

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—This paper presents the modeling and simulation of the drive using Field-Oriented Control (FOC), comparing its performance when applied to a Brushless AC (BLAC) motor, for which it was designed, and to a Brushless DC (BLDC) motor. The mathematical model in the dq reference frame is derived for both machines, and the control strategy employed is the Proportional-Integral (PI) controller, following the development of [4]. The simulation, developed in Python, demonstrates that, although FOC maintains precise speed control for both motors, it introduces significant torque ripple in the BLDC motor due to its trapezoidal back-electromotive force. The results highlight the limitations of standard FOC for BLDC machines in applications requiring low torque ripple.

Index Terms—PMSM, BLDC, BLAC, FOC, Simulation, Python, Motor Control

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 de 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 frequência 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.

A organização deste trabalho é apresentada a seguir. A Seção II descreve o modelo matemático do sistema e as diferenças entre os motores. A Seção III detalha a estratégia de controle e o projeto dos controladores. A Seção IV apresenta a implementação do simulador. A Seção V discute os resultados obtidos e, finalmente, a Seção VI conclui o artigo.

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:

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

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

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

A matriz de indutâncias 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:

$$L_{aa} = L_{al} + L_{aa0} + L_{g2} \cos(2\theta_e) \quad (3)$$

$$L_{bb} = L_{al} + L_{aa0} + L_{g2} \cos(2\theta_e + 2\pi/3) \quad (4)$$

$$L_{cc} = L_{al} + L_{aa0} + L_{g2} \cos(2\theta_e - 2\pi/3) \quad (5)$$

$$L_{ab} = -\frac{1}{2}L_{aa0} + L_{g2} \cos(2\theta_e - 2\pi/3) \quad (6)$$

$$L_{bc} = -\frac{1}{2}L_{aa0} + L_{g2} \cos(2\theta_e) \quad (7)$$

$$L_{ca} = -\frac{1}{2}L_{aa0} + L_{g2} \cos(2\theta_e + 2\pi/3) \quad (8)$$

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:

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

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

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

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

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

Expandindo a derivada do produto::

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

Substituindo na equação (11):

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

A matriz inversa da transformada de Park é definida como:

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

Sua derivada em relação ao tempo é:

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

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 :

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

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

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

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

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:

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

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

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

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

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:

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

Substituindo (22) em (13):

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

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

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

A forma matricial explícita no referencial dq torna-se:

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

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

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

Isolando as derivadas para cada componente:

$$\frac{dI_d}{dt} = \frac{1}{L_d}(V_d - R_s I_d + \omega_e L_q I_q - e_d) \quad (27)$$

$$\frac{dI_q}{dt} = \frac{1}{L_q}(V_q - R_s I_q - \omega_e L_d I_d - e_q) \quad (28)$$

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:

$$e_d = 0 \quad (29)$$

$$e_q = \omega_e \lambda_m \quad (30)$$

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.

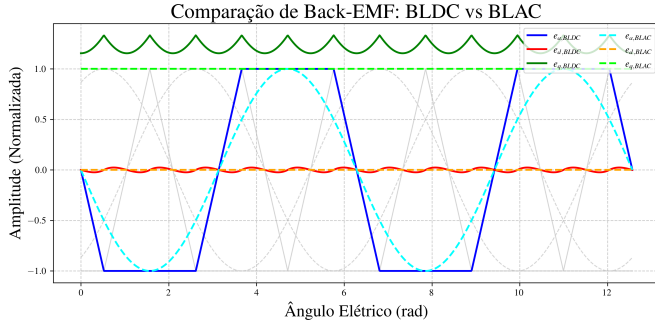


Figura 1. Formas de onda da força contra-eletromotriz trapezoidal no referencial trifásico (e_a, e_b, e_c) e suas componentes no referencial síncrono (e_d, e_q).

C. Equação de Torque

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

$$P_{abc} = v_a i_a + v_b i_b + v_c i_c \quad (31)$$

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

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

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

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

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

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

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:

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

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:

$$T_e = \frac{P_{em}}{\omega_m} = \frac{P P_{em}}{\omega_e} \quad (36)$$

Substituindo a expressão de P_{em} , 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:

$$T_e = \underbrace{\frac{3}{2} P \frac{e_d i_d + e_q i_q}{\omega_e}}_{T_{sinc}} + \underbrace{\frac{3}{2} P (L_d - L_q) i_d i_q}_{T_{relut}} \quad (37)$$

O termo T_{relut} 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:

$$T_e = \frac{3}{2} P \lambda_m I_q \quad (38)$$

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:

$$J \frac{d\omega_m}{dt} = T_e - T_L - B \omega_m - T_c \quad (39)$$

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) projetado para um motor BLAC. 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.

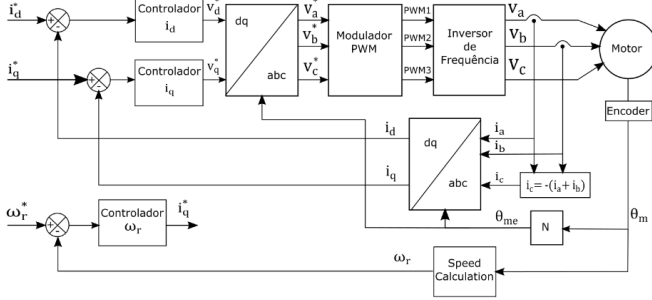


Figura 2. Diagrama de blocos geral do controle FOC.

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 é mostrado na Fig. 3.

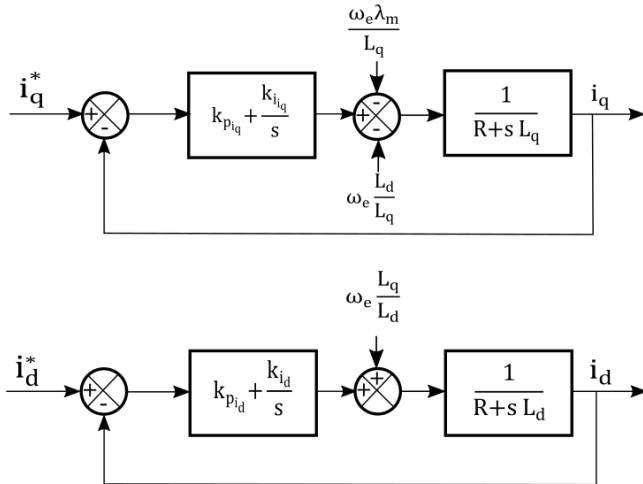


Figura 3. Diagrama de blocos do controlador de corrente [4].

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 $\xi = 4$, seguindo o controlador desenvolvido por [4]. Os ganhos resultantes são calculados, 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 resistência do motor seja muito menor que o ganho do controlador:

$$k_p = \frac{\xi \omega_b 2 L_q}{\sqrt{2\xi^2 + 1} + \sqrt{(1 + 2\xi)^2 + 1}} \quad (40)$$

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

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.

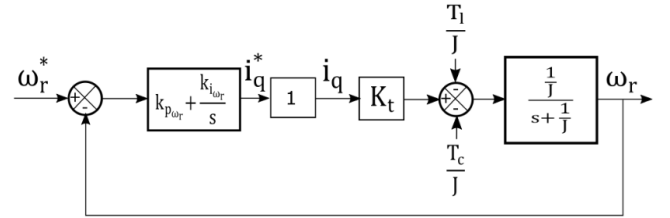


Figura 4. Diagrama de blocos do controlador de velocidade [4].

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 $\xi = 1.0$, novamente seguindo o controlador desenvolvido por [4]. 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:

$$k_p = \frac{\xi \omega_b 2 J}{K_t \sqrt{2\xi^2 + 1} + \sqrt{(1 + 2\xi)^2 + 1}} \quad (42)$$

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

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

IV. SIMULADOR

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 o motor BLDC e para o BLAC, ambas herdando 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 segue 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.

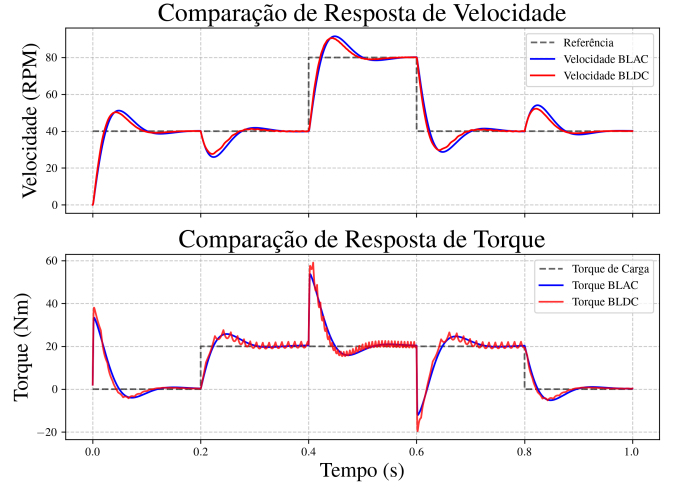


Figura 5. Comparação de Velocidade e Torque entre BLAC e BLDC.

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.

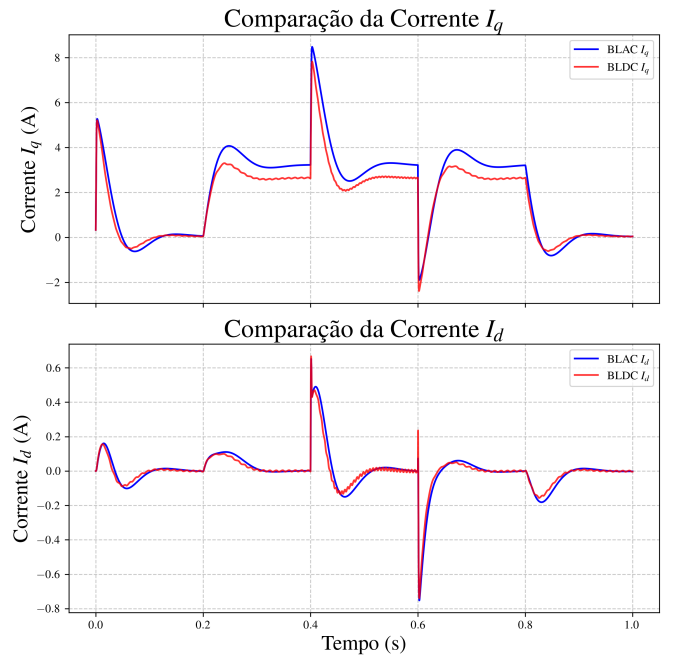


Figura 6. Comparação das Correntes I_d e I_q entre BLAC e BLDC.

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 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.

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 Bevilacqua, "Implementação do Controle de Velocidade de Motores Síncronos a Ímãs Permanentes em Plataforma LabVIEW FPGA", Universidade do Estado de Santa Catarina, 2015.

APÊNDICE A CÓDIGO DA SIMULAÇÃO

Os códigos fonte da simulação desenvolvida em Python são apresentados a seguir. O projeto completo também pode ser acessado no repositório GitHub: <https://github.com/FelipeLenschow/Artigo-FOC-no-BLAC-vs-BLDC>.

A. Simulate.py

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3 import math
4 from BLACMotor import BLACMotor
5 from BLDCMotor import BLDCMotor
6 from FOCController import FOCController
7 from Inverter import Inverter
8 from Sensors import Sensors
9
10 if __name__ == "__main__":
11     Ts = 1e-4
12     t_end = 1.0
13
14     # motor = BLACMotor(Ts)
15     motor = BLDCMotor(Ts)
16     controller = FOCController(Ts)
17     inverter = Inverter()
18     sensors = Sensors()
19
20     num_steps = int(t_end / Ts)
21
22     history = {
23         'time': np.zeros(num_steps),
24         'rpm_ref': np.zeros(num_steps),
25         'rpm_act': np.zeros(num_steps),
26         'Iq': np.zeros(num_steps),
27         'Id': np.zeros(num_steps),
28         'Te': np.zeros(num_steps),
29         'Tload': np.zeros(num_steps),
30         'Vbus': np.zeros(num_steps)
31     }
32
33     print("Starting Simulation...")
34
35     t = 0.0
36     for k in range(num_steps):
37         t = k * Ts
38
39         # -----
40         # 1. INPUTS & PROFILE
41         # -----
42         RPMref = 40.0
43         Tload = 0.0
44
45         if t > 0.2:
46             Tload = 20.0
47         if t > 0.4:
48             RPMref = 80.0
49         if t > 0.6:
50             RPMref = 40.0
51         if t > 0.8:
52             Tload = 0.0
53
54         V_bus = 311.0
55
56         # -----
57         # 2. SENSING STEP
58         # -----
59         Ia, Ib, Ic, theta_e, Wr_meas = sensors.measure(motor, RPMref)
60
61         # -----
62         # 3. CONTROLLER STEP
63         # -----
64         Va_ref, Vb_ref, Vc_ref = controller.control_step(
65             RPMref, Wr_meas, Ia, Ib, Ic, theta_e, V_bus
66         )
67
68         # -----
69         # 4. INVERTER STEP
70         # -----
71         Va, Vb, Vc = inverter.step(Va_ref, Vb_ref, Vc_ref, V_bus)
72
73         # -----
74         # 5. MOTOR PHYSICS STEP
75         # -----
76         Te = motor.physics_step(Va, Vb, Vc, Tload, Ia, Ib, Ic)
77
78         # -----
79         # 6. DATA LOGGING
80         # -----
81         history['time'][k] = t
82         history['rpm_ref'][k] = RPMref
83         history['rpm_act'][k] = motor.Wr * 60 / (2*math.pi)
84         history['Iq'][k] = motor.Iq
85         history['Id'][k] = motor.Id
86         history['Te'][k] = Te
87         history['Tload'][k] = Tload
88         history['Vbus'][k] = V_bus
89
90     data = history
91
92     fig, (ax1, ax2, ax3) = plt.subplots(3, 1, figsize=(10, 12), sharex=True)
93
94     ax1.plot(data['time'], data['rpm_ref'], 'r--', label='RPM Ref')
95     ax1.plot(data['time'], data['rpm_act'], 'b-', label='RPM Actual')
96     ax1.set_ylabel('Speed (RPM)')
97     ax1.set_title('FOC Simulation')
98     ax1.legend()
99     ax1.grid(True)
100
101     ax2.plot(data['time'], data['Te'], 'g-', label='Electromagnetic Torque')
102     ax2.plot(data['time'], data['Tload'], 'k--', label='Load Torque')
103     ax2.set_ylabel('Torque (Nm)')
104     ax2.legend()
105     ax2.grid(True)
106
107     ax3.plot(data['time'], data['Iq'], 'c-', label='Iq (A)')
108     ax3.plot(data['time'], data['Id'], 'm-', label='Id (A)')

```

```

109 ax3.set_ylabel('Current (A)')
110 ax3.set_xlabel('Time (s)')
111 ax3.legend()
112 ax3.grid(True)
113
114 plt.tight_layout()
115 plt.show()

```

B. BLACMotor.py

```

1 import math
2 import Transforms
3
4 class BLACMotor:
5     def __init__(self, Ts):
6         self.Ts = Ts
7         self.Npp = 21.0
8         self.Rs = 4.485
9         self.Ld = 0.0548
10        self.Lq = 0.0548
11        self.Lambda_m = 0.201
12        self.Bn = 0.0057
13        self.J = 0.1444
14        self.Tc = 0.3006
15
16        self.Id = 0.0
17        self.Iq = 0.0
18        self.Wr = 0.0
19        self.theta = 0.0
20        self.theta_e = 0.0
21
22    def physics_step(self, Va, Vb, Vc, Tload, Ia, Ib, Ic):
23        We = self.Npp * self.Wr
24        self.theta_e = self.Npp * self.theta
25        self.theta_e = self.theta_e % (2 * math.pi)
26
27        Vd_ref, Vq_ref = Transforms.abc_to_dq(Va, Vb, Vc, self.theta_e)
28
29        Id_meas, Iq_meas = Transforms.abc_to_dq(Ia, Ib, Ic, self.theta_e)
30
31        ed = 0.0
32        eq = We * self.Lambda_m
33
34        dId = (1.0/self.Ld) * (Vd_ref - self.Rs*Id_meas + We*self.Lq*Iq_meas - ed)
35        dIq = (1.0/self.Lq) * (Vq_ref - self.Rs*Iq_meas - We*self.Ld*Id_meas - eq)
36
37        Id_next = Id_meas + self.Ts * dId
38        Iq_next = Iq_meas + self.Ts * dIq
39
40        if abs(We) > 1e-3:
41            Te = 1.5 * self.Npp * (ed * Id_next + eq * Iq_next) / We + \
42                1.5 * self.Npp * (self.Ld - self.Lq) * Id_next * Iq_next
43        else:
44            Te = 1.5 * self.Npp * self.Lambda_m * Iq_next
45
46        Tc_dir = self.Tc if self.Wr > 0 else (-self.Tc if self.Wr < 0 else 0)
47
48        accel = (Te - Tload - (self.Bn * self.Wr) - Tc_dir) / self.J
49        self.Wr += accel * self.Ts
50
51        self.theta += self.Wr * self.Ts
52        self.theta = self.theta % (2*math.pi)
53
54        self.Id = Id_next
55        self.Iq = Iq_next
56
57        return Te

```

C. BLDCMotor.py

```

1 import math
2 import Transforms
3
4 class BLDCMotor:
5     def __init__(self, Ts):
6         self.Ts = Ts
7         self.Npp = 21.0
8         self.Rs = 4.485
9         self.Ld = 0.0548
10        self.Lq = 0.0548
11        self.Lambda_m = 0.201
12        self.Bn = 0.0057
13        self.J = 0.1444
14        self.Tc = 0.3006
15
16        self.Id = 0.0
17        self.Iq = 0.0
18        self.Wr = 0.0
19        self.theta = 0.0
20        self.theta_e = 0.0
21
22    def _trapezoidal_shape(self, theta):
23        t = theta % (2 * math.pi)
24        pi = math.pi
25
26        if t < pi/6:
27            return t * (6/pi) # 0 to 1
28        elif t < 5*pi/6:
29            return 1.0
30        elif t < 7*pi/6:
31            return 1.0 - (t - 5*pi/6) * (6/pi) # 1 to -1
32        elif t < 11*pi/6:

```

```

33        return -1.0
34    else:
35        return -1.0 + (t - 11*pi/6) * (6/pi) # -1 to 0
36
37    def physics_step(self, Va, Vb, Vc, Tload, Ia, Ib, Ic):
38        We = self.Npp * self.Wr
39        self.theta_e = self.Npp * self.theta
40        self.theta_e = self.theta_e % (2 * math.pi)
41
42        E_mag = We * self.Lambda_m
43
44        ea = -E_mag * self._trapezoidal_shape(self.theta_e)
45        eb = -E_mag * self._trapezoidal_shape(self.theta_e - 2*math.pi/3)
46        ec = -E_mag * self._trapezoidal_shape(self.theta_e + 2*math.pi/3)
47
48        ed, eq = Transforms.abc_to_dq(ea, eb, ec, self.theta_e)
49        Vd_ref, Vq_ref = Transforms.abc_to_dq(Va, Vb, Vc, self.theta_e)
50        Id_meas, Iq_meas = Transforms.abc_to_dq(Ia, Ib, Ic, self.theta_e)
51
52        dId = (1.0/self.Ld) * (Vd_ref - self.Rs*Id_meas + We*self.Lq*Iq_meas - ed)
53        dIq = (1.0/self.Lq) * (Vq_ref - self.Rs*Iq_meas - We*self.Ld*Id_meas - eq)
54
55        Id_next = Id_meas + self.Ts * dId
56        Iq_next = Iq_meas + self.Ts * dIq
57
58        if abs(We) > 1e-3:
59            Te = 1.5 * self.Npp * (ed * Id_next + eq * Iq_next) / We + \
60                1.5 * self.Npp * (self.Ld - self.Lq) * Id_next * Iq_next
61        else:
62            Te = 1.5 * self.Npp * self.Lambda_m * Iq_next
63
64        Tc_dir = self.Tc if self.Wr > 0 else (-self.Tc if self.Wr < 0 else 0)
65
66        accel = (Te - Tload - (self.Bn * self.Wr) - Tc_dir) / self.J
67        self.Wr += accel * self.Ts
68
69        self.theta += self.Wr * self.Ts
70        self.theta = self.theta % (2*math.pi)
71
72        self.Id = Id_next
73        self.Iq = Iq_next
74
75        return Te

```

D. FOCController.py

```

1 import math
2 import Transforms
3
4 class FOCController:
5     def __init__(self, Ts, Imax=8.0):
6         self.Ts = Ts
7         self.Imax = Imax
8
9         self.Kps = 1.25
10        self.Kis = 55.0
11        self.KpId = 119.0
12        self.KiId = 4015.0
13        self.KpIq = 119.0
14        self.KiIq = 4015.0
15
16        self.Ui_s = 0.0
17        self.Ui_Id = 0.0
18        self.Ui_Iq = 0.0
19
20    def control_step(self, RPMref, Wr, Ia, Ib, Ic, theta_e, Vbus):
21        Id, Iq = Transforms.abc_to_dq(Ia, Ib, Ic, theta_e)
22
23        # Controlador de velocidade
24        error_speed = (RPMref * 2 * math.pi / 60.0) - Wr
25        Up_s = self.Kps * error_speed
26        Ui_s_next = self.Ui_s + (self.Kis * self.Ts * error_speed)
27        Iq_ref = Up_s + Ui_s_next
28        self.Ui_s = Ui_s_next
29
30        Id_ref = 0.0
31
32        # Controlador de corrente Iq
33        err_Iq = Iq_ref - Iq
34        Up_Iq = self.KpIq * err_Iq
35        Ui_Iq_next = self.Ui_Iq + (self.KiIq * self.Ts * err_Iq)
36        Vq_ref = Up_Iq + Ui_Iq_next
37        self.Ui_Iq = Ui_Iq_next
38
39        # Controlador de corrente Id
40        err_Id = Id_ref - Id
41        Up_Id = self.KpId * err_Id
42        Ui_Id_next = self.Ui_Id + (self.KiId * self.Ts * err_Id)
43        Vd_ref = Up_Id + Ui_Id_next
44        self.Ui_Id = Ui_Id_next
45
46        Va_ref, Vb_ref, Vc_ref = Transforms.dq_to_abc(Vd_ref, Vq_ref, theta_e)
47
48        return Va_ref, Vb_ref, Vc_ref

```

E. Inverter.py

```

1 import math
2
3 class Inverter:
4     def __init__(self):

```

```

5     pass
6
7     def step(self, Va_ref, Vb_ref, Vc_ref, Vbus):
8         limit = Vbus / 2.0
9
10        Va = max(-limit, min(limit, Va_ref))
11        Vb = max(-limit, min(limit, Vb_ref))
12        Vc = max(-limit, min(limit, Vc_ref))
13
14        return Va, Vb, Vc

```

F. Sensors.py

```

1 import math
2 import Transforms
3
4 class Sensors:
5     def __init__(self):
6         pass
7
8     def measure(self, motor, RPMref):
9         theta_e = motor.theta_e
10
11        Ia, Ib, Ic = Transforms.dq_to_abc(motor.Id, motor.Iq, theta_e)
12
13        Wr_meas = motor.Wr
14
15        return Ia, Ib, Ic, theta_e, Wr_meas

```

G. Transforms.py

```

1 import math
2
3 def abc_to_dq(Xa, Xb, Xc, theta):
4     # Clarke transform
5     X_alpha = (2/3) * (Xa - 0.5 * Xb - 0.5 * Xc)
6     X_beta = (2/3) * (math.sqrt(3)/2 * (Xb - Xc))
7
8     # Park transform
9     Xd = X_alpha * math.cos(theta) + X_beta * math.sin(theta)
10    Xq = -X_alpha * math.sin(theta) + X_beta * math.cos(theta)
11
12    # Direct Park Transform (abc -> dq)
13    # Xd = 2/3 * (Xa * math.cos(theta) + Xb * math.cos(theta - 2*math.pi/3) + Xc
14    #       * math.cos(theta + 2*math.pi/3))
15    # Xq = 2/3 * (-Xa * math.sin(theta) - Xb * math.sin(theta - 2*math.pi/3) -
16    #       Xc * math.sin(theta + 2*math.pi/3))
17
18    return Xd, Xq
19
20 def dq_to_abc(Xd, Xq, theta):
21     # Inverse Park transform
22     X_alpha = Xd * math.cos(theta) - Xq * math.sin(theta)
23     X_beta = Xd * math.sin(theta) + Xq * math.cos(theta)
24
25     # Inverse Clarke transform
26     Xa = X_alpha
27     Xb = (-X_alpha + math.sqrt(3) * X_beta) / 2
28     Xc = (-X_alpha - math.sqrt(3) * X_beta) / 2
29
30     # Direct Inverse Park Transform (dq -> abc)
31     # Xa = math.cos(theta) * Xd - math.sin(theta) * Xq
32     # Xb = math.cos(theta - 2*math.pi/3) * Xd - math.sin(theta - 2*math.pi/3) *
33     #       Xq
34     # Xc = math.cos(theta + 2*math.pi/3) * Xd - math.sin(theta + 2*math.pi/3) *
35     #       Xq
36
37     return Xa, Xb, Xc

```