POLSKO-JAPOŃSKA AKADEMIA TECHNIK KOMPUTEROWYCH

Wydział Informatyki

Robotyka

Marcin Ziółkowski
Nr albumu s27597

Kulista Maszyna Mobilna

Praca inżynierska
Promotor Maciej Maciejewski

Warszawa, luty, 2026

Spis treści

ROZDZIAŁ 1

Wstęp

1.1 Motywacja i tło projektu

Współczesna robotyka mobilna poszukuje rozwiązań pozwalających na poruszanie się w trudnych, nieustrukturyzowanych środowiskach. Klasyczne roboty kołowe, mimo swojej prostoty, często borykają się z problemem wywrócenia się lub zablokowania na przeszkodach.

Robot kulisty, dzięki swojej specyficznej budowie, oferuje unikalną odporność na tego typu zdarzenia. Obudowa jest jednocześnie ochroną oraz elementem trakcyjnym, co sprawia, że jest on naturalnie chroniony przed czynnikami zewnętrznymi.

Aktualność tematyki robotów kulistych potwierdzają najnowsze wdrożenia w obszarze bezpieczeństwa publicznego. Przykładem pionierskiego rozwiązania jest chiński robot patrolowy RT-G, który na przełomie 2024 i 2025 roku rozpoczął służbę w prowincji Zhejiang. Urządzenie to, podobnie jak projekt opisany w niniejszej pracy, wykorzystuje mechanizm wewnętrznego wahadła do poruszania się, co pozwala mu na osiąganie dużych prędkości oraz sprawne manewrowanie zarówno na lądzie, jak i w środowisku wodnym. Zastosowanie napędu wahadłowego w robotach patrolowych pozwala na całkowitą izolację układów elektronicznych oraz eliminuje ryzyko przewrócenia się maszyny, co czyni go idealnym narzędziem do monitorowania trudnodostępnych przestrzeni miejskich.

1.2 Opis problemu technicznego

Głównym wyzwaniem w projektowaniu robota kulistego jest realizacja napędu przy zachowaniu szczelności i integralności sfery. Zastosowanie mechanizmu z wahadłem pozwala na zmianę położenia środka ciężkości wewnątrz kuli, co indukuje moment obrotowy i wymusza ruch toczny. Sterowanie takim układem jest nietrywialne pod kątem matematycznym, gdyż wymaga precyzyjnego operowania masą wahadła w celu uzyskania pożądanego wektora ruchu.

1.3 Cel i zakres pracy

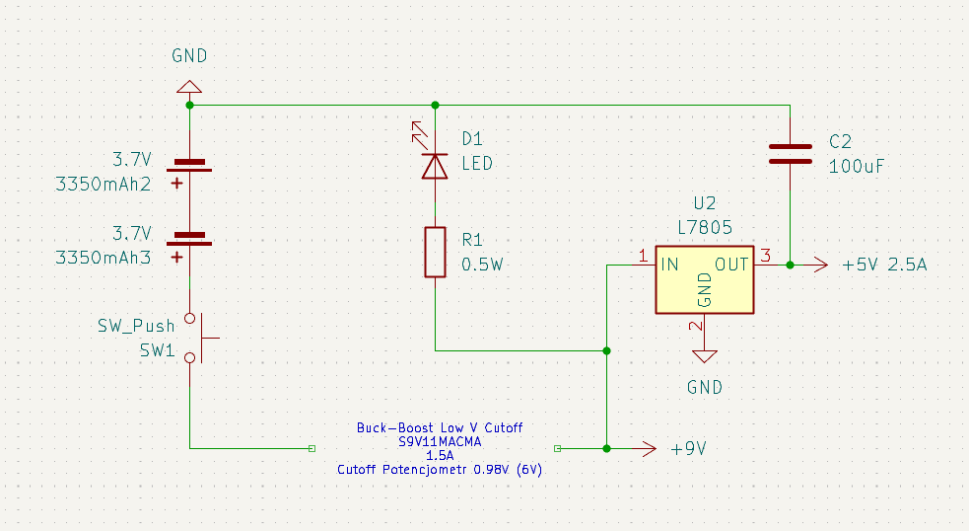
Celem niniejszej pracy jest zaprojektowanie oraz budowa prototypu robota kulistego sterowanego za pomocą wewnętrznego układu wahadłowego, opartego na jednostce Raspberry Pi Zero 2W. Zakres pracy obejmuje:

- Opracowanie architektury sprzętowej wykorzystującej piny PWM do sterowania silnikami prądu stałego.
- Zaprojektowanie mechanicznej konstrukcji robota
- Implementację autorskiego protokołu komunikacyjnego opartego na gniazdach TCP/IP.
- Stworzenie algorytmów kinematyki napędu różnicowego z uwzględnieniem płynnego narastania prędkości w celu ochrony mechanicznych elementów układu.
- Analizę stabilności połączenia i bezpieczeństwa pracy robota poprzez mechanizmy kontroli stanu klienta.

ROZDZIAŁ 2

Układ elektroniczny

2.1 Moduł zasilający



Rysunek 2.1: Moduł zasilający

2.2 Komputer i silniki

Fundamentem technicznym robota jest integracja wydajnej jednostki obliczeniowej z precyzyjnym układem wykonawczym.

2.2.1 Jednostka centralna – Raspberry Pi Zero 2W

Sercem układu sterowania jest minikomputer Raspberry Pi Zero 2W. Wybór tej platformy wynika z jej unikalnych cech:

- **Kompaktowe wymiary:** Niewielka obudowa pozwala na montaż elektroniki bezpośrednio na ramie wahadła, co jest kluczowe dla zachowania wyważenia wewnętrznego mechanizmu.
- **Łączność bezprzewodowa:** Zintegrowany moduł Wi-Fi umożliwia komunikację z komputerem operatora bez potrzeby stosowania zewnętrznych adapterów.
- **Możliwości sprzętowe:** Wielordzeniowy procesor pozwala na jednoczesną obsługę stosu sieciowego TCP/IP oraz generowanie precyzyjnych sygnałów sterujących silnikami.

2.2.2 Układ wykonawczy i mostki H L293D

Do sterowania silnikami prądu stałego wykorzystano zintegrowane mostki H typu **L293D**. Wybór ten umożliwia niezależne sterowanie kierunkiem oraz prędkością obrotową dwóch silników przez jeden układ scalony.

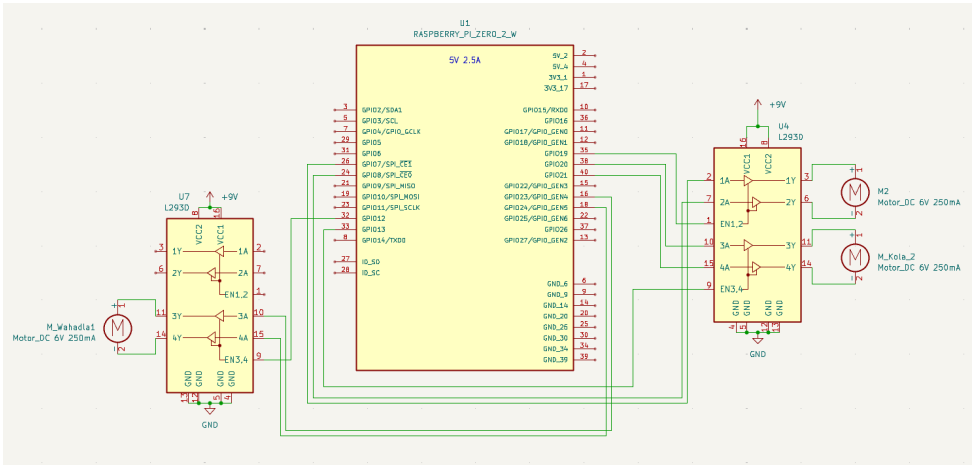
Mostek L293D pracuje w logice cztero-kanałowej i umożliwia:

- **Kontrolę kierunku:** Poprzez podanie stanów wysokich i niskich na wejścia binarne (piny *Input*), co odpowiada konfiguracji pinów *Forward* i *Backward* w projekcie.

- **Regulację prędkości:** Wykorzystanie pinu *Enable* do podania sygnału PWM (Pulse Width Modulation), co pozwala na płynne zarządzanie mocą przekazywaną na silniki.
- **Zabezpieczenie układu:** Wbudowane diody zabezpieczające chronią elektronikę sterującą przed przepięciami indukowanymi przez cewki silników podczas ich zatrzymywania lub zmiany kierunku.

2.2.3 Napęd i zasilanie

Napęd robota opiera się na trzech silnikach DC: dwóch trakcyjnych, realizujących ruch toczny kuli, oraz jednym silniku wahadła. Ten ostatni, poprzez zmianę położenia masy wewnętrznej, pozwala na dynamiczne sterowanie środkiem ciężkości, co jest kluczowe dla manewrowania jednostką sferyczną.



Rysunek 2.2: Komputer i silniki

ROZDZIAŁ 3

CAD

3.1 Projekt mechaniczny i modele CAD

Proces projektowania mechanicznego koncentrował się na zapewnieniu modułowości konstrukcji oraz optymalnym rozmieszczeniu masy wewnątrz sfery. Ze względu na specyfikę napędu wahadłowego, kluczowe było precyzyjne dopasowanie elementów nośnych do geometrii półkul. Poniżej przedstawiono opis głównych komponentów strukturalnych robota.

3.1.1 Moduł centralny i napędowy

Główny szkielet robota stanowią elementy odpowiedzialne za sztywne połączenie elektroniki z układem wykonawczym:

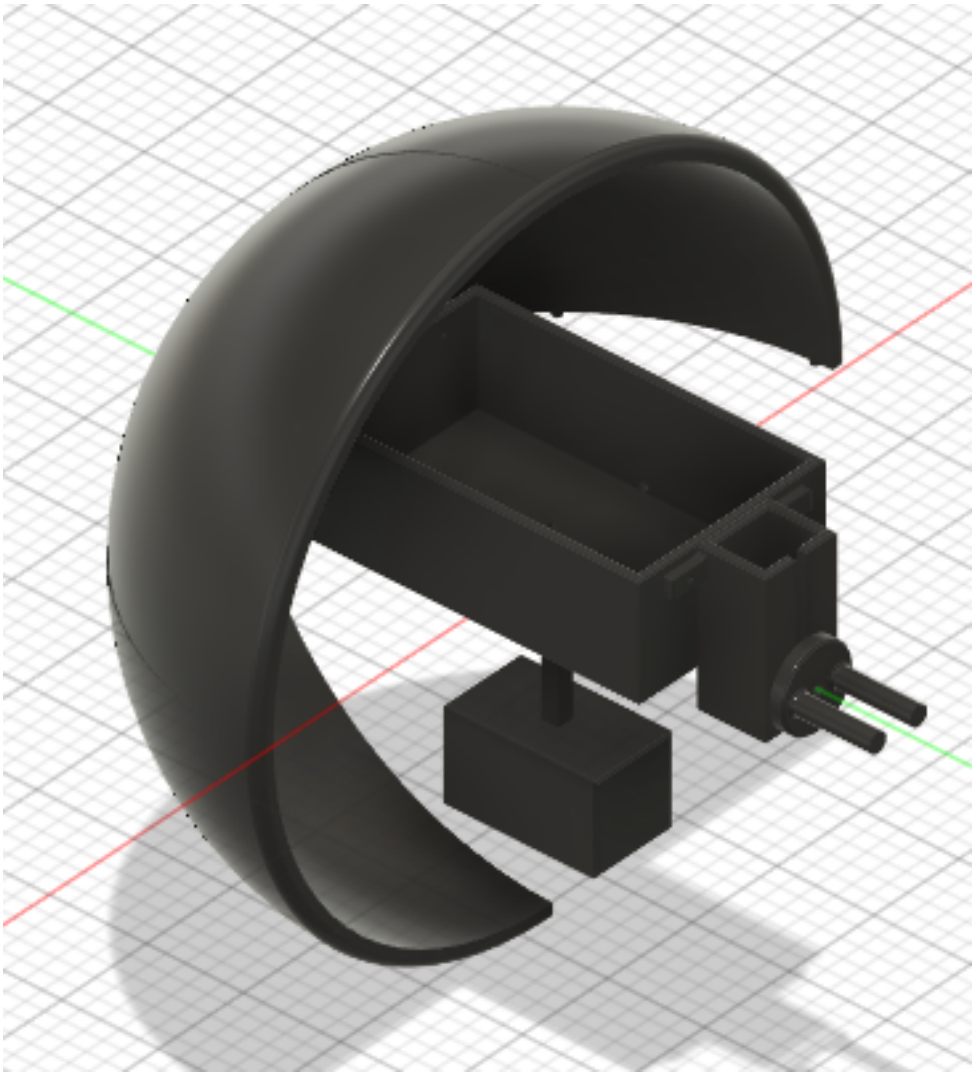
- **Obudowa jednostki centralnej (Box):** Centralny punkt robota, pełniący rolę ramy nośnej. Został zaprojektowany jako kontener przechowujący minikomputer Raspberry Pi oraz źródło zasilania w postaci pakietu baterii.
- **Obudowy silników trakcyjnych (Motor Box):** Dwa symetryczne moduły montowane po bokach jednostki centralnej. Ich zadaniem jest stabilne osadzenie silników DC odpowiedzialnych za wprawianie półkul w ruch obrotowy.
- **Łączniki napędu (Bridge):** Elementy pośredniczące (sprzęgła), które przenoszą moment obrotowy bezpośrednio z wałów silników na wewnętrzną strukturę półkul (Shell).

3.1.2 Układ wahadła i powłoka zewnętrzna

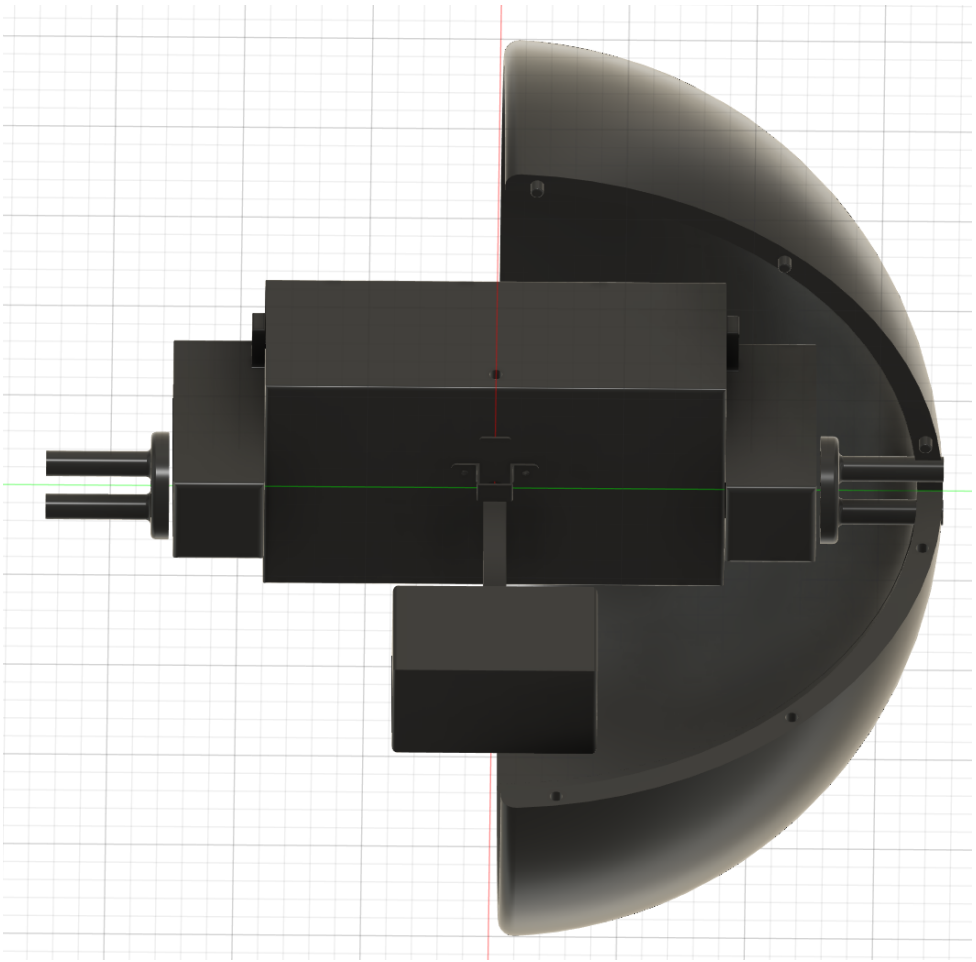
Za manewrowość oraz zewnętrzną formę robota odpowiadają następujące podzespoły:

- **Zaczep wahadła (Pendulum Hook):** Specjalistyczny wspornik montowany do dolnej części modułu *Box*. Stanowi on punkt montażowy dla trzeciego silnika, który operuje masą wahadła w osi poprzecznej robota.
- **Układ wahadła (Pendulum):** Składa się z dwóch integralnych części:
  - *Rękojeść:* Ramię łączące oś silnika z masą roboczą, determinujące promień zataczanego łuku przez wahadło.
  - *Młotek:* Obudowa końcowa z możliwością dociążenia zewnętrznymi ciężarkami, co pozwala na eksperymentalny dobór środka ciężkości układu.
- **Powłoka sferyczna (Shell):** Zewnętrzna obudowa robota o kształcie kuli. Została zaprojektowana jako konstrukcja złożona z czterech pasujących do siebie segmentów, co znacząco ułatwia proces wytwarzania robota.

3.2 Modele CAD



Rysunek 3.1: Model CAD 1



Rysunek 3.2: Model CAD 2

ROZDZIAŁ 4

Oprogramowanie

4.1 Aplikacja sterująca (Controller)

Aplikacja kliencka, uruchamiana na komputerze operatora, pełni rolę interfejsu sterującego, który przekłada interakcje użytkownika na komendy zrozumiałe dla jednostki wykonawczej Raspberry Pi. Architektura oprogramowania została zaprojektowana w sposób wielowątkowy, co zapewnia separację logiki komunikacji sieciowej od procesu monitorowania zdarzeń wejściowych.

4.1.1 Architektura klienta i komunikacja sieciowa

Rdzeniem modułu komunikacyjnego jest klasa `CommandClient`, odpowiedzialna za zarządzanie niskopoziomowym połączeniem TCP/IP. Implementacja wykorzystuje bibliotekę `socket` do nawiązywania strumieniowego połączenia z serwerem o określonym adresie IP i porcie (domyślnie 127.0.0.1:8080). Kluczowe funkcjonalności tego modułu obejmują:

- **Zarządzanie sesją:** Automatyczne próby nawiązania połączenia w pętli głównej do momentu uzyskania statusu `connected`
- **Serializacja danych:** Wykorzystanie modułu `protocol.py` do zamiany obiektów klasy `Command` na sformatowane ciągi znaków (np. `MOVE 1 0`) zakończone znakiem nowej linii, co jest zgodne z przyjętym standardem komunikacyjnym.
- **Mechanizm Heartbeat:** Regularne przesyłanie komendy `PING` w celu weryfikacji aktywności łącza i otrzymywania odpowiedzi `PONG` od serwera.

4.1.2 Przetwarzanie wejścia i mapowanie klawiszy

Interakcja z robotem odbywa się poprzez przechwytywanie zdarzeń klawiatury w czasie rzeczywistym. Za ten proces odpowiada dedykowany wątek `keyboard_listener`, który zapobiega blokowaniu głównej pętli programu przez operacje wejścia/wyjścia.

System sterowania wykorzystuje mapowanie `KEY_MAP`, które przypisuje klawiszom strzałek odpowiednie wartości wektorowe (dx, dy):

- **Ruch liniowy:** Klawisze `up` (1, 0) oraz `down` (-1, 0) generują komendy jazdy odpowiednio do przodu i do tyłu.
- **Ruch obrotowy:** Klawisze `left` (0, -1) oraz `right` (0, 1) odpowiadają za rotację robota wokół własnej osi.
- **Sterowanie złożone:** Logika pętli pozwala na sumowanie wektorów przy jednoczesnym wciśnięciu kilku klawiszy (np. góra i lewo), co umożliwia płynne sterowanie kierunkowe pod kątem 45 stopni.

W celu optymalizacji pasma sieciowego, aplikacja przesyła nową komendę `MOVE` tylko w przypadku zmiany stanu wejścia lub wykrycia ciągłego wciśnięcia klawisza, zachowując przy tym interwał 50 ms pomiędzy kolejnymi aktualizacjami. Gwarantuje to wysoką responsywność układu przy minimalnym obciążeniu procesora jednostki Raspberry Pi Zero 2W.

4.2 Oprogramowanie wykonawcze (Receiver)

Moduł `Receiver` stanowi kluczowy element systemu, rezydujący bezpośrednio na platformie sprzętowej Raspberry Pi Zero 2W. Jego głównym zadaniem jest utrzymywanie stabilnego serwera komunikacyjnego, interpretacja przychodzących ramek danych oraz transformacja abstrakcyjnych komend ruchu na fizyczne sygnały sterujące silnikami robota.

4.2.1 Architektura serwera i obsługa połączeń

Za warstwę komunikacyjną odpowiada klasa `CommandServer`, która inicjalizuje gniazdo sieciowe TCP/IP i nasłuchuje na zadanym porcie (domyślnie 8080). System został zaprojektowany z myślą o obsłudze jednego klienta sterującego w danym momencie, co zapewnia wyłączność nad kontrolą urządzenia.

W procesie obsługi klienta zaimplementowano mechanizmy zwiększające niezawodność:

- **Analiza strumienia danych:** Dane odbierane przez gniazdo są dekodowane i dzielone na poszczególne linie, co pozwala na poprawne przetwarzanie komend nawet przy spiętrzeniu pakietów w buforze sieciowym.
- **Zarządzanie stanem połączenia:** Klasa `ClientState` monitoruje aktywność klienta poprzez rejestrowanie czasu nadejścia ostatniej poprawnej ramki danych.
- **System bezpieczeństwa (Failsafe):** W przypadku przekroczenia zdefiniowanego limitu czasu oczekiwania (`SOCKET_TIMEOUT`), serwer automatycznie wywołuje procedurę zatrzymania silników i resetuje stan sterownika, co zapobiega niekontrolowanemu ruchowi robota po utracie łączności.

4.2.2 Logika sterowania i warstwa sprzętowa

Otrzymane komendy tekstowe są parsowane przez moduł `protocol.py` na obiekty klasy `Command`, które trafiają do głównego kontrolera robota.

Proces generowania ruchu przebiega w dwóch etapach:

1. **Obliczenia kinematyczne:** Klasa `DiffDriveKinematics` przelicza wektor ruchu (v, ω) na docelowe prędkości lewego i prawego koła oraz prędkość wahadła (`pendulum_speed`). Algorytm uwzględnia specyfikę robota kulistego, gdzie ruch obrotowy przy jeździe liniowej jest wspomagany wychyleniem wewnętrznego wahadła.
2. **Ramping prędkości:** Zastosowano metodę `_approach`, która w każdym cyklu pracy zbliża aktualną prędkość silników do wartości docelowej o stały krok określony przez `ACCELERATION_PERCENTAGE` (0.1). Pozwala to na uniknięcie gwałtownych szarpnięć, które mogłyby destabilizować ruch kuli lub uszkodzić mechanizmy wewnętrzne.

Bezpośrednią kontrolę nad sprzętem sprawuje moduł `RobotDriver`, wykorzystujący bibliotekę `gpiozero`. Klasa `Motor` abstrahuje sterowanie pojedynczym silnikiem, zarządzając pinami kierunkowymi oraz wypełnieniem sygnału PWM (Pulse Width Modulation) na dedykowanych pinach zdefiniowanych w konfiguracji systemu.

ROZDZIAŁ 5

Inicjalizacja

5.1 Inicjalizacja i procedura startowa systemu

Prawidłowe funkcjonowanie robota sferycznego wymaga skoordynowanej inicjalizacji wszystkich warstw systemowych – od niskopoziomowej konfiguracji pinów GPIO, przez logikę sterowania, aż po warstwę komunikacji sieciowej. Proces ten został zaprojektowany tak, aby zapewnić maksymalne bezpieczeństwo sprzętu już od momentu podania zasilania.

5.1.1 Inicjalizacja warstwy sprzętowej (Hardware Layer)

Pierwszym etapem po uruchomieniu skryptu głównego na jednostce Raspberry Pi jest inicjalizacja obiektu klasy `RobotDriver`. W tym kroku system wykonuje następujące operacje:

- **Konfiguracja pinów:** Na podstawie pliku `robotSettings.py` przypisywane są funkcje dla poszczególnych linii GPIO. Wykorzystanie biblioteki `gpiozero` pozwala na automatyczne ustawienie pinów w bezpiecznych stanach niskich, co zapobiega niekontrolowanym szarpnięciom silników przy starcie.
- **Instancjonowanie napędów:** Tworzone są trzy obiekty klasy `Motor` (lewy, prawy oraz wahadło). Dla każdego z nich inicjalizowane są dwa wyjścia cyfrowe (kierunek) oraz jedno wyjście z modulacją PWM (prędkość).
- **Reset kinematyki:** Obiekt `DiffDriveKinematics` zeruje wszystkie wewnętrzne bufory prędkości (`curr_left`, `curr_right`, `curr_pend`), przygotowując algorytm rampingu do płynnego startu od wartości zero.

5.1.2 Uruchomienie jednostki sterującej i serwera

Po pomyślnym przygotowaniu sterowników, system przechodzi do inicjalizacji logiki wysokopoziomowej w module `Controller`. Kontroler ten staje się pośrednikiem między siecią a sprzętem.

Ostatnim ogniwem procedury startowej jest uruchomienie `CommandServer`. Serwer otwiera gniazdo TCP/IP na porcie 8080 i przechodzi w stan nasłuchiwania (*listening*). W tym momencie robot jest widoczny w sieci lokalnej i oczekuje na pakiet inicjalizujący od aplikacji klienckiej (komputer operatora).

5.1.3 Ustanowienie połączenia i autoryzacja

Pełna operacyjność systemu następuje po wykonaniu tzw. "uścisku dłoni" (ang. *handshake*) z kontrolerem zewnętrznym:

1. Klient nawiązuje połączenie TCP.
2. Serwer inicjalizuje obiekt `ClientState`, ustawiając znacznik czasu `last_seen` na wartość aktualną.
3. Pierwsza odebrana komenda `PING` potwierdza drożność kanału komunikacyjnego, na co serwer odpowiada komunikatem `PONG`, sygnalizując gotowość do przyjmowania komend `MOVE`.

Wprowadzenie tej wieloetapowej inicjalizacji pozwala na uniknięcie błędów wynikających z prób sterowania niegotowym sprzętem oraz gwarantuje, że robot zareaguje na polecenia tylko wtedy, gdy wszystkie podsystemy pracują poprawnie.

ROZDZIAŁ 6

Podsumowanie

6.1 Zakończenie i wnioski

Niniejsza praca inżynierska poświęcona była procesowi projektowania, budowy oraz oprogramowania prototypu robota sferycznego z wewnętrznym układem wahadłowym. Realizacja projektu pozwoliła na zweryfikowanie założeń dotyczących mobilności jednostek kulistych oraz skuteczności sterowania opartego na systemie Linux w zastosowaniach czasu rzeczywistego.

6.1.1 Wnioski techniczne

Na podstawie przeprowadzonych prac oraz testów prototypu sformułowano następujące wnioski:

- **Skuteczność napędu wahadłowego:** Mechanizm zmiany środka ciężkości za pomocą wewnętrznego wahadła okazał się efektywnym sposobem na wymuszenie skrętu robota. Połączenie napędu różnicowego półkul z ruchem wahadła pozwala na uzyskanie dużej zwrotności, co jest kluczowe w ciasnych przestrzeniach.
- **Kluczowa rola rampingu:** Implementacja algorytmu narastania prędkości (*acceleration_percentage* = 0.1) w module `kinematics.py` znacząco wpłynęła na stabilność kuli. Bez płynnego przyrostu mocy, gwałtowne ruchy wahadła powodowały oscylacje kuli, utrudniając precyzyjne sterowanie.
- **Niezawodność TCP/IP:** Wykorzystanie gniazd sieciowych w architekturze klient-serwer zapewniło stabilną komunikację. Mechanizm *watchdog* (klasa `ClientState`) skutecznie chroni robota przed niekontrolowanym ruchem w przypadku utraty zasięgu sieci Wi-Fi.
- **Dobór komponentów:** Jednostka Raspberry Pi Zero 2W w połączeniu z mostkami L293D stanowi optymalny balans pomiędzy wydajnością obliczeniową a poborem energii i gabarytami, co jest krytyczne wewnątrz zamkniętej sfery.

6.1.2 Kierunki dalszego rozwoju

Mimo osiągnięcia założonych celów, projekt posiada potencjał do dalszej rozbudowy w następujących obszarach:

- **Autonomizacja:** Dodanie czujnika IMU (akcelerometru i żyroskopu) pozwoliłoby na implementację algorytmu aktywnej stabilizacji pionowej wahadła oraz automatyczne korygowanie toru jazdy.
- **System wizyjny:** Wykorzystanie dedykowanej kamery Raspberry Pi do strumieniowania obrazu w czasie rzeczywistym, co umożliwiłoby sterowanie robotem poza zasięgiem wzroku operatora.
- **Optymalizacja Shell:** Badania nad nowymi materiałami dla powłoki zewnętrznej (*Shell*), które zwiększyłyby przyczepność na gładkich powierzchniach oraz poprawiłyby odporność na uderzenia.
- **Integracja z systemem ROS:** Przeniesienie logiki sterowania do środowiska *Robot Operating System*, co ułatwiłoby współpracę z innymi systemami robotycznymi.

Podsumowanie końcowe

Projekt robota kulistego udowodnił, że koncepcja stosowana w najnowocześniejszych jednostkach patrolowych (takich jak chiński RT-G) może zostać z sukcesem zaimplementowana w warunkach laboratoryjnych przy użyciu druku 3D oraz popularnych mikrokontrolerów. Opracowany system stanowi solidną bazę do dalszych badań nad dynamicznie niestabilnymi układami mechanicznymi.