
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 [1], 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

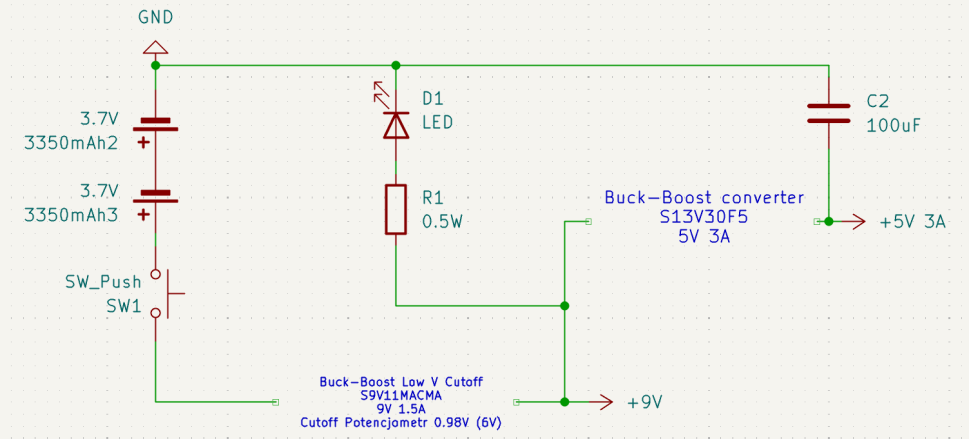
Celem niniejszej pracy jest zaprojektowanie oraz budowa prototypu robota kulistego sterowanego za pomocą wewnętrznego układu wahadłowego, opartego na jednostce Raspberry Pi Zero 2W. Zakres pracy obejmuje:

- 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

System zasilania robota kulistego został zaprojektowany z uwzględnieniem dwóch priorytetów: zapewnienia stabilnego napięcia dla jednostki logicznej (Raspberry Pi Zero 2W) oraz dostarczenia odpowiedniej mocy dla trzech silników prądu stałego. Ze względu na dużą bezwładność mechaniczną konstrukcji, moduł musi być odporny na nagłe skoki poboru prądu (tzw. inrush current) wywoływane przez silniki.

### 2.1.1 Architektura systemu i separacja napięć

Źródło zasilania stanowi pakiet ogniw litowo-jonowych (Li-ion). Ze względu na charakterystykę rozładowania tych ogniw oraz ryzyko ich trwałego uszkodzenia przy zbyt niskim napięciu, w systemie zastosowano dwustopniową stabilizację opartą na zaawansowanych regulatorach typu step-up/step-down (buck-boost).

### 2.1.2 Regulator logiki z funkcją ochrony ogniw – S9V11MACMA

Kluczowym elementem odpowiedzialnym za bezpieczeństwo całego systemu jest regulator Pololu S9V11MACMA [2]. Jest to układ typu step-up/step-down, który dostarcza stabilne napięcie wyjściowe niezależnie od tego, czy napięcie wejściowe jest wyższe, czy niższe od zadanego poziomu.

W niniejszym projekcie wykorzystano dwie unikalne cechy tego modułu:

- **Regulacja napięcia wyjściowego:** Za pomocą wbudowanego potencjometru wieloobrotowego napięcie wyjściowe ustawiono na poziomie 5.1V, co jest wartością optymalną dla Raspberry Pi, niwelującą spadki napięcia na przewodach zasilających.
- **Mechanizm Low-Voltage Cutoff:** Drugi potencjometr na module pozwala na precyzyjne ustawienie progu odcięcia zasilania (*cutoff*). W przypadku spadku napięcia na akumulatorze poniżej bezpiecznej granicy (ustalonej na 3.0V na ogniwo), regulator przechodzi w stan uśpienia, odcinając zasilanie od komputera i chroniąc pakiet Li-ion przed głębokim rozładowaniem.

### 2.1.3 Zasilanie układu wykonawczego – S13V30F5

Do zasilania mostków H (L293D) oraz silników wykorzystano wysokowydajny regulator Pololu S13V30F5 [3]. Wybór tego modułu był podyktowany koniecznością utrzymania stałej dynamiki ruchu robota, niezależnie od stopnia naładowania baterii.

Regulator ten charakteryzuje się:

- **Stałe napięcie 5V:** Zapewnia powtarzalne osiągi silników. Dzięki architekturze step-up/step-down, nawet gdy napięcie pakietu spadnie do 2.8V, silniki wciąż otrzymują pełne 5V, co zapobiega "pływaniu" parametrów sterowania pod koniec cyklu pracy baterii.
- **Wysoka wydajność prądowa:** Układ oferuje ciągły prąd wyjściowy na poziomie do 3A, co pozwala na jednoczesną pracę dwóch silników napędowych i silnika wahadła w momentach największego obciążenia (start, zmiana kierunku).

### 2.1.4 Analiza spadków napięcia i bezpieczeństwo operacyjne

W trakcie testów laboratoryjnych odnotowano, że bez odpowiedniej separacji, nagły rozruch silników powodował chwilowe spadki napięcia (voltage sags), które skutkowały restartem Raspberry Pi. Zastosowanie dwóch niezależnych regulatorów Pololu pozwoliło na stworzenie bufora bezpieczeństwa. Nawet jeśli regulator silników (S13V30F5) chwilowo obciąży baterię, regulator logiki (S9V11MACMA) dzięki szerokiemu zakresowi napięć wejściowych (już od 2V) jest w stanie utrzymać stabilną pracę procesora, co ma krytyczne znaczenie dla bezawaryjnej pracy systemu operacyjnego Linux.

## 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 [4]. 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

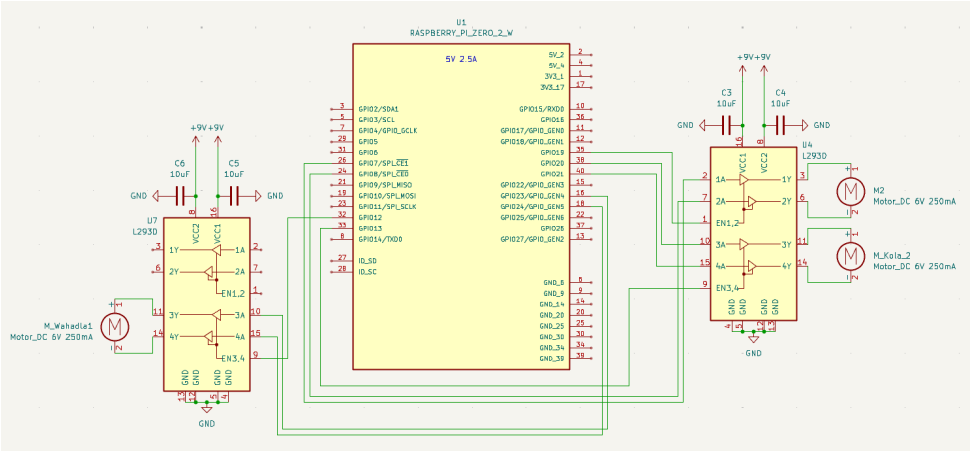
Do sterowania silnikami prądu stałego wykorzystano zintegrowane mostki H typu **L293D** [5]. Wybór ten umożliwia niezależne sterowanie kierunkiem oraz prędkością obrotową dwóch silników przez jeden układ scalony.

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

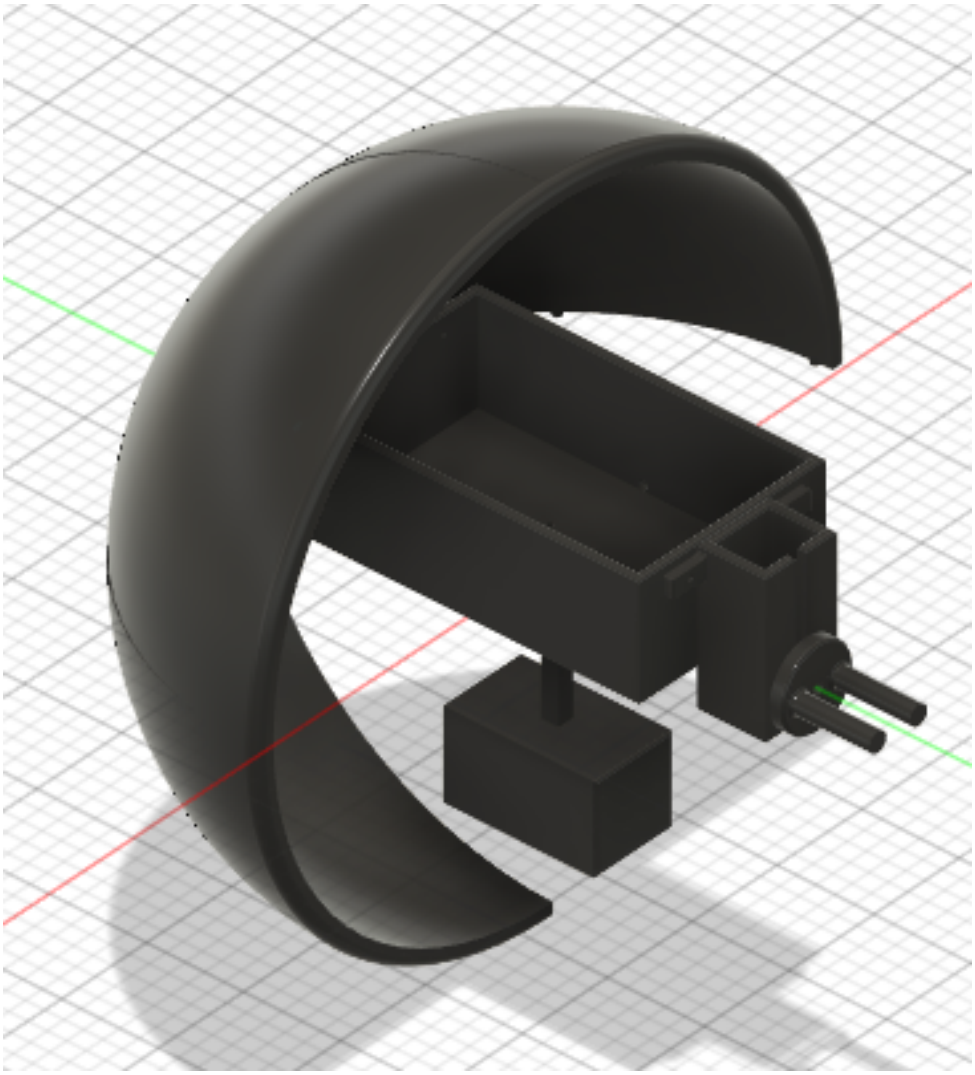
# CAD

## 3.1 Projekt mechaniczny i modele CAD

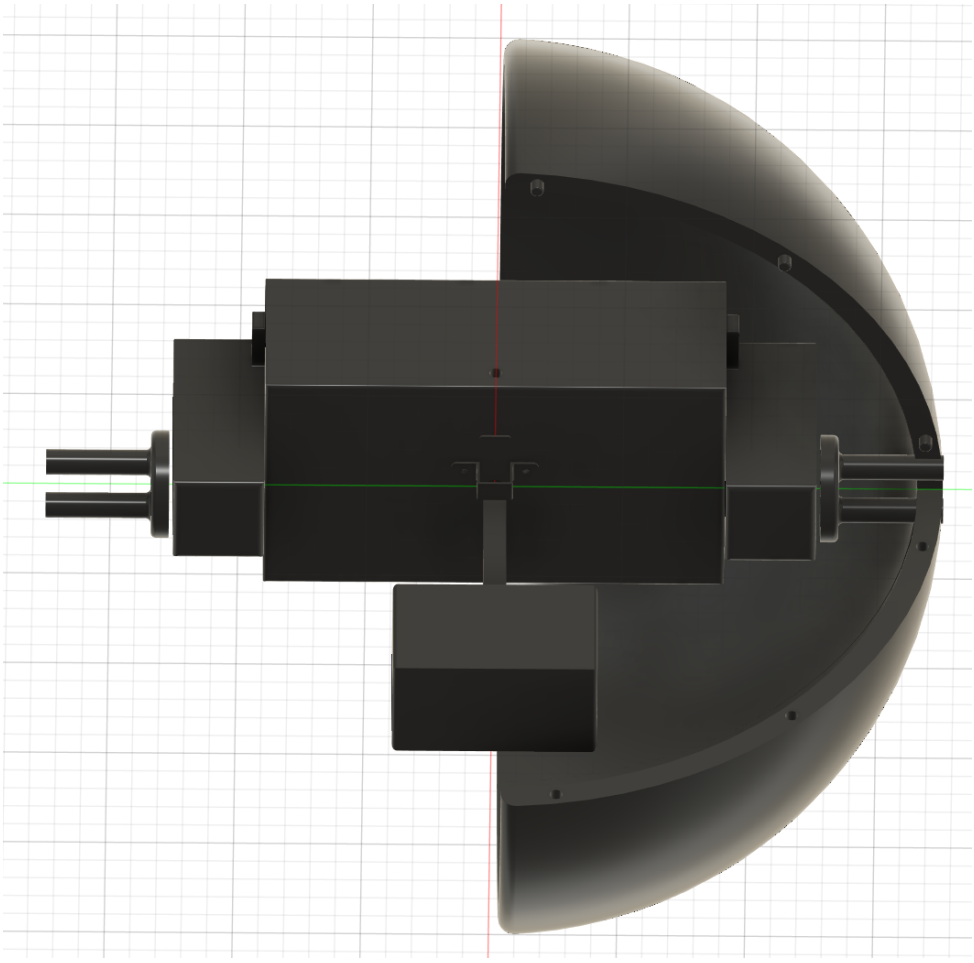
Proces projektowania mechanicznego koncentrował się na zapewnieniu modułowości konstrukcji oraz optymalnym rozmieszczeniu masy wewnątrz sfery. Ze względu na specyfikę napędu wahadłowego, kluczowe było precyzyjne dopasowanie elementów nośnych do geometrii półkul. Poniżej przedstawiono opis głównych komponentów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

### 3.1.1 Moduł centralny i napędowy

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

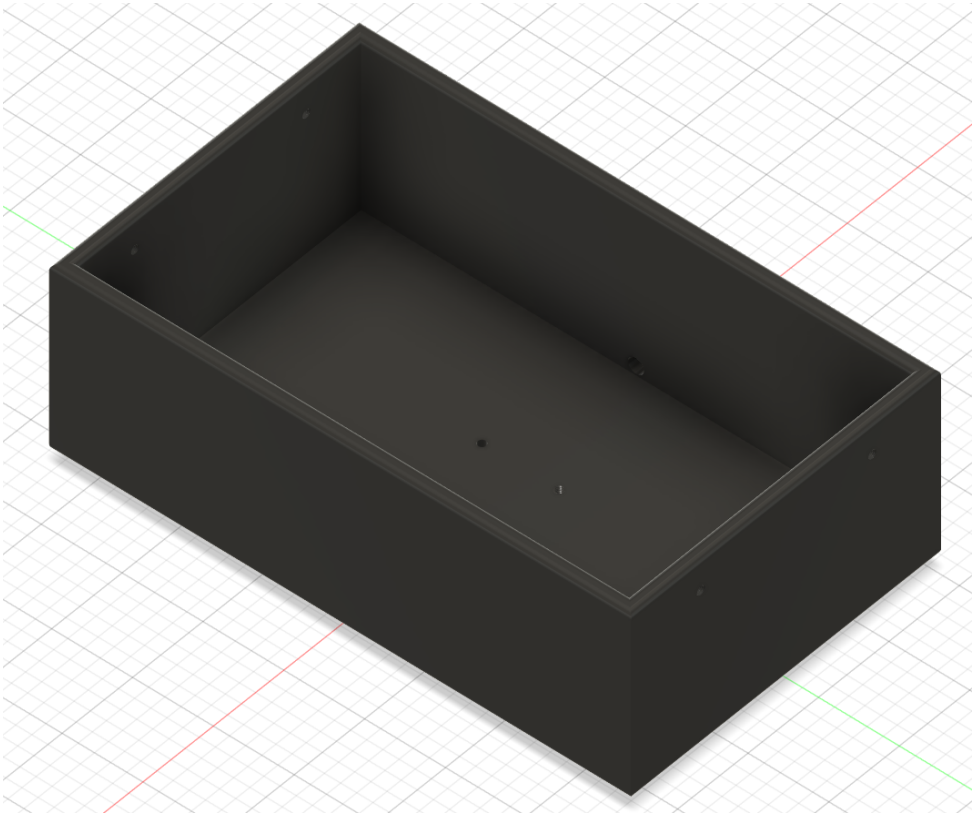


Rysunek 3.1: Model CAD 1



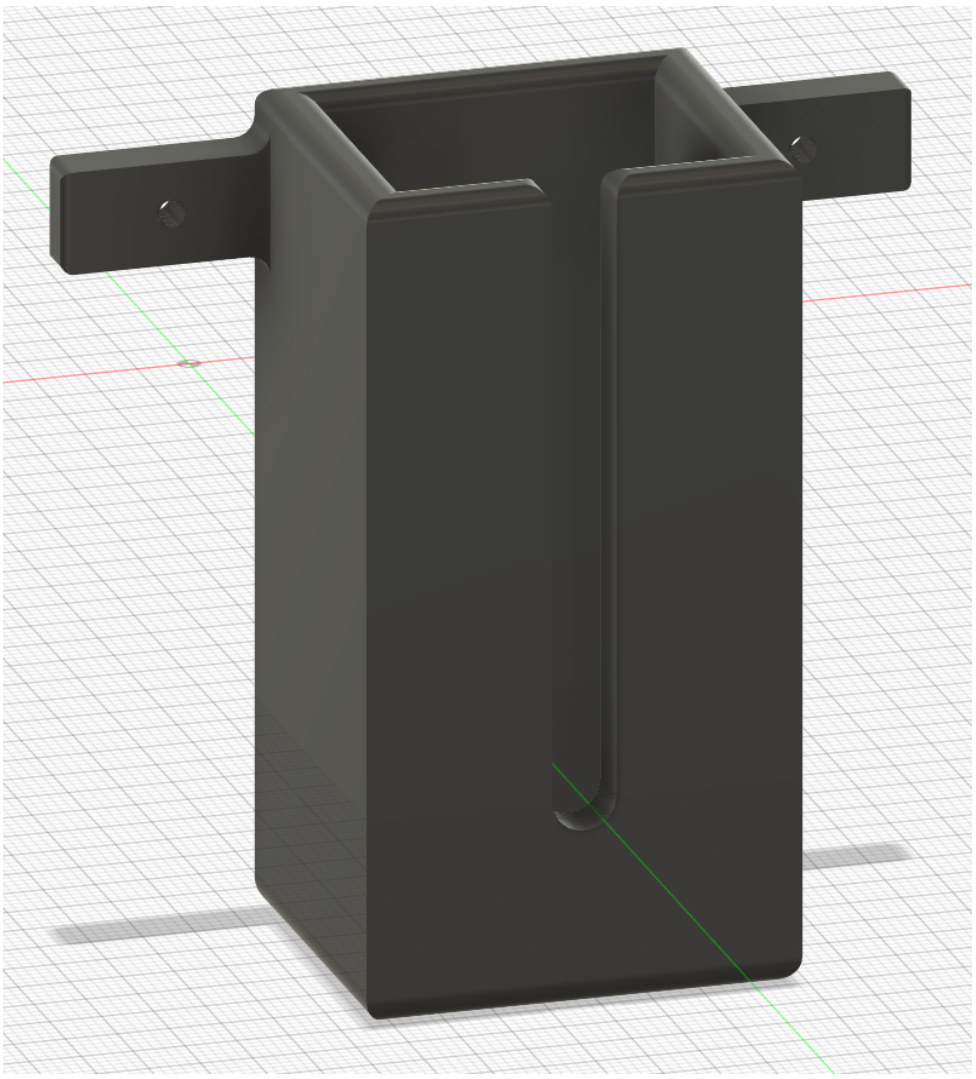
Rysunek 3.2: Model CAD 2

- **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.



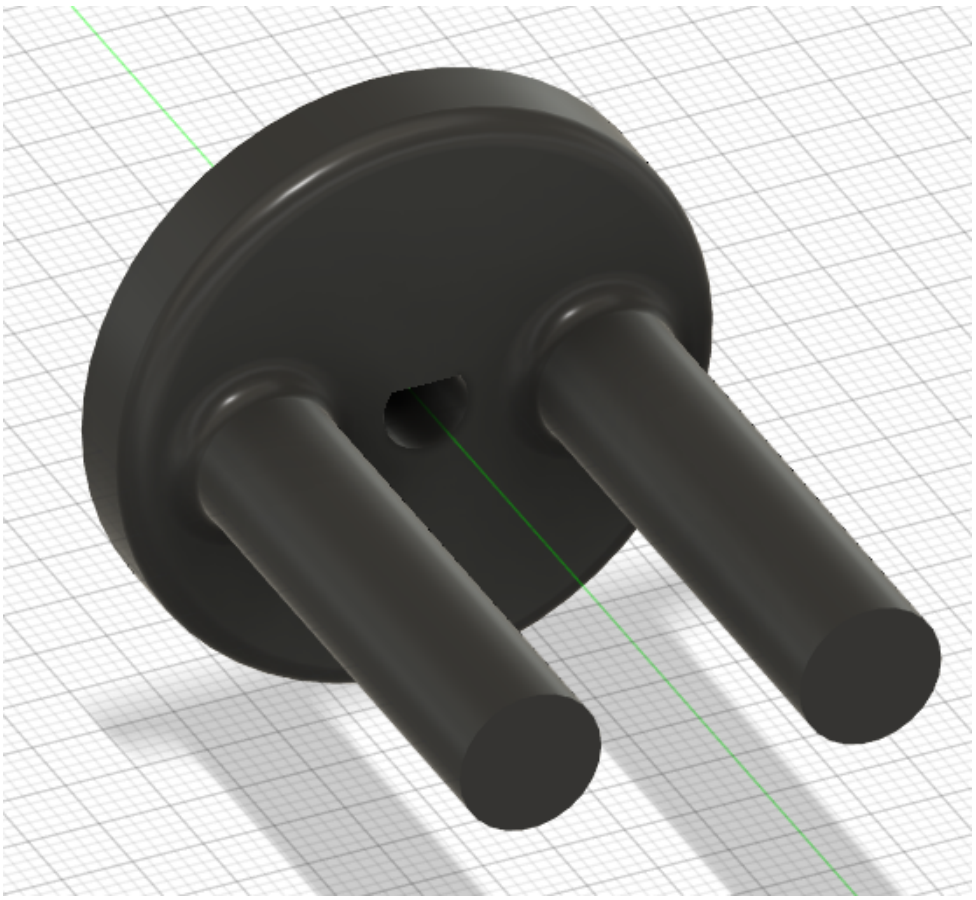
Rysunek 3.3: Box

- **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.



Rysunek 3.4: MotorBox

- **Łączniki napędu (Bridge):** Elementy pośredniczące (sprzęgła), które przenoszą moment obrotowy bezpośrednio z wałów silników na wewnętrzną strukturę półkul (Shell).

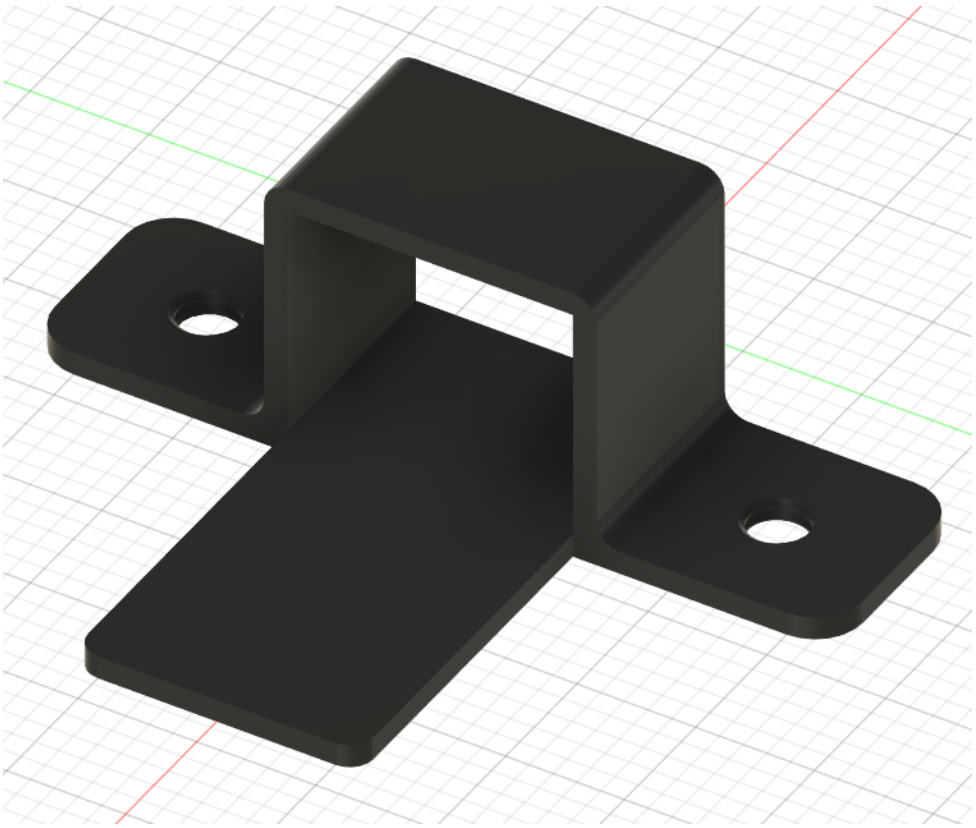


Rysunek 3.5: Bridge

### 3.1.2 Układ wahadła i powłoka zewnętrzna

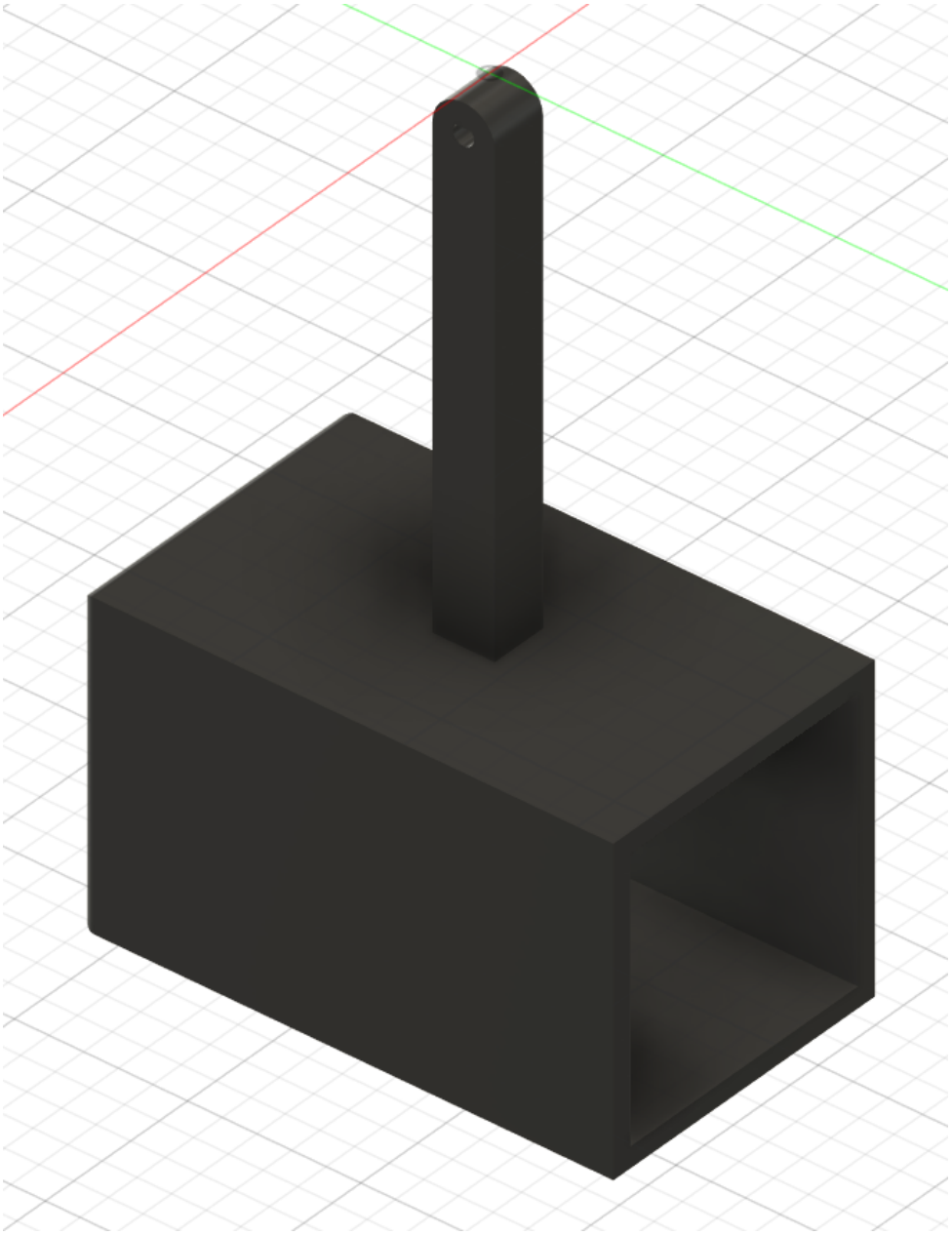
Za manewrowość oraz zewnętrzną formę robota odpowiadają następujące podzespoły:

- **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.



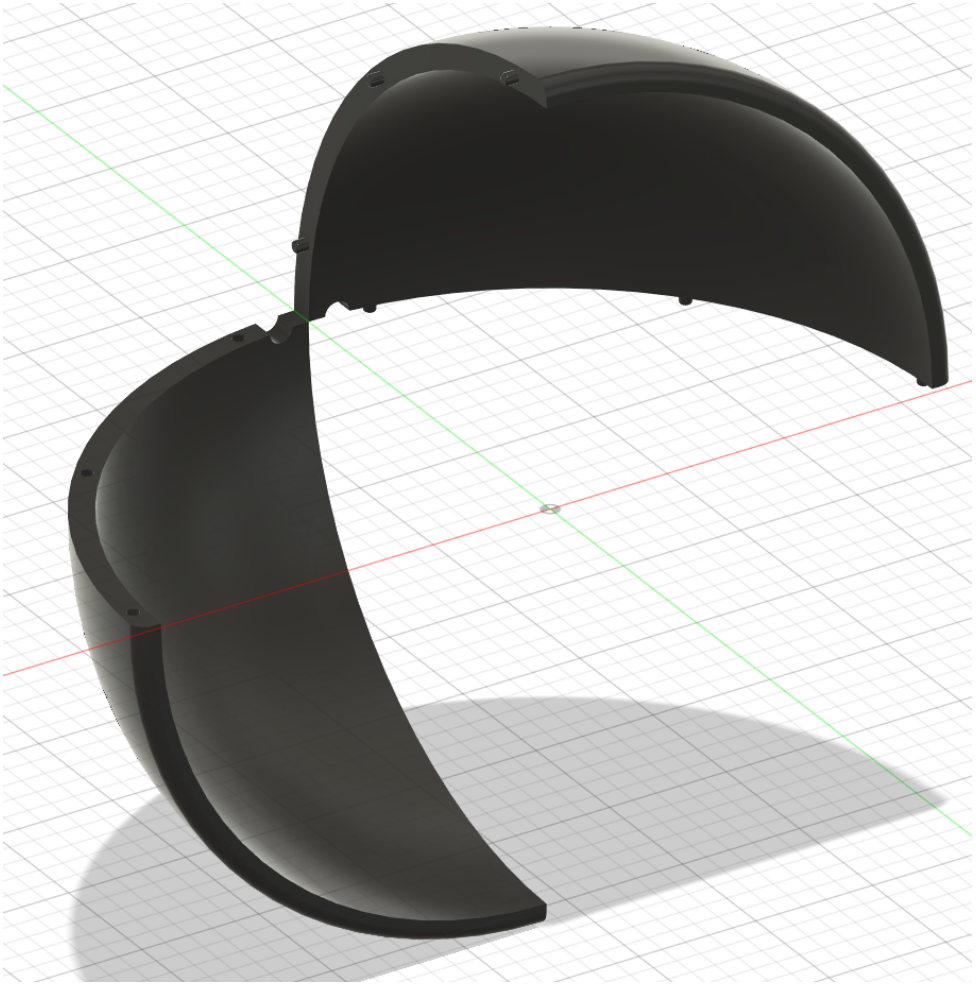
Rysunek 3.6: Pendulum Hook

- **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.



Rysunek 3.7: Pendulum

- **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.



Rysunek 3.8: Shell

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.

```
class CommandClient:
    def __init__(self, host: str, port: int, timeout: float =
    0.5):
        self.host = host
        self.port = port
        self.timeout = timeout
        self.sock = None
        self.connected = False

    def connect(self):
        self.sock = socket.socket(socket.AF_INET, socket.
    SOCK_STREAM)
        self.sock.settimeout(self.timeout)
        self.sock.connect((self.host, self.port))
        self.connected = True
```

Listing 4.1: Implementacja klienta sieciowego w pliku client.py

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 zakończone znakiem nowej linii.

```
def format_command(cmd: Command) -> bytes:
    if cmd.name == "MOVE":
        cmd_str = f"{cmd.name} {cmd.args[0]} {cmd.args[1]}"
    elif cmd.name == "INFO":
        cmd_str = f"{cmd.name} {' '.join(cmd.args)}"
    else:
        cmd_str = f"{cmd.name}"
    return (cmd_str + "\n").encode("utf-8")
```

Listing 4.2: Formatowanie komend w protocol.py

- **Mechanizm Heartbeat:** Regularne przesyłanie komendy `PING` w celu weryfikacji aktywności łącza.

### 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`. System wykorzystuje mapowanie `KEY_MAP`, które przypisuje klawiszom strzałek odpowiednie wartości wektorowe:

```
KEY_MAP = {
    'up': (1,0),
    'down': (-1,0),
    'left': (0,-1),
    'right': (0,1)
}
```

Listing 4.3: Mapowanie klawiatury w main.py

Logika przetwarzania wejścia uwzględnia priorytet klawisza spacji jako hamulca bezpieczeństwa (komenda `STOP`) oraz sumowanie wektorów dla ruchów złożonych:

```
while self.client.connected:
    if keyboard.is_pressed('space'):
        self.client.send_command(parse_command("STOP"))
        continue

    dx, dy = 0, 0
    for key in KEY_MAP:
        if keyboard.is_pressed(key):
            vx, vy = KEY_MAP[key]
            dx += vx
            dy += vy

    current = (dx, dy)
    if current != last:
        cmd = f"MOVE {dx} {dy}"
        self.client.send_command(parse_command(cmd))
        last = current
```

Listing 4.4: Pętla przetwarzania zdarzeń w keyboard_listener

W celu optymalizacji pasma sieciowego, aplikacja przesyła nową komendę `MOVE` tylko w przypadku zmiany stanu wejścia, zachowując przy tym interwał 50 ms pomiędzy kolejnymi aktualizacjami.

## 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).

```python
def start(self):
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as sock:
        sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        sock.bind((self.host, self.port))
        sock.listen(1)
        print(f"[COMM] Listening on {self.host}:{self.port}")

        while True:
            conn, addr = sock.accept()
            self.handle_client(conn)
```

Listing 4.5: Inicjalizacja i pętla nasłuchująca serwera w server.py

W procesie obsługi klienta zaimplementowano mechanizmy zwiększające niezawodność:

- **Analiza strumienia danych:** Dane są dekodowane i dzielone na poszczególne linie przed przekazaniem do parsera.
- **Zarządzanie stanem połączenia:** Klasa `ClientState` monitoruje aktywność poprzez metodę `update()`.

```python
class ClientState:
    def update(self):
        self.last_seen = time.monotonic()
        self.active = True

    def expired(self) -> bool:
        return (time.monotonic() - self.last_seen) > self.timeout
```

Listing 4.6: Mechanizm monitorowania aktywności w client_state.py

- **System bezpieczeństwa (Failsafe):** Wykrycie przekroczenia czasu oczekiwania skutkuje natychmiastowym zatrzymaniem silników.

```python
try:
    data = conn.recv(1024)
    if not data:
        self.controller.stop()
        break

except socket.timeout:
    if self.client_state.expired():
        self.controller.stop()
```

Listing 4.7: Logika Failsafe w pętli obsługi klienta (server.py)

### 4.2.2 Logika sterowania i warstwa sprzętowa

Otrzymane komendy tekstowe są parsowane na obiekty klasy `Command`, które trafiają do głównego kontrolera robota. Proces generowania ruchu przebiega w dwóch etapach:

1. **Obliczenia kinematyczne:** Klasa `DiffDriveKinematics` przelicza wektor $(v, \omega)$ na prędkości poszczególnych komponentów.

```
def calculate_wheel_speeds(self, v: float, omega: float):
    pendulum_speed = abs(v) * omega
    if v == 0:
        left_speed = v + omega
        right_speed = v - omega

    self.curr_left = self._approach(self.curr_left, left_speed *
   MAX_SPEED)
    self.curr_right = self._approach(self.curr_right, right_speed
    * MAX_SPEED)
    return self.curr_left, self.curr_right, self.curr_pend
```

Listing 4.8: Algorytm przeliczania prędkości w kinematics.py

1. **Ramping prędkości:** Metoda `_approach` zapewnia płynne narastanie sygnału PWM.

```
def _approach(self, current, target):
    if current < target:
        return min(target, current + robotSettings.
   ACCELERATION_PERCENTAGE)
    elif current > target:
        return max(target, current - robotSettings.
   ACCELERATION_PERCENTAGE)
    return target
```

Listing 4.9: Implementacja wygładzania ruchu (kinematics.py)

Bezpośrednią kontrolę nad sprzętem sprawuje moduł `RobotDriver`. Klasa `Motor` abstrahuje sterowanie sprzętowe przy pomocy biblioteki `gpiozero` [6]:

```
class Motor:
    def set_speed(self, speed):
        if speed > 0:
            self.fwd.on()
            self.back.off()
            self.pwm.value = speed
        elif speed < 0:
            self.fwd.off()
            self.back.on()
            self.pwm.value = abs(speed)
        else:
            self.stop()
```

Listing 4.10: Obsługa silników w driver.py

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

# Teoria

## 6.1 Zastosowane algorytmy i ich implementacja

W niniejszym rozdziale omówiono kluczowe algorytmy sterowania oraz mechanizmy logiczne, które odpowiadają za stabilną pracę robota. Opis obejmuje zarówno podstawy teoretyczne, jak i szczegóły implementacyjne w języku Python.

### 6.1.1 Kinematyka napędu różnicowego i sterowanie wahadłem

Głównym zadaniem warstwy kinematycznej jest przekształcenie wektora sterującego $(v, \omega)$, gdzie $v \in \{-1, 0, 1\}$ oznacza wejście liniowe, a $\omega \in \{-1, 0, 1\}$ wejście kątowe, na fizyczne sygnały sterujące. W badanym robocie kulistym zastosowano zmodyfikowany model napędu różnicowego zintegrowany z kontrolą wahadła.

**Teoria i model matematyczny:** Prędkości kół $(v_L, v_R)$ oraz prędkość wahadła $(v_p)$ są wyznaczane w zależności od trybu ruchu. Przyjmując $V_{max}$ jako maksymalne dopuszczalne wypełnienie PWM, relacje te opisują poniższe wzory.

1. **Obrót w miejscu ($v = 0$):** Robot wykonuje obrót wokół własnej osi pionowej poprzez przeciwbieżną pracę silników głównych:

$$v_L = \omega \cdot V_{max}, \quad v_R = -\omega \cdot V_{max}, \quad v_p = 0 \tag{6.1}$$

2. **Jazda do przodu ($v = 1$):** Implementacja wykorzystuje dyskretne mapowanie, które pozwala na sterowanie kierunkowe poprzez redukcję prędkości jednego z kół:

$$v_L = v \cdot \left( \left\lfloor \frac{v - \omega}{2} \right\rfloor + (v - \omega) \bmod 2 \right) \cdot V_{max} \tag{6.2}$$

$$v_R = v \cdot \left( \left\lfloor \frac{v + \omega}{2} \right\rfloor + (v + \omega) \bmod 2 \right) \cdot V_{max} \tag{6.3}$$

3. **Jazda do tyłu ($v = -1$):** Analogicznie do jazdy w przód, z uwzględnieniem wartości bezwzględnej dla poprawnego wyznaczenia kierunku obrotu silników:

$$v_L = v \cdot \left( \left\lfloor \frac{|v + \omega|}{2} \right\rfloor + (v + \omega) \bmod 2 \right) \cdot V_{max} \tag{6.4}$$

$$v_R = v \cdot \left( \left\lfloor \frac{|v - \omega|}{2} \right\rfloor + (v - \omega) \bmod 2 \right) \cdot V_{max} \tag{6.5}$$

Dodatkowo, dla każdego trybu jazdy z wektorem liniowym ($v \neq 0$), wyznaczana jest prędkość wspomagająca wahadła, która wychyla masę wewnętrzną robota w celu dynamicznej stabilizacji skrętu:

$$v_p = |v| \cdot \omega \cdot V_{max} \tag{6.6}$$

```python
def calculate_wheel_speeds(self, v: float, omega: float):
    # Obliczenie predkosci wspomagajacej wahadla
    pendulum_speed = abs(v) * omega

    if v == 0: # Obrot w miejscu
        left_speed = v + omega
        right_speed = v - omega
    elif v == 1: # Jazda do przodu z korekta kierunku
        left_speed = v * ((v-omega)//2 + (v-omega)%2)
        right_speed = v * ((v+omega)//2 + (v+omega)%2)
    elif v == -1: # Jazda do tylu z korekta kierunku
        left_speed = v * (abs(v+omega)//2 + (v+omega)%2)
        right_speed = v * (abs(v-omega)//2 + (v-omega)%2)

    # Skalowanie o predkosc maksymalna i aplikacja rampingu
    self.curr_left = self._approach(self.curr_left, left_speed *
robotSettings.MAX_SPEED_PERCENTAGE)
    self.curr_right = self._approach(self.curr_right, right_speed
    * robotSettings.MAX_SPEED_PERCENTAGE)
    self.curr_pend = self._approach(self.curr_pend,
pendulum_speed * robotSettings.MAX_SPEED_PERCENTAGE)

    return self.curr_left, self.curr_right, self.curr_pend
```

Listing 6.1: Algorytm obliczania prędkości w kinematics.py

### 6.1.2 Algorytm wygładzania ruchu (Soft Start)

Ze względu na wysoką bezwładność kuli oraz ograniczenia prądowe zasilania (ryzyko spadków napięcia przy rozruchu), zaimplementowano algorytm rampowania prędkości, znany jako *Soft Start*.

**Teoria:** Algorytm nie pozwala na skokową zmianę prędkości silnika. Zamiast tego, w każdym cyklu pętli sterującej, aktualna prędkość ($V_{curr}$) dąży do prędkości zadanej ($V_{target}$) o stały krok $\Delta$, określony parametrem `ACCELERATION_PERCENTAGE`. Proces ten można opisać wzorem:

$$V_{t+1} = \text{clamp}(V_{target}, V_t - \Delta, V_t + \Delta) \tag{6.7}$$

```python
def _approach(self, current, target):
    if current < target:
        return min(target, current + robotSettings.
ACCELERATION_PERCENTAGE)
    elif current > target:
        return max(target, current - robotSettings.
ACCELERATION_PERCENTAGE)
    return target
```

Listing 6.2: Implementacja metody _approach w kinematics.py

Ten algorytm jest wywoływany cyklicznie w osobnym wątku kontrolera z częstotliwością 20Hz (co 50ms), co zapewnia płynność ruchu i ochronę elektroniki przed nagłymi skokami poboru prądu.

### 6.1.3 Protokół komunikacyjny i parsowanie komend

Komunikacja między operatorem a robotem opiera się na autorskim protokole tekstowym przesyłanym przez strumień TCP.

**Teoria:** Zastosowano bezstanowy protokół typu *Request-Response* dla zapytań systemowych oraz jednokierunkowy strumień komend dla sterowania ruchem. Każda komenda składa się ze słowa kluczowego (np. `MOVE`, `STOP`) oraz opcjonalnych argumentów liczbowych.

```
def parse_command(raw: str) -> Command:
    parts = raw.split()
    name = parts[0].upper()

    if name == "MOVE":
        v = float(parts[1])
        omega = float(parts[2])
        return Command("MOVE", (v, omega))
    elif name == "STOP":
        return Command("STOP", ())
    # (...)
```

Listing 6.3: Logika parsera w protocol.py

### 6.1.4 Algorytm bezpieczeństwa (Failsafe)

Kluczowym elementem zapewniającym bezpieczeństwo jest algorytm wykrywania utraty łączności (tzw. *Watchdog*).

**Teoria:** Robot monitoruje czas nadejścia ostatniej poprawnej ramki danych. Jeśli odstęp czasu od ostatniej komunikacji przekroczy wartość `SOCKET_TIMEOUT`, system uznaje, że połączenie zostało przerwane (np. z powodu zasięgu Wi-Fi) i natychmiastowo zatrzymuje wszystkie procesy ruchowe.

```
except socket.timeout:
    if self.client_state.expired():
        self.controller.stop()
        self.client_state.reset()
```

Listing 6.4: Mechanizm Failsafe w server.py

Stan aktywności klienta jest zarządzany przez klasę `ClientState`, która wykorzystuje monotoniczny zegar systemowy do precyzyjnego odmierzania czasu bez ryzyka błędów wynikających ze zmiany godziny systemowej.

ROZDZIAŁ 7

# Podsumowanie

## 7.1 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.

### 7.1.1 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.

### 7.1.2 Uwagi eksploatacyjne i obserwacje (Notatki obserwatora)

Podczas testów prototypu odnotowano istotne aspekty wpływające na dynamikę urządzenia, które wynikają z fizycznych ograniczeń konstrukcji:

- **Zarządzanie budżetem energetycznym:** Zaobserwowano, że robot porusza się ze stosunkowo niską prędkością maksymalną. Jest to wynik celowego ograniczenia programowego poprzez parametr `MAX_SPEED_PERCENTAGE`. Ponieważ pakiet baterii zasila jednocześnie silniki oraz jednostkę obliczeniową Raspberry Pi, zbyt wysoki pobór prądu przez napęd powodował spadki napięcia prowadzące do restartów komputera. Ograniczenie prędkości pozwoliło zapewnić priorytet zasilania dla stabilnej pracy systemu operacyjnego i łączności bezprzewodowej.
- **Przyczepność i właściwości powłoki:** Gładka powierzchnia zewnętrznej obudowy (Shell) wykonanej z tworzywa sztucznego nie zapewnia wystarczającego współczynnika tarcia na wielu typach powierzchni. W obecnej formie robot ma trudności z pokonywaniem większych przeszkód terenowych oraz wzniesień, co sugeruje konieczność zastosowania materiałów o wyższej przyczepności lub gumowych pierścieni bieżnych.

### 7.1.3 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 powłoki:** Badania nad nowymi materiałami dla powłoki zewnętrznej, 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 [1]) 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.

# Bibliografia


[1] Jijo Malayil. Rolling justice: Chinas sci-fi spherical robot cop can capture criminals in style. *Interesting Engineering*, 2024. URL https://interestingengineering.com/innovation/chinas-spherical-robot-cop-captures-criminals.

[2] *S9V11MACMA - przetwornica step-up/step-down - 2,5-9V 1,5A - z funkcją odcięcia przy niskim napięciu - Pololu 2868*. Pololu, . URL https://botland.com.pl/przetwornice-step-up-step-down/10673-s9v11macma-przetwornica-step-upstep-down-25-9v-15a-z-funkcja-odciecia-przy-niskim-napieciu-pololu-2868-5904422305826.html.

[3] *S13V30F5 - przetwornica step-up/step-down - 5V 3A - Pololu 4082*. Pololu, . URL https://botland.com.pl/przetwornice-step-up-step-down/19768-s13v30f5-przetwornica-step-upstep-down-5v-3a-pololu-4082-5904422347529.html.

[4] *Raspberry Pi Zero 2 W Product Brief*. Raspberry Pi Foundation. URL https://datasheets.raspberrypi.com/rpizero2/raspberry-pi-zero-2-w-product-brief.pdf.

[5] *L293D STMicroelectronics-IC driver*. STMicroelectronics. URL https://www.tme.eu/pl/details/l293d/drivery-silnikowe-i-pwm/stmicroelectronics/.

[6] *Interactive pinout diagram for Raspberry Pi Zero 2 W*. Vercel. URL https://pinouts.vercel.app/board/raspberry-pi-zero-2-w.