IPV - ESTGV |
Instituto Politécnico de Viseu
Escola Superior de Tecnologia e Gestão de Viseu

Patrick Daniel Nunes Almeida
Pedro Miguel Moreira Almeida

Projeto de conversão de gerador automóvel em motor síncrono autopilotado com controlo remoto por web server baseado em esp32

Relatório de projeto:

Licenciatura em Engenharia Eletrotécnica

Orientador:
Eng: Luís Pestana

Julho de 2023

AGRADECIMENTOS

Com a realização deste Relatório de Projeto final, não podemos deixar de agradecer a algumas pessoas que, direta ou indiretamente, nos ajudaram neste percurso tão importante da nossa vida pessoal e profissional.

Ao Engenheiro Luís Pestana, por todo o apoio, disponibilidade e ensinamentos que nos transmitiu durante esta jornada no curso de Eng. Eletrotécnica assim como na realização do Projeto Final.

Agradecer a todos os docentes do curso de Engenharia Eletrotécnica da ESTGV por a dedicação, ensinamentos e compromisso ao longo destes anos, pois foram fundamentais para nosso crescimento académico e profissional. Agradecemos por compartilharem o conhecimento e por nos incentivarem a melhorar.

Às instituições, empresas e profissionais que nos receberam, oferecendo visitas técnicas. Estas experiências enriquecedoras permitiram vivenciar a aplicação prática dos conhecimentos adquiridos em sala de aula, consolidando assim a nossa formação e despertando ainda mais o interesse pela Engenharia Eletrotécnica.

Por fim, mas não menos importante, agradecer também aos nossos familiares e amigos, por todo o apoio, motivação e força que nos deram, permitindo superar obstáculos e concluir esta jornada.

Resumo

No presente relatório procuramos expor o desenvolvimento do projeto, que tem como objetivo principal explorar a utilização da tecnologia IoT na monitorização e controlo remoto de equipamentos, com foco específico em motores elétricos.

Com este projeto pretendemos fazer a recuperação de um alternador de automóvel e efetuar alterações de forma a funcionar como um motor síncrono do tipo BLDC – motor de corrente continua sem escovas, com controlo de velocidade por potenciómetro motorizado. Será também realizada a implementação de controlo de um motor de passo a partir do microcontrolador ESP32 (o motor de passo destina-se a variar o valor do potenciómetro). Para efetuar esse controlo foi criado um servidor - web-server que permite a conexão Wi-Fi com o microcontrolador ESP32.

Por fim, será criado um sistema de monitorização de velocidade, que transmitirá os dados para um servidor remoto, possibilitando a visualização das grandezas monitorizadas, poderá evoluir para análise de vibrações e deteção de avarias. Serão utilizados recursos disponíveis gratuitamente na internet, como o mencionado web-server.

ÍNDICE GERAL

ÍNDICE DE FIGURAS

ÍNDICE DE QUADROS

ABREVIATURAS E SIGLAS

UC	Unidade curricular
Iot	Internet of Things
BLDC	Brushless Direct Current
Wi-Fi	Wireless Fidelity
UART	Universal Asynchronous Receiver/Transmitter
PWM	Pulse Width Modulation
SPICE	Simulation Program With Integrated Circuit Emphasis
USB	Universal Serial Bus
GND	Ground
RAM	Random Access Memory
ADC	Analogic to Digital converter
DCA	Digital to Analogic converter
DC	Direct Current
HTML	HyperText Markup Language

1. Introdução

1.1 Enquadramento

Foi-nos proposto para Projeto final a conversão de um gerador automóvel em motor síncrono autopilotado (BLDC) com controlo remoto por web server baseado em microcontrolador esp32, é do nosso ponto de vista, um projeto interessante onde se demonstra um pouco esta evolução tecnológica ao longo das últimas décadas.
A base deste projeto é o microcontrolador Esp32, que apresenta algumas vantagens em relação a outos microcontroladores

A crescente adoção da tecnologia IoT tem impulsionado a monitorização de máquinas e o controlo remoto de equipamentos em diversas áreas, possibilitando o reaproveitamento de equipamentos, conferindo-lhes uma segunda vida por meio de requalificação com novos dispositivos de controlo e monitorização.

Para os engenheiros, o conhecimento e domínio de ferramentas relacionadas com a monitorização e controlo remoto de equipamentos são aspetos cada vez mais valorizados. Essas competências permitem não apenas a otimização de processos e a redução de custos, mas também contribuem para a tomada de decisões mais assertivas e a prevenção de falhas em tempo útil.

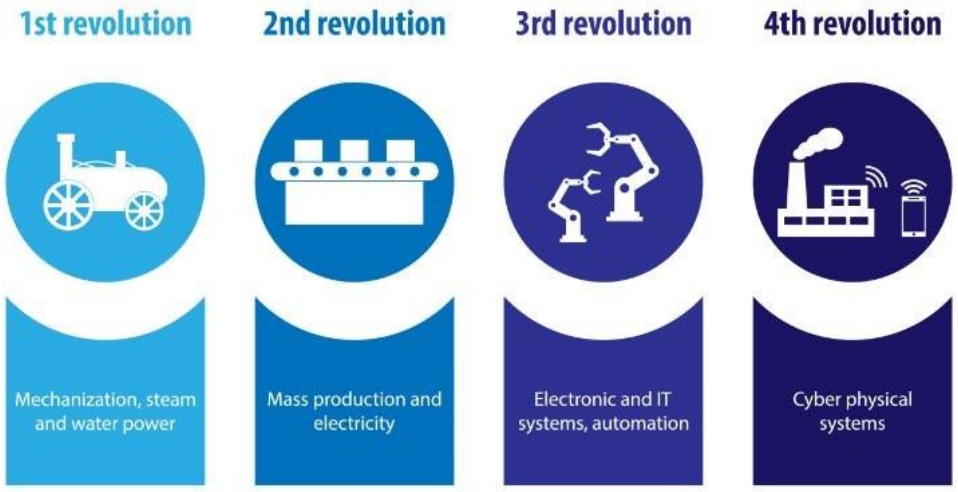


Figura 1 – Evolução Tecnológica

No âmbito deste relatório, exploraremos a aplicação prática do IoT na monitorização de máquinas e no controlo remoto de equipamentos. O nosso objetivo é analisar os benefícios, os desafios e as melhores práticas associadas a essa abordagem, assim como identificar oportunidades de melhoria e possíveis aplicações em diferentes setores industriais.

Ao longo do relatório, utilizaremos uma metodologia que combina pesquisa bibliográfica, estudos de caso e análise de dados. Dividiremos o conteúdo em seções distintas, abordando aspetos como o funcionamento do IoT, os métodos de monitorização de máquinas, os dispositivos de controlo remoto disponíveis no mercado e os benefícios obtidos com a implementação dessas tecnologias.

Esperamos que este relatório proporcione uma compreensão aprofundada das aplicações do IoT na monitorização de máquinas e no controlo remoto de equipamentos, bem como inspire reflexões e ações voltadas para a implementação dessas soluções em diferentes contextos industriais.

1.2 Objetivos

Com base nas informações fornecidas anteriormente, os objetivos do projeto podem ser resumidos da seguinte forma:

Recolher informação sobre o microcontrolador Esp32 e as suas capacidades, tipo de sinais utilizados e formas de comunicação. Isso implica entender as características e funcionalidades do microcontrolador específico utilizado no projeto.

Implementar pequenos exemplos de recolha de dados e transmissão para o exterior (webserver ou telemóvel) utilizando Wi-Fi ou Bluetooth. Isso envolve demonstrar a capacidade de recolher dados dos equipamentos monitorados e transmiti-los para um servidor remoto ou dispositivo móvel.

Recuperar o alternador de um automóvel e reconfigurá-lo para funcionar como motor síncrono do tipo BLDC com controlo de velocidade por potenciómetro. Isso implica transformar um componente existente (alternador de um automóvel) em um motor síncrono controlado eletronicamente.

Implementação de controle de motor de passo a partir do ESP32.

Criação de um sistema de monitorização de velocidade e transmissão para servidor remoto para visualização de grandezas monitorizadas. Isso envolve desenvolver um sistema que permita monitorar a velocidade dos motores e transmitir esses dados para um servidor remoto.

Esses são os objetivos delineados para o projeto com base nas informações fornecidas. É importante adaptar e ajustar esses objetivos conforme necessário, levando em consideração os requisitos e recursos disponíveis para a realização do projeto.

1.3 Estrutura deste documento

Este relatório está estruturado em 6 capítulos e 11 anexos. Neste primeiro é efetuada uma breve descrição do projeto e dos seus objetivos.

No segundo capítulo é abordado o funcionamento do alternador e como é que este pode ser transformado num motor do tipo BLCD.

No terceiro capítulo abordamos o que é o microcontrolador Esp, caraterísticas e as razões de escolha em comparação a outros microcontroladores.

No quarto capítulo é exposto a implementação do sequenciador de fases, os seus problemas e as respetivas soluções.

No quinto capítulo é descrito a programação envolvente durante a realização do projeto e os seus respetivos resultados.

No sexto capítulo encontra-se a conclusão onde é feito um resumo geral da aprendizagem ao longo desta UC.

2. Alternador / Motor CC

2.1 Introdução (Alternador)

Um alternador é um dispositivo eletromecânico que converte energia mecânica em energia elétrica. Normalmente utilizado em veículos automóveis para gerar eletricidade e recarregar a bateria do veículo enquanto o motor está em funcionamento.

O alternador é uma máquina elétrica de corrente alternada, normalmente a operar como gerador (daí advindo o seu nome – "alternador" - gerador de corrente alternada) sendo também uma máquina de tipo síncrono e que pode operar também como motor elétrico. A máquina funciona com base no princípio da indução eletromagnética e utiliza um campo magnético rotativo para gerar corrente elétrica.

O princípio de funcionamento básico da máquina como alternador é o seguinte:

O alternador possui um enrolamento de corrente continua na parte móvel da máquina - rotor, e que produz um campo magnético rotativo quando energizado e em movimento. O rotor é acionado pela correia do motor do veículo.

No estátor da máquina encontra-se um enrolamento trifásico, que quando sujeito a um campo girante (provocado pela bobine energizada do rotor em movimento), é alvo de força eletromotriz induzida, dando origem a um sistema trifásico de tensões.

Para utilização no automóvel, é necessário converter o sistema trifásico de tensões alternadas num sistema de corrente continua. Tal obtém-se por retificação. utilizando díodos retificadores (ponte retificadora trifásica) que permitem que a corrente flua em uma única direção.

O alternador também possui um regulador de tensão integrado, que controla a corrente elétrica gerada pelo alternador e mantém a tensão de saída dentro dos limites adequados para carregar a bateria do veículo e alimentar os sistemas elétricos.

Em resumo, um alternador é um tipo de motor síncrono que gera eletricidade por meio da indução eletromagnética. Ele utiliza um campo magnético rotativo gerado pelo rotor e bobines fixas do estator para produzir corrente elétrica alternada, que é retificada e regulada para alimentar os sistemas elétricos do veículo.

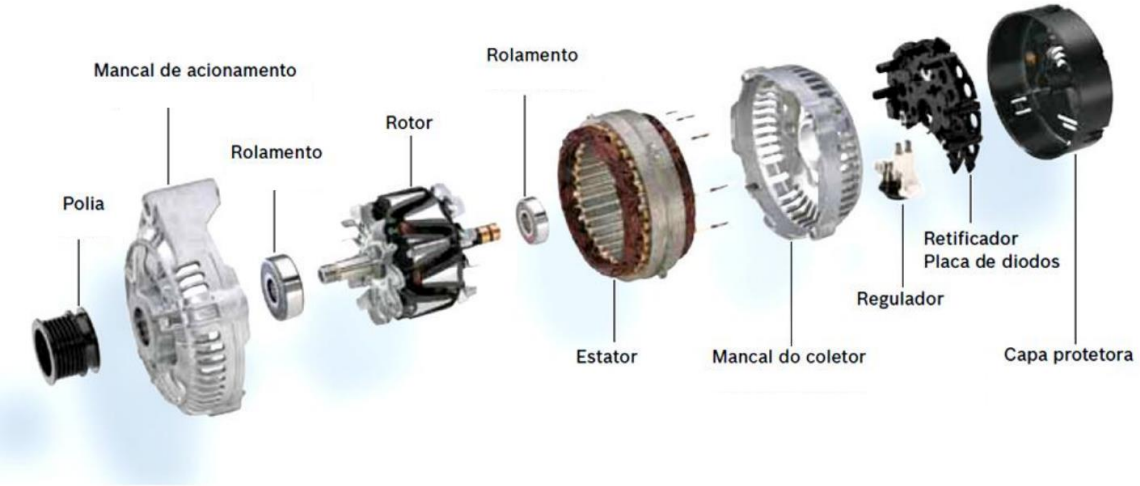


Figura 2 – Componentes de um Alterador

2.2 Transformar o alternador num Motor CC

Começamos por desmontar o alternador *(Figura 3)*, para realizar as modificações necessárias. Uma das primeiras etapas foi soldar um fio diretamente a cada escova no rotor, que seria utilizado para ligar uma pilha de 3.7V, para alimentar esse enrolamento.

O regulador *(Figura 4)*, desempenha um papel crucial no funcionamento do alternador ao controlar a tensão de saída do sistema elétrico do veículo, pois monitora a tensão e ajusta a corrente de excitação do rotor, garantindo uma tensão estável na saída do alternador. Isso é essencial para carregar adequadamente a bateria e garantir o funcionamento correto dos dispositivos elétricos.

No entanto, para o nosso projeto, o regulador não era necessário e decidimos aproveitar apenas a estrutura onde as escovas estão fixas e foram desfeitas as ligações internas no regulador, removendo os seus componentes. Essa modificação permitiu-nos simplificar o sistema e adaptar o alternador para a nova finalidade de motor, eliminando a necessidade do regulador no nosso projeto.

Figura 3 – Alternador Desmontado

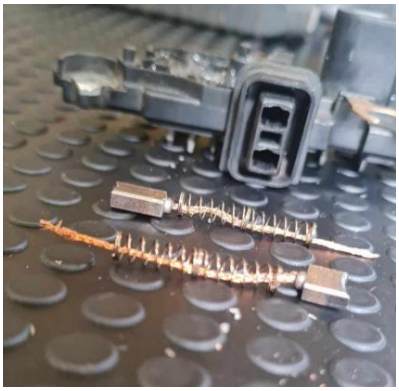


Figura 4 - Regulador

Após soldarmos os fios, realizamos a medição da resistência dos três enrolamentos do estator e constatámos que tinha cerca 0.1ohm cada um (valor extremamente reduzido). Essa medida é importante para verificar se havia algum desequilíbrio entre elas, assim como identificar os 3 enrolamentos.

Procedemos à ligação em estrela do circuito, pois este tipo de ligação é normalmente mais utilizado em sistemas de baixas tensões permitindo uma corrente de linha menor diminuindo assim a queda de tensão nas linhas e as perdas de energia

Após essas etapas, foi dada uma limpeza geral no alternador e procedemos a montagem, no entanto, não foi montado o retificador, que é a placa de díodos responsável pela retificação da corrente alternada em corrente contínua, uma vez que não era necessário na transformação do funcionamento da máquina de alternador para motor.

Na *figura 5* podemos ver o esquema de ligação em estrela:

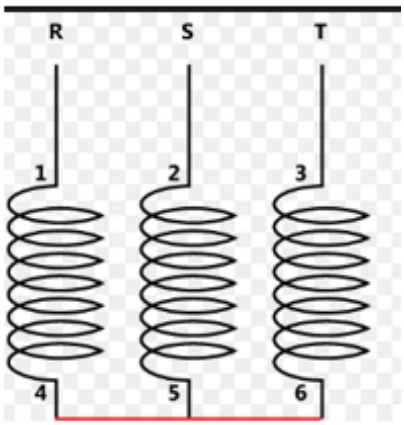


Figura 3 - Ligação Estrela

Na ligação em estrela (também conhecida como conexão em Y), os 3 enrolamentos são conectados num ponto comum, formando uma configuração em forma de estrela. O enrolamento é conectado entre uma extremidade do mesmo e o ponto comum.

Na ligação em triângulo (também conhecida como conexão em Δ), os enrolamentos são conectados numa configuração em forma de triângulo, ou seja, cada enrolamento está conectado em série com o próximo, formando um circuito fechado.

A principal diferença prática entre as duas ligações está na tensão entre fase e a tensão de linha. Na ligação em estrela, a tensão de fase é menor em relação à tensão de linha, enquanto na ligação em triângulo, a tensão de fase é igual à tensão de linha. Essa diferença é importante ao projetar e dimensionar sistemas elétricos, especialmente em termos de distribuição de tensão e potência.

Outra diferença está nas correntes de linha. Na ligação em estrela, as correntes de linha são iguais às correntes de fase. Na ligação em triângulo, as correntes de linha são $\sqrt{3}$ vezes superiores às correntes de fase.

.

3. ESP32s

3.1 Introdução (ESP32s)

O NodeMCU ESP32S é uma placa de desenvolvimento que combina as poderosas capacidades do microcontrolador ESP32 com recursos adicionais para facilitar o desenvolvimento de projetos IoT e eletrónicos em geral.

Uma das principais características do NodeMCU ESP32S é o seu microcontrolador ESP32, que possui um processador dual-core de alto desempenho. Isso permite a execução de múltiplas tarefas e a implementação de projetos mais complexos. Possui ainda uma memória RAM com cerca de 520KB e um Clock de 160-240MHz o que permite uma maior capacidade de processamento, uma resposta mais rápida, melhor desempenho em aplicações paralelas, e suporta taxas de transmissão mais altas que por sua vez é um aspeto importante quando se utiliza protocolos de comunicação rápida como o Wi-Fi, Ethernet e USB.

Oferece ainda conectividade Wi-Fi e Bluetooth integradas, o que permite a comunicação sem fio com outros dispositivos e redes, também possui uma ampla gama de pinos de E/S digitais e analógicos, permitindo a conexão de sensores, atuadores e outros periféricos. Isso proporciona flexibilidade na criação de projetos e na interação com o ambiente ao redor.

Outra característica notável do NodeMCU ESP32S é a sua compatibilidade com a plataforma de desenvolvimento Arduíno. Isso significa que os projetistas podem aproveitar a vasta biblioteca de código e recursos disponíveis na plataforma Arduíno para programar o NodeMCU com facilidade.

Além disso, o NodeMCU ESP32S oferece uma interface USB para programação e depuração, facilitando a conexão com um computador para desenvolvimento e teste de código.

Em resumo, o NodeMCU ESP32S é uma placa de desenvolvimento versátil e poderosa, ideal para projetos eletrónicos e em especial os relacionados com IoT, tirando partido do microcontrolador ESP32, conectividade Wi-Fi e Bluetooth, pinagem abrangente e compatibilidade com Arduíno.

3.2 Pinos ESP32s

A placa de desenvolvimento NodeMCU ESP32S é baseada no microcontrolador ESP32, e os principais pinos disponíveis são:

- Pinos de Alimentação: Possui pinos de alimentação de 3.3V, 5V e GND (terra) para fornecer energia aos componentes conectados.

- Pinos de Comunicação Série: Suporta comunicação serial usando os pinos RX (receção) e TX (transmissão). Esses pinos podem ser usados para comunicação com outros dispositivos por meio de protocolos como UART ou RS-232.

- Pinos de E/S Digital: Existem vários pinos digitais que podem ser usados para entrada ou saída digital. Eles podem ser configurados como entradas para ler o estado de sensores ou como saídas para acionar atuadores, LEDs e outros dispositivos.

- Pinos de PWM: Possui pinos com capacidade de PWM, que permitem gerar sinais para controlar a intensidade luminosa de LEDs, velocidade de motores e outros dispositivos que suportam controle por PWM.

- Pinos Analógicos: O NodeMCU ESP32S possui alguns pinos que podem ser usados para leitura de sinais analógicos (ADC). Eles são capazes de converter uma tensão analógica em um valor digital para fins de medição ou controle.

- Conectividade sem fio, como Wi-Fi e Bluetooth para transmitir e receber dados sem fio, permitindo a comunicação com outros dispositivos ou redes.

Na figura abaixo podemos ver onde se encontram os pintos de E/S do NodeMCU ESP32S:

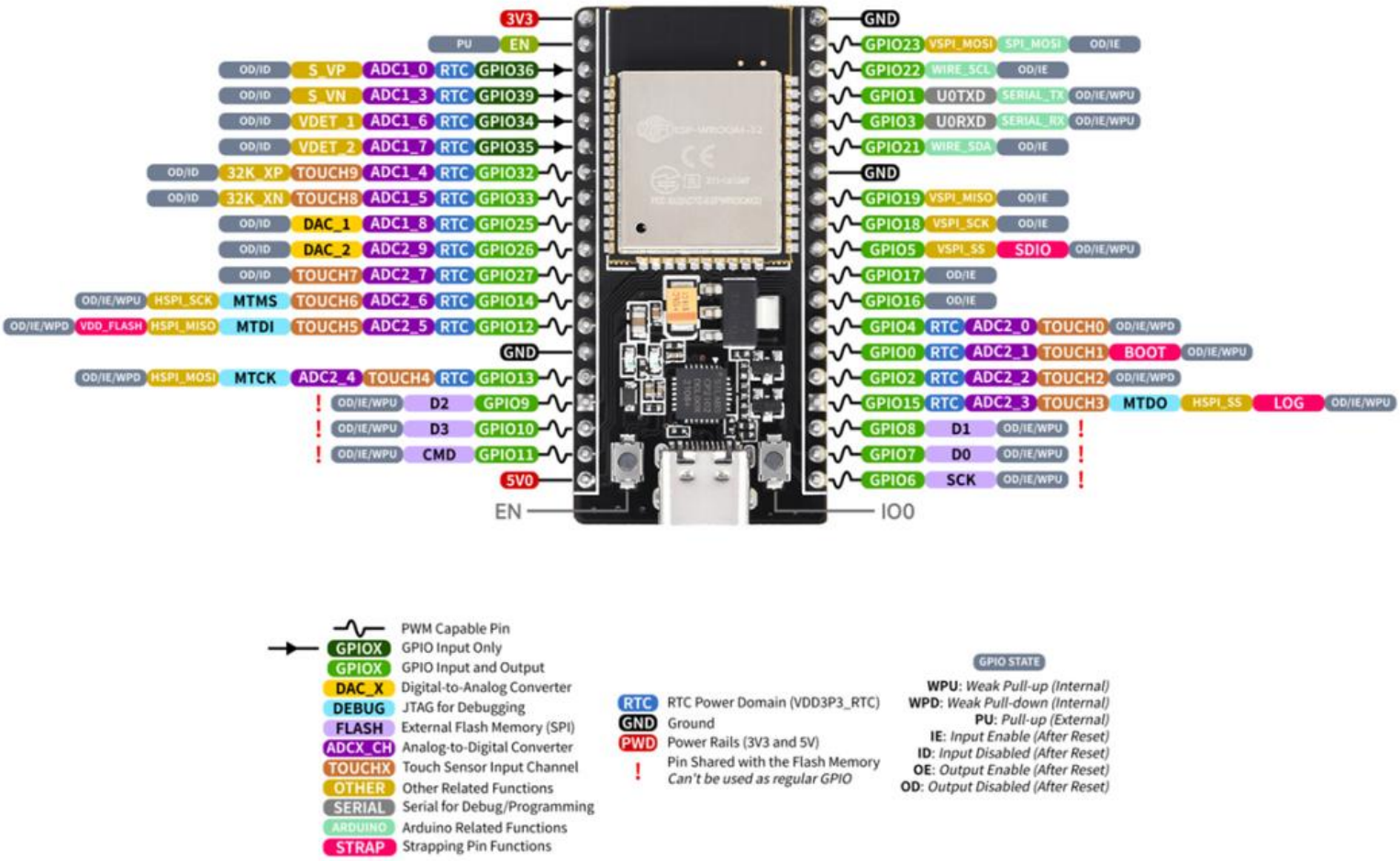


Figura 4 – Pinos E/S do NodeMCU ESP32S

3.3 Caraterísticas do ESP32s

As principais características do NodeMCU ESP32S:

- Possui um processador dual-core, arquitetura de 32 bits e velocidade de clock de até 240 MHz. Essa poderosa unidade de processamento permite executar várias tarefas simultaneamente e processar dados de forma eficiente.

- Conectividade integrada para redes Wi-Fi (802.11 b/g/n) e Bluetooth (v4.2 BR/EDR e BLE). Essa capacidade de comunicação sem fio permite que o dispositivo conecte a redes locais, como a Internet, e comunique com outros dispositivos habilitados para Bluetooth.

- Interface USB que permite a conexão com um computador para programação. Isso facilita o processo de desenvolvimento, carregamento de código e monitoramento de saída de dados.

- Oferece uma ampla variedade de pinos de E/S digitais e analógicos que podem ser configurados como entradas ou saídas, permitindo a conexão de sensores, atuadores e outros dispositivos eletrónicos. Esses pinos são versáteis e podem ser programados para realizar diversas funções.

- O NodeMCU ESP32S é compatível com a plataforma de desenvolvimento Arduíno, o que facilita a programação e o compartilhamento de código. Isso permite que os projetistas aproveitem a vasta biblioteca de funções e recursos disponíveis.

- Possui recursos avançados de gestão de energia, permitindo uma utilização eficiente de energia. Isso é particularmente importante para dispositivos alimentados por bateria, onde a vida útil da bateria é um aspeto crucial.

3.4 Razão da escolha do NodeMCU ESP32S

Escolhemos este microcontrolador devido às características anteriormente referidas sobretudo devido à sua memoria RAM pois o NodeMCU possui recursos de conectividade sem fio, e é necessário armazenar o código do programa, variáveis, chamada de funções, comunicação e outros dados em tempo de execução. Quanto mais RAM estiver disponível, mais espaço haverá para armazenar e manipular dados durante a execução do programa.

A quantidade de RAM disponível no ESP32 pode afetar o desempenho e a capacidade do dispositivo de lidar com tarefas mais complexas. Se o programa requer muita memória para executar as funções, pode haver limitações na quantidade de dados que podem ser processados ou na complexidade das operações que o ESP32 pode realizar.

Outra principal característica foi o Clock de 240MHz pois é uma medida da velocidade na qual o processador realiza as suas operações. Ele determina a taxa na qual as instruções são executadas e os dados são processados pelo microcontrolador, o Clock afeta diretamente o desempenho do processador e, consequentemente, a velocidade de execução do programa. Quanto maior a frequência, mais rápido o ESP32 será capaz de executar as instruções e realizar operações. Isso pode ser importante em situações em que o tempo de resposta é crítico.

Outra das razões que nos levaram a escolher este microcontrolador foram as entradas ADC`s que tem como finalidade converter sinais analógicos em sinais digitais, possibilitando a leitura de valores de grandezas analógicas. A resolução dos ADCs é de 12 bits, significa que o ADC é capaz de converter a tensão de entrada em até 4096 valores digitais diferentes. Esses valores são distribuídos uniformemente dentro da faixa de tensão que o ADC é capaz de ler.

	ESP32S	**ESP8266**	**ARDUINO UNO**
Cores	2	1	1
Arquitetura	32 bits	32 bits	8 bits
Clock	240MHz	80Mhz	16Mhz
Wi-Fi	Sim	Sim	Não
Bluetooth	Sim	Não	Não
RAM	512KB	160KB	2KB
Flash	16MB	16MB	32KB
GPIO	36	17	14
Interfaces	SPI/I2C/UART/I2S/CAN	SPI/I2C/UART/I2S	SP1/I2C/UART
ADC	16	1	6
DAC	2	0	0

Tabela 1 – Diferenças entre microcontroladores

4. Sequenciador de Fases

4.1 Introdução

Uma parte deste Projeto foi realizado numa placa de circuito impresso, placa essa onde foram soldados os devidos componentes como os Mosfets, Resistências/Potenciómetro e Díodos.

Numa primeira fase foi implementado na saída 3 Leds e mais tarde os 3 enrolamentos do Motor. Através de testes práticos com os Leds, conseguimos ver a sequência que é feita através dos Mosfets aumentando ou diminuindo o tempo através do potenciómetro, que tem como objetivo aumentar ou diminuir a velocidade do motor.

4.2 Explicação do funcionamento da placa

Este circuito impresso tem como objetivo o aumento e diminuição de velocidade do motor, através de um potenciómetro. É constituído por 6 díodos IN4007, 3 Mosfets Irf3205, 1 resistência de 4.7k ohm, 1 potenciómetro de 100k ohm, e 3 resistências de 10k ohm.

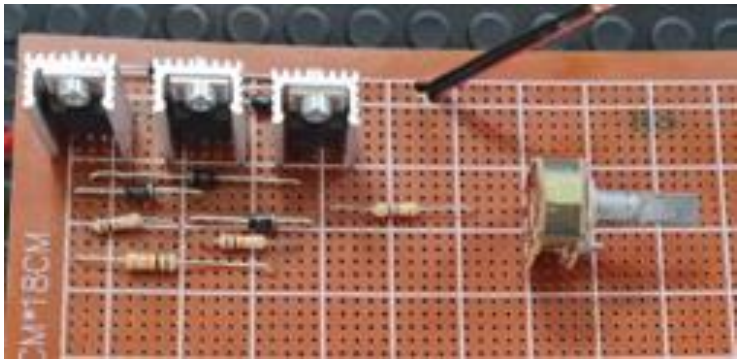


Figura 5 – Placa de Circuito Impresso

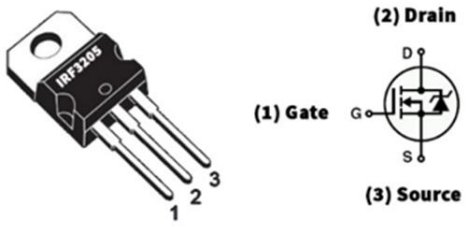


Figura 6 – Polarização do Mosfet

A Source (pino 3) de cada Mosfet foi ligada à massa que por sua vez também foi conectada a um potenciómetro.

Entre a Gate (pino 1) e o Dreno (pino 2) de cada Mosfet estão ligados os 3 díodos.

As 3 resistências de 10k têm como função limitar a corrente que influencia o tempo de subida (passagem a "ON" do Mosfet), ajuda também na limitação da oscilação.

As Gates dos 3 Mosfets estão ligadas a 3 díodos que por sua vez vão ligar no potenciómetro.
A alimentação do circuito é realizada através dos enrolamentos do motor que vai colocar um dos mosfets ativo, onde sequencialmente vai desligar os outros fazendo assim um Loop.

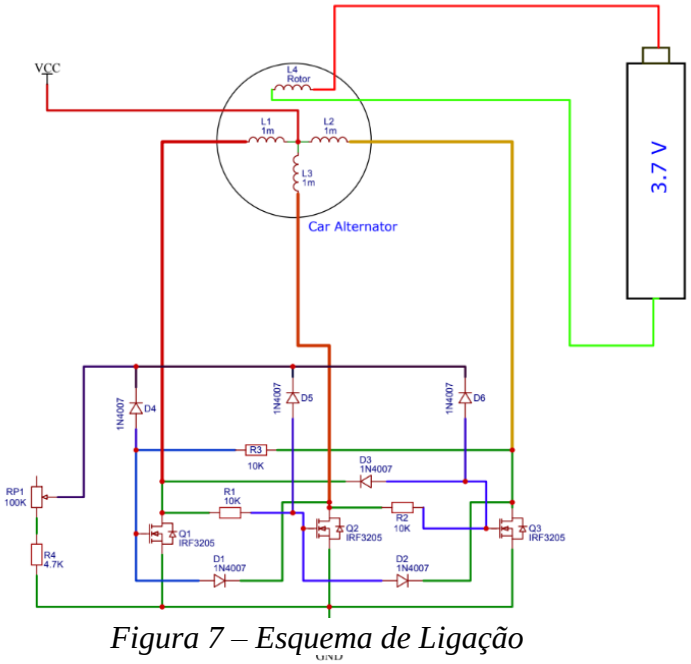


Figura 7 – Esquema de Ligação

4.3 Simulação no Multisim

A simulação foi executada no Multisim que é um software de simulação de esquemas eletrônicos analógicos, digitais e de potencia que funciona com base no SPICE.

Através desta simulação, que mais tarde foi implementada na prática, foi possível visualizar a respetiva sequência que os Mosfets realizavam, ligando os leds sequencialmente.

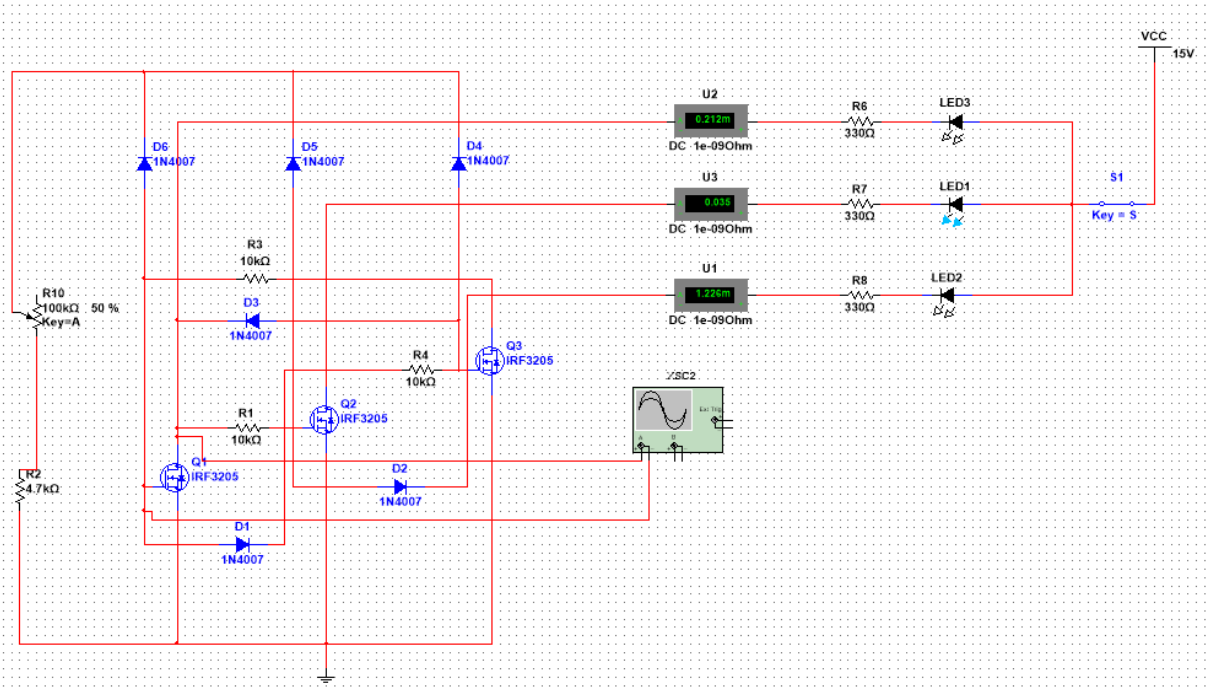


Figura 8 – Implementação do circuito no Multisim

4.4 Dificuldades durante a implementação da placa de circuito impresso

A principal adversidade na placa de circuito impresso foi o sobre aquecimento dos Mosfets que fez com que se queimassem, mosfets estes que aguentam 55V e 110A.

Numa primeira fase foi feita uma análise no circuito onde concluímos que todos os componentes e respetivas ligações estavam bem realizadas.

Numa segunda fase foram realizados testes com o osciloscópio onde tentámos reduzir o tempo que o Mosfet demorava a passar do estado 0 para 1. Observámos que esse tempo era de 4 microssegundos. Voltamos a dimensionar novas resistências e conseguimos reduzir esse tempo para 3.6 microssegundos, ou seja, praticamente igual.

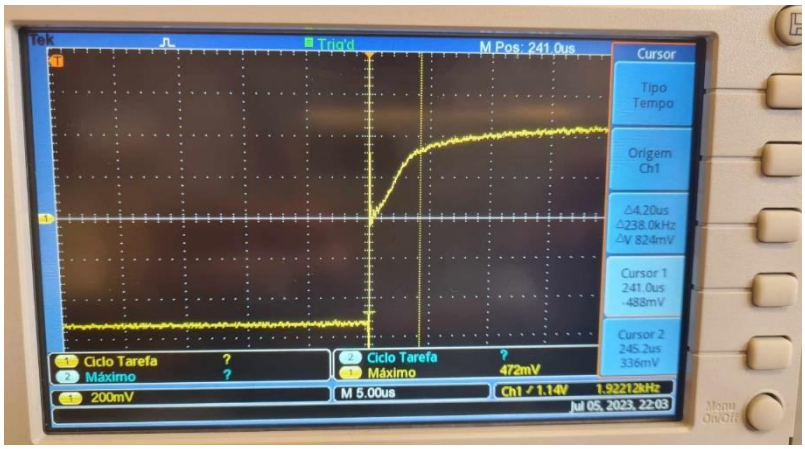


Figura 9 – Análise no Osciloscópio

Numa terceira fase procedemos á medição de resistência de cada enrolamento e concluímos que era de 0.1 ohm o que era um valor muito reduzido.

$$\mathrm{I} = \frac{U}{R} = \frac{10}{0.1} = 100\mathrm{A}$$

É de salientar que como a resistência é muito reduzida qualquer tensão que seja aplicada faz com haja uma corrente muito elevada e perante este problema a solução mais indicada foi a ligação de 3 reóstatos em série, um em cada enrolamento reduzindo assim a corrente.

$$\mathrm{I} = \frac{U}{R} = \frac{10}{1} = 10A$$

Com esta diminuição de corrente em 10 vezes já permitiu uma corrente menor na placa de circuito impresso, podendo assim reduzir o problema de sobreaquecimento.

Nos testes efetuados com o motor, a utilização de uma pilha de 3,7V, traduziu-se na produção de um campo magnético rotórico muito forte, que prendia o rotor de sobremaneira. Foi então necessário utilizar uma fonte de alimentação eletrónica, mas não se mostrou adequada e havia o risco de danificar a fonte. Em substituição foi utilizada a saída retificadora de um autotransformador de bancada (*Figura 10*), disponível no laboratório de instalações elétricas e própria para enrolamentos de excitação magnética de máquinas elétricas.

Figura 10 - Autotransformador

Para a alimentação dos 3 enrolamentos do motor foi necessário usar uma fonte DC (Figura 11), capaz de fornecer 0-15V com uma corrente máxima de 30A, esta fonte possui a capacidade de limitar a tensão e corrente através de dois potenciómetros.

Figura 11 – Fonte de Alimentação DC

O fato de precisarmos recorrer a essa fonte DC e ao autotransformador de bancada acabou por limitar a implementação tornando-a mais demorada, uma vez que, em casa, a placa de circuito era testada apenas com LEDs, o que não resultava no aquecimento excessivo que enfrentamos no laboratório.

A encomenda dos materiais a fornecedores da China tornou-se demorada, mas esta solução apresentava uma relação custo/benefício favorável em relação a outras lojas. Esse material incluía Mosfets, sensores, placas de circuito impresso e um potenciómetro de 6 pinos.

Após todas estas dificuldades, prosseguimos com os testes do motor no laboratório, utilizando os reóstatos regulados a 1ohm, porém o problema de sobreaquecimento persistiu.

Perante este problema de temperatura dos Mosfets a solução encontrada foi aumentar os dissipadores, com o objetivo de reduzir a temperatura dos componentes. Para o dimensionamento utilizamos o datasheet do Mosfet, onde pudemos observar que possui uma potencia dissipada de 50W.

A fórmula geral para calcular a resistência térmica é:

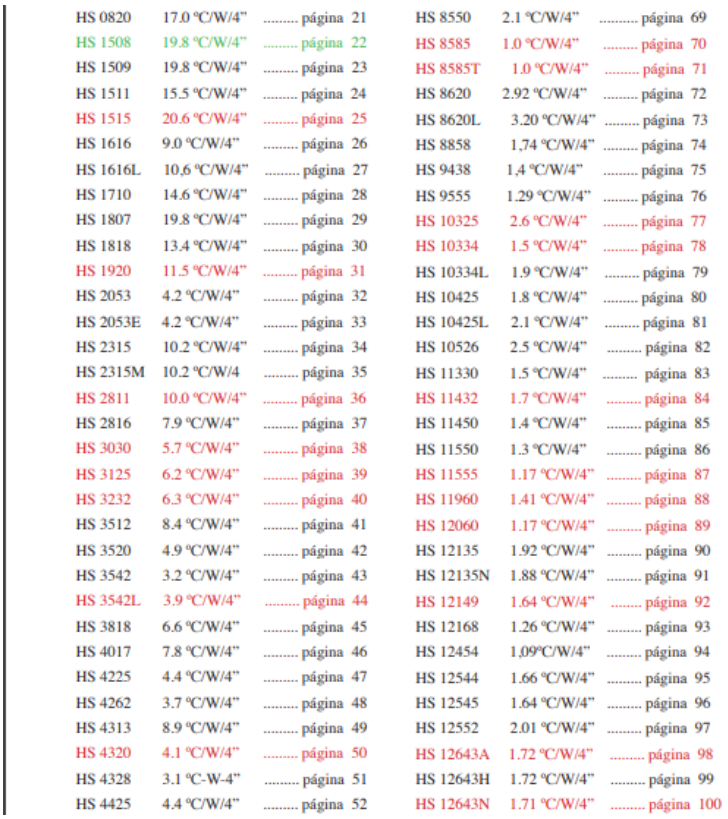
Resistência Térmica (θ) = (Temperatura do Componente - Temperatura do Ambiente) / Potência Dissipada

Supondo que a temperatura do componente seja de 175°C (valor máximo), a temperatura ambiente seja de 25°C e a potência dissipada seja de 50W, podemos usar a fórmula para calcular a resistência térmica:

θ = (175°C - 25°C) / 50W
θ = 150°C / 50W
θ = 3°C/W

Concluímos que a resistência térmica é de 3ºC/W, ou seja, por cada Watt dissipado haverá um aumento de 3º C de temperatura.

Na seguinte figura encontramos uma tabela de um fornecedor de dissipadores onde conseguimos ver os inúmeros dissipadores com resistência térmica diferentes.



HS 0820	17.0 °C/W/4"	página 21	HS 8550	2.1 °C/W/4"	página 69
HS 1508	19.8 °C/W/4"	página 22	HS 8585	1.0 °C/W/4"	página 70
HS 1509	19.8 °C/W/4"	página 23	HS 8585T	1.0 °C/W/4"	página 71
HS 1511	15.5 °C/W/4"	página 24	HS 8620	2.92 °C/W/4"	página 72
HS 1515	20.6 °C/W/4"	página 25	HS 8620L	3.20 °C/W/4"	página 73
HS 1616	9.0 °C/W/4"	página 26	HS 8858	1,74 °C/W/4"	página 74
HS 1616L	10,6 °C/W/4"	página 27	HS 9438	1,4 °C/W/4"	página 75
HS 1710	14.6 °C/W/4"	página 28	HS 9555	1.29 °C/W/4"	página 76
HS 1807	19.8 °C/W/4"	página 29	HS 10325	2.6 °C/W/4"	página 77
HS 1818	13.4 °C/W/4"	página 30	HS 10334	1.5 °C/W/4"	página 78
HS 1920	11.5 °C/W/4"	página 31	HS 10334L	1.9 °C/W/4"	página 79
HS 2053	4.2 °C/W/4"	página 32	HS 10425	1.8 °C/W/4"	página 80
HS 2053E	4.2 °C/W/4"	página 33	HS 10425L	2.1 °C/W/4"	página 81
HS 2315	10.2 °C/W/4"	página 34	HS 10526	2.5 °C/W/4"	página 82
HS 2315M	10.2 °C/W/4	página 35	HS 11330	1.5 °C/W/4"	página 83
HS 2811	10.0 °C/W/4"	página 36	HS 11432	1.7 °C/W/4"	página 84
HS 2816	7.9 °C/W/4"	página 37	HS 11450	1.4 °C/W/4"	página 85
HS 3030	5.7 °C/W/4"	página 38	HS 11550	1.3 °C/W/4"	página 86
HS 3125	6.2 °C/W/4"	página 39	HS 11555	1.17 °C/W/4"	página 87
HS 3232	6.3 °C/W/4"	página 40	HS 11960	1.41 °C/W/4"	página 88
HS 3512	8.4 °C/W/4"	página 41	HS 12060	1.17 °C/W/4"	página 89
HS 3520	4.9 °C/W/4"	página 42	HS 12135	1.92 °C/W/4"	página 90
HS 3542	3.2 °C/W/4"	página 43	HS 12135N	1.88 °C/W/4"	página 91
HS 3542L	3.9 °C/W/4"	página 44	HS 12149	1.64 °C/W/4"	página 92
HS 3818	6.6 °C/W/4"	página 45	HS 12168	1.26 °C/W/4"	página 93
HS 4017	7.8 °C/W/4"	página 46	HS 12454	1,09°C/W/4"	página 94
HS 4225	4.4 °C/W/4"	página 47	HS 12544	1.66 °C/W/4"	página 95
HS 4262	3.7 °C/W/4"	página 48	HS 12545	1.64 °C/W/4"	página 96
HS 4313	8.9 °C/W/4"	página 49	HS 12552	2.01 °C/W/4"	página 97
HS 4320	4.1 °C/W/4"	página 50	HS 12643A	1.72 °C/W/4"	página 98
HS 4328	3.1 °C-W-4"	página 51	HS 12643H	1.72 °C/W/4"	página 99
HS 4425	4.4 °C/W/4"	página 52	HS 12643N	1.71 °C/W/4"	página 100

Figura 12 – Catálogo HS Dissipadores 2023

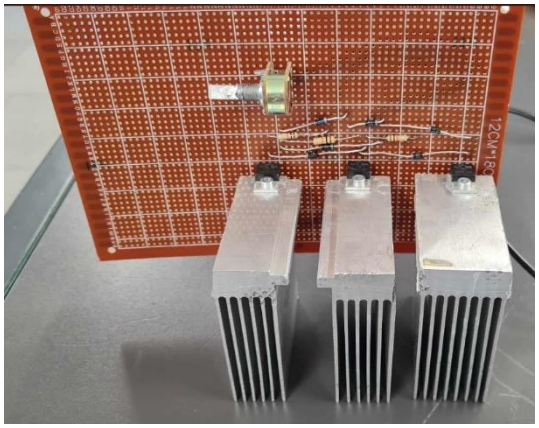


Figura 13 – Nova placa de Circuito Impresso

Devido ao preço elevado e difícil acesso destes dissipadores de calor, reutilizamos alguns que eram usados em processadores, dando assim uma segunda vida a estes dissipadores.

O problema de sobreaquecimento persistiu e como solução utilizamos um ccpm servo consistency master que é um controlador que varia a velocidade do motor através do potenciómetro colocando assim em funcionamento toda a programação anteriormente realizada.

Figura 14 – Testador/Controlador

O funcionamento de um ESC brushless é complexo. Tudo é controlado por um microcontrolador (*Figura 14*) que possui uma sequência binária pré-definida no seu firmware para o acionamento do circuito de MOSFET`s que irão energizar as bobines na devida sequência. O controle da velocidade é feito através de um sinal PWM (do testador) recebido e interpretado pelo microcontrolador, que converte em uma frequência diferente de chaveamento, ou seja, quanto maior o valor recebido, mais rapidamente os MOSFETs serão acionados, aumentando a velocidade de rotação do motor.

5. Programação

5.1 Motor de Passo

5.1.1 Introdução

Para controlar remotamente a velocidade do motor BLDC, através do potenciómetro (implementado na placa de circuito impresso) é necessário regular o seu valor à distância, para isso vamos usar um motor de passo, pois este é capaz de controlar o ângulo de forma precisa.

O motor de passo é um motor elétrico de corrente continua que converte pulsos elétricos em movimento mecânico angular preciso. Ao contrário dos motores de corrente contínua convencionais, que giram continuamente quando alimentados com corrente, um motor de passo gira em etapas discretas, avançando de um passo para o próximo/outro em resposta a pulsos elétricos.

Composto por um rotor (parte móvel) e um estator (parte fixa). O rotor é composto por um conjunto de ímãs permanentes, enquanto o estator é composto por bobines de fio enroladas em torno de polos magnéticos. Essas bobines são energizadas em uma sequência específica para criar campos magnéticos que atraem o rotor, movendo-o de um passo para o próximo/outro.

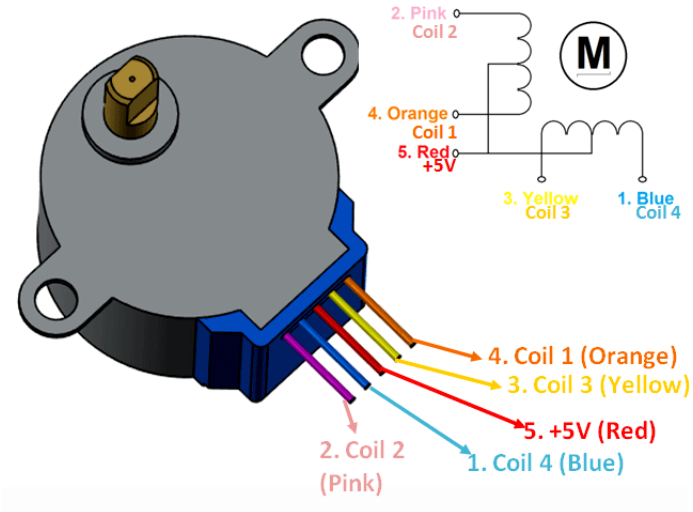


Figura 15 - Motor de passo

A maneira como se energiza as bobines do motor de passo determina como o motor opera:

- A sequência de pulsos determina a direção de rotação do motor. Ao inverter a sequência dos pulsos elétricos, é possível alterar a direção de rotação do motor, permitindo movimento no sentido horário ou anti-horário.
- A frequência dos pulsos determina a velocidade do motor. Quanto maior a frequência dos pulsos, mais rápido o motor girará. Ao controlar a taxa de pulsos elétricos enviados para as bobines, é possível ajustar a velocidade do motor de passo conforme necessário.
- O número de pulsos determina até onde o motor girará. Ao enviar um número específico de pulsos elétricos para as bobines, é possível controlar a quantidade de passos que o motor dará e, consequentemente, o ângulo de rotação. Cada passo corresponde a um determinado ângulo, permitindo um controle preciso do posicionamento do motor.

Caraterísticas

- Tensão de operação: 12V CC
- Número de fases: 4
- Relação da redução de engrenagens: 1/64 (mecanismo de redução)
- Angulo do passo: 5,625 graus => 64 passos/volta (360/64=5,625)
- Frequência: 100 Hz
- Torque de tração: 34,3 m N/m

5.1.2 Relação da redução de engrenagens

A relação de redução de engrenagens é uma medida que indica a diferença de velocidade entre a rotação do eixo de entrada (eixo do rotor) e o eixo de saída (eixo do motor exterior). Ela é expressa como uma proporção ou fração, representando quantas vezes o eixo de entrada precisa girar para que o eixo de saída complete uma rotação completa.

No caso deste motor de passo, a relação de redução de engrenagens é de 1/64. Isso significa que o eixo de entrada do motor precisa girar 64 vezes para que o eixo de saída complete uma rotação completa de 360 graus.

A relação da redução de engrenagens é calculada da seguinte maneira:

$$\frac{engrenagem\ acionada}{engrenagem\ de\ acionamento} = \frac{32}{9} * \frac{22}{11} * \frac{26}{9} * \frac{31}{10} = 63{,}68395$$

O valor obtido é arredondado para 64 voltas.

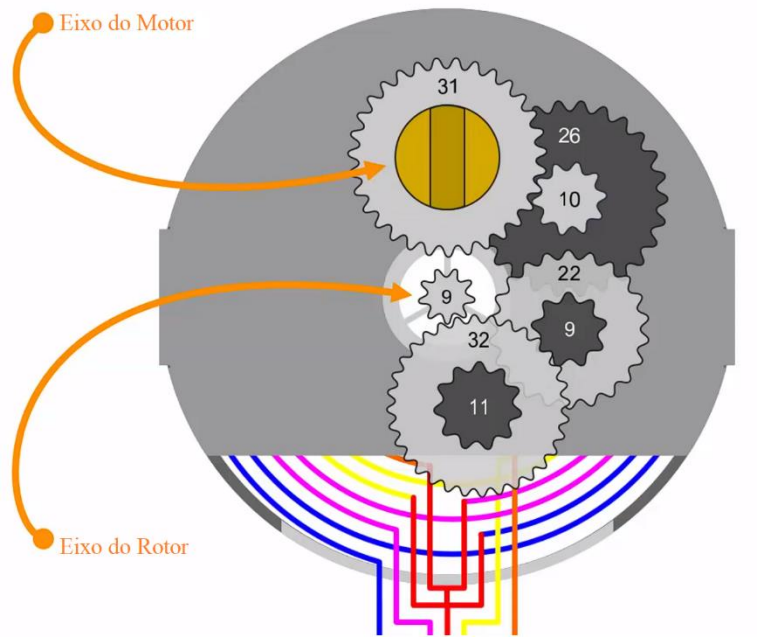


Figura 16 - Engrenagens do Motor de passo

5.1.3 Modo de acionamento do Motor

Existem diferentes modos de acionamento para motores de passo, como o Wave Driving (Acionamento em Onda), Full Stepping (Passo Completo) e o Half Stepping (Meio Passo). Cada um desses modos oferece características distintas em termos de torque, resolução e suavidade de movimento. Neste texto, vamos explorar cada um desses modos e entender as suas diferenças, vantagens e desvantagens.

Wave Driving (Acionamento em Onda):

Neste modo de acionamento, apenas uma bobine é ativada de cada vez. Isso resulta num movimento mais "áspero" e menos suave do motor de passo. O torque reduzido e o ângulo de passo maior limitam as aplicações deste motor de passo. O acionamento em onda é geralmente usado quando a precisão de posicionamento não é crítica e o motor é usado principalmente para aplicações de movimento simples.

Step	Azul	Rosa	Laranja	Amarelo
1	1			
2		1		
3			1	
4				1

Tabela 2 – Acionamento em onda

Full Stepping (Passo Completo):

No modo de acionamento de passo completo, cada passo do motor é ativado em sua sequência completa. Isso significa que duas bobines do motor são acionadas simultaneamente, proporcionando um torque máximo. Como resultado, o motor tem um movimento mais suave e é capaz de produzir mais força. No entanto, o angulo de passo é maior, isso faz resultar em uma resolução mais baixa e menor precisão de posicionamento.

Step	Azul	Rosa	Laranja	Amarelo
1	1			1
2	1	1		
3		1	1	
4			1	1

Tabela 3 – Passo completo

Half Stepping (Meio Passo):

O modo de acionamento de meio passo é a junção dos dois modos anteriores em um só. A sequência de acionamento varia entre alimentar uma bobine e depois duas bobines e assim sequencialmente. O que faz reduzir o ângulo de passo, Isso permite que o motor de passo alcance uma resolução e precisão de posicionamento mais alta. Consequentemente o torque do motor é reduzido em comparação com o modo de passo completo.

Step	Azul	Rosa	Laranja	Amarelo
1	1			
2	1	1		
3		1		
4		1	1	
5			1	
6			1	1
7				1
8	1			1

Tabela 4 – Meio Passo

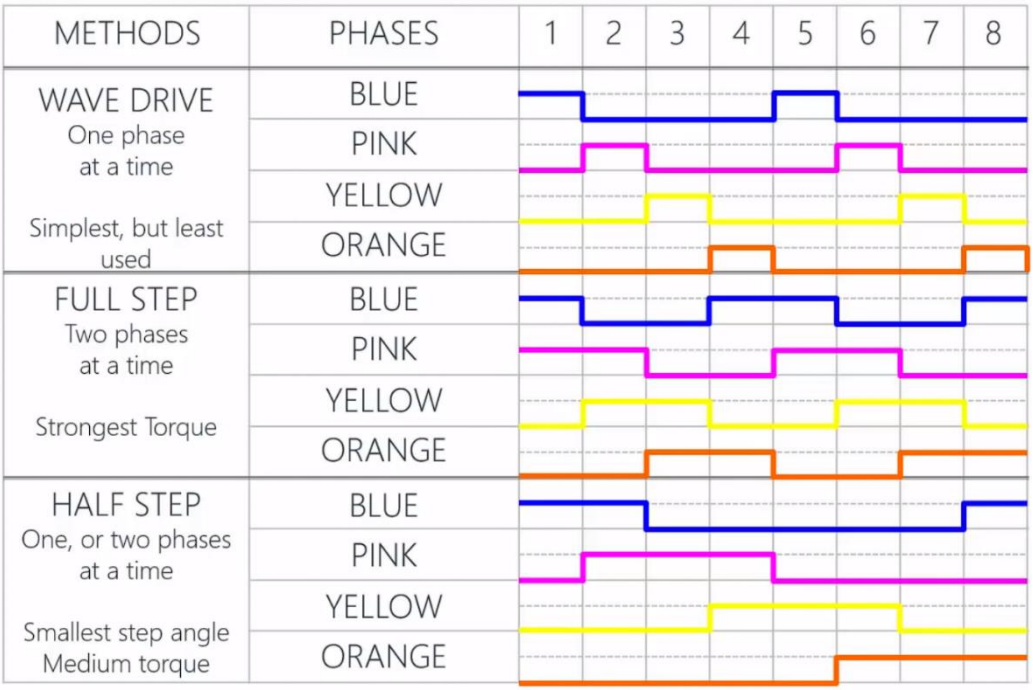


Figura 17 - Diferenças entre os Passos

Ângulo de passo

O ângulo de passo deste motor refere-se ao ângulo que o eixo do rotor se move a cada passo. No caso deste motor, o ângulo de passo é de 5,625 graus no modo de acionamento Half Stepping e de 11,25 graus no modo de acionamento Wave Driving ou Full Stepping.

$$\frac{360}{64} = 5{,}625\ graus \qquad \frac{360}{32} = 11{,}25\ graus$$

Para que seja realizada uma volta completa no motor de passo no modo Wave Driving ou Full Stepping é necessário 2048 passos, porque existe uma relação de relação de engrenagens de 1/64, e o eixo do rotor precisa de 32 passos para dar uma volta completa.

64 * 32 = 2048 passos

Para que seja realizada uma volta completa no motor de passo no modo Half Stepping é necessário 4096 passos, porque existe uma relação de redução de engrenagens de 1/64, e o eixo do rotor precisa de 64 passos, pois este modo tem uma resolução maior o que faz com que necessite do dobro dos passos para dar uma volta completa.

64 * 64 = 4096 passos

5.1.4 Driver ULN 2003

O ULN2003 é um circuito integrado de driver de transístor Darlington, onde contém sete drivers de alta tensão e corrente, projetados para acionar cargas indutivas, como motores de passo, relés, solenoides, entre outros dispositivos.

O transístor Darlington é um dispositivo eletrônico constituído por dois transístores bipolares, tem como função fornecer um ganho de corrente elevado e uma baixa corrente de base, permitindo que seja usado como um amplificador de corrente, é construído em cascata, onde o coletor de um transístor é conectado à base do outro transístor. Isso resulta em um ganho de corrente combinado muito maior do que o de um único transístor e daí ser utilizado onde é necessário amplificar uma corrente pequena para controlar uma corrente maior.

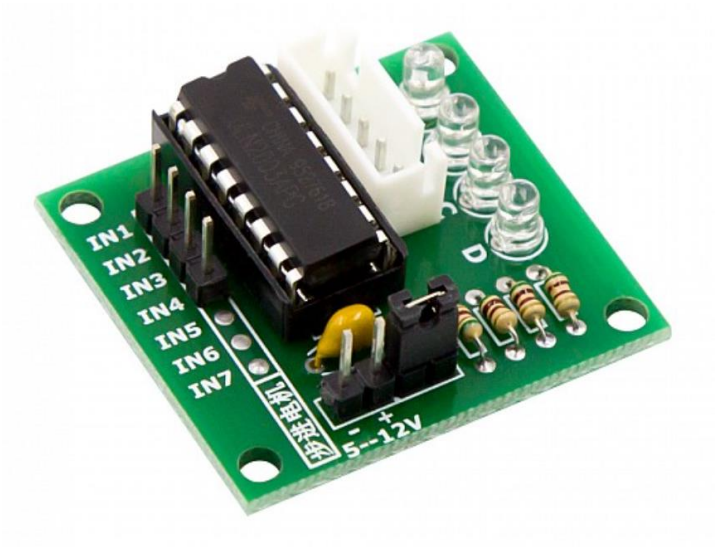


Figura 18 - Driver ULN 2003

Características do ULN2003:

- Configuração de transístor Darlington de sete canais.
- Tensão de alimentação: 5V a 50V.
- Corrente de saída por canal: até 500 mA.
- Proteção contra sobretensão
- Baixa queda de tensão na saída.

O uso do ULN2003 traz várias vantagens:

- Simplifica a interface com dispositivos de alta corrente e alta tensão, protegendo os componentes de controle contra danos.
- Fornece isolamento elétrico entre o circuito de controle e a carga, evitando interferências e danos nos componentes de controle.
- Permite o acionamento de cargas indutivas, que geralmente requerem correntes mais altas do que os pinos de saída dos microcontroladores podem fornecer diretamente.

Sem o ULN2003, seria necessário utilizar circuitos de acionamento mais complexos e potencialmente arriscados, como transístores individuais ou relés mecânicos. Isso exigiria um projeto mais elaborado e poderia resultar em maior custo e complexidade de montagem.

5.1.5 Resultados obtidos

Inicialmente criamos um programa para implementar este motor de passo, que é controlado inserindo o valor do ângulo pretendido no monitor série, caso este valor do ângulo seja superior a zero vai rodar para a direita se for inferior a zero vai rodar para a esquerda.
Foi utilizado o modo de acionamento de meio passo de forma a termos a maior precisão possível.

Na figura que se segue mostra os resultados obtidos no monitor serie:

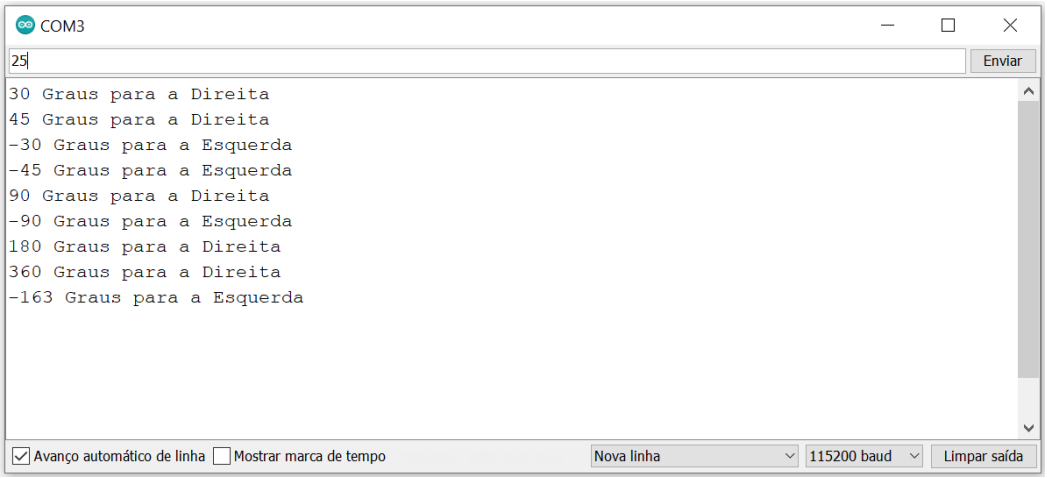


Figura 19 - Monitor série Motor de Passo

Durante a realização deste código a nossa maior dificuldade, foi em controlar o angulo de forma precisa sem utilizar nenhuma biblioteca. Após várias tentativas percebemos que para o motor

de passo dar uma volta completa eram precisos 512 passos e não os 4096 passos do modo de acionamento de Meio Passo.
Já que o motor com 512 passos não apresenta muita resolução, decidimos alimentar duas bobines em simultâneo (Passo completo) com finalidade aumentar o torque o que fez com que também houvesse uma diminuição das linhas de código do programa, que pode ser encontrado no Anexo 4.

5.1.6 Esquema de montagem

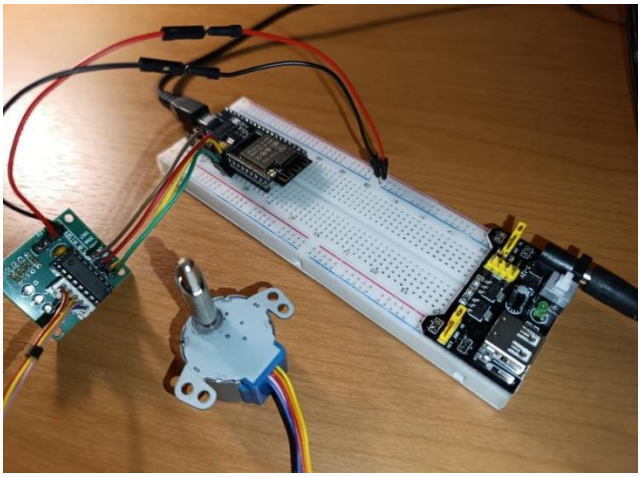


Figura 20 – Motor de passo

5.2 Potenciómetro

5.2.1 Potenciómetro linear rotativo de 6 pinos

Figura 21 - Potenciómetro de 6 pinos

Como o controlo do ângulo do motor de passo não está 100% preciso, houve a necessidade de encontrar uma forma simples de verificar a posição do potenciómetro, de modo que quando o motor de passo esteja conectado a ele, este não ultrapasse os seus limites.

Para isso foi utilizado um potenciómetro linear rotativo de 6 pinos, que nos é bastante útil pois com ele conseguimos controlar a velocidade do motor num potenciómetro e simultaneamente conseguimos ler o valor correspondente no outro potenciómetro (de medida) para sabermos sempre a posição de forma atualizada.

O potenciómetro teve de ser ligado a um ADC do esp32, permitindo que uma entrada analógica seja convertida em um valor digital de 12 bits (0 a 4096), representando a tensão no pino de entrada. Isso significa que o Esp32 pode medir com precisão a tensão do potenciómetro em diferentes posições.

O potenciómetro de medida possui três pinos. Um ligado ao ground do esp32, outro ligado a 3.3v e o pino do meio ligado à ADC do esp mais concretamente no pino 36.

5.2.2 Resultados obtidos

Inicialmente, através de pesquisa de informação e incorporação dessa informação foi criado um programa de leitura de valores do potenciómetro, este programa lê o valor do potenciómetro através da ADC do Esp32 e envia essas leituras para o monitor série.

Foi realizado um estudo sobre este potenciómetro para podermos fazer o controlo das condições de funcionamento do motor através das suas leituras e do valor do ângulo inserido no monitor série.

Após ser analisado foram inseridas linhas de código que estão presentes no Anexo 5, programa esse que mostra o motor de passo a atuar o potenciómetro de controlo de velocidade do motor síncrono, onde os pinos do potenciómetro de medida permitem ler o valor (posição) através da ADC do esp32.

Este programa faz com que o ângulo obtido no monitor serie faça rodar o motor de passo e quando o potenciómetro estiver a chegar ao seu limite (inferior ou superior) impeça o motor de passo de rodar mais, não danificando assim o potenciómetro.

Após declaração e inicialização de variável e leitura do valor inicial foi necessário a implementação destas linhas de programa que tem como finalidade rodar o motor. Caso o ângulo seja >0 executa o movimento para a direita, caso o ângulo seja <0 executa o movimento para a esquerda.

```
      // Verifica se o ângulo é positivo e executa o movimento para a direita
   if (angulo > 0) {
     while (graus > passos && valorPotenciometro >= valorMinimo && valorPotenciometro <
valorMaximo) {

   // Verifica se o ângulo é negativo e executa o movimento para a esquerda
   if (angulo < 0) {
     while (graus < passos && valorPotenciometro >56 valorMinimo && valorPotenciometro
<= valorMaximo) {
```

Sempre que o programa é inicializado ele lê o valor do potenciómetro e roda até obter o valor de 0, esta operação é realizada através do motor de passo que também está no anexo 5 mais concretamente nestas linhas de código seguintes.

```
 // Ajusta o potenciómetro para zero
 while (valorPotenciometro > valorMinimo) {
   Esquerda();
   valorPotenciometro = analogRead(potenciometro);
   delay(t);
 }
```

Enquanto o valor do potenciómetro for maior que o mínimo irá executar o movimento para a esquerda enquanto se verifica a seguinte condição “while (valorPotenciometro > valorMinimo) “.

Na figura que se segue mostra os resultados obtidos no monitor serie:

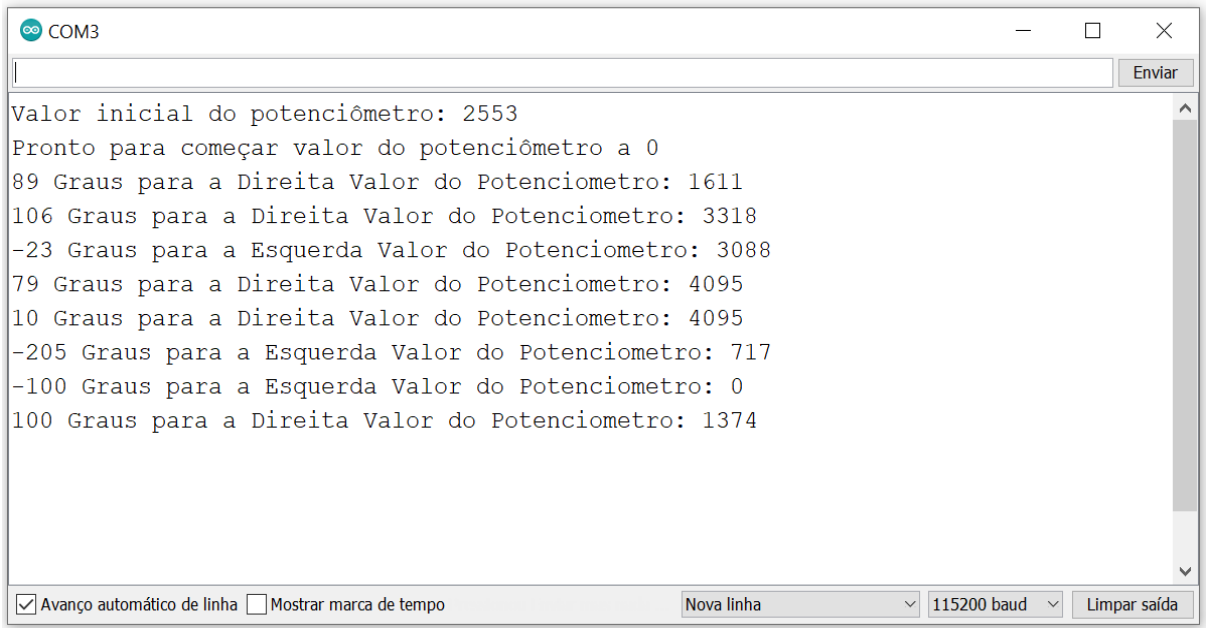


Figura 22 - Monitor série do Programa com potenciómetro

Através do monitor série (*Figura 22*) podemos observar que inicialmente o programa lê o valor do potenciómetro. Caso este valor não seja 0, o motor de passo vai rodar até ficar com o valor de zero.

Com base no valor do ângulo inserido na porta do monitor série é possível ver o motor de passo a rodar para a esquerda e para a direita, além disso conseguimos agora ver o valor lido pela ADC, valor esse que varia entre 0 e 4096.

O potenciómetro de 6 pinos foi uma mais-valia porque com ele conseguimos saber sempre o valor do ângulo do potenciómetro e isso permitiu-nos resolver o pequeno erro de imprecisão que o programa anterior possuía.

5.2.3 Esquema de montagem

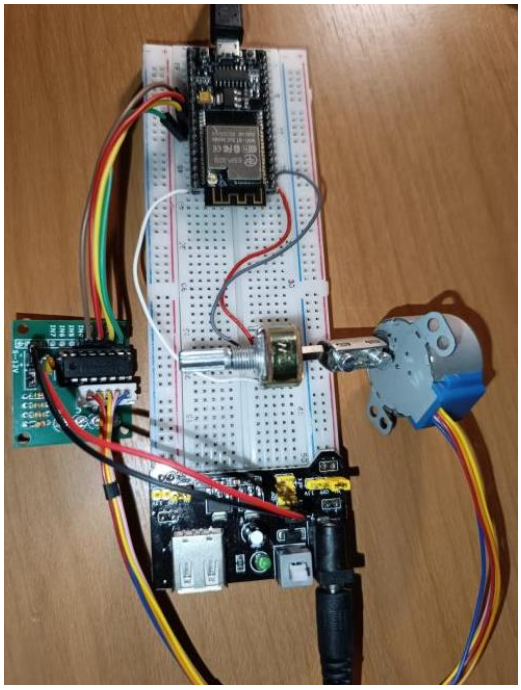


Figura 23 – Motor de Passo com o Potenciómetro

5.3 Web Server

5.3.1 Página web/HTML

Um Web Server é um software responsável por fornecer conteúdo e serviços através da internet, permitindo que os utilizadores acedam e interajam com recursos disponíveis num servidor por meio de um navegador web, para a criação do nosso web server usámos linguagem de código HTML.

O ESP32 cria um web server dentro da rede em que está conectado. Isso significa que o web server pode ser acedido apenas por dispositivos que estejam conectados à mesma rede local.

A programação em html não nos é “familiar”, mas após pesquisa e tentativas de implementação de exemplos de código em html, conseguimos implementar um código que se encontra no Anexo 6 deste relatório.

Usamos o Notepad++ para criar a página web em HTML, esta linguagem de programação é bastante versátil, permitindo enviar imagens, gráficos para a página web.

Inicialmente, através de pesquisa de informação e incorporação dessa informação foi criado um programa que através do web server controla um led.

Fomos inserindo linhas de programação aos poucos de modo que o web server enviasse o valor do angulo para o Esp32, este valor é controlado através de um slider que se encontra na página web (*Figura 24*).

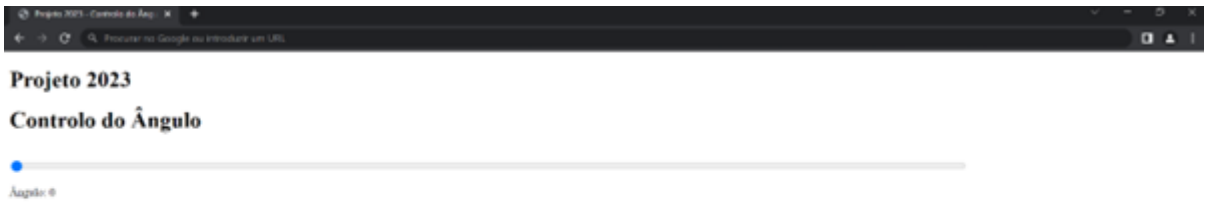


Figura 24 - Web server Controlo do Ângulo

Para a criação deste slider existiram dificuldades, mas após várias tentativas e após procura de informação conseguimos realizar a seguinte linha de código que se encontra no Anexo 6 de modo a criarmos este slider:

<input type='range' min='0' max='360' value='0' class='slider' id='angle-slider'>

Esta linha faz com que o slider trabalhe entre o valor de 0 e 360 enviando assim através do restante código esta informação ao Esp32.

5.3.2 Resultados obtidos

Na figura abaixo podemos ver os resultados obtidos do monitor série:

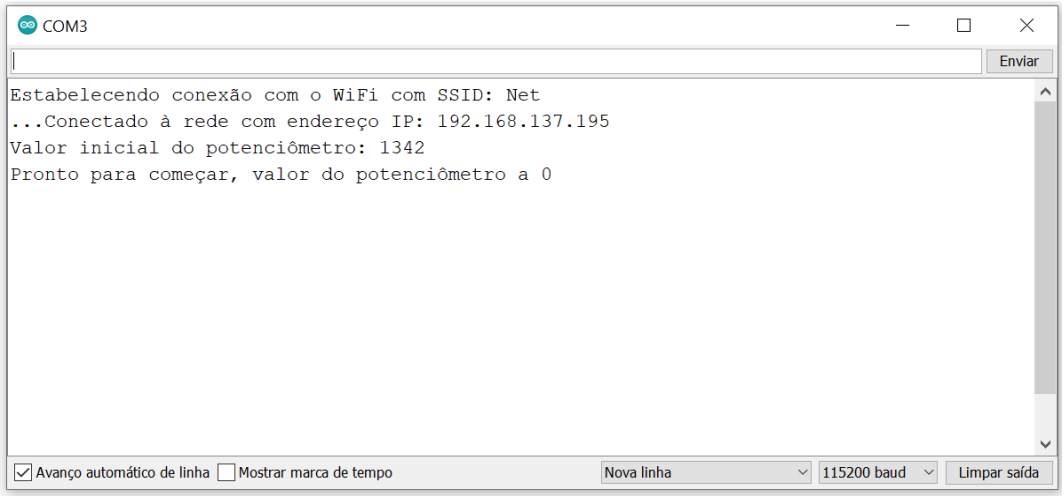


Figura 25 – Monitor série - Conexão Estabelecida

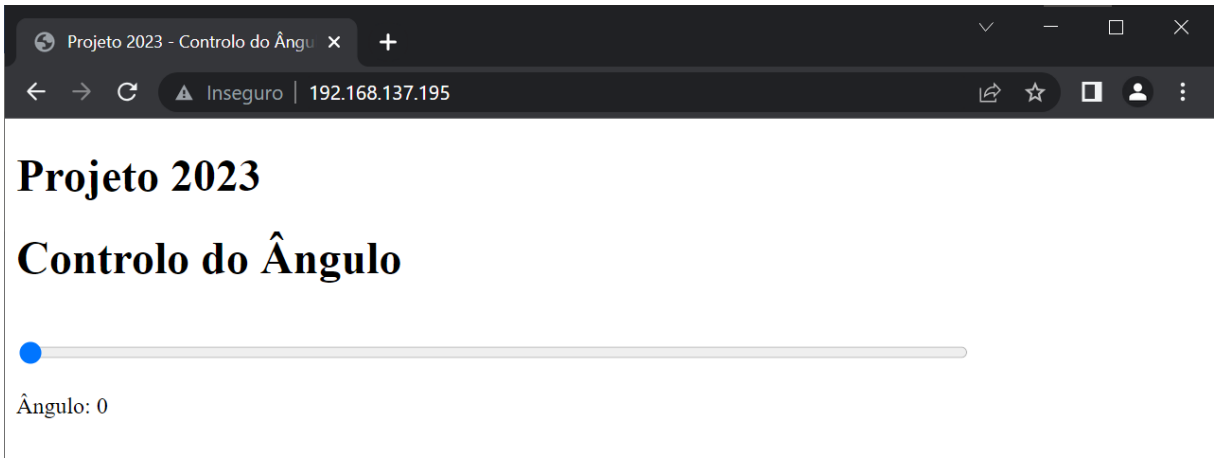


Figura 26 – Conexão da Página Web

Na Figura 25 é possível observar que após ser estabelecida a conexão, é criado o endereço IP. É necessário esse endereço IP para acessar a página web server (alojada no esp32) que permite o controlo do ângulo como podemos ver na Figura 26. É ainda possível observar que o Esp32 vai ler o valor inicial do potenciómetro e vai rodar o motor de passo até este valor ser 0.

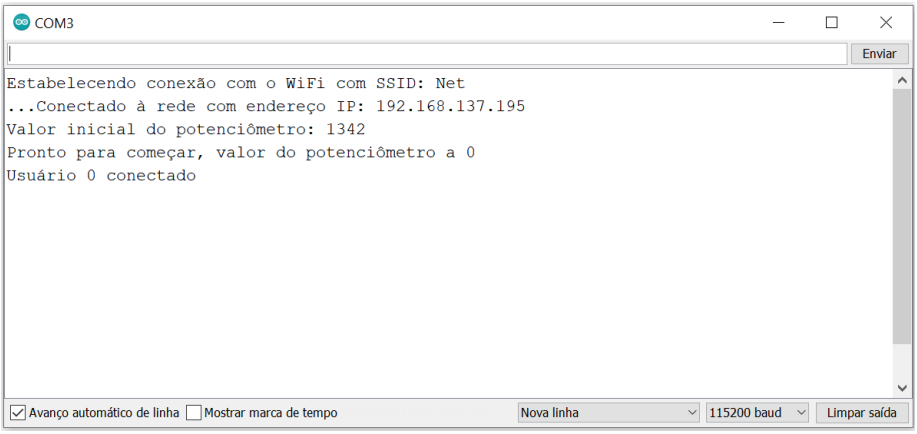


Figura 27 – Conexão do Usuário 0 e envio de dados do web server para o controlo do motor de passo, no Esp32

Após acessar a página web (*Figura 26*) vai ser visualizado no monitor série (*Figura 27*), que o usuário foi conectado ao web server.

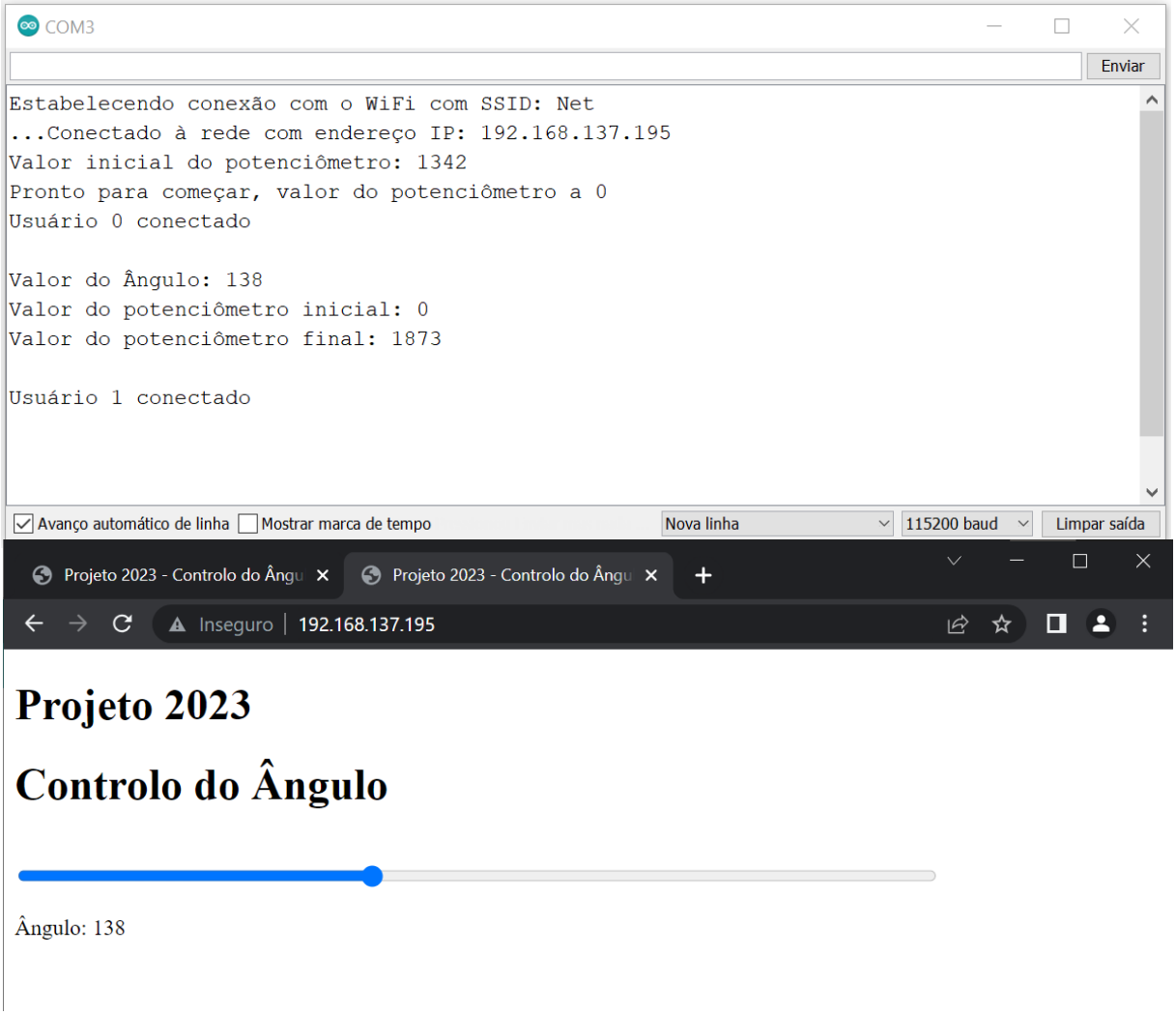


Figura 28 – Conexão do segundo usuário e envio de dados para o controlo do motor de passo, no Esp32

Se o usuário 0 alterar o valor do angulo para 138 vai ser possível visualizar no monitor série o valor do angulo selecionado no web server e o valor inicial do potenciómetro. Após o motor de passo rodar 138 graus aparece no monitor série o valor do potenciómetro correspondente. Caso seja conectado um novo usuário vai aparecer no monitor série que estabeleceu conexão e a página web vai mostrar o valor em que o ângulo se encontra.

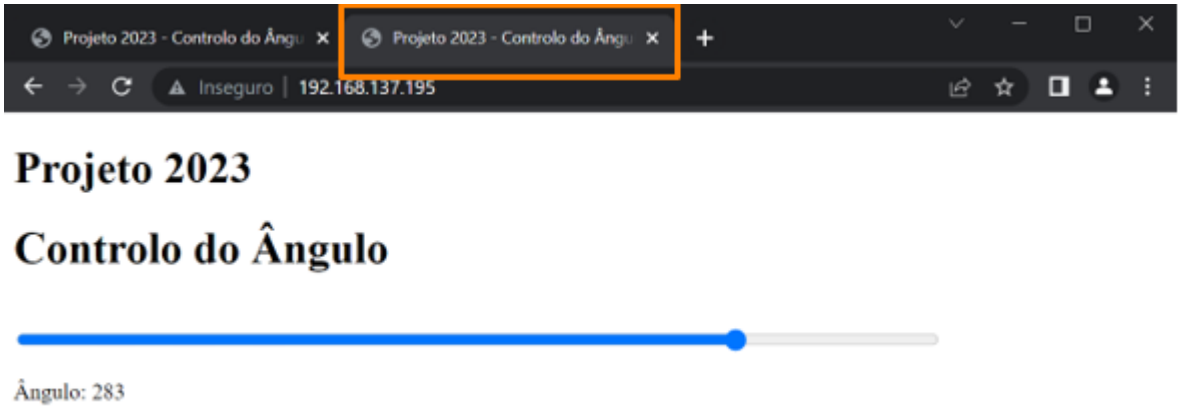


Figura 29 – Usuário1muda o valor do ângulo

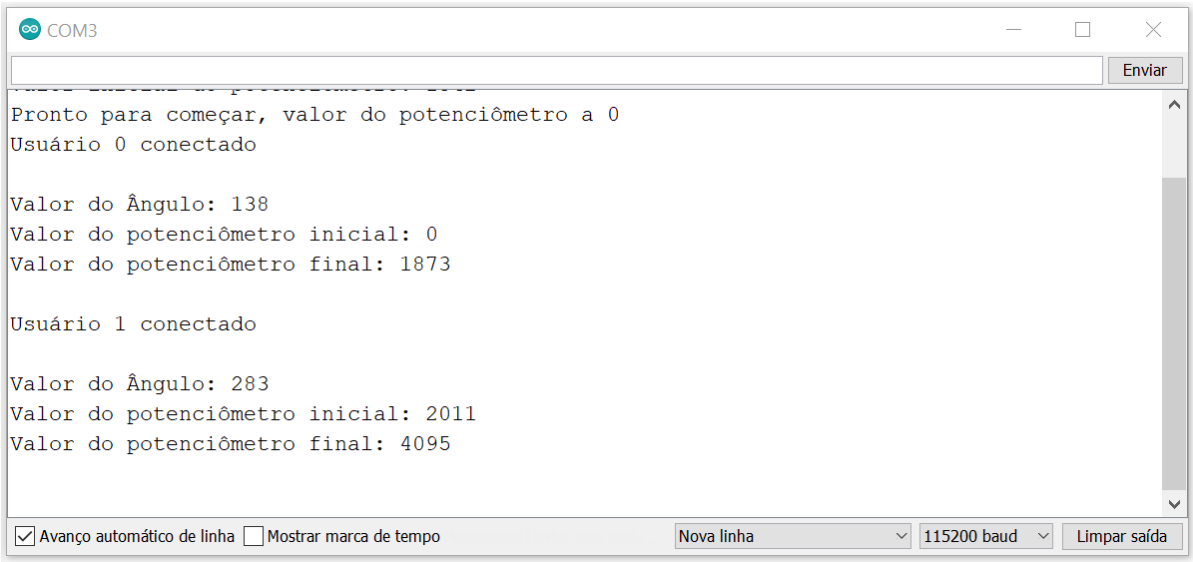


Figura 31 - Monitor série

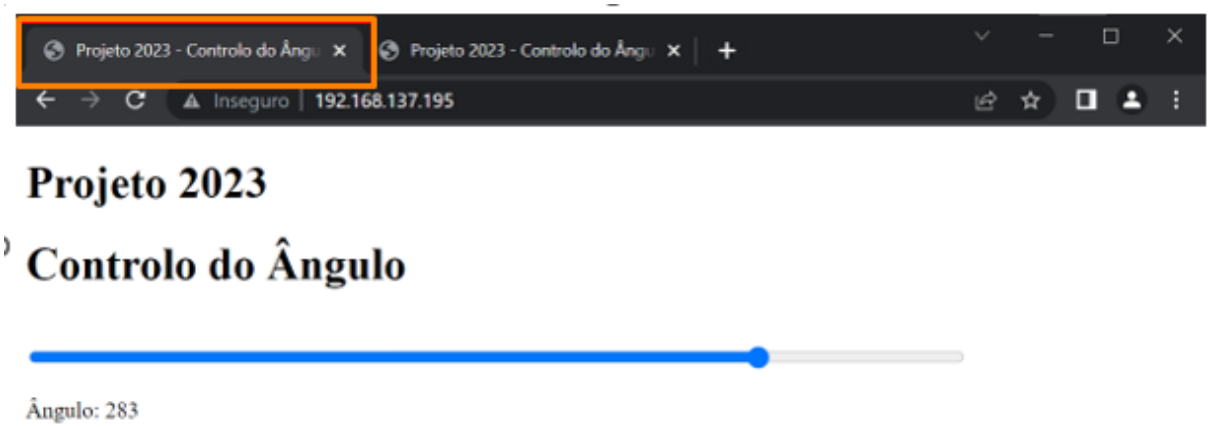


Figura 32 - Página web do usuário 0

Na figura 29 e 32 é possível ver que é aberto um novo separador, dando assim a entender que existe um novo usuário e caso este segundo usuário mude o valor do ângulo de 138 para 283 (*Figura 29*), o motor de passo vai rodar 145, mostrando no monitor série o valor correspondente do potenciómetro antes e depois de o motor de passo rodar os 145 graus. Após o motor de passo rodar o usuário 0 vai visualizar na página web o novo valor do ângulo (*Figura 32*).

Nesta implementação as maiores dificuldades foram o envio do valor do ângulo do web server para o controlo do motor de passo, no Esp32 onde resultou dum acréscimo de linhas de programação que estão presentes em Anexo 6.

Outra dificuldade surgiu após tentarmos conectar mais do que um usuário ao web server, pois o valor do ângulo que era visualizado era 0 mesmo tendo outro valor, mas após pesquisa e interiorização de informação foram inseridas as seguintes linhas de código que também estão no Anexo 6, linhas estas responsáveis por o envio de informação a vários clientes.

```
function processCommand(event) {
    var obj = JSON.parse(event.data);
    var type = obj.type;
  if (type.localeCompare("angulo") == 0) {
    var angulo = obj.value;
      console.log('Valor do Ângulo: ' + angulo);
      slider.value = angulo;
      output.innerHTML = angulo;
  }}

// Envia o valor atual do ângulo para o novo usuário conectado
```

```
sendJson("angulo", String(anguloAtual));
```

Outro imprevisto que tivemos durante a implantação deste programa que pode ser consultado no Anexo 6, foi o facto de haver transmissão de caracteres errados no web server, no qual foram corrigidos através da implementação da seguinte linha de código.

```
// Envio de caracteres corretos para o web server
server.sendHeader("Content-Type", "text/html; charset=UTF-8");
```

5.4 Sensor IR

5.4.1 Introdução

Um sensor infravermelho é um dispositivo eletrônico que é capaz de detetar e medir a radiação infravermelha emitida por objetos ao seu redor. A radiação infravermelha está na faixa do espectro eletromagnético entre a luz visível e as ondas de rádio, possuindo um comprimento de onda maior do que a luz visível.

Os sensores infravermelhos podem ser divididos em duas categorias principais: sensores passivos e sensores ativos.

5.4.2 Sensores Infravermelhos Passivos ou Ativos

- Sensores Infravermelhos Passivos:

Os sensores passivos não emitem radiação infravermelha, mas são projetados para detetar a radiação térmica emitida por objetos ao seu redor. Esses sensores são amplamente usados em sistemas de deteção de movimento, como em alarmes de segurança residenciais, onde são capazes de identificar mudanças no padrão de radiação infravermelha causadas pelo movimento de pessoas ou objetos.

- Sensores Infravermelhos Ativos:

Os sensores ativos, por outro lado, emitem luz infravermelha e também são capazes de detetar a luz refletida por objetos. Eles são comumente usados em aplicações como controle remoto, comunicação por infravermelho (por exemplo, entre dispositivos eletrónicos), sensoriamento de proximidade e em sistemas de seguidores de linha, como mencionado anteriormente.

Os sensores infravermelhos ativos geralmente consistem num emissor de infravermelho (que emite a luz infravermelha) e um recetor de infravermelho (que deteta a luz refletida). A quantidade de luz refletida captada pelo recetor é utilizada para determinar a presença, distância ou características do objeto em questão.

O Sensor IR Infravermelho que foi utilizado é composto por um emissor e um recetor IR, mais o CI comparador LM393, que facilita sua conexão com o microcontrolador, visto que sua tensão é de 3,3-5V.

O seu funcionamento é simples, pois quando algum obstáculo refletor (cor branca) é colocado à frente do Sensor IR Infravermelho, o sinal infravermelho é refletido para o recetor. Quando isso acontece, o pino de saída OUT é colocado em nível baixo (0), e o led verde do módulo é aceso, indicando que algum obstáculo foi detetado.

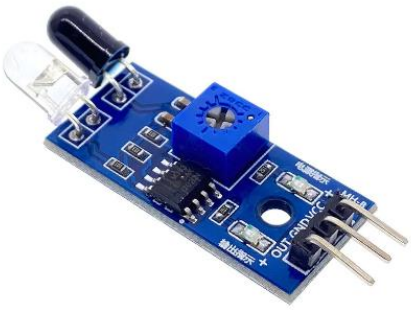


Figura 33 - Sensor IR Infravermelho

5.4.3 Especificações

- Sensor de obstáculo IR;
- Tensão de operação: 3.3 à 5V DC;
- Emissor e recetor IR;
- Distância de deteção: 2 a 80 cm;
- Potenciómetro para ajuste da distância;

5.4.4 Programa

Por último este programa foi um pouco mais complexo devido ao facto de que a informação encontrada estava dependente de uma biblioteca, biblioteca essa que não podia ser utilizada no Esp32, foi criado um programa de raiz de modo a não ser necessário a utilização de bibliotecas.

Na figura abaixo é possível observar o valor que é recebido no web server.

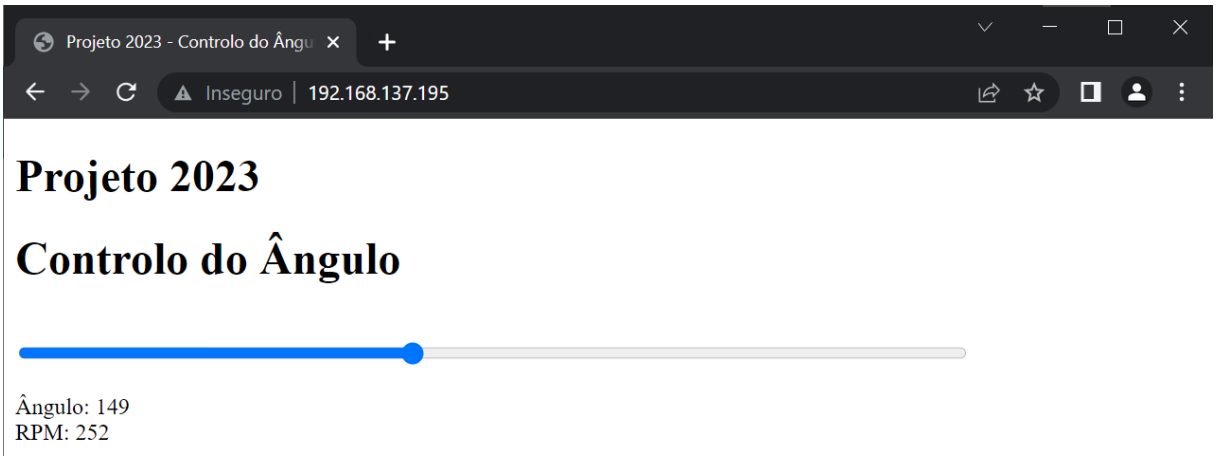


Figura 34 - Página web com controlo de ângulo e RPM

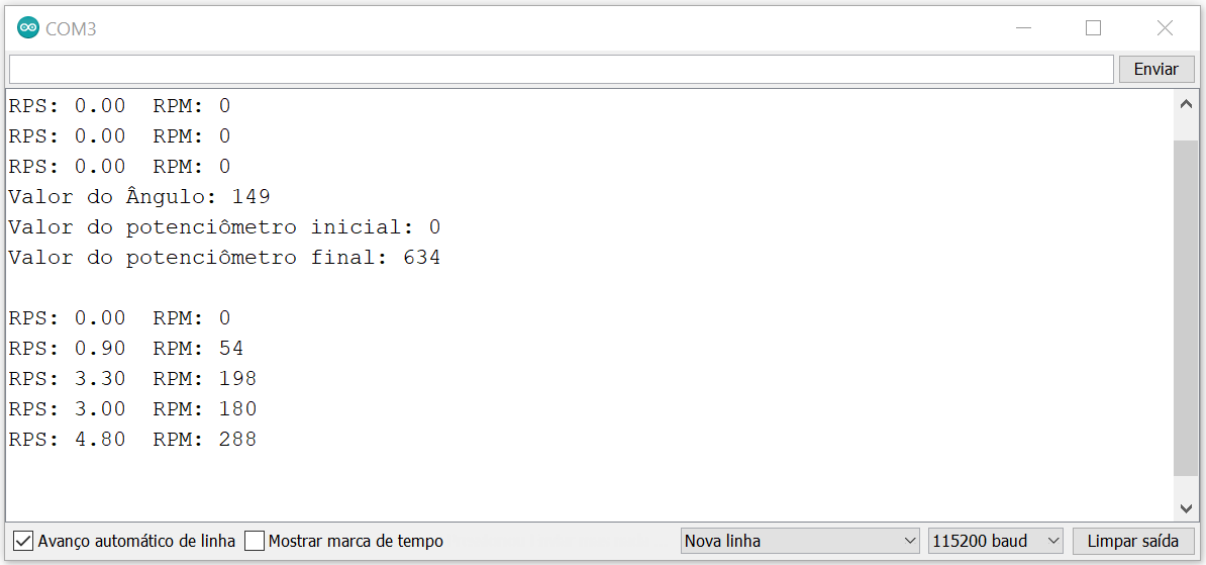


Figura 35 - Monitor série RPM

As funcionalidades do programa anterior mantiveram-se, fazendo apenas um acréscimo deste sensor, que é possível ver no Anexo 7.

É possível visualizar no web server (*Figura 34*), o valor de RPM em que o motor se encontra, este valor está disponível em todos os usuários que estabelecem conexão com o web server.

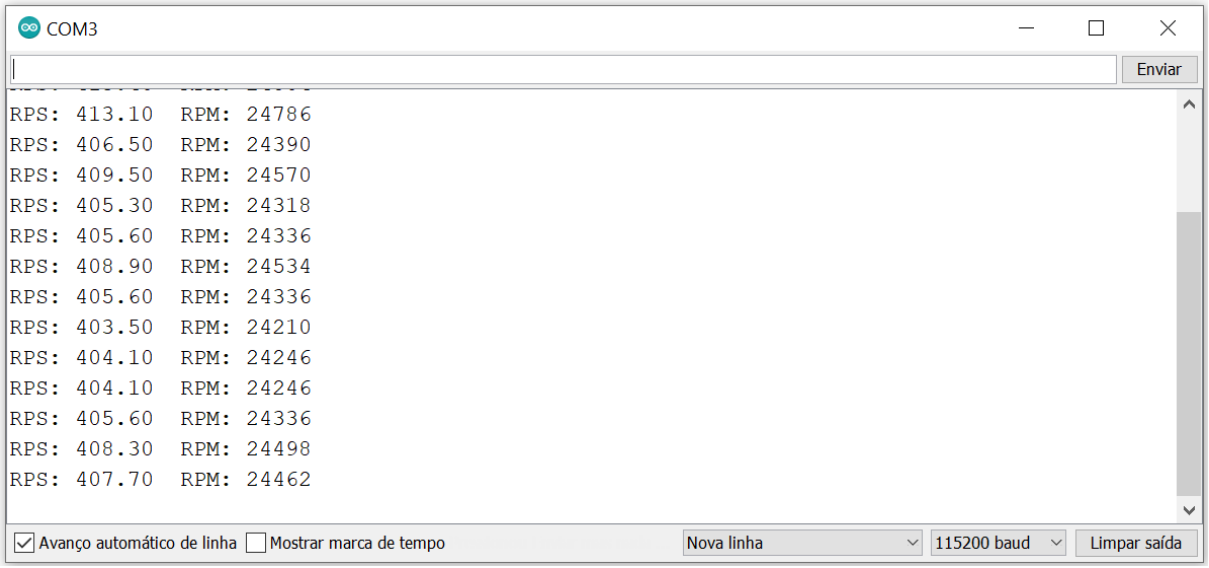


Figura 36 - *Valor de RPM com ruído*

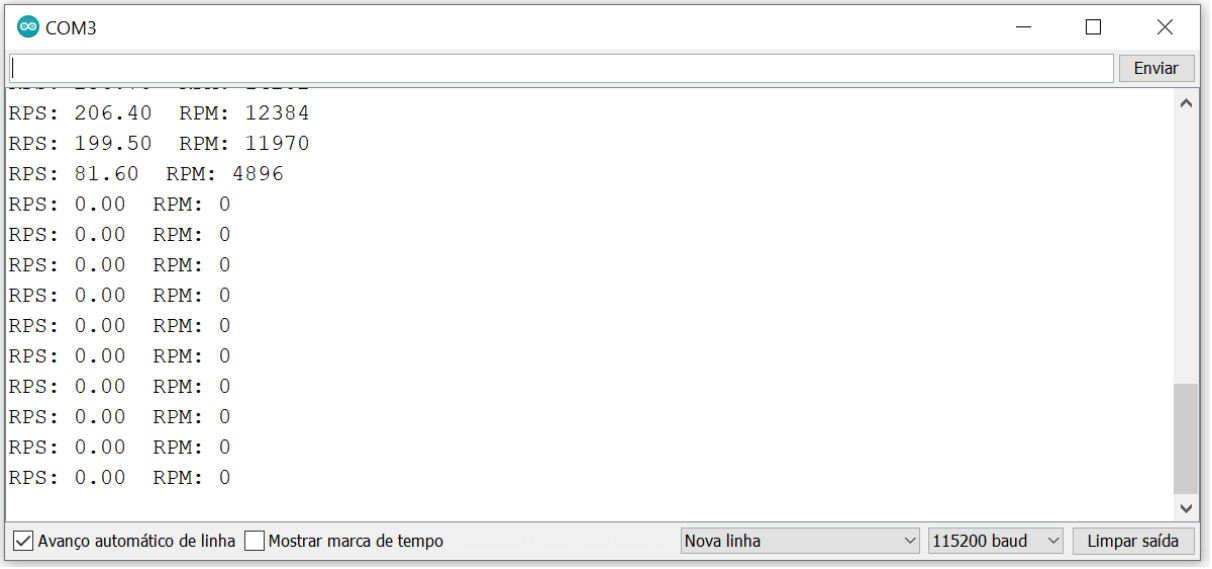


Figura 37 - *Valor de RPM sem ruído*

A maior dificuldade neste sensor foi a implementação prática, pois mesmo o sensor estando fora da zona de deteção, fazia medidas de como por exemplo 24 336 RPM, quando deveria apresentar o valor 0. Uma solução que foi encontrada foi a colocação de um condensador com capacidade de 100 nanofarad (nF), o que reduziu o ruido (*Figura 36*) e conseguiu filtrar dando assim valores de 0 quando não estava a medir (fora da zona de deteção) (*Figura 37*). O condensador foi colocado no circuito entre o ground e o pino de sinal.

Outra dificuldade foi o acréscimo de uma nova função capaz de enviar o valor das RPM para o web server, mas que ao fim de bastante tempo chegámos a conclusão que era necessário a introdução de mais algumas linhas de programação, linhas essas que podem ser encontradas com o resto do programa em Anexo 7.

```
//Função para receber e enviar informação para o web server sobre as rpm
void sendJsonRPM(String l_type, int l_rpm) {
  String jsonString = ""; // Variável para armazenar a string JSON
  const size_t CAPACITY = JSON_OBJECT_SIZE(2) + 30;  // Capacidade total do objeto JSON (2 campos + margem de segurança)
  StaticJsonDocument<CAPACITY> doc; // Criação de um objeto JSON estático

  doc["type"] = l_type; // Adiciona o tipo de dado ao objeto JSON
  doc["value"] = l_rpm; // Adiciona o valor ao objeto JSON

  serializeJson(doc, jsonString);     // Converter o objeto JSON em string
  webSocket.broadcastTXT(jsonString);  // Envia a string JSON para todos os clientes
}
```

5.4.5 Esquema de montagem

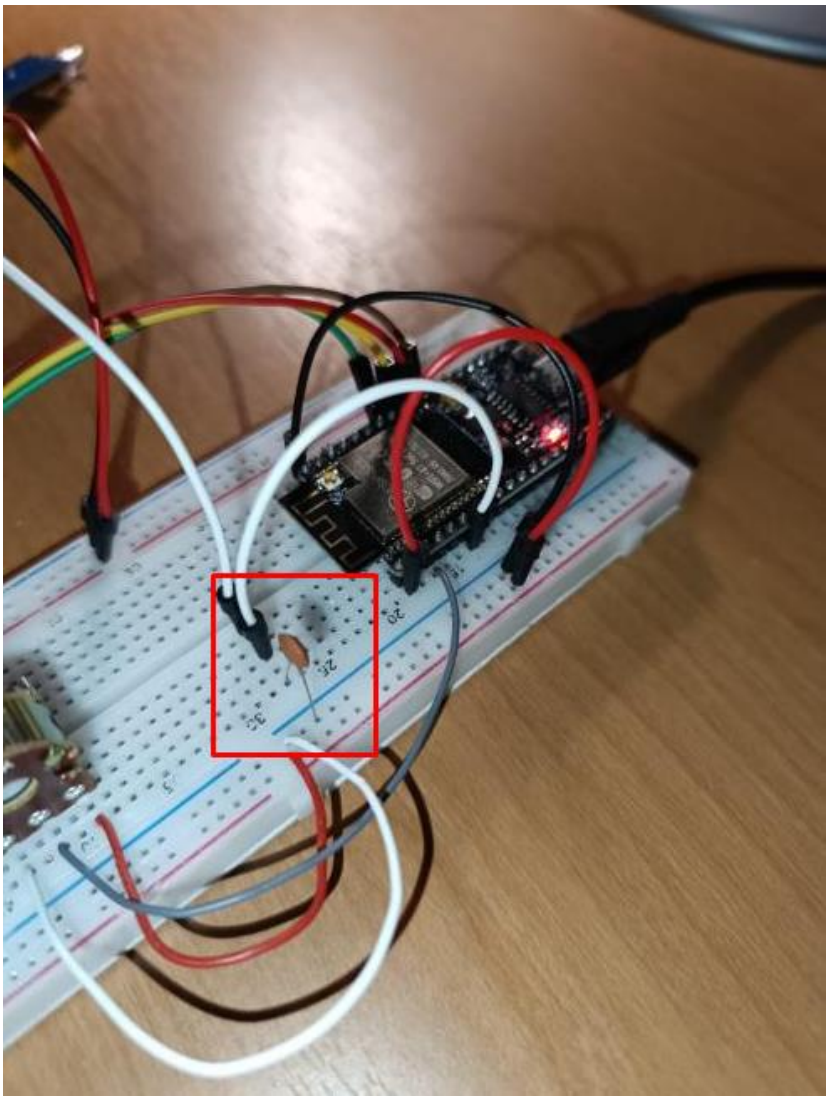


Figura 38 - *Valor de RPM sem ruído*

6. Conclusões

6.1 Introdução

O presente capítulo irá abordar alguns aspetos conclusivos, importantes para a exequibilidade deste projeto, mais concretamente na junção dos conhecimentos adquiridos ao longo desta unidade curricular.

Para além disso, ainda no conteúdo deste capítulo far-se-ão algumas considerações a determinados pontos que poderão ser no futuro, objeto de um desenvolvimento e análise mais detalhados.

Incorporar este conhecimento adquirido foi bastante importante pois conseguimos colocar a máquina a ser controlada manualmente, já remotamente foi um ponto que que não foi conseguido pois a ligação entre o motor de passo e o potenciómetro não está operacional. Foram realizadas as partes mais importantes como implementar web server e implementar o controlador do motor.

O material importado da China demorou a chegar e isso penalizou a execução do projeto, contudo a procura de informação, o processamento e a aplicação desta informação na prática assim como o arranjar soluções para os vários problemas com que nos deparámos, fez com que a Unidade Curricular de Projeto tenha sido enriquecedora para a nossa evolução.

6.2 Integração do Conhecimento

Apesar de o projeto não estar 100% funcional, conseguimos transmitir o conhecimento adquirido no presente relatório.

Foi realizada a conversão do alternador a motor e através do sequenciador de fases observámos que realmente ele funciona, concluindo assim que esta conversão foi bem conseguida.

Com o microcontrolador ESP32, foi implementado o controle do motor de passo através das leituras feitas pelo potenciómetro.

A transmissão de dados do web server para o ESP32, tais como as RPM e o controlo do ângulo, foram realizados com sucesso, porém o projeto não ficou 100% funcional devido à falta de

conexão física entre o potenciómetro e o controlador, onde seria possível demonstrar o projeto a funcionar como um todo.
As unidades curriculares de Máquinas Elétricas, Eletrónica e Eletrónica de Potência foram fundamentais para a realização do mesmo pois ajudou-nos a perceber o real funcionamento do alternador/motor, assim como perceber o que cada parte do alternador fazia, como por exemplo o regulador e a parte retificadora, o funcionamento do controlador do motor, a utilização de adc's no microcontrolador, etc.

Aliado a este projeto a U.C. de Programação Avançada também foi bastante importante quer na parte de programação C++ quer na programação HTML para a criação do web server.
Foi assim possível ver com maior acuidade a aplicação prática de matérias e conceitos lecionados durante todo o curso de licenciatura em Engenharia Eletrotécnica.

6.3 Perspetivas futuras

Como continuação do trabalho desenvolvido, seria interessante a implementação de sensores com o objetivo de garantir um melhor funcionamento:

- Colocar um sensor de corrente em cada linha de modo a proteger o motor (parte de potência), assim como a placa de circuito impresso (parte de comando) de sobreintensidades. Estes sensores também teriam o objetivo de verificar se o motor estaria equilibrado fazendo com que com a vida útil, aquecimento, operação e redução de ruídos fossem mais eficientes.
- Colocar sensor de vibração para ser realizado uma manutenção preditiva na máquina, de forma a evitar avarias futuras.

6.4 Em suma

Com a realização deste projeto foi nos proporcionado evoluir em diversas áreas do saber e sobretudo a ter espírito crítico, quer nos valores/medições que realizamos ao longo do tempo quer na escolha de soluções a adotar perante alguns problemas.

Foi necessário efetuar pesquisa diversa de informação em diferentes fontes e sobre os vários assuntos focados neste relatório assim como a seleção e utilização de componentes eletrónicos adequados à resolução dos problemas a resolver, até à organização destas linhas de texto que constituem o relatório.

REFERÊNCIAS

[1] Costa, G; "*Alternador do motor. O que é e como funciona?*", disponível em: https://www.razaoautomovel.com/autopedia/alternador-do-motor/, consultado em 6 de abril de 2023.

[2] ORACLE; "*Internet-of-things/what-is-iot*", disponível em: https://www.oracle.com/br/internet-of-things/what-is-iot, consultado em 7 de abril de 2023.

[3] MasterWalker ELETRONIC; "*conhecendo o nodemcu 32s esp32*", disponível em: https://blogmasterwalkershop.com.br/embarcados/esp32/conhecendo-o-nodemcu-32s-esp32, consultado em 12 de abril de 2023.

[4] pixFORCE; "*Impacto da tecnologia na industria conheça a revolução*", disponível em: https://www.pixforce.com.br/post/o-impacto-da-tecnologia-na-indústria-conheça-a revolução, consultado em 16 de abril de 2023.

[5] e-lee; "*Comparação estrela triangulo*", disponível em: http://e-lee.ist.utl.pt/realisations/CircuitsElectriques/SistemasTrifasicos/LigacaoCargas/3_aula.htm, consultado em 19 de abril de 2023.

[6] Micro Wire – Tech Lab; "*Sensor IR infravermelho*", disponível em: https://www.microwire.pt/sensor-ir-infravermelho-para-robot-seguidor-de-linha/, consultado em 22 de abril de 2023.

[7] MixTrónica; "*CCPM Servo Consistency Master*", disponível em: https://mixtronica.com/motor-modulos/20952-ccpm-servo-consistency-master.html, consultado em 6 de junho de 2023.

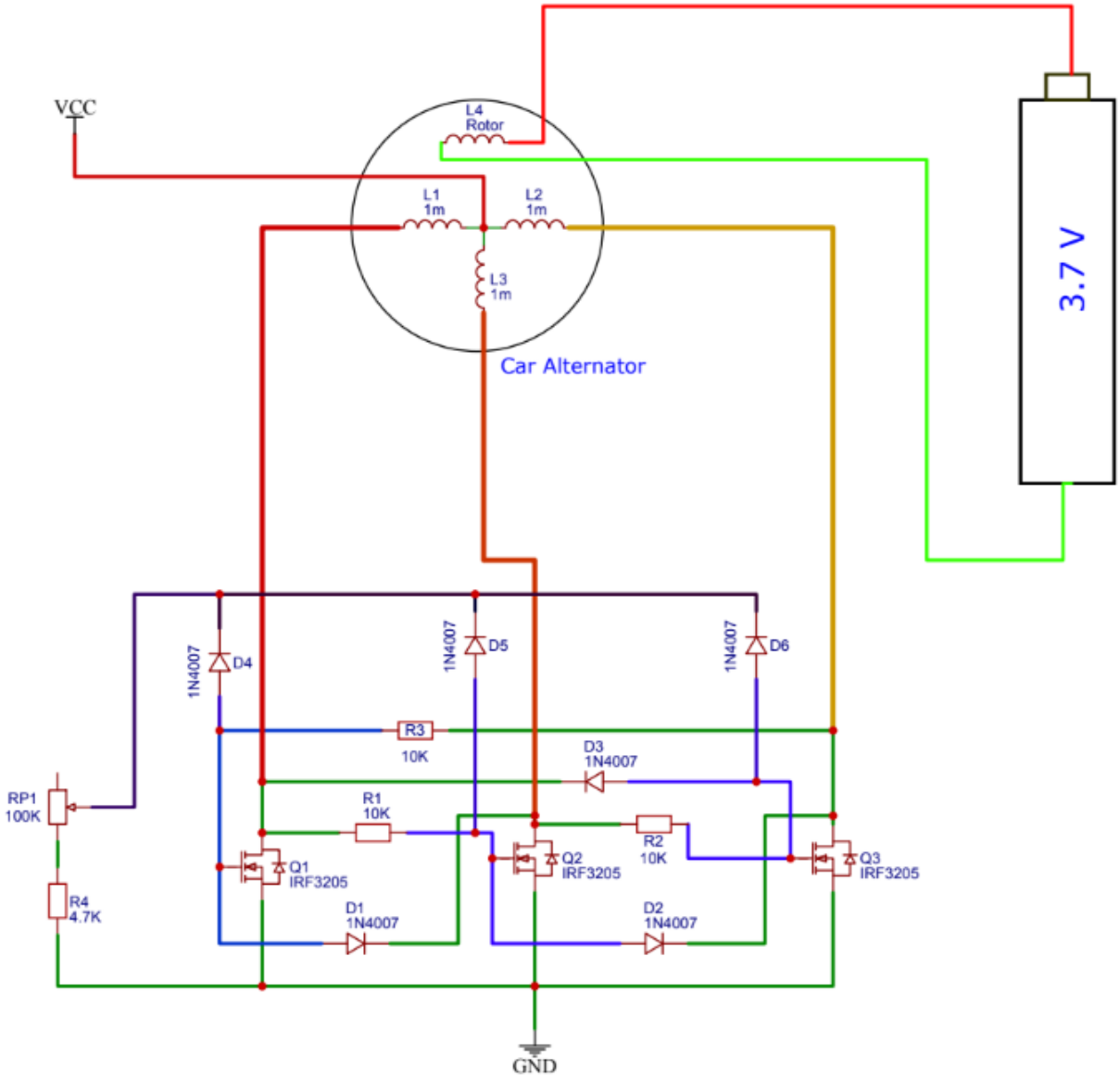
[8] aksotronik; "*Stepper motor 28BYJ-48 12V*", disponível em:https://www.aksotronik.com.pl/media/products_files/009433.pdf, consultado em 7 de julho de 2023.

[9] lastminuteengineers; "*Control 28BYJ-48 Stepper Motor with ULN2003 Driver & Arduino*", disponível em: In-Depth: Control 28BYJ-48 Stepper Motor with ULN2003 Driver & Arduino (lastminuteengineers.com), consultado em 7 de julho de 2023.

[10] Blog Eletrogate; "*Guia do Motor de Passo 28BYJ-48 + Driver ULN2003*",disponível em: https://blog.eletrogate.com/guia-completo-do-motor-de-passo-28byj-48-driver-uln2003/, consultado em 8 de julho de 2023.

[11] Bairros, P; "*Motor de Passo com Driver ULN2003*", disponível em: Motor de Passo com Driver ULN2003 - YouTube, consultado em 21 de junho de 2023.

Anexo 1 – Esquema de ligação



Anexo 2- Mosfet IRF3205

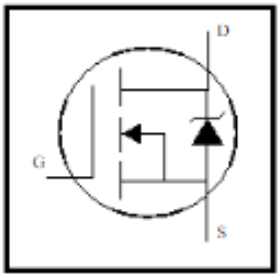
International
IOR Rectifier

PD-91279E

IRF3205

HEXFET® Power MOSFET

- l Advanced Process Technology
- l Ultra Low On-Resistance
- l Dynamic dv/dt Rating
- l 175°C Operating Temperature
- l Fast Switching
- l Fully Avalanche Rated



V_{DSS} = 55V

$R_{DS(on)}$ = 8.0mΩ

I_D = 110A

TO-220AB

Description

Advanced HEXFET® Power MOSFETs from International Rectifier utilize advanced processing techniques to achieve extremely low on-resistance per silicon area. This benefit, combined with the fast switching speed and ruggedized device design that HEXFET power MOSFETs are well known for, provides the designer with an extremely efficient and reliable device for use in a wide variety of applications.

The TO-220 package is universally preferred for all commercial-industrial applications at power dissipation levels to approximately 50 watts. The low thermal resistance and low package cost of the TO-220 contribute to its wide acceptance throughout the industry.

Absolute Maximum Ratings

	Parameter	Max.	Units
I_D @ T_C = 25°C	Continuous Drain Current, V_{GS} @ 10V	110	A
I_D @ T_C = 100°C	Continuous Drain Current, V_{GS} @ 10V	80	
I_{DM}	Pulsed Drain Current	390	
P_D @ T_C = 25°C	Power Dissipation	200	W
	Linear Derating Factor	1.3	W/°C
V_{GS}	Gate-to-Source Voltage	± 20	V
I_{AR}	Avalanche Current	62	A
E_{AR}	Repetitive Avalanche Energy	20	mJ
dv/dt	Peak Diode Recovery dv/dt	5.0	V/ns
T_J T_{STG}	Operating Junction and Storage Temperature Range	-55 to + 175	°C
	Soldering Temperature, for 10 seconds	300 (1.6mm from case)	
	Mounting torque, 6-32 or M3 srew	10 lbf•in (1.1N•m)	

Thermal Resistance

	Parameter	Typ.	Max.	Units
$R_{\theta JC}$	Junction-to-Case	.....	0.75	°C/W
$R_{\theta CS}$	Case-to-Sink, Flat, Greased Surface	0.50	.....	
$R_{\theta JA}$	Junction-to-Ambient	.....	62	

01/25/01

Electrical Characteristics @ T_J = 25°C (unless otherwise specified)

	Parameter	Min.	Typ.	Max.	Units	Conditions
$V_{(BR)DSS}$	Drain-to-Source Breakdown Voltage	55	—	—	V	V_{GS} = 0V, I_D = 250μA
$\Delta V_{(BR)DSS}/\Delta T_J$	Breakdown Voltage Temp. Coefficient	—	0.057	—	V/°C	Reference to 25°C, I_D = 1mA
$R_{DS(on)}$	Static Drain-to-Source On-Resistance	—	—	8.0	mΩ	V_{GS} = 10V, I_D = 62A ⓓ
$V_{GS(th)}$	Gate Threshold Voltage	2.0	—	4.0	V	V_{DS} = V_{GS}, I_D = 250μA
g_{fs}	Forward Transconductance	44	—	—	S	V_{DS} = 25V, I_D = 62A ⓓ
I_{DSS}	Drain-to-Source Leakage Current	—	—	25	μA	V_{DS} = 55V, V_{GS} = 0V
		—	—	250		V_{DS} = 44V, V_{GS} = 0V, T_J = 150°C
I_{GSS}	Gate-to-Source Forward Leakage	—	—	100	nA	V_{GS} = 20V
	Gate-to-Source Reverse Leakage	—	—	-100		V_{GS} = -20V
Q_g	Total Gate Charge	—	—	146	nC	I_D = 62A
Q_{gs}	Gate-to-Source Charge	—	—	35		V_{DS} = 44V
Q_{gd}	Gate-to-Drain ("Miller") Charge	—	—	54		V_{GS} = 10V, See Fig. 6 and 13
$t_{d(on)}$	Turn-On Delay Time	—	14	—	ns	V_{DD} = 28V
t_r	Rise Time	—	101	—		I_D = 62A
$t_{d(off)}$	Turn-Off Delay Time	—	50	—		R_G = 4.5Ω
t_f	Fall Time	—	65	—		V_{GS} = 10V, See Fig. 10 ⓓ
L_D	Internal Drain Inductance	—	4.5	—	nH	Between lead, 6mm (0.25in.) from package and center of die contact
L_S	Internal Source Inductance	—	7.5	—		
C_{iss}	Input Capacitance	—	3247	—	pF	V_{GS} = 0V
C_{oss}	Output Capacitance	—	781	—		V_{DS} = 25V
C_{rss}	Reverse Transfer Capacitance	—	211	—		f = 1.0MHz, See Fig. 5
E_{AS}	Single Pulse Avalanche Energy ⓔ	—	1050 ⓒ	264 ⓒ	mJ	I_{AS} = 62A, L = 138μH

Source-Drain Ratings and Characteristics

	Parameter	Min.	Typ.	Max.	Units	Conditions
I_S	Continuous Source Current (Body Diode)	—	—	110	A	MOSFET symbol showing the integral reverse p-n junction diode.
I_{SM}	Pulsed Source Current (Body Diode) ⓐ	—	—	390		
V_{SD}	Diode Forward Voltage	—	—	1.3	V	T_J = 25°C, I_S = 62A, V_{GS} = 0V ⓓ
t_{rr}	Reverse Recovery Time	—	69	104	ns	T_J = 25°C, I_F = 62A
Q_{rr}	Reverse Recovery Charge	—	143	215	nC	di/dt = 100A/μs ⓓ
t_{on}	Forward Turn-On Time	Intrinsic turn-on time is negligible (turn-on is dominated by L_S+L_D)				

Notes:

ⓐ Repetitive rating; pulse width limited by max. junction temperature. (See fig. 11)

ⓑ Starting T_J = 25°C, L = 138μH
R_G = 25Ω, I_{AS} = 62A. (See Figure 12)

ⓒ I_{SD} ≤ 62A, di/dt ≤ 207A/μs, V_{DD} ≤ $V_{(BR)DSS}$, T_J ≤ 175°C

ⓓ Pulse width ≤ 400μs; duty cycle ≤ 2%.

ⓔ Calculated continuous current based on maximum allowable junction temperature. Package limitation current is 75A.

ⓕ This is a typical value at device destruction and represents operation outside rated limits.

ⓖ This is a calculated value limited to T_J = 175°C.

Anexo 3- Díodos

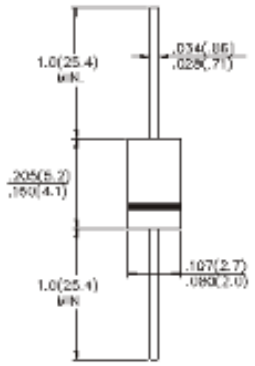
IN4001 thru IN4007

PLASTIC SILICON RECTIFIER

CHENG- YI
ELECTRONIC

VOLTAGE RANGE 50 TO 1000 Volts
CURRENT 1.0 Ampere

DO-41



Dimensions in inches and (millimeters)

FEATURE

- Low forward voltage
- High current capability
- Low leakage current
- High surge capability
- Low cost

MECHANICAL DATA

- Case:Molded plastic use UL 94V-0 recognized Flame retardant epoxy
- Terminals:Axial leads, solderable per MIL-STD-202, method 208
- Polarity:Color band denotes cathode
- Mounting Position:Any

MAXIMUM RATINGS AND ELECTRICAL CHARACTERISTICS

Single-phase, half-wave, 60Hz, resistive or inductive load

	IN4001	IN4002	IN4003	IN4004	IN4005	IN4006	IN4007	UNITS
Maximum Recurrent Peak Reverse Voltage	50	100	200	400	600	800	1000	V
Maximum RMS Voltage	35	70	140	280	420	560	700	V
Maximum DC Blocking Voltage	50	100	200	400	600	800	1000	V
Maximum Average Forward Rectified Current 3/8 Lead Length at T=75C	1.0							A
Maximum Overload Surge 8.3 ms single half sine-wave	50							A
Maximum Forward Voltage at 1.0A AC and 25 C	1.1							V
Maximum Full Load Reverse Current, Full Cycle Average at 75 C Ambient	30							μA
Maximum DC Reverse Current at 25 C at Rated DC Blocking Voltage at 75 C	5.0 50.0							μA μA
Typical Junction Capacitance (Note 1)	30							pF
Operating and Storage Temperature Range	-65 to +175							C

Notes : 1. Measured at 1.0MHz and applied reverse voltage of 4.0 VDC.
* JEDEC Registered Value.

Anexo 4- Programa Final motor de passo

```
//Variáveis
const int t = 5; // Tempo de atraso entre os passos
int angulo = 0; // Variável para armazenar o valor do ângulo
int passos = 0; // Variável para controlar o número de passos dados pelo motor
int graus = 0; // Variável para armazenar o valor do ângulo convertido em graus

// Configuração inicial
void setup() {
  // Inicializa os pinos de controle do motor como saídas
  pinMode(17, OUTPUT);
  pinMode(5, OUTPUT);
  pinMode(18, OUTPUT);
  pinMode(19, OUTPUT);

  // Inicia a comunicação série
  Serial.begin(115200);
}

// Loop principal
void loop() {
  // Verifica se há dados disponíveis na porta serial
  if (Serial.available() > 0) {
    // Lê o valor do ângulo da porta série
    angulo = Serial.parseInt();

    // Converter o ângulo em graus para o número de passos correspondentes
    graus = angulo * (512.0 / 360.0);

    // Verifica o valor do ângulo e controlar o motor de acordo com o número de graus
    if (angulo > 0) {
      while (graus > passos) {
        Direita(); // Gira para a direita
        passos = passos + 1;  // Incrementa o número de passos
        delay(t);
      }
      passos = 0;  // Reinicia o número de passos
      Serial.print(String(angulo) + " Graus para a Direita"+ "\n");
    }
```

```
    if (angulo < 0) {
      while (graus < passos) {
        Esquerda(); // Gira para a esquerda
        passos = passos - 1;  // Decrementa o número de passos
        delay(t);
      }
      passos = 0;  // Reinicia o número de passos
      Serial.print(String(angulo) + " Graus para a Esquerda"+ "\n");
    }
    else {
      Desligado();  // Motor desligado
    }
  }
}

  // Liga as bobines do motor girando-as para a esquerda
void Esquerda() {
  digitalWrite(17, LOW);
  digitalWrite(5, LOW);
  digitalWrite(18, HIGH);
  digitalWrite(19, HIGH);
  delay(t);
  digitalWrite(17, LOW);
  digitalWrite(5, HIGH);
  digitalWrite(18, HIGH);
  digitalWrite(19, LOW);
  delay(t);
  digitalWrite(17, HIGH);
  digitalWrite(5, HIGH);
  digitalWrite(18, LOW);
  digitalWrite(19, LOW);
  delay(t);
  digitalWrite(17, HIGH);
  digitalWrite(5, LOW);
  digitalWrite(18, LOW);
  digitalWrite(19, HIGH);
  delay(t);
}

// Liga as bobines do motor girando-as para a direita
void Direita() {
```

```
  digitalWrite(17, HIGH);
  digitalWrite(5, HIGH);
  digitalWrite(18, LOW);
  digitalWrite(19, LOW);
  delay(t);
  digitalWrite(17, LOW);
  digitalWrite(5, HIGH);
  digitalWrite(18, HIGH);
  digitalWrite(19, LOW);
  delay(t);
  digitalWrite(17, LOW);
  digitalWrite(5, LOW);
  digitalWrite(18, HIGH);
  digitalWrite(19, HIGH);
  delay(t);
  digitalWrite(17, HIGH);
  digitalWrite(5, LOW);
  digitalWrite(18, LOW);
  digitalWrite(19, HIGH);
  delay(t);
}

// Desliga todas as bobines do motor
void Desligado() {
  digitalWrite(17, LOW);
  digitalWrite(5, LOW);
  digitalWrite(18, LOW);
  digitalWrite(19, LOW);
}
```

Anexo 5 – Programa Final motor de passo com o potenciómetro

```
//Variáveis
const int t = 5; // Tempo de atraso entre os passos
int angulo = 0; // Variável para armazenar o valor do ângulo
int passos = 0; // Variável para controlar o número de passos dados pelo motor
int graus = 0; // Variável para armazenar o valor do ângulo convertido em graus

int potenciometro = 36;         // Pino do potenciómetro conectado ao ESP32
int valorMinimo = 0;            // Valor mínimo do potenciómetro
int valorPotenciometro = 0;  // Valor atual do potenciómetro
int valorMaximo = 4095;      // Valor máximo do potenciómetro

// Configuração inicial
void setup() {
  // Inicializa os pinos de controle do motor como saídas
  pinMode(17, OUTPUT);
  pinMode(5, OUTPUT);
  pinMode(18, OUTPUT);
  pinMode(19, OUTPUT);

  pinMode(potenciometro, INPUT);  // Configura o pino do potenciómetro como entrada

  // Inicia a comunicação série
  Serial.begin(115200);

  // Lê o valor inicial do potenciómetro
  valorPotenciometro = analogRead(potenciometro);
  Serial.print("Valor inicial do potenciómetro: " + String(valorPotenciometro) + "\n");

  // Ajusta o potenciómetro para zero
  while (valorPotenciometro > valorMinimo) {
    Esquerda();
    valorPotenciometro = analogRead(potenciometro);
    delay(t);
  }
  Desligado();
  Serial.print("Pronto para começar valor do potenciómetro a " + String(valorPotenciometro) +
"\n");
}
```

```
// Loop principal
void loop() {
  // Verifica se há dados disponíveis na porta série
  if (Serial.available() > 0) {
    angulo = Serial.parseInt(); // Lê o valor do ângulo da porta série

    // Converter o ângulo em graus para o número de passos correspondentes
    graus = angulo * (512.0 / 360.0);

    // Verifica se o ângulo é positivo e executa o movimento para a direita
    if (angulo > 0) {
      while (graus > passos && valorPotenciometro >= valorMinimo && valorPotenciometro < valorMaximo) {
        // Verifica se o número de graus restantes é maior que o número de passos já executados e se o valor do potenciómetro está dentro do intervalo permitido
        Direita(); // Executa o movimento para a direita
        valorPotenciometro = analogRead(potenciometro); // Lê o valor atual do potenciómetro
        passos = passos + 1; // Incrementa o número de passos
        delay(t);
      }
    }
      passos = 0;  // Reinicia o número de passos
      Serial.print(String(angulo) + " Graus para a Direita"+ " Valor do Potenciometro: " + String(valorPotenciometro) + "\n");
    }

    // Verifica se o ângulo é negativo e executa o movimento para a esquerda
    if (angulo < 0) {
      while (graus < passos && valorPotenciometro >56 valorMinimo && valorPotenciometro <= valorMaximo) {
        // Verifica se o número de graus restantes é maior que o número de passos já executados e se o valor do potenciómetro está dentro do intervalo permitido
        Esquerda(); // Gira para a esquerda
        valorPotenciometro = analogRead(potenciometro); // Lê o valor atual do potenciómetro
        passos = passos - 1;  // Decrementa o número de passos
        delay(t);
      }
      passos = 0;  // Reinicia o número de passos
      Serial.print(String(angulo) + " Graus para a Esquerda"+  " Valor do Potenciometro: " + String(valorPotenciometro) + "\n");
    }
```

```
    else {
      Desligado();  // Motor desligado
    }
  }
}

// Liga as bobines do motor girando-as para a esquerda
void Esquerda() {
  digitalWrite(17, LOW);
  digitalWrite(5, LOW);
  digitalWrite(18, HIGH);
  digitalWrite(19, HIGH);
  delay(t);
  digitalWrite(17, LOW);
  digitalWrite(5, HIGH);
  digitalWrite(18, HIGH);
  digitalWrite(19, LOW);
  delay(t);
  digitalWrite(17, HIGH);
  digitalWrite(5, HIGH);
  digitalWrite(18, LOW);
  digitalWrite(19, LOW);
  delay(t);
  digitalWrite(17, HIGH);
  digitalWrite(5, LOW);
  digitalWrite(18, LOW);
  digitalWrite(19, HIGH);
  delay(t);
}

// Liga as bobines do motor girando-as para a direita
void Direita() {
  digitalWrite(17, HIGH);
  digitalWrite(5, HIGH);
  digitalWrite(18, LOW);
  digitalWrite(19, LOW);
  delay(t);
  digitalWrite(17, LOW);
  digitalWrite(5, HIGH);
  digitalWrite(18, HIGH);
  digitalWrite(19, LOW);
```

```
  delay(t);
  digitalWrite(17, LOW);
  digitalWrite(5, LOW);
  digitalWrite(18, HIGH);
  digitalWrite(19, HIGH);
  delay(t);
  digitalWrite(17, HIGH);
  digitalWrite(5, LOW);
  digitalWrite(18, LOW);
  digitalWrite(19, HIGH);
  delay(t);
}

// Desliga todas as bobines do motor
void Desligado() {
  digitalWrite(17, LOW);
  digitalWrite(5, LOW);
  digitalWrite(18, LOW);
  digitalWrite(19, LOW);
}
```

Anexo 6 – Programa Final do motor de passo, com o potenciómetro e o web server

```
// Bibliotecas
#include <WiFi.h>              // Biblioteca para conetar ao WiFi
#include <WebServer.h>         // Biblioteca necessária para criar um servidor web simples
#include <WebSocketsServer.h>  // Biblioteca necessária para comunicação instantânea entre usuários e servidor por meio de websockets
#include <ArduinoJson.h>       // Biblioteca necessária para encapsular JSON (enviar várias variáveis com uma única string)

//Variáveis para controlar o ângulo
const int t = 5; // Tempo de atraso entre os passos
int angulo = 0; // Variável para armazenar o valor do ângulo
int passos = 0; // Variável para controlar o número de passos dados pelo motor
int graus = 0; // Variável para armazenar o valor do ângulo convertido em graus

//Variáveis para controlar o potenciómetro
int potenciometro = 36;     // Pino do potenciómetro conectado ao ESP32
int valorMinimo = 0;        // Valor mínimo do potenciómetro
int valorPotenciometro = 0; // Valor atual do potenciómetro
int valorMaximo = 4095;     // Valor máximo do potenciómetro

// Variáveis para controlar o WebSocket e a página HTML
String angleType = "Angulo";   // Tipo de ângulo
String dados;                  // Dados recebidos via WebSocket
int anguloAtual = 0;           // Valor atual do ângulo

// SSID e senha da conexão WiFi:
const char* ssid = "Net";
const char* password = "123456789";

// Página HTML
String website = R"=====(
<!DOCTYPE html>
<html>
<head>
  <title>Projeto 2023 - Controlo do Ângulo</title>
  <style>
    .slider {
```

```
      width: 80%;
      height: 50px;
    }
    .slider::-webkit-slider-thumb {
      height: 50px;
    }
  </style>
</head>
<body>
  <h1>Projeto 2023</h1>
  <h1>Controlo do Ângulo</h1>
  <input type='range' min='0' max='360' value='0' class='slider' id='angle-slider'><br>
  <label for='angle-value'> Ângulo: </label>
  <output id='angle-value'>0</output><br>

  <body onload='init()'>
    <script>
      var slider = document.getElementById('angle-slider');
    var output = document.getElementById('angle-value');
    var Socket;

      function init() {
        Socket = new WebSocket('ws://' + window.location.hostname + ':81/');

        Socket.onmessage = function(event) {
          processCommand(event);
        };

        const angleSlider = document.getElementById('angle-slider');
        const angleValue = document.getElementById('angle-value');

        angleSlider.addEventListener('input', () => {
          const angulo = angleSlider.value;
          angleValue.value = angulo;
          console.log('Ângulo: ' + angulo);

          var msg = { type: 'angulo', value: angulo };
          Socket.send(JSON.stringify(msg));
        });
      }
```

```
    function processCommand(event) {
      var obj = JSON.parse(event.data);
      var type = obj.type;

      if (type.localeCompare("angulo") == 0) {
        var angulo = obj.value;
        console.log('Valor do Ângulo: ' + angulo);
        slider.value = angulo;
        output.innerHTML = angulo;
      }
      else if (type.localeCompare("OutroTipo") == 0) {
        // Código para processar outro tipo de mensagem
      }
    }
    window.onload = function(event) {
    init();
    }
    </script>
  </body>
</html>
)=====";

WebServer server(80);                 // Servidor HTTP
WebSocketsServer webSocket = WebSocketsServer(81); // Servidor WebSocket

// Configuração inicial
void setup() {
  // Inicializa os pinos de controle do motor como saídas
  pinMode(17, OUTPUT);
  pinMode(5, OUTPUT);
  pinMode(18, OUTPUT);
  pinMode(19, OUTPUT);

  pinMode(potenciometro, INPUT);  // Configura o pino do potenciómetro como entrada

  Serial.begin(115200);  // Inicia a comunicação série

  // Inicia a conexão WiFi com as credenciais fornecidas (SSID e senha)
  WiFi.begin(ssid, password);

  // Escreve a mensagem informando que a conexão WiFi está sendo estabelecida
```

```
  Serial.println("Estabelecendo conexão com o WiFi com SSID: " + String(ssid));

  // Linha responsável por enviar os caracteres corretos para o web server
  server.sendHeader("Content-Type", "text/html; charset=UTF-8");

  // Aguarda até que a conexão WiFi seja estabelecida
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);  // Aguarda 1 segundo
    Serial.print(".");  // Escreve um ponto no monitor série de forma a indicar que a conexão está em andamento
  }

  // Escreve a mensagem indicando que a conexão WiFi foi estabelecida e exibe o endereço IP atribuído
  Serial.print("Conectado à rede com endereço IP: ");
  Serial.println(WiFi.localIP());

  // Configura a rota para a página principal
  server.on("/", []() {
    server.send(200, "text/html", website);
  });
  server.begin();

  // Inicia o servidor WebSocket
  webSocket.begin();
  webSocket.onEvent(webSocketEvent);

  // Aguarda até que a comunicação série esteja pronta
  while (!Serial);

  // Lê o valor inicial do potenciómetro
  valorPotenciometro = analogRead(potenciometro);
  Serial.print("Valor inicial do potenciómetro: " + String(valorPotenciometro) + "\n");

  // Ajusta o valor do potenciómetro para zero
  while (valorPotenciometro > valorMinimo) {
    Esquerda();
    valorPotenciometro = analogRead(potenciometro);
    delay(t);
  }
  Desligado();
```

```
  Serial.print("Pronto para começar, valor do potenciómetro a " + String(valorPotenciometro) +
"\n");
}

// Loop principal
void loop() {
  server.handleClient();   // Lida com as requisições HTTP dos clientes
  webSocket.loop();        // Lida com as conexões WebSocket
}

void webSocketEvent(byte num, WStype_t type, byte * payload, size_t length) {
  switch (type) {
    case WStype_DISCONNECTED:
      // Evento que mostra o que acontece quando um usuário é desconectado
      Serial.println("Usuário " + String(num) + " desconectado\n");
      break;
    case WStype_CONNECTED:
      // Evento que mostra o que acontece quando um usuário é conectado
      Serial.println("Usuário " + String(num) + " conectado\n");

      // Linha responsável por enviar os caracteres corretos para o web server
      server.sendHeader("Content-Type", "text/html; charset=UTF-8");

      // Envia o valor atual do ângulo para o novo usuário conectado
      sendJson("angulo", String(anguloAtual));
      break;
    case WStype_TEXT:
      // Evento que mostra o que acontece quando uma mensagem de texto é recebida do cliente
      StaticJsonDocument<200> doc;
      DeserializationError error = deserializeJson(doc, payload);
      if (error) {
        // Se ocorrer um erro, exibe uma mensagem de erro
        Serial.print(F("deserializeJson() falhou: \n"));
        Serial.println(error.f_str());
        return;
      }
      else {
        // Extrai o valor do ângulo recebido do objeto JSON (doc) e armazena na variável l_angulo
        const int l_angulo = doc["value"];

        Serial.print("Valor do Ângulo: " + String(l_angulo) + "\n");
```

```
// Lê o valor atual do potenciómetro
valorPotenciometro = analogRead(potenciometro);
Serial.print("Valor do potenciómetro inicial: " + String(valorPotenciometro) + "\n");

dados += String(l_angulo); // Dá o valor do ângulo (l_angulo) à variável dados


// Verifica se há dados suficientes para processar
if (dados.length() > 0) {
  // Converte os dados acumulados em graus (valor inteiro)
  graus = dados.toInt();

  delay(t);

  // Converter o ângulo em graus para o número de passos correspondentes
  graus = graus * (512.0 / 360.0);

  // Movimenta o motor para a posição desejada, conforme os graus calculados
  while (graus > passos && valorPotenciometro >= valorMinimo && valorPotenciometro < valorMaximo) {
    // Verifica se o número de graus restantes é maior que o número de passos já executados e se o valor do potenciómetro está dentro do intervalo permitido
    Direita(); // Executa o movimento para a direita
    valorPotenciometro = analogRead(potenciometro); // Lê o valor atual do potenciómetro
    passos = passos + 1; // Incrementa o número de passos
    delay(t);
  }

  while (graus < passos && valorPotenciometro > valorMinimo && valorPotenciometro <= valorMaximo) {
    // Verifica se o número de graus restantes é menor que o número de passos já executados e se o valor do potenciómetro está dentro do intervalo permitido
    Esquerda(); // Executa o movimento para a esquerda
    valorPotenciometro = analogRead(potenciometro); // Lê o valor atual do potenciómetro
    passos = passos - 1;  // Decrementa o número de passos
    delay(t);
  }
```

```
        anguloAtual = l_angulo; // Atualiza a variável anguloAtual com o valor recebido do WebSocket, representando o ângulo atual
        dados = ""; // Limpa os dados acumulados
        Desligado();  // Motor desligado
      }
       Serial.print("Valor do potenciómetro final: " + String(valorPotenciometro) + "\n");
    }
    // Após completar o movimento, envia o valor atual do ângulo para todos os clientes conectados
    sendJson("angulo", String(anguloAtual)); // Linha para envia o valor do ângulo
    Serial.println("");
    // Encerra o tratamento do evento WStype_TEXT
    break;
  }
}

//Função para enviar informação para os usuários Web
void sendJson(String l_type, String l_angulo) {
  String jsonString = "";  // Variável para armazenar a string JSON
  const size_t CAPACITY = JSON_OBJECT_SIZE(2) + 30; // Capacidade total do objeto JSON (2 campos + margem de segurança)
  StaticJsonDocument<CAPACITY> doc; // Criação de um objeto JSON estático
  doc["type"] = l_type; // Adiciona o tipo de dado ao objeto JSON
  doc["value"] = l_angulo; // Adiciona o valor ao objeto JSON
  // Converter os dados do objeto JSON em uma sequência de caracteres que segue a sintaxe do formato JSON para uma string JSON
  serializeJson(doc, jsonString);
  webSocket.broadcastTXT(jsonString);  // Envio da string JSON para todos os usuários WebSocket conectados
}

// Liga as bobines do motor girando-as para a esquerda
void Esquerda() {
  digitalWrite(17, LOW);
  digitalWrite(5, LOW);
  digitalWrite(18, HIGH);
  digitalWrite(19, HIGH);
  delay(t);
  digitalWrite(17, LOW);
  digitalWrite(5, HIGH);
  digitalWrite(18, HIGH);
```

```
  digitalWrite(19, LOW);
  delay(t);
  digitalWrite(17, HIGH);
  digitalWrite(5, HIGH);
  digitalWrite(18, LOW);
  digitalWrite(19, LOW);
  delay(t);
  digitalWrite(17, HIGH);
  digitalWrite(5, LOW);
  digitalWrite(18, LOW);
  digitalWrite(19, HIGH);
  delay(t);
}

// Liga as bobines do motor girando-as para a direita
void Direita() {
  digitalWrite(17, HIGH);
  digitalWrite(5, HIGH);
  digitalWrite(18, LOW);
  digitalWrite(19, LOW);
  delay(t);
  digitalWrite(17, LOW);
  digitalWrite(5, HIGH);
  digitalWrite(18, HIGH);
  digitalWrite(19, LOW);
  delay(t);
  digitalWrite(17, LOW);
  digitalWrite(5, LOW);
  digitalWrite(18, HIGH);
  digitalWrite(19, HIGH);
  delay(t);
  digitalWrite(17, HIGH);
  digitalWrite(5, LOW);
  digitalWrite(18, LOW);
  digitalWrite(19, HIGH);
  delay(t);
}

// Desliga todas as bobines do motor
void Desligado() {
  digitalWrite(17, LOW);
```

```
  digitalWrite(5, LOW);
  digitalWrite(18, LOW);
  digitalWrite(19, LOW);
}
```

Anexo 7 – Programa final - Programa do motor de passo, com o potenciómetro, web server e o sensor IR

```
// Bibliotecas
#include <WiFi.h>              // Biblioteca para conetar ao WiFi
#include <WebServer.h>         // Biblioteca necessária para criar um servidor web simples
#include <WebSocketsServer.h>  // Biblioteca necessária para comunicação instantânea entre usuários e servidor por meio de websockets
#include <ArduinoJson.h>       // Biblioteca necessária para encapsular JSON (enviar várias variáveis com uma única string)

//Variáveis para controlar o ângulo
const int t = 5; // Tempo de atraso entre os passos
int angulo = 0; // Variável para armazenar o valor do ângulo
int passos = 0; // Variável para controlar o número de passos dados pelo motor
int graus = 0; // Variável para armazenar o valor do ângulo convertido em graus

//Variáveis para controlar o potenciómetro
int potenciometro = 36;     // Pino do potenciómetro conectado ao ESP32
int valorMinimo = 0;        // Valor mínimo do potenciómetro
int valorPotenciometro = 0; // Valor atual do potenciómetro
int valorMaximo = 4095;     // Valor máximo do potenciómetro

// Variáveis para controlar o WebSocket e a página HTML
String angleType = "Angulo";   // Tipo de ângulo
String dados;                  // Dados recebidos via WebSocket
int anguloAtual = 0;           // Valor atual do ângulo

// SSID e senha da conexão WiFi:
const char* ssid = "Net";
const char* password = "123456789";

int Rotacao = 35;          // Pino do ESP32 conectado ao sensor IR
volatile unsigned long pulseCount = 0;   // Contador de pulsos
unsigned long lastTime = 0;     // Tempo da última leitura
unsigned long rpm = 0;          // RPM (rotações por minuto)
float rps = 0;                  // RPS (rotações por segundo)

// Página HTML
String website = R"=====(
```

```
<!DOCTYPE html>
<html>
<head>
  <title>Projeto 2023 - Controlo do Ângulo</title>
  <style>
    .slider {
      width: 80%;
      height: 50px;
    }
    .slider::-webkit-slider-thumb {
      height: 50px;
    }
  </style>
</head>
<body>
  <h1>Projeto 2023</h1>
  <h1>Controlo do Ângulo</h1>
  <input type='range' min='0' max='360' value='0' class='slider' id='angle-slider'><br>
  <label for='angle-value'>Ângulo:</label>
  <output id='angle-value'>0</output><br>
  <label for='rpm-value'>RPM:</label>
  <output id='rpm-value'>0</output>

  <body onload='init()'>
    <script>
      var slider = document.getElementById('angle-slider');
      var output = document.getElementById('angle-value');
      var rpmValue = document.getElementById('rpm-value'); // Corrigir o nome da variável
      var Socket;
      var rpm; // Declarar a variável "rpm" no escopo global

      function init() {
        Socket = new WebSocket('ws://' + window.location.hostname + ':81/');

        Socket.onmessage = function(event) {
          processCommand(event);
        };

        const angleSlider = document.getElementById('angle-slider');
        const angleValue = document.getElementById('angle-value');
        const rpmValue = document.getElementById('rpm-value');
```

```
        angleSlider.addEventListener('input', () => {
          const angulo = angleSlider.value;
          angleValue.value = angulo;
          console.log('Ângulo: ' + angulo);

          var msg = { type: 'angulo', value: angulo };
          Socket.send(JSON.stringify(msg));
        });


    Socket.addEventListener('message', (event) => {
          const data = JSON.parse(event.data);
          if (data.type === 'RPM') {
            rpm = data.value;
            rpmValue.value = rpm;
            console.log('RPM: ' + rpm);
          }
        });
      }

      function processCommand(event) {
       var obj = JSON.parse(event.data);
       var type = obj.type;

      if (type.localeCompare("angulo") == 0) {
         var angulo = obj.value;
         console.log('Valor do Ângulo: ' + angulo);
         slider.value = angulo;
         output.innerHTML = angulo;
       }
      }

    window.onload = function(event) {
       init();
    }
     </script>
   </body>
</html>
)=====";
```

```
WebServer server(80);                 // Servidor HTTP
WebSocketsServer webSocket = WebSocketsServer(81); // Servidor WebSocket

void Contar() {
  pulseCount++;   // Incrementa o contador de pulsos
}

// Configuração inicial
void setup() {
  Serial.begin(115200);  // Inicia a comunicação série

  // Inicializa os pinos de controle do motor como saídas
  pinMode(17, OUTPUT);
  pinMode(5, OUTPUT);
  pinMode(18, OUTPUT);
  pinMode(19, OUTPUT);

  pinMode(potenciometro, INPUT);  // Configura o pino do potenciómetro como entrada

  pinMode(Rotacao, INPUT);   // Configurar o pino do sensor IR como entrada

  // Registra a função de interrupção para detetar mudanças de estado do pino
  attachInterrupt(digitalPinToInterrupt(Rotacao), Contar, CHANGE);

  // Inicia a conexão WiFi com as credenciais fornecidas (SSID e senha)
  WiFi.begin(ssid, password);

  // Escreve a mensagem informando que a conexão WiFi está sendo estabelecida
  Serial.println("Estabelecendo conexão com o WiFi com SSID: " + String(ssid));

  // Aguarda até que a conexão WiFi seja estabelecida
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);  // Aguarda 1 segundo
    Serial.print(".");  // Escreve um ponto no monitor série de forma a indicar que a conexão está
em andamento
  }

  // Linha responsável por enviar os caracteres corretos para o web server
  server.sendHeader("Content-Type", "text/html; charset=UTF-8");
```

```
  // Escreve a mensagem indicando que a conexão WiFi foi estabelecida e exibe o endereço IP
atribuído
  Serial.print("Conectado à rede com endereço IP: ");
  Serial.println(WiFi.localIP());

  // Configura a rota para a página principal
  server.on("/", []() {
    server.send(200, "text/html", website);
  });
  server.begin();

  // Inicia o servidor WebSocket
  webSocket.begin();
  webSocket.onEvent(webSocketEvent);

  // Aguarda até que a comunicação série esteja pronta
  while (!Serial);

  // Lê o valor inicial do potenciómetro
  valorPotenciometro = analogRead(potenciometro);
  Serial.print("Valor inicial do potenciómetro: " + String(valorPotenciometro) + "\n");

  // Ajusta o valor do potenciómetro para zero
  while (valorPotenciometro > valorMinimo) {
    Esquerda();
    valorPotenciometro = analogRead(potenciometro);
    delay(t);
  }
  Desligado();
  Serial.print("Pronto para começar, valor do potenciómetro a " + String(valorPotenciometro) +
"\n");
}

// Loop principal
void loop() {
  server.handleClient();   // Lida com as requisições HTTP dos clientes
  webSocket.loop();        // Lida com as conexões WebSocket

  unsigned long currentTime = millis();   // Obtem o tempo atual em milissegundos

  if (currentTime - lastTime >= 1000) {   // Calcula as RPM a cada segundo
```

```
    rpm = ((pulseCount * 36000/ (currentTime - lastTime)))/2;   // Calcula as RPM
    rps = ((pulseCount * 600/ (float)(currentTime - lastTime)))/2;   // Calcula as RPS

    Serial.print("RPS: " + String(rps)+ "  RPM: " + String(rpm)+ "\n");

    pulseCount = 0;   // Reinicia o contador de pulsos
    lastTime = currentTime;   // Atualiza o tempo da última leitura
    sendJsonRPM("RPM", rpm);   // Chama a função sendJsonRPM() com o tipo "RPM" e o valor das RPM
  }
}

void webSocketEvent(byte num, WStype_t type, byte * payload, size_t length) {
  switch (type) {
    case WStype_DISCONNECTED:
      // Evento que mostra o que acontece quando um usuário é desconectado
      Serial.println("Usuário " + String(num) + " desconectado\n");
      break;
    case WStype_CONNECTED:
      // Evento que mostra o que acontece quando um usuário é conectado
      Serial.println("Usuário " + String(num) + " conectado\n");

      // Linha responsável por enviar os caracteres corretos para o web server
      server.sendHeader("Content-Type", "text/html; charset=UTF-8");

      // Envia o valor atual do ângulo para o novo usuário conectado
      sendJson("angulo", String(anguloAtual));

      break;
    case WStype_TEXT:
      // Evento que mostra o que acontece quando uma mensagem de texto é recebida do cliente
      StaticJsonDocument<200> doc;
      DeserializationError error = deserializeJson(doc, payload);
      if (error) {
        // Se ocorrer um erro, exibe uma mensagem de erro
        Serial.print(F("deserializeJson() falhou: \n"));
        Serial.println(error.f_str());
        return;
      }
      else {
```

```
// Extrai o valor do ângulo recebido do objeto JSON (doc) e armazena na variável l_angulo
const int l_angulo = doc["value"];

Serial.print("Valor do Ângulo: " + String(l_angulo) + "\n");

// Lê o valor atual do potenciómetro
valorPotenciometro = analogRead(potenciometro);
Serial.print("Valor do potenciómetro inicial: " + String(valorPotenciometro) + "\n");

dados += String(l_angulo); // Dá o valor do ângulo (l_angulo) à variável dados


// Verifica se há dados suficientes para processar
if (dados.length() > 0) {
  // Converte os dados acumulados em graus (valor inteiro)
  graus = dados.toInt();
  delay(t);

  // Converter o ângulo em graus para o número de passos correspondentes
  graus = graus * (512.0 / 360.0);

  // Movimenta o motor para a posição desejada, conforme os graus calculados
  while (graus > passos && valorPotenciometro >= valorMinimo && valorPotenciometro < valorMaximo) {
    // Verifica se o número de graus restantes é maior que o número de passos já executados e se o valor do potenciómetro está dentro do intervalo permitido
    Direita(); // Executa o movimento para a direita
    valorPotenciometro = analogRead(potenciometro); // Lê o valor atual do potenciómetro
    passos = passos + 1; // Incrementa o número de passos
    delay(t);
  }

  while (graus < passos && valorPotenciometro > valorMinimo && valorPotenciometro <= valorMaximo) {
    // Verifica se o número de graus restantes é menor que o número de passos já executados e se o valor do potenciómetro está dentro do intervalo permitido
    Esquerda(); // Executa o movimento para a esquerda
    valorPotenciometro = analogRead(potenciometro); // Lê o valor atual do potenciómetro
    passos = passos - 1;  // Decrementa o número de passos
```

```
            delay(t);
          }
          anguloAtual = l_angulo; // Atualiza a variável anguloAtual com o valor recebido do WebSocket, representando o ângulo atual
          dados = ""; // Limpa os dados acumulados
          Desligado();  // Motor desligado
        }
        Serial.print("Valor do potenciómetro final: " + String(valorPotenciometro) + "\n");
      }
      // Após completar o movimento, envia o valor atual do ângulo para todos os clientes conectados
      sendJson("angulo", String(anguloAtual)); // Linha que envia o valor do ângulo
      Serial.println("");
      // Encerra o tratamento do evento WStype_TEXT
      break;
  }
}

//Função para receber e enviar informação para o webserver sobre o angulo
void sendJson(String l_type, String l_angulo) {
  String jsonString = "";  // Variável para armazenar a string JSON
  const size_t CAPACITY = JSON_OBJECT_SIZE(2) + 30; // Capacidade total do objeto JSON (2 campos + margem de segurança)
  StaticJsonDocument<CAPACITY> doc; // Criação de um objeto JSON estático

  doc["type"] = l_type; // Adiciona o tipo de dado ao objeto JSON
  doc["value"] = l_angulo; // Adiciona o valor ao objeto JSON

  serializeJson(doc, jsonString); // Converter o objeto JSON em string
  webSocket.broadcastTXT(jsonString);  // Envio da string JSON para todos os usuários WebSocket conectados
}

//Função para receber e enviar informação para o webserver sobre as rpm
void sendJsonRPM(String l_type, int l_rpm) {
  String jsonString = ""; // Variável para armazenar a string JSON
  const size_t CAPACITY = JSON_OBJECT_SIZE(2) + 30;  // Capacidade total do objeto JSON (2 campos + margem de segurança)
  StaticJsonDocument<CAPACITY> doc; // Criação de um objeto JSON estático

  doc["type"] = l_type; // Adiciona o tipo de dado ao objeto JSON
```

```
  doc["value"] = l_rpm; // Adiciona o valor ao objeto JSON

  serializeJson(doc, jsonString);     // Converter o objeto JSON em string
  webSocket.broadcastTXT(jsonString);  // Envia a string JSON para todos os clientes
}

// Liga as bobines do motor girando-as para a esquerda
void Esquerda() {
  digitalWrite(17, LOW);
  digitalWrite(5, LOW);
  digitalWrite(18, HIGH);
  digitalWrite(19, HIGH);
  delay(t);
  digitalWrite(17, LOW);
  digitalWrite(5, HIGH);
  digitalWrite(18, HIGH);
  digitalWrite(19, LOW);
  delay(t);
  digitalWrite(17, HIGH);
  digitalWrite(5, HIGH);
  digitalWrite(18, LOW);
  digitalWrite(19, LOW);
  delay(t);
  digitalWrite(17, HIGH);
  digitalWrite(5, LOW);
  digitalWrite(18, LOW);
  digitalWrite(19, HIGH);
  delay(t);
}

// Liga as bobines do motor girando-as para a direita
void Direita() {
  digitalWrite(17, HIGH);
  digitalWrite(5, HIGH);
  digitalWrite(18, LOW);
  digitalWrite(19, LOW);
  delay(t);
  digitalWrite(17, LOW);
  digitalWrite(5, HIGH);
  digitalWrite(18, HIGH);
  digitalWrite(19, LOW);
```

```
  delay(t);
  digitalWrite(17, LOW);
  digitalWrite(5, LOW);
  digitalWrite(18, HIGH);
  digitalWrite(19, HIGH);
  delay(t);
  digitalWrite(17, HIGH);
  digitalWrite(5, LOW);
  digitalWrite(18, LOW);
  digitalWrite(19, HIGH);
  delay(t);
}

// Desliga todas as bobines do motor
void Desligado() {
  digitalWrite(17, LOW);
  digitalWrite(5, LOW);
  digitalWrite(18, LOW);
  digitalWrite(19, LOW);
}
```