Simulação do controle FOC de motor PMSM aplicado a um motor BLDC

1st Felipe Lenschow
Programa de Pós-Graduação em Engenharia Elétrica
Universidade do Estado de Santa Catarina
Joinville, Santa Catarina, Brasil
felipe.lenschow@edu.udesc.br

***Resumo*—Este artigo apresenta a modelagem e simulação do acionamento utilizando Controle Orientado a Campo (FOC), comparando seu desempenho quando aplicado a um motor de corrente alternada sem escovas (BLAC), para o qual foi projetado, e a um Motor de Corrente Contínua Sem Escovas (BLDC). O modelo matemático no referencial dq é derivado para ambas as máquinas, e uma estratégia de controle empregando controladores Proporcional-Integral (PI) é implementada. A simulação, desenvolvida em Python, demonstra que, embora o FOC mantenha um controle preciso de velocidade para ambos os motores, ele introduz um ripple de torque significativo no motor BLDC devido à sua força contra-eletromotriz trapezoidal. Os resultados evidenciam as limitações do FOC padrão para máquinas BLDC em aplicações que exigem baixo ripple de torque.**

***Index Terms*—PMSM, Controle Orientado a Campo, Simulação, Python, Acionamento de Motor**

I. Introdução

Motores Síncronos de Ímãs Permanentes (PMSMs) são amplamente utilizados em aplicações industriais, veículos elétricos e robótica devido à sua alta eficiência, alta densidade de potência e excelente desempenho dinâmico. Para alcançar um controle de alto desempenho, o Controle Orientado a Campo (FOC) é comumente empregado. O FOC permite o controle independente de fluxo e torque transformando as correntes trifásicas do estator para um referencial girante dq, alinhado com o fluxo do rotor [1], porém, tal método de controle apresenta limitações quando aplicado a motores de corrente contínua sem escovas (BLDC), que possuem uma força contra-eletromotriz trapezoidal, em vez senoidal como nos motores de corrente alternada sem escovas (BLACs).

Este artigo desenvolve um ambiente de simulação para um sistema de acionamento PMSM. A simulação inclui a física do motor, o inversor de fonte de tensão e o algoritmo FOC. O objetivo é fornecer uma compreensão clara da dinâmica do sistema, validando o controle desenvolvido em [4], bem como mostrar os efeitos do mesmo FOC na dinâmica do motor BLDC.

II. Modelo do Sistema

A. Modelo de Motores PMSC

O modelo dinâmico dos PMSMs pode ser derivado a partir das equações de tensão de fase. Conforme descrito em [2], as tensões nos enrolamentos do estator são definidas por:

$$\begin{bmatrix} v_a \\ v_b \\ v_c \end{bmatrix} = \begin{bmatrix} R_s & 0 & 0 \\ 0 & R_s & 0 \\ 0 & 0 & R_s \end{bmatrix} \begin{bmatrix} i_a \\ i_b \\ i_c \end{bmatrix} + \frac{d}{dt} \begin{bmatrix} \psi_a \\ \psi_b \\ \psi_c \end{bmatrix} \tag{1}$$

onde ψ representa o fluxo total concatenado em cada enrolamento, dado por:

$$\begin{bmatrix} \psi_a \\ \psi_b \\ \psi_c \end{bmatrix} = \begin{bmatrix} L_{aa} & L_{ab} & L_{ac} \\ L_{ba} & L_{bb} & L_{bc} \\ L_{ca} & L_{cb} & L_{cc} \end{bmatrix} \begin{bmatrix} i_a \\ i_b \\ i_c \end{bmatrix} + \begin{bmatrix} \psi_{am} \\ \psi_{bm} \\ \psi_{cm} \end{bmatrix} \tag{2}$$

A matriz de indutâncias $\mathbf{L}_{abc}$ contém termos que variam com a posição do rotor θ_m devido à saliência dos polos. Como apresentado por [4], as indutâncias próprias e mútuas são dadas por:

$$L_{aa} = L_{al} + L_{aa0} + L_{g2}\cos(2\theta_e) \tag{3}$$
$$L_{bb} = L_{al} + L_{aa0} + L_{g2}\cos(2\theta_e + 2\pi/3) \tag{4}$$
$$L_{cc} = L_{al} + L_{aa0} + L_{g2}\cos(2\theta_e - 2\pi/3) \tag{5}$$
$$L_{ab} = -\frac{1}{2}L_{aa0} + L_{g2}\cos(2\theta_e - 2\pi/3) \tag{6}$$
$$L_{bc} = -\frac{1}{2}L_{aa0} + L_{g2}\cos(2\theta_e) \tag{7}$$
$$L_{ca} = -\frac{1}{2}L_{aa0} + L_{g2}\cos(2\theta_e + 2\pi/3) \tag{8}$$

onde L_{al} é a indutância de dispersão, L_{aa0} é a componente constante da indutância mútua e L_{g2} representa a amplitude da variação de indutância devido à saliência.

Para simplificar a análise, aplica-se a Transformada de Park para converter as variáveis do referencial trifásico (abc) para o referencial síncrono girante ($dq0$). A transformação é definida por $\mathbf{x}_{dq0} = \mathbf{T}\mathbf{x}_{abc}$, onde $\mathbf{T}$ é a matriz de transformação dada por:

$$\mathbf{T} = \frac{2}{3}\begin{bmatrix} \cos(\theta_e) & \cos(\theta_e - \frac{2\pi}{3}) & \cos(\theta_e + \frac{2\pi}{3}) \\ -\sin(\theta_e) & -\sin(\theta_e - \frac{2\pi}{3}) & -\sin(\theta_e + \frac{2\pi}{3}) \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \end{bmatrix} \tag{9}$$

Aplicando a Transformada de Park na equação (1), obtemos a equação de tensão no referencial $dq0$:

$$\mathbf{v}_{dq0} = \mathbf{T}\mathbf{v}_{abc} = \mathbf{T}\mathbf{R}\mathbf{i}_{abc} + \mathbf{T}\frac{d\boldsymbol{\psi}_{abc}}{dt} \tag{10}$$

Sabendo que $\mathbf{i}_{abc} = \mathbf{T}^{-1}\mathbf{i}_{dq0}$ e $\boldsymbol{\psi}_{abc} = \mathbf{T}^{-1}\boldsymbol{\psi}_{dq0}$:

$$\mathbf{v}_{dq0} = \mathbf{R}\mathbf{i}_{dq0} + \mathbf{T}\frac{d}{dt}(\mathbf{T}^{-1}\boldsymbol{\psi}_{dq0}) \tag{11}$$

Expandindo a derivada do produto::

$$\frac{d}{dt}(\mathbf{T}^{-1}\boldsymbol{\psi}_{dq0}) = \mathbf{T}^{-1}\frac{d\boldsymbol{\psi}_{dq0}}{dt} + \frac{d\mathbf{T}^{-1}}{dt}\boldsymbol{\psi}_{dq0} \tag{12}$$

Substituindo na equação (11):

$$\mathbf{v}_{dq0} = \mathbf{R}\mathbf{i}_{dq0} + \frac{d\boldsymbol{\psi}_{dq0}}{dt} + \mathbf{T}\frac{d\mathbf{T}^{-1}}{dt}\boldsymbol{\psi}_{dq0} \tag{13}$$

A matriz inversa da transformada de Park é definida como:

$$\mathbf{T}^{-1} = \begin{bmatrix} \cos(\theta_e) & -\sin(\theta_e) & 1 \\ \cos(\theta_e - \frac{2\pi}{3}) & -\sin(\theta_e - \frac{2\pi}{3}) & 1 \\ \cos(\theta_e + \frac{2\pi}{3}) & -\sin(\theta_e + \frac{2\pi}{3}) & 1 \end{bmatrix} \tag{14}$$

Sua derivada em relação ao tempo é:

$$\frac{d\mathbf{T}^{-1}}{dt} = \omega_e \begin{bmatrix} -\sin(\theta_e) & -\cos(\theta_e) & 0 \\ -\sin(\theta_e - \frac{2\pi}{3}) & -\cos(\theta_e - \frac{2\pi}{3}) & 0 \\ -\sin(\theta_e + \frac{2\pi}{3}) & -\cos(\theta_e + \frac{2\pi}{3}) & 0 \end{bmatrix} \tag{15}$$

Portanto, o termo $\mathbf{T}\frac{d\mathbf{T}^{-1}}{dt}$ resulta em um acoplamento cruzado entre os eixos d e q devido à velocidade angular elétrica ω_e:

$$\mathbf{T}\frac{d\mathbf{T}^{-1}}{dt} = \omega_e \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{16}$$

Em seguida, aplica-se a transformada de Park na equação de fluxo (2):

$$\boldsymbol{\psi}_{dq0} = \mathbf{T}\boldsymbol{\psi}_{abc} = \mathbf{T}(\mathbf{L}_{abc}\mathbf{i}_{abc} + \boldsymbol{\psi}_{m,abc}) \tag{17}$$

$$\boldsymbol{\psi}_{dq0} = \mathbf{T}\mathbf{L}_{abc}\mathbf{T}^{-1}\mathbf{i}_{dq0} + \mathbf{T}\boldsymbol{\psi}_{m,abc} \tag{18}$$

A matriz de indutância no referencial $dq0$, $\mathbf{L}_{dq0}$, é obtida pela transformação de similaridade $\mathbf{T}\mathbf{L}_{abc}\mathbf{T}^{-1}$. Esta operação desacopla as fases e elimina a dependência da posição do rotor, resultando em uma matriz diagonal constante:

$$\mathbf{L}_{dq0} = \begin{bmatrix} L_d & 0 & 0 \\ 0 & L_q & 0 \\ 0 & 0 & L_{al} \end{bmatrix} \tag{19}$$

onde as indutâncias de eixo direto e quadratura são constantes, e dadas por:

$$L_d = L_{al} + \frac{3}{2}(L_{aa0} + L_{g2}) \tag{20}$$

$$L_q = L_{al} + \frac{3}{2}(L_{aa0} - L_{g2}) \tag{21}$$

Definindo o fluxo dos ímãs transformado como $\boldsymbol{\psi}_{m,dq0} = \mathbf{T}\boldsymbol{\psi}_{m,abc}$, a equação de fluxo no referencial $dq0$ é dada por:

$$\boldsymbol{\psi}_{dq0} = \mathbf{L}_{dq0}\mathbf{i}_{dq0} + \boldsymbol{\psi}_{m,dq0} \tag{22}$$

Substituindo (22) em (13):

$$\mathbf{v}_{dq0} = \mathbf{R}\mathbf{i}_{dq0} + \mathbf{L}_{dq0}\frac{d\mathbf{i}_{dq0}}{dt} + \mathbf{T}\frac{d\mathbf{T}^{-1}}{dt}\mathbf{L}_{dq0}\mathbf{i}_{dq0} + \mathbf{e}_{dq0} \tag{23}$$

onde $\mathbf{e}_{dq0}$ é a força contra-eletromotriz no referencial $dq0$, dada por:

$$\mathbf{e}_{dq0} = \frac{d\boldsymbol{\psi}_{m,dq0}}{dt} + \mathbf{T}\frac{d\mathbf{T}^{-1}}{dt}\boldsymbol{\psi}_{m,dq0} \tag{24}$$

A forma matricial explícita no referencial dq torna-se:

$$\frac{d\mathbf{i}_{dq0}}{dt} = \mathbf{A}\mathbf{i}_{dq0} + \mathbf{B}(\mathbf{v}_{dq0} - \mathbf{e}_{dq0}) \tag{25}$$

onde as matrizes de estado $\mathbf{A}$ e de entrada $\mathbf{B}$ são dadas por:

$$\mathbf{A} = \begin{bmatrix} -\frac{R_s}{L_d} & \omega_e\frac{L_q}{L_d} \\ -\omega_e\frac{L_d}{L_q} & -\frac{R_s}{L_q} \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} \frac{1}{L_d} & 0 \\ 0 & \frac{1}{L_q} \end{bmatrix} \tag{26}$$

Isolando as derivadas para cada componente:

$$\frac{dI_d}{dt} = \frac{1}{L_d}(V_d - R_sI_d + \omega_eL_qI_q - e_d) \tag{27}$$

$$\frac{dI_q}{dt} = \frac{1}{L_q}(V_q - R_sI_q - \omega_eL_dI_d - e_q) \tag{28}$$

B. Diferenças entre BLDC e BLAC

Ambos são motores síncronos de ímãs permanentes, mas a principal distinção entre o motor BLDC e o BLAC reside na distribuição do fluxo magnético e na forma de onda da força contra-eletromotriz (Back-EMF).

1) BLAC: O Motor Síncrono de Ímãs Permanentes (PMSM), ou BLAC, possui uma distribuição de fluxo senoidal. No referencial síncrono dq, as componentes da força contra-eletromotriz tornam-se constantes:

$$e_d = 0 \tag{29}$$

$$e_q = \omega_e\lambda_m \tag{30}$$

Isso simplifica o controle, pois as referências de corrente podem ser constantes para torque constante.

2) BLDC: O motor BLDC possui uma distribuição de fluxo trapezoidal. Ao contrário do BLAC, as componentes e_d e e_q no referencial síncrono não são constantes, mas variam com a posição do rotor, apresentando ondulações características.

A Fig. 1 ilustra que e_q apresenta ondulações (harmônicos de ordem $6k$) em vez de ser um valor DC puro, e e_d também oscila em torno de zero. Essas variações introduzem ripple de torque se o controle FOC padrão (projetado para PMSM) for aplicado sem compensação.

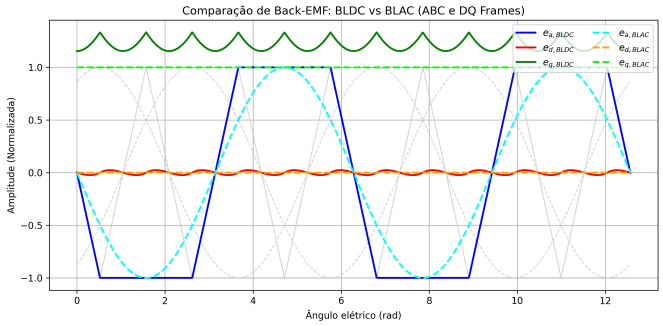

Figura 1. Formas de onda da força contra-eletromotriz trapezoidal no referencial trifásico (e_a, e_b, e_c) e suas componentes no referencial síncrono (e_d, e_q).

C. Equação de Torque

A potência instantânea de entrada nos terminais do estator no referencial abc é dada por:

$$P_{abc} = v_a i_a + v_b i_b + v_c i_c \tag{31}$$

Utilizando a transformação de Park, e considerando a transformação invariante em amplitude e a componente zero nula, a potência pode ser reescrita no referencial $dq0$:

$$P_{dq} = \frac{3}{2}(v_d i_d + v_q i_q) \tag{32}$$

Substituindo as equações de tensão v_d e v_q na equação de potência (13 em 32):

$$\begin{aligned} P_{dq} = \frac{3}{2}\Bigg[& i_d \left(R_s i_d + L_d \frac{di_d}{dt} - \omega_e L_q i_q + e_d \right) \\ & + i_q \left(R_s i_q + L_q \frac{di_q}{dt} + \omega_e L_d i_d + e_q \right) \Bigg] \end{aligned} \tag{33}$$

Reagrupando os termos, podemos identificar três componentes distintas de potência:

$$\begin{aligned} P_{dq} = & \underbrace{\frac{3}{2} R_s (i_d^2 + i_q^2)}_{P_{cu}} + \underbrace{\frac{3}{2} \left(L_d i_d \frac{di_d}{dt} + L_q i_q \frac{di_q}{dt} \right)}_{P_{mag}} \\ & + \underbrace{\frac{3}{2} \left[(e_d i_d + e_q i_q) + \omega_e (L_d - L_q) i_d i_q \right]}_{P_{em}} \end{aligned} \tag{34}$$

onde P_{cu} representa as perdas resistivas no cobre, P_{mag} é a taxa de variação da energia magnética armazenada, e P_{em} é a potência eletromecânica convertida. Portanto, a potência eletromecânica é:

$$P_{em} = \frac{3}{2}(e_d i_d + e_q i_q) + \frac{3}{2}\omega_e (L_d - L_q) i_d i_q \tag{35}$$

O torque eletromagnético T_e relaciona-se com a potência mecânica pela velocidade angular do rotor ω_m. Considerando $\omega_e = P\omega_m$, onde P é o número de pares de polos:

$$T_e = \frac{P_{em}}{\omega_m} = \frac{P P_{em}}{\omega_e} \tag{36}$$

Substituindo a expressão de P_{em}:

$$T_e = \frac{3}{2} P \left[\frac{e_d i_d + e_q i_q}{\omega_e} + (L_d - L_q) i_d i_q \right] \tag{37}$$

Podemos separar o torque em duas componentes: o torque de alinhamento (ou síncrono), devido à interação com a força contra-eletromotriz, e o torque de relutância, devido à assimetria do rotor:

$$T_e = \underbrace{\frac{3}{2} P \frac{e_d i_d + e_q i_q}{\omega_e}}_{T_{sinc}} + \underbrace{\frac{3}{2} P (L_d - L_q) i_d i_q}_{T_{rel}} \tag{38}$$

O termo $T_{relutancia}$ surge em máquinas com saliência $(L_d \neq L_q)$, sendo proporcional a $(L_d - L_q) I_d I_q$.

Para máquinas de ímãs permanentes com $L_d = L_q$, o torque de relutância se torna zero.

No caso do BLAC, onde $e_d = 0$ e $e_q = \omega_e \lambda_m$, a equação simplifica-se para a forma clássica:

$$T_e = \frac{3}{2} P \lambda_m I_q \tag{39}$$

Para o motor BLDC, como visto anteriormente, e_d e e_q variam com a posição do rotor, resultando em ondulações de torque se as correntes I_d e I_q forem mantidas constantes. A simulação utiliza a forma geral baseada nas componentes de força contra-eletromotriz para capturar este comportamento.

A dinâmica mecânica é descrita por:

$$J \frac{d\omega_m}{dt} = T_e - T_L - B\omega_m - T_c \tag{40}$$

onde J é o momento de inércia, ω_m é a velocidade mecânica, T_L é o torque de carga, B é o coeficiente de atrito viscoso, e T_c é o torque de atrito de Coulomb.

D. Parâmetros do Motor

Os parâmetros do motor utilizado neste trabalho, baseados em [4], são apresentados na Tabela I.

Tabela I
PARÂMETROS DO MOTOR BLAC

Parâmetro	Símbolo	Valor
Pares de polos	P	21
Resistência do estator	R_s	4.485 Ω
Indutância de eixo direto	L_d	54.8 mH
Indutância de eixo em quadratura	L_q	54.8 mH
Fluxo magnético	λ_m	0.201 Wb
Momento de inércia	J	0.1444 kg·m^2
Atrito viscoso	B	0.0057 Nms/rad

III. Estratégia de Controle

A estratégia de controle adotada é o Controle Orientado a Campo (FOC), que permite o controle desacoplado de torque e fluxo. O diagrama de blocos geral do sistema de controle é apresentado na Fig. 2. O sistema consiste em uma malha externa de velocidade e duas malhas internas de corrente (I_d e I_q).

A referência de velocidade é comparada com a velocidade medida, gerando uma referência de corrente de quadratura I_q^* através de um controlador PI. A referência de corrente de eixo direto I_d^* é mantida em zero para maximizar o torque por ampère (estratégia MTPA para SPMSM). As correntes medidas são transformadas para o referencial síncrono e controladas por controladores PI independentes, gerando as tensões de referência para o algoritmo SVPWM.

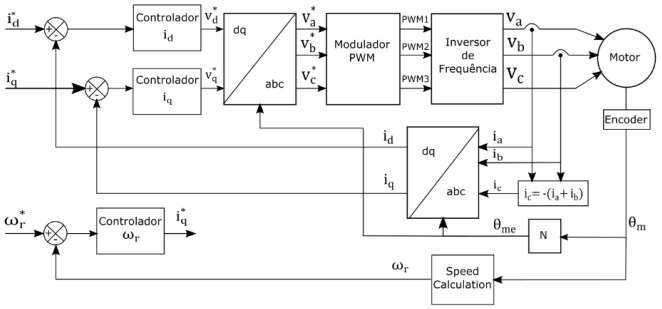

Figura 2. Diagrama de blocos geral do controle FOC.

A. Modelo do Inversor

O Inversor de Fonte de Tensão (VSI) trifásico é modelado idealmente, assumindo que as tensões de referência geradas pelo controlador são aplicadas com precisão aos terminais do motor, limitadas apenas pela tensão do barramento CC V_{bus}. Os limites da Modulação por Largura de Pulso Vetorial Espacial (SVPWM) são considerados saturando a magnitude do vetor de tensão para $V_{bus}/\sqrt{3}$ para um movimento suave.

B. Projeto do Controlador de Corrente

O controle de corrente é realizado no referencial síncrono dq, utilizando dois controladores PI independentes para regular as correntes I_d e I_q. O diagrama de blocos do controlador de corrente implementado é mostrado na Fig. 3.

A banda passante (ω_b) da malha de corrente é projetada para ser aproximadamente 10 vezes maior que a da malha de velocidade, garantindo o desacoplamento dinâmico. Neste trabalho, adotou-se $\omega_b = 350$ Hz e $\xi = 4$. Os ganhos resultantes são calculados conforme as equações analíticas apresentadas em [4], tendo como premissas que o acoplamento entre os eixos d e q é tratado como um distúrbio a ser compensado, que a dinâmica dos eixos d e q são iguais (válido para L_d = L_q), e que a resistencia do motor seja muito menor que o ganho do controlador:

$$k_p = \frac{\xi\omega_b 2L_q}{\sqrt{2\xi^2+1+\sqrt{(1+2\xi)^2+1}}} \tag{41}$$

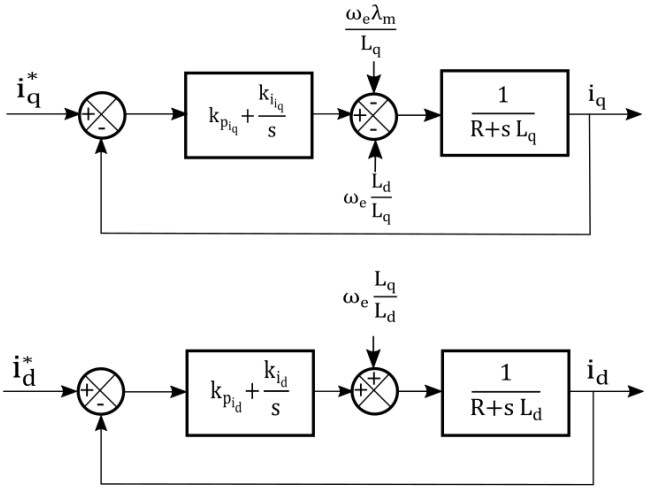

Figura 3. Diagrama de blocos do controlador de corrente [4].

$$k_i = \frac{L_q\omega_b^2}{2\xi^2+1+\sqrt{(1+2\xi)^2+1}} \tag{42}$$

Substituindo os parâmetros, obtém-se $k_p = 119$ e $k_i = 4015$.

C. Projeto do Controlador de Velocidade

O controlador de velocidade é responsável por regular a velocidade mecânica do rotor, gerando a referência de corrente I_q^* para a malha interna. Utiliza-se um controlador PI. O diagrama de blocos do controlador de velocidade é apresentado na Fig. 4. A referência de corrente de eixo direto I_d^* é mantida em zero.

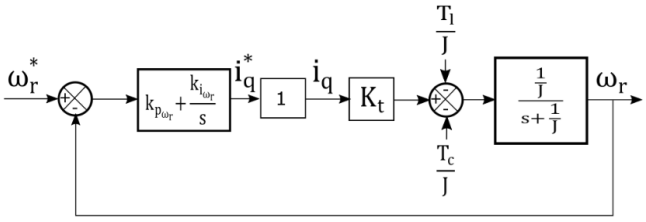

Figura 4. Diagrama de blocos do controlador de velocidade [4].

O projeto do controlador assume que a dinâmica da malha de corrente é suficientemente rápida para ser considerada ideal (ganho unitário) na faixa de frequências de interesse da malha de velocidade. A banda passante escolhida foi $\omega_b = 35$ Hz com um fator de amortecimento $\xi = 1.0$. As equações para os ganhos, considerando o torque de carga e atrito de Coulomb como distúrbios e o atrito viscoso desprezível frente à inércia, são dadas por:

$$k_p = \frac{\xi\omega_b 2J}{K_t\sqrt{2\xi^2+1+\sqrt{(1+2\xi)^2+1}}} \tag{43}$$

$$k_i = \frac{J\omega_b^2}{K_t(2\xi^2 + 1 + \sqrt{(1+2\xi)^2+1})} \tag{44}$$

Os ganhos obtidos para o controlador de velocidade são $k_p = 1.25$ e $k_i = 55$.

IV. Simulação

A. Implementação

Para validar a análise comparativa, foi desenvolvido um simulador modular em Python, projetado para flexibilidade e facilidade de manutenção, pode ser visto no apêndice A. A estrutura do simulador é baseada em componentes independentes que interagem através de interfaces bem definidas, permitindo a substituição de modelos de motor e estratégias de controle sem a necessidade de reescrever o código principal.

A arquitetura do simulador é composta pelos seguintes módulos principais:

- **Motor**: Define a física da máquina elétrica. Foram implementadas classes distintas para `BLDCMotor` e `BLACMotor`, ambas herdando ou seguindo uma interface comum que expõe métodos para o cálculo da dinâmica eletromecânica. Isso permite a troca direta entre os modelos de motor no script principal.
- **Controller**: Implementa a lógica de controle, como o FOC. Este módulo recebe as referências e as medições dos sensores para calcular as tensões de referência.
- **Inverter**: Modela o comportamento do inversor de potência, convertendo as tensões de referência em tensões aplicadas aos terminais do motor, considerando a tensão do barramento DC.
- **Sensors**: Simula a aquisição de dados, fornecendo medições de corrente, posição e velocidade, podendo incluir ruídos ou atrasos para maior realismo.

O ciclo de simulação, executado em `Simulate.py`, opera em passos de tempo discretos (T_s). Em cada iteração, o fluxo de dados segue a sequência:

1) **Definição de Perfil**: Atualização das referências de velocidade e torque de carga conforme o tempo de simulação.
2) **Sensoriamento**: O módulo de sensores lê o estado atual do motor.
3) **Controle**: O controlador calcula as ações de controle baseadas nas leituras dos sensores e nas referências.
4) **Atuação**: O inversor aplica as tensões resultantes ao motor.
5) **Física**: O modelo do motor atualiza suas variáveis de estado (correntes, velocidade, posição) resolvendo as equações diferenciais do sistema.
6) **Registro**: As variáveis de interesse são armazenadas para análise posterior.

Essa abordagem modular facilitou a comparação direta entre o desempenho do FOC em motores BLDC e BLAC, garantindo que as mesmas condições de controle e simulação fossem aplicadas a ambos os casos, isolando as diferenças intrínsecas das máquinas. Os parâmetros do motor utilizados são apresentados na Tabela I.

O perfil de simulação seque o padrão desenvolvido em [4], que consiste em:

- $t = 0.0s$: Início em 40 RPM.
- $t = 0.2s$: Degrau de torque de carga de 20 Nm aplicado.
- $t = 0.4s$: Degrau de referência de velocidade para 80 RPM.
- $t = 0.6s$: Degrau de referência de velocidade de volta para 40 RPM.
- $t = 0.8s$: Torque de carga removido.

V. Resultados

A Fig. 5 e a Fig. 6 mostram a resposta do sistema comparando o motor BLAC e o motor BLDC sob o mesmo controle FOC.

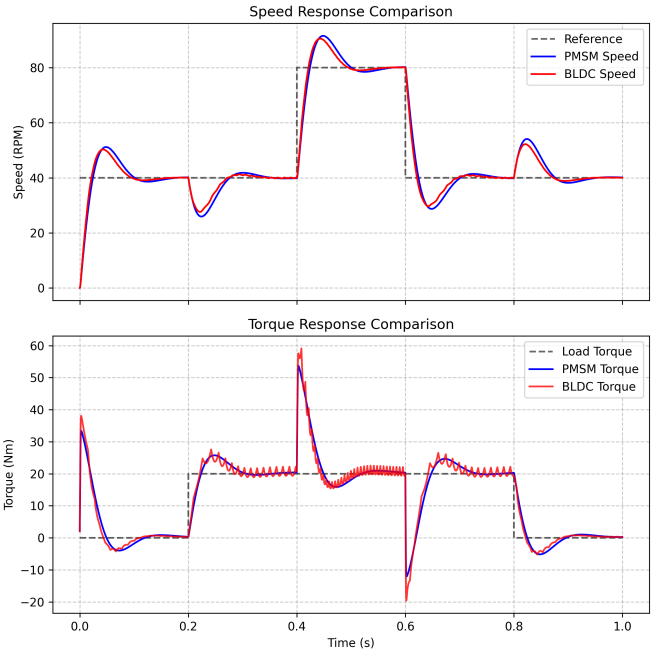

Figura 5. Comparação de Velocidade e Torque entre BLAC e BLDC.

O controlador de velocidade rastreia a referência de RPM com precisão para ambos os motores, mesmo tendo uma leve oscilação e ter o comportamento levemente mais rápido no caso do BLDC. Já para o torque, observa-se que o motor BLDC apresenta um ripple significativo em comparação com o BLAC. Este ripple é intrínseco à aplicação do controle FOC, projetado para formas de onda senoidais, em uma máquina com força contra-eletromotriz trapezoidal. A frequência fundamental deste ripple é de 6 vezes a frequência elétrica, correspondendo às comutações que ocorreriam em um controle trapezoidal padrão, efeito esse que pode ser observado no intervalo $0.4 < t < 0.6$ s, em que a velocidade é dobrada, e a frequência do ripple de torque também é dobrada.

As correntes I_d e I_q no caso do BLDC (Fig. 6) mostram oscilações claras. O controlador PI de corrente tenta compensar a variação da Back-EMF trapezoidal, que não é constante

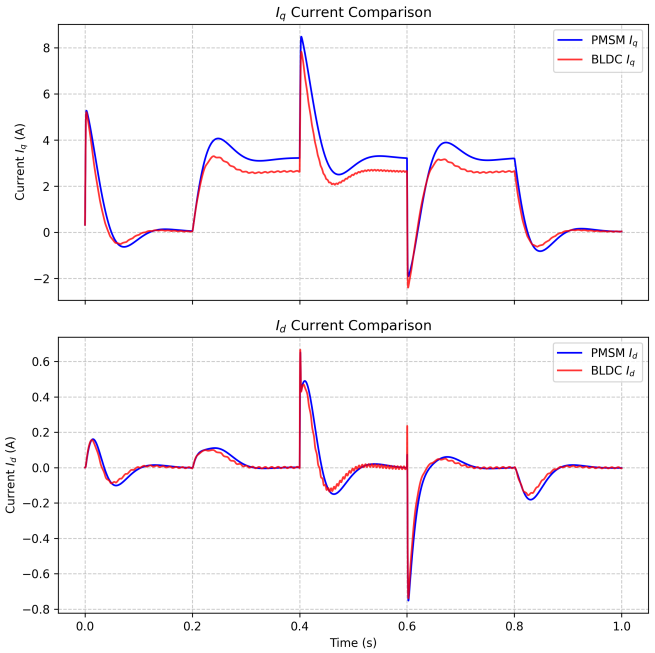

Figura 6. Comparação das Correntes I_d e I_q entre BLAC e BLDC.

no referencial dq, resultando nessas ondulações. Embora a velocidade média seja mantida, a qualidade do torque é degradada, o que pode gerar vibrações e ruído acústico em aplicações reais.

VI. Conclusão

Uma simulação completa de um acionamento PMSM usando FOC foi apresentada e comparada com a aplicação da mesma estratégia em um motor BLDC. A implementação modular em Python permitiu uma análise detalhada das diferenças de desempenho.

Os resultados confirmam que, embora o FOC possa controlar a velocidade de um motor BLDC com eficácia, ele introduz um ripple de torque considerável devido ao descasamento entre a estratégia de controle (baseada em modelos senoidais) e a física do motor (Back-EMF trapezoidal). Para aplicações de alto desempenho utilizando motores BLDC, estratégias de minimização de ripple de torque ou o uso de controle trapezoidal dedicado seriam mais apropriados. O ambiente de simulação desenvolvido serve como uma ferramenta valiosa para estudar e validar tais estratégias de controle avançado.

Apêndice A
Código da Simulação

Os códigos fonte da simulação desenvolvida em Python são apresentados a seguir.

A. *Simulate.py*

```python
import numpy as np
import matplotlib.pyplot as plt
import math
from BLACMotor import BLACMotor
from BLDCMotor import BLDCMotor
from FOCController import FOCController
from Inverter import Inverter
from Sensors import Sensors

if __name__ == "__main__":
    Ts = 1e-4
    t_end = 1.0

    # motor = BLACMotor(Ts)
    motor = BLDCMotor(Ts)
    controller = FOCController(Ts)
    inverter = Inverter()
    sensors = Sensors()

    num_steps = int(t_end / Ts)

    history = {
        'time': np.zeros(num_steps),
        'rpm_ref': np.zeros(num_steps),
        'rpm_act': np.zeros(num_steps),
        'Iq': np.zeros(num_steps),
        'Id': np.zeros(num_steps),
        'Te': np.zeros(num_steps),
        'Tload': np.zeros(num_steps),
        'Vbus': np.zeros(num_steps)
    }

    print("Starting Simulation...")

    t = 0.0
    for k in range(num_steps):
        t = k * Ts
        history['time'][k] = t

        # ------------------------------------------------------------
        # 1. INPUTS & PROFILE
        # ------------------------------------------------------------
        RPMref = 40.0
        Tload = 0.0

        if t > 0.2:
            Tload = 20.0
        if t > 0.4:
            RPMref = 80.0
        if t > 0.6:
            RPMref = 40.0
        if t > 0.8:
            Tload = 0.0

        V_bus = 311.0

        # ------------------------------------------------------------
        # 2. SENSORING STEP
        # ------------------------------------------------------------
        Ia, Ib, Ic, theta_e, Wr_meas = sensors.measure(motor, RPMref)

        # ------------------------------------------------------------
        # 3. CONTROLLER STEP
        # ------------------------------------------------------------
        Va_ref, Vb_ref, Vc_ref = controller.control_step(
            RPMref, Wr_meas, Ia, Ib, Ic, theta_e, V_bus
        )

        # ------------------------------------------------------------
        # 4. INVERTER STEP
        # ------------------------------------------------------------
        Va, Vb, Vc = inverter.step(Va_ref, Vb_ref, Vc_ref, V_bus)

        # ------------------------------------------------------------
        # 5. MOTOR PHYSICS STEP
        # ------------------------------------------------------------
        Te = motor.physics_step(Va, Vb, Vc, Tload, Ia, Ib, Ic)

        # ------------------------------------------------------------
        # 6. DATA LOGGING
        # ------------------------------------------------------------
        history['rpm_ref'][k] = RPMref
        history['rpm_act'][k] = motor.Wr * 60 / (2*math.pi)
        history['Iq'][k] = motor.Iq
        history['Id'][k] = motor.Id
        history['Te'][k] = Te
        history['Tload'][k] = Tload
        history['Vbus'][k] = V_bus

    data = history

    fig, (ax1, ax2, ax3) = plt.subplots(3, 1, figsize=(10, 12), sharex=True)

    ax1.plot(data['time'], data['rpm_ref'], 'r--', label='RPM Ref')
    ax1.plot(data['time'], data['rpm_act'], 'b-', label='RPM Actual')
    ax1.set_ylabel('Speed (RPM)')
    ax1.set_title('FOC Simulation')
    ax1.legend()
    ax1.grid(True)

    ax2.plot(data['time'], data['Te'], 'g-', label='Electromagnetic Torque')
    ax2.plot(data['time'], data['Tload'], 'k--', label='Load Torque')
    ax2.set_ylabel('Torque (Nm)')
    ax2.legend()
    ax2.grid(True)

    ax3.plot(data['time'], data['Iq'], 'c-', label='Iq (A)')
    ax3.plot(data['time'], data['Id'], 'm-', label='Id (A)')
    ax3.set_ylabel('Current (A)')
    ax3.set_xlabel('Time (s)')
    ax3.legend()
    ax3.grid(True)

    plt.tight_layout()
    plt.show()
```

B. BLACMotor.py

```python
import math
import numpy as np

class BLACMotor:
    def __init__(self, Ts):
        self.Ts = Ts
        self.Npp = 21.0
        self.Rs = 4.485
        self.Ld = 0.0548
        self.Lq = 0.0548
        self.Lambda_m = 0.201
        self.Bn = 0.0057
        self.J = 0.1444
        self.Tc = 0.3006

        self.Id = 0.0
        self.Iq = 0.0
        self.Wr = 0.0
        self.theta = 0.0
        self.theta_e = 0.0

    def physics_step(self, Va, Vb, Vc, Tload, Ia, Ib, Ic):
        We = self.Npp * self.Wr
        self.theta_e = self.Npp * self.theta
        self.theta_e = self.theta_e % (2 * math.pi)

        cos_t = math.cos(self.theta_e)
        sin_t = math.sin(self.theta_e)
        cos_t_m = math.cos(self.theta_e - 2*math.pi/3)
        sin_t_m = math.sin(self.theta_e - 2*math.pi/3)
        cos_t_p = math.cos(self.theta_e + 2*math.pi/3)
        sin_t_p = math.sin(self.theta_e + 2*math.pi/3)

        Vd_ref = (2.0/3.0) * (Va * cos_t + Vb * cos_t_m + Vc * cos_t_p)
        Vq_ref = (2.0/3.0) * (-Va * sin_t - Vb * sin_t_m - Vc * sin_t_p)

        I_alpha = Ia
        I_beta = (Ia + 2.0*Ib) / math.sqrt(3.0)

        Id_meas = I_alpha * cos_t + I_beta * sin_t
        Iq_meas = -I_alpha * sin_t + I_beta * cos_t

        g11 = 1 - (self.Ts * (self.Rs / self.Ld))
        g12 = (We * self.Lq * self.Ts) / self.Ld
        g21 = -We * self.Ld * self.Ts / self.Lq
        g22 = 1 - self.Rs * self.Ts / self.Lq
        h11 = self.Ts / self.Ld
        h22 = self.Ts / self.Lq
        i2 = -We * self.Lambda_m * self.Ts / self.Lq

        ed = 0.0
        eq = We * self.Lambda_m

        Id_next = g11 * Id_meas + g12 * Iq_meas + h11 * Vd_ref
        Iq_next = g21 * Id_meas + g22 * Iq_meas + h22 * Vq_ref + i2

        Te = 1.5 * self.Npp * Iq_next * (self.Lambda_m + (self.Ld - self.Lq) *
              Id_next)

        Tc_dir = self.Tc if self.Wr > 0 else (-self.Tc if self.Wr < 0 else 0)

        accel = (Te - Tload - (self.Bn * self.Wr) - Tc_dir) / self.J
        self.Wr += accel * self.Ts

        self.theta += self.Wr * self.Ts
        self.theta = self.theta % (2*math.pi)

        self.Id = Id_next
        self.Iq = Iq_next

        return Te
```

C. BLDCMotor.py

```python
import math
import numpy as np

class BLDCMotor:
    def __init__(self, Ts):
        self.Ts = Ts
        self.Npp = 21.0
        self.Rs = 4.485
        self.Ld = 0.0548
        self.Lq = 0.0548
        self.Lambda_m = 0.201
        self.Bn = 0.0057
        self.J = 0.1444
        self.Tc = 0.3006

        self.Id = 0.0
        self.Iq = 0.0
        self.Wr = 0.0
        self.theta = 0.0
        self.theta_e = 0.0

    def _trapezoidal_shape(self, theta):
        t = theta % (2 * math.pi)
        pi = math.pi

        if t < pi/6:
            return t * (6/pi) # 0 to 1
        elif t < 5*pi/6:
            return 1.0
        elif t < 7*pi/6:
            return 1.0 - (t - 5*pi/6) * (6/pi) # 1 to -1
        elif t < 11*pi/6:
            return -1.0
        else:
            return -1.0 + (t - 11*pi/6) * (6/pi) # -1 to 0

    def physics_step(self, Va, Vb, Vc, Tload, Ia, Ib, Ic):
        We = self.Npp * self.Wr
        self.theta_e = self.Npp * self.theta
        self.theta_e = self.theta_e % (2 * math.pi)

        E_mag = We * self.Lambda_m

        ea = -E_mag * self._trapezoidal_shape(self.theta_e)
        eb = -E_mag * self._trapezoidal_shape(self.theta_e - 2*math.pi/3)
        ec = -E_mag * self._trapezoidal_shape(self.theta_e + 2*math.pi/3)

        cos_t = math.cos(self.theta_e)
        sin_t = math.sin(self.theta_e)

        e_alpha = (2/3) * (ea - 0.5*eb - 0.5*ec)
        e_beta = (2/3) * (math.sqrt(3)/2 * (eb - ec))

        ed = e_alpha * cos_t + e_beta * sin_t
        eq = -e_alpha * sin_t + e_beta * cos_t

        cos_t_m = math.cos(self.theta_e - 2*math.pi/3)
        sin_t_m = math.sin(self.theta_e - 2*math.pi/3)
        cos_t_p = math.cos(self.theta_e + 2*math.pi/3)
        sin_t_p = math.sin(self.theta_e + 2*math.pi/3)

        Vd_ref = (2.0/3.0) * (Va * cos_t + Vb * cos_t_m + Vc * cos_t_p)
        Vq_ref = (2.0/3.0) * (-Va * sin_t - Vb * sin_t_m - Vc * sin_t_p)

        I_alpha = Ia
        I_beta = (Ia + 2.0*Ib) / math.sqrt(3.0)

        Id_meas = I_alpha * cos_t + I_beta * sin_t
        Iq_meas = -I_alpha * sin_t + I_beta * cos_t

        dId = (1.0/self.Ld) * (Vd_ref - self.Rs*Id_meas + We*self.Lq*Iq_meas - ed
              )
        dIq = (1.0/self.Lq) * (Vq_ref - self.Rs*Iq_meas - We*self.Ld*Id_meas - eq
              )

        Id_next = Id_meas + self.Ts * dId
        Iq_next = Iq_meas + self.Ts * dIq

        if abs(We) > 1e-3:
            Te = 1.5 * self.Npp * (ed * Id_next + eq * Iq_next) / We
        else:
            Te = 1.5 * self.Npp * self.Lambda_m * Iq_next


        Tc_dir = self.Tc if self.Wr > 0 else (-self.Tc if self.Wr < 0 else 0)

        accel = (Te - Tload - (self.Bn * self.Wr) - Tc_dir) / self.J
        self.Wr += accel * self.Ts

        self.theta += self.Wr * self.Ts
        self.theta = self.theta % (2*math.pi)

        self.Id = Id_next
        self.Iq = Iq_next

        return Te
```

D. FOCController.py

```python
import math

class FOCController:
    def __init__(self, Ts, Imax=8.0):
        self.Ts = Ts
        self.Imax = Imax

        self.Kps = 1.25
        self.Kis = 55.0
        self.KpId = 119.0
        self.KiId = 4015.0
        self.KpIq = 119.0
        self.KiIq = 4015.0

        self.Ui_s = 0.0
        self.Ui_Id = 0.0
        self.Ui_Iq = 0.0

    def control_step(self, RPMref, Wr, Ia, Ib, Ic, theta_e, Vbus):
        I_alpha = Ia
        I_beta = (Ia + 2.0*Ib) / math.sqrt(3.0)

        cos_t = math.cos(theta_e)
        sin_t = math.sin(theta_e)

        Id = I_alpha * cos_t + I_beta * sin_t
        Iq = -I_alpha * sin_t + I_beta * cos_t

        error_speed = (RPMref * 2 * math.pi / 60.0) - Wr

        Up_s = self.Kps * error_speed
        Ui_s_next = self.Ui_s + (self.Kis * self.Ts * error_speed)

        Iq_ref = Up_s + Ui_s_next
        self.Ui_s = Ui_s_next

        Id_ref = 0.0

        err_Iq = Iq_ref - Iq
        Up_Iq = self.KpIq * err_Iq
```

```
        Ui_Iq_next = self.Ui_Iq + (self.KiIq * self.Ts * err_Iq)
        Vq_ref = Up_Iq + Ui_Iq_next
        self.Ui_Iq = Ui_Iq_next

        err_Id = Id_ref - Id
        Up_Id = self.KpId * err_Id
        Ui_Id_next = self.Ui_Id + (self.KiId * self.Ts * err_Id)
        Vd_ref = Up_Id + Ui_Id_next
        self.Ui_Id = Ui_Id_next

        cos_t = math.cos(theta_e)
        sin_t = math.sin(theta_e)

        Va_ref = cos_t * Vd_ref - sin_t * Vq_ref
        Vb_ref = math.cos(theta_e - 2*math.pi/3) * Vd_ref - math.sin(theta_e - 2*
            math.pi/3) * Vq_ref
        Vc_ref = math.cos(theta_e + 2*math.pi/3) * Vd_ref - math.sin(theta_e + 2*
            math.pi/3) * Vq_ref

        V_mag = math.sqrt(Vd_ref**2 + Vq_ref**2)
        max_V = Vbus / math.sqrt(3)

        if V_mag > max_V:
            ratio = max_V / V_mag
            Vd_ref *= ratio
            Vq_ref *= ratio

        return Va_ref, Vb_ref, Vc_ref
```

E. *Inverter.py*

```
import math

class Inverter:
    def __init__(self):
        pass

    def step(self, Va_ref, Vb_ref, Vc_ref, Vbus):
        limit = Vbus / 2.0

        Va = max(-limit, min(limit, Va_ref))
        Vb = max(-limit, min(limit, Vb_ref))
        Vc = max(-limit, min(limit, Vc_ref))

        return Va, Vb, Vc
```

F. *Sensors.py*

```
import math

class Sensors:
    def __init__(self):
        pass

    def measure(self, motor, RPMref):
        cos_t = math.cos(motor.theta_e)
        sin_t = math.sin(motor.theta_e)

        I_alpha = motor.Id * cos_t - motor.Iq * sin_t
        I_beta = motor.Id * sin_t + motor.Iq * cos_t

        Ia = I_alpha
        Ib = -0.5 * I_alpha + (math.sqrt(3)/2.0) * I_beta
        Ic = -0.5 * I_alpha - (math.sqrt(3)/2.0) * I_beta

        theta_e = motor.theta_e

        Wr_meas = motor.Wr

        return Ia, Ib, Ic, theta_e, Wr_meas
```

REFERÊNCIAS

[1] P. Pillay and R. Krishnan, ”Modeling, simulation, and analysis of permanent-magnet motor drives. I. The permanent-magnet synchronous motor drive,”IEEE Transactions on Industry Applications, vol. 25, no. 2, pp. 265-273, March-April 1989.

[2] MathWorks, ”BLDC - Three-winding brushless direct current motor with trapezoidal flux distribution,”[Online]. Available: https://www.mathworks.com/help/sps/ref/bldc.html.

[3] MathWorks, ”PMSM - Permanent Magnet Synchronous Motor,”[Online]. Available: https://www.mathworks.com/help/sps/ref/pmsm.html.

[4] Matheus Alexandre Bevilaqua, ”Implementação do Controle de Velocidade de Motores Síncronos a Ímãs Permanentes em Plataforma LabVIEW FPGA”, Universidade do Estado de Santa Catarina, 2015.