmisyjne do każdego z nich.

- Środowisko ROS2, które poprzez protokół UART wysyła dane o zadanym położeniu do silników, a także umożliwia wymianę danych pomiędzy modułami programu poprzez stworzone tematy. Dla przykładu istotnym elementem jest temat odpowiadający za przesyłanie rozkazów użytkownika do algorytmu sterowania.

Rozdział 4

Specyfikacja zewnętrzna

W celu uruchomienia przygotowanej aplikacji należy spełnić wiele wymagań, zarówno sprzętowych, jak i programowych, jednak dołożono wszelkich starań, aby wymagany sprzęt był możliwie jak najtańszy, a oprogramowanie miało otwarte źródła i było aktywnie wspierane przez społeczność, dzięki czemu projekt ten ma szansę być wspierany i udoskonalany przez wiele lat.

4.1 Wymagania sprzętowe

W celu uruchomienia aplikacji należy spełnić następujące wymagania sprzętowe:

- Raspberry Pi 4B z Ubuntu Server - używana w tym projekcie wersja posiada 8 GB pamięci RAM (ang. *Random Access Memory*), ale przetestowano również model posiadający 4 GB pamięci i nie zauważono żadnych odchyleń w kwestii wydajności. Stworzona aplikacja jest na tyle kompaktowa, że wersja 2 GB RAM również powinna spełniać swoje zadanie, jednak nie zostało to przetestowane (1 sztuka),
- Moduł Raspberry Pi Pico W oparty na mikrokontrolerze RP2040 wymagany do stworzenia access point (1 sztuka),
- Konwerter "USB to TTL", który wykorzystywany jest do wysyłania wiadomości do serwomechanizmów (1 sztuka),
- Platforma HEXAPOD z silnikami Robotis Dynamixel AX-12A [18] (1 sztuka),
- Zasilacz lub baterie (litowo-polimerowe, Li-Po) o napięciu od 10 do 12 V, które zasilać będą serwomechanizmy (co najmniej 1 sztuka),
- Zasilacz lub baterie (typu powerbank) o napięciu 5 V, do zasilania komputera Raspberry Pi (1 sztuka).

4.2 Wymagane oprogramowanie

Na komputerze Raspberry Pi należy zainstalować aplikacje **Docker** oraz **Git**.

- Instalacja Git:

```
1 sudo apt-get install git -y
```

Listing 4.1: Komenda do instalacji aplikacji git

- Instalacja Docker:

```
1 curl -fsSL https://get.docker.com -o get-docker.sh
2 chmod +x get-docker.sh
3 sudo apt-get purge docker-ce docker-ce-cli containerd.io -y
4 ./get-docker.sh
5 sudo usermod -aG docker pi # (pi to nazwa naszego uzytkownika)
6 sudo systemctl unmask docker
7 sudo chmod 666 /var/run/docker.sock
8 pip3 -v install docker-compose
9 sudo systemctl start docker
10 sudo init 6
```

Listing 4.2: Zbiór komend do instalacji aplikacji Docker

4.3 Zdalne podłączenie - prywatna sieć

Aby prawidłowo podłączyć się do prywatnej sieci, co znacząco zwiększa bezpieczeństwo komunikacji pomiędzy klientem (naszym komputerem) a robotem, w pierwszej kolejności należy podłączyć do zasilania mikrokontroler Raspberry Pi Pico W i wgrać program, który widoczny jest na listingu 4.3.

```
1 try:
2     import usocket as socket    #importing socket
3 except:
4     import socket
5 import network    #importing network
6 import gc # garbage collection
7 gc.collect()
8 ssid = 'HEXAPOD Projekt'    #Set access point name
9 password = 'hexapodDevEnv061' #Set your access point password
10 ap = network.WLAN(network.AP_IF)
11 ap.config(essid=ssid, password=password)
12 ap.active(True)           #activating
13 while ap.active() == False:
14     pass
15 print('Connection is successful')
```

```

16 print(ap.ifconfig())
17 def web_page():
18     html = """
19     <html>
20         <head>
21             <meta name="viewport" content="width=device-width, initial-
22 scale=1">
23             <style>
24                 h1 {
25                     margin-top: 30px;
26                     text-align: center;
27                 }
28                 p {
29                     text-align: center;
30                 }
31                 a~{
32                     text-decoration: none;
33                     color: royalblue;
34                 }
35             </style>
36         </head>
37         <body>
38             <h1>Welcome to the HEXAPOD Project Webpage!</h1>
39             <p>Come visit our GitHub at: <a href="https://github.com/
40 revalew/HEXAPOD" target="_blank">HEXAPOD</a>!</p>
41         </body>
42     </html>"""
43     return html
44
45 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)      #creating socket
46     object
47 s.bind(('', 80))
48 s.listen(5)
49 while True:
50     conn, addr = s.accept()
51     print('Got a connection from %s' % str(addr))
52     request = conn.recv(1024)
53     print('Content = %s' % str(request))
54     response = web_page()
55     conn.send(response)
56     conn.close()

```

Listing 4.3: Kod konfigurujący Raspberry Pi Pico W jako Access Point

Kolejnym krokiem jest podłączenie Raspberry Pi oraz komputera do prywatnej sieci, za pomocą danych oznaczonych przez *ssid* oraz *password* (widoczne w linii 8 i 9 listingu 4.3).

4.4 Pierwsze uruchomienie programu

Po zainstalowaniu wymaganego oprogramowania (sekcja 4.2) oraz skonfigurowaniu punktu dostępowego (sekcja 4.3) należy wykonać następujące czynności:

- Przejść do wybranego przez siebie katalogu (np. `/home/pi/`, gdzie pi to nazwa użytkownika)

```
1 cd /home/pi/
```

lub utworzyć własny w dowolnym miejscu (np. `/root/`),

```
1 mkdir ~/HEXAPOD/
```

- Sklonować zawartość repozytorium za pomocą git

```
1 git clone https://github.com/revalew/HEXAPOD.git
```

- Przejść do folderu `docker` w sklonowanym wcześniej repozytorium

```
1 cd ~/HEXAPOD/docker
```

- Upewnić się, że konwerter TTL jest podpięty do portu USB i ma adres `/dev/ttyUSB0`,

```
1 dmesg | grep tty
```

jeżeli konwerter ma inny adres, należy zmienić go na odpowiedni w pliku `docker-compose.yml` w sekcji `devices`.

- Utworzyć obraz docker zgodnie z instrukcjami pliku `dockerfile`

```
1 docker build -t ros2 .
```

`ros2` to nazwa naszego kontenera.

- Uruchomić nowy kontener z ustawieniami zawartymi w `docker-compose.yml`

```
1 docker-compose up -d
```

- Uruchomić interaktywną sesję dla działającego kontenera

```
1 docker exec -it docker_ros2_1 bash
```

jeżeli występuje błąd nazwy (`docker_ros2_1` się nie zgadza), należy sprawdzić nazwę kontenera

```
1 docker ps
```

- Przejść do katalogu *hexapod_controller* w sklonowanym repozytorium, jeżeli kontener automatycznie się w nim nie uruchomi

```
1 cd ~/HEXAPOD/hexapod_controller
```

- Skompilować program do wersji wykonywalnej przez środowisko ROS2

```
1 colcon build
```

- Upewnić się, że wszystkie źródła są aktualne (do pliku .bashrc zostały dodane wszystkie zależne źródła w trakcie tworzenia obrazu Docker - plik *dockerfile* od linii 45)

```
1 source ~/.bashrc
```

- Uruchomić moduł główny robota

```
1 ros2 launch hexapod_controller hexapod.launch.py
```

- Uruchomić moduł sterowania z klawiatury w nowym terminalu (dodatkowa sesja SSH), dla którego należy również połączyć się do aktywnej sesji Docker

```
1 docker exec -it docker_ros2_1 bash
```

```
1 ros2 run hexapod_controller keyboard_node
```

Po zakończonej pracy można zamknąć aplikację w obu terminalach (każdy osobno) za pomocą kombinacji "Ctrl + C".

Wspomniany moduł sterowania robotem za pomocą klawiatury pozwala na łatwe testowanie funkcjonalności robota oraz manipulację jego położenia w przestrzeni. Zaprojektowany system posiada wiele trybów pracy, które można łączyć dowolnie ze sobą. Jedynym ograniczeniem w kwestii łączenia jest to, że robot może wykonywać tylko jeden ze zdefiniowanych stanów, np. nie można jednocześnie wprowadzić hexapoda w stan obracania lub chodu i wyłączyć moment obrotowy na serwomechanizmach. Taka kombinacja mogłaby spowodować w najgorszym wypadku uszkodzenie robota. Lista zdefiniowanych akcji, które dostępne są użytkownikowi widoczna jest na listingu 4.4.

```
1 , ,
2 This node takes keypresses from the keyboard and publishes them as
   BodyIKCalculate message.
3 -----
4 |           List of keys used to change the values
5 -----
6 |   Translation:          |   Rotation:          |   Movement direction:
7 |     Q   W   E          |     U   I             |     1
8 |     A   S   D          |     J   K             |     2
9 |                         |     M   ,             |     3
10|                         |
11| -----
12|   W / S : X translation |   U / I : X rotation |   1 : move forward
13|   A / D : Y translation |   J / K : Y rotation |   2 : move backward
14|   Q / E : Z translation |   M / , : Z rotation |   3 : move stop
15|                         |
16| -----
17|
18| -----
19|           List of keys used to change state of the robot
20| -----
21|   Control mode / state:
22|     4   8
23|     5   9
24|     6   0
25|     7
26|
27| -----
28|   4 : walk in a given direction (can be changed with 1 and 2)
29|   5 : rotate the robot counter-clockwise
30|   6 : rotate the robot clockwise
31|   7 : enter the body manipulation mode (standing still)
32|   8 : enable the torque on every servo
33|   9 : disable the torque on every servo
34|
35|   0 : idle state, do nothing
36|
37| -----
38|
39 anything else : back to initial position
40 CTRL-C       : go back to initial position and quit
41 , ,
```

Listing 4.4: Wiadomość wyjaśniająca sposób obsługi hexapoda za pomocą klawiatury

4.5 Wyłączanie programu

Po zakończonej pracy można zamknąć aplikację w obu terminalach (każdy osobno) za pomocą kombinacji "Ctrl + C". Po wykonaniu tej operacji możemy wyjść z kontenera,

```
1 exit
```

a następnie wyłączyć aktywny obraz.

```
1 docker stop docker_ros2_1
```

Po wyłączeniu obrazu możemy bezpiecznie wyłączyć Raspberry.

```
1 sudo shutdown now
```

Po zgaśnięciu zielonej diody na obudowie komputera Raspberry Pi, która migra przez chwilę po wyłączeniu, możemy odłączyć przewód zasilający. Dioda ta sygnalizuje aktywny zapis na karcie SD, więc odłączenie przewodu zbyt szybko może skutkować utratą danych.

4.6 Kolejne uruchomienia programu

Gdy raz już zbudujemy kontener i skompilujemy program, to możemy go znacznie szybciej uruchamiać:

- Upewnić się, że konwerter TTL jest podpięty do portu USB i ma adres `/dev/ttyUSB0`,

```
1 dmesg | grep tty
```

jeżeli konwerter ma inny adres, należy zmienić go na odpowiedni w pliku *docker-compose.yml* w sekcji *devices*.

- Przejść do folderu *docker* w sklonowanym wcześniej repozytorium

```
1 cd ~/HEXAPOD/docker
```

- Uruchomić kontener z ustawieniami zawartymi w *docker-compose.yml*

```
1 docker-compose up -d
```

- Uruchomić interaktywną sesję dla działającego kontenera

```
1 docker exec -it docker_ros2_1 bash
```

- Uruchomić moduł główny robota

```
1 ros2 launch hexapod_controller hexapod.launch.py
```

- Uruchomić moduł sterowania z klawiatury w nowym terminalu (dodatkowa sesja SSH), dla którego należy również połączyć się do aktywnej sesji Docker

```
1 docker exec -it docker_ros2_1 bash
```

```
1 ros2 run hexapod_controller keyboard_node
```

4.7 Możliwe problemy

4.7.1 Zmiana konwertera USB2TTL spowodowała, że program przestał działać.

Gdyby wymagana była zmiana portu konwertera TTL lub wydarzyła się ona samostnie (przypisanie innego adresu przez system macierzysty Raspberry Pi), to po jej dokonaniu należy sprawdzić, jaki otrzymał on numer.

```
1 dmesg | grep tty
```

Po ustaleniu numeru należy go zmienić w ustawieniach (w pliku *docker-compose.yml*) i na nowo uruchomić kontener. Tutaj jednak może się okazać, że nasz program nie działa, pomimo uruchomienia kontenera. W takim przypadku należy wykorzystać plik o nazwie "whyDoIhaveToDoThis.txt", znajdujący się w folderze `docker/` i skopiować całą jego zawartość. Skopiowaną treść należy wkleić w oknie terminala (w uruchomionym kontenerze) i wcisnąć przycisk "Enter".

4.7.2 Wywołanie polecenia `colcon build` skutkuje pojawiением się błędów i aplikacja nie działa.

Należy usunąć katalogi instalacyjne

```
1 rm -rf ~/HEXAPOD/hexapod_controller/build ~/HEXAPOD/hexapod_controller/
      install ~/HEXAPOD/hexapod_controller/log
```

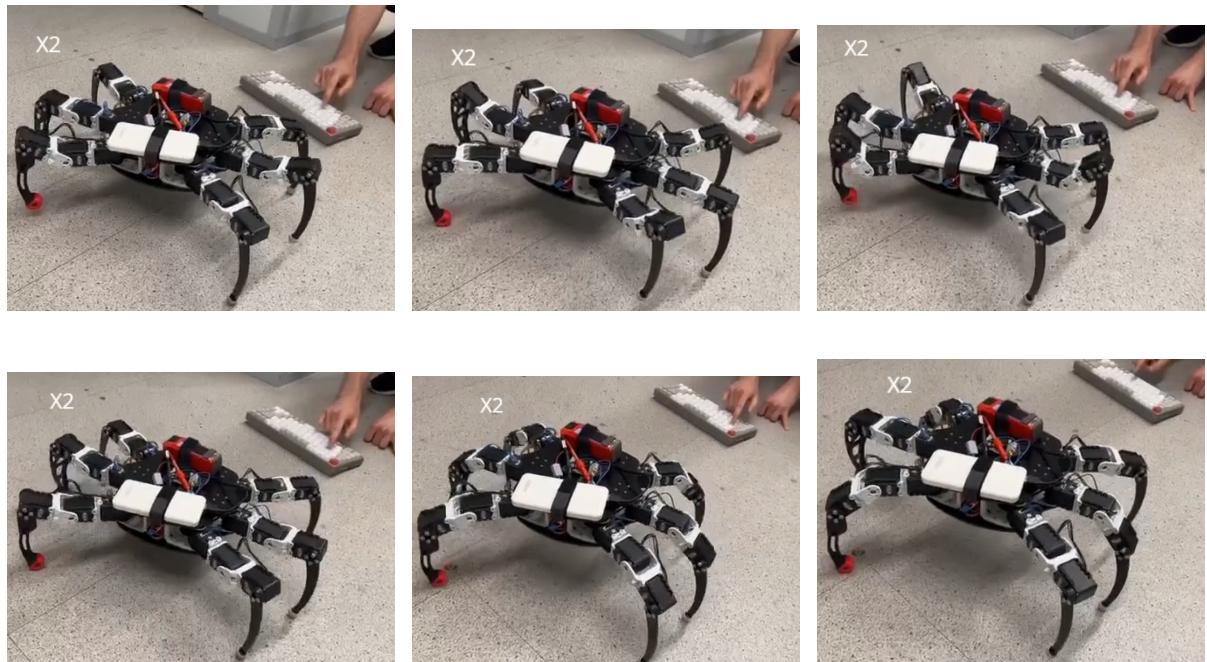
i jeszcze raz wywołać polecenie

```
1 colcon build
```

4.8 Przykładowe działanie robota

4.8.1 Zmiana położenia względem osi Z w trakcie chodu

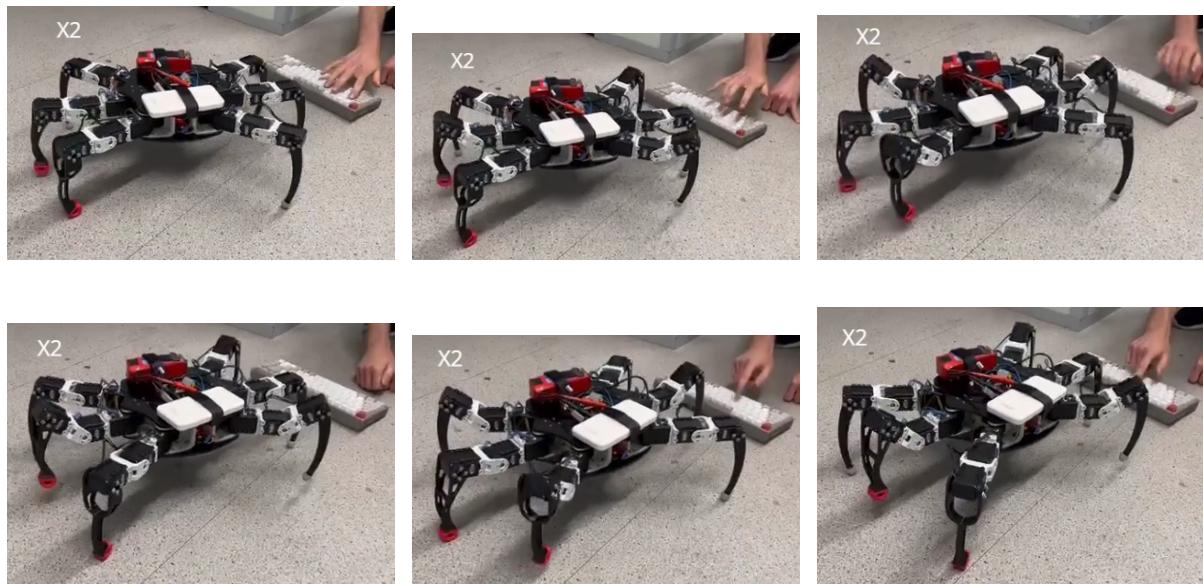
Na rysunku 4.1 można zauważyc zmianę położenia ciała wzdłuż osi Z (translacja) względem powierzchni. Zmiana ta występuje w trakcie chodu i jest kontrolowana za pomocą sygnału z klawiatury, którą można zauważyc z prawej strony każdego ze zdjęć. W pierwszej części prezentacji możemy zauważyc zgięcie odnóży w części femur-tibia (porównując budowę do nogi ludzkiej - "kolano"), co oznacza, że ciało robota jest bliżej ziemi. Dwa ostatnie zdjęcia (w rzędzie dolnym) pokazują robota, który się podnosi, czyli zwiększa dystans pomiędzy swoim ciałem, a podłożem.



Rysunek 4.1: Zmiana położenia ciała w trakcie chodu

4.8.2 Rotacja przeciwnie do ruchu wskazówek zegara

Na rysunku 4.2 możemy zaobserwować w jaki sposób hexapod obraca się wokół osi Z (rotacja) w kierunku przeciwnym do ruchu wskazówek zegara. Podobnie jak w przypadku translacji (sekcja 4.8.1) ruch ten kontrolowany jest przez operatora za pomocą klawiatury, jednak w przypadku tej operacji możliwe są jedynie dwa stany, czyli obracanie lub też brak obrotu (operacja możliwa w obu kierunkach). Nie mamy takiej swobody jak w przypadku translacji, którą można wykonywać ze stałym krokiem w wybranym kierunku (zwiększenie lub zmieszanie odległości od podłoża).



Rysunek 4.2: Rotacja ciała

Rozdział 5

Szczegóły techniczne

W tej pracy wykorzystano kinematykę odwrotną kończyny, jak i ciała, obliczaną za pomocą równań trygonometrycznych zamiast macierzy przejścia. Jest to łatwiejszy, jednak dający taki sam rezultat, sposób wyznaczania docelowego położenia każdej ze stóp oraz ciała w przestrzeni. Przygotowany skrypt wyznaczający te punkty wzorowany jest na dostępnych, wyznaczonych już w wielu pracach, opisach matematycznych i nie był od podstaw wyprowadzony przez autora [5, 7, 8, 15, 24, 25, 26]. Opis matematyczny odnóża robota to opis manipulatora OOO¹. Rozwiązania te pomogły w stworzeniu funkcjonalnego algorytmu, który wyznacza kąty, pod którymi ustawione powinny być silniki. Wyliczane kąty zamienane są na wartości zrozumiałe dla serwomechanizmów. Połączenie kinematyki odwrotnej kończyny z kinematyką ciała sprawiło, że możliwa jest jednoczesna zmiana ustawienia ciała (zarówno translacja, jak i rotacja) w trakcie chodu hexapoda. Kinematyka odwrotna odnóża została również wykorzystana do obliczania trajektorii, w której to poprzez interpolację tworzymy pośrednie punkty wzdłuż kształtu elipsy (bardziej szczegółowy opis znajduje się w sekcji 5.2, pozycja nr 4 na liście).

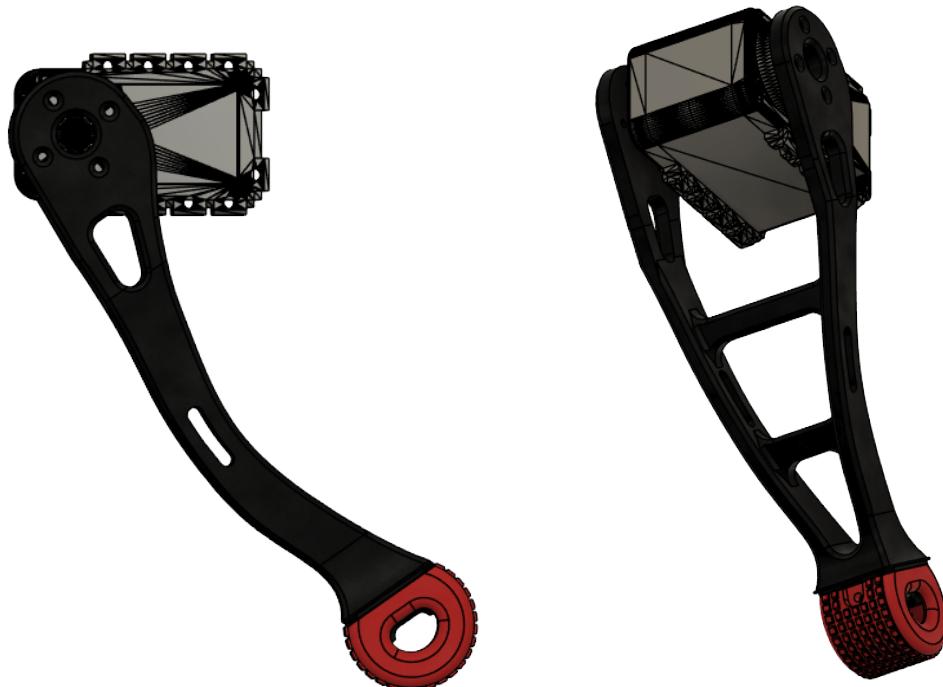
5.1 Synteza układu

W przypadku niniejszego projektu synteza obejmuje złożony proces łączący aspekty mechaniczne, elektroniczne oraz informatyczne. Każdy z wymienionych elementów jest połączony ze sobą, a ich współpraca jest kluczowa dla poprawności działania robota.

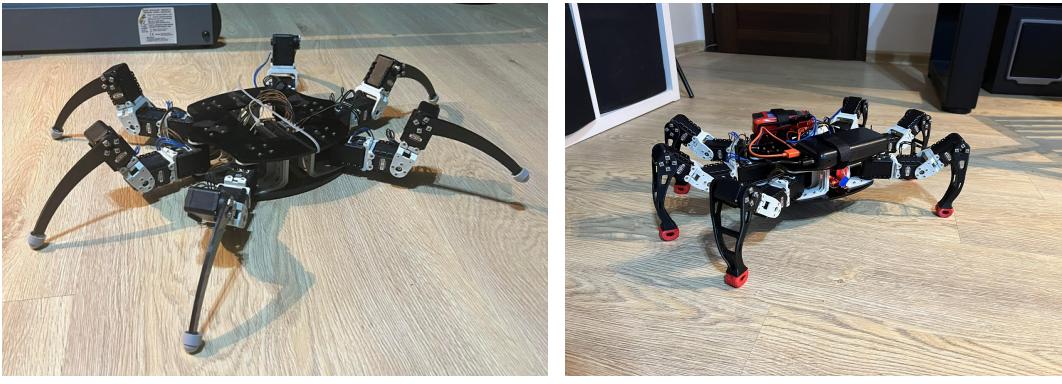
Rozważając aspekty mechaniczne robota, należy zwrócić uwagę na to, że jego odnóża ułożone są wokół ciała w układzie pajaka, o czym wspomniano w sekcji 2.2.2, a każde z sześciu odnóży składa się z trzech silników, co daje ostatecznie osiemnaście serwomechanizmów. Platforma wykorzystywana w tym projekcie była mechanicznie kompletna i nie została stworzona od podstaw przez autora, jednak wprowadzone zostały usprawnienia w kwestii odnóży robota, które zostały całkowicie zmienione. Nowo powstałym projekt miał

¹Manipulator OOO - manipulator o trzech obrotowych parach kinematycznych

na celu usunięcie znaczącej wady konstrukcyjnej, którą było odnóża nieznajdujące się w osi silnika oraz jego końcówka wykonana z tworzywa sztucznego (plastiku). Zastosowanie takiego materiału sprawiło, że nie powstawało wystarczająco duże tarcie pomiędzy stopą a podłożem, aby umożliwić robotowi płynne poruszanie. Te błędy konstrukcyjne mogły powodować chwilową utratę stabilności w trakcie chodu. Problem ten został rozwiązyany poprzez zaprojektowanie własnego modelu i wydrukowanie go za pomocą drukarki 3D. Do druku odnóży wykorzystano sztywny plastik typu PLA (ang. *Polylactic Acid*), natomiast stopy wydrukowano za pomocą filamentu elastycznego TPU (ang. *Thermoplastic Polyurethane*). Nowe stopy są giętkie, a ich odkształcanie się pod wpływem nacisku dodatkowo zwiększa tarcie, a co za tym idzie, także stabilność i szybkość chodu, ponieważ robot nie posuwa się bezwładnie po powierzchni. Prezentacja cyfrowego modelu oraz gotowego, zamontowanego zestawu odnóży znajduje się odpowiednio na rysunku 5.1 oraz 5.2.



Rysunek 5.1: Projekt nowego odnóża robota



Oryginalne odnóża

Usprawniony model

Rysunek 5.2: Graficzne porównanie oryginalnej i nowej konstrukcji odnóży

Przechodząc do aspektów elektronicznych hexapoda, możemy wyróżnić trzy najważniejsze elementy:

- Kontrolery Raspberry Pi 4B oraz Pico W,
- Baterie zasilające komputer i servomechanizmy wraz z płytą drukowaną PCB (ang. *Printed Circuit Board*) do zasilania i połączenia silników w magistralę,
- Konwerter USB2TTL, do transmisji szeregowej UART.

Układ sterowania oparto na mikrokontrolerach firmy Raspberry, co umożliwia synchroniczne sterowanie wszystkimi silnikami. Początkowo do komunikacji pomiędzy robotem a użytkownikiem (stworzenie prywatnej sieci) wykorzystywano układ ESP8266, jednak w trakcie trwania projektu zdecydowano się na jego zmianę na wydajniejszy układ oparty na kontrolerze RP2040, który posiada aż dwa rdzenie oraz 2 MB pamięci Flash, ale co najważniejsze posiada on także moduł Bluetooth. Daje to lepsze perspektywy przyszłego rozwoju i możliwości integracji z innymi systemami.

Częścią platformy wykorzystywanej w tej pracy była także unikalna płyta PCB, która umożliwia podłączenie servomechanizmów w magistralę. Płyta ma wyprowadzenia służące do podłączenia zewnętrznego zasilania do silników, a także komunikacji z nimi. Poprzez drobną modyfikację wyprowadzonych z płytki przewodów, udało się przystosować robota do zasilania z baterii litowo-polimerowych (Li-Po) o napięciu 12 V i pojemności 2400 mAh, na których można było pracować przez kilka godzin przed ich rozładowaniem. Natomiast z racji standardowego napięcia 5 V, wykorzystywanego do zasilania Raspberry Pi, użyto zwykłego banku energii (ang. *powerbank*). W trakcie testów, przy stałym napięciu z zasilacza, zauważono, że układ w trakcie pracy i znaczącym obciążeniu, pobiera maksymalnie 2 do 3 A, więc w najgorszym możliwym wypadku bateria powinna wystarczać na godzinę. Pobór prądu przez Raspberry nie jest znany, jednak według specyfikacji zawartej w dokumentacji producenta, maksymalne skoki natężenia mogą wynosić do 3 A.

Wykorzystywane serwomechanizmy to Dynamixel AX-12A [18] firmy Robotis. W celu zapewnienia z nimi komunikacji użyto konwertera *USB to TTL*, który za pomocą transmisji szeregowej UART Half-Duplex, wysyła zdefiniowane polecenia do silników, a także odbiera od nich informację zwrotną, czy też ich aktualny status w innej, dowolnie wybranej, chwili. Zastosowanie tego konwertera było bardzo istotnym elementem w procesie wymiany danych pomiędzy programem i silnikami, ponieważ umożliwia on wybór poziomu logicznego (napięcia) pomiędzy 3.3 oraz 5 V. Jest to bardzo ważna funkcja, gdyż serwomechanizmy pracują na logice 5 V do wymiany danych [19], a wszystkie dostępne porty szeregowe na Raspberry Pi operują na logice 3.3 V, więc bezpośrednie połączenie tych dwóch interfejsów ze sobą mogłoby spowodować wypalenie uniwersalnych wyjść GPIO (ang. *General Purpose Input Output*) umieszczonych na płytce komputera Raspberry. Innym sposobem na rozwiązywanie tego problemu, mogłoby być zastosowanie konwertera poziomów logicznych i wbudowanych wyjść GPIO, połączonych ze sobą np. na płytce prototypowej, jednak aktualna propozycja dominuje w aspekcie estetycznym, jak i pod względem łatwości diagnozowania problemów i ich naprawy. Dodatkowo konwerter zapewnia kolejny poziom bezpieczeństwa, ponieważ układ ten posiada swoje własne zabezpieczenia przeciwprzepięciowe.

Jedynym problemem, czy też czynnikiem ograniczającym, którego nie rozwiązano w procesie wymiany danych, a jedynie zminimalizowano jego negatywne skutki, jest sam sposób jej realizacji, czyli transmisja w trybie Half-Duplex. Zastosowana modyfikacja, umożliwiająca w ogóle pracę, polegała na połączeniu ze sobą wyprowadzeń Rx (odbieranie danych) i Tx (wysyłanie danych) konwertera za pomocą rezystora $10\text{ k}\Omega$. Rezystor ten jest potrzebny, ponieważ, w chwili odbierania, wyjście Tx powinno być w stanie wysokiej impedancji, żeby nie uległo uszkodzeniu. Takie rozwiązanie jednak może powodować przepelenie magistrali i nachodzenie na siebie pakietów, ponieważ obie linie zajmują się obsługą danych w tym samym momencie. Tutaj innym rozwiązaniem mogłoby być zastosowanie wcześniej wspomnianego konwertera poziomów logicznych oraz cyfrowego bufora pamięci [23], przy bezpośrednim połączeniu do wyjść GPIO na Raspberry Pi. Umożliwiłoby to wysyłanie danych i jednoczesny odbiór informacji zwrotnej, jednak jest to rozwiązanie mniej odporne na błędy i trudniejsze w utrzymaniu. Prowadziłoby to też do większej złożoności samego układu, jak i kodu źródłowego, który musiałby wtedy dodatkowo przełączać bufor pomiędzy odczytem i zapisem. Wymagałoby to dopasowania czasów przełączania, żeby nie zniekształcić którejś z wiadomości (ucinanie bitów, czy też ich nadpisywanie).

Poza konwerterem istotnym czynnikiem jest również sam program, który powstał na podstawie udostępnionej przez firmę Robotis dokumentacji oraz paczki narzędzi DynamixelSDK [20], której kod źródłowy jest w pełni otwarty i dostępny na platformie GitHub [21]. Początkowe testy działania interfejsu wymiany danych przeprowadzono na gotowych przykładach, napisanych w języku Python, które modyfikowano w razie potrzeb. W ten

sposób łatwo można było zrozumieć sposób działania protokołu komunikacyjnego oraz wykorzystywanych przez niego zależności. Po przetestowaniu i potwierdzeniu funkcjonalności wszystkich silników z wykorzystaniem gotowych funkcji zaczęto pisanie własnego kodu na podstawie przedstawionych klas i ich metod.

5.2 Oprogramowanie

Oprogramowanie wykorzystywanego systemu zostało starannie zaprojektowane i na bieżąco modyfikowane w momencie wykrycia nowych błędów. W jego skład wchodzi kilka kluczowych elementów, które zapewniają nie tylko efektywne, ale i efektywne sterowanie ruchem hexapoda. Do najważniejszych modułów można zaliczyć:

1. Główny program

W celu usprawnienia sterowania, stworzono maszynę stanów, która pozwala na łatwe określanie aktualnego zadania robota. Dzięki takiemu rozwiązaniu możliwe jest szybkie i przewidywalne przełączanie pomiędzy zadaniami. Zaimplementowano wykonywanie podstawowych akcji, do których zaliczyć można:

- *idle* - w tym stanie robot zatrzymuje się i oczekuje na następne polecenie operatora,
- *walk* - rozpoczęcie chodu w zadanym kierunku (kierunek domyślny poruszania to "w przód"),
- *rotate_left* - rozpoczęcie obracania ciała robota w kierunku przeciwnym do ruchu wskazówek zegara,
- *rotate_right* - rozpoczęcie obracania ciała robota zgodnie z ruchem wskazówek zegara,
- *torque_enable* - włączenie momentu obrotowego na wszystkich serwomechanizmach robota (część inicjalizacji),
- *torque_disable* - wyłączenie momentu obrotowego na wszystkich serwomechanizmach robota (komenda używana w momencie wyłączania robota),
- *body_manipulation* - zatrzymanie i przełączenie w tryb zmiany pozycji ciała (kontrolowanie translacji i rotacji).

Stworzona maszyna jest elementem nadzorującym pracę robota, ponieważ odpowiada za wydawanie komend i uruchamianie odpowiednich funkcji. Rodzaj rozkazu jest sprawdzany z dużą częstotliwością, co zapewnia niemal natychmiastową reakcję układu. Zastosowanie maszyny stanów pozwoliło na dziewięciokrotne zwiększenie prędkości transmisji danych do silników, w porównaniu do początkowego rozwiązania, co zniwelowało błędy synchronizacji kończyn w trakcie pracy.

2. Moduł inicjalizujący komunikację z silnikami

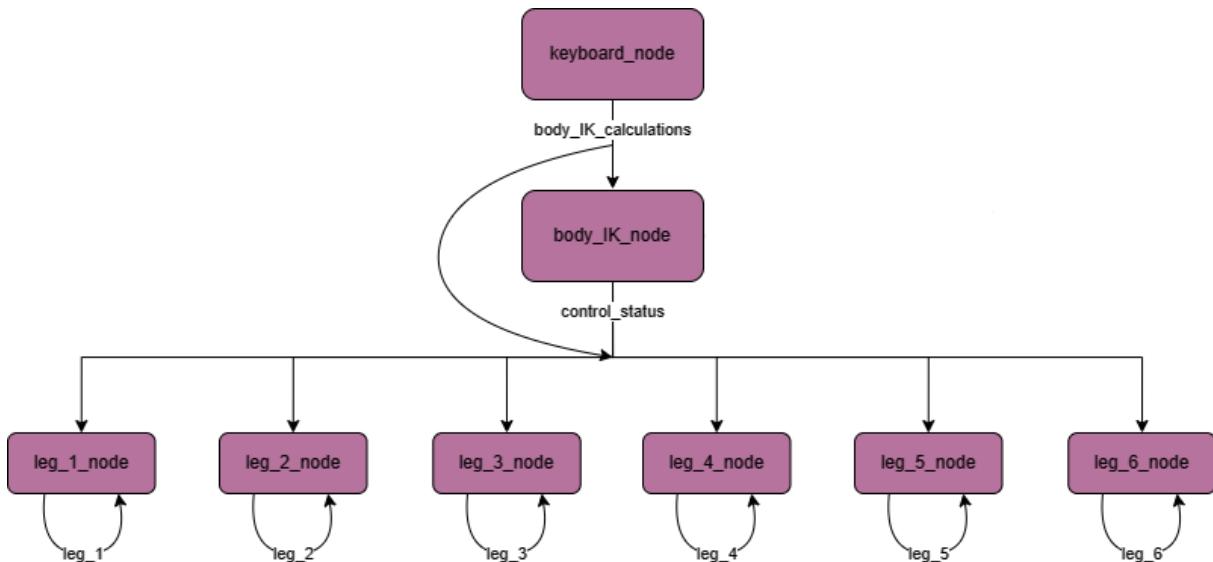
W sekcji 5.1 z większymi szczegółami opisano już wykorzystanie konwertera, który wymagany jest do poprawnego działania tego modułu.

Ten fragment oprogramowania odpowiedzialny jest za odczytanie ustawień, w tym portu, do którego został przydzielony konwerter, oraz ich późniejsze zaaplikowanie w momencie otwierania toru transmisyjnego. W tym pliku wprowadzane są również odpowiednie adresy pamięci silników, które są odpowiedzialne za zapis i odczyt danych, a także szybkość transmisji. Gdy wszystkie ustawienia zostaną odczytane i utworzone zostaną obiekty klasy odpowiedzialne za sterowanie portem i obsługę paczek, otwierany jest kanał komunikacyjny. Jak wspominano wcześniej jest to transmisja szeregową UART, w której dane wysyłane są bit za bitem w jednym kierunku jednocześnie. Cały program wymaga otwarcia tylko i wyłącznie jednego portu, a czynność ta jest częścią procesu inicjalizacji robota.

Wiadomości wysyłane do silników grupowane są w paczki po trzy serwomechanizmy (jedno całe odnóża), co przyczynia się nie tylko do zwiększenia szybkości, ale także niezawodności transmisji, ponieważ wysyłanych jest mniej danych i linia przesyłowa nie ulega tak łatwemu zapchaniu.

3. Moduł pozwalający na sterowanie robotem

Teleoperacja robotem, czyli zdalne sterowanie, możliwa jest za pomocą klawiatury bezprzewodowej, która podłączona została do komputera sterującego. Napisany program umożliwia łatwą i intuicyjną zmianę trybów (stanów) i parametrów, które definiują ciało robota (translacja i rotacja), a także kierunek poruszania się. Skrypt obsługujący klawiaturę powstał na podstawie kodu udostępnionego przez twórców środowiska ROS w ramach jednego ze szkoleń (ang. *tutorial*) [14]. W sekcji 4.4 (listing 4.4) przedstawiono wiadomość, która jest fragmentem kodu odpowiedzialnego za sterowanie i dokładnie opisuje użytkownikowi, jakie akcje może podjąć w trakcie pracy układu. Informacje wprowadzone przez użytkownika przesyłane są za pomocą mechanizmu wiadomości (sekcja 3.3.1, pozycja nr 3 na liście) do modułu głównego, który na ich podstawie przesyła dane dalej. Miejscem, do którego są one wysyłane, jest moduł obliczający kinematykę. Schemat przepływu informacji widoczny jest na rysunku 5.3.



Rysunek 5.3: Wizualizacja przepływu informacji

4. Moduł umożliwiający poruszanie

Moduł ten odpowiedzialny jest za odbieranie danych przesyłanych przez program główny i wyliczenie na ich podstawie docelowych położenie każdego z silników. Po otrzymaniu wartości oczekiwanych parametrów następuje wysłanie informacji o położeniu do serwomechanizmów.

Moduł pozwala na łatwą zmianę wzorca chodu (sekcja 2.2.4), jednakże projekt ten w głównej mierze testowany był przy użyciu wzorca trójpodporowego (ang. *tripod gait*), gdyż jest on najbardziej naturalny, a także zapewnia odpowiedni kompromis pomiędzy szybkością poruszania się i stabilnością platformy. Listing 5.1 obrazuje, jak stworzona architektura kodu pozwala na łatwą zmianę tego wzorca.

```

1 '''VISUAL REPRESENTATION OF GAIT PATTERNS:
2 leg1 === --- === --- === ---
3 leg2 --- === --- === --- ===
4 leg3 === --- === --- === --- ===
5 leg4 --- === --- === --- ===
6 leg5 === --- === --- === --- ===
7 leg6 --- === --- === --- ===
8 row => leg
9 column => indicating whether the leg should be in the air or on the
   ground
10 === => the leg is up,    --- => the leg is down
11 in the matrix this is represented by:
12   1 => the leg is up,      0 => the leg is down'''
13 tripod_gait = [[1,0,1,0,1,0], [0,1,0,1,0,1], [1,0,1,0,1,0],
14                   [0,1,0,1,0,1], [1,0,1,0,1,0], [0,1,0,1,0,1]]
  
```

Listing 5.1: Opis wzorców chodu

Poza określeniem sposobu poruszania, czyli kolejności podnoszenia odnóży, należy również obliczyć, w jaki sposób każda z nich ma się poruszać, zarówno w powietrzu, jak i po ziemi. W tym celu należy wyznaczyć trajektorię, czyli zbiór punktów, w których ma się znaleźć stopa robota. W tym przypadku punkty te wyznaczane są poprzez obliczanie wartości według wzoru na elipsę (ruch w powietrzu) oraz wzoru na prostą (ruch po ziemi), z wykorzystaniem danych oddalonych o stały krok. Oznacza to, że dzielimy elipsę i linię na równą liczbę punktów. Dzięki temu silniki poruszają się płynnie, a układ ma czas na wykonanie ruchu. Bez wyznaczania trajektorii, silniki miałyby jedynie dwa punkty docelowe i robot nie byłby w stanie chodzić, a nawet utrzymywać pozycję stabilną.

Takie rozwiązanie pozwala w łatwy sposób zmieniać zarówno wzorzec chodu, jak i parametry elipsy, oraz ilość punktów w trajektorii.

5.3 Architektura proponowanego rozwiązania

Architektura projektu obejmuje część sprzętową oraz programową, które tworzą spójny system sterowania robotem. Zaproponowana architektura zapewnia efektywną współpracę wszystkich jego elementów, co pozwala na precyzyjną kontrolę ruchów hexapoda.

5.3.1 Aspekty sprzętowe

Do części sprzętowej zaliczyć możemy kontrolery Raspberry Pi oraz platformę z serwomechanizmami. Komputer jednopłytkowy jest główną jednostką obliczeniową, która odpowiedzialna jest za sterowanie całym robotem i integrację wszystkich systemów ze sobą. Mikrokontroler Pico W odpowiedzialny jest tylko i wyłącznie za tworzenie prywatnej sieci umożliwiającej połączenie użytkownika z systemem, jednak daje on wiele możliwości rozwoju w przyszłości. Serwomechanizmy są elementami mechanicznymi, które kontrolują fizyczne ruchy odnóży robota na podstawie danych wysłanych z Raspberry za pomocą konwertera.

5.3.2 Aspekty programowe

Do części programowej zaliczają się podstawowe elementy oraz algorytmy. System operacyjny działa na głównym komputerze i umożliwia zarządzanie zasobami i uruchamianie aplikacji. Wirtualne środowisko ROS2 jest elementem pośredniczącym, który łączy stworzone moduły programu za pomocą wbudowanych mechanizmów komunikacyjnych (wiadomości). Aplikacja sterująca zawiera moduł umożliwiający sterowanie, czyli generowanie

sekwencji ruchu na podstawie otrzymanych rozkazów, a także moduł sterujący, który te rozkazy generuje i zapewnia interaktywność.

5.3.3 Elementy łączące

Stworzone rozwiązanie zawiera dwa elementy, które łączą ze sobą warstwę sprzętową i programową. Są to aplikacja Docker oraz konwerter USB2TTL. Za pomocą kontenera zapewniamy izolację oprogramowania i systemu operacyjnego działającego na Raspberry, a także łączymy fizyczny konwerter z wirtualnym środowiskiem ROS2. Wykorzystanie Docker nie tylko ułatwia replikację programu, ale także zapewnia bezpieczeństwo. Konwerter zapewnia sposób komunikacji pomiędzy aplikacją i silnikami za pomocą protokołu UART.

5.4 Algorytmy

W ramach projektu zaimplementowano dwa istotne algorytmy: chodu i komunikacji (sterowania za pomocą klawiatury).

5.4.1 Algorytm chodu

Jak wspomniano w sekcji 5, stworzony algorytm bazuje na wielu istniejących i udowodnionych już opisach matematycznych, jednak został zmieniony i dostosowany nie tylko do bieżących potrzeb, ale także do łatwej modyfikacji w przyszłości.

Dzięki tej implementacji jesteśmy w stanie poruszać robotem poprzez zadanie mu fizycznego punktu w przestrzeni roboczej opisanej względnym układem współrzędnych. Kod umożliwia nam wyznaczanie kątów, pod którymi powinna być ustaliona każda część odnóża, a także zamienia tę wartość na pozycję zrozumiałą dla silnika, w której ma się znaleźć każdy z nich. Algorytm daje również możliwość zmienianie położenia ciała robota (translacja i rotacja), a także liczenia trajektorii chodu w zależności od wartości parametrów wejściowych. Poprawnie zaimplementowana kinematyka pozwoliła na połączenie tych dwóch przypadków ze sobą, czyli zmianę położenia ciała w trakcie poruszania się, a także daje możliwość rozszerzenia funkcjonalności robota w przyszłości, poprzez zastosowanie korekcji wychylenia ciała np. za pomocą czujnika IMU² (ang. *Inertial Measurement Unit*). Innym wykorzystaniem tej zależności może być zastosowanie czujników nacisku w każdej ze stóp, dzięki czemu robot będzie wiedział, kiedy dane odnóża dotyka ziemi, więc będzie mógł poruszać się po nierównym terenie, czy wchodzić po schodach.

²IMU umożliwia pomiar przyspieszenia oraz orientacji przestrzennej za pomocą 3-osiowego akcelerometru i 3-osiowego żyroskopu w konfigurowalnych zakresach.

Stworzony algorytm, w połączeniu z narzędziami środowiska ROS2, pozwolił na synchronizację silników i zwiększenie płynności poruszania się robota. Wykorzystywane narzędzia, o których mowa, to:

- niestandardowy interfejs (typ wiadomości),
- launchfile (plik uruchamiający wiele węzłów).

Wykorzystanie pierwszego z tych narzędzi, czyli własnego typu wiadomości, było zadaniem łatwym w implementacji. Do tej prostoty przyczyniła się bardzo rozbudowana dokumentacja środowiska ROS2 [12], a także jeden z wielu poradników, który dostępne są na znanym forum robotyki [16]. Używając niestandardowego interfejsu do przesyłu wiadomości pomiędzy węzłami, udało się jednocześnie zmieniać zadane pozycje wszystkich serwomechanizmów. Jednak sama zmiana pozycji nie wystarcza, ponieważ należy tę informację również przesłać. Wysyłanie wiadomości do silników polega na nadawaniu odpowiednich pakietów, w których można zawrzeć wiele informacji, także innych niż pozycja. W tej sytuacji problemem staje się ilość wysyłanych paczek, ponieważ jest ich aż osiemnaście i wszystkie muszą być wysyłane jednocześnie, więc zaczynają na siebie nachodzić lub też są pomijane. Kluczowym elementem pozwalającym rozwiązać problem, okazała się gotowa klasa pakietu DynamixelSDK [20, 21] o nazwie "GroupSyncWrite", która pozwala na połączenie trzech wiadomości w jedną, w formie tablicy bajtów (ang. *byte array*), więc liczba wysyłanych paczek spada do sześciu (z początkowych osiemnastu). Dzięki temu każde z odnóży otrzymuje po jednej paczce i nie występuje nachodzenie się pakietów. Takie rozwiązanie jest jednak nieidealne, ponieważ, w tak opisanej formie, wiadomości wysyłane są jedna za drugą (za pomocą zasobnika pętli `for`). Ich liczba została ograniczona trzykrotnie, jednak opóźnienia pomiędzy kolejnymi odnóżami były widoczne gołym okiem, pomimo krótkich czasów oddzielających wysyłane wiadomości. Rozwiązaniem tego problemu okazało się drugie z wymienionych narzędzi pakietu środowiska ROS2, czyli launchfile [6, 17]. Dzięki stworzeniu pliku *hexapod.launch.py* udało się wywołać wszystkie węzły, czyli "body" i sześć instancji "leg" (od 1 do 6) o numerze odpowiadającym poszczególnym odnóżom robota, jednocześnie. Plik taki uruchamia niezależny proces dla każdego z wykonywalnych węzłów skonfigurowanych w pliku *setup.py*, czyli pliku konfiguracyjnym stworzonej paczki (definicja paczki w sekcji 3.3.1, pozycja nr 3 na liście).

Połączenie tych dwóch narzędzi pozwoliło osiągnąć zadowalające efekty, zarówno pod względem szybkości komunikacji, jak i optymalizacji algorytmu sterowania.

5.4.2 Algorytm komunikacji

Po utworzeniu algorytmu chodu i wprowadzeniu maszyny stanów, możliwe było wdrożenie algorytmu komunikacyjnego, czyli modułu wprowadzającego zdalne sterowanie. Jak wspomniano w sekcji 5.2, pozycja nr 2, stworzenie skryptu obsługującego klawiaturę okazało się zadaniem stosunkowo prostym, a implementacja sterowania na podstawie komend z klawiatury pozwoliła również dokładnie przetestować przytaczane wcześniej połączenie chodu z zachowaniem położenia ciała, czyli połączenie niezależnej kinematyki odwrotnej ciała i odnóża. Tutaj udało się potwierdzić, że połączenie rotacji i zachowania pozycji ciała, niezależnie od kierunku, również działa zgodnie z oczekiwaniami. Dało to również możliwość przełączania robota pomiędzy różnymi stanami np. bezczynności, chodu, czy obracania, w zależności od zadania wybranego przez operatora, wg. instrukcji przedstawionej na wspominanym listingu 4.4. Aby w pełni wykorzystać funkcjonalność tego skryptu, rozszerzono także niestandardowy interfejs wspominany w sekcji 5.4.1, który można zobaczyć na listingu 5.2.

```

1 # COMMUNICATE BETWEEN THE KEYBOARD AND BODY TO CALCULATE THE IK
2 # INITIALIZE ALL OF THE VALUES WITH 0 - STARTING POSITION
3 int16[6] position_of_the_body [0, 0, 0, 0, 0, 0]
4 int16 move_direction 1
5 string robot_state "idle"
6
7 # position_of_the_body (by index):
8 # 0: transX
9 # 1: transY
10 # 2: transZ
11 # 3: rotX
12 # 4: rotY
13 # 5: rotZ
14
15 # move_direction
16 # 1: forward
17 # -1: backward
18 # 0: STOP
19
20 # robot_state
21 # "idle": do nothing
22 # "walk": walking in a specified direction
23 # "rotate_left": rotate the body anti-clockwise
24 # "rotate_right": rotate the body clockwise
25 # "body_manipulation": manipulate the position of the body (ONLY BODY)
26 # "torque_enable": enable the torque on every servo
27 # "torque_disable": disable the torque on every servo
28

```

Listing 5.2: Niestandardowy interfejs (typ wiadomości) sterowania robotem

Rozdział 6

Weryfikacja i walidacja

Proces testowania proponowanego rozwiązania obejmował zarówno testy jednostkowe, jak i integracyjne. W ramach testów jednostkowych sprawdzana była każda z funkcji w module, dla każdego z modułów, i proces ten był nieodłączną częścią tworzenia programu. Weryfikowanie integracji obejmowało sprawdzanie, czy wszystkie ze stworzonych komponentów należycie ze sobą współpracują. Często zdarzało się, że testowanie nowej funkcji wymagało jednoczesnego nadzorowania i usprawniania elementów integrujących, np. modyfikowanie interfejsów (typów wiadomości) ROS.

Przed wprowadzeniem modułu odpowiedzialnego za sterowanie przy pomocy klawiatury, testy algorytmu odbywały się poprzez ręczne zmienianie danych w programie, a także wykorzystując mechanizm wiadomości środowiska ROS i bezpośrednie ich nadawanie przy pomocy terminalu. W kontekście walidacji moduł zdalnej kontroli znacząco ułatwił i usprawnił cały proces twórczy. Wykorzystując klawiaturę, sprawdzano płynność ruchów, a także synchronizację silników. Dzięki wielu testom, wykonywanym na bieżąco, udało się usprawnić wiele funkcji, do których zaliczyć można między innymi:

- Wysyłanie wiadomości do serwomechanizmów

Proces ten został usprawniony poprzez grupowanie wiadomości w paczki. Dzięki temu linia transmisyjna obciążana jest jedynie sześcioma, a nie osiemnastoma, rozkazami.

- Synchronizacja odnóży

Dzięki testom udało się dopasować czas przesyłu wiadomości do poszczególnych odnóży robota i zminimalizować opóźnienia.

- Połączenie kinematyki ciała i kończyny

Wykonywanie dokładnych testów w trakcie tworzenia programu umożliwiło poprawną implementację połączenia tych dwóch algorytmów, dzięki czemu robot jest w stanie nie tylko chodzić, ale także się obracać, przy zachowaniu zadanej pozycji ciała.

Po stworzeniu wszystkich niezbędnych modułów przeprowadzono testy jeszcze raz, aby potwierdzić poprawność ich wykonania i współpracy pomiędzy nimi. Do testowanych przypadków zaliczyć można:

- komunikacja z silnikami poprzez konwerter,
- komunikacja pomiędzy operatorem i robotem,
- chód,
- poprawność wyliczanej trajektorii,
- stabilność w trakcie chodu i po jego wstrzymaniu,
- udźwig robota,
- obrót ciała,
- translacja i rotacja robota w pozycji stacjonarnej (tryb manipulacji).

6.1 Wykryte i poprawione błędy

Podejmując tak trudny temat, którym jest sterowanie mobilnym robotem kroczącym, należy spodziewać się, że wystąpi wiele komplikacji i niedogodności. Nie wszystkie błędy da się całkowicie usunąć, jednak trzeba zrobić wszystko, co się da, aby zniwelować ich negatywne skutki. W trakcie testów należy identyfikować błędy funkcjonalne oraz wydajnościowe, a następnie niezwłocznie podjąć próbę ich usunięcia.

W trakcie tworzenia programu napotkano wiele problemów, które początkowo wydawały się niemożliwe do rozwiązania, jednak wraz z upływem czasu i lepszym zrozumieniem sposobu funkcjonowania wszystkich systemów, większość z nich udała się naprawić. Możemy do nich zaliczyć:

- Stworzenie kontenera Docker

Pierwszym problemem było stworzenie środowiska skonteneryzowanego zawierającego wszystkie wymagane biblioteki. Jego rozwiązaniem okazało się wykorzystanie dokumentacji producentów Docker oraz ROS2, którzy to wspólnym wysiłkiem stworzyli wiele instrukcji szczegółowo opisujących to zagadnienie [11, 13]. Dokumentacja znaczco usprawniła proces przygotowania narzędzi, jednak wciąż należało się zastanowić jakich dodatkowych bibliotek i przyrządów należy użyć w projekcie (np. edytor tekstu "nano", czy też menedżer plików "Midnight Commander"). Ostatecznie postanowiono zainstalować jedynie bibliotekę twórców silników (firmy Robotis) o nazwie "dynamixel_sdk" [21]. W trakcie tworzenia rozwiązania do konfiguracji kontenera dodano skrypty odpowiadające za poprawne funkcjonowanie elementów

środowiska ROS2. Skrypt ten wczytuje funkcje i zmienne środowiskowe w momencie uruchomienia nowej instancji terminala systemowego.

- Architektura kodu

Początkowo architektura aplikacji opierała się na programowaniu funkcjonalnym, jednak wprowadziło to wiele powtarzających się elementów. Aby rozwiązać ten problem, zastosowano ideę programowania obiektowego, czyli stworzono klasę zawierającą te same elementy i różniącą się jedynie identyfikatorem. Sprawiło to, że każde z odnóży dzieli ten sam kod i inicjalizowane jest w ten sam sposób. Dzięki temu wystarczy zmienić jedną funkcję, aby efekt widoczny był dla każdej instancji klasy.

- Przesyłanie wiadomości i synchronizacja silników

Wysyłanie rozkazów do silników okazało się bardzo istotnym problemem, ponieważ istniało wiele czynników wpływających na sposób i jakość komunikacji. Największym problemem okazał się protokół transmisji danych UART half-duplex, którego nie można zmienić. Z tego powodu wymagane było zastosowanie konwertera USB2TTL, oraz łączenie wiadomości w większe paczki. Jak wspominano w sekcji 5.1, problemu nie udało się w pełni rozwiązać, ale zminimalizowano jego wpływ do tego stopnia, że jego efekty są niezauważalne. W rozwiązaniu przydatne było zarówno grupowanie wiadomości, jak i mechanizm *launchfile* opisane w sekcji 5.4.1.

- Sposób obliczania trajektorii

W przypadku obliczania trajektorii, problemem było wyznaczenie odpowiednich parametrów określających elipsę, a także zmiana ruchu w zależności od tego, czy odnóża porusza się w powietrzu, czy po ziemi.

Z pomocą długich testów udało się znaleźć odpowiednie wartości parametrów, dla których długość i kierunek kroku są zadowalające. Implementacja "odpychania", czyli zmiana rodzaju ruchu w zależności od tego, czy odnóża znajduje się w powietrzu, okazała się znacznie łatwiejsza, ponieważ wystarczyło dodać warunek, który tworzy punkty na linii zamiast na kształcie elipsy, gdy odnóża jest na ziemi.

- Kontrola kierunku poruszania

Wprowadzenie zmiennej decydującej o kierunku poruszania (w przód lub w tył) sprawiło, że widoczne stały się błędy w implementacji ruchu. Przykładowo zmiana kierunku na ruch w tył i rozpoczęcie sekwencji obracania zmieniało ruch odnóży w taki sposób, że zamiast się obracać, robot szedł przed siebie. Kolejnym przykładem może być to, że przy zmianie kierunku i jednoczesnej modyfikacji pozycji ciała, niektóre z silników wykonywały ruch przeciwny do oczekiwanej. Rozwiązanie tego problemu okazało się łatwe, jednak bez nieustannych testów jednostkowych

nie udałoby się go wykryć tak szybko i mógłby wystąpić efekt kuli śnieżnej¹.

- Brak tarcia w trakcie chodu

Problem ten został już szerzej opisany w sekcji 5.1 i dotyczy się wadliwej konstrukcji odnóża, którego to końcówka została wykonana z plastiku, który sprawia, że robot nie jest w stanie odepchnąć się od podłoża. Rozwiązaniem okazało się zaprojektowanie własnego modelu i jego wydrukowanie. Projekt nowego odnóża widoczny jest na rysunku 5.1.

- Zasilanie i mobilność

Hexapody należą do grupy robotów mobilnych, więc nie powinny być one ograniczone przez zasilanie sieciowe. Początkowo projekt wykorzystywał regulowany zasilacz napięcia stałego w celu dostarczenia napięcia do serwomechanizmów, jednak wraz z rozwojem projektu, pojawiła się potrzeba zapewnienia źródła zasilania, które jest lekkie i łatwe w transporcie. Stosowany do tej pory zasilacz był bardzo ciężki (10 kg), więc oczywistym rozwiązaniem stało się zastosowanie baterii. Jak wspomniano już w rozdziale 5.1, częścią platformy hexapoda była płytka rozprowadzająca zasilanie do silników. Wykorzystując wiedzę nabycią w trakcie projektu i ogólne doświadczenie z elektroniką, zmodyfikowano układ tak, aby możliwe było nie tylko podłączenie baterii w celu zasilenia robota, ale także ich szybka wymiana. Przygotowane rozwiązanie, pomimo prostoty, okazało się efektywnym sposobem na rozwiązanie problemu i usprawnienie projektu.

¹Proces, który rozpoczyna się na małą skalę i stopniowo rozrasta, stając się większym [2].

Rozdział 7

Podsumowanie i wnioski

Osiągnięte wyniki będą poddane analizie w kontekście założonych celów, co pozwoli ocenić skuteczność i efektywność rozwiązania.

Podczas realizacji projektu udało się osiągnąć wszystkie podstawowe cele i wymagania, jakie przed nim postawiono. Udało się stworzyć funkcjonalny system sterowania, który został przetestowany pod kątem różnych funkcji, do których zalicza się chód, interaktywna komunikacja z użytkownikiem, a także integracja systemu za pomocą platformy Raspberry Pi i otwartych standardów.

Osiągnięto pełną implementację algorytmu chodu i przygotowano strukturę programu w taki sposób, aby możliwe było wprowadzenie kolejnych, bardziej różnorodnych wzorców chodu. Testy układu dla wzorca trójpodporowego potwierdziły poprawność kodu i zachowanie stabilności układu.

W kwestii komunikacji udało się zarówno połączenie programu z serwomechanizmami, jak i połączenie operatora (użytkownika) z programem. Stworzony interfejs bazujący na mechanizmie wiadomości i protokole UART sprawia, że użytkownik może płynnie sterować robotem za pomocą klawiatury.

Wykorzystanie komputera Raspberry Pi 4B jako główna jednostka obliczeniowa okazało się efektywnym i elastycznym rozwiązaniem, dzięki czemu projekt przebiegł pomyślnie. Łatwość integracji z platformą Docker i środowiskiem ROS2 zapewniły zadowalający rezultat w kwestii modułowości.

Przeprowadzone testy oprogramowania potwierdziły funkcjonalność i efektywność stworzonych rozwiązań. Udało się zweryfikować poprawność wszystkich stworzonych modułów i trybów, a wykryte błędy zostały poprawione.

7.1 Rozbudowa funkcjonalna

Platforma ta ma wciąż wielki potencjał i istnieje wiele możliwości rozwoju, które mogą zostać wprowadzone do przygotowanego rozwiązania w przyszłości. Możemy do nich zaliczyć:

1. Stabilizacja ciała poprzez kompensację wychylenia za pomocą czujnika IMU

Jak wspomniano w sekcji 5.4.1, dodanie tej funkcjonalności umożliwi chodzenie robota po nierównym terenie, co może okazać się kluczowe przy badaniach środowiska, ponieważ czujnik ten zapewni równoległe ustawienie względem płaszczyzny podłożu.

2. Sterowanie za pomocą kontrolera

Rozwój tej funkcjonalności opiera się na zamianie sposobu kontroli robota z klawiatury na kontroler bezprzewodowy, np. Xbox. Do wdrożenia takiego rozwiązania należałoby stworzyć opis robota za pomocą narzędzi ROS2 takich jak URDF (ang. *Unified Robotics Description Format*) i paczki "ros2_control".

3. Skanowanie środowiska za pomocą skanera LiDAR

Wykorzystanie skanera LiDAR pomoże w wykrywaniu przeszkód. Będzie to preludium do autonomicznej pracy robota, a także mapowania otoczenia. Dane odczytane przez ten czujnik mogą też posłużyć do uczenia algorytmu planowania ścieżki (ang. *path planning*).

4. Rozpoznawanie obiektów - wizja komputerowa

Implementacja danej funkcjonalności polegać będzie na wykrywaniu i rozpoznawaniu obiektów za pomocą kamery wizyjnej (wizja komputerowa). Pozwoli to na podążanie za obiektem, czy też wykrywanie zagrożenia. Rozwiązanie to może być zaimplementowane razem ze skanerem LiDAR, dzięki czemu robot będzie mógł połączyć obrazy z otoczeniem z informacją o głębi.

5. Uczenie maszynowe

Po wprowadzeniu wizji kolejnym krokiem rozwoju może być wprowadzenie analizy danych otrzymanych z kamery, wykorzystując sieci neuronowe. Dzięki temu robot będzie w stanie wykryć anomalię, zanim się ona będzie miała szansę wydarzyć. Dodatkowo wykorzystując skaner LiDAR, możliwe będzie uczenie robota, w jaki sposób należy omijać przeszkody, w miarę jak będzie się do nich zbliżać.

6. Autonomiczna praca

Zwieńczeniem projektu może stać się połączenie wszystkich pomysłów, zarówno tych zrealizowanych, jak i przyszłych, czyli:

- Chód,
- Stabilizacja ciała,
- Wizja komputerowa,
- Skanowanie otoczenia,
- Uczenie maszynowe.

Koordynacja wszystkich tych funkcjonalności umożliwi wprowadzenie autonomicznej pracy robota. Dzięki temu będzie mógł zostać wykorzystany do różnych zadań:

- Prowadzenie osób niewidomych z punktu A do punktu B,
- Poszukiwanie obiektów lub osób w trudnych warunkach,
- Transport przedmiotów.

7.2 Napotkane problemy

W trakcie pracy pojawiło się wiele wyzwań, które wynikały zarówno z ograniczonych zasobów, jak i błędów technicznych, czy też implementacyjnych. Jak zostało to już opisane w sekcji 6.1, wszystkie błędy udało się rozwiązać, a osiągnięty wynik jest zadowalający. Pomimo negatywnych konotacji, problemy, które wystąpiły w trakcie tworzenia programu, były kluczowym elementem w procesie zrozumienia szczegółów projektowych.

Podsumowując, uzyskane wyniki potwierdzają skuteczność zaproponowanego rozwiązania, jednocześnie pozostawiając możliwość jego przyszłego rozwoju i doskonalenia. Problemy napotkane w trakcie pracy stanowiły cenną lekcję, a ich analiza może przyczynić się do udoskonalenia procesu projektowego w przyszłości.

Bibliografia

- [1] Fraś J. Czarnowski J. „Opracowanie, budowa i oprogramowanie sześcionożnego robota kroczącego”. Praca dyplomowa inżynierska. Warszawa: Politechnika Warszawska, 2013.
- [2] Efekt kuli śnieżnej. Wikipedia : wolna encyklopedia. https://pl.wikipedia.org/wiki/Efekt_kuli_Śnieżnej. [Online; dostęp: 12.12.2023]. 2019.
- [3] Forbot, Zuk. Robaty kroczące – teoria i podstawy projektowania. <https://forbot.pl/blog/robaty-kroczace-teoria-podstawy-projektowania-id976>. [Online; dostęp: 20.11.2023]. 2023.
- [4] Toshio Fukuda, Fei Chen i Qing Shi. „Special feature on bio-inspired robotics”. W: *Applied Sciences* 8.5 (2018), s. 817.
- [5] Grzegorz Gałazka. „Analiza chodu sześcionożnego robota kroczącego”. Praca dyplomowa magisterska. Gliwice: Politechnika Śląska, 2016.
- [6] Hummingbird Robotics. *Hands-On ROS2 - Part 3 (Launch Files)*. <https://youtu.be/dY9aZVMC-JM>. [Online; dostęp: 20.10.2023]. 2022.
- [7] Syamsul Ma’arif, Dyah Ari Susanti i Muchamad Malik. „Implementation of Inverse Kinematics Method for Self-Moving on Hexapod Robot”. W: *ICSET: International Conference on Sustainable Engineering and Technology*. T. 1. 1. 2022, s. 41–51.
- [8] Piątek Marcin. „Problemy sterowania robotami kroczącymi — generatory chodu hexapoda”. Rozprawa doktorska. Kraków: Akademia Górnictwa Hutnicza w Krakowie, 2012.
- [9] mithi. *Mithi’s Hexapod Robot Simulator*. <https://hexapod.netlify.app/walking-gaits>. [Online; dostęp: 14.10.2023]. 2023.
- [10] mithi. *Mithi’s hexapod robot simulator for the web*. <https://github.com/mithi/hexapod>. [Online; dostęp: 14.10.2023]. 2023.
- [11] Muhammad Luqman. *Getting started with ROS and Docker*. <http://wiki.ros.org/docker/Tutorials/Docker>. [Online; dostęp: 04.10.2023]. 2022.
- [12] Open Robotics. *Interfaces*. <https://docs.ros.org/en/rolling/Concepts/Basic/About-Interfaces.html>. [Online; dostęp: 15.10.2023]. 2023.

- [13] Open Robotics. *ROS2 Documentation*. <https://docs.ros.org/en/humble/index.html>. [Online; dostęp: 01.10.2023]. 2022.
- [14] Open Robotics. *ROS2 Tutorials - Turtlesim*. https://github.com/ros/ros_tutorials/blob/humble/turtlesim/tutorials/teleop_turtle_key.cpp. [Online; dostęp: 06.11.2023]. 2022.
- [15] Oscar Liang. *Inverse Kinematics for Hexapod and Quadruped Robots*. <https://oscarliang.com/inverse-kinematics-implementation-hexapod-robots/>. [Online; dostęp: 12.10.2023]. 2012.
- [16] Robotics Back-End. *ROS2 Create Custom Message (Msg/Srv)*. <https://roboticsbackend.com/ros2-create-custom-message/>. [Online; dostęp: 15.10.2023]. 2023.
- [17] Robotics Back-End. *ROS2 Python Launch File Example - How to Start All Your Nodes at Once*. <https://youtu.be/xJ3WAs8GndA>. [Online; dostęp: 20.10.2023]. 2023.
- [18] ROBOTIS. *AX-12A*. <https://emanual.robotis.com/docs/en/dxl/ax/ax-12a/>. [Online; dostęp: 07.10.2023]. 2022.
- [19] ROBOTIS. *AX-12A Communication Circuit*. <https://emanual.robotis.com/docs/en/dxl/ax/ax-12a/#communication-circuit>. [Online; dostęp: 07.10.2023]. 2022.
- [20] ROBOTIS. *DYNAMIXEL SDK*. https://emanual.robotis.com/docs/en/software/dynamixel/dynamixel_sdk/overview/. [Online; dostęp: 07.10.2023]. 2022.
- [21] ROBOTIS-GIT. *Dynamixel SDK*. <https://github.com/ROBOTIS-GIT/DynamixelSDK>. [Online; dostęp: 07.10.2023]. 2022.
- [22] A Roennau, T Kerscher, M Ziegenmeyer, JM Zoellner i R Dillmann. „Six-legged walking in rough terrain based on foot point planning”. W: *Mobile robotics: solutions and challenges*. World Scientific, 2010.
- [23] Texas Instruments. *SN74LS241N*. <https://www.ti.com/product/SN74LS241/part-details/SN74LS241N>. [Online; dostęp: 07.10.2023]. 2023.
- [24] Toglefritz. *Hexapod Inverse Kinematics Equations*. <https://toglefritz.com/hexapod-inverse-kinematics-equations/>. [Online; dostęp: 14.10.2023]. 2016.
- [25] Toglefritz. *Hexapod Inverse Kinematics Simulator*. <https://toglefritz.com/hexapod-inverse-kinematics-simulator/>. [Online; dostęp: 14.10.2023]. 2018.
- [26] Marco Zangrandi, Stefano Arrigoni i Francesco Braghin. „Control of a hexapod robot considering terrain interaction”. W: *arXiv preprint arXiv:2112.10206* (2021).

Dodatki

Spis skrótów i symboli

API interfejs programistyczny do aplikacji (ang. *Application Programming Interface*)

CI/CD ciągła integracja / ciągłe wdrażanie, sposoby organizacji przepływu pracy (ang. *Continuous Integration / Continuous Delivery*)

Docker platforma do konteneryzacji aplikacji

Git system kontroli wersji

GPIO wejścia i wyjścia użytku ogólnego (ang. *General Purpose Input/Output*)

GUI graficzny interfejs użytkownika (ang. *Graphical User Interface*)

HEXAPOD sześcionożny robot mobilny

IMU czujnik pomiaru przyspieszenia (ang. *Inertial Measurement Unit*)

Li-Po ogniwko litowo-polimerowe

OOO manipulator o trzech obrotowych parach kinematycznych

PCB płytka drukowana (ang. *Printed Circuit Board*)

PLA kwas polilaktydowy (ang. *Polylactic Acid*)

RAM pamięć podręczna (ang. *Random Access Memory*)

ROS2 ang. *Robot Operating System 2*

SBC komputer jednopłytkowy (ang. *Single Board Computer*)

SSH protokół komunikacyjny do bezpiecznego zdalnego połączenia (ang. *Secure Shell*)

TPU termoplastyczny poliuretan (ang. *Thermoplastic Polyurethane*)

UART uniwersalny protokół transmisji szeregowej (ang. *Universal Asynchronous Receiver-Transmitter*)

USB port szeregowy (ang. *Universal Serial Bus*)

USB2TTL konwerter portu szeregowego na wyjście o logice tranzystorowej (ang. *USB to TTL*,
Universal Serial Bus to Transistor-Transistor Logic)

Wi-Fi standard sieci bezprzewodowej (ang. *Wireless Fidelity*)

Lista dodatkowych plików, uzupełniających tekst pracy

W systemie do pracy dołączono dodatkowe pliki zawierające:

- źródła programu,
- film demonstrujący działanie opracowanego oprogramowania.

Spis rysunków

2.1	Przedstawienie wielokąta podparcia na podstawie symulacji [9, 10]	8
4.1	Zmiana położenia ciała w trakcie chodu	29
4.2	Rotacja ciała	30
5.1	Projekt nowego odnóża robota	32
5.2	Graficzne porównanie oryginalnej i nowej konstrukcji odnóży	33
5.3	Wizualizacja przepływu informacji	37