UNIVERSIDADE FEDERAL DE VIÇOSA
CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

FELIPE MATIAS DA SILVA NORONHA

Desenvolvimento de um sistema de controle de velocidade para acionamento de motores de corrente contínua sem escovas

VIÇOSA
2023

FELIPE MATIAS DA SILVA NORONHA

Desenvolvimento de um sistema de controle de velocidade para acionamento de motores de corrente contínua sem escovas

Monografia apresentada ao Departamento de Engenharia Elétrica do Centro de Ciências Exatas e Tecnológicas da Universidade Federal de Viçosa, para a obtenção dos créditos da disciplina ELT 402 – Projeto de Engenharia II – e cumprimento do requisito parcial para obtenção do grau de Bacharel em Engenharia Elétrica.

Orientador: Prof. Heverton Augusto Pereira

VIÇOSA
2023

FELIPE MATIAS DA SILVA NORONHA

Desenvolvimento de um circuito ESC (Electronic Speed Control) para acionamento de motores de corrente contínua sem escovas (BLDC)

Monografia apresentada ao Departamento de Engenharia Elétrica do Centro de Ciências Exatas e Tecnológicas da Universidade Federal de Viçosa, para a obtenção dos créditos da disciplina ELT 490 – Monografia e Seminário – e cumprimento do requisito parcial para obtenção do grau de Bacharel em Engenharia Elétrica.

Aprovada em 07 de julho de 2023.

COMISSÃO EXAMINADORA

gov.br Documento assinado digitalmente
HEVERTON AUGUSTO PEREIRA
Data: 07/07/2023 17:53:05-0300
Verifique em https://validar.iti.gov.br

Prof. Dr. Heverton Augusto Pereira
Universidade Federal de Viçosa

gov.br Documento assinado digitalmente
DIUARY GONCALVES
Data: 08/07/2023 10:42:23-0300
Verifique em https://validar.iti.gov.br

Eng. Diuary Gonçalves - Membro
Centro Federal de Educação Tecnológica de Minas Gerais

gov.br Documento assinado digitalmente
JOAO VICTOR GUIMARAES FRANCA
Data: 07/07/2023 19:04:28-0300
Verifique em https://validar.iti.gov.br

Eng. João Victor Guimarães França - Membro
Centro Federal de Educação Tecnológica de Minas Gerais

“Se você quer encontrar os segredos do universo, pense em termos de energia, frequência e vibração.”

(Nikola Tesla)

A minha família, meu bem maior.

Agradecimentos

Agradeço em primeiro lugar a Deus e minha família por sempre estarem ao meu lado me dando apoio para continuar e seguir sempre na realização dos meus sonhos.

Agradeço aos meus amigos de curso, com menção especial aos do grupo Mama nós e Malhas, por todos os momentos que passamos juntos, sempre um ajudando o outro a encarar os desafios da graduação em Engenharia Elétrica, assim como os da vida morando longe de casa e da família. Foram tantos os feriados, finais de semana, madrugadas que passamos estudando juntos, mas unidos com momentos de descontração e risadas, o que tornava tudo muito mais leve.

Um grande agradecimento aos amigos do Cool do Charmois por todos os momentos e aventuras vividas durante o nosso intercâmbio.

Gostaria de agradecer aos professores da UFV por todo o conhecimento compartilhado e que me dão uma base sólida e destaque para competir no mercado de trabalho. Agradeço também aos professores da ENSEM, pelo acolhimento em sua universidade, a paciência por explicar as coisas complexas para um aluno internacional, os conselhos para conseguir um estágio, sempre dispostos a ajudar e me auxiliando a abrir as portas para o meu crescimento.

Agradeço aos meus chefes do meu primeiro estágio, Roland e Laurent, na Ian Motion, que me ajudaram na execução desse trabalho. Fui muito bem acolhido por eles que se tornaram grandes amigos meus, me dando todo o suporte no período do estágio, momentos de descontração, de conhecer coisas novas da França e compartilhando a experiência profissional deles comigo. Me trouxeram uma boa visão sobre o que é ser um engenheiro, me mostrando a importância de ser curioso e de compreender bem cada etapa de um projeto, sendo sempre atencioso, testando cada etapa e verificando seu funcionamento além de sempre deixar tudo registrado para não que informações não sejam perdidas.

Por fim, gostaria de agradecer a CAPES pelo apoio financeiro concedido, que através do programa BRAFITEC, possibilitou que eu tivesse uma das melhores experiências de crescimento profissional e pessoal através do intercâmbio realizado na França.

Resumo

O presente trabalho abordou diferentes etapas relacionadas ao estudo e desenvolvimento de circuitos de controle eletrônico de velocidade (ESC, *Electronic Speed Controller*) para motores de corrente contínua sem escovas (BLDC, *Brushless Direct Current*). Inicialmente, foi realizada uma análise do circuito de *bootstrap* para o acionamento dos MOSFETs em configuração *high-side*, visando melhorar a eficiência e o desempenho do acionamento dos motores. Em seguida, foi confeccionado o primeiro protótipo, no qual um motor BLDC de drone foi acionado sem a utilização de sensores de efeito Hall, aproveitando o fenômeno de *back electromotive force* (*back EMF)* para realizar as comutações.

Posteriormente, na segunda etapa foi utilizado sensores de efeito Hall para controle de um motor BLDC. Esses sensores foram integrados ao circuito ESC, permitindo uma maior precisão no controle da velocidade e direção do motor. Além disso, na terceira etapa, foram estudadas estratégias para a realização da frenagem regenerativa, aproveitando a energia gerada durante a desaceleração para recarregar a bateria ou fornecer energia para outros componentes do sistema.

Durante todas as etapas, foram realizados testes e análises dos resultados obtidos, buscando aprimorar o desempenho e a eficiência do sistema de acionamento dos motores BLDC. Ao final do trabalho, foi possível observar que o acionamento dos motores tanto no modo *sensorless* como no modo *sensored*, bem como a implementação da frenagem regenerativa, trouxeram benefícios significativos, como maior eficiência energética, maior controle e menor desgaste dos componentes.

O estudo realizado neste trabalho contribui para o avanço da tecnologia de acionamento de motores BLDC, especialmente no contexto da mobilidade elétrica, onde a eficiência e a recuperação de energia são aspectos fundamentais. A partir das estratégias desenvolvidas e dos resultados obtidos, novas oportunidades de pesquisa e aprimoramentos surgem, como a otimização dos códigos de controle, a utilização de microcontroladores mais robustos e o projeto de circuitos de proteção adicionais. Essas melhorias podem impactar positivamente o desempenho e a aplicação dos circuitos ESC em diversas áreas, contribuindo para a evolução da mobilidade elétrica e seus benefícios ambientais e tecnológicos.

Abstract

The present study encompassed various stages related to the study and development of Electronic Speed Control (ESC) circuits for Brushless DC (BLDC) motors. Initially, an analysis of the bootstrap circuit for driving the high-side MOSFETs was conducted, aiming to improve the motor control efficiency and performance. Subsequently, the first prototype was constructed, where a drone's BLDC motor was driven without the use of Hall effect sensors, utilizing the back electromotive force (back EMF) phenomenon for commutations.

Further progress was made in implementing a BLDC motor using Hall effect sensors. These sensors were integrated into the ESC circuit, enabling greater precision in controlling motor speed and direction. Additionally, strategies for regenerative braking were studied, harnessing the energy generated during deceleration to recharge the battery or provide power to other system components.

Throughout the entire process, tests and analyses of the obtained results were carried out, aiming to enhance the performance and efficiency of the BLDC motor control system. Ultimately, it was observed that both sensorless and sensored motor control, as well as the implementation of regenerative braking, yielded significant benefits, such as increased energy efficiency, improved control, and reduced component wear.

This study contributes to the advancement of BLDC motor drive technology, particularly in the context of electric mobility, where efficiency and energy recovery are fundamental aspects. Based on the developed strategies and obtained results, new research opportunities and enhancements emerge, such as optimizing control codes, utilizing more robust microcontrollers, and designing additional protection circuits. These improvements can positively impact the performance and application of ESC circuits in various fields, contributing to the evolution of electric mobility and its environmental and technological benefits.

Sumário

Lista de Figuras

Lista de Tabelas

1 Introdução do Trabalho

Os circuitos ESC (*Electronic Speed Controller*), ou controladores eletrônicos de velocidade, são componentes essenciais em veículos e equipamentos controlados eletronicamente, como drones, carros de controle remoto, aeromodelos, patinetes elétricos e robôs. Esses circuitos têm a função de controlar a velocidade e direção de motores elétricos, convertendo o sinal de controle em sinais elétricos apropriados para o motor.

A mobilidade elétrica tem ganhado destaque como uma solução sustentável para reduzir a dependência de combustíveis fósseis e mitigar os impactos ambientais causados pelos veículos convencionais. Nesse contexto, os motores elétricos BLDC (*Brushless Direct Current*) têm se mostrado uma escolha popular devido à sua eficiência, confiabilidade e controle preciso da velocidade e torque. Os motores de corrente contínua sem escovas (BLDC), foram inventados em 1962 por T. G. Wilson e P. H. Trickey [1], aumentando a vida útil, eficiência e relação potência-peso, ao mesmo tempo em que reduzem a necessidade de manutenção em comparação com os motores de corrente contínua com escovas.

Isso abriu muitas novas aplicações para motores elétricos. Um campo em crescimento para os motores BLDC é em veículos aéreos pequenos não tripulados (UAVs, *Unmanned Aerial Vehicles*). Na maioria das aplicações usadas atualmente, o motor é otimizado para peso ou eficiência. No entanto, em veículos elétricos, como UAVs, a otimização tanto do peso quanto da eficiência é essencial para a viabilidade do projeto. Além disso, veículos elétricos frequentemente requerem aceleração total para decolagem e subida, mas redução significativa de potência durante o voo de cruzeiro. Em um sistema de eletrônica de potência, a aceleração total pode ter uma eficiência acima de 90%, mas em regime de baixa potência, a eficiência pode diminuir consideravelmente. Considerando que o voo de cruzeiro pode durar até 99% ou mais do tempo da missão, uma melhoria na eficiência do voo de cruzeiro aumentaria significativamente a autonomia do UAV. Essa aplicação em crescimento reabriu e estimulou novas pesquisas em sistemas de eletrônica de potência para aumentar a eficiência dos motores de corrente contínua sem escovas [2].

Em uma correspondência com a *Castle Creations*, um grande fabricante de ESC para entusiastas, eles responderam: "A eficiência geral dos nossos ESCs varia entre 98,5% e 99,5%. Baixas rotações resultam em eficiências mais baixas, enquanto altas rotações chegam

perto de 100% de eficiência." (Bernie Wolfard, 2014 apud Green, C. R., & McDonald, R. A., 2015, p. 3191.)

Um motor CC (corrente contínua) convencional utiliza de escovas de contato elétrico deslizando sobre comutadores eletromecânicos para energizar os campos magnéticos. O estator é um imã permanente, o rotor tem os enrolamentos, que são excitados por uma corrente. A corrente no rotor é invertida para criar um campo elétrico rotativo ou móvel por meio de um comutador dividido e com escovas. Por outro lado, em um motor BLDC, os enrolamentos estão no estator e o rotor é um imã permanente. Para girar o rotor, deve haver um campo elétrico rotativo. Como não possuem escovas, a comutação nos motores BLDC é feita eletronicamente [3].

Entre as vantagens dos Motores BLDC está sua alta eficiência, a capacidade de eliminar desgastes e perdas de energia, manutenção reduzida e muitos outros benefícios em comparação aos motores CC escovados e motores de indução, incluindo melhor velocidade em comparação ao torque, ausência de ionização do comutador, menor interferência eletromagnética (EMI), resposta dinâmica mais rápida, operação silenciosa e maiores faixas de velocidade [4].

Os circuitos ESC são essenciais para o funcionamento adequado dos motores BLDC, pois controlam a velocidade, direção e eficiência desses motores. Eles são responsáveis por converter a energia fornecida pela fonte de alimentação em sinais adequados para os motores. Esses circuitos utilizam algoritmos de controle avançados, como o controle PWM (*Pulse Width Modulation*), para variar a velocidade do motor.

A modulação por largura de pulso (PWM, *Pulse Width Modulation*) é uma técnica amplamente utilizada para controlar a velocidade e direção de motores elétricos. Nessa técnica, o sinal de controle é convertido em uma série de pulsos com largura variável, em que a proporção entre o tempo de pulso e o período total do sinal determina a intensidade média do sinal de controle. O PWM é particularmente eficiente em termos de energia, pois permite o controle da potência entregue ao motor de forma precisa e eficaz. Quando a largura do pulso é reduzida, a potência média fornecida ao motor é diminuída, resultando em uma velocidade mais baixa. Por outro lado, quando a largura do pulso é aumentada, a potência média aumenta e a velocidade do motor aumenta.

Um ESC controla a forma de onda trifásica aplicada ao motor. Ele utiliza uma série de pelo menos seis transistores de potência e diodos para modular o sinal trifásico, como pode

ser observado na figura 1. O acionamento de MOSFETs em configuração de meia-ponte apresenta numerosos desafios para os projetistas. Um desses desafios é o de gerar um sinal para ativar o FET em *high-side* (lado alto). [5] O início do capítulo de materiais e métodos trará informações sobre a estratégia utilizada nesse trabalho para chavear os MOSFETs em configuração *high-side*.

Os transistores aplicam tensão a uma fase e conectam outra ao terra, enquanto a terceira fase fica desconectada. A corrente gerada através das bobinas do motor cria um campo magnético que atrai os ímãs permanentes. A corrente é alternada ao redor do motor em seis etapas, arrastando o rotor consigo. A terceira fase desconectada ainda experimenta o campo magnético do rotor em rotação. Isso é chamado de força eletromotriz de retorno ou *back EMF* [3].

No ESC sem sensor (*sensorless*), o controlador utiliza essa *back EMF* como sensor de posição para comutar. Uma outra implementação, que é mais frequentemente usada, é a utilização de sensores de efeito hall (*sensored*) que detectam a posição do imã do rotor. A figura 2 ilustra as seis regiões ou setores distintos nos quais dois enrolamentos distintos são excitados e o capítulo 2 trará mais detalhes sobre o funcionamento do ESC em modo *sensorless* e modo *sensored*.

Um ESC pode ser programado para produzir uma onda senoidal PWM, o que está se tornando mais popular devido ao seu menor ruído e suposta maior eficiência em baixa potência. Os controladores de velocidade com onda senoidal tendem a ser mais caros devido ao aumento na complexidade do *software*. A forma de onda de saída do controlador é, portanto, um sinal de tensão trapezoidal trifásico unipolar. O sinal possui uma frequência de comutação de 8 a 32 kHz, com uma frequência de saída geralmente entre 100 e 1000 Hz, dependendo do motor e de sua velocidade [2].

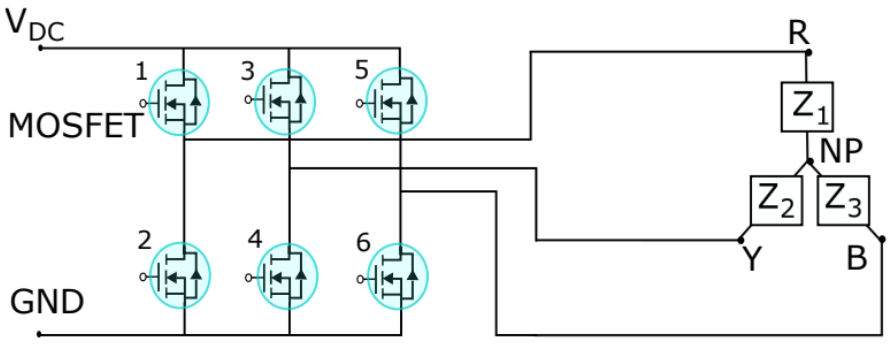


Figura 1 – Inversor com 6 transistores de potência [2].

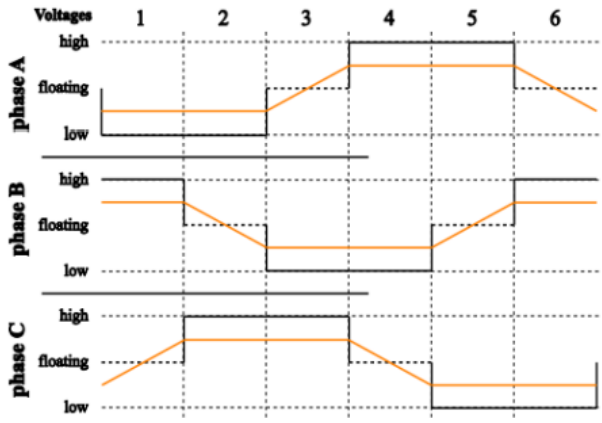


Figura 2 – Padrão de comutação em 6 passos e correspondente *back EMF* (linha laranja).[6]

Por fim, a frenagem regenerativa é um recurso inovador presente em muitos sistemas de propulsão elétrica, que permite que a energia cinética do veículo seja convertida em energia elétrica e armazenada nas baterias. Esse processo é realizado por meio dos motores elétricos atuando como geradores durante a desaceleração, recuperando parte da energia que normalmente seria dissipada como calor nos sistemas de freios convencionais. Essa abordagem oferece vantagens significativas, como a melhoria na eficiência energética, o aumento da autonomia dos veículos elétricos e a redução do desgaste dos freios, contribuindo para uma condução mais sustentável e econômica. A seção 1.1 traz mais detalhes sobre o estudo realizado e a estratégia utilizada para se obter a frenagem regenerativa no sistema ESC.

Figura 3 - Motor de Corrente Contínua Sem Escovas.

1.1 Frenagem Regenerativa

Como o próprio nome sugere, a frenagem regenerativa permite regenerar energia. No caso dos veículos elétricos e híbridos, esta frenagem é utilizada para regenerar a energia dissipada durante a desaceleração de forma a transformá-la em eletricidade. Na verdade, o princípio de funcionamento é muito simples. Para que um veículo se mova, seu motor transformará a corrente elétrica em energia cinética para mover as rodas. Agora, quando o veículo desacelera ou o motorista freia, o motor gira no sentido contrário e se torna um gerador de energia [7].

O motor elétrico simplesmente se opõe à rotação das rodas. Assim, terá o efeito de desacelerá-los, transformando sua energia cinética (energia relacionada à sua rotação) em energia elétrica. A eletricidade é então transmitida para a bateria para ser armazenada lá, conforme mostrado na figura 4. Em caso de aceleração, o motor elétrico reaproveitará essa eletricidade para funcionar e mover as rodas com ela (figura 5).

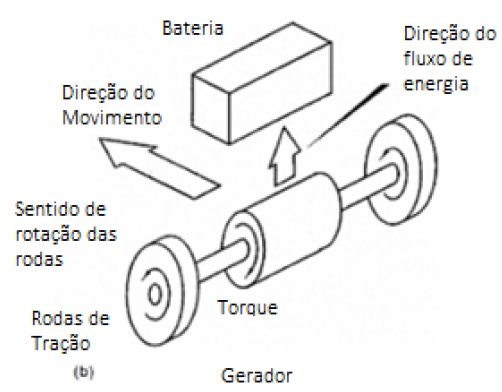


Figura 4 – Ação regenerativa durante frenagem [8].

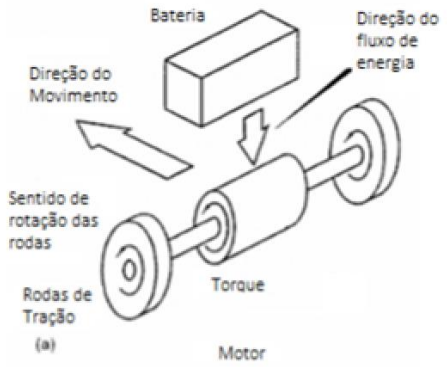


Figura 5 – Veículo acelerando [8].

1.2 Justificativa e Objetivos Específicos

A justificativa para realizar um estudo sobre os circuitos ESC (Controladores Eletrônicos de Velocidade) é baseada na crescente aplicação de motores elétricos, na importância da eficiência energética para otimizar o consumo de energia e reduzir o desperdício, no controle preciso e flexível que esses circuitos proporcionam permitindo uma ampla gama de aplicações, desde movimentação precisa em robótica até controle de velocidade em veículos elétricos e, além disso, estudar os circuitos ESC permite acompanhar a rápida evolução da eletrônica de potência e as novas tecnologias e conceitos que estão constantemente surgindo e contribuir para o desenvolvimento de tecnologias emergentes.

Os objetivos específicos são:

- Contrução das placas de circuito impresso e programação do microcontrolador para acionamento do motor em dois modos (sem sensores e com sensores de efeito Hall);
- Testes do circuito de modulação;
- Revisão bibliográfica dos modos de acionamento e sobre o tema frenagem regenerativa;
- Atualizações dos códigos para implementação de uma estratégia de frenagem regenerativa buscando maior eficiência do sistema;
- Identificar oportunidades de melhoria e inovação aprofundando o conhecimento sobre os circuitos ESC, explorando suas possibilidades de desenvolvimento.

2 Materiais e Métodos

Este capítulo apresentará em detalhes a execução dos três protótipos desenvolvidos neste projeto. O primeiro protótipo consistiu em um circuito ESC alimentado por uma bateria de 12 V para acionamento de um motor BLDC de drone sem o uso de sensores. O segundo protótipo foi um circuito ESC também alimentado por uma bateria de 12 V, porém, com a incorporação de sensores de efeito Hall para o acionamento do motor BLDC. Por fim, o terceiro protótipo foi um circuito ESC alimentado por uma bateria de 24 V, que utilizou tanto os sensores de efeito Hall quanto a estratégia de frenagem regenerativa.

A primeira seção deste capítulo abordará em detalhes o funcionamento do circuito de *Bootstrap* empregado para acionar os MOSFETs. Serão apresentadas as etapas de operação e os componentes envolvidos nesse processo, destacando a importância desse circuito para o controle eficiente dos motores. A segunda seção do capítulo discutirá os dois modos de controle utilizados para o motor (com e sem sensores), explicando o funcionamento de ambos.

2.1 Chaveamento de um MOSFET em configuração High-Side

Os requisitos de acionamento de porta para um MOSFET ou IGBT de potência utilizado como um interruptor de alta tensão pode ser resumido a seguir [5]:

1. A tensão no *Gate* deve ser de 10 V à 15 V superior a tensão na *Source*. Como se trata de um interruptor *high-side*, essa tensão no *Gate* deve ser superior a tensão no barramento c.c., que é geralmente a tensão mais elevada disponível no sistema.

2. A tensão do *Gate* deve ser controlável a partir da lógica do circuito de controle, que normalmente é referenciada ao terra. Assim, os sinais de controle têm que ser deslocados de nível para a *Source* do dispositivo *high-side*, que, na maioria das aplicações, oscila entre os dois barramentos.

3. A potência absorvida pelo circuito do *gate driver* não deve afetar de maneira significativa a eficiência global.

Um dos meios mais comuns e econômicos para os projetistas de fazer o acionamento de um MOSFET em configuração *high-side* é através da utilização de um um circuito de

Bootstrap. Esse circuito é constituído por um capacitor, um diodo, uma resistência e um capacitor de derivação (*bypass*).

Os esquemas de *bootstrap* podem aumentar a tensão de uma fonte de tensão fornecida, Vcc, em um nível de potencial flutuante e podem fornecer corrente suficiente para os *gate drivers* do MOSFET high-side. Recentemente, os esquemas *bootstrap* tornaram-se populares na indústria. Além disso, esquemas de *bootstrap* podem ser implementados em circuitos integrados [9].

- Princípio de funcionamento do Circuito de *Bootstrap*

Um circuito de *bootstrap* é usado em configurações de meia ponte para fornecer polarização ao FET *high-side*. A Figura 6 mostra o caminho de carregamento de um circuito *bootstrap* em uma configuração simplificada de meia ponte. Quando o FET *low-side* está ligado (o FET *high-side* está desligado), o pino HS e o nó do *switch* são conectados ao terra; a fonte de polarização VDD, através do capacitor de *bypass*, carrega o capacitor de *bootstrap* C_{BOOT} através do diodo D_{BOOT} e resistor de *bootstrap* R_{BOOT} [10].

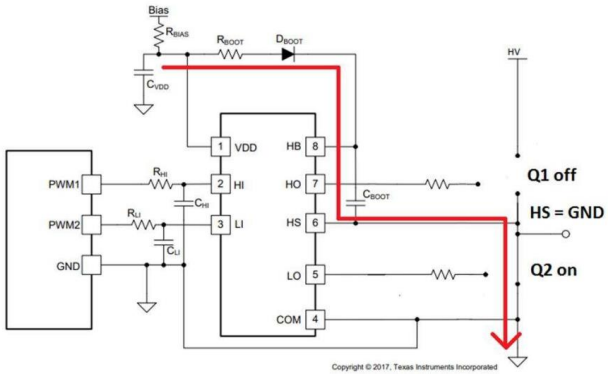


Figura 6 – Caminho de carregamento de um circuito *bootstrap* [10].

Quando o FET *low-side* é desabilitado e o FET *high-side* é habilitado, o pino HS do *gate driver* e o nó do *switch* são puxados para o barramento HV de alta tensão; o capacitor de *bootstrap* despeja parte da tensão armazenada (acumulada durante a sequência de carregamento) para o FET *high-side* por meio dos pinos HO e HS do *gate driver*, como mostrado na figura 7 [10].

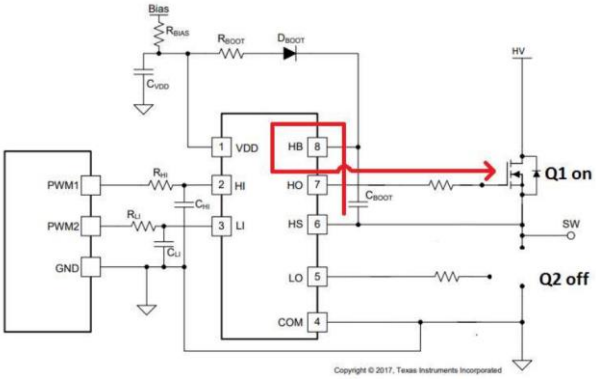


Figura 7 – Caminho de descarregamento de um circuito *boostrap* [10].

2.2 Modo Sensorless e Modo Sensored

No sistema sem sensores, o enrolamento flutuante é usado para detectar o cruzamento com zero, então a combinação de todos os 3 pontos de cruzamento por zero é usada para gerar a sequência. No total tem-se 6 eventos: Passagem por zero da Fase A: alto para baixo e baixo para alto; Passagem por zero da Fase B: alto para baixo e baixo para alto; Passagem por zero da Fase C: alto para baixo e baixo para alto.

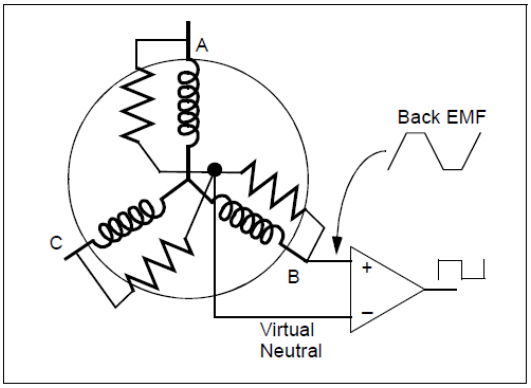


Figura 8 – Circuito para deteção de eventos de passagem por zero [11].

O ponto neutro virtual é o mesmo para todos os 3 comparadores, ele é gerado usando 3 resistores. Quando a *back EMF* (força contra eletromotriz) gerada no enrolamento flutuante (aberto) cruza o ponto zero para o lado positivo, a saída do comparador transita de baixo para alto. Quando a *back EMF* gerada no enrolamento flutuante cruza o ponto zero para o lado negativo, a saída do comparador faz uma transição de alto para baixo. Por ter três desses circuitos comparadores, um em cada uma das fases fornece três sinais digitais correspondentes ao sinal *back EMF* nos enrolamentos. A combinação desses três sinais é usada para evoluir a sequência de comutação [11].

No sistema com sensores de efeito Hall, a função dos sensores que estão conectados às três fases do motor (distribuídas uniformemente ao redor do estator a uma distância de 120°) é indicar quando fazer a comutação. Existem 6 comutações (101, 001, 011, 010, 110, 100) que se repetem para dar uma volta completa no motor. Cada valor de código representa um setor em que o rotor está atualmente e, portanto, fornece informações sobre quais enrolamentos precisam ser energizados. Assim, uma simples tabela de consulta pode ser utilizada pelo programa para determinar quais dois enrolamentos específicos energizar e, assim, girar o rotor.

Esses tipos de dispositivos são baseados na teoria do efeito Hall, que afirma que, se um condutor que transporta corrente elétrica é mantido em um campo magnético, o campo magnético exerce uma força transversal nos portadores de carga em movimento que tende a empurrá-los para um lado do condutor. Um acúmulo de carga nas laterais dos condutores equilibrará essa influência magnética, produzindo uma tensão mensurável entre os dois lados do condutor [12].

A figura 9 mostra a relação entre o sinal dos sensores de efeito Hall e a *back EMF* nas fases do motor.

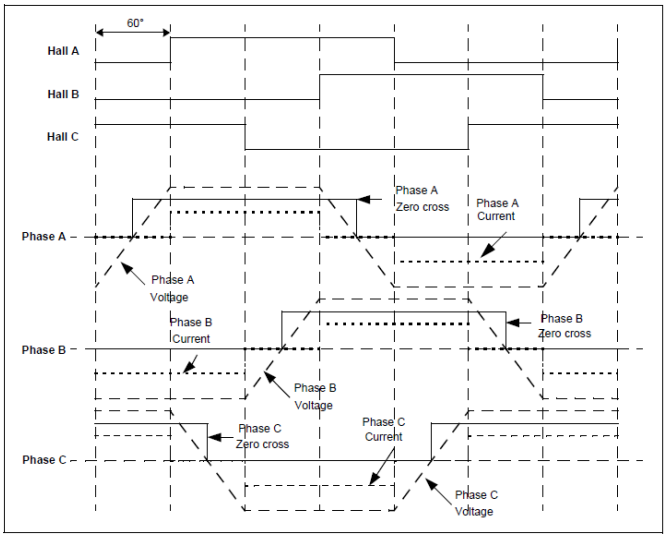


Figura 9 – Relação entre os sensores de efeito Hall e a força contra eletromotriz [11].

2.3 Desenvolvimento do primeiro protótipo.

O primeiro protótipo consiste no acionamento de um motor BLDC de um drone, alimentado através de uma bateria de 12 V e utilizando um circuito ESC *sensorless* e detecção dos pontos de cruzamento com zero (*zero-crossing*) com o auxílio de um Arduino.

Um motor BLDC *sensorless* não possui sensores Hall embutidos para detectar a posição do rotor. Em vez disso, ele utiliza a técnica de detecção dos pontos de cruzamento com zero (*zero-crossing*) da *back EMF* (força eletromotriz de retroalimentação) para estimar a posição do rotor. Quando o rotor gira, as bobinas do estator produzem uma tensão conhecida como *back EMF*. A magnitude e a forma dessa tensão variam de acordo com a posição do rotor. Durante a comutação das fases do motor, ocorrem os pontos de cruzamento com zero, onde a tensão *back EMF* é nula. Esses pontos de cruzamento são utilizados para estimar a posição do rotor.

O Arduino é utilizado para ler a tensão de *back EMF* do motor. Ele monitora a tensão nas bobinas do estator e identifica os pontos de cruzamento com zero, onde a tensão muda de positiva para negativa ou vice-versa. Esses pontos indicam a posição aproximada do rotor.

O esquema do circuito é mostrado na figura 10. O circuito possui dois botões do tipo *pushbutton*, sendo um deles utilizado para aumentar a velocidade do motor e o outro para diminui-la. Os três primeiros resistores de 33 kΩ (conectados as fases do motor) e os três resistores de 10 kΩ, são utilizados como pontes divisoras de tensão, porque não se pode fornecer 12 V ao microcontrolador. Os outros 3 resistores de 10 kΩ geram o ponto neutro virtual.

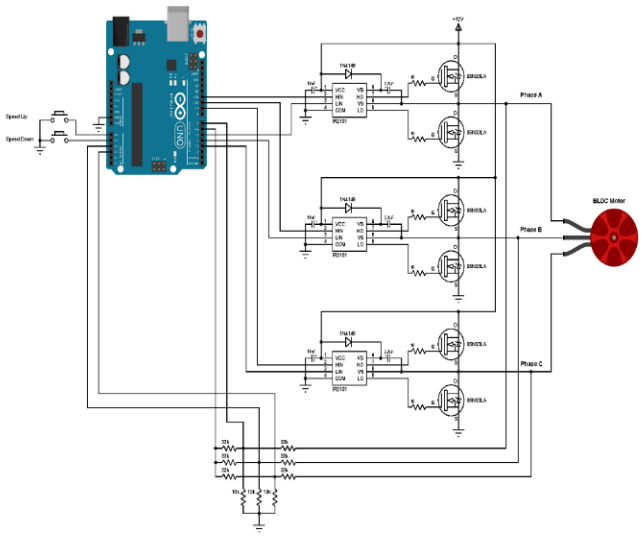


Figura 10 – Esquema eletrônico do Protótipo 1.

O ponto neutro virtual é uma referência fixa que representa a posição inicial do rotor. Ele é estabelecido durante o processo de inicialização do motor. O Arduino utiliza os pontos de cruzamento com zero detectados para estimar a posição do rotor em relação a esse ponto neutro virtual. Com base na estimativa da posição do rotor em relação ao ponto neutro virtual, o circuito ESC aciona as fases do motor em uma sequência adequada para gerar o campo magnético rotativo. A comutação das fases ocorre em intervalos regulares, conforme determinado pela velocidade de rotação desejada.

A placa do Arduino UNO é baseada no microcontrolador ATmega328P que possui um comparador analógico. A entrada positiva deste comparador está no pino 6 (AIN0) do Arduino e a entrada negativa pode ser o pino 7 (AIN1), A0 (ADC0), A1 (ADC1), A2 (ADC2), A3 (ADC3), A4 (ADC4) ou A5 (ADC5). O ponto neutro virtual é então conectado ao pino 6 do Arduino, a fase A ao pino 7 (AIN1), a fase B ao pino A2 e a fase C ao pino A3. O comparador fica comparando o ponto neutro virtual com a *back EMF* de uma fase (isso é feito no *software*).

O comparador analógico compara a entrada positiva AIN0 (pino 6 do Arduino) com a entrada negativa que pode ser AIN1 (pino 7), ADC2 (pino A2) ou ADC3 (pino A3). Quando a tensão do pino positivo é maior que a tensão do pino negativo, a saída do comparador analógico (ACO) é ativada. Quando a tensão do pino positivo é menor que a tensão do pino negativo, a saída do comparador analógico é desativada. Nesse projeto, a interrupção do comparador analógico é utilizada para detectar as transições de baixo para alto e de alto para baixo (subida e queda) nas tensões comparadas. Essas transições correspondem aos eventos de cruzamento com zero da *back EMF* do motor. Ao ocorrer um cruzamento com zero, uma interrupção é gerada no microcontrolador, interrompendo o fluxo normal do programa e permitindo que o código associado a essa interrupção seja executado.

Os *gate drivers* IR2101S são componentes utilizados para controlar os MOSFETs presentes no lado alto e no lado baixo de cada fase do motor. Esses *gate drivers* permitem alternar entre o lado alto e o lado baixo de acordo com as linhas de controle HIN e LIN. No contexto desse projeto, as linhas HIN dos três IR2101S estão conectadas aos pinos 11, 10 e 9 do Arduino, correspondendo à fase A, fase B e fase C, respectivamente.

Os pinos 9, 10 e 11 do Arduino têm a capacidade de gerar sinais PWM. Esses pinos estão conectados aos módulos de temporização (*Timers*) do Arduino, sendo o pino 9 (OC1A) e o pino 10 (OC1B) conectados ao módulo *Timer1*, e o pino 11 (OC2A) conectado ao módulo *Timer2*. Os módulos *Timer* são configurados para gerar sinais PWM com uma frequência

aproximada de 31 kHz e uma resolução de 8 bits. Cada um dos temporizadores possui um pré-escalonador que gera o relógio do temporizador dividindo o relógio do sistema por um fator de pré-escalonamento (N), como 1, 8, 64, 256 ou 1024.

O Arduino possui um relógio do sistema de 16 MHz ($f_{clkI/O}$) e a frequência do relógio do temporizador será a frequência do relógio do sistema dividida pelo fator de pré-escalonamento. A frequência do PWM para a saída dos pinos pode ser calculada pela equação seguinte:

$$f_{OCnxPWM} = \frac{f_{clkI/O}}{N \times 256} \qquad (1)$$

No código, o fator de pré-escalonamento utilizado é N igual a 1. O valor 256 representa o máximo do contador do temporizador utilizado para gerar a forma de onda PWM. Esse valor é específico para a configuração de 8 bits do contador do temporizador e determina o período da forma de onda PWM.

Os ciclos de trabalho desses sinais PWM são atualizados quando um botão é pressionado, seja para acelerar ou desacelerar o motor, e essas atualizações são feitas escrevendo nos registros correspondentes (OCR1A, OCR1B e OCR2A).

Segue a explicação de algumas partes chave do código:

1. **Configuração de pinos:**

- Os pinos 3, 4 e 5 do Arduino Uno são configurados como saídas e conectados à entrada LIN do driver de porta.

- Os pinos 9, 10 e 11 são configurados como saídas e conectados à entrada HIN do driver de porta.

2. **Configuração do timer:**

- O *Timer1* e o *Timer2* são configurados para usar a fonte de *clock* clkI/O sem prescalar (frequência máxima).

- Esses *timers* serão usados para gerar os sinais PWM para controlar os MOSFETs do motor BLDC.

3. **Configuração do comparador analógico:**

- O comparador analógico é configurado para gerar interrupções com base em mudanças nos sinais de retroalimentação de *back EMF* dos três enrolamentos do motor BLDC (A, B e C).

- Os sinais de retroalimentação são amostrados usando os pinos AIN1 (D7), A2 e A3 do Arduino Uno.

4. **Funções de movimento e comutação do motor BLDC:**

- A função *bldc_move()* é chamada a partir da interrupção do comparador analógico e determina qual estágio da comutação do motor BLDC deve ser executado com base no valor de *bldc_step*.

- Existem seis casos (0 a 5) correspondentes a seis estágios diferentes de comutação do motor BLDC. Cada caso chama uma função específica para configurar os pinos corretos do *driver* de porta e definir a direção do sinal de *back EMF*.

5. **Função *loop()*:**

- A função *loop()* é a função principal do programa.

- Ela começa configurando o PWM inicial com um ciclo de trabalho definido por *PWM_START_DUTY*.

- Em seguida, ocorre uma sequência de partida do motor BLDC em malha aberta, onde os enrolamentos são energizados em uma sequência pré-determinada.

- Depois disso, a velocidade do motor pode ser ajustada usando dois botões (*SPEED_UP* e *SPEED_DOWN*).

- O ciclo de trabalho do PWM é ajustado com base na direção do botão pressionado, aumentando ou diminuindo a velocidade do motor em incrementos de 1.

- A função *SET_PWM_DUTY()* é usada para configurar o ciclo de trabalho do PWM para os pinos 9, 10 e 11.

Para obter um entendimento completo do código utilizado nesse projeto, é importante consultar a folha de dados do microcontrolador ATmega328.

Uma PCB (*Printed Circuit Board*) foi projetada usando o *software DesignSparks* e encomendada através do site da *JLCPCB*. Em seguida, os componentes foram soldados na placa e é seu *layout* assim como todo o projeto que pode ser visualizado na figura 11:

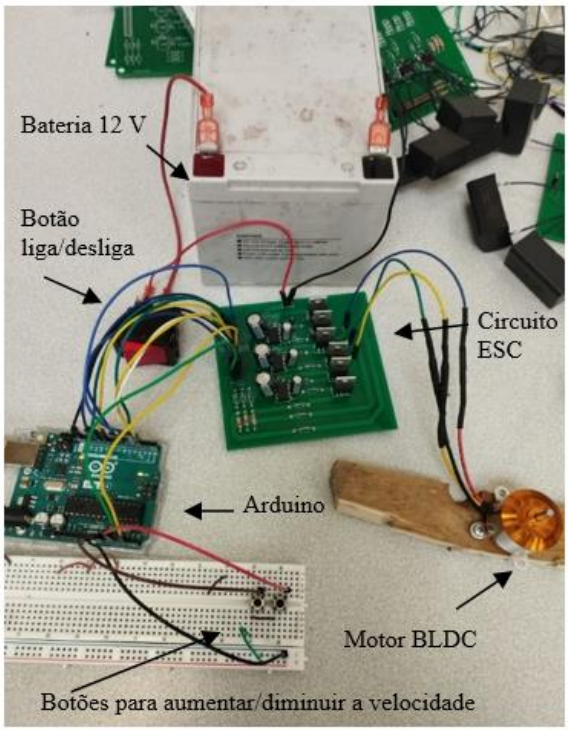


Figura 11 – *Setup* do Protótipo 1.

2.4 Desenvolvimento do segundo protótipo.

Para este protótipo foi utilizado um novo motor BLDC. Três sensores de efeito Hall são pré-instalados dentro do motor. Esses sensores Hall são distribuídos uniformemente ao redor do estator (distância de 120°). Esses sensores são posicionados de forma que a polaridade dos ímãs mude antes mesmo de o rotor estar na posição para a próxima comutação, evitando assim que o rotor trave. Os três sensores são alimentados com +5 V através de um resistor *pull-up* de 10 kΩ.

A interrupção do Arduino está habilitada para os pinos 5, 6 e 7, que são as entradas dos sensores de efeito Hall, visando uma comutação mais precisa. A saída do sensor é uma saída digital e pode ser nível alto (5 V) ou nível baixo (0 V).

As portas dos MOSFETs do lado alto recebem os sinais PWM e, portanto, as entradas HIN dos *drivers* de cada fase são conectadas às saídas 9, 10 e 11 do arduino. As portas dos MOSFETs do lado baixo recebem um sinal contínuo e as entradas LIN são conectadas às saídas 2 (fase A), 3 (fase B) e 4 (fase C) do Arduino.

A tabela abaixo resume os pinos ativos do Arduino de acordo com os estados dos sensores de efeito Hall (pinos: 5, 6 e 7):

Tabela 1 – Pinos ativos do Arduino de acordo com os estados dos sensores de efeito hall.

Pino / Etapa	**7**	**6**	**5**	**11**	**10**	**9**	**4**	**3**	**2**
3	0	0	1	0	0	PWM	1	0	0
2	0	1	0	0	PWM	0	0	0	1
6	0	1	1	0	PWM	0	1	0	0
4	1	0	0	PWM	0	0	0	1	0
5	1	0	1	0	0	PWM	0	1	0
1	1	1	0	PWM	0	0	0	0	1

O diagrama do circuito do projeto é mostrado na figura a seguir. O potenciômetro é usado para aumentar e diminuir a velocidade do motor BLDC variando o ciclo de trabalho.

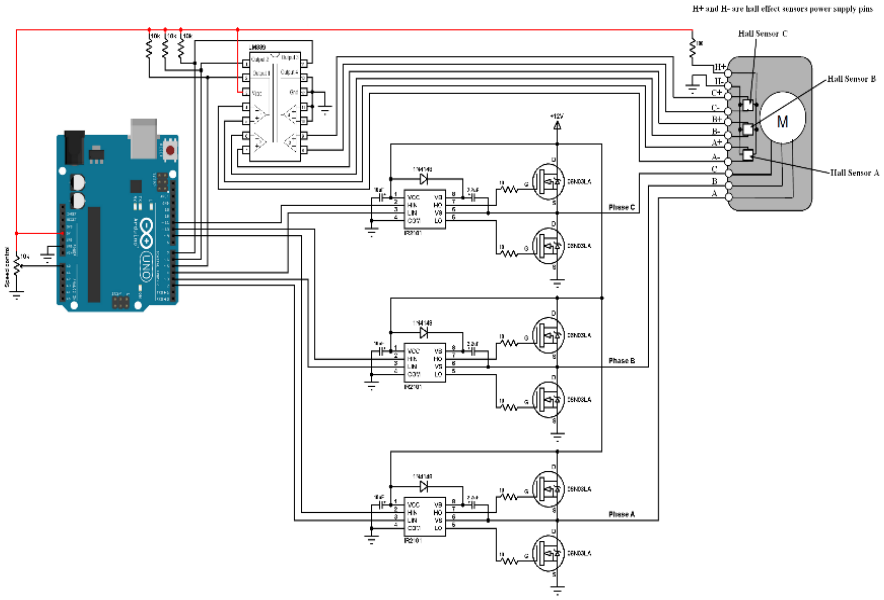


Figura 12 – Esquema eletrônico do Protótipo 2.

A diferença principal entre o código que utiliza sensores Hall e o código que não utiliza sensores Hall está na detecção do estado atual do motor BLDC.

No código que utiliza sensores Hall, a detecção do estado do motor é feita através da leitura dos valores dos sensores de efeito Hall. Esses sensores fornecem informações sobre a posição do rotor do motor, permitindo determinar qual bobina precisa ser acionada em cada momento. Isso é importante para o correto funcionamento do motor BLDC, pois as bobinas devem ser acionadas em sequência específica para gerar o movimento.

Por outro lado, no código que não utiliza sensores Hall, a detecção do estado do motor é baseada em um esquema de comutação pré-definido. Nesse caso, as bobinas são acionadas em uma sequência fixa, sem depender da posição do rotor. Isso significa que o código assume uma sequência de acionamento das bobinas e a repete continuamente para gerar o movimento do motor.

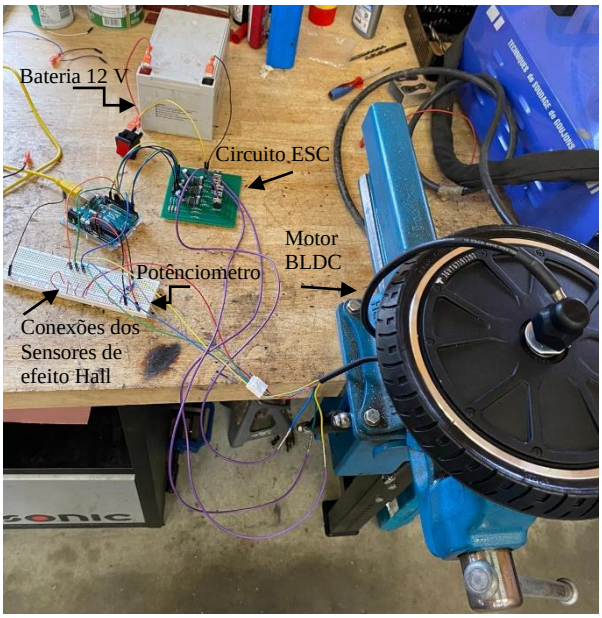


Figura 13 – *Setup* do Protótipo 2. Foi utilizado um motor rodando a 12 V e uma protoboard para conectar os sensores ao Arduino através de resistores *pull-up*.

2.5 Desenvolvimento do terceiro protótipo.

Desta vez, foi desenvolvida uma nova placa de circuito impresso para permitir o acionamento do motor a 24 V. Para isso, foram utilizadas duas baterias: uma de 12 V para alimentar os *drivers* dos MOSFETs e outra de 24 V para o dreno dos MOSFETs do lado alto (barramento CC). Os componentes utilizados no sistema operando em 12 V foram mantidos.

O circuito utilizado é semelhante ao do protótipo anterior (conforme mostrado na figura 12), com exceção dessas modificações. Além disso, foi adicionado um capacitor de 10uF na entrada da fonte de alimentação para filtrar e eliminar os efeitos da indutância da fiação entre a fonte de alimentação e o circuito de teste.

Devido à necessidade de utilizar três portas adicionais capazes de enviar sinais PWM (portas 3, 5 e 6), tanto o código quanto o circuito foram modificados para acomodar essa

alteração. Como resultado, os pinos do lado inferior do motor foram alterados de 2, 3 e 4 para 3, 5 e 6, pois esses pinos são capazes de enviar sinais PWM. Adicionalmente, os pinos do sensor Hall foram alterados de 5, 6 e 7 para 2, 4 e 7.

Essas modificações permitiram a operação do motor a 24 V, garantindo o adequado acionamento dos MOSFETs, a filtragem da fonte de alimentação e a utilização das portas necessárias para o controle do motor com os sensores Hall.

Foram pesquisadas estratégias para aplicar uma frenagem regenerativa nesse sistema. Um método simples e eficaz é a comutação independente em conjunto com a modulação por largura de pulso (PWM) para se obter um controle eficaz da frenagem. Na comutação independente, todos os dispositivos eletrônicos de comutação são desabilitados quando a frenagem regenerativa é aplicada. Os mosfets do lado baixo são colocados chaveados recebendo o sinal PWM para controlar a corrente que será injetada na bateria. Todos os interruptores superiores são mantidos desligados. Tudo isso é feito alterando a comutação dos MOSFETs no código do Arduino, agora, sendo dividido em uma parte para aceleração, e outra para frenagem.

Ao fazer isso, os MOSFETs do lado inferior, juntamente com o indutor e o resistor do motor, transformam o circuito em um conversor *Boost*. No esquema baseado no conversor *Boost*, a *back EMF* é aumentada e a bateria é carregada. Em particular, a *back EMF* de um motor é sempre menor que a tensão da bateria. Portanto, é necessário aumentar a amplitude dela para transferir energia para a bateria. Para realizar a operação de carregamento, o conversor *Boost* é conectado ao terminal do motor e a maior tensão é obtida controlando o ciclo de trabalho dos pulsos de comutação.

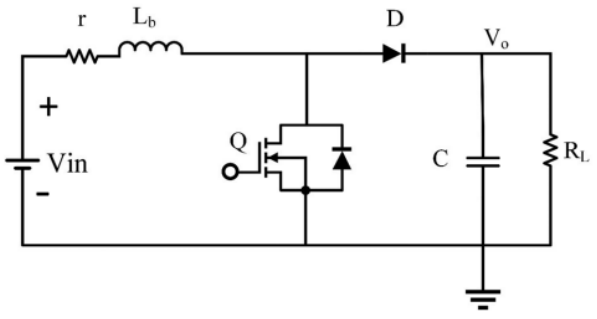


Figura 14 – Topologia do Conversor *Boost* [14].

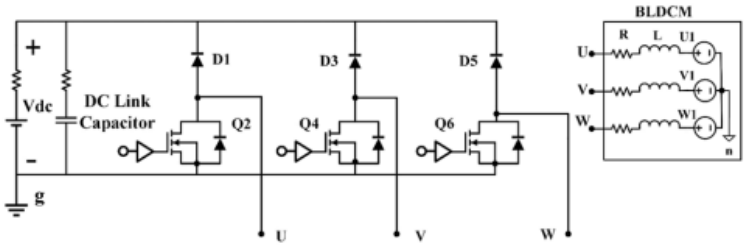


Figura 15 – Esquema do circuito quando os MOSFETs *high-side* são desativados [14].

Para este projeto foram necessárias várias alterações de código, que serão explicadas a seguir. Cada etapa do código será explicada mostrando as alterações feitas a cada passo percorrido até chegar a um código final seja bem explicado.

1. ***Setup* das variáveis:**

O código permite que os pinos 3, 5, 6, 9, 10 e 11 do Arduino gerem sinais PWM. Esses pinos são conectados aos módulos *Timer* do Arduino da seguinte forma: os pinos 5 (fase B *low-side*) e 6 (fase C *low-side*) estão ligados ao módulo *Timer* 0 (OC0A e OC0B), os pinos 9 (fase A *high-side*) e 10 (fase B *high-side*) estão ligados ao módulo *Timer* 1 (OC1A e OC1B) e os pinos 3 (fase A *low-side*) e 11 (fase C *high-side*) estão ligados ao módulo *Timer* 2 (OC2B e OC2A).

Os três módulos *Timer* são configurados para gerar sinais PWM com uma frequência de 31 kHz e uma resolução de 8 bits. Os ciclos de trabalho dos sinais PWM são atualizados com base na leitura dos dados analógicos do canal 0, que está conectado ao pino A0 do Arduino (ligado a um potenciômetro). Os ciclos de trabalho são escritos nos respectivos registros (OCR0A, OCR0B, OCR1A, OCR1B, OCR2A e OCR2B) para atualizar os sinais PWM.

O código define constantes para o valor máximo do ciclo de trabalho do PWM (*PWM_MAX_DUTY*), bem como os valores máximos (*PWM_BRAKE_MAX*) e mínimos (*PWM_BRAKE_MIN*) do ciclo de trabalho do PWM para a frenagem regenerativa.

Além disso, a interrupção para a troca de fase é habilitada nos pinos 2, 4 e 7, que são as entradas dos sensores de efeito Hall. Essa interrupção melhora o processo de comutação. Inicialmente, um botão conectado ao pino 13 foi utilizado para ativar o modo de frenagem e a variável booleana "*brake*" é criada para indicar se o modo de frenagem está ativado.

Existem algumas variáveis globais para armazenar informações relacionadas à comutação do motor, como *bldc_step* (etapa atual da comutação), *motor_speed* (velocidade

do motor), *brake_speed* (velocidade de frenagem), *bldc_step1*, *bldc_step2* e *bldc_step3* (valores das entradas do sensor de efeito Hall).

A função *setup()* é executada uma vez no início e é usada para configurar as portas e os registradores necessários para o controle do motor. Ela configura as portas como saídas e define os registradores de controle dos *timers*. No fim ela chama a função *bldc_move()* para realizar o primeiro movimento do motor, como pode ser visualizado na figura 16.

```
#define PWM_MAX_DUTY      255         // duty cycle = 100%
//#define PWM_MIN_DUTY      0           // duty cycle = 0%
#define PWM_BRAKE_DUTY    150         // duty cycle = 59%

bool brake;

byte bldc_step, motor_speed, brake_speed, bldc_step1, bldc_step2, bldc_step3; // A byte stores an 8-bit unsigned number, from 0 to 255.

void setup() {

  //Serial.begin(9600);
  DDRD  |= 0x68;            // DDRD:The Port D Data Direction Register. Configure pins 3, 5 and 6 as outputs.
  PORTD  = 0x00;            // PORTD are Arduino uno pins: 0 ... 7. Initialize all pins in 0(low).
  DDRB  |= 0x0E;            // Configure pins 9, 10 and 11 as outputs. 0x0E = 0b1110.
  PORTB  = 0xD1;            // PORTB are Arduino uno pins: 8 ... 13. Initialize pins 9, 10, 11 and 13 in 0 (low).
  // Timer0 (pins 5 and 6) module setting: set clock source to clkI/O / 1 (no prescaling)
  TCCR0A = 0x00;               // Timer/Counter1 Control Register A. Normal port operation.
  TCCR0B = 0x01;            // Timer/Counter2 Control Register B.
  // Timer1 (pins 9 and 10) module setting: set clock source to clkI/O / 1 (no prescaling)
  TCCR1A = 0x00;               // Timer/Counter1 Control Register A. Normal port operation.
  TCCR1B = 0x01;            // Timer/Counter2 Control Register B.
  // Timer2 (pins 3 and 11) module setting: set clock source to clkI/O / 1 (no prescaling)
  TCCR2A = 0x00;               // Timer/Counter2 Control Register A. Normal port operation.
  TCCR2B = 0x01;            // Timer/Counter2 Control Register B.
  // ADC module configuration (ADC - analog-to-digital converter)
  ADMUX  = 0x60;            // Configure ADC module and select channel 0 (pg.217 datasheet Atmel).
                            // Write one to ADLAR to left adjust the result.

  ADCSRA = 0x84;            // Enable ADC module with 16 division factor (ADC clock = 1MHz)(pg.208 datasheet Atmel)
  // Pin change interrupt configuration
  PCICR  = 4;               // Enable pin change interrupt for pins 0 to 7  (PCINT23..16)
  /* When the PCIE2 bit is set (one) and the I-bit in the status register (SREG) is set (one), pin change interrupt 2 is enabled.
   * Any change on any enabled PCINT23..16 pin will cause an interrupt.*/
  PCMSK2 = 0x94;            // Enable pin change interrupt for pins 2, 4 and 7 - hall effect sensor inputs
  pinMode(13,   INPUT);

  // BLDC motor first move
  bldc_step1 = (PIND >> 2) & 1;
  bldc_step2 = (PIND >> 3) & 2;
  bldc_step3 = (PIND >> 5) & 4;
  bldc_step = bldc_step1 | bldc_step2 | bldc_step3;   // Read hall effect sensors status (PIND: read from PORTD which is arduino pins 0..7)
  bldc_move();                      // Move the BLDC motor (first move)
}
```

Figura 16 – *Setup* das variáveis.

2. Rotina de Interrupção:

O operador de deslocamento à direita (>>) desloca os bits do operando esquerdo para a direita pelo número de posições especificado pelo operando direito. O operador de E bit-a-bit (&) produz 1 na saída se ambos os bits de entrada forem 1, caso contrário, a saída será 0. O operador de OU bit-a-bit (|) produz 1 na saída se pelo menos um dos bits de entrada for 1, caso contrário, a saída será 0.

Portanto, se as entradas do sensor hall (bits 2, 4 e 7) forem, por exemplo: 10000100 com operador de deslocamento à direita e operador &, obtêm-se:

bldc_step1 = 00100001&00000001 = 00000001;

bldc_step2 = 00100000&00000000=00000000;

bldc_step3 = 00000100&00000100=00000100.

Fazendo a operação: *bldc_step = bldc_step1 | bldc_step2 | bldc_step3*

Resultará em:

bldc_step = 00000001 | 00000000 | 00000100 = 00000101.

A figura 17 mostra essas operações e a rotina de interrupção:

```
////////////////////////////////////////////////////////////////////////////////
ISR (PCINT2_vect){    // Interrupt Service Routine. Any change on PCINT2 vector will cause an interrupt.
  bldc_step1 = (PIND >> 2) & 1;
  bldc_step2 = (PIND >> 3) & 2;
  bldc_step3 = (PIND >> 5) & 4;
  bldc_step = bldc_step1 | bldc_step2 | bldc_step3;      // Read hall effect sensors status (PIND: read from PORTD which is arduino pins 0..7)
  bldc_move();
  }

////////////////////////////////////////////////////////////////////////////////
```

Figura 17 – Rotina de Interrupção.

Essa interrupção é acionada quando ocorre uma mudança nos pinos de interrupção PCINT23..16 (no caso, pinos 2, 4 e 7). Ela atualiza as variáveis relacionadas aos sensores de efeito Hall e chama a função *bldc_move()* para realizar o movimento do motor BLDC de acordo com o novo estado dos sensores.

3. **Função *bldc_move()*:**

A função *bldc_move()* é responsável pela comutação do motor BLDC.

Se o botão estiver ligado (*HIGH*), entramos no modo de frenagem, no qual os MOSFETs do lado alto são desligados e os MOSFETs do lado baixo são controlados por PWM. Caso contrário, entramos no modo de comutação do modo de aceleração (modo motor).

Durante o modo motor, os sinais PWM são enviados para as portas dos MOSFETs do lado alto, conectando as entradas HIN dos *drivers* de cada fase às saídas 9 (fase A), 10 (fase B) e 11 (fase C) do Arduino. As portas dos MOSFETs do lado baixo recebem um sinal contínuo e suas entradas LIN são conectadas às saídas 3 (fase A), 5 (fase B) e 6 (fase C) do Arduino.

A tabela a seguir resume os pinos ativos do Arduino de acordo com os estados dos sensores de efeito hall nos pinos 2, 4 e 7 e a figura 18 traz como isso é desenvolvido no código da função *bldc_move()* :

Tabela 2 – Pinos ativos do Arduino de acordo com os estados dos sensores de efeito hall durante o modo motor.

Pino / Fase	**7**	**4**	**2**	**11**	**10**	**9**	**6**	**5**	**3**
3	0	1	1	0	PWM	0	1	0	0
2	0	1	0	0	PWM	0	0	0	1
6	1	1	0	PWM	0	0	0	0	1
4	1	0	0	PWM	0	0	0	1	0
5	1	0	1	0	0	PWM	0	1	0
1	0	0	1	0	0	PWM	1	0	0

Tabela 3 – Pinos ativos do Arduino de acordo com os estados dos sensores de efeito hall durante o modo frenagem.

Pino / Fase	**7**	**4**	**2**	**11**	**10**	**9**	**6**	**5**	**3**
3	0	1	1	0	0	0	PWM	PWM	PWM
2	0	1	0	0	0	0	PWM	PWM	PWM
6	1	1	0	0	0	0	PWM	PWM	PWM
4	1	0	0	0	0	0	PWM	PWM	PWM
5	1	0	1	0	0	0	PWM	PWM	PWM
1	0	0	1	0	0	0	PWM	PWM	PWM

```
void bldc_move(){        // BLDC motor commutation function

      if (brake == HIGH){
        switch(bldc_step){
          case 1:
            AL_BL_CL();
            break;
          case 2:
            AL_BL_CL();
            break;
          case 3:
            AL_BL_CL();
            break;
          case 4:
            AL_BL_CL();
            break;
          case 5:
            AL_BL_CL();
            break;
          case 6:
            AL_BL_CL();
            break;
          default:
            PORTD = 0;
            break;
      }

      }else{
        switch(bldc_step){
          case 1:
            AH_CL();
            break;
          case 2:
            BH_AL();
            break;
          case 3:
            BH_CL();
            break;
          case 4:
            CH_BL();
            break;
          case 5:
            AH_BL();
            break;
          case 6:
            CH_AL();
            break;
          default:
            PORTD = 0;
            break;
        }
      }
  }
```

Figura 18 – Função *bldc_move()*.

4. **Função *loop()*:**

Dentro da função *loop* o botão será lido toda vez que for executada e após se ter seu estado uma decisão é tomada. Novamente, se ligado (nível alto) o código chama a função *SET_PWM_DUTY_BRAKE*, se desligado (nível baixo) o código chama a função *SET_PWM_DUTY* que tem como entrada *motor_speed*, que consiste no valor analógico lido do potenciômetro e que tem a função de controlar o ciclo de trabalho.

Durante a frenagem, em um primeiro tempo, um ciclo de trabalho fixo é determinado para os MOSFETs do lado baixo. Posteriormente serão explicadas as alterações do código que mostram outra forma de controlar o ciclo de trabalho durante a frenagem regenerativa a fim de obter mais eficiência no processo.

Ao executar o código presente na figura 19, tem-se um problema pois apertando o botão uma vez para fazer a frenagem o código não consegue sair do freio e o motor não gira mais quando o botão não é mais pressionado e deseja-se voltar ao modo motor.

```
void loop() {
  ADCSRA |= 1 << ADSC;  // Start conversion
  /* The first conversion must be started by writing a logical one to the ADSC bit in ADCSRA.
   *  In this mode the ADC will perform successive conversions independently of whether the ADC interrupt flag, ADIF is cleared or not. */
  while(ADCSRA & 0x40); // Wait for conversion complete.
  motor_speed = ADCH;   // Read ADC data (8 bits)
  //Serial.println(motor_speed);
  brake = digitalRead(13);
  if(brake==LOW){
    SET_PWM_DUTY(motor_speed); // motor_speed is a variable type byte and a variable byte stores an 8-bit unsigned number, from 0 to 255.
  }else{
    SET_PWM_DUTY_BRAKE(motor_speed);
  }
}
```

Figura 19 – Função *loop()* que não consegue sair do modo freio.

Para corrigir isso, depois de fazer a frenagem, o código lê o estado do motor novamente e, em seguida, chama a função *bldc_move* para que o motor possa passar para a próxima etapa no modo motor e girar novamente, como pode ser visto na figura 20.

```
void loop() {
  ADCSRA |= 1 << ADSC;
  while(ADCSRA & 0x40);
  motor_speed = ADCH;
  //Serial.println(motor_speed);
  brake = digitalRead(13);
  if(brake==LOW){
    SET_PWM_DUTY(motor_speed);
  }else{
    while(brake == HIGH){
      SET_PWM_DUTY_BRAKE(motor_speed);
      brake = digitalRead(13);
    }
    bldc_move();
  }
}
```

Figura 20 – Função *loop()* corrigida.

5. Funções de comutação das fases do motor:

As funções *AH_BL()*, *AH_CL()*, *BH_CL()*, *BH_AL()*, *CH_AL()*, *CH_BL()* e *AL_BL_CL()* são responsáveis por configurar os pinos do Arduino para ativar/desativar as fases do motor BLDC de acordo com a sequência correta de comutação tanto no modo motor como no modo frenagem. A figura 21 mostra a configuração dos pinos em cada etapa de comutação do motor.

```
void AH_BL(){

  TCCR2A =  0;       // Turn pin 11 &  pin 3 OFF
  TCCR0A = 0x21;     // Turn pin 5 ON & 6 OFF
  TCCR1A =  0x81;    // Turn pin 9 (OC1A) PWM ON (pin 10 OFF) - pin 9 - phase A

}
void AH_CL(){

  TCCR2A =  0;       // Turn pin 11 & pin 3 OFF
  TCCR0A = 0x81;     // Turn pin 6 ON & 5 OFF
  TCCR1A =  0x81;    // Turn pin 9 (OC1A) PWM ON (pin 10 OFF) - pin 9 - phase A

}
void BH_CL(){

  TCCR2A =  0;        // Turn pin 11 & pin 3 OFF
  TCCR0A = 0x81;      // Turn pin 6 ON & 5 OFF
  TCCR1A =  0x21;     // Turn pin 10 (OC1B) PWM ON (pin 9 OFF) - pin 10 - phase B

}
void BH_AL(){

  TCCR0A = 0;         // Turn pin 5 and 6 OFF
  TCCR2A =  0x21;     // Turn pin 11 OFF &  pin 3 ON
  TCCR1A =  0x21;     // Turn pin 10 (OC1B) PWM ON (pin 9 OFF) - pin 10 - phase B

}
void CH_AL(){

  TCCR0A = 0;        // Turn pin 5 and 6 OFF
  TCCR1A =  0;       // Turn pin 9 & pin 10 OFF
  TCCR2A =  0xA1;    // Turn pin 11 (OC2A) PWM ON - phase C & pin 3 (OC2B)ON - Phase A

}
void CH_BL(){

  TCCR1A =  0;           // Turn pin 9 & pin 10 OFF
  TCCR0A = 0x21;         // Turn pin 5 ON & 6 OFF
  TCCR2A =  0x81;        // Turn pin 11 (OC2A) PWM ON - phase C

}

void AL_BL_CL(){

  TCCR0A = 0xA1;
  TCCR1A = 0x00;
  TCCR2A = 0x21;
}
```

Figura 21 – Funções de comutação dos MOSFETs.

6. **Funções SET PWM:**

Durante o modo motor, as portas referentes ao lado baixo recebem PWM máximo (255) que é equivalente a um sinal contínuo. As portas do lado alto recebem um sinal PWM, que tem seu ciclo de trabalho controlado pelo potenciômetro conectado à entrada A0 do Arduino.

Durante o modo de freio, as portas referentes ao lado baixo recebem um valor PWM fixo atribuído à variável *PWM_DUTY_BRAKE*. As portas referentes ao lado alto recebem 0 e, portanto, são desabilitadas, como pode ser verificado nas linhas de código presentes na figura 22.

```
void SET_PWM_DUTY(byte duty){
  OCR2B = PWM_MAX_DUTY;                    // Set pin 3 PWM duty cycle
  OCR0B = PWM_MAX_DUTY;                    // Set pin 5 PWM duty cycle
  OCR0A = PWM_MAX_DUTY;                    // Set pin 6 PWM duty cycle

  OCR1A = duty;                   // Set pin 9 PWM duty cycle
  OCR1B = duty;                   // Set pin 10 PWM duty cycle
  OCR2A = duty;                   // Set pin 11 PWM duty cycle

}

void SET_PWM_DUTY_BRAKE(byte duty){
  OCR2B = PWM_BRAKE_DUTY;                    // Set pin 3 PWM duty cycle
  OCR0B = PWM_BRAKE_DUTY;                    // Set pin 5 PWM duty cycle
  OCR0A = PWM_BRAKE_DUTY;                    // Set pin 6 PWM duty cycle

  OCR1A = 0;                   // Set pin 9 PWM duty cycle
  OCR1B = 0;                   // Set pin 10 PWM duty cycle
  OCR2A = 0;                   // Set pin 11 PWM duty cycle

}
```

Figura 22 – Funções que definem o sinal PWM.

O código passou por algumas modificações para que alguns testes pudessem ser feitos e pudesse ser melhorado até chegar a uma versão final. Entre eles estão:

- **Passagem do modo motor ao modo frenagem sem a utilização de um botão e utilizando o potênciometro**

Lembrando que o potenciômetro envia um sinal de 0 a 5 V que equivale a um valor de 0 a 255 bits, foi escolhido então um valor de 25 bits (equivalente a 0,5 V) para fazer essa transição entre os dois modos. Se o potenciômetro enviar um sinal maior ou igual a este valor, o motor estará em modo motor, caso contrário, o motor estará em modo de frenagem regenerativa. A figura 25 mostra essa configuração presente na função *bldc_move()*.

```
void bldc_move(){        // BLDC motor commutation function

  if (motor_speed >= 25){
    switch(bldc_step){
       case 1:
         AH_CL();
         break;
       case 2:
         BH_AL();
         break;
       case 3:
         BH_CL();
         break;
       case 4:
         CH_BL();
         break;
       case 5:
         AH_BL();
         break;
       case 6:
         CH_AL();
         break;
       default:
         PORTD = 0;
         break;
    }
    }else{
       switch(bldc_step){
           case 1:
             AL_BL_CL();
             break;
           case 2:
             AL_BL_CL();
             break;
           case 3:
             AL_BL_CL();
             break;
           case 4:
             AL_BL_CL();
             break;
           case 5:
             AL_BL_CL();
             break;
           case 6:
             AL_BL_CL();
             break;
           default:
             PORTD = 0;
             break;
       }
     }
}
```

Figura 23 – Função *bldc_move()* que alterna entre modo motor e modo freio via um potênciometro.

Na função *loop()*, da mesma forma que antes, ao entrar no modo freio, o código deve reler a entrada (antes era o botão e agora o potenciômetro) ao final da função e chamar a função *bldc_move()* (ver figura 24).

```
void loop() {
  ADCSRA |= 1 << ADSC;                  //
  /* The first conversion must be start
   *  ADC interrupt flag, ADIF is clear
  while(ADCSRA & 0x40);                 //
  motor_speed = ADCH;                   //
  //Serial.println(motor_speed);
  if (motor_speed >= 25){
    SET_PWM_DUTY(motor_speed);
  }
  else{
    while(motor_speed < 25){
      SET_PWM_DUTY_BRAKE(motor_speed);
      while(ADCSRA & 0x40);
      ADCSRA |= 1 << ADSC;
      motor_speed = ADCH;
    }
}
```

Figura 24 – Função *loop()* que alterna entre modo motor e modo freio via um potênciometro.

- **Cálculo da velocidade eRPM – capturando a duração de uma etapa do motor**

Foi desenvolvido um código para otimizar a eficiência da recuperação de energia durante a frenagem regenerativa do motor. Esse código utiliza a captura da unidade de captura de entrada através do timer 1 para calcular a velocidade elétrica do motor, permitindo que o controle PWM seja realizado durante a frenagem nessa velocidade específica.

Para implementar esse código, um botão foi conectado entre o pino 8 do módulo e o terra, conforme ilustrado na figura 25. É importante declarar o pino 8 como uma entrada e ativar o resistor *pull-up* correspondente, como pode ser visualizado na figura 26. Além disso, o programa utiliza o monitor serial do IDE para exibir informações toda vez que o botão é pressionado.

Ao pressionar o botão, o programa exibirá as seguintes informações no monitor serial:

- O número de ciclos completos decorridos do *timer* 1.

- A data da etapa anterior registrada usando a função *micros()*.

- A data da próxima etapa registrada usando a função *micros()*.

- A diferença entre as duas datas, representando a duração entre as etapas de comutação do motor.

Dessa forma, o código permite monitorar e calcular com precisão a velocidade elétrica do motor durante a frenagem regenerativa, melhorando a eficiência do processo de recuperação de energia.

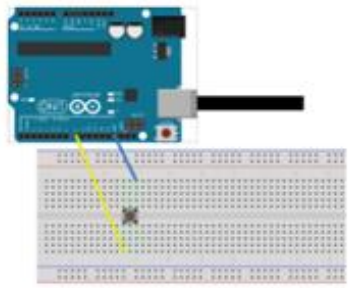


Figura 25 – Botão utilizado para realizar a captura da duração entre as etapas do motor [16].

O resistor de *pull-up* sendo ativado no pino 8, pressionando o botão fará com que o pino 8 vá do estado *HIGH* para o estado *LOW*; a unidade de captura de entrada é, portanto, configurada para reagir a uma borda descendente. A redução de ruído não é usada. A rotina de interrupção acionada pela unidade de captura de entrada calcula e exibe os dados no monitor serial. As interrupções devem ser autorizadas global e localmente (*overflow* e *capture*) [15].

```
#define PWM_MAX_DUTY      255         // duty cycle = 100%
//#define PWM_MIN_DUTY      0           // duty cycle = 0%
#define PWM_BRAKE_DUTY    100         // duty cycle = 59%

bool brake;
unsigned long step_top_e = 0;
unsigned long step_top_b = 0;
unsigned long step_duration = 0;
byte bldc_step, motor_speed, brake_speed, bldc_step1, bldc_step2, bldc_step3;
```

Figura 26 – Definição das novas variáveis.

A figura 27 as mudanças na função *setup()*para autorizar a captura feita pelo botão, a figura 28 mostra o que irá aparecer no monitor serial da IDE e a figura 29 mostra os cálculos realizados para encontra a velocidade desejada.

```
void setup() {
  Serial.begin(9600);
  DDRD  |= 0x60;           // DDRD:The Port D Data Direction Register. Configure pins 3, 5 and 6 as outputs. The
  PORTD  = 0x00;           // PORTD are Arduino uno pins: 0 ... 7. Initialize all pins in 0(low).
  DDRB  |= 0x0E;           // Configure pins 9, 10 and 11 as outputs. 0x0E = 0b1110.
  PORTB  = 0xD1;           // PORTB are Arduino uno pins: 8 ... 13. Initialize pins 9, 10, 11 and 13 in 0 (low).
  // Timer0 (pins 5 and 6) module setting: set clock source to clkI/O / 1 (no prescaling)
  TCCR0A = 0x00;              // Timer/Counter1 Control Register A. Normal port operation.
  TCCR0B = 0x01;           // Timer/Counter2 Control Register B. The Timer/Counter can be clocked by an internal
  // Timer1 (pins 9 and 10) module setting: set clock source to clkI/O / 1 (no prescaling)
  TCCR1A = 0x00;              // Timer/Counter1 Control Register A. Normal port operation.
  TCCR1B = 0x01;           // Timer/Counter2 Control Register B. The Timer/Counter can be clocked by an internal
  TIMSK1 = 0b00100000;     // autorisation interruption TOV et Input Capture
  TCNT1 = 0;
  ICR1 = 0;
  // Timer2 (pins 3 and 11) module setting: set clock source to clkI/O / 1 (no prescaling)
  TCCR2A = 0x00;              // Timer/Counter2 Control Register A. Normal port operation.
  TCCR2B = 0x01;           // Timer/Counter2 Control Register B.
  // ADC module configuration (ADC - analog-to-digital converter)
  ADMUX  = 0x60;           // Configure ADC module and select channel 0 (pg.217 datasheet Atmel). Bit 0 at REFS1
                           // The ADLAR bit affects the presentation of the ADC conversion result in the ADC dat
  ADCSRA = 0x84;           // Enable ADC module with 16 division factor (ADC clock = 1MHz)(pg.208 datasheet Atme
  // Pin change interrupt configuration
  PCICR  = 4;              // Enable pin change interrupt for pins 0 to 7  (PCINT23..16)
  /* When the PCIE2 bit is set (one) and the I-bit in the status register (SREG) is set (one), pin change interr
   * The corresponding interrupt of pin change interrupt request is executed from the PCI2 interrupt vector. PCI
  PCMSK2 = 0x94;           // Enable pin change interrupt for pins 2, 4 and 7 - hall effect sensor inputs(pg.57
  //pinMode(13,    INPUT);
  pinMode(8, INPUT_PULLUP);
```

Figura 27 – Inicialização do monitor serial, autorização da interrupção de captura e inicialização do botão.

```
ISR(TIMER1_CAPT_vect){
  Serial.println(step_top_e);
  Serial.println(step_top_b);
  Serial.println(step_duration);
}
```

Figura 28 – Captura da Interrupção.

```
case 2:
  step_top_e = micros();
  step_duration = step_top_e - step_top_b;
  AL_BL_CL();
  break;
case 3:
  step_top_b = micros();
  AL_BL_CL();
  break;
```

Figura 29 – Cálculo do tempo de um *step* utilisando dois *steps* consecutivos.

- **Controle PWM durante a frenagem usando dados de duração do passo e aprimoramento do modo do motor**

Como a duração de uma etapa já é conhecida, não é necessário mais capturá-la e imprimi-la no monitor serial, agora o que é buscado é usar essa informação para controlar o ciclo de trabalho dos MOSFETs low-side durante a frenagem, de modo que a recuperação de energia seja mais eficiente. Portanto, a parte de interrupção de captura (figura 28) foi desativada do código, mas o cálculo (figura 29) ainda continua porque é necessário.

A ideia é calcular o PWM de acordo com o seguinte gráfico (exemplo):

Figura 30 – Exemplo de comportamento PWM durante a frenagem em comparação com a duração de uma etapa do motor.

- Abaixo de um limite "*step_duration_min*" (no exemplo 1300): o tempo todo igual a 0 (*PWM_BRAKE_MIN*).

- Acima de um limite "*step_duration_max*" (no exemplo 6000): o tempo todo igual a um valor X (no exemplo, igual à 150 (*PWM_BRAKE_MAX*)).

- Entre dois limites "*step_duration_min*" e "*step_duration_max*" : proporcional. Para isso foi utilizada a função *map()*.

Assim, foram definidas as variáveis *PWM_BRAKE_MAX* e *PWM_BRAKE_MIN* como pode ser visto na figura 31, bem como duas outras variáveis *mappedPWMMOTOR* e *mappedPWMBRAKE* que serão utilizadas com a função *map()*.

```
#define PWM_MAX_DUTY       255         // duty cycle = 100%
#define PWM_BRAKE_MAX      250
#define PWM_BRAKE_MIN        0

int mappedPWMMOTEUR;
int mappedPWMBRAKE;
bool brake;
unsigned long step_top_e = 0;
unsigned long step_top_b = 0;
unsigned long step_duration = 0;
byte bldc_step, motor_speed, brake_speed, bldc_step1, bldc_step2, bldc_step3;
```

Figura 31 – Definição das variáveis *mappedPWMMOTOR* e *mappedPWMBRAKE*.

A função map do Arduino transfere um valor do intervalo atual para um novo intervalo de valores especificado pelos parâmetros - *map(value, fromLow, fromHigh, toLow, toHigh)*. O comando *map()* só funciona com números inteiros. A parte fracionária de um número não é arredondada de acordo com as regras matemáticas da transferência proporcional, mas simplesmente descartada [16].

Sintaxe:

map(input_value, input_lower, input_upper, output_lower, output_upper);

input_value: este é o número para mapear (para transformar)

lower_entry: este é o limite inferior do intervalo atual

upper_in : este é o limite superior do intervalo atual

lower_output: este é o limite inferior do intervalo resultante da alteração

upper_output: este é o limite superior do intervalo resultante da mudança

Por exemplo, no código anterior, quando um ciclo de trabalho de 25 é atingido, o sistema passou diretamente do modo de frenagem para o modo motor e, portanto,

repentinamente foi obtido 0,5 V nos MOSFETs do lado alto resultando em um ruído. Para fazer essa transição de forma mais suave foi utilizada a função map. O valor do duty é o valor a ser “mapeado” e após a alteração quando um ciclo de trabalho de 25 for atingido, esse valor será equivalente a 0. A figura 32 traz a utilziação da função *map()* no modo motor.

```
void SET_PWM_DUTY(byte duty){

  mappedPWMMOTEUR = map(duty, 25, 255, 0, 255);

  OCR2B = PWM_MAX_DUTY;                    // Set pin 3 PWM duty cycle
  OCR0B = PWM_MAX_DUTY;                    // Set pin 5 PWM duty cycle
  OCR0A = PWM_MAX_DUTY;                    // Set pin 6 PWM duty cycle

  OCR1A = mappedPWMMOTEUR;                    // Set pin 9 PWM duty cycle
  OCR1B = mappedPWMMOTEUR;                    // Set pin 10 PWM duty cycle
  OCR2A = mappedPWMMOTEUR;                    // Set pin 11 PWM duty cycle

}
```

Figura 32 – PWM modo motor utilizando a função *map()*.

O mesmo foi feito de forma semelhante com o PWM em modo frenagem mas desta vez o valor a transformar é *step_duration* porque deseja-se ter um comportamento como o mostrado no gráfico da figura 30 (PWM em função da velocidade do motor) . A função de restrição *constrain()* foi usada para garantir que os limites inferior e superior não sejam excedidos. A figura 33 mostra a utilização da função *map()* no modo frenagem.

```
void SET_PWM_DUTY_BRAKE(){

  if (step_duration <= 60000){
    OCR2B = PWM_BRAKE_MIN;                    // Set pin 3 PWM duty cycle
    OCR0B = PWM_BRAKE_MIN;                    // Set pin 5 PWM duty cycle
    OCR0A = PWM_BRAKE_MIN;                    // Set pin 6 PWM duty cycle

    OCR1A = 0;                    // Set pin 9 PWM duty cycle
    OCR1B = 0;                    // Set pin 10 PWM duty cycle
    OCR2A = 0;                    // Set pin 11 PWM duty cycle

  }
  else if(step_duration > 60000 && step_duration < 900000){

    step_duration = constrain(step_duration,60000,900000);
    mappedPWMBRAKE = map(step_duration, 60000, 900000, PWM_BRAKE_MIN, PWM_BRAKE_MAX);

    //Serial.println(mappedPWMBRAKE);

    OCR2B = mappedPWMBRAKE;                    // Set pin 3 PWM duty cycle
    OCR0B = mappedPWMBRAKE;                    // Set pin 5 PWM duty cycle
    OCR0A = mappedPWMBRAKE;                    // Set pin 6 PWM duty cycle

    OCR1A = 0;                    // Set pin 9 PWM duty cycle
    OCR1B = 0;                    // Set pin 10 PWM duty cycle
    OCR2A = 0;                    // Set pin 11 PWM duty cycle

  }

  else{

    OCR2B = PWM_BRAKE_MAX;                    // Set pin 3 PWM duty cycle
    OCR0B = PWM_BRAKE_MAX;                    // Set pin 5 PWM duty cycle
    OCR0A = PWM_BRAKE_MAX;                    // Set pin 6 PWM duty cycle

    OCR1A = 0;                    // Set pin 9 PWM duty cycle
    OCR1B = 0;                    // Set pin 10 PWM duty cycle
    OCR2A = 0;                    // Set pin 11 PWM duty cycle

  }
}
```

Figura 33 – PWM modo frenagem utilizando a função *map()*.

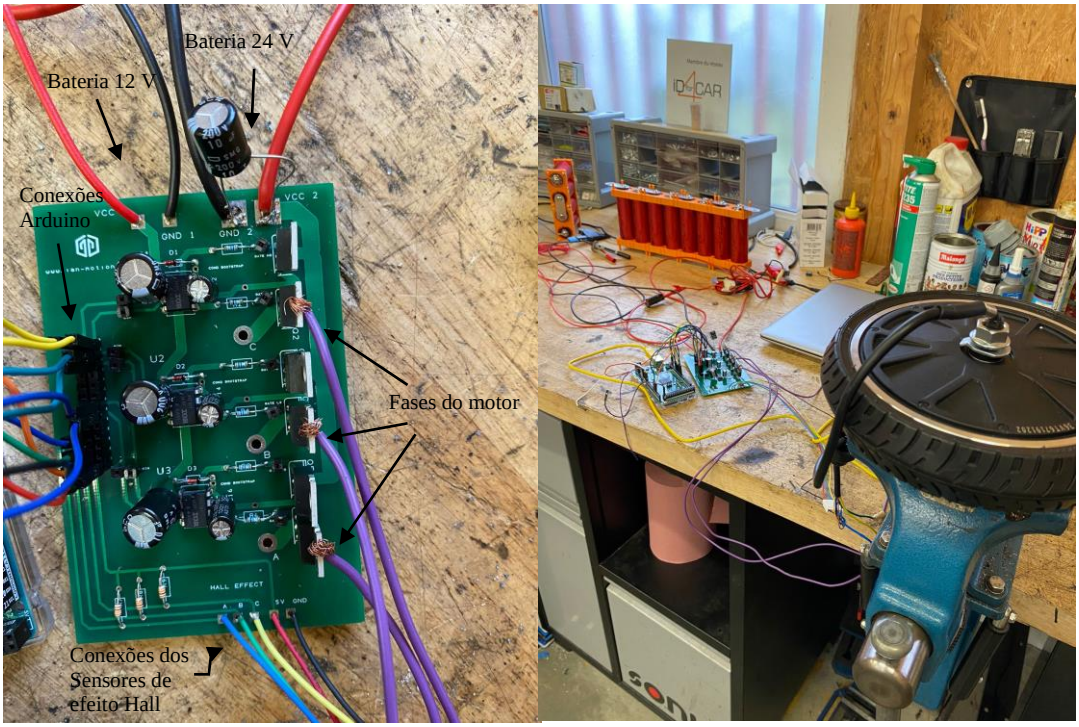


Figura 34 – *Setup* e PCB do protótipo 3.

3 Resultados e Discussão

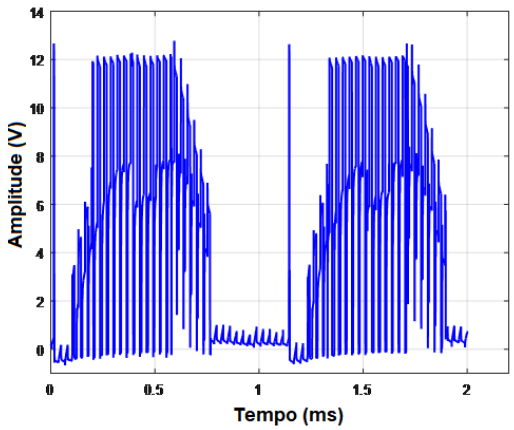
Os resultados serão apresentados a seguir separados em três seções, se referindo aos três protótipos desenvolvidos e apresentados na metodologia desse trabalho. O primeiro se refere ao circuito ESC para acionar um motor BLDC de drone em modo *sensorless* e utilizando uma bateria de 12 V, o segundo se refere ao acionamento de um motor BLDC no modo *sensored* com a utilização de sensores de efeito hall e utilizando uma bateria de 12 V, já o terceiro se refere ao acionamento do mesmo motor do segundo protótipo mas dessa vez utilizando uma bateria de 24 V e o estudo de estratégias de recuperação de energia durante a frenagem. A aquisição dos dados foi feita utilizando um osciloscópio.

3.1 Primeiro Protótipo

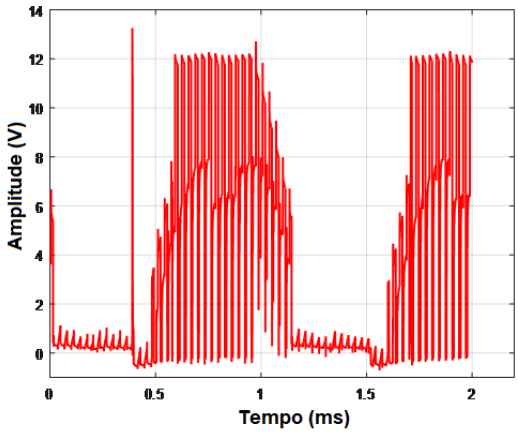
A figura 35 capturada com um osciloscópio mostra a mudança de fase de 120º entre as 3 fases do motor BLDC e a operação do circuito *bootstrap*. Nota-se bastante ruído nas ondas de saída e isso será corrigido posteriormente no terceiro protótipo com a inclusão de um capacitor na entrada do barramento CC.

A falta de sensores de efeito Hall pode afetar o funcionamento adequado do circuito, especialmente no que diz respeito ao controle do motor BLDC. A utilização da *back EMF* para o controle requer técnicas mais avançadas de processamento de sinal e algoritmos de controle, uma vez que os sinais podem ser mais sensíveis a ruídos e imprecisões.

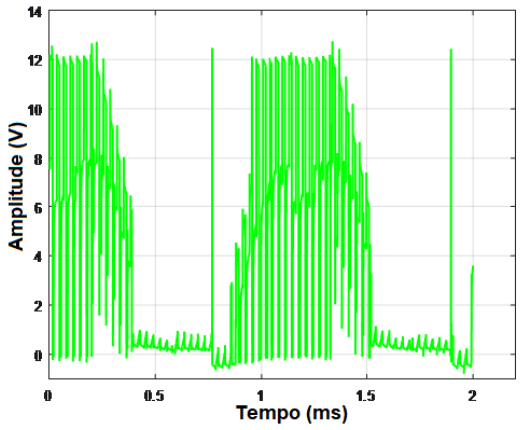
As figuras 35 e 36 mostram que há uma imprecisão e instabilidade no controle do motor visto que tem-se tensões de 24 V com uma parte em 12 V e, além disso, tem-se um degrau no meio da onda, influenciado pelas comutações do circuito de controle.



(a)



(b)

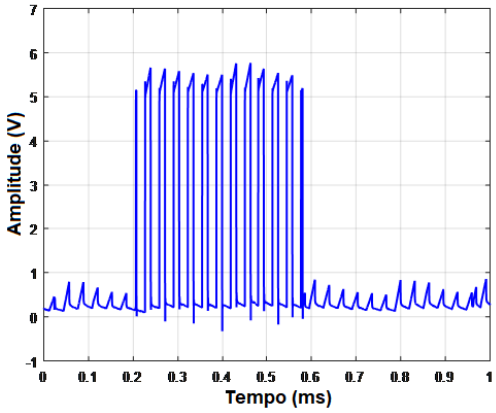


(c)

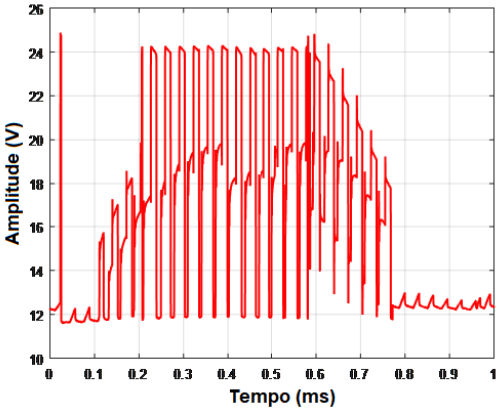
Figura 35 – Tensão de saída do motor (a) Fase A, (b) Fase B, (c) Fase C.

Para acionar um MOSFET de lado alto, a tensão da porta deve ser 10 V a 15 V maior que a tensão da fonte, portanto, essa tensão deve ser maior que a tensão VCC, que geralmente é a tensão mais alta disponível no sistema [1]. É possível evidenciar nas figuras 36 e 37 que o

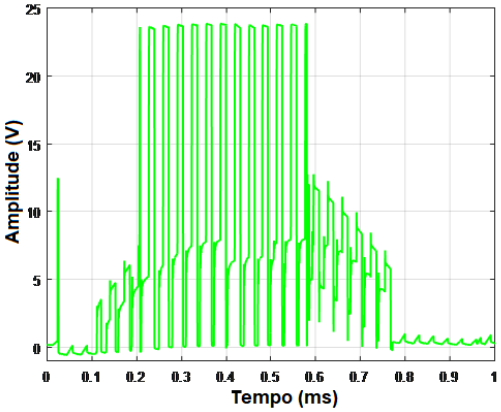
circuito de *bootstrap* eleva a tensão no gate a uma tensão maior que a tensão VCC de 12 V porém, o sistema sem sensores, se mostra bem instável, ocasionando em uma forma de onda bem ruidosa e resultando também em um ruído sonoro durante a rotação do motor.



(a)

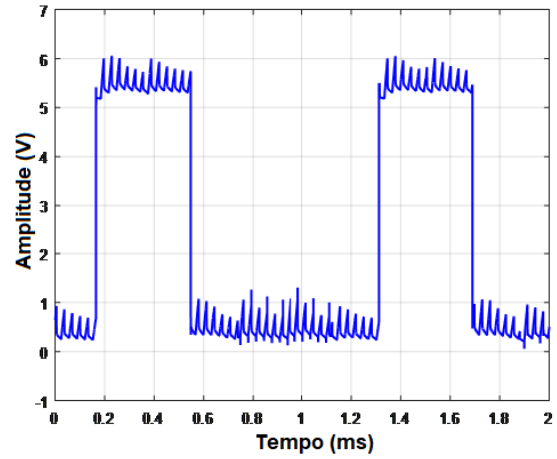


(b)

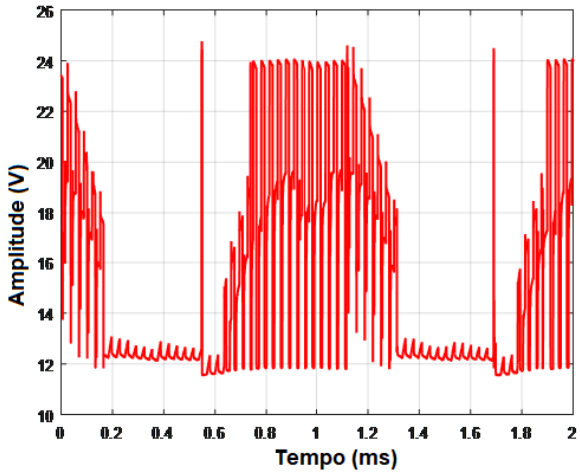


(c)

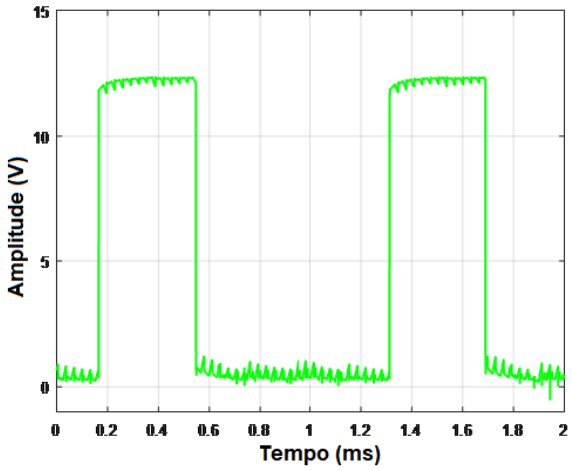
Figura 36 – (a) Entrada PWM Fase A. (b) Tensão de saída do capacitor de *bootstrap* da fase A. (c) Tensões de saída do *gate* do MOSFET *high-side* da fase A – H0.



(a)



(b)



(c)

Figura 37 – (a) Entrada *low-Side* Fase A. (b) Tensão de saída do capacitor de *bootstrap* da fase A. (c) Tensões de saída do *gate* do MOSFET *low-side* da fase A - L0.

3.2 Segundo Protótipo

A saída do sensor é uma saída digital e pode ser nível alto (5 V) ou nível baixo (0 V). Pode-se verificar o deslocamento de fase dos sensores conforme a figura a seguir que foi obtida com um osciloscópio e o motor sendo girado com a mão:

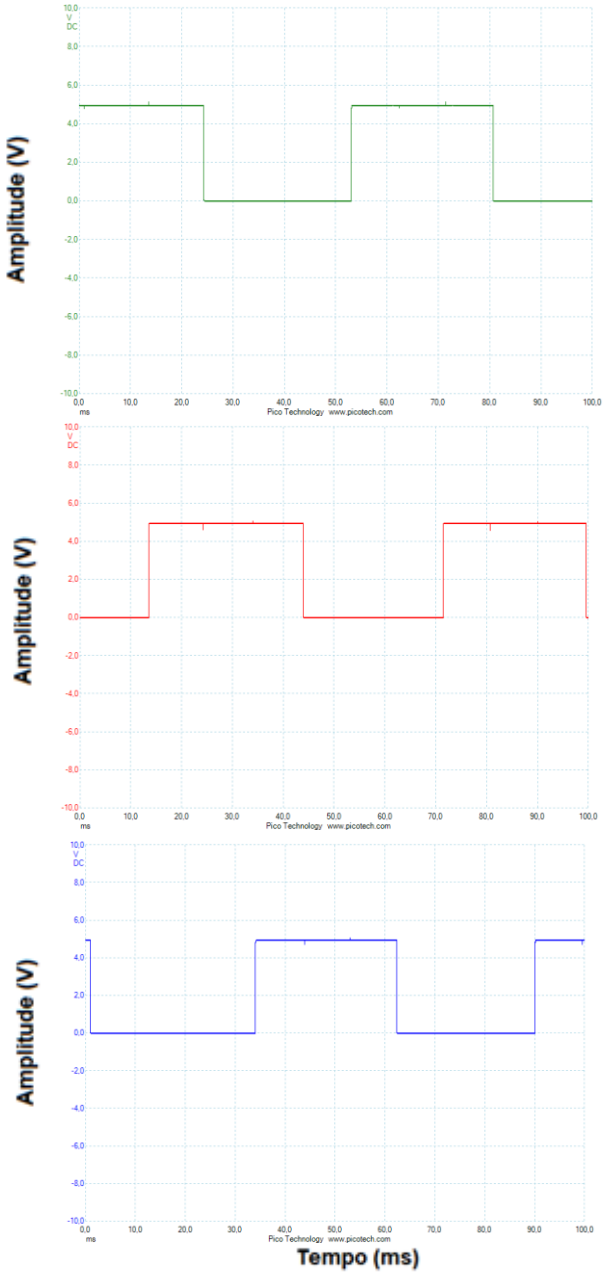


Figura 38 – Saídas do sensores de efeito hall fases A, B e C respectivamente.

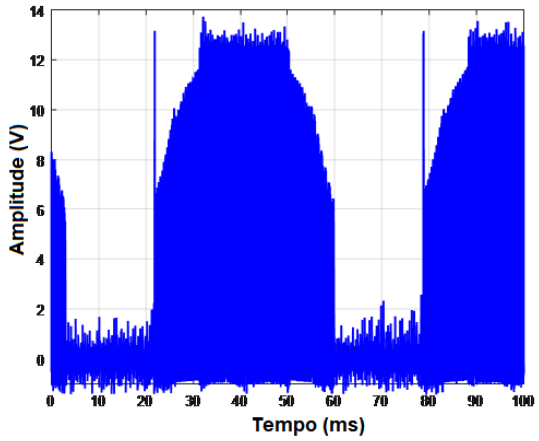
A escolha de usar sensores Hall ou não depende das necessidades do sistema e do motor em questão. Sensores Hall proporcionam um controle mais preciso e *feedback* em

tempo real sobre a posição do rotor, mas requerem *hardware* adicional e complexidade adicional no código.

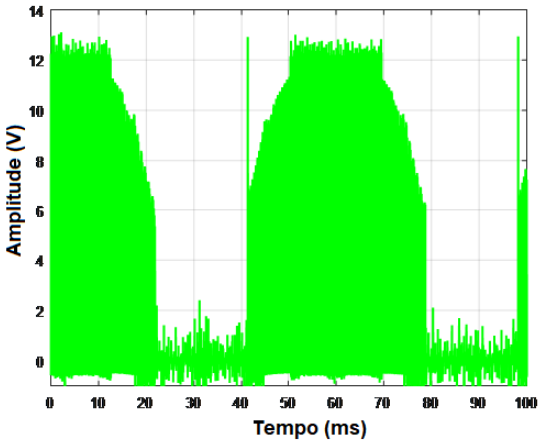
No caso do protótipo 1, onde não são utilizados sensores Hall, é necessário fornecer um impulso inicial para iniciar o movimento do motor e ajudá-lo a encontrar a posição correta. Esse impulso é necessário porque, sem os sensores Hall, o sistema não tem informações diretas sobre a posição do rotor e, portanto, não sabe qual bobina deve ser acionada inicialmente.

No entanto, com sensores Hall, o sinal dos sensores fornece informações precisas sobre a posição do rotor em tempo real. Isso permite que o sistema localize a posição inicial do motor com base nos sinais dos sensores e comece a girar a partir dessa posição, eliminando a necessidade de fornecer um impulso inicial. Portanto, a utilização de sensores Hall simplifica o processo de inicialização do motor e melhora a precisão e o controle do sistema.

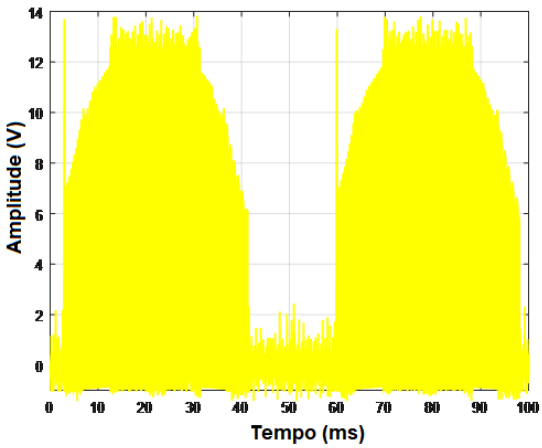
A figura 39 mostra as tensões de saída obtidas com esse protótipo. Lembrando que o ruído observado foi atenuado somente no terceiro protótipo. Para entender melhor o funcionamento do circuito de *bootstrap*, foi aplicado um zoom maior nas ondas de tensão medidas em três intervalos diferentes. O resultado pode ser observado nas figuras 40 a-c, 41 a-c, 42 a-c.



(a)



(b)



(c)

Figura 39 – Tensões de Saída do motor (a) Fase A. (b) Fase B. (c) Fase C.

Sinal PWM:

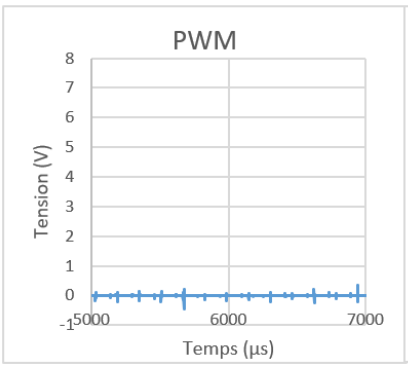


Figura 40a – Sinal antes do PWM

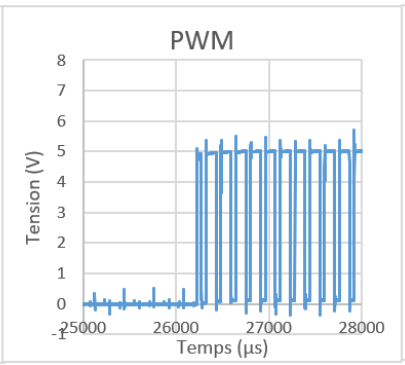


Figura 40b – Sinal durante PWM

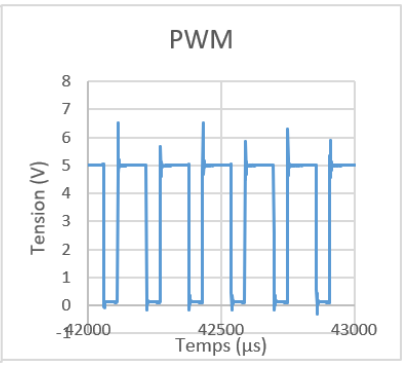


Figura 40c – Zoom durante o PWM

Tensão no capacitor de *bootstrap*:

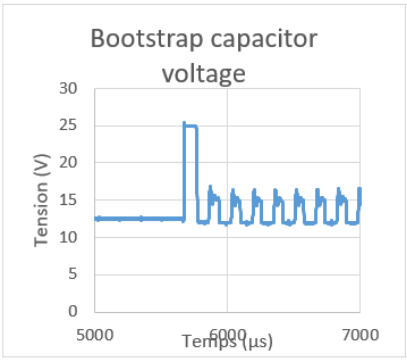


Figura 41a – Sinal antes do PWM

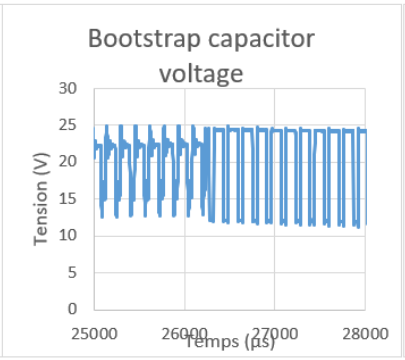


Figura 41b – Sinal durante PWM

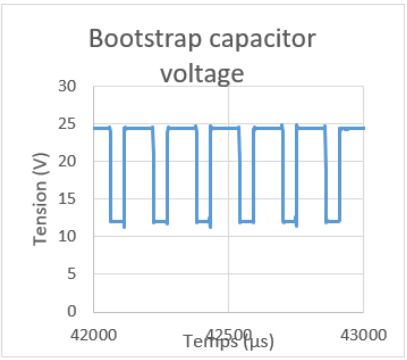


Figura 41c – Zoom durante o PWM

Tensão no *gate* do MOSFET *high-side*:

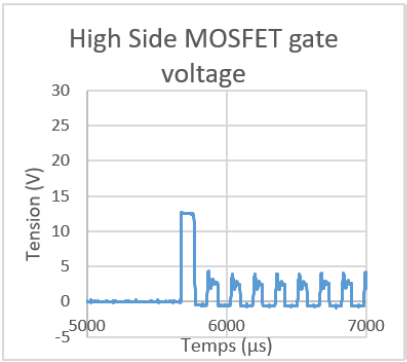


Figura 42a – Sinal antes do PWM

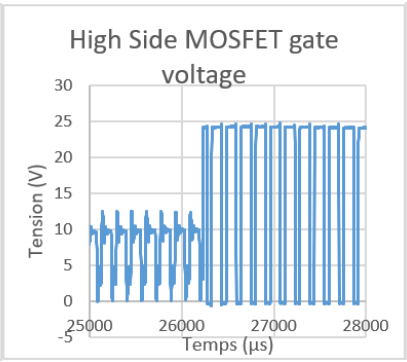


Figura 42b – Sinal durante PWM

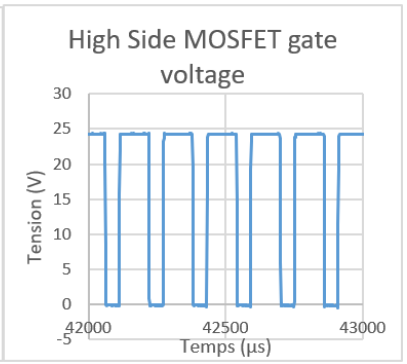


Figura 42c – Zoom durante o PWM

É possível ver que as ondas de tensão da entrada do PWM, do capacitor de *bootstrap* e da tensão de *gate* do lado alto estão em fase uma com a outra. O capacitor de *bootstrap* carrega aumentando sua tensão e fornecendo tensão suficiente para acionar o *gate* do MOSFET de lado alto.

Nas Figuras 41b e 42b vemos que durante o intervalo entre 25 e 26 ms, mesmo sem o sinal PWM ativado (Figura 40b), temos tensão no capacitor de *bootstrap* e no *gate* do MOSFET do lado alto. O que acontece é que nesse momento a fase de análise fica flutuante e o que causa essa tensão é a influência da *back EMF*.

A Figura 43 mostra um gráfico de osciloscópio da tensão do terminal do motor versus o terra do sistema (lado negativo do barramento CC). Sobreposto no gráfico está uma representação da *back EMF* como se veria se o motor fosse simplesmente girado com algum tipo de acionador mecânico. Durante ambas as fases de flutuação, podemos ver claramente que o ponto baixo (durante o tempo de desligamento do PWM) da tensão forma um envelope que segue a *back EMF* [13].

O início do intervalo flutuante do lado esquerdo é marcado por um pico até o nível do barramento CC. Este pico de tensão é causado pela comutação da ponte quando o MOSFET de lado baixo desta fase desliga e a corrente do motor é imediatamente transferida para o diodo de roda livre na parte superior. Como o diodo ligado ao MOSFET *high-side* conduz, a tensão no motor é "fixada" ao barramento CC e a tensão permanecerá lá até que a corrente de fase caia para zero [13].

Este intervalo é referido como compensação ou tempo de desmagnetização. Após o tempo de demag, a *back EMF* real seria negativa, mas, devido aos diodos de roda livre no lado baixo da ponte, a tensão é fixada em uma queda na tensão do diodo abaixo do GND e parece plana. No ponto onde a polaridade da *back EMF* se torna positiva, podemos ver que o lado baixo do envelope de tensão começa a subir, e este é o instante que detectamos como o cruzamento zero da *back EMF* crescente [13].

Durante o tempo PWM ligado, a tensão do terminal é essencialmente a *back EMF* mais Vbus/2. Como esperado, durante a fase condutiva, a tensão terminal é ou Vbus ou zero porque é acionada ativamente [13].

O início da fase flutuante do lado direito é marcado por outro intervalo demag. Nesse caso, a comutação é marcada pelo desligamento do MOSFET do lado alto, fazendo com que a corrente seja transferida através do diodo de roda livre no lado baixo. A tensão será assim

"fixada" ao terra até que a corrente caia para zero. Durante este intervalo, seguimos a tensão esperando pelo cruzamento zero da borda descendente.

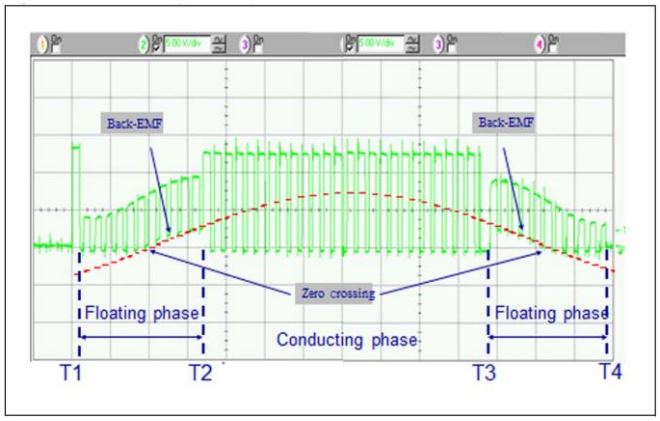


Figura 43 – Plot de uma das fases do motor [13].

O *setup* com um motor BLDC foi utilizado para constatar o efeito da back EMF como mostrado anteriormente. Ao girar o motor observando em um osciloscópio a tensão em A em relação a B, uma tensão senoidal é observada, conforme mostrado na Figura 44. Essa é a força contra-eletromotriz ou *back EMF* do motor. Para um determinado motor, a magnitude e a frequência da *back EMF* são diretamente proporcionais à velocidade do rotor.

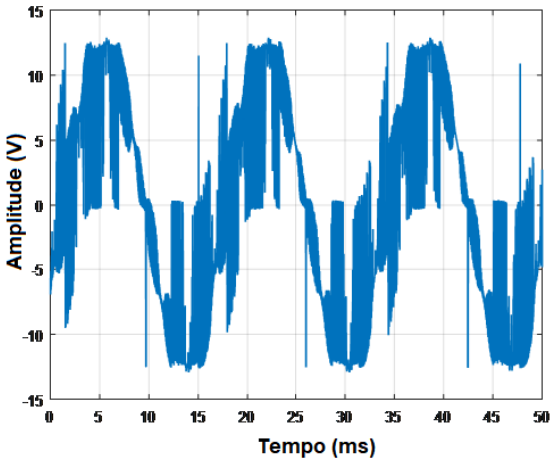


Figura 44 – *Back EMF* entre fases A e B.

A Figura 45 mostra as medições feitas com um osciloscópio que mostram a interferência da *back EMF* na tensão da fase C do motor:

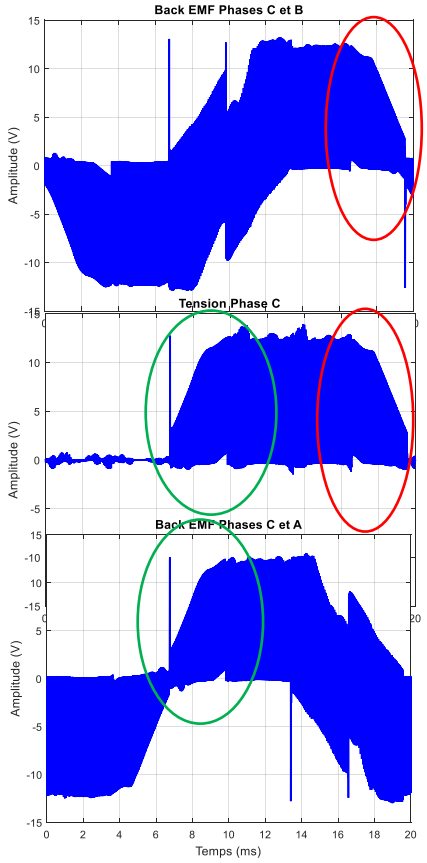


Figura 45 – Influência da back EMF na tensão do motor.

3.3 Terceiro Protótipo

No terceiro protótipo deste trabalho, foi desenvolvido um circuito ESC alimentado por uma bateria de 24 V para o acionamento de um motor BLDC. Nesse protótipo, além do uso de sensores de efeito Hall para o controle preciso da velocidade e direção do motor, foram implementadas estratégias de frenagem regenerativa. A frenagem regenerativa permite a recuperação de energia durante a desaceleração do motor, proporcionando maior eficiência energética e aumentando a autonomia do sistema. Os resultados obtidos nesse protótipo serão apresentados a seguir.

Na figura 46, é possível observar a influência do capacitor colocado na entrada do barramento CC nas fases do motor.

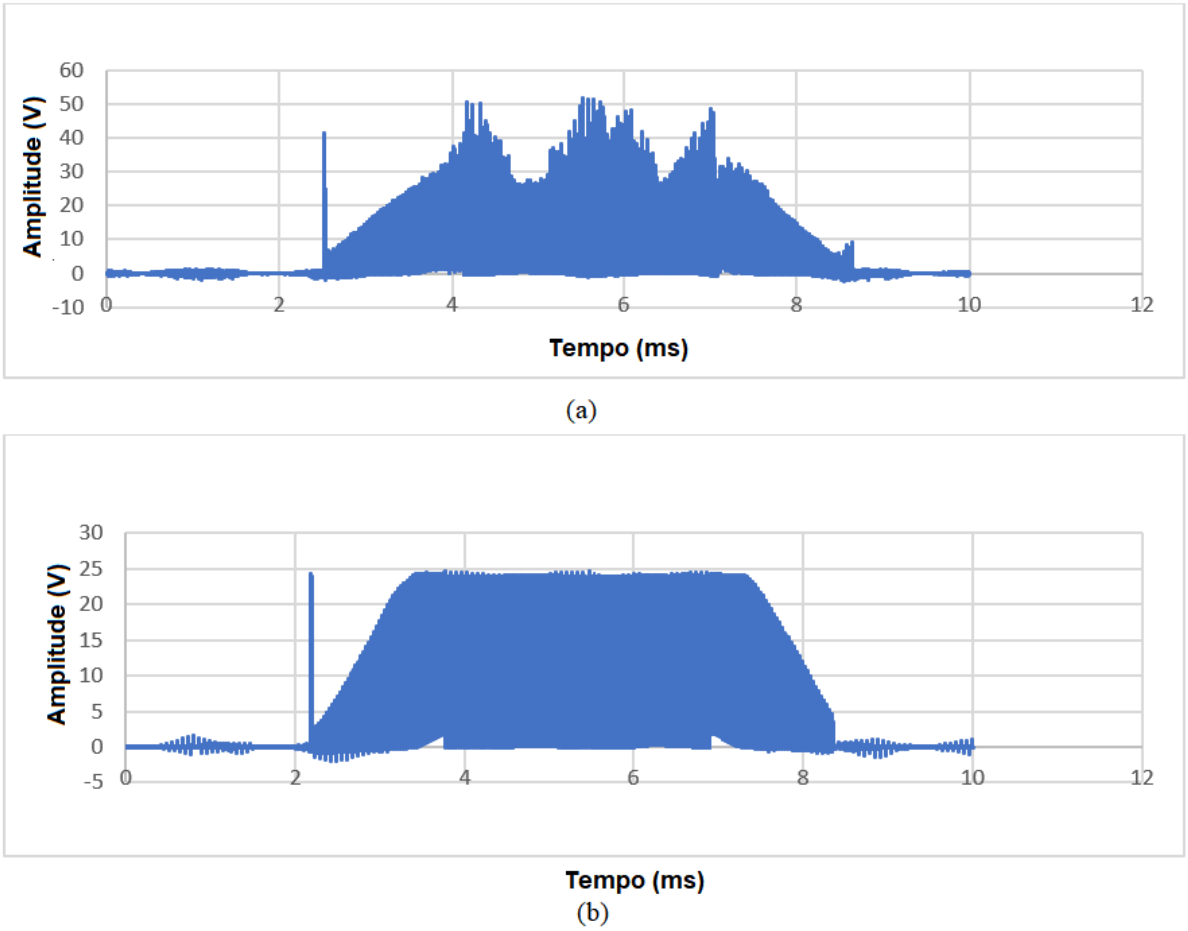
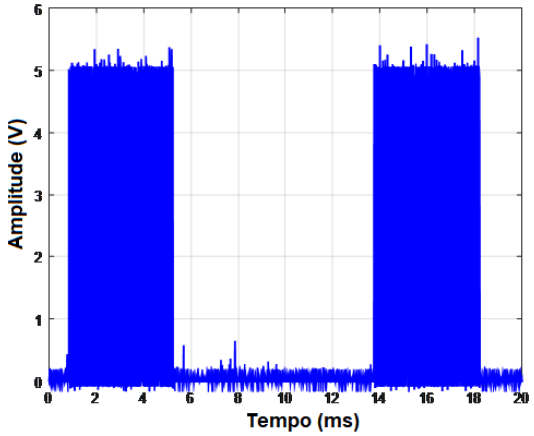
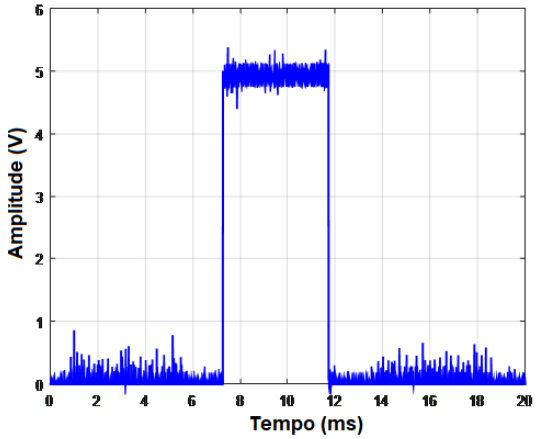


Figura 46 – Influência do capacitor colocado na entrada do barramento CC nas fases do motor. (a) Circuito sem capacitor na entrada do barramento CC. (b) Circuito com capacitor na entrada do barramento CC.

Nas imagens a seguir, vemos o comportamento das entradas do lado alto e do lado baixo dos *drivers* MOSFET no modo motor e no modo de frenagem regenerativa. É possível ver que no modo motor o sistema funciona como nos últimos projetos, o lado alto recebe um sinal PWM defasado em 120º em cada fase e o lado baixo recebe um sinal CC defasado também em 120º em cada fase. Além disso, o lado alto e o lado baixo nunca são ativados ao mesmo tempo na mesma fase. No modo de frenagem regenerativa, o lado alto está desligado e todos os pinos do lado baixo estão habilitados para PWM.

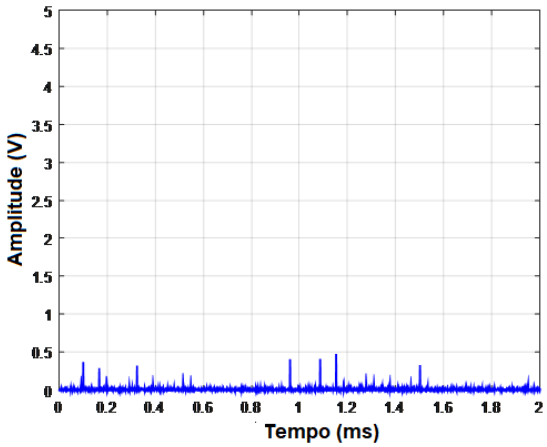


(a)

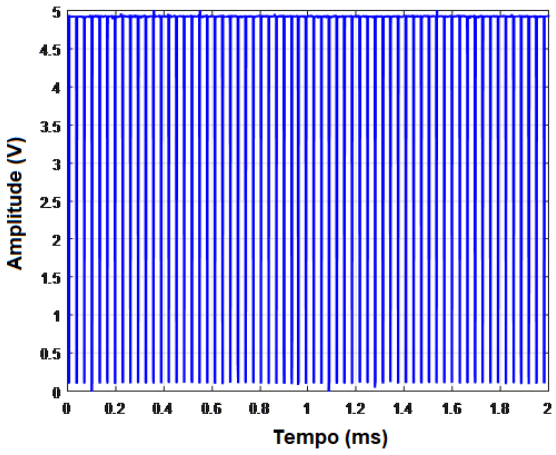


(b)

Figura 47 – Comportamento (a) High-Side (b)Low-Side modo motor.



(a)



(b)

Figura 48 – Comportamento (a) High-Side (b) Low-Side modo frenagem.

Na figura 49 é possível ver a transição entre o modo motor e o modo frenagem regenerativa, em que nota-se um aumento na tensão no barramento CC, que vai diminuindo gradualmente com o tempo. Essa energia pode ser utilizada para recarregar a bateria, mas para isso tanto o *hardware* como o *firmware* devem ser melhorados para potencializar ainda mais essa energia e enviá-la de forma segura para a bateria.

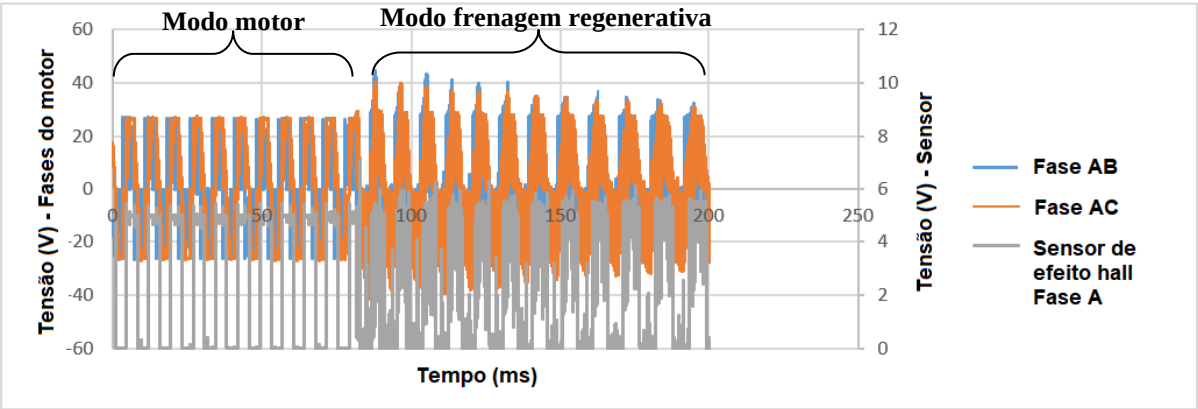


Figura 49 – Transição entre modo motor e modo frenagem regenerativa.

4 Conclusões

Em conclusão, este estudo abordou o acionamento de motores BLDC utilizando controladores eletrônicos de velocidade (ESC) nos modos *sensorless* e *sensored*. Foi comprovado que ambos os métodos são viáveis para o controle do motor, proporcionando flexibilidade e precisão no acionamento. No entanto, é importante ressaltar que ainda existem diversas oportunidades para aprimorar e desenvolver esse projeto.

O primeiro protótipo (*sensorless*) mostrou que a utilização da back EMF para o controle requer técnicas mais avançadas de processamento de sinal e algoritmos de controle, uma vez que os sinais podem ser mais sensíveis a ruídos e imprecisões. Portanto, é importante garantir que o circuito esteja adequadamente projetado e calibrado para compensar a falta dos sensores de efeito Hall.

Uma das melhorias sugeridas é otimizar os códigos do Arduino utilizados no controle do ESC. Isso envolve refinamentos nas estratégias de controle, utilizando técnicas de controle avançado, como controle PID (Proporcional, Integral e Derivativo), para melhorar a resposta e estabilidade do sistema, implementação de algoritmos mais avançados e aprimoramento da eficiência energética durante a frenagem regenerativa.

Outro aspecto a ser considerado é a utilização de um microcontrolador mais robusto, capaz de lidar com maiores cargas de trabalho e oferecer maior poder de processamento. Isso permitiria uma maior flexibilidade no projeto e a incorporação de recursos adicionais, como comunicação sem fio e interfaces de usuário mais avançadas.

Além disso, é fundamental projetar circuitos de proteção adequados para garantir a segurança do sistema e evitar danos aos componentes eletrônicos. Esses circuitos podem incluir proteção contra sobrecorrente, sobretensão, subtensão e curto-circuito, entre outros.

Referências Bibliográficas

[1] T. G. Wilson and P. H. Trickey. “D-C machine with solid-state commutation”. In: IEEE 81.11 (1962), pp. 879–884.

[2] Green, C. R., & McDonald, R. A. (2015). Modeling and test of the efficiency of electronic speed controllers for brushless dc motors. In 15th AIAA aviation technology, integration, and operations conference (p. 3191).

[3] D’SOUZA, Stan, AN957 Sensored BLDC Motor Control Using dsPIC30F2010 – Microchip Technology Inc., 2004.

[4] Srinu, V., Mounica, P., Varalakshmi, S. Kumar S., Teja, K., A Novel Speed Control of Brushless DC Motor Using Arduino UNO R3 and BOT. Asian Journal of Applied Science and Technology (AJAST). 2017

[5] IR International Rectifier. Application Note AN-978 “HV Floating MOS-Gate Driver ICs”. Disponível em: http://www.irf.com/technicalinfo/appnotes/an-978.pdf. Acesso em: [Março 2023].

[6] B. Tefay, et al., "Design of an integrated electronic speed controller for compact robotic vehicles," in Proceedings of the 2011 Australasian Conference on 4/4 ESC Robotics and Automation, 2011.

[7] Khalil Bouguerra(2020). Em InsideEVs, “Como funciona a frenagem regenerativa” [traduzido do francês]. Recuperado de URL: https://insideevs.fr/features/397741/freinage-regeneratif-recuperatif-electrique-hybride/#:~:text=Comme%20son%20nom%20l'indique,de%20fonctionnement%20est%20tr%C3%A8s%20simple. Acesso em: [Março 2023]

[8] Yoong, M. K., et al. "Studies of regenerative braking in electric vehicle." 2010 IEEE Conference on Sustainable Utilization and Development in Engineering and Technology. IEEE, 2010.

[9] R. L. Lin and F. C. Lee, "Single-power-supply-based transformerless IGBT/MOSFET gate driver with 100% high-side turn-on duty cycle operation performance using auxiliary bootstrapped charge pumper," PESC97. Record 28th Annual IEEE Power Electronics Specialists Conference. Formerly Power Conditioning Specialists Conference 1970-71. Power Processing and Electronic Specialists Conference 1972, 1997, pp. 1205-1209 vol.2, doi: 10.1109/PESC.1997.616904.

[10] Application Report “Bootstrap Circuitry Selection for Half-Bridge Configurations” Disponível em: https://www.ti.com/lit/an/slua887/slua887.pdf?ts=1647874006556&ref_url=https%253A%252F%252Fwww.go ogle.com%252F. Acesso em: [Março 2023]

[11] Padmaraja Yedamale, AN970: Using the PIC18F2431 for Sensorless BLDC Motor Control - Microchip Technology Inc., 2015. Recuperado de [https://www.microchip.com/en-us/application-notes/an970] Acesso em: [Março 2023]

[12] (2013). Em Newton C. Braga (Ed.), Como funcionam os sensores de efeito Hall (artigo nº 1050). Recuperado de URL: https://www.newtoncbraga.com.br/index.php/como-funciona/6640-como-funcionam-os-sensores-de-efeito-hall-art1050 Acesso em: [Março 2023]

[13] Application Note AN-4420 “Sensorless six-step BLDC commutation” Disponível no site:

https://www.st.com/content/ccc/resource/technical/document/application_note/26/42/24/a5/f4/7c/4b/90/DM00072008.pdf/files/DM00072008.pdf/jcr:content/translations/en.DM00072008.pdf. Acesso em: [Março 2023]

[14] Bahrami, M.; Mokhtari, H.; Dindar, A. Energy regeneration technique for electric vehicles driven by a brushless DC motor. IET Power Electron. 2019, 12, 3397–3402.

[15] Christian(2017), LOCOduino. Artigo 198 – Os timers (tradução propria do francês). Disponível em: https://www.locoduino.org/spip.php?article198. Acesso em: Março 2023.

[16] Arduino. Referência – Linguagem – função – Math - Map(). Disponível em: https://www.arduino.cc/reference/en/language/functions/math/map/. Acesso em: Março 2023.