IMPLEMENTAÇÃO DE GPC DE CÔMPUTO RÁPIDO COM RESTRIÇÕES BASEADA EM MICROCONTROLADOR

João Pedro Brunoni [*] **Vinícius Berndsen Peccin** [**]
Rodolfo César Costa Flesch [*] **Julio Elias Normey-Rico** [*]

[*] *Universidade Federal de Santa Catarina, Florianópolis, Santa Catarina, Brasil, (jpbrunoni@gmail.com, rodolfo.flesch@ufsc.br, julio.normey@ufsc.br).*
[**] *Instituto Federal de Santa Catarina, Chapecó, Santa Catarina, Brasil, (vinicius.peccin@ifsc.edu.br).*

Abstract: Recent advances in algorithms and embedded systems have created possibilities to meet the demand for advanced controllers in processes with fast dynamics. In general, advanced controllers are very costly from a computational point of view and end up being used in processes with slow dynamics or need to be implemented in sophisticated computers. The implementation of advanced controllers in low-cost embedded systems, such as microcontrollers, can contribute to the use of these techniques in a greater number of processes. This paper provides a proposal of implementation of a generalized predictive control algorithm of fast computation with constraints in a microcontroller platform. Issues of automatic code generation and the improvement of the control signal computation time and required memory space are evaluated. The implemented controller was experimentally simulated and tested in a case study of a multivariable system with first order transfer functions. The computation time obtained was about 1,2 ms.

Resumo: Os recentes avanços em algoritmos e sistemas embarcados abrem possibilidades para se atender a demanda por controladores avançados em processos com dinâmicas rápidas. Em geral, os controladores avançados são bastante custosos do ponto de vista computacional e acabam sendo utilizados em processos com dinâmicas mais lentas ou em computadores mais sofisticados. A implementação de controladores avançados em sistemas embarcados de baixo custo, como microcontroladores, pode contribuir para a utilização dessas técnicas em uma maior quantidade de processos. Este trabalho apresenta uma proposta de implementação de um algoritmo de controle preditivo generalizado de cômputo rápido com restrições em uma plataforma microcontrolada. São avaliadas questões da geração automática de código e estratégias para a melhoria no tempo de cômputo do sinal de controle e espaço de memória requerida. O controlador implementado foi simulado e testado experimentalmente em um estudo de caso de uma planta multivariável com funções de transferência de primeira ordem. O tempo de cômputo obtido foi da ordem de 1,2 ms.

Keywords: Predictive control; Advanced control; Optimized GPC algorithms; Execution Time; Microcontroller.

Palavras-chaves: Controle preditivo; Controle avançado; Algoritmos otimizados de GPC; Tempo de execução; Microcontrolador.

1. INTRODUÇÃO

A utilização de técnicas avançadas de controle tem sido o foco de muitos pesquisadores nos últimos anos, e entre essas técnicas está o controle preditivo baseado em modelo (MPC, do inglês *Model Predictive Control*). Desenvolvido no fim dos anos 70, o MPC tem uma participação expressiva no controle de processos da indústria petroquímica, por possibilitar aplicações em sistemas multivariáveis (MIMO, do inglês *Multiple Inputs Multiple Outputs*), facilitar o tratamento de restrições na entrada e saída do sistema, e poder compensar intrinsecamente atrasos de transporte (Normey-Rico e Camacho, 2007, Camacho e Bordons, 2004).

O MPC pode ser formulado de diversas maneiras, e entre as mais populares na indústria está o controle preditivo generalizado (GPC, do inglês *Generalized Predictive Control*) (Camacho e Bordons, 2004). O GPC utiliza a função de transferência como modelo para realizar o cálculo das predições, e deste modo, é uma das formulações MPC mais utilizadas na indústria, pois conceitos comuns neste setor, como tempo morto, constantes de tempo e ganhos, são mais facilmente representados por meio desta formulação (Peccin et al., 2021).

Devido ao alto custo computacional, o MPC com restrições é geralmente aplicado em processos com dinâmicas lentas, nos quais os períodos de amostragem se estabelecem na ordem de segundos ou minutos (Peccin et al., 2020). Na indústria petroquímica, por conta desta característica, a estrutura de controle costuma seguir o que é abordado em Zanin (2001), em que o MPC opera numa frequência menor que controladores regulatórios clássicos, apenas controlando as suas referências. Entretanto, melhorias e modificações para o MPC têm sido o foco de muitos pesquisadores a fim de diminuir o tempo de execução da técnica e, deste modo, possibilitar que o controle preditivo possa ser implementado diretamente nas variáveis do processo, sem a necessidade de um controlador clássico em cascata. Com isso, o sistema de controle é simplificado, tornando-o mais confiável, uma vez que evita-se problemas de comunicação entre os controladores. Para alcançar estas melhorias, diferentes plataformas computacionais estão sendo testadas para a implementação deste modelo, como processadores de sinais digitais (DSPs, do inglês *Digital Signal Processor*) e arranjos de portas programáveis em campo (FPGA, do inglês *Field Programmable Gate Arrays*) (Elmorshedy et al., 2021, Rodriguez et al., 2013). Os FPGAs possuem vantagens em termos de velocidade de execução, paralelização e determinismo se comparados aos processadores (Peccin et al., 2020), devido à possibilidade de realizar um processamento paralelo e de haver diversos elementos lógicos em um único chip (Sharma et al., 2016). Quando integrados com algoritmos eficientes para o GPC, os FPGAs podem otimizar o controle para operar com um pior caso de tempo de execução (WCET, do inglês *Worst Case Execution Time*) na ordem de microssegundos. Por exemplo, em Peccin et al. (2021), um FPGA foi utilizado com algoritmos otimizados de GPC, e a validação foi feita utilizando como estudo de caso um inversor trifásico, alcançando WCETs na ordem de dezenas de microssegundos. Por outro lado, os microcontroladores são mais simples de programar, mais baratos e bastante populares entre os desenvolvedores.

Implementações com aritmética de ponto fixo trazem vantagens relacionadas ao menor atraso computacional e menos utilização de memória. Entretanto, a mesma pode apresentar problemas de *overflow* e *underflow*, de acordo com a seleção do número de bits para representar a parte inteira e a parte fracionária. Outro problema apresentado por essa representação é o erro relativo grande, comum em cálculos que utilizam essa aritmética (Peccin et al., 2020), podendo ocasionar erros no cálculo da ação de controle.

Este trabalho visa contribuir na forma de implementação do GPC, aplicado em um dos algoritmos do estado da arte de cômputo rápido, o GPCGPAD proposto em Peccin et al. (2021), em uma plataforma de controle baseada em microcontrolador. Outro ponto avaliado é a comparação com a implementação em aritmética de ponto flutuante e análise da relevância do erro causado pela aritmética de ponto fixo para um caso específico da aplicação em controle realimentado.

2. CONTROLE PREDITIVO GENERALIZADO

Tanto para sistemas monovariáveis (SISO, do inglês *Single Input Single Output*), quanto para sistemas multivariáveis (MIMO, do inglês *Multiple Input Multiple Output*), é possível aplicar a formulação GPC. A diferença entre as aplicações consiste em aumentar as dimensões dos parâmetros utilizados. Por exemplo, em um sistema SISO, um dos elementos utilizados para o cálculo da resposta forçada do sistema é uma matriz. Entretanto, em um sistema MIMO, este mesmo elemento será representado por uma matriz de diversas matrizes (Peccin et al., 2020).

Para desenvolver a formulação GPC, será considerado um sistema MIMO generalizado, com n_c entradas e n_o saídas, apresentado no modelo autorregressivo com média móvel, integrador e entrada controlada (CARIMA, do inglês *Controlled Autoregressive Integrated Moving Average*), retratado em Camacho e Bordons (2004). A representação de um sistema MIMO pelo modelo CARIMA é dada por:

$$\begin{aligned} A(z^{-1})y(k) =& D(z^{-1})B(z^{-1})u(k-1) \\ &+ C(z^{-1})\frac{e(k)}{\Delta}, \end{aligned} \tag{1}$$

onde $A(z^{-1})$, $B(z^{-1})$ e $C(z^{-1})$ são matrizes polinomiais de dimensões $n_o \times n_o$, $n_o \times n_c$ e $n_o \times n_o$, respectivamente. A primeira, está relacionada com os denominadores das funções de transferência do modelo, a segunda com os numeradores com atraso de uma amostra, e a matriz $C(z^{-1})$, pode ser utilizada para sintonizar o sistema e é normalmente utilizada como uma matriz identidade (Camacho e Bordons, 2004). A matriz $D(z^{-1})$ é diagonal com dimensão $n_o \times n_o$, e cada elemento representa o menor tempo morto entre determinadas entradas e saídas do sistema. A saída do sistema é representada pelo vetor $y(k)$, com dimensão n_o, enquanto a entrada é dada por u, um vetor de dimensão n_c, que para o modelo CARIMA possui uma amostra de atraso. Utiliza-se, ainda, um ruído branco de média zero denominado $e(k)$ e um polinômio para a modelagem de perturbações não estacionárias, representado por $\Delta = 1 - z^{-1}$.

A partir do modelo CARIMA do sistema, encontra-se a equação da saída predita do sistema, que pode ser dada como:

$$\hat{y} = G\Delta u + f, \tag{2}$$

onde $(G\Delta u)$ corresponde à resposta forçada, e f à resposta livre do sistema. Entre as principais características de um GPC, está a capacidade de obter o sinal de controle através da minimização de uma função custo do sistema, de modo que, para referências e perturbações constantes, o erro possa ser anulado em regime permanente. Nessa função, o projetista consegue sintonizar os ganhos q_δ e q_λ, que correspondem ao peso das parcelas do erro e da variação do sinal de controle, respectivamente. A função custo é dada por:

$$\begin{aligned} J =& \sum_{p=1}^{n_o} \sum_{j=N_1^{(p)}}^{N_2^{(p)}} q_\delta^{(p)} [\hat{y}^{(p)}(k+j|k) - w^{(p)}(k+j)]^2 + \\ & \sum_{l=1}^{n_c} \sum_{j=1}^{N_u^{(l)}} q_\lambda^{(l)} [\Delta u^{(l)}(k+j-1)]^2, \end{aligned} \tag{3}$$

onde o erro é representado pela subtração entre a saída p predita no instante de tempo discreto futuro $k+j$, representada por $\hat{y}^{(p)}(k+j|k)$, e a trajetória de referência no mesmo instante de tempo para a saída p, representada por $w^{(p)}(k+j)$. Deste modo, $q_\delta^{(p)}$ corresponde à ponderação

do erro. O incremento de controle é representado por $\Delta u^{(l)}$, e por conta disso, é acompanhado pela variável $q_\lambda^{(l)}$, que pondera a variação do sinal de controle. $N_1^{(p)}$, $N_2^{(p)}$, e $N_u^{(l)}$ correspondem aos horizontes de predição e de controle utilizados no projeto. As ponderações podem ser agrupadas em matrizes bloco diagonais positivas definidas por $Q_\delta^{(p)}$ e $Q_\lambda^{(l)}$.

Conforme Camacho e Bordons (2004), partindo da função custo representada em (3), é possível encontrar o problema de programação quadrática (QP, do inglês, *Quadratic Programming*), típico do GPC, representado como:

$$\begin{aligned} \min_{\Delta u} \quad & J = \min_{\Delta u} \quad (\frac{1}{2}\Delta u^T H \Delta u + b^T \Delta u) \\ \text{s.a.} \quad & \bar{R}\Delta u \leq \bar{r}, \end{aligned} \tag{4}$$

sendo $\bar{R} \in \mathbb{R}^{n_{rin} \times N_u}$ e $\bar{r} \in \mathbb{R}^{n_{rin}}$, respectivamente, uma matriz e um vetor que definem as restrições impostas, e n_{rin} equivalente ao número de restrições de desigualdade utilizadas. Pode-se perceber que, caso não sejam consideradas restrições, o sistema pode ser implementado resolvendo (4) de forma analítica, entretanto, os sistemas práticos normalmente possuem restrições relacionadas às suas entradas ou saídas.

A resolução deste QP concentra uma boa parcela do tempo de execução do algoritmo de controle e, portanto, destaca-se a importância de trabalhos que implementam maneiras mais eficientes no cômputo desta solução, como é o caso dos algoritmos abordados em Peccin et al. (2020), Peccin et al. (2021), mas particularmente do Algoritmo GPCGPAD, proposto em Peccin et al. (2021), que baseia-se no método GPAD, apresentado em Patrinos e Bemporad (2014). Por não possuir divisões em seus cálculos que necessitam ser embarcados, o algoritmo é adequado para implementações em ponto fixo, pois deste modo não acumula erro.

3. PROPOSTA DE IMPLEMENTAÇÃO EMBARCADA DO ALGORITMO GPCPAD

Nesta seção são discutidos os detalhes de implementação do algoritmo GPCGPAD em arquitetura embarcada com o intuito de acelerar o tempo de cômputo. O algoritmo do GPCGPAD para plantas MIMO foi apresentado em Peccin et al. (2021) e está reproduzido no Algoritmo 1 para a comodidade ao leitor.

3.1 Estrutura do código

Para fins de implementação, o algoritmo GPCGPAD pode ser separado em duas partes, sendo a primeira de cômputo *offline*, na qual são feitas as inicializações das variáveis e a montagem das matrizes que definem o GPC. Nessa etapa são, também, configuradas as formas de entrada e saída do controlador, como, por exemplo, o Conversor Analógico/Digital (ADC, do inglês *Analog to Digital Converter*) para as entradas e os parâmetros das saídas analógicas, feitas por meio de sinais com modulação por largura de pulso (PWM, do inglês *Pulse Width Modulation*).

A segunda parte do algoritmo consiste no cômputo *online*, ou seja, todos os cálculos e atualizações de variáveis que devem ser feitos a cada novo período de amostragem.

Algoritmo 1: Algoritmo GPCGPAD (Fonte: Peccin et al. (2021))

Entrada: $A^{(1)}$, ..., $A^{(n_o)}$, $B^{(1,1)}$, ..., $B^{(n_c,n_o)}$, $\mathbf{D}$, $\mathbf{R}$, $\mathbf{r}$
Saída: $u^{(1)}(k)$, ..., $u^{(n_c)}(k)$
Dados: n_o, n_c, $\mathbf{Q}_\lambda$, $\mathbf{Q}_\delta$, $N_u^{(1)}$, ..., $N_u^{(n_c)}$, $N_1^{(1)}$, ..., $N_1^{(n_o)}$, $N_2^{(1)}$, ..., $N_2^{(n_o)}$, $\psi^0 = \psi^{-1} = \mathbf{0}$, $\epsilon_\mathbf{f}$, $\epsilon_\mathbf{o}$, $j_{\max}$
início
  para $p = 1 : n_o$ **faça**
    $\tilde{A}^{(p)}\left(z^{-1}\right) := (1 - z^{-1})A^{(p)}\left(z^{-1}\right)$;
    para $l = 1 : n_c$ **faça**
      para $i = 1 : N_2^{(p)}$ **faça**
        $g_i^{(p,l)} := z\left(1 - A^{(p)}(z^{-1})\right)g_{i-1}^{(p,l)} + B^{(p,l)}\left(z^{-1}\right)\bar{u}_i$;
      para $i = 1 : N_u^{(l)}$ **faça**
        $\mathbf{G}^{(p,l)}_{i:N_2^{(p)}-N_1^{(p)}+1,i} = \mathbf{g}^{(p,l)}_{N_1^{(p)}:N_2(p)+1-i}$;
  $\mathbf{H} := 2(\mathbf{G}^T\mathbf{Q}_\delta\mathbf{G} + \mathbf{Q}_\lambda)$; $\breve{\mathbf{H}} := \mathbf{H}^{-1}$;
  $L := \sqrt{\sum_{i,j=1}^{n_{rin}} |\mathbf{R}\breve{\mathbf{H}}\mathbf{R}^T|^2}$;
  $k := 0$;
  enquanto *modo automático* **faça**
    se *tempo de amostragem* **então**
      para $p = 1 : n_o$ **faça**
        Adquirir saídas $y^{(p)}(k)$ e ref. $w^{(p)}(k)$;
        $f_0^{(p)} := y^{(p)}(k)$;
        para $i = 1 : N_2^{(p)}$ **faça**
          $f_i^{(p)} := z\left(1 - \tilde{A}^{(p)}\left(z^{-1}\right)\right)f_{i-1}^{(p)} + \sum_{l=1}^{n_c} B^{(p,l)}\left(z^{-1}\right)\Delta u^{(l)}(k - \mathbf{D}^{(p)} + i - 1)$;
      $\mathbf{f} = [\mathbf{f}^{(1)}_{N_1^{(1)}:N_2^{(1)}} \ \ldots \ \mathbf{f}^{(n_o)}_{N_1^{(n_o)}:N_2^{(n_o)}}]^T$;
      $\mathbf{b}^T = 2(\mathbf{f} - \mathbf{w})^T\mathbf{Q}_\delta\mathbf{G}$;
      enquanto $j < j_{\max}$ **faça**
        se $j = 0$ **então**
          $\beta := 0$
        senão
          $\beta := \frac{j-1}{j+2}$;
        $\mathbf{w} := \psi_j + \beta(\psi_j - \psi_{j-1})$;
        $\Delta\mathbf{u} := -\breve{\mathbf{H}}(\mathbf{R}^T\mathbf{w} + \mathbf{b}^T)$;
        $\mathbf{s} := (1/L)(\mathbf{R}\Delta\mathbf{u} - \mathbf{r})$;
        se $\mathbf{s}_i \leq \epsilon_f/L, \ \forall i = 1,..,n_{rin}$ **então**
          se $-\mathbf{w}^T\mathbf{s} < \epsilon_o/L$ **então**
            retorna;
        $\psi_{j+1} := \max(\mathbf{w} + \mathbf{s}, \mathbf{0})$;
        $j := j + 1$;
      $i = 1$;
      para $l = 1 : n_c$ **faça**
        $\Delta u^{(l)}(k) = \Delta\mathbf{u}_i$;
        $i = i + N_u^{(l)}$;
        $u^{(l)}(k) := u^{(l)}(k-1) + \Delta u^{(l)}(k)$;
      $k := k + 1$;

O Algoritmo 2 descreve a implementação embarcada do GPCGPAD por meio de um pseudocódigo. Desta forma, é possível visualizar que os únicos valores calculados pelo microcontrolador são aqueles que necessitam do cômputo *online*, a fim de reduzir o tempo de execução do código. Estes valores estão localizados dentro da função da interrupção, ferramenta utilizada em sistemas microcontrolados para garantir a frequência de amostragem desejada.

Para o caso implementado, por possuir restrições no valor máximo e mínimo de sinal de controle, o vetor de restrições $\bar{r}$ também passou a ser atualizado em todo instante de amostragem, uma vez que para determinar se a variação

é aceitável ou não, o valor máximo deve ser alterado de acordo com o valor atual. Caso esta restrição fosse na variação do sinal de controle, e não nos limites do sinal como um todo, este vetor poderia ser calculado previamente e inserido no código com os valores preenchidos. Observa-se que caso a variação do sinal for menor que o limite estabelecido, independente do sinal atual, ela é aceita, o que torna possível implementar de modo *offline*. A variável β, mesmo sendo atualizada em cada iteração, é pré-calculada até o valor de j_{max}, visto que é possível prever seus valores, acelerando a resposta do sistema.

Algoritmo 2: Pseudocódigo do programa

início
- Incluir as bibliotecas;
- Configurar as GPIOs;
- Preparar a função *handler* pra interrupção;
- Configurar a utilização do PWM e do ADC;

Valores *offline*: dim_A, dim_B, $\tilde{A}_{flip,1}$, $\tilde{A}_{flip,2}$, $B_{flip,1}$, $B_{flip,2}$, $B_{flip,3}$, $B_{flip,4}$, β, N_u, $N_{u,1}$, $N_{u,2}$, N_{11}, N_{12}, N_{21}, N_{22}, N, N_{rin}, $N_{rin,1}$, $N_{rin,2}$, G, H^{-1}, $Q_\delta G$, j_{max}, $\bar{R}$, L, $\frac{1}{L}$, e_f, e_o, ref;

`FUNÇÃO DA INTERRUPÇÃO()`

início
- Limpar a *flag* da interrupção;
- Atualizar o vetor de saídas y com a leitura do ADC;
- Calcular o vetor f;
- Calcular o vetor b;
- Calcular o vetor $\bar{r}$;

enquanto $loop < j_{max}$ **faça**
- Calcular o vetor w;
- Calcular o vetor $\Delta u(k)$;
- Calcular o vetor s;
- Calcular o critério de parada aux_w;
- Verificar o critério de parada através de s e aux_w;
- Calcular o vetor ψ;
- Incrementar "1"ao valor da variável *loop*;

- Calcular o vetor de entrada u;
- Calcular e atualizar o valor do *duty cycle*;

A multiplicação matricial, para implementações em sistemas embarcados, não pode ser feita de maneira trivial, pois na maioria destas plataformas, não é possível inserir matrizes como elementos de uma multiplicação. Dentro do GPCGPAD, estas operações são frequentemente utilizadas e, para implementá-las, é necessário utilizar laços de repetição *for*. Deste modo, o código fica mais extenso, e o tempo de execução do algoritmo é proporcional aos limites utilizados nestes blocos. O Algoritmo 3 demonstra o cálculo do vetor $\Delta u(k)$, onde é possível observar que o número de laços necessários neste cálculo é proporcional ao horizonte de controle utilizado no projeto. Nesse algoritmo, aux_R é um vetor auxiliar, utilizado para obter $\bar{r}$.

Algoritmo 3: Multiplicação matricial

- Cálculo de $\Delta u(k)$:

para $j \leftarrow 0$ ***até*** *Hor. de Controle* **faça**
  para $i \leftarrow 0$ ***até*** *Hor. de Controle* **faça**
    $\Delta u[j][0] \mathrel{-}= H^{-1}[j][i] \times (\text{aux}_R[i][0] + b[i][0])$;

3.2 Geração automática de código

As dimensões dos vetores e das matrizes que serão implementados no microcontrolador podem variar bastante para diferentes modelos de processos, número de entradas e saídas, e possíveis valores diferentes nas escolhas dos horizontes. Portanto, os espaços de memória reservados para as variáveis geralmente são definidos com base nos valores máximos e na maior parte dos casos são subutilizados. Para evitar esse problema, foi proposto e implementado um gerador automático do código do microcontrolador, que ainda auxilia o projetista na sintonização de controle. Dessa forma, o código gerado é produzido de acordo com os parâmetros do problema específico, ocupando um espaço reduzido de memória. Para essa implementação foi utilizado o software MATLAB®, entretanto, qualquer linguagem de alto nível poderia ser utilizada, assim como o microcontrolador, que também pode ser escolhido pelo projetista, desde que as funções e registradores sejam adaptados para outros correspondentes. Dessa forma, o *software* de geração automática de código pode ser dividido em duas partes. A primeira parte, calcula todos os itens que montam o problema de controle do GPC, partindo da função de transferência que modela a planta, até chegar ao gráfico da resposta do sistema em malha fechada com o controlador simulado. Deste modo, é possível realizar a simulação do sistema rapidamente, avaliando as mudanças de comportamento através da alteração de parâmetros do controle, além de auxiliar na correção de eventuais erros de modelagem. A segunda parte do código é responsável por gerar o *firmware* para o sistema embarcado, com suas funções necessárias, incluindo bibliotecas e funções intermediárias, com todas as variáveis declaradas para as dimensões adequadas ao problema em questão, de forma a minimizar o espaço de memória requerido para a implementação. Além disso, o código possibilita acionar degraus de tensão enquanto cada cálculo é realizado, de modo que se torne possível, com o auxílio de um instrumento de medição de sinais elétricos, a aferição do tempo de execução de determinada seção do código. O algoritmo é totalmente generalizado, e possibilita que o usuário possa gerar em poucos minutos um código para implementação embarcada do seu controlador GPC de cômputo rápido.

3.3 Representação numérica

Para aplicações embarcadas, é comum a implementação da aritmética de ponto fixo, tanto para minimizar o tempo de cômputo, quanto por conta da limitação do *hardware*. Esta técnica matemática consiste em escalonar números reais por um mesmo fator, possibilitando a representação de números fracionários por números inteiros. No meio computacional, é comum a utilização de números denominados por "potências de 2" como fator de escala, pois isso facilita algumas operações na representação binária.

Quando se utiliza a aritmética de ponto fixo, alguns parâmetros devem ser definidos para a execução dos cálculos. Entre eles, está o tamanho da palavra utilizada, ou seja, quantos bits estarão disponíveis para armazenar o número no total. Além disso, deve-se definir se o número será representado com sinal ou não, pois caso seja necessário representar números com alternância de sinal, é necessário reservar um bit para armazenar o sinal. A última definição é selecionar, dentro dos bits restantes, quantos estarão alocados para a parte inteira e quantos serão utilizados na parte fracionária.

Por exemplo, ao utilizar um número de 8 bits, com sinal e 4 bits para a parte fracionária, é definido que: 1 bit é reservado para o sinal e 7 bits para representação numérica, sendo 3 bits para a parte inteira e 4 para a parte fracionária. Com essa escolha, podem ser representados valores no intervalo $[-8; +7{,}9375]$ com uma resolução de 0,0625. Por fim, os valores são codificados em números inteiros no intervalo $[-128; +127]$.

Neste trabalho, esta aritmética foi utilizada para minimizar o tempo de cômputo, uma vez que as operações matemáticas são feitas entre números inteiros, o que é relativamente mais rápido que operar com números do tipo ponto flutuante. Também foi avaliado se a perda de resolução correspondente ao arredondamento para a utilização da técnica interferiria na resposta, pois mesmo na utilização de números em ponto flutuante, existem perdas em relação aos valores reais, uma vez que as entradas e saídas analógicas também possuem limitações referentes à resolução.

3.4 Aceleração do cômputo

Mesmo com a utilização de ponto fixo, durante os testes com o algoritmo GPCGPAD, foi observado que algumas operações ou seções do código ainda poderiam ser melhoradas do ponto de vista da implementação embarcada, mantendo as propriedades e qualidades do controle, e diminuindo o tempo de execução médio do código.

É importante destacar que para garantir a funcionalidade genérica das melhorias planejadas, assume-se que as plantas podem ser normalizadas para sinais de entrada e saída variando de 0 a 1, onde estes valores significam os limites mínimos e máximos do sistema.

A primeira melhoria proposta foi referente ao modo como as operações de ponto fixo são implementadas. Por exemplo, no caso de uma multiplicação com esta aritmética, caso as palavras utilizadas sejam de 16 bits, existirá uma operação intermediária realizando o *cast* das mesmas para palavras de 32 bits, para garantir que não ocorra *overflow* no meio do cálculo. Entretanto, foi observado que em alguns casos particulares, este *cast* não é necessário, e um certo tempo pode ser economizado evitando esta etapa. Neste estudo de caso, foi possível implementar essa simplificação dentro do cálculo do vetor de resposta livre, envolvendo os vetores $B_{flip,n}$ e Δu_{aux}, pois com valores originais bem inferiores a 1, mesmo na multiplicação intermediária não haveria riscos em ocorrer o *overflow*. Entretanto, a diminuição do tempo através desta adaptação não é tão significativa, uma vez que o cálculo do vetor de resposta livre, conforme pode ser visto no Algoritmo 2, é realizado fora do laço interno de otimização do controle.

Uma segunda melhoria proposta, com resultados bem mais impactantes, foi relacionada às multiplicações envolvendo a matriz de restrições R. Para as restrições mais comuns no contexto do GPC, como a limitação do esforço de controle, da taxa de variação do controle e da saída, a matriz é esparsa e constituída por elementos 0 e 1. Dessa forma, é possível transformar as multiplicações em apenas operações de soma e subtração entre os elementos que seriam multiplicados por 1. Um exemplo pode ser visto no Algoritmo 4. Além de realizar operações mais simples e rápidas para o processador, as multiplicações são realizadas para encontrar os valores de $\Delta u(k)$ e s, vetores presentes no laço interno de controle, portanto, a diferença entre a implementação convencional e com esta melhoria se torna maior nos piores casos de controle, nos quais mais de uma repetição é necessária.

Algoritmo 4: $aux_R = R \times \Delta u$

- Cálculo convencional de aux_R:

para $j \leftarrow 0$ ***até*** *(Nº de Restrição)* -1 **faça**
  para $i \leftarrow 0$ ***até*** *(Horiz. de Controle)* -1 **faça**
    $aux_R[j][0] += R[j][i] \times \Delta u(k)[i][0]$

- Cálculo proposto para aux_R:

para $j \leftarrow 0$ ***até*** *(Hor. de Controle 1)* -1 **faça**
  para $i \leftarrow 0$ ***até*** j **faça**
    $aux_R[j][0] += \Delta u(k)[i][0]$
    $aux_R[j + N_{u,1}][0] -= \Delta u(k)[i][0]$
para $j \leftarrow 0$ ***até*** *(Hor. de Controle 1)* -1 **faça**
  para $i \leftarrow 0$ ***até*** j **faça**
    $aux_R[j + N_{rin,1}][0] += \Delta u(k)[iN_{u,1}][0]$
    $aux_R[j + N_{u,1} + N_{rin,1}][0] -= \Delta u(k)[iN_{u,1}][0]$

No algoritmo descrito, $N_{u,1}$ e $N_{rin,1}$ são, respectivamente, o horizonte de controle e o número de restrições para a entrada 1. Percebe-se que nesta implementação, os laços de repetição do tipo "para" são escolhidos para não percorrer todo o vetor, passando apenas por onde os valores de R não seriam nulos. Desta forma, para implementações distintas com outras matrizes de restrições, estes laços devem ser modificados de acordo com a implementação.

4. ESTUDO DE CASO

4.1 Planta utilizada

Visto que o foco deste projeto é a área de controle preditivo, optou-se por uma planta de baixa complexidade, voltando o foco do projeto sobre o controle implementado. Deste modo, visando uma implementação de baixo custo e simplicidade de instrumentação, um circuito elétrico foi projetado para ser utilizado como uma planta MIMO. Esse circuito pode ser visto na Figura 1.

As funções de transferência do circuito são genéricas e também podem descrever o comportamento de diversos sistemas mecânicos, elétricos e similares. Desta forma, podemos dizer que o circuito projetado é um sistema análogo de outras plantas, com uma interface elétrica simples que auxilia a manter o foco no controle, e não se preocupar

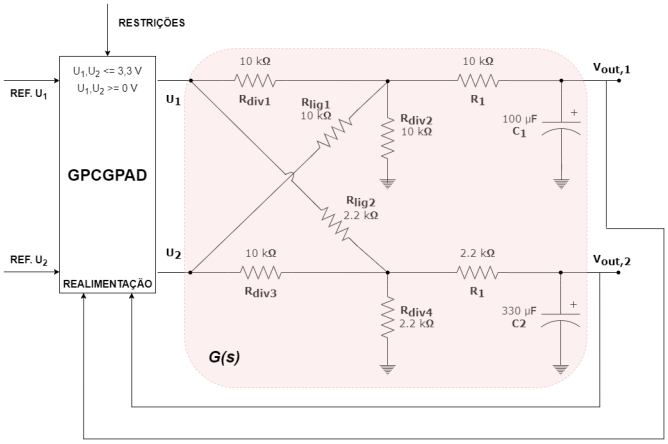

Figura 1. Circuito elétrico projetado.

com problemas mais sofisticados de instrumentação, por exemplo.

Para encontrar o modelo do sistema, por conta das divergências entre os valores teóricos e reais dos componentes, foram aplicados degraus em ambas as entradas, analisando os valores de constante de tempo e ganho estático resultantes. Deste modo, a matriz de funções de transferências do sistema foi encontrada, conforme a seguir:

$$G_s = \begin{bmatrix} \frac{0{,}331}{1{,}26s+1} & \frac{0{,}334}{1{,}31s+1} \\ \frac{0{,}443}{1{,}11s+1} & \frac{0{,}103}{0{,}97s+1} \end{bmatrix}. \quad (5)$$

Para possibilitar a implementação através do microcontrolador, a G_s descrita em (5) foi discretizada com emprego do método "Sustentador de Ordem Zero" (ZOH, do inglês *Zero-Order Hold*), utilizando um período de amostragem (T_s) de 48,5 ms, que satisfaz o critério de ser 20 vezes mais rápido que a constante de tempo mais rápida. Desta forma, a matriz de FT discretizada é dada por:

$$G_d = \begin{bmatrix} \frac{0{,}0125}{z-0{,}9622} & \frac{0{,}01214}{z-0{,}9637} \\ \frac{0{,}01894}{z-0{,}9572} & \frac{0{,}005023}{z-0{,}9512} \end{bmatrix}. \quad (6)$$

4.2 Hardware utilizado

Para realizar os testes descritos a seguir, além da planta projetada, foi utilizada uma Raspberry Pi Pico para armazenar e calcular a lei de controle. A placa é baseada em microcontrolador, com um chip RP2040 e dois núcleos *ARM Cortex-M0+*, com frequências de *clock* configuradas em 133 MHz. Possui 256 kB de memória RAM e 2 MB de memória *Flash onboard*. Dentre suas 36 portas multifuncionais disponíveis, 30 podem gerar um sinal por PWM. Ainda sobre suas GPIOs (do inglês, *General Purpose Input / Output*), são 4 entradas analógicas disponíveis, sendo uma exclusiva ao sensor de temperatura interno da Pico. Uma das principais características da placa está relacionada ao seu ADC, que pode realizar até 500 mil amostras por segundo e possui 12 bits de resolução, que é maior do que o convencional para placas de baixo custo.

4.3 Parâmetros configurados

Para os testes realizados, os horizontes foram selecionados, suficientes para não comprometer a estabilidade do sistema, sendo eles: $N_{U,1} = 3$, $N_{U,2} = 3$, $N_{1,1} = 1$, $N_{1,2} = 1$, $N_{2,1} = 20$, $N_{2,2} = 20$, $N_{rin,1} = 6$ e $N_{rin,2} = 6$. As restrições foram definidas de acordo com a limitação do ADC da Raspberry Pi Pico, sendo a tensão máxima suportada de 3,3 V, e a tensao mínima de 0 V, ambas aplicadas às duas saídas do sistema. Quanto à referência, o valor foi setado como 1,5 V para a saída 1, e 1 V para a saída 2. Deste modo, para o algoritmo "P. F. Proposto", com a planta normalizada, onde 3,3 V passou a valer 1 (adimensional), as referências passaram a ser 0,4545 e 0,303, respectivamente.

A ponderação do erro do sistema manteve-se como $\delta = 1$, para ambas as saídas, enquanto a de esforço de controle, λ, que ficou livre para servir como uma ferramenta de ajuste, foi definida em 10 para as duas entradas. Quanto aos critérios de parada, para ambas as tolerâncias, de factibilidade e de otimalidade, os valores foram definidos como 0,001.

4.4 Testes realizados

A fim de validar as técnicas propostas neste trabalho, três algoritmos foram implementados no circuito projetado. O primeiro foi o GPCGPAD em sua formulação convencional, com aritmética de ponto flutuante. No segundo algoritmo, a única diferença está na aritmética utilizada, que passou a ser a de ponto fixo. Por último, foi implementado um código com aritmética de ponto fixo, entretanto, com as melhorias propostas para redução de tempo de execução. Este algoritmo é denominado neste trabalho como "Ponto Fixo Proposto". Deste modo, a primeira análise foi quanto ao desempenho dos algoritmos analisando as saídas do sistema, mostradas na Figura 2.

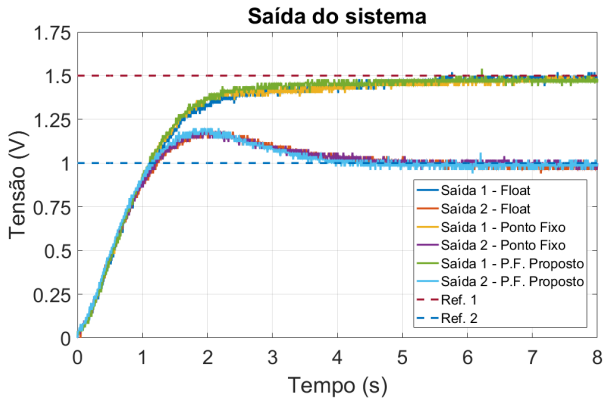

Figura 2. Comparação das respostas reais.

Avaliando a Figura 2, observa-se um comportamento semelhante das saídas do sistema entre as diferentes implementações. Este resultado contribui com a validação das otimizações propostas, uma vez que as alterações não interferiram na resposta dinâmica do sistema de maneira significativa. Além disso, este resultado demonstra que o erro causado pela aritmética de ponto fixo pode ser desconsiderado, pois além do sistema realimentado corrigir o erro gerado, um sistema com representação em ponto flutuante

também tem seus problemas de resolução referente aos periféricos utilizados pelo microcontrolador.

Uma simulação com o modelo dinâmico também foi implementada, e pode ser vista na Figura 3. Esta abordagem permite a validação do modelo calculado com o sistema real por conta da proximidade entre as saídas. Outra vantagem em simular o sistema está na possibilidade de avaliar o comportamento esperado para o sinal de controle, como por exemplo, verificar se o mesmo está trabalhando dentro das restrições impostas pelo projeto. Vale ressaltar que para a simulação, os valores de entrada e saída foram normalizados, ou seja, estão representados de forma adimensional, variando de 0 a 1. Por conta disso, restrições impostas de tensão máxima em 3,3 V no sinal de controle, passam a ser 1.

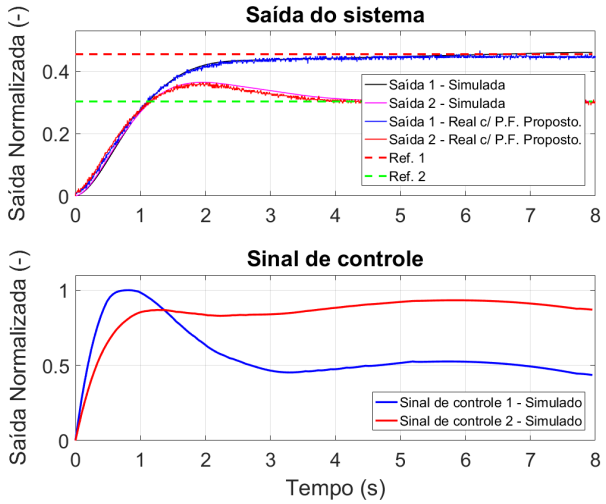

Figura 3. Análise com a simulação.

Além da análise dos valores de entradas e saídas, os tempos médios de execução do algoritmo também foram analisados e são descritos na Tabela 1.

Na Tabela 1, o tempo de interrupção total corresponde ao tempo gasto pelo algoritmo inteiro quando o número de iterações do GPCGPAD é igual a 1. Entretanto, para o cálculo do pior caso de tempo de execução, segue-se a definição mostrada na Figura 4, onde a última parte em verde corresponde às atualizações de entradas e saídas (I/O).

Tabela 1. Descrição dos tempos médios (em microssegundos)

Seção	T.E. Float	T.E. P. Fixo	T.E. P.F. Prop.
y	12	6	6
f	420	229	172
b	416	236	230
r	11	2	2
w	27	11	7
u	196	100	39
s	137	67	11
crit. parada	27	7	7
att. de I/O	10	3	3
GPAD (total)	387	185	64
GPC (total)	869	476	413
interrup. (total)	1256	661	477

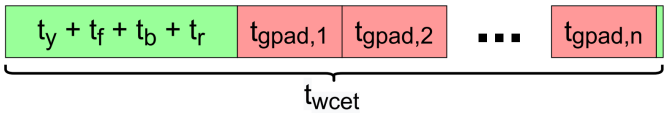

Figura 4. Cálculo do pior caso de tempo de execução observado

Para este estudo de caso, através de simulações, obteve-se 12 iterações (n) do GPAD para o pior caso de tempo de execução observado, com a normalização da planta. Deste modo, os piores tempos de execução observados foram de 5,513 ms para a utilização com ponto flutuante, 2,696 ms com a aritmética de ponto fixo e 1,181 ms com a aritmética de ponto fixo melhorada. Esta diferença pode ser ainda maior, pois com a ampliação do horizonte de controle, por exemplo, o cálculo do vetor Δu aumentaria a diferença entre os tempos consumidos.

A última análise realizada foi sobre a utilização de memória conforme os horizontes de predição e controle se expandiam. Os resultados dessa análise são apresentados na Figura 5.

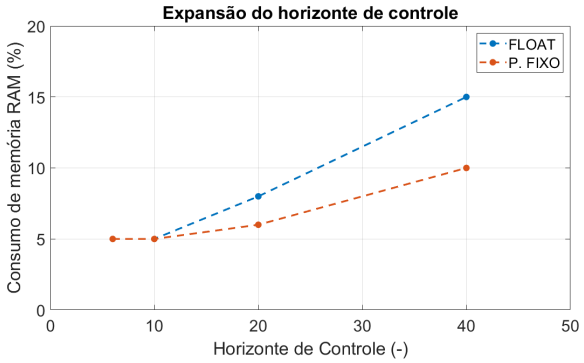

Figura 5. Variação da memória com a expansão de N_U.

Para o consumo de memória Flash, variando ambos os horizontes, o consumo oscilou pouco se mantendo na mesma porcentagem de 2%, independente da técnica de implementação. Para a memória RAM, dentre os aproximadamente 200 kB disponíveis, o programa variou em uma margem satisfatória, comprovando que mesmo para aplicações com horizontes mais amplos, a memória não se tornaria um problema. A variação do horizonte de controle apresentou mais variação no consumo de memória RAM, e a Figura 5 mostra a comparação entre o consumo das aritméticas, com o horizonte de predição fixo em 40 amostras.

Através deste experimento, é perceptível que além da melhoria no tempo de execução, o consumo de memória RAM utilizando ponto fixo é significativamente menor, o que demonstra uma maior aplicabilidade do algoritmo utilizando esta aritmética, uma vez que é possível implementá-lo em sistemas com mais entradas e saídas, além de permitir a escolha de horizontes de controle maiores.

5. CONCLUSÃO

Neste trabalho foi apresentada uma implementação embarcada de um algoritmo GPC de cômputo rápido baseado em GPAD. O algoritmo foi adaptado para ser executado em um microcontrolador, utilizando aritmética de ponto fixo,

e teve como principal contribuição a melhoria de operações do laço interno, que envolviam o vetor de restrições $\bar{r}$. Com a normalização das variáveis do projeto, também foi possível adaptar cálculos com ponto fixo, uma vez que sabia-se o valor máximo de determinadas variáveis, e com isso, podia-se economizar poder computacional para lidar com algumas operações. O resultado obtido foi satisfatório, com o pior caso de tempo de execução observado reduzido para 1,181 ms, para uma planta MIMO, com modelos de primeira ordem. Para uma plataforma baseada em um microcontrolador simples e com resultados rápidos, o conjunto entre o *hardware* e o algoritmo se mostrou viável para diversas utilizações em plantas reais, fazendo com que o GPC passe a se aproximar cada vez mais de aplicações reais com dinâmicas rápidas.

A utilização da aritmética de ponto fixo não comprometeu a resposta do sistema, visto que, em relação à técnica com números em ponto flutuante, tanto as dinâmicas em regime transitório, quanto o valor final em regime permanente, mantiveram valores semelhantes.

Os resultados obtidos possibilitam afirmar que com as melhorias propostas neste presente trabalho, seria possível implementar um controle preditivo que aja diretamente nos parâmetros da planta, e não indiretamente, alterando a referência de um controlador regulatório mais simples, como costuma ser implementado atualmente na indústria petroquímica. Com isso, estima-se ainda uma melhora na confiabilidade do sistema, pois são extintos os problemas relacionados à comunicação entre os controladores.

6. AGRADECIMENTOS

Os autores gostariam de agradecer a Agência Nacional de Petróleo, Gás Natural e Biocombustíveis (ANP), o Programa de Recursos Humanos PRH-2.1 e o Instituto Federal de Santa Catarina pelo apoio financeiro recebido.

REFERÊNCIAS

Camacho, E.F. e Bordons, C. (2004). *Model predictive control*. Advanced textbooks in control and signal processing. Springer, Londres.

Elmorshedy, M.F., Xu, W., El-Sousy, F.F.M., Islam, M.R., e Ahmed, A.A. (2021). Recent Achievements in Model Predictive Control Techniques for Industrial Motor: A Comprehensive State-of-the-Art. 9, 58170–58191. doi: 10.1109/ACCESS.2021.3073020.

Normey-Rico, J.E. e Camacho, E.F. (2007). *Control of dead-time processes*. Advanced textbooks in control and signal processing. Springer, Londres.

Patrinos, P. e Bemporad, A. (2014). An Accelerated Dual Gradient-Projection Algorithm for Embedded Linear Model Predictive Control. *IEEE Transactions on Automatic Control*, 59(1), 18–33. doi:10.1109/TAC.2013.2275667.

Peccin, V.B., Lima, D.M., Flesch, R.C.C., e Normey-Rico, J.E. (2020). Fast Constrained Generalized Predictive Control with ADMM Embedded in an FPGA. *IEEE Latin America Transactions*, 18(02), 422–429. doi:10.1109/TLA.2020.9085299.

Peccin, V.B., Lima, D.M., Flesch, R.C.C., e Normey-Rico, J.E. (2021). Fast algorithms for constrained generalised predictive control with on-line optimisation. *IET Control Theory & Applications*, 15(4), 545–558. doi: 10.1049/cth2.12060.

Rodriguez, J., Kazmierkowski, M.P., Espinoza, J.R., Zanchetta, P., Abu-Rub, H., Young, H.A., e Rojas, C.A. (2013). State of the Art of Finite Control Set Model Predictive Control in Power Electronics. *IEEE Transactions on Industrial Informatics*, 9(2), 1003–1016. doi: 10.1109/TII.2012.2221469.

Sharma, B.L., Khatri, N., e Sharma, A. (2016). An analytical review on FPGA based autonomous flight control system for small UAVs. In *2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT)*, 1369–1372. doi:10.1109/ICEEOT.2016.7754907.

Zanin, A.C. (2001). *Implementação industrial de um otimizador em tempo real*. Tese de doutorado, Escola Politécnica da Universidade de São Paulo, Departamento de Engenharia Química, São Paulo. doi:10.11606/T.3.2001.tde-06082001-150353.