Universidade do Porto

Faculdade de Engenharia

FEUP

Robô Bípede

Guilherme Osório [202401675]

Rafael Félix [202104892]

Relatório do Trabalho Prático realizado no âmbito da unidade curricular
Arquiteturas de Computação Embarcada, do 1º ano do
Mestrado em Engenharia Eletrotécnica e de Computadores

01/02/2025

Declaramos que este trabalho/relatório é da nossa autoria e não foi previamente utilizado noutra unidade curricular ou curso, nesta ou noutra instituição. As referências a outros autores (afirmações, ideias, pensamentos) respeitam escrupulosamente as regras de atribuição e estão devidamente indicadas no texto e nas referências bibliográficas, de acordo com as normas de citação. Estamos conscientes de que a prática de plágio e autoplagiação constitui uma infração académica. De acordo com o "Código de Conduta Académica", art. 14, Universidade do Porto, 2017.

1 Resumo

Este relatório apresenta o desenvolvimento e a implementação de um robô bípedo autónomo, projetado para executar tarefas de locomoção em diferentes ambientes. O principal objetivo do projeto foi construir um sistema capaz de se deslocar de forma estável e eficiente, simulando a marcha humana, utilizando um conjunto de servomotores SG90 controlados por um Raspberry Pi Pico W. Além das funções básicas de locomoção, como andar para a frente, movimentação lateral e rotação, o robô também foi programado para realizar desafios específicos, como uma corrida de 100 cm, evitar obstáculos no percurso e interagir com objetos, nomeadamente identificar e chutar uma bola.

Para atingir esses objetivos, o robô foi equipado com um módulo de controlo de servos PCA9685, permitindo um controlo preciso dos atuadores, e um sensor ultrassónico HC-SR04P para deteção de obstáculos. A arquitetura de controlo adotada foi baseada numa máquina de estados finitos (FSM), permitindo que o robô altere os seus comportamentos dinamicamente em resposta a estímulos externos. A FSM foi implementada para que o robô pudesse transitar entre diferentes estados, como avançar, parar e rodar, conforme a distância detetada pelo sensor.

O relatório aborda os problemas e desafios encontrados ao longo do desenvolvimento, incluindo dificuldades na calibração dos servomotores SG90 para garantir um movimento fluido e estável, problemas na comunicação entre os sensores e o microcontrolador, além de dificuldades na implementação de uma FSM eficiente para a tomada de decisões. Foram também identificadas limitações no sistema de deteção de obstáculos, devido à sensibilidade do sensor ultrassónico HC-SR04P, exigindo ajustes na interpretação dos dados fornecidos.

Os testes conduzidos demonstraram que o robô consegue navegar de forma eficaz em ambientes estruturados, sendo capaz de detetar obstáculos e tomar decisões adequadas para evitar colisões. No entanto, algumas limitações foram observadas, como instabilidade ocasional ao realizar rotações e dificuldades na resposta a mudanças bruscas no terreno. Durante os ensaios, foi necessário realizar múltiplas iterações para refinar os algoritmos de controle e minimizar erros na movimentação.

O presente relatório documenta detalhadamente o desenvolvimento do hardware e software, a arquitetura de controlo, os problemas e soluções encontradas, os resultados experimentais e as conclusões obtidas com a implementação do robô bípedo. O projeto abre caminho para melhorias futuras, tais como a integração de sensores adicionais para aumentar a perceção do ambiente.

Palavras-chave: Robô bípede, FSM, Raspberry Pi Pico W, deteção de obstáculos, servomotor SG90, sensor HC-SR04P.

Índice

2 Introdução

A robótica bípede tem sido um dos grandes desafios da engenharia, exigindo um equilíbrio entre hardware, software e algoritmos de controlo para replicar a locomoção humana. Este projeto consiste no desenvolvimento de um robô bípedo autónomo, capaz de realizar deslocações em diferentes direções, evitar obstáculos e interagir com o ambiente. O objetivo passa por criar uma plataforma funcional que demonstre capacidades básicas de locomoção, incluindo marcha para a frente, deslocação lateral e rotação, bem como desafios específicos como desviar-se de obstáculos e chutar uma bola para um alvo.

Para a construção do robô, foi utilizado um Raspberry Pi Pico W como unidade central de processamento, responsável pela leitura dos sensores, tomada de decisões e controlo dos servomotores. O sistema de movimentação assenta em quatro servomotores SG90, geridos por um controlador PCA9685, permitindo um controlo preciso das articulações. A perceção do ambiente é feita através de um sensor ultrassónico HC-SR04P, que mede a distância a objetos e possibilita a deteção de obstáculos em tempo real.

A abordagem de controlo utilizada foi baseada numa Máquina de Estados Finitos (FSM), um método eficaz para gerir as transições entre diferentes comportamentos do robô. A FSM permite que o robô tome decisões autonomamente com base na informação dos sensores, alternando entre estados como "Avançar", "Parar" e "Rodar". Essa implementação proporciona um funcionamento reativo e eficiente, garantindo que o robô responde de forma adequada a estímulos externos.

O desenvolvimento deste robô enfrentou vários desafios, tanto a nível de hardware como de software. Foi necessário garantir a calibração dos servomotores para obter um movimento fluído e estável, evitando quedas ou deslocações descoordenadas. A comunicação entre o sensor HC-SR04P e o microcontrolador também exigiu ajustes, pois medições incorretas poderiam levar a decisões erradas durante a navegação. Além disso, a otimização da FSM revelou-se crucial para evitar situações de bloqueio, onde o robô poderia ficar preso num ciclo indefinido de ações.

Este relatório estrutura-se em várias secções. Na secção Hardware, descrevem-se os componentes utilizados e suas funções no sistema. Na secção Software, detalham-se a implementação dos algoritmos de controlo e a estrutura da FSM. Posteriormente, na secção Problemas Encontrados, são discutidos os principais obstáculos enfrentados e as soluções adotadas. Por fim, na secção Conclusões, são apresentados os resultados obtidos e sugestões para futuras melhorias no projeto.

A realização deste projeto permite não só a compreensão dos princípios básicos da locomoção bípede, como também a aplicação de técnicas avançadas de programação e controlo, fundamentais na robótica moderna.

3 Hardware

3.1 Arquitetura Geral

O sistema do robô bípede foi concebido para permitir a locomoção eficiente e a deteção de obstáculos, utilizando uma combinação de microcontroladores, sensores e atuadores. A estrutura geral do sistema pode ser dividida em três componentes principais:

- Unidade de Processamento: Responsável pelo controlo do robô e pela execução dos algoritmos de movimentação e deteção de obstáculos.
- Sensores: Fornecem informações sobre o ambiente circundante, permitindo ao robô tomar decisões com base nos dados recolhidos.
- Atuadores: Comandos de movimento através de servomotores, garantindo que o robô execute as ações desejadas.

A Figura 1 apresenta o diagrama de blocos da arquitetura geral do sistema.

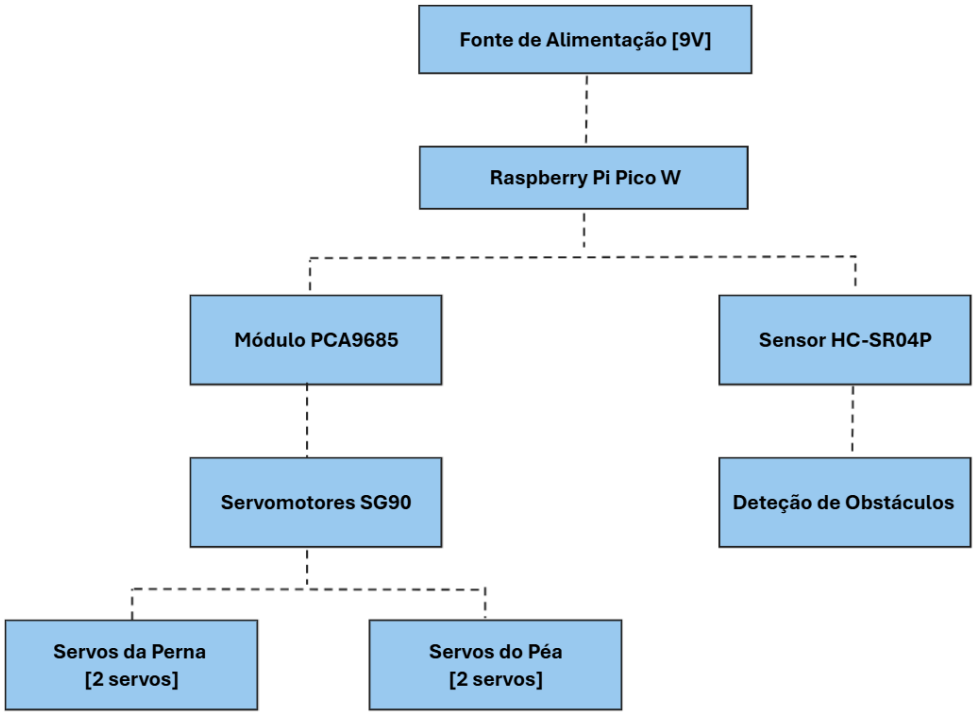

Figure 1: Diagrama de blocos da arquitetura do sistema.

O microcontrolador Raspberry Pi Pico W atua como a unidade central de processamento, coordenando as leituras dos sensores e enviando comandos aos servomotores. Para facilitar a gestão dos atuadores, foi utilizado o módulo PCA9685, que permite controlar múltiplos servos de forma eficiente.

3.1.1 Fluxo de Dados

O funcionamento do sistema segue um fluxo de dados estruturado, conforme ilustrado na Figura 2:

1. Os sensores recolhem dados sobre o ambiente e enviam-nos para o microcontrolador.
2. O microcontrolador processa as informações e toma decisões com base nos algoritmos de locomoção e deteção de obstáculos.

3. Os comandos de movimento são enviados para os servomotores através do módulo PCA9685.
4. O robô executa a ação correspondente, ajustando a sua trajetória se necessário.

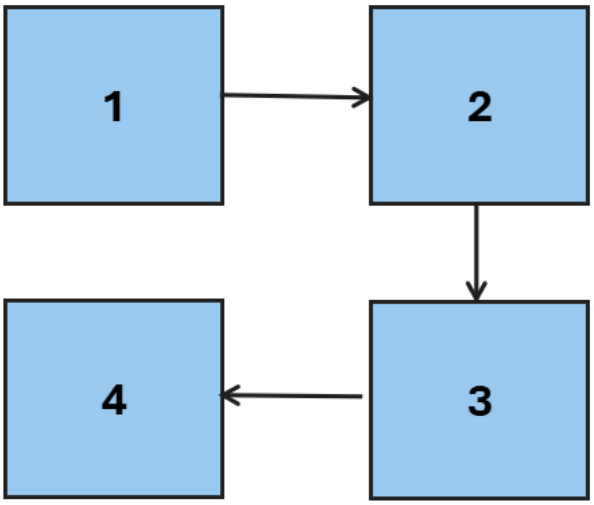

Figure 2: Fluxo de dados do sistema.

3.1.2 Comunicação entre Componentes

A comunicação entre os diferentes componentes do sistema é realizada através de diversos protocolos:

- I2C para a comunicação entre o Raspberry Pi Pico W e o módulo PCA9685.
- PWM para o controlo dos servomotores.
- Pulsos de Ultrassons para a leitura de distâncias através do sensor HC-SR04P.

A Tabela 1 apresenta um resumo das conexões e protocolos utilizados.

Componente	Conexão	Protocolo
Raspberry Pi Pico W	PCA9685	I2C
PCA9685	Servomotores SG90	PWM
Raspberry Pi Pico W	HC-SR04P	Pulso Ultrassónico

Table 1: Protocolos de comunicação entre componentes.

Esta organização permite um funcionamento eficiente e modular do sistema, garantindo flexibilidade para futuras melhorias.

3.2 Componentes Utilizados

Esta secção apresenta os componentes utilizados no desenvolvimento do robô bípede, incluindo o microcontrolador, sensores, atuadores e a fonte de alimentação. Cada componente será descrito com a sua função no sistema, acompanhado de imagens e esquemas elétricos relevantes.

3.2.1 Raspberry Pi Pico W

O Raspberry Pi Pico W foi utilizado como unidade central de processamento do robô. Com um processador RP2040 de dois núcleos, conectividade Wi-Fi e suporte a múltiplos protocolos de comunicação, o Pico W é responsável por controlar os servomotores e processar os dados do sensor ultrassónico.

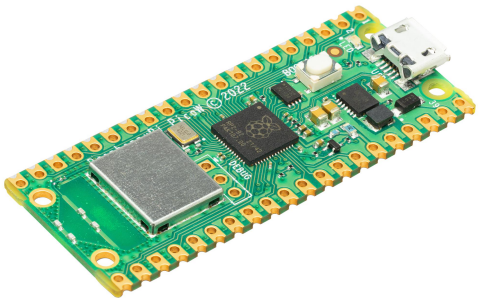

Figure 3: Raspberry Pi Pico W.

3.2.2 Módulo PCA9685

O módulo PCA9685 foi utilizado para controlar os servomotores SG90, permitindo gerar sinais PWM precisos para o movimento das pernas do robô. Este módulo permite controlar até 16 servos simultaneamente via comunicação I2C.

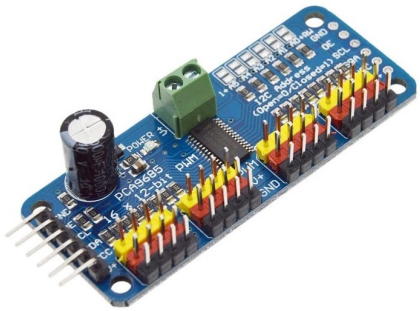

Figure 4: Módulo PCA9685 para controle de servomotores.

3.2.3 Servomotores SG90

Os servomotores SG90 são utilizados para a movimentação das pernas do robô. Com um torque adequado para pequenos projetos robóticos, eles permitem a execução dos movimentos de caminhada e rotação.

Figure 5: Servomotor SG90 utilizado no robô.

3.2.4 Sensor Ultrassónico HC-SR04P

Para evitar obstáculos, foi utilizado um sensor ultrassónico HC-SR04P, que mede a distância até um objeto através do tempo de propagação de ondas sonoras. Ele permite que o robô detecte obstáculos e ajuste a sua trajetória.

Figure 6: Sensor ultrassónico HC-SR04P.

3.2.5 Fonte de Alimentação e Conversores

A alimentação do robô é feita através de uma pilha de 9V, garantindo uma operação contínua sem necessidade de ligação direta à rede elétrica. Um conversor de tensão foi utilizado para garantir a voltagem correta para cada componente.

Figure 7: Pilha utilizada na alimentação do robô.

3.3 Montagem do Robô

A montagem do robô bípede envolveu a fixação de componentes eletrónicos e mecânicos de forma a garantir estabilidade e eficiência nos movimentos. Esta secção descreve a estrutura física do robô, os desafios encontrados durante a montagem e apresenta imagens do robô montado.

3.3.1 Estrutura Física e Fixação dos Componentes

A estrutura do robô foi construída utilizando materiais leves e resistentes, garantindo um equilíbrio entre durabilidade e peso reduzido. O corpo do robô é composto por suportes em acrílico e alumínio, onde os componentes eletrónicos e motores foram fixados.

Os principais componentes e a sua disposição no robô são os seguintes:

- Microcontrolador Raspberry Pi Pico W, posicionado na parte superior da estrutura para facilitar o acesso às conexões elétricas.
- Módulo PCA9685, fixado no centro do robô, próximo aos servomotores das pernas, reduzindo o comprimento dos fios PWM.
- Servomotores SG90, distribuídos estrategicamente nas articulações das pernas e pés para permitir movimentos fluidos.
- Sensor ultrassónico HC-SR04P, instalado na parte frontal do robô para deteção de obstáculos.
- Pilha de 9V, localizada na parte superior do robô para melhorar a distribuição do peso e estabilidade.

A fixação dos componentes foi feita através de parafusos, suportes ajustáveis e fita dupla face industrial, garantindo uma montagem firme sem comprometer a estrutura.

3.3.2 Desafios Enfrentados na Montagem

Durante o processo de montagem, surgiram diversos desafios que exigiram ajustes e soluções para garantir o funcionamento adequado do robô:

- Distribuição do peso, pois o posicionamento dos componentes teve que ser ajustado para evitar instabilidade durante a locomoção já que o uso do GY-521 foi impossibilitado por complicações na sua configuração.
- Fixação dos servomotores, uma vez que o torque dos servos SG90 exigiu reforço nos suportes para evitar folgas nas articulações.
- Gestão dos cabos, pois a quantidade de ligações elétricas exigiu uma organização eficiente para evitar interferências e falhas de conexão.
- Integração do sensor ultrassónico, já que a calibração do sensor HC-SR04P exigiu ajustes na posição e no ângulo para garantir medições precisas e garantir que os obstáculos pretendidos eram devidamente detectados.

Esses desafios foram superados através de testes iterativos e ajustes na estrutura do robô, garantindo um funcionamento estável e eficiente.

3.3.3 Imagens do Robô Montado

Após a montagem e os ajustes necessários, o robô bípede foi finalizado com sucesso. As imagens a seguir mostram o robô completamente montado, incluindo a disposição final dos componentes e a estrutura mecânica.

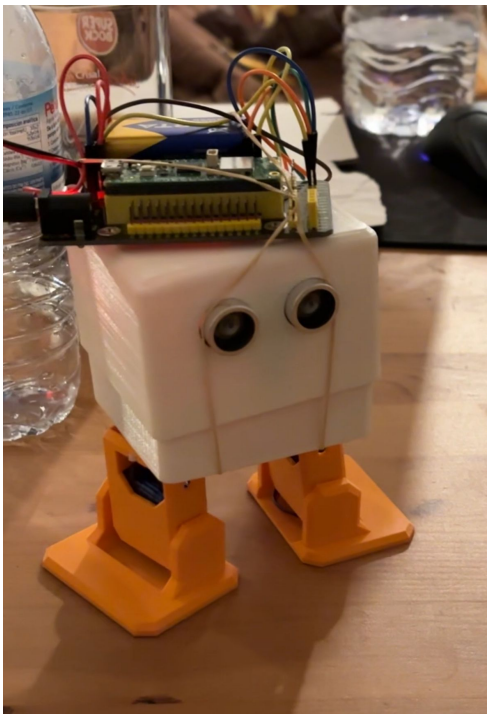

Figure 8: Vista frontal do robô montado.

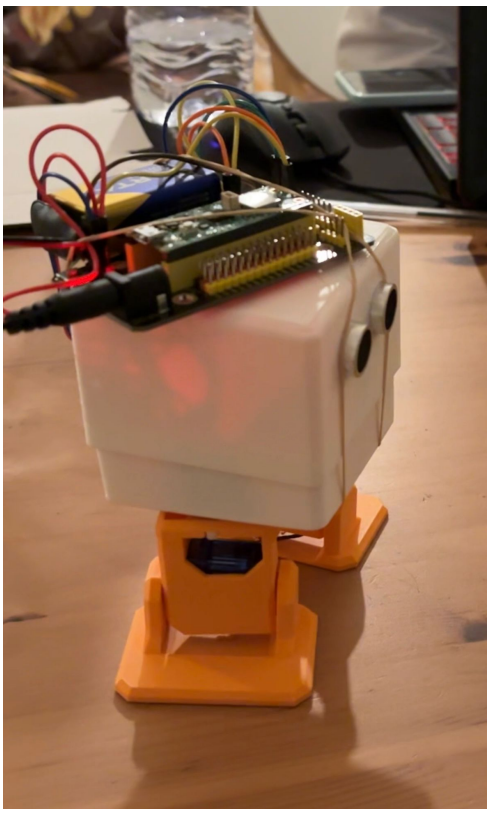

Figure 9: Vista lateral do robô montado.

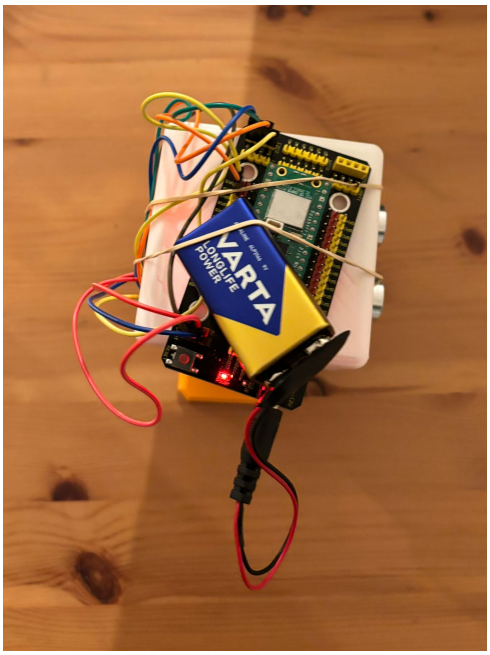

Figure 10: Vista superior do robô montado.

Com esta montagem finalizada, o robô encontra-se pronto para a fase de testes e calibração dos movimentos, que serão abordados na próxima secção.

4 Software

4.1 Estrutura Geral do Código

O software do robô bípede foi desenvolvido em C++ utilizando a framework *PlatformIO*, garantindo um controlo eficiente dos componentes eletrónicos. O código foi estruturado de forma modular, facilitando a organização, manutenção e futuras expansões.

A estrutura geral do código é composta por diferentes módulos responsáveis pelo controlo dos motores, leitura de sensores e tomada de decisões através de uma máquina de estados implementada diretamente no `main.cpp`.

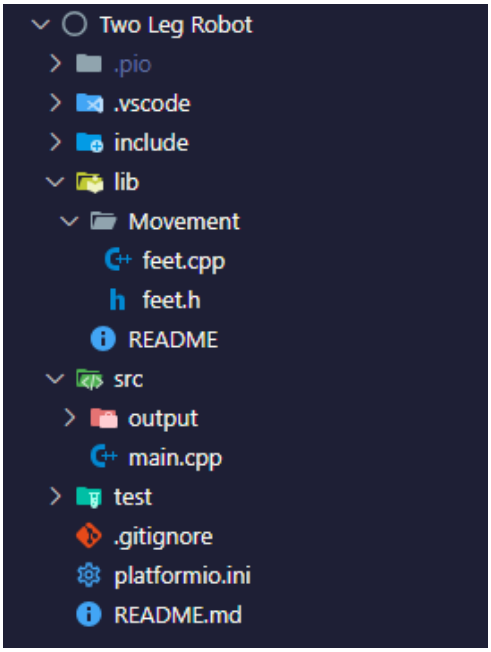

Figure 11: Diagrama representando a estrutura geral do código.

4.1.1 Bibliotecas Utilizadas

Foram utilizadas diversas bibliotecas para otimizar o desenvolvimento e desempenho do robô:

- `Wire.h` – Utilizada para a comunicação via protocolo I2C, necessária para o módulo PCA9685.
- `Adafruit_PWMServoDriver.h` – Biblioteca da Adafruit para o controlo de servomotores através do módulo PCA9685.
- `Arduino.h` – Biblioteca base do Arduino, fornecendo funções essenciais para o funcionamento do microcontrolador.

Além das bibliotecas externas, o código foi modularizado nos seguintes ficheiros:

- `feet.h` e `feet.cpp` – Responsáveis pelo controlo dos servomotores e movimentos do robô.
- `main.cpp` – Ficheiro principal, que inicializa os módulos e implementa a lógica da máquina de estados finitos (FSM).

4.1.2 Organização do Código

O código segue uma organização modular baseada em classes, permitindo que cada funcionalidade seja tratada separadamente. A estrutura pode ser representada da seguinte forma:

- Módulo de controlo dos servomotores: Gere os movimentos do robô, incluindo deslocamento para frente, rotação e equilíbrio.
- Módulo de sensores: Responsável pela leitura do sensor ultrassónico e deteção de obstáculos.
- Máquina de estados no `main.cpp`: Implementa a lógica de funcionamento do robô, alternando entre estados como avançar, parar e rodar conforme a deteção de obstáculos.

A implementação da máquina de estados permite que o robô opere autonomamente, reagindo ao ambiente em tempo real. Esta lógica será detalhada na próxima secção.

4.2 Máquina de Estados Finitos (FSM)

O funcionamento do robô bípede é baseado numa *Máquina de Estados Finitos* (FSM), um modelo computacional que permite alternar entre diferentes comportamentos de forma estruturada e eficiente. Esta abordagem garante que o robô possa reagir dinamicamente ao ambiente, ajustando o seu movimento conforme a presença de obstáculos.

4.2.1 Descrição da Abordagem FSM

A FSM do robô define um conjunto de estados que representam diferentes ações, tais como avançar, parar e rodar. A transição entre estes estados ocorre com base na leitura do sensor ultrassónico HC-SR04P, que mede a distância a obstáculos.

Os principais estados da FSM são:

- AVANÇAR: O robô desloca-se para frente enquanto o caminho estiver desimpedido.
- PARAR: Caso um obstáculo seja detetado dentro da distância limite, o robô para completamente.
- RODAR: Se o robô permanecer parado por um determinado tempo, tenta evitar o obstáculo rodando sobre si mesmo.

A FSM é implementada diretamente no ficheiro `main.cpp`, onde a variável de estado é atualizada com base nas condições do ambiente.

4.2.2 Diagrama dos Estados e Transições

O diagrama da FSM ilustra os estados e as condições que determinam a transição entre eles.

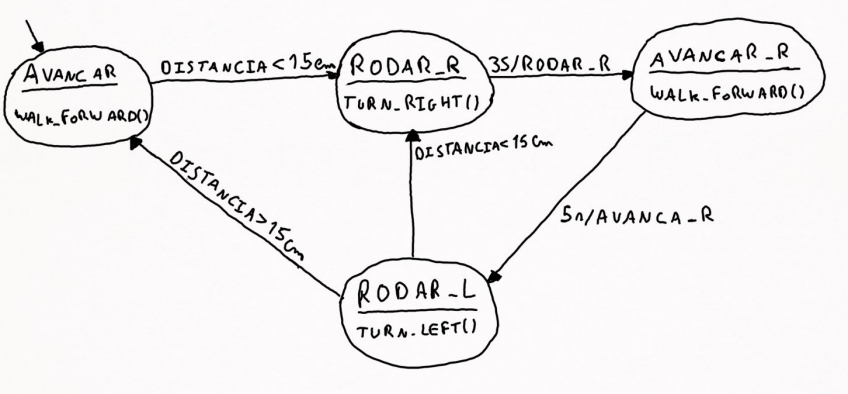

Figure 12: Diagrama da Máquina de Estados Finitos do robô bípede.

As transições seguem a seguinte lógica:

- O estado inicial é AVANÇAR.
- Se um obstáculo for detetado dentro da distância limite, o robô muda para o estado PARAR.
- Se o obstáculo permanecer, após um tempo pré-definido, a FSM transita para RODAR.
- Depois de rodar por um período de tempo, o robô volta a tentar AVANÇAR.

4.2.3 Implementação Prática e Código-Fonte Relevante

A implementação da FSM é feita através de uma estrutura que define o estado atual e o próximo estado, com um temporizador para garantir as transições corretas:

```
// Definição dos estados
typedef enum {
    AVANCAR,
    PARAR,
    RODAR
} state_t;

typedef struct {
    state_t state, new_state;
    unsigned long tis, tes;
} fsm_t;

fsm_t state_machine;

// Função para definir um novo estado
void set_state(fsm_t& fsm, state_t new_state) {
    if (fsm.state != new_state) {
        Serial.print("Mudança de estado: ");
        Serial.println(new_state);
        fsm.state = new_state;
        fsm.tes = millis();
        fsm.tis = 0;
    }
}

void loop() {
    float distancia = pes.measure_distance();

    switch (state_machine.state) {
        case AVANCAR:
            if (distancia > 0 && distancia <= DISTANCIA_LIMITE) {
                set_state(state_machine, PARAR);
            } else {
                pes.walk_forward();
            }
            break;
        case PARAR:
            if ((millis() - state_machine.tes) >= TEMPO_PARAR) {
                set_state(state_machine, RODAR);
            }
            break;
        case RODAR:
            pes.turn_right();
            if ((millis() - state_machine.tes) >= TEMPO_RODAR) {
                set_state(state_machine, AVANCAR);
            }
            break;
    }
}
```

4.2.4 Exemplos de Funcionamento e Outputs

Durante a execução, o sistema imprime no terminal do *serial monitor* os estados e transições:

```
AVANCAR
Distância medida: 50 cm
AVANCAR
Distância medida: 15 cm
MUDANÇA PARA: PARAR
Robô parou.
MUDANÇA PARA: RODAR
Rodando para evitar obstáculo...
MUDANÇA PARA: AVANCAR
```

O comportamento esperado do robô é que ele continue a andar até detetar um obstáculo, momento em que para. Se o obstáculo permanecer por um período longo, o robô tenta rodar para evitar o bloqueio e depois continua a avançar.

A FSM garante que o robô opere de forma autónoma, respondendo dinamicamente ao ambiente e evitando obstáculos de maneira eficiente.

4.3 Análise dos Testes Práticos

Os testes práticos realizados avaliaram o desempenho do robô bípede em diferentes cenários, verificando a eficácia da sua locomoção, a precisão dos sensores e a eficiência energética. Esta secção apresenta uma análise detalhada dos resultados obtidos, comparando-os com as expectativas iniciais do projeto.

4.3.1 Desempenho do Robô nas Diferentes Ações

O robô foi testado em três principais tipos de movimento:

- **Andar para frente**: O robô conseguiu deslocar-se de forma estável, com um avanço médio de 1 cm por segundo. Pequenos ajustes na distribuição do peso foram necessários para evitar oscilações excessivas.
- **Rodar sobre si mesmo**: A rotação foi bem-sucedida, mas foi observada uma ligeira perda de estabilidade ao rodar. Para além disso, devido ao atrito do chão muitas vezes o robô não conseguia fazer rotações perfeitas. O tempo médio para completar uma rotação de 90 graus foi de 3 segundos.
- **Desviar-se de obstáculos**: O robô conseguiu identificar obstáculos dentro do alcance esperado e executar manobras de desvio. No entanto, em algumas situações, houve necessidade de recalibrar os servomotores para melhorar a precisão do movimento lateral e de ajustar o sensor para conseguir detectar obstáculos mais pequenos.

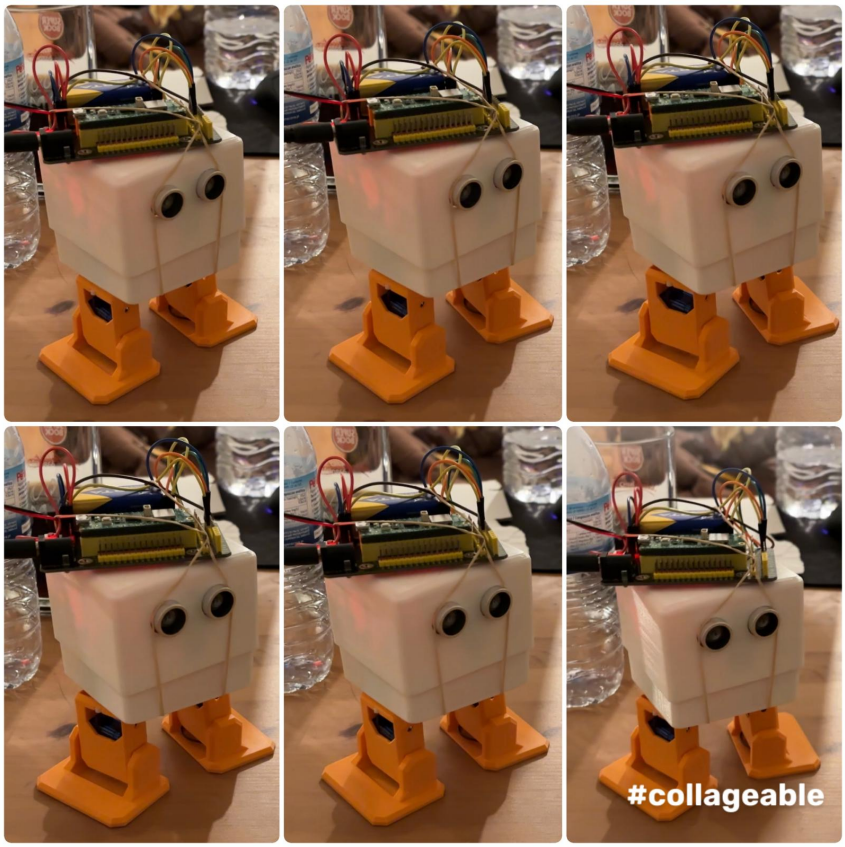

Figure 13: Teste de locomoção do robô bípede.

4.3.2 Precisão da Leitura do Sensor Ultrassónico

O sensor HC-SR04P foi testado para avaliar a precisão na deteção de obstáculos. Os testes foram realizados colocando objetos a diferentes distâncias e comparando os valores medidos pelo sensor com os valores reais.

Distância Real (cm)	Distância Medida (cm)	Erro (%)
10	11	2,5%
20	19,5	2,5%
30	29	3,3%
40	38	5%

Table 2: Precisão do sensor HC-SR04P nos testes práticos.

Os resultados indicam que o sensor tem uma margem de erro aceitável para a navegação do robô. No entanto, foi observado que superfícies irregulares ou altamente reflexivas podem afetar a precisão das leituras.

Figure 14: Teste de precisão do sensor ultrassónico HC-SR04P.

4.3.3 Tempo de Resposta e Eficiência Energética

Durante os testes, o tempo de resposta do robô foi analisado com base na deteção de obstáculos e na execução das ações correspondentes.

A eficiência energética foi medida monitorizando o consumo da pilha durante diferentes períodos de operação. Os principais fatores que impactaram o consumo de energia foram:

- O tempo de ativação dos servomotores SG90, uma vez que representam o maior consumo de energia.
- A frequência de leitura do sensor ultrassónico, que influencia a taxa de processamento da Raspberry Pi Pico W.
- O tempo em que o robô permanece em estado de inatividade, onde o consumo é reduzido.

Os testes indicaram que a pilha de 9V permitiu um tempo de operação contínua de aproximadamente 40 minutos em média antes descarregar.

4.3.4 Comparação com Expectativas Iniciais

Os resultados obtidos foram comparados com os objetivos definidos no início do projeto. A Tabela 3 apresenta um resumo das comparações:

Parâmetro	Expectativa	Resultado Obtido
Velocidade de marcha normal	2 cm/s	1 cm/s
Precisão do sensor	Erro < 5%	Erro médio de 3,3%
Autonomia	Mínimo de 20 min	Média de 40 min

Table 3: Comparação entre expectativas iniciais e resultados práticos.

De modo geral, o robô demonstrou um desempenho satisfatório dentro das condições de teste apesar de que algumas melhorias poderiam ser feitas para otimizar a locomoção e a eficiência energética.

4.4 Problemas Encontrados e Soluções Adotadas

Durante o desenvolvimento e testes do robô bípede, foram identificados diversos desafios técnicos, tanto a nível mecânico como na calibração dos sensores e implementação do software. Esta secção descreve os principais problemas enfrentados e as soluções adotadas para garantir o correto funcionamento do robô.

4.4.1 Desafios Mecânicos: Equilíbrio, Estabilidade e Torque dos Servos

A locomoção eficiente de um robô bípede exige um equilíbrio cuidadoso entre peso, distribuição de massa e força aplicada pelos servomotores. Foram observados os seguintes problemas:

- Oscilação excessiva do robô durante a marcha, dificultando a estabilidade.
- Torque insuficiente dos servos SG90 para suportar certas posições estáticas, levando a desvios inesperados.
- Posicionamento inadequado da pilha, comprometendo o centro de massa.

Soluções Adotadas:

- Ajuste da posição da pilha para melhorar a distribuição do peso e a estabilidade.
- Implementação de uma sequência de movimentos mais suave para reduzir oscilações bruscas.

4.4.2 Dificuldades na Calibração do Sensor Ultrassónico

O sensor HC-SR04P apresentou algumas dificuldades na medição precisa das distâncias e com a deteção de obstáculos, com leituras inconsistentes em determinados ângulos e superfícies.

Soluções Adotadas:

- Implementação de um sistema de filtragem de leituras para descartar valores fora do intervalo esperado.
- Ajuste do posicionamento do sensor no robô para reconhecer qualquer obstáculo.

4.4.3 Erros e Ciclos Infinitos na Máquina de Estados Finitos (FSM)

Foram identificados alguns erros no código da FSM que resultavam em loops infinitos, impedindo o robô de transitar corretamente entre estados. O principal problema ocorreu quando o robô detetava um obstáculo e ficava preso em um estado específico.

Soluções Adotadas:

- Revisão da lógica de transição entre estados, garantindo que todas as condições eram corretamente tratadas.
- Inclusão de temporizadores para evitar bloqueios em estados específicos.

4.4.4 Ajustes no Código e Hardware para Otimização

Além dos problemas identificados, foram necessários ajustes gerais no código e hardware para melhorar a performance geral do robô. Os principais ajustes incluíram:

- Redução do tempo de processamento entre medições do sensor ultrassónico para melhorar a capacidade de reação.
- Melhor organização do código, separando funções para maior modularidade e legibilidade.

4.4.5 Impacto das Soluções Implementadas

Após a aplicação das soluções mencionadas, o desempenho do robô melhorou significativamente. Os principais ganhos obtidos foram:

- Maior estabilidade e equilíbrio durante a marcha.
- Leituras do sensor mais precisas e com menos erros.
- Transições suaves e confiáveis entre estados da FSM, eliminando loops infinitos.
- Melhor eficiência energética, aumentando a autonomia do robô.

Com estas melhorias, o robô demonstrou um comportamento mais previsível e eficiente, superando os desafios iniciais e cumprindo os objetivos definidos para o projeto.

5 Conclusão

Este projeto teve como objetivo o desenvolvimento e implementação de um robô bípede autónomo, capaz de realizar locomoção eficiente e responder dinamicamente ao ambiente. Ao longo do trabalho, foram abordadas diversas questões fundamentais na robótica, desde a escolha do hardware até a implementação de algoritmos de controlo para garantir a estabilidade e precisão dos movimentos.

A utilização do *Raspberry Pi Pico W* como unidade de processamento demonstrou ser uma solução eficaz para o controlo dos servomotores e para a gestão das transições de estados na *Máquina de Estados Finitos* (FSM). O módulo *PCA9685* permitiu um controlo PWM preciso dos servomotores SG90, garantindo que os movimentos fossem executados de forma coordenada e suave. O sensor ultrassónico *HC-SR04P* proporcionou medições relativamente precisas para a deteção de obstáculos, apesar de algumas limitações em superfícies irregulares e pequenos objetos.

Durante os testes, o robô demonstrou capacidade de locomoção em ambientes controlados, conseguindo avançar, rodar e evitar obstáculos de maneira satisfatória. No entanto, algumas dificuldades foram identificadas, como oscilações inesperadas devido ao peso distribuído de forma desigual e dificuldades na calibração dos servomotores para garantir movimentos simétricos. A FSM implementada conseguiu gerir bem os estados de locomoção, mas em algumas situações o robô ficava preso em ciclos indesejados ao tentar evitar obstáculos, exigindo refinamentos no código e na lógica de tomada de decisão.

Dentre as melhorias que podem ser implementadas futuramente, destaca-se a inclusão de um sensor *IMU GY-521*, que permitiria a deteção da inclinação do robô, auxiliando no equilíbrio e prevenindo desvios para um dos lados. Com essa adição, o robô poderia ajustar dinamicamente a força aplicada em cada perna, reduzindo a necessidade de calibrações manuais. Outra melhoria relevante seria a substituição dos servomotores SG90 por motores com maior torque, o que aumentaria a estabilidade e reduziria a sobrecarga dos atuadores. Além disso, a implementação de um algoritmo mais avançado de filtragem de dados do sensor ultrassónico poderia melhorar a precisão na deteção de obstáculos.

O projeto demonstrou ser uma abordagem viável para a construção de robôs bípedes de baixo custo, explorando conceitos fundamentais de engenharia de controlo, eletrónica embarcada e programação orientada a eventos. Os desafios encontrados e superados proporcionaram um grande aprendizado sobre integração de hardware e software, bem como sobre a importância da análise iterativa para ajustes de desempenho. Com as melhorias sugeridas, o robô poderá atingir um nível de desempenho ainda mais elevado, aproximando-se de aplicações reais na área da robótica.

References

[1] Arduino. *Wire Library - I2C Communication*. Disponível em: https://www.arduino.cc/reference/en/language/functions/communication/wire/. Acesso em: 01 Fev. 2025.

[2] Adafruit. *Adafruit 16-Channel Servo Driver with I2C Interface*. Disponível em: https://learn.adafruit.com/16-channel-pwm-servo-driver. Acesso em: 01 Fev. 2025.

[3] Raspberry Pi Foundation. *Raspberry Pi Pico W Datasheet*. Disponível em: https://datasheets.raspberrypi.com/picow/pico-w-datasheet.pdf. Acesso em: 01 Fev. 2025.

[4] NXP Semiconductors. *PCA9685 - 16-channel, 12-bit PWM Fm+ I2C-bus LED controller*. Disponível em: https://www.nxp.com/docs/en/data-sheet/PCA9685.pdf. Acesso em: 01 Fev. 2025.

[5] TowerPro. *SG90 Micro Servo Motor Datasheet*. Disponível em: https://datasheetspdf.com/pdf/923349/TowerPro/SG90/1. Acesso em: 01 Fev. 2025.

[6] Elecfreaks. *HC-SR04P Ultrasonic Sensor Datasheet*. Disponível em: https://www.elecrow.com/download/HC-SR04-Ultrasonic-Module-User-Guide.pdf. Acesso em: 01 Fev. 2025.

[7] Otto DIY. *Otto DIY - Open Source Biped Robot*. Disponível em: https://www.ottodiy.com/. Acesso em: 01 Fev. 2025.

[8] Github. *Bipedal Robot Projects*. Disponível em: https://github.com/topics/biped-robot. Acesso em: 01 Fev. 2025.

6 Anexos

Esta secção apresenta anexos relevantes para a documentação completa do projeto, incluindo os esquemas elétricos detalhados, o código-fonte completo e imagens da montagem e circuitos utilizados.

6.1 Código-Fonte Completo

O código-fonte utilizado para programar o robô bípede está disponível nos ficheiros abaixo. Este código inclui a implementação da máquina de estados finitos (FSM), a gestão dos servomotores e a leitura dos sensores.

6.1.1 Anexo I - `main.cpp`

```
#include <Wire.h>
#include <Adafruit_PWMServoDriver.h>
#include <Arduino.h>
#include <feet.h>

// Definição dos tempos e distância limite
#define TEMPO_PARAR 3000  // 15 segundos parado antes de tomar decisão
#define TEMPO_RODAR 5000   // Tempo a rodar antes de avançar novamente
#define DISTANCIA_LIMITE 15.0  // Distância limite para evitar colisão (cm)
#define TIME_BETWEEN_CYCLES 100 // Tempo entre ciclos da FSM

// Máquina de estados
typedef enum {
    AVANCAR,
    PARAR,
    RODAR
} state_t;

// Estrutura para armazenar o estado atual e o próximo estado
typedef struct {
    state_t state, new_state;
    unsigned long tis, tes;
} fsm_t;

Feet pes;
fsm_t state_machine;

unsigned long last_cycle = 0;
unsigned long tempo_inicio_estado = 0;

// **Converte o estado para string (para debug)**
const char* stateToString(state_t state) {
    switch (state) {
        case AVANCAR: return "AVANCAR";
        case PARAR: return "PARAR";
        case RODAR: return "RODAR";
        default: return "UNKNOWN";
    }
}
```

```
// **Define o novo estado**
void set_state(fsm_t& fsm, state_t new_state) {
    if (fsm.state != new_state) {
        Serial.println("-------------------------------------------------");
        Serial.print("Mudança de estado: "); Serial.print(stateToString(fsm.state));
        Serial.print(" → "); Serial.println(stateToString(new_state));
        Serial.println("-------------------------------------------------");

        fsm.state = new_state;
        fsm.tes = millis();
        fsm.tis = 0;
        tempo_inicio_estado = millis();  // Guarda o tempo em que entrou no estado

        if (new_state == PARAR) {
            pes.move_all_initial(); // Para todos os motores
        }
    }
}

void setup() {
    Serial.begin(9600);
    Wire.begin();
    pes.initialize_feet();
    pes.move_all_initial();
    delay(2000);

    Serial.println("Robô iniciado. Começando FSM...");

    state_machine.new_state = AVANCAR; // Começa a avançar automaticamente
    set_state(state_machine, state_machine.new_state);
}

void loop() {
    uint32_t now = millis();
    if (now - last_cycle > TIME_BETWEEN_CYCLES) {
        state_machine.tis += (now - last_cycle);
        last_cycle = now;

        // **Lê a distância do sensor**
        static float last_distance = DISTANCIA_LIMITE + 10; // Iniciar com um valor alto

        float distancia = pes.measure_distance();
        if (distancia == 0 || distancia < 2.0) {
            distancia = last_distance; // Usa a última leitura válida
        } else {
            last_distance = distancia; // Atualiza a última leitura válida
        }

        // **Verifica se os motores estão a ser chamados corretamente**
        if (state_machine.state == AVANCAR) {
```

```
    Serial.println("Executando: AVANCAR");
} else if (state_machine.state == PARAR) {
    Serial.println("Executando: PARAR");
} else if (state_machine.state == RODAR) {
    Serial.println("Executando: RODAR");
}

// **Transições de estado**
if (state_machine.state == AVANCAR) {
    if (distancia > 0 && distancia <= DISTANCIA_LIMITE) {
        state_machine.new_state = PARAR;
    }
}
else if (state_machine.state == PARAR) {
    if (distancia > DISTANCIA_LIMITE) {
        state_machine.new_state = AVANCAR;
    } else if ((millis() - tempo_inicio_estado) >= TEMPO_PARAR) { // Espera o tempo
        state_machine.new_state = RODAR;
    }
}

else if (state_machine.state == RODAR) {
        if (pes.measure_distance() > DISTANCIA_LIMITE) { // Só avança se a distância
            state_machine.new_state = AVANCAR;
        } else {
            state_machine.new_state = RODAR; // Continua a rodar até estar livre
        }
    }
}


set_state(state_machine, state_machine.new_state);

// **Execução dos estados**
switch (state_machine.state) {
    case AVANCAR:
        pes.hello_all();
        //delay(2000);
        Serial.println("Iniciando movimento para frente...");
        //pes.walk_forward(); // Move para frente

        break;

    case PARAR:
        Serial.println("Robô parou.");
        pes.move_all_initial(); // Para completamente
        break;

    case RODAR:
        Serial.println("Rodando para evitar obstáculo...");
        pes.turn_right();
```

```
                //pes.hello_all();
 // Simula rotação lateral
                break;
        }

}
```

6.1.2 Anexo II - `feet.h`

```
#ifndef FEET_H
#define FEET_H

#pragma once

//PÉS
#define L_foot 0
#define R_foot 1

#define INICIAL_R 1000
#define INICIAL_L 1225
#define MIN_R 1150
#define MIN_L 1000
#define MAX_R 800
#define MAX_L 1400
#define PCA9685_I2C_ADDRESS 0x40
#define MPU_I2C_ADRESS 0x68


//PERNAS
#define L_leg 3
#define R_leg 2

#define INICIAL_R_P 2200
#define INICIAL_L_P 1000
#define MAX_FRONT_R 1700
#define MAX_BACK_R 2400
#define MAX_FRONT_L 700
#define MAX_BACK_L 450




//OLHOS

#define TRIG_PIN 17
#define ECHO_PIN 16

#define MPU_SDA 6  // GPIO onde está o SDA
#define MPU_SCL 7  // GPIO onde está o SCL

#include <Wire.h>
#include <Adafruit_PWMServoDriver.h>
```

```
#include <Arduino.h>



class Feet
{
    public:
    // Declaração externa do PCA9685 para evitar redefinições
    Adafruit_PWMServoDriver pca9685;

    int step = 2;
    int pwm_l_atual = 0;
    int pwm_r_atual = 0;
    int pwm_l_leg_atual = 0;
    int pwm_r_leg_atual = 0;


    void initialize_feet();

    void walk_forward();
    void move_l_foot_up();
    void move_l_foot_down();
    void move_r_foot_down();
    void move_r_foot_up();

    void move_l_leg_backward();
    void move_r_leg_forward();
    void move_r_leg_backward();
    void move_l_leg_forward();

    void move_l_foot_inicial_pos();
    void move_r_foot_inicial_pos();
    void move_l_leg_inicial_pos();
    void move_r_leg_inicial_pos();
    void move_all_initial();

    int measure_distance();

    void turn_right();
    void turn_around();
    void move_l_leg_backward_full();
    void move_r_leg_backward_full();
    void move_l_leg_forward_full();
    void move_r_leg_forward_full();

    void  move_l_foot_up_full();
    void  move_l_foot_down_full();
    void  move_r_foot_up_full();
    void  move_r_foot_down_full();

    void hello_all();
```

```
    int find_ServoDriver(int addr);

};

#endif
```

6.1.3 Anexo III - `feet.cpp`

```
#include <feet.h>

Adafruit_PWMServoDriver pca9685 = Adafruit_PWMServoDriver(PCA9685_I2C_ADDRESS, Wire);



void Feet::move_l_foot_inicial_pos() {
    Serial.println("Retornando pé esquerdo à posição inicial...");
    Serial.print("Pé esquerdo atual antes de mover: "); Serial.println(pwm_l_atual);

    if (pwm_l_atual > INICIAL_L) {
        for (int i = pwm_l_atual; i >= INICIAL_L; i -= step) {
            pca9685.writeMicroseconds(L_foot, i);
        }
    } else if (pwm_l_atual < INICIAL_L) {
        for (int i = pwm_l_atual; i <= INICIAL_L; i += step) {
            pca9685.writeMicroseconds(L_foot, i);
        }
    }

    pwm_l_atual = INICIAL_L;  // Atualiza o valor correto
    Serial.print("Novo valor de pwm_l_atual: "); Serial.println(pwm_l_atual);
    Serial.println("Pé esquerdo na posição inicial.");
}

void Feet::move_r_foot_inicial_pos() {
    Serial.println("Retornando pé direito à posição inicial...");
    Serial.print("Pé direito atual: "); Serial.println(pwm_r_atual);

    if (pwm_r_atual > INICIAL_R) {
        for (int i = pwm_r_atual; i >= INICIAL_R; i -= step) {  // Correção aqui
            pca9685.writeMicroseconds(R_foot, i);
        }
    } else if (pwm_r_atual < INICIAL_R) {
        for (int i = pwm_r_atual; i <= INICIAL_R; i += step) {  // Correção aqui
            pca9685.writeMicroseconds(R_foot, i);
        }
    }
    pwm_r_atual = INICIAL_R;
    Serial.println("Pé direito na posição inicial.");
}

void Feet::move_l_leg_inicial_pos() {
```

```
    Serial.println("Retornando perna esquerda à posição inicial...");
    Serial.print("Perna esquerda atual: "); Serial.println(pwm_l_leg_atual);

    if (pwm_l_leg_atual > INICIAL_L_P) {
        for (int i = pwm_l_leg_atual; i >= INICIAL_L_P; i -= step) {
            pca9685.writeMicroseconds(L_leg, i);
        }
    } else if (pwm_l_leg_atual < INICIAL_L_P) {
        for (int i = pwm_l_leg_atual; i <= INICIAL_L_P; i += step) {
            pca9685.writeMicroseconds(L_leg, i);
        }
    }
    pwm_l_leg_atual = INICIAL_L_P;
    Serial.println("Perna esquerda na posição inicial.");
}

void Feet::move_r_leg_inicial_pos() {
    Serial.println("Retornando perna direita à posição inicial...");
    Serial.print("Perna direita atual: "); Serial.println(pwm_r_leg_atual);

    if (pwm_r_leg_atual > INICIAL_R_P) {
        for (int i = pwm_r_leg_atual; i >= INICIAL_R_P; i -= step) {
            pca9685.writeMicroseconds(R_leg, i);
        }
    } else if (pwm_r_leg_atual < INICIAL_R_P) {
        for (int i = pwm_r_leg_atual; i <= INICIAL_R_P; i += step) {
            pca9685.writeMicroseconds(R_leg, i);
        }
    }
    pwm_r_leg_atual = INICIAL_R_P;
    Serial.println("Perna direita na posição inicial.");
}

void Feet::move_all_initial() {
    Serial.println("Movendo todas as partes para a posição inicial...");
    move_l_foot_inicial_pos();
    move_r_foot_inicial_pos();
    move_l_leg_inicial_pos();
    move_r_leg_inicial_pos();
    Serial.println("Posição inicial concluída.");
}

void Feet::move_l_foot_up() {
    Serial.println("Levantando pé esquerdo...");
    Serial.print("Pé esquerdo atual antes de mover: "); Serial.println(pwm_l_atual);

    int novo_pwm = pwm_l_atual;
    while (novo_pwm <= MAX_L) {  // Corrigido: agora inclui o MAX_L corretamente
        pca9685.writeMicroseconds(L_foot, novo_pwm);
        novo_pwm += step;
    }
```

```
    pwm_l_atual = MAX_L;  // Atualiza o valor corretamente
    Serial.print("Novo valor de pwm_l_atual: "); Serial.println(pwm_l_atual);
}

void Feet::move_l_foot_down() {
    Serial.println("Baixando pé esquerdo...");

    int novo_pwm = pwm_l_atual + step;
    while (novo_pwm > MIN_L) {  // Garante que não ultrapassa o limite inferior
        pca9685.writeMicroseconds(L_foot, novo_pwm);
        //delay(10);
        novo_pwm -= step;
    }
    pwm_l_atual = MIN_L;  // Atualiza a posição atual do servo
}

void Feet::move_r_foot_down() {
    Serial.println("Baixando pé direito...");

    int novo_pwm = pwm_r_atual + step;
    while (novo_pwm < MIN_R) {
        pca9685.writeMicroseconds(R_foot, novo_pwm);
        //delay(5);
        novo_pwm += step;
    }
    pwm_r_atual = MIN_R;
}

void Feet::move_r_foot_up() {
    Serial.println("Levantando pé direito...");

    int novo_pwm = pwm_r_atual - step;
    while (novo_pwm > MAX_R) {
        pca9685.writeMicroseconds(R_foot, novo_pwm);
        //delay(5);
        novo_pwm -= step;
    }
    pwm_r_atual = MAX_R;
}

void Feet::move_l_leg_forward() {
    Serial.println("Movendo perna esquerda para frente...");

    int novo_pwm_l = pwm_l_leg_atual + step;
    while (novo_pwm_l < MAX_FRONT_L) {
        pca9685.writeMicroseconds(L_leg, novo_pwm_l);
        //delay(10);
        novo_pwm_l += step;
    }
    pwm_l_leg_atual = MAX_FRONT_L;  // Atualiza a posição da perna esquerda
```

```
}

void Feet::move_l_leg_backward() {
    Serial.println("Movendo perna esquerda para trás...");

    int novo_pwm_l = pwm_l_leg_atual - step;
    while (novo_pwm_l > MAX_BACK_L) {
        pca9685.writeMicroseconds(L_leg, novo_pwm_l);
        //delay(10);
        novo_pwm_l -= step;
    }
    pwm_l_leg_atual = MAX_BACK_L;  // Atualiza a posição da perna esquerda
}

void Feet::move_r_leg_forward() {
    Serial.println("Movendo perna direita para frente...");

    int novo_pwm_r = pwm_r_leg_atual - step;
    while (novo_pwm_r > MAX_FRONT_R) {
        pca9685.writeMicroseconds(R_leg, novo_pwm_r);
        //delay(10);
        novo_pwm_r -= step;
    }
    pwm_r_leg_atual = MAX_FRONT_R;  // Atualiza a posição da perna direita
}

void Feet::move_r_leg_backward() {
    Serial.println("Movendo perna direita para trás...");

    int novo_pwm_r = pwm_r_leg_atual + step;
    while (novo_pwm_r < MAX_BACK_R) {
        pca9685.writeMicroseconds(R_leg, novo_pwm_r);
        //delay(10);
        novo_pwm_r += step;
    }
    pwm_r_leg_atual = MAX_BACK_R;  // Atualiza a posição da perna direita
}

void Feet::walk_forward() {
    move_all_initial();
    move_l_foot_down();
    move_l_leg_forward();
    move_l_foot_inicial_pos();
    move_r_foot_up();
    move_r_leg_backward();
    move_r_foot_inicial_pos();
    move_all_initial();

}

void Feet::initialize_feet() {
```

```
    while (!find_ServoDriver(PCA9685_I2C_ADDRESS)) {
        Serial.println("No PCA9685 found ... check your connections");
        delay(200);
    }

    Serial.println("Found PCA9685");
    pca9685.begin();
    pca9685.setPWMFreq(50); // Frequência padrão para servos (50 Hz)
}

void Feet::turn_around() {
    Serial.println("Rodando 180 graus...");

    for (int i = 0; i < 4; i++) {  // 4 passos para uma volta completa
        move_r_foot_up();           // Levanta o pé direito
        move_l_leg_backward();      // Move a perna esquerda para trás
        move_r_foot_inicial_pos();  // Volta o pé direito para a posição inicial

        move_l_foot_up();           // Levanta o pé esquerdo
        move_r_leg_forward();       // Move a perna direita para frente
        move_l_foot_inicial_pos();  // Volta o pé esquerdo para a posição inicial
    }

    move_all_initial(); // Garante que os pés ficam na posição inicial após a rotação
    Serial.println("Rotação completa.");
}

int Feet::measure_distance() {
    Serial.println("Medindo distância...");

    // Garante que TRIG está baixo antes de enviar o pulso
    digitalWrite(TRIG_PIN, LOW);
    delayMicroseconds(2);

    // Envia pulso de 10µs no TRIG para ativar a medição
    digitalWrite(TRIG_PIN, HIGH);
    delayMicroseconds(10);
    digitalWrite(TRIG_PIN, LOW);

    // Mede o tempo de resposta do ECHO
    long duration = pulseIn(ECHO_PIN, HIGH);

    // Calcula a distância (tempo * velocidade do som / 2)
    int distancia = (duration * 0.0343) / 2;

    Serial.print("Distância medida: ");
    Serial.print(distancia);
    Serial.println(" cm");

    return distancia;
}
```

```
void Feet::move_l_leg_forward_full() {
    Serial.println("Movendo perna esquerda para frente...");

    int novo_pwm_l = pwm_l_leg_atual + step;
    while (novo_pwm_l < 2000) {
        pca9685.writeMicroseconds(L_leg, novo_pwm_l);
        //delay(10);
        novo_pwm_l += step;
    }
    pwm_l_leg_atual = 2000;  // Atualiza a posição da perna esquerda
}

void Feet::move_l_leg_backward_full() {
    Serial.println("Movendo perna esquerda para trás...");

    int novo_pwm_l = pwm_l_leg_atual - step;
    while (novo_pwm_l > 500) {
        pca9685.writeMicroseconds(L_leg, novo_pwm_l);
        //delay(10);
        novo_pwm_l -= step;
    }
    pwm_l_leg_atual = 500;  // Atualiza a posição da perna esquerda
}

void Feet::move_r_leg_forward_full() {
    Serial.println("Movendo perna direita para frente...");

    int novo_pwm_r = pwm_r_leg_atual - step;
    while (novo_pwm_r > 1150) {
        pca9685.writeMicroseconds(R_leg, novo_pwm_r);
        //delay(10);
        novo_pwm_r -= step;
    }
    pwm_r_leg_atual = 1150;  // Atualiza a posição da perna direita
}

void Feet::move_r_leg_backward_full() {
    Serial.println("Movendo perna direita para trás...");

    int novo_pwm_r = pwm_r_leg_atual + step;
    while (novo_pwm_r < 2600) {
        pca9685.writeMicroseconds(R_leg, novo_pwm_r);
        //delay(10);
        novo_pwm_r += step;
    }
    pwm_r_leg_atual = 2600;  // Atualiza a posição da perna direita
}

void Feet::move_l_foot_up_full() {
    Serial.println("Levantando pé esquerdo...");
```

```
    Serial.print("Pé esquerdo atual antes de mover: "); Serial.println(pwm_l_atual);

    int novo_pwm = pwm_l_atual;
    while (novo_pwm <= 1800) {  // Corrigido: agora inclui o MAX_L corretamente
        pca9685.writeMicroseconds(L_foot, novo_pwm);
        novo_pwm += step;
    }

    pwm_l_atual = 1800;  // Atualiza o valor corretamente
    Serial.print("Novo valor de pwm_l_atual: "); Serial.println(pwm_l_atual);
}

void Feet::move_l_foot_down_full() {
    Serial.println("Baixando pé esquerdo...");

    int novo_pwm = pwm_l_atual + step;
    while (novo_pwm > 800) {  // Garante que não ultrapassa o limite inferior
        pca9685.writeMicroseconds(L_foot, novo_pwm);
        //delay(10);
        novo_pwm -= step;
    }
    pwm_l_atual = 800;  // Atualiza a posição atual do servo
}

void Feet::move_r_foot_down_full() {
    Serial.println("Baixando pé direito...");

    int novo_pwm = pwm_r_atual + step;
    while (novo_pwm < 1500) {
        pca9685.writeMicroseconds(R_foot, novo_pwm);
        //delay(5);
        novo_pwm += step;
    }
    pwm_r_atual = 1500;
}

void Feet::move_r_foot_up_full() {
    Serial.println("Levantando pé direito...");

    int novo_pwm = pwm_r_atual - step;
    while (novo_pwm > 500) {
        pca9685.writeMicroseconds(R_foot, novo_pwm);
        //delay(5);
        novo_pwm -= step;
    }
    pwm_r_atual = 500;
}


void Feet::turn_right(){
```

```
    move_r_foot_up();
    move_l_foot_up();
    move_r_foot_down();
    move_l_foot_down();

    Serial.println("90 graus para a direita...");

    move_all_initial();      // Garante que os pés começam alinhados
    move_r_foot_up_full();        // Levanta o pé direito para reduzir atrito
    move_l_leg_forward_full();  // Move a perna esquerda para frente
    move_r_leg_backward_full(); // Move a perna direita para trás
    move_r_foot_inicial_pos();  // Coloca o pé direito no chão

    move_all_initial();      // Retorna à posição base

    Serial.println("90 graus completa.");




}

void Feet::hello_all() {
    Serial.println("Executando hello_all() 3 vezes...");

    for (int i = 0; i < 9; i++) {  // Repete 3 vezes
        Serial.print("Execução número: "); Serial.println(i + 1);

        move_l_foot_down_full();  // Movimento correto
        move_l_foot_inicial_pos();

        delay(500);  // Pequena pausa para evitar sobrecarga mecânica
    }

    Serial.println("Finalizado hello_all() 3 vezes.");
}

//void Feet::move_sideway(){




int Feet::find_ServoDriver(int addr)
{

  // test PCA9685 presence
  // write8(PCA9685_MODE1, MODE1_RESTART);

  Wire.beginTransmission(addr);
  Wire.write(PCA9685_MODE1);
```

```
  Wire.write(MODE1_RESTART);

  int err = Wire.endTransmission();

  return !err;

}
```