Fast Constrained Generalized Predictive Control with ADMM Embedded in an FPGA

V. Berndsen, D. Martins, R. Costa, and J. Normey

***Abstract*—Constrained model predictive control (MPC) usually requires the computation of a quadratic programming problem (QP) at each sampling instant. This is computationally expensive and becomes a limitation to embed and use MPC in plants with fast sampling rates. Several special solvers for MPC problems have been proposed in the last years, but most of them focus on state-space formulations, which are very popular in academia. This paper proposes a solution based on alternated direction method of multipliers, tailored for embedded systems and applied to generalized predictive control (GPC), which is a very popular formulation in industry. Implementations issues of parallel computation are discussed in order to accelerate the time required for the operations. The implementation in an FPGA proved to be quite fast, with the observed worst case execution time of $11,54\,\mu$s for the presented example. These results contribute to embed GPC applications in processes that are typically controlled by classical controllers because of their fast dynamics.**

***Index Terms*—Alternated direction method of multipliers, Embedded MPC, FPGA application, Fast GPC.**

I. Introdução

COM o advento de melhorias nos métodos e tecnologias de sistemas embarcados, muitas soluções que antes eram restritas às aplicações com computadores poderosos, atualmente se tornam viáveis de serem embarcadas. O controle preditivo baseado em modelo (MPC, do inglês *Model Predictive Control*) é uma delas. É bem difundido que o MPC, em sua formulação convencional, só é aplicável para processos com dinâmica lenta, com período de amostragem da ordem de segundos ou minutos [1]. Essa característica advém do fato de que cada nova ação de controle é obtida por meio do cômputo de um problema de otimização *online* com restrições em um horizonte predefinido. Com os avanços da indústria, é latente a necessidade de controladores avançados nos níveis mais baixos da pirâmide de produção que, consequentemente, devem operar com períodos de amostragem da ordem de mili ou microssegundos em sistemas embarcados. Apesar de ter sido desenvolvido há décadas, o PID ainda é a técnica com mais impacto na indústria, seguida pelo MPC, e grande parte disso se deve a um distanciamento entre a comunidade acadêmica e a aplicação prática das técnicas propostas [2]. Dessa forma, é importante que os algoritmos propostos também levem em consideração características relevantes para a aplicação prática em sistemas embarcados.

Existem diversas formulações de MPC, que se diferenciam pelo modelo utilizado na predição, pelas perturbações consideradas e pela função objetivo [3]. Entre as formulações mais utilizadas na indústria, destaca-se o controle preditivo generalizado (GPC, do inglês *Generalized Predictive Control*) [4], que utiliza um modelo de função de transferência discreta para o cálculo das predições. Outra formulação MPC bastante difundida, em especial na academia, emprega a modelagem em espaço de estados, que permite uma abordagem mais fácil para impor garantias de estabilidade e robustez no sistema em malha fechada. Entretanto, as formulações baseadas em funções de transferência são bastante utilizadas na indústria, uma vez que conceitos como tempo morto, constantes de tempo e ganhos são geralmente mais familiares no contexto industrial do que os conceitos de espaço de estados [4]. Outro ponto fundamental é que o tempo morto, frequente em processos industriais, é mais facilmente representado em funções de transferência do que em outras abordagens [3]. Entretanto, percebe-se que a grande maioria dos trabalhos que exploram algoritmos de MPC de cômputo rápido empregam a formulação em espaço de estados. Dessa forma, há uma lacuna entre os algoritmos do estado da arte de otimização e as formulações de MPC amplamente utilizadas na indústria. Apesar de as formulações poderem ser equivalentes em alguma medida, elas ainda guardam algumas particularidades de implementação, como o número de variáveis para modelar o problema, a esparsidade e dimensão das matrizes, a necessidade de observador de estados etc.

Para computar um problema de programação quadrática com restrições (QP, do inglês *Quadratic Programming*), no contexto de MPC, alguns algoritmos foram propostos de modo a explorar suas particularidades de forma eficiente. Uma abordagem possível é o cômputo *offline* de um conjunto de QPs e posterior armazenamento dos valores obtidos em estruturas do tipo *look-up table*, tal como o MPC explícito apresentado em [5]. Contudo, esse tipo de solução é restrita a problemas de pequena dimensão devido a limitações de memória [6]. Outro tipo de abordagem tem base na exploração da estrutura do problema de otimização resultante. Entre os métodos para o cômputo *online* do QP, os de primeira e segunda ordem aparecem em destaque. Alguns trabalhos propõem métodos de segunda ordem para MPC de cômputo rápido, tais como os baseados no método de conjunto ativo (ver [7], [8]) e no método de ponto interior (ver [1]). Entretanto, os métodos de segunda ordem requerem

V. B. Peccin, R. C. C. Flesch, and J. E. Normey-Rico are with the Department of Automation and Systems, Federal University of Santa Catarina, Florianópolis, Santa Catarina, Brazil e-mail: (vinicius.peccin@ifsc.edu.br; rodolfo.flesch@ufsc.br; julio.normey@ufsc.br).

D. M. Lima is with the Department of Control, Automation and Computing, Federal University of Santa Catarina, Blumenau, Santa Catarina, Brazil e-mail: (daniel.lima@ufsc.br).

This work was supported in part by the National Council for Scientific and Technological Development (CNPq/Brazil), under grants 305785/2015-0 and 309244/2018-8, and in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES/Brazil) under PrInt/UFSC Program.

rotinas relativamente complexas de álgebra linear, que acabam resultando em algoritmos complexos para serem embarcados e que necessitam de uma grande quantidade de memória para armazenar o código resultante [9]. Por outro lado, os métodos de primeira ordem requerem menos recursos computacionais e, dessa forma, têm se tornado uma opção interessante para MPC embarcado de cômputo rápido. Os métodos mais populares de primeira ordem são o método de projeção de gradiente [10],[11] e o método do operador de partição [12], que tem como uma das implementações mais comuns o método dos multiplicadores em direções alternadas (ADMM, do inglês *Alternating Direction Method of Multipliers*).

Para que o MPC embarcado se torne uma opção viável para aplicações industriais, o seu *hardware* necessita ser simples. Os *hardwares* mais comuns apresentados em pesquisas de controladores embarcados são os processadores e os arranjos de portas programáveis em campo (FPGAs, do inglês *Field Programmable Gate Arrays*) [13]. Os FPGAs têm vantagens em termos de velocidade de execução, paralelização e determinismo. As principais soluções com FPGA propõem a implementação do otimizador diretamente na lógica com emprego de alguma linguagem de descrição de *hardware*, como o Verilog e o VHDL. Entretanto, Os FPGAs possuem muitas limitações de recursos. Dessa forma, as aplicações embarcadas devem apresentar boas características tais como: (i) simplicidade para gerar um código passível de ser verificado, validado e certificado e (ii) a limitação no espaço de memória necessário para armazenar os dados que definem o problema e o código que implementa a solução. Outra característica desejada para sistemas de controle embarcados é o determinismo com um pior caso de tempo de execução (WCET, do inglês *Worst Case Execution Time*) previsível, de modo a atender os requisitos *hard* de tempo real.

Baseado nos requisitos apresentados, neste trabalho é proposto um algoritmo chamado GPCADMM. A formulação do algoritmo ADMM para GPC foi previamente apresentada em [14] e neste trabalho é proposta uma versão estendida com a integração do algoritmo do GPC com o ADMM, com a proposição de uma arquitetura de implementação embarcada em FPGA e com a discussão de detalhes que tornam a solução adequada para sistemas com dinâmicas rápidas. Portanto, as principais contribuições deste artigo são:

- a proposição de um algoritmo eficiente para GPC, integrado com o otimizador de QP e dedicado para a implementação embarcada em um hardware paralelo de alta velocidade;
- a proposição de uma arquitetura de implementação para FPGA;
- a discussão dos detalhes de implementação embarcada que tornam a solução adequada para sistemas com dinâmicas rápidas;
- os resultados experimentais em um *hardware* paralelo de alta velocidade.

O restante do artigo está organizado da seguinte forma: a Seção II faz um breve revisão do GPC e a sua generalização para sistemas com múltiplas entradas e múltiplas saídas (MIMO, do inglês *Multiple Input Multiple Output*). A Seção III apresenta uma revisão do ADMM. Na seção IV são apresentados a formulação GPC com a versão acelerada do ADMM e o respectivo algoritmo proposto. Na Seção V, são descritos alguns detalhes de implementação do GPCADMM em arquitetura paralela. Na Seção VI, uma implementação embarcada em FPGA é apresentada e, a partir de um estudo de caso, os requisitos de tempo de computação são discutidos. A Seção VII conclui o artigo.

II. Controle Preditivo Generalizado

Esta seção apresenta a formulação do GPC empregada para derivar o GPCADMM. Inicialmente é apresentado o caso de uma entrada e uma saída (SISO, do inglês *Single Input Single Output*), como proposto em [15], e no fim da seção é apresentada a extensão para o caso MIMO.

Muitos sistemas SISO podem ser modelados pelo chamado modelo autorregressivo com média móvel, integrador e entrada controlada (CARIMA, do inglês *Controlled Autoregressive Integrated Moving Average*), o qual é dado por:

$$A(z^{-1})y(k) = B(z^{-1})z^{-d}u(k-1) + T(z^{-1})\frac{e(k)}{\Delta}, \quad (1)$$

com

$$\begin{aligned} A(z^{-1}) &= 1 + a_1 z^{-1} + a_2 z^{-2} + ... + a_{n_a} z^{-n_a}, \\ B(z^{-1}) &= b_0 + b_1 z^{-1} + b_2 z^{-2} + ... + b_{n_b} z^{-n_b}, \\ \Delta &= 1 - z^{-1}, \end{aligned}$$

onde $u(k)$ é o sinal de controle, $y(k)$ é a saída do sistema, $e(k)$ é um ruído branco de média zero, A e B são polinômios no operador de deslocamento z^{-1} e d é o atraso de transporte. O polinômio Δ é utilizado para uma melhor modelagem de perturbações não estacionárias, que são muito presentes em aplicações industriais [15]. O polinômio $T(z^{-1})$, que é difícil de ser obtido através de dados da planta, pode ser utilizado como um parâmetro de sintonização, pois afeta diretamente a robustez de malha fechada [4]. Uma vez que a escolha de $T(z^{-1})$ não afeta o desenvolvimento proposto neste trabalho, $T(z^{-1}) = 1$ é utilizado, que é uma opção comum na formulação GPC.

A função custo do GPC, para o caso SISO, é dada por:

$$\begin{aligned} J_{GPC} = &\sum_{j=N_1}^{N_2} \delta(j) \left[\hat{y}(k+j|k) - w(k+j)\right]^2 + \\ &\sum_{j=1}^{N_u} \lambda(j) \left[\Delta u(k+j-1)\right]^2, \end{aligned} \quad (2)$$

onde $\hat{y}(k+j|k)$ representa uma predição da saída no instante futuro em tempo discreto $k+j$, determinado em k, $w(k+j)$ é a referência em $k+j$, Δu é o incremento no sinal de controle, $\delta(j)$ e $\lambda(j)$ são as sequências de ponderação no erro e no esforço de controle, respectivamente, $N = N_2 - N_1 + 1$ define o horizonte de predição e N_u é o horizonte de controle.

De modo a obter as predições de saída necessárias para encontrar o sinal de controle ótimo, as saídas futuras do

sistema podem ser divididas em duas parcelas, chamadas de resposta forçada ($\mathbf{G}\Delta\mathbf{u}$) e resposta livre ($\mathbf{f}$):

$$\mathbf{y} = \mathbf{G}\Delta\mathbf{u} + \mathbf{f}, \tag{3}$$

com $\mathbf{y} \in \mathbb{R}^N$, $\Delta\mathbf{u} \in \mathbb{R}^{N_u}$, $\mathbf{G} \in \mathbb{R}^{N \times N_u}$ e $\mathbf{f} \in \mathbb{R}^N$ dados por:

$$\mathbf{y} = \begin{bmatrix} \hat{y}(k+N_1|k) \\ \hat{y}(k+N_1+1|k) \\ \vdots \\ \hat{y}(k+N_2|k) \end{bmatrix}, \qquad \Delta\mathbf{u} = \begin{bmatrix} \Delta u(k) \\ \Delta u(k+1) \\ \vdots \\ \Delta u(k+N_u-1) \end{bmatrix},$$

$$\mathbf{G} = \begin{bmatrix} g_{N_1} & g_{N_1-1} & \cdots & g_{N_1-N_u+1} \\ g_{N_1+1} & g_{N_1} & \cdots & g_{N_1-N_u+2} \\ \vdots & \vdots & \vdots & \vdots \\ g_{N_2} & g_{N_2-1} & \cdots & g_{N_2-N_u+1} \end{bmatrix}, \ \mathbf{f} = \begin{bmatrix} f_{N_1} \\ f_{N_1+1} \\ \vdots \\ f_{N_2} \end{bmatrix}.$$

Para a aplicação em algoritmos, $\mathbf{G}$ e $\mathbf{f}$ podem ser diretamente obtidos de forma recursiva [4, p. 51, pp. 56-57] tal que:

$$\begin{aligned} g_i &= z(1 - A(z^{-1}))g_{i-1} + B(z^{-1})\bar{u}(i), \\ f_i &= z(1 - \tilde{A}(z^{-1}))f_{i-1} + B(z^{-1})\Delta u(k-d+i-1), \end{aligned} \tag{4}$$

com $i = 1, ..., N_2$, $\bar{u}(i)$ sendo um degrau unitário, $g_i = 0$ para $i \leq 0$, $f_i = y(k+i)$ para $i \leq 0$ e $\Delta u(k+j) = 0$ para $j \geq 0$.

O próximo passo consiste em aplicar as sequências de predição de saída (3) na função custo do GPC (2), que pode ser reescrita como:

$$J_{GPC} = \frac{1}{2}\Delta\mathbf{u}^T\mathbf{H}\Delta\mathbf{u} + \mathbf{b}^T\Delta\mathbf{u} + f_0, \tag{5}$$

onde

$$\begin{aligned} \mathbf{H} &= 2(\mathbf{G}^T\mathbf{Q}_\delta\mathbf{G} + \mathbf{Q}_\lambda), \\ \mathbf{b}^T &= 2(\mathbf{f} - \mathbf{w})^T\mathbf{Q}_\delta\mathbf{G}, \\ f_0 &= (\mathbf{f} - \mathbf{w})^T(\mathbf{f} - \mathbf{w}), \end{aligned}$$

$\mathbf{Q}_\delta$ e $\mathbf{Q}_\lambda$ são matrizes de ponderação bloco diagonais positivas definidas e o vetor de trajetórias de referência $\mathbf{w}$ segue a mesma estrutura de $\mathbf{f}$.

Por fim, o vetor de incrementos de controle do GPC $\Delta\mathbf{u}$ pode ser obtido, em cada iteração de controle, a partir do QP:

$$\begin{aligned} \min_{\Delta\mathbf{u}} \quad & f(\Delta\mathbf{u}) = \frac{1}{2}\Delta\mathbf{u}^T\mathbf{H}\Delta\mathbf{u} + \mathbf{b}^T\Delta\mathbf{u} \\ \text{s.a.} \quad & \bar{\mathbf{R}}\Delta\mathbf{u} \leq \bar{\mathbf{r}}, \end{aligned} \tag{6}$$

com a matriz de restrições $\bar{\mathbf{R}} \in \mathbb{R}^{n_{rin} \times N_u}$, o vetor $\bar{\mathbf{r}} \in \mathbb{R}^{n_{rin}}$ e n_{rin} representando a quantidade de restrições de desigualdade. A forma concisa de representar as restrições $\bar{\mathbf{R}}\Delta\mathbf{u} \leq \bar{\mathbf{r}}$ é genérica e permite a representação de muitos tipos de restrição, tais como de magnitude, taxa de variação e sobressinal, tanto nas variáveis de saída quanto de entrada [4].

A generalização do GPC para sistemas MIMO é direta. A matriz de transferência genérica para um sistema MIMO de n_i entradas e n_o saídas pode ser representada como:

$$\mathbf{M}_t = \begin{bmatrix} \dfrac{N_{11}(z^{-1})}{D_{11}(z^{-1})}z^{-d_{11}} & \cdots & \dfrac{N_{1n_i}(z^{-1})}{D_{1n_i}(z^{-1})}z^{-d_{1n_i}} \\ \vdots & \ddots & \vdots \\ \dfrac{N_{n_o1}(z^{-1})}{D_{n_o1}(z^{-1})}z^{-d_{n_o1}} & \cdots & \dfrac{N_{n_on_i}(z^{-1})}{D_{n_on_i}(z^{-1})}z^{-d_{n_on_i}} \end{bmatrix},$$

onde $\mathbf{M}_t \in \mathbb{R}^{n_o \times n_i}$ e $N_{pl}(z^{-1})$, $D_{pl}(z^{-1})$ e d_{pl} representam o numerador, denominador e atraso de transporte da função de transferência que relaciona a saída p com a entrada l, respectivamente.

A matriz $\mathbf{M}_t$ pode ser decomposta em uma representação chamada de descrição matricial fracionária (MFD, do inglês *Matrix Fraction Description*), como segue:

$$\mathbf{M}_t = \mathbf{D}(z^{-1})\mathbf{A}(z^{-1})^{-1}\mathbf{B}(z^{-1}),$$

com $\mathbf{A}(z^{-1}) \in \mathbb{R}^{n_o \times n_o}$ e $\mathbf{B}(z^{-1}) \in \mathbb{R}^{n_o \times n_i}$ representando matrizes polinomiais. $\mathbf{D}(z^{-1}) \in \mathbb{R}^{n_o \times n_o}$ é uma matriz diagonal com cada elemento representando o menor atraso de transporte entre as entradas relacionadas com a saída correspondente. A matriz $\mathbf{A}(z^{-1})$ é uma matriz diagonal com os elementos $pl : p = l$ sendo o mínimo múltiplo comum entre os denominadores da linha p de $\mathbf{M}_t$. A matriz $\mathbf{B}(z^{-1})$ pode ser calculada como $\mathbf{B}(z^{-1}) = \mathbf{A}(z^{-1})\mathbf{M}_t$.

Para representar n_o saídas, os vetores associados com cada saída podem ser agrupados no vetor $\mathbf{y} \in \mathbb{R}^{\sum_{p=1}^{n_o} N_2^{(p)} - N_1^{(p)} + 1}$ e os n_i sinais de controle no vetor $\mathbf{u} \in \mathbb{R}^{\sum_{l=1}^{n_i} N_u^{(l)}}$, dados por:

$$\mathbf{y} = [\mathbf{y}^{(1)} \ \mathbf{y}^{(2)} \ ... \ \mathbf{y}^{(n_o)}]^T, \quad \mathbf{u} = [\mathbf{u}^{(1)} \ \mathbf{u}^{(2)} \ ... \ \mathbf{u}^{(n_i)}]^T. \tag{7}$$

A partir das representações das matrizes polinomiais $\mathbf{A}(z^{-1})$ e $\mathbf{B}(z^{-1})$ e dos vetores generalizados para o caso MIMO, $\mathbf{y}$ e $\mathbf{u}$, é então possível generalizar a representação do modelo CARIMA para sistemas MIMO:

$$\mathbf{A}(z^{-1})\mathbf{y}(k) = \mathbf{B}(z^{-1})\mathbf{D}(z^{-1})\mathbf{u}(k-1) + \mathbf{C}(z^{-1})\frac{e(k)}{\Delta}. \tag{8}$$

Com o modelo CARIMA generalizado, o modelo de predição segue os mesmos passos desenvolvidos para o caso SISO e obtém-se a mesma forma de (3). No caso MIMO, a matriz $\mathbf{G}$ é formada por outras matrizes $\mathbf{G}_{pl}$, onde $\mathbf{G}_{pl}$ é uma matriz de coeficientes da resposta ao degrau da saída p em relação à entrada l. Portanto, a matriz $\mathbf{G}$ é dada por:

$$\mathbf{G} = \begin{bmatrix} \mathbf{G}_{11} & \cdots & \mathbf{G}_{1n_i} \\ \mathbf{G}_{21} & \cdots & \mathbf{G}_{2n_i} \\ \vdots & \ddots & \vdots \\ \mathbf{G}_{n_o1} & \cdots & \mathbf{G}_{n_on_i} \end{bmatrix}.$$

Para se obter o vetor de resposta livre generalizado $\mathbf{f}$, as respostas livres para cada saída do sistema podem ser agrupadas como:

$$\mathbf{f} = \left[\mathbf{f}^{(1)} \ \mathbf{f}^{(2)} \ \ldots \ \mathbf{f}^{(n_o)}\right]^T,$$

onde $\mathbf{f}^{(i)}$ tem a mesma forma de (4). Portanto, o vetor de resposta livre para cada saída p pode ser obtido recursivamente como:

$$\begin{aligned} f_i^{(p)} :=& z(1 - \tilde{\mathbf{A}}_{pp}(z^{-1}))f_{i-1}^{(p)} + \\ &+ \sum_{l=1}^{n_i} \mathbf{B}_{pl}(z^{-1})\Delta u^{(l)}(k - \mathbf{D}_{pp} + i - 1), \end{aligned} \tag{9}$$

com $i = 1, ..., N_2^{(p)}$, $p = 1, ..., n_o$, $l = 1, ..., n_i$, $f_i^{(p)} = y^{(p)}(k+i)$ para $i \leq 0$ e $\Delta u^{(l)}(k+j) = 0$ para $j \geq 0$.

Seguindo a mesma lógica para se obter a lei de controle para o caso SISO e generalizando $\mathbf{w}$ a partir do agrupamento

das referências futuras para cada saída predita similarmente a (7), o problema de otimização para o caso MIMO pode ser representado da mesma forma padrão de (6).

III. Método dos Multiplicadores em Direções Alternadas

O ADMM é um algoritmo simples, mas poderoso, que se adapta muito bem para problemas de otimização convexa. Ele se apresenta na forma de um algoritmo de decomposição e coordenação, no qual as soluções de pequenos subproblemas locais são coordenadas para encontrar uma solução para um problema global de grande escala [12]. ADMM pode ser visto como uma tentativa de colher os benefícios de ambos, a decomposição dual e o método de Lagrange aumentado, para otimização com restrições [16].

O ADMM foi desenvolvido para solucionar problemas na seguinte forma:

$$\begin{aligned} \min_{\mathbf{x},\mathbf{y}} \quad & h(\mathbf{x}) + g(\mathbf{z}) \\ \text{s.a.} \quad & \bar{\mathbf{C}}\mathbf{x} + \bar{\mathbf{D}}\mathbf{z} = \bar{\mathbf{c}}, \end{aligned} \tag{10}$$

com $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{z} \in \mathbb{R}^m$, $\bar{\mathbf{C}} \in \mathbb{R}^{n_{req} \times n}$, $\bar{\mathbf{D}} \in \mathbb{R}^{n_{req} \times m}$, $\bar{\mathbf{c}} \in \mathbb{R}^{n_{req}}$. As funções h e g são convexas.

Para resolver o problema (10), o ADMM pode ser computado de forma iterativa, alternando entre as variáveis $\mathbf{x}$ e $\mathbf{z}$ e atualizando os multiplicadores de Lagrange associados às restrições de igualdade, ν:

$$\mathbf{x}_{j+1} := \operatorname{argmin}\ \mathcal{L}_\rho(\mathbf{x}_j, \mathbf{z}_j, \nu_j), \tag{11}$$

$$\mathbf{z}_{j+1} := \operatorname{argmin}\ \mathcal{L}_\rho(\mathbf{x}_{j+1}, \mathbf{z}_j, \nu_j), \tag{12}$$

$$\nu_{j+1} := \nu_j + \rho(\bar{\mathbf{C}}\mathbf{x}_{j+1} + \bar{\mathbf{D}}\mathbf{z}_{j+1} - \bar{\mathbf{c}}), \tag{13}$$

onde $\mathcal{L}_\rho$ é a função lagrangiana aumentada. O primeiro e segundo passos são equivalentes à minimização do lagrangiano aumentado em relação às variáveis $\mathbf{x}$ e $\mathbf{z}$, respectivamente. O uso da versão aumentada do lagrangiano resulta em uma convergência mais rápida devido a uma melhor regularização do problema através do termo quadrático [17]. O último passo é uma atualização do multiplicador dual que garante a convergência do algoritmo a partir da imposição de consenso das sequências de atualizações $\{\bar{\mathbf{C}}\mathbf{x}_j - \bar{\mathbf{c}}\}$ com $\{\bar{\mathbf{D}}\mathbf{z}_j\}$.

O ADMM pode ser representado de uma forma mais compacta e que pode ser mais conveniente para a implementação. Essa forma é chamada de escalonada e pode ser obtida através da definição de uma nova variável $\phi_j = (\frac{1}{\rho})\nu_j$, denominada variável escalonada dual [16]. Assim, o ADMM escalonado pode ser computado iterativamente da seguinte forma:

$$\begin{aligned} \mathbf{x}_{j+1} &:= \operatorname{argmin}\left(h(\mathbf{x}_j) + \frac{\rho}{2}\|\bar{\mathbf{C}}\mathbf{x}_j + \bar{\mathbf{D}}\mathbf{y}_j - \bar{\mathbf{c}} + \phi_j\|_2^2\right), \\ \mathbf{z}_{j+1} &:= \operatorname{argmin}\left(g(\mathbf{y}_j) + \frac{\rho}{2}\|\bar{\mathbf{C}}\mathbf{x}_{j+1} + \bar{\mathbf{D}}\mathbf{y}_j - \bar{\mathbf{c}} + \phi_j\|_2^2\right), \\ \phi_{j+1} &:= \phi_j + (\bar{\mathbf{C}}\mathbf{x}_{j+1} + \bar{\mathbf{D}}\mathbf{y}_{j+1} - \bar{\mathbf{c}}). \end{aligned} \tag{14}$$

IV. Algoritmo GPCADMM

A partir do QP do GPC (6), o primeiro passo proposto é representar as restrições por meio de uma função indicadora $I_{\_}(\mathbf{x})$. Pode-se definir $f(\Delta\mathbf{u}) = \frac{1}{2}\Delta\mathbf{u}^T\mathbf{H}\Delta\mathbf{u} + \mathbf{b}^T\Delta\mathbf{u}$ e reescrever o problema rearranjando as restrições com uso da função indicadora $I_{\_}(x)$ na função objetivo:

$$\min_{\Delta\mathbf{u}} \quad f(\Delta\mathbf{u}) + I_{\_}(\bar{\mathbf{R}}\Delta\mathbf{u} - \bar{\mathbf{r}}). \tag{15}$$

Para se obter a forma padrão do ADMM (10), pode-se definir uma nova variável $\mathbf{z} = \bar{\mathbf{R}}\Delta\mathbf{u} - \bar{\mathbf{r}}$ para a segunda equação e inserir uma restrição de igualdade para manter o acoplamento entre as variáveis:

$$\begin{aligned} \min_{\Delta\mathbf{u},\mathbf{z}} \quad & f(\Delta\mathbf{u}) + I_{\_}(\mathbf{z}) \\ \text{s.a.} \quad & \bar{\mathbf{R}}\Delta\mathbf{u} - \mathbf{z} = \bar{\mathbf{r}}. \end{aligned} \tag{16}$$

A partir da forma padrão (16), o lagrangiano aumentado $\mathcal{L}_\rho$ pode ser definido como:

$$\begin{aligned} \mathcal{L}_\rho(\Delta\mathbf{u}, \mathbf{z}, \nu) = f(\Delta\mathbf{u}) + I_{\_}(\mathbf{z}) + \nu^T(\bar{\mathbf{R}}\Delta\mathbf{u} - \mathbf{z} - \bar{\mathbf{r}}) + \\ + \frac{\rho}{2}\|\bar{\mathbf{R}}\Delta\mathbf{u} - \mathbf{z} - \bar{\mathbf{r}}\|^2. \end{aligned}$$

Com o lagrangiano aumentado e os passos básicos do ADMM escalonado (14), o GPCADMM pode ser computado:

$$\Delta\mathbf{u}_{j+1} := \operatorname{argmin}\left(f(\Delta\mathbf{u}_j) + \frac{\rho}{2}\|\bar{\mathbf{R}}\Delta\mathbf{u} - \mathbf{z}_j - \bar{\mathbf{r}} + \phi_j\|^2\right), \tag{17}$$

$$\mathbf{z}_{j+1} := \operatorname{argmin}\left(I_{\_}(\mathbf{z}) + \frac{\rho}{2}\|\bar{\mathbf{R}}\Delta\mathbf{u}_{j+1} - \mathbf{z}_j - \bar{\mathbf{r}} + \phi_j\|^2\right), \tag{18}$$

$$\phi_{j+1} := \phi_j + (\bar{\mathbf{R}}\Delta\mathbf{u}_{j+1} - \mathbf{z}_{j+1} - \bar{\mathbf{r}}). \tag{19}$$

Para computar o passo da equação (17) pode-se abrir o termo da norma e agrupar da seguinte forma:

$$\begin{aligned} \Delta\mathbf{u}_{j+1} := \operatorname{argmin}\Bigg(\frac{1}{2}\Delta\mathbf{u}^T(\mathbf{H} + \bar{\mathbf{R}}^T\bar{\mathbf{R}})\Delta\mathbf{u} + \\ + (\mathbf{b}^T + \bar{\mathbf{R}}^T(\phi - \mathbf{z} - \bar{\mathbf{r}}))\Delta\mathbf{u}\Bigg). \end{aligned}$$

Portanto, se forem definidos $\tilde{\mathbf{H}} = \mathbf{H} + \bar{\mathbf{R}}^T\bar{\mathbf{R}}$ e $\tilde{\mathbf{b}} = \mathbf{b}^T + \bar{\mathbf{R}}^T(\phi - \mathbf{z} - \bar{\mathbf{r}}))$ recai-se num problema padrão de programação quadrática sem restrições:

$$\min_{\Delta\mathbf{u}} \quad \frac{1}{2}\Delta\mathbf{u}^T\tilde{\mathbf{H}}\Delta\mathbf{u} + \tilde{\mathbf{b}}^T\Delta\mathbf{u}, \tag{20}$$

que possui solução analítica:

$$\Delta\mathbf{u}_{j+1} := -\tilde{\mathbf{H}}^{-1}\tilde{\mathbf{b}}.$$

Como em um problema típico de GPC a matriz $\tilde{\mathbf{H}}$ pode ser computada *offline*, então a sua inversa pode ser armazenada *offline* e a solução desse passo pode ser computada apenas com uma multiplicação matricial.

Para computar o passo da equação (18), pode-se verificar que ela pode ser reescrita através do operador de proximidade [17] da seguinte forma:

$$\mathbf{z}_{j+1} := \operatorname{prox}_{I_{\_},\rho}(\bar{\mathbf{R}}\Delta\mathbf{u}_{j+1} - \bar{\mathbf{r}} + \phi_j). \tag{21}$$

Dado que $I_{\_}(\mathbf{z})$ é uma função indicadora do ortante negativo $\mathbb{R}^n_-$, o qual é um conjunto convexo fechado e não vazio, o operador de proximidade torna-se uma projeção em $\mathbb{R}^n_-$ [18]. Dessa forma, (21) pode ser obtida a partir da seleção dos elementos negativos de cada elemento ou zero:

$$\mathbf{z}_{j+1} := \min(\bar{\mathbf{R}}\Delta\mathbf{u}_{j+1} - \bar{\mathbf{r}} + \phi_j, \mathbf{0}).$$

Os resíduos das condições de otimalidade podem ser usados como critério de parada. O resíduo primal $\mathbf{r}$ e o resíduo dual $\mathbf{s}$ são dados por:

$$\mathbf{r}_{j+1} := \bar{\mathbf{R}}\Delta\mathbf{u}_{j+1} - \mathbf{z}_{j+1} - \bar{\mathbf{r}},\ \mathbf{s}_{j+1} := \rho\bar{\mathbf{R}}^T(\mathbf{z}_{j+1} - \mathbf{z}_j). \quad (22)$$

Portanto, um critério de parada razoável é que os resíduos devem ser menores que um valor de tolerância desejado:

$$\|\mathbf{r}_j\| \leq \epsilon_p \quad \text{e} \quad \|\mathbf{s}_j\| \leq \epsilon_d. \quad (23)$$

A partir dos desenvolvimentos apresentados, pode-se resumir o GPCADMM, para o caso SISO, no Algoritmo 1. A generalização do algoritmo para o caso MIMO é direta, como foi apresentada na Seção II.

Algoritmo 1: GPCADMM

Entradas: A, B, d, $\bar{\mathbf{R}}$, $\bar{\mathbf{r}}$
Saída: $u(k)$
Dados: λ, δ, N_u, N_1, N_2, $\rho > 0$, $\phi_0 = \mathbf{0}$, ϵ_p, ϵ_d, j_{max}
início
  $\tilde{A}(z^{-1}) = (1 - z^{-1})A(z^{-1})$;
  para $i = 1 : N_2$ **faça**
    $g_i := z(1 - A(z^{-1}))g_{i-1} + B(z^{-1})\bar{u}_i$;
  para $i = 1 : N_u$ **faça**
    $\mathbf{G}_{i:N,i} = \mathbf{g}_{N_1:N_2+1-i}$;
  $\mathbf{H} = 2(\mathbf{G}^T\mathbf{G} + \lambda\mathbf{I})$;
  $\tilde{\mathbf{H}} = \mathbf{H} + \bar{\mathbf{R}}^T\rho\bar{\mathbf{R}}$; $\breve{\mathbf{H}} = \tilde{\mathbf{H}}^{-1}$;
  $k = 0$;
  enquanto *modo automático* **faça**
    se *tempo de amostragem* **então**
      Adquire saída $y(k)$ e referências futuras;
      $f_0 = y(k)$;
      para $i = 1 : N_2$ **faça**
        $f_i := z(1 - \tilde{A}(z^{-1}))f_{i-1} +$
        $B(z^{-1})\Delta u(k - d + i - 1)$;
      $\mathbf{b}^T = 2(\mathbf{f}_{N_1:N_2} - \mathbf{w}_{N_1:N_2})^T\mathbf{G}$;
      $\mathbf{z}_0 = \bar{\mathbf{R}}\Delta\mathbf{u}_0 - \bar{\mathbf{r}}$;
      $j = 0$;
      enquanto $j < j_{max}$ **faça**
        $\tilde{\mathbf{b}} := \mathbf{b}^T + \rho\bar{\mathbf{R}}^T(\phi_j - \mathbf{z}_j - \bar{\mathbf{r}})$;
        $\Delta\mathbf{u} := -\breve{\mathbf{H}}\tilde{\mathbf{b}}$;
        $\mathbf{z}_{j+1} := \min(\bar{\mathbf{R}}\Delta\mathbf{u} - \bar{\mathbf{r}} + \phi_j, \mathbf{0})$;
        $\phi_{j+1} := \phi_j + (\bar{\mathbf{R}}\Delta\mathbf{u} - \mathbf{z}_{j+1} - \bar{\mathbf{r}})$;
        se $\|\bar{\mathbf{R}}\Delta\mathbf{u} - \mathbf{z}_{j+1} - \bar{\mathbf{r}}\|_2 \leq \epsilon_p$ **e**
        $\|\rho\bar{\mathbf{R}}^T(\mathbf{z}_{j+1} - \mathbf{z}_j)\|_2 \leq \epsilon_d$ **então**
          retorna;
        $j = j + 1$;
      $\Delta u(k) = [1 \quad 0 \quad \cdots \quad 0]_{1\times N_u}\Delta\mathbf{u}$;
      $u(k) = u(k-1) + \Delta u(k)$;
      $k = k + 1$;
fim

V. Detalhes da implementação embarcada

Esta seção descreve os detalhes da implementação para possibilitar embarcar o algoritmo em arquitetura paralela, os quais contribuem para um ganho de velocidade no tempo de computação e pouco uso de memória.

A. Aritmética de Ponto Fixo

A representação numérica geralmente é feita escolhendo-se entre a aritmética de ponto fixo e a aritmética de ponto flutuante. A aritmética de ponto fixo tem vantagens para sistemas embarcados, tais como introduzir um menor atraso computacional e ocupar menos espaço de memória em relação à de ponto flutuante. O pequeno atraso nas operações de aritmética de ponto fixo permite que as implementações atinjam tempos de execução menores que os que seriam alcançados com uma representação em ponto flutuante. Por outro lado, a aritmética de ponto fixo possui uma faixa dinâmica mais limitada. Dessa forma, a opção pela aritmética de ponto fixo requer uma análise numérica cuidadosa para a determinação do número de bits para representar a parte inteira e a parte fracionária tais que *overflows* e o acúmulo de erros de arredondamento mantenham-se suficientemente pequenos [13].

A implementação da aritmética de ponto fixo é adequada para a utilização no algoritmo GPCADMM graças a algumas características. A primeira delas é que, em aplicações de controle, o modelo da planta e suas variáveis de entrada e saída podem ser normalizados. Além disso, as limitações físicas como a resolução dos transdutores e conversores analógico-digital (A/D) e os limites dos atuadores fazem com que a faixa dinâmica necessária seja pequena e, nesse caso, uma representação com poucos bits é suficiente. A segunda característica é que o algoritmo proposto não requer operações de divisão e, dessa forma, o problema de erro relativo grande, comum na representação de números pequenos em aritmética de ponto fixo, não é amplificado.

B. Multiplicação Matricial Paralela

Uma das operações mais custosas do ponto de vista computacional, no contexto dos algoritmos propostos, é a multiplicação matricial. Algoritmos clássicos de multiplicação de matrizes densas, para um processador simples, são computados dentro de três laços de repetição aninhados, com n repetições cada. Tais algoritmos possuem uma complexidade $O(n^3)$, o que representa um crescimento acentuado do número de operações com a dimensão da matriz [19].

O uso de computação paralela pode diminuir consideravelmente o tempo necessário para essa operação. Existem formas bastante sofisticadas de implementação paralela, como a utilização da arquitetura sistólica, que se propõe a otimizar três métricas, a velocidade, a área ocupada e o consumo de energia [20], [21]. Neste trabalho é proposta uma forma de implementação mais simples, focada na métrica de velocidade.

A ideia geral é agrupar as operações independentes, executá-las de forma paralela e repassar os resultados intermediários de forma sequencial por meio de registradores, formando assim uma estrutura de *pipeline*. Um esquemático da proposta de arquitetura pode ser visto na Fig. 1. Neste exemplo, é apresentado o cômputo de um elemento de uma matriz de ordem $n = 10$ com um bloco somador de 10 elementos. Como cada elemento pode também ser computado em paralelo, a

multiplicação entre duas matrizes 10×10 apresenta uma latência de 2 pulsos de *clock* (para uma aritmética de ponto fixo). O algoritmo paralelo proposto apresenta uma propriedade de complexidade dada por $O(\lceil \log_{10}(n) \rceil + 1)$. Para comparação, uma multiplicação entre duas matrizes 10×10 no algoritmo paralelo necessita de 2 pulsos de *clock*, enquanto o algoritmo sequencial requer $1\,000$ pulsos.

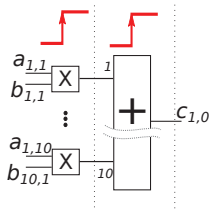

Fig. 1. Exemplo de multiplicação matricial paralela entre duas matrizes 10×10 em apenas 2 pulsos de *clock*.

C. Arquitetura de Implementação do ADMM

De modo a implementar o algoritmo do ADMM em arquitetura paralela, foi proposta uma solução a partir de uma máquina de estados finita. A solução foi centrada no controle sequencial das ações através da máquina de estados, de modo que os valores intermediários de cômputo fossem coordenados enquanto as operações de soma, subtração e multiplicação matriciais executam paralelamente. Na Fig. 2 são representadas de forma esquemática a implementação em máquina de estados e as operações correspondentes. No estado 11, o critério de parada é verificado e, em caso positivo, após o passo 12, o processo é então terminado, sinalizando o incremento de controle encontrado. Dessa forma, pode-se fazer uma estimativa do tempo de execução do algoritmo a partir da seguinte equação:

$$t = T_c(2+n_{iter}(8+1(\lceil \log_{10}(N_u) \rceil+1)+3(\lceil \log_{10}(n_{rin}) \rceil+1))),$$

onde T_c é o período de *clock* utilizado pelo *hardware* e n_{iter} é o número de iterações necessários pelo algoritmo atingir o critério de parada.

Para ilustrar a utilização, toma-se um cenário com um horizonte de controle $N_u = 5$ e com restrições de limite superior e inferior no incremento de controle $n_{rin} = 10$. Neste cenário obtém-se um total de 18 pulsos de *clock* para uma iteração completa. Para um *hardware* com frequência de *clock* de 50 MHz, o período é $T_c = 0,02\,\mu$s e portanto o tempo total de uma iteração é de $0,36\,\mu$s

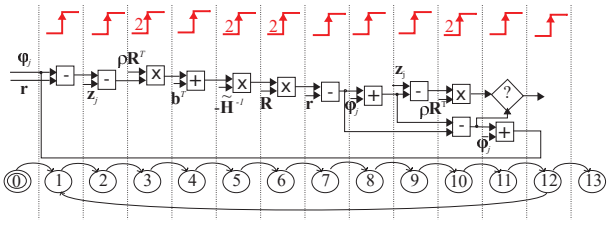

Fig. 2. Arquitetura de implementação do GPCADMM com máquina de estados finita.

VI. Implementação em FPGA

Para se avaliar o tempo de cômputo do algoritmo embarcado, o Algoritmo 1 foi implementado em um FPGA. O FPGA utilizado foi um Altera® MAX10® modelo 10M50DAF484C7G. A família MAX 10 apresenta conversores A/D embarcados no FPGA e $50\,000$ elementos lógicos programáveis. O sinal de *clock* utilizado foi de $50\,$MHz na placa de desenvolvimento DE10-Lite. Essa é uma solução de baixo custo e com uma velocidade modesta se comparada com FPGAs de ponta, como o Stratix® IV, que pode ser utilizado com sinais de *clock* de $600\,$MHz. O algoritmo foi implementado em Verilog. Para a representação numérica, foi implementada uma aritmética de ponto fixo com uma palavra de 16 bits, com 10 bits para os decimais. Máquinas de estados foram utilizadas para sincronizar os módulos e todas as operações de multiplicação matricial foram implementadas para execução em paralelo. Um esquemático da arquitetura implementada pode ser visto na Fig. 3. Os parâmetros de configuração do controlador são passados por meio de comunicação externa (USB-Blaster) com o processador NIOS II, em sua versão econômica, implementado no FPGA. A comunicação do software embarcado no NIOS II com o bloco de lógica GPC foi feito através do barramento Avalon®. O bloco GPC instancia os blocos ADMM e Resposta livre. As bibliotecas de multiplicação matricial e as operações básicas com aritmética de ponto fixo também foram implementadas diretamente em lógica customizada. A implementação foi compilada no software Intel Quartus Prime 17.1.0 e resultou em uma ocupação de $30\,662$ elementos lógicos (62%), $7\,691$ registradores, $49,125\,$KB de memória (24%) e 40 multiplicadores embarcados (14%). Por se tratar de um FPGA de baixo custo, a ocupação de 62% pode ser considerada boa e com margem para o aumento no tamanho dos horizontes se necessário. Os requisitos temporais foram obtidos com valores positivos de folga, sendo os piores casos, com folgas menores, de $1,034\,$ns de *setup*, $0,074\,$ns de *hold*, $15,394\,$ns de *recovery* e $0,406\,$ns de *removal*.

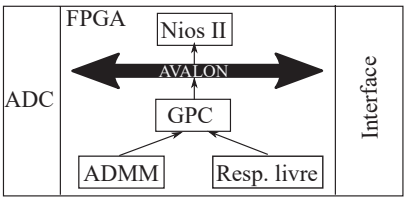

Fig. 3. Arquitetura de implementação em FPGA.

O estudo de caso utilizado foi o mesmo de [14] cuja planta foi modelada com uma função de transferência discreta de segunda ordem:

$$G(z) = \frac{0,035z + 0,0307}{z^2 - 1,6375z + 0,6703} \tag{24}$$

com um horizonte de controle $N_u = 5$, horizontes de predição $N_1 = 1$ e $N_2 = 20$, ponderação no esforço de controle $\lambda = 10$, ponderação no erro $\delta = 1$ e referências futuras conhecidas $w(t+j)$, $j = N1, ..., N2$.

Foram também definidas restrições para os incrementos de controle $|\Delta \mathbf{u}(k+j)| \leq 0,05$ com $j = 0, ..., N_u - 1$. O

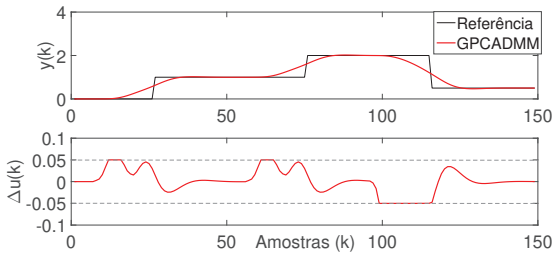

Fig. 4. Resultado da simulação com a saída da planta e o sinal de incremento de controle para o algoritmo proposto.

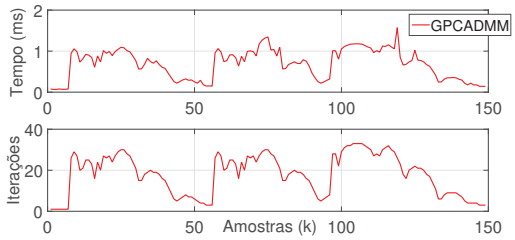

Fig. 5. Resultado da simulação com o tempo de execução e o número de iterações do algoritmo proposto.

problema de otimização resulta em uma matriz $\mathbf{H} \in \mathbb{R}^{5\times5}$ e 10 restrições. O GPCADMM foi sintonizado com $\rho = 50$ e com as tolerâncias dos resíduos primal e dual $\epsilon_p = \epsilon_d = \sqrt{N_u}0,001$.

O estudo de caso foi primeiramente simulado em MATLAB® de modo a se obter o pior caso no tempo de execução. A simulação foi realizada por 150 amostras discretas e com três trocas de referência do tipo degrau. Não foi utilizado *warm-start* para se avaliar o WCET. As simulações foram feitas em um computador com processador Core i3 de $2,40\,\text{GHz}$ e $12\,\text{GB}$ RAM. Os tempos de execução do algoritmo foram medidos através do comando *tic-toc* do MATLAB®.

Os comportamento do sistema de controle pode ser visto na Fig. 4, na qual são apresentados a saída da planta e o incremento de controle aplicado. Nas trocas de referência, pode-se perceber que as restrições estão ativas limitando o incremento de controle em $\pm0,05$.

O desempenho do algoritmo é ilustrado na Fig. 5, na qual o tempo de execução e o número de iterações são apresentados. Apesar de o GPCADMM apresentar um aumento no número de iterações durante as trocas de referência, cada iteração é pouco custosa do ponto de vista computacional. Essa é uma característica desejada para o algoritmo obter um bom desempenho ao ser embarcado.

A Tabela I apresenta os tempos de execução médio e máximo, obtidos a partir de 10 simulações. Dos dados de tempo de execução coletados, o tempo máximo observado pelo GPCADMM, com 33 iterações, foi de 1,30 ms.

O pior caso observado na simulação foi replicado na implementação embarcada em FPGA. O tempo gasto por iteração em cada módulo foi de $0,14\,\mu\text{s}$ para o GPC, $0,36\,\mu\text{s}$ para o ADMM e $0,80\,\mu\text{s}$ para resposta livre, totalizando $1,30\,\mu\text{s}$ para 1 iteração completa do GPCADMM. Os resultados obtidos

TABELA I
COMPARAÇÃO DOS TEMPOS DE CÔMPUTO

	GPCADMM
Tempo médio	0,66 ms
Tempo Máximo (iter.)	1,30 ms (33)

para o pior caso, com 33 iterações do ADMM, podem ser vistos na Tabela II. Devido ao paralelismo implementado, pode-se perceber que o tempo de execução é bastante rápido, com valor máximo de $11,54\,\mu\text{s}$.

Esse resultado pode ser considerado competitivo com outras soluções para MPC. Em [11], um algoritmo GPC com o método de projeção de gradiente acelerado dual (GPAD, do inglês *Accelerated Dual Gradient Projection*) foi testado com a mesma planta de exemplo (24) e com o mesmo FPGA. O WCET observado nesse trabalho foi de $11,76\,\mu\text{s}$ com 21 iterações do algoritmo de otimização. Percebe-se que a arquitetura GPCADMM proposta foi mais rápida, mesmo com um número maior de iterações. Esse desempenho pode ser explicado, em grande parte, graças ao ganho obtido com a arquitetura de multiplicação matricial, implementada com o somador de 10 entradas. Vale ressaltar que os métodos acelerados de primeira ordem, como o GPAD, sofrem de acumulação de erro quando aplicados com aritmética de ponto fixo [22]. Um estudo sobre a representação numérica de ponto fixo adequada para o ADMM pode ser vista em [23]. Já para as formulações em espaço de estados, em [24], um algoritmo MPC com o método de ponto interior foi aplicado em uma planta de controle de vibração em uma haste flexível. A planta foi modelada com 14 estados e o horizonte de predição de 14 amostras. O tempo de cômputo em um FPGA com $70\,\text{MHz}$ e aritmética de ponto flutuante foi de $30,0\,\mu\text{s}$. Em [25], um MPC com o método de conjunto ativo foi aplicado em um servomotor modelado com 2 estados. O horizonte de controle utilizado foi de 3 amostras e o tempo de cômputo em um FPGA com *clock* de $100\,\text{MHz}$ foi de $20,0\,\mu\text{s}$.

TABELA II
TEMPOS DE EXECUÇÃO EM FPGA

	Tempo máximo com 33 iterações
GPC	$0,14\,\mu\text{s}$
ADMM	$10,60\,\mu\text{s}$
Resp. livre	$0,80\,\mu\text{s}$
GPCADMM	$11,54\,\mu\text{s}$

VII. CONCLUSÃO

Neste trabalho foi apresentada uma implementação embarcada de um algoritmo GPC de cômputo rápido baseado em ADMM. Como o algoritmo proposto utiliza apenas operações básicas em cada iteração, a implementação embarcada em FPGA se torna bastante adequada. O GPCADMM embarcado em FPGA mostrou-se bastante rápido para o estudo de caso apresentado, validando a arquitetura proposta e as acelerações propostas nos cômputos das operações matemáticas. Os resultados obtidos contribuem para a viabilidade da utilização do

GPC em processos que tipicamente são controlados por PIDs por conta da dinâmica rápida, trazendo assim as vantagens inerentes do método.

REFERÊNCIAS

[1] Y. Wang and S. Boyd, "Fast model predictive control using online optimization," *IEEE Trans. Control Syst. Technol.*, vol. 18, no. 2, pp. 267–278, Mar. 2010.

[2] T. Samad, "A survey on industry impact and challenges thereof [technical activities]," *IEEE Control Syst. Mag.*, vol. 37, no. 1, pp. 17–18, Feb. 2017.

[3] J. E. Normey-Rico and E. F. Camacho, *Control of Dead-time Processes*. London: Springer, 2007.

[4] E. Camacho and C. Bordons, *Model Predictive Control*, ser. Advanced Textbooks in Control and Signal Processing. London: Springer, 2004.

[5] R. Oberdieck, N. A. Diangelakis, and E. N. Pistikopoulos, "Explicit model predictive control: A connected-graph approach," *Automatica*, vol. 76, pp. 103–112, 2017.

[6] X. Cai, M. J. Tippett, L. Xie, and J. Bao, "Fast distributed MPC based on active set method," *Comput. Chem. Eng.*, vol. 71, pp. 158–170, 2014.

[7] M. Herceg, C. N. Jones, and M. Morari, "Dominant speed factors of active set methods for fast MPC," *Optim. Contr. Appl. Met.*, vol. 36, no. 5, pp. 608–627, 2015.

[8] A. G. Wills, G. Knagge, and B. Ninness, "Fast linear model predictive control via custom integrated circuit architecture," *IEEE Trans. Control Syst. Technol.*, vol. 20, no. 1, pp. 59–71, Jan. 2012.

[9] S. D. Cairano, M. Brand, and S. A. Bortoff, "Projection-free parallel quadratic programming for linear model predictive control," *Int. J. Control*, vol. 86, no. 8, pp. 1367–1385, 2013.

[10] P. Patrinos and A. Bemporad, "An accelerated dual gradient-projection algorithm for embedded linear model predictive control," *IEEE Trans. Autom. Control*, vol. 59, no. 1, pp. 18–33, Jan. 2014.

[11] V. B. Peccin, D. M. Lima, R. C. C. Flesch, and J. E. Normey-Rico, "Fast generalized predictive control based on accelerated dual gradient projection method," in *Proc. 12th IFAC Symp. Dynamics and Control of Process Syst., incl. Biosystems (DYCOPS)*, 2019, pp. 474 – 479.

[12] B. O'Donoghue, G. Stathopoulos, and S. Boyd, "A splitting method for optimal control," *IEEE Trans. Control Syst. Technol.*, vol. 21, no. 6, pp. 2432–2442, Nov. 2013.

[13] H. Peyrl, A. Zanarini, T. Besselmann, J. Liu, and M.-A. Boéchat, "Parallel implementations of the fast gradient method for high-speed MPC," *Control Eng. Practice*, vol. 33, pp. 22–34, 2014.

[14] V. B. Peccin, D. M. Lima, R. C. C. Flesch, and J. E. Normey-Rico, "Implementação de GPC e DMC para sistemas rápidos usando ADMM," in *Anais XXII Congresso Brasileiro de Automática (CBA)*, 2018.

[15] D. W. Clarke, C. Mohtadi, and P. S. Tuffs, "Generalized predictive control - part 1: The basic algorithm," *Automatica*, vol. 23, no. 2, pp. 137–148, Mar. 1987.

[16] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Found. Trends Mach. Learn.*, vol. 3, no. 1, pp. 1–122, Jan. 2011.

[17] G. Stathopoulos, H. Shukla, A. Szucs, Y. Pu, and C. N. Jones, "Operator splitting methods in control," *Found. Trends Syst. and Control*, vol. 3, no. 3, pp. 249–362, 2016.

[18] N. Parikh and S. Boyd, "Proximal algorithms," *Found. Trends Optimiz.*, vol. 1, no. 3, pp. 127–239, 2014.

[19] S. M. Qasim, S. A. Abbasi, and B. A. Almashary, "Hardware realization of matrix multiplication using field programmable gate array," *MASAUM J. of Computing*, vol. 1, no. 1, pp. 21–25, 2009.

[20] M. Kumm, M. Hardieck, and P. Zipf, "Optimization of constant matrix multiplication with low power and high throughput," *IEEE Trans. Comput.*, vol. 66, no. 12, pp. 2072–2080, Dec. 2017.

[21] J. Xie, P. K. Meher, M. Sun, Y. Li, B. Zeng, and Z. Mao, "Efficient FPGA implementation of low-complexity systolic Karatsuba multiplier over $GF(2^m)$ based on NIST polynomials," *IEEE Trans. Circuits Syst. I*, vol. 64, no. 7, pp. 1815–1825, Jul. 2017.

[22] O. Devolder, F. Glineur, and Y. Nesterov, "First-order methods of smooth convex optimization with inexact oracle," *Math. Program.*, vol. 146, no. 1, pp. 37–75, Aug. 2014.

[23] J. L. Jerez, P. J. Goulart, S. Richter, G. A. Constantinides, E. C. Kerrigan, and M. Morari, "Embedded online optimization for model predictive control at megahertz rates," *IEEE Trans. Autom. Control*, vol. 59, no. 12, pp. 3238–3251, Dec. 2014.

[24] A. Wills, A. Mills, and B. Ninness, "FPGA implementation of an interior-point solution for linear model predictive control," in *Proc. 18th IFAC World Congr.*, 2011.

[25] N. Yang, D. Li, J. Zhang, and Y. Xi, "Model predictive controller design and implementation on FPGA with application to motor servo system," *Control Eng. Practice*, vol. 20, no. 11, pp. 1229–1235, 2012.

Vinícius Berndsen Peccin received the B.E. degree in control and automation engineering in 2009, and the M.E. degree in automation and systems engineering in 2013 from Federal University of Santa Catarina (UFSC), Florianópolis, Brazil. He is currently a Ph.D. candidate in automation and systems engineering from UFSC.

He is a professor at Federal Institute of Education, Science and Technology of Santa Catarina, Chapecó, Brazil. His current research includes model predictive control and process control applications.

Daniel Martins Lima received the B.E. in Control and Automation Engineering in 2011, M.E. and Ph.D. in Automation and Systems Engineering in 2013 and 2015, respectively, from the Federal University of Santa Catarina, Florianópolis, Brazil. He is currently a Professor at Federal University of Santa Catarina, Blumenau, Brazil. His current research interests include model predictive control and dead-time compensators.

Rodolfo César Costa Flesch received the B.E. degree in control and automation engineering from the Federal University of Santa Catarina (UFSC), Florianópolis, Brazil, in 2006, the B.S. degree in business administration from the Santa Catarina State University, Florianópolis, Brazil, in 2007, and both the M.Eng. and Dr.Eng. degrees in automation and systems engineering from UFSC, in 2009 and 2012, respectively. He is currently a Professor with the Department of Automation and Systems Engineering, UFSC, and a researcher for the Brazilian National Council for Scientific and Technological Development, Brasília, Brazil. His main research interests are process control (time delay processes and model predictive control), instrumentation, and automation of tests.

Julio Elias Normey-Rico Received the Ph.D. degree from the Dept. of System Eng. and Automatic Control, University of Seville, Spain, in 1999. He is currently Full Professor at the Dept. of Automation and Systems Engineering, Federal University of Santa Catarina (UFSC), Brazil; researcher of the Brazilian National Research Council CNPq and director of several research projects with industry in Brazil. His current research interests include model predictive control, time delay systems, PID controllers and process control applications. In the last years his research has been focussed in the control and optimization of energy systems including petroleum and natural gas production and renewable energies such as solar, wind, and biomass. He is the author or co-author of about 260 conference and journal papers, the book Control of Dead-Time Processes (Springer, 2007), and has supervised 60 Ph.D./M.Sc. graduates.