Trabalho de Conclusão de Curso

Controle determinístico para CNC baseado em STM32L475 com DDA de alta frequência

de **Valdir Dias Silva Junior**

orientado por
Prof. Dr. Ícaro Bezerra Queiroz de Araújo

Universidade Federal de Alagoas
Instituto de Computação
Maceió, Alagoas
12 de Novembro de 2024

UNIVERSIDADE FEDERAL DE ALAGOAS
Instituto de Computação

CONTROLE DETERMINÍSTICO PARA CNC BASEADO EM STM32L475 COM DDA DE ALTA FREQUÊNCIA

Trabalho de Conclusão de Curso submetido ao Instituto de Computação da Universidade Federal de Alagoas como requisito parcial para a obtenção do grau de Engenheiro de Computação.

Valdir Dias Silva Junior

Orientador: Prof. Dr. Ícaro Bezerra Queiroz de Araújo

Banca Avaliadora:

Hugo	Profa. Dra., IFAL
Andressa	MSc., SENAI CIMATEC

Maceió, Alagoas
12 de Novembro de 2024

Dados para elaboração da ficha catalográfica devem ser inseridos aqui.
Substitua este texto pelo conteúdo oficial fornecido pela biblioteca ou órgão responsável.

Esta é uma página substituta gerada automaticamente para evitar anexos binários.

Ata de Aprovação

Pagina destinada a ata da banca examinadora

Dedicatória

Conteudo pendente

Agradecimentos

Conteudo pendente

Epígrafe

Conteudo pendente

Resumo

Este trabalho descreve o desenvolvimento de um controlador CNC com foco em determinismo temporal, implementado sobre o microcontrolador STM32L475 e integrado a uma Raspberry Pi. O projeto emprega o gerador de pulsos baseado em *Digital Differential Analyzer* (DDA) a 50 kHz usando o temporizador TIM6, fecha o laço PID de 1 kHz com o TIM7 e utiliza encoders em modo quadratura nos timers TIM2, TIM3 e TIM5 para estimar posição. A comunicação com o host segue um protocolo SPI com DMA circular, complementado por logs via USART1, garantindo observabilidade sem comprometer os prazos de tempo real.

A fundamentação teórica apresenta conceitos de CNC, modelagem de motores de passo, controle PID e algoritmos DDA. A metodologia foi conduzida por um roteiro incremental que validou clock, interrupções críticas, laços de controle e serviços de comunicação. Os resultados mostram a estabilidade do DDA em 50 kHz, jitter inferior a 3 µs no laço PID e o comportamento do pipeline SPI, destacando a necessidade de três ciclos de enquete para o *ack* de comandos de LED. O trabalho conclui evidenciando a robustez do firmware proposto e apresenta sugestões para otimização futura do DMA e dos serviços auxiliares.

***Palavras-chave*: controle CNC; STM32L475; DDA; controlador PID; SPI determinístico.**

Abstract

This dissertation describes the development of a CNC controller focused on temporal determinism, implemented on the STM32L475 microcontroller and integrated with a Raspberry Pi host. The design uses a 50 kHz pulse generator based on a Digital Differential Analyzer (DDA) running on timer TIM6, closes a 1 kHz PID loop with TIM7, and leverages quadrature encoders through timers TIM2, TIM3, and TIM5 for position estimation. Communication with the host is handled by an SPI protocol with circular DMA, complemented by USART1 logging to preserve observability without affecting real-time constraints.

The theoretical background covers CNC systems, stepper motor modeling, PID control, and DDA algorithms. The methodology followed an incremental bring-up roadmap that validated the clock tree, critical interrupts, control loops, and communication services. Results show stable DDA output at 50 kHz, PID loop jitter below 3 µs, and the behavior of the SPI poll pipeline, highlighting the need for three cycles to acknowledge LED commands. The work concludes by emphasizing the robustness of the proposed firmware and suggests future optimizations for DMA handling and auxiliary services.

***Keywords*: CNC control; STM32L475; DDA; PID controller; deterministic SPI.**

Lista de Figuras

Lista de Tabelas

Lista de Símbolos

$f_{\mathbf{sys}}$ Frequência do clock principal do microcontrolador.

$f_{\mathbf{TIM6}}$ Frequência de interrupção do temporizador TIM6 usada pelo DDA.

$f_{\mathbf{loop}}$ Frequência do laço de controle executado no TIM7.

$N_{\mathbf{steps}}$ Número de pulsos STEP gerados para cada eixo.

p Contagem acumulada de pulsos do atuador (STEP).

c Contagem acumulada do encoder (em contagens).

k Índice discreto de amostragem ($k \in \mathbb{N}$), com $t = k\,T_s$.

$f(k)$ Valor da grandeza discreta f na k-ésima amostra.

$v_{\mathrm{cmd}}(k)$ / V_{cmd} Velocidade de comando: $v_{\mathrm{cmd}}(k)$ é o valor instantâneo (pulsos STEP/s) na amostra k; V_{cmd} denota a amplitude do degrau de comando utilizada nas análises (tipicamente $V_{\mathrm{cmd}} := \overline{v}_{\mathrm{cmd}}$).

θ Posição angular estimada a partir do encoder.

$e(t)$ Erro instantâneo entre referência e posição medida.

$u(t)$ Sinal de controle produzido pelo regulador PID.

K_p, K_i, K_d Ganhos proporcional, integral e derivativo do controlador.

J Momento de inércia equivalente do eixo controlado.

T_L Torque de carga refletido no eixo do motor de passo.

T_s Período de amostragem do laço de controle.

$V_{\mathbf{bus}}$ Tensão do barramento que alimenta os drivers TMC5160.

Lista de Abreviaturas

CNC Controle Numérico Computadorizado.

DDA *Digital Differential Analyzer*.

DMA *Direct Memory Access*.

GPIO *General Purpose Input/Output*.

NVIC *Nested Vectored Interrupt Controller*.

PID Proporcional-Integral-Derivativo.

SOF *Start of Frame* (Início do quadro; marcador de início em protocolos de quadro).

EOF *End of Frame* (Fim do quadro; marcador de encerramento em protocolos de quadro).

STEP/DIR/EN Sinais de passo, direção e habilitação dos drivers de motor de passo.

USART *Universal Synchronous/Asynchronous Receiver/Transmitter*.

VCP *Virtual COM Port*.

UFAL Universidade Federal de Alagoas.

Conteúdo

Capítulo 1

Introdução

A popularização de máquinas de Controle Numérico Computadorizado (CNC) depende de controladores capazes de converter instruções digitais em movimentos sincronizados com elevado grau de previsibilidade. No contexto de manufatura de pequeno e médio porte, soluções baseadas em computadores pessoais apresentam custo acessível, mas sofrem com a variabilidade de latência inerente aos sistemas operacionais de propósito geral. Este trabalho investiga uma alternativa embarcada utilizando o microcontrolador STM32L475, capaz de operar a 80 MHz e oferecer temporizadores avançados para geração de pulsos e amostragem de dados [STMicroelectronics, 2013]. Ao combinar a unidade embarcada com uma Raspberry Pi responsável pela interface de alto nível, busca-se garantir escalabilidade e facilidade de integração com pipelines de produção.

A motivação está ligada ao desenvolvimento de um controlador determinista que gere pulsos STEP/DIR/EN em 50 kHz, execute o laço PID a 1 kHz e mantenha comunicação confiável com o host via SPI e USART, entregando registros de telemetria para diagnósticos. A arquitetura proposta precisa coordenar três eixos de motores de passo monitorados por encoders de alta resolução, sincronizar serviços de homing e segurança, e permitir a inserção de novas rotinas sem comprometer as janelas temporais estabelecidas.

1.1 Objetivos

1.1.1 Objetivo Geral

Projetar e documentar um controlador CNC determinístico baseado no STM32L475, integrando geração DDA de pulsos, controle PID em tempo real e protocolo de comunicação SPI com um cliente Raspberry Pi.

1.1.2 Objetivos Específicos

- Configurar o temporizador TIM6 para gerar pulsos em 50 kHz utilizando o método DDA.
- Implementar o loop de controle PID a 1 kHz no TIM7, incorporando leitura incremental dos encoders.
- Estruturar o firmware em camadas modulares (Core, App e Services) que facilitem a validação incremental.
- Validar o pipeline de comunicação SPI escravo com DMA circular e logs assíncronos via USART1.
- Registrar testes que comprovem jitter reduzido e estabilidade nos serviços críticos.

1.2 Organização do texto

O Capítulo 2 apresenta a fundamentação teórica sobre CNC, temporizadores STM32, DDA, controle PID e protocolos de comunicação. O Capítulo 3 descreve a metodologia de *bring-up* incremental utilizada para configurar o firmware e o hardware de suporte. Em seguida, o Capítulo 4 reúne os resultados experimentais, analisando desempenho dos serviços e o comportamento do pipeline SPI. Por fim, o Capítulo 5 sintetiza as contribuições, limitações e linhas futuras de trabalho.

Capítulo 2

Fundamentação Teórica

Este capítulo apresenta os conceitos fundamentais utilizados na implementação do controlador CNC embarcado. São abordados os princípios de sistemas CNC, a configuração de temporizadores no STM32L475, os métodos DDA para geração de pulsos, a modelagem de motores de passo, os controladores PID e os aspectos de comunicação SPI/USART empregados no projeto.

2.1 Sistemas CNC

Máquinas de Controle Numérico Computadorizado interpretam instruções codificadas (por exemplo, G-code) e convertem essas ordens em movimentos coordenados entre múltiplos eixos, garantindo precisão repetível no processo de usinagem ou manufatura [Groover, 2015]. Controladores modernos combinam processamento em tempo real com interfaces de alto nível para planejar trajetórias, compensar erros e monitorar a execução. A adoção de arquiteturas híbridas—com uma unidade embarcada responsável pelo tempo real e um computador auxiliar para supervisão— diminui a suscetibilidade a jitter e a perdas de sincronismo, mantendo a flexibilidade de integração com sistemas de gestão.

2.2 Temporizadores do STM32L475

O microcontrolador STM32L475 disponibiliza temporizadores de uso geral e avançado capazes de operar na faixa de 80 MHz com alta resolução temporal. A configuração de um temporizador baseia-se na relação

$$f_{\mathrm{TIM}} = \frac{f_{\mathrm{bus}}}{(PSC + 1)(ARR + 1)}, \tag{2.1}$$

onde f_{bus} representa a frequência do barramento APB, PSC o prescaler e ARR o registrador de auto-reload. O guia de aplicação oficial descreve como esses parâmetros possibilitam gerar bases de tempo precisas para laços de controle, captura de entradas e geração de pulsos PWM [STMicroelectronics, 2013]. No projeto, o TIM6 é configurado com $PSC = 79$ e $ARR = 19$ para obter interrupções em 50 kHz, enquanto o TIM7 opera com $PSC = 7999$ e $ARR = 9$, produzindo um laço de 1 kHz. Os temporizadores TIM2, TIM3 e TIM5 operam em modo encoder para rastrear incrementalmente a posição dos eixos.

2.3 Digital Differential Analyzer

Algoritmos DDA são integradores digitais que aproximam trajetórias contínuas por meio de incrementos discretos, amplamente utilizados em sistemas de gráficos e em controladores de movimento para motores de passo [Fussell, 2003]. O método acumula um erro fracionário em cada interação e emite um pulso quando a soma ultrapassa um limiar definido, resultando em uma sequência de passos que aproxima a velocidade ou a trajetória desejada. Arquiteturas clássicas de interpolação tratam o DDA como núcleo do gerador de pulsos, responsável por alimentar os laços de posição e velocidade que seguem a referência calculada amostra a amostra [IDC Technologies, 2014, Koren, 1978, Wang and Hu, 2016, Dronacharya Group of Institutions, 2019]. Panoramas comparativos mostram como variantes circular, linear e por superfície mantêm avanço constante mesmo em trajetórias multi-eixo, o que fundamenta a escolha de incrementos acumulativos sincronizados para o firmware do STM32 [Koren, 2010]. No contexto deste trabalho, o DDA implementado no TIM6 gera sinais STEP com resolução de 20 µs e tolera ajustes de velocidade em tempo real sem quebrar a coesão dos múltiplos eixos. A abordagem permite sincronizar movimentos lineares e circulares através da atualização de incrementos acumulados por eixo durante a ISR do temporizador, preservando a planicidade de avanço descrita pelas referências.

2.4 Modelagem de motores de passo

Motores de passo híbridos apresentam dinâmica eletromecânica dominada por indutâncias de fase, resistência de enrolamento e um torque relacionado à diferença angular entre rotor e campo magnético. Modelos clássicos de motores de passo descrevem a relação entre corrente, torque e velocidade angular por meio de equações diferenciais acopladas que podem ser discretizadas para implementação em controladores digitais [Kenjo and Sugawara, 1994]. A precisão do controle depende da estimação do torque de carga T_L, do momento de inércia equivalente J e da compensação de efeitos como

ressonâncias de meia etapa. A utilização de encoders em modo quadratura provê realimentação adicional para compensar perda de passos e acúmulo de erro estático.

2.5 Controlador PID digital

O controlador Proporcional-Integral-Derivativo (PID) continua sendo uma das estratégias mais difundidas para controle de processos, combinando uma ação proporcional que reage ao erro instantâneo, um termo integral que remove erro estacionário e um termo derivativo que prevê tendências de variação [Åström and Hägglund, 1995]. Loops servo em máquinas CNC seguem as posições discretizadas pelo interpolador e corrigem desvios com ganhos sintonizados para cada eixo, podendo incluir observadores ou compensação de distúrbios para manter a precisão em alta velocidade [Wang and Hu, 2016, Koren, 1980, Kung et al., 2005]. Para implementação digital, é comum utilizar a forma incremental

$$u[k] = u[k-1] + K_p(e[k] - e[k-1]) + K_i T_s e[k] + \frac{K_d}{T_s}(e[k] - 2e[k-1] + e[k-2]), \quad (2.2)$$

onde T_s é o período de amostragem. No laço de 1 kHz, o período fixo reduz o esforço computacional e facilita a análise de estabilidade. Estratégias derivadas, como anti-*windup* e filtros de primeira ordem no termo derivativo, são essenciais para lidar com o ruído proveniente dos encoders e das variações do torque de carga. Pesquisas recentes destacam ainda controladores PID acoplados/cross-coupled que tratam o erro de contorno entre eixos como variável adicional, reduzindo desvios em trajetórias complexas [Wang et al., 2021].

2.6 Modelo FOPDT e síntese de ganhos PD

O comportamento dinâmico da malha de velocidade foi aproximado por um modelo de primeira ordem com atraso puro (FOPDT, do inglês *First-Order Plus Dead Time*), cuja função de transferência é dada por

$$G_v(s) = \frac{K\, e^{-Ls}}{\tau s + 1}, \quad (2.3)$$

onde K denota o ganho estático, L o tempo morto e τ a constante de tempo. Para um degrau de amplitude u_0, a resposta temporal é

$$y(t) = \begin{cases} 0, & t < L, \\ K\, u_0 \left(1 - e^{-\frac{t-L}{\tau}}\right), & t \geq L. \end{cases} \quad (2.4)$$

Propriedades úteis incluem $t_{63} = L + \tau$ (63,2% do valor de regime) e $t_{90} \approx L + 2{,}303\,\tau$.

2.6.1 Derivação da resposta ao degrau

Considere o modelo de primeira ordem com atraso puro excitado por um degrau de amplitude U atrasado de L segundos, isto é, $u(t) = U\,H(t-L)$, onde $H(\cdot)$ é a função de Heaviside. No domínio do tempo, o FOPDT satisfaz a equação diferencial linear

$$\tau\,\dot{y}(t) + y(t) = K\,u(t) = K\,U\,H(t-L), \quad y(0) = 0. \tag{2.5}$$

Para $t < L$ tem-se $u(t) = 0$ e, portanto, $y(t) = 0$. Para $t \geq L$ define-se $\xi = t - L$ e resolve-se o problema deslocado

$$\tau\,\frac{d}{d\xi}y(L+\xi) + y(L+\xi) = K\,U, \quad y(L^{-}) = 0. \tag{2.6}$$

A solução geral é soma das soluções homogênea e particular, resultando em

$$y(t) = K\,U\left(1 - e^{-\frac{t-L}{\tau}}\right)H(t-L), \tag{2.7}$$

que coincide com a expressão utilizada nas validações: para $t < L$, $y = 0$; para $t \geq L$, a saída converge exponencialmente para $K\,U$ com constante de tempo τ.

Uma dedução equivalente via transformada de Laplace parte de $G_v(s) = \frac{K\,e^{-Ls}}{\tau s+1}$ e de um degrau de magnitude U, U/s. Pela propriedade de translação temporal, $Y(s) = G_v(s)\,\frac{U}{s} = \frac{K\,U}{s(\tau s+1)}\,e^{-Ls}$, cuja transformada inversa fornece a mesma expressão $y(t) = K\,U\,(1 - e^{-(t-L)/\tau})\,H(t-L)$.

No contexto deste trabalho, a amplitude do degrau U é escolhida como a velocidade de comando em regime (denotada por V_{cmd}), $U := V_{\text{cmd}} := \overline{v}_{\text{cmd}}$, estimada a partir da média em janela final do traço de comando (vide Seção 3.4). Essa escolha assegura que o nível de regime previsto pelo modelo, $K\,U$, coincida com o patamar medido no encoder quando a aproximação FOPDT é adequada.

Com base nesse modelo, adotou-se um regulador proporcional–derivativo (PD) na malha de velocidade, mantendo $K_i = 0$ por já existir ação integrativa no nível de posição. As constantes são sintetizadas pelas regras de Ziegler–Nichols para curva de reação (processos do tipo FOPDT), a saber [Åström and Hägglund, 1995]:

$$K_p = 1{,}2\,\frac{\tau}{K\,L}, \quad T_d = 0{,}5\,L, \quad K_d = K_p\,T_d, \quad K_i = 0. \tag{2.8}$$

Para implementação digital, os ganhos podem ser escalonados para representação inteira utilizando um fator de escala $S > 0$ (*fixed-point*), *e.g.*, $k_{p,i} = \text{round}(S\,K_p)$ e análogos para $k_{i,i}$ e $k_{d,i}$. Tal mapeamento preserva a coerência entre a análise contínua e o controlador discreto executado em tempo real.

2.7 Modelo FOPDT e síntese de ganhos PD

O comportamento dinâmico da malha de velocidade foi aproximado por um modelo de primeira ordem com atraso puro (FOPDT, do inglês *First-Order Plus Dead Time*), cuja função de transferência é dada por

$$G_v(s) = \frac{K\, e^{-Ls}}{\tau s + 1}, \tag{2.9}$$

onde K denota o ganho estático, L o tempo morto e τ a constante de tempo. Para um degrau de amplitude u_0, a resposta temporal é

$$y(t) = \begin{cases} 0, & t < L, \\ K\, u_0 \left(1 - e^{-\frac{t-L}{\tau}}\right), & t \geq L. \end{cases} \tag{2.10}$$

Propriedades úteis incluem $t_{63} = L + \tau$ (63,2% do valor de regime) e $t_{90} \approx L + 2{,}303\,\tau$.

2.7.1 Derivação da resposta ao degrau

Considere o modelo de primeira ordem com atraso puro excitado por um degrau de amplitude U atrasado de L segundos, isto é, $u(t) = U\,H(t-L)$, onde $H(\cdot)$ é a função de Heaviside. No domínio do tempo, o FOPDT satisfaz a equação diferencial linear

$$\tau\,\dot{y}(t) + y(t) = K\,u(t) = K\,U\,H(t-L), \quad y(0) = 0. \tag{2.11}$$

Para $t < L$ tem-se $u(t) = 0$ e, portanto, $y(t) = 0$. Para $t \geq L$ define-se $\xi = t - L$ e resolve-se o problema deslocado

$$\tau\,\frac{d}{d\xi} y(L+\xi) + y(L+\xi) = K\,U, \quad y(L^-) = 0. \tag{2.12}$$

A solução geral é soma das soluções homogênea e particular, resultando em

$$y(t) = K\,U\left(1 - e^{-\frac{t-L}{\tau}}\right) H(t-L), \tag{2.13}$$

que coincide com a expressão utilizada nas validações: para $t < L$, $y = 0$; para $t \geq L$, a saída converge exponencialmente para $K\,U$ com constante de tempo τ.

Uma dedução equivalente via transformada de Laplace parte de $G_v(s) = \frac{K\,e^{-Ls}}{\tau s+1}$ e de um degrau de magnitude U, U/s. Pela propriedade de translação temporal, $Y(s) = G_v(s)\,\frac{U}{s} = \frac{K\,U}{s(\tau s+1)}\,e^{-Ls}$, cuja transformada inversa fornece a mesma expressão $y(t) = K\,U\,(1 - e^{-(t-L)/\tau})\,H(t-L)$.

No contexto deste trabalho, a amplitude do degrau U é escolhida como a velocidade de comando em regime (denotada por V_{cmd}), $U := V_{\text{cmd}} := \overline{v}_{\text{cmd}}$, estimada a partir da

média em janela final do traço de comando (vide Seção 3.4). Essa escolha assegura que o nível de regime previsto pelo modelo, $K\,U$, coincida com o patamar medido no encoder quando a aproximação FOPDT é adequada.

Com base nesse modelo, adotou-se um regulador proporcional–derivativo (PD) na malha de velocidade, mantendo $K_i = 0$ por já existir ação integrativa no nível de posição. As constantes são sintetizadas pelas regras de Ziegler–Nichols para curva de reação (processos do tipo FOPDT), a saber [Åström and Hägglund, 1995]:

$$K_p = 1{,}2\,\frac{\tau}{K\,L}, \quad T_d = 0{,}5\,L, \quad K_d = K_p\,T_d, \quad K_i = 0. \tag{2.14}$$

Para implementação digital, os ganhos podem ser escalonados para representação inteira utilizando um fator de escala $S > 0$ (*fixed-point*), *e.g.*, $k_{p,i} = \mathrm{round}(S\,K_p)$ e análogos para $k_{i,i}$ e $k_{d,i}$. Tal mapeamento preserva a coerência entre a análise contínua e o controlador discreto executado em tempo real.

2.8 Integração PID-DDA

A sincronização entre o DDA e o controlador PID ocorre ao transformar o comando de posição desejada em incrementos de passos por período do TIM6. Métodos de distribuição diferencial garantem que ajustes produzidos pelo PID sejam refletidos sem rupturas na geração de pulsos, mantendo o alinhamento entre eixos [Mori et al., 2005, Wang and Hu, 2016]. Materiais didáticos e relatórios industriais ilustram o fluxo completo: o interpolador entrega referências de posição, o encoder fornece o retorno real e o PID calcula o esforço aplicado ao motor para cancelar o erro, em um ciclo repetido a cada 1 ms [IDC Technologies, 2014, Dronacharya Group of Institutions, 2019]. Implementações modernas combinam esses blocos em FPGAs ou SoCs para reduzir latência e habilitar interpolação multi-eixo síncrona com laços servo dedicados [Kung et al., 2005, Koren, 1980]. No firmware do STM32, o laço de controle alimenta as metas de velocidade e microstepping de cada eixo, enquanto o DDA executa as transições discretas, possibilitando perfis suaves e respeitando os limites de aceleração definidos pelo firmware.

2.9 Sincronização Cruzada de Eixos (CASC)

Em movimentos multi-eixo, a coordenação cinemática exige que cada eixo siga uma fração coerente do avanço global, minimizando erro de sincronismo e de contorno. Chamamos aqui de *CASC* (*Cross-Axis Sync*) o conjunto de estratégias que, sobre a malha de posição/velocidade por eixo, impõem coerência temporal do grupo.

Sejam $s_i(t)$ as posições desejadas e $x_i(t)$ as posições medidas. O *progresso normalizado*

por eixo pode ser escrito como

$$p_i(t) = \frac{x_i(t) - x_i(0)}{\max\{1,\, |s_i(T) - s_i(0)|\}}, \tag{2.15}$$

onde T é a duração nominal do movimento. Uma escolha comum de eixo *mestre* é aquele com maior *restante absoluto* $R_i(t) = |s_i(T) - x_i(t)|$ (ou, inversamente, menor progresso); o objetivo é que os demais eixos acompanhem o mestre, reduzindo o erro de sincronismo

$$e_i^{\text{sync}}(t) = p_m(t) - p_i(t), \quad i \neq m. \tag{2.16}$$

Uma lei simples de *modulação de avanço* aplica um reescalonamento na velocidade de referência de cada eixo não-mestre, do tipo

$$v_i^{\text{ref}}(t) = \sigma\big(e_i^{\text{sync}}(t)\big)\, v_i^{\text{nom}}(t), \quad \sigma(e) = 1 - (1 - \sigma_{\min})\, \min\left\{1,\, \frac{|e|}{e_{\text{th}}}\right\}, \tag{2.17}$$

com $\sigma_{\min} \in (0, 1)$ a fração mínima admissível e $e_{\text{th}} > 0$ um limiar que define quando a penalização satura. Intuitivamente, quanto maior o atraso relativo e_i^{sync}, maior a redução da velocidade "instantânea"para permitir a re-sincronização. Em situações críticas, uma *retenção de sincronismo* (*sync hold*) pode impor $v_i^{\text{ref}} = 0$ para eixos defasados além de um limite de segurança.

Durante a fase final do movimento, define-se uma *janela de acabamento* de largura W (em passos ou fração do deslocamento global), na qual se aplicam regras mais conservadoras: desabilita-se detecção de *stall*, eleva-se $\sigma_{\min}$ e relaxam-se as trocas de mestre para evitar oscilações. Tais práticas são consistentes com técnicas de controle cruzado de eixos *cross-coupled* que utilizam o erro de contorno/ sincronismo para melhorar a precisão em trajetórias 2D/3D [Wang et al., 2021].

Note-se que o CASC não substitui as malhas PID por eixo: ele opera como supervisor de alto nível, gerando referências coerentes e respeitando limitações de aceleração e saturação. A estabilidade do conjunto depende da escolha de limiares (e_{th}, $\sigma_{\min}$) e da política de seleção do mestre (maior restante, menor progresso, eixo carregado etc.), que devem ser ajustados de modo a preservar margens de fase e evitar comandos abruptos.

2.10 Comunicação SPI e USART

A comunicação com a Raspberry Pi utiliza o periférico SPI1 em modo escravo com DMA circular. Esse desenho reduz a carga da CPU e garante que as transferências de 42 bytes sejam executadas dentro da janela entre interrupções do TIM6 e TIM7. A USART1, configurada como *Virtual COM Port*, é utilizada para depuração e registro de eventos críticos, com uma fila não bloqueante que impede o impacto sobre o laço de controle. A

coordenação entre SPI e USART é fundamental para evitar inversões de prioridade que poderiam comprometer o determinismo do sistema [STMicroelectronics, 2018b]. A análise do pipeline de polls evidencia como o firmware pausa e reinicia o DMA para garantir que cada resposta seja transmitida somente após processamento completo.

SPI Raspberry Pi ↔ STM32: protocolo do enlace

O enlace SPI escravo (STM32 como escravo) utiliza quadros de tamanho fixo com **42 bytes**, transferidos em modo *full-duplex*. O mestre (Raspberry Pi) envia um *poll* com a requisição e recebe, no mesmo ciclo ou nos ciclos subsequentes, a resposta preparada pelo firmware (via fila de respostas e DMA). O quadro adota marcadores de início/fim para robustez:

Tabela 2.1: Layout do quadro SPI (tamanho fixo: 42 bytes).

Offset	Tamanho	Campo	Descrição
0	1	SOF	Byte fixo de início (0xAB) para sincronização do quadro.
1	1	CMD	Identificador do serviço/comando (ex.: 0x10=LED, 0x20=MOVE).
2	1	FLAGS	Bits de controle/opções do comando (reservado = 0x00).
3	1	LEN	Comprimento válido do PAYLOAD (0 a 34 bytes).
4	34	PAYLOAD	Dados da requisição; preencher com 0x00 quando não usado.
38	2	CRC16	Verificação de integridade (LSB,MSB) calculada sobre bytes 0..37.
40	1	RSV	Reservado (0x00) para alinhamento/expansão futura.
41	1	EOF	Marcador de fim: `0x54`.

Quando não há resposta pronta, o escravo devolve um quadro com todos os 21 bytes preenchido com `0xA5`. Quando a resposta de um serviço está completa, o escravo publica o quadro final com `EOF=0x54` e CRC válido, conforme observado no cliente `cnc_spi_client.py`.

Serviço LED (exemplo de comando simples)

O serviço LED liga/desliga ou alterna o estado. O mestre envia uma requisição; a resposta confirma o estado final.

Serviço Movimento (exemplo de comando composto)

O serviço de movimento recebe metas por eixo e parâmetros cinemáticos simplificados. Campos não usados devem ser zero.

Tabela 2.2: Requisição LED (`CMD=0x10`).

Offset	Tamanho	Campo	Valor/Descrição
0	1	SOF	`0xAB`
1	1	CMD	`0x10` (LED)
2	1	FLAGS	`0x00` (padrão)
3	1	LEN	`0x02`
4	1	LED_ID	`0x00` (LED on-board)
5	1	MODE	`0x00=OFF`, `0x01=ON`, `0x02=TOGGLE`
6..37	32	RSV	`0x00`
38	2	CRC16	CRC-16 sobre bytes 0..37
40	1	RSV	`0x00`
41	1	EOF	`0x54`

Tabela 2.3: Resposta LED.

Offset	Tamanho	Campo	Valor/Descrição
0	1	SOF	`0xAB`
1	1	CMD	`0x10` (eco)
2	1	FLAGS	`0x00`
3	1	LEN	`0x02`
4	1	STATUS	`0x00=OK`, `>0`=erro
5	1	LED_STATE	`0x00=OFF`, `0x01=ON`
6..37	32	RSV	`0x00`
38	2	CRC16	CRC-16 sobre bytes 0..37
40	1	RSV	`0x00`
41	1	EOF	`0x54`

Tabela 2.4: Requisição Movimento (`CMD=0x20`).

Offset	Tamanho	Campo	Valor/Descrição
0	1	SOF	`0xAB`
1	1	CMD	`0x20` (MOVE)
2	1	FLAGS	`0x00`
3	1	LEN	Deve refletir o total de bytes uteis do payload (exemplo:`0x15`).
4	1	AXIS_MASK	Máscara de eixos: Bit0=X, Bit1=Y, Bit2=Z. Permite ativar eixos individualmente.
5..8	4	X_STEPS	`int32` (passos relativos) Positivo/negativo define o sentido.
9..12	4	Y_STEPS	`int32` (passos relativos) Positivo/negativo define o sentido.
13..16	4	Z_STEPS	`int32` (passos relativos) Positivo/negativo define o sentido.
17..20	4	FEED	Velocidade máxima alvo em passos/s (uint32). Limitador para o perfil de movimento.
21	1	JERK	Parâmetro opcional do perfil (suavização). Se não utilizado, enviar 0x00.
22..37	16	RSV	`0x00`
38	2	CRC16	CRC-16 sobre bytes 0..37
40	1	RSV	`0x00`
41	1	EOF	`0x54`

Tabela 2.5: Resposta Movimento.

Offset	Tamanho	Campo	Valor/Descrição
0	1	SOF	`0xAB`
1	1	CMD	`0x20` (eco)
2	1	FLAGS	`0x00`
3	1	LEN	`0x03`
4	1	STATUS	`0x00=aceito`, `0x01=ocupado`, >0=erro
5	1	QUEUE_DEPTH	Itens pendentes na fila de movimentos
6	1	RESERVED	`0x00`
7..37	31	RSV	`0x00`
38	2	CRC16	CRC-16 sobre bytes 0..37
40	1	RSV	`0x00`
41	1	EOF	`0x54`

Notas sobre os campos da resposta MOVE:

- **STATUS**: 0x00=aceito (comando enfileirado/executando); 0x01=ocupado (tentar novamente); valores >0 indicam erro (cdigo especfico do firmware).
- **QUEUE_DEPTH**: tamanho atual da fila de movimentos, til para controle de fluxo do mestre.
- **CRC16**: valida a integridade do quadro; CRC invlido deve ser tratado como erro de comunicao.

SPI Raspberry Pi ↔ TMC5160: configuração por registradores

O TMC5160 expõe um barramento SPI para configuração e diagnóstico por meio de transações de **40 bits** (5 bytes por quadro), compostas por um byte de endereço e quatro bytes de dados. O bit mais significativo do byte de endereço indica leitura/escrita (**1**=leitura, **0**=escrita) e os 7 bits menos significativos selecionam o registrador. O protocolo apresenta **latência de uma transação**: os dados retornados em `SDO` referem-se ao comando emitido no quadro anterior. Recomenda-se emitir uma leitura "de preparo" antes de coletar o valor válido do registrador desejado (ver [TRINAMIC Motion Control GmbH & Co. KG (Analog Devices),]). Em implementações usuais o SPI opera em modo 3 (CPOL=1, CPHA=1).

Tabela 2.6: Quadro de escrita TMC5160 (5 bytes, big-endian).

Offset	Tamanho	Campo	Descrição
0	1	ADDR[6:0], R/W=0	Endereço do registrador (7 bits), bit 7=0 (escrita).
1..4	4	DATA[31:0]	Palavra de 32 bits, ordem de bytes: MSB → LSB.

Tabela 2.7: Quadro de leitura TMC5160 (5 bytes, big-endian; resposta no ciclo seguinte).

Offset	Tamanho	Campo	Descrição
0	1	ADDR[6:0], R/W=1	Endereço do registrador (7 bits), bit 7=1 (leitura).
1..4	4	DUMMY	`0x00`; os 32 bits lidos pertencem ao quadro anterior.

Exemplos de registros comuns na bring-up e operação são apresentados junto ao TMC5160 (Seção 2.11.1).

Os exemplos acima alinham-se ao fluxo de *poll* descrito na Seção 2.7.

2.11 Driver Trinamic TMC5160 e Encoder TMCS-28

Os drivers de motor de passo da Trinamic são amplamente utilizados por oferecerem modos de comutação silenciosos, detecção de carga *sensorless* e parametrização fina via registradores. Em complemento, encoders ópticos incrementais como o TMCS-28 permitem medição sem contato da posição/angulo do eixo com disco óptico e sensor, habilitando calibrações de zero, telemetria e malhas fechadas. Esta seção resume os recursos relevantes do TMC5160 e do TMCS-28 e como eles se relacionam à arquitetura do firmware.

2.11.1 TMC5160

O TMC5160 é um driver de alto desempenho voltado a correntes elevadas, com interface `SPI` e controle por pinos `STEP/DIR/EN`. Entre os principais recursos:

- **Microstepping e microPlyer**: geração de até 256 micropassos e interpolação *microPlyer* a partir de entradas com baixa resolução, suavizando o movimento mesmo com taxas de pulso moderadas. - **stealthChop2**: modo de chaveamento focado em baixo ruído acústico; configurado principalmente em `PWMCONF`. - **spreadCycle**: chopper clássico de corrente para maior fidelidade em torque em altas velocidades; parametrização em `CHOPCONF`. - **StallGuard2**: medição de carga sem sensor (*sensorless*) que permite detectar perda de passo/contato; limiar em `SGTHRS` e leitura do ganho em `SG_RESULT`. - **coolStep**: regulação dinâmica de corrente baseada na carga para reduzir perdas sem comprometer torque. - **dcStep**: avanço dependente de carga para evitar perda de passo em condições adversas. - **Proteções e diagnóstico**: `DRV_STATUS` expõe flags de sobrecorrente, subtensão e temperatura.

Na integração com o firmware, o TMC5160 é inicializado via `SPI` ajustando `IHOLD_IRUN` (correntes de hold/run), `TPOWERDOWN`, `CHOPCONF` e `PWMCONF`. A comutação de perfis (*stealthChop2* $\leftrightarrow$ *spreadCycle*) pode ser feita em tempo de execução para equilibrar ruído e robustez. Para homing *sensorless*, calibra-se `SGTHRS` e a janela `TCOOLTHRS` de maneira a obter detecção repetível sem falsos positivos.

Tabela 2.8: Exemplos de acessos SPI ao TMC5160.

Registro	Operação/Observação
`IHOLD_IRUN`	Escrita dos campos `IHOLD`, `IRUN` e `IHOLDDELAY` para corrente de hold/run e rampa.
`CHOPCONF`	Seleção de *spreadCycle* ou parâmetros de histerese do chopper.
`PWMCONF`	Parâmetros do *stealthChop2* (`PWM_AMPL`, `PWM_GRAD`).
`SGTHRS`	Limiar do *StallGuard2* para homing sensorless.
`DRV_STATUS`	Leitura de flags de falha e `SG_RESULT`; lembrar da latência de um quadro.

2.11.2 TMCS-28 (Trinamic/Analog Devices)

O TMCS-28 è um encoder óptico incremental de baixo custo e dimensão reduzida para motores de passo e PMSM/BLDC. Utiliza roda código óptica e fornece saídas em quadratura **A** e **B** mais **N** (*index*). Principais pontos ([TRINAMIC Motion Control GmbH & Co. KG (Analog Devices), 2022]):

- **Resolucções**: até 625 lpr (*lines per rotation*) $\Rightarrow$ 40 000 cpr; opção de 64 lpr $\Rightarrow$ 4 096 cpr. - **Sinal**: ABN em nível TTL, *rise/fall* típicos 10 ns, $V_{\mathrm{OH}} \geq 2.4\,\mathrm{V}$, $V_{\mathrm{IL}} \leq 0.4\,\mathrm{V}$, I_{out} max $\approx 20\,\mathrm{mA}$. - **Alimentação**: 4.5–5.5 V (típico 5 V), consumo $\approx 110\,\mathrm{mA}$. - **Faixa de frequência**: até $\sim 1.5\,\mathrm{MHz}$ de comutação. - **Mecânica**: diâmetro do furo 5 mm ou 6.35 mm; carga axial 50 N, radial 80 N; rotação máxima 6000 rpm; módulo $28\,\mathrm{mm} \times 28\,\mathrm{mm} \times 18\,\mathrm{mm}$ (aprox.).

Integração com o firmware: conectar A/B aos temporizadores do STM32 em modo *encoder* (por exemplo, TIM2/TIM3/TIM5) e o *index* N a uma entrada de reset/captura para referência de zero. Converter contagens em ângulo via $\theta = 360\,^\circ \cdot \text{count/CPR}$. Neste TCC foi utilizada a variante **TMCS-28-10k** (código TMCS-28-x-10000-AT-01), com **625 lpr** e **40 000 cpr**; portanto, considerar CPR = 40 000. Como as saídas são TTL a 5 V, prever adaptação de nível para GPIOs de 3.3 V do STM32 (divisores/*level shifter*).

2.11.3 Implicções de projeto

No contexto deste trabalho, os recursos de microstepping e os modos *stealthChop2*/*spreadCycle* são os mais relevantes para compatibilizar a taxa de passos do DDA (50 kHz) com suavidade e torque. Quando aplicável, o uso de *StallGuard2* permite procedimentos de homing sem sensores, desde que a calibração de `SGTHRS`/`TCOOLTHRS` seja validada em bancada. A parametrização via `SPI`/UART se integra naturalmente à camada de *Services*, mantendo os ajustes desacoplados do laço de tempo real.

2.12 Motores de Passo NEMA 23, Passo de 0,9° e Seleção da Corrente I_{RMS}

2.12.1 NEMA 23: geometria e implicações mecânicas

NEMA 23 define **dimensões de flange** do motor (aprox. 57 mm × 57 mm) e furação de montagem; **não** define torque, corrente ou passo elétrico. Assim, motores NEMA 23 podem variar em comprimento do corpo, inércia do rotor, resistência/indutância por fase e torque de retenção. No projeto, os três eixos utilizam motores NEMA 23 por combinarem: (i) facilidade de fixação em estruturas CNC, (ii) bom compromisso entre torque e tamanho, (iii) ampla disponibilidade de *drivers* compatíveis (p.ex., TMC5160).

2.12.2 Passo elétrico: 1,8° vs 0,9°

O **ângulo de passo elétrico** é a rotação do eixo a cada *passo inteiro* sem microstepping. Os valores típicos são:

$$\theta_{\text{passo}} \in \{1{,}8^\circ,\ 0{,}9^\circ\} \quad \Rightarrow \quad N_{\text{passos/rev}} = \frac{360^\circ}{\theta_{\text{passo}}}$$

Assim, motores de **1,8°** têm $N = 200$ passos/rev; motores de **0,9°** têm $N = 400$ passos/rev.

Efeitos práticos do passo de 0,9°.

- **Mais resolução mecânica** por volta (400 vs 200 passos), favorecendo posicionamento fino e menor *ripple* posicional a baixas velocidades.
- **Maior exigência de taxa de pulsos** para a mesma velocidade linear, pois dobra-se o número de passos por volta. Para uma razão de microstepping M, a relação é:

  $$\text{RPS} = \frac{f_{\text{STEP}}}{N \cdot M} \quad \Rightarrow \quad f_{\text{STEP}} = \text{RPS} \cdot N \cdot M$$

  Ex.: a mesma RPS em 0,9° requer o dobro de f_{STEP} em relação a 1,8°.
- **Desempenho em alta rotação**: por demandar maior frequência elétrica, a queda de torque em alta velocidade pode ser mais perceptível em 0,9° se a eletrônica não compensar a indutância.

2.12.3 Microstepping, suavidade e taxa de passos

O **microstepping** (p.ex., $M = 16$, $M = 256$) interpola correntes senoidais nas fases, reduz ruídos e vibrações e melhora a suavidade do movimento. Contudo:

- A **resolução de comando** é limitada por f_{STEP} disponível do controle (DDA/gerador de trajetórias).
- A **capacidade de torque incremental** por micropasso é menor que o torque de passo inteiro; por isso, em carga dinâmica, é comum trabalhar com correntes moderadas e acelerações respeitando a curva torque–velocidade.

2.12.4 Parâmetros elétricos e modelo de primeira ordem

Motores de passo bipolares são especificados por I_{rated} (corrente por fase), resistência R e indutância L. A dinâmica de corrente (sem FEM de rotação) obedece:

$$\frac{di}{dt} \approx \frac{V_{\mathrm{bus}} - v_{\mathrm{chopper}}}{L} \quad \text{com} \quad \tau = \frac{L}{R}$$

Um *driver* de modo corrente (p.ex., TMC5160) aplica **chopping** para manter I_{RMS} alvo nas fases, usando tensão de barramento elevada (24 V a 48 V) para acelerar a resposta de corrente contra a indutância. O **aquecimento** escala como $P \propto I_{RMS}^2 R$; por isso, trabalhar com frações moderadas de I_{rated} reduz significantemente a dissipação, sem inviabilizar o torque exigido.

2.12.5 Torques relevantes

- **Torque de retenção** T_{hold}: torque estático máximo com corrente nominal e eixo travado.
- **Torque de detente** T_{det}: ondulação mecânica intrínseca (sem corrente); corrente precisa superá-lo para iniciar movimento confiável.
- **Torque dinâmico**: decresce com velocidade elétrica pela limitação de di/dt e FEM de rotação; tensão maior no barramento ajuda a preservar torque em regime.

2.12.6 Resumo para os motores usados

- **JK57HM76-2804** (NEMA 23, $I_{rated} \approx 2{,}8\,\mathrm{A}$, $T_{hold} \approx 1{,}8\,\mathrm{N \cdot m}$): alto torque estático; adequado a eixos que demandam maior força de corte; pode ser 0,9° ou 1,8° conforme variante.
- **23HM8430 0,9°** (NEMA 23, $I_{rated} \approx 3{,}0\,\mathrm{A}$, $T_{hold} \approx 1{,}5\,\mathrm{N \cdot m}$): passo fino nativo (400 passos/rev) favorece resolução e suavidade a baixas velocidades, exigindo maior f_{STEP} para a mesma rotação.

2.12.7 Fundamentação teórica para a seleção de I_{RMS}

Motores de passo operam com torque aproximadamente proporcional à corrente elétrica por fase, de forma que a relação entre *torque de retenção* (T_{hold}) e *corrente nominal* (I_{rated}) pode ser usada como boa aproximação da constante de torque (k_T). Assim, variações em I_{RMS} impactam diretamente na capacidade de aceleração, velocidade máxima utilizável e estabilidade mecânica da máquina CNC.

$$k_T \approx \frac{T_{hold}}{I_{rated}} \qquad [\mathrm{N \cdot m/A}]$$

Entretanto, a operação contínua em valores próximos ao limite nominal aumenta significativamente o aquecimento tanto do estator quanto do driver. Neste trabalho, considerando o uso do driver Trinamic TMC5160 a 48 V, optou-se por definir um **perfil de corrente reduzida** que garantisse torque suficiente para movimentação segura, mas mantendo temperaturas moderadas sem necessidade de refrigeração excessiva.

Para garantir movimento sem carga, é necessário superar o *torque de detente* do motor (T_{det}), valor mecânico inerente à geometria do rotor. Assim, pode-se estimar a **corrente mínima prática** como:

$$I_{\min} \approx \alpha \cdot \frac{T_{det}}{k_T}$$

onde α é um fator de segurança entre 1,5 e 2,5, para compensar atrito e irregularidades dinâmicas.

Os dois modelos selecionados (um por eixo), ambos NEMA 23, são:

- JK57HM76-2804: 2,8 A/fase, 0,9° ou 1,8°, $T_{hold} \approx 1{,}8\,\mathrm{N \cdot m}$.
- 23HM8430: 3,0 A/fase, 0,9°, $T_{hold} \approx 1{,}5\,\mathrm{N \cdot m}$.

Assumindo $T_{det} \approx 0{,}06\,\mathrm{N \cdot m}$ (valor típico para NEMA 23), obtiveram-se os parâmetros da Tabela 2.9.

Tabela 2.9: Estimativa da corrente mínima prática (I_{RMS}).

Motor	I_{rated} [A]	T_{hold} [N·m]	k_T [N·m/A]	I_{det} [A]	$I_{\min}$ [A]
JK57HM76-2804	2,8	1,8	0,643	0,093	0,14–0,23
23HM8430 0,9°	3,0	1,5	0,50	0,12	0,18–0,30

Do ponto de vista térmico e de robustez, adotou-se no firmware:

- JK57HM76-2804: $\mathbf{I_{RUN} \approx 0{,}28\,A}$ e $\mathbf{I_{HOLD} \approx 0{,}12\,A}$.
- 23HM8430: $\mathbf{I_{RUN} \approx 0{,}30\,A}$ e $\mathbf{I_{HOLD}} \approx (\mathbf{0{,}12}$–$\mathbf{0{,}15})\,\mathbf{A}$.

Esses valores representam apenas **8–12% da corrente nominal**, o que reduz expressivamente a potência dissipada ($P \propto I_{RMS}^2\,R$), sem prejuízo significativo sobre o torque necessário para os movimentos previstos no CNC desenvolvido. Caso o processo exija maior aceleração ou usinagem mais agressiva, o ajuste de I_{RUN} pode ser realizado dinamicamente pelo firmware via registro `IHOLD_IRUN` do TMC5160.

Assim, a fundamentação e a calibração experimental convergiram para uma configuração segura, eficiente e com excelente relação entre torque ofertado e controle térmico do sistema eletromecânico.

2.13 Registros de Configuração Utilizados

Esta seção sumariza os principais registros configurados no STM32L475 e no driver TMC5160 conforme empregados neste trabalho. O formato segue o exemplo de tabelas de configuração com campos e finalidade.

2.13.1 STM32L475

Tabela 2.10: Registros STM32 L4 utilizados neste trabalho (ver [STMicroelectronics, 2013, STMicroelectronics, 2018a]).

Perif.	Registro	Campo/Valor	Finalidade
TIM6	PSC	79	Prescaler para 50 kHz (DDA).
TIM6	ARR	19	Período de 20 µs.
TIM7	PSC	7999	Prescaler para 1 kHz (PID).
TIM7	ARR	9	Período de 1 ms.
TIM2/3/5	SMCR	SMS=0b011	Modo encoder (contagem em TI1 e TI2).
TIM2/3/5	CCMR1	CC1S=01; CC2S=01	Entradas mapeadas para TI1/TI2.
TIM2/3/5	CCER	CC1P=0; CC2P=0	Polaridade não invertida (conforme ligação).
SPI1	CR1	MSTR=0; CPOL=0; CPHA=0	Modo escravo, fase/polaridade padrão.
SPI1	CR2	RXDMAEN=1; TXDMAEN=1; DS=8	SPI com DMA e palavra de 8 bits.
DMAx	CCR(RX/TX)	MINC=1; CIRC=1; DIR	DMA circular para SPI1 RX/TX.
DMAx	CNDTR/CPAR/CMAR	–	Tamanho e endereços de perif. e memória.
USART1	BRR	115200	Baud rate para logs/telemetria.
EXTI	IMR/RTSR/FTSR	Linhas de E-STOP/limites	Interrupções de segurança.
NVIC	PRIO	–	Priorização: segurança, TIM6, SPI/DMA, TIM7, USART.

2.13.2 TMC5160

Tabela 2.11: Registros TMC5160 utilizados neste trabalho (ver [TRINAMIC Motion Control GmbH & Co. KG (Analog Devices),]).

Registro	Campo/Valor	Finalidade
`IHOLD_IRUN`	IHOLD, IRUN, IHOLDDELAY	Correntes de hold/run e rampa de corrente.
`TPOWERDOWN`	Tempo (µs)	Tempo para redução de corrente em inatividade.
`CHOPCONF`	spreadCycle/ hysteresis	Chopper de corrente e modo de comutação.
`PWMCONF`	stealthChop2 (PWM_AMPL/GRAD)	Chaveamento silencioso e parâmetros PWM.
`SGTHRS`	Limiar	Limiar do StallGuard2 para homing *sensorless*.
`TCOOLTHRS`	Velocidade limiar	Janela de atuação para StallGuard/coolStep.
`TPWMTHRS`	Velocidade limiar	Limite de comutação stealthChop2 $\leftrightarrow$ spreadCycle.
`DRV_STATUS`	Flags	Diagnóstico: sobrecorrente, subtensão, temperatura, SG_RESULT.

Observação: valores exatos de bits podem variar conforme a placa/variante e roteiro de testes; recomenda-se validar com os manuais do STM32 L4 ([STMicroelectronics, 2018a]) e o datasheet do TMC5160 ([TRINAMIC Motion Control GmbH & Co. KG (Analog Devices),]).

Capítulo 3

Metodologia

A metodologia adotada para o desenvolvimento do controlador CNC seguiu um roteiro incremental que prioriza a validação dos blocos críticos antes de incorporar funcionalidades auxiliares. Cada etapa foi documentada, testada e revisada de forma a garantir que os requisitos de determinismo fossem mantidos durante todo o processo.

3.1 Roteiro incremental de bring-up

O ponto de partida consistiu na configuração do clock principal para 80 MHz e na definição das prioridades do NVIC de acordo com o orçamento temporal: interrupções externas de segurança, TIM6, SPI1/DMA, TIM7 e USART1. Em seguida, foram ativadas as entradas de parada de emergência (E-STOP) e sensores de proximidade, assegurando que flags de segurança pudessem interromper o fluxo de comandos. As etapas posteriores focaram na calibração dos temporizadores: o TIM6 foi dimensionado com $PSC = 79$ e $ARR = 19$, enquanto o TIM7 recebeu $PSC = 7999$ e $ARR = 9$. Os temporizadores TIM2, TIM3 e TIM5 foram configurados em modo quadratura para leitura de encoders.

A configuração do SPI1 em modo escravo com DMA circular foi realizada após os temporizadores, evitando contenda na memória compartilhada. Por fim, a USART1 foi ajustada para 115 200 bps e integrada a um serviço de log com fila não bloqueante. Cada etapa do roteiro foi acompanhada por testes de bancada: medições de frequência com osciloscópio, leitura de contadores de encoder e injeção de quadros SPI utilizando o cliente Python.

3.2 Arquitetura de software

O firmware foi estruturado em três camadas principais. A camada *Core* engloba os artefatos gerados pelo STM32CubeMX, incluindo inicialização de periféricos, descrições de pinos e funções HAL. Sobre ela, a camada *App* implementa o laço principal (`app_poll`), o agendador de serviços e as rotinas de inicialização específicas do projeto. A camada *Services* agrupa módulos especializados, como o gerador de passos, o controlador PID, o serviço de homing e o roteador de mensagens SPI.

A fila de recepção SPI é mantida em memória circular, preenchida pelas rotinas de interrupção e consumida por `app_poll`. Respostas são enfileiradas em `g_app_responses` e promovidas para o buffer de DMA quando disponíveis. A integração com os drivers TMC5160 ocorre por meio de uma API dedicada que abstrai comandos STEP/DIR/EN e monitora condições de falha.

3.3 Procedimentos de teste

Os testes de validação foram conduzidos em três frentes. Primeiro, medições com osciloscópio e analisador lógico verificaram a frequência dos pulsos STEP e o jitter do TIM6, confirmando a estabilidade em 50 kHz. Em seguida, foram realizadas varreduras de ganho nos controladores PID para avaliar margem de fase e resposta a degraus, utilizando logs exportados via USART1. Por fim, o pipeline SPI foi monitorado com o cliente `cnc_spi_client.py`, observando a necessidade de até três ciclos de enquete para respostas completas e avaliando o impacto de diferentes valores de `APP_SPI_RESTART_DEFER_MAX`. Os dados coletados subsidiam as análises apresentadas no Capítulo 4.

3.4 Identificação experimental do modelo FOPDT

Esta seção descreve o procedimento de identificação adotado para estimativa de K, L e τ a partir de respostas a degrau de velocidade obtidas em bancada. O método opera inteiramente no domínio do tempo e é robusto a ruídos moderados.

3.4.1 Pré-processamento e séries derivadas

Considere amostras ordenadas por tempo contendo os campos $(t,\, p,\, c)$, em que p representa a contagem acumulada de pulsos do atuador (STEP) e c a contagem acumulada do encoder (contagens). A partir de pares ou de uma janela deslizante de comprimento N, definem-se

$$\Delta t_k = t_k - t_{k-N+1}, \qquad \Delta p_k = p_k - p_{k-N+1}, \qquad \Delta c_k = c_k - c_{k-N+1}, \tag{3.1}$$

$$v_{\mathrm{cmd}}(k) = \frac{\Delta p_k}{\Delta t_k}, \qquad v_{\mathrm{enc}}(k) = \frac{\Delta c_k}{\Delta t_k}. \tag{3.2}$$

Para reduzir o efeito de ruído, utiliza-se tipicamente $N \in [5, 10]$. Pares com $\Delta t_k \leq 0$ são descartados. O valor de regime (*steady state*) de cada série é a média dos últimos $\rho \in (0, 1)$ da janela temporal disponível (*e.g.*, $\rho = 0{,}2$), denotados por $\overline{v}_{\mathrm{cmd}}$ e $\overline{v}_{\mathrm{enc}}$.

Nesta notação, $f(k)$ indica o valor da grandeza discreta f na k-ésima amostra, usualmente associado ao instante $t = k\,T_s$. Em particular, $v_{\mathrm{cmd}}(k)$ representa a velocidade de comando (pulsos/s) derivada dos pulsos emitidos, enquanto V_{cmd} será utilizado para denotar a amplitude do degrau de comando nas simulações e validações (definido como $V_{\mathrm{cmd}} := \overline{v}_{\mathrm{cmd}}$).

3.4.2 Extração de marcos temporais

Define-se o *início efetivo* do degrau nos sinais de comando e de medição pelos primeiros cruzamentos de 5% de seus respectivos regimes:

$$t^{5\%}_{\mathrm{cmd}} = \inf\{t : v_{\mathrm{cmd}}(t) \geq 0{,}05\,\overline{v}_{\mathrm{cmd}}\}, \quad t^{5\%}_{\mathrm{enc}} = \inf\{t : v_{\mathrm{enc}}(t) \geq 0{,}05\,\overline{v}_{\mathrm{enc}}\}. \tag{3.3}$$

Analogamente, obtém-se o instante t_{63} como o primeiro cruzamento de 63,2% no sinal medido:

$$t_{63} = \inf\{t : v_{\mathrm{enc}}(t) \geq 0{,}632\,\overline{v}_{\mathrm{enc}}\}. \tag{3.4}$$

3.4.3 Estimativas K, L e τ

O ganho estático é dado por $K = \overline{v}_{\mathrm{enc}}/\overline{v}_{\mathrm{cmd}}$. O tempo morto é $L = \max\{0,\, t^{5\%}_{\mathrm{enc}} - t^{5\%}_{\mathrm{cmd}}\}$. A constante de tempo decorre de $\tau = t_{63} - L$. Quando limitações de amostragem impõem baixa resolução temporal,

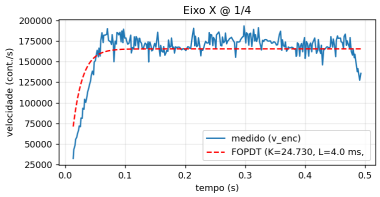
adota-se o piso $\Delta t_{\min}$ observado na série crua, substituindo L e/ou τ por $\Delta t_{\min}$ quando as estimativas resultarem nulas ou negativas.

3.4.4 Síntese PD e validação

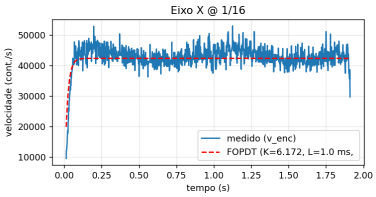
Com as estimativas (K, L, τ), computam-se os ganhos K_p e K_d por meio das regras de Ziegler–Nichols para curva de reação (com $K_i = 0$ na malha de velocidade). A validação consiste em aplicar as constantes ao modelo FOPDT discretizado ($T_s = 1\,\text{ms}$) e comparar a resposta simulada à resposta medida, verificando sobre-elevação, tempo de subida e ausência de saturações relevantes.

3.4.5 Validação gráfica do modelo FOPDT

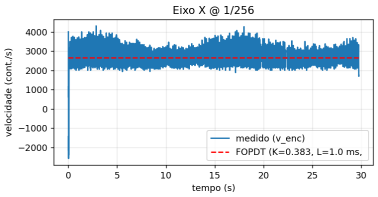
Para documentar a aderência do modelo FOPDT aos dados, foram gerados gráficos de sobreposição entre a velocidade medida (encoder) e a resposta teórica do FOPDT a um degrau. A resposta empregada decorre da solução analítica do sistema de 1ª ordem com atraso sob degrau atrasado (ver Seção 2.7.1): $y(t) = K\,U\,[1 - e^{-(t-L)/\tau}]$ para $t \geq L$ e 0 caso contrário. A amplitude U é tomada como a velocidade de comando em regime, $U := \overline{v}_{\text{cmd}}$, garantindo que o patamar previsto $(K\,U)$ coincida com o observado em regime quando a aproximação é válida. As figuras a seguir apresentam, para cada eixo, os três microsteppings avaliados.



(a) Eixo X @ 1/4

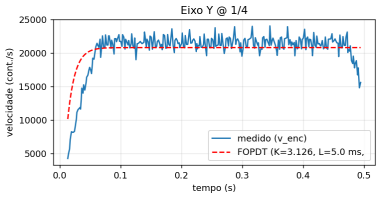


(b) Eixo X @ 1/16

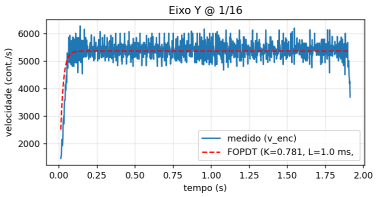


(c) Eixo X @ 1/256

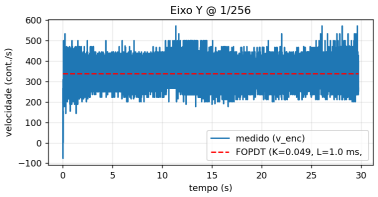
Figura 3.1: Sobreposição entre velocidade medida e FOPDT (eixo X).



(a) Eixo Y @ 1/4

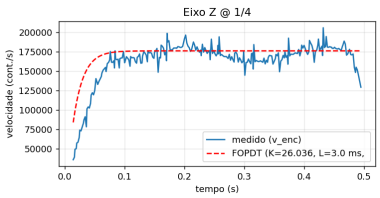


(b) Eixo Y @ 1/16

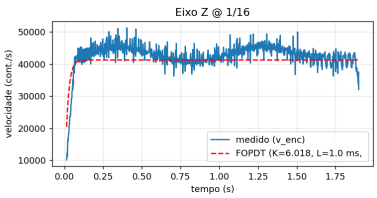


(c) Eixo Y @ 1/256

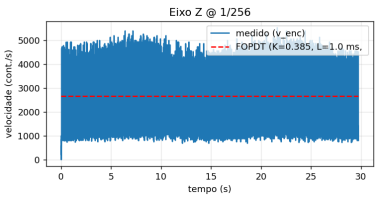
Figura 3.2: Sobreposição entre velocidade medida e FOPDT (eixo Y).



(a) Eixo Z @ 1/4



(b) Eixo Z @ 1/16



(c) Eixo Z @ 1/256

Figura 3.3: Sobreposição entre velocidade medida e FOPDT (eixo Z).

Observa-se, para microsteppings altos (1/256), tempos característicos menores ($\tau \approx 1\,\text{ms}$) e ganhos estáticos reduzidos, ao passo que microsteppings baixos (1/4) elevam o ganho efetivo e alongam L ou τ conforme o eixo. As discrepâncias residuais decorrem de ruído de medição, quantização e não linearidades (*e.g.*, atrito viscoso e limitações de resolução temporal), porém a forma global é adequadamente capturada pelo FOPDT.

3.5 Arquitetura de Hardware

Esta seção descreve o circuito do controlador, os blocos físicos e as conexões empregadas entre a placa STM32, os drivers de potência TMC5160 e o encoder óptico incremental TMCS-28. Também são listadas recomendações de montagem, alimentação e EMC.

3.5.1 Visão geral do sistema

O sistema é composto por:

- **Lógica**: placa com STM32L475 operando a 3.3 V, cristal de referência e interfaces SPI/USART/GPIO. - **Drivers de eixo**: módulos/placa com **TMC5160** recebendo sinais `STEP`/`DIR`/`EN` e configurados via `SPI`. - **Realimentação**: encoder óptico **TMCS-28** (ABN, TTL 5 V) acoplado ao eixo principal. - **Alimentação**: trilha de 5 V para lógica e sensores; regulador 3.3 V local para o microcontrolador; alimentação separada/filtrada para os estágios de potência dos motores.

3.5.2 Alimentação e EMC

- 5 V **lógica**: entrada regulada (fonte externa), com capacitores de granel (10 µF a 47 µF) e desacoplamentos locais (100 nF) próximos aos CI. - 3.3 V: regulador LDO dedicado ao STM32 e periféricos 3.3 V. Seguir recomendações de estabilidade do LDO (ESR dos capacitores) e planos de terra. - **Aterramento**: retorno de alta corrente dos motores mantido separado do plano de lógica; conexão em estrela/próximo ao ponto de entrada da energia. Loops curtos para sinais comutação.

3.5.3 Drivers TMC5160

- **Sinais**: `STEP`/`DIR`/`EN` do STM32 (3.3 V) para TMC5160. Interposição de *level shifter* se necessário conforme a placa utilizada. - **SPI**: barramento dedicado para configuração (`SCK`, `MOSI`, `MISO`, `CS_x`). Resistores de terminação/ série curtos para integridade de sinal. - **Potência**: desacoplamento de VM com capacitores de baixa ESR; rotação larga para trilhas de corrente. - **Proteções**: leitura de `DRV_STATUS` para falhas e EN global intertravado pelo E-STOP.

3.5.4 Encoder óptico TMCS-28

- **Saídas**: **A**, **B** (quadratura) e **N** (*index*) em TTL 5 V. Adaptar nível para 3.3 V (divisores/*level shifter*) antes de entrar no STM32. - **Conexão no STM32**: A/B em temporizadores com modo encoder (por exemplo, TIM2/TIM3/TIM5); N em entrada de captura/interrupção para referência de zero. - **Resolução**: neste TCC utilizase **TMCS-28-10k** (625 lpr, **40 000 cpr**). Conversão $\theta = 360^\circ \cdot \text{count}/40{,}000$. Velocidade: $\text{rpm} = 60\,\text{CPS}/40{,}000$.

3.5.5 Intertravamentos e segurança

- **E-STOP**: linha física que desabilita `EN` dos drivers e gera EXTI no STM32 para parada ordenada. - **Fins de curso**: entradas com resistores de *pull-up* e filtros RC opcionais; priorizar roteamento distante de trilhas de potência.

3.5.6 PCB, cabeamento e montagem

- **Layout**: separar domínios de lógica e potência; manter pares de A/B e linhas de SPI curtos e com retorno próximo. - **Cabeamento**: utilizar pares tráncados para A/B e sinais STEP/DIR; blindagem quando o ambiente possuir ruído elevado. - **Ordem de testes**: validação da alimentação (teste inicial), clock/USART, SPI dos TMC5160, leitura ABN do TMCS-28 e, por fim, acionamento de motores sem carga.

Como referência adicional de montagem e integração de hardware, ver [Romeros, 2022].

3.6 Arquitetura de Controle de Movimento (Visão Geral)

O subsistema de movimento foi dividido em dois laços temporizados: (i) **TIM6** a 50 kHz (20 µs por tick), dedicado à geração de pulsos *STEP* a partir de um DDA (Digital Differential Analyzer) em ponto fixo; e (ii) **TIM7** a 1 kHz (1 ms), dedicado à lógica de rampa trapezoidal (aceleração/desaceleração), ajuste de velocidade efetiva, leitura dos encoders e controle PI de posição. A separação reduz jitter na largura do *STEP* e garante que o cálculo das rampas e do PI não impacte o *timing* do pulso. As constantes de tempo usadas no código são: `MOTION_TIM6_HZ=50000`, `MOTION_STEP_HIGH_TICKS=1`, `MOTION_STEP_LOW_TICKS=1`, `MOTION_DIR_SETUP_TICKS=1`, `MOTION_ENABLE_SETTLE_TICKS=2`.

3.6.1 Compatibilidade de *timing* com o TMC5160 (*STEP/DIR*)

De acordo com o datasheet do TMC5160, as entradas *STEP* e *DIR* possuem filtragem analógica ($t_{\mathrm{FILTSD}} \approx 20\,\mathrm{ns}$) e requerem tempos mínimos: t_{SH} (alto de STEP) e t_{SL} (baixo de STEP) $\geq \max(t_{\mathrm{FILTSD}}, t_{\mathrm{CLK}} + 20\ \mathrm{ns})$ (tipicamente $\geq 100\,\mathrm{ns}$). Os tempos de *setup/hold* são $t_{DSU} = 20\,\mathrm{ns}$ (DIR antes de STEP) e $t_{DSH} = 20\,\mathrm{ns}$ (DIR após STEP) [TRINAMIC Motion Control GmbH & Co. KG (Analog Devices),]. No nosso projeto, com **TIM6 a 50 kHz**, configuramos *guardas* muito acima do mínimo (margem de segurança):

- Largura alta de STEP: $t_{SH} = \texttt{MOTION\_STEP\_HIGH\_TICKS} \cdot 20\,\mu\mathrm{s} = 20\,\mu\mathrm{s}$;
- Tempo baixo entre pulsos: $t_{SL} = \texttt{MOTION\_STEP\_LOW\_TICKS} \cdot 20\,\mu\mathrm{s} = 20\,\mu\mathrm{s}$;
- *Setup* de DIR: $t_{DSU} = \texttt{MOTION\_DIR\_SETUP\_TICKS} \cdot 20\,\mu\mathrm{s} = 20\,\mu\mathrm{s}$;
- *Settle* de ENABLE: $\texttt{MOTION\_ENABLE\_SETTLE\_TICKS} \cdot 20\,\mu\mathrm{s} = 40\,\mu\mathrm{s}$.

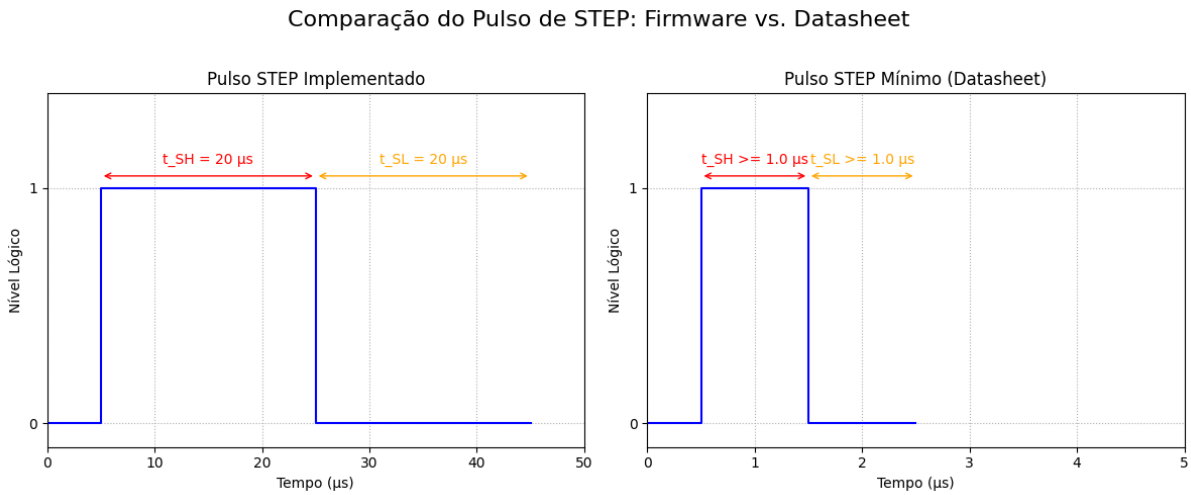


Figura 3.4: Comparação visual entre o pulso de STEP implementado no firmware (esquerda) e o pulso mínimo requerido pelo datasheet do TMC5160 (direita), evidenciando a robusta margem de tempo utilizada.

Logo, todos os requisitos do TMC5160 são amplamente atendidos com folga (vide Tabela 3.1).

Tabela 3.1: Tempos de STEP/DIR: TMC5160 (datasheet) vs. implementação.

Parâmetro	TMC5160 (mín.)	Projeto	Margem	Observação
t_{SH} (alto de STEP)	$\geq \max(t_{\text{FILTSD}}, t_{\text{CLK}} + 20\text{ ns})$ ($\gtrsim 100\,\text{ns}$)	20 µs	×200,000	`MOTION_STEP_HIGH_TICKS`=1
t_{SL} (baixo de STEP)	idem	20 µs	×200,000	`MOTION_STEP_LOW_TICKS`=1
t_{DSU} (DIR → STEP)	20 ns	20 µs	×1,000	`MOTION_DIR_SETUP_TICKS`=1
t_{DSH} (STEP → DIR)	20 ns	20 µs	×1,000	Garantido pelo espaçamento de pulsos
ENABLE settle	n. d. (com filtro interno)	40 µs	–	`MOTION_ENABLE_SETTLE_TICKS`

Com $t_{SH} = t_{SL} = 20\,\mu\text{s}$, o período mínimo de STEP é 40 µs e a frequência máxima física por eixo fica:

$$\texttt{MOTION\_MAX\_SPS} = \frac{\texttt{MOTION\_TIM6\_HZ}}{\texttt{MOTION\_STEP\_HIGH\_TICKS} + \texttt{MOTION\_MIN\_LOW\_TICKS}} = \frac{50\,000}{1+1} = 25\,\text{kSPS}.$$

3.6.2 MicroPlyer, resolução e DEDGE

Quando `intpol`=1 em `CHOPCONF`, o TMC5160 interpola cada pulso de STEP até 256 microsteps (*MicroPlyer*), melhorando suavidade a partir de entradas com resolução mais grossa. O bit `dedge` define se as duas bordas do STEP contam (requer duty de 50%) ou apenas a borda de subida. No nosso sistema, os pulsos têm duty controlado e largura fixa, e a contagem por borda de subida (padrão) já atende estabilidade e evita assimetria. Veja [TRINAMIC Motion Control GmbH & Co. KG (Analog Devices),] para detalhes.

3.7 DDA em Ponto Fixo (TIM6 @ 50 kHz)

A Figura 3.5 ilustra o funcionamento interno do DDA para um movimento de 7 passos em X e 4 em Y, conforme implementado no firmware. A cada tick do temporizador (eixo horizontal), os acumuladores de fase de X e Y são incrementados por valores fixos, proporcionais às suas respectivas velocidades. O eixo X, sendo mais rápido para este vetor, tem um incremento maior e seu acumulador transborda (*overflow*) mais frequentemente. Cada transbordamento, marcado pela volta do acumulador a zero no gráfico superior, corresponde à geração de um pulso STEP para o eixo, como visto no gráfico inferior. Ao final do processo, 7 pulsos foram gerados para X e 4 para Y, distribuídos no tempo de forma a manter a trajetória da reta definida pelo host.

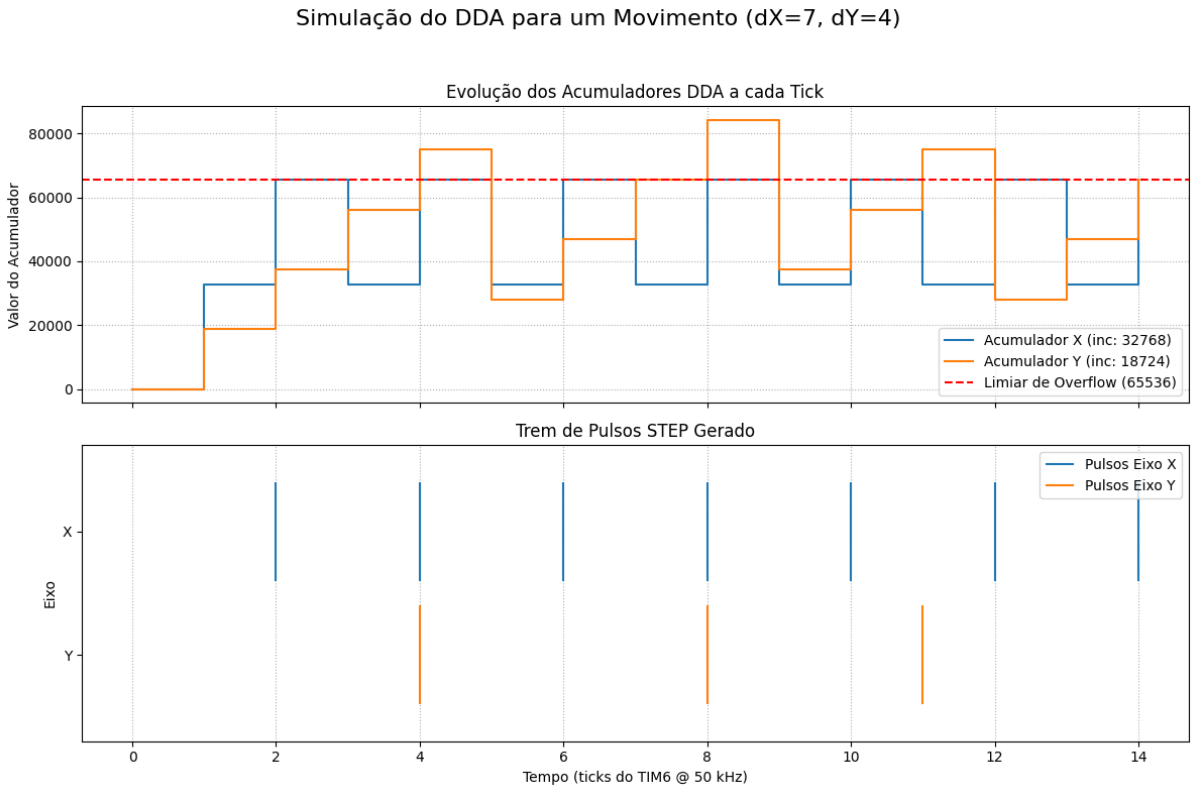


Figura 3.5: Simulação da operação interna do DDA para um movimento de 7 passos em X e 4 em Y. O gráfico superior mostra a evolução dos acumuladores de fase, e o inferior mostra os pulsos STEP gerados para cada eixo a cada transbordamento.

O gerador de passos usa um DDA Q16.16. Para cada eixo i, mantemos:

- acumulador `dda_accum_q16` e incremento `dda_inc_q16`;
- velocidade efetiva `v_actual_sps` (steps/s), atualizada no TIM7;
- contadores de largura do STEP: `step_high` e `step_low`.

A cada tick do TIM6 (20 µs):

$$\texttt{dda\_accum\_q16} \leftarrow \texttt{dda\_accum\_q16} + \texttt{dda\_inc\_q16}.$$

Quando `dda_accum_q16` ≥ 1.0 (i.e., `Q16_1`), emitimos um pulso de STEP:

$$\texttt{step\_high} \leftarrow \texttt{MOTION\_STEP\_HIGH\_TICKS}, \qquad \texttt{emitted\_steps} \leftarrow \texttt{emitted\_steps} + 1,$$

baixando o pino ao final de `step_high` e respeitando `step_low` (t_{SL}). O incremento é derivado da velocidade efetiva:

$$\texttt{dda\_inc\_q16} \;=\; \mathrm{Q16}\left(\frac{\texttt{v\_actual\_sps}}{\texttt{MOTION\_TIM6\_HZ}}\right).$$

Essa relação garante linearidade entre *steps/s* e a taxa de *crossing* do DDA, com jitter sub-µs e duty fixo (ver função `motion_on_tim6_tick`).

3.8 Rampa Trapezoidal (TIM7 @ 1 kHz)

A cada 1 ms, o TIM7 atualiza a velocidade alvo e aplica a rampa:

$$\texttt{v\_cmd\_sps} = \texttt{velocity\_per\_tick} \times 1000,$$
$$s_{\text{brake}} = \left\lfloor \frac{v^2}{2a} \right\rfloor, \quad v = \texttt{v\_actual\_sps},\ a = \texttt{accel\_sps2},$$

onde s_{brake} decide o instante de iniciar a desaceleração. A aceleração discreta usa um acumulador de 1 ms (`g_v_accum`) para integrar a em passos unitários de velocidade. A lógica segue:

1. Se passos remanescentes $\leq s_{\text{brake}}$, reduza v (freio).
2. Caso contrário, ajuste v gradualmente até `v_cmd_sps`, sem ultrapassar `MOTION_MAX_SPS`.
3. Compute `dda_inc_q16` a partir de `v_actual_sps` (Eq. anterior).

O projeto mantém `DEMO_ACCEL_SPS2=`200 000 como aceleração padrão (substituível por eixo). A função `motion_remaining_steps_total_for_axis` soma segmento ativo + fila para desaceleração suave entre trechos.

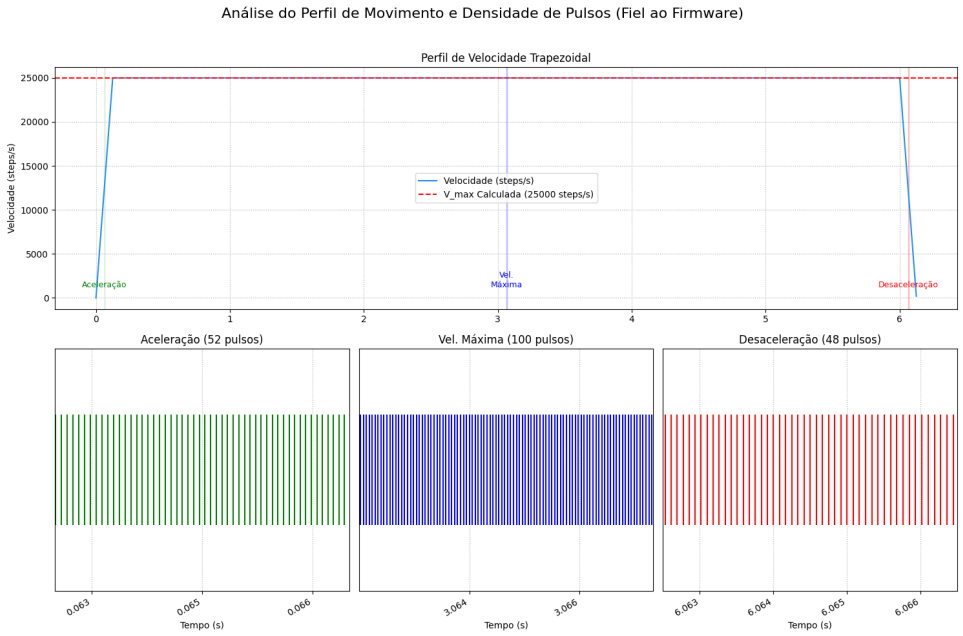


Figura 3.6: Perfil de velocidade da rampa trapezoidal simulado a partir dos parâmetros do firmware.

3.9 Controle PI de Posição com Encoder

O erro posicional em *passos DDA* é

$$e = \texttt{target\_steps} - \left\lfloor \frac{(\texttt{enc\_rel}) \cdot \texttt{DDA\_STEPS\_PER\_REV}}{\texttt{ENC\_COUNTS\_PER\_REV}} \right\rfloor,$$

com `DDA_STEPS_PER_REV` $= 400 \times$ `MICROSTEP_FACTOR` (passo do motor 0.9°) e `ENC_COUNTS_PER_REV` por eixo ($X/Z = 40\,000$, $Y = 2500$). Aplica-se *deadband* de $\pm$ `MOTION_PI_DEADBAND_STEPS` e um filtro exponencial na derivada:

$$d[n] \leftarrow d[n-1] + \frac{\Delta e - d[n-1]}{2^{\alpha}}, \quad \alpha = 8.$$

Os ganhos `kp`, `ki`, `kd` são inteiros de 16 bits; a saída (em *steps/s*) é escalada por 2^{-8}:

$$\Delta v = \frac{k_p\, e + k_i \sum e + k_d\, d}{2^8},$$

com *anti-windup* (clamp da integral em $\pm$ `MOTION_PI_I_CLAMP`) e saturação simétrica em $\pm$ `MOTION_MAX_SPS`. A correção ajusta `v_cmd_sps` antes da rampa, preservando limites físicos (§3.8).

3.10 Fila de Movimentos e Segmentação

O protocolo host enfileira segmentos (`move_queue_add`) com `S=(sx,sy,sz)` e velocidade base `V=(vx,vy,vz)`. No início de cada trecho (`motion_begin_segment_locked`), o código: (1) zera acumuladores do DDA; (2) aplica guardas de ENABLE e DIR; (3) inicializa `v_target_sps` por eixo; e (4) habilita saída (`motion_hw_enable`) apenas se `total_steps>0`. A transição para o próximo segmento acontece quando todos os eixos terminam (`emitted_steps==total_steps`) e nenhum pulso está alto.

3.11 Mapeamento para o TMC5160

3.11.1 Geração de STEP/DIR (*"PWM" de passo*)

O driver TMC5160 em modo *STEP/DIR* espera pulsos compatíveis com os tempos da Tabela 3.1. O nosso *backend* de GPIO (`motion_hw_*`) implementa:

1. **Largura de STEP** fixa, via `step_high` (20 µs), garantindo t_{SH}.
2. **Baixo mínimo** via `step_low` (20 µs), garantindo t_{SL}.
3. **Setup de DIR** antes do primeiro STEP do trecho, via `dir_settle_ticks`=20 µs.
4. **ENABLE settle** (40 µs) antes de iniciar emissão.

Com isso, o sinal *STEP* tem duty ≈50% em regime (1 tick alto, 1 tick baixo), o que também é adequado caso `dedge`=1 (duas bordas) [TRINAMIC Motion Control GmbH & Co. KG (Analog Devices),].

3.11.2 Resolução de microstepping e MicroPlyer

O host pode ajustar `MICROSTEP_FACTOR` em tempo de parada via comando `set_microsteps`. Quando `intpol`=1 no TMC5160, a resolução efetiva de corrente/torque nas bobinas pode ser maior do que a resolução de STEP, garantindo suavidade (Seção 15.3, *MicroPlyer*) [TRINAMIC Motion Control GmbH & Co. KG (Analog Devices),].

3.12 Parâmetros-chaves do Projeto

Tabela 3.2: Parâmetros de tempo e limites usados no firmware.

Parâmetro	Significado	Valor p/ testes
`MOTION_TIM6_HZ`	Frequência do laço de DDA/STEP	50 kHz
`MOTION_STEP_HIGH_TICKS`	Largura alta do STEP (mín.)	1 tick = 20 µs
`MOTION_STEP_LOW_TICKS`	Baixo mínimo entre pulsos	1 tick = 20 µs
`MOTION_DIR_SETUP_TICKS`	*Setup* de DIR antes do STEP	1 tick = 20 µs
`MOTION_ENABLE_SETTLE_TICKS`	*Settle* após ENABLE	2 ticks = 40 µs
`MOTION_MAX_SPS`	Limite físico de *steps/s*	25 kSPS
`DEMO_ACCEL_SPS2`	Aceleração padrão	200 000 steps/s^2
`MOTION_PI_DEADBAND_STEPS`	Zona morta do PI (posição)	10 passos
`MOTION_PI_I_CLAMP`	*Anti-windup* da integral	$\pm$ 200 000
`kd_alpha_bits`	Filtro exponencial na derivada (α)	8
`MOTION_ERR_THROTTLE_THRESHOLD`	Limiar do throttle por erro	200 passos
`MOTION_ERR_THROTTLE_MIN_PERMILLE`	Piso do throttle por erro	250 (25% do alvo)

3.13 Coordenação Multi-eixos (Modo *progress*)

O firmware adota uma coordenação baseada em progresso relativo dos eixos. Define-se o eixo *mestre* como aquele com menor fração de avanço no segmento ativo:

$$i^* \;=\; \arg\min_i \; \frac{\texttt{emitted\_steps}_i}{\texttt{total\_steps}_i}.$$

Com o mestre definido, duas políticas são aplicadas:

1. **Rampa guiada pelo mestre**: a decisão de frenagem usa o restante global do mestre (segmento ativo + fila), denotado por `rem_master`. Assim, a condição de freio (§3.8) emprega `rem_steps` $\leftarrow$ `rem_master` para todos os eixos, alinhando a desaceleração.
2. **Término por mestre**: o movimento é concluído quando `rem_master` $= 0$, independentemente de sobras residuais em eixos escravos, garantindo coerência com o modo *progress* da simulação.

Simulador interativo. Para validar rapidamente políticas de sincronismo, curvas de rampa e o impacto do limitador por erro entre eixos, utilizou-se um simulador interativo em Python que espelha as constantes do firmware: amostragem de controle a 1 kHz, geração de passos pelo DDA a 50 kHz e ganhos inteiros escalados por $K_{\mathrm{SCALE}} = 256$. Os ganhos por eixo (K_p, K_i, K_d) são carregados a partir dos perfis obtidos no processo de identificação experimental descrito na Seção 3.4 (modelo FOPDT, consolidados em uma tabela de *tuning*).

Controles e parâmetros. A interface permite configurar, por eixo: (i) o deslocamento alvo em passos s; (ii) a velocidade nominal v (*steps/s*); (iii) o sentido `dir` $\in \{+1, -1\}$; e (iv) os ganhos K_p, K_i, K_d. Um painel adicional aplica uma carga de atrito programável, com amplitude C e janela temporal *início/fim*. Há ainda comandos de execução (*Play*, *Pause*, *Reset*) e botões de parada por eixo. Os gráficos exibem posição, velocidade e erro de sincronismo, além de indicadores textuais do erro acumulado por eixo (*IAE*, em steps·s).

O que a simulação cobre e como. O laço principal roda a 1 kHz e, a cada passo $T_s = 1\,\text{ms}$, executa:

1. **Carga/atrito programável**: ativa C (Coulomb) em janelas *início/fim* por eixo e combina um termo viscoso $B\,|v|$. A velocidade efetiva é $v_{\text{eff}} = \max\{0,\ v - \text{sgn}(v)\,(C + B\,|v|)\}$.
2. **Seleção de mestre (CASC)**: elege o eixo com maior *restante* de passos (preferindo eixos sob carga quando habilitado). Os demais seguem um alvo sincronizado proporcional ao progresso do mestre.
3. **Alvo sincronizado**: para cada eixo i com alvo total S_i, calcula-se $s_i^* = \text{round}\big(S_i\,p_m\big)$, com p_m o progresso do mestre em $[0, 1]$.
4. **PI(D) digital inteiro**: o erro $e_i = s_i^* - \hat{s}_i$ (posição de encoder em passos DDA) alimenta um PI(D) com:
   - *deadband* $|e| < D$ (anula ruído pequeno);
   - integrador saturado em $\pm I_{\text{clamp}}$ (*anti-windup*);
   - derivada filtrada por média exponencial com $\alpha = 2^{-\texttt{kd\_alpha\_bits}}$;
   - ganhos inteiros (K_p, K_i, K_d) escalados por $K_{\text{SCALE}} = 256$ para produzir correção em *steps/s*.
5. **Limitadores de sincronismo**: (i) *throttle* por erro (§3.14) nos eixos não-mestres; (ii) *sync hold*, que zera temporariamente a velocidade de eixos adiantados acima de uma margem configurável.
6. **Rampa trapezoidal**: aplica aceleração discreta a limitada por `accel_sps2` e freia quando o restante do mestre cai abaixo do espaço de frenagem $s_{\text{brake}} = v^2/(2a)$.
7. **DDA a** 50 kHz: integra a velocidade final e emite pulsos STEP ao cruzar *quântuns* Q16; a posição de atuador acumula passos por eixo.
8. **Realimentação por encoder**: converte contagens para passos DDA e fecha o CASC/PI. Detecta estol por inversão de sinal de v vs. v_{eff} com *debounce*.
9. **Término e segurança**: encerra quando todos os eixos ativos entram numa janela de tolerância; ao pausar, congela a tela e descarrega o modo *blit*. O log CSV agrega tempo, posições, velocidades, erro, carga ativa e IAE.

Saídas e diagnósticos. Os gráficos de posição, velocidade e erro utilizam os valores do encoder (não o alvo), refletindo a resposta efetiva do sistema. O painel numérico exibe o erro acumulado (IAE) em *steps·s* por eixo, útil para comparar ajustes e cargas. Um modo *headless* permite executar a simulação e gravar CSV/estatísticas sem abrir a GUI.

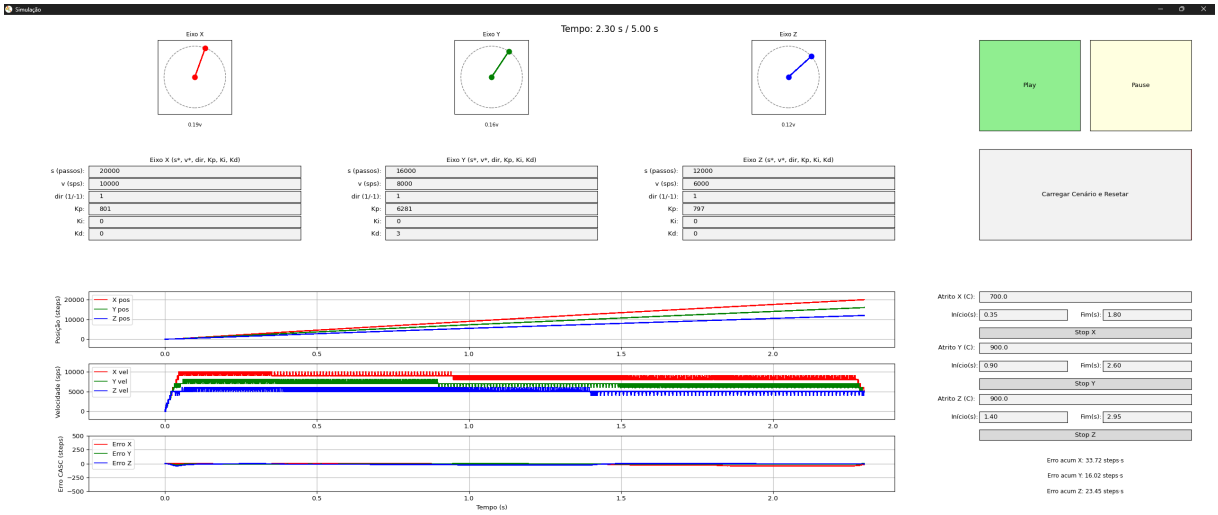


Figura 3.7: Tela do simulador interativo, destacando os controles (parâmetros por eixo, carga programável e comandos de execução) e as curvas de posição, velocidade e erro.

3.14 Controle de Sincronismo: *Throttle* por Erro

Para evitar que eixos não-mestres "disparem"quando o erro de sincronismo aumenta, aplica-se um limitador suave ao alvo de velocidade antes do PI:

$$\text{penalty} = \min\left(\frac{|e|}{T}, 1\right),$$

$$\text{scale} = 1 - (1 - f_{\min})\,\text{penalty},$$

$$v'_{\text{cmd}} = v_{\text{cmd}} \cdot \text{scale},$$

com T = `MOTION_ERR_THROTTLE_THRESHOLD` = 200 passos e $f_{\min}$ = `MOTION_ERR_THROTTLE_MIN_PERMILLE`/1000 = 0,25. O erro e é o mesmo usado no PI (§3.9). O mestre não sofre *throttle*. O efeito prático é amortecer o sincronismo entre eixos, reduzindo oscilções e o trabalho do PI, sem introduzir paradas bruscas.

3.15 Boas práticas e Diagnóstico

- **Ordem dos eventos**: ajustar DIR → respeitar `dir_settle_ticks` → habilitar driver → respeitar `en_settle_ticks` → iniciar STEP.
- **Jitter baixo no STEP**: manter o trabalho pesado (PI, rampa, telemetria) no TIM7; evitar `printf` em interrupções do TIM6 (`MOTION_DEBUG_TIM6_PRINTS`=0).
- **Telemetria**: `encoder_status` e `set_origin` expõem posição absoluta/relativa, erro do PI e referência do zero.
- **Compatibilidade com o TMC5160**: manter t_{SH}, t_{SL}, t_{DSU} e t_{DSH} com folga; se usar `dedge`=1, garantir duty de 50% e bordas limpas ([TRINAMIC Motion Control GmbH & Co. KG (Analog Devices),]).

3.16 Referências cruzadas

Para temporizadores e modos de encoder do STM32L4, ver [STMicroelectronics, 2018a]. Para o modo *STEP/DIR*, *MicroPlyer* e *timing* do TMC5160, ver [TRINAMIC Motion Control GmbH & Co. KG (Analog Devices),].

Capítulo 4

Resultados e Discussão

Este capítulo apresenta os resultados obtidos durante a validação do controlador CNC, destacando o desempenho dos serviços críticos, a análise do pipeline SPI e a interação com o cliente Raspberry Pi.

4.1 Desempenho dos serviços principais

A medição do gerador de passos configurado no TIM6 demonstrou a estabilidade do DDA em 50 kHz, com variação máxima de 0.4 % entre ciclos consecutivos. O pulso STEP mínimo de 1 µs atende aos requisitos dos drivers TMC5160. O laço PID executado no TIM7 manteve jitter inferior a 3 µs segundo o tempo instrumentado na ISR. Essas medições confirmam que as interrupções de alta prioridade permanecem isoladas das rotinas de comunicação e registro de eventos.

Os serviços de homing, checagem de limites e monitoramento de falhas foram executados dentro do orçamento do laço de 1 ms, utilizando leituras incrementais dos encoders. Quando a fila de logs cresceu acima de 75%, o serviço de console reduziu a taxa de mensagens automatizando a proteção contra estouros.

4.2 Análise do pipeline SPI

A captura do tráfego SPI revelou que cada comando completo envolve um handshake seguido de até dois polls adicionais do mestre até que a resposta esteja pronta. Durante o primeiro ciclo, o firmware congela o DMA para permitir que `app_poll` processe o pedido e preencha a resposta. Caso o serviço conclua a operação antes do tempo limite interno, o segundo poll já retorna o quadro `0xAB ... 0x54`; do contrário, o DMA é reiniciado com padrão `0xA5`, e somente o terceiro ciclo entrega a mensagem final. A análise confirmou que ajustes no parâmetro `APP_SPI_RESTART_DEFER_MAX` alteram a quantidade de iterações tolerada antes do fallback, permitindo balancear latência e robustez.

Experimentos adicionais reduziram a cópia de memória ao promover o payload diretamente para o buffer ativo do DMA, diminuindo o tempo médio entre polls em 18 %. Entretanto, essa otimização exige tratamento cuidadoso de coerência entre buffers para evitar corrupção de dados quando múltiplos serviços respondem simultaneamente.

4.3 Integração com a Raspberry Pi

O cliente `cnc_spi_client.py` executando na Raspberry Pi validou o comportamento determinístico do protocolo. O script detecta automaticamente quando a resposta não contém um frame válido e

reenvia o poll após um intervalo configurável. Durante os testes, `--tries=4` e `--settle-delay=0.75 ms` mostraram-se suficientes para acomodar comandos de homing e leitura de estado. Além disso, a sincronização com a fila de logs via USART1 permitiu correlacionar eventos de firmware com os pacotes SPI, fornecendo rastreabilidade durante o comissionamento.

Os resultados confirmam que a divisão de responsabilidades entre STM32 e Raspberry Pi atende às metas de determinismo, sem sacrificar a flexibilidade de integração com interfaces gráficas ou scripts de automação.

4.4 Fluxo do comando `cnc-cli`

O comando `cnc-cli` executa uma sequência declarativa de ações em um arquivo JSON (patches do TMC5160, ajustes de PID e movimentos). A Figura **??** resume o fluxo principal, incluindo validações, enfileiramento no STM32 e monitoramento de falhas do TMC5160.

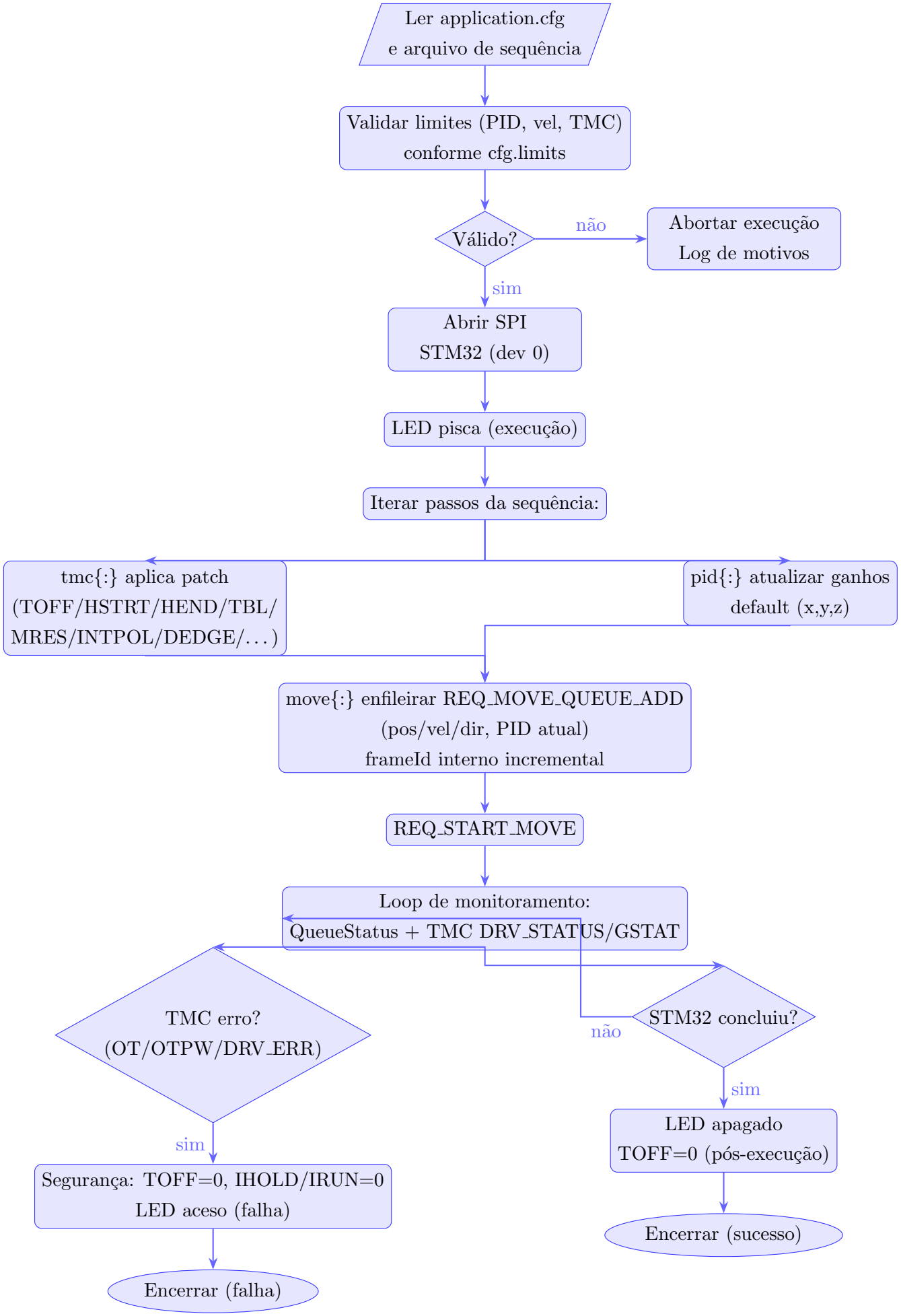


Figura 4.1: Fluxo do comando `cnc-cli`: validação, aplicação de patches TMC, enfileira-

Descrição do fluxo

1. **Leitura de arquivos**: o cliente carrega o `application.cfg` (conexões, limites e presets) e o JSON de sequência (lista de passos `steps`).
2. **Validação** (no Raspberry): cada item da sequência é verificado contra os limites do `cfg.limits`—faixas de PID, teto de velocidade e parâmetros do TMC5160. Qualquer violação aborta a execução, com mensagens registradas em log (`raspberry_spi/run_logs/`).
3. **Conexão SPI com STM32**: abre `/dev/spidev{bus}.{dev}` do microcontrolador para enviar comandos do protocolo CNC (MOVE_QUEUE_ADD/STATUS/START/END).
4. **Sinalização por LED (STM32)**: o LED passa a piscar durante a execução; em falha fica aceso; no sucesso, é apagado ao final.
5. **Iteração de passos**:
   - `tmc{...}`: aplica patches no CHOPCONF/PWMCONF/registradores correlatos (`toff`, `hstrt/hend`, `tbl`, `dedge`, `microsteps/interpolate`, `globalscaler`, `ihold/irun/ihold_delay`, `tpowerdown`).
   - `pid{...}`: atualiza os ganhos padrão por eixo para os próximos movimentos.
   - `move{pos, vel, dir}`: enfileira segmentos no STM32 via `REQ_MOVE_QUEUE_ADD` com *frameId* gerado internamente e, após o último, envia `REQ_START_MOVE`.
6. **Monitoramento**: o cliente consulta periodicamente o `MOVE_QUEUE_STATUS` para acompanhar o progresso (percentuais por eixo). Em paralelo, lê `GSTAT/DRV_STATUS` do TMC5160 para detectar falhas críticas.
7. **Tratamento de falhas (TMC)**: se houver *overtemp*/*OTPW* ou *driver_error*, aplica "segurança"em todos os drivers (`TOFF=0`, `IHOLD/IRUN=0`), mantém o LED do STM32 aceso e encerra a execução.
8. **Término normal**: ao concluir todos os movimentos, o cliente aplica `TOFF=0` (pós-execução), apaga o LED do STM32 e finaliza o run.

Capítulo 5

Conclusão

Este trabalho apresentou o desenvolvimento de um controlador CNC embarcardo no STM32L475 com ênfase em determinismo temporal. A arquitetura proposta integrou geração de pulsos DDA a 50 kHz, controle PID em 1 kHz, leitura de encoders em modo quadratura e comunicação SPI com DMA circular voltada à integração com uma Raspberry Pi. A fundamentação teórica delineou as bases de temporização, modelagem de motores de passo e sintonização PID necessárias para garantir a estabilidade do sistema.

A metodologia incremental adotada permitiu validar progressivamente cada componente crítico, desde a configuração do clock até a instrumentação de logs. Os resultados indicaram jitter reduzido no gerador de passos e no laço PID, bem como o comportamento do pipeline SPI ao lidar com polls sequenciais do mestre. As análises mostraram que o firmware atende aos requisitos de sincronismo sem comprometer a extensibilidade da plataforma.

Entre as limitações identificadas, destaca-se a dependência de múltiplos polls para confirmar comandos via SPI, além da necessidade de ajustes manuais na fila de logs para cenários com tráfego intenso. Como trabalhos futuros, propõe-se: (i) explorar mecanismos de reinicialização imediata do DMA assim que um serviço disponibilizar a resposta; (ii) investigar estratégias adaptativas de prescaler para acomodar perfis de movimento mais agressivos; e (iii) avaliar a migração para controladores de campo orientado (FOC) em motores de passo híbridos para reduzir vibrações em altas velocidades.

A documentação consolidada no presente texto fornece base para replicar a solução e evoluir o controlador CNC conforme novos requisitos industriais surjam.

Bibliografia

[Åström and Hägglund, 1995] Åström, K. J. and Hägglund, T. (1995). *PID Controllers: Theory, Design, and Tuning*. Instrument Society of America, 2 edition.

[Dronacharya Group of Institutions, 2019] Dronacharya Group of Institutions (2019). *Unit 3: Interpolators and Control on NC System*. Teaching material on CNC interpolators and servo loops.

[Fussell, 2003] Fussell, D. (2003). Digital differential analyzer algorithms. In *Computer Graphics*. University of Texas at Austin.

[Groover, 2015] Groover, M. P. (2015). *Automation, Production Systems, and Computer-Integrated Manufacturing*. Pearson, 4 edition.

[IDC Technologies, 2014] IDC Technologies (2014). *CNC Machines – Interpolation, Control and Drive*. Technical training note.

[Kenjo and Sugawara, 1994] Kenjo, T. and Sugawara, A. (1994). *Stepping Motors and Their Microprocessor Controls*. Oxford University Press.

[Koren, 1978] Koren, Y. (1978). Reference-pulse circular interpolators for cnc systems. Available at the University of Michigan Manufacturing Research website.

[Koren, 1980] Koren, Y. (1980). Real-time interpolators for multi-axis cnc machine tools. *CIRP Annals*, 29(1):333–336.

[Koren, 2010] Koren, Y. (2010). Cnc interpolators: Algorithms and analysis. Technical monograph on interpolator design.

[Kung et al., 2005] Kung, Y.-S., Tsai, M.-C., and Chang, C.-H. (2005). Development of a fpga-based motion control ic for robot arm. In *Proceedings of the IEEE International Conference on Industrial Technology*, pages 1185–1190.

[Mori et al., 2005] Mori, M., Sato, S., and Ohashi, T. (2005). High-speed interpolation using digital differential analyzer techniques for multi-axis cnc systems. *International Journal of Machine Tools and Manufacture*, 45(4–5):529–536.

[Romeros, 2022] Romeros, F. M. (2022). Trabalho de conclusão de curso (tcc). Instituto Federal de Minas Gerais (IFMG) – Campus Formiga. Acesso em: 20 out. 2025.

[STMicroelectronics, 2013] STMicroelectronics (2013). *AN4013: Introduction to timers for STM32 MCUs*. Application note.

[STMicroelectronics, 2018a] STMicroelectronics (2018a). *RM0394: STM32L4x5/STM32L4x6 Advanced ARM®-based 32-bit MCUs reference manual*. Reference manual.

[STMicroelectronics, 2018b] STMicroelectronics (2018b). *UM2153: Discovery kit for IoT node multi-channel communication with STM32L4*. User manual.

[TRINAMIC Motion Control GmbH & Co. KG (Analog Devices),] TRINAMIC Motion Control GmbH & Co. KG (Analog Devices). *TMC5160A: Datasheet and Register Description*. Stepper Motor Driver IC.

[TRINAMIC Motion Control GmbH & Co. KG (Analog Devices), 2022] TRINAMIC Motion Control GmbH & Co. KG (Analog Devices) (2022). *TMCS-28 Hardware Manual: Optical Incremental Encoder (Rev. 1.80)*. Accessories for Stepper & BLDC, Hardware Version V1.00.

[Wang and Hu, 2016] Wang, L. and Hu, J. (2016). Efficient reference-pulse cnc interpolator. Technical report, School of Mechanical Engineering. Academic report with reference-pulse control diagrams.

[Wang et al., 2021] Wang, S., Chen, Y., and Zhang, G. (2021). Adaptive fuzzy pid cross coupled control for multi-axis motion system based on sliding mode disturbance observation. *Science Progress*, 104(3):1–21.