UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO DE JOINVILLE
CURSO DE ENGENHARIA MECATRÔNICA

ANGELINE MELCHIORS

PROJETO E DESENVOLVIMENTO DE FIRMWARE PARA MOTORES BLDC.

Joinville
2024

ANGELINE MELCHIORS

PROJETO E DESENVOLVIMENTO DE FIRMWARE PARA MOTORES BLDC.

Trabalho apresentado como requisito parcial para obtenção do título de bacharel em Engenharia Mecatrônica, no Curso de Engenharia Mecatrônica, do Centro Tecnológico de Joinville, da Universidade Federal de Santa Catarina.

Orientador(a): Dr(a). Gian Ricardo Berkenbrock

Joinville
2024

Dedico este trabalho a meus pais, que sempre disseram que eu podia ser o que eu quisesse, e ao Higino, meu companheiro que embarcou comigo no projeto "ser engenheira" e me deu todo o apoio necessário, em diversos aspectos.

AGRADECIMENTOS

Tenho muito a agradecer, sou rodeada de pessoas que me apoiam, me incentivam e me inspiram para buscar sempre ser melhor.
Agradeço muito a minha família, por todo o incentivo, todas as discussões técnicas e conversas motivadoras, e paciência nos finais de semana de estudo. Agradeço ao meu namorado Higino pela seu incentivo, confiança de que eu seria capaz, paciência nas intermináveis horas que passei estudando e suporte neste período de estudante.
Agradeço aos amigos que fiz na faculdade, que tornaram as coisas um pouco mais fáceis e até divertidas.
Aos professores, agradeço especialmente ao meu orientador, professor Gian, por todo o incentivo no estágio, para realização deste trabalho e pelas oportunidades de aprendizado no projeto Rota 2030 e no Laboratório de Sistemas Embarcados. Agradeço ao professor Dalton pelo suporte técnico no TCC. Agradeço aos professores Wagner, Maria Simone e Diego, pela confiança nos projetos que me oportunizaram participar.
Não vou citar um por um, mas diversos professores do Campus, além dos mencionados, serviram de inspiração, pelo seu conhecimento, generosidade e vontade de ensinar.

RESUMO

Os motores BLDC são os mais utilizados em veículos elétricos, pela baixa manutenção e desempenho. Suas principais características são a ausência de escovas e o rotor de imãs permanente. São motores trifásicos de corrente alternada alimentados com corrente contínua, o que torna necessário um circuito de acionamento com inversor de frequência e controlador. Diante disso, o objetivo deste trabalho foi desenvolver um firmware para controle de um inversor de frequência para acionar um motor BLDC. Foi feita revisão da literatura para identificar os atributos de software para sistemas embarcados e compreender o funcionamento do conjunto de hardware a ser controlado. Para isso foi utilizado um microcontrolador STM32F407G, programado em linguagem C usando biblioteca CMSIS e fluxo por interrupção. Dentre outras funções, foi criada uma função para verificar a posição inicial do rotor e inicializar o inversor, e o firmware permite o funcionamento do inversor em modo motor e modo gerador. O controle de velocidade é feito através da variação do ciclo de trabalho em PWM complementar com retificação síncrona. O firmware foi desenvolvido priorizando o desempenho, que foi verificado com auxílio de um microcontrolador auxiliar e potenciômetros para gerar os sinais de entrada e um analisador lógico digital para analisar o comportamento dos sinais de saída enviados ao inversor. O trabalho é organizado em fundamentação teórica, no qual se descrevem os atributos e recursos necessários ao software e o funcionamento do conjunto inversor/motor; em materiais e métodos são definidos os requisitos de projeto, materiais utilizados e feita a modelagem do sistema com definição dos comportamentos; e análise e resultados apresenta as funções desenvolvidas e o resultado dos testes.

Palavras-chave: Sistemas embarcados. Firmware. Inversor de frequência. Motor BLDC.

ABSTRACT

BLDC engines are the most used in electric vehicles, due to their low maintenance and performance. Its main features are the absence of brushes and the permanent magnet rotor. They are three-phase alternating current motors powered by direct current, which requires a drive circuit with a frequency inverter and controller. Therefore, the objective of this work was to develop a firmware to control a frequency inverter to drive a BLDC motor. A literature review was carried out to verify the software attributes for embedded systems and understand the functioning of the hardware set to be controlled. For this, an STM32F407G microcontroller was used, programmed in C language using the CMSIS library and interrupt flow. Among other functions, a function was created to check the initial position of the rotor and initialize the inverter, and the firmware allows the inverter to operate in motor mode and generator mode. Speed control is done by varying the duty cycle in complementary PWM with synchronous rectification. The firmware was developed prioritizing performance, which was verified with the help of an auxiliary microcontroller and potentiometers to generate the input signals and a digital logic analyzer to analyze the behavior of the output signals sent to the inverter. The work is organized on a theoretical basis, in which the attributes and resources necessary for the software and the operation of the inverter/motor set are described; in materials and methods, project requirements are defined, materials used and the system is modeled with definition of behaviors; and analysis and results presents the functions developed and the results of the tests.
Keywords: Embedded systems. Firmware. Variable frequency drive. Brushless DC motor.

LISTA DE FIGURAS

Figura 1 – Componentes básicos de um microcontrolador. 15
Figura 2 – Modalidades PWM. 18
Figura 3 – Sensores *Hall* em motor BLDC. 19
Figura 4 – Exemplos de máquinas rotativas: a) máquina CC b) máquina síncrona. 24
Figura 5 – Enrolamento trifásico máquina com dois polos. 25
Figura 6 – Forma de onda tensão alimentação BLDC. 26
Figura 7 – Comportamento da tensão trapezoidal com PWM. 27
Figura 8 – Possibilidades dos conversores estáticos de energia. 28
Figura 9 – Funcionamento conversor buck. 29
Figura 10 – Inversores monofásicos. 30
Figura 11 – Inversor trifásico ponte completa. 31
Figura 12 – Fluxograma para controle de motor BLDC com sensores. 32
Figura 13 – Representação dos componentes do sistema físico. 37
Figura 14 – Modos de operação do sistema. 38
Figura 15 – Sequência de ativação sensores *Hall*. 40
Figura 16 – Sequência de acionamento das chaves de acordo com a posição do rotor. 40
Figura 17 – Braco de inversor ativo - posição 5. 41
Figura 18 – Braço de inversor em modo gerador. 42
Figura 19 – Foto do conjunto de teste. 43
Figura 20 – Ilustração da montagem circuito de teste. 44
Figura 21 – Estrutura de arquivos. 45
Figura 22 – Fluxo. 46
Figura 23 – Detalhe do PWM complementar. 50
Figura 24 – Modo motor com PWM complementar com retificação síncrona. . . 52
Figura 25 – Modo gerador com acionamento de uma chave. 53
Figura 26 – Fluxo 56
Figura 27 – Comportamento do ciclo de trabalho sob aceleração. 56
Figura 28 – Tempo de execução da função *callback*. 57

LISTA DE QUADROS

Quadro 1 – Especificações elétricas do motor JK42BLS01 36
Quadro 2 – Resumo das ações possíveis 38
Quadro 3 – Posicionamento sensores *Hall* com base na Figura 15 40
Quadro 4 – Sequência de acionamento dos transistores em modo gerador . . 41

LISTA DE CÓDIGOS

1 main.c: Configuração dos periféricos. 45
2 BLDC.h: Protótipos das funções. 46
3 BLDC_users_defines.h: Informações do usuário. 47
4 Função get_position. 48
5 Função set_variables. 48
6 Função set_start. 48
7 Função position_next. 50
8 Macro para ativação simultânea das bobinas. 51
9 Configuração modo motor. 51
10 Cálculo ciclo de trabalho modo motor. 53
11 Cálculo ciclo de trabalho modo gerador. 54

LISTA DE ABREVIATURAS E SIGLAS

AC	*alternate current* - corrente alternada
ADC	*analog to digital converter* - conversor analógico digital
BLDC	*brushless direct current* - corrente contínua sem escovas
CAN	*control area network*
CH	canal gerador PWM
CHN	canal complementar gerador PWM
CMSIS	cortex microcontroller software standard
DC	*direct currente* - corrente contínua
EFICEM	Equipe de Eficiência Energética
FCEM	força contraeletromotriz
GPIO	*general purpose input/output* - entradas e saídas de propósito geral
HAL	*hardware abstraction layer* - camada de abstração de hardware
I/O	*inputs and outpust* - entradas e saídas
ISR	*interrupt service routine* - rotina de serviço de interrupção
DMA	*direct memory access* - acesso direto à memória
PCB	*printed circuit board* - placa de circuito impresso
PWM	*pulse width modulation* - modulação de largura de pulso
RPM	rotações por minuto
RTOS	*real time operation system* - sistema operacional de tempo real
SO	sistema operacional
UART	*universal asynchronous receiver-transmitter* - receptor-transmissor universal assíncrono
USB	*universal serial bus*
μC	microcontrolador
UFSC	Universidade Federal de Santa Catarina

SUMÁRIO

1 INTRODUÇÃO 12
1.1 OBJETIVO 13
1.1.1 Objetivo Geral 14
1.1.2 Objetivos Específicos 14

2 FUNDAMENTAÇÃO TEÓRICA 15
2.1 SISTEMAS EMBARCADOS 15
2.1.1 Microcontrolador 15
2.1.2 PWM 17
2.1.3 Sensores *Hall* 18
2.2 ARQUITETURA DE CÓDIGO 19
2.2.1 Componentes do código 20
2.2.2 Fluxo de programação 21
2.2.3 Atributos de qualidade 22
2.3 MOTORES ELÉTRICOS 23
2.3.1 Motores BLDC 24
2.3.2 Geradores elétricos e frenagem regenerativa 27
2.4 CONVERSORES E INVERSORES 28
2.4.1 Conversores CC-CC 29
2.4.2 Inversores CC-CA 29
2.5 DESIGN FIRMWARE CONTROLE BLDC 31
2.6 TRABALHOS RELACIONADOS 32

3 MATERIAIS E MÉTODOS 35
3.1 REQUISITOS DE PROJETO 35
3.2 MATERIAIS 36
3.2.1 Motor 36
3.3 PROJETO 36
3.3.1 Sistema físico 37
3.3.2 Modos de operação 37
3.3.3 Sequência de chaveamento BLDC 39
3.3.4 Sequência modo motor 39
3.3.5 Sequência modo gerador 41
3.3.6 Definição do ciclo de trabalho 42
3.4 ESTRUTURA PARA TESTES 43

4 ANÁLISE E RESULTADOS 45

4.1 ORGANIZAÇÃO DOS ARQUIVOS 45
4.1.1 Fluxo Principal 45
4.1.2 Funções para Controle do Motor 46
4.2 CONFIGURAÇÕES E FUNCIONAMENTO 47
4.2.1 PWM 49
4.2.2 Sequência de acionamento das bobinas 50
4.2.3 Modo de operação 51
4.2.4 Cálculo do ciclo de trabalho 53
4.2.5 Monitoramento tensão e corrente 54
4.2.6 Freio regenerativo 55
4.2.7 Função *callback* 55
4.2.8 *Watchdog* 56
4.3 CONSIDERAÇÕES FINAIS 57

5 CONCLUSÕES 58

REFERÊNCIAS 60

APÊNDICE A 63

1 INTRODUÇÃO

Uma das medidas tomadas na tentativa de conter o aquecimento global é buscar substituir os veículos movidos a combustíveis fósseis por veículos elétricos, juntamente com a busca por fontes de energia renováveis para fornecimento de energia elétrica. Com isso tem-se uma redução na emissão de gás carbônico, o que fez com que os veículos elétricos ganhassem popularidade na última década (SUBBARAO et al., 2024).

No Campus da UFSC Joinville, existem diversas equipes de competição formadas pelos estudantes, e a Equipe de Eficiência Energética (EFICEM) compete na Shell Eco-marathon, patrocinada pela Shell. Conforme Shell Eco-marathon (2024) é um dos maiores programas de engenharia de eficiência energética para estudantes. Uma das modalidades disputadas é a eficiência energética em veículos alimentados por bateria elétrica, e no regulamento em seu artigo 67 consta que o controlador do motor, e quando houver PCB (placa de circuito impressa) incorporada ao controlador, além do software, estes devem ser desenvolvidos especificamente para a competição (Shell Eco-marathon, 2024).

A equipe de competição conta com uma placa controladora, descrita em CHRIST (2023) que atende aos requisitos da competição, porém existe necessidade de melhorar o software. Considerando que a desempenho do veículo é dada pela eficiência do motor, escolhido pela equipe de competição, e pela eficiência do sistema de controle (SUBBARAO et al., 2024), surge o problema de pesquisa, que consiste em desenvolver um firmware para controle do motor utilizado pela equipe, atualmente um motor de imãs permanentes BLDC (brushless DC) trifásico.

A metodologia de trabalho consiste em pesquisa bibliográfica sobre sistemas embarcados e boas práticas, o funcionamento dos motores, seu controle e aspectos relacionados. Em trabalhos relacionados serão buscadas estratégias e verificadas formas de implementação que funcionaram ou não para nortear o desenvolvimento do firmware. Após as pesquisas de base concluídas, a próxima etapa é o desenvolvimento de um modelo e definição dos componentes, partindo para o desenvolvimento do software seguido de testes.

O motor BLDC é um motor de corrente alternada trifásico alimentado por corrente contínua, necessitando de um inversor de frequência para fazer a conversão CC-AC e fornecimento das três fases. O inversor, controlado por firmware, faz o acionamento das fases que precisam ser acionadas e alimenta o motor com tensão modulada por largura de pulso (PWM). A frequência do PWM é fixa, o controle da velocidade é feito através do ciclo de trabalho: quanto maior o ciclo de trabalho, maior a

tensão fornecida e maior será a velocidade de rotação do motor. A posição das bobinas a serem acionadas é dada pela leitura de sensores *Hall*, acionados pela proximidade do rotor composto de imãs, e pela combinação dos sensores o controlador identifica a localização do rotor e comanda o acionamento das bobinas seguintes para manter o giro. Além dos sensores *Hall*, o firmware prevê a leitura de sensores de aceleração, tensão da bateria e corrente. A definição do ciclo de trabalho se dá proporcional à aceleração.

O controle é um sistema embarcado, motivo pelo qual trabalha com limitação de memória e precisa ter baixo consumo. O desenvolvimento é feito com um microcontrolador STM32F407G, utilizando biblioteca HAL e programado em fluxo de interrupção, utilizando diversos recursos da placa, como PWM, GPIOs (general purpose input / output), DMA (direct memory access), ADC (analog to digital converter), timers, entre outros.

No desenvolvimento é dado ênfase à modificabilidade e ao desempenho. A modificabilidade é a facilidade com que o sistema pode ser modificado e adaptado, uma vez que pode atender à diferentes equipes de competição, ser utilizado com motores, inversores e baterias diferentes, e o código foi dividido em funções menores, chamadas pela função principal, que consiste no *callback* de uma interrupção, o que permite um melhor entendimento caso seja modificado por outro desenvolvedor. O desempenho consiste na capacidade do software desempenhar aquilo para o que foi desenvolvido, que é o acionamento do hardware. Esse desempenho será verificado com apoio do analisador lógico, verificando as respostas do sistema em relação às entradas fornecidas.

Como proposta de melhoria e na intenção de gerar vantagem competitiva para as equipes da UFSC, o projeto também implementa o freio regenerativo, para melhorar o desempenho no quesito de consumo energético, com ciclo de trabalho controlado pela corrente. O termo motor é utilizado de forma genérica, para referir-se à máquina elétrica. Quando o texto se referir ao modo motor, em oposição ao modo gerador, será especificado como *modo motor*.

Para verificar o funcionamento do firmware, foram realizados testes de simulação fornecendo valores para simular as posições dos sensores *Hall* e verificadas as saídas geradas, com alteração nos valores analógicos dos sensores. As saídas foram lidas com analisador lógico digital e apresentaram o comportamento esperado, tanto em formato como em tempo de resposta.

1.1 OBJETIVO

Para resolver a problemática de desenvolver um firmware para controle do motor BLDC trifásico, propõe-se neste trabalho os seguintes objetivos:

1.1.1 Objetivo Geral

O objetivo deste trabalho é desenvolver um software para acionamento e controle de motor de imãs permanentes BLDC.

1.1.2 Objetivos Específicos

Os objetivos específicos são:

- compreender o funcionamento do conjunto de hardware que será controlado pelo dispositivo;
- definir os requisitos que devem ser atendidos pelo software;
- desenvolver o firmware para acionamento de motor BLDC;
- testar o software para verificar o seu funcionamento.

O trabalho está organizado da seguinte forma: o capítulo 2 - fundamentação teórica - aborda os principais temas cujo entendimento é necessário, como sistemas embarcados, arquitetura de código, motores elétricos, conversores e inversores de frequência, e o resumo de alguns trabalhos relacionados. No capítulo 3 - materiais e métodos - são definidos os requisitos do projeto, recursos de software e hardware utilizados, os modelos do sistema e definição de comportamentos. O capítulo 4 - análise e resultados traz a explicação das funções implementadas e os resultados dos testes. Por fim, o capítulo 5 - Conclusões, verifica o atendimento dos objetivos e resumo os resultados obtidos, juntamente com sugestões para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo aborda os conceitos cuja compreensão é necessária para p desenvolvimento deste trabalho, como características de software para sistemas embarcados, funcionamento do motor BLDC e do inversor de frequência a ser controlado, e trabalhos relacionados, que traz alguns trabalhos que trouxeram informações e ideias relevantes.

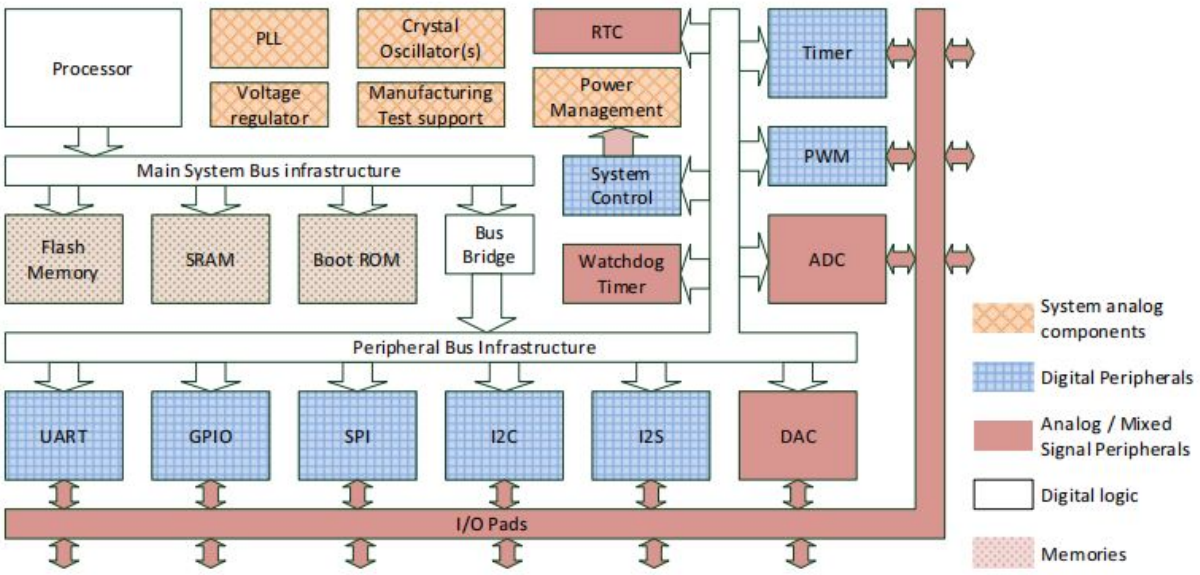
2.1 SISTEMAS EMBARCADOS

Os sistemas embarcados são sistemas eletrônicos microprocessados desenvolvidos para uma aplicação específica (ALMEIDA et al., 2023).

2.1.1 Microcontrolador

Os microcontroladores são chips que contém processadores desenvolvidos para aplicações específicas. São programáveis e contém diversos blocos funcionais - os periféricos - para permitir sua interação com o ambiente (YIU, 2020). A coleta dos dados dos periféricos e o controle das interfaces do microcontrolador é feita pelo software que roda no processador. Os principais componentes de um microcontrolador simples são apresentados na Figura 1.

Figura 1 – Componentes básicos de um microcontrolador.



Fonte: (YIU, 2020)

Nesta seção são definidos alguns periféricos apresentados na Figura 1 e mais alguns recursos dos microcontroladores cujo uso depende de escolhas do programador e podem ser aplicados na implementação do inversor de frequência e seus dispositivos auxiliares:

- ADC - *analog to digital converter*: converte sinal analógico em sinal digital;
- DAC - *digital to analog converter*: converte sinal digital em analógico;
- DMA - *direct memory access*: é uma abordagem de interface que transfere dados direto da/para a memória, pode ser utilizado para implementar aquisição de dados de alta velocidade, quando é importante ter banda larga ou baixa latência;
- Gerenciador de interrupções: controla o aninhamento de interrupções, o que permite interromper interrupções de prioridade mais baixa, garantindo que interrupções com maior prioridade sejam executadas com menor latência;
- GPIO - general purpose Input/Output: periféricos com uma interface de dados paralela, para permitir a leitura de sinais e o controle de dispositivos externos. Os pinos da GPIO são programáveis, podendo ser inputs e outputs digitais e analógicos e configurados para protocolos específicos. Os sensores *Hall*, descritos na seção 2.1.3 passam a informação para o microcontrolador via pinos de entrada;
- Protocolos de comunicação - Os microcontroladores usualmente possuem suporte para implementação de diversos protocolos de comunicação serial, como UART, SPI, I2C, CAN, RS232, USB e podem ser utilizados adaptadores para permitir modalidades não previstas inicialmente;
- PWM (Pulse with modulation): periférico que tem como sinal de saída uma onda com ciclo de trabalho (duty cycle) programável. A saída PWM é digital, mas pode ser utilizada como saída analógica, conforme o tempo de resposta do sistema controlado, conforme explicado na seção 2.1.2;
- SysTick: permite acionar interrupções programadas para ocorrerem em um intervalo regular de tempo, controladas pelo incremento monotônico do clock;
- Timers: Podem ser configurados, de maneira independente, para executar operações periódicas ou de execução única (one-shot), permitindo acionar interrupções;
- Watchdog: dispositivo com timer programável para assegurar que o processador esteja rodando um programa. Se o programa falha, após um tempo predefinido o dispositivo pode reiniciar o processador ou emitir alerta de erro. Especialmente útil em programação bare-metal, para evitar que alguma aplicação bloqueie o laço (loop) principal (ALMEIDA et al., 2023; LACAMERA, 2018; VALVANO, 2012; VALVANO, 2017; YIU, 2020).

2.1.2 PWM

O recurso de PWM é um recurso fundamental no controle do inversor de frequência. Quando a tensão de alimentação é fixa e se faz necessária variá-la, como no acionamento de um motor, um recurso é utilizar o controle da modulação por largura de pulso (PWM). Isso é feito variando a duração da aplicação da tensão de entrada à máquina, e que se feito rapidamente funciona como se fosse alimentação contínua (KRISHNAN, 2015). Rashid (2014) especifica que a tensão média é dada pela equação 1, na qual V_{dc} é a tensão de alimentação e T é o período de operação do conversor.

$$V_o = \frac{t_{on} \times V_{dc}}{T} \tag{1}$$

Dessa forma, é possível controlar a tensão de saída através do ciclo de trabalho do conversor, dado pela equação 2 uma vez que o período T é a soma de T_{on} com T_{off}.

$$d = \frac{t_{on}}{T}. \tag{2}$$

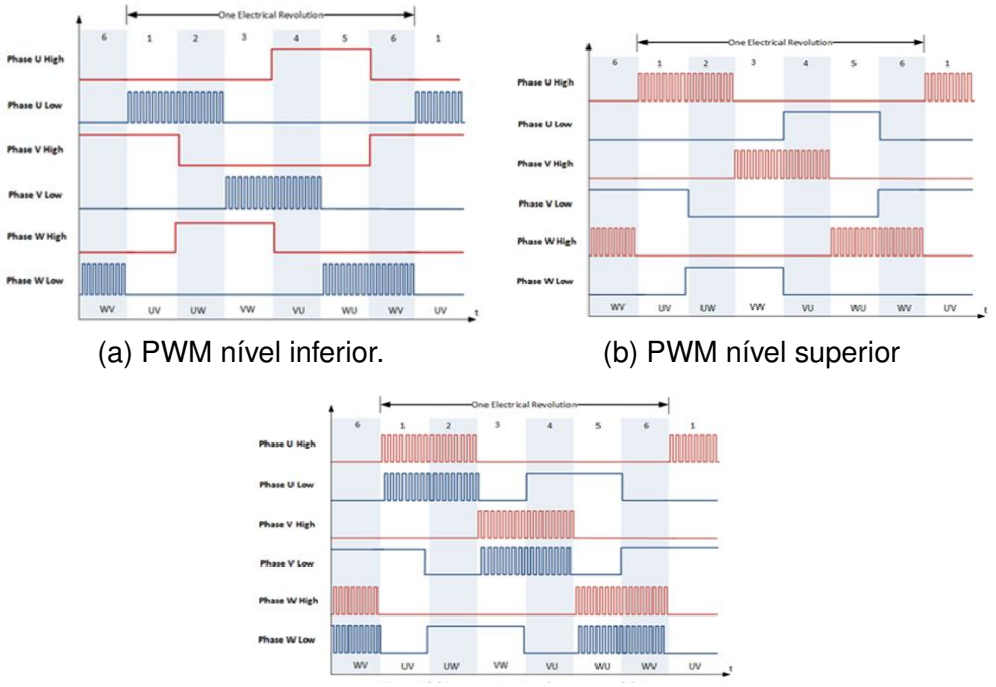
A frequência do PWM deve ser adequada para atender aos requisitos de frequência elétrica e as características de chaveamento do circuito de acionamento. Buchi (2012) afirma que a frequência de chaveamento pode ser configurada como parâmetro, e deve considerar a indutância das bobinas, a velocidade de rotação e o número de polos do motor, uma vez que a frequência da corrente precisa ser maior que a rotação do motor e a frequência de pulso deve ser ainda maior. Conforme o autor, para motores de alta indutância a frequência deve ser menor do que para motores de baixa indutância e é preferível configurar o chaveamento em frequência mais baixa, para evitar as perdas de comutação.

A detecção do sinal de pulso se dá pela borda de subida do pulso, não sendo possível trabalhar com ciclo de trabalho de zero ou de 100%. Mesmo um ciclo de trabalho de 99% não é viável em função de atraso no chaveamento, por isso, na prática, a parte mais estreita do pulso deve ser maior que 250ns. Se o ciclo de trabalho for zerado, o motor para (XIA, 2012). Considerando que os dispositivos de chaveamento necessitam de um tempo mínimo para ligar e desligar, o ciclo de trabalho pode ser controlado apenas em uma faixa de valores, limitando assim os valores de potência mínima e máxima (RASHID, 2014).

Infineon Technologies AG (2017) menciona três tipos de PWM suportados pelos BLDCs: Modulação no nível superior, modulação no nível inferior, e modulação no nível superior com retificação síncrona, que podem ser vistos na Figura 2:

No PWM com modulação no nível superior, a comutação é aplicada somente às chaves da parte superior do inversor enquanto as da parte inferior permanecem constantes. Na modulação da parte inferior, a comutação é aplicada somente às chaves

Figura 2 – Modalidades PWM.



(a) PWM nível inferior.

(b) PWM nível superior

(c) PWM nível superior com retificação síncrona.

Fonte: (INFINEON TECHNOLOGIES AG, 2017).

inferiores. Na modulação no nível superior com retificação síncrona, a modulação é aplicada nas chaves do lado superior com comutação complementar nas chaves do lado inferior (INFINEON TECHNOLOGIES AG, 2017). Segundo os autores, esta última modalidade ajuda a reduzir as perdas nos diodos.

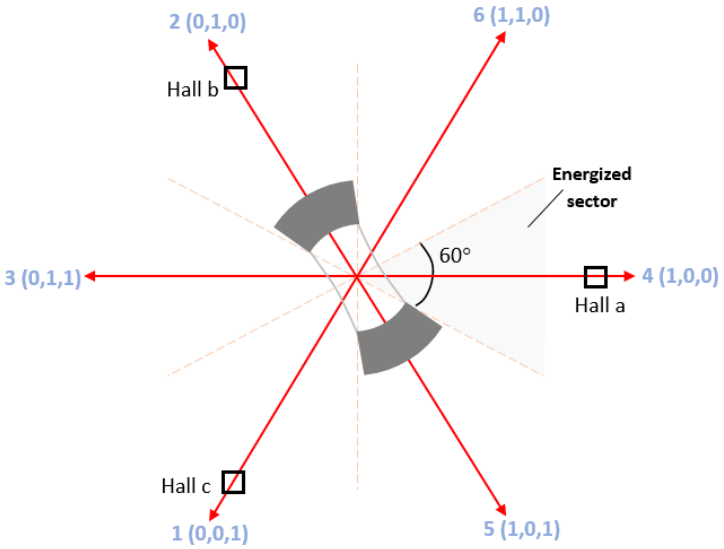
Descrever o processo de geração de PWM foge do escopo deste trabalho, uma vez que será utilizado microcontrolador com a função PWM.

2.1.3 Sensores *Hall*

Para detectar a posição do rotor e melhorar o seu controle, é possível utilizar sensores. Os mais comumente utilizados são os sensores de efeito *Hall*, assim o controlador sabe exatamente a posição do rotor e sua velocidade, podendo agir imediatamente em caso de desvio do esperado (BUCHI, 2012).

O efeito *Hall* consiste na reação de uma partícula semicondutora atravessada por uma corrente elétrica sobre um campo magnético perpendicular à sua superfície. Isso gera uma diferença de potencial elétrico na partícula, que é perpendicular, simultaneamente, à corrente e ao campo magnético. Essa diferença de potencial elétrico é chamada de tensão de *Hall*. A polaridade da tensão inverte segundo a

Figura 3 – Sensores *Hall* em motor BLDC.



Fonte: (The MathWorks, 2021)

direção da corrente, de forma que, posicionado junto a um rotor de ímãs permanentes, o sensor *Hall* pode detectar precisamente a posição dos polos, informando a tensão *Hall* (KENJO; NAGAMORI, 1985).

Na Figura 3 é possível ver a disposição de três sensores *Hall* em BLDC. Cada sensor é posicionado junto a uma das bobinas e as seis identificações no formato (a, b, c) indicam a leitura de cada bobina, quando o polo norte do rotor está apontado para aquela direção. Os sensores são conectados a portas de entrada da GPIO do microcontrolador.

Em algumas situações a limitação de espaço pode dificultar o uso de sensores de posição. Existem técnicas que permitem controlar a posição do motor indiretamente através da medição de tensão ou corrente, porém para este trabalho será utilizado o controle com sensores *Hall*.

2.2 ARQUITETURA DE CÓDIGO

O desenvolvimento de software para sistemas embarcados precisa levar em consideração diversas questões, além do atendimento da aplicação para o qual foi desenvolvido, como limitação de espaço em memória, RAM insuficiente para lidar com buffers grandes, processador lento para a aplicação, necessidade de baixo consumo de energia. Depurar um firmware de software embarcado também é complexo, pois geralmente não há interface para interação com humanos. Isso tudo precisa ser considerado no desenvolvimento do firmware, para evitar problemas quando o sistema estiver em uso, exigindo técnicas e rotinas de programação diferentes das utilizadas

em computadores convencionais (ALMEIDA et al., 2023; BASS; CLEMENTS; KAZMAN, 2021; LACAMERA, 2018; YIU, 2020).

2.2.1 Componentes do código

Um projeto para sistema embarcado precisa conter todo o código-fonte, bibliotecas de terceiros, dados, scritpts e automações para construir a imagem final. Entre os elementos principais, temos (LACAMERA, 2018):

- Drives para abstração de hardware: fornecidos pelo fabricante do microcontrolador, para permitir o controle do dispositivo. Normalmente são distribuídos no formato de código-fonte ou bibliotecas, que podem ser usadas como referência para abstração de hardware e adaptadas para o controlador final;
- Middleware: Bibliotecas de terceiros, protocolos de comunicação, rede e mensageria, RTOS (sistemas operacionais de tempo real), entre outros. São soluções prontas e já testadas que podem ser integradas ao projeto. Necessário atentar para consumo de memória, tamanho do código e compatibilidade das licenças;
- Código da aplicação: código principal, coordena a execução desde o nível mais alto do projeto até os demais módulos envolvidos. Bem desenvolvido permite visualizar claramente os blocos componentes do sistema, suas inter-relações e o tempo de execução dos componentes.

Sobre a organização de um código para sistemas embarcados, (LACAMERA, 2018) orienta que seja estruturado da seguinte forma:

- manter bibliotecas autossuficientes em diretórios separados para facilitar atualizações;
- makefiles e outros scripts podem ficar no diretório raiz;
- o código da aplicação deve ser curto e sintético, acessando outros módulos para realizar as funcionalidades;
- módulos funcionais descrevem o processo e ocultam detalhes da implementação;
- módulos pequenos, autossuficientes e abstratos, o que facilita os testes;
- separar lógica dos componentes da aplicação da implementação de hardware, para melhorar portabilidade;
- abstrair demais requer esforço e recursos, buscar equilíbrio.

Martin (2017) define componentes como as menores unidades que podem ser implementadas como partes de um sistema e estabelece alguns princípios para organizá-los, dentre os quais a coesão e o acoplamento. O autor considera classes e diz que, para as classes poderem ser agrupadas em um mesmo componente, devem pertencer a um grupo coeso, para poderem ser atualizadas em conjunto, que mudem ao mesmo tempo, e pelas mesmas razões. Da mesma forma, não se deve agrupar classes que não tenham essa afinidade, para não haver atualizações desnecessárias.

O acoplamento de componentes considera as relações de dependência entre eles. Para evitar problemas a relação deve ser unidirecional, com os componentes dispostos como nós em um grafo, não deve ser possível retornar a um componente de nível superior, isso evita ciclos de dependência. Caso seja necessário, uma solução seria criar um novo componente com as classes comuns. Outro aspecto diz respeito à volatilidade dos componentes para permitir alterações no software, porém deve-se ter cuidado para não ter componentes estáveis dependendo de componentes voláteis, isso tornaria a aplicação difícil de atualizar. Por outro lado, componentes com muitos dependentes precisam ser estáveis, para não ser necessário atualizar todas as suas dependências (MARTIN, 2017).

2.2.2 Fluxo de programação

Uma abordagem para desenvolvimento do código da aplicação é fazer bare-metal, ou seja, programar diretamente no hardware, sem sistema operacional. A aplicação é construída baseada em um laço infinito que não pode ser suspenso, apenas pelas chamadas de interrupções. Como sua execução é sequencial, qualquer função chamada pelo laço principal deve retornar o mais rapidamente possível. O ideal é que cada componente que interaja com o laço conforme o paradigma orientado a eventos. A vantagem da programação bare-metal é que toda memória está acessível a qualquer função no código (LACAMERA, 2018).

Em sistemas maiores nos quais é necessário lidar com diversos periféricos e eventos simultaneamente é conveniente fazer uso de um sistema operacional (SO) para coordenar e sincronizar as atividades (LACAMERA, 2018). Almeida et al. (2023) também propões o uso de algum SO para evitar erros, facilitar a organização e melhorar a reutilização do código. O autor explica que o SO funciona como uma camada de abstração e provê funcionalidades para aplicações de alto nível, para que a aplicação não precise ser alterada quando houver mudança no hardware. Com o uso de um SO, as aplicações são implementadas como processos gerenciados pelo núcleo do SO, que os executa em uma sequência definida pelos escalonadores.

De forma intermediária, a programação pode ser feita com o uso de uma camada de abstração de Hardware (HAL - Hardware Abstraction Layer).No caso dos dispositivos da ARM, toda a linha Cortex-M suporta a Cortex Microcontroller Software Interface Standard (CMSIS), o que faz com que os sistemas desenvolvidos com a CMSIS funcionem em todos os μC Cortex-M (LACAMERA, 2018).

Yiu (2020) coloca que existem diferentes maneiras de estruturar o fluxo de um programa para microcontroladores e lista os que considera principais:

- método de *Polling*, ou super loop: usualmente implementado em um laço, dentro do qual o microcontrolador verifica continuamente as entradas para verificar se alguma ação é requerida (LIPOVSKI, 2004). Yiu (2020) explica que quando a

demanda é complexa ou exige alto desempenho, *pollings* não são aplicáveis, além do elevado consumo de energia, uma vez que o *polling* roda continuamente, mesmo quando nenhuma ação é requerida.

- método de interrupção: interrupção é a transferência da execução do software em resposta a um evento de hardware (trigger) assíncrono com a execução atual. Esses eventos podem ser externos, recebidos por algum dispositivo de entrada, como UART, ou internos, como falta de memória ou timer periódico (VALVANO, 2012). Lipovski (2004) define as interrupções como algum evento que exige processamento e saída ou sinal de erro, executado em uma subrotina, chamada de rotina de interrupção - ISR (do inglês *Interrupt Service Routine*). Isso interrompe o processamento que está sendo realizado e depois o retoma exatamente onde parou.
- combinação de *polling* e interrupção: com o uso de variáveis as informações podem ser transferidas entre as ISR e os processos do loop (YIU, 2020). Almeida et al. (2023) coloca que as interrupções podem ser disparadas por fim de conversão AD, estouro de contagem ou comparação de valor de timer, mudanças nos valores de portas I/O, fim de transmissão, recepção de valor ou colisão de mensagens em portas seriais, entre outros.
- processos concorrentes: utilizado quando uma aplicação pode ser muito longa, e ela é interrompida para permitir que outras tarefas sejam executadas. Isso pode ser feito usando um RTOS para lidar com as tarefas ou dividindo o processo longo em uma sequência de estados, permitindo a execução de outras aplicações entre os estados (YIU, 2020).

2.2.3 Atributos de qualidade

Para mensurar a qualidade de um software são considerados os atributos de qualidade. Um atributo de qualidade é uma propriedade de um sistema que pode ser testada ou medida, para verificar como o sistema atende as necessidades das partes envolvidas através de suas funções básicas, ou simplesmente como a utilidade do sistema em algum aspecto específico para um usuário (BASS; CLEMENTS; KAZMAN, 2021).

Os principais atributos de qualidade de um software são listados na sequência, juntamente com as estratégias que podem ser utilizadas para alcançá-los (BASS; CLEMENTS; KAZMAN, 2021):

- Disponibilidade: capacidade do software executar sua tarefa quando for requisitado. Algumas estratégias para obter disponibilidade são: redundância, interrupção de funcionamento por watchdog ou reset, próximo estado válido;
- Implantação: viabilidade de instalação, atualização e desinstalação do software. Pode ser obtida com arquitetura de microsserviço, substituição gradual, teste com

pequenos grupos, testes de preferência de usuários;

- Eficiência energética: relação entre consumo e eficiência e disponibilidade. Pode ser obtida com uso de sensores de baixo consumo, monitorar e desativar serviços desnecessários, desligar aplicativos que estejam consumindo muito;
- Integração: facilidade para integrar o software com outros módulos ou sistemas. Se vale de encapsulamento, pontes, componentes distribuídos, descoberta dinâmica de provedores;
- Modificabilidade: considera os custos e riscos de realizar uma alteração no software. Estratégia de servidor provendo serviços para múltiplos clientes, núcleo central com plugins, organização em camadas com relação unidirecional, e comunicação por mensagens;
- Desempenho: refere-se a qual velocidade o software executa suas tarefas. Pode ser obtida com gestão e controle da comunicação dos microsserviços, fazendo distribuição balanceada das demandas, limitar acesso aos recursos para evitar sobrecargas e com técnicas de ordenação e análise para abundância de dados;
- Segurança: capacidade de um sistema de evitar entrar em estados que causem danos, lesões ou morte para os atores em seu ambiente, ou então detectar e se recuperar destes estados o mais rapidamente possível para minimizar os efeitos. Os estados inseguros podem ser causados por omissão, ocorrência indesejável, tempo errado, problema com valores do sistema, omissão e ocorrência indesejável em sequência, eventos fora de ordem. Atua-se com sensores redundantes, monitor para conferir cálculos do atuador, separar parte crítica do sistema, categorizar o design em níveis de criticidade;
- Proteção: medida da capacidade do sistema de proteger dados e informações de acessos não autorizados e permitir o acesso a quem é devido. Costuma validar mensagens antes de entregá-las ao destinatário e verificar atividades suspeitas;
- Testabilidade: é a facilidade com que cada sistema pode demonstrar suas falhas mediante testes. As técnicas são injeção de dependências, padrão de estratégia e filtro interceptador;
- Usabilidade: é a facilidade com que o usuário consegue utilizar o sistema e o suporte provido. Utilizada separar o modelo da realização, e verificar a capacidade do sistema retomar uma atividade.

Pode não ser possível ou viável conciliar todos os atributos, sendo necessário optar pelos mais importantes para o projeto em questão.

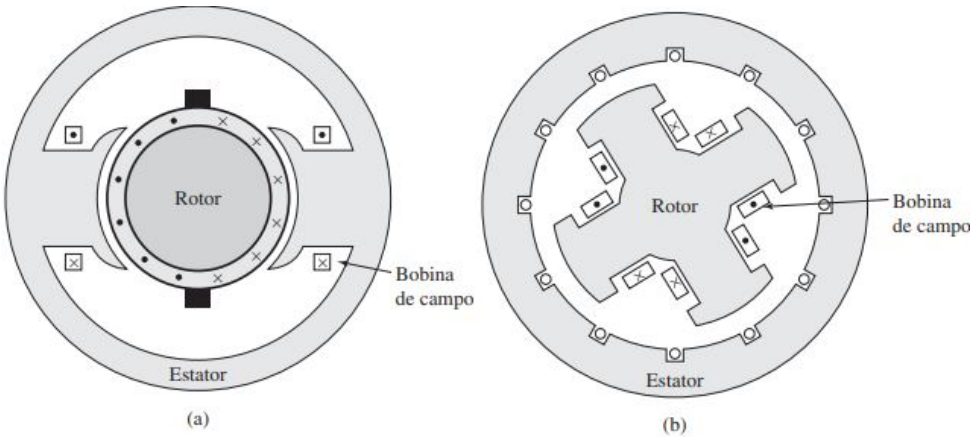
2.3 MOTORES ELÉTRICOS

Um motor elétrico é um dispositivo capaz de converter energia elétrica em energia mecânica, geralmente através da ação de campos magnéticos. São

extremamente populares porque a eletricidade é uma fonte de energia limpa e eficiente, e fácil de controlar e transmitir a grandes distâncias. Além disso, os motores elétricos não requerem ventilação nem combustíveis constantes (CHAPMAN, 2013).

Um dos princípios do funcionamento do motor elétrico é a Lei de Ampere, que afirma que um campo magnético é produzido pela passagem da corrente elétrica por um fio, e a corrente em uma bobina de fio enrolado em um núcleo produz um fluxo magnético nesse núcleo. As polaridades do fluxo magnético dependem da direção da corrente, conforme a Lei de Lenz (CHAPMAN, 2013). Geralmente, as máquinas rotativas possuem dois grupos de enrolamentos: enrolamentos de campo e enrolamentos de armadura, e a rotação do motor é causada pela interação entre os campos magnéticos das duas partes, sendo o enrolamento fixo no estator e no rotor a parte girante (UMANS, 2014), conforme os exemplos na Figura 4.

Figura 4 – Exemplos de máquinas rotativas: a) máquina CC b) máquina síncrona.



Fonte: (UMANS, 2014)

Os motores elétricos podem ser classificados segundo a sua alimentação, dividindo-se em máquinas de corrente alternada (CA) e máquinas de corrente contínua (CC) (UMANS, 2014). Também podem ser classificados conforme o número de fases da alimentação, sendo normalmente monofásicos ou trifásicos.

Neste trabalho serão considerados os motores BLDC trifásicos que, apesar da alimentação CC, são máquinas CA.

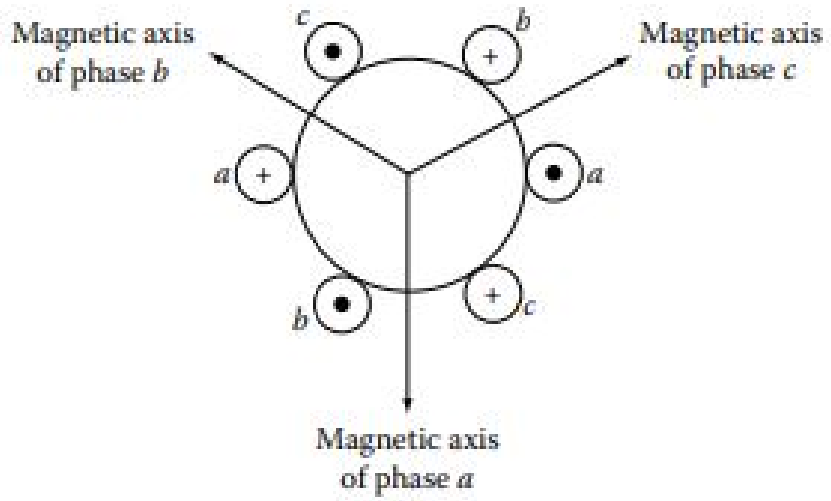
2.3.1 Motores BLDC

Atualmente o motor BLDC se tornou o mais utilizado em veículos elétricos, em função de alta eficiência, alta densidade de potência e taxa torque/peso e flexibilidade operacional (KARNAVAS; TOPALIDIS; DRAKAKI, 2019). Conforme Chapman (2013), o princípio de funcionamento de um motor CA é criar dois campos magnéticos, um no estator e um no rotor, o que induzirá um torque no rotor que fará com que ele gire e se alinhe com o campo magnético do rotor. Para fazer com que o rotor se mantenha em

rotação, é necessário fazer o campo magnético do estator girar, de forma que o campo do rotor persiga o campo do estator. Isso é possível quando correntes trifásicas com a mesma intensidade e defasagem de $120\circ$ entre si fluírem por um enrolamento trifásico. Com isso, um campo magnético girante de intensidade constante será produzido.

Como pode ser visto na Figura 5, o enrolamento trifásico é formado por três enrolamentos espaçados entre si em $120\circ$ elétricos em volta da superfície da máquina. Na figura, um enrolamento com dois polos, pois produz apenas um polo norte e um polo sul.

Figura 5 – Enrolamento trifásico máquina com dois polos.



Fonte: (KRISHNAN, 2015).

A especificidade do motor BDLC é que possui ímãs permanentes no lugar dos enrolamentos de campo (no rotor) (CHAPMAN, 2013). Umans (2014) explica que as máquinas CA com ímãs permanentes são conhecidas com motores CC sem escovas pela semelhança das características de velocidade $\times$ conjugado da máquina CC com o BLDC, quando acionado por um sistema de frequência e tensão variáveis, com um sistema eletrônico que faz o papel das escovas no chaveamento das fases.

Usualmente um motor BLDC consiste em três partes: a estrutura do motor, o circuito de acionamento de potência e o sensor de posição. Para controlar a velocidade e a direção de rotação do motor, um sensor de posição do rotor e um circuito de controle juntamente com um inversor de potência precisam ser incluídos no sistema do motor BLDC (XIA, 2012).

Os sensores de posição instalados no motor detectam a posição do rotor e a informam ao circuito de chaveamento para garantir a comutação correta das chaves. A modalidade mais utilizada de sensor é o sensor *Hall*, pelo seu baixo custo, volume reduzido e facilidade de operação (XIA, 2012).

As bobinas do estator podem ser conectadas em estrela ou triângulo, porém

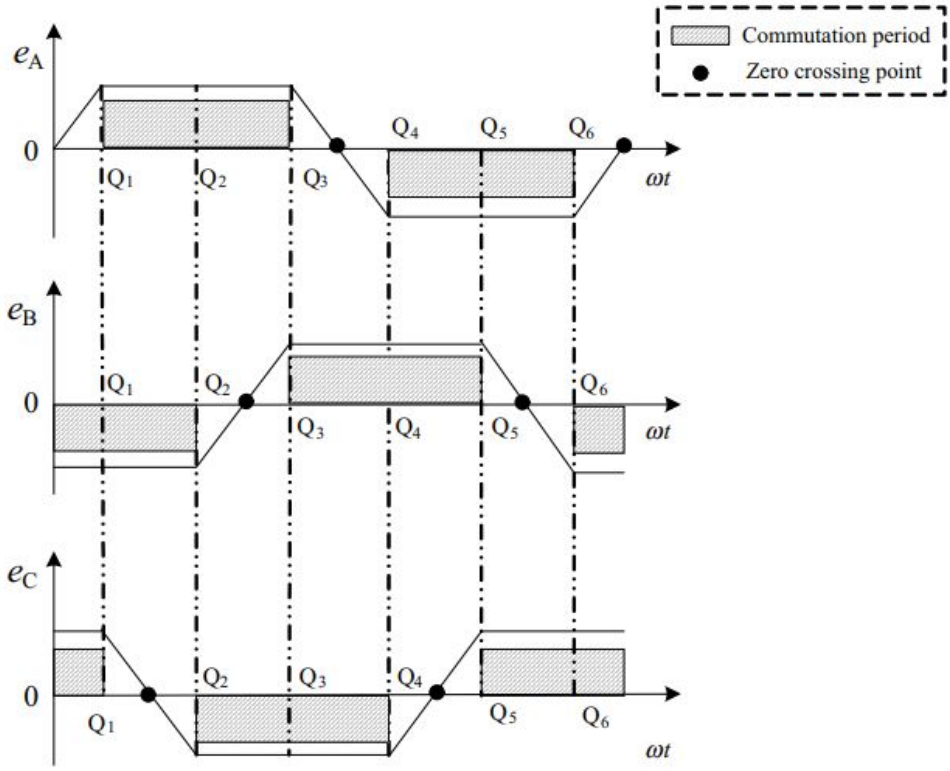
considerando custo do sistema e performance, costuma-se fazer as conexões em estrela. O acionamento pode ser feito por circuitos em meia ponte ou ponte completa com condução bifásica, ou trifásica (XIA, 2012).

Para fazer a rotação suave, o motor BLDC trifásico precisaria de alimentação com corrente senoidal, porém isso é possível apenas de forma aproximada, pois a tensão tem um formato trapezoidal, como pode ser visto na Figura 6. Apesar disso, a alimentação trapezoidal funciona bem, em função da indutância das bobinas que suaviza os cantos na curva da corrente em relação à tensão, dando-lhe um formato aproximadamente senoidal (XIA, 2012).

A frequência elétrica de um motor BLDC, em RPM, é dada pela equação 3 na qual f_{rot} é a frequência de rotação do motor em RPM na tensão máxima e polos é o número de polos magnéticos do rotor (BUCHI, 2012).

$$f_{el} = \frac{f_{rot} \times polos}{2}. \tag{3}$$

Figura 6 – Forma de onda tensão alimentação BLDC.



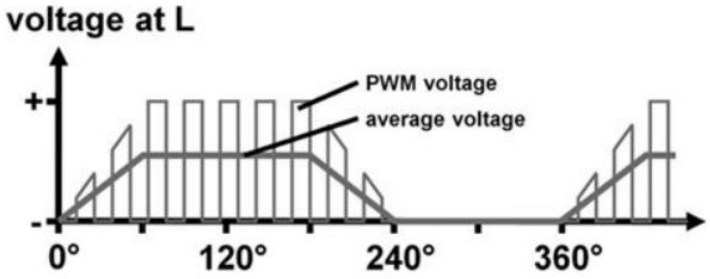
Fonte: (XIA, 2012).

Buchi (2012) explica que uma das características mais importantes a se considerar em um BLDC é o valor de RPM/V, que significa rotações por minuto por volt, ou kV (uma constante k que multiplica a tensão em Volts, com a mesma unidade - RPM/V). Com essa informação e a tensão pode se obter a velocidade de rotação do motor desconsiderando as perdas, porém essa característica atrela a

velocidade do motor à tensão da bateria, seja negativa ou positiva, mas não valores intermediários.

Para permitir um controle de velocidade, é necessário variar a tensão, por PWM. Buchi (2012) escreve que fazendo a oscilação entre valor positivo e negativo da bateria suficientemente rápido, a corrente não reage na mesma velocidade em função da indutância, mas tem-se um valor intermediário de tensão, conforme demonstrado na Figura 7.

Figura 7 – Comportamento da tensão trapezoidal com PWM.



Fonte: (BUCHI, 2012).

Quanto maior a frequência de comutação do controlador para gerar o PWM, menor a variação na corrente, porém cada comutação gera perdas de potência e variação no campo magnético. A indutância das bobinas, a velocidade de rotação e o número de polos são fatores que devem ser considerados para definir frequência do PWM (BUCHI, 2012).

2.3.2 Geradores elétricos e frenagem regenerativa

Os motores elétricos convertem energia mecânica em energia elétrica, e os geradores fazem a transformação inversa, convertendo energia mecânica em energia elétrica, a partir da energia cinética do motor (UMANS, 2014).

A força contraeletromotriz (FCEM), conhecida como a tensão da velocidade, é dada por 4 na qual K_v é a constante de tensão em $V/(A \times rad/s)$, ω é a velocidade angular do motor em rad/s e i é a corrente (RASHID, 2014; SANTOS JR.; SILVA, 2010).

$$e_g = K_v \times \omega \times i_f \tag{4}$$

Dreher e Rosa (2013) explica que a energia cinética do veículo, no processo de frenagem, aplica energia mecânica no eixo do rotor, que passa a funcionar como gerador, sendo possível, com uma estratégia adequada de controle do BLDC, aproveitar essa energia para recarregar as baterias. Esse processo é chamado de frenagem regenerativa.

Normalmente a tensão induzida é menor que a tensão da bateria, sendo necessário usar conversor para aumentar a tensão ao nível necessário para carregar a

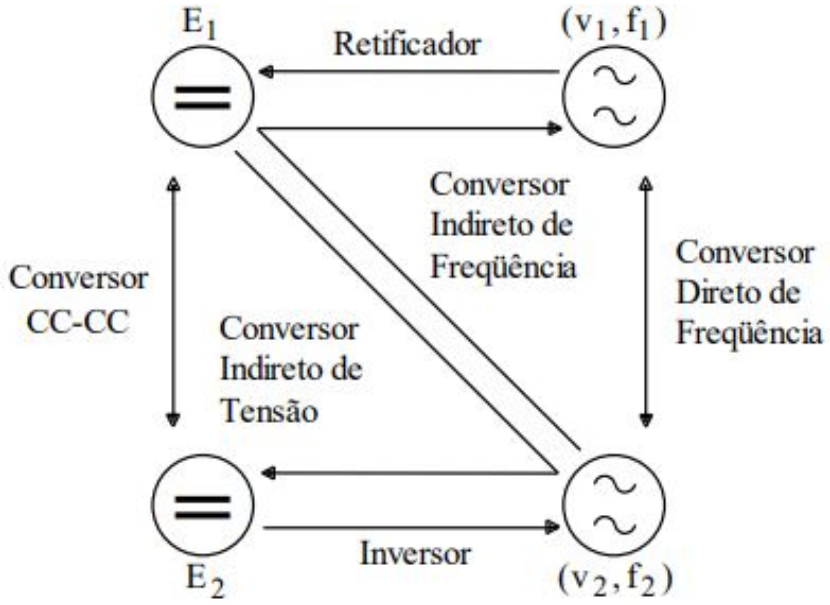
bateria (CHEN; C.; CHENG, 2011). A implementação da frenagem regenerativa pode ser feita de diversas formas, sendo a recuperação da energia acionada pelo pedal do acelerador, do freio ou ambos (COCRON et al., 2013).

De Santos Jr. e Silva (2010) tem-se que quando a FCEM é alta, a frequência de chaveamento do conversor deve ser baixa e vice-versa, em função da corrente.

2.4 CONVERSORES E INVERSORES

As fontes de energia elétrica podem gerar tensões alternadas (geralmente produzidas por geradores síncronos) ou contínuas (geradores fotovoltaicos, baterias, etc). Para adaptar essas tensões às necessidades do equipamento utilizado, são encontrados conversores estáticos de energia elétrica, que consistem em sistemas constituídos por semicondutores que atuam como chaves e dispositivos magnéticos, como capacitores e indutores. A Figura 8 representa os tipos de conversão que podem ser realizadas: Conversores CC-CC diretos ou com passagem por tensão alternada, CC-CA e CA-CC diretos e CA-CA com passagem por estágio de corrente contínua. Os conversores CC-CA são chamados de inversores e os conversores CA-CC são chamados de retificadores (BARBI, 2022).

Figura 8 – Possibilidades dos conversores estáticos de energia.



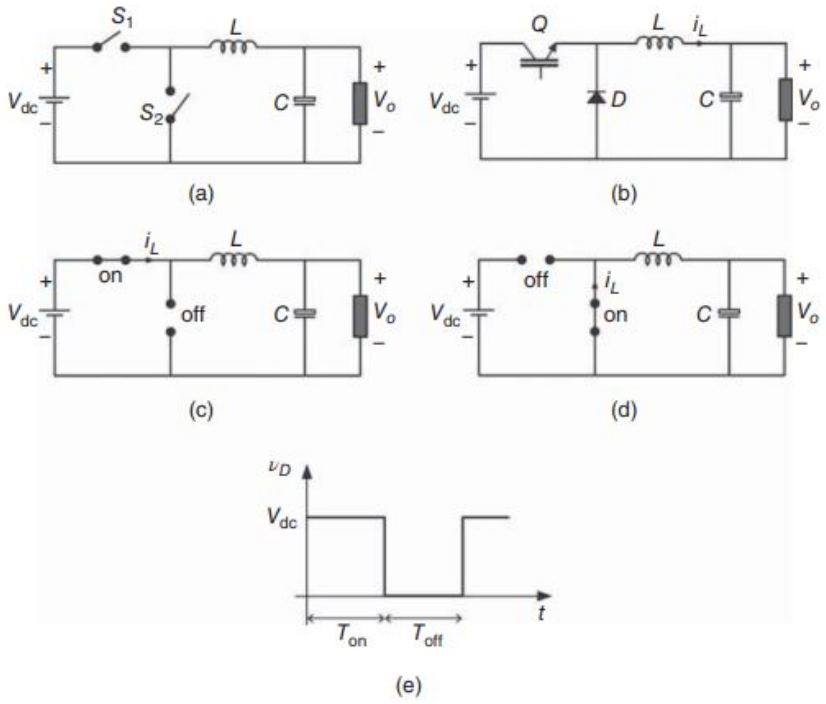
Fonte: (BARBI, 2022).

Este trabalho, como a alimentação é a bateria e o BLDC é uma máquina CA, irá considerar o acionamento de um inversor.

2.4.1 Conversores CC-CC

Os conversores CC diretos são também conhecidos como chopper, e podem reduzir a tensão (conversores buck) e elevar a tensão (conversores boost) (SANTOS JR.; SILVA, 2010). A Figura 9 apresenta o esquema de funcionamento das chaves de um conversor buck monofásico para gerar o PWM.

Figura 9 – Funcionamento conversor buck.



Fonte: (SANTOS JR.; SILVA, 2010).

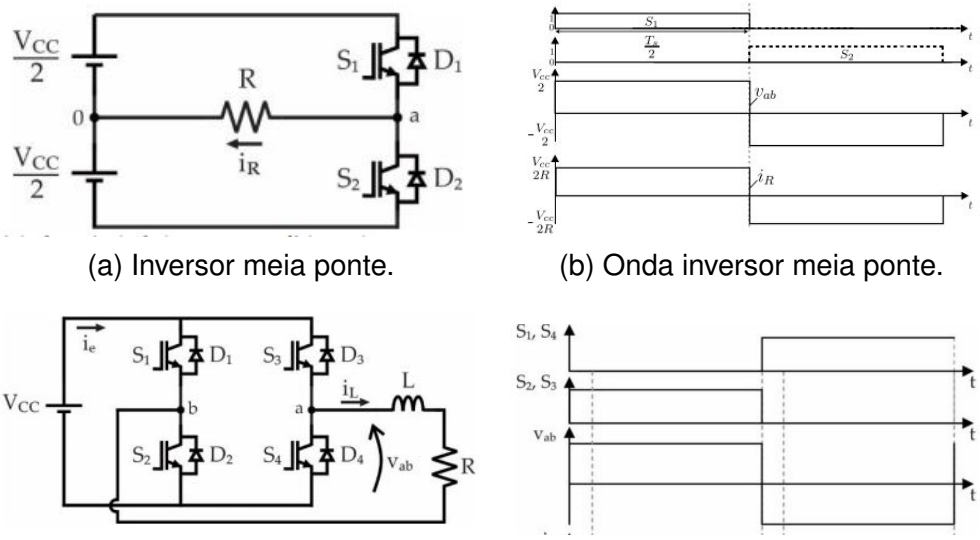
Conforme Santos Jr. e Silva (2010), o funcionamento do conversor *buck* se dá pelo funcionamento alternado de duas chaves (S_1 e S_2) representadas na Figura 9a, que consistem em um transistor e um diodo (Figura 9b) que operam de forma complementar (Figuras 9c e 9d), uma vez que são dispostos para apenas um elemento permitir passagem de corrente. Quando o transistor está conduzindo a carga está ligada à fonte (T_{on}) e quando o diodo conduz a tensão é nula (T_{off}), gerando o pulso apresentado na Figura 9e. Dessa forma é possível controlar a tensão média fornecida ao motor controlando o ciclo de trabalho, conforme explicado em 2.1.2.

2.4.2 Inversores CC-CA

A função dos inversores é transformar uma tensão de entrada CC em uma saída CA simétrica, com a amplitude e frequência necessárias. Quando a tensão de alimentação é fixa (inversor alimentado por tensão), a saída pode ser variada com o controle da modulação por PWM (RASHID, 2014).

Os principais tipos de inversores são monofásicos, trifásicos, meia ponte e ponte completa. Na Figura 10 são apresentados os conversores monofásicos meia ponte e ponte completa, que são as topologias mais simples.

Figura 10 – Inversores monofásicos.



(a) Inversor meia ponte. (b) Onda inversor meia ponte.

(c) Inversor ponte completa. (d) Onda inversor ponte completa.

Fonte: (BARBI, 2022).

No inversor de meia ponte, as chaves funcionam de modo complementar, assim como os diodos, e apenas uma malha conduz por vez, tendo como resultado uma tensão limitada à metade da tensão da fonte. Nos inversores de ponte completa a tensão pode chegar ao valor da fonte (ciclo de trabalho de 100%), sendo que as chaves do mesmo braço funcionam de forma complementar e as chaves paralelas funcionam alternadamente, superior de um braço com inferior do outro braço (BARBI, 2022).

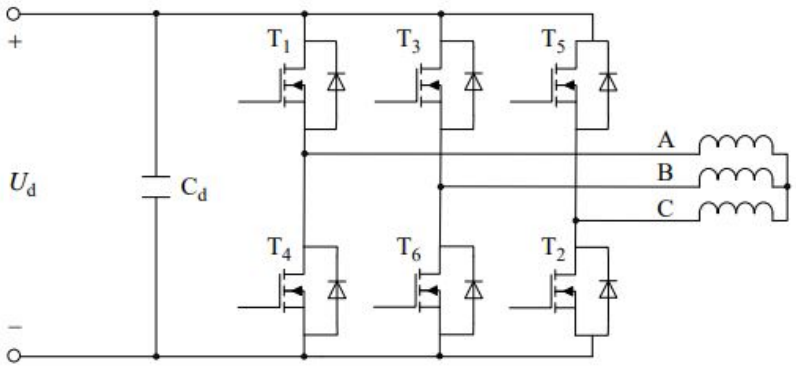
Um inversor trifásico, três inversores meia ponte ou ponte completa podem ser conectados em paralelo, como na Figura 11, na qual consta o inversor ponte completa. Os sinais dos inversores devem ser atrasados ou adiantados em 120^o em relação aos outros para gerar tensão trifásica equilibrada.

O motor BLDC necessita de acionamento trifásico. Um inversor em meia ponte tem a vantagem de ter menos componentes, ser mais barato e fácil de controlar, porém, segundo Xia (2012), tem as desvantagens de maior oscilação de torque e baixa utilização das bobinas.

Neste trabalho, considerando os requisitos de projeto será utilizado um circuito ponte completa trifásico, com condução em duas fases. A ordem de condução e o momento de ativação é determinada pela informação de posição gerada pelos sensores do rotor, que funcionam conforme explicado na Seção 2.1.3.

A condução em duas fases significa que sempre há duas bobinas conduzindo

Figura 11 – Inversor trifásico ponte completa.



Fonte: (XIA, 2012).

simultaneamente e a terceira bobina está suspensa. A cada 60^o elétricos, uma chave comuta, mudando uma das bobinas. Dessa forma, cada bobina permanece conduzindo continuamente por 120^o elétricos. Sempre há uma chave no nível superior e uma no nível inferior, o que faz com que haja uma bobina com tensão positiva e uma com tensão negativa acionadas (XIA, 2012).

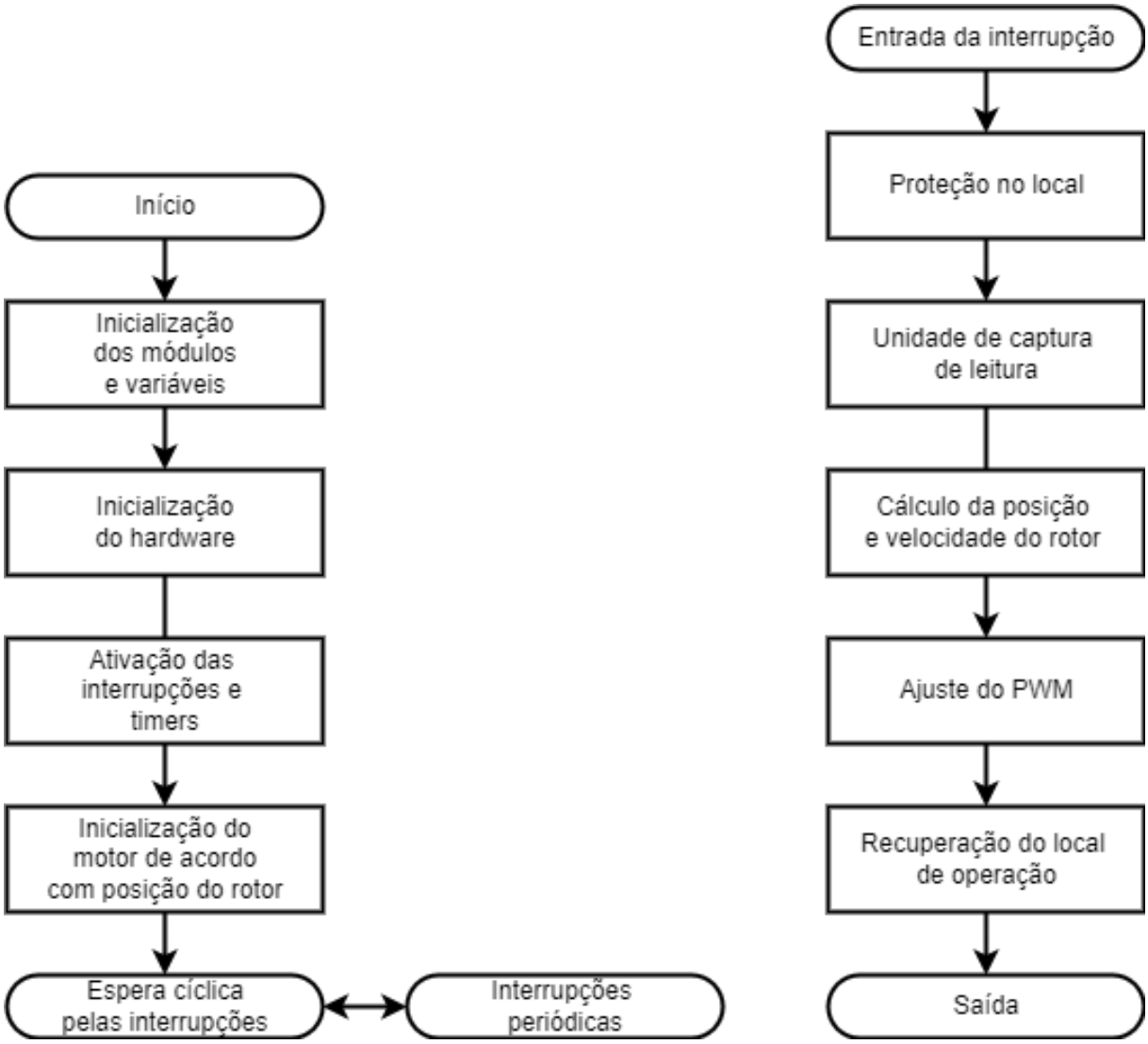
Xia (2012) explica que na condução em três fases, sempre há três bobinas conduzindo, pois o intervalo de condução é 180^o. Isso reduz a oscilação de torque e aumenta uso das bobinas, porém pode permitir que as chaves superior e inferior, do mesmo braço, entrem em condução simultaneamente, o que causaria curto-circuito.

2.5 DESIGN FIRMWARE CONTROLE BLDC

Unindo o conhecimento em sistemas embarcados, motores e conversores de frequência, Xia (2012) propõe um fluxograma básico para implementação de um controle para BLDC com sensores de posição, sendo o caso em estudo, apresentado na Figura 12.

Destacando alguns aspectos do fluxograma da Figura 12, no módulo de inicialização é feita a inicialização do relógio do sistema (*clock*), o *watchdog*, as portas I/O, interrupção e outros sistemas de hardware, bem como inicializadas as variáveis. Fora da interrupção, são feitas apenas as configurações, e o funcionamento do motor ocorre dentro da interrupção. O autor destaca que o timer geral deve ser configurado com a frequência de amostragem após a inicialização que a subrotina de interrupção seja desativada durante a inicialização, para prevenir interrupções indevidas (XIA, 2012).

Figura 12 – Fluxograma para controle de motor BLDC com sensores.



Fonte: Adaptado de (XIA, 2012)

2.6 TRABALHOS RELACIONADOS

Foram pesquisados trabalhos que descrevem o desenvolvimento de firmware para acionamento de conversores baseados em microcontroladores para motores BLDC, bem como trabalhos que estudaram as características necessárias para um bom controle destes motores.

O trabalho de conclusão de curso de CHRIST (2023) fez o levantamento das características requeridas para um inversor para o veículo de competição da equipe EFICEM da UFSC Campus Joinville e desenvolveu um inversor ponte completa trifásico baseado no microcontrolador ATMega2560, para acionar um BLDC com controle de posição por sensores *Hall*. O trabalho apresenta todo o circuito elétrico do

inversor e pinagem. Na configuração escolhida para chaveamento, foi utilizado PWM complementar na fase positiva e a fase negativa foi acionada sem a modulação. Foi utilizado tempo morto (*dead time*) entre as fases e não foi possível fazer a comutação das chaves conforme a leitura dos sensores de posição por defeitos construtivos no motor utilizado.

O trabalho de Meireles (2023) apresenta o desenvolvimento de conjunto de circuitos para acionamento de um motor BLDC, com tratamento dos sinais dos sensores *Hall*, circuito de acionamento e inversor trifásico de ponte completa. Foi utilizado um microcontrolador DSP TMS320F28027 para fazer o acionamento. O estudo comparou o método de acionamento convencional, no qual duas chaves são mantidas em estado alto por 120^o conforme o setor correspondente, e acionamento por PWM unipolar, no qual uma chave recebe o sinal pulsado por 60^o e se mantém estável no nível alto por mais 60^o, mostrando a coerência dos dois métodos.

O artigo de Karnavas, Topalidis e Drakaki (2019) faz análise de aspectos de hardware e software para implementação de controle de motor BLDC baseado em microcontrolador, fazendo a implementação prática, com ênfase em baixo custo e simplicidade. O Hardware desenvolvido faz o controle da posição do motor por força contraeletromotriz (sem sensores *Hall*, detecta o momento em que a tensão na bobina desativada é exatamente zero).

O hardware desenvolvido por Karnavas, Topalidis e Drakaki (2019) consiste em um inversor trifásico de ponte completa com comutação de seis etapas e condução de 120^o elétricos. Destaca que se as chaves não comutam perfeitamente entre *on* e *off* pode ser necessário um tempo de transição (tempo morto) para evitar curto-circuito nos braços, porém os autores utilizam acionamento de chaves do mesmo braço com 60^o elétricos de separação. Os autores justificam a escolha do microcontrolador com uma série de critérios e, a partir do atendimento dos critérios, qual dos modelos utilizar é uma "decisão pessoal", e optam pelo Microchip dsPIC30F4011. O software de controle é implementado em linguagem *C*, constituído basicamente por seis rotinas, sendo uma para acionamento, duas para funcionamento do motor e três interrupções para conversões de dados.

Hui, Basu e Subbiah (2003) dividem as técnicas de controle do motor BLDC em cinco categorias principais: Método de controle de comutação, de velocidade, de partida, detecção de posição e controle de desempenho, levantando aspectos que devem ser levados em consideração na implementação do controle. Os autores explicam que o controle de comutação consiste em manipular a corrente pelas bobinas na direção e tempo corretos, podendo ser utilizadas as técnicas de controle de meia ponte, 120^o (também conhecido como duas fases), $180^o graus$ (três fases). O controle de velocidade é feito pela magnitude da tensão aplicada, aumentando a tensão aumenta a corrente nas bobinas, fortalecendo o campo magnético e forçando o alinhamento dos campos

de rotor e estator mais rapidamente, através do aumento da velocidade de rotação, e a pulsação do torque aumenta muito com a redução do tempo de fase. A tensão pode ser controlada pela variação da tensão de alimentação ou por PWM. O controle de acionamento consiste em identificar a posição do rotor e tomar precauções para sobrecorrente, que na partida é dez vezes maior, e caso o rotor não esteja posicionado entre duas bobinas, a sobrecarga será maior. O texto descreve alguns métodos para reposicionar o rotor na partida. O controle de partida pode ser feito principalmente por dois métodos, com sensor (*Hall* ou encoder ótico) e sem sensor, no qual o controle é feito detectando a força contraeletromotriz do motor.

O artigo de Mukherjee, Ray e Das (2018) descreve a implementação de um software para drive baseado em microcontrolador de baixo custo para um BLDC trifásico com tensão trapezoidal, e algoritmo de controle baseado em PWM adaptável para meia ponte e duas fases, para permitir velocidade de operação considerando os efeitos de carga constante. O microcontrolador utilizado é um PIC18F4331, e foi utilizado Proteus VSM para simular o programa antes da implementação em hardware. As funções básicas do microcontrolador consistem em manter a sequência dos seis canais de PWM em sincronismo com o posicionamento dados pelos sensores *Hall* para fazer a comutação das fases e ajustar o ciclo de trabalho do PWM para controlar a tensão fornecida ao motor. O texto explica que o BLDC é fácil de controlar em função da característica quase linear da relação tensão $\times$ velocidade, apenas ajustando o ciclo de trabalho do PWM, e apresenta o fluxograma de um algoritmo de controle, mostrando os cálculos necessários para estabelecer/manter os parâmetros. Finaliza demonstrando por simulação o funcionamento do software desenvolvido.

Chen, C. e Cheng (2011) faz um comparativo de diferentes formas de chaveamento no inversor para implementação de freio regenerativo em um motor BLDC. Inicialmente coloca que a implementação pode ser feita acoplando um conversor book ou boost ao conversor, porém, considerando custo de componentes, o experimento enfatiza o uso de um inversor ponte completa, afirmando que com a lógica correta de chaveamento é possível fazer a carga da bateria. O experimento consiste em utilizar o mesmo inversor que aciona o BLDC apenas mudando o chaveamento, com acionamento de apenas uma chave, duas chaves e três chaves. No estudo o PWM foi regulado pela corrente, quanto menor a velocidade de rotação do motor, maior o ciclo de trabalho. Os resultados ficaram todos ficarem próximos de 50% de recuperação na frenagem, sendo que para velocidades constantes, o conversor fazendo o ciclo por três chaves teve melhor desempenho, e para velocidades variando, o acionamento da reversão por uma chave apenas apresentou o melhor resultado.

3 MATERIAIS E MÉTODOS

Nesta seção são descritos os requisitos do projeto, os recursos utilizados e a modelagem feita. Também são apresentados os procedimentos definidos para testes.

3.1 REQUISITOS DE PROJETO

Com base nos requisitos da competição Shell Eco Marathon, combinados com a teoria estudada para o uso do hardware e os princípios de qualidade de software, foram definidos os requisitos de projeto, listados a seguir.

Requisitos funcionais:

RF1: Alimentação por bateria;
RF2: Acionamento de motor trifásico;
RF3: Reconhecimento de sensor *Hall*;
RF4: Geração de PWM com ciclo de trabalho variável;
RF5: Reconhecimento de sinal de acelerador;
RF6: Frenagem regenerativa;
RF7: Limitar corrente no modo motor;
RF8: Limitar corrente no modo gerador;
RF9: Monitoramento da corrente;
RF10: Monitoramento do nível da bateria.

Requisitos não funcionais:

RNF1: Programado em linguagem C;
RNF2: Ser compatível com a placa inversora da equipe EFICEM;
RNF3: Características do motor configuráveis;
RNF4: Tensão de alimentação configurável;
RNF5: Estratégia de controle Six-steps;
RNF6: Topologia de operação do inversor em 120º;
RNF7: Correntes máximas configuráveis;
RNF8: PWM complementar na operação em modo motor;
RNF9: Acionamento de uma chave por ciclo na operação gerador.

Regras de negócio:

RN1: Freio regenerativo acionado por ausência de aceleração;
RN2: Limite mínimo de carga na bateria.

3.2 MATERIAIS

O firmware foi desenvolvido em linguagem C, fazendo uso do ambiente de desenvolvimento Cube IDE [1], com uso da biblioteca HAL (hardware abstraction layer) CMSIS. Para o teste foi utilizado o software Logic [2].

Os recursos de hardware utilizados são placa de desenvolvimento com microcontrolador STM32F407G com processador ARM, potenciômetros para simular acelerador, sensor de tensão e sensor de corrente, circuito NAND com CIs HEF4011B, analisador lógico digital e microcontrolador Arduino Uno para simular as informações dos sensores *Hall*.

3.2.1 Motor

O motor considerado como referência para os testes é um Nema17 JK42BLS01, da fabricante Jkong Motor, com sensores *Hall* acoplados. As principais características do motor são apresentados no quadro 1.

Quadro 1 – Especificações elétricas do motor JK42BLS01

	Especificações elétricas	
Especificação	Unidade	JK42BLS01
Número de fases	fases	3
Número de polos	polos	8
Tensão nominal	V	24
Velocidade nominal	RPM	4.000
Corrente nominal	A	1,8
Corrente de pico	A	5,4
FCEM	V/RPM	4,1

Fonte: Adaptado de (JKONG MOTOR CO., LTDA, 2020).

O motor foi apenas tomado como referência de parâmetros, uma vez que os testes foram simulados.

3.3 PROJETO

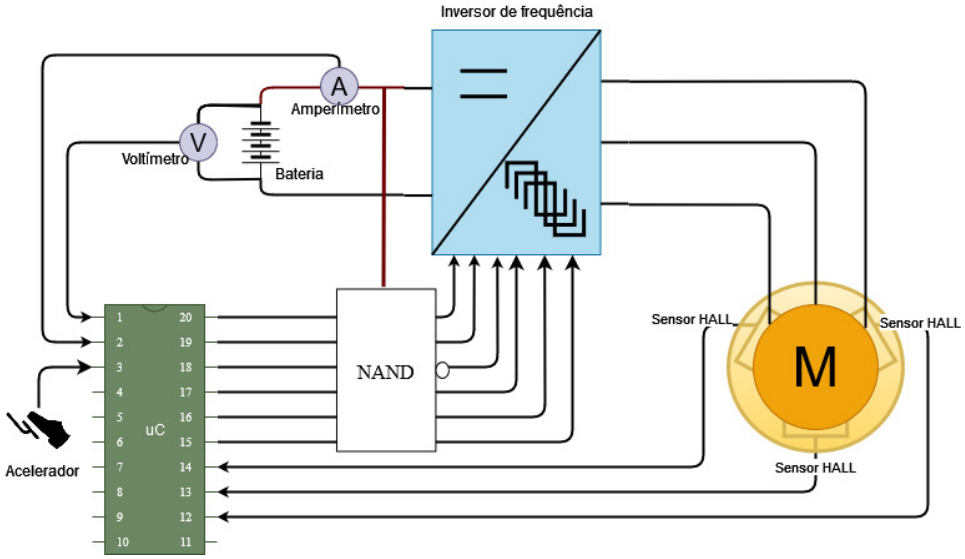
O projeto consiste na modelagem do sistema, compreendendo a relação entre os componentes e definindo critérios para as funções principais: a sequência de chaveamento e geração dos pulsos para acionar o motor. O sistema foi modelado para compreender as relações entre os componentes e seus comportamentos.

[1] https://www.st.com/en/development-tools/stm32cubeide.html
[2] https://www.saleae.com/pages/downloads

3.3.1 Sistema físico

O sistema físico considerado compreende motor (M), placa inversora de frequência, bateria, microcontrolador, circuito NAND, três sensores *Hall* embutidos no motor, acelerador, sensor de corrente e sensor de tensão. A visão geral do sistema está disponível na Figura 13.

Figura 13 – Representação dos componentes do sistema físico.



Fonte: A autora.

A bateria alimenta todo o sistema, a placa que contém o inversor faz a redução de tensão para alimentar o microcontrolador e os sensores. O microcontrolador controla o modo de operação - se motor ou gerador, e gera o PWM, de acordo com as informações do acelerador (modo motor) e sensores de corrente e tensão (modo gerador) e coordena a sequência de chaveamento dos transistores a partir das informações dos sensores *Hall*. O sensor de tensão monitora o nível da bateria e o sensor de corrente, além da informação para o PWM no modo gerador, serve para alertar o microcontrolador em caso de sobrecarga no circuito.

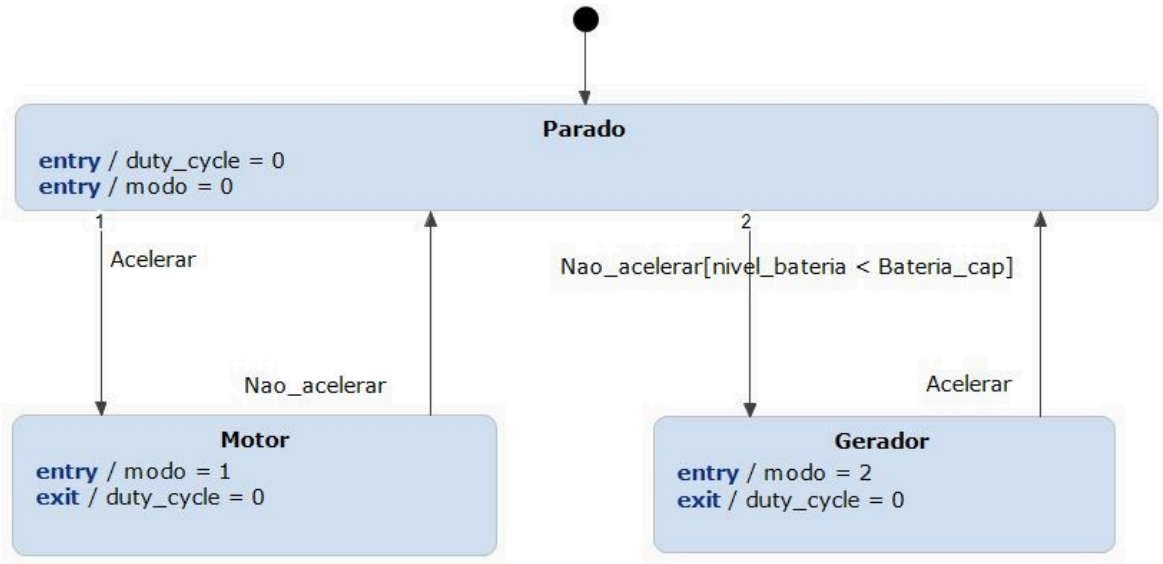
Não existe malha de controle fechada, pois o controle é feito diretamente pelo condutor do veículo, através do acelerador, sem automação alguma.

3.3.2 Modos de operação

O sistema, quando em funcionamento, pode estar em modo motor ou em modo gerador, conforme representação na Figura 14, que apresenta apenas os estados gerais do sistema e os eventos que provocam mudança de estado e são dependentes do usuário.

O estado *modo motor* é o funcionamento do sistema quando o veículo está andando acelerado. O estado *modo gerador* significa que não está sendo aplicada

Figura 14 – Modos de operação do sistema.



Fonte: A autora.

aceleração sobre o motor, e este está funcionando como gerador, através da inversão do sentido da corrente no inversor de frequência.

Existem quatro ações possíveis sobre o sistema, das quais três são relacionadas com o modo motor - veículo acelerado, e uma com o modo gerador - veículo não acelerado. As ações que mantém o veículo acelerado, e o motor em modo motor são: acelerar, manter aceleração e reduzir aceleração (sem zerar). Todas as ações devem ser consideradas, apesar de que apenas *acelerar* provoca mudança de estado, mas as demais tem efeito sobre o ciclo de trabalho. Na prática, significam que o motorista do veículo realiza algum pressão no pedal do acelerador.

A ação de não acelerar significa que não há pressão alguma no pedal do acelerador, ou, no caso da simulação deste estudo, o potenciômetro estará com resistência zerada. O quadro 2 apresenta um resumo das ações.

Quadro 2 – Resumo das ações possíveis

Ação	Veículo acelerado	Transição de estado	Modo	Ciclo de trabalho
Acelerar	sim	sim	motor	aumenta
Manter aceleração	sim	não	motor	mantém
Reduzir aceleração	sim	não	motor	reduz
Não acelerar	não	sim	gerador	aumenta

Fonte: A autora.

No quadro 2 está explícito se a ação mantém o veículo acelerado, se pode

levar à transição de estado, a qual estado leva ou mantém o motor e seu efeito sobre o ciclo de trabalho. O ciclo de trabalho é diferente em modo motor e modo gerador, e na transição de estados ele é zerado.

O ciclo de trabalho considera ainda a definição de um intervalo para o ciclo de trabalho entre zero e 100%, mesmo havendo aceleração ou não aceleração, e independente do valor da corrente, atingido o limite o ciclo de trabalho se mantém no limite até alguma ação que inverta a situação.

Caso seja implementado modo cruzeiro, é necessário revisar a forma com que o ciclo de trabalho é definido.

3.3.3 Sequência de chaveamento BLDC

Coordenar a sequência de chaveamento dos transistores do inversor de frequência, juntamente com o controle do ciclo de trabalho são as principais funções que precisam ser desempenhadas pelo controlador do sistema.

A sequência de chaveamento determina a ordem em que as bobinas serão energizadas para permitir a rotação do motor, e depende da informação recebida dos sensores *Hall*, conforme explicado na seção 2.3.1. Apesar da rotação do motor permanecer no mesmo sentido nos modos motor e gerador, o acionamento dos transistores é diferente, pois no modo motor a corrente flui sentido bateria-inversor, e no modo gerador é necessário acionar as chaves para permitir que a corrente faça o caminho inverso.

3.3.4 Sequência modo motor

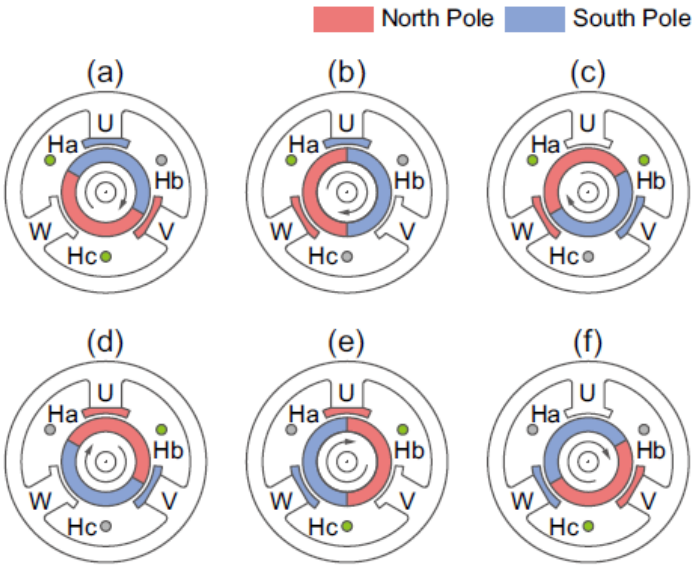
A sequência de posição do rotor informada pelos sensores *Hall* é $5-4-6-2-3-1$, considerando a rotação no sentido horário. A origem dessa sequência é apresentada na Figura 15.

Analisando a Figura 15 é possível ver que cada sensor é ativado pela proximidade de um dos polos do rotor, tendo sempre dois sensores na posição *'1'* e um na posição *'0'*, com isso gerando um valor com três bits, conforme demonstrado no quadro 3.

Os valores em binário, conforme detalhado no quadro 3, informam para o microcontrolador a posição do rotor, e a partir disto é definido quais são as próximas bobinas que precisam ser energizadas. Para energizar as bobinas, o microcontrolador ativa os transistores que irão alimentá-las, sendo sempre uma bobina do nível superior e uma bobina do nível inferior, não complementares.

Por definição de projeto, cada bobina deve permanece ativa por 120^o (topologia de operação em 120^o) e foi estabelecido que bobinas complementares não serão ativadas em sequência, para evitar curtos-circuitos. Com isso foi definida a sequência

Figura 15 – Sequência de ativação sensores *Hall*.



Fonte: (TANG, 2021).

Quadro 3 – Posicionamento sensores *Hall* com base na Figura 15

Sub-figura	*Hall*			Posição
	A	B	C	
a	1	0	1	5
b	1	0	0	4
c	1	1	0	6
d	0	1	0	2
e	0	1	1	3
f	0	0	1	1

Fonte: A autora.

apresentada no quadro da Figura 16.

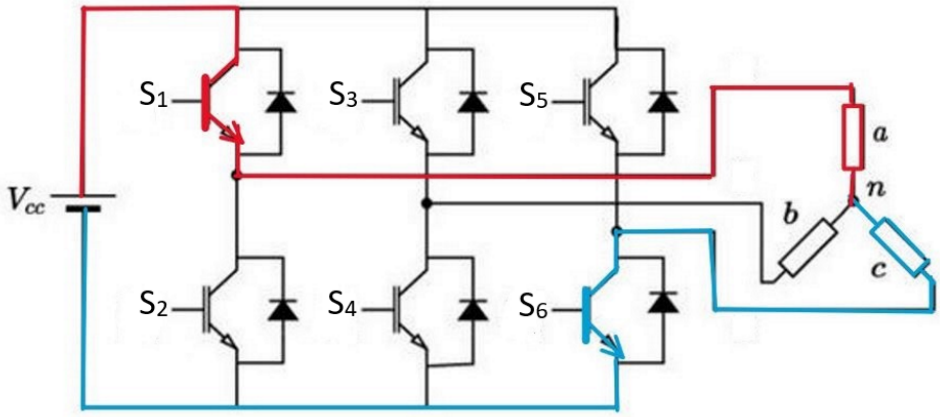
Figura 16 – Sequência de acionamento das chaves de acordo com a posição do rotor.

HIGH			LOW			HALL
A	B	C	A	B	C	
S1	S3	S5	S2	S4	S6	
1	0	0	0	0	1	5
0	1	0	0	0	1	4
0	1	0	1	0	0	6
0	0	1	1	0	0	2
0	0	1	0	1	0	3
1	0	0	0	1	0	1

Fonte: Adaptado de (FIGUEIREDO et al., 2022).

Na Figura 17 pode ser visto o exemplo de um braço do inversor acionado com o fluxo da corrente.

Figura 17 – Braco de inversor ativo - posição 5.



Fonte: A autora.

A Figura 17 apresenta o fluxo da corrente quando o rotor está na posição 5, com os transistores $S1$ do nível superior e $S6$ no nível inferior ativos, com a corrente fluindo do polo positivo para o polo negativo da bateria.

3.3.5 Sequência modo gerador

Quando o motor estiver funcionando em modo gerador, é necessário que a corrente flua do motor para o a bateria. Seguindo os resultados obtidos no estudo de Chen, C. e Cheng (2011), será utilizado o inversor existente, sem adição do conversor boost e realizando a frenagem regenerativa com acionamento de apenas um transistor, sempre no nível inferior do inversor, conforme detalhado no quadro 4.

Quadro 4 – Sequência de acionamento dos transistores em modo gerador

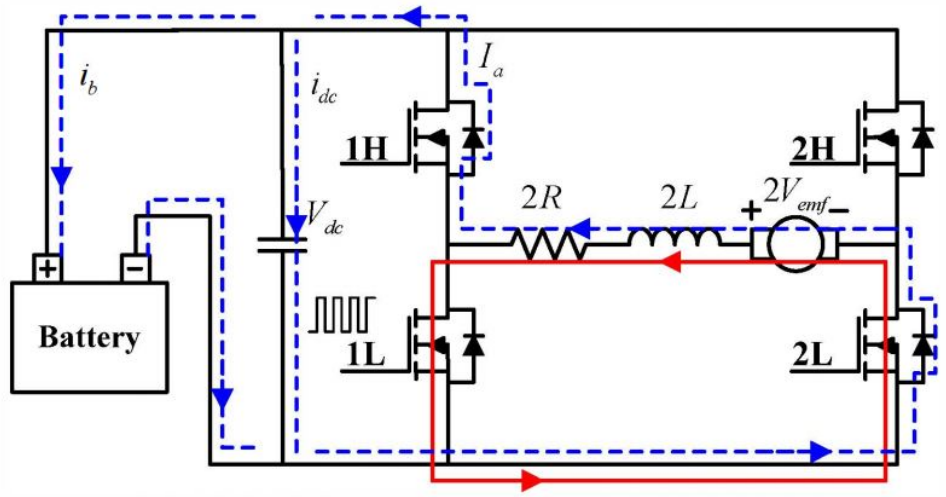
Posição rotor	Chave ativa
5	S6 (C)
4	S6 (C)
6	S4 (B)
2	S4 (B)
3	S2 (A)
1	S2 (A)

Fonte: (DREHER; ROSA, 2013).

A Figura 18 mostra o fluxo da corrente com uso de apenas uma chave para gerar o PWM em cada posição.

O braço apresentado na Figura 18 é uma simplificação com um inversor de dois quadrantes, porém a lógica é a mesma para o inversor de três quadrantes. A linha contínua, em vermelho, mostra o caminho da corrente quando a chave está ativa e a

Figura 18 – Braço de inversor em modo gerador.



Fonte:(CHEN; C.; CHENG, 2011).

linha azul, pontilhada, representa o fluxo da corrente com a chave inativa, dentro de um pulso de PWM.

3.3.6 Definição do ciclo de trabalho

A definição do ciclo de trabalho do PWM também é diferente nos modos motor e gerador.

No modo motor, a lógica é que quanto mais o motorista do veículo acelerar, mais velocidade ele precisa por isso maior deve ser a tensão entregue ao motor, o que demanda um ciclo de trabalho maior. Dessa forma, o valor do ciclo de trabalho é dado pela proporcionalidade direta da variação na aceleração. No caso do potenciômetro esse intervalo é de 0 a $5V$, se considerar ciclo de trabalho de 0 a 100%, para cada $50mV$ o ciclo de trabalho aumenta 1 ponto percentual, e reduz proporcionalmente quando é diminuída a resistência do potenciômetro (ou pressão no acelerador).

Em modo gerador, o PWM serve para controlar a corrente: em rotações altas, a tensão gerada é maior, e o ciclo de trabalho precisa ser menor para evitar sobrecarga. Quanto menor a velocidade de rotação, maior o ciclo de trabalho. A corrente pode ser calculada pela lei de Ohm, sendo necessário conhecer a diferença de potencial entre a geração e a bateria e a resistência do estator. A tensão gerada é a força contraeletromotriz, obtida em função da velocidade e geralmente pode ser obtida no *datasheet* do motor, bem como a resistência. A corrente que o inversor suporta e a capacidade de carta da bateria devem ser levadas em consideração.

No caso dos equipamentos considerados neste projeto, a força contraeletromotriz na rotação máxima é inferior à capacidade da bateria. Isso faz com que seja necessário um conversor CC-CC para ser possível carregar a

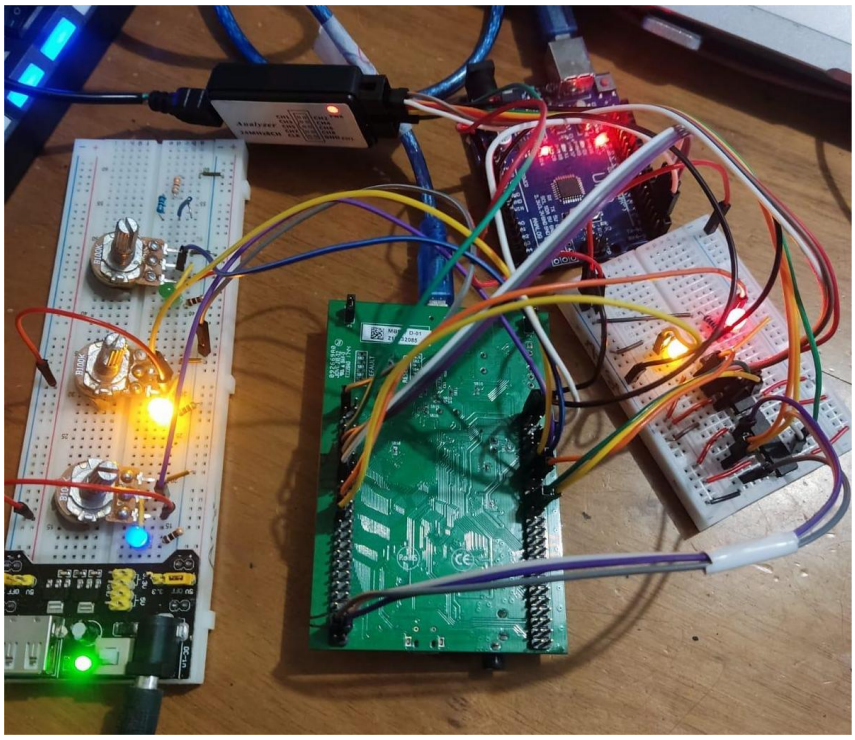
bateria, o que está além do escopo, porém a possibilidade do freio regenerativo já é considerada no firmware para caso de futura implementação.

A implementação será feita buscando o máximo aproveitamento de carga e o ciclo de trabalho será definido pela corrente, respeitando o intervalo estabelecido para o ciclo de trabalho. Enquanto a corrente estiver dentro do limite máximo, o ciclo de trabalho será calculado com base na relação entre a corrente máxima e a corrente medida.

3.4 ESTRUTURA PARA TESTES

Para verificar o funcionamento do software foi utilizado um circuito com potenciômetros para simular os sensores do veículo - acelerador, tensão da bateria e corrente, e uma placa Arduíno Uno para gerar um sinal simulando os sinais dos sensores *Hall*. O conjunto de teste utilizado é mostrado na Figura 19.

Figura 19 – Foto do conjunto de teste.

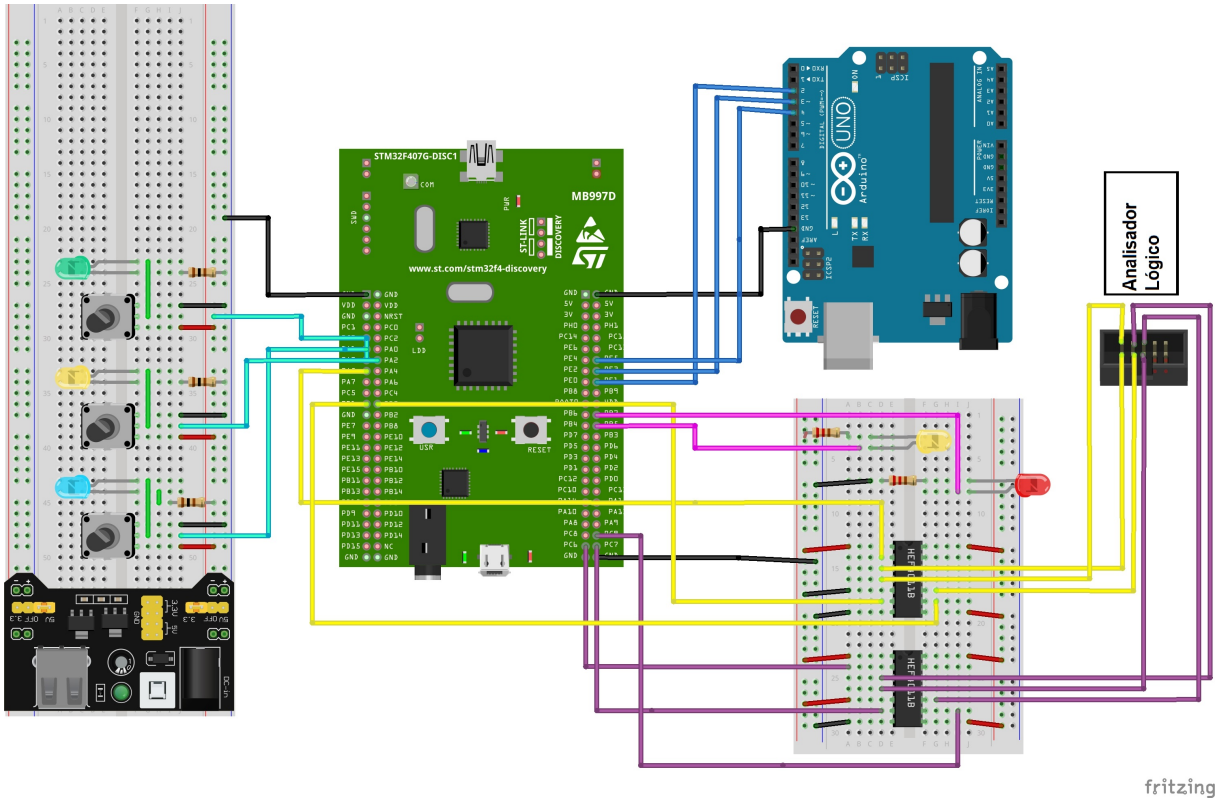


Fonte:A autora.

O resultado foi verificado através do analisador lógico digital conectado ao computador. Na Figura 20 temos uma ilustração do conjunto, com os componentes e conexões.

Na placa de prototipagem a esquerda estão os potenciômetros para gerar as entradas analógicas simulando os sensores - os leds utilizados são apenas para sinalizar a posição do potenciômetro durante o teste. A placa verde é o μC que aciona o inversor - o STM32, e o Arduino gera os sinais digitais. A placa de prototipagem

Figura 20 – Ilustração da montagem circuito de teste.



Fonte: A autora.

menor possui o circuito inversor de sinal (NAND), e as saídas ligadas ao analisador lógico são as que serão conectadas ao inversor, além das saídas nos leds amarelo e vermelho que indicam corrente alta e bateria baixa, respectivamente. A pinagem para conexão do STM32 consta no apêndice 5.

Foram simuladas diversas situações, como aceleração, ausência de aceleração, aumento e diminuição da corrente, bateria cheia e bateria no nível baixo. Os resultados são descritos no capítulo 4.

4 ANÁLISE E RESULTADOS

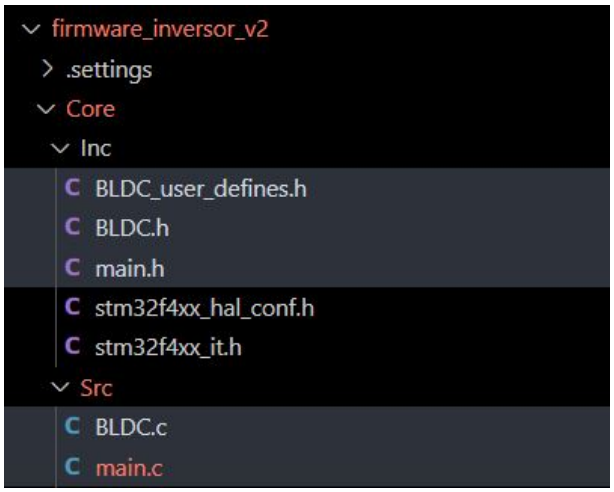
Atendendo ao objetivo, foi desenvolvido o firmware para controle de um inversor para acionamento de motor BLDC. Com o uso do ambiente de desenvolvimento da STM32, as configurações de ativação dos recursos são feitos pelo Cube IDE, de forma que neste capítulo serão destacadas as configurações e funções feitas manualmente.

Foi utilizada a biblioteca HAL CMSIS e o fluxo do programa é baseado em interrupção. Dos recursos do μC foram utilizados DMA, interrupções, GPIO, watchdog e o ADC, cuja aplicação será descrita em conjunto com as funções nas quais estão envolvidos.

4.1 ORGANIZAÇÃO DOS ARQUIVOS

As funções programadas especificamente para o BLDC estão reunidas em cinco arquivos, conforme Figura 21.

Figura 21 – Estrutura de arquivos.



Fonte: A autora.

Os arquivos *main.h* e *main.c* são criados pela IDE. *BLDC.h* e *BLDC.c* foram criados para organizar as funções específicas para o BLDC, e *BLDC_user_defines* é o arquivo no qual o usuário informa os parâmetros para sua aplicação.

4.1.1 Fluxo Principal

O arquivo *.h* contém a declaração das variáveis e protótipos das funções criadas nas configurações pela IDE, e os defines das entradas e saídas da GPIO. No arquivo *.c* são configurados os periféricos.

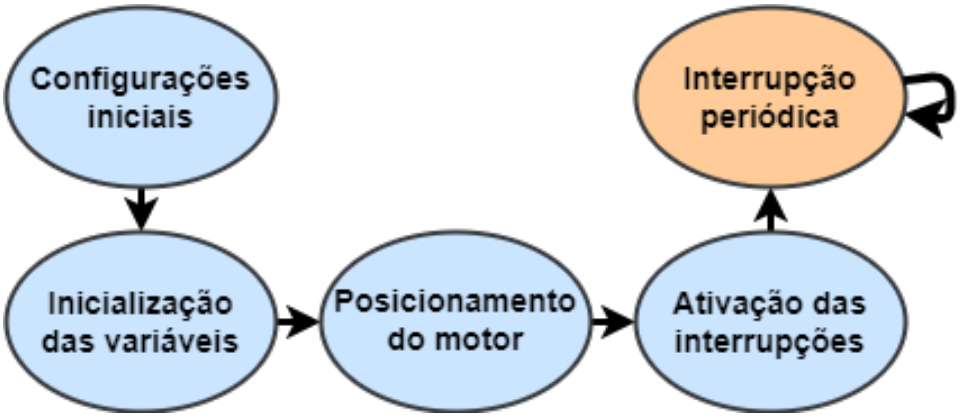
```
1     void SystemClock_Config(void);
```

```
2     static void MX_GPIO_Init(void);
3     static void MX_DMA_Init(void);
4     static void MX_ADC1_Init(void);
5     static void MX_TIM8_Init(void);
6     static void MX_TIM10_Init(void);
7     static void MX_TIM13_Init(void);
8     static void MX_IWDG_Init(void);
```

Código 1 – main.c: Configuração dos periféricos.

Na função *main* são chamadas as funções de inicialização do inversor explicadas na seção 4.1.2 e ativado o DMA e as interrupções. A Figura 22 apresenta de forma sucinta o comportamento invocado no arquivo *main.c*.

Figura 22 – Fluxo.



Fonte: A autora.

A execução do código se dá via interrupção, cuja função de *callback* se encontra em *main.c*, não sendo utilizado *pooling*.

4.1.2 Funções para Controle do Motor

No arquivo de cabeçalho constam os protótipos das funções específicas, listadas em 2.

```
1     void set_variables();
2     void set_start();
3     void modo_motor(uint8_t);
4     void modo_gerador(uint8_t);
5     void parado(void);
6     uint8_t position_next(uint8_t);
7     float calc_duty_cycle_motor(uint16_t);
8     float calc_duty_cycle_generator();
```

Código 2 – BLDC.h: Protótipos das funções.

A implementação das funções é feita no arquivo *.c*, e é explicada a partir da seção 4.2.

O arquivo *BLDC_users_defines.h* foi criado para o usuário poder informar as definições de características do equipamento e preferências, com as opções mostradas em Código 3.

```
1   #define CURRENT_SENSOR_RANGE  10 //(+- 5A)
2   #define DUTY_CYCLE_MAX        95
3   #define DUTY_CYCLE_MIN         2
4   #define I_MAX_MOTOR          1.2
5   #define I_MAX_GERADOR        0.7
6   #define PRECISION_ADC       1023 //2^n_bits - 1
7   #define V_MIN_BATERIA          4
8   #define V_BATERIA           24.0
9   #define VIN                    5
10  #define VREF                 5.0
```

Código 3 – BLDC_users_defines.h: Informações do usuário.

Os valores apresentados são os que foram utilizados nos testes.

4.2 CONFIGURAÇÕES E FUNCIONAMENTO

Nesta seção são descritas as configurações e o funcionamento do firmware, com apresentação das principais funções.

O firmware utiliza duas interrupções:

- **tim10**: é a parte principal do firmware, que realmente executa a operação, ela controla o funcionamento geral do sistema: modo de operação e ciclo de trabalho. A frequência do clock para esta interrupção é 96MHz.
- **tim13**: acionada quando a corrente está acima do nível estabelecido. Vai periodicamente zerar o ciclo de trabalho, é desativada quando a corrente estiver em valor aceitável novamente. Não interrompe o funcionamento do sistema. Utiliza clock com frequência de 48MHz.

A frequência da interrupção *tim10* foi calculada de forma que seja 10 vezes superior à frequência de mudança de fase das bobinas, e em cada ativação, além de verificar se precisa mudar a ativação das bobinas, recalcula o ciclo de trabalho e verifica o modo de operação. O funcionamento da interrupção *tim10* é explicado na seção 4.2.7.

As entradas analógicas dos sensores de aceleração, corrente e nível de bateria, simulados com potenciômetros, foram tratadas por ADC configurado para precisão de 10 bits, e os valores foram salvos e acessados por DMA, o que evita uso do processador, e acessados diretamente pelas funções que utilizaram as informações.

As entradas dos sensores *Hall* são lidas dentro da função da interrupção, fazendo chamada da função *get_position* que utiliza os pinos de entrada do GPIO:

```
1     uint8_t get_position(){
2     return (HAL_GPIO_ReadPin(HB_GPIO_Port, HA_Pin) << 2) | (
          HAL_GPIO_ReadPin(HB_GPIO_Port, HB_Pin) << 1) |
          HAL_GPIO_ReadPin(HC_GPIO_Port, HC_Pin);
3 }
```

Código 4 – Função get_position.

Dentro da função *main* são chamadas duas funções para inicializar variáveis e preparar o inversor: *set_variables* e *set_start*.

A função *set_variables* converte alguns dos valores informados pelo usuário para valores em bits, para agilizar a comparação com a leitura dos ADCs durante a execução:

```
1 void set_variables(){
2   Ig_max = (PRECISION_ADC/VREF)*(-I_MAX_GERADOR*VIN/
        CURRENT_SENSOR_RANGE + VIN/2.0)*(68.0/101.0);
3   Im_max = (PRECISION_ADC/VREF)*( I_MAX_MOTOR  *VIN/
        CURRENT_SENSOR_RANGE + VIN/2.0)*(68.0/101.0);
4   batery_min = V_MIN_BATERIA * PRECISION_ADC / (VIN*5); //o 5 eh da
        formula do sensor
5   batery_max = V_BATERIA * PRECISION_ADC / (VIN*5);
6   dc = 0;
7 }
```

Código 5 – Função set_variables.

Essa função depende dos sensores utilizados, podendo ser necessário adequá-la em caso do uso de outros sensores ou intervalos de operação. No caso, todos os sensores considerados fornecem sinais analógicos.

A função *set_start* identifica a posição inicial do rotor, para fazer o acionamento das bobinas corretas ao iniciar a rotação e, além de fazer a configuração inicial do inversor, verifica a situação da bateria e aciona alerta luminoso.

```
1 void set_start(){
2   if (data_adc1[2] < V_MIN_BATERIA) {
3     HAL_GPIO_WritePin(LED_bateria_GPIO_Port, LED_bateria_Pin, 1);
4   };
5   active_coils = 1;
6   while (active_coils != get_position()) {
7     active_coils = position_next(active_coils);
8     dc = DUTY_CYCLE_MIN;
9     modo_motor(active_coils);
```

```
10      HAL_Delay(5);
11      parado();
12      HAL_IWDG_Refresh(&hiwdg);
13    }
14  }
```

Código 6 – Função set_start.

O PWM e *tim10* são acionados após a *set_start*.

4.2.1 PWM

Para gerar o PWM foi utilizado o recurso do microcontrolador de gerar PWMs complementares, que permite implementar o PWM com retificação síncrona, demonstrado na Figura 2c. Para isso foram utilizados três canais do Timer 8 configurados como "PWM Generation CHx e CHxN", e cada canal resulta em duas saídas, o que atende às seis posições dos braços do inversor. O canal principal é configurado e o complementar faz o inverso. O timer 8 está ligado ao clock com frequência de 96MHz.

Em algumas posições no ciclo do inversor, o comportamento do canal complementar(CHN) não deve ser igual ao complemento do canal principal(CH), e isso exigiu algumas adaptações, uma vez que nível alto do inversor só pulsa ou permanece em baixa, e o nível baixo tem três posições, porém apenas é possível comandar o canal principal.

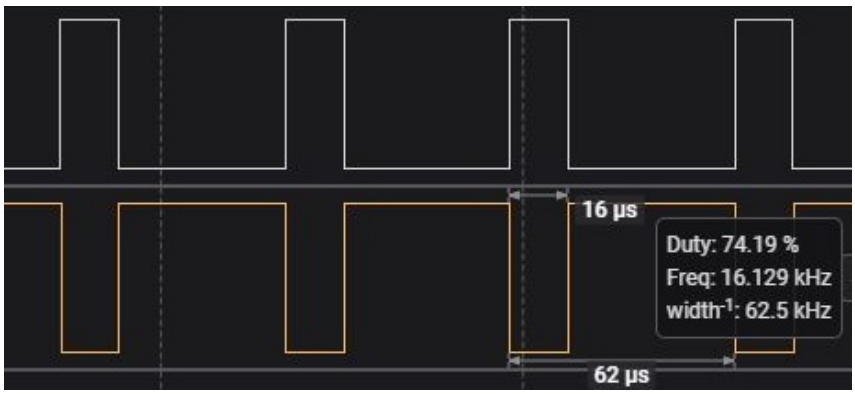
Adaptações:

- Foi utilizado o CHN como principal e o CH como secundário;
- PWM configurado como modo 2 que faz com os pulsos sejam invertidos (uma vez que CH foi para o braço inferior e CHN foi para o braço superior);
- Necessidade de um circuito inversor de sinal, pois, quando o comportamento não é complementar, o CHN é desativado, o que faz com que se mantenha em alta por definição do microcontrolador, e é necessário que ele se mantenha em baixa quando fora de funcionamento;
- Ajuste no ciclo de trabalho.

Foi feita tentativa de gerar o PWM complementar utilizando seis canais independentes, porém havia deslocamento entre os pulsos, e seria necessário configurar um tempo morto, o que diminuiria a eficiência do sistema. De qualquer forma, caso seja necessário utilizar tempo morto, o microcontrolador utilizado oferece a possibilidade de definir tempo morto diretamente nas configurações iniciais do PWM, na parte gráfica do ambiente de desenvolvimento.

A frequência de pulsação foi definida para 16kHz, e o resultado obtido é apresentado na Figura 23.

Figura 23 – Detalhe do PWM complementar.



Fonte: A autora.

No momento da captura da imagem da Figura 23 a aceleração era de cerca de 20% da tensão de referência.

4.2.2 Sequência de acionamento das bobinas

A função *set_start* identifica a posição inicial do rotor para fazer o acionamento das bobinas corretas na sequência de rotação, e uma vez que a interrupção *tim10* é acionada, sempre que é lida posição igual a das bobinas acionadas, as próximas bobinas são ativadas. Esse fluxo é feito por um *switch case* na função *position_next*:

```
1     //5-4-6-2-3-1 //
2     uint8_t position_next(uint8_t p){
3         switch (p) {
4                 case 5: return 4;
5                 case 4: return 6;
6                 case 6: return 2;
7                 case 2: return 3;
8                 case 3: return 1;
9                 case 1: return 5;
10                default: return 0;
11        };
12 }
```

Código 7 – Função position_next.

Esse posicionamento vale tanto para modo motor como para modo gerador, uma vez que a rotação do motor é sempre no mesmo sentido.

4.2.3 Modo de operação

O modo de operação é dado pela entrada do sensor de aceleração. Foi estabelecido um valor mínimo de 30 (num intervalo de 0 a 1023) para iniciar e manter o modo motor para dar mais estabilidade ao sistema, em função de ruídos. Com leitura entre 0 e 30 o veículo entra em modo gerador. O modo parado é ativado diretamente, por condições de sistema, zerando o ciclo de trabalho em todas as bobinas.

Essa operação se dá através da atribuição do ciclo de trabalho às bobinas, de 0, 100% e *dc*, que representam posição alta, baixa e pulsando.

Para que as bobinas da mesma posição do rotor sejam ativadas simultaneamente, foi necessário fazer a configuração diretamente nos registradores, e criou-se uma macro para isso, apresentada em Código 8.

```
1   #define __HAL_TIM_SET_MULTIPLE_COMPARE(__HANDLE__, __COMPARE1__
        , __COMPARE2__, __COMPARE3__) \
2     do { \
3       (__HANDLE__)->Instance->CCR1 = (__COMPARE1__); \
4       (__HANDLE__)->Instance->CCR2 = (__COMPARE2__); \
5       (__HANDLE__)->Instance->CCR3 = (__COMPARE3__); \
6     } while(0)
```

Código 8 – Macro para ativação simultânea das bobinas.

Os registros CCR1 a CCR3 configuram, com o valor em *_COMPAREX__*, o ciclo de trabalho de cada pulso.

A configuração do modo motor é feita em Código 9.

```
1  void modo_motor(uint8_t ac){
2    mode = 1;
3    TIM8->CCER &= ~TIM_CCER_CC1E;
4    TIM8->CCER &= ~TIM_CCER_CC2E;
5    TIM8->CCER &= ~TIM_CCER_CC3E;
6    switch (ac) {
7      case 5: //AC
8        TIM8->CCER |= TIM_CCER_CC1E;
9        __HAL_TIM_SET_MULTIPLE_COMPARE(&htim8, dc, 100, 0);
10       break;
11     case 4: //BC
12       TIM8->CCER |= TIM_CCER_CC2E;
13       __HAL_TIM_SET_MULTIPLE_COMPARE(&htim8, 100, dc, 0);
14       break;
15     case 6: //BA
16       TIM8->CCER |= TIM_CCER_CC2E;
17       __HAL_TIM_SET_MULTIPLE_COMPARE(&htim8, 0, dc, 100);
```

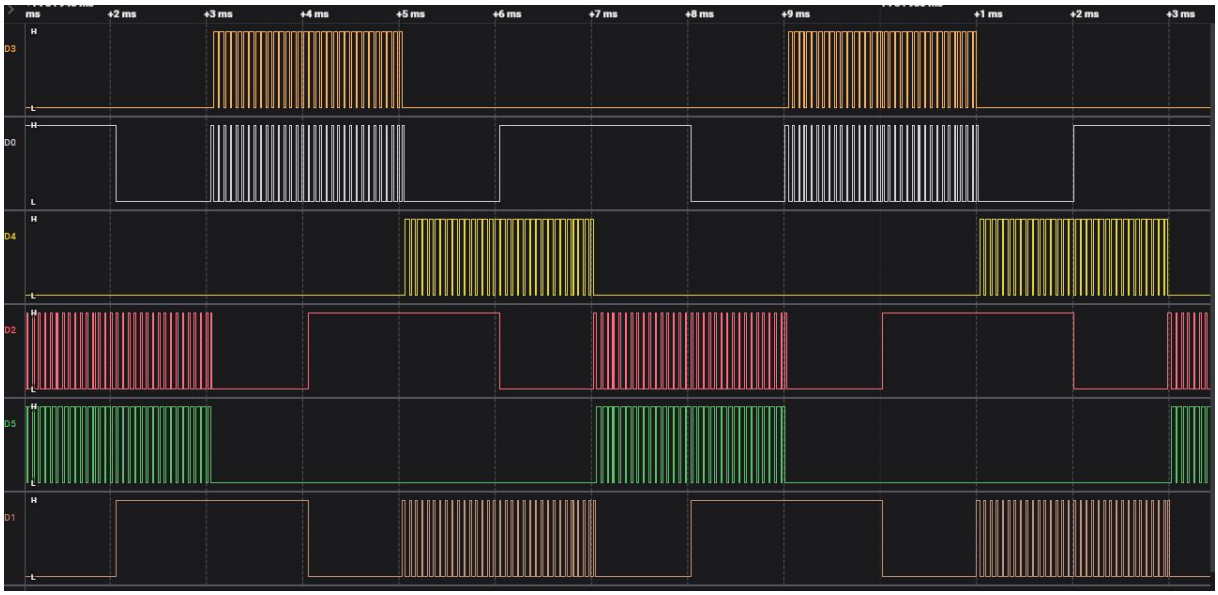
```
18        break;
19      case 2: //CA
20        TIM8->CCER |= TIM_CCER_CC3E;
21        __HAL_TIM_SET_MULTIPLE_COMPARE(&htim8, 0, 100, dc);
22        break;
23      case 3: //CB
24        TIM8->CCER |= TIM_CCER_CC3E;
25        __HAL_TIM_SET_MULTIPLE_COMPARE(&htim8, 100, 0, dc);
26        break;
27      case 1: //AB
28        TIM8->CCER |= TIM_CCER_CC1E;
29        __HAL_TIM_SET_MULTIPLE_COMPARE(&htim8, dc, 0, 100);
30        break;
31      default:
32        break;
33    }
34  }
```

Código 9 – Configuração modo motor.

Na função *modo_motor* inicialmente são desativados todos os canais complementares (os principais permanecem ativos), e é ativado somente o canal que irá pulsar, sendo atribuídos os valores adequados ao canal principal.

As saídas resultantes do modo motor são apresentadas na Figura 24.

Figura 24 – Modo motor com PWM complementar com retificação síncrona.



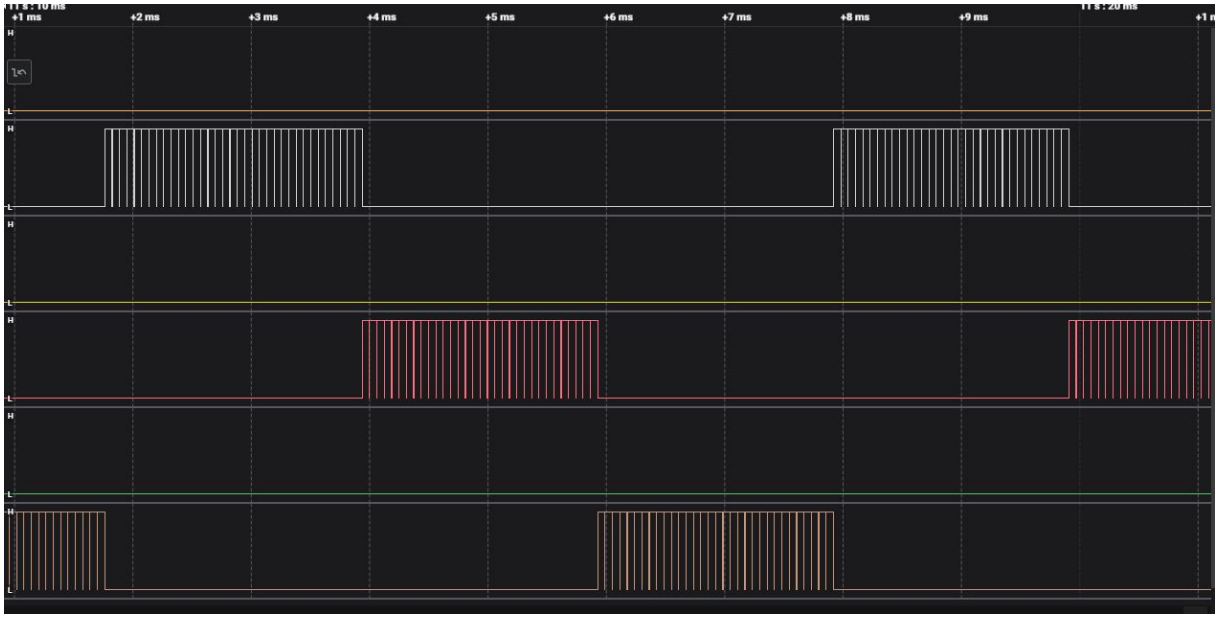
Fonte: A autora.

Na Figura 24 é possível verificar o funcionamento em *six steps*, topologia de operação em $120\circ$ e a alternância entre as bobinas, cujas bobinas sequenciais não são

acionadas para que não haja curtos-circuitos.

Em modo gerador não é utilizado o complemento do PWM, seguindo o ciclo da Figura 18. Para isso, todos os canais ChN são desativados, uma vez que os canais principais estão ligados ao nível inferior do inversor, e o sinal de saída é o mostrado na Figura 25.

Figura 25 – Modo gerador com acionamento de uma chave.



Fonte: A autora.

Em modo parado, todos os ciclos de trabalho são zerados.

```
1 void parado(void){
2   mode = 0;
3   dc = 100;
4   __HAL_TIM_SET_MULTIPLE_COMPARE(&htim8, dc, dc, dc); //por causa
       dos inversores na saida
5 }
```

Reforça-se que em função do uso do circuito inversor, conforme explicado na seção 4.2.1, o valor 100 representa ciclo de trabalho igual a zero e vice-versa.

4.2.4 Cálculo do ciclo de trabalho

O ciclo de trabalho utiliza duas funções: uma para funcionamento em modo motor (Código 10), e outra quando funcionando em moto gerador (Código 11).

- *calc_duty_cycle_motor* faz a proporção direta da leitura do motor com o intervalo definido para o ciclo de trabalho:

```
1 float calc_duty_cycle_motor(uint16_t data_adc_read){
2   float dc_aux;
```

```
3   dc_aux = DUTY_CYCLE_MIN + (DUTY_CYCLE_MAX-DUTY_CYCLE_MIN)*
        data_adc_read/PRECISION_ADC;
4   if (dc_aux > DUTY_CYCLE_MAX) return 100-DUTY_CYCLE_MAX;
5   else if (dc_aux < DUTY_CYCLE_MIN) return 100-DUTY_CYCLE_MIN;
6   else return 100-dc_aux;
7 }
```

Código 10 – Cálculo ciclo de trabalho modo motor.

- *calc_duty_cycle_gerador* procura manter o ciclo de trabalho no maior valor possível sem ultrapassar a corrente máxima permitida e respeitando o limite de carga da bateria.

```
1 float calc_duty_cycle_generator(){
2   float dc_aux;
3   dc_aux = DUTY_CYCLE_MAX*Ig_max/data_adc1[1];
4   if (dc_aux > DUTY_CYCLE_MAX) return 100-DUTY_CYCLE_MAX;
5   else if (dc_aux < DUTY_CYCLE_MIN) return 100-DUTY_CYCLE_MIN;
6   else return 100-dc_aux;
7 }
```

Código 11 – Cálculo ciclo de trabalho modo gerador.

Em caso de modo gerador e bateria cheia, simplesmente é ativado o modo parado.

4.2.5 Monitoramento tensão e corrente

O monitoramento da tensão objetiva proteger a bateria de carga além do limite em modo gerador e avisar o condutor do veículo quando a bateria está em nível baixo. Não é tomada nenhuma ação além de acender o alerta luminoso em caso de bateria em nível baixo, deixando a decisão de continuar, parar ou administrar o modo de condução a cargo do motorista.

O sensor de corrente está ligado à interrupção *tim13*, que é ativada por corrente acima do limite. Após o tempo estabelecido, se a corrente não estiver dentro do valor estabelecido, a interrupção ativa modo parado. Na próxima execução da interrupção *tim10* o sistema volta a funcionar em modo motor ou gerador e verifica a corrente novamente, reativando a interrupção caso não seja adequada. Também não toma ação definitiva, cabendo ao condutor do veículo decidir, caso o alerta luminoso permaneça, qual ação tomar.

Ambas as condições foram testadas com os potenciômetros e o sistema respondeu de acordo. A sinalização dos LEDs pode ser vista na Figura 19.

O uso de sensores e sinais luminosos pode ser suprimido, com as devidas exclusões no sistema, uma vez que cada componente adicional aumenta o consumo

do conjunto.

4.2.6 Freio regenerativo

O freio regenerativo apenas muda a sequência de chaveamento no inversor, de forma que a corrente faça o caminho do motor até a bateria, entrando em funcionamento quando o veículo entra em modo gerador.

Não é realizado nenhum travamento do motor ou atuação sobre sistema mecânica para parar o carro, isso depende de instalações físicas. Caso seja necessário esse tipo de comando em alguma alteração futura, é necessário alterar o firmware também para operar desta forma.

Para utilizar o freio regenerativo, dependendo das configurações do motor, pode ser necessário acoplar um conversor boost ao inversor.

Caso exista algum controle sobre a velocidade na ausência de aceleração, como por exemplo controle de cruzeiro, a forma que o freio regenerativo foi implementado deverá ser alterado para possibilitar a manutenção da velocidade.

4.2.7 Função *callback*

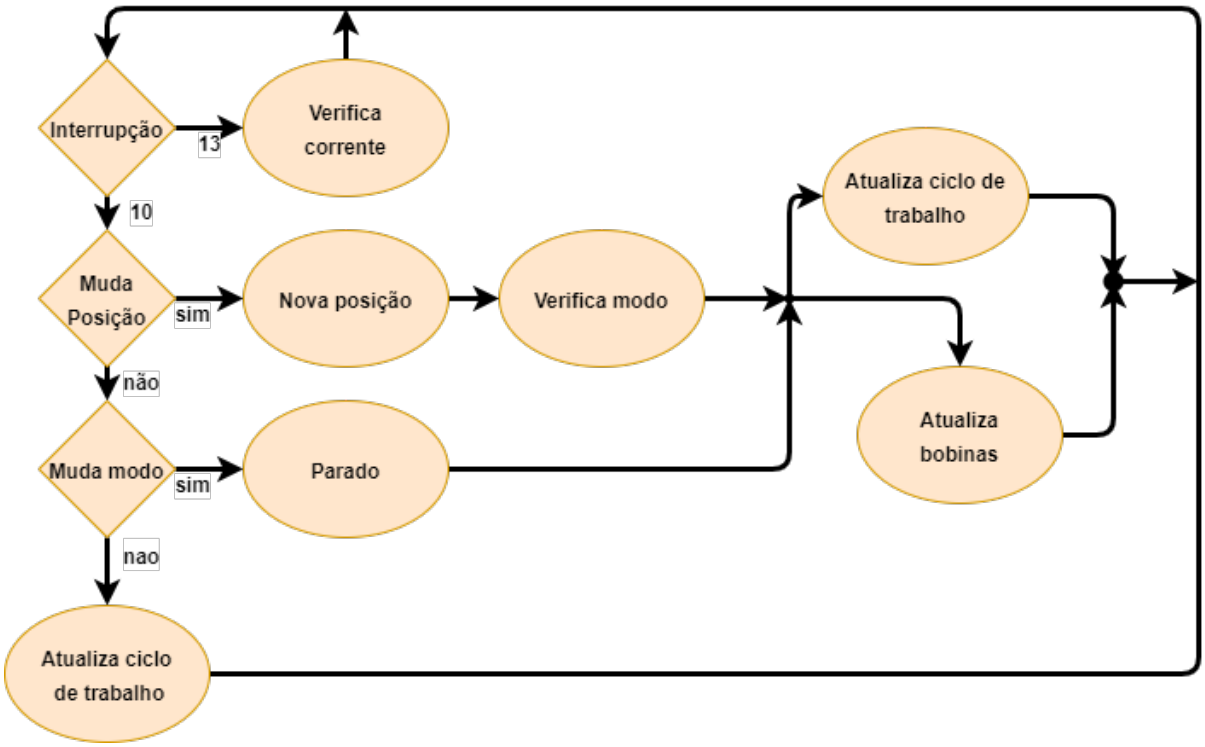
A função *HAL_TIM_PeriodElapsedCallback* é chamada quando ocorre uma interrupção periódica. Inicia verificando a origem da interrupção: se *tim13* ou *tim10*. A lógica interna da função *callback* é explicado de forma simplificada na Figura 26, na qual são omitidas as tarefas de leituras de sensores e watchdog para melhor visualização.

Caso a função *callback* tenha sido chamada pela *tim10*, é enviado sinal para o watchdog. Em seguida a função verifica se a leitura dos sensores *Hall* é igual à posição do rotor, caso afirmativo, calcula a próxima posição das bobinas, verifica o modo - se motor ou gerador, calcula o ciclo de trabalho adequado e faz o acionamento das bobinas. Se o rotor ainda não tiver atingido a próxima posição, é verificada a situação da bateria, e em seguida, conforme o modo, é calculado o ciclo de trabalho e atualizado no chaveamento. Na Figura 27 é possível ver a mudança do ciclo de trabalho em uma mesma posição de chaveamento, em função da aceleração.

Também, conforme o modo de operação, é verificada a corrente, se necessário é ativada a *tim13*. Caso o sistema esteja em modo gerador e carga da bateria estiver completa é chamado o modo parado, lembrando que *parado* não para o motor, apenas zera o ciclo de trabalho.

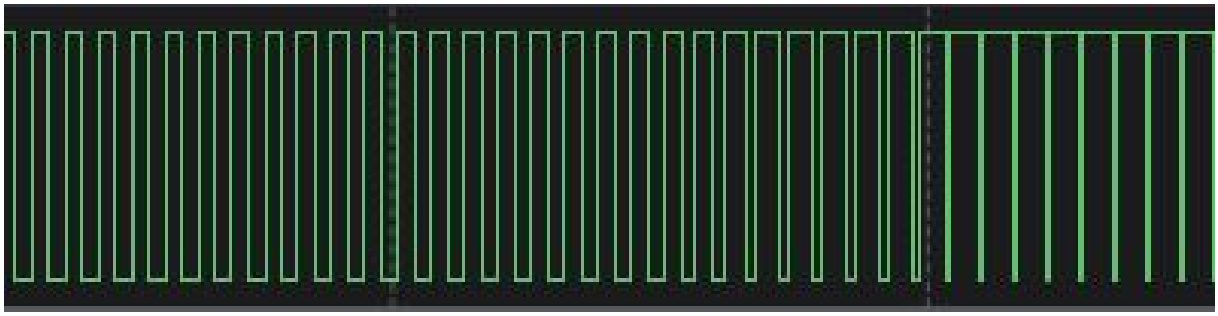
Pela extensão da função e quantidade de funções chamadas internamente, houve a preocupação da execução ser realizada em tempo hábil, uma vez que a *tim10* é chamada com uma frequência de 5kHz. Para fazer a verificação foi utilizado um pino configurado como saída, ao qual foi atribuído "1" no início da eecução da função *callback* e "0" no término, e fez se a leitura do pino com o analisador lógico. A Figura 28

Figura 26 – Fluxo .



Fonte: A autora.

Figura 27 – Comportamento do ciclo de trabalho sob aceleração.



Fonte: A autora.

apresenta o resultado, nos pulsos em azul.

Pode ser confirmado que a interrupção está acontecendo com a frequência esperada, e a função ocupada cerca de 3% do intervalo entre chamadas, assegurando a execução com segurança.

4.2.8 *Watchdog*

O *watchdog* foi implementado para reiniciar o sistema caso fique 100ms sem receber sinal. O sinal para o *watchdog* é enviado dentro da função *start* e quando ocorre

Figura 28 – Tempo de execução da função *callback*.

Fonte: A autora.

a interrupção *tim10*. Caso o sistema não chame a *tim10*, o que indicaria um travamento, o controlador é reiniciado, reiniciando a sua execução a partir das configurações iniciais. O *watchdog* foi testado através da execução da função sem o comando para envio do sinal dentro da interrupção, e respondeu de acordo.

4.3 CONSIDERAÇÕES FINAIS

O funcionamento do firmware segue, em linhas gerais, o fluxo proposto por Xia na Figura 12, chegando à estrutura similar, na qual a operação ocorre em uma interrupção que controla a posição do motor e ajusta a modulação.

O teste procurou identificar se o firmware produzia as respostas esperadas, através da análise das saídas em relação às entradas. A geração das posições - sinal de saída dos sensores *Hall* com Arduino foi feita em velocidade constante, porém com velocidade de 1kHz, acima da velocidade máxima de mudança de posição calculada para o motor, que seria de 500Hz. Isso pode ser conferido na Figura 24.

5 CONCLUSÕES

Este trabalho foi desenvolvido visando criar um firmware para acionar um motor sem escovas com imãs permanentes - BLDC. Os motores BLDC tem ampla aplicação em veículos elétricos, pelo seu desempenho e baixa manutenção.

Inicialmente foi realizada pesquisa para compreender requisitos necessários para um bom software para sistemas embarcados, que possui particularidades em relação a outros tipos de software, os recursos disponíveis e as estratégias possíveis de serem utilizadas. Em relação às particularidades do sistema embarcado tem-se o pouco espaço de memória, importância do baixo consumo energético, velocidade reduzida. Por outro lado, existem recursos para facilitar o trabalho, como as GPIOs, ADC, DMA, interrupções, bibliotecas, e diferentes estratégias de programação. Nos atributos de software, foi optado por priorizar o desempenho.

A característica marcante do BLDC é o fato de ser um motor CA acionado por uma fonte CC, necessitando de inversor de frequência para funcionar. No modelo considerado, os sensores *Hall* informam ao controlador a posição do rotor e este comanda o inversor para acionar as bobinas da próxima posição que o rotor deve ocupar para manter a rotação. O controle de velocidade de rotação é feito regulando a tensão com PWM, cuja frequência de comutação se mantém constante e a largura do ciclo de trabalho varia.

Com isso tem-se um resumo do que o controlador deve fazer: monitorar a posição do rotor e acionar as bobinas corretas, enviando um sinal com PWM com ciclo de trabalho adequado para que o motor tenha a rotação esperada. Como incremento, foi implementado o freio regenerativo, que visa recuperar energia para a bateria quando o motor não está gerando torque, o que também é feito com o controle do chaveamento no inversor.

A implementação foi feita utilizando o ambiente de desenvolvimento STM32 CubeIDE e o microcontrolador STM32F407G, programado em linguagem C com uso da biblioteca HAL CMSIS. O fluxo de programa é baseado em interrupção, e foram utilizados recursos da placa como *GPIO*, clock do sistema, *timers*, ADC, DMA, PWM e *watchdog*.

Visando obter um bom desempenho energético alguns cuidados foram tomados na implementação:

- freio regenerativo;
- implantação de PWM complementar com retificação síncrona, o que, de acordo com Infineon Technologies AG (2017), reduz as perdas nos diodos;
- não acionar chaves sequenciais para evitar curtos circuitos;

- PWM complementar gerado diretamente no microcontrolador na tentativa de evitar usar tempo morto.

O freio regenerativo depende de alteração do hardware para ser implementado. A dispensa de tempo morto precisa ser testada na prática e, caso seja necessário utilizar, é possível implementá-lo por configuração dos periféricos.

Os testes foram realizados utilizando um μC auxiliar que gerou sinais simulando os sensores *Hall*, e os sinais que comandam o inversor de frequência foram analisados com um analisador lógico digital para verificar o seu comportamento.

O desempenho foi validado pelos testes realizados, o firmware fornece as respostas esperadas para as situações abordadas, seja modo de operação, modulação de pulso e exibição de alerta. Seguindo as recomendações de desenvolvimento de software, foi criado arquivos específicos para as definições do usuário e para as funções de controle do motor. Também foram feitos módulos pequenos, chamando funções para a execução, e foi feita a separação do código em arquivos fonte e arquivos de cabeçalho.

Em relação aos objetivos específicos do trabalho, a pesquisa realizada proporcionou a compreensão dos dispositivos, foi feito o levantamento dos requisitos a serem atendidos pelo software e o software foi desenvolvido, com o funcionamento verificado por testes.

Como indicação para trabalhos futuros ficam as sugestões: desenvolver os atributos proteção e testabilidade, implementação de método de identificação e reação em caso de falha em bobinas no estator, medição de consumo de diferentes controladores para buscar opção mais econômica e implementação de um modo de motor em estado de cruzeiro.

REFERÊNCIAS

ALMEIDA, R. M. A. et al. **Programação de sistemas embarcados: desenvolvendo software para microcontroladores em linguagem C**. 2. ed. Rio de Janeiro: LTC, 2023.

BARBI, I. **Eletrônica de Potência: Inversores Monofasicos**. Florianopolis: Edição do autor, 2022.

BASS, L.; CLEMENTS, P.; KAZMAN, R. **Software Architecture in Practice**. 4. ed. [S.l.]: Pearson Education, Limited, 2021. ISBN 9780136886099.

BUCHI, R. **Brushless motors and controllers**. Norderstedt: Books on Demand, 2012.

CHAPMAN, S. J. **Máquinas elétricas**. 5. ed. Porto Alegre: Mc Graw Hill, 2013.

CHEN, C.; C., W.-C.; CHENG, M. Regenerative braking control for light electric vehicles. In: **2011 IEEE Ninth International Conference on Power Electronics and Drive Systems**. [S.l.]: IEEE, 2011.

CHRIST, J. M. R. **Projeto e implementação de um inversor trifasico para motor bldc aplicado a prototipo veicular de competição universitaria**. Trabalho de Conclusão de Curso (Graduação em Engenharia Meatrônica) — Centro Tecnologico de Joinville, Universidade Federal de Santa Catarina, Joinville, 2023.

COCRON, P. et al. Energy recapture through deceleration – regenerative braking in electric vehicles from a user perspective. **Ergonomics**, Informa UK Limited, v. 56, n. 8, p. 1203–1215, ago. 2013. ISSN 1366-5847.

DREHER, J. C.; ROSA, A. Frenagem regenerativa aplicada em motores brushless dc utilizados em veiculos eletricos. In: **4 Seminario de pesquisa, extensão e inovação do IFSC**. Gaspar: Anais, 2013. Disponível em: https://eventoscientificos.ifsc.edu.br/index.php/sepei/sepei2014/paper/view/472. Acesso em: 05 jun. 2024.

FIGUEIREDO, R. et al. Robust finite control set predictive control for induction machine drives. **Eletrônica de Potência**, v. 27, p. 1–8, 09 2022.

HUI, T.; BASU, K.; SUBBIAH, V. Permanent magnet brushless motor control techniques. In: **Proceedings. National Power Engineering Conference, 2003. PECon 2003**. [S.l.]: IEEE, 2003. p. 133–138. ISBN 0780382080.

INFINEON TECHNOLOGIES AG. **BLDC motor control software using XMC**. Munique, 2017. Disponível em: https://www.infineon.com/dgdl/Infineon-AP32359_BLDC_Motor_Control_Software-AN-v01_00-EN.pdf?fileId=5546d46258fc0bc101596988325e3f87. Acesso em: 25 abr. 2024.

JKONG MOTOR CO., LTDA. **JKM 42BLS Square BLDC Motor**. Changzou, 2020. Disponível em: https://www.jkongmotor.com/Product/JKM-42BLS-Square-BLDC-Motor.html. Acesso em: 26 maio 2024.

KARNAVAS, Y. L.; TOPALIDIS, A. S.; DRAKAKI, M. Development and implementation of a low cost c- based brushless dc motor sensorless controller: A practical analysis of hardware and software aspects. **Electronics**, MDPI AG, v. 8, n. 12, p. 1456, dez. 2019. ISSN 2079-9292.

KENJO, T.; NAGAMORI, S. **Permanent-magnet and brushless DC motors**. Oxford: Clarendon Press, 1985.

KRISHNAN, R. **Permanent magnet synchronous and brushless DC motor drives**. Boca Raton: CRC Press, 2015.

LACAMERA, D. **Embedded Systems Architecture: Explore architectural concepts, pragmatic design patterns, and best practices to produce robust systems**. [S.l.]: Packt Publishing - ebooks Account, 2018. 324 p. ISBN 9781788832502.

LIPOVSKI, G. J. **Introduction to Microcontrollers**: Architecture, programming, and interfacing for the freescale 68hc12 (academic press series in engineering). 2. ed. [S.l.]: Academic Press, 2004. 488 p. ISBN 9780124518384.

MARTIN, R. C. **Clean Architecture**: A craftsman's guide to software structure and design. [S.l.]: Pearson, 2017. 432 p. ISBN 9780134494166.

MEIRELES, J. P. **Acionamento de um motor bldc com modulação PWM trapezoidal**. Trabalho de Conclusão de Curso (Graduação em Engenharia Meatrônica) — Centro Tecnologico de Joinville, Universidade Federal de Santa Catarina, Joinville, 2023.

MUKHERJEE, A.; RAY, S.; DAS, A. Microcontroller based speed control and speed regulation scheme for bldc motor under variable loading conditions. v. 3, p. 2018, 07 2018.

RASHID, M. H. **Eletrônica de Potência: Dispositivos, circuitos e aplicações**. 4. ed. São Paulo: Pearson, 2014.

SANTOS JR., E. C.; SILVA, E. R. C. **Advanced power electronics converter: PWM converters processing AC voltages**. Picataway: IEEE Press, 2010.

Shell Eco-marathon. **About Shell Eco-marathon**. 2024. Disponível em: https://www.shellecomarathon.com/about.html. Acesso em: 28 abr. 2024.

SUBBARAO, M. et al. Design, control and performance comparison of pi and anfis controllers for bldc motor driven electric vehicles. **Measurement. Sensors**, Elsevier, v. 31, p. 101001, 2024. ISSN 2665-9174.

TANG, J. **Technical Manual Series Brushless Motor Structure and Rotation Principles**. Torrance, 2021. Disponível em: https://blog.orientalmotor.com/technical-manual-series-brushless-motor-structure-and-rotation-principles. Acesso em: 08 abr. 2024.

The MathWorks. **Six-Step Commutation of BLDC Motor Using Sensor Feedback**. 2021. Disponível em: https://www.mathworks.com/help/mcb/gs/six-step-commutation-bldc-motor-using-position-sensor.html. Acesso em: 17 abr. 2024.

UMANS, S. D. **Máquinas elétricas de Fitzgerald e Kingslehy**. 7. ed. São Paulo: Mc Graw Hill, 2014.

VALVANO, J. **Embedded Systems Real - Time Operating Systems for Arm Cortex M Microcontrollers**. [S.l.]: CreateSpace Independent Publishing Platform, 2017. 488 p. ISBN 9781466468863.

VALVANO, J. W. **Embedded Systems - Introduction to Arm® Cortex™-M Microcontrollers**. 5. ed. [S.l.]: Createspace Independent Publishing Platform, 2012. 507 p. ISBN 9781477508992.

XIA, C. **Permanent magnet brushless DC motors drives and controls**. Noida: Wiley, 2012.

YIU, J. **Definitive Guide to Arm Cortex-M23 and Cortex-M33 Processors**. Cambridge: Elsevier Science Technology, 2020. ISBN 9780128207352.

APÊNDICE A - PINOS DE CONEXÃO STM32

Função	**Pino**	**Direção**
Sensor Hall A	PE1	entrada
Sensor Hall B	PE3	entrada
Sensor Hall C	PE5	entrada
Acelerador	PA1	entrada
Corrente	PA2	entrada
Tensão	PA3	entrada
S1	PA5	saída
S3	PB0	saída
S5	PB1	saída
S2	PC6	saída
S4	PC7	saída
S6	PC8	saída
LED corrente	PB5	saída
LED bateria	PB7	saída