



UNIVERSIDADE FEDERAL DE MINAS GERAIS
ESCOLA DE ENGENHARIA

Relatório Final de Iniciação Científica

Kit Didático para Estudo de Controle: Pêndulo Invertido Sobre Rodas

Alunos: Gustavo Vieira Barbosa (PIBIC)
Jéssica Luana de Oliveira Brandão (ICV)
Orientador: Fernando de Oliveira Souza

Belo Horizonte - MG
2024

Agradecimentos

Ao Professor Doutor Fernando de Oliveira Souza, orientador do projeto, por seu constante apoio e disponibilidade ao longo da Iniciação Científica, sempre presente para resolver qualquer necessidade, dúvida ou indagação, pela sua amizade.

Ao Professor Doutor Leonardo Antônio Borges Tôrres, co-orientador do projeto, pelo suporte teórico, fruto de sua vasta experiência e conhecimento, mesmo passando por tribulações pessoais sempre nos alegrou com seu entusiasmo e carisma, pela sua amizade.

Ao laboratório FABLAB do Departamento de Engenharia Eletrônica da UFMG, por possibilitar a impressão dos primeiros modelos 3D nas etapas iniciais do projeto.

À FAPEMIG, CNPq e PRPq, sem o apoio financeiro das instituições de fomento a conclusão do projeto não seria possível.

Ao colega Tales Emanuel Moreira Marques, por sua contribuição em várias etapas de desenvolvimento do projeto, sempre presente para auxiliar na construção dos algoritmos empregados, como também na elaboração do controle, sem seus *insights* a completude do projeto não seria alcançável.

Resumo

O trabalho apresenta o desenvolvimento do kit do pêndulo invertido sobre rodas para estudo de controle. Inicialmente, uma breve introdução do projeto é feita para contextualizar seu desenvolvimento, em sequência são apresentados os componentes utilizados para a montagem do pêndulo, juntamente com suas especificações, o modelo de impressão 3D e as conexões elétricas. Para os testes dos componentes, códigos em arduino foram desenvolvidos para leitura da velocidade das rodas com os encoders utilizando um filtro de primeira ordem, para eventual controle dessa grandeza com um controlador PI, um filtro complementar foi aplicado à IMU para leitura do ângulo do pêndulo invertido. Uma modelagem à partir das equações de Newton é apresentada com testes para descoberta dos parâmetros juntamente com o projeto do controlador por um compensador de avanço e atraso.

Palavras-chave: Compensador de avanço e atraso, arduino, pêndulo invertido, ensino de engenharia, discretização, modelagem Newtoniana.

Abstract

The work presents the development of an wheeled inverted pendulum kit for control study. Initially, a brief introduction to the project is provided to contextualize its development. Then, the components used for the assembly of the pendulum are presented along with their specifications, the 3D printing model and the electrical connections. For component testing, Arduino codes were developed to read the wheel speed using encoders with a first-order filter for the control using a PI controller. A complementary filter was applied to the IMU for reading the angle of the inverted pendulum. A model based on Newton's equations is presented with tests to determine the parameters, along with the design of the controller using a lead-lag compensator.

Keywords: Lead-lag compensator, arduino, inverted pendulum, engineering teaching, discretization, Newtonian modeling.

Sumário

| | | |
|----------|--|-----------|
| 1 | Introdução | 7 |
| 2 | Montagem do Dicíclo Autônomo (DIA) | 8 |
| 2.1 | Lista de componentes | 8 |
| 2.1.1 | ESP32 | 8 |
| 2.1.2 | Motor DC | 9 |
| 2.1.3 | Ponte H/Regulador de tensão | 10 |
| 2.1.4 | IMU | 11 |
| 2.1.5 | Estrutura | 12 |
| 2.1.6 | Orçamento | 13 |
| 2.2 | Montagem | 14 |
| 2.3 | Ligações Elétricas | 14 |
| 2.4 | Ambiente de Programação | 16 |
| 3 | Testes | 20 |
| 3.1 | Motores | 20 |
| 3.1.1 | Encoder | 20 |
| 3.1.2 | PWM | 21 |
| 3.1.3 | Filtro na velocidade | 21 |
| 3.1.4 | Controle de velocidade | 22 |
| 3.1.5 | Implementação do Controle de velocidade: experimento | 24 |
| 3.2 | IMU | 26 |
| 3.2.1 | Acelerômetro | 26 |
| 3.2.2 | Giroscópio | 26 |
| 3.2.3 | Filtro complementar | 27 |
| 3.2.4 | Implementação do Filtro complementar: experimentos | 27 |
| 3.2.5 | Teste de Equilíbrio do DIA | 28 |
| 4 | Projeto do Controlador | 29 |
| 4.1 | Modelagem Matemática | 29 |
| 4.1.1 | Estudo de forças na roda | 30 |
| 4.1.2 | Estudo de forças no corpo | 31 |
| 4.2 | Identificação de parâmetros | 34 |
| 4.2.1 | Identificação de parâmetros: experimentos | 34 |
| 4.3 | Controlador | 37 |
| 4.3.1 | Controlador Discretizado | 41 |
| 5 | Conclusão | 43 |
| A | Código de Exemplo e Teste dos encoderes | 44 |

| | |
|--|----|
| B Código de Exemplo e Teste da IMU | 47 |
| C Código de modelagem de motor | 49 |
| D Código de controle da velocidade do motor | 52 |
| E Código Teste de Equilíbrio do DIA | 56 |
| F Código Experimento de Modelagem | 65 |
| G Código do Controlador Compensador de Avanço e Atraso | 79 |
| Referências | 92 |

Listas de Tabelas

| | |
|--|----|
| 1 Componentes principais e suas respectivas quantidades. | 8 |
| 2 Considerações sobre os pinos do ESP32. | 9 |
| 3 Conexões do motor DC. | 10 |
| 4 Conexões da ponte H Dupla L298N. | 11 |
| 5 Conexões do sensor MPU-6050. | 12 |
| 6 Orçamento médio do díscio autônomo. | 13 |
| 7 Ligações entre os componentes. | 15 |

Listas de Figuras

| | |
|---|----|
| 1 Módulo WiFi ESP32 Bluetooth com 30 pinos. | 8 |
| 2 Mini motor DC. | 9 |
| 3 Ponte H Dupla L298N. | 10 |
| 4 Principais pinos de conexão da Ponte H. | 11 |
| 5 Sensor MPU-6050 acelerômetro e giroscópio. | 12 |
| 6 Modelo da base do DIA. | 13 |
| 7 Vistas do díscio autônomo com as posições dos componentes. | 14 |
| 8 Ligações elétricas entre os componentes. | 16 |
| 9 Interface do Arduino IDE. | 17 |
| 10 Abrindo o exemplo Blink no Arduino IDE. | 19 |
| 11 Na parte superior temos a curva de velocidade do motor e na parte inferior, a entrada de PWM correspondente. | 23 |
| 12 Curva da velocidade do motor com um ponto em 63.21% do valor final. | 23 |
| 13 Resposta ao degrau do sistema controlado. | 24 |

| | | |
|----|---|----|
| 14 | O gráfico à direita mostra a resposta do motor a múltiplas referências e o gráfico à esquerda é um zoom na resposta do motor à terceira referência. | 25 |
| 15 | Eixos da MPU-6050. | 27 |
| 16 | Funcionamento do arco tangente. | 27 |
| 17 | Definição dos eixos e coordenadas iniciais. | 29 |
| 18 | Decomposição de forças no conjunto completo, no corpo e em uma roda. | 30 |
| 19 | Estudo das forças em uma roda. | 31 |
| 20 | Estudo das forças no corpo. | 32 |
| 21 | Decomposição das forças 2P e 2H nos eixos de inclinação do corpo. | 33 |
| 22 | Captura de tela do CoolTerm. | 35 |
| 23 | Mapeamento de zeros e polos da planta de θ . | 37 |
| 24 | Lugar das raízes da planta de θ em malha fechada. | 39 |
| 25 | Mapeamento de zeros e polos da planta de ϕ . | 39 |
| 26 | Lugar das raízes da planta de ϕ em malha fechada. | 40 |

1 Introdução

O projeto “Kit Didático para Estudo de Controle: Pêndulo Invertido Sobre Rodas” surgiu como forma de criar uma planta “*Do it yourself*” relativamente barata e que permitisse o ensino de engenharia de controle e com o avanço do projeto, o objetivo inicial se manteve, mas agora o kit incorporaria o Laboratório de Controle e Automação do Departamento de Engenharia Eletrônica (DELT).

Em coordenação com os professores Fernando Oliveira Souza e Leonardo Antônio Borges Tôrres o dicíclo autônomo foi desenvolvido com o uso de impressões 3D, itens conhecidos de kits de arduino e o microcontrolador ESP32. Ao utilizar esses itens, a ideia central é permitir que o aluno construa sua planta com os conhecimentos adquiridos ao longo do curso e consiga colocar em prática as teorias de controle com a modificação do *software* no ESP32.

Dessa forma, esse relatório apresenta ao longo de seus capítulos todas as partes necessárias para montar e colocar o dicíclo para rodar de forma funcional. O capítulo 2 apresenta os componentes, a montagem, as ligações elétricas e o ambiente de programação arduino, o capítulo 3 demonstra os testes que precisam ser realizados com os componentes para a identificação do sistema, o capítulo 4 mostra a modelagem matemática da planta juntamente com o projeto do controlador e por fim, o capítulo 5 faz uma breve conclusão do projeto.

2 Montagem do Dicíclo Autônomo (DIA)

Este capítulo apresenta os componentes da montagem do dicíclo autônomo e sua estrutura juntamente com uma breve descrição e o orçamento médio total. Posteriormente, o esquema de montagem e o esquema elétrico de conexão dos componentes são comentados e por fim, o ambiente de programação “Arduino IDE” (utilizado para os testes e controle) é introduzido.

2.1 Lista de componentes

Para a montagem do dicíclo autônomo foram utilizados componentes de fácil acesso e amplamente difundidos em projetos de arduino. A Tabela 1 mostra os componentes e suas respectivas quantidades de forma simplificada.

| Quantidade | Componente |
|------------|--------------------------|
| 1 | ESP32 |
| 2 | Motor DC |
| 1 | Ponte H |
| 1 | IMU |
| 1 | Suporte de baterias |
| 2 | Baterias de lítio 3,7V |
| 1 | Botão chave On/Off |
| 1 | Base do dicíclo autônomo |

Tabela 1: Componentes principais e suas respectivas quantidades.

2.1.1 ESP32

O ESP32 é um microcontrolador de baixo custo e alto desempenho com Wi-Fi, memória Flash e Bluetooth integrados. O ESP32 usado neste projeto possui 30 pinos de entrada e saída, algumas considerações sobre cada pino está disponível na Tabela 2. O software Arduino IDE foi usado para programar o microcontrolador.

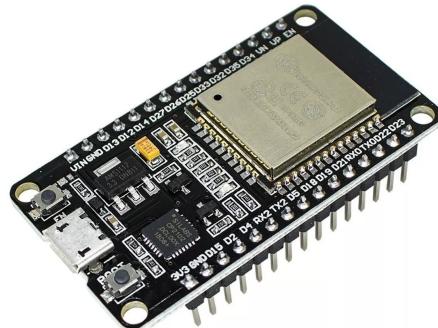


Figura 1: Módulo WiFi ESP32 Bluetooth com 30 pinos.

| Label | GPIO | Use? | Motivo |
|-----------|---------|------|--|
| D0 | 0 | ! | Deve ser HIGH durante boot e LOW para programar |
| TX0 | 1 | X | Tx pin, usado para flashing e debugging |
| D2 | 2 | ! | Deve ser LOW durante boot e é conectado no LED |
| RX0 | 3 | X | Rx pin, usado para flashing e debugging |
| D4 | 4 | ✓ | |
| D5 | 5 | ! | Deve ser HIGH durante boot |
| D6 - D11 | 6 - 11 | X | Conectado na memoria Flash |
| D12 | 12 | ! | Deve ser LOW durante boot |
| D13 | 13 | ✓ | |
| D14 | 14 | ✓ | |
| D15 | 15 | ! | Deve ser HIGH durante boot, impede o log de inicialização se estiver LOW |
| RX2 | 16 | ✓ | |
| TX2 | 17 | ✓ | |
| D18 - D33 | 18 - 33 | ✓ | Exceto 20, 24, 28, 29, 30, e 31 |
| D34 - D35 | 34 - 35 | ! | Apenas entrada, não pode ser configurado como saída |
| VP | 36 | ! | Apenas entrada, não pode ser configurado como saída |
| VN | 39 | ! | Apenas entrada, não pode ser configurado como saída |

Tabela 2: Considerações sobre os pinos do ESP32.

2.1.2 Motor DC

Um dicíclo possui duas rodas alinhadas pelo eixo e para o acionamento de cada uma é necessário utilizar motores com uma boa relação de velocidade e torque. O motor escolhido para o projeto foi o mini motor DC com caixa de redução 6V 500RPM com encoder.

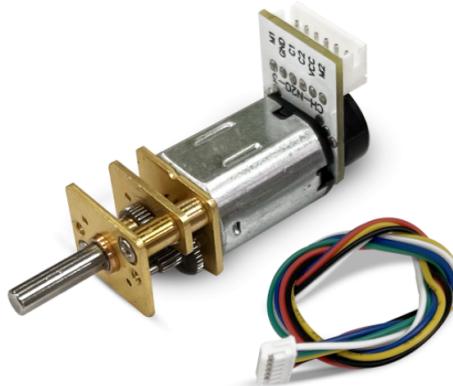


Figura 2: Mini motor DC.

O motor mostrado na Figura 2 tem um encoder já acoplado em seu eixo com leitura do sentido de rotação do motor, isso permite medições mais precisas das interrupções nos códigos de arduino, facilitando o controle de velocidade e consequentemente o de equilíbrio.

A Tabela 3 apresenta os pinos de conexão do motor e suas funções. A combinação de leitura de C1 e C2 permite o cálculo da velocidade do motor e seu sentido de rotação e a combinação de M1 e M2 permite determinar a velocidade do motor e seu sentido de rotação.

| Label | Função |
|-------|------------------------------------|
| M1 | Sinal 1 para movimentação do motor |
| GND | Aterrramento do encoder do motor |
| C1 | Feedback + do encoder |
| C2 | Feedback - do encoder |
| VCC | Alimentação do encoder do motor |
| M2 | Sinal 2 para movimentação do motor |

Tabela 3: Conexões do motor DC.

2.1.3 Ponte H/Regulador de tensão

Para acionar os mini motores DC de forma simples com a variação do PWM é preciso utilizar um regulador de tensão. Um regulador de tensão possui um pino denominado *enable* que ao receber um sinal, envia tensão para o motor e ao utilizar pinos de PWM no ESP32 é possível selecionar o *duty cycle* desejado de 0 a 100%, determinando a porcentagem em um período de tempo que o pino *enable* fica energizado, regulando dessa forma o valor de tensão recebido pelo motor.

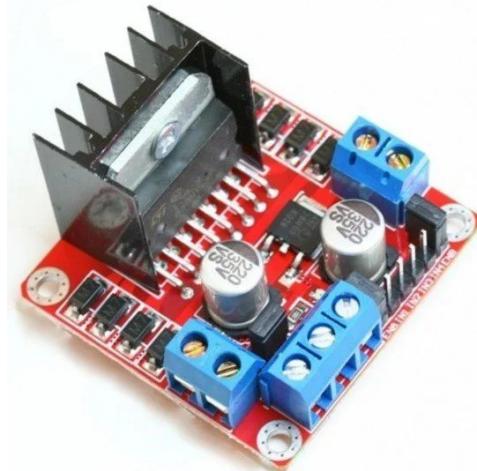


Figura 3: Ponte H Dupla L298N.

A Figura 3 mostra a Ponte H L298N amplamente utilizada em projetos de arduino e comumente encontrada em lojas de componentes eletrônicos. A tabela 4 mostra os pinos de conexão da ponte H e suas respectivas funções.

| Label | Função |
|-------|--|
| OUT1 | Sinal 1 para o motor A |
| OUT2 | Sinal 2 para o motor A |
| OUT3 | Sinal 1 para o motor B |
| OUT4 | Sinal 2 para o motor B |
| 12V | Alimentação para os motores |
| GND | Aterrramento da placa |
| 5V | Quando 5VEN está com jumper possui saída de 5V |
| 5VEN | Enable da alimentação do chip da ponte H |
| ENA | Ativação do motor A |
| ENB | Ativação do motor B |
| IN1 | Controle 1 da saída do motor A |
| IN2 | Controle 2 da saída do motor A |
| IN3 | Controle 1 da saída do motor B |
| IN4 | Controle 2 da saída do motor B |

Tabela 4: Conexões da ponte H Dupla L298N.

Nesse projeto, os terminais do motor A são conectados à OUT1 e OUT2 e os terminais do motor B à OUT3 e OUT4, a alimentação 12V vem das baterias, o pino 5VEN possui jumper, os pinos ENA e ENB **não** possuem jumper e juntamente com IN1, IN2, IN3 e IN4 são conectados ao ESP32 para controle da velocidade e sentido de rotação dos motores. A Figura 4 mostra as posições dos principais pinos de conexão da ponte H.

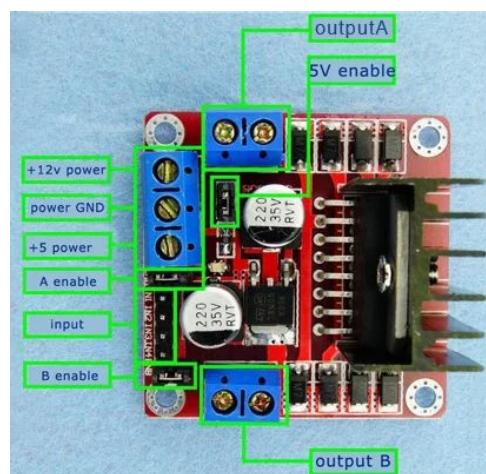


Figura 4: Principais pinos de conexão da Ponte H.

2.1.4 IMU

A unidade de medição inercial (*Inertial Measurement Unit*) é um componente comumente empregado em projetos de arduino e consiste em um acelerômetro e um giroscópio embutidos em uma placa de circuito impresso compacta.

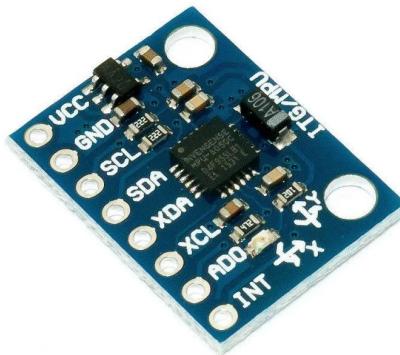


Figura 5: Sensor MPU-6050 acelerômetro e giroscópio.

O sensor MPU-6050 mostrado na Figura 5 foi escolhido para ser utilizado no projeto por sua facilidade de compra e simplicidade de uso. A tabela 5 mostra os pinos de conexão da ponte H e suas respectivas funções

| Label | Função |
|-------|---|
| VCC | Alimentação do sensor |
| GND | Aterramento do sensor |
| SCL | Pin de clock para interface I2C |
| SDA | Pino de dados para a interface I2C |
| XDA | Pino de dados para sensores externos |
| XCL | Pino de clock para sensores externos |
| ADO | Mudança de endereço para leitura de dados |
| INT | Pino de interrupção de leitura |

Tabela 5: Conexões do sensor MPU-6050.

Vale ressaltar que apenas os pinos VCC, GND, SCL e SDA são utilizados no projeto, porque só precisamos da tensão de alimentação da placa e seu aterramento e dos pinos para leitura dos dados do acelerômetro e giroscópio.

2.1.5 Estrutura

Uma base foi desenvolvida através de softwares de criação de modelos 3D para impressão com o objetivo de alocar os componentes da melhor forma possível no dicíclo autônomo. A figura 6 mostra a base do DIA com os encaixes necessários para cada componente utilizado. Vale pontuar que o arquivo .stl está disponível no repositório https://github.com/JessicaLuana1377/diciclo_autonomo.

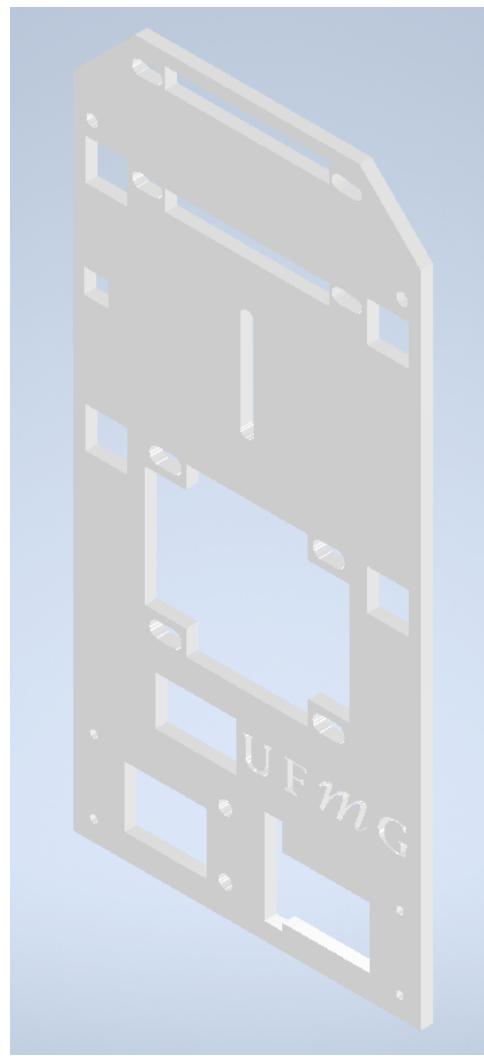


Figura 6: Modelo da base do DIA.

2.1.6 Orçamento

Um orçamento médio pode ser montado considerando todos os componentes apresentados e suas respectivas quantidades levando em conta os preços médios de mercado no Aliexpress.

| Componente | Preço Médio |
|---------------------|-------------|
| ESP32 | R\$22,00 |
| 2 motores DC | R\$53,00 |
| Ponte H | R\$15,00 |
| IMU | R\$16,00 |
| Suporte de baterias | R\$6,00 |
| 2 Baterias | R\$17,00 |
| Botão chave On/Off | R\$2,00 |
| Total | R\$131,00 |

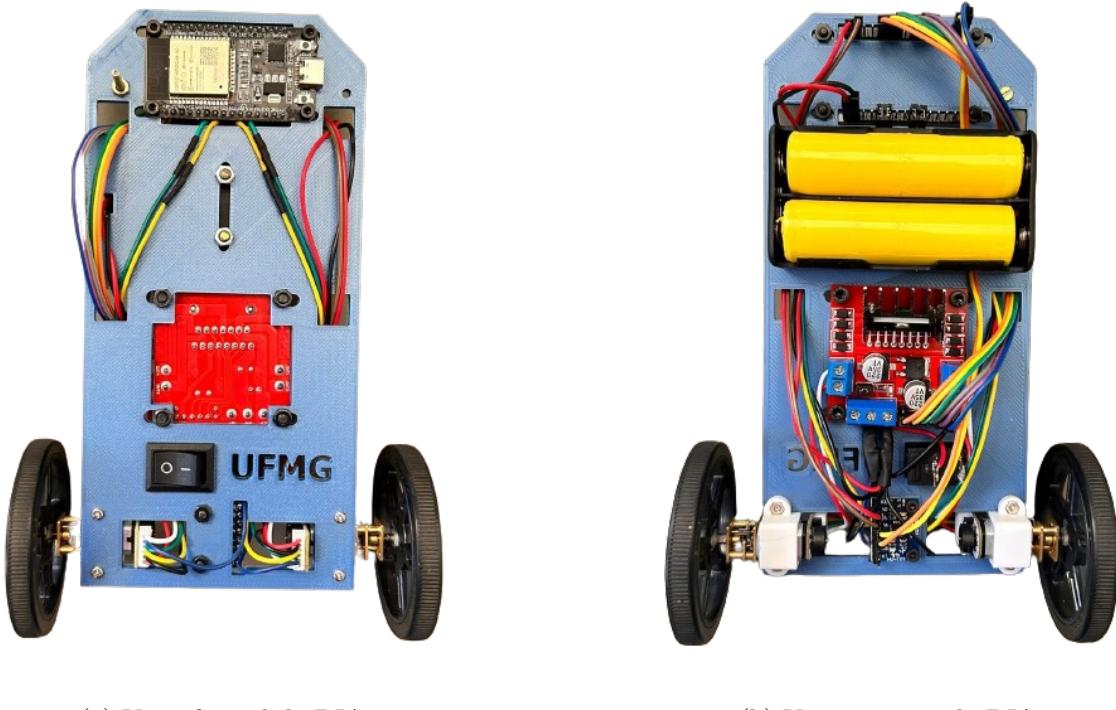
Tabela 6: Orçamento médio do dicíclo autônomo.

Como apresentado pela Tabela 6, o orçamento médio total para montar o dicíclo

autônomo é de R\$131,00, o componente mais caro é o motor DC, por causa da sua qualidade e facilidade de uso, o valor total é razoável considerando seu alto potencial em agregar conhecimento nas áreas de identificação, linearização e controle de sistemas. Vale ressaltar também que o preço dos componentes eletrônicos sempre varia e que nesse caso foram considerados os melhores preços na época de pesquisa.

2.2 Montagem

A base principal do dicíclo autônomo foi concebida para alocar os componentes da melhor forma possível, facilitando as conexões dos cabos. As Figuras 7a e 7b mostram a vista frontal e traseira do dicíclo autônomo, respectivamente.



(a) Vista frontal do DIA.

(b) Vista traseira do DIA.

Figura 7: Vistas do diciclo autônomo com as posições dos componentes.

Observando as vistas frontais, fica claro as posições de cada componente na base do diciclo autônomo e a quantidade de cabos que são utilizados para interligá-los (vale ressaltar que as cores dos cabos não são iguais aos apresentados na Figura 8).

2.3 Ligações Elétricas

As ligações elétricas devem ser realizadas conforme mostrado na Figura 8. A Tabela 7 apresenta os mesmos dados da Figura 8, destacando o nome de cada pino ou terminal dos componentes e suas respectivas conexões.

Um ponto de atenção ao efetuar as conexões elétricas é o sentido de rotação dos motores e de leitura dos encoders. Para verificar a correta conexão dos pinos M1 e M2

| Pino | Componente | Coneção | Componente |
|------|------------------|-------------------|----------------|
| C1 | Motor direita | GPIO 32 (D32) | ESP32 |
| C2 | Motor direita | GPIO 33 (D33) | ESP32 |
| C1 | Motor esquerda | GPIO 14 (D14) | ESP32 |
| C2 | Motor esquerda | GPIO 27 (D27) | ESP32 |
| VCC | MPU-6050 | 3.3V | ESP32 |
| GND | MPU-6050 | GND | ESP32 |
| SCL | MPU-6050 | GPIO 22 (D22) | ESP32 |
| SDA | MPU-6050 | GPIO 21 (D21) | ESP32 |
| 5V | Ponte H L298N | Vin | ESP32 |
| GND | Ponte H L298N | GND | ESP32 |
| ENA | Ponte H L298N | GPIO 4 (D4) | ESP32 |
| IN1 | Ponte H L298N | GPIO 16 (RX2) | ESP32 |
| IN2 | Ponte H L298N | GPIO 17 (TX2) | ESP32 |
| IN3 | Ponte H L298N | GPIO 18 (D18) | ESP32 |
| IN4 | Ponte H L298N | GPIO 19 (D19) | ESP32 |
| ENB | Ponte H L298N | GPIO 23 (D23) | ESP32 |
| 5V | Ponte H L298N | VCC | Motor direita |
| GND | Ponte H L298N | GND | Motor direita |
| OUT1 | Ponte H L298N | M1 | Motor direita |
| OUT2 | Ponte H L298N | M2 | Motor direita |
| OUT3 | Ponte H L298N | M1 | Motor esquerda |
| OUT4 | Ponte H L298N | M2 | Motor esquerda |
| 5V | Ponte H L298N | VCC | Motor esquerda |
| GND | Ponte H L298N | GND | Motor esquerda |
| 12V | Ponte H L298N | Qualquer terminal | Chave Gangorra |
| VCC | Suporte de pilha | Qualquer terminal | Chave Gangorra |
| GND | Suporte de pilha | GND | Ponte H L298N |

Tabela 7: Ligações entre os componentes.

dos motores, aplique um sinal positivo e observe se a rotação segue a regra da mão direita: com o polegar alinhado ao motor apontando no sentido do eixo, a roda deve girar na direção dos outros dedos da mão. Se a rotação ocorrer no sentido contrário, inverta as conexões dos pinos M1 e M2 no ESP32. Lembre-se de que os motores devem girar em sentidos opostos quando o PWM é positivo.

Para verificar se a conexão dos encoders está correta, consulte a seção 3.1.1.

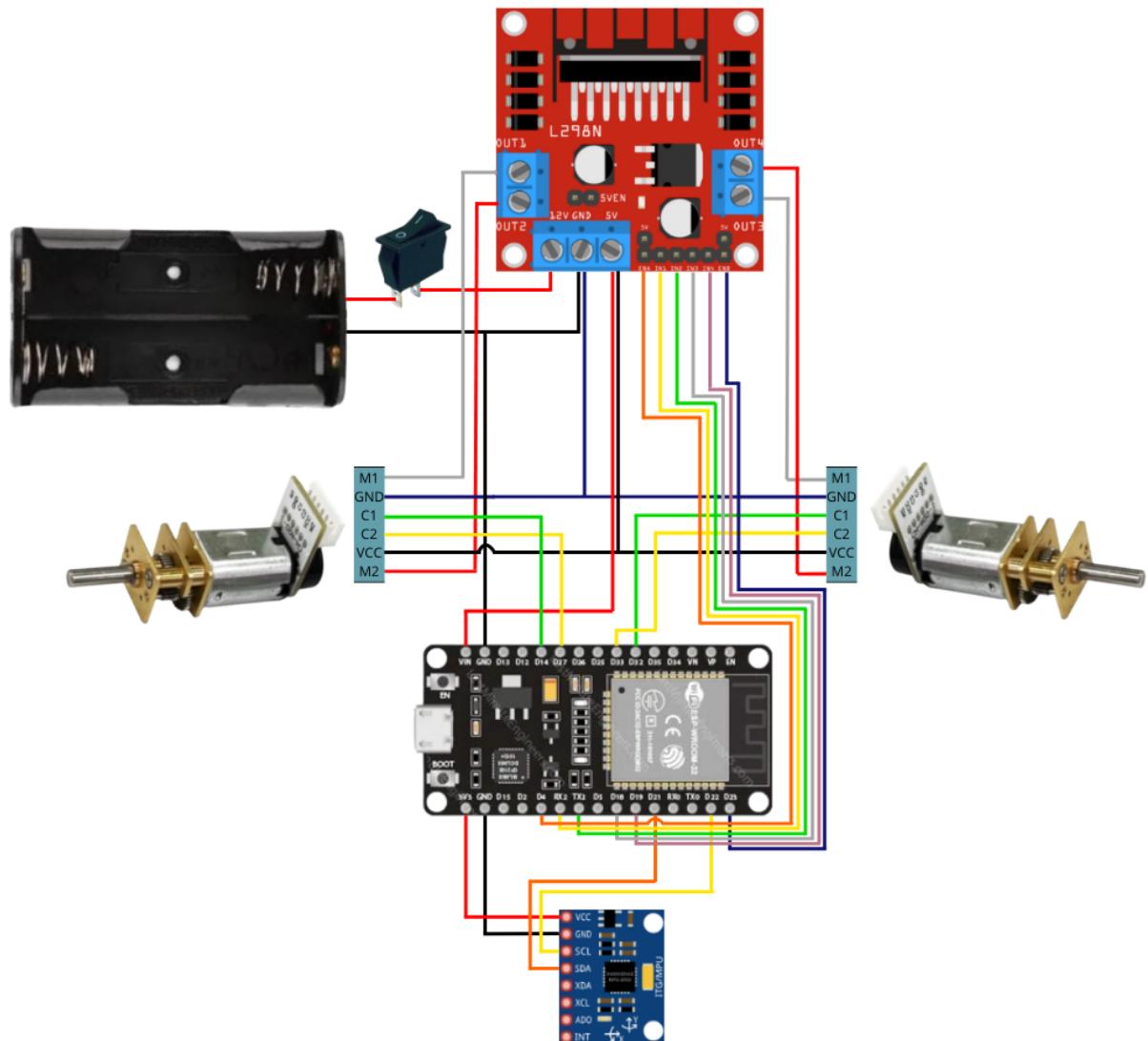


Figura 8: Ligações elétricas entre os componentes.

2.4 Ambiente de Programação

Arduino IDE (Integrated Development Environment) é um software de código aberto que facilita a programação de placas de microcontroladores Arduino e compatíveis. Ele oferece uma interface amigável onde você pode escrever, compilar e carregar códigos diretamente para a placa.

Instalação do Arduino IDE:

1. Acesse o site oficial do Arduino (<https://www.arduino.cc/en/software>) e baixe a versão do Arduino IDE correspondente ao seu sistema operacional;



Figura 9: Interface do Arduino IDE.

2. Siga as instruções de instalação fornecidas pelo site;
3. Após a instalação, abra o Arduino IDE. A Figura 9 mostra a interface do Arduino IDE

Em alguns casos, o computador não reconhece o ESP32, portanto, é necessário instalar um driver que comunique com o ESP32.

Instalação do driver do ESP32:

1. Acesse o site oficial do Silicon labs (<https://www.silabs.com/developers/usb-to-uart-bridge-vcp-drivers>) na aba downloads baixe o arquivo CP210x Windows Drivers;
2. Descompacte a pasta;
3. Instale o driver pelo arquivo CP210xVCPInstaller_x64.exe.

Uma vez terminada a instalação do Arduino IDE e, se necessário, a instalação do driver do ESP32, é necessário instalar a placa do ESP32 no Arduino IDE.

Instalação da placa:

1. Vá em Tools > Board > Boards Manager..., ou acesse pelo segundo ícone disponibilizado no menu à esquerda;
2. Pesquise por esp32 by Espressif Systems versão 2.0.17 e clique em “Instalar”.

Após realizar todos os preparativos necessários para o seu projeto, é importante testar se tudo está funcionando corretamente. Uma maneira eficaz de fazer isso é programando o ESP32 com um exemplo do Arduino IDE.

O Arduino IDE (Integrated Development Environment) oferece uma variedade de exemplos de código que podem ser usados para verificar a funcionalidade básica do seu hardware. Por exemplo, o código de "Blink" pode ser utilizado para testar se a conexão e a comunicação entre o ESP32 e o seu computador estão operando conforme o esperado. Este exemplo básico faz um LED piscar em intervalos regulares, permitindo que você confirme rapidamente se o ESP32 está programado e funcionando corretamente. Isso ajuda a garantir que a configuração básica está correta antes de avançar no projeto.

Para verificar o funcionamento correto do programa, siga os passos abaixo:

1. Abra o Arduino IDE;
2. Selecione o Exemplo Blink:
 - Vá até a aba File (Arquivo) > Examples (Exemplos) > 0.1Basics > Blink.
Como mostra a Figura 10.
3. Configure a Placa e a Porta:
 - Em Tools (Ferramentas), selecione Board (Placa) > ESP32 Dev Module;
 - Em Port (Porta), selecione a porta à qual o ESP32 está conectado. Se houver várias opções, desconecte o ESP32 e veja qual opção desaparece para identificar a correta.
4. Ajuste o Código;
 - No código do exemplo, substitua 'LED_BUILTIN' por '2'.
5. Faça o Upload do Código:
 - Clique na seta de upload no canto superior esquerdo do Arduino IDE para carregar o código na placa ESP32.
 - Após a compilação do código, o software tentará se conectar ao ESP32. Em alguns casos, é necessário manter pressionado o botão BOOT do ESP32 até que a conexão seja estabelecida.

Este projeto também faz uso de algumas bibliotecas disponíveis no Arduino IDE. Para que o código compile, é necessário instalar essas bibliotecas.

Instalação das bibliotecas:

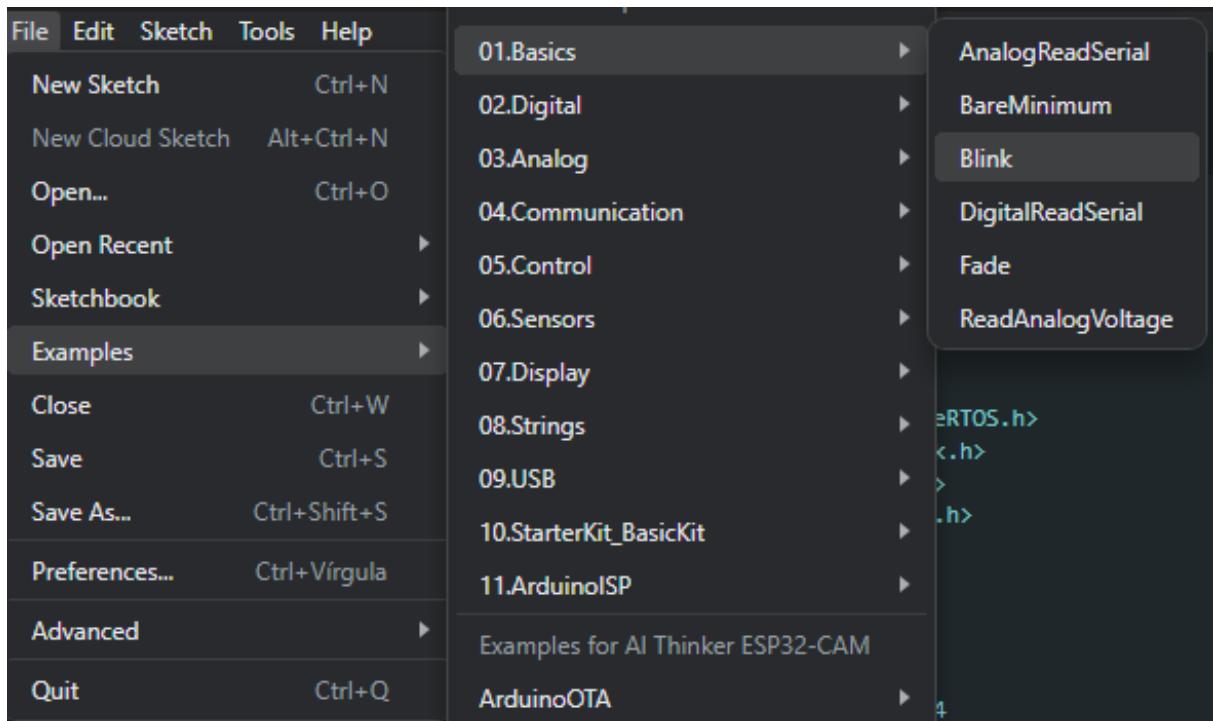


Figura 10: Abrindo o exemplo Blink no Arduino IDE.

1. Vá em Sketch > Include Library > Manage Libraries, ou acesse pelo terceiro ícone disponibilizado no menu à esquerda;
2. Pesquise por ESP32Encoder by Kevin Harrington versão 0.11.6 e clique em “Instalar”.
3. Ainda no gerenciador de bibliotecas, pesquise por Adafruit MPU6050 by Adafruit versão 2.2.6, clique em “Instalar” e aceite todas as dependências (Adafruit BusIO, Adafruit GFX Library, Adafruit SSD1306, Adafruit Unified Sensor);

Além do Arduino IDE, o programa CoolTerm foi usado para capturar os dados da serial. Este programa não precisa ser instalado, apenas é necessário baixar a pasta “CoolTermWin64Bit.zip” disponibilizada no repositório https://github.com/JessicaLuana1377/diciclo_autonomo, descompactar e executar o arquivo “CoolTerm.exe”.

3 Testes

Esta seção descreve os testes necessários para o funcionamento dos componentes básicos, primeiramente é apresentada a configuração da biblioteca para leitura do encoder e o uso dos comandos para PWM que serão aplicados nos motores, posteriormente o filtro para leitura da velocidade do motor é apresentado, juntamente com um teste simples para o controle dessa grandeza e sua implementação em um código para arduino. Por fim, a calibração da IMU e a implementação do filtro complementar para leitura do ângulo são demonstrados e um teste para o equilíbrio do DIA é realizado.

3.1 Motores

Para controle da velocidade dos motores as configurações para leitura do sinal do encoder e envio do PWM são apresentados, juntamente com um controle de velocidade e sua implementação em código.

3.1.1 Encoder

No projeto, é usado um encoder com duas leituras defasadas para monitorar a direção de rotação de cada roda. A leitura do encoder é realizada com o auxílio da biblioteca ESP32Encoder, que utiliza os contadores internos do ESP32 para realizar a contagem dos pulsos e métodos para acessar os valores do contador. Configurando o encoder com a função attachFullQuad, é possível detectar mudanças nos sinais A e B, permitindo determinar se a roda está girando para a frente ou para trás.

Como usar a biblioteca ESP32Encoder:

1. Declare uma variável ESP32Encoder;
2. Use “ESP32Encoder::useInternalWeakPullResistors = puType::up;” na função “setup” para configurar a biblioteca. Essa configuração deve estar presente no código apenas uma vez;
3. Use o método attachFullQuad(Pin A, Pin B) para configurar os dois pinos do encoder conectados no ESP32;
4. Após configurar os pinos, use o método clearCount() para zerar o contador;
5. Por fim, use o método getCount() para obter a contagem atual do encoder.

No Anexo A está disponível um código de exemplo e teste. Esse código aplica um PWM de 10% no motor esquerdo, 100% no motor direito, lê a posição atual da roda e imprime os valores no console a cada 500 milissegundos. Se a montagem estiver correta, ambos os valores dos encoders devem aumentar, caso algum esteja errado, inverta os

pinos do C1 e do C2 do encoder correspondente no ESP32. É possível comentar as linhas de código que aplicam o PWM nos motores e testar os encoderes individualmente.

3.1.2 PWM

A velocidade do motor é controlada usando PWM. Ao variar a largura dos pulsos de energia enviados ao motor é possível simular uma tensão DC menor que a tensão alternada realmente enviada para os motores. O ciclo de trabalho (duty cycle) do PWM é a proporção do tempo em que o sinal está em nível alto (ligado) em relação ao tempo total do ciclo. O PWM costuma ser representado em porcentagem, por exemplo, 100% de PWM significa que o sinal de tensão alternada (7,4 V) enviado para os motores é 100% do tempo alta, já um PWM de 60% significa que o sinal enviado fica 60% do tempo alto e 40% baixo. Na prática, o que acontece é que o motor filtra a frequência do PWM e funciona como se estivesse recebendo uma tensão DC, nos dois casos citados acima, o motor recebe uma tensão DC de 7,4 V e de 4,44 V, respectivamente.

O ESP32 possui 16 canais de PWM que necessitam ser configurados para serem usados.

Configuração do PWM no ESP32:

1. Use a função `ledcSetup(X_PWM_CHANNEL, X_PWM_FREQ, X_PWM_RESOLUTION)` para configurar um dos canais do ESP32, onde:

X_PWM_CHANNEL corresponde ao canal do ESP32 que será usado;

X_PWM_FREQ é a frequência do PWM;

X_PWM_RESOLUTION é a resolução do PWM em bits.

2. Use a função `ledcAttachPin(X_PWM_OUT, X_PWM_CHANNEL)` para associar um pino do ESP32 a um canal de PWM, onde:

X_PWM_OUT é um pino do ESP32;

X_PWM_CHANNEL é o canal de PWM.

3. Use a função `ledcWrite(X_PWM_CHANNEL, X_PWM)` para escrever um valor de PWM no canal, onde:

X_PWM_CHANNEL é o canal de PWM;

X_PWM é o valor do PWM.

3.1.3 Filtro na velocidade

As medidas de velocidade costumam ser ruidosas devido a folgas e vibrações. Para filtrar estes ruídos e suavizar a leitura de velocidade é aplicado um filtro de primeira ordem no formato

$$v_{filtrado}[k] = \alpha v_{filtrado}[k - 1] + (1 - \alpha)v[k],$$

onde v é a velocidade lida, $v_{filtrado}$ é a velocidade filtrada, e k é o índice que representa os valores atuais (k) e os valores passados ($k - 1$). Além disso, há a constante de peso α que pondera entre manter o valor lido (α próximo de 1) ou responda mais rapidamente as alterações em v (α próximo a 0). Um valor típico para α utilizado neste projeto é de 0.9, ou seja, 90%.

3.1.4 Controle de velocidade

Um motor pode ser representado por um sistema de segunda ordem, onde um polo é associado à indutância do motor e o outro à sua inércia. Entretanto, a dinâmica da indutância é muito rápida, por isto ela é desconsiderada em muitos casos práticos. Assim, o modelo do motor é frequentemente aproximado por um sistema de primeira ordem:

$$G = \frac{K}{\tau s + 1}; \quad (1)$$

onde K é o ganho e τ é a constante de tempo.

O experimento para obter o modelo do motor consiste em aplicar um degrau de PWM no motor, com este parado, e esperar estabilizar. O ganho do motor corresponde ao valor final da velocidade estabilizada dividida pelo valor de PWM aplicado e a constante de tempo corresponde ao tempo decorrido entre o momento em que o degrau foi aplicado e o momento em que o motor alcançou 63.21% da velocidade final.

A Figura 11 exemplifica o gráfico a ser obtido aplicando 100% de PWM no início da simulação e -100% após 0,4 segundos. No Anexo C está disponível o código deste experimento.

É possível observar na Figura 12 que o motor estabiliza em, aproximadamente, 55 rad/s. 63.21% desse valor corresponde a 34.77 rad/s e o tempo para alcançar esse valor é de, aproximadamente, 0.05 s. Como o PWM aplicado é 100%, $K = 0.55$ e $\tau = 0.05$. O modelo do motor pode ser representado pela equação (1) com os dados calculados, entretanto, para obter um modelo mais preciso foi usada a função *tfest* do software MATLAB:

$$G = \frac{0.5536}{0.0463s + 1},$$

ou, dividindo todos os termos por 0.0463, temos:

$$G = \frac{11.91}{s + 21.6}.$$

Como o sistema é de primeira ordem, para obter uma resposta controlada com erro nulo, apenas é necessário um PI. Várias técnicas podem ser empregadas para obter um bom controlador, entretanto, o controlador desenvolvido nesta seção foi obtido de forma experimental. A função transferência de um PI é:

$$C = K \left(\frac{s + z}{s} \right), \quad (2)$$

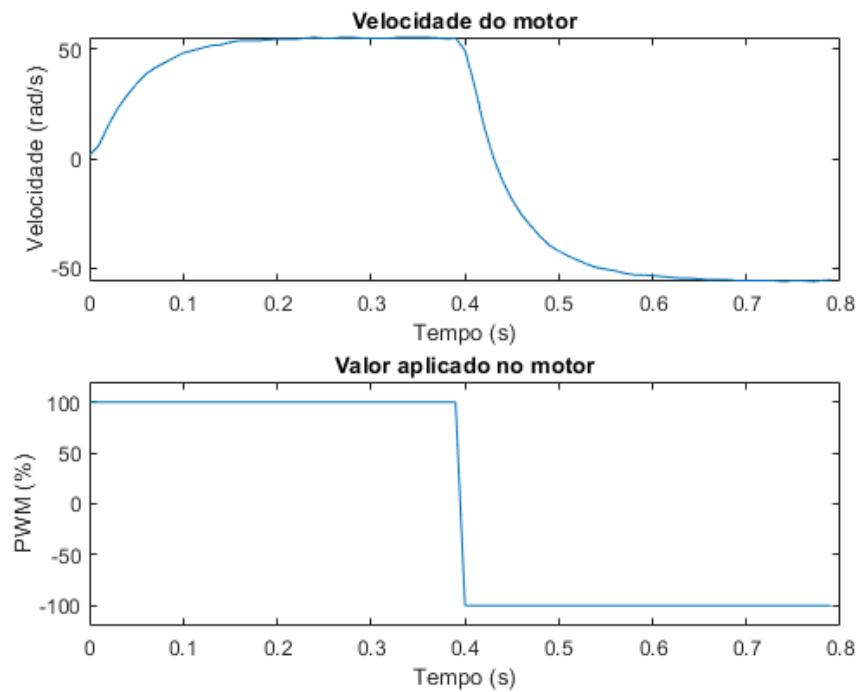


Figura 11: Na parte superior temos a curva de velocidade do motor e na parte inferior, a entrada de PWM correspondente.

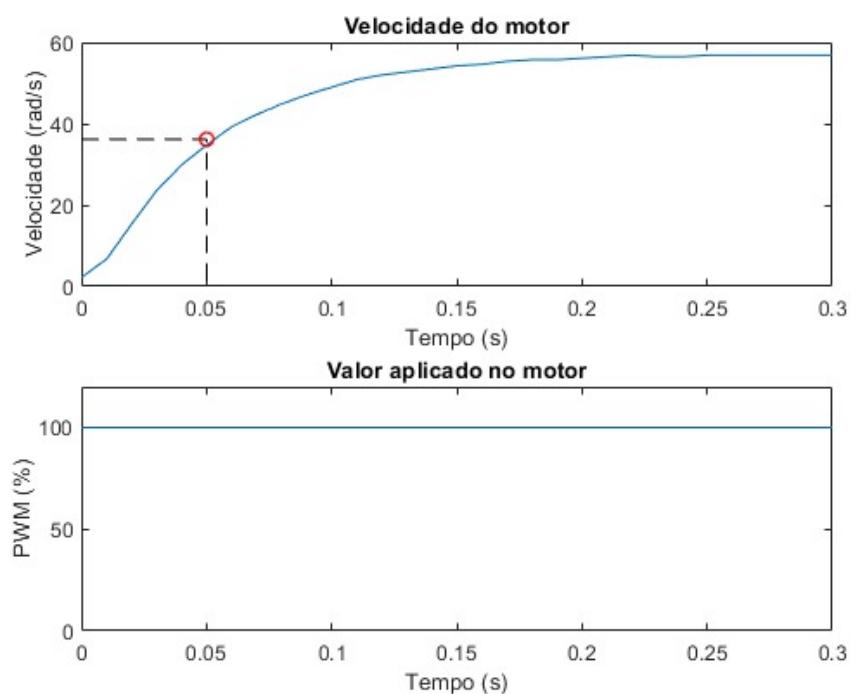


Figura 12: Curva da velocidade do motor com um ponto em 63.21% do valor final.

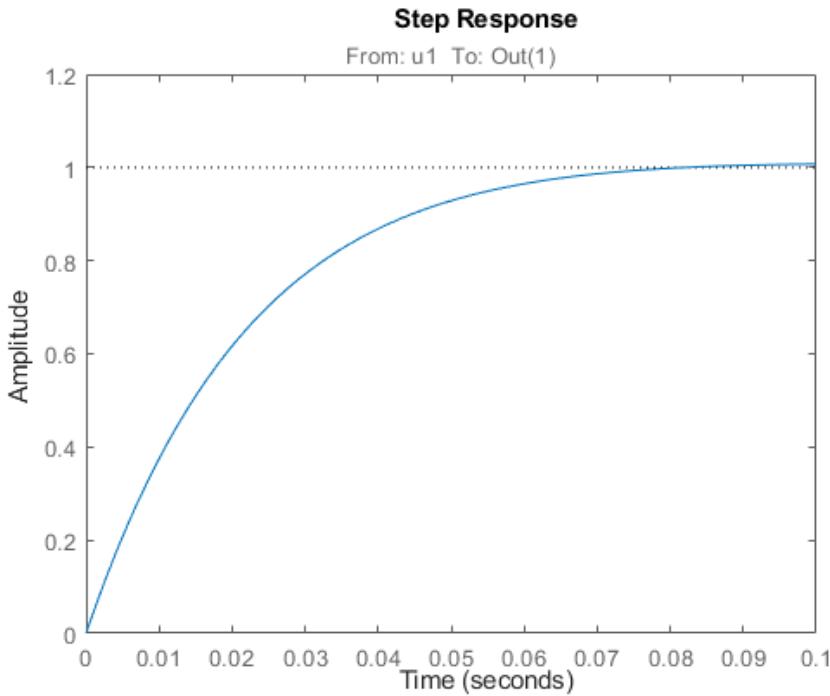


Figura 13: Resposta ao degrau do sistema controlado.

onde K é um ganho e z é o zero do controlador. O zero foi escolhido como sendo 25, maior que o polo do modelo, e o ganho foi escolhido de forma que os polos em malha fechada do sistema sejam mais rápidos que o polo em malha aberta do motor e reais, para evitar oscilações, neste caso, o ganho corresponde a 3.75.

A resposta do modelo controlado para uma entrada de degrau unitário está disponível na Figura 13. Podemos observar que o sistema tem erro nulo e o tempo de subida é de 0.04 segundos, enquanto em malha aberta, o tempo de subida do motor é de 0.10 segundos.

Aplicando esse controlador no motor, foram obtidos os gráficos da Figura 14, onde podemos observar a resposta rápida e o erro nulo do controlador. Além disso, está presente um sobressinal na resposta da planta e ruido de medição. Como o tempo de amostragem foi de 10 milissegundos e o motor é muito rápido, o uso do encoder para obter velocidade é muito discretizado, uma vez que não tem muito tempo para ocorrer uma grande quantidade de pulsos.

Uma forma de resolver o ruido de medição é aplicando um filtro para a leitura de velocidade, como demonstrado na seção 3.1.3. Já para o caso do sobressinal, é possível minimizar sua ocorrência com um integrador anti wind-up e com um filtro de referência, essas técnicas serão estudadas em disciplinas posteriores no curso.

3.1.5 Implementação do Controle de velocidade: experimento

O controlador foi desenvolvido no domínio da frequência, mas sua implementação deve ser realizada no domínio do tempo. Além disso, o controlador obtido foi projetado para operar em tempo contínuo; no entanto, computadores e microcontroladores funcionam

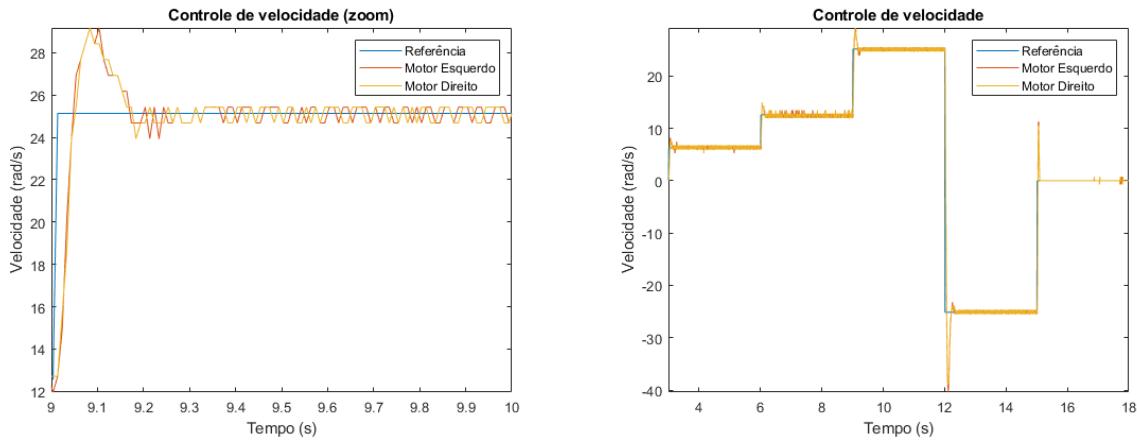


Figura 14: O gráfico à direita mostra a resposta do motor a múltiplas referências e o gráfico à esquerda é um zoom na resposta do motor à terceira referência.

em tempo discreto, ou seja, de forma digital. Portanto, algumas considerações devem ser feitas:

- A entrada do controlador é o erro, ou seja, a referência subtraída do valor medido de velocidade.
- Uma integral é representada no domínio da frequência como:

$$\text{integral} = \frac{1}{s},$$

já no domínio do tempo discreto é uma soma que acumula a medição (y) multiplicada pela diferença de tempo entre as medições:

$$\text{integral} = \sum_{\text{inicio}}^k [y(k)(t(k) - t(k-1))].$$

- Para a implementação, iremos converter o formato da equação do controlador:

$$C = K \left(\frac{s + z}{s} \right) = K \left(1 + \frac{1}{T_i s}, \right)$$

onde T_i é o tempo integrativo e corresponde a $T_i = 1/z$.

A implementação final do controlador é:

$$PWM = K(e + \frac{\text{soma}_e T_s}{T_i});$$

onde T_s é o tempo de amostragem e e é o erro.

Para implementar o controlador em código confira:

- Lembre de calcular o erro e a soma dos erros (soma_e) a cada ciclo;

- Inicialize a variável de soma com 0;
- Confira se a lógica do tempo de amostragem está correta e se o controlador consegue manter esse valor de amostragem.

No Anexo D está disponibilizado o código de controle usado.

3.2 IMU

A IMU (Inertial Measurement Unit), no nosso caso é a MPU-6050, é um dispositivo que combina vários sensores, como acelerômetros e giroscópios, para medir a aceleração linear, a velocidade angular e, em alguns casos, a temperatura ao redor de um objeto. No contexto do pêndulo invertido, a MPU-6050 é usada para medir o ângulo de inclinação do pêndulo, ajudando a manter o equilíbrio.

3.2.1 Acelerômetro

O acelerômetro é um componente da IMU que mede a aceleração linear em uma ou mais direções. Ele detecta a força da gravidade e outros tipos de movimento linear. No projeto, o acelerômetro é usado para medir a inclinação do pêndulo, capturando a aceleração nos eixos Y e Z, que pode ser convertida para um ângulo de inclinação usando arco tangente. Os eixos da MPU-6050 estão disponíveis na Figura 15 e a Figura 16 demonstra como fazer arco tangente.

O acelerômetro pode precisar de calibração. Para calibrar o acelerômetro, siga estes passos:

1. Deixe o sensor parado.
2. Realize inúmeras medições (por exemplo, 3000) em todos os eixos.
3. Identifique o eixo afetado pela gravidade.

A calibração envolve:

- Para os eixos que não sofrem a ação da gravidade: subtrair a média das medições desses eixos da leitura atual do acelerômetro.
- Para o eixo que sofre a ação da gravidade: subtrair a gravidade à média das medições anteriores e então subtrair essa média da leitura atual do acelerômetro.

3.2.2 Giroscópio

O giroscópio é um componente da IMU que mede a velocidade angular em um ou mais eixo. No projeto, ele é usado para medir a inclinação do pêndulo, em conjunto com o acelerômetro. O sentido positivo dos eixos está disponível na Figura 15.

O giroscópio pode precisar de calibração. Para calibrar o giroscópio, siga estes passos:

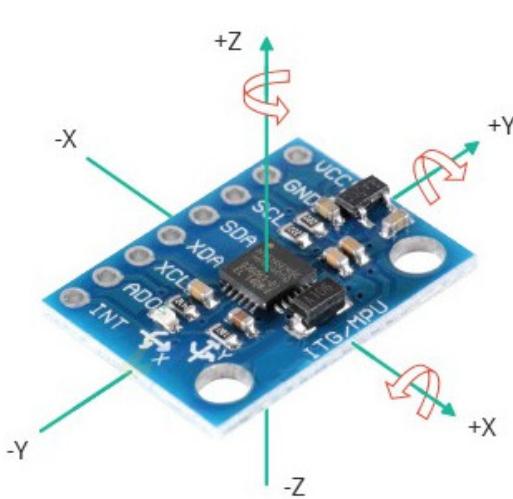


Figura 15: Eixos da MPU-6050.

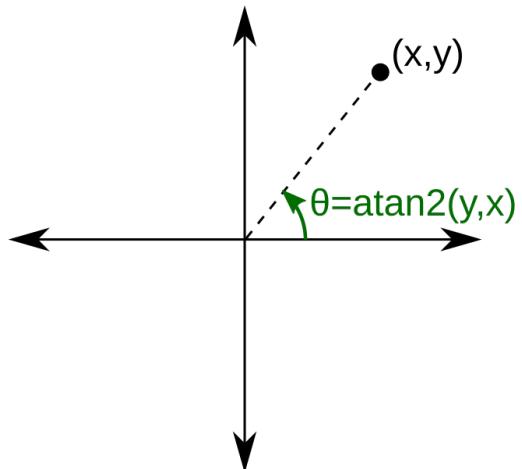


Figura 16: Funcionamento do arco tangente.

1. Deixe o sensor parado.
2. Realize inúmeras medições (por exemplo, 3000) em todos os eixos.
3. Calcule a média em cada eixo.
4. Subtraia a média nas seguintes medições.

3.2.3 Filtro complementar

O sensor MPU-6050 combina um acelerômetro e um giroscópio em um único dispositivo, fornecendo medições de aceleração e rotação. Cada um desses sensores tem suas próprias características e limitações. O acelerômetro é bom para medir a inclinação estática, pois mede a gravidade, mas é sensível a movimentos rápidos e vibrações. Por outro lado, o giroscópio mede a taxa de rotação, sendo bom para detectar movimentos rápidos, mas tende a apresentar deriva ao longo do tempo.

O filtro complementar é uma técnica que combina as leituras do acelerômetro e do giroscópio para obter uma estimativa mais precisa da orientação.

3.2.4 Implementação do Filtro complementar: experimentos

O filtro pode ser implementado como uma ponderação entre os valores da medida do giroscópio e do acelerômetro:

$$\theta = \alpha g + (1 - \alpha)a;$$

onde θ é a medição final, α é o coeficiente da ponderação, g é a medição do giroscópio e a é a medição do acelerômetro. Aumentar o α resulta em um sinal menos ruidoso que demora mais para convergir para o valor correto e diminuir o α resulta em um sinal mais ruidoso que converge mais rápido. O coeficiente pode ser obtido empiricamente e depende

do objetivo de medição, mas um valor comumente usado em filtros complementares de junção de acelerômetro e giroscópio é de 0.98.

O Anexo B tem um código de exemplo e teste da IMU e do filtro complementar. Ao deixar o pêndulo em pé, o angulo deve ser próximo de zero, ao inclinar o pêndulo para a frente, o ângulo deve aumentar e para trás, diminuir.

3.2.5 Teste de Equilíbrio do DIA

Para testar o equilíbrio do díscio, coloque o código disponível no Anexo

4 Projeto do Controlador

Nessa seção, a modelagem matemática baseada nas equações de Newton é apresentada juntamente com a identificação dos parâmetros da planta e posteriormente, considerando as equações do modelo, o compensador de avanço e atraso é projetado para o DIA e por fim, discretizado para ser implementado no ESP32.

4.1 Modelagem Matemática

A abordagem adotada para modelagem matemática do díctalo autônomo foi o uso das equações de Newton para decomposição de forças. Primeiramente, o eixo de coordenadas e as principais variáveis são definidas em um modelo simplificado, que considera o mesmo torque aplicado pelas duas rodas e o pêndulo se moendo para frente:

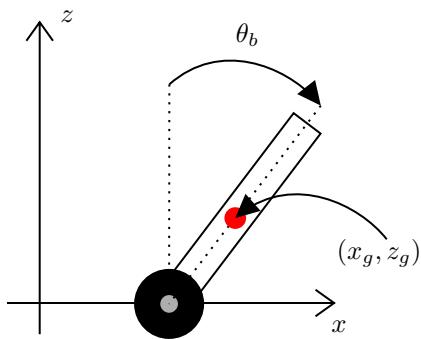


Figura 17: Definição dos eixos e coordenadas iniciais.

- m_b = massa do corpo e motores;
- J_b = momento de inércia do corpo e motores (segundo o centro de massa, *CM*);
- m_w = massa do conjunto das rodas;
- J_w = momento de inércia do conjunto de rodas (segundo o ponto $(x, 0)$);
- r = raio das rodas.

Avaliando o desenho e considerando que o centro das rodas se encontra em uma posição horizontal x , as coordenadas do centro de massa podem ser inferidas como

$$x_g = x + l \sin(\theta_b), \quad (3)$$

$$z_g = l \cos(\theta_b). \quad (4)$$

Assumindo que as rodas esquerda e direita movimentam-se simultaneamente e com o mesmo torque, podemos separar o estudo das forças no DIA em duas partes:

- Corpo principal (impressão 3D, componentes e estator do motor);
- Rodas (rodas, rotor do motor e caixa de redução);

O diagrama de corpo livre do díctico autônomo completo pode ser deduzido utilizando do conhecimento da segunda e terceira leis de Newton. Da mesma maneira, esse diagrama inicial pode ser separado em duas partes, o corpo principal e as rodas, como citado anteriormente.

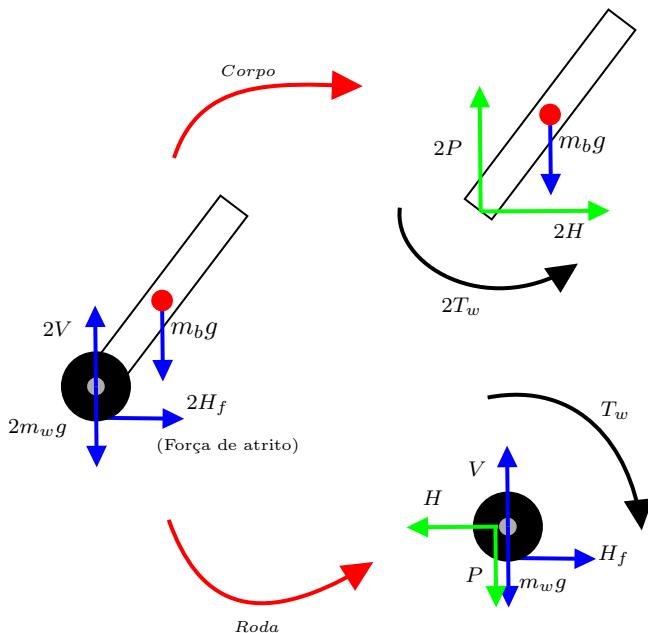


Figura 18: Decomposição de forças no conjunto completo, no corpo e em uma roda.

A Figura 18 mostra a decomposição de forças no conjunto completo e nas duas partes que o compõem. As forças H e P aparecem devido à separação do corpo e das rodas para estudo sendo a força mecânica nos rolamentos dentro dos motores e o torque T_w aparece pela força magnética do rotor e do estator. Essas forças podem ser inferidas logicamente ao considerarmos o diagrama do corpo sem elas, sendo que o corpo está se movimentando para frente devido ao torque aplicado pelas rodas, dessa maneira, alguma força precisa aparecer para justificar esse movimento. Por fim, a força V é a força normal de reação ao toque das duas rodas no chão e H_f a força de atrito de cada roda (oposto ao movimento das mesmas).

4.1.1 Estudo de forças na roda

Como a roda apresenta apenas movimento horizontal, podemos desconsiderar as forças verticais, porque elas estão em equilíbrio. Dessa maneira obtemos a Figura 19 que representa as variáveis necessárias para estudo da translação e da rotação.

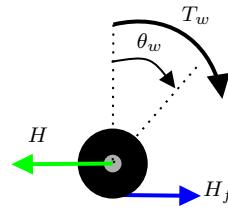


Figura 19: Estudo das forças em uma roda.

- **Translação**

Considerando o movimento da roda como sendo horizontal e para a direita, de acordo com a segunda lei de Newton teremos

$$F_r = ma,$$

$$H_f - H = m_w \ddot{x}, \quad (5)$$

- **Rotação**

Por ser um corpo em movimento de rotação, é preciso fazer o estudo do torque de acordo com a relação do momento de inércia e momento angular,

$$\begin{aligned} \tau &= J\alpha, \\ T_w - H_f r &= J_w \ddot{\theta}_w. \end{aligned}$$

Considerando que não há escorregamento da roda quando ela se move, a relação $x = \theta_w r$ é verdadeira, então chegamos à equação

$$\frac{T_w}{r} - H_f = \frac{J_w}{r^2} \ddot{x}. \quad (6)$$

Por fim, as equações 5 e 6 podem ser somadas chegando à relação:

$$H = \frac{1}{r} T_w - \left(m_w + \frac{1}{r^2} J_w \right) \ddot{x}. \quad (7)$$

4.1.2 Estudo de forças no corpo

A figura 20 apresenta o diagrama de forças no corpo do díscio autônomo em que será feito o mesmo estudo das relações translação e rotação.

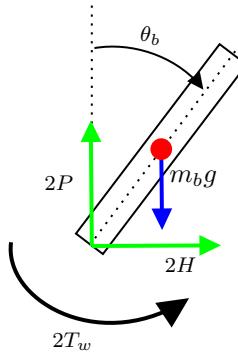


Figura 20: Estudo das forças no corpo.

- Translação - Eixo x

Considerando o deslocamento do corpo para a direita por causa do movimento das rodas, de acordo com a segunda lei de Newton teremos

$$F_r = ma,$$

$$2H = m_b \ddot{x}_g. \quad (8)$$

Através da equação 3 podemos deduzir que

$$\begin{aligned} x_g &= x + l \operatorname{sen}(\theta_b), \\ \dot{x}_g &= \dot{x} + l \dot{\theta}_b \cos(\theta_b), \\ \ddot{x}_g &= \ddot{x} + l(-\dot{\theta}_b^2 \operatorname{sen}(\theta_b) + \ddot{\theta}_b \cos(\theta_b)), \\ \ddot{x}_g &= \ddot{x} + l \ddot{\theta}_b \cos(\theta_b) - l \dot{\theta}_b^2 \operatorname{sen}(\theta_b). \end{aligned}$$

Dessa forma substituindo a equação para a aceleração no eixo x do centro de massa na equação 8 obtemos

$$m_b \ddot{x} + m_b l \ddot{\theta}_b \cos(\theta_b) - m_b l \dot{\theta}_b^2 \operatorname{sen}(\theta_b) = 2H. \quad (9)$$

- Translação - Eixo y

Fazendo a análise das forças verticais na barra obtemos pela segunda lei de Newton

$$2P - m_b g = m_b \ddot{z}_g. \quad (10)$$

Analogamente à análise feita para a posição do centro massa no eixo x , temos para o eixo y

$$\begin{aligned}
z_g &= l \cos(\theta_b), \\
\dot{z}_g &= -l \dot{\theta}_b \sin(\theta_b), \\
\ddot{z}_g &= -l (\dot{\theta}_b^2 \cos(\theta_b) + \ddot{\theta}_b \sin(\theta_b)), \\
\ddot{z}_g &= -l \ddot{\theta}_b \sin(\theta_b) - l \dot{\theta}_b^2 \cos(\theta_b).
\end{aligned}$$

Substituindo essa relação na equação 10 obtemos

$$m_b g - m_b l \ddot{\theta}_b \sin(\theta_b) - m_b l \dot{\theta}_b^2 \cos(\theta_b) = 2P. \quad (11)$$

• Rotação

Agora, é preciso analisar o movimento de rotação do corpo para encontrar a relação do momento de inércia e momento angular. Analisando a imagem 20 podemos perceber que o peso do corpo não causa movimento de rotação, porque é exercido sobre o centro de massa, já as forças H e P são exercidas fora do centro de massa e consequentemente exercem torque.

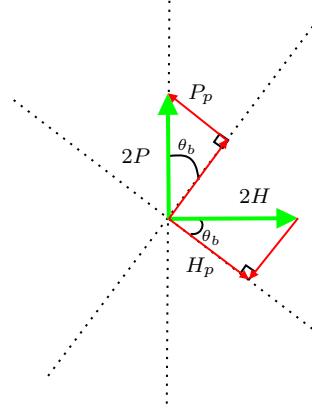


Figura 21: Decomposição das forças $2P$ e $2H$ nos eixos de inclinação do corpo.

À partir da Figura 21, ao decompor as forças $2H$ e $2P$ nos eixos de inclinação do corpo obtemos duas projeções para cada uma, sendo que apenas as forças P_p e H_p exercem torque no corpo e suas equações são:

$$P_p = 2P \sin(\theta_b), \quad (12)$$

$$H_p = 2H \cos(\theta_b). \quad (13)$$

Utilizando da relação do torque para o movimento rotacional, temos

$$2P \sin(\theta_b) - 2H \cos(\theta_b) - 2T_w = J_b \ddot{\theta}_b. \quad (14)$$

- Equações Finais

Substituindo as equações 9 e 11 na equação 14 e simplificando, chegamos a uma das relações que regem a dinâmica do DIA:

$$(J_b + m_b l^2) \ddot{\theta}_b = m_b g \sin(\theta_b) - m_b l \cos(\theta_b) \ddot{x} - 2T_w. \quad (15)$$

Por fim, a segunda relação que rege a dinâmica do díctico autônomo é obtida com a substituição da equação 7 na equação 9, chegando a

$$[m_b + 2(m_w + \frac{1}{r^2} J_w)] \ddot{x} = -m_b l \ddot{\theta}_b \cos(\theta_b) + m_b l \dot{\theta}_b^2 \sin(\theta_b) + 2\frac{1}{r} T_w. \quad (16)$$

- Adequação e Manipulação Algébrica

Após encontrar as equações 15 e 16, algumas manipulações algébricas, substituições e simplificações são feitas para encontrar as funções de transferência de θ e ϕ . Todas as manipulações são descritas no arquivo "Modelo_novo mlx" dentro da pasta "Modelo" no repositório https://github.com/JessicaLuana1377/diciclo_autonomo.

4.2 Identificação de parâmetros

4.2.1 Identificação de parâmetros: experimentos

O código utilizado para os experimentos está disponível no Anexo F.

Passos antes de começar o experimento:

- O primeiro passo para a realização dos experimentos é garantir que a construção está correta. Para isso, siga todos os passos indicados anteriormente;
- Caso não tenham sido usados os mesmos pinos do ESP32 descritos acima, é necessário atualizá-los no código. Entre no arquivo 'tools.h' e altere os pinos dos encoders e dos motores para os correspondentes;
- Caso tenha realizado uma calibração no giroscópio, acesse o arquivo 'tools.cpp' e altere o valor da variável g_x_offset para o valor de offset encontrado para o eixo x do giroscópio.

Garantindo que todos os passos anteriores estão corretos, inicie os preparos para os experimentos. No início do arquivo 'ExperimentoArduinoIDE' estão dispostas as variáveis:

f0 - Frequência inicial da chirp;

f1 - Frequência final da chirp;

```

ets Jul 29 2019 12:21:46

rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:1
load:0x3fff0030,len:1344
load:0x40078000,len:13964
load:0x40080400,len:3600
entry 0x400805f0
File count init = 0
StartMode:1
Configuration Mode (l:list, f:format, r:read and format, R:read only)

```

Figura 22: Captura de tela do CoolTerm.

ts - Tempo de amostragem em milissegundos;

T - Tempo total da chirp;

SIMULATIONS - Quantidade de chirps aplicadas;

MAX_PWM - Máximo de PWM aplicado nos motores.

Todas possuem valores padrões, com f0, f1 e MAX_PWM específicas para cada experimento (experimento do corpo ou das rodas), mas podem ser alteradas caso haja necessidade. Para realizar o experimento das rodas, coloque o código da pasta ‘ExperimentoArduinoIDE’ no ESP32 do díscio, descomente as linhas referentes ao f0, f1 e MAX_PWM do experimento das rodas e coloque o díscio em uma superfície estável, onde as rodas estejam livres. Clique no botão ‘En’ do ESP32 e, quando uma luz azul piscar, clique no outro botão (‘Boot’). Espere o experimento terminar.

Com o ESP32 conectado no computador, abra a ferramenta ‘CoolTerm.exe’. Em ‘Options’, no campo ‘Baudrate’, selecione a velocidade da serial configurada no ESP32; caso não tenha alterado, é 115200. Para salvar os dados em um arquivo, vá em ‘Connection’, ‘Capture to text/binary file’, ‘Start’, escolha o diretório do arquivo sendo a pasta ‘Rodas’ e coloque o nome ‘exp_rodas_bruto’. Para conectar a ferramenta com o ESP32, confira se o *Serial Monitor* de todas as abas do Arduino IDE está fechado, clique em ‘Connect’ e espere o ESP32 parar de piscar a luz azul. A Figura 22 mostra como deve estar o ‘CoolTerm.exe’ nesse instante.

Ao clicar em ‘l’, serão listados todos os arquivos presentes na flash do ESP32. A cada nova simulação, é criado um novo arquivo, começando com o nome ‘Arquivo0’, ‘Arquivo1’, etc. A letra ‘f’ apaga todos os arquivos. A letra ‘r’ lê e, em seguida, apaga todos os arquivos e ‘R’ apenas lê, mantendo todos na memória.

Realize a leitura com ‘r’ ou ‘R’ e, quando terminar a leitura, pare a gravação do arquivo em ‘*Connection*’, ‘*Capture to text/binary file*’, ‘*Stop*’. Vá no arquivo ‘exp_rodas_bruto.txt’ e apague as linhas desnecessárias, deixando apenas o cabeçalho e os dados (vá ao final do arquivo e apague todas as linhas em branco).

No arquivo ‘TratarDados mlx’ descomente as linhas referentes ao experimento das rodas presentes no início e no final do arquivo e execute. Nele estão dispostos gráficos para verificação dos dados, alguns pontos devem ser conferidos:

- A frequência inicial deve ser pequena o suficiente para que o início do gráfico de velocidade apresente um ganho constante;
- A frequência final não deve ser muito alta para não capturar muito ruído;
- O tempo da chirp deve ser grande o suficiente para dar tempo do motor reagir à entrada;
- Repita os experimentos com novos valores para as variáveis, se necessário.

Ao final do arquivo ‘TratarDados mlx’ é gerado um arquivo de dados com extensão do MATLAB que será usado em ‘ExperimentoRodas mlx’. Execute-o e serão apresentados 4 modelos: dois baseados na posição fornecida pelo encoder de cada roda e dois baseados na velocidade de cada roda. Escolha o que melhor corresponde ao experimento e substitua no final. Execute o arquivo novamente e serão salvos os dados do modelo correto para serem usados no modelo do pêndulo.

Para realizar o experimento do corpo, siga os mesmos passos anteriores. O experimento deve ser realizado segurando as rodas do díctico e deixando-o pendurado, livre para que o corpo possa balançar. Mantenha-o estável durante todo o experimento e sem o cabo para não prejudicar a captura de dados. Se, em algum momento, o corpo do díctico bater em algum lugar, diminua a variável MAX_PWM e recomece o experimento. Ao final do experimento, realize o mesmo passo a passo com o ‘CoolTerm.exe’ para a captura de dados. Coloque o nome do arquivo como ‘exp_corpo_bruto.txt’ na pasta ‘Corpo’.

Comente as linhas referentes ao experimento das rodas no arquivo ‘TratarDados mlx’ e descomente as linhas referentes ao experimento do corpo. Os mesmos pontos devem ser conferidos nesse experimento. Ao final do arquivo ‘TratarDados mlx’ será gerado o arquivo de dados necessário para ‘ExperimentoCorpo mlx’. Execute o ‘ExperimentoCorpo mlx’ e serão apresentados 6 modelos: 3 baseados na posição das rodas (2 encoders e a MPU) e 3 nas respectivas velocidades. Antes de escolher um modelo, recorte os dados para a evitar ruídos que possam ter ocorrido no início da simulação e na desaceleração da chirp. Realize a seleção dos dados tanto na posição quanto na velocidade. Execute o código, escolha o modelo que melhor corresponde ao experimento e substitua no final do arquivo. Execute novamente o código e o modelo escolhido será salvo para ser usado no modelo do pêndulo.

4.3 Controlador

As funções de transferência para as variáveis θ e ϕ do modelo encontradas após a execução dos testes foram

$$G_\theta(s) = \frac{-17.422s}{(s - 8.441)(s + 4.423)(s + 19.21)}, \quad (17)$$

$$G_\phi(s) = \frac{-2.9209(s - 7.961)(s + 7.961)}{s^2}. \quad (18)$$

O projeto do controle do sistema consiste em um compensador de avanço e atraso para as plantas de θ e ϕ como mostrado na equação 19. O compensador de avanço e atraso foi escolhido porque o compensador de atraso não possui *windup* no integrador, o compensador de avanço possui atenuação para altas frequências e sua implementação é relativamente simples.

$$C_{lead-lag}(s) = K \frac{(s + Z_{lead}) (s + Z_{lag})}{(s + P_{lead}) (s + P_{lag})}. \quad (19)$$

Um compensador de avanço e atraso consiste em dois zeros e dois polos, sendo que $Z_{lead} < P_{lead}$ e $Z_{lag} > P_{lag}$, dessa forma o projeto é feito de forma a alocar esses polos e zeros em locais adequados para atingir os objetivos de controle.

- Controlador de θ

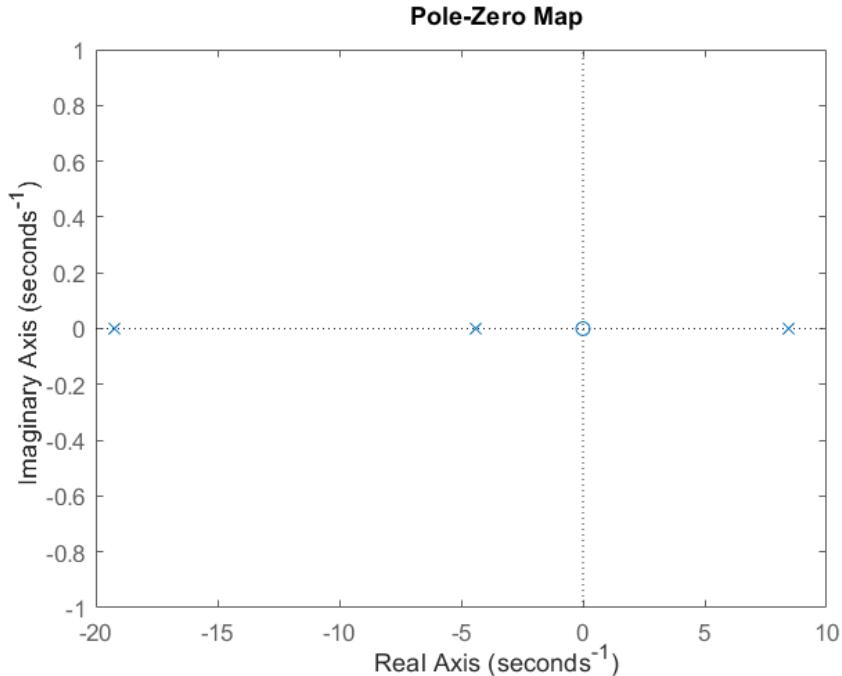


Figura 23: Mapeamento de zeros e polos da planta de θ .

Analizando a equação 17 podemos fazer o mapeamento de zeros e polos no plano s e analisar boas posições potenciais para os zeros e polos do compensador de avanço e atraso. A Figura 23 mostra que para o ângulo θ temos um sistema instável, como o esperado pois o díctico autônomo não se equilibra sozinho.

Para o projeto do compensador é levada em conta as seguintes relações:

$$Z_{lead} = \frac{\omega_c}{\sqrt{\sigma}},$$

$$P_{lead} = \omega_c \sqrt{\sigma}.$$

Após diversos testes no *software Matlab*, a razão de magnitude entre o polo e o zero do compensador de avanço, $\sigma = \frac{P_{lead}}{Z_{lead}}$, foi escolhido como sendo 6. Esse valor garante uma grande faixa de frequências em que o compensador de avanço está em efeito, mas suficientemente pequeno para garantir que seu zero esteja distante do zero do compensador de atraso.

A frequência de corte, ω_c , é a frequência na qual a magnitude do ganho do sistema em malha aberta tem o valor de 0dB. Após alguns testes com o projeto do compensador, esse valor foi escolhido como sendo $3,5 \text{ Hz} = 7\pi \text{ rad/s}$.

De acordo com a Figura 23 temos um zero dominante no sistema de θ que deixa a resposta mais lenta, por este motivo, como ainda era preciso alocar o zero e polo do compensador de atraso, o polo foi alocado na origem para "cancelar" a influência do zero da origem na resposta. Já o zero do compensador foi alocado em -20, dessa forma, o compensador de atraso apenas substitui o zero lento por um de maior magnitude, acelerando o sistema.

Por fim, em subsequentes testes de estabilidade no *Matlab* o ganho encontrado para estabilizar o sistema com uma boa margem foi de -72,9759. A equação 20 mostra o compensador de avanço e atraso para a variável θ em sua forma completa.

$$C_\theta(s) = -72,9759 \frac{(s + 8,978)}{(s + 53,87)} \frac{(s + 20)}{s}. \quad (20)$$

A Figura 24 mostra o lugar das raízes da planta de θ em malha fechada, analisando a imagem é possível perceber a eficácia do controlador em estabilizar a planta. Vale ressaltar que o arquivo "*Compensador mlx*" dentro da pasta "*Controle*" no repositório https://github.com/JessicaLuana1377/diciclo_autonomo ainda realiza mais testes de estabilidade checando o diagrama de Bode em malha fechada e a resposta do sistema a um degrau.

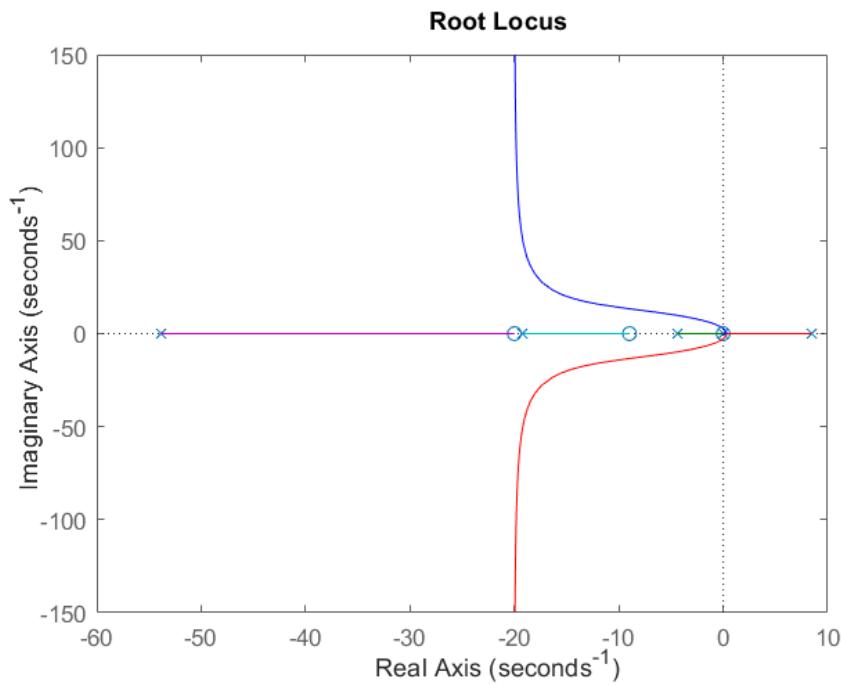


Figura 24: Lugar das raízes da planta de θ em malha fechada.

- Controlador de Φ

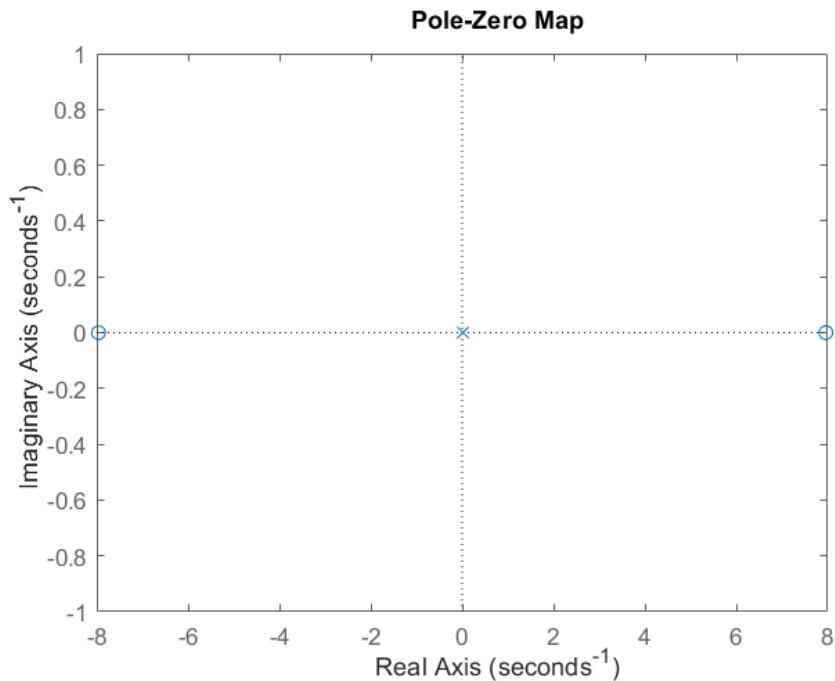


Figura 25: Mapeamento de zeros e polos da planta de ϕ .

A Figura 25 mostra a posição dos polos e zeros da planta, que consiste em dois zeros no eixo real igualmente espaçados, sendo um deles de fase não mínima e dois polos na origem.

A abordagem para o projeto do compensador de avanço e atraso para a planta de ϕ é a mesma tomada para o projeto do controlador de θ , com algumas pequenas diferenças.

Como a planta possui um número igual de zeros e polos, o zero do compensador de avanço foi removido para garantir um número de polos maior que o número de zeros, permitindo assim que a fase do sistema controlado passe por 180° apenas uma vez. Dessa forma, para o projeto não foi preciso se preocupar com a relação de σ , portanto após alguns testes no *Matlab* o valor escolhido para a frequência de corte foi de $0,5\text{Hz} = \pi$ garantindo também que a malha externa (ϕ) seja mais lenta que a malha interna (θ).

O compensador de atraso foi projetado analisando a resposta em frequência do sistema com o diagrama de Bode para garantir uma boa margem de fase e ganho no sistema. Os valores obtidos após algumas iterações foram de 0,1 para o zero e 15 para o polo do compensador de atraso. Da mesma forma, o ganho do sistema foi encontrado levando-se em conta o ganho necessário para estabilizar o sistema de acordo com a análise do lugar das raízes do sistema em malha fechada.

$$C_\phi(s) = 0,993 \frac{(s + 0,1)}{(s + 53,87)(s + 3,142)}. \quad (21)$$

Por fim, a Equação 21 descreve o compensador de avanço e atraso para ϕ e a Figura 26 mostra o lugar das raízes do sistema em malha fechada mostrando sua estabilidade alcançada com o consequente aumento do ganho.

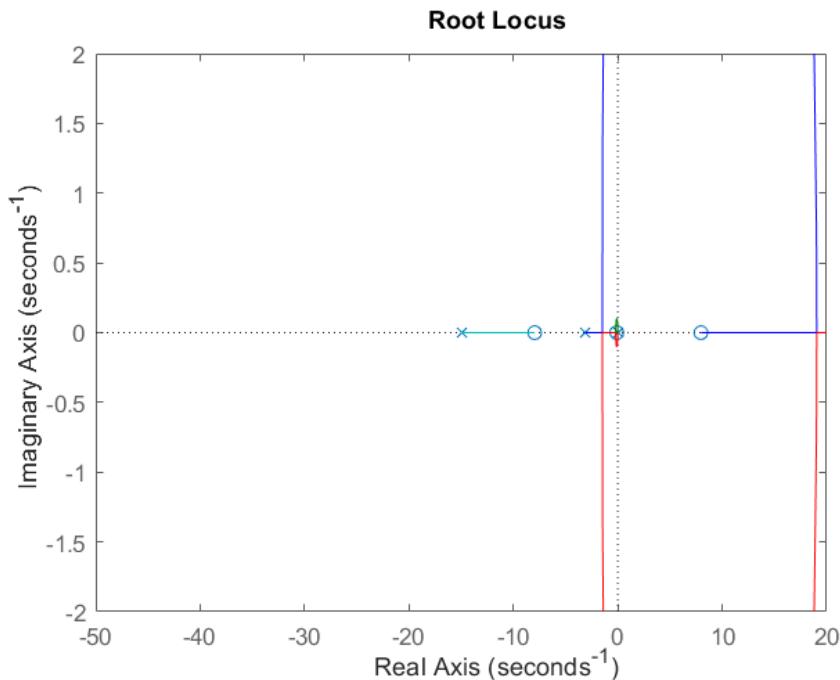


Figura 26: Lugar das raízes da planta de ϕ em malha fechada.

4.3.1 Controlador Discretizado

O último passo para implementar o controle no ESP32 consiste em discretizar os controladores e transformá-los em equações de diferenças que relacionam a entrada e a saída. Na literatura, existem diferentes métodos para discretização de uma planta ou controlador, o método escolhido foi o de Tustin que faz a substituição:

$$s = \frac{2}{T_s} \left(\frac{1 - z^{-1}}{1 + z^{-1}} \right). \quad (22)$$

Em que T_s consiste no tempo de amostragem que devido à natureza do ESP32 foi escolhido como sendo 0,005 segundos. Essa discretização pode ser facilmente implementada no *Matlab* com o comando "*c2d*".

Aplicando a discretização através do método de Tustin chegamos no modelo discretizado do controladores de θ e ϕ :

$$C_\theta(z) = -69,046 \frac{(z - 0,9048)(z - 0,9561)}{(z - 1)(z - 0,7626)}. \quad (23)$$

$$C_\phi(z) = 0,0023898 \frac{(z - 1)(z + 1)}{(z - 0,9277)(z - 0,9844)}. \quad (24)$$

As equações 23 e 24 representam os controladores no plano Z , ou seja, discretizados, contudo o algoritmo escrito em C apenas "compreende" a linguagem temporal. Dessa maneira, de acordo com a passagem das iterações no código no ESP32 o algoritmo precisa ser capaz de expressar o efeito do controlador, e isso é feito convertendo as equações no domínio Z para o domínio do tempo. O processo de conversão é o mesmo para os dois controladores, então o cálculo será feito para o controlador de θ .

Reescrevendo a equação 23 multiplicando todos os termos, obtemos

$$C_\theta(z) = \frac{Y(Z)}{E_\theta(Z)} = \frac{-69,046z^2 + 128,488z - 59,730}{z^2 - 1,763z + 0,763}. \quad (25)$$

A equação 25 mostra também que o controlador consiste na relação de sua saída pelo erro medido. O próximo passo consiste em fazer a multiplicação cruzada dos termos e multiplicar ambos os lados por z^{-2}

$$Y(Z)[1 - 1,763z^{-1} + 0,763z^{-2}] = E_\theta(Z) - 69,046 + 128,488z^{-1} - 59,730z^{-2}. \quad (26)$$

Por definição das relações do domínio da frequência e do tempo, um sinal discretizado multiplicado por z^{-1} consiste em olhar a amostra anterior desse valor, aplicando essa definição e convertendo a equação 26 para o domínio do tempo e isolando $y[k]$ obtemos

$$y[k] = 1,763y[k-1] - 0,763y[k-2] - 69,046e_\theta[k] + 128,488e_\theta[k-1] - 59,730e_\theta[k-2]. \quad (27)$$

Por fim, a equação 27 mostra o valor de saída do controlador na amostra atual de acordo com uma amostra do erro de θ atual e amostras passadas do mesmo erro e das saídas. Essa relação é implementada no código disponibilizado na pasta "*Controle*" e "*controle_lead_lag*" no repositório https://github.com/JessicaLuana1377/diciclo_autonomo.

Aplicando o mesmo método para o controlador de ϕ obtemos outra equação de diferenças que possibilita sua implementação no código do ESP32

$$y[k] = 1,912y[k-1] - 0,913y[k-2] + 0,00239e_\phi[k] + 1,195 \cdot 10^{-6}e_\phi[k-1] - 0,002389e_\phi[k-2]. \quad (28)$$

Vale ressaltar que outros métodos de discretização podem ser utilizados e os controladores discretizados precisam ser novamente testados no *software Matlab* para garantir que estabilizaram o sistema. Outro ponto importante é o cálculo correto das equações de diferenças e sua

5 Conclusão

Por fim, o objetivo inicial do trabalho era criar um kit didático para estudo de engenharia de controle que fosse economicamente viável para reprodução. Em coordenação com os professores Fernando Oliveira Souza e Leonardo Antônio Borges Tôrres o díctico autônomo foi desenvolvido com o uso de impressões 3D, itens conhecidos de kits de arduino e o microcontrolador ESP32 chegando a um orçamento médio de 131 reais.

O relatório apresentou uma recapitulação de todo o processo de desenvolvimento do díctico autônomo, passando por todos os componentes e suas especificações, como também pela montagem e ligações elétricas. Foram descritos os procedimentos para testar os motores e a unidade de medição inercial (IMU) com a disponibilização dos códigos para arduino e concluindo o trabalho, a modelagem matemática utilizada foi apresentada juntamente com a identificação de parâmetros e o projeto do compensador de avanço e atraso contínuo e discreto.

Vale ressaltar que o projeto não teria sido possível sem o apoio financeiro das instituições de fomento à pesquisa e a intenção do projeto era beneficiar a comunidade acadêmica com a inserção do aluno no contexto de controle da planta que cobre áreas multidisciplinares, como apresentado por este relatório.

A Código de Exemplo e Teste dos encoders

Este código está presente no repositório https://github.com/JessicaLuana1377/diciclo_autonomo.git, na pasta 'CódigosDisponíveis'.

```
#include <ESP32Encoder.h>

// Pinos do encoders
#define ENCODER_C1_R 32
#define ENCODER_C2_R 33
#define ENCODER_C1_L 14
#define ENCODER_C2_L 27
#define TS 500 // Tempo de amostragem em milissegundos

// Pinos dos motores
#define IN1 16
#define IN2 17
#define ENA 4

#define IN3 18
#define IN4 19
#define ENB 23

// Canais de PWM
#define L_CHANNEL 0
#define R_CHANNEL 1

// Variaveis dos encoder
ESP32Encoder encoder_r;
ESP32Encoder encoder_l;

unsigned long t0 = millis();
void setup() {
    Serial.begin(115200);
    delay(100);

    // Configuração dos pinos
    pinMode(IN1, OUTPUT);
    pinMode(IN2, OUTPUT);
    pinMode(IN3, OUTPUT);
    pinMode(IN4, OUTPUT);
```

```
// Configuração do PWM
ledcSetup(L_CHANNEL, 200, 8);
ledcSetup(R_CHANNEL, 200, 8);
ledcAttachPin(ENA, L_CHANNEL);
ledcAttachPin(ENB, R_CHANNEL);
ledcWrite(L_CHANNEL, 0);
ledcWrite(R_CHANNEL, 0);

// Configuração dos encoders
ESP32Encoder::useInternalWeakPullResistors = puType::up;
encoder_r.attachFullQuad(ENCODER_C1_R, ENCODER_C2_R);
encoder_l.attachFullQuad(ENCODER_C1_L, ENCODER_C2_L);
encoder_r.clearCount();
encoder_l.clearCount();
}

void loop() {
    if (millis() - t0 > TS){
        t0 = millis();
        float position_r = encoder_r.getCount();
        float position_l = encoder_l.getCount();

        // Comente essas duas linhas para testar o encoder individualmente
        motor(10,L_CHANNEL);
        motor(100,R_CHANNEL);

        Serial.print(" Encocer direito: ");
        Serial.print(position_r);
        Serial.print(" Encocer esquerdo: ");
        Serial.print(position_l);
        Serial.println();
    }
}

void motor(int PWM, int channel)
{
    int p1, p2;
    if (channel == L_CHANNEL)
    {
        p1 = IN1;
```

```
p2 = IN2;
} else {
    p1 = IN3;
    p2 = IN4;
}
PWM = constrain(PWM, -100, 100);
PWM = map(PWM, -100, 100, -255, 255);
if (PWM == 0)
{
    digitalWrite(p1, LOW);
    digitalWrite(p2, LOW);
    return;
} else if (PWM > 0)
{
    digitalWrite(p1, LOW);
    digitalWrite(p2, HIGH);
} else
{
    PWM = -PWM;
    digitalWrite(p1, HIGH);
    digitalWrite(p2, LOW);
}
ledcWrite(chanel, PWM);
}
```

B Código de Exemplo e Teste da IMU

Este código está presente no repositório https://github.com/JessicaLuana1377/diciclo_autonomo.git, na pasta 'CódigosDisponíveis'.

```
#include <Adafruit_MPU6050.h>
#include <Adafruit_Sensor.h>
#include <Wire.h>

#define TS 5000 // microssegundos
unsigned long t = micros();

// MPU6050 mpu;
Adafruit_MPU6050 mpu;
float g_x_offset = -0.05 + 0.097;
sensors_event_t a, g, temp;

float g_x, a_pitch, theta;
float alpha = 0.98;
void setup() {
    Serial.begin(115200);
    delay(500);
    Wire.begin();

    if (!mpu.begin()) {
        Serial.println("Failed to find MPU6050 chip");
        while (1) {
            delay(10);
        }
    }
    mpu.setAccelerometerRange(MPU6050_RANGE_8_G);
    mpu.setGyroRange(MPU6050_RANGE_500_DEG);
    mpu.setFilterBandwidth(MPU6050_BAND_184_HZ);

    delay(1000);
    mpu.getEvent(&a, &g, &temp);
    theta = atan2(a.acceleration.z, a.acceleration.y);
}

void loop() {
    if (micros() - t >= TS){
```

```
t = micros();  
  
mpu.getEvent(&a, &g, &temp);  
g_x = -(g.gyro.x + g_x_offset);  
a_pitch = atan2(a.acceleration.z, a.acceleration.y);  
  
theta = alpha*(theta + g_x*(TS/1e6)) + (1-alpha)*a_pitch;  
  
Serial.print(" Theta: ");  
Serial.print(theta);  
Serial.println();  
}  
}
```

C Código de modelagem de motor

Este código está presente no repositório https://github.com/JessicaLuana1377/diciclo_autonomo.git, na pasta 'CódigosDisponíveis'.

```
#include <ESP32Encoder.h>
#include <math.h>
#include <stdio.h>
#include <driver/pcnt.h>

#define ENCODER_C1_R 14
#define ENCODER_C2_R 27
#define ENCODER_C1_L 33
#define ENCODER_C2_L 32

#define IN1 17
#define IN2 16
#define ENA 4
#define IN3 18
#define IN4 19
#define ENB 23

#define L_CHANNEL 0
#define R_CHANNEL 1
#define SIMULATION_TIME 0.4

// Funções auxiliares
void motor(int PWM, int channel);
void PinSetup();
// Variáveis Globais
const float TS = 10/1e3; // 10 milisegundos
float t0, t;
unsigned long last_time;
ESP32Encoder encoder_r;
ESP32Encoder encoder_l;
float theta_ml, theta_mr; // posição angular do motor esquerdo e direito
float u_ml, u_mr; // entrada dos motores

void setup() {
    Serial.begin(115200);
    PinSetup();
```

```
motor(0, L_CHANNEL);
motor(0, R_CHANNEL);
delay(100);
t0 = micros()/1e6;
Serial.println("t;u_ml;theta_ml;u_mr;theta_mr");
}

unsigned long diff;
void loop() {
    if ((diff = micros() - last_time) >= TS*1e6) {
        last_time = micros();
        t = micros()/1e6 - t0;

        theta_ml = 2*PI * encoder_l.getCount()/(float)(4*7*30);
        theta_mr = 2*PI * encoder_r.getCount()/(float)(4*7*30);

        if (t < SIMULATION_TIME) {
            u_ml = 100;
            u_mr = 100;
        } else if (t < SIMULATION_TIME*2) {
            u_ml = -100;
            u_mr = -100;
        } else {
            motor(0, L_CHANNEL);
            motor(0, R_CHANNEL);
            while (true) delay(1000);
        }
        motor(u_ml, L_CHANNEL);
        motor(u_mr, R_CHANNEL);
        printf("%.04f;%.04f;%.04f;%.04f;%.04f\n",
               t, u_ml, theta_ml, u_mr, theta_mr);
    }
}

void PinSetup() {
    // Configuração dos pinos
    pinMode(IN1, OUTPUT);
    pinMode(IN2, OUTPUT);
    pinMode(IN3, OUTPUT);
    pinMode(IN4, OUTPUT);
```

```
// Configuração do PWM
ledcSetup(L_CHANNEL, 200, 8);
ledcSetup(R_CHANNEL, 200, 8);
ledcAttachPin(ENA, L_CHANNEL);
ledcAttachPin(ENB, R_CHANNEL);
ledcWrite(L_CHANNEL, 0);
ledcWrite(R_CHANNEL, 0);

// Configuração dos encoders
ESP32Encoder::useInternalWeakPullResistors = puType::up;
encoder_r.attachFullQuad(ENCODER_C1_R, ENCODER_C2_R);
encoder_l.attachFullQuad(ENCODER_C1_L, ENCODER_C2_L);
encoder_r.clearCount();
encoder_l.clearCount();

}

void motor(int PWM, int channel) {
    int p1, p2;
    if (channel == L_CHANNEL) {
        p1 = IN1;
        p2 = IN2;
    } else {
        p1 = IN3;
        p2 = IN4;
    }
    PWM = constrain(PWM, -100, 100);
    PWM = map(PWM, -100, 100, -255, 255);
    if (PWM > 0) {
        digitalWrite(p1, LOW);
        digitalWrite(p2, HIGH);
    } else if (PWM < 0) {
        PWM = -PWM;
        digitalWrite(p1, HIGH);
        digitalWrite(p2, LOW);
    } else {
        digitalWrite(p1, LOW);
        digitalWrite(p2, LOW);
    }
    ledcWrite(channel, PWM);
}
```

D Código de controle da velocidade do motor

Este código está presente no repositório https://github.com/JessicaLuana1377/diciclo_autonomo.git, na pasta 'CódigosDisponíveis'.

```
#include <ESP32Encoder.h>
#include <math.h>
#include <stdio.h>

#define ENCODER_C1_R 14
#define ENCODER_C2_R 27
#define ENCODER_C1_L 33
#define ENCODER_C2_L 32

#define IN1 17
#define IN2 16
#define ENA 4
#define IN3 18
#define IN4 19
#define ENB 23

#define L_CHANNEL 0
#define R_CHANNEL 1

// Funções auxiliares
void motor(int PWM, int channel);
void PinSetup();

// Variáveis Globais
const float TS = 10/1e3; // 10 milisegundos
const float Ti = 1/25.0;
const float Kp = 3.75;

float t0, t;
unsigned long last_time;
ESP32Encoder encoder_r;
ESP32Encoder encoder_l;

float theta_ml, theta_mr; // posição angular dos motores
float last_theta_ml, last_theta_mr;
float omega_ml, omega_mr; // velocidade angular dos motores
```

```
float u_ml, u_mr; // entrada dos motores
float error_ml, error_mr;
float sum_error_ml = 0, sum_error_mr = 0;

float ref = 0; // rad/s;

void setup() {
    Serial.begin(115200);
    PinSetup();

    motor(0, L_CHANNEL);
    motor(0, R_CHANNEL);

    printf("t;ref;u_ml;omega_ml;u_mr;omega_mr\n");

    delay(100);

    t0 = micros()/1e6;
}

unsigned long diff;
void loop() {
    if ((diff = micros() - last_time) >= TS*1e6) {
        last_time = micros();
        t = micros()/1e6 - t0; // tempo de simulação em segundos

        theta_ml = 2*PI * encoder_l.getCount()/(float)(4*7*30);
        theta_mr = 2*PI * encoder_r.getCount()/(float)(4*7*30);

        // calculo da velocidade:
        // variação da posição sobre a variação do tempo -> (dtheta/dt)
        omega_ml = (theta_ml - last_theta_ml)/TS;
        omega_mr = (theta_mr - last_theta_mr)/TS;

        error_ml = ref - omega_ml;
        error_mr = ref - omega_mr;
        // integral do erro:
        // somatório das áreas dos trapézios -> sum(erro * dt)
        sum_error_ml += error_ml * TS;
        sum_error_mr += error_mr * TS;
    }
}
```

```
u_ml = Kp * (error_ml + sum_error_ml/Ti);
u_mr = Kp * (error_mr + sum_error_mr/Ti);

motor(u_ml, L_CHANNEL);
motor(u_mr, R_CHANNEL);

// Descomente para observar as saídas no Arduino Serial Plot
//printf("LI:-30,LS:30,ref:%.04f,omega_ml:%.04f,omega_mr:%.04f\n",
//      ref, omega_ml, omega_mr);

// Ou descomente este para ler os dados no CoolTerm
// printf("%.04f;%.04f;%.04f;%.04f;%.04f;%.04f\n",
//        t, ref, u_ml, omega_ml, u_mr, omega_mr);

last_theta_ml = theta_ml;
last_theta_mr = theta_mr;
}

if (t >= 3 && t < 6) { // referência 1
    ref = 2*PI;
} else if (t >= 6 && t < 9) { // referência 2
    ref = 4*PI;
} else if (t >= 9 && t < 12) { // referência 3
    ref = 8*PI;
} else if (t >= 12 && t < 15) { // referência 4
    ref = -8*PI;
} else { // referência final
    ref = 0;
}

void PinSetup() {
    // Configuração dos pinos
    pinMode(IN1, OUTPUT);
    pinMode(IN2, OUTPUT);
    pinMode(IN3, OUTPUT);
    pinMode(IN4, OUTPUT);

    // Configuração do PWM
```

```
ledcSetup(L_CHANNEL, 200, 8);
ledcSetup(R_CHANNEL, 200, 8);
ledcAttachPin(ENA, L_CHANNEL);
ledcAttachPin(ENB, R_CHANNEL);
ledcWrite(L_CHANNEL, 0);
ledcWrite(R_CHANNEL, 0);

// Configuração dos encoders
ESP32Encoder::useInternalWeakPullResistors = puType::up;
encoder_r.attachFullQuad(ENCODER_C1_R, ENCODER_C2_R);
encoder_l.attachFullQuad(ENCODER_C1_L, ENCODER_C2_L);
encoder_r.clearCount();
encoder_l.clearCount();
}

void motor(int PWM, int channel) {
    int p1, p2;
    if (channel == L_CHANNEL) {
        p1 = IN1;
        p2 = IN2;
    } else {
        p1 = IN3;
        p2 = IN4;
    }
    PWM = constrain(PWM, -100, 100); // saturador
    // altera a escala de -100 a 100 para -255 a 255:
    PWM = map(PWM, -100, 100, -255, 255);
    if (PWM > 0) {
        digitalWrite(p1, LOW);
        digitalWrite(p2, HIGH);
    } else if (PWM < 0) {
        PWM = -PWM;
        digitalWrite(p1, HIGH);
        digitalWrite(p2, LOW);
    } else {
        digitalWrite(p1, LOW);
        digitalWrite(p2, LOW);
    }
    ledcWrite(channel, PWM);
}
```

E Código Teste de Equilíbrio do DIA

Este código está presente no repositório https://github.com/JessicaLuana1377/diciclo_autonomo.git, na pasta ‘Controle’ → ‘ControleDLQGI’. Ele é dividido em três arquivos:

- **ControleDLQGI.ino**

```
#include <stdio.h>
#include <freertos/FreeRTOS.h>
#include <freertos/task.h>
#include <esp_system.h>
#include <ESP32Encoder.h>
#include <math.h>
#include <Adafruit_MPU6050.h>
#include <Adafruit_Sensor.h>
#include <Wire.h>
#include "FS.h"
#include "SPIFFS.h"

#include "tools.h"

void PinSetup();
void taskBlinkLed(void* pvParameter);
int fileCountInit();
void listDir(fs::FS &fs, const char * dirname, uint8_t levels);

// CONST VARIABLES
const float ts = 3e-3;
const float alpha_w = 0.5;
const float r_w = 0.03;
const float w = 11e-2;
const float K[3] = {-44.9181, -30.9875, -6.1045};
const float Ki = 35.3553; // original

// GLOBALS VARIABLES
ESP32Encoder encoder_r;
ESP32Encoder encoder_l;

Modes startMode = INIT;
```

```
// MPU6050 mpu;
Adafruit_MPU6050 mpu;
sensors_event_t a, g, temp;
float gy_offset = -0.01000992;

int count = 0;

Kalman kalman_theta(ts);

float position_r, position_l, x, dx, theta, psi;
float last_position_r, last_position_l, last_theta, Vr, Vl;
float g_x, g_y, g_z;
float a_pitch, a_yaw, a_roll;
float dwheel_f_r, dwheel_f_l;
float V, DeltaV;
int pwm_value;

float error_psi, error_psi_int;
float ref_psi_f, ref_psi = 0;

float q[3] = {0};
float error_dx_int = 0;
float ref_dx = 0.0;

void taskControl(void* pvParameter) {

    for (int i=0; i<round(2.0/ts); i++) {
        mpu.getEvent(&a, &g, &temp);
        g_x = -g.gyro.x;
        a_pitch = atan2(a.acceleration.z, a.acceleration.y);
        theta = kalman_theta.getAngle(a_pitch, g_x);
        delay(1000*ts);
    }

    TickType_t init_loop_time;
    float t0 = micros()/1e6;
    float t = 0;

    while(1) {
        init_loop_time = xTaskGetTickCount();
```

```
t = micros()/1e6 - t0;

position_r = 2*PI*encoder_r.getCount()/(float)(4*7*30);
position_l = 2*PI*encoder_l.getCount()/(float)(4*7*30);
x = r_w * (position_l - position_r)/2;

mpu.getEvent(&a, &g, &temp);
g_x = -g.gyro.x;
g_y = g.gyro.y - gy_offset;
a_pitch = atan2(a.acceleration.z, a.acceleration.y);

// direita
dwheel_f_r = alpha_w*dwheel_f_r +
(1-alpha_w)*(position_r - last_position_r)/ts;
// esquerda
dwheel_f_l = alpha_w*dwheel_f_l +
(1-alpha_w)*(position_l - last_position_l)/ts;
dx = r_w * (dwheel_f_l - dwheel_f_r)/2;

// corpo
theta = kalman_theta.getAngle(a_pitch, g_x);
psi += g_y*ts;
// printf("%.04f,%.04f\n", psi, -r_w * (position_l + position_r)/w);

q[0] = theta;
q[1] = dx;
q[2] = (theta - last_theta)/ts;
error_dx_int += (ref_dx - dx)*ts;

// psi
ref_psi_f = 0.9962*ref_psi_f + 0.0038*ref_psi;
error_psi = ref_psi_f - psi;
error_psi_int += error_psi*ts;

V = -(K[0]*q[0] + K[1]*q[1] + K[2]*q[2]) - Ki*error_dx_int;
DeltaV = -1.5694*(error_psi + error_psi_int/0.7867);
Vl = (V + DeltaV);
Vr = -(V - DeltaV);

if (abs(theta) >= PI/4) {
```

```
    error_dx_int = 0;
    error_psi_int = 0;
    Vr = 0; Vl = 0;
    psi = 0;
}

pwm_value = mapfloat(Vr, -7.4, 7.4, -100, 100);
motor(pwm_value, R_CHANNEL);
pwm_value = mapfloat(Vl, -7.4, 7.4, -100, 100);
motor(pwm_value, L_CHANNEL);

last_theta = theta;
last_position_r = position_r;
last_position_l = position_l;

// printf("%.02f\n", 1000*((micros()/1e6 - t0) - t));
vTaskDelayUntil(&init_loop_time, pdMS_TO_TICKS(ts*1000));
}

vTaskDelete(NULL);
}

void setup() {
    Serial.begin(921600);
    PinSetup();
    Wire.begin();

    if (!mpu.begin()) {
        Serial.println("Failed to find MPU6050 chip");
        while (1) {
            delay(10);
        }
    }
    mpu.setAccelerometerRange(MPU6050_RANGE_4_G);
    mpu.setGyroRange(MPU6050_RANGE_500_DEG);
    mpu.setFilterBandwidth(MPU6050_BAND_184_HZ);
    delay(100);

    digitalWrite(2, LOW);
    xTaskCreate(&taskControl, "task_control", 4096, NULL, 10, NULL);
}
```

```
void loop() {
    delay(5000);
}

void PinSetup() {
    // Configuração dos pinos
    pinMode(IN1, OUTPUT);
    pinMode(IN2, OUTPUT);
    pinMode(IN3, OUTPUT);
    pinMode(IN4, OUTPUT);

    // Configuração do PWM
    ledcSetup(L_CHANNEL, 50000, 8);
    ledcSetup(R_CHANNEL, 50000, 8);
    ledcAttachPin(ENA, L_CHANNEL);
    ledcAttachPin(ENB, R_CHANNEL);
    ledcWrite(L_CHANNEL, 0);
    ledcWrite(R_CHANNEL, 0);

    // Configuração dos encoders
    ESP32Encoder::useInternalWeakPullResistors = puType::up;
    encoder_r.attachFullQuad(ENCODER_C1_R, ENCODER_C2_R);
    encoder_l.attachFullQuad(ENCODER_C1_L, ENCODER_C2_L);
    encoder_r.clearCount();
    encoder_l.clearCount();

    pinMode(2, OUTPUT);
    digitalWrite(2, LOW);
}

void taskBlinkLed(void* pvParameter)
{
    pinMode(2, OUTPUT);
    pinMode(0, INPUT);

    unsigned long init = millis();
    bool state = true;
    startMode = INIT;

    while (millis() - init <= 5000) {
```

```
    digitalWrite(2, state);
    state = !state;

    if (digitalRead(0) == LOW) {
        startMode = RUN;
        break;
    }

    vTaskDelay(pdMS_TO_TICKS(100));
}

if (startMode == INIT) startMode = CONFIG;
vTaskDelete(NULL);
}
```

- tools.cpp

```
#include "tools.h"
#include <ESP32Encoder.h>

float mapfloat(float x, float in_min, float in_max,
               float out_min, float out_max)
{
    return (x-in_min)*(out_max-out_min)/(in_max-in_min)+out_min;
}

void motor(int PWM, int channel) {
    int p1, p2;
    if (channel == L_CHANNEL) {
        p1 = IN1;
        p2 = IN2;
    } else {
        p1 = IN3;
        p2 = IN4;
    }
    bool PWM_Zero = PWM == 0 ? true : false;
    PWM = constrain(PWM, -100, 100);
    PWM = (PWM >= 0 ? 1 : -1) * mapfloat(abs(PWM), 0, 100, MIN_PWM, 100);
    PWM = map(PWM, -100, 100, -255, 255);
    if (PWM > 0) {
        digitalWrite(p1, LOW);
        digitalWrite(p2, HIGH);
    } else {
        digitalWrite(p1, HIGH);
        digitalWrite(p2, LOW);
    }
}
```

```
    } else if (PWM < 0) {
        PWM = -PWM;
        digitalWrite(p1, HIGH);
        digitalWrite(p2, LOW);
    } else if (PWM_Zero) {
        digitalWrite(p1, LOW);
        digitalWrite(p2, LOW);
        PWM = 0;
    }
    ledcWrite(chanel, PWM);
}
```

- tools.h

```
#ifndef TOOLS
#define TOOLS

#include <Wire.h>
#include <MPU6050.h>

#define ENCODER_C1_R 32
#define ENCODER_C2_R 33

#define ENCODER_C1_L 14
#define ENCODER_C2_L 27

#define IN1 16
#define IN2 17
#define ENA 4

#define IN3 18
#define IN4 19
#define ENB 23

#define L_CHANNEL 0
#define R_CHANNEL 1

#define MIN_PWM 20

// #define USE_WIFI
```

```
float mapfloat(float x, float in_min, float in_max,
              float out_min, float out_max);
typedef enum { INIT, CONFIG, RUN, BUFFER_OVERFLOW } Modes;
void motor(int PWM, int channel);

class Kalman {
public:
    Kalman(float _dt) {
        angle = 0;
        bias = 0;
        dt = _dt;
        P[0][0] = 1;
        P[0][1] = 0;
        P[1][0] = 0;
        P[1][1] = 1;
    }
    float getAngle(float newAngle, float newRate) {
        float rate = newRate - bias;
        angle += dt * rate;

        P[0][0] += dt * (dt * P[1][1] - P[0][1] - P[1][0] + Q_angle);
        P[0][1] -= dt * P[1][1];
        P[1][0] -= dt * P[1][1];
        P[1][1] += Q_bias * dt;

        float S = P[0][0] + R_measure;
        float K[2];
        K[0] = P[0][0] / S;
        K[1] = P[1][0] / S;

        float y = newAngle - angle;
        angle += K[0] * y;
        bias += K[1] * y;

        float P00_temp = P[0][0];
        float P01_temp = P[0][1];

        P[0][0] -= K[0] * P00_temp;
        P[0][1] -= K[0] * P01_temp;
        P[1][0] -= K[1] * P00_temp;
```

```
P[1][1] -= K[1] * P01_temp;

    return angle;
}

float bias;

private:
    float dt;
    float angle;
    float P[2][2];
    float Q_angle = 0.001;      // angular data confidence
    float Q_bias = 0.005;       // angular velocity confidence
    float R_measure = 1.0;      // measure confidence
};

#endif
```

F Código Experimento de Modelagem

Este código está presente no repositório https://github.com/JessicaLuana1377/diciclo_autonomo.git, na pasta ‘Modelo’ → ‘ExperimentoArduinoIDE’. Ele é dividido em três arquivos:

- **ExperimentoArduinoIDE.ino**

```
#include "tools.h"

// CONST VARIABLES
// Experimento do corpo
const float f0 = 0.5;           // Frequência inicial
const float f1 = 15;            // Frequência final
#define MAX_PWM 30              // Máximo de PWM aplicado

// Experimento das rodas
// const float f0 = 0.2;         // Frequência inicial
// const float f1 = 20;          // Frequência final
// #define MAX_PWM 50            // Máximo de PWM aplicado

const int ts = 10;
const int T = 120;               // Tempo de uma simulação
const int uT = T*1000000;        // T em microsegundos
#define SIMULATIONS 2           // Quantidade de simulações

void setup() {

    Serial.begin(115200);
    delay(500);

    InitSetup();

    Serial.print("StartMode:");
    Serial.println(startMode);

    // Modo simulação - Apertar botão BOOT
    if (startMode == RUN) {
        digitalWrite(2, LOW);
        printf("Running Mode...\n");
    }
}
```

```
delay(2000);
xTaskCreate(&taskControl, "task_control", 4096, NULL, 4, NULL);
}

// Modo configuração (leitura, formatar) - esperar 5 segundos
else if (startMode == CONFIG) {
    digitalWrite(2, HIGH);
    printf("Configuration Mode
        (l:list, f:format, r:read and format, R:read only)\n");
}
}

void taskControl(void* pvParameter) {
    pinMode(2, OUTPUT);
    // thread para salvar os dados do experimento na flash
    xTaskCreate(&taskFlash, "task_flash", 4096, NULL, 2, NULL);

    Data data;                      // struct com os dados do experimento
    TickType_t init_loop_time;      // variavel para manter o ts
    unsigned long t0 = micros();    // tempo inicial da simulação
    unsigned long time = t0;        // tempo usado para calcular o PWM
    mpu.getEvent(&a, &g, &temp);
    a_pitch = atan2(a.acceleration.z, a.acceleration.y);
    float dtheta = a_pitch; // ângulo inicial do pêndulo
    while(1) {
        init_loop_time = xTaskGetTickCount();
        time = micros();

        float position_r = encoder_r.getCount();
        float position_l = encoder_l.getCount();

        mpu.getEvent(&a, &g, &temp);
        g_x = -(g.gyro.x + g_x_offset);
        dtheta = g_x;

        // calculo do tempo triangular
        int temp = (time - t0 + uT) % (2 * uT) - uT;
        float t = abs(temp) / usTos;
        // calculo da chirp
        int pwm_value = (int)(MAX_PWM *
```

```
sin(-2 * PI * f0 * f1 * T / (f1 - f0) *
    log(1 - (f1 - f0) / (f1 * T) * t));

motor(pwm_value, L_CHANNEL);
motor(-pwm_value, R_CHANNEL);

// salvar dados
data.t = (time - t0)/usT0s;
data.u = pwm_value;
data.yr = position_r;
data.yl = position_l;
data.freq = f0 * f1 * T / ((f0 - f1) * t + f1 * T);
data.dtheta = dtheta;
data_array[count % BUFFER_SIZE] = data;
count++;

// caso ocorra overflow - não costuma acontecer
if (startMode == BUFFER_OVERFLOW) {
    motor(0, L_CHANNEL);
    motor(0, R_CHANNEL);
    printf("Buffer Overflow! Stop Simulation\n");
    while (1) {
        digitalWrite(2, HIGH);
        vTaskDelay(pdMS_TO_TICKS(500));
        digitalWrite(2, LOW);
        vTaskDelay(pdMS_TO_TICKS(500));
    }
}

// fim das simulações
if (time - t0 >= uT*SIMULATIONS) {
    motor(0, L_CHANNEL);
    motor(0, R_CHANNEL);

    startMode = CONFIG;
    digitalWrite(2, HIGH);

    printf("End Control\n");
    printf("Configuration Mode
        (l:list, f:format, r:read and format, R:read only)\n");
    break;
}
```

```
}

    vTaskDelayUntil(&init_loop_time, pdMS_TO_TICKS(ts));
}

vTaskDelete(NULL);
}

void loop() {
    if (Serial.available() >= 1) {
        char command = Serial.read();

        if (command == 'l') {
            listDir(SPIFFS, "/", 0);
        }

        if (command == 'f') {
            SPIFFS.format();
            Serial.println("Memória formatada");
        }

        if (command == 'r' || command == 'R') {
            File root = SPIFFS.open("/");
            if(!root){
                Serial.println("- failed to open directory");
                return;
            }
            if(!root.isDirectory()){
                Serial.println(" - not a directory");
                return;
            }

            Data data_read;
            File file = root.openNextFile();
            while(file){
                if (!file.isDirectory()) {
                    String filename = "/" + String(file.name());
                    File file = SPIFFS.open(filename.c_str());
```

```
if(!file || file.isDirectory()){
    Serial.println("- failed to open file for reading");
    break;
}
if (!filename.equals(String(FILE_COUNT_BKP))) {
    Serial.println("Reading: " + filename);

    Serial.println("- read from file:");
    Serial.println("t;u;yr;yl;freq;dtheta");
    while(file.available()){
        // Serial.write(file.read());
        if (file.read((uint8_t*)&data_read,
                      sizeof(Data)) != sizeof(Data))
        {
            break;
        }
        sprintf(buffer, "%.4f;%d;%ld;%ld;%4f;%4f\n",
                data_read.t, data_read.u, data_read.yr, data_read.yl,
                data_read.freq, data_read.dtheta);
        Serial.write(buffer);
    }
}
file.close();
}
file = root.openNextFile();
}

if (command == 'r') {
    SPIFFS.format();
    Serial.println("Memória formatada");
}
}
}
```

- tools.cpp

```
#include "tools.h"

// GLOBALS VARIABLES
ESP32Encoder encoder_r;
```

```
ESP32Encoder encoder_1;

Data *data_array;
Modes startMode;

// MPU6050 mpu;
Adafruit_MPU6050 mpu;
float g_x_offset = -0.05 + 0.097;
sensors_event_t a, g, temp;
float alpha;
float dtheta, a_pitch, g_x;

char buffer[50];
int count;
int flash_count;
int file_count;

void listDir(fs::FS &fs, const char * dirname, uint8_t levels)
{
    Serial.printf("Listing directory: %s\r\n", dirname);

    File root = fs.open(dirname);
    if(!root){
        Serial.println("- failed to open directory");
        return;
    }
    if(!root.isDirectory()){
        Serial.println(" - not a directory");
        return;
    }

    File file = root.openNextFile();
    while(file){
        if(file.isDirectory()){
            Serial.print(" DIR : ");
            Serial.println(file.name());
            if(levels){
                listDir(fs, file.path(), levels -1);
            }
        } else {

```

```
        Serial.print(" FILE: ");
        Serial.print(file.name());
        Serial.print("\tSIZE: ");
        Serial.println(file.size());
    }
    file = root.openNextFile();
}
}

void motor(int PWM, int channel)
{
    int p1, p2;
    if (channel == L_CHANNEL)
    {
        p1 = IN1;
        p2 = IN2;
    } else {
        p1 = IN3;
        p2 = IN4;
    }
    PWM = constrain(PWM, -100, 100);
    PWM = map(PWM, -100, 100, -255, 255);
    if (PWM == 0)
    {
        digitalWrite(p1, LOW);
        digitalWrite(p2, LOW);
        return;
    } else if (PWM > 0)
    {
        digitalWrite(p1, LOW);
        digitalWrite(p2, HIGH);
    } else
    {
        PWM = -PWM;
        digitalWrite(p1, HIGH);
        digitalWrite(p2, LOW);
    }
    ledcWrite(channel, PWM);
}
```

```
int fileCountInit()
{
    int file_count = 0;

    File file = SPIFFS.open(FILE_COUNT_BKP, FILE_READ);
    if(!file || file.isDirectory()){
        file_count = 0;
    }
    while(file.available()){
        file.read((uint8_t*)&file_count, sizeof(file_count));
    }
    file.close();

    Serial.println("File count init = " + String(file_count));

    return file_count;
}

void PinSetup() // realiza o PinOut de todos os dispositivos
{
    // Configuração dos pinos
    pinMode(IN1, OUTPUT);
    pinMode(IN2, OUTPUT);
    pinMode(IN3, OUTPUT);
    pinMode(IN4, OUTPUT);

    // Configuração do PWM
    ledcSetup(L_CHANNEL, 200, 8);
    ledcSetup(R_CHANNEL, 200, 8);
    ledcAttachPin(ENA, L_CHANNEL);
    ledcAttachPin(ENB, R_CHANNEL);
    ledcWrite(L_CHANNEL, 0);
    ledcWrite(R_CHANNEL, 0);

    // Configuração dos encoders
    ESP32Encoder::useInternalWeakPullResistors = puType::up;
    encoder_r.attachFullQuad(ENCODER_C1_R, ENCODER_C2_R);
    encoder_l.attachFullQuad(ENCODER_C1_L, ENCODER_C2_L);
    encoder_r.clearCount();
    encoder_l.clearCount();
```

```
pinMode(2, OUTPUT);
digitalWrite(2, LOW);
}

void taskFlash(void* pvParameter)
{
    String filename = "/arquivo" + String(file_count) + ".csv";
    Serial.println(filename);
    File file = SPIFFS.open(filename, FILE_WRITE);
    if(!file){
        Serial.println("- failed to open file for writing");
        return;
    }

    Data data_read;
    while(1) {
        if (flash_count < count) {
            data_read = data_array[flash_count % BUFFER_SIZE];

            if (count - flash_count > BUFFER_SIZE) startMode = BUFFER_OVERFLOW;

            file.write((const uint8_t*)&data_read, sizeof(Data));

            flash_count++;
        } else if (startMode == CONFIG ||
                   (startMode == BUFFER_OVERFLOW && flash_count == count)) {
            file.close();

            file_count++;
            file = SPIFFS.open(FILE_COUNT_BKP, FILE_WRITE);
            if(!file || file.isDirectory()) {
                Serial.println("- failed to open file bkp for something");
            }
            file.write((uint8_t*)&file_count, sizeof(file_count));
            file.close();

            printf("End Flash\n");
            break;
        }
    }
}
```

```
vTaskDelay(pdMS_TO_TICKS(1));
}

vTaskDelete(NULL);
}

void taskBlinkLed(void* pvParameter)
{
    pinMode(2, OUTPUT);
    pinMode(0, INPUT);

    unsigned long init = millis();
    bool state = true;
    startMode = INIT;

    while (millis() - init <= 5000) {
        digitalWrite(2, state);
        state = !state;

        if (digitalRead(0) == LOW) {
            startMode = RUN;
            break;
        }
    }

    vTaskDelay(pdMS_TO_TICKS(100));
}

if (startMode == INIT) startMode = CONFIG;

vTaskDelete(NULL);
}

void InitSetup()
{
    // Inicialização de variáveis
    startMode = INIT;
    alpha = 0.98;
    count = 0;
    flash_count = 0;
    file_count = 0;
```

```
PinSetup();

Wire.begin();

delay(1000);

if (!mpu.begin()) {
    Serial.println("Failed to find MPU6050 chip");
    while (1) {
        delay(10);
    }
}

mpu.setAccelerometerRange(MPU6050_RANGE_8_G);
mpu.setGyroRange(MPU6050_RANGE_1000_DEG);
mpu.setFilterBandwidth(MPU6050_BAND_184_HZ);
// calibrate_MPU();
delay(1000);

if(!SPIFFS.begin(FORMAT_SPIFFS_IF_FAILED)) {
    Serial.println("Falha ao montar SPIFFS");
    return;
}
file_count = fileCountInit();

data_array = (Data*)heap_caps_malloc(BUFFER_SIZE, sizeof(Data),
                                     MALLOC_CAP_8BIT | MALLOC_CAP_INTERNAL);
if (data_array == NULL) {
    Serial.println("Malloc for 'data_array' has failed!");
    return;
}

// Serial.println("Criando a TaskBlinkLed");
xTaskCreate(&taskBlinkLed, "blink_led", 2048, NULL, 1, NULL);
while (startMode == INIT) delay(100);
}

• tools.h

#ifndef TOOLS
#define TOOLS
```

```
#include <Wire.h>
#include <Adafruit_MPU6050.h>
#include <Adafruit_Sensor.h>

#include <stdio.h>
#include <freertos/FreeRTOS.h>
#include <freertos/task.h>
#include <esp_system.h>
#include <ESP32Encoder.h>
#include <math.h>
#include "FS.h"
#include "SPIFFS.h"

// Pinos dos encoders
#define ENCODER_C1_R 32
#define ENCODER_C2_R 33
#define ENCODER_C1_L 14
#define ENCODER_C2_L 27

// Pinos dos motores
#define IN1 16
#define IN2 17
#define ENA 4

#define IN3 18
#define IN4 19
#define ENB 23

// Canais de PWM
#define L_CHANNEL 0
#define R_CHANNEL 1

// Definições para o arquivo na flash
#define FORMAT_SPIFFS_IF_FAILED true
#define BUFFER_SIZE 2000
#define FILE_COUNT_BKP "/file_count.bkp"

// Microssegundos para segundos
#define usTOS 1000000.0
```

```
typedef struct {
    float t;
    int u;
    long int yr;
    long int yl;
    float freq;
    float dtheta;
} Data;

// Modo de operação do ESP32
typedef enum { INIT, CONFIG, RUN, BUFFER_OVERFLOW } Modes;

// GLOBALS VARIABLES
extern ESP32Encoder encoder_r;
extern ESP32Encoder encoder_l;

extern Data *data_array;
extern Modes startMode;

extern Adafruit_MPU6050 mpu;
extern float g_x_offset;
extern sensors_event_t a, g, temp;
extern float alpha;      // filtro para leitura
extern float dtheta, a_pitch, g_x; // ângulo lido

// variaveis para arquivo
extern char buffer[50];
extern int count;
extern int flash_count;
extern int file_count;

// lista os arquivos presentes na flash
void listDir(fs::FS &fs, const char * dirname, uint8_t levels);
// aplica o PWM em um motor
void motor(int PWM, int channel);
// inicializa o arquivo fileCount
int fileCountInit();
```

```
void InitSetup () ; // inicialização  
  
// thread que salva os valores na flash  
void taskFlash(void* pvParameter);  
// thread que pisca o led e espera o acionamento do botão  
void taskBlinkLed(void* pvParameter);  
  
#endif // TOOLS
```

G Código do Controlador Compensador de Avanço e Atraso

Este código está presente no repositório https://github.com/JessicaLuana1377/diciclo_autonomo.git, na pasta ‘Controle’ → ‘controle_lead_lag’. Ele é dividido em três arquivos:

- **controle_lead_lag.ino**

```
#include "tools.h"

// CONST VARIABLES
const int ts = 5;
const float alpha_w = 0.5; // filtro da velocidade das rodas
const float alpha_b = 0.95; // filtro da derivada de theta
const float alpha = 0.98; // filtro de theta
const float r_w = 0.03; // raio da roda
const int reduction = 25; // 1 dado será salvo em ‘reduction’ dados

const float k_A_theta[3] = {1.0000, -1.7639, 0.7639};
const float k_B_theta[3] = {-72.9757, 136.5322, -63.8437};
const float k_A_phi[3] = {1.0000, -1.3270, 0.4037};
const float k_B_phi[3] = {0, 0.0323, -0.0321};

float position_r, position_l, phi, theta;
float last_position_r, last_position_l, last_theta, ur, ul;
float dwheel_f_r, dwheel_f_l, dtheta_f, dwheel;
int pwm_value;

float u[3] = {0};
float error_theta[3] = {0};
float theta_ref[3] = {-0.062, -0.062, -0.062};
float error_phi[3] = {0};

void setup() {
    Serial.begin(115200);
    delay(500);

    InitSetup();

    delay(100);;
```

```
digitalWrite(2, LOW);
delay(3000);

if (!BootPressed){
    printf("Running Mode...\n");

    // Running Mode
    Data data;
    TickType_t init_loop_time;
    unsigned long t0 = micros();

    while(!BootPressed) {
        init_loop_time = xTaskGetTickCount();

        position_r = 2*PI*encoder_r.getCount()/(float)(4*7*30);
        position_l = 2*PI*encoder_l.getCount()/(float)(4*7*30);
        phi = (position_l - position_r)/2;

        mpu.getEvent(&a, &g, &temp);
        g_x = -(g.gyro.x + g_x_offset);
        a_pitch = atan2(a.acceleration.z, a.acceleration.y);

        // direita
        dwheel = (position_r - last_position_r)/(ts/1e3);
        dwheel_f_r = alpha_w*dwheel_f_r + (1-alpha_w)*dwheel;

        // esquerda
        dwheel_f_l = alpha_w*dwheel_f_l +
                      (1-alpha_w)*(position_l - last_position_l)/(ts/1e3);

        if (true) {
            error_phi[0] = 0 - phi;

            theta_ref[0] = -1*(k_A_phi[1]*theta_ref[1] +
                               k_A_phi[2]*theta_ref[2]) +
                           1*(k_B_phi[1]*error_phi[1] +
                               k_B_phi[2]*error_phi[2]);
            theta_ref[0] = constrain(theta_ref[0], -PI/4, PI/4);
        }
    }
}
```

```
    error_phi[2] = error_phi[1];
    error_phi[1] = error_phi[0];

    theta_ref[2] = theta_ref[1];
    theta_ref[1] = theta_ref[0];

}

// corpo
theta = alpha*(data.theta + g_x*(ts/1e3)) + (1-alpha)*a_pitch;
dtheta_f = alpha_b*dtheta_f + (1-alpha_b)*(g_x);

error_theta[0] = theta_ref[0] - theta;

u[0] = -1*(k_A_theta[1]*u[1] + k_A_theta[2]*u[2]) +
1*(k_B_theta[0]*error_theta[0] + k_B_theta[1]*error_theta[1] +
k_B_theta[2]*error_theta[2]);
u[0] = constrain(u[0], -7.4, 7.4);

ul = u[0];
ur = -ul;

if (abs(theta) >= PI/4) {
    ur = 0; ul = 0;
    theta_ref[0] = 0;
}

pwm_value = map(ur, -7.4, 7.4, -100, 100);
motor(pwm_value, R_CHANNEL);

pwm_value = map(ul, -7.4, 7.4, -100, 100);
motor(pwm_value, L_CHANNEL);

data.t = (micros() - t0)/usT0s;
data.u = pwm_value;
data.theta = theta;
data.dtheta = dtheta_f;
data.wheel = position_r;
data.dwheel = dwheel;
data.dwheel_f = dwheel_f_r;
```

```
// Salvar dados
if (count_save >= BUFFER_SIZE) {
    BlueLED = !BlueLED;
    digitalWrite(BLUE_LED, BlueLED);
} else if(count % reduction == 0){
    data_array[count_save % BUFFER_SIZE] = data;
    count_save++;
    // printf("Tempo: %0.4f\n", data.t);
}

last_theta = theta;
last_position_r = position_r;
last_position_l = position_l;

error_theta[2] = error_theta[1];
error_theta[1] = error_theta[0];

u[2] = u[1];
u[1] = u[0];

count++;
vTaskDelayUntil(&init_loop_time, pdMS_TO_TICKS(ts));
}

motor(0, R_CHANNEL);
motor(0, L_CHANNEL);

printf("Saving data\n");
DataSave();
}

printf("Configuration Mode
(l:list, f:format, r:read, R:read and format)\n\n");
digitalWrite(BLUE_LED, HIGH);
}

void loop() {
    if (Serial.available() >= 1) {
        char command = Serial.read();

        if (command == 'l') {
            listDir(SPIFFS, "/", 0);
        }
    }
}
```

```
}

if (command == 'f') {
    SPIFFS.format();
    Serial.println("Memória formatada");
}

if (command == 'r' || command == 'R') {
    File root = SPIFFS.open("/");
    if(!root){
        Serial.println("- failed to open directory");
        return;
    }
    if(!root.isDirectory()){
        Serial.println(" - not a directory");
        return;
    }

    Data data_read;
    File file = root.openNextFile();
    while(file){
        if (!file.isDirectory()) {
            String filename = "/" + String(file.name());

            File file = SPIFFS.open(filename.c_str());

            if(!file || file.isDirectory()){
                Serial.println("- failed to open file for reading");
                break;
            }
            if (!filename.equals(String(FILE_COUNT_BKP))) {
                Serial.println("Reading: " + filename);

                Serial.println("- read from file:");

                Serial.println("t;u;theta;dtheta;wheel;dwheel;dwheel_f");
                while(file.available()){
                    // Serial.write(file.read());
                    if (file.read((uint8_t*)&data_read,
                        sizeof(Data)) != sizeof(Data))
```

```
{  
    break;  
}  
printf("%.4f;%d;%.4f;%.4f;%.4f;%.4f;%.4f\n",  
    data_read.t, data_read.u, data_read.theta, data_read.dtheta,  
    data_read.wheel, data_read.dwheel, data_read.dwheel_f);  
}  
}  
file.close();  
}  
file = root.openNextFile();  
}  
  
if (command == 'R') {  
    SPIFFS.format();  
    Serial.println("Memória formatada");  
}  
}  
}  
}
```

- tools.cpp

```
#include "Arduino.h"  
#include "SPIFFS.h"  
#include "tools.h"  
  
// GLOBALS VARIABLES  
ESP32Encoder encoder_r;  
ESP32Encoder encoder_l;  
  
Data data_array[BUFFER_SIZE];  
  
// MPU6050 mpu;  
Adafruit_MPU6050 mpu;  
float g_x_offset = -0.05 + 0.097;  
sensors_event_t a, g, temp;  
float dtheta, a_pitch, g_x;  
  
int count = 0;  
int count_save = 0;
```

```
int flash_count = 0;
int file_count = 0;

volatile bool BootPressed = false;
volatile bool BlueLED = false;

void listDir(fs::FS &fs, const char * dirname, uint8_t levels)
{
    Serial.printf("Listing directory: %s\r\n", dirname);

    File root = fs.open(dirname);
    if(!root){
        Serial.println("- failed to open directory");
        return;
    }
    if(!root.isDirectory()){
        Serial.println(" - not a directory");
        return;
    }

    File file = root.openNextFile();
    while(file){
        if(file.isDirectory()){
            Serial.print(" DIR : ");
            Serial.println(file.name());
            if(levels){
                listDir(fs, file.path(), levels -1);
            }
        } else {
            Serial.print(" FILE: ");
            Serial.print(file.name());
            Serial.print("\tSIZE: ");
            Serial.println(file.size());
        }
        file = root.openNextFile();
    }
}

void motor(int PWM, int channel) {
    int p1, p2;
```

```
if (chanel == L_CHANNEL) {  
    p1 = IN1;  
    p2 = IN2;  
} else {  
    p1 = IN3;  
    p2 = IN4;  
}  
PWM = constrain(PWM, -100, 100);  
PWM = map(PWM, -100, 100, -255, 255);  
if (PWM > 0) {  
    digitalWrite(p1, LOW);  
    digitalWrite(p2, HIGH);  
} else if (PWM < 0) {  
    PWM = -PWM;  
    digitalWrite(p1, HIGH);  
    digitalWrite(p2, LOW);  
} else {  
    digitalWrite(p1, LOW);  
    digitalWrite(p2, LOW);  
}  
ledcWrite(chanel, PWM);  
}  
  
void PinSetup() {  
    // Configuração dos pinos  
    pinMode(IN1, OUTPUT);  
    pinMode(IN2, OUTPUT);  
    pinMode(IN3, OUTPUT);  
    pinMode(IN4, OUTPUT);  
  
    // Configuração dos motores  
    motor(0, L_CHANNEL);  
    motor(0, R_CHANNEL);  
  
    // Configuração do PWM  
    ledcSetup(L_CHANNEL, 200, 8);  
    ledcSetup(R_CHANNEL, 200, 8);  
    ledcAttachPin(ENA, L_CHANNEL);  
    ledcAttachPin(ENB, R_CHANNEL);  
    ledcWrite(L_CHANNEL, 0);  
}
```

```
ledcWrite(R_CHANNEL, 0);

// Configuração dos encoders
ESP32Encoder::useInternalWeakPullResistors = puType::up;
encoder_r.attachFullQuad(ENCODER_C1_R, ENCODER_C2_R);
encoder_l.attachFullQuad(ENCODER_C1_L, ENCODER_C2_L);
encoder_r.clearCount();
encoder_l.clearCount();

pinMode(BLUE_LED, OUTPUT);
digitalWrite(BLUE_LED, BlueLED);
}

void InitSetup() {
PinSetup();

Wire.begin();

delay(1000);

if (!mpu.begin()) {
Serial.println("Failed to find MPU6050 chip");
while (1) {
delay(10);
}
}
mpu.setAccelerometerRange(MPU6050_RANGE_8_G);
mpu.setGyroRange(MPU6050_RANGE_1000_DEG);
mpu.setFilterBandwidth(MPU6050_BAND_184_HZ);
// calibrate_MPU();
delay(1000);

if(!SPIFFS.begin(FORMAT_SPIFFS_IF_FAILED)) {
Serial.println("Falha ao montar SPIFFS");
return;
}
file_count = fileCountInit();

pinMode(BUTTON_BOOT, INPUT_PULLUP);
attachInterrupt(digitalPinToInterrupt(BUTTON_BOOT),
```

```
        StopControl, FALLING);  
    }  
  
void DataSave () {  
    String filename = "/arquivo" + String(file_count) + ".csv";  
    Serial.println(filename);  
    File file = SPIFFS.open(filename, FILE_WRITE);  
    if(!file){  
        Serial.println("- failed to open file for writing");  
        return;  
    }  
  
    Data data_read;  
    while(1) {  
        if (flash_count < count_save) {  
            data_read = data_array[flash_count % BUFFER_SIZE];  
  
            file.write((const uint8_t*)&data_read, sizeof(Data));  
  
            flash_count++;  
        } else {  
            file.close();  
  
            file_count++;  
            file = SPIFFS.open(FILE_COUNT_BKP, FILE_WRITE);  
            if(!file || file.isDirectory()) {  
                Serial.println("- failed to open file bkp for something");  
            }  
            file.write((uint8_t*)&file_count, sizeof(file_count));  
            file.close();  
  
            printf("End\n");  
            break;  
        }  
    }  
  
    int fileCountInit() {  
        int file_count = 0;
```

```
File file = SPIFFS.open(FILE_COUNT_BKP, FILE_READ);
if(!file || file.isDirectory()){
    file_count = 0;
}
while(file.available()){
    file.read((uint8_t*)&file_count, sizeof(file_count));
}
file.close();

Serial.println("File count init = " + String(file_count));

return file_count;
}

void IRAM_ATTR StopControl() {
    BootPressed = true;
}

void calibrate_MPU() {
    float x, y, z;
    for (int i=0; i<3000; i++) {
        mpu.getEvent(&a, &g, &temp);
        x += g.gyro.x;
        y += g.gyro.y;
        z += g.gyro.z;
        delay(3);
    }
    x /= 3000;
    y /= 3000;
    z /= 3000;

    g_x_offset = x;
}

• tools.h

#ifndef TOOLS
#define TOOLS

#include <stdio.h>
#include <freertos/FreeRTOS.h>
```

```
#include <freertos/task.h>
#include <esp_system.h>
#include <ESP32Encoder.h>
#include <math.h>
#include "FS.h"
#include "SPIFFS.h"

#include <Adafruit_MPU6050.h>
#include <Adafruit_Sensor.h>
#include <Wire.h>

#include <driver/pcnt.h>

// Pinos dos encoderes
#define ENCODER_C1_R 32
#define ENCODER_C2_R 33
#define ENCODER_C1_L 14
#define ENCODER_C2_L 27

// Pinos dos motores
#define IN1 16
#define IN2 17
#define ENA 4
#define IN3 18
#define IN4 19
#define ENB 23

// Canais de PWM
#define L_CHANNEL 0
#define R_CHANNEL 1

// Definições para o arquivo na flash
#define FORMAT_SPIFFS_IF_FAILED true
#define BUFFER_SIZE 2000
#define FILE_COUNT_BKP "/file_count.bkp"

#define usT0s 1000000.0
#define BUTTON_BOOT 0
#define BLUE_LED 2
extern volatile bool BlueLED;
```

```
extern volatile bool BootPressed;

typedef struct {
    float t;
    float dwheel; //
    float dwheel_f; //
    float dtheta; //
    float theta; //
    float wheel; //
    int u; //
} Data;
extern Data data_array[BUFFER_SIZE];

// GLOBALS VARIABLES
extern ESP32Encoder encoder_r;
extern ESP32Encoder encoder_l;

extern Adafruit_MPU6050 mpu;
extern float g_x_offset;
extern sensors_event_t a, g, temp;
extern float dtheta, a_pitch, g_x; // ângulo lido

// variaveis para arquivo
extern int count;
extern int count_save;
extern int flash_count;
extern int file_count;

void listDir(fs::FS &fs, const char * dirname, uint8_t levels);
void motor(int PWM, int channel);
int fileCountInit();
void DataSave();
void InitSetup ();
void calibrate_MPU();
void IRAM_ATTR StopControl();
void taskFlash(void* pvParameter);
#endif
```

Referências