

UNIVERSIDADE FEDERAL DO RIO GRANDE

ALEXANDRE MARQUES BALDEZ JUNIOR,
RODRIGO DE SOUZA TORMA

**SISTEMA DE COORDENADAS PARA POSICIONAMENTO DE INSTRUMENTOS DE
MEDIÇÃO EM UM TÚNEL DE VENTO**

**RIO GRANDE, RIO GRANDE DO SUL
2021**

ALEXANDRE MARQUES BALDEZ JUNIOR,
RODRIGO DE SOUZA TORMA

SISTEMA DE COORDENADAS PARA POSICIONAMENTO DE INSTRUMENTOS DE
MEDIÇÃO EM UM TÚNEL DE VENTO

Monografia apresentada à Universidade Federal do Rio
Grande como requisito parcial para obtenção de título de
Engenheiro Mecânico.

Orientador: Prof. Dr. Gustavo da Cunha Dias

Co-Orientador: Prof. Me. Letieri Rodrigues de Ávila

RIO GRANDE, RIO GRANDE DO SUL
2021

370

S2373d

Marques Baldez Junior, Alexandre

Sistema de Coordenadas para Posicionamento de Instrumentos de
Medição em um Túnel de Vento / Alexandre Marques Baldez Junior. –
Rio Grande, 2021.

102f.: il.

Trabalho de conclusão de curso (graduação) – Universidade Federal do
Rio Grande. Escola de Engenharia. Rio grande, 2021.

Orientador Gustavo Da Cunha Dias; Co-orientador: Letieri Rodrigues
de Ávila

1. Mesa cartesiana 2. Túnel de vento 3. Tubo de Pitot

I. Da Cunha Dias, Gustavo II. Rodrigues de Ávila, Letieri III. Titulo

UNIVERSIDADE FEDERAL DO RIO GRANDE

Reitor: Danilo Giroldo

Pró-Reitor de Graduação: Sibele da Rocha Martins

Coordenador do Curso: Márcio Ulguim Oliveira

**ALEXANDRE MARQUES BALDEZ JUNIOR,
RODRIGO DE SOUZA TORMA**

**SISTEMA DE COORDENADAS PARA POSICIONAMENTO DE INSTRUMENTOS DE
MEDIÇÃO EM UM TÚNEL DE VENTO**

Monografia apresentada à Universidade Federal do Rio Grande como requisito parcial para obtenção de título de Engenheiro Mecânico.

Aprovada em: 20 de maio de 2021

BANCA EXAMINADORA

Prof. Dr. Gustavo da Cunha Dias (Orientador)
Escola de Engenharia
Universidade Federal do Rio Grande

Prof. Me. Letieri Rodrigues de Ávila (Co-Orientador)
Escola de Engenharia
Universidade Federal do Rio Grande - FURG

Prof. Me. Fernanda Mazuco Clain
Escola de Engenharia
Universidade Federal do Rio Grande - FURG

AGRADECIMENTOS

Aos meus amados pais Alexandre Marques Baldez e Gladistani Malta Vaz Baldez por compreenderem a minha ausência enquanto eu me dedicava a realização deste trabalho, por me ensinarem o valor da educação e pelo apoio incondicional.

A Fernanda Chaves Lopes que também me ajudou, compreendeu e ensinou nessa caminhada.

Alexandre

Parte do Rodrigo

Rodrigo

Em especial gostaríamos de agradecer aos nossos queridos e incansáveis orientadores Gustavo da Cunha Dias e Letieri Rodrigues de Ávila por acreditarem em nossa capacidade, nos dar o suporte, amizade e terem sido indispensáveis neste ciclo.

Agradecemos a professora Fernanda Mazuco Clain que aceitou ser banca deste trabalho colaborando para sua melhoria e nossa formação.

À Universidade Federal do Rio Grande por todo crescimento pessoal e profissional que vem nos proporcionando

Aos amigos e companheiros de profissão que convivemos ao longo desses anos de curso, que nos incentivaram e tiveram impacto em nossas trajetórias.

RESUMO

O presente trabalho de graduação apresenta o projeto de um sistema de coordenadas cartesianas para posicionamento de instrumentos de medição situado no túnel de vento do laboratório da Universidade Federal do Rio Grande. A concepção do projeto partiu da necessidade de precisão e repetibilidade no deslocamento e posicionamento de instrumentos de medição que caracterizam o campo de velocidade e ou pressão na seção de teste do canal aerodinâmico. Para o desenvolvimento deste sistema, foram utilizadas plataformas de hardware livre, tais como o arduino, visando o baixo custo e fácil reprodução. O sistema consiste em um aplicativo que receberá a coordenada para onde a mesa deve se movimentar. O projeto pode ser dividido em três partes: mecânica, na qual envolve a disposição dos componentes mecânicos; elétrica, na qual se projetou os circuitos elétricos; e a programação, na qual foi desenvolvido o sistema de comando do aplicativo e a mesa. Logo, para a montagem do sistema eletrônico foram utilizados uma placa controladora Arduino Uno, drivers de potência, motores de passo, optoacopladores, encoders. Já para o sistema mecânico foram utilizados eixos, mancais, rolamentos, fusos e estrutura base/pórtico em alumínio.

Palavras-chave: Túnel de vento. Tubo de Pitot. Arduino. Mesa cartesiana

ABSTRACT

The present graduation work presents the project of a system of Cartesian coordinates for positioning measuring instruments located in the wind tunnel of the laboratory of the Federal University of Rio Grande. The design of the project started from the need for precision and repeatability in displacement and positioning of measuring instruments that characterize the speed and / or pressure field in the test section of the streamline. For the development of this system, free hardware platforms were used, such as arduino, aiming at low cost and easy reproduction. The system consists of an application that will receive the coordinate to where the table should move. The project can be divided into three parts: mechanical, in which it involves the disposition of mechanical components; electrical, on which the electrical circuits were designed; and the programming, in which the application's command system was developed and the cartesian table. Therefore, for the composition of the electronic system an Arduino Uno controller board, power drivers, stepper motors, optocouplers and encoders were used. For the mechanical system, axes, shafts, bearings, spindles and aluminum structure (base, tower) were used.

Keywords: Wind tunnel. Pitot tube. Arduino. Cartesian table.

LISTA DE ILUSTRAÇÕES

Figura 1 – Funcionamento das pressões dentro do tubo de Pitot.	16
Figura 2 – Mesa acionada por fuso.	18
Figura 3 – Mesa acionada por correias.	18
Figura 4 – Perfil v_slot 20x40 mm em alumínio.	21
Figura 5 – Dimensões do perfil 20x40 mm.	22
Figura 6 – Placa de conexão interna de 90°.	22
Figura 7 – Dimensões da placa de conexão interna de 90°.	23
Figura 8 – Estrutura da mesa cartesiana.	23
Figura 9 – Detalhe do encaixe a 45° da base da estrutura.	24
Figura 10 – Perfil v_slot 20x20 mm em alumínio.	24
Figura 11 – Dimensões do perfil 20x20 mm.	25
Figura 12 – Placa T simples de aço.	25
Figura 13 – Dimensões da placa T simples.	26
Figura 14 – Fluxograma do sistema eletrônico.	28
Figura 15 – Descrição dos componentes da placa Arduino.	30
Figura 16 – Driver A4988.	32
Figura 17 – Portas do driver A4988.	33
Figura 18 – Conceito didático do motor de passo.	35
Figura 19 – Esquema elétrico do motor de passo.	36
Figura 20 – Fonte do sistema.	40
Figura 21 – Funcionamento do acoplador no sistema.	40
Figura 22 – Ambiente de desenvolvimento integrado Arduino.	42
Figura 23 – Fluxo de execução do software.	43
Figura 24 – Diagrama de classes do sistema de software presente no Arduino.	45
Figura 25 – Diagrama da organização geral do software.	50
Figura 26 – Fluxograma para apresentar a integração sistemas.	51

LISTA DE TABELAS

Tabela 1 – Parâmetros do driver de potência.	32
Tabela 2 – Lista de terminais do driver A4988.	34
Tabela 3 – Sequência de passos com uma fase (wavestep) para movimentação no sentido horário.	37
Tabela 4 – Sequência de passos com duas fases (fullstep) para movimentação no sentido horário.	37
Tabela 5 – Sequência de passos com meio passo (halfstep) para movimentação no sentido horário.	37
Tabela 6 – Sequência de passos com uma fase (wavestep) para movimentação no sentido anti-horário.	37
Tabela 7 – Sequência de passos com duas fases (fullstep) para movimentação no sentido anti-horário.	38
Tabela 8 – Sequência de passos com meio passo (halfstep) para movimentação no sentido anti-horário.	38
Tabela 9 – Parâmetros dos motores de passo.	38
Tabela 10 – Demanda de energia elétrica de cada componente do sistema.	39
Tabela 11 – Declaração e funcionalidade dos atributos e métodos da classe Sigmoidal. .	46
Tabela 12 – Declaração e funcionalidade dos atributos e métodos da classe Pino. . . .	47
Tabela 13 – Declaração e funcionalidade dos atributos e métodos da classe Driver. . . .	48
Tabela 14 – Declaração e funcionalidade dos atributos e métodos da classe Eixo. . . .	49

LISTA DE ABREVIATURAS E SIGLAS

CNC Controle Numérico Computadorizado

EEPROM Electrically Erasable Programmable Read-Only Memory

ICSP In Circuit Serial Programming

IDE Integrated Development Environment

PWM Pulse Width Modulation

RAM Random Access Memory

RISC Reduced Instruction Set Computer

USB Universal Serial Bus

LISTA DE SÍMBOLOS

g	Força gravitacional
ρ_{ar}	Massa específica do ar
ρ	Massa específica do fluido
P_{atm}	Pressão atmosférica
p_1	Pressão estática no ponto 1
pd	Pressão dinâmica
pe	Pressão estática
pt	Pressão total
p_2	Pressão total
V_1	Velocidade do fluído no ponto 1

SUMÁRIO

1	INTRODUÇÃO	13
1.1	OBJETIVO GERAL	13
1.2	OBJETIVOS ESPECÍFICOS	14
1.3	ESTRUTURA DO TRABALHO	14
2	REFERENCIAL TEÓRICO	15
2.1	TÚNEL DE VENTO	15
2.2	TUBO DE PITOT	16
2.3	MESA DE POSICIONAMENTO	17
2.4	TRABALHOS RELACIONADOS	19
3	METODOLOGIA	21
3.1	SISTEMA MECÂNICO	21
3.1.1	Estrutura	21
3.1.2	Sistema de transmissão	26
3.1.3	Acionador	26
3.2	SISTEMA ELETRÔNICO	27
3.2.1	Placa de prototipagem eletrônica Arduino	28
3.2.2	Drivers de potência	31
3.2.3	Atuadores	34
3.2.4	Fonte de alimentação	38
3.2.5	Acopladores ópticos	40
3.2.6	Encoders	41
3.2.7	Chaves fim de curso	41
3.3	SISTEMA DE SOFTWARE	42
3.3.1	Plataforma de prototipação Arduino IDE	42
3.3.2	Logica de programação	43
3.3.3	Diagrama de classes	44
3.4	INTEGRAÇÃO DOS SISTEMAS	50
4	RESULTADOS E DISCUSSÃO	52
4.1	SISTEMA MECÂNICO	52
4.2	SISTEMA ELETRÔNICO	52

4.3	SISTEMA DE SOFTWARE	52
4.3.1	Código header da classe Pino	52
4.3.2	Código header da classe Sigmoidal	53
4.3.3	Código header da classe Driver	54
4.3.4	Código header da classe Eixo	57
4.3.5	Detalhamento do arquivo principal (MesaCartesiana.ino)	58
5	CONSIDERAÇÕES FINAIS	60
5.1	CRÍTICAS E SUGESTÕES DE TRABALHOS FUTUROS	60
	REFERÊNCIAS	62
	APÊNDICES	63
	APÊNDICE A – CÓDIGO PRINCIPAL PARA O CONTROLE DA MESA CARTESIANA	64
	APÊNDICE B – CÓDIGO DA CLASSE EIXO PARA O CONTROLE DO EIXO	69
	APÊNDICE C – CÓDIGO DA CLASSE DRIVER PARA O CONTROLE DO DRIVER DE POTÊNCIA	72
	APÊNDICE D – CÓDIGO DA CLASSE PINO PARA O CONTROLE DOS PINOS	76
	APÊNDICE E – CÓDIGO DA CLASSE SIGMOIDAL PARA O CON- TROLE DA ACELERAÇÃO DOS MOTORES	78
	APÊNDICE F – CÓDIGO DO HEADER STDIVER PARA O INCLUDE DE CLASSES	80

1 INTRODUÇÃO

A mecânica dos fluidos é uma área muito complexa da engenharia, mesmo com todo o avanço tecnológico e computacional nem sempre é possível projetar com precisão sem se valer de uma análise prévia da ação de esforços sobre algum material. O estudo da ação do ar sobre estruturas pode ser um fator determinante entre o sucesso e o fracasso de um projeto.

A análise aerodinâmica, se bem conduzida, pode apresentar dados confiáveis ao projetista para o apoio na tomada de decisão. Uma das maneiras de realizar estes estudos é através das leis de similaridade que aplicam os adimensionais de fatores de escala para replicar resultados em escalas reais. Assim, de uma forma menos onerosa é possível se fazer esse estudo em escala reduzida e com condições controladas em laboratório. Os túneis de vento são as bancadas de testes para estudos de escoamento de ar, onde é possível simular cenários e avaliar a interação do fluido e estrutura.

Complementar a um controle de escoamento no túnel de vento estão os instrumentos para as grandezas físicas (i. e. pressão e velocidade). O tubo de Pitot e a Sonda de anemômetro de fio quente são exemplos de instrumentos de medição, usados dentro desses canais aerodinâmicos.

A operação desses equipamentos dentro do túnel de vento pode ser de forma manual ou automatizada. A primeira tende a gerar imprecisões, como o posicionamento incorreto do sensor em relação ao seus eixos vertical e horizontal, além de um gasto considerável de tempo e energia, pois o operador, para fazer o reposicionamento do equipamento, deve desligar o túnel, abri-lo e posicionar o equipamento para fazer a próxima medição, gerando assim incertezas de operação. Já a atuação automatizada resolve todos os problemas acima citados, porém o processo de implantação é mais dispendioso.

A realização desse trabalho justifica-se por desenvolver um equipamento que agregue um sistema de coordenadas bidimensional para posicionamento de instrumentos de medição no túnel de vento do Laboratório de Sistemas Térmicos da Universidade Federal do Rio Grande.

1.1 OBJETIVO GERAL

Desenvolver um dispositivo para o posicionamento de equipamentos de medições dentro de um túnel de vento para facilitar o processo de avaliação de velocidades e pressões de forma automatizada.

1.2 OBJETIVOS ESPECÍFICOS

- Projetar a mesa cartesiana.
- Criar o sistema de comunicação entre a mesa e o software.
- Desenvolver o software que comandará a mesa cartesiana.

1.3 ESTRUTURA DO TRABALHO

Na primeira seção são apresentados: o tema do projeto, os objetivos, a justificativa e a estrutura do trabalho.

A segunda seção apresenta a revisão bibliográfica a fim de ser referência neste estudo e para fundamentar a base teórica utilizada no trabalho.

A terceira seção apresenta a metodologia e detalhamento dos componentes do projeto em si, que envolve o projeto do sistema mecânico, sistema elétrico, desenvolvimento do software e a integração dos sistemas.

A quarta seção apresenta os resultados referentes a cada sistema.

A quinta seção apresenta as considerações finais, críticas e sugestões para trabalhos futuros.

Por último são dispostas as referências bibliográficas e apêndices.

2 REFERENCIAL TEÓRICO

Esta seção dispõe de uma breve revisão bibliográfica de assuntos referentes ao tema do projeto, que são os seguintes: túneis de vento, tubos de Pitot e mesas de posicionamento. Por fim, apresenta trabalhos relacionados que envolvem estes assuntos.

2.1 TÚNEL DE VENTO

Os túneis de vento são estruturas que propiciam a simulação para o desenvolvimento de estudos que relacionam o efeito do movimento de ar em torno de objetos, como turbinas, aviões, carros e edificações. Sua estrutura é composta por um duto de diâmetro adequado onde o ar é empurrado ou succionado por um ventilador. No interior do duto, o ar é analisado através de instrumentos de medição.

O primeiro túnel de vento foi construído na Inglaterra em 1871 por Frank H. Wenham (1824-1908), engenheiro naval britânico e membro da Sociedade Aeronáutica da Grã-Bretanha, esse túnel era de circuito fechado e acionado por uma máquina a vapor. Os estudos de Wenham permitiram um aperfeiçoamento no alongamento de uma asa relacionado à força de sustentação (CARMINATTI; KONRATH, 2019).

Em 1897 foi construído por Konstantin Tsiolkovsky o primeiro túnel de vento Russo que era de circuito aberto com um ventilador centrífugo e determinou os coeficientes de arrasto de placas planas, cilindros e esferas (JOGLEKAR; MOURYA, 2014).

Devido às guerras, a produção de túneis de vento teve uma demanda aumentada, pois era necessário a execução de ensaios em aeronaves militares. Já após o período de guerras, os túneis de vento ganharam relevância quando o objetivo foi aumentar a eficiência na aerodinâmica dos carros (SANTOS et al., 2014).

Os túneis de vento podem ser classificados quanto ao circuito que pode ser aberto ou fechado, quanto a velocidade de escoamento em relação à velocidade do som que é chamada de número de Mach (Ma) definindo os escoamentos como sônico, subsônico, supersônico e hipersônico e quanto ao sentido do escoamento que nos túneis de vento de circuito aberto podem ser soprador e sugador, sendo definido pela condição de trabalho do ventilador (PRITCHARD; MITCHELL, 2005).

O túnel de vento tratado neste trabalho está situado junto ao Laboratório de Sistemas Térmicos da Universidade Federal do Rio Grande e é de característica subsônica, circuito aberto

e do tipo soprador.

2.2 TUBO DE PITOT

O tubo de Pitot foi um equipamento criado por Henri Pitot em 1732 para medição da vazão do rio Sena. Pitot de maneira intuitiva apresentou que a altura de uma coluna de líquido conectada ao seu tubo era proporcional à raiz quadrada da velocidade. Ele desenvolveu a técnica mais comum para determinação da velocidade de um fluido, pois a utilização desse tubo é simplificada, além do baixo custo.

O tubo de Pitot apresenta vantagens como sua flexibilidade na utilização de diferentes faixas de velocidade desde o regime subsônico ou supersônico, sendo possível a obtenção de velocidades com alta precisão. No entanto apresenta desvantagens de falta de precisão em baixas velocidade, impossibilidade de medição em escoamentos reversos e dificuldade de obtenção de medições em alta frequência.

Com essa técnica, Pitot obteve a velocidade em um escoamento incompressível em uma área pontual, sendo que é necessário o tubo ser posicionado de modo a ficar alinhado com o escoamento. Com isso se mede a pressão estática e a pressão total ou de estagnação. A subtração da pressão total da estática resulta na pressão dinâmica do escoamento (PRITCHARD; MITCHELL, 2005).

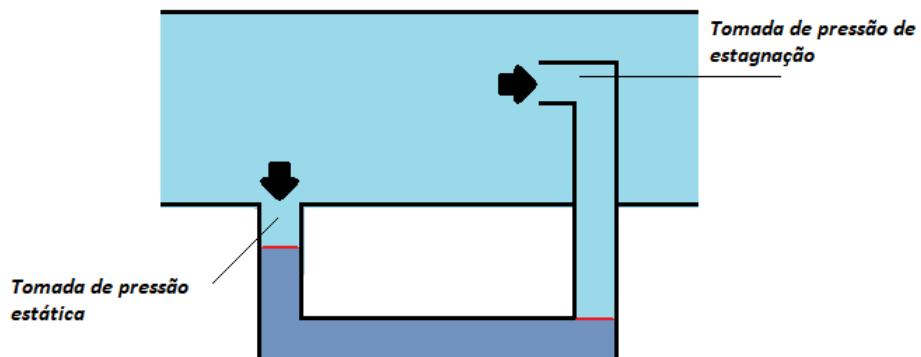


Figura 1 – Funcionamento das pressões dentro do tubo de Pitot.

Fonte: Próprio autor.

Seu princípio de funcionamento está baseado na conhecida equação de Bernoulli onde a pressão dinâmica (p_d) é igual a pressão total (p_t) menos a pressão estática (p_e) (PRITCHARD; MITCHELL, 2005).

$$pd = pt - pe \quad (2.1)$$

$$\frac{p_1}{\rho} + \frac{1}{2} \cdot (V_1)^2 + g \cdot z_1 = \frac{p_2}{\rho} + \frac{1}{2} \cdot (V_2)^2 + g \cdot z_2 = cte \quad (2.2)$$

$$p_e + \frac{1}{2} \cdot \rho \cdot (V)^2 + \rho \cdot g \cdot z_e = p_t + \frac{1}{2} \cdot \rho \cdot (0)^2 + \rho \cdot g \cdot z_t = cte \quad (2.3)$$

$$V = \sqrt{\frac{2 \cdot (p_t - p_e)}{\rho}} \quad (2.4)$$

$$\rho_{ar} = \frac{P_{atm}}{R_{ar} \cdot T_{ar}} \quad (2.5)$$

Sendo que p_1 é a pressão estática no ponto 1, V_1 é a velocidade do fluido no ponto 1, ρ é a massa específica do fluido, g é a força gravitacional e p_2 é a pressão total.

A equação 2.4 serve para obter a velocidade a partir da pressão e é obtida através da equação de Bernoulli para regime permanente, incompressível e sem atrito conforme a equação 2.2. Percebe-se que a massa específica é função da condição de estado do ar durante os testes sendo determinada a partir da Equação 2.5. Onde R_{ar} é e T_{ar} é...

2.3 MESA DE POSICIONAMENTO

Utilizada para várias finalidades como, posicionamento de peças que serão usinadas em máquinas de Controle Numérico Computadorizado (CNC), automação de laboratórios, armazenamento de cargas, impressões 3D entre outros tantos propósitos, as mesas cartesianas tem como função principal o posicionamento de alguma ferramenta, para executar algum tipo de serviço.

Também conhecidas como mesa de posicionamento XY, pois como o nome sugere, é uma estrutura com dois eixos de liberdade que permite o posicionamento da peça ou da ferramenta em algum lugar de um plano pré-definido. Podem ser classificadas em dois tipos com relação a

sua transmissão: as mesas acionadas por fusos conforme apresenta a Figura 2 e as acionadas por correias sincronizadas demonstradas na Figura 3.

As mesas de posicionamentos de fuso possuem um alto rendimento, próximo de 95%, um baixo desgaste e uma velocidade máxima de 3 m/s, já as mesas acionadas por correias sincronizadas podem desenvolver velocidades de até 5 m/s, conseguindo altas acelerações devido a sua inércia (ROCHA et al., 2015).

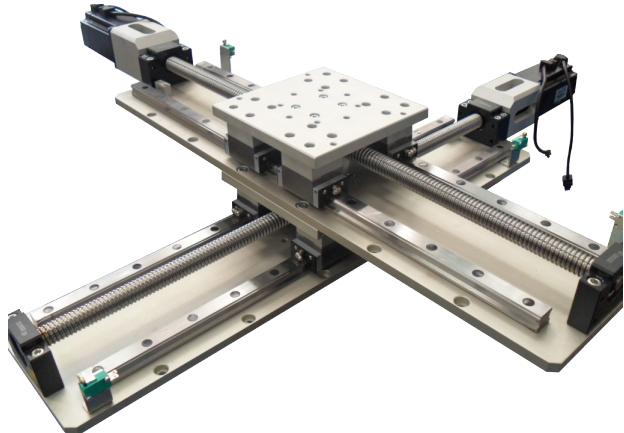


Figura 2 – Mesa acionada por fuso.

Fonte: <https://www.kalatec.com.br/mesa-de-coordenada-xy/>



Figura 3 – Mesa acionada por correias.

Fonte: <https://www.ccmrails.com/2019/01/26/packing-automation/>

Outro componente importante nesse mecanismo é o acionador, que pode ser um motor de passo ou um servomotor. Os motores de passo são máquinas que exercem um papel muito importante atualmente, são utilizados em aplicações onde é requerido um alto grau de precisão no movimento e que este movimento seja feito em passos fixos, referentes a uma fração de

ângulo. Empregados normalmente quando se deseja controlar uma combinação entre a posição do rotor (ângulo), com a devida velocidade e o sincronismo (JÚNIOR; SILVA, 2018).

O funcionamento desse motor se dá através dos princípios do eletromagnetismo, com estatores bobinados e um rotor formado por ímãs permanentes ligados ao eixo. Quando o estator é energizado cria um campo magnético e o rotor move-se para alinhar os ímãs (pólos norte e sul) com as linhas de fluxo magnético, formados pelo estator, movendo o eixo em um ângulo pequeno chamado de passo e continua a girar conforme o incremento angular controlado por circuitos eletrônicos. Esses circuitos digitais tem como função repassar a informação, um pulso, recebido pelo sistema de controle para o motor, que gira com grande precisão conforme seu controle. Como característica marcante, os motores com ímãs permanentes apresentam um torque estático quando não submetidos à tensão devido a força magnética entre os ímãs e o estator, servindo como freio para o sistema.

2.4 TRABALHOS RELACIONADOS

Nesta seção serão apresentados trabalhos relacionados ao tema deste projeto apresentando os objetivos e resultados de cada um.

O trabalho realizado por BUTIGNOL (2017), tinha o objetivo de desenvolver uma adequação de uma mesa XYZ didática acionada por motores de passo para o estudo de programação em microcontroladores e seu posicionamento em duas dimensões.

Como resultado do trabalho, BUTIGNOL (2017) desenvolveu um aparato eletromecânico de posicionamento de dois eixos e um atuador no terceiro eixo capaz de auxiliar no ensino de microcontroladores. Por fim, o projeto está disponível em <<https://mesaxydidatica.blogspot.com.br>> para sua montagem em outras instituições de ensino.

O trabalho realizado por CAMARGO et al. (1988), tinha o objetivo de desenvolver um sistema posicionador de baixo custo com CNC, utilizando componentes nacionalizados e com uma complexidade mínima no sistema de comando.

Como resultado do trabalho, CAMARGO et al. (1988) fez uma análise comparando diversos parâmetros como: massa, frequência natural amortecida, amplitude média da curva de resposta, perdas de passo, erros de posicionamento, vibração entre outros a fim de demonstrar que a concepção projetada e executada para a mesa de coordenadas XY presente em seu estudo, se justifica com os resultados obtidos.

O trabalho realizado por RAMOS (2018), tinha o objetivo de projetar e construir uma

mesa cartesiana para ser colocada nos túneis aerodinâmicos existentes no Departamento de Engenharia Mecânica e Industrial da Faculdade de Ciências e Tecnologia Universidade Nova de Lisboa, para assim, dar a capacidade de deslocamento de sensores como: tubo de Pitot, anemômetro de fio quente, anemômetro laser-Doppler para qualquer ponto na seção escolhida.

Como resultado do trabalho, RAMOS (2018) construiu uma mesa cartesiana funcional com capacidade de efetuar medições de velocidade em uma malha de cem posições, podendo inserir coordenadas através da porta serial do Arduino. Ainda comprovou por testes experimentais a precisão dos motores de passo ao comprovar que não existe diferença mensurável entre o valores medidos com os ideais.

O trabalho realizado por HOSS (2018), tinha o objetivo a instrumentação e desenvolvimento do sistema de controle de velocidade do vento de um túnel de vento com propulsão por motor a combustão interna, existente no laboratório de Conformação do IFSC Campus Chapecó.

Como resultado do trabalho, HOSS (2018) alcançou o objetivo permitindo medir variáveis de forma confiável com incertezas pequenas dentro dos requisitos estabelecidos, mesmo com emprego de sensores de baixo custo e média precisão.

3 METODOLOGIA

A metodologia desse projeto será dividida em quatro fases: sistema mecânico, sistema eletrônico, desenvolvimento do sistema de software e integração dos sistemas.

3.1 SISTEMA MECÂNICO

Nessa sessão será desenvolvido o projeto de sistema mecânico que se divide em estrutura e componentes. Para compreensão facilitada do projeto, os cálculos de resistência e equilíbrio foram desconsiderados, já que o equipamento é de pequena escala.

3.1.1 Estrutura

A estrutura ou mesa cartesiana, como é chamada neste trabalho, compõem o sistema mecânico junto com seus componentes, é a parte responsável por manter a união das peças que ela compõem e também servir de base para o posicionamento dentro da área de testes do túnel de vento. Como o túnel de vento do Laboratório de Sistemas Térmicos da Universidade Federal do Rio Grande é utilizado para atividades de pesquisa em energia renovável e fenômenos de transporte, optou-se por projetar uma estrutura móvel, que pode ser colocada e retirada de dentro do túnel, quando há a necessidade da caracterização do canal aberto. Para que essa operação se torne prática a mesa deve ser leve e resistente.

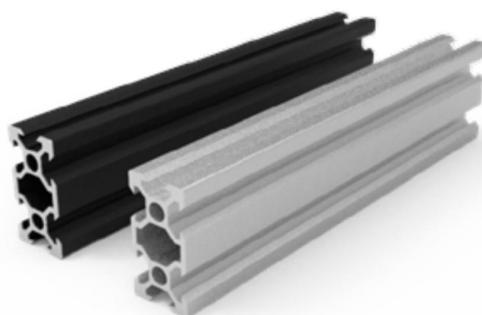


Figura 4 – Perfil v_slot 20x40 mm em alumínio.

Fonte: www.forsetisolucoes.com.br

Por ser um metal leve, durável e resistente, o alumínio se torna uma boa opção para a confecção desta estrutura. Com uma alta relação resistência/peso, confere um excepcional desempenho, além de dar um ótimo acabamento à peça. A estrutura de metal será dividida em duas partes, a base e o pórtico. A base será o componente que terá a função de dar estabilidade a

mesa e evitar que esta venha a tombar, terá um formato retangular de 500x400 mm e será feito com um perfil v_slot de 20x40 mm em alumínio, conforme apresentado na Figura 4. Para fazer a união dos perfis, que serão cortados em tamanho adequado com cantos esquadrejados em ângulo de 45 graus, para dar um melhor acabamento à peça, será utilizada uma placa de conexão interna de 90 graus, conforme Figura 6, que unirá os cantos dos perfis por meio de parafusos Allen sem cabeça M5.

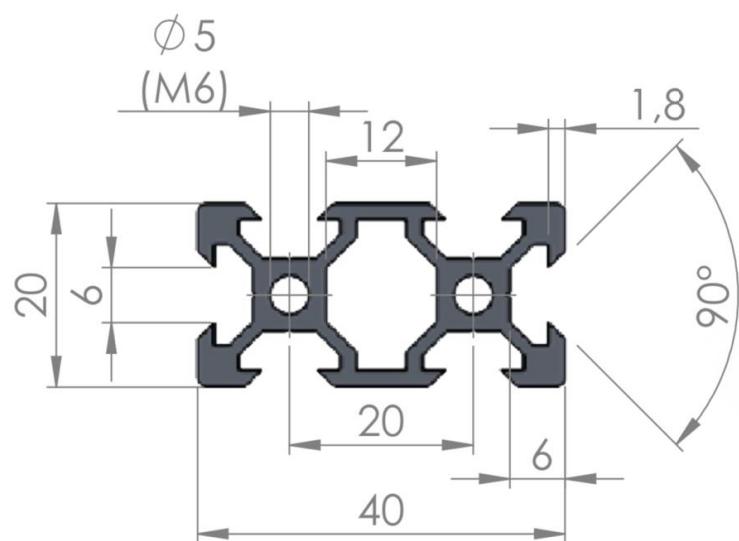


Figura 5 – Dimensões do perfil 20x40 mm.

Fonte: www.forsetisolucoes.com.br

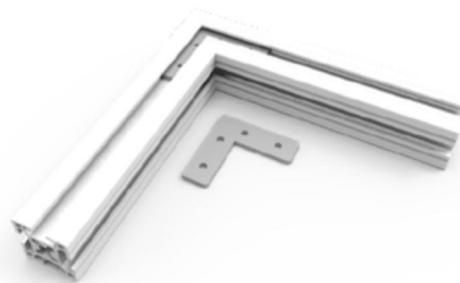
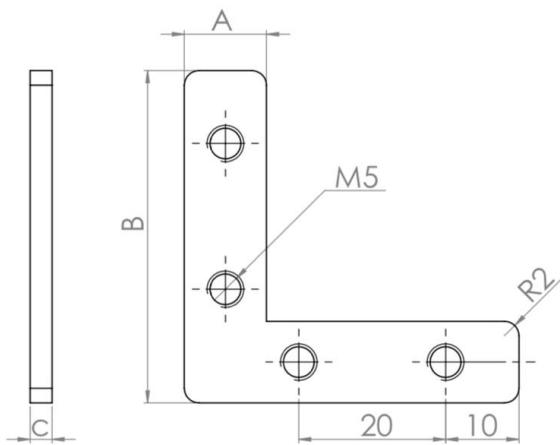


Figura 6 – Placa de conexão interna de 90°.

Fonte: www.forsetisolucoes.com.br



CÓDIGO	A	B	C	BASE
CNI20-01	9,5	44,7	3	20
CNI20-11	8	40	3	V-SLOT
CNI30-01	15	47,5	3	30
CNI45-01	13,4	37,2	9,5	45 PARAUSO

*medidas em milímetros

Figura 7 – Dimensões da placa de conexão interna de 90°.
Fonte: www.forsetisolucoes.com.br

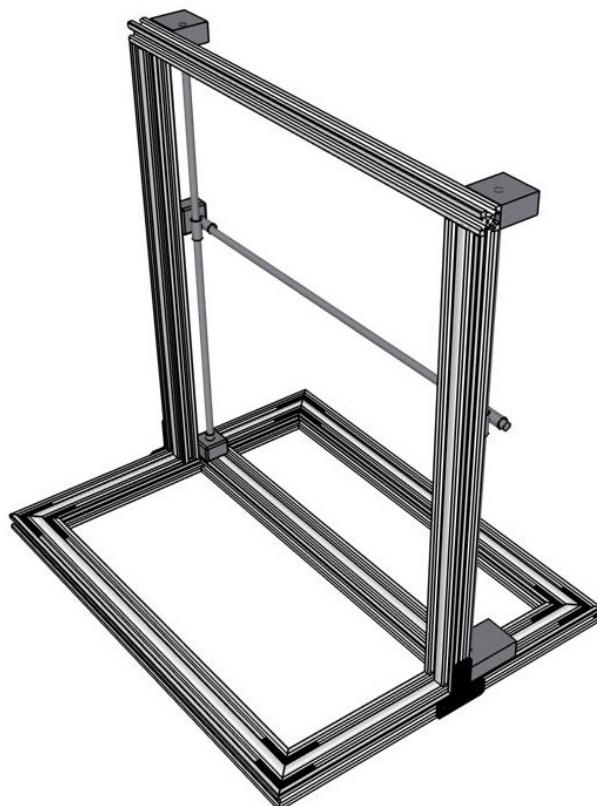


Figura 8 – Estrutura da mesa cartesiana.
Fonte: Próprio autor

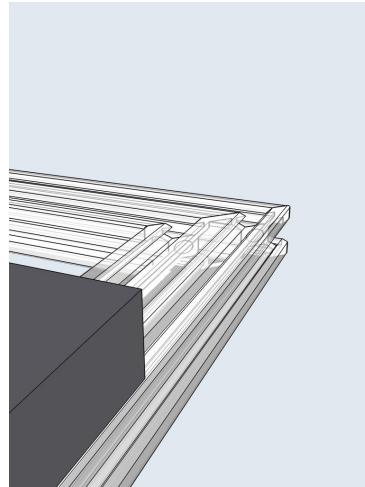


Figura 9 – Detalhe do encaixe a 45° da base da estrutura.
Fonte: Próprio autor

O pórtico será feito com o perfil v_slot de 20x20 mm em alumínio conforme apresenta a Figura 10, e terá as dimensões de 500x480 mm. Sua função é sustentar e estabilizar o sistema de transmissão. Para fixação do pórtico à base será utilizada uma placa T simples de aço. Na montagem da parte superior do pórtico a união se dará por meio de parafusos M6 diretamente nos perfis.



Figura 10 – Perfil v_slot 20x20 mm em alumínio.
Fonte: www.forsetisolucoes.com.br

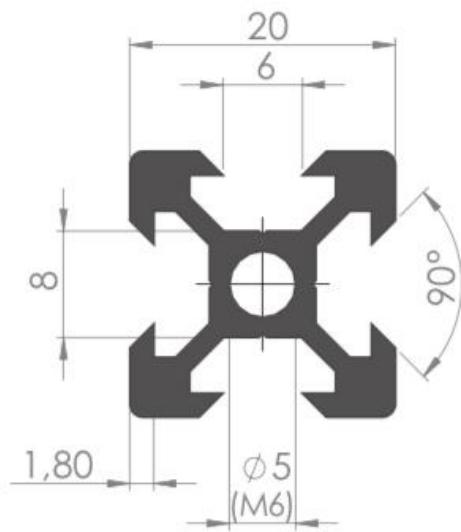


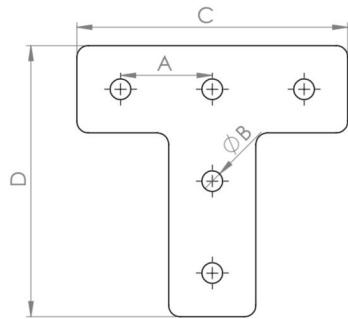
Figura 11 – Dimensões do perfil 20x20 mm.

Fonte: www.forsetisolucoes.com.br



Figura 12 – Placa T simples de aço.

Fonte: www.forsetisolucoes.com.br



CÓDIGO	A	B	C	D	MATERIAL	BASE
PA15-01	15	3,2	44	44	Alumínio	15
PLT-20	20	6,6	58	58	Aço	20
PLT-30	30	6,6	88	88	Aço	30
PLT-40	40	9	118	118	Aço	40/45

*medidas em milímetros

Figura 13 – Dimensões da placa T simples.

Fonte: www.forsetisolucoes.com.br

3.1.2 Sistema de transmissão

Segundo BUDYNAS; NISBETH (2016), o parafuso de potência é um dispositivo usado para transformar o movimento angular em movimento linear e, usualmente, para transmitir potência. Para fazer o movimento dos carros do eixo X e do eixo Y, componentes que farão o efetivo deslocamento do equipamento de medição dentro da área de teste do túnel de vento, o fuso foi escolhido como elemento de transmissão, que vai transformar o movimento giratório do motor de passo em deslocamento linear na estrutura da mesa cartesiana. Acoplados aos motores de passo, por meio de acopladores de eixo e na outra extremidade por mancal para fuso de 8 mm, os fusos farão o transporte dos carros horizontal e vertical.

Para que se tenha um deslocamento mais rápido das castanhas, elemento que estarão em contato direto com o fuso e o carro, foi selecionado o fuso trapezoidal de 8 mm de diâmetro e 8 mm de passo. Será utilizado uma guia para o deslocamento horizontal, para o deslocamento vertical será utilizado um fuso acoplado ao motor é uma guia no lado oposto, apenas para dar suporte ao elemento que fixará o fuso horizontal.

3.1.3 Acionador

Os motores elétricos são máquinas capazes de transformar energia elétrica em energia mecânica, essa energia se dá em forma de movimento angular. São equipamentos versáteis, muito eficientes e amplamente utilizados. Para fazer a movimentação dos carros, nos eixos X e

Y, optou-se por motores de passo, por que é um motor que possibilita o controle da velocidade e o posicionamento preciso, pois rotaciona em ângulos bem definidos, chamados de passos. O ponto negativo desses motores é que o controle é mais complexo, necessitando de um comando eletrônico digital para fazê-lo funcionar, por outro lado apresenta uma grande precisão no seu movimento. Os motores serão acoplados aos perfis da estrutura de modo a possibilitar o movimento dos fusos e consequentemente o equipamento de medição, que estará preso a ele por meio de uma guia com roldana.

3.2 SISTEMA ELETRÔNICO

O sistema eletrônico de uma mesa cartesiana foi dividido em módulos para um melhor detalhamento: placa de prototipagem eletrônica Arduino, drivers de potência, atuadores elétricos, fonte de alimentação, optoacopladores, encoders e um computador. É importante lembrar que para menor custo do projeto optou-se pela criação de dispositivos.

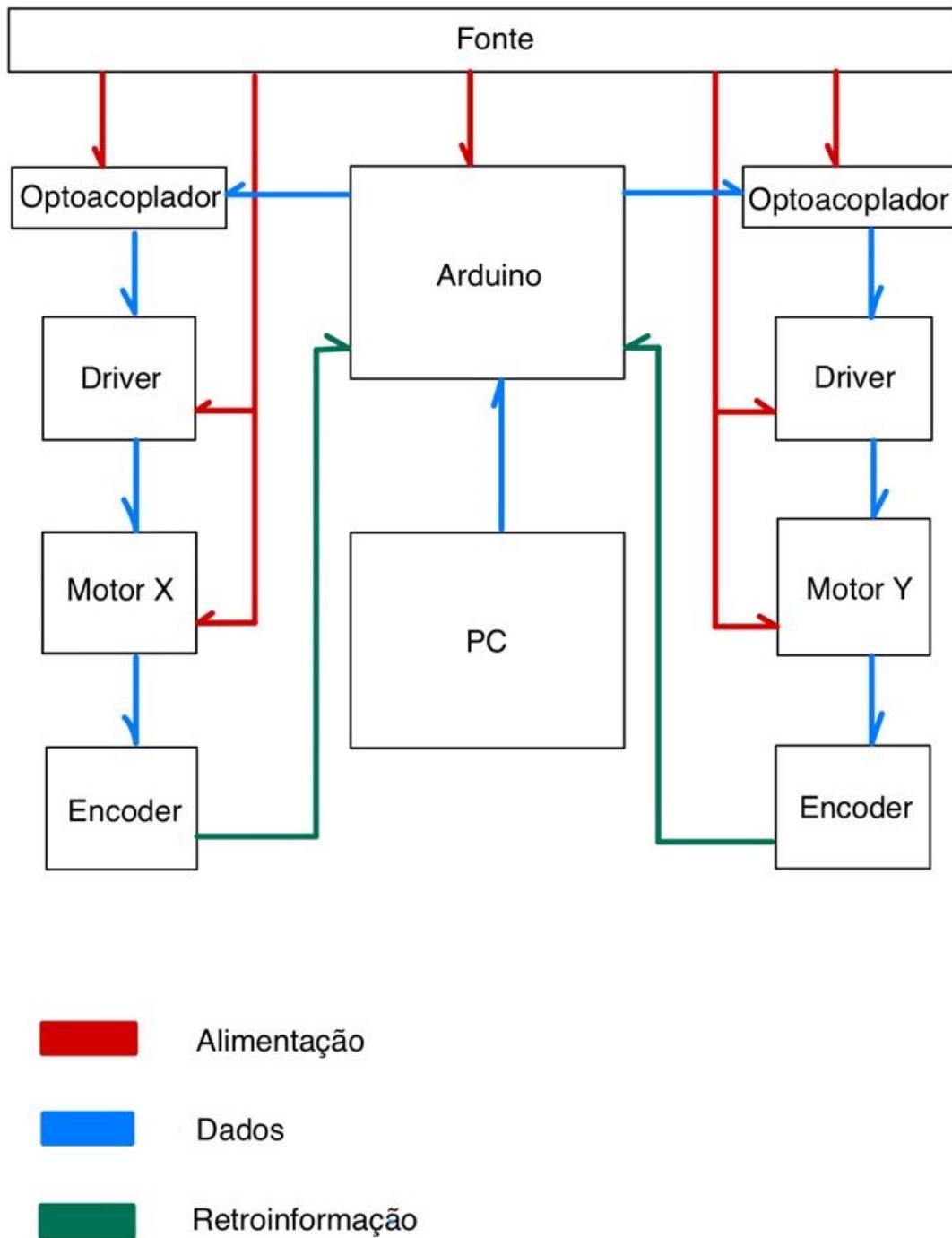


Figura 14 – Fluxograma do sistema eletrônico.
Fonte: Próprio autor.

3.2.1 Placa de prototipagem eletrônica Arduino

A placa de prototipagem eletrônica Arduino é responsável pela recepção e tratamento dos dados provenientes da interface computacional. O controle dos motores está fundamentado

na programação do microcontrolador de acordo com as necessidades definidas inicialmente para operação da mesa cartesiana.

Com o objetivo de elaborar uma interface de prototipagem de baixo custo para uso em projetos escolares, o arduino foi criado por um grupo de pesquisadores na Itália em 2005. Na sua concepção, para ter maior flexibilidade a diversos tipos de projetos, isto é, para que qualquer projetista pudesse personalizá-lo, foi adotado o conceito de hardware livre.

Em sua composição, a placa Arduino UNO contém um microcontrolador ATMEG328, que é um dispositivo de 8 bits da família AVR com arquitetura Reduced Instruction Set Computer (RISC) avançada e com encapsulamento DIP28, além de quatorze portas digitais de entrada e saída, sendo que seis delas com capacidade de Pulse Width Modulation (PWM), seis entradas analógicas e 32 KB de memória flash, 2 KB de Random Access Memory (RAM) e 1 KB de Electrically Erasable Programmable Read-Only Memory (EEPROM).

A alimentação é via porta Universal Serial Bus (USB) ou por conector tipo Jack de alimentação externa que trabalha entre os limites de 6 V e 20 V sendo recomendável o uso de 7 V a 12 V para não danificar. Para o fornecimento de tensão contínua para alimentação dos circuitos e Shields, a placa Arduino UNO tem um regulador de tensão de 3,3 V.

Além de alimentar, a porta USB é a via de comunicação com o computador para o envio do código de máquina gerado pelo compilador. O código compilado é enviado pelo microcontrolador ATMEGA16U2 que está conectado a dois LEDs chamados de TX e RX cuja função é a indicação do envio e recepção dos dados da placa para o computador.

A placa Arduino é programada via Integrated Development Environment (IDE), utilizando uma linguagem baseada em C/C++.

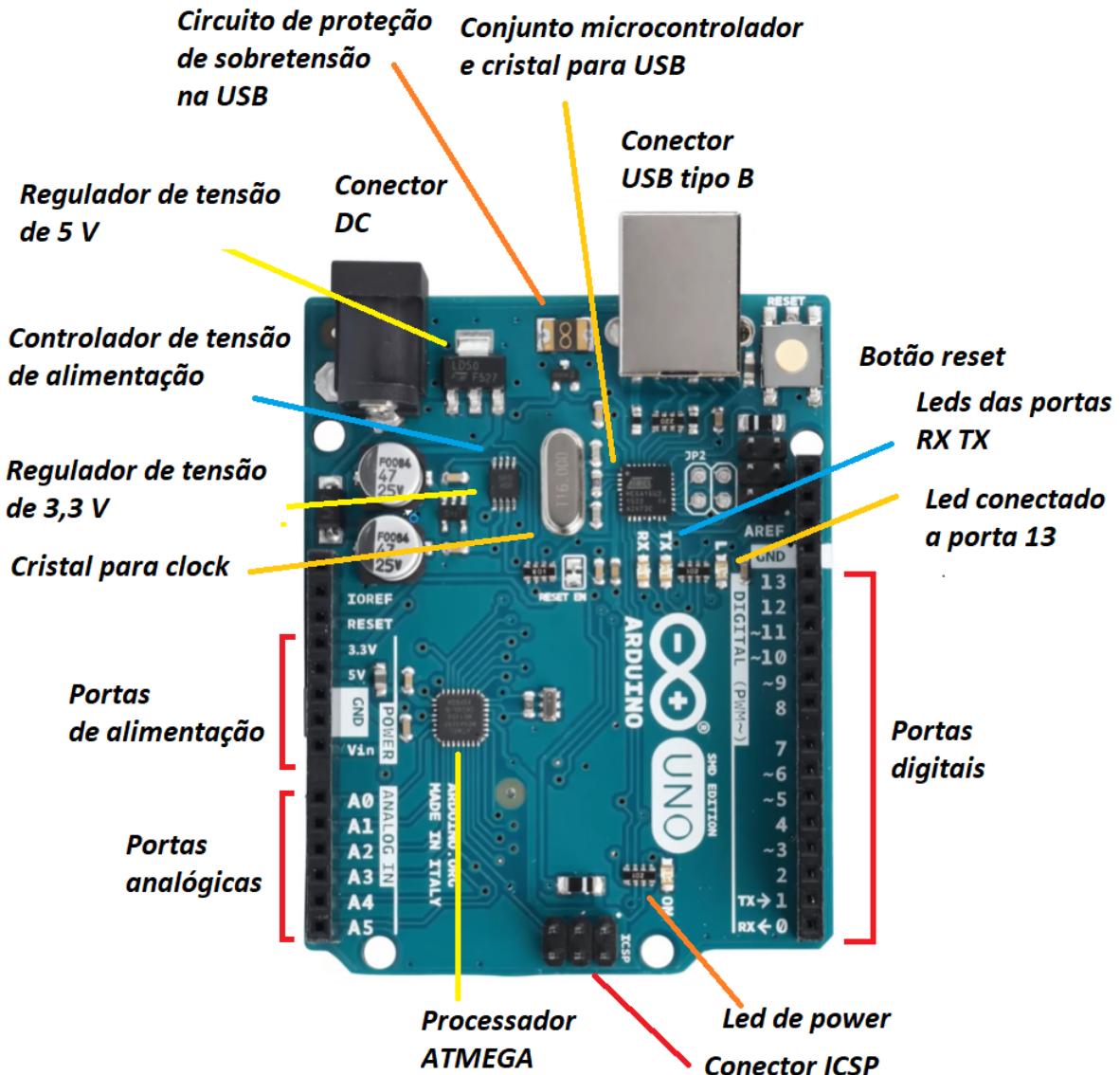


Figura 15 – Descrição dos componentes da placa Arduino.

Fonte: Próprio autor.

- **Botão reset**: o botão reset serve para reiniciar a placa arduino.
- **Conector USB tipo B**: o conector USB tipo B serve para conectar a placa Arduino ao computador.
- **Conector DC**: o conector DC serve para alimentação externa do Arduino.
- **Portas digitais**: são às portas que trabalham com os sinais digitais que são sinais com valores discretos, no arduino temos às constantes LOW que significa 0 V e também HIGH que significa 5 V, no código são 0 e 1 respectivamente. Dentro das portas digitais temos às portas PWM que são portas que modulam o sinal pela largura do pulso, no arduino podemos variar de 0 a 255. Resumidamente, a placa Arduino possui quatorze portas

digitais, sendo que seis delas são portas PWM como dito anteriormente, os pinos delas são 3, 5, 6, 9, 10 e 11.

- **Portas analógicas:** são às portas que trabalham com os sinais analógicos que são sinais com valores contínuos, no arduino podemos variar de 0 a 1023.
- **Portas de alimentação:** são às portas de saída de tensão do arduino, temos três portas: porta 3,3 V cuja saída trabalha em 3,3 V, porta 5 V cuja saída trabalha em 5 V e a porta Vin cuja saída trabalha com a tensão de entrada do arduino.
- **Led porta 13:** led conectado a porta 13 do arduino.
- **Leds das portas RX TX:** leds conectados às portas RX TX, sendo que o led TX serve para indicar a transmissão de dados e o RX para indicar a recepção de dados.
- **Processador ATMEGA328:** o processador é responsável pelo processamento da lógica de programação.
- **Led de power:** led que acende quando o arduino está ligado.
- **Conector In Circuit Serial Programming (ICSP):** o conector ICSP, que se refere a capacidade de programar Arduinos diretamente dos seus microcontroladores.
- **Conjunto microcontrolador e cristal para USB:** esse conjunto possui um microcontrolador ATMEGA16U2 e um cristal externo de 16 MHz, e é responsável pelo gerenciamento da porta USB.
- **Circuito de proteção de sobretensão na USB:** responsável por proteger a entrada USB de sobretensão.
- **Controlador de tensão de alimentação:** esse controlador é responsável por verificar se a tensão DC está presente, se não estiver, deixa que a tensão da USB alimente o circuito.
- **Regulador de tensão de 5 V:** serve para regular a tensão em 5 V.
- **Regulador de tensão de 3,3 V:** serve para regular a tensão em 3,3 V.
- **Cristal para clock:** serve para gerar o clock.

3.2.2 Drivers de potência

Os drivers de potência são dispositivos que conservam sinais fundamentais de entrada em suas saídas, potencializando e fornecendo maior corrente elétrica para equipamentos atuadores. Os drivers de potência podem ser formados por elementos eletromecânicos (relés) e semicondutores (diodos, transistores, e circuitos integrados).

No presente projeto, os drivers têm a função de, a partir dos sinais originados pelo

Arduino, atender a demanda dos motores de passo utilizados.

A construção dos drivers que controlam os motores de passo da mesa cartesiana foi baseada em um circuito eletrônico para motores que trabalham com tensão de 12 V DC e até 10 A de corrente elétrica.

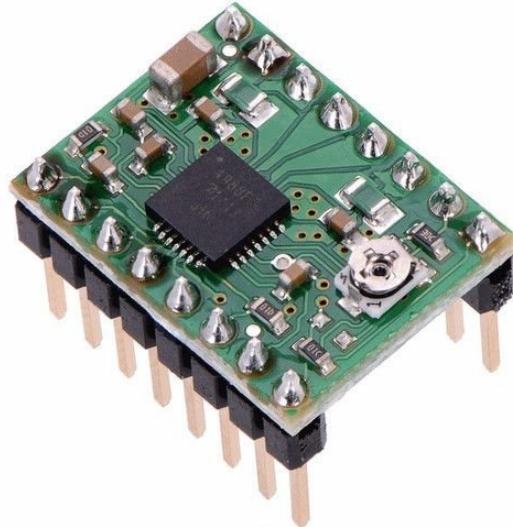


Figura 16 – Driver A4988.
Fonte: <https://www.eletrogate.com>.

Tabela 1 – Parâmetros do driver de potência.

Parâmetro	Magnitude
Tensão lógica mínima	3 V
Tensão lógica máxima	5,5 V
Corrente contínua por fase	1 A
Corrente máxima por fase	2 A
Tensão de operação mínima	8 V
Tensão de operação máxima	35 V

Fonte: Próprio Autor, 2021.

O driver foi utilizado para controlar motores de passo e pode operar com tensões entre 8 V e 35 V e entregar até 35 V por bobina. A Figura 17 mostra o driver com os respectivos componentes.

Pin-out Diagram

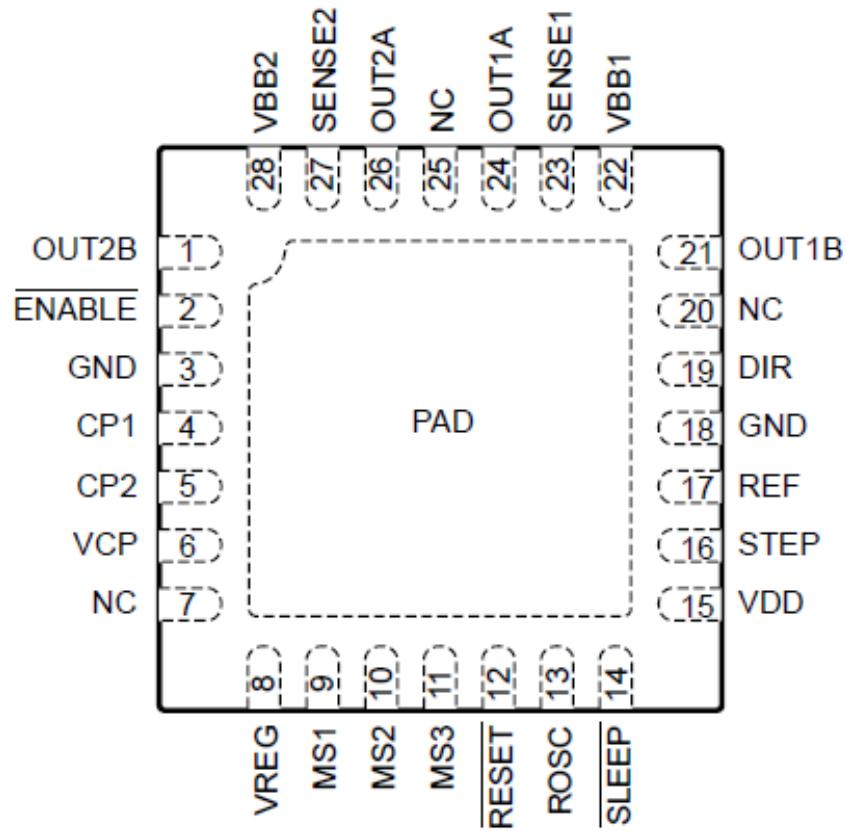


Figura 17 – Portas do driver A4988.

Fonte: www.alldatasheet.com

Tabela 2 – Lista de terminais do driver A4988.

Nome	Número	Descrição
CP1	4	Terminal do capacitor da bomba de carga
CP2	5	Terminal do capacitor da bomba de carga
VCP	6	Terminal do capacitor do reservatório
VREG	8	Terminal de desacoplamento regulador
MS1	9	Entrada lógica
MS2	10	Entrada lógica
MS3	11	Entrada lógica
RESET	12	Entrada lógica
ROSC	13	Ajuste de tempo
SLEEP	14	Entrada lógica
VDD	15	Fonte lógica
STEP	16	Entrada lógica
REF	17	Entrada de tensão de referência Gm
GND	3, 18	Terra
DIR	19	Entrada lógica
OUT1B	21	DMOS ponte completa 1 saída B
VBB1	22	Abastecimento de carga
SENSE1	23	Ponte terminal do resistor do sensor 1
OUT1A	24	DMOS ponte completa 1 saída A
OUT2A	26	DMOS ponte completa 2 saída A
SENSE2	27	Ponte terminal do resistor do sensor 2
VBB2	28	Abastecimento de carga
OUT2B	1	DMOS ponte completa 2 saída B
ENABLE	2	Entrada lógica
NC	7, 20, 25	Sem conexão
PAD	-	Almofada exposta para dissipação térmica aprimorada

Fonte: www.alldatasheet.com.

3.2.3 Atuadores

Atuadores são equipamentos ou dispositivos elétricos que convertem energia hidráulica, pneumática ou elétrica em energia mecânica. A energia gerada nos atuadores é transformada em movimento em vários tipos de processos.

Os atuadores utilizados neste projeto foram os elétricos que se dividem em motores elétricos de corrente alternada e contínua, servomotores e motores de passo. Esses atuadores convertem pulsos elétricos recebidos em seus terminais em energia mecânica transformando-a em movimento rotativo do motor, no caso do projeto os motores de passo que controlam o movimento dos fusos da mesa cartesiana. Os atuadores elétricos atendem a comandos manuais ou programáveis, localmente ou remotamente. Eles se destacam por uma transmissão de potência simplificada e eficiente do ponto de vista de energia.

Os motores de passo são atuadores eletromagnéticos com capacidade de converter pulsos elétricos digitais recebidos em movimento de rotação incremental do eixo do motor. Sua aplicação é necessária em movimentos rotativos com alta precisão que necessitam de controle

de posição e velocidade. A sua escolha foi definida pela precisão, baixo custo de aquisição e manutenção.

A composição de um motor de passo contém um rotor e um estator que é a parte fixa do gerador elétrico, nessa está situado um conjunto de bobinas responsáveis pelo giro do rotor. Para que haja o movimento, as bobinas estão ligadas aos terminais do motor, organizadas em pares, interligadas entre si e posicionadas em sentidos opostos para que quando forem energizadas, dê o movimento de rotação do motor devido às interações magnéticas.

O atributo que distingue o motor de passo dos demais motores elétricos é a capacidade de realizar passos, que são rotações discretas incrementais e precisas. Os passos são definidos por um número fixo de pólos magnéticos de dente de engrenagens do motor determinando assim, a precisão de ângulo de rotação do motor de passo. Para que haja um controle de quantos passos serão dados, o motor necessita de largura de pulso a fim de enviar a corrente adequada para cada passo. A Figura 18 é um exemplo para melhor entendimento do conceito de passo.

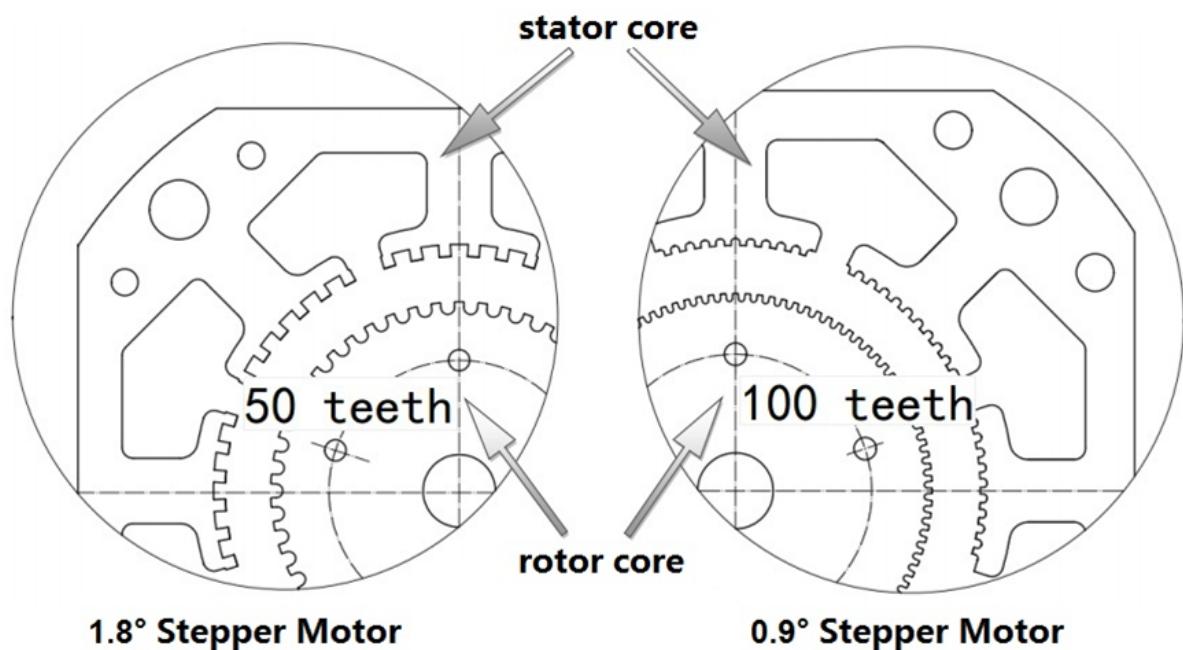


Figura 18 – Conceito didático do motor de passo.

Fonte: <https://www.fernandok.com>.

A Figura 18 mostra um motor de passo com 50 dentes de precisão, portanto combinando o número de dentes com as 4 fases de um motor bipolar temos 200 passos.

A equação 3.1 calcula o número de passos de uma volta executada pelo motor onde, o

número de passos (n_p) é igual ao número de dentes (n_d) mutiplicado pelo número de fases (n_f)

$$n_p = n_d \cdot n_f \quad (3.1)$$

A precisão do motor é definida por 360° divididos pelo número de passos. Portanto, 360 divididos por 200 é igual a $1,8^\circ$ de precisão. Já o segundo motor de passo possui 100 dentes de precisão, portanto a precisão é de $0,9^\circ$, o que indica que é um motor com maior precisão que o de 50 dentes.

Para aumentar a precisão dos motores é possível subdividir o passo de um motor em menores passos fornecendo ao mesmo tempo corrente elétrica em duas fases. Essa forma de trabalho é chamada de micropassos.

Logo, quando é fornecido a mesma magnitude de corrente a duas fases, um campo magnético de mesma magnitude é gerado, resultando em um movimento angular com a metade do passo original, essa configuração é chamada de meio passo. Assim, ao aplicar magnitudes de corrente diferentes a duas fases, o rotor se desloca proporcionalmente ao campo eletromagnético mais forte.

Os motores de passo utilizados no projeto da mesa cartesiana possuem 5 kgf.cm de torque, passo de 1.8° e corrente máxima de 2 Ampères por fase. O incremento de rotação e o torque são definidos conforme o modo de excitação que pode ser por passo completo, meio passo e micropasso explicadas anteriormente. O passo é dividido em quatro, oito ou dezesseis na maioria das vezes, mas também, atualmente se encontra sistemas capazes de dividir um passo em milhares de vezes.

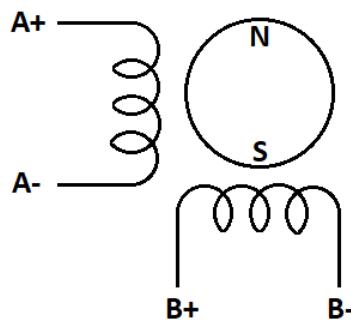


Figura 19 – Esquema elétrico do motor de passo.
Fonte: Próprio autor.

As Tabelas 3, 4 e 5 mostram a sequência de polaridades aplicadas no motor no movimento de sentido horário.

Tabela 3 – Sequência de passos com uma fase (wavestep) para movimentação no sentido horário.

Passo	A+	B+	A-	B-	Decimal
1	0	0	0	1	1
2	0	0	1	0	2
3	0	1	0	0	4
4	1	0	0	0	8

Fonte: Próprio Autor, 2021.

Tabela 4 – Sequência de passos com duas fases (fullstep) para movimentação no sentido horário.

Passo	A+	B+	A-	B-	Decimal
1	1	0	0	1	9
2	0	0	1	1	3
3	0	1	1	0	6
4	1	1	0	0	12

Fonte: Próprio Autor, 2021.

Tabela 5 – Sequência de passos com meio passo (halfstep) para movimentação no sentido horário.

Passo	A+	B+	A-	B-	Decimal
1	1	0	0	1	9
2	0	0	0	1	1
3	0	0	1	1	3
4	0	0	1	0	2
5	0	1	1	0	6
6	0	1	0	0	4
7	1	1	0	0	12
8	1	0	0	0	8

Fonte: Próprio Autor, 2021.

As Tabelas 6, 7 e 8 mostram a sequência de polaridades aplicadas no motor no movimento de sentido anti-horário.

Tabela 6 – Sequência de passos com uma fase (wavestep) para movimentação no sentido anti-horário.

Passo	A+	B+	A-	B-	Decimal
1	1	0	0	0	8
2	0	1	0	0	4
3	0	0	1	0	2
4	0	0	0	1	1

Fonte: Próprio Autor, 2021.

Tabela 7 – Sequência de passos com duas fases (fullstep) para movimentação no sentido anti-horário.

Passo	A+	B+	A-	B-	Decimal
1	1	1	0	0	12
2	0	1	1	0	6
3	0	0	1	1	3
4	1	0	0	1	9

Fonte: Próprio Autor, 2021.

Tabela 8 – Sequência de passos com meio passo (halfstep) para movimentação no sentido anti-horário.

Passo	A+	B+	A-	B-	Decimal
1	1	0	0	0	8
2	1	1	0	0	12
3	0	1	0	0	4
4	0	1	1	0	6
5	0	0	1	0	2
6	0	0	1	1	3
7	0	0	0	1	1
8	1	0	0	1	9

Fonte: Próprio Autor, 2021.

A Tabela 9 apresenta alguns parâmetros dos motores de passo.

Tabela 9 – Parâmetros dos motores de passo.

Parâmetros	Magnitude
Número de passos por revolução	200 (1,8 graus por passo)
Corrente de operação	800 mA
Tensão de alimentação	12 V
Configuração das bobinas	Bipolar (4 fios)

Fonte: Próprio Autor, 2021.

3.2.4 Fonte de alimentação

A fonte de alimentação projetada para transformar a tensão elétrica alternada em corrente contínua.

Para a escolha correta da fonte de alimentação é necessário definir a demanda de energia elétrica que os dispositivos que são alimentados pela fonte necessitam.

A Tabela 10 apresenta a demanda de energia elétrica de cada dispositivo.

Tabela 10 – Demanda de energia elétrica de cada componente do sistema.

Componente	Quantidade	Tensão	Consumo	Consumo Total
Arduino	1	7 a 12 V	800 mA	800 mA
Driver	2	8 a 35 V	2 A	4 A
Motor de passo	2	5 a 36 V	2 A	4 A
Optoacoplador	2	5 a 35 V	5 mA	10 mA
Resultado	-	12 V	-	8,18 A

Fonte: Próprio Autor, 2021.

A Tabela 10 indica que a placa controladora Arduino opera no intervalo de tensão de 7 V a 12 V, os drivers de potência no intervalo de 8 V a 35 V, os motores de passo no intervalo de 5 V a 36 V e o optoacoplador no intervalo de 5 V a 35 V. A tensão de saída de 12 V foi determinada como melhor opção para o projeto.

Outro parâmetro que a Tabela 10 apresenta é a corrente elétrica mínima para a operação do circuito. A placa controladora necessita de 800 mA, os drivers de potência necessitam de 2 A cada, como são 2 drivers, 4 A são necessários, os motores de passo consomem 2 amperes cada, como o projeto contém 2 motores, 4 Amperes são necessários e os optoacopladores necessitam de 10 mA cada, como o projeto contém 2 optoacopladores, 20 mA são necessários. Decidiu-se acrescentar uma margem de segurança adicional de 30% na corrente, então, considerou-se uma corrente mínima necessária de 10,64 A.

A equação 3.2 calcula a demanda total de corrente elétrica mínima para a operação do circuito onde: demanda total (d_{total}), demanda do arduino ($d_{arduino}$), demanda do driver de potência (d_{driver}), demanda do motor de passo (d_{motor}) e demanda do optoacoplador (d_{opto}).

$$d_{total} = 1,3 \cdot (d_{arduino} + 2 \cdot d_{driver} + 2 \cdot d_{motor} + 2 \cdot d_{opto}) \quad (3.2)$$

Conforme a determinação da tensão de saída e o cálculo de corrente necessária, é possível determinar que a fonte deve ter 12 V e 10,64 A.



Figura 20 – Fonte do sistema.

Fonte: Próprio autor.

3.2.5 Acopladores ópticos

Os acopladores ópticos ou optoacopladores são dispositivos que realizam a transferência de sinais de um circuito para outro por meio de um feixe de luz sem a ligação elétrica.

A aplicação dentro do projeto é o isolamento elétrico que pode ser estabelecido entre os circuitos de controle de potência, protegendo os circuitos sensíveis a uma alta tensão como a placa controladora Arduino.

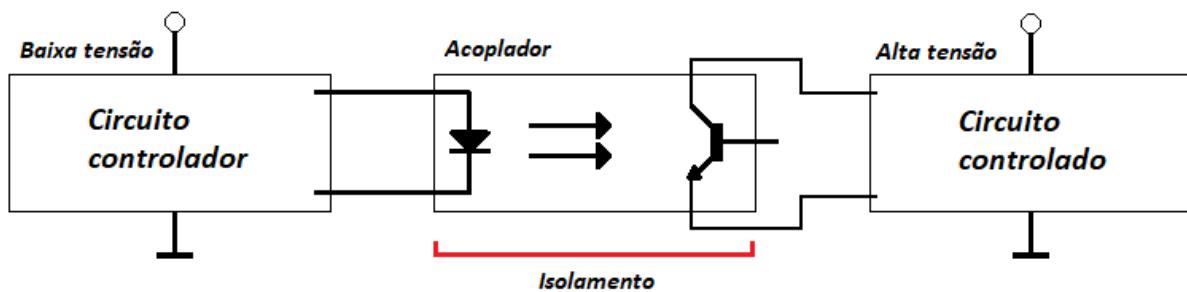


Figura 21 – Funcionamento do acoplador no sistema.

Fonte: Próprio autor.

A sua composição contém uma fonte emissora de luz (LED) e um sensor fototransistor

sensível às variações espectrais da fonte emissora. Seu funcionamento é baseado no efeito fotoelétrico, onde o diodo LED produz um feixe de luz infravermelha polarizando a base do fototransistor impondo uma condução entre base e emissor.

O optoacoplador escolhido para o projeto foi o PC 817 por já estar disponível no Laboratório de Sistemas Térmicos.

3.2.6 Encoders

Geradores de pulsos ou encoders são sensores/transdutores eletromecânicos responsáveis pelo sistema de controle de posição transformando a medida de posição de algum objeto, seja linear ou angular, em sinal elétrico digital que é transmitida à placa controladora, também conseguem converter movimentos circulares ou lineares em pulso elétricos.

Um encoder tem a capacidade de quantização de distâncias, controle de velocidades, medição de ângulos, medição de posição, medição de deslocamento relativo e etc. Sua composição contém um disco com marcações, um emissor e um receptor. Conforme o disco gira, vão sendo contadas às marcações e o emissor envia um sinal à placa controladora que por sua vez executa cálculos de distâncias, velocidades, ângulos, número de rotações e etc.

O princípio de funcionamento é dividido em três tipos, sendo eles: tacômetro, incremental e absoluto.

O encoder do tipo tacômetro possui um sinal de saída digital responsável em emitir um pulso para cada incremento captado no deslocamento. Este é utilizado na medição de velocidade e também no deslocamento angular unidirecional. O encoder incremental conta com sistema eletrônico externo responsável pela interpretação de posição, este dispositivo utiliza dois ou mais elementos geradores de sinal possuindo a capacidade de rotacionar por quantas revoluções forem necessárias.

E o encoder absoluto utiliza várias faixas de saídas com leitura em paralelo e são limitados a uma revolução. Os dados podem ser recuperados, se uma falha no sistema ocorrer, devido a representação binária da posição angular do eixo.

3.2.7 Chaves fim de curso

As chaves fim de curso são dispositivos eletromecânicos usados para limitação de campo de movimento de eixos, como os presentes na mesa cartesiana. Esses componentes têm a capacidade de mudança de estado de conexão em circuitos, alternando o estado de aberto para

fechado e vice-versa (ALCIATORE; HISTAND, 2014). Seu estado inicialmente pode ser tanto normalmente aberto como normalmente fechado alterando seu estado por pino, gatilho, roldana, haste alavanca, etc.

Uma chave fim de curso é composta basicamente por três elementos, sendo eles:

- a) **Caixa:** Pode ser metálica ou plástica, dependendo do tipo e abriga os contatos e o atuador.
- b) **Contato:** É usado dentro do circuito a fim de fazer com que a atuação da chave fim de curso interrompa ou acione algum outro dispositivo.
- c) **Atuador:** Recebe a força externa exercida para o acionamento da troca de estado.

3.3 SISTEMA DE SOFTWARE

A seguir será descrito o desenvolvimento do sistema de software.

3.3.1 Plataforma de prototipação Arduino IDE

A plataforma de prototipação Arduino IDE é um software que permite o desenvolvimento e envio de códigos compilados direto para o microcontrolador. Essa plataforma tem a flexibilidade de ser utilizada em vários sistemas operacionais e foi desenvolvida na linguagem Java oferecendo suporte de desenvolvimento na linguagem C e C++. O download da plataforma foi realizado através do link: <https://www.arduino.cc/en/software>

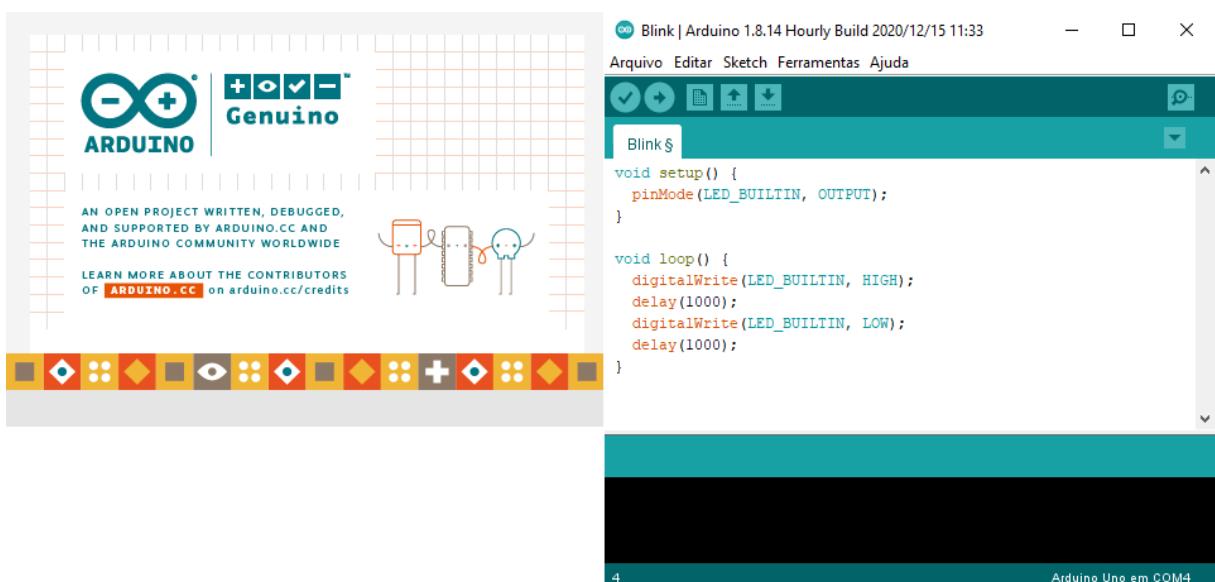


Figura 22 – Ambiente de desenvolvimento integrado Arduino.

Fonte: Próprio autor.

3.3.2 Logica de programação

O desenvolvimento da lógica de programação da placa controladora foi organizado de maneira modular usando orientação a objetos para uma manutenção facilitada e um entendimento mais claro do código.

As operações que o software deve executar foram apresentadas no fluxograma abaixo.

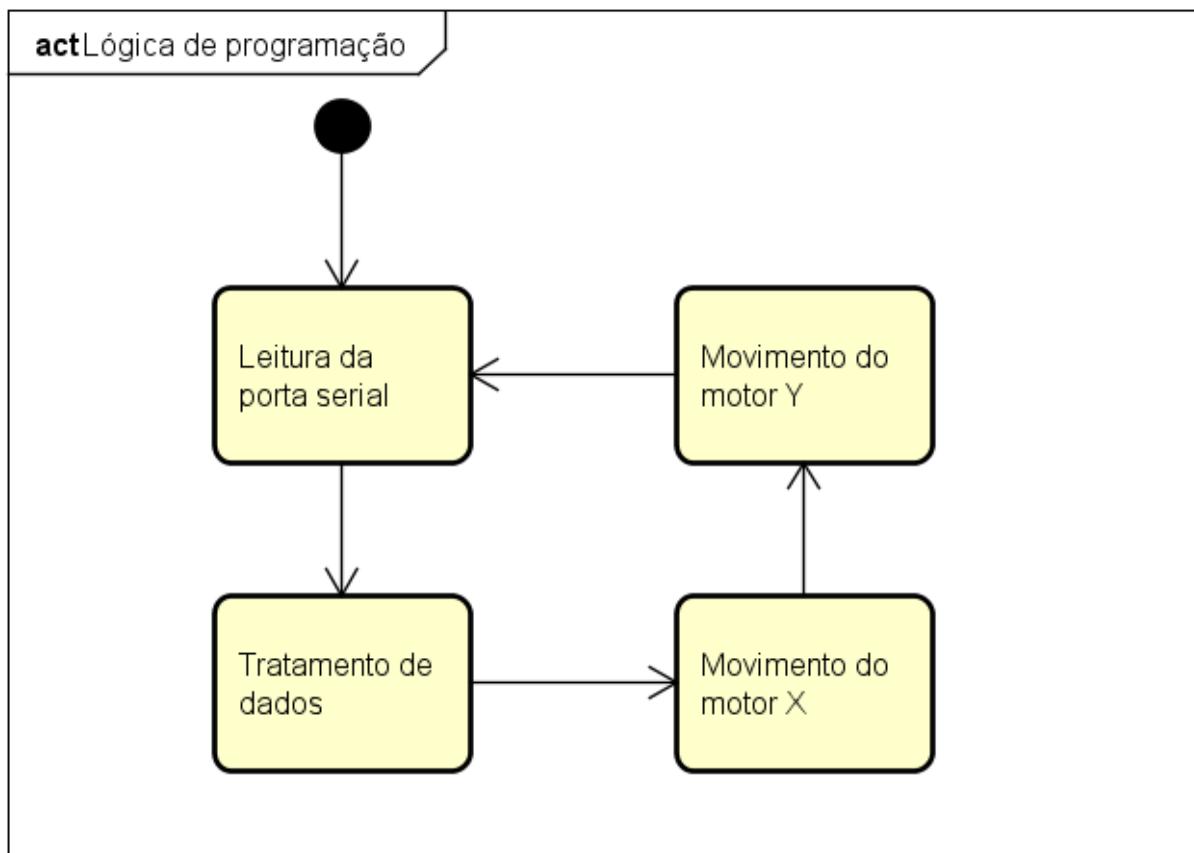


Figura 23 – Fluxo de execução do software.

Fonte: Próprio autor.

A Figura 23 apresenta um fluxograma simplificado da lógica de programação presente na placa controladora Arduino que inicialmente tem como primeiro comando ficar esperando o envio de dados pela porta Serial, se caso há esse envio, então o programa faz a leitura da porta serial, armazenando os dados em variáveis para em seguida realizar o tratamento de dados. Após o tratamento de dados o software dará o comando de movimentação do motor que controla o fuso do eixo horizontal (eixo X), em seguida, assim que o motor do eixo X parar sua rotação, o comando de movimentação do motor do eixo vertical (eixo Y) será acionado. Finalmente, após o fim da execução do motor que controla o eixo Y, o software volta a analisar se é enviado dados pela porta serial.

3.3.3 Diagrama de classes

Diagrama de classe é uma representação da estrutura e relações entre classes que um software possui facilitando e servindo de modelo para criação de objetos. Esse diagrama permite modelar classes com seus atributos e métodos além da relação entre objetos.

Para o desenvolvimento do software da placa controladora cuja linguagem é C++ que é fundamentada em orientação a objetos, foi definido que seria a melhor opção realizar um diagrama de classes antes do desenvolvimento do código. Sendo assim a Figura 24 apresentada abaixo é o resultado do que foi modelado.

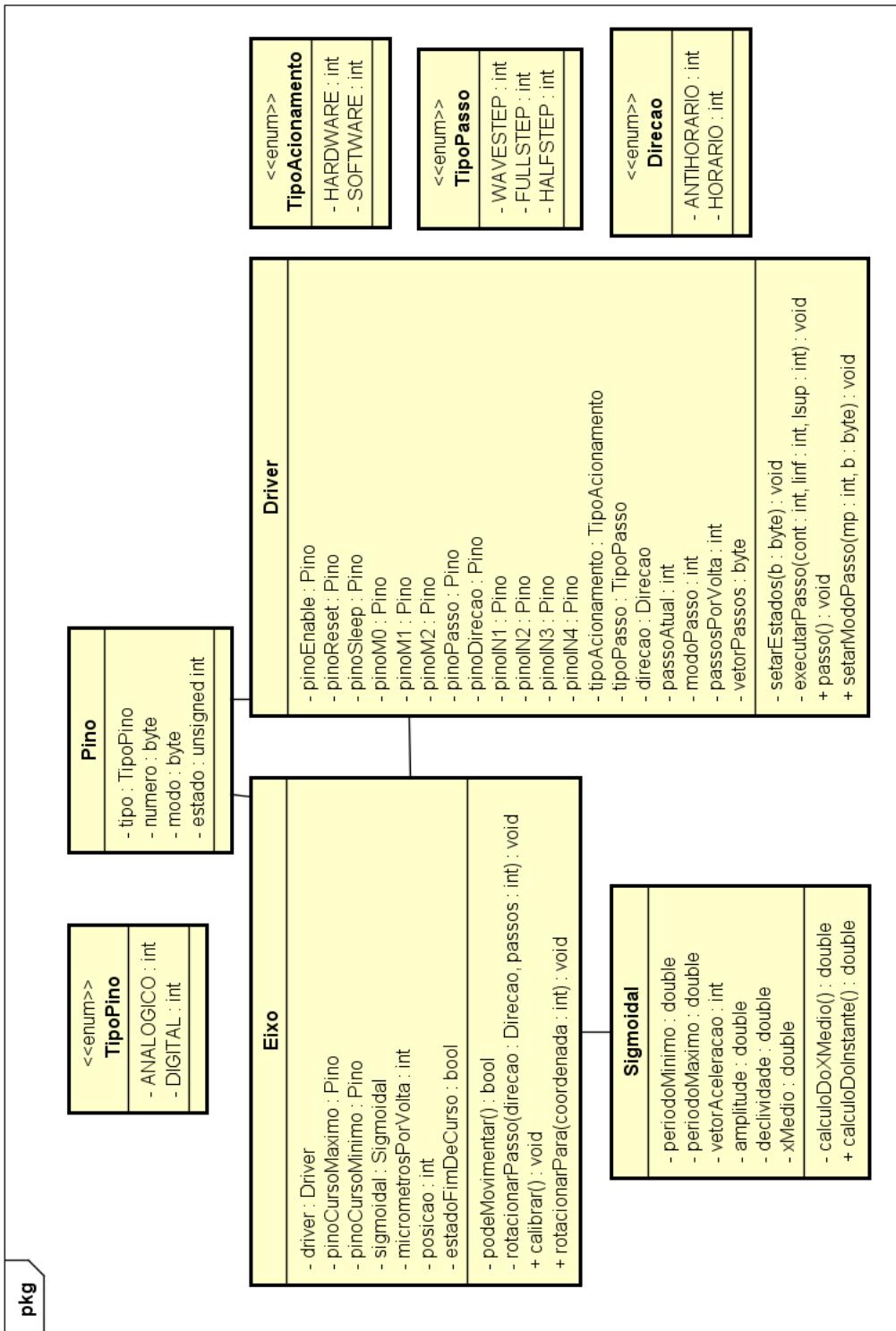


Figura 24 – Diagrama de classes do sistema de software presente no Arduino.

Fonte: Próprio autor.

A seguir será explicado este diagrama classe a classe.

“Sigmoidal” é a classe responsável pela aceleração sigmoidal nos motores de passo, essa classe utiliza a função sigmoidal que é uma função matemática com o gráfico parecido com a letra S. Para o desenvolvimento da mesa cartesiana houve a preocupação dos projetistas em configurar os motores de passo com uma aceleração variável para maior vida útil dos equipamentos, já que com aceleração constante pode causar forças desnecessárias durante o processo de rotação.

A Tabela 11 apresenta a declaração e funcionalidade dos atributos e métodos da classe “Sigmoidal”.

Tabela 11 – Declaração e funcionalidade dos atributos e métodos da classe Sigmoidal.

Declaração	Funcionalidade
- periodoMinimo: double	Período máximo do passo do motor.
- periodoMaximo: double	Período mínimo do passo do motor.
- vetorAceleracao: int	Determina se a curva será de aceleração (1) ou desaceleração (-1)
- amplitude: double	Amplitude da aceleração, diferença do período máximo menos o período mínimo
- declividade: double	Inclinação da curva de aceleração, o quanto rápido o motor irá acelerar
- xMedio: double	É a metade do número de iterações necessários para percorrer a curva sigmoidal
- calculoDoXMedio(): double	Calcula o número de iterações necessários para percorrer metade da curva sigmoidal.
+ calculoDoInstante(): double	Calcula um instante na curva tendo como entrada uma iteração

Fonte: Próprio Autor, 2021.

A classe “Sigmoidal” tem como atributos privados: “periodoMaximo” e o “periodoMinimo” que são os períodos que o motor sofrerá os pulsos nas bobinas, “vetorAceleracao” que determina se a curva sigmoidal terá um comportamento de aceleração ou desaceleração, “amplitude” que é a diferença entre o “periodoMaximo” e o “periodoMinimo” e é utilizada na equação do cálculo do X médio e do cálculo do instante, “declividade” que determina a inclinação da curva de aceleração (o quanto rápido o motor irá acelerar), “xMedio” que é a metade do número de passos (iterações) que o motor precisará para acelerar e é utilizado no cálculo do instante.

Como operações, a classe “Sigmoidal” tem os métodos de acesso getters e setters que acessam os atributos privados citados acima, esses métodos são necessários para cumprir o princípio de encapsulamento da orientação a objetos protegendo a lógica da classe. A classe “Sigmoidal” dispõe de duas operações, as quais são definidas por: “calculoDoXMedio”, que calcula o valor do atributo xMedio, “calculoDoInstante”, que calcula o período a ser aplicado em um determinado instante, esse cálculo é utilizado na lógica do método “rotacionarPasso” da

classe “Eixo”.

“Pino” é a classe que representa os pinos da placa Arduino, pois devido ao número alto de vezes que foi necessário a utilização dos pinos, foi definido que seria melhor a criação de uma classe para facilitar as operações com pinos.

A Tabela 12 apresenta a declaração e funcionalidade dos atributos e métodos da classe “Pino”.

Tabela 12 – Declaração e funcionalidade dos atributos e métodos da classe Pino.

Declaração	Funcionalidade
- tipo: TipoPino	Tipo do pino (ANALOGICO, DIGITAL).
- numero: byte	Número do pino.
- modo: byte	Modo do pino (OUTPUT, INPUT).
- estado: unsigned int	Estado do pino (LOW, HIGH) (0, 255) (0, 1023).

Fonte: Próprio Autor, 2021.

A classe “Pino” possui como atributos privados: “tipo”, que é o tipo de entrada do pino que pode ser analogico ou digital, “numero”, que é o número de uma determinada porta na placa, “modo” que define se a porta é de entrada ou saída de dados, e “estado”, que pode estar ligado ou desligado nas portas digitais, ter valores de 0 a 255 nas portas PWM e de 0 a 1023 nas portas analogicas. Como operações, a classe “Pino” tem os métodos de acesso getters e setters que acessam os atributos privados citados acima.

“Driver” é a classe responsável pelo controle digital do driver de potência, com ela é possível definir o modo do passo dos motores, executar o pulso que dará movimento ao motor de passo, ligar e desligar o driver, deixá-lo no modo “sleep” e também “reseta-lo”.

A Tabela 13 apresenta a declaração e funcionalidade dos atributos e métodos da classe “Driver”

Tabela 13 – Declaração e funcionalidade dos atributos e métodos da classe Driver.

Declaração	Funcionalidade
- pinoEnable: Pino	Ativar e desativar o driver.
- pinoReset: Pino	Resetar o driver.
- pinoSleep: Pino	Ativar e desativar o modo sleep do driver.
- pinoM0: Pino	Pino M0 do modo de passo do driver.
- pinoM1: Pino	Pino M1 do modo de passo do driver.
- pinoM2: Pino	Pino M2 do modo de passo do driver.
- pinoPasso: Pino	Pino de execução do passo do driver.
- pinoDirecao: Pino	Pino de configuração da direção do driver.
- pinoIN1: Pino	Pino IN1 do motor de passo.
- pinoIN2: Pino	Pino IN2 do motor de passo.
- pinoIN3: Pino	Pino IN3 do motor de passo.
- pinoIN4: Pino	Pino IN4 do motor de passo.
- tipoAcionamento: TipoAcionamento	Tipo de acionamento do driver: (SOFTWARE, HARDWARE).
- tipoPasso: TipoPasso	Tipo do passo do driver: (WAVESTEP, FULLSTEP, HALFSTEP).
- direcao: Direcao	Configuração da direção do driver: (ANTIHORARIO, HORARIO).
- passoAtual: int	PassoAtual é o índice de acesso as informações do vetor de passos.
- modoPasso: int	Pino do modo de passo do driver.
- passosPorVolta: int	Quantidade de passos a cada volta do motor de passo.
- vetorPasso: byte	Vetor de bytes de sequência dos passos.
- setarEstadoModo(pM0:bool, pM1:bool, pM2:bool): void	Definir o modo de passo do driver passando como parâmetros o estado dos pinos M0, M1 e M2.
- setarEstados(b: byte):void	Definir estados dos pinos IN1, IN2, IN3, IN4 do motor de passo.
- passo():void	Executar um passo do motor.
+ executarPasso(cont:int, linf:int, lsup:int):void	Executar um passo do motor com a lógica de sequência dos passos.
+ setarModoPasso(mp:int, b:byte):void	Definir o modo de passo do driver.

Fonte: Próprio Autor, 2021.

A classe “Driver” possui os atributos privados: “pinoEnable”, que permite ligar ou desligar o driver, “pinoReset”, que reinicia o driver, “pinoSleep”, que ativa o modo sleep do driver, “pinoM0”, “pinoM1” e “pinoM2”, que definem o modo do Passo, “pinoPasso” e “pinoDirecao” que são os pinos de execução de passo e configuração de direção para tipo de acionamento via hardware, “pinoIN1”, “pinoIN2”, “pinoIN3” e “pinoIN4” que são os pinos referentes ao controle das bobinas do motor de passo para tipo de acionamento via software, “tipoAcionamento” para configuração do tipo de acionamento do driver que pode ser via software ou hardware conforme a necessidade de cada tipo de driver utilizado, “tipoPasso” para configuração do tipo de passo do driver que pode ser (wavestep, fullstep, halfstep), “direcao”, que define o sentido do movimento, “passoAtual” que é o índice de acesso às informações do vetor de passos, “modoPasso”, que define o modo do passo do driver, “passosPorVolta”, que define a quantidade de passos a cada

volta do motor como por exemplo, 200 passos de $1,8^\circ$ resultando em 360° .

Como operações, a classe “Driver” tem os métodos: métodos de acesso getters e setters que acessam os atributos privados da classe, os métodos: “setarEstados”, que define os estados dos pinos IN1, IN2, IN3, IN4 do motor de passo, “passo” que executa um passo do motor, “setarModoPasso”, que define o modo de passo do driver e “executarPasso”, executa um passo do motor com a lógica de configurações de passos.

“Eixo” é a classe que foi desenvolvida a lógica de movimentação dos eixos da mesa cartesiana. Com ela é possível rotacionar definindo a coordenada cartesiana de preferência, verificar se a posição do eixo está ativando a chave de fim de curso.

A Tabela 14 apresenta a declaração e funcionalidade dos atributos e métodos da classe “Eixo”.

Tabela 14 – Declaração e funcionalidade dos atributos e métodos da classe Eixo.

Declaração	Funcionalidade
- driver: Driver	Define as configurações do driver que controla os motores de passo.
- sigmoidal: Sigmoidal	Define a aceleração sigmoidal dos motores de passo.
- pinoCursoMinimo: Pino	Pino do curso mínimo da chave fim de curso.
- pinoCursoMaximo: Pino	Pino do curso máximo da chave fim de curso.
- posicao: int	Pino da posição atual do eixo.
- estadoFimDeCurso: bool	Pino do estado de fim de curso que pode inicializar LOW ou HIGH.
- podeMovimentar(direcao:bool):bool	Verificar se o motor está no fim de curso.
- rotacionarPassos(direcao:bool,passos:int):void	Rotacionar o motor, tendo como parâmetros de entrada a direção e a quantidade de passos.
+ rotacionarPara(coordenada:double):void	Rotacionar o motor tendo como parâmetro a coordenada cartesiana do eixo.

Fonte: Próprio Autor, 2021.

A classe “Eixo” possui os atributos privados: “driver”, que é responsável pelas configurações e operações do driver de potência que controla o motor de passo acoplado ao eixo, “sigmoidal” que define a aceleração variável do motor, “pinoCursoMinimo” e “pinoCursoMaximo”, que são os pinos referentes às chaves fim de curso de cada eixo, “estadoFimDeCurso”, que indica o estado que a chave fim de curso inicializará estando desativada.

Como operações, a classe tem os métodos: métodos de acesso getters e setters que acessam os atributos privados da classe, os métodos: “podeMovimentar”, que indica se o motor de passo pode executar o próximo passo ou se já chegou ao fim do curso, “rotacionarPassos”,

que rotaciona o eixo tendo como parâmetros de entrada uma direção e uma quantidade de passos a serem executadas e “rotacionarPara”, que rotaciona o eixo tendo como parâmetros de entrada uma coordenada cartesiana.

Além das classes utilizadas, o software possui a sketch principal que é responsável pela configuração das informações iniciais, criação dos objetos na memória, configuração dos atributos de cada objeto, execução da leitura pela porta serial, tratamento de dados recebidos e a movimentação dos motores. A Figura 25 apresenta um diagrama geral da organização do software.

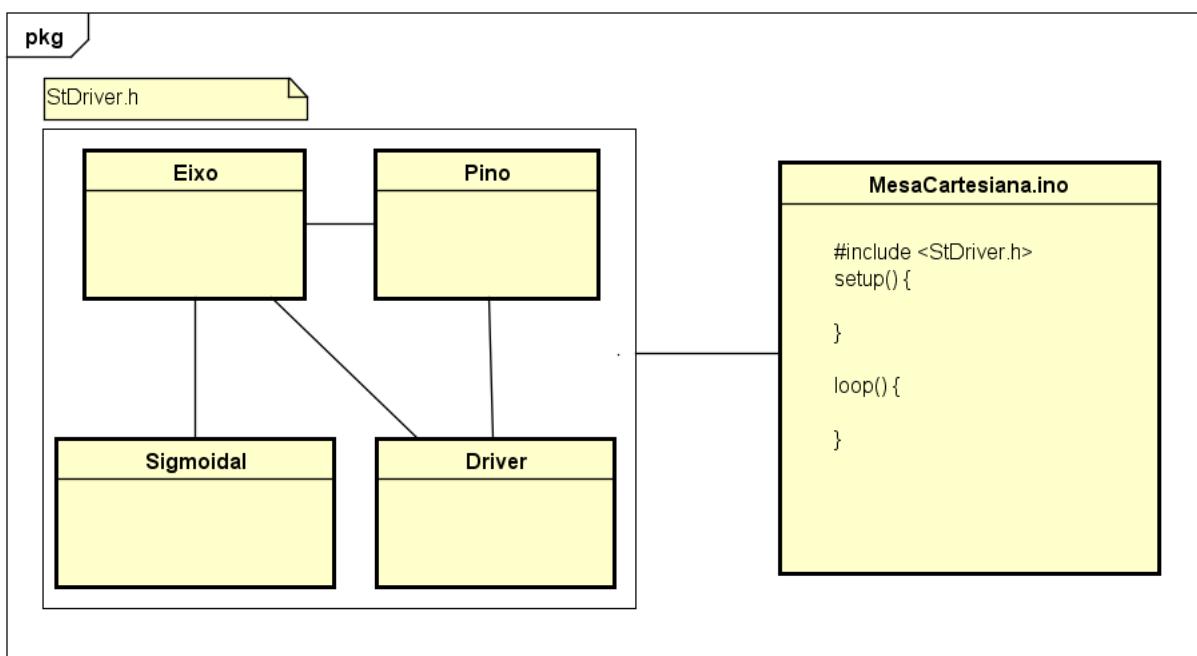


Figura 25 – Diagrama da organização geral do software.

Fonte: Próprio autor.

O arquivo “MesaCartesiana.ino” é o principal, ele instancia os objetos das classes: “Pino”, “Sigmoidal”, “Driver” e “Eixo” e também tem a responsabilidade de receber às coordenadas através da comunicação serial e às enviar para o objeto da classe “Eixo” que fará a rotação dos motores.

3.4 INTEGRAÇÃO DOS SISTEMAS

Nesta seção será descrito como os sistemas são integrados mostrando como os três sistemas se comunicam. A Figura 26 é um fluxograma que descreve de maneira gráfica a integração dos sistemas.

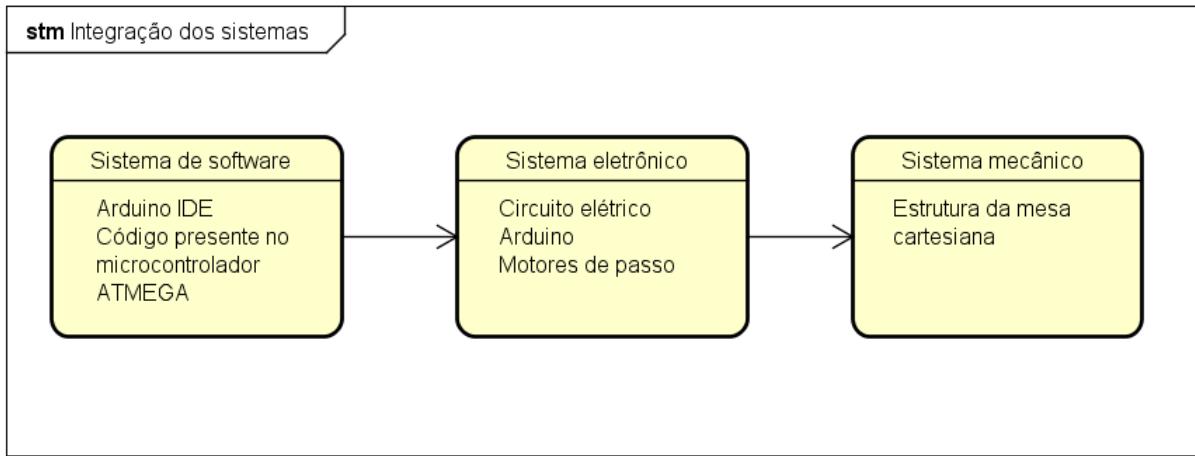


Figura 26 – Fluxograma para apresentar a integração sistemas.

Fonte: Próprio autor.

O sistema de software se comunica com o sistema eletrônico através do envio de dados pela comunicação Serial presente no Arduino IDE para o software presente no microcontrolador ATMEGA da placa Arduino. Por sua vez, o microcontrolador envia um comando elétrico ao sistema eletrônico que acionará os motores de passo que convertem a energia elétrica em mecânica transmitindo o movimento dos fusos da estrutura do sistema mecânico.

4 RESULTADOS E DISCUSSÃO

Para apresentação dos resultados encontrados ao longo da implantação da mesa cartesiana no laboratório de sistemas térmicos e uma posterior discussão, os temas foram divididos de acordo com a metodologia (seção 3). Ou seja, primeiramente é apresentado os resultados do sistema mecânico, posteriormente os resultados do sistema eletrônico, em seguida é mostrado os resultados da programação do sistema de software. Por fim, é realizada uma análise dos testes experimentais.

4.1 SISTEMA MECÂNICO

A Figura X mostra uma fotografia da mesa cartesiana possibilitando uma melhor compreensão do funcionamento dos seus componentes.

4.2 SISTEMA ELETRÔNICO

A Figura X mostra um esquema elétrico que comanda os motores de passos presentes na estrutura da mesa cartesiana possibilitando uma melhor compreensão do funcionamento dos seus componentes.

4.3 SISTEMA DE SOFTWARE

O sistema de software do projeto foi fundamentado na metodologia, sendo assim para o desenvolvimento do código foi utilizado a plataforma de prototipação Arduino IDE respeitando o diagrama de classes mostrado na seção 3.3.3.

A seguir serão mostrados os arquivos header de cada classe presente no sistema a fim de resumir o código fonte da sua versão completa. Esta versão se encontra nos apêndices do trabalho.

4.3.1 Código header da classe Pino

Segue abaixo o arquivo header (Pino.h) da classe “Pino”.

```

1 #ifndef PINO_H_INCLUDED
2 #define PINO_H_INCLUDED
3 class Pino {

```

```

4 public :
5 enum TipoPino {
6 ANALOGICO = 0,
7 DIGITAL = 1,
8 };
9 ~Pino (void) {}
10 Pino (TipoPino t, byte n, byte m, unsigned int e);
11 void setModo (byte m);
12 void setEstado (unsigned int e);
13 TipoPino getTipo ();
14 byte getNumero ();
15 byte getModo ();
16 unsigned int getEstado ();
17 private :
18 TipoPino tipo = TipoPino :: DIGITAL;
19 byte numero = 0;
20 byte modo = OUTPUT;
21 unsigned int estado = LOW;
22 void setTipo (TipoPino t);
23 void setNumero (byte n);
24 };
25 #endif

```

Assim para se utilizar as funções presentes na classe “Pino” é criado um objeto através do comando abaixo com seus devidos parâmetros de entrada.

```

1 Pino *pinoEnableX;
2 pinoEnableX = new Pino (Pino :: DIGITAL, PINO_ENABLE_X, OUTPUT, LOW);

```

4.3.2 Código header da classe Sigmoidal

Segue abaixo o arquivo header (Sigmoidal.h) da classe “Sigmoidal”.

```

1 #ifndef SIGMOIDAL_H_INCLUDED
2 #define SIGMOIDAL_H_INCLUDED
3 #define E 2.71828182845904523536
4 class Sigmoidal {
5 public :
6 ~Sigmoidal (void) {}
7 Sigmoidal (double pmax, double pmin, double de);

```

```

8 double calculoDoXMedio();
9 void setVetorAceleracao();
10 void setPeriodoMaximo(double pmax);
11 void setPeriodoMinimo(double pmin);
12 void setDeclividade(double de);
13 double getPeriodoMaximo();
14 double getPeriodoMinimo();
15 int getVetorAceleracao();
16 double getAmplitude();
17 double getDeclividade();
18 double getXMedio();
19 double calculoDoInstante();
20 private:
21 double periodoMaximo = 600, periodoMinimo = 100,
22 amplitude = this->periodoMaximo - this->periodoMinimo,
23 declividade = 0, xMedio = 0;
24 int vetorAceleracao = 1;
25 };
26 #endif

```

Assim para se utilizar as funções presentes na classe “Sigmoidal” é criado um objeto através do comando abaixo com seus devidos parâmetros de entrada.

```

1 Sigmoidal sigmoidalX;
2 sigmoidalX = new Sigmoidal(SIG_PERIODO_MAXIMO_X, SIG_PERIODO_MINIMO_X,
   SIG_DECLIVIDADE_X);

```

4.3.3 Código header da classe Driver

Segue abaixo o arquivo header (Driver.h) da classe “Driver”.

```

1 #ifndef DRIVER_H_INCLUDED
2 #define DRIVER_H_INCLUDED
3 class Driver {
4 public:
5 enum TipoAcionamento {
6 SOFTWARE = 0,
7 HARDWARE = 1
8 };
9 enum TipoPasso {

```

```

10 WAVESTEP = 0,
11 FULLSTEP = 1,
12 HALFSTEP = 2
13 };
14 enum Direcao {
15 ANTIHORARIO = -1,
16 HORARIO = 1
17 };
18 ~Driver(void) {}
19 Driver(Pino* pinoIN1, Pino* pinoIN2, Pino* pinoIN3, Pino* pinoIN4, byte*
      vetorPassos);
20 Driver(Pino *pinoEnable, Pino *pinoReset, Pino *pinoSleep, Pino *pinoM0,
      Pino *pinoM1, Pino *pinoM2, Pino *pinoPasso, Pino *pinoDirecao);
21 Driver(TipoPasso tipoPasso, Pino* pinoEnable, Pino* pinoReset, Pino*
      pinoSleep, Pino* pinoM0, Pino* pinoM1, Pino* pinoM2, Pino* pinoIN1, Pino
      * pinoIN2, Pino* pinoIN3, Pino* pinoIN4, byte *vetorPassos);
22 void setDirecao(Direcao d);
23 Pino* getPinoEnable();
24 Pino* getPinoReset();
25 Pino* getPinoSleep();
26 Pino* getPinoPasso();
27 Pino* getPinoDirecao();
28 Direcao getDirecao();
29 int getModoPasso();
30 int getPassosPorVolta();
31 void setarModoPasso(int mp, byte b);
32 void passo(void);
33 private:
34 Pino *pinoEnable, *pinoReset, *pinoSleep, *pinoM0, *pinoM1, *pinoM2, *
      pinoPasso, *pinoDirecao, *pinoIN1, *pinoIN2, *pinoIN3, *pinoIN4;
35 TipoAcionamento tipoAcionamento = TipoAcionamento::SOFTWARE;
36 TipoPasso tipoPasso = TipoPasso::WAVESTEP;
37 Direcao direcao = Direcao::HORARIO;
38 int passoAtual = 0, modoPasso = 1, passosPorVolta = 200;
39 byte *vetorPassos;
40 void setarEstados(byte b);
41 void executarPasso(int contador, int limiteInferior, int limiteSuperior);
42 };
43 #endif

```

Para flexibilização na utilização de diferentes tipos de drivers, foi desenvolvido duas construções diferentes para utilizar as funções da classe “Driver”.

A primeira é passando como parâmetros o pino do passo e da direção como no código abaixo. Essa construção é para drivers que se responsabilizam pelo controle do estado das bobinas através do seu hardware.

```

1 Pino *pinoEnableX, *pinoResetX, *pinoSleepX, *pinoM0X, *pinoM1X, *pinoM2X,
   *pinoPassoX, *pinoDirecao;
2 pinoEnableX = new Pino(Pino::DIGITAL, PINO_ENABLE_X, OUTPUT, LOW);
3 pinoResetX = new Pino(Pino::DIGITAL, PINO_RESET_X, OUTPUT, LOW);
4 pinoSleepX = new Pino(Pino::DIGITAL, PINO_SLEEP_X, OUTPUT, LOW);
5 pinoM0X = new Pino(Pino::DIGITAL, PINO_M0_X, OUTPUT, LOW);
6 pinoM1X = new Pino(Pino::DIGITAL, PINO_M1_X, OUTPUT, LOW);
7 pinoM2X = new Pino(Pino::DIGITAL, PINO_M2_X, OUTPUT, LOW);
8 pinoPassoX = new Pino(Pino::DIGITAL, PINO_PASSO_X, OUTPUT, LOW);
9 pinoDirecaoX = new Pino(Pino::DIGITAL, PINO_DIRECAO_X, OUTPUT, LOW);
10 Driver *driverX;
11 driverX = new Driver(pinoEnableX, pinoResetX, pinoSleepX, pinoM0X, pinoM1X,
   pinoM2X, pinoPassoX, pinoDirecaoX);
```

Já a segunda é passando como parâmetros os pinos das bobinas como no código abaixo. Essa construção é para drivers que terceirizam para o software o controle do estado das bobinas.

```

1 byte vetorPassos [8] = {9, 1, 3, 2, 6, 4, 12, 8};
2 Pino *pinoEnableX, *pinoResetX, *pinoSleepX, *pinoM0X, *pinoM1X, *pinoM2X,
   pinoIN1X, pinoIN2X, pinoIN3X, pinoIN4X;
3 Driver *driverX;
4 pinoEnableX = new Pino(Pino::DIGITAL, PINO_ENABLE_X, OUTPUT, LOW);
5 pinoResetX = new Pino(Pino::DIGITAL, PINO_RESET_X, OUTPUT, LOW);
6 pinoSleepX = new Pino(Pino::DIGITAL, PINO_SLEEP_X, OUTPUT, LOW);
7 pinoM0X = new Pino(Pino::DIGITAL, PINO_M0_X, OUTPUT, LOW);
8 pinoM1X = new Pino(Pino::DIGITAL, PINO_M1_X, OUTPUT, LOW);
9 pinoM2X = new Pino(Pino::DIGITAL, PINO_M2_X, OUTPUT, LOW);
10 pinoIN1X = new Pino(Pino::DIGITAL, PINO_IN1_X, OUTPUT, LOW);
11 pinoIN2X = new Pino(Pino::DIGITAL, PINO_IN2_X, OUTPUT, LOW);
12 pinoIN3X = new Pino(Pino::DIGITAL, PINO_IN3_X, OUTPUT, LOW);
13 pinoIN4X = new Pino(Pino::DIGITAL, PINO_IN4_X, OUTPUT, LOW);
14 Driver *driverX;
15 driverX = new Driver(Driver::WAVESTEP, pinoEnableX, pinoResetX,
   pinoSleepX, pinoM0X, pinoM1X, pinoM2X, pinoIN1X, pinoIN2X, pinoIN3X,
```

```
pinoIN4X , vetorPassos );
```

4.3.4 Código header da classe Eixo

Segue abaixo o arquivo header (Eixo.h) da classe “Eixo”.

```

1 #ifndef EIXO_H_INCLUDED
2 #define EIXO_H_INCLUDED
3 class Eixo {
4 public:
5 ~Eixo( void ) {}
6 Eixo( Driver *driver , Pino *pinoCursoMaximo , Pino *pinoCursoMinimo ,
7       Sigmoidal *sigmoidal );
8 Eixo( Driver *driver , byte milimetrosPorVolta , bool estadoFimDeCurso , Pino *
9       pinoCursoMaximo , Pino *pinoCursoMinimo , Sigmoidal *sigmoidal );
10 void setPosicao( double p );
11 Driver* getDriver();
12 Pino* getPinoCursoMaximo();
13 Pino* getPinoCursoMinimo();
14 bool getEstadoFimDeCurso();
15 double getPosicao();
16 Sigmoidal* getSigmoidal();
17 void rotacionarPara( int coordenada );
18 void calibrar();
19 private:
20 Driver *driver;
21 Pino *pinoCursoMaximo , *pinoCursoMinimo ;
22 Sigmoidal *sigmoidal;
23 int posicao = 0;
24 int micrometrosPorVolta = 8000;
25 bool estadoFimDeCurso = LOW;
26 unsigned long previousMillis = 0;
27 bool podeMovimentar();
28 void rotacionarPasso( Driver::Direcao direcao , int passos );
29 };
30 #endif
```

Assim para se utilizar as funções presentes na classe “Eixo” é criado um objeto através do comando abaixo com seus devidos parâmetros de entrada.

```

1 Pino *pinoEnableX, *pinoResetX, *pinoSleepX, *pinoM0X, *pinoM1X, *pinoM2X,
  *pinoPassoX, *pinoDirecao;
2 pinoEnableX = new Pino(Pino::DIGITAL, PINO_ENABLE_X, OUTPUT, LOW);
3 pinoResetX = new Pino(Pino::DIGITAL, PINO_RESET_X, OUTPUT, LOW);
4 pinoSleepX = new Pino(Pino::DIGITAL, PINO_SLEEP_X, OUTPUT, LOW);
5 pinoM0X = new Pino(Pino::DIGITAL, PINO_M0_X, OUTPUT, LOW);
6 pinoM1X = new Pino(Pino::DIGITAL, PINO_M1_X, OUTPUT, LOW);
7 pinoM2X = new Pino(Pino::DIGITAL, PINO_M2_X, OUTPUT, LOW);
8 pinoPassoX = new Pino(Pino::DIGITAL, PINO_PASSO_X, OUTPUT, LOW);
9 pinoDirecaoX = new Pino(Pino::DIGITAL, PINO_DIRECAO_X, OUTPUT, LOW);
10 Driver *driverX;
11 driverX = new Driver(pinoEnableX, pinoResetX, pinoSleepX, pinoM0X, pinoM1X,
  pinoM2X, pinoPassoX, pinoDirecaoX);
12 Pino *cursoMaximoX, *cursoMinimoX;
13 Sigmoidal *sigmoidalX;
14 cursoMaximoX = new Pino(Pino::DIGITAL, PINO_CURSOMAXIMO_X, INPUT, HIGH);
15 cursoMinimoX = new Pino(Pino::DIGITAL, PINO_CURSOMINIMO_X, INPUT, HIGH);
16 sigmoidalX = new Sigmoidal(SIG_PERIODO_MAXIMO_X, SIG_PERIODO_MINIMO_X,
  SIG_DECLIVIDADE_X);
17 Eixo *eixoX;
18 eixoX = new Eixo(driverX, MILIMETROS_POR_VOLTA_X, ESTADO_FIM_DE_CURSO_X,
  cursoMaximoX, cursoMinimoX, sigmoidalX);

```

4.3.5 Detalhamento do arquivo principal (MesaCartesiana.ino)

O arquivo principal tem a responsabilidade de gerenciar toda aplicação, neste arquivo serão construídos todos objetos necessários. Segue abaixo um resumo com todas funcionalidades que este possui.

```

1 void inicializarEixos(Driver::TipoAccionamento tp);
2 void interpretarComandos(String comando);
3 void movimentarMesa(String coordenada);
4 void escolherModoPasso(Eixo *e, String modoPasso);
5 void setup();
6 void loop();
7 void serialEvent();

```

O método “incializarEixos” é responsável por construir os objetos necessários para

controle dos eixos da aplicação, neste é possível escolher se o tipo de acionamento do driver é via software ou hardware.

O método “interpretarComandos” é responsável pela interpretação de qual comando foi enviado pelo usuário, como por exemplo o comando “MOVER 2,4” que movimenta o eixo X para a ordenada 2 e o eixo Y para a ordenada 4. Outros comandos necessários para uma aplicação como a mudança de estados de algumas configurações devem ser interpretadas por este método.

O método “movimentarMesa” é responsável pela chamada da funcionalidade de rotação presente na classe “Eixo” para isso recebe como parâmetro a coordenada enviada pelo usuário.

O método “escolherModoPasso” é responsável por setar qual o modo do passo que o eixo irá operar.

O método “setup” é responsável pela inicialização de toda aplicação estando nele uma chamada para o método de inicialização dos eixos.

O método “serialEvent” é responsável pela recepção dos dados enviados pelo usuário estando nele uma chamada para o método de interpretação de comandos.

O método “loop” é responsável pela execução da aplicação de forma que ela nunca acabe, já que é necessário que o método “serialEvent” esteja sempre em operação para quando o usuário solicitar alguma ação no sistema esta seja executada.

5 CONSIDERAÇÕES FINAIS

O objetivo geral do projeto que consiste na movimentação de instrumentos de medição em um sistema de coordenadas cartesianas foi atendido. Nesse sentido, os objetivos específicos de projetar a mesa cartesiana, criar o sistema de comunicação “mesa-software” e desenvolver o software de comando foram contemplados.

A mesa cartesiana foi projetada com recursos disponíveis do mercado indicando que é possível construir sistemas com propósitos parecidos que atendem solicitações de laboratórios.

A documentação deste trabalho foi realizada a fim de facilitar o entendimento para o desenvolvimento de outros projetos parecidos e assim estender o impacto deste trabalho em outras instituições de ensino. Isso permitirá que um maior número de pesquisadores possam realizar a captura de dados para experimentos de forma automatizada em túneis de vento.

Assim, o código fonte presente na placa de prototipação Arduino está disponível em um repositório online no endereço <<https://github.com/abaldezjr/MesaCartesiana>>. Também está disponível no endereço <<https://github.com/abaldezjr/tcc>>, uma versão em Latex com todo o material necessário para a montagem do trabalho, este tem o objetivo de auxiliar a montagem de trabalhos de conclusão de curso futuros.

Com este trabalho, foi possível utilizar diversos dos conhecimentos adquiridos ao longo do curso de Engenharia Mecânica utilizando-os de forma integrada, visto que envolveu a concepção de um sistema mecânico movimentado por motores elétricos, além de aproveitar-se do desenvolvimento de software na integração do projeto como um todo.

5.1 CRÍTICAS E SUGESTÕES DE TRABALHOS FUTUROS

Algumas críticas podem ser levantadas sobre o protótipo do projeto como:

- A versão do projeto ainda não possui um sistema completamente automatizado, sendo possível realizar somente a captura dos dados de uma coordenada por vez.
- Os dados ainda não foram capturados através de algum sensor digital.
- Os dados não são apresentados de forma gráfica.
- O sistema eletrônico não tem um case para seu abrigo estando desprotegido, além de ter sido desenvolvido em uma versão de protótipo com a placa Arduino.

Algumas melhorias podem ser sugeridas para uma próxima versão do projeto como:

- Capturar os dados de forma digital através de um sensor.

- Automatizar a captura de dados para que seja possível o usuário planejar uma rotina e saber o seu tempo estimado.
- Integrar o sistema de software da mesa cartesiana a um aplicativo que faça geração de relatórios, para uma visualização dos dados de forma gráfica.
- Desenvolver o circuito eletrônico em uma placa de circuito impresso com os devidos componentes.
- Projetar um case (caixa) para abrigar os componentes do circuito eletrônico.

REFERÊNCIAS

- ALCIATORE, D. G.; HISTAND, M. B. **Introdução à Mecatrônica e aos Sistemas de Medições.** [S.I.]: AMGH Editora, 2014.
- BUDYNAS, R. G.; NISBETH, J. K. **Elementos de Máquinas de Shigley-10^a Edição.** [S.I.]: McGraw Hill Brasil, 2016.
- BUTIGNOL, M. R. Adequação de uma mesa xyz para fins didáticos. 2017. Monografia (Bacharel em Engenharia Mecatrônica), IFSC (Instituto Federal de Educação, Ciência e Tecnologia de Santa Catarina), Florianópolis, Brazil. Disponível em: <https://repositorio.ifsc.edu.br/bitstream/handle/123456789/571/ADEQUACAO_DE_UMA_MESA_XYZ_PARA_FINS_DIDATICOS.pdf?sequence=1&isAllowed=y>.
- CAMARGO, L. F. S. et al. Mesa de coordenadas xy para aplicação em microengenharia com cnc: projeto e analise. 1988. Disponível em: <<https://repositorio.ufsc.br/xmlui/bitstream/handle/123456789/75448/80422.pdf?sequence=1&isAllowed=y>>.
- CARMINATTI, L. J.; KONRATH, R. Desenvolvimento de um túnel de vento subsônico com foco no ensino didático. **Anais da Engenharia Mecânica/ISSN 2594-4649**, v. 4, n. 1, p. 17–33, 2019. Disponível em: <<https://uceff.edu.br/anais/index.php/engmec/article/view/229/221>>.
- HOSS, D. L. Implantação de controle de velocidade em túnel de vento movido à motor de combustão para testes de turbinas eólicas. p. 121, 2018. Disponível em: <<https://repositorio.ifsc.edu.br/bitstream/handle/123456789/532/Implementaç~ao%20de%20controle%20de%20velocidade%20em%20túnel%20de%20vento.pdf?sequence=1>>.
- JOGLEKAR, B.; MOURYA, R. M. Design, construction and testing open circuit low speed wind tunnel. **International Journal of Engineering Research and Reviews**, v. 2, n. 4, p. 1–9, 2014.
- JÚNIOR, L. C. d. F.; SILVA, R. S. da. **Máquinas Elétricas.** [S.I.]: Editora e Distribuidora Educacional S.A., 2018.
- PRITCHARD, P. J.; MITCHELL, J. W. **Fox and McDonald's introduction to fluid mechanics.** John Wiley & Sons, 2005. Disponível em: <http://ftp.demec.ufpr.br/disciplinas/TM240/Marchi/Bibliografia/Pritchard-Fox-McDonalds_2011_8ed_Fluid-Mechanics.pdf>.
- RAMOS, L. G. d. R. Desenvolvimento de um sistema de deslocamento bi-axial para aplicação em túnel aerodinâmico. 2018.
- ROCHA, F.; SERRANTOLA, W.; LOPEZ, G. N.; TORGÀ, D.; CARVALHO, M.; SOUZA, G. et al. Retrofitting de uma mesa xy. v. 501778, 2015. Disponível em: <<http://swge.inf.br/SBAI2015/anais/554.pdf>>.
- SANTOS, A. R. dos; SILVA, B. W. X. da; NETO, F. S.; LOPES, L. D.; DIONÍSIO, T. H. Elaboração de túnel de vento para aplicações de ensaios aerodinâmicos. p. 39, 2014. Disponível em: <https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&ved=2ahUKEwjO5pzx7_nuAhV_HLkGHT9GCXkQFjAAegQIAhAD&url=https%3A%2F%2Fsjc.ifsp.edu.br%2Fbiblioteca%2Findex.php%2Fcomponent%2Fphocadownload%2Fcategory%2F5-mecanica%3Fdownload%3D4%3Aelaboracao-de-tunel-de-vento-para-aplicacoes-de-ensaios-aerodinamicos&usg=AQvVaw2RnEW_mEs0Z9K7xVBjPVyZ>.

APÊNDICES

APÊNDICE A – CÓDIGO PRINCIPAL PARA O CONTROLE DA MESA CARTESIANA

```

1 #include "Arduino.h"
2 #include "StDriver/src/StDriver.h"
3 #define MODO_PASSO_X 1
4 #define PINO_ENABLE_X 30
5 #define PINO_RESET_X 40
6 #define PINO_SLEEP_X 50
7 #define PINO_M0_X 60
8 #define PINO_M1_X 70
9 #define PINO_M2_X 80
10 #define PINO_PASSO_X 200
11 #define PINO_DIRECAO_X 200
12 #define PINO_IN1_X 0
13 #define PINO_IN2_X 0
14 #define PINO_IN3_X 0
15 #define PINO_IN4_X 0
16 #define PINO_CURSOMAXIMO_X 3
17 #define PINO_CURSOMINIMO_X 4
18 #define ESTADO_FIM_DE_CURSO_X LOW
19 #define MILIMETROS_POR_VOLTA_X 8
20 #define LIMITE_MAXIMO_X 10
21 #define SIG_PERIODO_MAXIMO_X 600
22 #define SIG_PERIODO_MINIMO_X 100
23 #define SIG_DECLIVIDADE_X 0.1
24 #define MODO_PASSO_Y 1
25 #define PINO_ENABLE_Y 15
26 #define PINO_RESET_Y 16
27 #define PINO_SLEEP_Y 17
28 #define PINO_M0_Y 18
29 #define PINO_M1_Y 19
30 #define PINO_M2_Y 20
31 #define PINO_PASSO_Y 34
32 #define PINO_DIRECAO_Y 44
33 #define PINO_IN1_Y 0
34 #define PINO_IN2_Y 0
35 #define PINO_IN3_Y 0
36 #define PINO_IN4_Y 0
37 #define PINO_CURSOMAXIMO_Y 8

```

```

38 #define PINO_CURSOMINIMO_Y 9
39 #define ESTADO_FIM_DE_CURSO_Y LOW
40 #define MILIMETROS_POR_VOLTA_Y 8
41 #define LIMITE_MAXIMO_Y 10
42 #define SIG_PERIODO_MAXIMO_Y 600
43 #define SIG_PERIODO_MINIMO_Y 100
44 #define SIG_DECLIVIDADE_Y 0.1
45 String leitura;
46 int x = 0, y = 0;
47 byte vetorPassos [8] = {9, 1, 3, 2, 6, 4, 12, 8};
48 Pino *pino13, *pinoEnableX, *pinoResetX, *pinoSleepX,
49 *pinoM0X, *pinoM1X, *pinoM2X,* pinoPassoX, *pinoDirecaoX ,
50 *pinoIN1X, *pinoIN2X, *pinoIN3X, *pinoIN4X,*cursoMaximoX, *cursoMinimoX ;
51 Sigmoidal *sigmoidalX;
52 Driver *driverX;
53 Eixo *eixoX;
54 Pino *pinoEnableY, *pinoResetY, *pinoSleepY ,
55 *pinoM0Y, *pinoM1Y, *pinoM2Y,* pinoPassoY, *pinoDirecaoY ,
56 *pinoIN1Y, *pinoIN2Y, *pinoIN3Y, *pinoIN4Y ,
57 *cursoMaximoY, *cursoMinimoY;
58 Sigmoidal *sigmoidalY;
59 Driver *driverY;
60 Eixo *eixoY;
61 void inicializarEixos(Driver::TipoAcionamento tp);
62 void interpretarComandos(String comando);
63 void movimentarMesa(String coordenada);
64 void escolherModoPasso(Eixo *e, String modoPasso);
65 void setup(){
66   Serial.begin(9600);
67   pino13 = new Pino(Pino::DIGITAL, 13, OUTPUT, LOW);
68   inicializarEixos(Driver::SOFTWARE);
69   delay(3000);
70 }
71 void loop(){
72
73 }
74 void serialEvent(){
75   if(Serial.available() > 0){
76     leitura = Serial.readStringUntil('\n');

```

```

77  if (!leitura .equals("")){
78      interpretarComandos(leitura );
79      leitura = "";
80  }
81 }
82 }
83 void inicializarEixos(Driver ::TipoAccionamento tp){
84     pinoEnableX = new Pino(Pino ::DIGITAL, PINO_ENABLE_X, OUTPUT, LOW);
85     pinoResetX = new Pino(Pino ::DIGITAL, PINO_RESET_X, OUTPUT, LOW);
86     pinoSleepX = new Pino(Pino ::DIGITAL, PINO_SLEEP_X, OUTPUT, LOW);
87     pinoM0X = new Pino(Pino ::DIGITAL, PINO_M0_X, OUTPUT, LOW);
88     pinoM1X = new Pino(Pino ::DIGITAL, PINO_M1_X, OUTPUT, LOW);
89     pinoM2X = new Pino(Pino ::DIGITAL, PINO_M2_X, OUTPUT, LOW);
90     cursoMaximoX = new Pino(Pino ::DIGITAL, PINO_CURSOMAXIMO_X, INPUT, HIGH);
91     cursoMinimoX = new Pino(Pino ::DIGITAL, PINO_CURSOMINIMO_X, INPUT, HIGH);
92     sigmoidalX = new Sigmoidal(SIG_PERIODO_MAXIMO_X, SIG_PERIODO_MINIMO_X,
93                                 SIG_DECLIVIDADE_X);
93     pinoEnableY = new Pino(Pino ::DIGITAL, PINO_ENABLE_Y, OUTPUT, LOW);
94     pinoResetY = new Pino(Pino ::DIGITAL, PINO_RESET_Y, OUTPUT, LOW);
95     pinoSleepY = new Pino(Pino ::DIGITAL, PINO_SLEEP_Y, OUTPUT, LOW);
96     pinoM0Y = new Pino(Pino ::DIGITAL, PINO_M0_Y, OUTPUT, LOW);
97     pinoM1Y = new Pino(Pino ::DIGITAL, PINO_M1_Y, OUTPUT, LOW);
98     pinoM2Y = new Pino(Pino ::DIGITAL, PINO_M2_Y, OUTPUT, LOW);
99     cursoMaximoY = new Pino(Pino ::DIGITAL, PINO_CURSOMAXIMO_Y, INPUT, HIGH);
100    cursoMinimoY = new Pino(Pino ::DIGITAL, PINO_CURSOMINIMO_Y, INPUT, HIGH);
101    sigmoidalY = new Sigmoidal(SIG_PERIODO_MAXIMO_Y, SIG_PERIODO_MINIMO_Y,
102                                SIG_DECLIVIDADE_Y);
102    if (tp == Driver ::HARDWARE){
103        pinoPassoX = new Pino(Pino ::DIGITAL, PINO_PASSO_X, OUTPUT, LOW);
104        pinoDirecaoX = new Pino(Pino ::DIGITAL, PINO_DIRECAO_X, OUTPUT, LOW);
105        pinoPassoY = new Pino(Pino ::DIGITAL, PINO_PASSO_Y, OUTPUT, LOW);
106        pinoDirecaoY = new Pino(Pino ::DIGITAL, PINO_DIRECAO_Y, OUTPUT, LOW);
107        driverX = new Driver(pinoEnableX, pinoResetX, pinoSleepX, pinoM0X,
108                             pinoM1X, pinoM2X, pinoPassoX, pinoDirecaoX);
109        driverY = new Driver(pinoEnableY, pinoResetY, pinoSleepY, pinoM0Y,
110                             pinoM1Y, pinoM2Y, pinoPassoY, pinoDirecaoY);
111    } else {
112        pinoIN1X = new Pino(Pino ::DIGITAL, PINO_IN1_X, OUTPUT, LOW);
113        pinoIN2X = new Pino(Pino ::DIGITAL, PINO_IN2_X, OUTPUT, LOW);

```

```

112     pinoIN3X = new Pino(Pino::DIGITAL, PINO_IN3_X, OUTPUT, LOW);
113     pinoIN4X = new Pino(Pino::DIGITAL, PINO_IN4_X, OUTPUT, LOW);
114     pinoIN1Y = new Pino(Pino::DIGITAL, PINO_IN1_Y, OUTPUT, LOW);
115     pinoIN2Y = new Pino(Pino::DIGITAL, PINO_IN2_Y, OUTPUT, LOW);
116     pinoIN3Y = new Pino(Pino::DIGITAL, PINO_IN3_Y, OUTPUT, LOW);
117     pinoIN4Y = new Pino(Pino::DIGITAL, PINO_IN4_Y, OUTPUT, LOW);
118     driverX = new Driver(Driver::WAVESTEP, pinoEnableX, pinoResetX,
119     pinoSleepX, pinoM0X, pinoM1X, pinoM2X, pinoIN1X, pinoIN2X, pinoIN3X,
120     pinoIN4X, vetorPassos);
121     driverY = new Driver(Driver::WAVESTEP, pinoEnableY, pinoResetY,
122     pinoSleepY, pinoM0Y, pinoM1Y, pinoM2Y, pinoIN1Y, pinoIN2Y, pinoIN3Y,
123     pinoIN4Y, vetorPassos);
124 }
125 eixoX = new Eixo(driverX, MILIMETROS_POR_VOLTA_X, ESTADO_FIM_DE_CURSO_X,
126 cursoMaximoX, cursoMinimoX, sigmoidalX);
127 eixoY = new Eixo(driverY, MILIMETROS_POR_VOLTA_Y, ESTADO_FIM_DE_CURSO_Y,
128 cursoMaximoY, cursoMinimoY, sigmoidalY);
129 }
130 void interpretarComandos(String comando){
131     comando.toUpperCase();
132     if (comando.indexOf("MOVER") > -1) movimentarMesa(comando.substring(
133         comando.indexOf(" ") + 1));
134     if (comando.indexOf("L") > -1) pino13->setEstado(HIGH);
135     if (comando.indexOf("D") > -1) pino13->setEstado(LOW);
136     if (comando.indexOf("MODOPASSOX") > -1) escolherModoPasso(eixoX, comando.
137         substring(comando.indexOf(" ") + 1));
138     if (comando.indexOf("MODOPASSOY") > -1) escolherModoPasso(eixoY, comando.
139         substring(comando.indexOf(" ") + 1));
140 }
141 void movimentarMesa(String coordenada){
142     if (!(coordenada.indexOf(",") == -1)){
143         x = coordenada.substring(0, coordenada.indexOf(",")).toDouble();
144         y = coordenada.substring(coordenada.indexOf(",") + 1).toDouble();
145         if(x >= 0 && x <= 10 && y >= 0 && y <= 10){
146             if(x != eixoX->getPosicao()) {
147                 eixoX->rotacionarPara(x);
148                 Serial.println(eixoX->getPosicao());
149             }
150             if(y != eixoY->getPosicao()) {
151                 eixoY->rotacionarPara(y);
152                 Serial.println(eixoY->getPosicao());
153             }
154         }
155     }
156 }

```

```
142     eixoY->rotacionarPara(y);
143     Serial.println(eixoY->getPosicao());
144 }
145 }
146 }
147 }
148 void escolherModoPasso(Eixo *e, String modoPasso){
149     switch(modoPasso.toInt()){
150         case 1: e->getDriver()->setarModoPasso( 1,0);break;
151         case 2: e->getDriver()->setarModoPasso( 2,4);break;
152         case 4: e->getDriver()->setarModoPasso( 4,2);break;
153         case 8: e->getDriver()->setarModoPasso( 8,6);break;
154     }
155 }
```

Exemplo de código A.1 – Código principal para o controle da mesa cartesiana.

APÊNDICE B – CÓDIGO DA CLASSE EIXO PARA O CONTROLE DO EIXO

```

1 #ifndef EIXO_H_INCLUDED
2 #define EIXO_H_INCLUDED
3
4 class Eixo {
5     public:
6         ~Eixo(void) {}
7         Eixo(Driver *driver, Pino *pinoCursoMaximo, Pino *pinoCursoMinimo,
8               Sigmoidal *sigmoidal) {
9             this->driver = driver;
10            this->pinoCursoMaximo = pinoCursoMaximo;
11            this->pinoCursoMinimo = pinoCursoMinimo;
12            this->sigmoidal = sigmoidal;
13        }
14        Eixo(Driver *driver, byte milimetrosPorVolta, bool estadoFimDeCurso,
15              Pino *pinoCursoMaximo, Pino *pinoCursoMinimo, Sigmoidal *sigmoidal) {
16            this->driver = driver;
17            this->milimetrosPorVolta = milimetrosPorVolta * 1000;
18            this->estadoFimDeCurso = estadoFimDeCurso;
19            this->pinoCursoMaximo = pinoCursoMaximo;
20            this->pinoCursoMinimo = pinoCursoMinimo;
21            this->sigmoidal = sigmoidal;
22        }
23        void setPosicao(double p) {
24            this->posicao = p;
25        }
26        Driver* getDriver(void) const {
27            return this->driver;
28        }
29        Pino* getPinoCursoMaximo(void) const {
30            return this->pinoCursoMaximo;
31        }
32        Pino* getPinoCursoMinimo(void) const {
33            return this->pinoCursoMinimo;
34        }
35        bool getEstadoFimDeCurso(void) const {
36            return this->estadoFimDeCurso;
37        }
38        double getPosicao(void) const {
39
40    }

```

```

36     return this->posicao;
37 }
38 Sigmoidal* getSigmoidal(void) const {
39     return this->sigmoidal;
40 }
41 void rotacionarPara(int coordenada){
42     this->rotacionarPasso(
43         (coordenada*1000 - this->posicao) > 0? Driver::HORARIO: Driver::
44 ANTIHORARIO,
45         (fabs(coordenada*1000 - this->posicao) * this->driver->
46 getPassosPorVolta()) / this->micrometrosPorVolta
47     );
48 }
49 void calibrar(){
50     this->driver->setDirecao(Driver::ANTIHORARIO);
51     this->posicao = 0;
52     while(this->pinoCursoMinimo->getEstado()){
53         this->driver->passo();
54         delayMicroseconds(this->sigmoidal->getPeriodoMaximo());
55     }
56 }
57 private:
58     Driver *driver;
59     Pino *pinoCursoMaximo, *pinoCursoMinimo;
60     Sigmoidal *sigmoidal;
61     int posicao { 0 };
62     int micrometrosPorVolta { 8000 };
63     bool estadoFimDeCurso { LOW };
64     unsigned long previousMillis { 0 };
65     bool podeMovimentar(void) const {
66         return ( this->estadoFimDeCurso && ((this->driver->getDirecao() && !
67             this->pinoCursoMaximo->getEstado()) || (!this->driver->getDirecao() && !
68             this->pinoCursoMinimo->getEstado())) ) || (!this->estadoFimDeCurso && ((
69             this->driver->getDirecao() && this->pinoCursoMaximo->getEstado() || (
70             this->driver->getDirecao() && this->pinoCursoMinimo->getEstado())));
71     }
72     void rotacionarPasso(Driver::Direcao direcao, int passos){
73         this->driver->setDirecao(direcao);
74     }
75 }
```

```
68     for(int i = 0, iSig = 0; i < passos; i++, i <= ((passos * this->  
69         driver->getModoPasso()) / 2)? iSig++: iSig--){  
70         if(this->podeMovimentar()){  
71             this->driver->passo();  
72             delayMicroseconds(this->sigmoidal->calculoDoInstante(iSig));  
73             this->posicao += direcao * (this->micrometrosPorVolta / this->  
74                 driver->getPassosPorVolta());  
75         } else{  
76             i = passos;  
77             delay(3000);  
78             this->calibrar();  
79         }  
80     }  
81 }  
#endif
```

Exemplo de código B.1 – Código da classe eixo para o controle do eixo.

APÊNDICE C – CÓDIGO DA CLASSE DRIVER PARA O CONTROLE DO DRIVER DE POTÊNCIA.

```

35     this ->pinoM0 = pinoM0;
36     this ->pinoM1 = pinoM1;
37     this ->pinoM2 = pinoM2;
38     this ->pinoPasso = pinoPasso;
39     this ->pinoDirecao = pinoDirecao;
40     this ->tipoAcionamento = TipoAcionamento ::HARDWARE;
41 }
42 Driver(TipoPasso tipoPasso, Pino* pinoEnable, Pino* pinoReset, Pino*
43 pinoSleep, Pino* pinoM0, Pino* pinoM1, Pino* pinoM2, Pino* pinoIN1, Pino
44 * pinoIN2, Pino* pinoIN3, Pino* pinoIN4, byte *vetorPassos){
45     this ->setarModoPasso(1, 0);
46     this ->tipoPasso = tipoPasso;
47     this ->pinoEnable = pinoEnable;
48     this ->pinoReset = pinoReset;
49     this ->pinoSleep = pinoSleep;
50     this ->pinoM0 = pinoM0;
51     this ->pinoM1 = pinoM1;
52     this ->pinoM2 = pinoM2;
53     this ->pinoIN1 = pinoIN1;
54     this ->pinoIN2 = pinoIN2;
55     this ->pinoIN3 = pinoIN3;
56     this ->pinoIN4 = pinoIN4;
57     this ->direcao = Driver::HORARIO;
58     this ->vetorPassos = vetorPassos;
59     this ->tipoAcionamento = TipoAcionamento ::SOFTWARE;
60 }
61 void setDirecao(Direcao d){
62     this ->direcao = d;
63     if (this ->tipoAcionamento == TipoAcionamento ::HARDWARE)
64         this ->pinoDirecao ->setEstado((this ->direcao == Driver::HORARIO) ?
65 HIGH: LOW);
66 }
67 Pino* getPinoEnable(void) const {
68     return this ->pinoEnable;
69 }
70 Pino* getPinoReset(void) const {
71     return this ->pinoReset;
72 }
73 Pino* getPinoSleep(void) const {
74 }
```

```

71     return this->pinoSleep;
72 }
73 Pino* getPinoPasso(void) const {
74     return this->pinoPasso;
75 }
76 Pino* getPinoDirecao(void) const {
77     return this->pinoDirecao;
78 }
79 Direcao getDirecao(void) {
80     if(this->tipoAcionamento == TipoAcionamento::HARDWARE)
81         this->direcao = this->pinoDirecao->getEstado() ? Direcao::HORARIO:
82 Direcao::ANTIHORARIO;
83     return this->direcao;
84 }
85 int getModoPasso(void) const {
86     return this->modoPasso;
87 }
88 int getPassosPorVolta(void) const {
89     return this->passosPorVolta;
90 }
91 void setarModoPasso(int mp, byte b){
92     this->modoPasso = mp;
93     this->passosPorVolta = 200 * this->modoPasso;
94     this->pinoM0->setEstado((b >> 2) & 1);
95     this->pinoM1->setEstado((b >> 1) & 1);
96     this->pinoM2->setEstado((b >> 0) & 1);
97     this->passoAtual = TipoPasso::FULLSTEP? 1 :0;
98 }
99 void passo(void){
100     if(this->tipoAcionamento == TipoAcionamento::HARDWARE){
101         this->pinoPasso->setEstado(this->pinoPasso->getEstado() ?LOW:HIGH);
102     } else {
103         switch(this->tipoPasso){
104             case TipoPasso::WAVESTEP: this->executarPasso(2,1,7); break;
105             case TipoPasso::FULLSTEP: this->executarPasso(2,0,6); break;
106             case TipoPasso::HALFSTEP: this->executarPasso(1,0,7); break;
107         }
108     }

```

```

109 private:
110     Pino *pinoEnable, *pinoReset, *pinoSleep, *pinoM0, *pinoM1, *pinoM2, *
111     pinoPasso, *pinoDirecao, *pinoIN1, *pinoIN2, *pinoIN3, *pinoIN4;
112     TipoAcionamento tipoAcionamento { TipoAcionamento::SOFTWARE };
113     TipoPasso tipoPasso { TipoPasso::WAVESTEP };
114     Direcao direcao { Direcao::HORARIO };
115     int passoAtual { 0 }, modoPasso { 1 }, passosPorVolta { 200 };
116     byte *vetorPassos;
117     void setarEstados(byte b){
118         this->pinoIN1->setEstado((b >> 0) & 1);
119         this->pinoIN2->setEstado((b >> 1) & 1);
120         this->pinoIN3->setEstado((b >> 2) & 1);
121         this->pinoIN4->setEstado((b >> 3) & 1);
122     }
123     void executarPasso(int contador, int limiteInferior, int limiteSuperior
124 ){
125         this->setarEstados(this->vetorPassos[ this->passoAtual ]);
126         this->passoAtual = this->direcao == Direcao::HORARIO? this->
127             passoAtual + contador :this->passoAtual - contador;
128         if(this->passoAtual >= limiteSuperior) this->passoAtual =
129             limiteInferior;
130         if(this->passoAtual < limiteInferior) this->passoAtual =
131             limiteSuperior;
132     }
133 };
134 #endif

```

Exemplo de código C.1 – Código da classe driver para o controle do driver de potência

APÊNDICE D – CÓDIGO DA CLASSE PINO PARA O CONTROLE DOS PINOS

```

1 #ifndef PINO_H_INCLUDED
2 #define PINO_H_INCLUDED
3
4 class Pino {
5     public:
6         enum TipoPino {
7             ANALOGICO = 0,
8             DIGITAL    = 1,
9         };
10        ~Pino(void){}
11        Pino(TipoPino t, byte n, byte m, unsigned int e){
12            this->setTipo(t);
13            this->setNumero(n);
14            this->setModo(m);
15            this->setEstado(e);
16        }
17        void setModo(byte m){
18            this->modo = m;
19            pinMode( this->numero, this->modo );
20        }
21        void setEstado(unsigned int e){
22            this->estado = e;
23            this->tipo == TipoPino::DIGITAL? digitalWrite(this->numero, this->
24            estado): analogWrite(this->numero, this->estado);
25        }
26        TipoPino getTipo(void) const {
27            return this->tipo;
28        }
29        byte getNumero(void) const {
30            return this->numero;
31        }
32        byte getModo(void) const {
33            return this->modo;
34        }
35        unsigned int getEstado(void){
36            this->estado = this->tipo == TipoPino::DIGITAL? digitalRead(this->
37            numero): analogRead(this->numero);
38            return this->estado;
39        }

```

```
36     }
37 private:
38     TipoPino tipo { TipoPino ::DIGITAL };
39     byte numero { 0 };
40     byte modo { OUTPUT };
41     unsigned int estado { LOW };
42     void setTipo(TipoPino t){
43         this ->tipo = t;
44     }
45     void setNumero(byte n){
46         this ->numero = n;
47     }
48 };
49 #endif
```

Exemplo de código D.1 – Código da classe Pino para o controle dos pinos.

APÊNDICE E – CÓDIGO DA CLASSE SIGMOIDAL PARA O CONTROLE DA ACELERAÇÃO DOS MOTORES

```

1 #ifndef SIGMOIDAL_H_INCLUDED
2 #define SIGMOIDAL_H_INCLUDED
3 #define E 2.71828182845904523536
4 class Sigmoidal {
5     public:
6         ~Sigmoidal(void){}
7         Sigmoidal(double pmax, double pmin, double de){
8             this->periodoMaximo = pmax;
9             this->periodoMinimo = pmin;
10            this->amplitude = this->periodoMaximo - this->periodoMinimo;
11            this->vetorAceleracao = 1;
12            this->declividade = de;
13            this->xMedio = this->calculoDoXMedio();
14        }
15        double calculoDoXMedio(void) const {
16            return ((-1)*log( this->amplitude / ((this->periodoMaximo * 0.99999)
17            - this->periodoMinimo) -1)) / this->declividade;
18        }
19        void setVetorAceleracao(int va){
20            this->vetorAceleracao = va;
21        }
22        void setPeriodoMaximo(double pmax){
23            this->periodoMaximo = pmax;
24        }
25        void setPeriodoMinimo(double pmin){
26            this->periodoMinimo = pmin;
27        }
28        void setDeclividade(double de){
29            this->declividade = de;
30        }
31        double getPeriodoMaximo(void) const {
32            return this->periodoMaximo;
33        }
34        double getPeriodoMinimo(void) const {
35            return this->periodoMinimo;
36        }

```

```

36 int getVetorAceleracao(void) const {
37     return this->vetorAceleracao;
38 }
39 double getAmplitude(void) const {
40     return this->amplitude;
41 }
42 double getDeclividade(void) const {
43     return this->declividade;
44 }
45 double getXMedio(void) const {
46     return this->xMedio;
47 }
48 double calculoDoInstante(int i) const {
49     return (this->amplitude / (1 + pow(E, (this->vetorAceleracao * (
50         this->declividade * (i - this->xMedio)))))) + this->periodoMinimo;
51 }
52 private:
53     double periodoMaximo { 600 }, periodoMinimo { 100 }, amplitude { this->
54     periodoMaximo - this->periodoMinimo }, declividade { 0 }, xMedio { 0 };
55     int     vetorAceleracao { 1 };
56 };
57 #endif

```

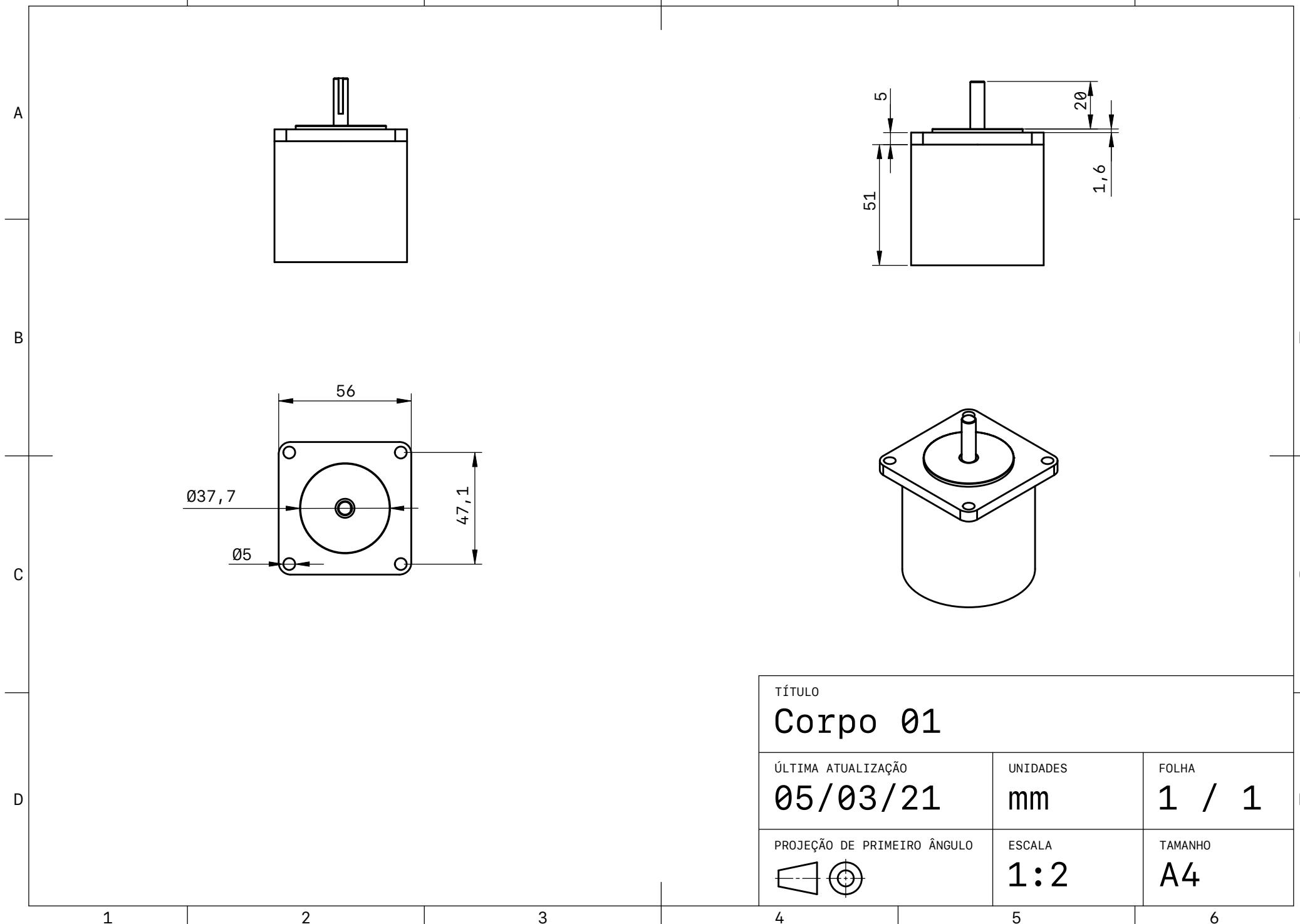
Exemplo de código E.1 – Código da classe Sigmoidal para o controle da aceleração dos motores.

APÊNDICE F – CÓDIGO DO HEADER STDIVER PARA O INCLUDE DE CLASSES

```
1 #ifndef STEPPERDRIVER_H
2 #define STEPPERDRIVER_H
3 #include <math.h>
4 #include "Pino.h"
5 #include "Sigmoidal.h"
6 #include "Driver.h"
7 #include "Eixo.h"
8#endif
```

Exemplo de código F.1 – Código do header StDriver para o include de classes.

1 2 3 4 5 6



TÍTULO

Corpo 01

ÚLTIMA ATUALIZAÇÃO

05/03/21

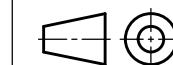
UNIDADES

mm

FOLHA

1 / 1

PROJEÇÃO DE PRIMEIRO ÂNGULO

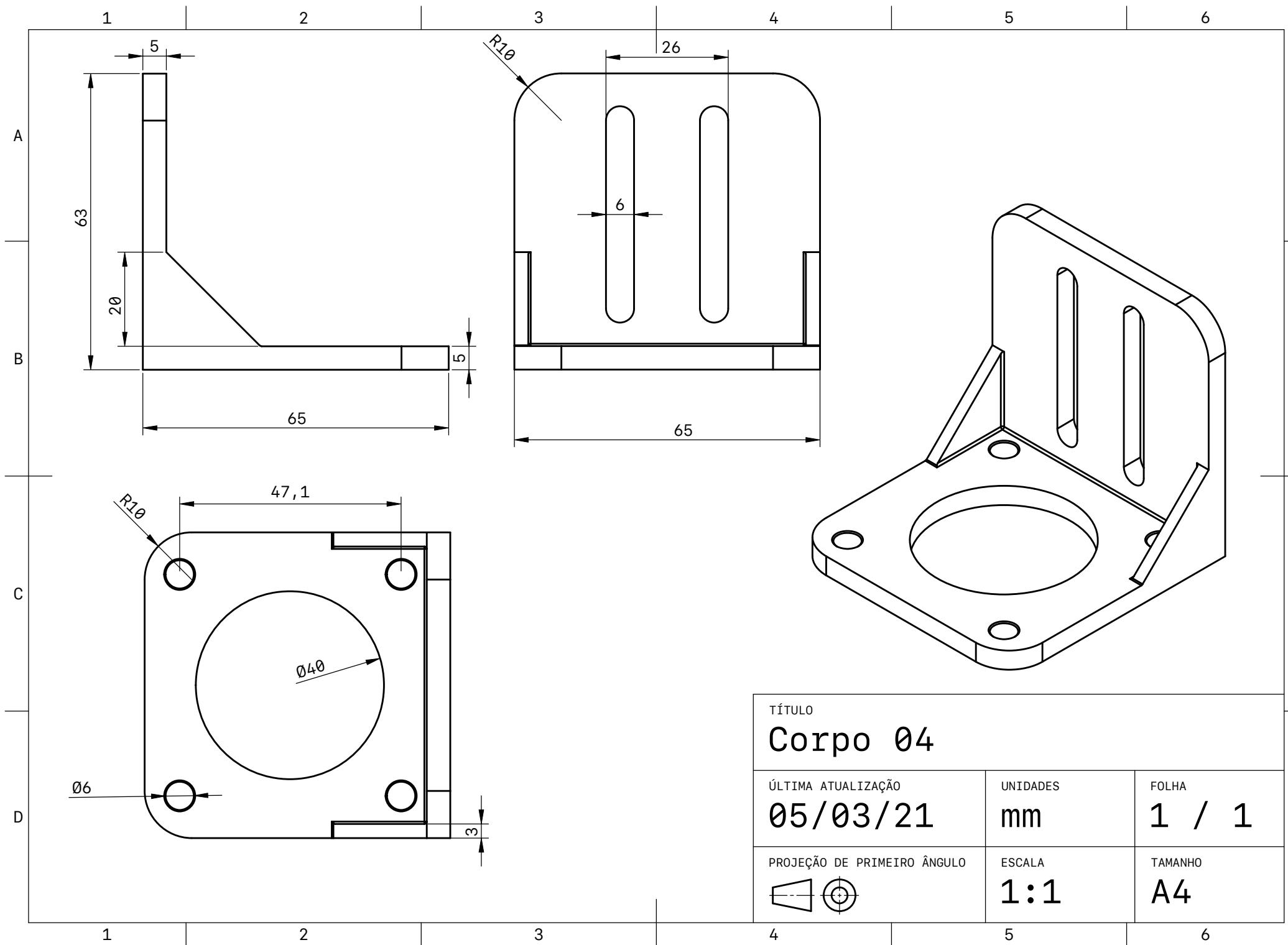


ESCALA

1:2

TAMANHO

A4



TÍTULO

Corpo 04

ÚLTIMA ATUALIZAÇÃO

05/03/21

UNIDADES

mm

FOLHA

1 / 1

PROJEÇÃO DE PRIMEIRO ÂNGULO



ESCALA

1:1

TAMANHO

A4

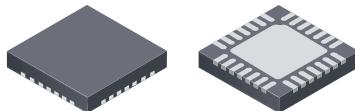
DMOS Microstepping Driver with Translator And Overcurrent Protection

Features and Benefits

- Low $R_{DS(ON)}$ outputs
- Automatic current decay mode detection/selection
- Mixed and Slow current decay modes
- Synchronous rectification for low power dissipation
- Internal UVLO
- Crossover-current protection
- 3.3 and 5 V compatible logic supply
- Thermal shutdown circuitry
- Short-to-ground protection
- Shorted load protection
- Five selectable step modes: full, $1/2$, $1/4$, $1/8$, and $1/16$

Package:

28-contact QFN
with exposed thermal pad
5 mm × 5 mm × 0.90 mm
(ET package)



Approximate size

Description

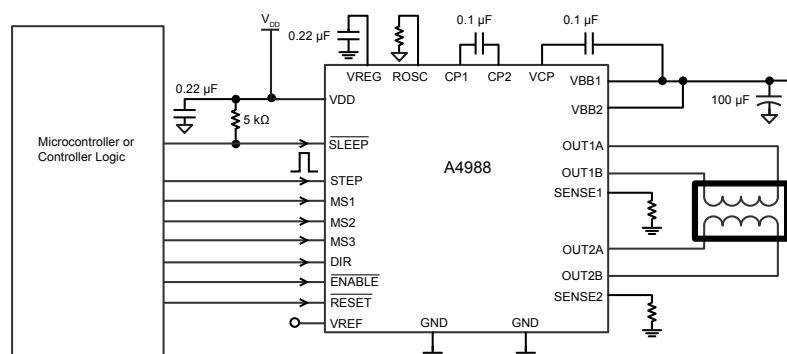
The A4988 is a complete microstepping motor driver with built-in translator for easy operation. It is designed to operate bipolar stepper motors in full-, half-, quarter-, eighth-, and sixteenth-step modes, with an output drive capacity of up to 35 V and ± 2 A. The A4988 includes a fixed off-time current regulator which has the ability to operate in Slow or Mixed decay modes.

The translator is the key to the easy implementation of the A4988. Simply inputting one pulse on the STEP input drives the motor one microstep. There are no phase sequence tables, high frequency control lines, or complex interfaces to program. The A4988 interface is an ideal fit for applications where a complex microprocessor is unavailable or is overburdened.

During stepping operation, the chopping control in the A4988 automatically selects the current decay mode, Slow or Mixed. In Mixed decay mode, the device is set initially to a fast decay for a proportion of the fixed off-time, then to a slow decay for the remainder of the off-time. Mixed decay current control results in reduced audible motor noise, increased step accuracy, and reduced power dissipation.

Continued on the next page...

Typical Application Diagram



Description (continued)

Internal synchronous rectification control circuitry is provided to improve power dissipation during PWM operation. Internal circuit protection includes: thermal shutdown with hysteresis, undervoltage lockout (UVLO), and crossover-current protection. Special power-on sequencing is not required.

The A4988 is supplied in a surface mount QFN package (ES), 5 mm × 5 mm, with a nominal overall package height of 0.90 mm and an exposed pad for enhanced thermal dissipation. It is lead (Pb) free (suffix -T), with 100% matte tin plated leadframes.

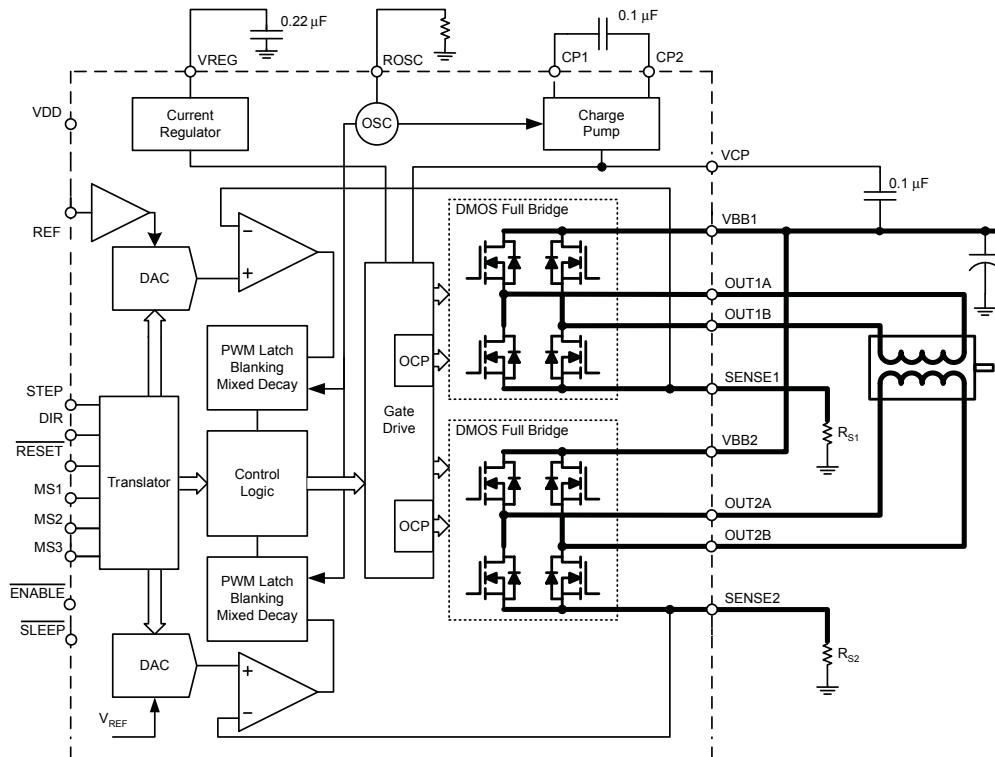
Selection Guide

Part Number	Package	Packing
A4988SETTR-T	28-contact QFN with exposed thermal pad	1500 pieces per 7-in. reel

Absolute Maximum Ratings

Characteristic	Symbol	Notes	Rating	Units
Load Supply Voltage	V_{BB}		35	V
Output Current	I_{OUT}		± 2	A
Logic Input Voltage	V_{IN}		-0.3 to 5.5	V
Logic Supply Voltage	V_{DD}		-0.3 to 5.5	V
Motor Outputs Voltage			-2.0 to 37	V
Sense Voltage	V_{SENSE}		-0.5 to 0.5	V
Reference Voltage	V_{REF}		5.5	V
Operating Ambient Temperature	T_A	Range S	-20 to 85	°C
Maximum Junction	$T_J(max)$		150	°C
Storage Temperature	T_{stg}		-55 to 150	°C

Functional Block Diagram



A4988

DMOS Microstepping Driver with Translator And Overcurrent Protection

ELECTRICAL CHARACTERISTICS¹ at $T_A = 25^\circ\text{C}$, $V_{BB} = 35\text{ V}$ (unless otherwise noted)

Characteristics	Symbol	Test Conditions	Min.	Typ. ²	Max.	Units
Output Drivers						
Load Supply Voltage Range	V_{BB}	Operating	8	–	35	V
Logic Supply Voltage Range	V_{DD}	Operating	3.0	–	5.5	V
Output On Resistance	$R_{DS(ON)}$	Source Driver, $I_{OUT} = -1.5\text{ A}$	–	320	430	$\text{m}\Omega$
		Sink Driver, $I_{OUT} = 1.5\text{ A}$	–	320	430	$\text{m}\Omega$
Body Diode Forward Voltage	V_F	Source Diode, $I_F = -1.5\text{ A}$	–	–	1.2	V
		Sink Diode, $I_F = 1.5\text{ A}$	–	–	1.2	V
Motor Supply Current	I_{BB}	$f_{PWM} < 50\text{ kHz}$	–	–	4	mA
		Operating, outputs disabled	–	–	2	mA
Logic Supply Current	I_{DD}	$f_{PWM} < 50\text{ kHz}$	–	–	8	mA
		Outputs off	–	–	5	mA
Control Logic						
Logic Input Voltage	$V_{IN(1)}$		$V_{DD} \times 0.7$	–	–	V
	$V_{IN(0)}$		–	–	$V_{DD} \times 0.3$	V
Logic Input Current	$I_{IN(1)}$	$V_{IN} = V_{DD} \times 0.7$	-20	<1.0	20	μA
	$I_{IN(0)}$	$V_{IN} = V_{DD} \times 0.3$	-20	<1.0	20	μA
Microstep Select	R_{MS1}	MS1 pin	–	100	–	$\text{k}\Omega$
	R_{MS2}	MS2 pin	–	50	–	$\text{k}\Omega$
	R_{MS3}	MS3 pin	–	100	–	$\text{k}\Omega$
Logic Input Hysteresis	$V_{HYS(IN)}$	As a % of V_{DD}	5	11	19	%
Blank Time	t_{BLANK}		0.7	1	1.3	μs
Fixed Off-Time	t_{OFF}	OSC = VDD or GND	20	30	40	μs
		$R_{OSC} = 25\text{ k}\Omega$	23	30	37	μs
Reference Input Voltage Range	V_{REF}		0	–	4	V
Reference Input Current	I_{REF}		-3	0	3	μA
Current Trip-Level Error ³	err_i	$V_{REF} = 2\text{ V}$, % $ I_{TripMAX} = 38.27\%$	–	–	± 15	%
		$V_{REF} = 2\text{ V}$, % $ I_{TripMAX} = 70.71\%$	–	–	± 5	%
		$V_{REF} = 2\text{ V}$, % $ I_{TripMAX} = 100.00\%$	–	–	± 5	%
Crossover Dead Time	t_{DT}		100	475	800	ns
Protection						
Overcurrent Protection Threshold ⁴	I_{OCPST}		2.1	–	–	A
Thermal Shutdown Temperature	T_{TSD}		–	165	–	$^\circ\text{C}$
Thermal Shutdown Hysteresis	T_{TSDHYS}		–	15	–	$^\circ\text{C}$
VDD Undervoltage Lockout	V_{DDUVLO}	V_{DD} rising	2.7	2.8	2.9	V
VDD Undervoltage Hysteresis	$V_{DDUVLOHYS}$		–	90	–	mV

¹For input and output current specifications, negative current is defined as coming out of (sourcing) the specified device pin.

²Typical data are for initial design estimations only, and assume optimum manufacturing and application conditions. Performance may vary for individual units, within the specified maximum and minimum limits.

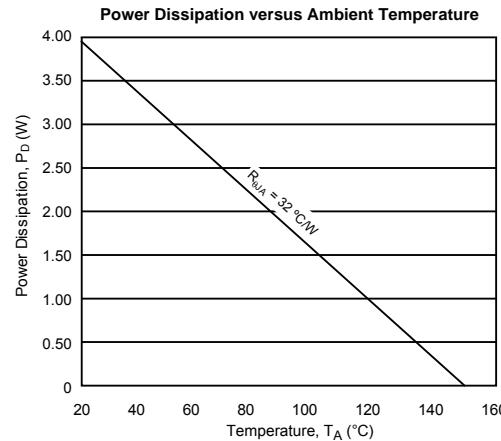
$$\text{V}_{\text{ERR}} = [(V_{\text{REF}}/8) - V_{\text{SENSE}}] / (V_{\text{REF}}/8).$$

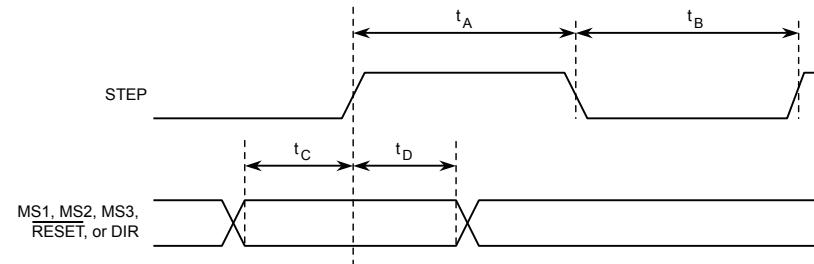
³Overcurrent protection (OCP) is tested at $T_A = 25^\circ\text{C}$ in a restricted range and guaranteed by characterization.

THERMAL CHARACTERISTICS

Characteristic	Symbol	Test Conditions*	Value	Units
Package Thermal Resistance	R_{QJA}	Four-layer PCB, based on JEDEC standard	32	$^{\circ}\text{C/W}$

*Additional thermal information available on Allegro Web site.





Time Duration	Symbol	Typ.	Unit
STEP minimum, HIGH pulse width	t_A	1	μs
STEP minimum, LOW pulse width	t_B	1	μs
Setup time, input change to STEP	t_C	200	ns
Hold time, input change to STEP	t_D	200	ns

Figure 1. Logic Interface Timing Diagram

Table 1. Microstepping Resolution Truth Table

MS1	MS2	MS3	Microstep Resolution	Excitation Mode
L	L	L	Full Step	2 Phase
H	L	L	Half Step	1-2 Phase
L	H	L	Quarter Step	W1-2 Phase
H	H	L	Eighth Step	2W1-2 Phase
H	H	H	Sixteenth Step	4W1-2 Phase

Functional Description

Device Operation. The A4988 is a complete microstepping motor driver with a built-in translator for easy operation with minimal control lines. It is designed to operate bipolar stepper motors in full-, half-, quarter-, eighth, and sixteenth-step modes. The currents in each of the two output full-bridges and all of the N-channel DMOS FETs are regulated with fixed off-time PWM (pulse width modulated) control circuitry. At each step, the current for each full-bridge is set by the value of its external current-sense resistor (R_{S1} and R_{S2}), a reference voltage (V_{REF}), and the output voltage of its DAC (which in turn is controlled by the output of the translator).

At power-on or reset, the translator sets the DACs and the phase current polarity to the initial Home state (shown in figures 8 through 12), and the current regulator to Mixed Decay Mode for both phases. When a step command signal occurs on the STEP input, the translator automatically sequences the DACs to the next level and current polarity. (See table 2 for the current-level sequence.) The microstep resolution is set by the combined effect of the MSx inputs, as shown in table 1.

When stepping, if the new output levels of the DACs are lower than their previous output levels, then the decay mode for the active full-bridge is set to Mixed. If the new output levels of the DACs are higher than or equal to their previous levels, then the decay mode for the active full-bridge is set to Slow. This automatic current decay selection improves microstepping performance by reducing the distortion of the current waveform that results from the back EMF of the motor.

Microstep Select (MSx). The microstep resolution is set by the voltage on logic inputs MSx, as shown in table 1. The MS1 and MS3 pins have a 100 k Ω pull-down resistance, and the MS2 pin has a 50 k Ω pull-down resistance. When changing the step mode the change does not take effect until the next STEP rising edge.

If the step mode is changed without a translator reset, and absolute position must be maintained, it is important to change the step mode at a step position that is common to both step modes in order to avoid missing steps. When the device is powered down, or reset due to TSD or an over current event the translator is set to the home position which is by default common to all step modes.

Mixed Decay Operation. The bridge operates in Mixed decay mode, at power-on and reset, and during normal running according to the ROSC configuration and the step sequence, as shown in figures 8 through 12. During Mixed decay, when the trip point is reached, the A4988 initially goes into a fast decay mode for 31.25% of the off-time, t_{OFF} . After that, it switches to Slow decay mode for the remainder of t_{OFF} . A timing diagram for this feature appears on the next page.

Typically, mixed decay is only necessary when the current in the winding is going from a higher value to a lower value as determined by the state of the translator. For most loads automatically-selected mixed decay is convenient because it minimizes ripple when the current is rising and prevents missed steps when the current is falling. For some applications where microstepping at very low speeds is necessary, the lack of back EMF in the winding causes the current to increase in the load quickly, resulting in missed steps. This is shown in figure 2. By pulling the ROSC pin to ground, mixed decay is set to be active 100% of the time, for both rising and falling currents, and prevents missed steps as shown in figure 3. If this is not an issue, it is recommended that automatically-selected mixed decay be used, because it will produce reduced ripple currents. Refer to the Fixed Off-Time section for details.

Low Current Microstepping. Intended for applications where the minimum on-time prevents the output current from regulating to the programmed current level at low current steps. To prevent this, the device can be set to operate in Mixed decay mode on both rising and falling portions of the current waveform. This feature is implemented by shorting the ROSC pin to ground. In this state, the off-time is internally set to 30 μ s.

Reset Input (RESET). The \overline{RESET} input sets the translator to a predefined Home state (shown in figures 8 through 12), and turns off all of the FET outputs. All STEP inputs are ignored until the RESET input is set to high.

Step Input (STEP). A low-to-high transition on the STEP input sequences the translator and advances the motor one increment. The translator controls the input to the DACs and the direc-

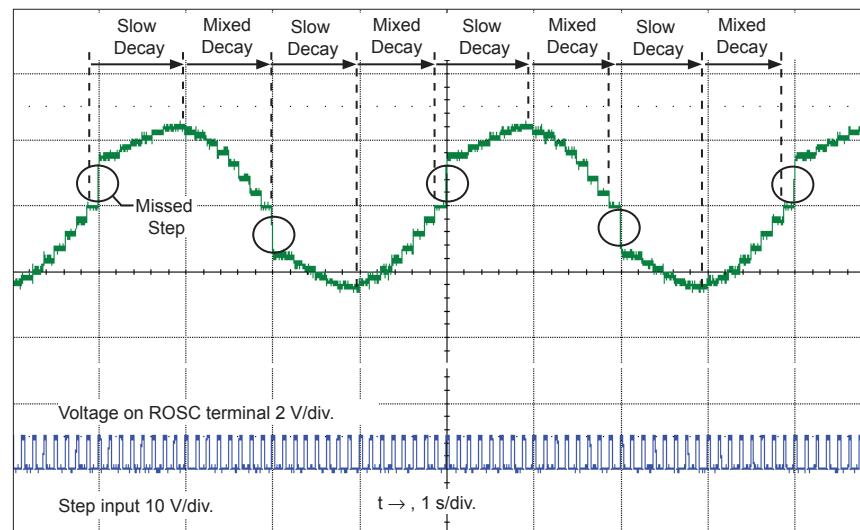


Figure 2. Missed steps in low-speed microstepping

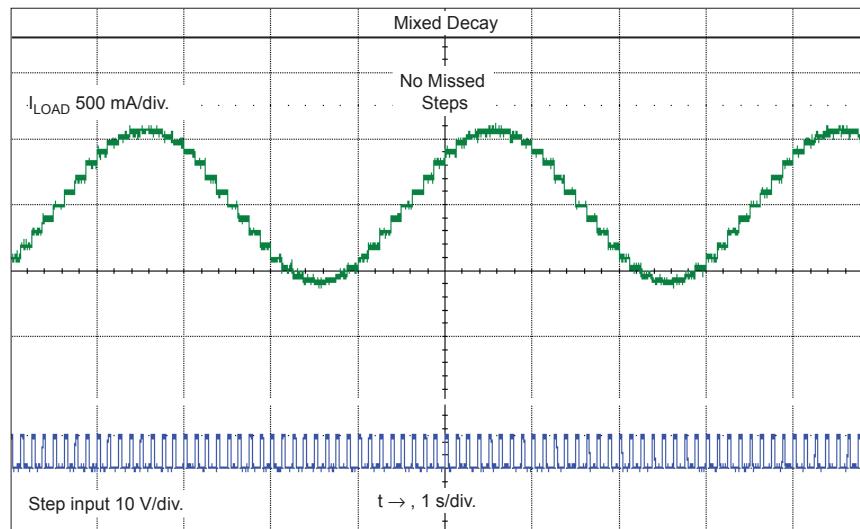


Figure 3. Continuous stepping using automatically-selected mixed stepping (ROSC pin grounded)

tion of current flow in each winding. The size of the increment is determined by the combined state of the MSx inputs.

Direction Input (DIR). This determines the direction of rotation of the motor. Changes to this input do not take effect until the next STEP rising edge.

Internal PWM Current Control. Each full-bridge is controlled by a fixed off-time PWM current control circuit that limits the load current to a desired value, I_{TRIP} . Initially, a diagonal pair of source and sink FET outputs are enabled and current flows through the motor winding and the current sense resistor, R_{SX} . When the voltage across R_{SX} equals the DAC output voltage, the current sense comparator resets the PWM latch. The latch then turns off the appropriate source driver and initiates a fixed off time decay mode

The maximum value of current limiting is set by the selection of R_{SX} and the voltage at the VREF pin. The transconductance function is approximated by the maximum value of current limiting, $I_{TripMAX}$ (A), which is set by

$$I_{TripMAX} = V_{REF} / (8 \times R_S)$$

where R_S is the resistance of the sense resistor (Ω) and V_{REF} is the input voltage on the REF pin (V).

The DAC output reduces the V_{REF} output to the current sense comparator in precise steps, such that

$$I_{trip} = (%I_{TripMAX} / 100) \times I_{TripMAX}$$

(See table 2 for % $I_{TripMAX}$ at each step.)

It is critical that the maximum rating (0.5 V) on the SENSE1 and SENSE2 pins is not exceeded.

Fixed Off-Time. The internal PWM current control circuitry uses a one-shot circuit to control the duration of time that the DMOS FETs remain off. The off-time, t_{OFF} , is determined by the ROSC terminal. The ROSC terminal has three settings:

- ROSC tied to VDD — off-time internally set to 30 μ s, decay mode is automatic Mixed decay except when in full step where decay mode is set to Slow decay
- ROSC tied directly to ground — off-time internally set to 30 μ s, current decay is set to Mixed decay for both increasing and decreasing currents for all step modes.

- ROSC through a resistor to ground — off-time is determined by the following formula, the decay mode is automatic Mixed decay for all step modes.

$$t_{OFF} \approx R_{OSC} / 825$$

Where t_{OFF} is in μ s.

Blanking. This function blanks the output of the current sense comparators when the outputs are switched by the internal current control circuitry. The comparator outputs are blanked to prevent false overcurrent detection due to reverse recovery currents of the clamp diodes, and switching transients related to the capacitance of the load. The blank time, t_{BLANK} (μ s), is approximately

$$t_{BLANK} \approx 1 \mu\text{s}$$

Shorted-Load and Short-to-Ground Protection.

If the motor leads are shorted together, or if one of the leads is shorted to ground, the driver will protect itself by sensing the overcurrent event and disabling the driver that is shorted, protecting the device from damage. In the case of a short-to-ground, the device will remain disabled (latched) until the \overline{SLEEP} input goes high or VDD power is removed. A short-to-ground overcurrent event is shown in figure 4.

When the two outputs are shorted together, the current path is through the sense resistor. After the blanking time ($\approx 1 \mu$ s) expires, the sense resistor voltage is exceeding its trip value, due to the overcurrent condition that exists. This causes the driver to go into a fixed off-time cycle. After the fixed off-time expires the driver turns on again and the process repeats. In this condition the driver is completely protected against overcurrent events, but the short is repetitive with a period equal to the fixed off-time of the driver. This condition is shown in figure 5.

During a shorted load event it is normal to observe both a positive and negative current spike as shown in figure 3, due to the direction change implemented by the Mixed decay feature. This is shown in figure 6. In both instances the overcurrent circuitry is protecting the driver and prevents damage to the device.

Charge Pump (CP1 and CP2). The charge pump is used to generate a gate supply greater than that of VBB for driving the source-side FET gates. A 0.1 μ F ceramic capacitor, should be connected between CP1 and CP2. In addition, a 0.1 μ F ceramic capacitor is required between VCP and VBB, to act as a reservoir for operating the high-side FET gates.

Capacitor values should be Class 2 dielectric $\pm 15\%$ maximum, or tolerance R, according to EIA (Electronic Industries Alliance) specifications.

V_{REG} (VREG). This internally-generated voltage is used to operate the sink-side FET outputs. The nominal output voltage of the VREG terminal is 7 V. The VREG pin must be decoupled with a 0.22 μ F ceramic capacitor to ground. V_{REG} is internally monitored. In the case of a fault condition, the FET outputs of the A4988 are disabled.

Capacitor values should be Class 2 dielectric $\pm 15\%$ maximum, or tolerance R, according to EIA (Electronic Industries Alliance) specifications.

Enable Input (ENABLE). This input turns on or off all of the FET outputs. When set to a logic high, the outputs are disabled. When set to a logic low, the internal control enables the outputs as required. The translator inputs STEP, DIR, and MSx, as well as the internal sequencing logic, all remain active, independent of the ENABLE input state.

Shutdown. In the event of a fault, overtemperature (excess T_J) or an undervoltage (on VCP), the FET outputs of the A4988 are disabled until the fault condition is removed. At power-on, the UVLO (undervoltage lockout) circuit disables the FET outputs and resets the translator to the Home state.

Sleep Mode (SLEEP). To minimize power consumption when the motor is not in use, this input disables much of the internal circuitry including the output FETs, current regulator, and charge pump. A logic low on the SLEEP pin puts the A4988 into Sleep mode. A logic high allows normal operation, as well as start-up (at which time the A4988 drives the motor to the Home microstep position). When emerging from Sleep mode, in order to allow the charge pump to stabilize, provide a delay of 1 ms before issuing a Step command.

Mixed Decay Operation. The bridge operates in Mixed Decay mode, depending on the step sequence, as shown in figures 8 through 12. As the trip point is reached, the A4988 initially goes into a fast decay mode for 31.25% of the off-time, t_{OFF}. After that, it switches to Slow Decay mode for the remainder of t_{OFF}. A timing diagram for this feature appears in figure 7.

Synchronous Rectification. When a PWM-off cycle is triggered by an internal fixed-off time cycle, load current recirculates according to the decay mode selected by the control logic. This synchronous rectification feature turns on the appropriate FETs during current decay, and effectively shorts out the body diodes with the low FET R_{DS(ON)}. This reduces power dissipation significantly, and can eliminate the need for external Schottky diodes in many applications. Synchronous rectification turns off when the load current approaches zero (0 A), preventing reversal of the load current.

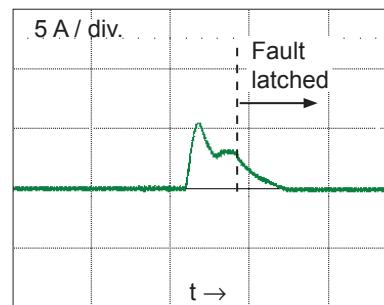


Figure 4. Short-to-ground event

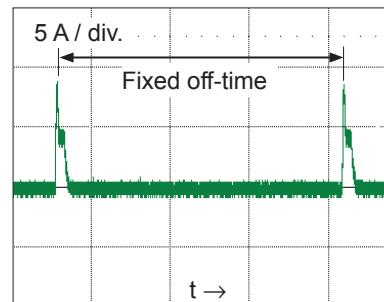


Figure 5. Shorted load (OUTxA → OUTxB) in Slow decay mode

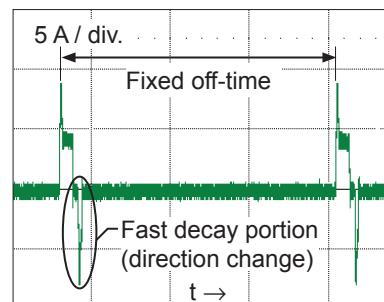
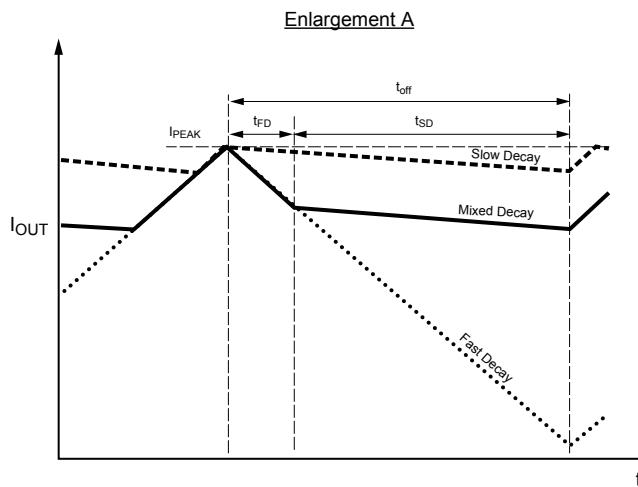
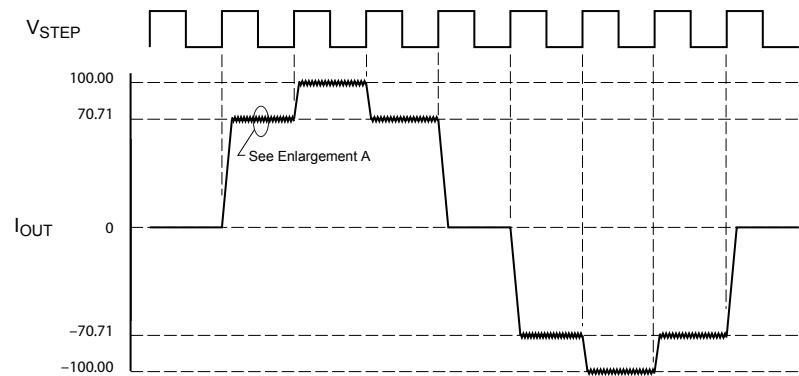


Figure 6. Shorted load (OUTxA → OUTxB) in Mixed decay mode



Symbol	Characteristic
t_{off}	Device fixed off-time
I_{PEAK}	Maximum output current
t_{SD}	Slow decay interval
t_{FD}	Fast decay interval
I_{OUT}	Device output current

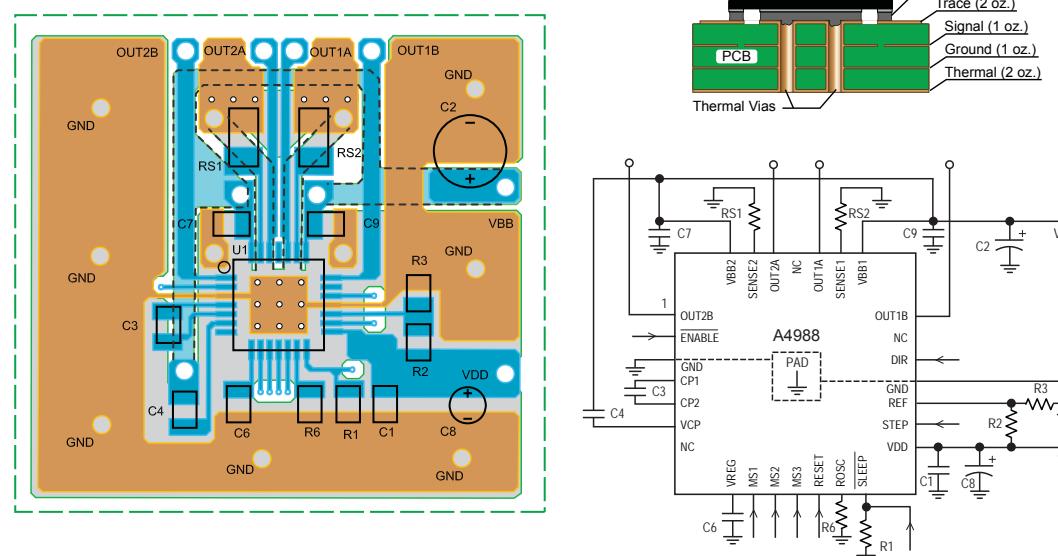
Figure 7. Current Decay Modes Timing Chart

Application Layout

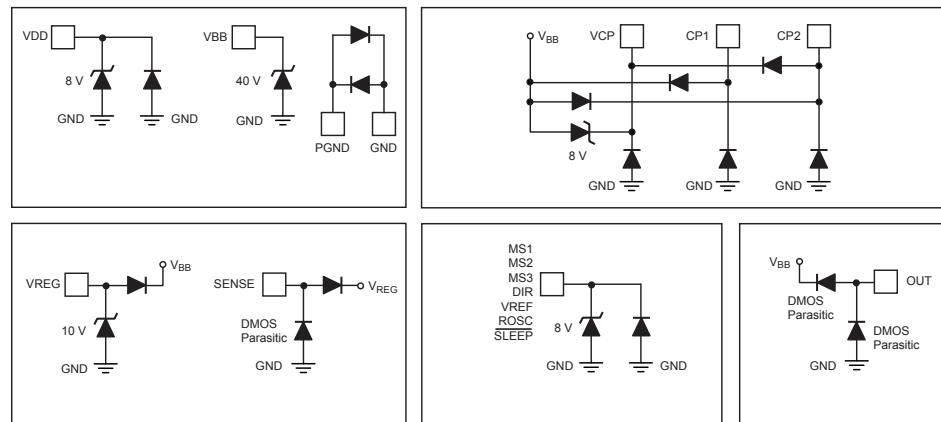
Layout. The printed circuit board should use a heavy ground-plane. For optimum electrical and thermal performance, the A4988 must be soldered directly onto the board. Pins 3 and 18 are internally fused, which provides a path for enhanced thermal dissipation. These pins should be soldered directly to an exposed surface on the PCB that connects to thermal vias are used to transfer heat to other layers of the PCB.

In order to minimize the effects of ground bounce and offset issues, it is important to have a low impedance single-point ground, known as a *star ground*, located very close to the device. By making the connection between the pad and the ground plane directly under the A4988, that area becomes an ideal location for a star ground point. A low impedance ground will prevent ground bounce during high current operation and ensure that the supply voltage remains stable at the input terminal.

The two input capacitors should be placed in parallel, and as close to the device supply pins as possible. The ceramic capacitor (CIN1) should be closer to the pins than the bulk capacitor (CIN2). This is necessary because the ceramic capacitor will be responsible for delivering the high frequency current components. The sense resistors, RS_x, should have a very low impedance path to ground, because they must carry a large current while supporting very accurate voltage measurements by the current sense comparators. Long ground traces will cause additional voltage drops, adversely affecting the ability of the comparators to accurately measure the current in the windings. The SENSE_x pins have very short traces to the RS_x resistors and very thick, low impedance traces directly to the star ground underneath the device. If possible, there should be no other components on the sense circuits.



Pin Circuit Diagrams



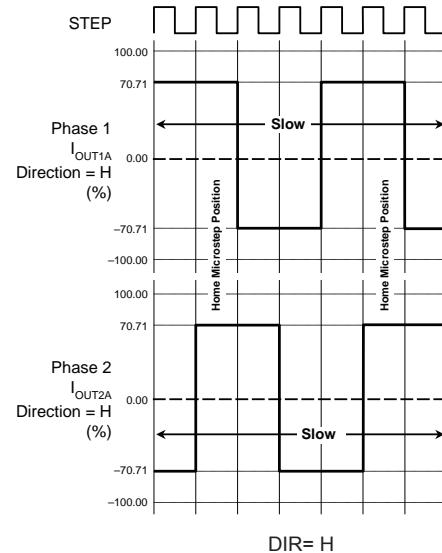


Figure 8. Decay Mode for Full-Step Increments

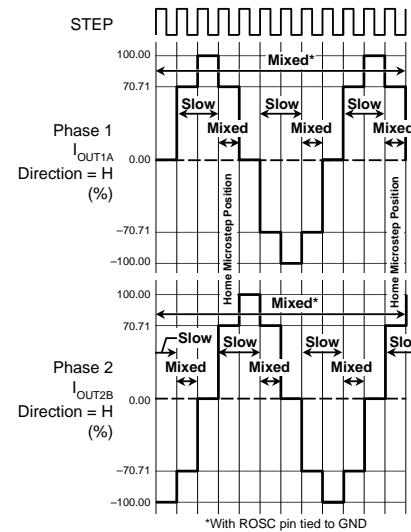


Figure 9. Decay Modes for Half-Step Increments

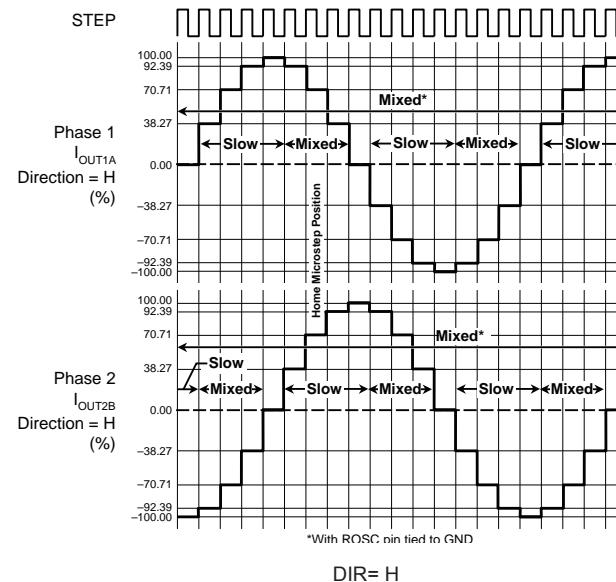


Figure 10. Decay Modes for Quarter-Step Increments

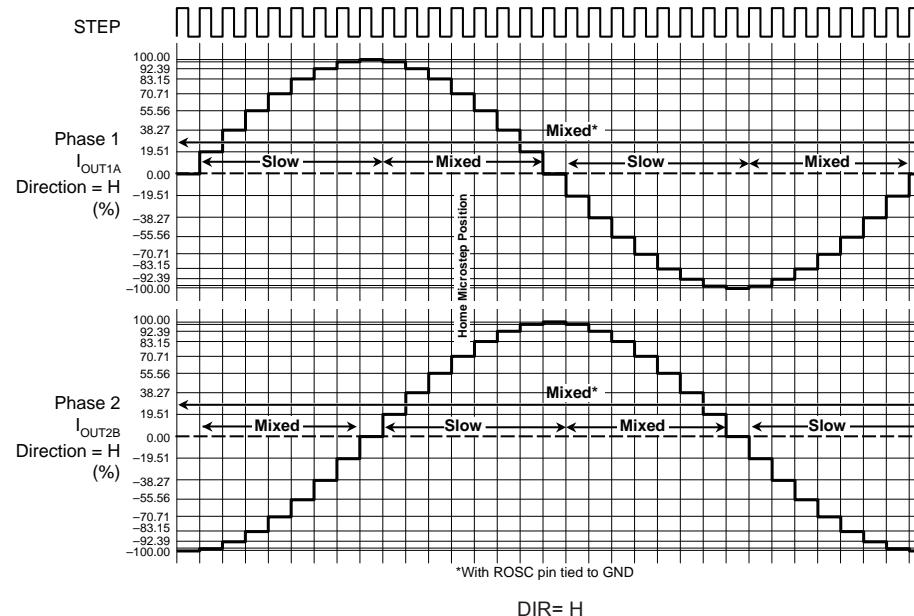


Figure 11. Decay Modes for Eighth-Step Increments

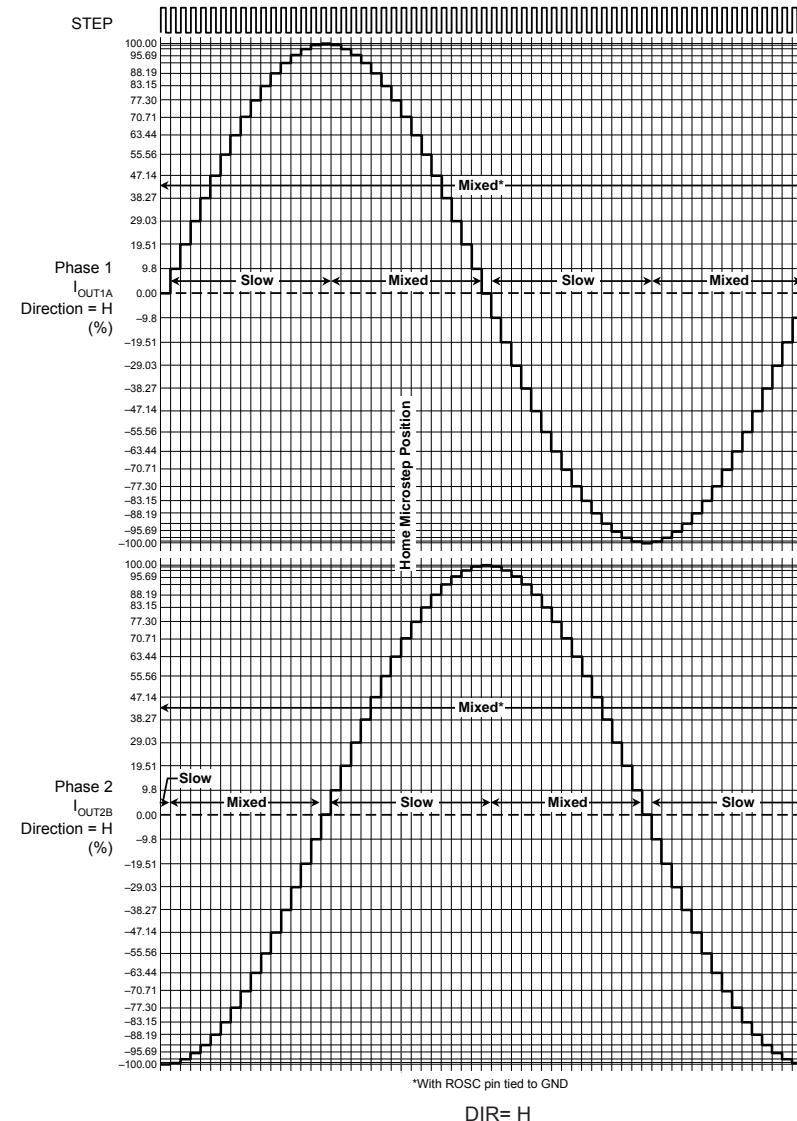


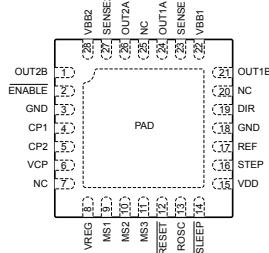
Figure 12. Decay Modes for Sixteenth-Step Increments

Table 2. Step Sequencing Settings

Home microstep position at Step Angle 45°; DIR = H

Full Step #	Half Step #	1/4 Step #	1/8 Step #	1/16 Step #	Phase 1 Current [% I _{tripMax}] (%)	Phase 2 Current [% I _{tripMax}] (%)	Step Angle (°)	Full Step #	Half Step #	1/4 Step #	1/8 Step #	1/16 Step #	Phase 1 Current [% I _{tripMax}] (%)	Phase 2 Current [% I _{tripMax}] (%)	Step Angle (°)	
1	1	2	1	1	100.00	0.00	0.0	5	9	17	33	-100.00	0.00	180.0		
			2	2	99.52	9.80	5.6			34	-99.52	-9.80	-9.80	185.6		
			2	3	98.08	19.51	11.3			18	35	-98.08	-19.51	191.3		
				4	95.69	29.03	16.9			36	-95.69	-29.03	-29.03	196.9		
			2	3	92.39	38.27	22.5			10	19	37	-92.39	-38.27	202.5	
				5	88.19	47.14	28.1			38	-88.19	-47.14	-47.14	208.1		
				6	83.15	55.56	33.8			20	39	-83.15	-55.56	213.8		
				7	77.30	63.44	39.4			40	-77.30	-63.44	-63.44	219.4		
1	2	3	5	9	70.71	70.71	45.0	3	6	11	21	41	-70.71	-70.71	225.0	
				10	63.44	77.30	50.6			42	-63.44	-77.30	-77.30	230.6		
				6	55.56	83.15	56.3			22	43	-55.56	-83.15	236.3		
				11	47.14	88.19	61.9			44	-47.14	-88.19	-88.19	241.9		
				12	38.27	92.39	67.5			12	23	45	-38.27	-92.39	247.5	
				13	29.03	95.69	73.1			46	-29.03	-95.69	-95.69	253.1		
				14	19.51	98.08	78.8			24	47	-19.51	-98.08	258.8		
				15	9.80	99.52	84.4			48	-9.80	-99.52	-99.52	264.4		
			3	5	0.00	100.00	90.0			7	13	25	0.00	-100.00	270.0	
				9	-9.80	99.52	95.6				50	9.80	-99.52	-99.52	275.6	
				17	-19.51	98.08	101.3				26	51	19.51	-98.08	281.3	
				18	-29.03	95.69	106.9				52	29.03	-95.69	-95.69	286.9	
				19	-38.27	92.39	112.5				14	27	53	38.27	-92.39	292.5
				21	-47.14	88.19	118.1				54	47.14	-88.19	-88.19	298.1	
				22	-55.56	83.15	123.8				28	55	55.56	-83.15	303.8	
				23	-63.44	77.30	129.4				56	63.44	-77.30	-77.30	309.4	
2	4	7	13	25	-70.71	70.71	135.0	4	8	15	29	57	70.71	-70.71	315.0	
				26	-77.30	63.44	140.6				58	77.30	-63.44	-63.44	320.6	
				27	-83.15	55.56	146.3				30	59	83.15	-55.56	326.3	
				28	-88.19	47.14	151.9				60	88.19	-47.14	-47.14	331.9	
				29	-92.39	38.27	157.5				16	31	61	92.39	-38.27	337.5
				30	-95.69	29.03	163.1				62	95.69	-29.03	-29.03	343.1	
				31	-98.08	19.51	168.8				32	63	98.08	-19.51	348.8	
				32	-99.52	9.80	174.4				64	99.52	-9.80	-9.80	354.4	

Pin-out Diagram

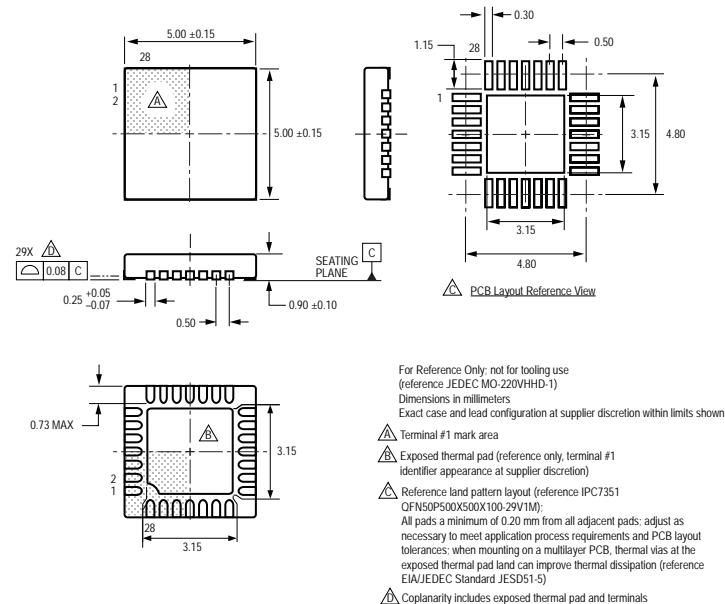


Terminal List Table

Name	Number	Description
CP1	4	Charge pump capacitor terminal
CP2	5	Charge pump capacitor terminal
VCP	6	Reservoir capacitor terminal
VREG	8	Regulator decoupling terminal
MS1	9	Logic input
MS2	10	Logic input
MS3	11	Logic input
<u>RESET</u>	12	Logic input
ROSC	13	Timing set
<u>SLEEP</u>	14	Logic input
VDD	15	Logic supply
STEP	16	Logic input
REF	17	G_m reference voltage input
GND	3, 18	Ground*
DIR	19	Logic input
OUT1B	21	DMOS Full Bridge 1 Output B
VBB1	22	Load supply
SENSE1	23	Sense resistor terminal for Bridge 1
OUT1A	24	DMOS Full Bridge 1 Output A
OUT2A	26	DMOS Full Bridge 2 Output A
SENSE2	27	Sense resistor terminal for Bridge 2
VBB2	28	Load supply
OUT2B	1	DMOS Full Bridge 2 Output B
<u>ENABLE</u>	2	Logic input
NC	7, 20, 25	No connection
PAD	-	Exposed pad for enhanced thermal dissipation*

*The GND pins must be tied together externally by connecting to the PAD ground plane under the device.

ET Package, 28-Pin QFN with Exposed Thermal Pad



Revision History

Revision	Revision Date	Description of Revision
Rev. 4	January 27, 2012	Update I_{OCPST}

Copyright ©2009-2012, Allegro MicroSystems, Inc.

Allegro MicroSystems, Inc. reserves the right to make, from time to time, such departures from the detail specifications as may be required to permit improvements in the performance, reliability, or manufacturability of its products. Before placing an order, the user is cautioned to verify that the information being relied upon is current.

Allegro's products are not to be used in life support devices or systems, if a failure of an Allegro product can reasonably be expected to cause the failure of that life support device or system, or to affect the safety or effectiveness of that device or system.

The information included herein is believed to be accurate and reliable. However, Allegro MicroSystems, Inc. assumes no responsibility for its use; nor for any infringement of patents or other rights of third parties which may result from its use.

For the latest version of this document, visit our website:
www.allegromicro.com

