DISSERTAÇÃO DE MESTRADO

IMPLEMENTAÇÃO EM FPGA DE ESTRATÉGIAS DE CONTROLE PREDITIVO PARA UM QUADRIRROTOR

Alceu Bernardes Castanheira de Farias

Brasília, Março de 2019

UNIVERSIDADE DE BRASÍLIA

FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA MECÂNICA

IMPLEMENTAÇÃO EM FPGA DE ESTRATÉGIAS DE CONTROLE PREDITIVO PARA UM QUADRIRROTOR

Alceu Bernardes Castanheira de Farias

DISSERTAÇÃO SUBMETIDA AO DEPARTAMENTO DE ENGENHARIA MECÂNICA DA FACULDADE DE TECNOLOGIA DA UNIVERSIDADE DE BRASÍLIA COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM SISTEMAS MECATRÔNICOS

APROVADA POR:

Prof. Dr. André Murilo de Almeida, PPMEC/UnB
Orientador

Prof. Dr. Renato Vilela Lopes, FGA/UnB
Coorientador

Prof. Dr. Daniel Maurício Muñoz Arboleda, PPMEC/UnB
Membro Interno

Prof. Dr. Renato Coral Sampaio, FGA/UnB
Membro Externo

BRASÍLIA/DF, 29 DE MARÇO DE 2019

Bernardes Castanheira de Farias, Alceu
Implementação em FPGA de estratégias de controle preditivo para um quadrirrotor / Alceu Bernardes Castanheira de Farias. –Brasil, 2019.
120 p.

Orientador: André Murilo de Almeida Pinto
Coorientador: Renato Vilela Lopes
Dissertação (Mestrado) – Universidade de Brasília – UnB
Faculdade de Tecnologia – FT
Programa de Pós-Graduação em Sistemas Mecatrônicos – PPMEC, 2019.

1. Sistemas de Controle. 2. Sistemas Embarcados. 3. Controle Preditivo Baseado em Modelo. 4. FPGA. 5. Quadrirrotores. I. André Murilo de Almeida Pinto, orientador. II. Renato Vilela Lopes, coorientador. III. Universidade de Brasília. IV. Faculdade de Tecnologia.

Agradecimentos

Em primeiro lugar a Deus, por permitir que eu pudesse realizar mais essa etapa da minha trajetória acadêmica e por tudo que me proporcionou em minha vida.

À minha mãe, Máuria Maria Castanheira, por toda sua luta para permitir que eu pudesse chegar aonde cheguei. Tudo que eu conquistei e hei de conquistar é graças ao seu amor e tudo que me ensinou.

À minha namorada, Mônica, pelo apoio e compreensão durante todo esse período do mestrado. Sem seu carinho e sua ajuda eu não teria conseguido cumprir mais esta etapa na minha vida.

Aos meus orientadores André Murilo e Renato Lopes, cujas orientações foram fundamentais para a conclusão deste trabalho. Obrigado pelo auxílio, pelas conversas e por terem aceito este desafio junto comigo.

Aos professores Daniel Muñoz e Renato Coral, cujos conhecimentos e auxílios prestados foram fundamentais para alcançar os resultados obtidos neste trabalho.

A todos os meus familiares, amigos e colegas de pós-graduação, que auxiliaram e contribuíram de alguma forma, seja com o trabalho em si ou por meio de experiências vividas durante esse período.

A todos os demais professores e funcionários do PPMEC que me ajudaram em diversos pontos dessa caminhada e permitiram que mais esta etapa pudesse ser concluída.

Por fim, à CAPES, pela bolsa que me auxiliou financeiramente durante todo meu período de mestrado.

Alceu Bernardes Castanheira de Farias

Resumo

Uma das técnicas de controle que mais impactou tanto a indústria quanto a academia nos últimos anos foi a estratégia de controle conhecida como Controle Preditivo baseado em Modelo (MPC). A principal razão para o interesse nesse tipo de controlador é o fato de o mesmo ser capaz de controlar um sistema respeitando as restrições que são impostas ao mesmo, independente da natureza delas (normas de segurança, faixa de operação dos atuadores, normas de qualidade de um produto, etc.), ao mesmo tempo que encontra uma solução ótima que satisfaz os objetivos de controle desejados.

Entretanto, o MPC possui uma grande desvantagem: seu elevado custo computacional. Isso limita a implementação do MPC a sistemas de dinâmicas mais lentas, o que tem levado a um aumento no estudo de técnicas de otimização e no desenvolvimento de sistemas mais rápidos e com mais recursos, capazes de acelerar o MPC para aplicá-lo a sistemas de dinâmicas mais rápidas.

Neste cenário, este trabalho apresenta o desenvolvimento de uma arquitetura em *hardware* do MPC, utilizando FPGA (do inglês, *Field Programmable Gate Array*) para sua implementação. Este tipo de dispositivo reconfigurável tem ganho notoriedade na literatura por permitir a aceleração de algoritmos, por meio da paralelização dos mesmos ao implementá-los diretamente em *hardware*. A aplicação escolhida para este trabalho consiste em um Veículo Aéreo Não Tripulado (VANT) quadrirrotor, um tipo de sistema amplamente utilizado em diversas áreas e muito estudado na atualidade, já que é considerado um problema de controle desafiador para a aplicação do controlador MPC, devido às dimensões do seu modelo e suas dinâmicas rápidas, que também são acopladas e inerentemente instáveis.

Neste trabalho, são apresentados resultados referentes à implementação do MPC sem aplicação de restrições e com aplicação de restrições implementados em FPGA, com o objetivo de rastrear trajetórias pré-definidas para o quadrirrotor. Todos os algoritmos são desenvolvidos manualmente e implementados utilizando artimética em ponto flutuante. Os resultados mostram que o MPC sem restrições consome menos recursos de *hardware* do que a solução com restrições, mas não é capaz de lidar com cenários nos quais restrições não podem ser violadas. Por outro lado, O MPC com restrições respeita essas restrições, ao custo de um consumo de *hardware* maior. Por conta da sua complexidade, o MPC com restrições também exige maior precisão em ponto flutuante, de forma que os melhores resultados foram alcançados utilizando tamanho de dados de 32 *bits* em ponto flutuante.

- **Palavras-chave:** Sistemas de controle, Sistemas embarcados, Controle Preditivo Baseado em Modelo, Quadrirrotores, FPGAs

Abstract

One of the most impactful control techniques both in industry and academia through the last years is the control strategy known as Model Predictive Control (MPC). The main reason for such interest on this type of controller is due to the fact that it is capable of controlling a system while handling constraints imposed to it, regardless of the nature of such constraints (security, operation range of the actuators, quality norms of a product, etc.), while still finding an optimal solution that satisfies the desired control objectives.

However, MPC has a major drawback: its elevated computational cost. This limits MPC implementation to systems with slower dynamics, which has lead to an increase of studies regarding new optimization techniques and the development of faster and more resourceful systems, capable of accelerating MPC in order to apply it to systems with faster dynamics.

Regarding this scenario, this work presents the development of a hardware architecture for MPC, using FPGA (Field Programmable Gate Array) for its implementation. This type of reconfigurable device has gained notoriety in literature for allowing the speedup of embedded algorithms, by parallelising and implementing them directly in hardware. The chosen application for this work consists of a quadrotor Unmanned Aerial Vehicle (UAV), a type of system that has been widely used today in many fields of application and vastly studied, as it is considered a challenging control problem for MPC, due to the dimensions of its model and fast dynamics, which are also coupled and inherently unstable.

In this work, results regarding both unconstrained MPC and constrained MPC implemented using FPGA are presented, with the goal of tracking predefined trajectories for the quadrotor. All control algorithms were developed manually and using floating point arithmetic. The results obtained show that unconstrained MPC consumes less hardware resources than the constrained solution, but it is not capable of handling scenarios in which constraints must not be violated. On the other hand, constrained MPC is capable of handling these constraints, at the cost of a higher hardware consumption. Due to its complexity, constrained MPC also demands a higher floating point precision, so that the best results were obtained using a 32 bits data width in floating point.

- **Keywords:** Control systems, Embedded systems, Model Predictive control, Quadrotors, FPGAs

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

AMPC	Adaptative Model Predictive Control
CLB	*Configurable Logic Block*
CPLD	*Complex Programmable Logic Device*
DMC	*Dynamic Matrix Control*
DSP	*Digital Signal Processing*
EDA	*Electronic Design Automation*
FB	*Feedback Control*
FFW	*Feedforward Control*
FPGA	*Field Programmable Array*
FSM	*Finite State Machine*
GAL	*Generic Logic Arrays*
GE	Gradiente com Expansão
GPU	*Graphics Processing Unit*
HIL	*Hardware in the Loop*
I/O	*Inputs/Outputs*
LNS	*Logarithmic Number System*
LQ	*Linear Quadratic Control*
LQG	*Linear-Quadratic-Gaussian Control*
LUT	*Look Up Table*
MHPC	*Model Heuristic Predictive Control*
MPC	*Model Predictive Control*
MIMO	*Multiple Input Multiple Output*
MSB	*Most Significant Bit*
NMPC	*Non-linear Model Predictive Control*

PAL	*Programmable Logic Arrays*
PDIP	*Primal Dual Interior Point*
PLD	*Programmable Logic Device*
PLL	*Phase Lock Loop*
PSO	*Particle Swarm Operation*
PVC	Policoreto de Vinila
QFT	*Quantitative Feedback Theory*
PGE	Projeção do Gradiente com Expansão
QP	*Quadratic Problem*
RAM	*Random Access Memory*
RBFNN	*Radial Basis Function Neural Network*
RC	*Ratio Control*
RHC	*Receding Horizon Control*
ROM	*Read Access Memory*
SoC	*System on Chip*
SQP	*Sequential Quadratic Problem*
TSR	*Truncated Step Response*
UAV	*Unmanned Aerial Vehicle*
USB	*Universal Serial Bus*
VANT	Veículo Aéreo Não-Tripulado
VHDL	*Very High Speed Integrated Circuit Hardware Description Language*
VTOL	*Vertical Take-Off and Landing*

LISTA DE SÍMBOLOS

ω_i	Velocidade angular do i-ésimo motor do quadrirotor
b	Coeficiente de empuxo das hélices dos motores do quadrirotor
d	Coeficiente de arrasto do quadrirotor
e	Número de *bits* do expoente de um número representado em ponto flutuante
f	Número de *bits* da mantissa de um número representado em ponto flutuante
g	Gravidade local
I_{xx}	Coeficiente de empuxo das hélices dos motores do quadrirotor
I_{yy}	Coeficiente de empuxo das hélices dos motores do quadrirotor
I_{zz}	Coeficiente de empuxo das hélices dos motores do quadrirotor
L	Meia envergadura do quadrirotor
m	Massa do quadrirotor
N	Horizonte de predição
N_{iter}	Número de iterações do *solver* PGE
n_u	Número de variáveis de controle do sistema
n_x	Número de variáveis de estado do sistema
n_y	Número de saídas do sistema
s	*Bit* de sinal de um número representado em ponto flutuante

Capítulo 1

Introdução

Uma das áreas que mais impactou a indústria e a academia nos últimos anos, contribuindo de maneira fundamental com diversos campos da engenharia, é a área de sistemas de controle e automação (KOZÁK, 2014). Segundo Frank (2012), a tecnologia relacionada a sistemas de controle e automação pode ser entendida como sendo composta por métodos, processos, instalações (tanto de *hardware* quanto de *software*) e, obviamente, as estratégias de controle capazes de cumprir objetivos previamente estabelecidos sem a interferência humana, de maneira independente.

Aplicações de sistemas de controle e automação estão presentes nos mais diversos campos do dia-a-dia, desde os mais variados segmentos da indústria (como sistemas de telecomunicações, automóveis, aeronaves, satélites, sistemas mecatrônicos) até a área biomédica (KOZÁK, 2014). Essa variedade de aplicações têm motivado o desenvolvimento de sistemas de controle cada vez mais otimizados e avançados. De fato, a teoria de controle evoluiu de forma a permitir uma grande variedade de técnicas de controle, conforme pode ser visto na Tabela 1.1.

Tabela 1.1: Tipos de desenvolvimento de estratégias de controle. Fonte: (KOZÁK, 2014).

A. Controle PID Convencional	B. Controle Avançado I	C. Controle Avançado II	D. Controle Avançado III
Controle Manual	Adaptativo e *selftuning*	Métodos de Controle Ótimo (LQ e LQG)	Controle Preditivo Híbrido
Feedback (FB)	*Gain Scheduling*	Controle Robusto	Controlador *Fuzzy*
Controle em Cascata (CC)	Multivariável (Espaço de Estados e Funções de Transferência)	Controle Preditivo Baseado em Modelo (MPC)	Controle por Rede Neural
Feedforward (FFW)	*Decoupling*	Controle Decentralizado	Controle de Eventos Discretos
Ratio Control (RC)	Alocação de Pólos	Síntese Polinomial	*Soft Computing* híbrido não linear
Estrutura Combinada (CC + FFW + RC)	Controle Não Linear	Controle Robusto QFT	*Expert Control*

Dentre as técnicas de controle apresentadas, uma das que mais tem recebido destaque nos últimos anos é o Controle Preditivo baseado em Modelo ou MPC (do inglês, *Model Predictive Control*). O objetivo desta técnica de controle é obter uma estimativa do comportamento futuro do sistema a ser controlado, obtendo a cada instante de tempo uma ação de controle ótima, que busca minimizar uma determinada função custo para o problema de controle, dentro de um número finito de passos (EKAPUTRI; SYAICHUROHMAN, 2013). Isso ocorre a cada instante de amostragem do sistema, permitindo que uma ação de controle otimizado possa ser aplicada ao processo que está sendo controlado (HARTLEY et al., 2014).

O MPC têm tido um enorme impacto na indústria nos últimos anos (LEE, 2011), por conta das van-

tagens que essa técnica de controle oferece. A primeira delas é a capacidade de lidar com sistemas de múltiplas entradas e saídas (do inglês, *Mutiple Input Multiple Output* ou MIMO) (VOUZIS et al., 2009). Já a segunda, e uma das principais características que diferenciam o MPC em relação a outras técnicas de controle, é a capacidade de levar em consideração as restrições de entrada e saída do sistema, de tal forma que, segundo Alamir (2013), a única estratégia de controle avançada que possibilita lidar de maneira sistemática com restrições é o MPC. Isso é de fato uma grande vantagem, uma vez que, em problemas de controle reais, os componentes de um sistema possuem restrições inerentes às suas faixas de operação, sejam elas relacionadas a restrições físicas dos atuadores do sistema, restrições de seguranças em algumas variáveis-chave (temperatura e pressão, por exemplo) ou até mesmo restrições contratuais, como normas de qualidade do produto.

Ainda segundo Alamir (2013), outras características importantes do MPC envolvem generalidade (ou seja, a metodologia da técnica de controle pode ser aplicada a diferentes tipos de sistemas sem grandes modificações) e a capacidade de lidar com sistemas não lineares (dando origem ao Controle Preditivo Baseado em Modelo Não Linear, do inglês *Nonlinear Model Predictive Control*, ou NMPC). Essa última característica não é uma exclusividade dessa técnica de controle, mas graças à generalidade descrita anteriormente, há uma facilidade maior em utilizar essa técnica para desenvolver algoritmos de controle para sistemas não lineares.

Essas características têm contribuído cada vez mais para a utilização do MPC como estratégia de controle embarcada em diversos tipos de sistema. A Figura 1.1 mostra algumas das principais áreas de aplicação do MPC embarcado.

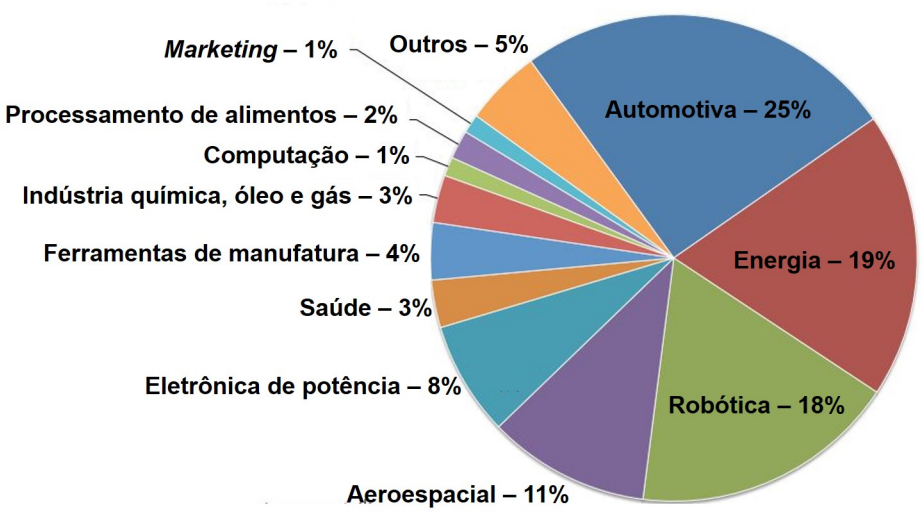

Figura 1.1: Áreas industriais de aplicação do MPC embarcado em plataformas de *hardware*. Fonte: (DOMAHIDI et al., 2016).

A literatura também mostra que há uma grande quantidade de trabalhos que utilizam o MPC nas mais diversas áreas. Uma das áreas que mais aplicam o MPC é a indústria automotiva, na qual é possível destacar as contribuições de Shourkry, El-Kharashi e Hammad (2010), que utiliza o MPC embarcado em um coprocessador para controlar um sistema de suspensão ativa; Spivey e Edgar (2012), que utiliza o

MPC no controle de células combustíveis de óxido sólido; Murilo, Alamir e Alberer (2014), que define um *framework* de utilização do NMPC para o controle do caminho de ar de um motor a diesel; Takács *et al.* (2016), cujo trabalho é voltado para a aplicação do MPC para controle de vibração em veículos; entre outros.

Já na área aeroespacial, é possível destacar aplicações do MPC em satélites, como abordado nos trabalhos de Chen e Wu (2011) e de Hartley e Maciejowski (2013), além de aeronaves, como no trabalho já citado de Hartley *et al.* (2014). Outras áreas de destaque correspondem à robótica, com aplicações voltadas ao controle de robôs móveis diferenciais (WORTHMANN et al., 2015), robôs de pêndulo invertido no trabalho (OHHIRA; SHIMADA, 2017) e exoesqueletos reconfiguráveis de membros inferiores (RODRIGUEZ; PONCE; MOLINA, 2017); energia e eletrônica de potência, área na qual, segundo Rodriguez *et al.* (2009), o MPC é considerado uma técnica de controle simples mas poderosa, possibilitando o controle de diversas aplicações; e até mesmo na área da saúde, abordando o estudo de regulação de glicose para pacientes com diabetes tipo 1 (VOUZIS et al., 2009) e uma adaptação do MPC para o controle de uma célula pancreática artificial, conhecida como *Multi-Zone MPC*, respectivamente (GROSMAN et al., 2011).

Apesar de todas as vantagens já supracitadas, há uma grande desvantagem inerente à formulação do MPC: o seu elevado custo computacional. Essa característica explica o fato de que o MPC foi inicialmente implementado em sistemas de dinâmica mais lenta, cujas taxas de amostragem não eram muito elevadas. Portanto, há um grande número de áreas de aplicação que poderiam se beneficiar do MPC, mas devido à característica citada acima, isso acaba se tornando inviável (DUA et al., 2008).

Um exemplo desse tipo de sistema é o chamado Veículo Aéreo Não Tripulado ou VANT (em inglês, *Unmanned Aerial Vehicle* ou UAV). A área de projeto de controladores para esse tipo de sistema tem sido amplamente estudada nos últimos anos, tendo em vista sua grande aplicabilidade na atualidade em atividades remotas ou autônomas. Esse tipo de sistema é inerentemente instável, não linear e possui múltiplas variáveis de entrada e saída. Apesar disso, quando um controlador apropriado é projetado, VANTs apresentam boa estabilidade durante o vôo (LOPES et al., 2011).

Levando em consideração as características já citadas sobre o MPC, é possível perceber que o mesmo se mostra uma alternativa de controle interessante para VANTs, sendo capaz de lidar com sistemas de múltiplas entradas e saídas, não lineares e lidar com restrições (de forma a não permitir que o sistema alcance regiões de instabilidade ao violar essas restrições). Adicionalmente, a complexidade do modelo desse tipo de sistema (por conta de suas dimensões) torna desafiadora a implementação do MPC para o caso de um VANT, o que por sua vez, torna este um problema de controle interessante de ser estudado.

Por conta dessa problemática, diversos estudos vêm sendo realizados nos últimos anos, no sentido de possibilitar soluções mais eficientes e capazes de lidar com a elevada carga computacional exigida pelo MPC (DUA et al., 2008). Uma dessas soluções que tem ganho bastante destaque recentemente é a utilização de FPGAs (do inglês, *Field Programmable Gate Arrays*), dispostivos lógicos programáveis, compostos por uma tecnologia que permite o desenvolvimento de sistemas embarcados baseados em arquiteturas paralelas, que por sua vez, podem ser implementadas diretamente em *hardware*, oferecendo soluções com alta capacidade de processamento (MUÑOZ, 2012).

FPGAs têm sido utilizados para paralelizar algoritmos, permitindo não somente que os mesmos sejam acelerados, mas também implementados em sistemas embarcados. Logo, é possível encontrar trabalhos

na literatura que mostram como esse tipo de *hardware* pode ser utilizado para implementar o MPC, sejam eles problemas lineares, como proposto em Jerez *et al.* (2012) e nos já citados trabalhos de Vouzis *et al.* (2009) e Hartley *et al.* (2014), ou até mesmo em casos não lineares, como nos trabalhos de Kaepernick *et al.* (2014) e Ayala *et al.*(2016).

Com base nas discussões efetuadas até o momento, é possível perceber que há na literatura trabalhos relacionados à utilização do MPC em VANTs e quadrirrotores e à utilização de FPGAs para implementações do MPC. Entretanto, não há muitos trabalhos na literatura que buscam unir essas vertentes: utilizar FPGAs para embarcar o MPC e utilizá-las com VANTs e quadrirrotores. Isso pode ser evidenciado por meio de uma pesquisa em bases de dados dos periódicos CAPES, conforme mostrado na Figura 1.2.

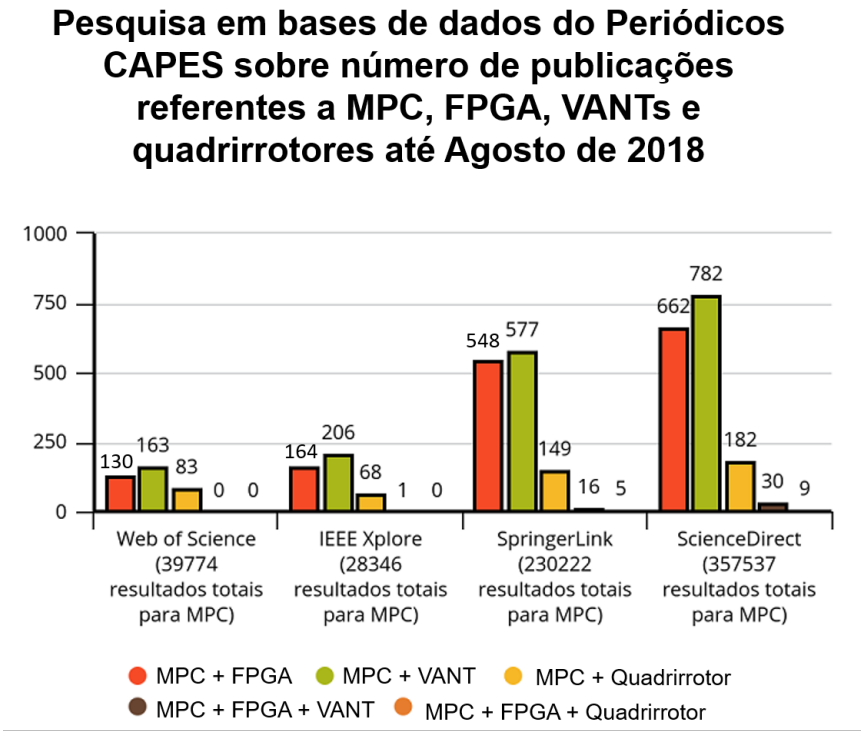

Figura 1.2: Pesquisa em bases de dados disponíveis no periódicos CAPES sobre o número de publicações referentes a MPC, FPGA e quadrirrotores.

É importante salientar que essa pesquisa foi realizada com os termos UAV e *quadrotor*, no lugar dos termos VANT e quadrirrotor, por se tratarem de bases de dados internacionais. Mesmo assim, a Figura 1.2 mostra que há bastante espaço para contribuição na literatura neste segmento de pesquisa.

Neste cenário, este trabalho propõe a implementação em FPGA de estratégias de controle preditivo, sem e com aplicação de restrições, para rastreamento de trajetórias pré-definidas para um quadrirrotor. A Figura 1.2 mostra que esse tipo de sistema possui ainda menos publicações na literatura, uma vez que trata-se de um problema de controle desafiador. Entretanto, a melhoria contínua dos recursos computacionais para aplicações de tempo real e o desenvolvimento de algoritmos numéricos eficientes para a resolução de problemas de otimização em conjunto com o desenvolvimento de tecnologia dos FPGAs, motiva e possibilita esse trabalho.

Para realizar tal implementação, este trabalho se baseia na formulação do MPC apresentada em Alamir (2013). A estratégia de otimização do MPC adotada se dá por meio do algoritmo PGE (do inglês, *Projected Gradient Expansion*), que também encontra-se disponível em Alamir (2013) no formato de algoritmos em MATLAB. O FPGA utilizado para implemetação do MPC corresponde a um *kit* KC-705 da Xilinx, contendo um FPGA da família Kintex 7. A linguagem de descrição de *hardware* utilizada para a implementação do controlador é o VHDL (do inglês, *Very High Speed Integrated Circuit Hardware Description Language*). A formulação do modelo do quadrirrotor adotada corresponde ao modelo linearizado do sistema, que pode ser encontrado em Lopes *et al.* (2011).

Assim, utilizando ferramentas da Xilinx, como o Vivado e ferramentas do MATLAB para integração com FPGAs, como o HDL Verifier, pretende-se validar as estratégias de controle desenvolvidas, bem como estimar a quantidade de recursos de *hardware* necessários para a implementação das mesmas.

1.1 Objetivos

1.1.1 Objetivo geral

- Implementar em FPGA estratégias de controle preditivo linear, sem e com restrições, aplicadas a um veículo quadrirrotor.

1.1.2 Objetivos específicos

- Desenvolver manualmente todos os algoritmos em linguagem de descrição de *hardware* utilizados para a implementação do MPC em FPGA;
- Desenvolver arquiteturas de *hardware* com precisão em ponto flutuante para implementação do MPC em FPGA;
- Implementar em FPGA o MPC sem aplicação de restrições para controle de um veículo quadrirrotor;
- Implementar em FPGA o MPC aplicando restrições ao controle de um veículo quadrirrotor utilizando como algoritmo de otimização o algoritmo PGE e utilizando parametrização exponencial.

1.2 Contribuições do trabalho

A primeira contribuição deste trabalho refere-se ao tema escolhido, devido à pouca quantidade de trabalhos existentes que embarcam o MPC em FPGA para controle de aeronaves do tipo quadrirrotor. Isso é importante, tendo em vista a complexidade do problema de controle envolvendo o quadrirrotor (um sistema com muitos estados e variáveis de controle quando comparado com outros sistemas controlados pelo MPC disponíveis na literatura).

Este trabalho propõe ainda mais duas contribuições importantes: (a) a implementação da arquitetura do MPC utilizando o algoritmo de otimização PGE e utilizando aritmética de ponto flutuante; (b) o desenvolvimento manual dos algoritmos de controle em linguagem de descrição de *hardware*. Estas são

contribuições importantes pois a maioria das soluções de MPC embarcado em FPGA encontradas na literatura utilizam ponto fixo e são desenvolvidas a partir de ferramentas de geração automática de código, como o HDL Coder do MATLAB.

É possível citar ainda como contribuição deste trabalho o desenvolvimento de ferramentas de parametrização automática dos algoritmos desenvolvidos, por meio de *scripts* em MATLAB. Esses *scripts* permitem que todos os algoritmos de controle desenvolvidos adaptem o tamanho das entradas, saídas e sinais que compõem os mesmos, de forma a atender as características e dimensões do problema de controle com que se está trabalhando e preservando a lógica que foi desenvolvida manualmente em cada um deles. Assim, é possível realizar testes alterando parâmetros dos controladores de maneira mais rápida e eficiente.

Portanto, com base em tudo que foi apresentado até este momento, é possível perceber que por meio destas atividades este trabalho almeja realizar contribuições para uma área atual e relevante da literatura, por meio de uma aplicação atual, relevante e que possui muito espaço para publicações, como é o controle de quadrirrotores.

1.3 Organização do manuscrito

Este trabalho está organizado da maneira que se segue: o primeiro e presente capítulo apresenta a introdução e justificativa do trabalho, bem como os objetivos gerais e específicos do mesmo; o segundo capítulo apresenta toda a fundamentação teórica e a revisão bibliográfica referente às três grandes áreas que abrangem este trabalho: MPC, quadrirrotores e FPGA; o terceiro capítulo corresponde à metodologia, ou seja, quais as atividades que foram desenvolvidas para alcançar os objetivos do trabalho; o quarto capítulo apresenta os resultados obtidos, assim como discussões acerca desses resultados; por fim, o quinto e último capítulo apresenta as conclusões finais referentes a tudo que foi apresentado no trabalho e propostas de trabalhos futuros.

Capítulo 2

Revisão bibliográfica

2.1 Controle Preditivo Baseado em Modelo (MPC)

2.1.1 Introdução ao MPC

A técnica de controle denominada Controle Preditivo Baseado em Modelo (MPC), também conhecida como Controle de Horizonte Deslizante (do inglês, *Receding Horizon Control* ou RHC) corresponde a uma estratégia na qual a ação de controle atual é obtida através da solução *online* de um problema ótimo de controle em malha aberta com um horizonte finito, a cada instante de amostragem, utilizando o estado atual da planta/processo como estado inicial. Esse problema de otimização produz uma sequência de controle ótima, da qual somente o primeiro elemento é aplicado à planta (MAYNE et al., 2000).

A estratégia de controle do MPC envolve também a definição de uma função custo (ALAMIR, 2013). Essa função é utilizada para expressar o objetivo do controle, ou seja, quais as variáveis mais relevantes para o problema em questão, de modo que as mesmas possam ser minimizadas para alcançar os resultados ótimos descritos no parágrafo anterior.

De maneira resumida, as etapas do MPC podem ser descritas da seguinte forma (ALAMIR, 2013):

1. Definição de uma função custo que expresse o objetivo do controle;
2. A cada instante de amostragem, medir os estados da planta/processo sendo controlado;
3. Computar a sequência de ações futuras utilizando a função custo desenvolvida;
4. Aplicar a primeira ação da sequência de controle ótimo obtida no passo anterior;
5. Repetir os itens 2, 3 e 4 a cada novo instante de amostragem.

De maneira geral, a minimização da função custo envolve a otimização de um problema de programação quadrática, comumente conhecido como QP (do inglês, *Quadratic Programming*). Se o problema é linear, um QP é resolvido a cada instante de amostragem. Já no caso de um problema não linear, mais de um QP deve ser resolvido a cada instante de amostragem, dando origem à chamada programação quadrática sequencial (do inglês, *Sequencial Quadratic Programming* ou SQP) (FERREAU et al., 2017).

Felizmente, problemas de QP possuem soluções já consolidadas, tanto na literatura quanto em bibliotecas numéricas, para utilização em algoritmos de controle do MPC (ALAMIR, 2013).

O MPC não é um novo método de controle. Trata-se da resolução de problemas de controle ótimo, mas feita de maneira *online*, utilizando os estados atuais da planta/processo sendo controlado, diferentemente de outros tipos de controladores que realizam operações desta natureza de maneira *offline*, ou seja antes do momento de execução do algoritmo de controle (MAYNE et al., 2000). Isso, associado às características já citadas no capítulo anterior (capacidade de lidar com restrições, sistemas multivariáveis e sistemas não lineares) tornaram o MPC uma das técnicas de controle mais importantes das últimas décadas.

As próximas subseções deste trabalho são responsáveis por apresentar os fundamentos teóricos referentes ao MPC. A primeira delas apresenta o desenvolvimento histórico do MPC, enquanto a segunda é responsável por mostrar a formulação do MPC que foi utilizada neste trabalho.

2.1.2 Desenvolvimento histórico do MPC

O MPC foi introduzido há quase quatro décadas atrás e, desde então, vem se desenvolvendo bastante e sendo introduzido cada vez mais em sistemas de controle industriais (LEE, 2011). De fato, segundo esse mesmo autor, é possível dividir a história do MPC em três períodos distintos:

1. **Primeiro período:** introdução do MPC na indústria
2. **Segundo período:** fundamentação e solidificação da teoria do MPC
3. **Terceiro período:** diversificação e otimização do MPC

O primeiro período corresponde às primeiras citações das idéias de horizonte de controle deslizante e controle preditivo apresentadas na literatura. Segundo Lee (2011), características essenciais do MPC podem ser encontrados em projeto ainda mais antigos, que datam da década de 1950, com várias indústrias petroquímicas, como Texaco, Monsanto, Riverside Cement, Union Carbide e muitas outras (STOUT; WILLIAMS, 1995). Nos anos 60 e 70, a idéia do MPC continuou a se expandir e a aparecer esporadicamente na literatura, em trabalhos como Propoi (1963) e Lee(1967), que propuseram a programação de sistemas de controle lineares com restrições e a intenção de obter controladores em malha aberta que fossem capazes de medir o estado atual do processo e computá-lo rapidamente para a utilização no sistema de controle, respectivamente. É possível perceber que o último trata-se de uma das etapas base do MPC, apresentada na subseção 2.1.1.

Entretanto, somente no fim da década de 1970 / início da década de 1980 temos as primeiras publicações referentes ao MPC, sendo difícil atribuir o seu surgimento a uma pessoa ou grupo específico, uma vez que essas publicações surgiram de forma independente, mais ou menos no mesmo período (LEE, 2011). Aqui, destacam-se os trabalhos de Richalet *et al.* (1978) e de Cutler e Ramaker (1980), que introduziram as técnicas conhecidas como Controle Preditivo Baseado em Modelo Heurístico (em inglês, *Model Heuristic Predictive Control* ou MHPC) e Controle Dinâmico de Matrizes (do inglês, *Dynamic Matrix Control* ou DMC). O primeiro foi aplicado com sucesso em estudos de casos reportados em Richalet *et al.* (1978),

incluindo colunas de quebra de fluídos catalíticos, um gerador a vapor e uma planta de produção de policloreto de vinela (PVC), plástico proveniente do petróleo. Já o trabalho de Cutler e Ramaker (1980) obteve sucesso com aplicações de sistemas multivariáveis diversos.

É possível fazer algumas observações interessantes sobre esse período. A primeira delas corresponde ao fato de que os primeiros trabalhos referentes ao MPC foram publicados no final da década de 70 e no início da década de 80. Um dos fatores mais preponderantes para isso é que, a partir dessa época, a eletrônica havia alcançado um ponto de desenvolvimento que possibilitou o surgimento de microprocessadores mais poderosos e baratos, permitindo que a alta demanda computacional exigida por essa estratégia de controle começasse a ser suprida (LEE, 2011). Além disso, a segunda observação corresponde às aplicações desse período, principalmente relacionadas à indústria petroquímica, o que pode ser explicado graças à capacidade do MPC de lidar com sistemas multivariáveis e por, de maneira geral, a dinâmica desses tipos de sistemas serem mais lentas, o que viabiliza mais tempo de cálculo para o algoritmo de controle.

O segundo período de desenvolvimento do MPC corresponde a duas etapas importantes: avanços teóricos na formulação do MPC e a introdução de algoritmos comerciais do MPC em uma variedade maior de áreas industriais (LEE, 2011).

Graças aos avanços referentes à primeira etapa, uma formulação do MPC em espaço de estados passou a ser amplamente utilizada, o que possibilitou não somente uma base sólida para a utilização do MPC, como o surgimento de novas técnicas derivadas do mesmo. Destaca-se aqui o surgimento do NMPC, que até hoje é grande objeto de estudo na literatura por conta da sua dificuldade de implementação para certos tipos de sistema. Apesar de os resultados de estabilidade para o MPC serem válidos também para o NMPC, de modo que alguns dos trabalhos mais antigos que buscam resultados de estabilidade do MPC o fazem utilizando modelos genéricos não lineares como nos trabalhos de Lee e Markus (1967) e de Mayne e Michalska (1990), a implementação desses sistemas é em geral restrita, devida à complexidade computacional demandada para resolver os problemas de otimização não convexos inerentes a algumas aplicações (LEE, 2011).

No que se refere às aplicações do MPC durante esse período, obteve-se um crescimento da aplicação dessa técnica de controle em mais áreas da indústria. Uma pesquisa realizada em Qin e Badgewell (1997) com cinco grandes vendedores de soluções de controle contendo MPC mostrou um número total de 2233 aplicações que incluem aplicações na indústria petroquímica (ainda grande maioria), alimentícia, mineiração, papel, química e outros. Outra pesquisa com os mesmos vendedores (QIN; BADGWELL, 2003), realizada aproximadamente cinco anos depois da pesquisa citada anteriormente, revelou o dobro do número total de aplicações do MPC na indústria, incluindo um aumento de aplicações do NMPC, em aplicações como indústria química, ar/gás e indústria de polímeros (LEE, 2011).

Por fim, o último período de desenvolvimento histórico do MPC corresponde aos trabalhos que buscam otimizar e diversificar o MPC, possibilitando sua implementação em sistemas de dinâmicas mais rápidas. Algumas dessas tentativas levaram à criação de uma vertente do MPC, conhecida como MPC explícito (do inglês, *Explicit* MPC). Essa técnica, introduzida no trabalho de Bemporad, Morari e Dua (2002), possui o objetivo de determinar a solução do problema de otimização do MPC por meio de uma programação multi-paramétrica linear ou programação multi-paramétrica quadrática linear, realizada *offline*, ou seja, realizada antes da execução do algoritmo de controle. Esse método busca implementar a lei de controle por meio

de memórias e *Lookup tables*, mapeando diferentes realimentações de estado para cada situação que surja durante a execução do algoritmo, conhecida como região poliédrica (do inglês, *polyhedral region*).

O problema com essa abordagem é que, por conta da dimensão dos problemas de controle e, por ser necessário estabelecer todas as situações que ocorrerão durante a execução do algoritmo de antemão, é necessária uma grande quantidade de recursos de *hardware* e memória, o que torna difícil sua implementação em sistemas de controle reais. Entretanto, existem diversos trabalhos na literatura que buscam uma solução para esse tipo de problema, tornando o MPC explícito uma opção mais viável para implementação em sistemas de controle. Em Johansen e Crancharova (2003), um método que realiza uma aproximação explícita linear por partes para a realimentação dos estados é proposto. Utiliza-se uma estrutura de árvore de busca para diminuir a complexidade computacional de implementação do MPC explícito logaritmicamente.

Há ainda trabalhos que apresentam estratégias para evitar o mapeamento de situações desnecessárias analisando a geometria de cada região poliédrica, consumindo menos recursos de *hardware* para a implementação do MPC explícito (TØNDEL; JOHANSEN; BEMPORAD, 2003), e um método para resolver o problema de programação multi-paramétrica quadrática de maneira aproximada (BEMPORAD; FILIPPI, 2003). Isso melhora o desempenho do algoritmo de controle e propõe um *tradeoff* entre otimização e um menor número de regiões poliédricas a serem mapeadas, respectivamente.

A segunda principal vertente que foi desenvolvida durante esse período é o ganho de desempenho nas estratégias de otimização *online* do MPC, ou seja, durante a execução do algoritmo. Existem diversos tipos de algoritmos que foram e continuam sendo propostos na literatura. Segundo Patrascu e Necoara (2016), esses algortimos podem ser divididos em diferentes tipos:

- **Método do ponto interior:** do inglês, *interior point method*, foi proposto por Rao, Wright e Rawlings (1998). São métodos de segunda ordem que removem as restrições de inequalidade da função custo por meio de outra função (denominada função barreira) e penalizam as violações de restrições. De maneira geral, esses métodos são utilizados para sistemas de pequena e média escala, pois sua complexidade aumenta consideravelmente de acordo com a dimensão do problema. Entretanto, soluções têm surgido nas últimas décadas que utilizam métodos de ponto interior para gerar *solvers* de QP voltados para controle preditivo, como CVXGEN (MATTINGLEY; BOYD, 2009) e FORCES (DOMAHIDI et al., 2012), buscando aplicar esse método a sistemas de maiores dimensões. Trabalhos na literatura que utilizam esse método podem ser encontrados em Domahidi *et al.* (2012), que propõe implementações mais eficientes e detalhes que otimizam os algoritmos dessa classe, mostrando resultados numéricos referentes ao desempenho do algoritmo; e Zhang, Ferranti e Keviczky (2017), que propõe uma implementação diferente unindo o método do ponto interior com o método de gradiente rápido de Nesterov para obter uma melhora no desempenho do mesmo.

- **Métodos de *Active set***: são algoritmos baseados na equivalência entre soluções de um QP com restrições e um sistema linear específico gerado pelas condições de otimização do QP em questão. O objetivo principal dessa classe de algoritmos é estimar um conjunto de restrições que são satisfeitas quando avaliada a solução do problema original, denominado *active set*. É composto por duas fases:

    1. Escolha de um ponto factível que é computado pelo algoritmo, sem levar em consideração informações da função de custo do problema.

2. A função de custo do problema é minimizada, levando em consideração o ponto factível escolhido na primeira etapa.

Essa classe de algoritmos geralmente é indicada para sistemas de pequena e média ordem, mas assim como no caso do método do ponto interior, vêm recebendo significativas contribuições de otimização para aplicação do mesmo a sistemas de maiores dimensões. Exemplos de algoritmos que aplicam desse método *active set* são dados pela função *quadprog* do *software* MATLAB, bem como em Ferreau, Bock e Diehl (2008), que mais tarde daria origem ao *solver* conhecido como qpOASES (FERREAU et al., 2014), muito utilizado hoje em dia para implementação do MPC embarcado em diferentes plataformas de *hardware*. Trabalhos na literatura, como Khan e Rossiter (2012) e Cai *et al.* (2014) apresentam, respectivamente, um estudo comparativo de diferentes estratégias de controle baseadas no método *active set* e uma proposição de implementação mais rápida do método, por meio de uma estratégia dual: se a solução sem restrições encontra-se distante do ponto de referência do sistema, a lei de controle aplicada é a solução sem restrições movimentando-se para o limite da restrição; caso contrário, a solução sem restrições projetada para o limite da restrição é utilizada como ponto inicial da próxima iteração do algoritmo.

- **Métodos de primeira ordem:** do inglês, *first order method*, esse tipo de algoritmo utiliza informação de gradiente a cada iteração, de forma a encontrar uma solução para o problema sem restrições. Uma vez que essa solução é obtida, as restrições são aplicadas. Para um sistema com restrições simples e pouco limitadores, é relativamente fácil realizar essa operação. Entretanto, se as restrições do sistema são difíceis de computar, uma alternativa é a utilização de técnicas de relaxamento de Lagrange, o que gera um algoritmo do tipo ordem dual.

  Um exemplo de *solver* que utiliza esse tipo de estratégia é o *open-source* FiOrdOs (ULLMANN, 2011). O trabalho de Kouzoupis *et al.* (2015) analisa a complexidade desse tipo de algoritmo e, por meio do controle de um sistema não linear de pêndulo invertido, mostra que é possível obter ganhos de desempenho satisfatórios em termos de tempo de execução do MPC. Posteriormente, o trabalho de Kufoalor *et al.* (2017) apresentou uma evolução desse método por meio de uma modificação na formulação das restrições e penalizações do sistema. Com isso, foi possível obter um resultado que levou a um desempenho melhor que os demais métodos de primeira e segunda ordem presentes na literatura, aplicados a um sistema separador compactador, que separa um liquido composto por água e óleo.

- **Método de ordem dual:** do inglês, *dual order method*, trata-se de uma abordagem em que são utilizados multiplicadores de Lagrange para resolver restrições mais complexas que inviabilizam a utilização do método descrito anteriormente (primeira ordem). O maior esforço computacional corresponde à resolução de um problema QP Lagrangiano a cada instante de tempo, com restrições simples aplicadas a um multiplicador de Lagrange. Essas operações são efetuadas por meio de operações matriciais, que normalmente envolvem a inversão de uma matriz. Portanto, quanto maior a ordem do sistema, mais complexo torna-se o problema e mais difícil torna-se respeitar as restrições do mesmo a cada instante de tempo.

Analisando todas as etapas do desenvolvimento do MPC, é interessante notar que um fator permeia toda a evolução do algoritmo: o objetivo de diminuir a complexidade computacional para implementação

do mesmo. O MPC possui inúmeras vantagens que são interessantes para diversas aplicações, mas por conta da alta necessidade de desempenho computacional para solucionar os problemas de otimização de QP *online* a cada instante de amostragem, trata-se de um grande desafio para sistemas de altas dimensões e tempos de amostragem baixos. Por isso, a literatura continua recebendo publicações com esse objetivo: otimização do MPC para aplicação do mesmo a sistemas cada vez mais complexos e de dinâmicas cada vez mais rápidas.

2.1.3 Fundamentação teórica do MPC

Este trabalho aborda a formulação linear do MPC. As formulações e demonstrações matemáticas dessa seção são baseadas nos capítulos de 2 a 6 de Alamir (2013). Para maiores detalhes acerca desses tópicos, o leitor é convidado a consultar a obra original.

A formulação do MPC será divida em cinco partes: os conceitos básicos para fomulação do MPC, sua aplicação a sistemas lineares e invariantes no tempo (do inglês, *Linear Time Invariant* ou LTI), formulação do MPC inicialmente sem restrições, formulação do MPC com aplicação das restrições e, por fim, a parametrização de controle que é aplicada ao MPC (utilizada para diminuir a complexidade computacional de implementação do mesmo).

2.1.3.1 Conceitos básicos do MPC

O primeiro conceito básico que é importante para o entendimento do MPC é o de controle em espaço de estados. Este conceito é importante pois a formulação linear do MPC baseia-se no controle de sistemas LTI, e uma das maneiras mais comuns de se trabalhar com esse tipo de sistema, e desenvolver algoritmos de controle para os mesmos, é justamente por meio de uma formulação em espaço de estados. Essa formulação é muito utilizada na teoria de controle moderno, pois oferece uma metodologia para trabalhar com sistemas multivariáveis, que podem ter interconexões complexas entre suas entradas e saídas. A análise em espaço de estados é dada através da representação das equações dinâmicas de um sistema por meio de uma equação diferencial com notação vetor-matricial de primeira ordem (OGATA, 2002). Os chamados estados são as variáveis necessárias para descrever o comportamento do sistema em um determinado espaço de tempo. O espaço *n*-dimensional que consiste de vários eixos que representam cada variável de estado é denominado espaço de estados, de modo que cada estado do sistema pode ser representado por um ponto nesse espaço de estados.

Abaixo, na Equação 2.1, é apresentada a formulação de um sistema linear genérica em espaço de estados que será utilizada como base para as demais formulações desta seção.

$$\begin{cases} x(k+1) &= \mathbf{A}x(k) + \mathbf{B}u(k) \\ y_r(k) &= \mathbf{C_r}x(k) \\ y_c(k) &= \mathbf{C_c}x(k) + \mathbf{D_c}u(k) \end{cases} . \tag{2.1}$$

Na Equação 2.1, há matrizes (em negrito) e vetores que são necessários para modelar o comportamento dinâmico de um sistema em espaço de estados. Essas matrizes e vetores são:

- x: vetor de estados do sistema;
- u: vetor de variáveis de controle do sistema;
- y_r: vetor de saídas reguladas do sistema;
- y_c: vetor de saídas restringidas, ou seja, saídas sob as quais são aplicadas as restrições do sistema;
- $\mathbf{A}$: matriz de estados do sistema;
- $\mathbf{B}$: matriz de entradas do sistema;
- $\mathbf{C_r}$: matriz de saídas reguladas do sistema;
- $\mathbf{C_c}$: matriz de saídas restringidas do sistema;
- $\mathbf{D_c}$: matriz de transmissão direta das saídas restringidas do sistema.

Em muitas das aplicações de sistemas LTI, a matriz $\mathbf{D_c}$ corresponde a uma matriz nula. O mesmo se aplica ao sistema estudado neste trabalho, o quadrirrotor, que será abordado com mais detalhes na próxima seção.

Com as definições acima, é possível dar sequência aos demais conceitos básicos da formulação do MPC. Conforme abordado em seções anteriores, o MPC calcula, a cada instante de amostragem, uma sequência de variáveis de controle, que será denominada $\widetilde{u}$. Cada elemento dessa sequência corresponde a uma ação de controle futura que será aplicada em cada instante de amostragem, sendo o número total de ações futuras utilizadas para determinação das ações de controle um parâmetro denominado horizonte de predição, cuja notação adotada será *N*. Portanto, o MPC, a cada instante de amostragem, calcula uma sequência de N ações futuras de controle e armazena cada uma no vetor $\widetilde{u}$, que pode ser visto na Equação 2.2.

$$\widetilde{u}(k) := \begin{pmatrix} u(k) \\ u(k+1) \\ \vdots \\ u(k+N-1) \end{pmatrix} \in \mathbb{R}^{(N.n_u)}. \tag{2.2}$$

Na Equação acima, n_u refere-se ao número de variáveis de controle do sistema. O índice k refere-se ao instante de amostragem atual do sistema. Da mesma forma, $k-1$ refere-se ao instante de amostragem anterior e $k+1$, ao próximo instante de amostragem do sistema.

De maneira análoga, o sistema define uma sequência de estados calculada ao longo do horizonte de predição, denominada $\widetilde{x}$, que pode ser vista na Equação 2.3.

$$\widetilde{x}(k) := \begin{pmatrix} x(k+1) \\ x(k+2) \\ \vdots \\ x(k+N) \end{pmatrix} \in \mathbb{R}^{(N.n_x)}. \tag{2.3}$$

De maneira análoga a n_u, n_x refere-se ao número de estados do sistema.

Com as definições de $\widetilde{x}$ e $\widetilde{u}$, é possível estabelecer um mapa de predição de N passos a frente: para cada sequência de controle $\widetilde{u}$ calculada, é possível obter a trajetória correspondente do sistema $\widetilde{x}(k|\widetilde{u}(k))$, ou seja, dos estados do sistema, dado o seu estado inicial. O mapa de predição descrito pode ser evidenciado na Figura 2.1.

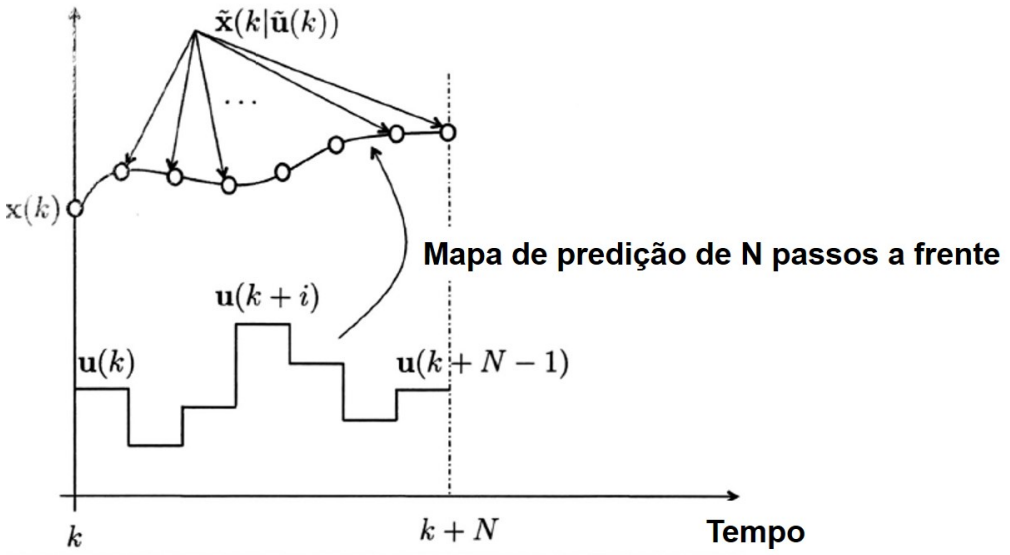

Figura 2.1: Mapa de predição de *N* passos a frente: para cada sequência de controle $\widetilde{u}$ calculada, uma trajetória $\tilde{\mathbf{x}}(\mathbf{k}|\widetilde{\mathbf{u}}(\mathbf{k}))$ correspondente é obtida com base no estado inicial $x(k)$ do sistema, ao longo de todo o horizonte de predição adotado. Fonte: (ALAMIR, 2013).

O símbolo | em $\widetilde{x}(k|\widetilde{u}(k))$ indica que $\widetilde{x}(k)$ é calculada utilizando a informação de $\widetilde{u}(k)$, que também está disponível no instante de amostragem k.

Uma característica importante do MPC é que o mesmo calcula variáveis de controle referentes a *N* ações futuras, mas aplica somente a(s) primeira(s) variável(is) de controle da sequência $\widetilde{u}$. Isso ocorre porque, a cada instante de amostragem, o sistema está sujeito a novos tipos de intereferência e/ou perturbações que não foram levadas em consideração no cálculo anterior de $\widetilde{u}$. Assim, aplicando somente o primeiro elemento da sequência do sistema, aumenta-se a robustez do controlador contra esse tipo de situação.

Pode-se pensar que isso gera um cálculo desnecessário, já que são computadas várias ações de controle que não serão aplicadas ao sistema, mas quanto mais ações futuras o MPC tém acesso (ou seja, quanto maior o valor de *N*), maiores as chances de estabilidade no cálculo das variáveis de controle. Em outras palavras, quanto mais informação o MPC possui para tomar as decisões de controle que minimizam a função custo do problema, melhor será a solução encontrada. Entretanto, mesmo que horizontes de predição maiores auxiliem na estabilidade do sistema, quanto maior o valor de *N*, maiores as dimensões do problema e maior é a demanda computacional exigida para a implementação do MPC.

A seleção do primeiro elemento da sequência de controle $\widetilde{u}$ é feita através da matriz de seleção $\Pi_i^{(n_i,N)}$. Essa matriz é responsável por selecionar o i-ésimo elemento de uma matriz composta pela concatenação de *N* vetores, ou seja, para uma sequência de N variáveis de controle, a matriz seleciona a i-ésima ação de controle calculada. A operação da matriz de seleção pode ser evidenciado na Figura 2.2.

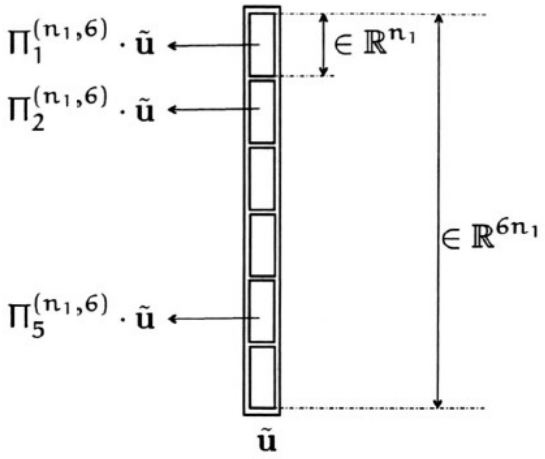

Figura 2.2: Funcionamento da matriz de seleção $\mathbf{\Pi_i^{(n_l,N)}}$: seleciona o i-ésimo vetor de dimensão n_l de um vetor composto por *N* concatenações de vetores. Na figura, tém-se o exemplo de um problema de controle com horizonte de predição *N* = 6. Fonte: (ALAMIR, 2013).

Portanto, se deseja-se aplicar no próximo instante de amostragem a(s) primeira(s) variável(is) de controle calculadas, têm que o vetor de controle $u(k+i)$ é dado pela Equação 2.4:

$$u(k+i-1) = \mathbf{\Pi_i^{(n_u,N)}}.\widetilde{u}(k). \tag{2.4}$$

A próxima etapa corresponde à formulação da função custo do problema de controle. Essa função deve retornar um escalar, que permita ao algoritmo de controle comparar as diferentes sequências de ações futuras que podem ser executadas e possuir um parâmetro para realizar uma escolha, ou seja, selecionar qual ação futura minimiza os objetivos de controle expressos na função custo. Um exemplo de função custo que pode ser adotada é dada pela Equação 2.5.

$$J(\widetilde{u}|x(k)) := ||\mathbf{\Pi_N^{(n,N)}}.\widetilde{x}(k|\widetilde{u})||^2_{Q_f} + \sum_{i=1}^{(N-1)} ||\mathbf{\Pi_i^{(n,N)}}.\widetilde{x}(k|\widetilde{u}) - x^d||^2_Q + \sum_{i=1}^{(N-1)} ||\mathbf{\Pi_i^{(n_u,N)}}.\widetilde{u} - u^d||^2_R. \tag{2.5}$$

Na Equação 2.5, a notação $||v||^2_Q$ corresponde à operação v^TQv, que resulta em um escalar. As matrizes $\mathbf{Q_f}$, $\mathbf{Q}$ e $\mathbf{R}$ são matrizes de ponderação, definidas e sintonizadas para cada problema de controle estudado. Os termos x^d e u^d correspondem aos valores desejados para os estados e variáveis de controle do sistema, respectivamente.

Essa função custo é interessante para problemas de controle, uma vez que cada termo que compõem a mesma é responsável por um aspecto que representa o objetivo de controle desejado. O primeiro termo ($||\mathbf{\Pi_N^{(n,N)}}.\widetilde{x}(k|\widetilde{u}||^2_{Q_f}$), penaliza o erro final do sistema, também conhecido como erro de estado estacionário, ou seja, o quão longe o valor ótimo atual dos estados encontra-se do valor desejado dos mesmos (x^d). O segundo termo ($\sum_{i=1}^{(N-1)} ||\mathbf{\Pi_i^{n,N}}.\widetilde{x}(k|\widetilde{u}) - x^d||^2_Q$), penaliza os erros intermediários do sistema, ou seja, o somatório da diferença entre cada estado ótimo atual e o valor desejado para o mesmo. Por fim, o terceiro

e último termo ($\sum_{i=1}^{N-1} ||\mathbf{\Pi_i^{(n_u,N)}}.\widetilde{u} - u^d||_R^2$) é responsável pela penalização da divergência de controle, ou seja, o quanto as variáveis de controle calculadas ($\widetilde{u}$) estão distantes do valor desejado para as mesmas (u^d).

Por fim, é necessário que sejam aplicadas as restrições do sistema. É comum expressar essas restrições por meio da Equação 2.6.

$$g(\widetilde{u}|x(k)) \leq 0. \tag{2.6}$$

A Equação 2.6 significa que o conjunto de sequências de ações possíveis (admissíveis) que minimizam a função custo do problema de controle são as sequências de controle $\widetilde{u}$ que satisfazem a restrição de desigualdade $g(\widetilde{u}|x(k)) \leq 0$, e somente essas sequências de controle podem ser escolhidas pelo MPC.

Portanto, para o MPC linear, têm-se o seguinte problema de otimização $P(x(k))$, que deve ser resolvido a cada instante de amostragem k do sistema, expresso na Equação 2.7.

$$P(x(k)) : \min_{\widetilde{u} \in \mathbb{R}^{(N.n_u)}} J(\widetilde{u}|x(k)) \quad \text{mediante} \quad g(\widetilde{u}|x(k)) \leq 0. \tag{2.7}$$

2.1.3.2 Aplicação do MPC a sistemas LTI

Uma vez que os conceitos básicos do MPC e o problema de otimização que deve ser resolvido a cada instante de amostragem (Equação 2.7) foram apresentados, é possível aplicar o MPC aos sistemas que deseja-se controlar.

Segundo as definições apresentadas no capítulo 3 de Alamir(2013), ao aplicar o MPC a sistemas LTI, a definição da função custo pode ser modificada, conforme mostrado na Equação 2.8 abaixo.

$$\begin{aligned} J(\widetilde{u}|x(k), \widetilde{y}_r^d(k)) &= \widetilde{u}^T \left[\sum_{i=1}^{N} \mathbf{\Psi_i}^T \mathbf{C_r}^T \mathbf{Q_y} \mathbf{C_r} \mathbf{\Psi_i}\right] \widetilde{u} \\ &+ \left[\left(2\sum_{i=1}^{N} \mathbf{\Psi_i}^T \mathbf{C_r}^T \mathbf{Q_y} \mathbf{C_r} \mathbf{\Phi_i}\right) x(k) - \left(2\sum_{i=1}^{N} \mathbf{\Psi_i}^T \mathbf{C_r}^T \mathbf{Q_y} \mathbf{\Pi_i^{(n_r,N)}}\right) \widetilde{y}_r^d(k)\right]^T \widetilde{u} \\ &+ ||\sum_{i=1}^{N} \left(\mathbf{C_r} \mathbf{\Phi_i} x(k) - y_r^d(k+i)\right)||_{\mathbf{Q_y}}^2. \end{aligned} \tag{2.8}$$

A Equação 2.8 possui a matriz de ponderação das saídas $\mathbf{Q_y}$, que é sintonizada para cada problema de controle, e as matrizes $\mathbf{\Psi_i}$ e $\mathbf{\Phi_i}$, que são constantes e contêm manipulações algébricas envolvendo as matrizes de estados $\mathbf{A}$ e de entradas $\mathbf{B}$ do modelo do sistema sendo controlado em espaço de estados. É possível ainda perceber que a função custo é uma função quadrática na variável de decisão $\widetilde{u}$, o QP citado em subseções anteriores. Esse tipo de função quadrática é geralmente dado na forma expressa na Equação 2.9:

$$\frac{1}{2}\widetilde{u}^T \mathbf{H} \widetilde{u} + \mathbf{F}^T \widetilde{u} + \text{constante}. \tag{2.9}$$

Na Equação 2.9, há a presença de uma matriz $\mathbf{H}$, conhecida como matriz Hessiana, que é um elemento

muito importante para a resolução de QPs. De acordo com a Equação 2.8, $\mathbf{H}$ pode ser definido como na Equação 2.10:

$$\mathbf{H} = \left[2 \sum_{i=1}^{N} \mathbf{\Psi_i}^T \mathbf{C_r}^T \mathbf{Q_y} \mathbf{C_r} \mathbf{\Psi_i} \right] \in \mathbb{R}^{(N.n_r).(N.n_r)}. \tag{2.10}$$

O problema de otimização exige que a função custo seja minimizada, e para que isso seja possível, a matriz $\mathbf{H}$ deve ser positivo-definida. Caso isso não ocorra, a função custo pode decair indefinidamente ou permanecer constante, o que não permite obter a solução ótima desejada. As condições para a matriz $\mathbf{H}$ ser considerada positivo-definida, podem ser vistas logo abaixo, na Equação 2.11.

$$\widetilde{u} \in \mathbb{R}^{(N.n_u)}, \quad \widetilde{u}^T \mathbf{H} \widetilde{u} \geq \lambda_{min} ||\widetilde{u}||^2 \quad \text{para algum} \;\; \lambda_{min} > 0. \tag{2.11}$$

É interessante notar que, para $\mathbf{H}$ ser positivo-definida, não é suficiente que a matriz de ponderação $\mathbf{Q_y}$ seja positivo-definida. Para garantir essa condição, adiciona-se o termo presente na Equação 2.5, tratando-se justamente da parcela que penaliza o desvio de controle. Expandindo esse segundo termo, é possível obter a Equação 2.12.

$$\sum_{i=1}^{N} \left\| \mathbf{\Pi_i}^{(n_u,N)} \widetilde{u} - u^d \right\|_{Q_u}^2 = \widetilde{u}^T \left[\sum_{i=1}^{N} \left(\mathbf{\Pi_i^{(n_u,N)}} \right)^T \mathbf{Q_u} \left(\mathbf{\Pi_i^{(n_u,N)}} \right) \right] \widetilde{u} + \left[2 \sum_{i=1}^{N} \left(\mathbf{\Pi_i^{(n_u,N)}} \right)^T \mathbf{Q_u} u^d \right]^T \widetilde{u} + \left\| u^d \right\|_{Q_u}^2. \tag{2.12}$$

Com isso, as matrizes $\mathbf{H}$ e $\mathbf{F}$, associadas à função custo quadrática do problema de otimização do MPC, podem ser expressas como disposto abaixo, na Equação 2.13:

$$\begin{cases} \mathbf{H} &= 2 \sum_{i=1}^{N} \left[\mathbf{\Psi_i}^T \mathbf{C_r}^T \mathbf{Q_y} \mathbf{C_r} \mathbf{\Psi_i} + \left(\mathbf{\Pi_i^{(n_u,N)}} \right)^T \mathbf{Q_u} \left(\mathbf{\Pi_i^{(n_u,N)}} \right) \right] \\ \mathbf{F} &= \mathbf{F_1} x(k) + \mathbf{F_2} \widetilde{y}_r^d + \mathbf{F_3} u^d \\ \mathbf{F_1} &= 2 \sum_{i=1}^{N} \left[\mathbf{\Psi_i}^T \mathbf{C_r}^T \mathbf{Q_y} \mathbf{C_r} \mathbf{\Phi_i} \right] \\ \mathbf{F_2} &= -2 \sum_{i=1}^{N} \left[\mathbf{\Psi_i}^T \mathbf{C_r}^T \mathbf{Q_y} \mathbf{\Pi_i^{(n_r,N)}} \right] \\ \mathbf{F_3} &= 2 \sum_{i=1}^{N} \left[\left(\mathbf{\Pi_i^{(n_u,N)}} \right)^T \mathbf{Q_u} \right] \end{cases}. \tag{2.13}$$

Assim como na subseção anterior, para aplicar o MPC a sistemas lineares é necessário aplicar os conceitos das restrições do sistema. Para isso, utiliza-se a última linha da Equação 2.1 ($y_c(k+1) = \mathbf{C_c} x(k) + \mathbf{D_c} u(k)$), que refere-se justamente às saídas restringidas do sistema. As restrições aplicadas ao sistema são divididas em três categorias: restrição nas saídas, na variação das variáveis de controle e nas variáveis de controle em si. Cada uma dessas categorias de restrição é delimitada por valores mínimos e máximos, conforme mostrado na Equação 2.14 abaixo.

$$\begin{cases} y_c^{min} \leq y_c \leq y_c^{max} \\ \delta^{min} \leq u(k+i) - u(k+i-1) \leq \delta^{max} \\ \widetilde{u}^{min} \leq \widetilde{u} \leq \widetilde{u}^{max} \end{cases}. \tag{2.14}$$

Substituindo $y_c(k+1) = \mathbf{C_c}x(k) + \mathbf{D_c}u(k)$ na primeira linha da Equação 2.14, obtêm-se a Equação 2.15, que pode ser manipulada e expandida para obter a Equação 2.19.

$$y_c^{min} \leq y_c(k+i) = \mathbf{C_c}x(k+i) + \mathbf{D_c}u(k+i-1) \leq y_c^{max}, \tag{2.15}$$

$$y_c^{min} \leq \mathbf{C_c}\mathbf{\Pi_i^{(n,N)}}\widetilde{x}(k) + \mathbf{D_c}\mathbf{\Pi_i^{(n_u,N)}}\widetilde{u}(k) \leq y_c^{max}, \tag{2.16}$$

$$\begin{pmatrix} y_c^{min} \\ \vdots \\ y_c^{min} \end{pmatrix} \leq \begin{pmatrix} \mathbf{C_c}\mathbf{\Pi_1^{(n,N)}} \\ \vdots \\ \mathbf{C_c}\mathbf{\Pi_N^{(n,N)}} \end{pmatrix} \underbrace{\widetilde{x}(k)}_{\mathbf{\Phi}x(k)+\mathbf{\Psi}\widetilde{u}} + \begin{pmatrix} \mathbf{D_c}\mathbf{\Pi_1^{(n_u,N)}} \\ \vdots \\ \mathbf{D_c}\mathbf{\Pi_N^{(n_u,N)}} \end{pmatrix} \widetilde{u}(k) \leq \begin{pmatrix} y_c^{max} \\ \vdots \\ y_c^{max} \end{pmatrix}, \tag{2.17}$$

$$\begin{pmatrix} y_c^{min} \\ \vdots \\ y_c^{min} \end{pmatrix} \leq \begin{pmatrix} \mathbf{C_c}\mathbf{\Pi_1^{(n,N)}}\mathbf{\Phi} \\ \vdots \\ \mathbf{C_c}\mathbf{\Pi_N^{(n,N)}}\mathbf{\Phi} \end{pmatrix} x(k) + \begin{pmatrix} \mathbf{C_c}\mathbf{\Pi_1^{(n,N)}}\mathbf{\Psi} + \mathbf{D_c}\mathbf{\Pi_1^{(n_u,N)}} \\ \vdots \\ \mathbf{C_c}\mathbf{\Pi_N^{(n,N)}}\mathbf{\Psi} + \mathbf{\Psi}\mathbf{D_c}\mathbf{\Pi_N^{(n_u,N)}} \end{pmatrix} \widetilde{u} \leq \begin{pmatrix} y_c^{max} \\ \vdots \\ y_c^{max} \end{pmatrix}, \tag{2.18}$$

$$\underbrace{\begin{pmatrix} +\mathbf{C_c}\mathbf{\Psi_1} + \mathbf{D_c}\mathbf{\Pi_1^{n_u,N}} \\ \vdots \\ +\mathbf{C_c}\mathbf{\Psi_N} + \mathbf{D_c}\mathbf{\Pi_N^{n_u,N}} \\ -\mathbf{C_c}\mathbf{\Psi_1} - \mathbf{D_c}\mathbf{\Pi_1^{n_u,N}} \\ \vdots \\ -\mathbf{C_c}\mathbf{\Psi_N} - \mathbf{D_c}\mathbf{\Pi_N^{n_u,N}} \end{pmatrix}}_{\mathbf{A_{ineq}^{(1)}}} \widetilde{u} \leq \underbrace{\begin{pmatrix} -\mathbf{C_c}\mathbf{\Phi_1} \\ \vdots \\ -\mathbf{C_c}\mathbf{\Phi_N} \\ +\mathbf{C_c}\mathbf{\Phi_1} \\ \vdots \\ +\mathbf{C_c}\mathbf{\Phi_N} \end{pmatrix}}_{\mathbf{G_1^{(1)}}} x(k) + \underbrace{\begin{pmatrix} +y_c^{max} \\ \vdots \\ +y_c^{max} \\ -y_c^{min} \\ \vdots \\ -y_c^{min} \end{pmatrix}}_{\mathbf{G_3^{(1)}}}. \tag{2.19}$$

De maneira análoga, as restrições de variação do comando ($\delta^{min} \leq u(k+i) - u(k+i-1) \leq \delta^{max}$) podem ser expandidas e rearranjadas, originando a Equação 2.21.

$$\begin{pmatrix} \delta^{min} \\ \delta^{min} \\ \vdots \\ \delta^{min} \end{pmatrix} \leq \begin{pmatrix} +\mathbb{I} & \mathbb{0} & \mathbb{0} & \dots & \mathbb{0} & \mathbb{0} \\ -\mathbb{I} & +\mathbb{I} & \mathbb{0} & \dots & \mathbb{0} & \mathbb{0} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \mathbb{0} & \mathbb{0} & \mathbb{0} & \dots & -\mathbb{I} & +\mathbb{I} \end{pmatrix} \widetilde{u} + \begin{pmatrix} -\mathbb{I} \\ \mathbb{0} \\ \vdots \\ \mathbb{0} \end{pmatrix} u(k-1) \leq \begin{pmatrix} \delta^{max} \\ \delta^{max} \\ \dots \\ \delta^{max} \end{pmatrix}, \tag{2.20}$$

$$\underbrace{\begin{pmatrix} +\mathbb{I} & \mathbb{O} & \mathbb{O} & \dots & \mathbb{O} & \mathbb{O} \\ -\mathbb{I} & +\mathbb{I} & \mathbb{O} & \dots & \mathbb{O} & \mathbb{O} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \mathbb{O} & \mathbb{O} & \mathbb{O} & \dots & -\mathbb{I} & +\mathbb{I} \\ -\mathbb{I} & \mathbb{O} & \mathbb{O} & \dots & \mathbb{O} & \mathbb{O} \\ +\mathbb{I} & -\mathbb{I} & \mathbb{O} & \dots & \mathbb{O} & \mathbb{O} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \mathbb{O} & \mathbb{O} & \mathbb{O} & \dots & \mathbb{I} & -\mathbb{I} \end{pmatrix}}_{\mathbf{A_{ineq}^{(2)}}} \widetilde{u} \leq \underbrace{\begin{pmatrix} +\mathbb{I} \\ \mathbb{O} \\ \vdots \\ \mathbb{O} \\ -\mathbb{I} \\ \mathbb{O} \\ \vdots \\ \mathbb{O} \end{pmatrix}}_{\mathbf{G_2^{(2)}}} u(k-1) + \underbrace{\begin{pmatrix} +\delta^{max} \\ +\delta^{max} \\ \vdots \\ +\delta^{max} \\ -\delta^{min} \\ -\delta^{min} \\ \vdots \\ -\delta^{min} \end{pmatrix}}_{\mathbf{G_3^{(2)}}} . \tag{2.21}$$

As restrições aplicadas às variáveis de controle podem ser arranjadas de maneira mais simples, conforme mostrado na Equação 2.23 abaixo:

$$\underbrace{\begin{pmatrix} u^{min} \\ \vdots \\ u^{min} \end{pmatrix}}_{\widetilde{u}^{min}} \leq \widetilde{u} \leq \underbrace{\begin{pmatrix} u^{max} \\ \vdots \\ u^{max} \end{pmatrix}}_{\widetilde{u}^{max}} \in \mathbb{R}^{Nn_u}, \tag{2.22}$$

$$\widetilde{u}^{min} \leq \widetilde{u} \leq \widetilde{u}^{max}. \tag{2.23}$$

Com todas as matrizes de restrição elaboradas, pode-se reorganizar todas as restrições, obtendo-se as Equações 2.24 e 2.25:

$$\begin{cases} \mathbf{A_{ineq}}\widetilde{u} \leq \mathbf{B_{ineq}} \\ \widetilde{u}^{min} \leq \widetilde{u} \leq \widetilde{u}^{max} \end{cases}, \tag{2.24}$$

$$\begin{cases} \mathbf{A_{ineq}} = \begin{pmatrix} \mathbf{A_{ineq}^{(1)}} \\ \mathbf{A_{ineq}^{(2)}} \end{pmatrix} \\ \mathbf{B_{ineq}} = \mathbf{G_1}x(k) + \mathbf{G_2}u(k-1) + \mathbf{G_3} \\ \mathbf{G_1} = \begin{pmatrix} \mathbf{G_1^{(1)}} \\ \mathbb{O}_{(2N.n_u).n} \end{pmatrix} \\ \mathbf{G_2} = \begin{pmatrix} \mathbb{O}_{(2N.n_c).n_u} \\ \mathbf{G_2^{(2)}} \end{pmatrix} \\ \mathbf{G_3} = \begin{pmatrix} \mathbf{G_3^{(1)}} \\ \mathbf{G_3^{(2)}} \end{pmatrix} \end{cases} . \tag{2.25}$$

Portanto, as Equações 2.13 e 2.25 apresentam todas as matrizes necessárias para a resolução do QP inerente ao MPC. Algumas dessas matrizes podem ser computadas *offline*, ou seja, permanecem constantes durante a execução do algoritmo de controle: $\mathbf{A_{ineq}}$, $\mathbf{H}$, $\mathbf{F_1}$, $\mathbf{F_2}$, $\mathbf{F_3}$, $\mathbf{G_1}$, $\mathbf{G_2}$ e $\mathbf{G_3}$. Já a computação das matrizes $\mathbf{F}$ e $\mathbf{B_{ineq}}$ devem ser calculadas *online*, pois a cada instante de amostragem estão sendo modificados por dependerem dos valores dos estados e variáveis de controle do sistema. Sejam as matrizes de custo e restrição do MPC calculadas *online* ou *offline*, elas são utilizadas na resolução do QP e são parte fundamental para a implementação do MPC.

2.1.4 Formulação do MPC sem restrições

As matrizes calculadas na subseção anterior podem ser divididas em dois grupos: as matrizes de custo, representadas por $\mathbf{F1}$, $\mathbf{F2}$ e $\mathbf{F3}$; e as matrizes de restrições do MPC, compostas por $\mathbf{A_{ineq}}$, $\mathbf{B_{ineq}}$, $\mathbf{G1}$, $\mathbf{G2}$ e $\mathbf{G3}$.

As matrizes de custo estão relacionadas com as ponderações de saída e de controle, e são utilizadas na formulação do MPC para encontrar o valor que satisfaça os objetivos de controle, minimizando a função custo, definida na Equação 2.8. Já as matrizes de restrições são responsáveis por incorporar as restrições do problema à formulação do controlador, garantindo que a função custo seja minimizada e que as restrições do problema de controle sejam respeitadas. É a combinação de ambas que faz com que o MPC seja capaz de controlar o sistema, garantindo as restrições do mesmo, a principal vantagem desta estratégia de controle.

Entretanto, caso não deseje-se incorporar as restrições do problema de controle (seja por motivos de complexidade desse cenário ou no caso de uma aplicação na qual dificilmente condições de restrição são atingidas), é possível desenvolver uma lei de controle que minimize a função custo, sem levar em consideração as matrizes de restrições. Assim, é possível establizar o sistema nos valores desejados, seguindo referências desejadas, mas não se tem a garantia de que as restrições do problema de controle serão cumpridas.

A abordagem do MPC sem restrições pode parecer incoerente, uma vez que ela não possui a grande vantagem do MPC: o respeito às restrições. Entretanto, por se tratar de uma formulação mais simples do que a formulação completa do MPC, esta se mostra útil para validar os módulos desenvolvidos neste trabalho antes de utilizá-los no desenvolvimento do controlador MPC completo. Por isso, mesmo não sendo o objetivo final do trabalho, o MPC sem restrições é uma etapa que ajuda na validação deste trabalho e por isso será explicado rapidamente nesta seção, com base no desenvolvimento apresentado no capítulo 4 de Alamir (2013).

A lei de controle que deseja-se encontrar pode ser expressa no formato a seguir, mostrado na Equação 2.26.

$$u(k) = -\mathbf{K_N} x(k). \tag{2.26}$$

Utilizando as equações de espaço de estados definidas na Equação 2.1, a lei de controle expressa na Equação 2.26 leva à Equação 2.27:

$$x(k+1) = (\mathbf{A} - \mathbf{B}\mathbf{K_N}x(k)). \tag{2.27}$$

Assim, deseja-se utilizar o modelo linear do sistema a ser controlado, descrito pelas equações de espaço de estados (Equação 2.1), para alcançar determinados espaços estacionários de acordo com o problema de controle escolhido, de forma que o regime desejado pode ser descrito pelo par $(x^d, u^d) \in \mathbb{R}^n.\mathbb{R}^{n+u}$, em que x^d representa o valor desejado para os estados do sistema e u^d representa o valor desejado para as variáveis de controle.

Da mesma forma, y_r^d corresponde a referência de rastreamento que o sistema controlado deve seguir, definido como $y_r^d = \mathbf{C_c}x^d$. Assumindo que a trajetória desejada está disponível ao longo de todo o horizonte de predição, pode-se descrever essa trajetória por meio do vetor $\tilde{y}_r^d(k)$ da forma mostrada abaixo, na Equação 2.28:

$$\tilde{y}_r^d(k) = \begin{pmatrix} y_r^d(k+1) \\ \vdots \\ y_r^d(k+N) \end{pmatrix}. \tag{2.28}$$

A melhor sequência de ações futuras de controle $\tilde{u}^{opt}(x(k)$ é o vetor $\tilde{u}$, que minimiza a seguinte função custo expressa pela Equação 2.29, derivada da Equação 2.8.

$$\frac{1}{2}\tilde{u}^T\mathbf{H}\tilde{u} + [\mathbf{F1}x(k) + \mathbf{F2}\tilde{y}_r^d + \mathbf{F3}u^d]^T\tilde{u} + \text{constante}. \tag{2.29}$$

Na ausência de restrições, a sequência ótima de controle $\tilde{u}^{opt}(x(k))$ pode ser obtida igualando o gradiente da função custo definida na Equação 2.29 a 0, resultando na Equação 2.30:

$$\mathbf{H}\tilde{u} + \mathbf{F1}x(k) + \mathbf{F2}\tilde{y}_r^d + \mathbf{F3}u^d = 0. \tag{2.30}$$

Portanto, a sequência ótima de controle pode ser definida de tal forma que resulta na Equação 2.31:

$$\tilde{u}^{opt} = \mathbf{H}^{-1}[\mathbf{F1}x(k) + \mathbf{F2}\tilde{y}_r^d + \mathbf{F3}u^d]. \tag{2.31}$$

Como o cálculo das variáveis de controle do MPC é baseado na utilização do primeiro componente do vetor ótimo de sequência de controle, a variável de controle do MPC sem restrições pode ser dada pela Equação 2.34:

$$u(k) = \mathbf{K_{MPC}}(x(k)), \tag{2.32}$$

$$u(k) = -\Pi_1^{(n_u,N)}\mathbf{H}^{-1}[\mathbf{F1}x(k) + \mathbf{F2}\tilde{y}_r^d + \mathbf{F3}u^d] = -\underbrace{\Pi_1^{(n_u,N)}\mathbf{H}^{-1}\mathbf{F1}}_{\mathbf{K_N}}x^d\ \underbrace{-\Pi_1^{(n_u,N)}\mathbf{H}^{-1}\mathbf{F2}}_{\mathbf{G_N}}\tilde{y}_r^d\ \underbrace{-\Pi_1^{(n_u,N)}\mathbf{H}^{-1}\mathbf{F3}}_{\mathbf{L_N}}u^d, \tag{2.33}$$

$$u(k) = -\mathbf{K_N}x(k) + \mathbf{G_N}\tilde{y}_r^d + \mathbf{L_N}u^d. \tag{2.34}$$

2.1.5 Formulação do MPC com restrições

Assim, como nas subseções anteriores, as demonstrações do algoritmo de solver de QP utilizado neste trabalho serão baseadas nas demonstrações contidas no capítulo 5 de Alamir (2013). Para informações mais detalhadas a cerca do formulação do algoritmo, o leitor é convidado a recorrer à bibliografia original.

Para entender o funcionamento do algoritmo que realiza a minimização do QP neste cenário, é conveniente analisar as etapas do algoritmo individualmente e gradualmente acrescentando as modificações necessárias para melhorar o desempenho do mesmo e lidar com as restrições inerentes ao MPC. Primeiramente, é introduzida a iteração gradiente, algoritmo base utilizado para resolver um problema de minimização do tipo quadrático, na mesma forma que encontra-se presente na formulação linear do MPC apresentada nas subseções anteriores. Depois, o algoritmo é melhorado por meio da introdução de uma etapa conhecida como expansão, que permite ao algoritmo obter uma convergência mais rápida que a formulação anterior (iteração gradiente). Logo em seguida, são introduzidas as restrições do problema, que são tratadas por meio de uma estratégia de penalização/saturação das mesmas. Por fim, obtêm-se o algoritmo final denominado PGE, envolvendo todas as etapas de cálculo apresentadas até então.

Cada uma dessas etapas é abordada nas subseções que seguem abaixo.

2.1.5.1 Iteração gradiente

Primeiramente, considera-se uma função custo (ainda sem as restrições do problema), que tem como base uma variável de decisão $p \in \mathbb{R}^{n_p}$, em que n_p representa o número total de variáveis de decisão, utilizada no processo de minimização inerente ao problema de otimização da mesma. Essa função custo pode ser escrita na seguinte forma, expressa na Equação 2.35:

$$J(p) \quad ; \quad p \in \mathbb{R}^{n_p}. \tag{2.35}$$

Supondo que essa função custo seja diferenciável e, mais precisamente, duplamente diferenciável, é possível realizar uma expansão de Taylor de segunda ordem em torno dos valores atuais das variáveis de decisão do problema de otimização ($p^{(0)}$), obtendo-se a Equação 2.36. Entretanto, para simplificar a notação utilizada, é possível realizar algumas substituições, que levam à Equação 2.37.

$$J(p) = J(p^{(0)}) + \underbrace{\frac{\partial J}{\partial p}(p^{(0)})}_{G(p)=\frac{\partial J}{\partial p}}(p - p^{(0)}) + \frac{1}{2}(p - p^{(0)})^T \underbrace{\frac{\partial^2 J}{\partial p^2}(p^{(0)})}_{H(p)=\frac{\partial^2 J}{\partial p^2}}(p - p^{(0)}) + ..., \tag{2.36}$$

$$J(p) = J(p^{(0)}) + G(p^{(0)})(p - p^{(0)}) + \frac{1}{2}(p - p^{(0)})^T H(p^{(0)})(p - p^{(0)}) + ... \tag{2.37}$$

O objetivo da função custo J é minimizar o problema de controle de maneira iterativa, ou seja, adotando a cada nova iteração uma melhor estimativa do valor das variáveis de decisão $p^{(i+1)}$ quando comparado com as variáveis de decisão utilizadas na iteração atual $p^{(i)}$. Portanto, torna-se necessário definir uma regra para que seja possível escolher melhores novas estimativas $p^{(i+1)}$.

Para isso, primeiramente utiliza-se a notação da Equação 2.37, o que permite escrever para qualquer variável de decisão $p^{(i+1)}$ a seguinte expansão de Taylor (Equação 2.38):

$$J(p^{(i+1)}) = J(p^{(i)}) + G(p^{(i)})(p^{(i+1)} - p^{(i)}) + ... \tag{2.38}$$

Assumindo que para um determinado valor de $p^{(i)}$ têm-se que $G(p^{(i)}) \neq 0$, pode-se adotar um passo suficientemente pequeno α na direção oposta à do gradiente, resultando na Equação 2.39. Essa estimativa de $p^{(i+1)}$ é considerada melhor que $p^{(i)}$, pois acarreta em um valor menor na função custo, ou seja, $J(p^{(i+1)}) < J(p^{(i)})$.

$$p^{(i+1)} = p^{(i)} - \alpha G(p^{(i)}). \tag{2.39}$$

É justamente essa a idéia por trás da iteração gradiente: adotando-se um pequeno passo α na direção oposta ao gradiente da função custo, é possível diminuir a função custo a cada iteração e, portanto, minimizar a mesma. Entretanto, percebe-se que esta estratégia é dependente do valor de α e a escolha desse fator é crucial para o sucesso da iteração gradiente. Um valor de α muito pequeno faz com que a função custo diminua de maneira muito devagar e, portanto, faz com que o problema de minimização demore muito para convergir para o valor desejado. Por outro lado, caso α seja muito alto, o valor da função custo pode não diminuir, mas sim aumentar. Como consequência, $p^{(i+1)}$ pode ser maior que $p^{(i)}$, o que faz com que o problema de minimização encontre uma solução além do ponto considerado como ótimo.

A Figura 2.3 ilustra o comportamento da função custo dado um passo α muito grande.

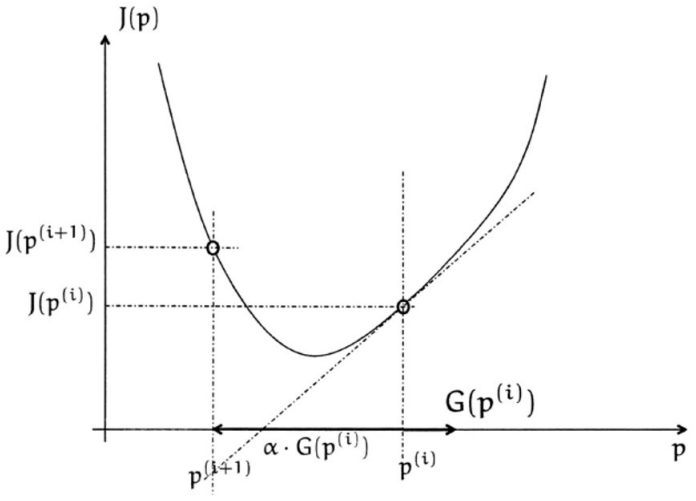

Figura 2.3: Aumento do valor da função custo que pode ser causado ao adotar-se um passo α muito grande. Adaptado de: (ALAMIR, 2013).

Com essa problemática em questão, torna-se necessário determinar um limite para o tamanho de passo α que pode ser adotado. Para isso, pode-se utilizar o teorema do valor central: para cada $\alpha > 0$ existe um valor $\alpha_0 \leq \alpha$ tal que resulte na Equação 2.40:

$$\frac{d\bar{J}}{d\alpha}(\alpha) = \frac{d\bar{J}}{d\alpha}(0) + \left[\frac{d^2\bar{J}}{d\alpha^2}(\alpha_0)\right]\alpha. \tag{2.40}$$

Esse resultado é importante porque, assumindo um limite máximo h_{max} conforme a Equação 2.41, é possível escrever o teorema do valor central de tal forma (como mostrado na Equação 2.42) que a derivada de $\bar{J}(\alpha)$ não mude de sinal enquanto o passo α não ultrapassar o limite estipulado na Equação 2.43.

$$\forall p \in \mathbb{R}^{n_p}; \left\| \frac{\partial J}{\partial p^2}(p) \right\| \leq h_{max}, \tag{2.41}$$

$$\frac{d\bar{J}}{d\alpha}(\alpha) = \left\| G(p^{(i)}) \right\|_2^2 .[-1 + h_{max}\alpha], \tag{2.42}$$

$$\alpha \leq \frac{1}{h_{max}}. \tag{2.43}$$

Para o caso particular de uma função custo quadrática (como no caso do MPC), a condição 2.41 é satisfeita sempre adotando a seguinte definição de h_{max}, expressa na Equação 2.44.

$$h_{max} = \|H\|_2 . \tag{2.44}$$

Portanto, a utilização de $\alpha = \frac{1}{\|H\|_2}$ garante sempre uma próxima iteração que gera um menor valor de função custo na iteração gradiente. Isso pode ser demonstrado utilizando o teorema do valor central para expressar o valor assumido pela função custo utilizando esse valor de α (mostrado na Equação 2.45), o que resulta na Equação 2.46.

$$\bar{J}(\alpha) \leq \bar{J}(0) - \frac{1}{h_{max}} \left\| G(p^{(i)}) \right\|_2^2 + \frac{1}{2h_{max}} \left\| G(p^{(i)}) \right\|_2^2, \tag{2.45}$$

$$J(p^{(i+1)}) \leq J(p^{(i)}) - \frac{1}{2h_{max}} \left\| G(p^{(i)}) \right\|_2^2. \tag{2.46}$$

Assim, utilizando-se da iteração gradiente, a função custo diminui a cada iteração desde que o gradiente permaneça diferente de 0. Com isso, o algoritmo converge assintoticamente a um ponto estacionário p que satisfaz $G(p) = 0$. Na maioria dos casos, isso corresponde ao mínimo local do problema de otimização sem restrições. Como trata-se de um problema de otimização quadrático e convexo, garantido com o uso de uma matriz $\mathbf{H}$ positivo-definida, tém-se que a solução global do problema de otimização é justamente dada por p.

2.1.5.2 Gradiente com Expansão (GE)

Apesar da formulação da iteração gradiente apresentada convergir para a solução global de um problema de otimização quadrático e convexo, o algoritmo possui limitações. Essas limitações surgem por conta da definição do passo α: quando o mesmo está próximo de autovalores que convergem mais rápido, tem-se uma resposta bastante rápida por parte do algoritmo; entretanto, para autovalores mais lentos, a convergência do algoritmo também é muito lenta.

Uma maneira de melhorar essa adversidade da iteração gradiente é checar de maneira sistemática se é possível utilizar passos ainda maiores de α, na forma da Equação 2.47

$$p_{\gamma}^{(i+1)} = p^{(i)} - \frac{\gamma}{h_{max}} G(p^{(i)}); \quad \gamma > 1. \tag{2.47}$$

Assim, é possível comparar os valores dados pelas duas funções custo: $J(p_1^{(i+1)})$, que é dado na Equação 2.46, e $J(p_{\gamma}^{(i+1)})$, que é dado calculando-se o custo das variáveis de decisão expressas na Equação 2.47, verificando qual deles resulta em um menor valor. Se $J(p_{\gamma}^{(i+1)}) < J(p_1^{(i+1)})$, a expansão pode assumir um valor maior, dado pela Equação (Eq. 2.48), e os valores atualizados de p são dados por $p_{\gamma}^{(i+1)}$:

$$\gamma = \beta^{+}\gamma; \quad \beta^{+} > 1. \tag{2.48}$$

Entretanto, se $J(p_{\gamma}^{(i+1)}) > J(p_1^{(i+1)})$, a expansão é muito grande e o valor do parâmetro γ deve ser reduzido para o valor mostrado na Equação 2.49, e o valor atualizado de p é dado por $p_1^{(i+1)}$.

$$y = max(\gamma_{min}, \beta^{-}.\gamma); \quad 0 < \beta^{-} < 1; \quad \gamma_{min} > 1. \tag{2.49}$$

Esse algoritmo é conhecida como Gradiente com Expansão (do inglês, *Gradient Expansion* ou simplesmente GE) e pode ser sumarizado conforme mostrado no pseudocódigo refente ao Algoritmo 1.

Algorithm 1 Algoritmo Expansão do Gradiente (GE) [p^{+},γ^{+}] = GE(p,γ)

Funções necessárias: J e seu gradiente G
Parâmetros necessários : $h_{max} > 0$, $\gamma_{min} > 1$, $\beta^{+} > 1$, $\beta^{-} \in]0, 1[$

$p_{cand}^{1} \leftarrow p - 1/h_{max}G(p)$, $J^{(1)} \leftarrow J(p_{cand}^{1})$
$p_{cand}^{\gamma} \leftarrow p - \gamma/h_{max}G(p)$, $J^{(\gamma)} \leftarrow J(p_{cand}^{\gamma})$

if $J^{(\gamma)} < J^{(1)}$ **then**
    $p^{+} \leftarrow p^{(\gamma)}$
    $\gamma^{+} \leftarrow \beta^{+}.\gamma$
else
    $p^{+} \leftarrow p^{(1)}$
    $\gamma^{+} \leftarrow \max\{\gamma_{min}, \beta^{-}.\gamma\}$
end

2.1.5.3 Projeção do Gradiente com Expansão (PGE)

Para iniciar a implementação das restrições inerentes ao MPC no algoritmo de *solver* apresentado até aqui (algoritmo PGE), é importante analisar o problema de otimização por meio da seguinte formulação, representada na Equação 2.50:

$$\min_{p\in\mathbb{R}^{n_p}} J_0(p) \quad \text{sob} \quad g(p) \leq 0 \in \mathbb{R}^{n_g} \quad \text{e} \quad p^{min} \leq p \leq p^{max}. \tag{2.50}$$

Em outras palavras, um problema de otimização restringido por inequalidades que deve ser ser aplicada a todos os componentes do vetor p que contém as variáveis de decisão. Uma maneira muito utilizada para forçar com que as restrições sejam respeitadas é ampliar a função custo para incluir os termos adicionais apresentados na Equação 2.51.

$$J(p|\rho) = J_0(p) + \sum_{i=1}^{n_g} \rho_i [max(0, g_i(p))]^2. \tag{2.51}$$

Na Equação 2.51, J_0 corresponde à função custo original apresentada em 2.50 e o vetor $\rho \in \mathbb{R}_+^{n_g}$, cujo número total de variáveis de decisão é representado por n_g, é dado pela Equação 2.52:

$$\rho = \begin{pmatrix} \rho_1 \\ \vdots \\ \rho_{n_g} \end{pmatrix}. \tag{2.52}$$

Essa nova formulação permite expressar que toda vez que uma violação às restrições do problema de controle ocorrer, a mesma introduz um custo adicional e, é justamente esse custo adicional, que exclui essa possibilidade durante o processo de otimização da nova função custo sob as condições de saturação e somente sob elas, resultando na Equação 2.53.

$$\min_{p\in\mathbb{R}^{n_p}} J(p|\rho) \quad \text{sob} \quad p^{min} \leq p \leq p^{max}. \tag{2.53}$$

O novo problema de otimização parametrizado com base em ρ pode ser resolvido utilizando-se um método denominado Projeção do Gradiente, constituído de duas etapas:

1. Dado o vetor correspondente à iteração atual $p^{(i)} \in [p^{min}, p^{max}]$ e o valor atual do vetor de penalização ρ, o valor atualizado de p é calculado primeiramente sem levar em consideração as restrições por meio de uma iteração gradiente com expansão, resultando na Equação 2.54:

$$[p_{cand}^{(i+1)}, \gamma^{(i+1)}] = G_E(p^{(i)}, \gamma^{(i)}). \tag{2.54}$$

2. Atualiza-se o valor de $p^{(i+1)}$ por meio da etapa de projeção, cujo operador é definido na Equação 2.55 e resulta na Equação 2.56.

$$P_r = \mathbb{R}^{n_p}[p^{min}, p^{max}]; \quad p^r = P_r(p), \quad \forall i p_i^r = min(p_i^{max}, max(p_i^{min}, p_i)), \tag{2.55}$$

$$p^{(i+1)} = P_r(p_{cand}^{(i+1)}). \tag{2.56}$$

A esse algoritmo dá-se o nome de Projeção do Gradiente com Expansão (do inglês *Projected Gradient Expansion*) ou PGE. Se a função custo $J(p|\rho)$ for convexa, o que é o caso da abordagem linear do MPC, o algoritmo PGE converge assintoticamente para o mínimo global da função custo aumentada $J(p|\rho)$ e, consequentemente, o torna um bom candidato para utilização como *solver* de QP do MPC, ou seja, o algoritmo de minimização de QP.

O pseudocódigo do Algoritmo PGE (algoritmo 2) encontra-se abaixo.

Algorithm 2 Algoritmo Expansão do Gradiente Projetado (PGE) [p^+,γ^+] = PGE(p,γ,ρ)

Funções necessárias: J e seu gradiente G
Parâmetros necessários : $h_{max} > 0$, $\gamma_{min} > 1$, $\beta^+ > 1$, $\beta^- \in]0, 1[$, p^{min}, p^{max}

$p_{cand}^1 \leftarrow p - 1/h_{max} G(p)$, $J^{(1)} \leftarrow J(p_{cand}^1|\rho)$
$p_{cand}^\gamma \leftarrow p - \gamma/h_{max} G(p)$, $J^{(\gamma)} \leftarrow J(p_{cand}^\gamma|\rho)$

if $J^{(\gamma)} < J^{(1)}$ **then**
    $p_{cand}^+ \leftarrow p^{(\gamma)}$
    $\gamma^+ \leftarrow \beta^+.\gamma$
else
    $p_{cand}^+ \leftarrow p^{(1)}$
    $\gamma^+ \leftarrow \max\{\gamma_{min}, \beta^-.\gamma\}$
end

for $\sigma = 1 \ldots n_p$ **do**
    $p_\sigma^{(i+1)} \leftarrow \min(p_\sigma^{max}, \max(p_\sigma^{min}, p_{cand_\sigma}^+))$
end

Entretanto, é interessante analisar o que determina se a solução encontrada está de acordo com as restrições do problema de controle, e isso é dado pelo parâmetro ρ. Se $\rho = 0$, o algoritmo PGE converge para a solução na qual nenhuma restrição é aplicada, ou seja, $G(p) = 0 \in \mathbb{R}^n$. Já para valores bem altos de ρ, a quantidade de restrições que podem ser violadas pela solução ótima p^{opt} do algoritmo PGE converge para zero, ou seja, ocorre o que é mostrado na Equação 2.57:

$$\lim_{\rho \to \inf} \max_{i=1}^{n_g} [\max(0, g_i(p^{opt}))] = 0. \tag{2.57}$$

Portanto, é interessante utilizar valores altos de ρ, uma vez que a Equação 2.57 mostra que para tais valores, as restrições do problema de controle são atendidas na solução ótima encontrada pelo algoritmo. Todavia, há uma limitação em utilizar valores muito altos para ρ: por mais que seja uma garantia do respeito às limitações do problema, altos valores de ρ também acabam gerando valores altos de h_{max}, já

que o termo $\rho\frac{\partial^2 g_i}{\partial p^2}$ está envolvido no cálculo da matriz Hessiana que possui como restrição justamente o fator h_{max}.

Altos valores de h_{max}, por sua vez, resultam em passos α muito pequenos. Isso poderia ser resolvido adotando-se valores mais altos para β^+, por exemplo, mas esse tipo de abordagem pode levar a problemas de convergência do algoritmo.

Para escolher então o valor apropriado de $\rho > 0$, pode-se utilizar os seguintes passos:

1. Inicialização de ρ com valores pequenos;
2. Incrementar ρ por um fator β_ρ^+ e submetido a uma restrição de valor máximo (ρ_{max}), como pode ser visto na Equação 2.58 a cada N_{iter} iterações.

$$\rho = \min\{\ \rho_{max}, \beta_\rho^+.\rho\} \qquad \beta_\rho^+ > 1. \tag{2.58}$$

Essa formulação final resulta no Algoritmo 3, cujo pseudocódigo é apresentado logo abaixo.

Algorithm 3 Otimização com aplicação das restrições p^{sol} = solver_ sat_ pen($p^{(0)}$,N_{iter})

Funções necessárias: J e seu gradiente G
Parâmetros necessários : $h_{max} > 0$, $h_{max}^g > 0$, $\gamma_{min} > 1$, $\beta^+ > 1$, $\beta^- \in]0,1[$, $p^{min}, p^{max} \in \mathbb{R}^{n_p}$, $\rho_{max}, \beta_\rho^+ > 1$

$i_\rho = 0$, $\gamma = 1.2$, $\rho = 5h_{max}/h_{max}^g$, $n_\rho = 5n_p$

for $(i = 1 : \text{Niter})$ **do**
    $[p, \gamma]$ = PGE(p,γ,ρ)
    $i_\rho \leftarrow i_\rho + 1$

    if $i_\rho = n_\rho$ **then**
        $i_\rho \leftarrow 1$
        $\rho \leftarrow \min(\rho_{max}, \beta_\rho^+.\rho)$
    end

end

$p^{sol} \leftarrow p$

2.1.5.4 Aplicação do algoritmo PGE como solver de QP do MPC

Com a definição do algoritmo PGE, têm-se todos os dados necessários para implementar o *solver* de QP a ser utilizado no MPC linear.

O QP a ser resolvido é definido da seguinte forma (Equação 2.59):

$$\min_{p \in [p^{min}, p^{max}]} \frac{1}{2} p^T \mathbf{H} p + \mathbf{F}^T p \quad \text{mediante as restrições} \quad \mathbf{A}p \leq \mathbf{B}. \tag{2.59}$$

As matrizes $\mathbf{H}$ e $\mathbf{F}$ do QP presentes na Equação 2.59 são as mesmas matrizes de custo $\mathbf{H}$ e $\mathbf{F}$ definidas na subseção 2.1.3. Já as matrizes $\mathbf{A}$ e $\mathbf{B}$ do QP são as matrizes de restrições $\mathbf{A_{ineq}}$ e $\mathbf{B_{ineq}}$, também definidas na subseção 2.1.3.

Com as matrizes $\mathbf{H}$ e $\mathbf{A}$ do QP, pode-se calcular o valor do parâmetro h_{max} (equações 2.60 - 2.61):

$$h^0_{\text{max}} = \|\mathbf{H}\|_2; \quad h^g_{\text{max}} = \|\mathbf{A}^T\mathbf{A}\|_2, \tag{2.60}$$

$$h_{max} = h^0_{\text{max}} + 2\rho.h^g_{\text{max}}. \tag{2.61}$$

Além desses dados, é necessário atribuir os parâmetros p^{min}, p^{max}, ρ_0, ρ^{max}, β^+_ρ,n_ρ e N_{iter}, que assumem os valores determinados na formulação do algoritmo de *solver* nesta subseção.

No capítulo 5 de Alamir (2013), há uma formulação deste *solver* de QP em MATLAB, denominada solver_ sat_ pen.m. Esse código é integrado a um *script* utilizado para calcular as matrizes do MPC e os dados que o QP necessita, listados anteriormente.

Por fim, é interessante notar que toda a formulação necessária para a implementação do MPC utilizando como *solver* de QP o algoritmo PGE foi apresentada nesta seção. Em outras palavras, isso quer dizer que, com essa formulação, é possível implementar o MPC sem a utilização de bibliotecas numéricas ou de otimização adicionais, que é a principal motivação para implementação deste algoritmo neste trabalho. É justamente essa estratégia de controle que foi implementada em linguagem de descrição de *hardware*, para que pudesse ser implementada em FPGA.

2.1.6 Parametrização do MPC

A parametrização de controle é uma técnica muito importante para a implementação do MPC embarcado em plataformas de *hardware*. Trata-se de uma técnica que diminui drasticamente o número de variáveis de decisão do problema de otimização inerente ao MPC, sem acarretar em uma redução significativa de desempenho do controlador, na maioria dos casos (ALAMIR, 2013).

A parametrização adotada neste trabalho é a parametrização exponencial. Entretanto, antes de explicar esse método específico de parametrização, é importante explicar alguns conceitos gerais às diferentes formas de parametrização de controle.

Na formulação do MPC aplicado a sistemas LTI apresentada na subseção anterior, é possível perceber que o número de graus de liberdade do algoritmo, ou seja, o número de variáveis que precisam ser determinadas para se obter uma solução do problema de controle, corresponde ao horizonte de predição *N*. Isso fica ainda mais claro quando observa-se que o MPC calcula uma sequência de controle composta por *N* elementos, que pode ser evidenciado analisando a Equação 2.2.

Pode-se então considerar uma sequência de controle em que existam somente dois graus de liberdade, como mostrado na Equação 2.62.

$$u(k+i-1) = \begin{cases} p_1 \text{ se } i = 1 \\ p_2 \text{ se } i \in 2, \ldots, 10 \end{cases} . \tag{2.62}$$

A Equação 2.62 pode ser reescrita da forma apresentada na Equação 2.63 abaixo:

$$\widetilde{u}(k) := \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{pmatrix} p =: \mathbf{\Pi_r}.p; \quad \mathbf{\Pi_r} \in \mathbb{R}^{2\text{x}10}. \tag{2.63}$$

Na Equação 2.63, a matriz $\mathbf{\Pi_r}$ é a matriz de parametrização em torno das variáveis de decisão alocadas no vetor p. Utilizando essa mesma Equação, é possível reescrever as matrizes $\mathbf{H}$, $\mathbf{F}$, $\mathbf{A_{ineq}}$ e $\mathbf{B_{ineq}}$ em função das novas variáveis de decisão p. Assim, a função custo do problema de controle pode ser reescrita, resultando na Equação 2.64:

$$\mathbf{J}(p) = \frac{1}{2}\widetilde{u}^T\mathbf{H}\widetilde{u} + \mathbf{F}^T\widetilde{u} = \frac{1}{2}p^T[\mathbf{\Pi_r}\mathbf{H}\mathbf{\Pi_r}]p + [\mathbf{\Pi_r}^T\mathbf{F}]^T p = \frac{1}{2}p^T\mathbf{H_r}p + \mathbf{F_r}^T p. \tag{2.64}$$

Analisando a Equação 2.64, é possível perceber que as novas matrizes reduzidas $\mathbf{H_r}$ e $\mathbf{F_r}$ são dadas pela Equação 2.65

$$\begin{cases} \mathbf{H_r} = \mathbf{\Pi_r}.\mathbf{H}.\mathbf{\Pi_r} \\ \mathbf{F_r} = \mathbf{\Pi_r}^T\mathbf{F} \end{cases} . \tag{2.65}$$

Utilizando um raciocínio análogo, as restrições aplicadas às variáveis de controle podem ser reescritas da forma apresentada na Equação 2.66:

$$\widetilde{u}^{min} \leq \widetilde{u} \leq \widetilde{u}^{max} = \widetilde{u}^{min} \leq \mathbf{\Pi_r}p \leq \widetilde{u}^{max}. \tag{2.66}$$

Assim, as inequalidades que continham as restrições na primeira linha da Equação 2.24 podem ser reescritas na maneira expressa na Equação 2.67:

$$\mathbf{A_{ineq}}\underbrace{\widetilde{u}}_{\mathbf{\Pi_r}p} \leq \mathbf{B_{ineq}} = \underbrace{\mathbf{A_{ineq}}\mathbf{\Pi_r}}_{\mathbf{A_r}} p \leq \mathbf{B_{ineq}}. \tag{2.67}$$

Injetando as novas restrições impostas na Equação 2.66 na Equação 2.67, é possível obter as novas

matrizes reduzidas de restrições $\mathbf{A_r}$ e $\mathbf{B_r}$, representadas pela Equação 2.68.

$$\begin{cases} \mathbf{A_r} = \begin{pmatrix} \mathbf{A_{ineq}}.\mathbf{\Pi_r} \\ -\mathbf{\Pi_r} \\ +\mathbf{\Pi_r} \end{pmatrix} \\ \\ \mathbf{B_r} = \begin{pmatrix} \mathbf{B_{ineq}} \\ -\widetilde{u}^{min} \\ \widetilde{u}^{max} \end{pmatrix} \end{cases} . \tag{2.68}$$

É interessante notar que na formulação adotada, a dimensão das variáveis de decisão não está vinculada com o horizonte de predição adotado para o problema. Essa é uma grande vantagem, pois em muitos sistemas o uso de um horizonte de predição relativamente grande pode ser essencial para manter a estabilidade do mesmo. Além disso, ainda que a estabilidade do sistema seja garantida, um maior horizonte de predição pode ser necessário para melhorar o desempenho do controlador. Portanto, é possível adotar um horizonte de predição maior sem impactar nos graus de liberdade necessários para obter a sequência de controle $\widetilde{u}$, o que contribui para diminuir a complexidade computacional do MPC.

Existem diversas maneiras de definir uma sequência de controle com um número menor de parâmetros, sendo um dos possíveis métodos para alcançar tal objetivo a utilização da parametrização exponencial. No caso desse tipo de parametrização, considera-se um sistema com n_u atuadores (a mesma dimensão do vetor de dimensões de controle u), cujos tempos de estabilização são dados pela Equação 2.69.

$$\tau_r \in \mathbb{R}^{n_u}. \tag{2.69}$$

O tempo de estabilização indica que o i-ésimo atuador do sistema alcança seu valor de referência dado pelo controlador em um intervalo de tempo igual ao i-ésimo componente de τ_r. Por conta disso, assume-se que o algoritmo de controle não deve possuir modos dinâmicos mais rápidos que $\frac{3}{\tau_r}$, ou seja:

$$\lambda = \frac{3}{\tau_r}. \tag{2.70}$$

Dado o período de amostragem do sistema τ, a parametrização exponencial do perfil de controle pode ser expressa de acordo com a Equação 2.1.6.

$$u_j(k+i) = \sum_{l=1}^{n_e^{(j)}} \left[e^{-\lambda_j (i\tau)/((l-1)\alpha+1)} \right] .p_l^{(j)}; \quad \alpha > 1. \tag{2.71}$$

Analisando a Equação , é possível notar que a mesma é uma combinação linear de exponenciais da forma $e^{-\lambda_j^{(l)t}}$, em que $\lambda_j^{(l)} = \frac{\lambda_j}{(l-1)\alpha+1}$. Assim, têm-se que $\lambda_j^{(1)} = \lambda_j$, $\lambda_j^2 = \frac{\lambda_j}{\alpha}$, $\lambda_j^{(3)} = \frac{\lambda_j}{2\alpha}$ e assim por diante.

Adotando-se o parâmetro $\alpha > 1$, os n_u perfis de controle podem ser definidos pelo vetor de parâmetros p dado pela Equação 2.72.

$$p = \begin{pmatrix} p^{(1)} \in \mathbb{R}^{n_e^{(1)}} \\ \vdots \\ p^{(n_u)} \in \mathbb{R}^{n_e^{(n_u)}} \end{pmatrix} \in \mathbb{R}^{n_p}. \tag{2.72}$$

O resultado disso consiste em uma variável de decisão p de dimensão $n_p = \sum_{j=1}^{n_u} n_e^{(j)}$, em que n_u corresponde ao número de atuadores do sistema e n_e é o número de exponenciais escolhidas para serem usadas em cada um dos atuadores. É importante ressaltar que este último corresponde a um importante grau de liberdade, uma vez que alguns dos atuadores do sistema podem se comportar de forma bastante dinâmica ao longo do horizonte de predição, ao passo que outros atuadores podem ser considerados constantes. Por isso, pode ser necessário utilizar diferentes números de exponenciais no perfil de controle parametrizado do sistema, buscando atender as especificidades do modelo a ser controlado.

Para implementar de fato a parametrização exponencial, ainda é necessário implementar a matriz de parametrização específica para esse tipo de técnica. Essa matriz é análoga à matriz $\mathbf{\Pi_r}$ dada na Equação 2.63. Aqui, essa matriz será representada por $\mathbf{\Pi_e}$, de forma que a sequência de controle ótima será dada pela Equação 2.73):

$$\widetilde{u}(k) = \begin{pmatrix} u(k) \\ \vdots \\ u(k+N-1) \end{pmatrix} = \mathbf{\Pi_e}.p(k). \tag{2.73}$$

Para isso, a Equação 2.1.6 será reescrita de forma mais compacta, conforme mostrado na Equação 2.74.

$$u_j(k+i) = \sum_{l=1}^{n_e^{(j)}} \underbrace{\left[e^{-\lambda_j(i\tau)/((l-1)\alpha+1)}\right]}_{m_{j,l}(i)}.p_l^{(j)}; \quad \alpha > 1 = \sum_{l=1}^{n_e^{(j)}} [m_{j,l}(i)].p_l^{(j)}. \tag{2.74}$$

A Equação 2.74 pode ser escrita na forma de produto matricial (Eq. 2.75), dada a expressão mais compacta mostrada na Equação 2.76

$$u_j(k+i) = [\mathbf{M_j}(i)].p^{(j)}; \quad p^{(j)} \in \mathbb{R}^{n_e^{(j)}}, \tag{2.75}$$

$$\mathbf{M_j}(i) = (m_{j,l}(i) \dots m_{j,n_e^{(j)}}(i)). \tag{2.76}$$

Reescrevendo a Equação 2.75 para $j = 1, \dots\ n_u$ leva ao vetor de entrada $u(k+i)$ no instante futuro $k+i$, cuja formulação é mostrada na Equação 2.77:

$$u(k+i) = \underbrace{BlockDiag\left(\mathbf{M_j}(i)_{j=1}^{n_u}\right)}_{\mathbf{M}(i)}. \begin{pmatrix} p^{(1)} \\ \vdots \\ p^{(n_u)} \end{pmatrix} = [\mathbf{M}(i)].p \tag{2.77}$$

Na Equação 2.77, apresentada acima, $BlockDiag$ indica uma matriz diagonal por bloco formada com os elementos dentro dos parênteses.

Por fim, concatenando a Equação 2.77 para $i = 0, \ldots, N-1$, obtêm-se a formulação da matriz de parametrização exponencial $\mathbf{\Pi_e}$, mostrada na Equação 2.78 abaixo.

$$\mathbf{\Pi_e} = \begin{pmatrix} \mathbf{M}(0) \\ \vdots \\ \mathbf{M}(N-1) \end{pmatrix}. \tag{2.78}$$

Apesar de introduzir mais cálculos matriciais na formulação do MPC, a parametrização exponencial reduz a complexidade computacional para o *solver* de QP do MPC. Isso porque as matrizes $\mathbf{\Pi_e}$, $\mathbf{H_r}$ e $\mathbf{A_r}$ podem ser calculadas *offline*, assim como as matrizes $\mathbf{H}, \mathbf{A_{ineq}}, \mathbf{F_1}, \mathbf{F_2}, \mathbf{F_3}, \mathbf{G_1}, \mathbf{G_2}$ e $\mathbf{G_3}$. As matrizes $\mathbf{F_r}$ e $\mathbf{B_r}$ são calculadas *online*, a cada iteração do MPC, pois dependem das matrizes $\mathbf{F}$ e $\mathbf{B_{ineq}}$, que também são calculadas *online*.

A parametrização de controle é muito importante para a implementação do MPC em tempo real. Utilizando essa técnica, é possível tornar uma solução antes não realizável em realizável do ponto de vista do tempo real e da compatibilidade com o *hardware* no qual pretende-se embarcar o MPC. E o principal, isso pode ser alcançado, na maioria das vezes, sem quedas drásticas na qualidade do controle do sistema em malha fechada.

2.2 Quadrirrotor

2.2.1 Revisão bibliográfica

Conforme apresentado anteriormente, o estudo de caso deste trabalho é um quadrirrotor. Esse tipo de sistema é uma das diversas categorias de VANT, veículos que realizam operações sem nenhum piloto a bordo, ou seja, são controlados remotamente ou autonomamente. A primeira aplicação de VANTs foi na área militar, e desde então, os mesmos se tornaram presença permanente nesse segmento (NISSER; WESTIN, 2006). Ainda segundo Nisser e Westin (2006) e também segundo Irizarry, Gheisari e Walker (2012), outras áreas de aplicação de VANTs incluem:

- Busca, investigação e resgate de pessoas durante / após desastres naturais como furacões, terremotos e incêndios;
- Controle e vigilância de fronteiras e territórios;
- Localização de incêndios em florestas ou nevascas em fazendas;
- Monitoramento de atividades criminais;
- Mineiração;
- Pesquisas científicas;

- Utilização em plataformas petroquímicas.

Aplicações mais recentes de VANTs incluem atividades comerciais e de *marketing* e empresas que prestam serviços de fotografia aérea e filmagens de eventos (IRIZARRY; GHEISARI; WALKER, 2012).

O crescente interesse por VANTs possibilitou o desenvolvimento de diversos modelos de aeronave, de diferentes formatos e tamanhos para operações em diferentes tipos de missão (VALAVANIS, 2008). Os principais tipos de VANT são apresentados na Figura 2.4.

Figura 2.4: Diferentes tipos de VANT disponíveis na atualidade. Adaptado de: (KANELLAKIS; NIKOLAKOPOULOS, 2017).

Dos tipos apresentados na Figura 2.4, merecem destaque devido à quantidade de publicações e aplicações disponíveis na literatura os chamados VANTs de asa fixa e multirrotores. O primeiro corresponde a um tipo de VANT ideal para vôos longos e aplicações que exijam a presença de cargas mais pesadas durante o vôo; já a segunda categoria de VANTs possui maior manobrabilidade por conta de suas lâminas rotatórias que permitem o vôo do dispositivo, bem como decolagem e aterrissagem vertical (do inglês, *Vertical Take-Off and Landing* ou VTOL) e a capacidade de pairar e voar a pequenas altitudes. Por conta dessas características, multirotores são utilizadas em uma faixa maior de aplicações do que os VANTs de asa fixa (KENDOUL, 2012).

A Tabela 2.1 sumariza as vantagens e desvantagens discutidas no parágrafo anterior, bem como para os demais tipos de VANT apresentados na Figura 2.4.

Tabela 2.1: Vantagens e desvantagens dos diferentes tipos de VANT. Adaptado de: (KANELLAKIS; NIKOLAKOPOULOS, 2017).

	Vantagens	Desvantagens
Unirrotor	• Vôo VTOL • Capacidade de pairar • Alta capacidade de carga • Durabilidade	• Cobertura em área • Mais periogoso
Multirrotor	• Vôo VTOL • Capacidade de pairar • Manobrabilidade • Ambientes internos e externos • Área pequena e vazada • *Design* simples	• Cobertura em área • Baixa capacidade de carga • Curto período de vôo
Asa fixa	• Durabilidade • Grande cobertura • Alta velocidade de vôo • Alta capacidade de carga	• Cobertura em área • Decolagem e aterrissagem em locais específicos • Não é possível pairar • Velocidade constante para voar
Asa fixa híbrido VTOL	• Durabilidade • Grande cobertura • Vôo VTOL	• Ainda sendo aprimorado • Transição entre pairar e voar

Apesar disso, o controle de multirrotores é desafiador. De fato, o desenvolvimento de VANTs para operações interiores e exteriores é considerado, há algum tempo, um dos grandes desafios da robótica (BECKER; BOUABDALLAH; SIEGWART, 2006). Primeiramente, esses sistemas são não lineares e subatuados, ou seja, possuem menos entradas de controle do que estados do sistema. Além disso, as dinâmicas são significativamente acopladas e há grande incerteza nos modelos desse tipo de sistema que surgem por conta da natureza aerodinâmica complicada que ocorre durante a geração de empuxo no vôo dos mesmos (ALVARENGA et al., 2015).

Essas questões motivam o estudo de sistemas e técnicas de controle para esse tipo de veículo, voltados para as mais diversas aplicações. A Figura 2.5 mostra o resultado de uma pesquisa realizada por (ALVARENGA et al., 2015), listando as principais estratégias de controle utilizadas em VANTs.

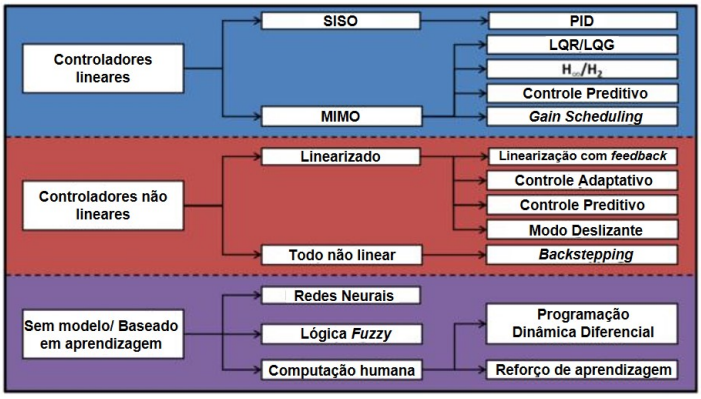

Figura 2.5: Técnicas de controle para VANT e suas respectivas classificações. Fonte: (ALVARENGA et al., 2015).

É possível perceber pela Figura 2.5, que os sistemas de controle desenvolvidos para VANTs são divididos em três categorias: linear, não linear e independentes de modelo, de acordo com o modelo utilizado para a implementação do VANT. Modelos não lineares são mais complexos e difíceis de implementar por conta da presença de equações diferenciais ou polinomiais de ordem elevada, mas é justamente esse tipo de complexidade que permite a descrições de dinâmicas com múltiplos pontos de equilíbrio e comportamentos que não podem ser previstos com modelos lineares. Sistemas de controle lineares geralmente realizam suposições sob algumas condições de operação específicas, que permitem a linearização do modelo. Isso simplifica o mesmo e facilita sua implementação, mas limita seus cenários de aplicação. Por último, sistemas independentes de modelo baseam-se em algoritmos de aprendizagem de máquina para, por meio de testes de vôo controlados por um piloto utilizando controle remoto, ensinar o algoritmo a reproduzir o comportamento do piloto e suas tomadas de decisão (ALVARENGA et al., 2015).

Focando na categoria de multirrotores, e mais especificamente na categoria de quadrirrotores, é possível perceber que esse tipo de sistema possui diversos trabalhos na literatura, com várias proposições de estratégia de controle diferentes. No trabalho de Castillo, Dzul e Lozano (2004), controladores mais tradicionais como o PID e estratégias de controle mais modernas como o LQR são propostas para estabilização de quadrirrotores, ou seja, para todo sinal de entrada com amplitude limitada que é introduzido no sistema, as saídas também assumem valores limitados, de forma que o sistema não diverge. Segundo Abdolhosseini, Zhang e Rabbath (2013), esses controladores funcionam bem com modelos linearizados do quadrirrotor, mas provaram não ser tão eficazes com modelos não lineares do mesmo.

Há ainda proposições de métodos como linearização de alimentação (MISTLER; BENALLEGUE, 2001), utilizado para que o quadrirrotor fosse capaz de seguir uma trajetória de referência (rastreamento de trajetória); controle de modo deslizante (XU; OZGUNER, 2006) aplicado a uma classe de sistemas subatuados, utilização da técnica conhecida como *backstepping* com um modelo simplificado do quadrirrotor (BOUABDALLAH; SIEGWART, 2005) e uma lei de controle baseada na suposição de que o quadrirrotor pode ser considerado como a interconexão de três subsistemas distintos (MADANI; BENALLEGUE, 2006a).

Apesar de todas essas diferentes estratégias de controle terem sido testadas com quadrirrotores e conseguirem resultados satisfatórios, nenhuma delas é capaz de lidar explicitamente com restrições de operação do quadrirrotor, que são fundamentais para manter a estabilidade e controlar o mesmo (ABDOLHOSSEINI; ZHANG; RABBATH, 2013). Essa é uma das principais razões que motivam a utilização do MPC aplicado a esse tipo de sistema. Primeiramente, estratégias híbridas foram propostas na literatura: MPC para rastrear trajetórias juntamente com um controlador $H_{\inf}$ para estabilização do sistema (RAFFO; ORTEGA; RUBIO, 2008) e MPC para estabelecer leis de controle para posição e um controlador *feedforward* para estabilização da aeronave (ALEXIS; NIKOLAKOPOULOS, 2010).

Nos últimos anos, entretanto, há uma crescente de trabalhos que buscam utilizar o MPC como único controlador, tanto para estabilizar o sistema quanto para alcançar os demais objetivos de controle estabelecidos. O trabalho de Lopes *et al.* (2011) propõe um controlador MPC para estabilização e rastreamento de trajetórias para um quadrirrotor, sendo a principal contribuição do trabalho justamente a utilização do MPC como único controlador do sistema. O desempenho do MPC é comparado à abordagem do PID e do controlador *backstepping* já citados em Castillo, Dzul e Lozano (2004) e Bouabdallah e Siegwart (2005),

respectivamente, obtendo melhor desempenho que o controlador PID, mas um tempo de convergência menor que o *backstepping*. Apesar disso, o MPC não necessita de atuadores com respostas tão rápidas quanto os do controlador *backstepping* e, por incorporar as restrições do sistema em sua formulação, torna-se uma opção mais viável e segura para esse tipo de sistema.

Trabalhos mais recentes como o propõem a aplicação do MPC a um quadrirrotor Qball-X4 da Quanser, buscando diminuir a demanda computacional do MPC por meio de um método de predição baseado em uma estrutura linear interna do modelo do sistema e tratando as dinâmicas de vôo translacionais como sendo independentes umas das outras (ABDOLHOSSEINI; ZHANG; RABBATH, 2013). A estratégia é validada tanto em simulação quanto por meio de testes experimentais de rastreamento de trajetória utilizando o Qball-X4.

Outra possibilidade explorada na literatura corresponde à proposição de uma estratégia de controle preditivo não linear baseada em *flatness*, através da utilização de um modelo linear equivalente (graças à propriedade de *flatness* do modelo não linear). Essa é a maior contribuição do trabalho e permite que o rastreamento de trajetória seja efetuado por parte do quadrirrotor, respeitando as restrições impostas ao sistema (Limaverde Filho, J. O. D. A. Lourenço et al., 2016).

O trabalho de Chikasha (2017) propõe a utilização de uma técnica conhecido como Controle Preditivo baseado em Modelo Adaptativo (em inglês, *Adaptive Model Predictive Control* ou AMPC). Os resultados mostram que o sistema consegue alcançar seus dois principais objetivos de controle: adaptação a mudanças na dinâmica do sistema e rastreamento de trajetória, sendo a principal dificuldade do método a sintonização do controlador.

Por fim, o trabalho de Murilo e Lopes (2018) apresenta uma metodologia de parametrização aplicada a quadrirrotores utilizando uma abordagem não linear (NMPC), de forma a reduzir significativamente o número de variáveis de decisão relacionadas ao problema de otimização e tornar mais fácil o cálculo da sequência ótima de controle do problema proposto. A metodologia permite trabalhar com diferentes tipos de modelagem de quadrirrotores, independente da complexidade do modelo não linear adotado.

É possível perceber que há muitas contribuições na literatura referentes à aplicação do MPC em quadrirrotores. É um campo de grande destaque, mas que ainda possui muito espaço para contribuições. Por isso, este trabalho propõe a utilização do MPC em quadrirrotores, buscando obter todas as vantagens da utilização dessa técnica (em especial o respeito às restrições do sistema, auxiliando na estabilidade do mesmo), mas explorando um campo que ainda foi pouco abordado na literatura: o desenvolvimento de estratégias de controle preditivo utilizando FPGA para aplicações em quadrirrotores. Mas antes disso, será apresentada a modelagem do quadrirrotor e os parâmetros do sistema que serão utilizados nas etapas de validação experimental deste trabalho.

2.2.2 Modelagem do quadrirrotor

Como foi abordado na seção 2.1.1, a formulação do MPC torna imprescindível a utilização de um modelo do sistema que deseja-se controlar. Além disso, a abordagem apresentada até o momento trata do MPC linear, e, consequentemente, caso o modelo do sistema que deseja-se controlar seja não linear, uma etapa necessária para a utilização de tal modelo é a sua linearização.

A subseção atual possui como objetivo apresentar a formulação do modelo matemático do quadrirrotor que é utilizado neste trabalho. A modelagem é baseada nos trabalhos de Santana e Braga (2008) e Santana e Borges (2009) e, caso o leitor queira procurar maiores informações a respeito deste tópico, o mesmo é convidado a consultar as referências originais.

Nestes trabalhos, os autores optaram por representar o modelo do quadrirrotor por ângulos de Euler, abordagem amplamente utilizada na literatura (SANTANA; BRAGA, 2008). Para a modelagem do sistema, algumas hipóteses são assumidas:

1. A estrutura do quadrirrotor e as hélices são rígidas;
2. O centro de gravidade e a origem do sistema de coordenadas do sistema coincidem;
3. A estrutura do quadrirrotor é simétrica;
4. Os atuadores do quadrirrotor são idênticos;
5. O arrasto e o empuxo aerodinâmicos do sistema são proporcionais ao quadrado das velocidades de rotação dos motores.

Primeiramente, uma breve explicação do sistema de coordenadas utilizado na modelagem é apresentada. Em seguida, os movimentos de rotação e translação são modelados individualmente, de forma que cada um é abordado em um tópico diferente. Em posse das equações dinâmicas que descrevem os movimentos do quadrirrotor, um processo de linearização é realizado para a obtenção de um modelo em espaço de estados linear do quadrirrotor, que pode ser aplicado à abordagem do MPC apresentada neste trabalho.

2.2.2.1 Sistemas de coordenadas do quadrirrotor

Para a correta modelagem matemática do sistema do quadrirrotor, é necessário estabelecer um sistema de coordenadas através do qual seja possível localizar o sistema no espaço, de modo a permitir que seus movimentos de rotação e translação sejam representados de maneira adequada. Para isso, são considerados dois tipos de coordenadas: o sistema de coordenadas fixo no corpo do quadrirrotor, denominado B, e o sistema de coordenadas fixo na Terra, denominado E. A Figura 2.6 representa ambos os sistemas de coordenadas descritos até aqui.

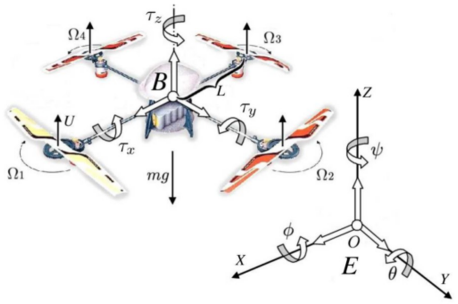

Figura 2.6: Sistema de coordenadas adotado na modelagem do quadrirrotor. Fonte: (SANTANA; BORGES, 2009).

As diferenças lineares e angulares entre esses dois sistemas de coordenadas é responsável pela definição do vetor ζ, definido na Equação 2.79:

$$\zeta = [x \quad y \quad z \quad \phi \quad \theta \quad \psi]^T. \tag{2.79}$$

Ao vetor ζ dá-se o nome de vetor de postura do quadrirrotor. Ele possui informações importantes para a modelagem do sistema:

- **x:** movimento de translação na direção x do sistema de coordenadas fixo na Terra E;
- **y:** movimento de translação na direção y do sistema de coordenadas fixo na Terra E;
- **z:** movimento de translação na direção z do sistema de coordenadas fixo na Terra E;
- ϕ**:** ângulo de rotação em torno do eixo x do sistema de coordenadas fixo na Terra E, conhecido como rolagem (do inglês, *roll*);
- θ**:** ângulo de rotação em torno do eixo y do sistema de coordenadas fixo na Terra E, conhecido como arfagem (do inglês, *pitch*);
- ψ: ângulo de rotação em torno do eixo z do sistema de coordenadas fixo na Terra E, conhecido como guinada (do inglês, *yaw*).

Assume-se que a translação entre os sistemas de coordenadas B e E não influencia na determinação das equações dinâmicas e, consequentemente, assume-se também que as origens de ambos os sistemas de coordenadas são coincidentes.

2.2.2.2 Equacionamento dinâmico do movimento de translação

A primeira etapa necessária para modelar os movimentos de translação do quadrirrotor corresponde à obteção da equação relaciona o empuxo vertical (U) direcionado para cima com a atuação simultânea dos quatro propulsores, resultando na Equação 2.80:

$$U = b \times (\Omega_1^2 + \Omega_2^2 + \Omega_3^2 + \Omega_4^2). \tag{2.80}$$

Na Equação 2.80, os termos Ω_i referem-se à velocidade angular do i-ésimo motor do sistema, enquanto o termo b refere-se ao coeficiente de empuxo das hélices dos motores.

Segundo Madani e Benallague (2006), a estrutura peculiar de distribuição de motores no quadrirrotor limita as maneiras de controlar o mesmo. A atuação nos movimentos de translação é feita com base na projeção do empuxo vertical nos eixos coordenados de acordo com a atitude do quadrirrotor, diferentemente da atuação nos momentos de rotação, que baseam-se nos princípios de conservação de momento angular e desbalanceio de torques, que é abordado no próximo tópico desta subseção.

A segunda equação necessária para descrever os movimentos de translação é a rotação do vetor de empuxo vertical (Equação 2.82). Para a obtenção da mesma, torna-se necessário multiplicar o vetor de empuxo vertical (medido no referencial B) pela matriz de rotação definida na Equação 2.81.

$$R_{\phi\theta\psi} = \begin{bmatrix} cos(\psi)cos(\theta) & cos(\psi)sin(\theta)sin(\phi) - sin(\psi)cos(\phi) & cos(\psi)sin(\theta)cos(\phi) + sin(\psi)sin(\phi) \\ sin(\psi)cos(\theta) & sin(\psi)sin(\theta)sin(\phi) + cos(\psi)cos(\phi) & sin(\psi)sin(\theta)cos(\phi) - sin(\phi)cos(\psi) \\ -sin(\theta) & cos(\theta)sin(\phi) & cos(\theta)cos(\phi) \end{bmatrix}, \tag{2.81}$$

$$U_{rot} = R_{\phi\theta\psi} \times \begin{bmatrix} 0 \\ 0 \\ U \end{bmatrix} = \begin{bmatrix} cos(\psi)sin(\theta)cos(\phi) + sin(\psi)sin(\theta) \\ sin(\psi)sin(\theta)cos(\phi) - sin(\phi)cos(\psi) \\ cos(\theta)cos(\phi) \end{bmatrix}. \tag{2.82}$$

Por fim, aplicando-se a segunda Lei de Newton a cada uma das componentes, obtém-se as Equações 2.83, 2.84 e 2.85.

$$\frac{d^2x}{dt^2} = (cos(\psi)sin(\theta)cos(\phi) + sin(\psi)sin(\theta)) \times \frac{U}{m}, \tag{2.83}$$

$$\frac{d^2y}{dt^2} = (sin(\psi)sin(\theta)cos(\phi) - sin(\phi)cos(\psi)) \times \frac{U}{m}, \tag{2.84}$$

$$\frac{d^2z}{dt^2} = -g + (cos(\theta)cos(\phi)) \times \frac{U}{m}. \tag{2.85}$$

Nas equações acima, m corresponde à massa total da estrutura do quadrirrotor e g corresponde à aceleração da gravidade local.

2.2.2.3 Equacionamento dinâmico do movimento de rotação

A segunda dinâmica necessária para a descrição matemática completa do quadrirrotor refere-se ao movimento de rotação do mesmo. O equacionamento dessa dinâmica é mais complexo do que o equacionamento das dinâmicas de rotação, principalmente por conta da abordagem do formalismo de Euler-Lagrange adotada em Santana e Braga (2008). Essa abordagem permite que as equações de rotação sejam obtidas de maneira mais simples do que utilizando-se as equações de Newton, com base na análise de energia e trabalho do sistema, mas envolve também muitas manipulações matemáticas, que tornam o equacionamento mais complexo.

O primeiro passo necessário para definir essas equações, é definir o conceito de Lagrangiano (mostrado na Equação 2.86):

$$\begin{cases} \mathcal{L} = E - V \\ \frac{d}{dt}\left(\frac{\partial L}{\partial \rho_i}\right) - \frac{\partial L}{\partial \rho_i} = \gamma_i \end{cases}. \tag{2.86}$$

Na Equação 2.86, E corresponde à energia cinética total, V é a energia potencial total, ρ_i é a i-ésima coordenada generalizada, ou seja, grau de liberdade e γ_i é a força resultante não-conservativa que é capaz de realizar trabalho na direção de ρ_i.

Para o segundo passo, é necessário considerar um ponto qualquer do quadrirrotor com coordenadas medidas no referencial B do quadrirrotor: $p_B = [x\ y\ z]^T$. A posição do ponto p_B em relação ao sistema de coordenadas fixo na Terra, E, é dada por p_E e pode ser determinada por meio da multiplicação de p_B pela matriz de rotação definida na Equação 2.81, resultando na Equação 2.87:

$$p_E = p_B \times R_{\phi\theta\psi} = \begin{bmatrix} (cos(\psi)cos(\theta))x + (cos(\psi)sin(\theta)\sin(\phi) - sin(\psi)cos(\phi))y + (cos(\psi)sin(\theta)cos(\phi) + sin(\psi)sin(\phi))z \\ (sin(\psi)cos(\theta))x + (sin(\psi)sin(\theta)sin(\phi) + cos(\psi)cos(\phi))y + (sin(\psi)sin(\theta)cos(\phi) - sin(\phi)cos(\psi))z \\ (-sin(\theta))x + (cos(\theta)sin(\phi))y + (cos(\theta)cos(\phi))z \end{bmatrix}. \tag{2.87}$$

A energia cinética do ponto p_E, denominada K_{p_E} pode ser calculada por meio da Equação 2.88:

$$K_{p_E} = \frac{1}{2}\left(\left\|\frac{dp_E}{dt}\right\|\right)^2 dm. \tag{2.88}$$

A Equação 2.88 possui o operador $\|\|$, que representa a norma Euclidiana, e dm, que representa o elemento de massa associado a p_E.

Para determinar o valor da energia cinética total do quadrirrotor (E), é necessário integrar a Equação 2.88 ao longo de toda a estrutura do mesmo, denominada $\mathcal{C}$, resultando na Equação 2.89.

$$\begin{aligned} E = \ & \tfrac{1}{2}\int_{\mathcal{C}}(y^2+z^2)dm(\dot{\phi}^2 - 2\dot{\psi}\dot{\phi}sin\theta + \dot{\psi}^2sin^2\theta) \\ & +\tfrac{1}{2}\int_{\mathcal{C}}(z^2+x^2)dm(\dot{\theta}^2cos^2(\theta) + 2\dot{\theta}\dot{\psi}sin(\phi)cos(\phi)cos(\theta) + \dot{\psi}^2sin^2(\phi)cos^2(\theta)) \\ & +\tfrac{1}{2}\int_{\mathcal{C}}(x^2+y^2)dm(\dot{\theta}^2sin^2(\phi) - 2\dot{\theta}\dot{\psi}sin(\phi)cos(\phi)cos(\theta) + \dot{\psi}^2cos^2(\phi)cos^2(\theta)) \\ & +\int_{\mathcal{C}} xydm(\dot{\psi}^2sin(\phi)sin(\theta)cos(\theta) + \dot{\psi}(cos(\phi)sin(\theta)\dot{\theta} - sin(\phi)cos(\theta)\dot{\phi}) - cos(\phi)\dot{\phi}\dot{\theta}) \\ & +\int_{\mathcal{C}} xzdm(\dot{\psi}^2cos(\phi)sin(\theta)cos(\theta) + \dot{\psi}(-cos(\phi)cos(\theta)\dot{\phi} - sin(\phi)sin(\theta)\dot{\theta}) + sin(\phi)\dot{\phi}\dot{\theta}) \\ & +\int_{\mathcal{C}} yzdm(-\dot{\psi}^2sin(\phi)cos(\phi)cos^2(\theta) + \dot{\psi}(sin^2(\phi)cos(\theta)\dot{\theta} - cos^2(\phi)cos(\theta)\dot{\theta}) + sin(\phi)cos(\phi)\dot{\theta}^2) \end{aligned}. \tag{2.89}$$

É importante notar que, para a obtenção da Equação 2.89, desprezou-se a energia de translação associada ao deslocamento de B em relação a E

As três primeiras integrais da Equação 2.89 correspondem aos momentos de inércia do quadrirrotor em torno dos eixos x, y e z, respectivamente. O momento de inércia em torno do eixo x é denominado I_{xx} e, de forma análoga, os momentos de inércia em torno dos eixos y e z são denominados I_{yy} e I_{zz}, respectivamente. Já as três últimas integrais presentes na Equação 2.89 estão relacionadas aos produtos de inércia do sistema. Como assume-se que o quadrirrotor possui uma estrutura simétrica, esses termos podem ser considerados nulos, o que permite rearranjar a Equação 2.89, dando origem à Equação 2.90.

$$
\begin{aligned}
E = \quad & \tfrac{1}{2} I_{xx} \left(\tfrac{d\phi}{dt} - \tfrac{d\psi}{dt} sin(\theta) \right)^2 \\
& + \tfrac{1}{2} I_{yy} \left(\tfrac{d\theta}{dt} cos(\phi) + \tfrac{d\psi}{dt} sin(\phi) cos(\theta) \right)^2 \\
& + \tfrac{1}{2} I_{zz} \left(\tfrac{d\theta}{dt} sin(\phi) - \tfrac{d\psi}{dt} cos(\phi) cos(\theta) \right)^2
\end{aligned} . \tag{2.90}
$$

Com a obtenção da Equação 2.90, é possível estabelecer uma comparação com a expressão geral da energia cinética de um corpo que gira livremente no espaço com velocidades ω_x(velocidade de rotação em torno do eixo x), ω_y(velocidade de rotação em torno do eixo y) e ω_z(velocidade de rotação em torno do eixo z). Essa Equação pode ser vista logo abaixo (Equação 2.91).

$$
E = \frac{1}{2} I_{xx} \omega_x^2 + \frac{1}{2} I_{yy} \omega_y^2 + \frac{1}{2} I_{zz} \omega_z^2 . \tag{2.91}
$$

Ao comparar-se as Equações 2.90 e 2.91, é possível derivar uma relação matricial (Eq. 2.92) entre as taxas de variação dos ângulos de Euler com as taxas reais de rotação (velocidades angulares) do quadrirrotor.

$$
\begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} = \begin{bmatrix} 1 & 0 & -sin(\theta) \\ 0 & cos(\phi) & sin(\phi)cos(\theta) \\ 0 & -sin(\phi) & cos(\phi)cos(\theta) \end{bmatrix} \times \begin{bmatrix} \frac{d\phi}{dt} \\ \frac{d\theta}{dt} \\ \frac{d\psi}{dt} \end{bmatrix} . \tag{2.92}
$$

Ao inverter-se a matriz definida na Equação 2.92, é possível calcular diretamente as velocidades angulares dos ângulos de Euler como função das velocidades angulares do quadrirrotor, resultando na Equação 2.93.

$$
\begin{bmatrix} \frac{d\phi}{dt} \\ \frac{d\theta}{dt} \\ \frac{d\psi}{dt} \end{bmatrix} = \begin{bmatrix} 1 & (sin(\theta) \frac{sin(\phi)}{(cos(\theta))(cos^2(\phi)+sin^2(\phi))}) & (sin(\theta)) \frac{cos(\phi)}{(cos(\theta))(cos^2(\phi)+sin^2(\phi))} \\ 0 & \frac{cos(\phi)}{cos^2(\phi)+sin^2(\phi)} & -\frac{sin(\phi)}{cos^2(\phi)+sin^2(\phi)} \\ 0 & \frac{sin(\phi)}{(cos(\theta))(cos^2(\phi)+sin^2(\phi))} & \frac{cos(\phi)}{(cos(\theta))(cos^2(\phi)+sin^2(\phi))} \end{bmatrix} \times \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} . \tag{2.93}
$$

A terceira etapa necessária para o equacionamento das dinâmicas de rotação do quadrirrotor é obter a energia potencial associada a esse tipo de movimento, que pode ser obtida de maneira menos complexa do que a energia cinética obtida anteriormente. Para tal, assume-se que a aceleração gravitacional g é a mesma em todos os pontos do quadrirrotor e, portanto, a energia potencial do ponto p_E pode ser expressa por meio da Equação 2.94.

$$
V_{p_E} = g \times ([0\ 0\ 1] \times p_E) \times dm. \tag{2.94}
$$

Na Equação 2.94, o vetor $[0\ 0\ 1]$ é utilizado para expressar que somente a coordenada de p_E no eixo z é considerada no cálculo da energia potencial.

O procedimento para obtenção da energia potencial total do quadrirrotor é o mesmo para a obtenção da energia cinética total: integrar a Equação associada ao ponto p_E, na Equação 2.94, ao longo de toda a

estrutura $\mathcal{C}$ do quadrirrotor, resultando na Equação 2.95:

$$\begin{aligned} V &= g\int_{\mathcal{C}}(-sin(\theta)x + sin(\phi)cos(\theta)y + cos(\phi)cos(\theta)z)dm \\ &= \left(\int_{\mathcal{C}} xdm\right)(-gsin(\theta)) + \left(\int_{\mathcal{C}} ydm\right)(gsin(\phi)cos(\theta)) + \left(\int_{\mathcal{C}} zdm\right)(gcos(\phi)cos(\theta)) \end{aligned} . \quad (2.95)$$

As integrais entre parênteses na Equação 2.95 estão relacionadas com as coordenadas x, y e z do centro de massa do quadrirrotor em relação ao sistema de coordenadas E. Entretanto, como as hipóteses assumidas no modelamento matemático do quadrirrotor incluem a coincidência das origens dos sistemas B e E, têm-se que essas integrais são nulas, ou seja, $V = 0$ e este termo pode ser eliminado dos próximos cálculos.

A quarta e última etapa para equacionar as dinâmicas de rotação do quadrirrotor corresponde à definição dos graus de liberdade ρ_i e as forças não conservativas γ_i. Acompanhando toda a formulação da dinâmica de rotação do quadrirrotor é natural assumir que os graus de liberdade ρ_i do sistema são dados pelos ângulos do mesmo (ϕ, θ e ψ) e as forças não conservativas γ_i associadas a eles são os torques em torno dos eixos x, y e z (τ_x, τ_y e τ_z, respectivamente). Dentre estes, ainda é necessário definir o equacionamento dos torques, dado pelo desbalanceamento de empuxo e rotações entre os quatro motores, conforme ilustrado nas equações abaixo (Equações 2.96 - 2.98):

$$\tau_x = bL(\Omega_4^2 - \Omega_2^2), \quad (2.96)$$

$$\tau_y = bL(\Omega_3^2 - \Omega_1^2), \quad (2.97)$$

$$\tau_z = d(\Omega_1^2 - \Omega_2^2 + \Omega_3^2 - \Omega_4^2). \quad (2.98)$$

Nas equações acima, b representa o coeficiente de empuxo das hélices do quadrirrotor, medido em $[Ns^2]$, L corresponde à meia envergadura do mesmo, medida em $[m]$, d é o coeficiente de arrasto, medido em $[Nms^2]$ e Ω_i é a velocidade de rotação do i-ésimo motor do sistema, medida em $[rad/s]$.

As equações 2.96, 2.97, 2.98 e a Equação 2.80, definida durante o equacionamento do movimento de translação do quadrirrotor, formam um sistema não linear com quatro equações e quatro incógnitas, cuja solução corresponde às velocidades de rotação que devem ser aplicadas aos motores das hélices do quadrirrotor para gerar os torques e o empuxo vertical necessários. Para tal, é necessário realizar uma substituição de variáveis da forma $\Omega_i^2 = v_i$, com a hipótese de que $\Omega_i \geq 0$,$i \in 1, 2, 3, 4$. Após resolver esse sistema de equações, ao retornar para as variáveis originais, obtêm-se o conjunto de Equações 2.99 abaixo, com as equações para a rotação de cada motor.

$$\begin{cases} \Omega_1 = \frac{1}{2}\sqrt{-\frac{-bL\tau_z+2d\tau_y-dLU}{bLd}} \\ \Omega_2 = \frac{1}{2}\sqrt{-\frac{-bL\tau_z-dLU+2d\tau_x}{bLd}} \\ \Omega_3 = \frac{1}{2}\sqrt{\frac{bL\tau_z+2d\tau_y+dLU}{bLd}} \\ \Omega_4 = \frac{1}{2}\sqrt{-\frac{bL\tau_z-dLU-2d\tau_x}{bLd}} \end{cases} . \tag{2.99}$$

Com posse de todas as equações descritas nessa subseção, é possível obter as equações diferenciais que descrevem a dinâmica do movimento de rotação do quadrirrotor. Para isso, deve-se substituir as equações 2.90 e 2.95 na Equação 2.86, resultando nas equações 2.100, 2.101 e 2.102, que descrevem as dinâmicas de rotação de ϕ, θ e ψ, respectivamente.

$$\begin{aligned} \frac{d^2\phi}{dt^2} = \quad & \frac{d^2\psi}{dt^2}sin(\theta) + \frac{\frac{d\Psi}{dt}\frac{d\theta}{dt}cos(\theta)(I_{xx}+(I_{yy}-I_{zz})(2cos(\phi)^2-1))}{I_{xx}} \\ & -\frac{1}{2}\frac{d\theta^2}{dt}sin(2\phi)\frac{I_{yy}-I_{zz}}{I_{xx}} + \frac{1}{2}\frac{d\psi^2}{dt}sin(2\phi)cos(\theta)^2\frac{I_{yy}-I_{zz}}{I_{xx}} + \frac{\tau_x}{I_{xx}} \end{aligned} , \tag{2.100}$$

$$\begin{aligned} \frac{d^2\theta}{dt^2} = \quad & \frac{-\frac{d^2\psi}{dt^2}\frac{1}{2}sin(2\phi)cos(\theta)(I_{yy}-Izz)}{I_{yy}cos(\phi)^2+I_{zz}sin(\phi)^2} \\ & -\frac{\frac{1}{2}\frac{d\psi}{dt}^2sin(2\theta)(-I_{xx}+I_{yy}sin(\phi)^2+I_{zz}cos(\phi)^2)}{I_{yy}cos(\phi)^2+I_{zz}sin(\phi)^2} \\ & +\frac{\frac{d\theta}{dt}\frac{d\phi}{dt}sin(2\phi)(I_{zz}-I_{yy})}{I_{yy}cos(\phi)^2+I_{zz}sin(\phi)^2} \\ & +\frac{\frac{d\psi}{dt}\frac{d\phi}{dt}cos(\theta)((cos(2\phi))(I_{yy}-I_{zz})+I_{xx})+\tau_y)}{I_{yy}cos(\phi)^2+I_{zz}sin(\phi)^2} \end{aligned} , \tag{2.101}$$

$$\begin{aligned} \frac{d^2\psi}{dt^2} = \quad & \frac{\frac{d^2\phi}{dt^2}sin(\theta)I_{xx}-\frac{d^2\theta}{dt^2}\frac{1}{2}sin(2\phi)cos(\theta)(I_{yy}-I_{zz})}{cos(\theta)^2(I_{zz}cos(\theta)^2)+I_{yy}sin(\phi)^2)+sin(\theta)^2I_{xx}} \\ & -\frac{\frac{d\theta}{dt}\frac{d\psi}{dt}sin(2\theta)(I_{xx}-I_{zz}cos(\phi)^2+I_{yy}sin(\phi)^2)}{cos(\theta)^2(I_{zz}cos(\phi)^2+I_{yy}sin(\phi)^2)+sin(\theta)^2I_{xx}} \\ & -\frac{\frac{d\theta}{dt}\frac{d\psi}{dt}sin(2\theta)(I_{xx}-I_{zz}cos(\phi)^2+I_{yy}sin(\phi)^2)}{cos(\theta)^2(I_{zz}cos(\phi)^2+I_{yy}sin(\phi)^2)+sin(\theta)^2I_{xx}} \\ & +\frac{\frac{d\psi}{dt}\frac{d\theta}{dt}sin(2\phi)cos(\theta)^2(I_{yy}-I_{zz})}{cos(\theta)^2(I_{zz}cos(\phi)^2+I_{yy}sin(\phi)^2)+sin(\theta)^2I_{xx}} \\ & -\frac{\frac{d\theta}{dt}\frac{d\phi}{dt}cos(\theta)(I_{xx}+(2cos(\phi)^2-1)(I_{yy}-I_{zz}))}{cos(\theta)^2(I_{zz}cos(\phi)^2+I_{yy}sin(\phi)^2)+sin(\theta)^2I_{xx}} \\ & +\frac{\frac{1}{2}\frac{d\theta}{dt}^2sin(2\phi)sin(\theta)(I_{yy}-I_{zz})+\tau_z}{cos(\theta)^2(I_{zz}cos(\phi)^2+I_{yy}sin(\phi)^2)+sin(\theta)^2I_{xx}} \end{aligned} . \tag{2.102}$$

2.2.2.4 Linearização do modelo

Com as equações dinâmicas que descrevem as dinâmicas do quadrirrotor, nosso sistema de interesse, é necessário representá-las em espaço de estados, para que seja possível aplicar o MPC da maneira que foi demonstrada na seção anterior (seção 2.1.1). Esse é o último passo para utilizar o modelo na formulação do controlador MPC utilizado neste trabalho.

Essas etapas correspondem ao processo que foi utilizado no trabalho de Lopes *et al.* (2011), resultando nas mesmas matrizes $\mathbf{A}$ e $\mathbf{B}$ apresentadas no mesmo.

A primeira etapa corresponde a uma aproximação por meio de uma expansão de Taylor de primeira ordem das equações dinâmicas do quadrirrotor em torno de um ponto de equilíbrio. Em Lopes *et al.* (2011), esse ponto de equilíbrio foi arbitrado como sendo $z = 10$m, o que implica em um vetor $\bar{\xi} = [x\ \dot{x}\ y\ \dot{y}\ z\ \dot{z}\ \phi\ \dot{\phi}\ \theta\ \dot{\theta}\ \psi\ \dot{\psi}] = [0\ 0\ 0\ 0\ 10\ 0\ 0\ 0\ 0\ 0\ 0\ 0]$, que corresponde a uma variação do vetor de postura ξ do quadrirrotor contendo informações adicionais referentes à variação de cada uma das grandezas presentes no vetor de postura ξ original.

Em outras palavras, o quadrirrotor foi linearizado com base em um ponto de equilíbrio que representa a situação em que a aeronave encontra-se pairando a 10m de altura, com todos os ângulos de atitude iguais a zero.

Para a etapa da expansão de Taylor de primeira ordem também é considerado um vetor de controle $\bar{u} = [\bar{u}_1\ \bar{u}_2\ \bar{u}_3\ \bar{u}_4]^T$, que corresponde a um condição de vôo em que o quadrirrotor encontra-se parado, planando no ar. As variáveis de controle do quadrirrotor correspondem às velocidades angulares de cada motor, de forma que cada elemento de $\bar{u}$ é dado em rad/s.

Essa etapa de linearização corresponde basicamente em utilizar as derivadas de cada variável de estado e derivá-las em função de cada variável de estado individual, conforme é mostrado na Equação 2.103, e derivar as mesmas equações em função de cada variável de controle individual, conforme mostrado na Equação 2.104. Essa etapa gera as matrizes de estado contínuas $\mathbf{A_c}$ e $\mathbf{B_c}$. O sub-índice c refere-se justamente ao fato de ambas serem matrizes contínuas.

$$\mathbf{A_c} = \begin{bmatrix}
\frac{\partial \dot{x}}{\partial x} & \frac{\partial \dot{x}}{\partial \dot{x}} & \frac{\partial \dot{x}}{\partial y} & \frac{\partial \dot{x}}{\partial \dot{y}} & \frac{\partial \dot{x}}{\partial z} & \frac{\partial \dot{x}}{\partial \dot{z}} & \frac{\partial \dot{x}}{\partial \phi} & \frac{\partial \dot{x}}{\partial \dot{\phi}} & \frac{\partial \dot{x}}{\partial \theta} & \frac{\partial \dot{x}}{\partial \dot{\theta}} & \frac{\partial \dot{x}}{\partial \psi} & \frac{\partial \dot{x}}{\partial \dot{\psi}} \\
\frac{\partial \ddot{x}}{\partial x} & \frac{\partial \ddot{x}}{\partial \dot{x}} & \frac{\partial \ddot{x}}{\partial y} & \frac{\partial \ddot{x}}{\partial \dot{y}} & \frac{\partial \ddot{x}}{\partial z} & \frac{\partial \ddot{x}}{\partial \dot{z}} & \frac{\partial \ddot{x}}{\partial \phi} & \frac{\partial \ddot{x}}{\partial \dot{\phi}} & \frac{\partial \ddot{x}}{\partial \theta} & \frac{\partial \ddot{x}}{\partial \dot{\theta}} & \frac{\partial \ddot{x}}{\partial \psi} & \frac{\partial \ddot{x}}{\partial \dot{\psi}} \\
\frac{\partial \dot{y}}{\partial x} & \frac{\partial \dot{y}}{\partial \dot{x}} & \frac{\partial \dot{y}}{\partial y} & \frac{\partial \dot{y}}{\partial \dot{y}} & \frac{\partial \dot{y}}{\partial z} & \frac{\partial \dot{y}}{\partial \dot{z}} & \frac{\partial \dot{y}}{\partial \phi} & \frac{\partial \dot{y}}{\partial \dot{\phi}} & \frac{\partial \dot{y}}{\partial \theta} & \frac{\partial \dot{y}}{\partial \dot{\theta}} & \frac{\partial \dot{y}}{\partial \psi} & \frac{\partial \dot{y}}{\partial \dot{\psi}} \\
\frac{\partial \ddot{y}}{\partial x} & \frac{\partial \ddot{y}}{\partial \dot{x}} & \frac{\partial \ddot{y}}{\partial y} & \frac{\partial \ddot{y}}{\partial \dot{y}} & \frac{\partial \ddot{y}}{\partial z} & \frac{\partial \ddot{y}}{\partial \dot{z}} & \frac{\partial \ddot{y}}{\partial \phi} & \frac{\partial \ddot{y}}{\partial \dot{\phi}} & \frac{\partial \ddot{y}}{\partial \theta} & \frac{\partial \ddot{y}}{\partial \dot{\theta}} & \frac{\partial \ddot{y}}{\partial \psi} & \frac{\partial \ddot{y}}{\partial \dot{\psi}} \\
\frac{\partial \dot{z}}{\partial x} & \frac{\partial \dot{z}}{\partial \dot{x}} & \frac{\partial \dot{z}}{\partial y} & \frac{\partial \dot{z}}{\partial \dot{y}} & \frac{\partial \dot{z}}{\partial z} & \frac{\partial \dot{z}}{\partial \dot{z}} & \frac{\partial \dot{z}}{\partial \phi} & \frac{\partial \dot{z}}{\partial \dot{\phi}} & \frac{\partial \dot{z}}{\partial \theta} & \frac{\partial \dot{z}}{\partial \dot{\theta}} & \frac{\partial \dot{z}}{\partial \psi} & \frac{\partial \dot{z}}{\partial \dot{\psi}} \\
\frac{\partial \ddot{z}}{\partial x} & \frac{\partial \ddot{z}}{\partial \dot{x}} & \frac{\partial \ddot{z}}{\partial y} & \frac{\partial \ddot{z}}{\partial \dot{y}} & \frac{\partial \ddot{z}}{\partial z} & \frac{\partial \ddot{z}}{\partial \dot{z}} & \frac{\partial \ddot{z}}{\partial \phi} & \frac{\partial \ddot{z}}{\partial \dot{\phi}} & \frac{\partial \ddot{z}}{\partial \theta} & \frac{\partial \ddot{z}}{\partial \dot{\theta}} & \frac{\partial \ddot{z}}{\partial \psi} & \frac{\partial \ddot{z}}{\partial \dot{\psi}} \\
\frac{\partial \dot{\phi}}{\partial x} & \frac{\partial \dot{\phi}}{\partial \dot{x}} & \frac{\partial \dot{\phi}}{\partial y} & \frac{\partial \dot{\phi}}{\partial \dot{y}} & \frac{\partial \dot{\phi}}{\partial z} & \frac{\partial \dot{\phi}}{\partial \dot{z}} & \frac{\partial \dot{\phi}}{\partial \phi} & \frac{\partial \dot{\phi}}{\partial \dot{\phi}} & \frac{\partial \dot{\phi}}{\partial \theta} & \frac{\partial \dot{\phi}}{\partial \dot{\theta}} & \frac{\partial \dot{\phi}}{\partial \psi} & \frac{\partial \dot{\phi}}{\partial \dot{\psi}} \\
\frac{\partial \ddot{\phi}}{\partial x} & \frac{\partial \ddot{\phi}}{\partial \dot{x}} & \frac{\partial \ddot{\phi}}{\partial y} & \frac{\partial \ddot{\phi}}{\partial \dot{y}} & \frac{\partial \ddot{\phi}}{\partial z} & \frac{\partial \ddot{\phi}}{\partial \dot{z}} & \frac{\partial \ddot{\phi}}{\partial \phi} & \frac{\partial \ddot{\phi}}{\partial \dot{\phi}} & \frac{\partial \ddot{\phi}}{\partial \theta} & \frac{\partial \ddot{\phi}}{\partial \dot{\theta}} & \frac{\partial \ddot{\phi}}{\partial \psi} & \frac{\partial \ddot{\phi}}{\partial \dot{\psi}} \\
\frac{\partial \dot{\theta}}{\partial x} & \frac{\partial \dot{\theta}}{\partial \dot{x}} & \frac{\partial \dot{\theta}}{\partial y} & \frac{\partial \dot{\theta}}{\partial \dot{y}} & \frac{\partial \dot{\theta}}{\partial z} & \frac{\partial \dot{\theta}}{\partial \dot{z}} & \frac{\partial \dot{\theta}}{\partial \phi} & \frac{\partial \dot{\theta}}{\partial \dot{\phi}} & \frac{\partial \dot{\theta}}{\partial \theta} & \frac{\partial \dot{\theta}}{\partial \dot{\theta}} & \frac{\partial \dot{\theta}}{\partial \psi} & \frac{\partial \dot{\theta}}{\partial \dot{\psi}} \\
\frac{\partial \ddot{\theta}}{\partial x} & \frac{\partial \ddot{\theta}}{\partial \dot{x}} & \frac{\partial \ddot{\theta}}{\partial y} & \frac{\partial \ddot{\theta}}{\partial \dot{y}} & \frac{\partial \ddot{\theta}}{\partial z} & \frac{\partial \ddot{\theta}}{\partial \dot{z}} & \frac{\partial \ddot{\theta}}{\partial \phi} & \frac{\partial \ddot{\theta}}{\partial \dot{\phi}} & \frac{\partial \ddot{\theta}}{\partial \theta} & \frac{\partial \ddot{\theta}}{\partial \dot{\theta}} & \frac{\partial \ddot{\theta}}{\partial \psi} & \frac{\partial \ddot{\theta}}{\partial \dot{\psi}} \\
\frac{\partial \dot{\psi}}{\partial x} & \frac{\partial \dot{\psi}}{\partial \dot{x}} & \frac{\partial \dot{\psi}}{\partial y} & \frac{\partial \dot{\psi}}{\partial \dot{y}} & \frac{\partial \dot{\psi}}{\partial z} & \frac{\partial \dot{\psi}}{\partial \dot{z}} & \frac{\partial \dot{\psi}}{\partial \phi} & \frac{\partial \dot{\psi}}{\partial \dot{\phi}} & \frac{\partial \dot{\psi}}{\partial \theta} & \frac{\partial \dot{\psi}}{\partial \dot{\theta}} & \frac{\partial \dot{\psi}}{\partial \psi} & \frac{\partial \dot{\psi}}{\partial \dot{\psi}} \\
\frac{\partial \ddot{\psi}}{\partial x} & \frac{\partial \ddot{\psi}}{\partial \dot{x}} & \frac{\partial \ddot{\psi}}{\partial y} & \frac{\partial \ddot{\psi}}{\partial \dot{y}} & \frac{\partial \ddot{\psi}}{\partial z} & \frac{\partial \ddot{\psi}}{\partial \dot{z}} & \frac{\partial \ddot{\psi}}{\partial \phi} & \frac{\partial \ddot{\psi}}{\partial \dot{\phi}} & \frac{\partial \ddot{\psi}}{\partial \theta} & \frac{\partial \ddot{\psi}}{\partial \dot{\theta}} & \frac{\partial \ddot{\psi}}{\partial \psi} & \frac{\partial \ddot{\psi}}{\partial \dot{\psi}}
\end{bmatrix}, \tag{2.103}$$

$$\mathbf{B_c} = \begin{bmatrix}
\frac{\partial \dot{x}}{\partial \Omega_1} & \frac{\partial \dot{x}}{\partial \Omega_2} & \frac{\partial \dot{x}}{\partial \Omega_3} & \frac{\partial \dot{x}}{\partial \Omega_4} \\
\frac{\partial \ddot{x}}{\partial \Omega_1} & \frac{\partial \ddot{x}}{\partial \Omega_2} & \frac{\partial \ddot{x}}{\partial \Omega_3} & \frac{\partial \ddot{x}}{\partial \Omega_4} \\
\frac{\partial \dot{y}}{\partial \Omega_1} & \frac{\partial \dot{y}}{\partial \Omega_2} & \frac{\partial \dot{y}}{\partial \Omega_3} & \frac{\partial \dot{y}}{\partial \Omega_4} \\
\frac{\partial \ddot{y}}{\partial \Omega_1} & \frac{\partial \ddot{y}}{\partial \Omega_2} & \frac{\partial \ddot{y}}{\partial \Omega_3} & \frac{\partial \ddot{y}}{\partial \Omega_4} \\
\frac{\partial \dot{z}}{\partial \Omega_1} & \frac{\partial \dot{z}}{\partial \Omega_2} & \frac{\partial \dot{z}}{\partial \Omega_3} & \frac{\partial \dot{z}}{\partial \Omega_4} \\
\frac{\partial \ddot{z}}{\partial \Omega_1} & \frac{\partial \ddot{z}}{\partial \Omega_2} & \frac{\partial \ddot{z}}{\partial \Omega_3} & \frac{\partial \ddot{z}}{\partial \Omega_4} \\
\frac{\partial \dot{\phi}}{\partial \Omega_1} & \frac{\partial \dot{\phi}}{\partial \Omega_2} & \frac{\partial \dot{\phi}}{\partial \Omega_3} & \frac{\partial \dot{\phi}}{\partial \Omega_4} \\
\frac{\partial \ddot{\phi}}{\partial \Omega_1} & \frac{\partial \ddot{\phi}}{\partial \Omega_2} & \frac{\partial \ddot{\phi}}{\partial \Omega_3} & \frac{\partial \ddot{\phi}}{\partial \Omega_4} \\
\frac{\partial \dot{\theta}}{\partial \Omega_1} & \frac{\partial \dot{\theta}}{\partial \Omega_2} & \frac{\partial \dot{\theta}}{\partial \Omega_3} & \frac{\partial \dot{\theta}}{\partial \Omega_4} \\
\frac{\partial \ddot{\theta}}{\partial \Omega_1} & \frac{\partial \ddot{\theta}}{\partial \Omega_2} & \frac{\partial \ddot{\theta}}{\partial \Omega_3} & \frac{\partial \ddot{\theta}}{\partial \Omega_4} \\
\frac{\partial \dot{\psi}}{\partial \Omega_1} & \frac{\partial \dot{\psi}}{\partial \Omega_2} & \frac{\partial \dot{\psi}}{\partial \Omega_3} & \frac{\partial \dot{\psi}}{\partial \Omega_4} \\
\frac{\partial \ddot{\psi}}{\partial \Omega_1} & \frac{\partial \ddot{\psi}}{\partial \Omega_2} & \frac{\partial \ddot{\psi}}{\partial \Omega_3} & \frac{\partial \ddot{\psi}}{\partial \Omega_4}
\end{bmatrix}. \tag{2.104}$$

É importante ressaltar que as derivadas das velocidades lineares e angulares ($\ddot{x}$, $\ddot{y}$, $\ddot{z}$, $\ddot{\phi}$, $\ddot{\theta}$ e $\ddot{\psi}$) são dadas pelas equações 2.83, 2.84 e 2.85, 2.101, 2.100 e 2.102, que foram apresentadas anteriormente. Já as derivadas das coordenadas espaciais e angulares do quadrirrotores são dadas simplesmente por $\dot{x}$, $\dot{y}$, $\dot{z}$, $\dot{\phi}$, $\dot{\theta}$ e $\dot{\psi}$.

Uma vez que as matrizes contínuas são obtidas $\mathbf{A_c}$ e $\mathbf{B_c}$, é necessário discretizá-las, para que o modelo possa implementado na prática. Para isso, foi adotado um período de amostragem $T_s = 50$ms e o sistema é discretizado utilizando um segurador de ordem zero (do inglês, *Zero Order Hold*), da mesma maneira que apresentado em Lopes *et al.* (2011).

Substituindo os valores dos parâmetros do quadrirrotor (apresentados na Tabela 2.2), como massa e momentos de inércia do mesmo em relação aos diferentes eixos de coordenadas, obtêm-se as matrizes de espaço de estados discretizadas $\mathbf{A}$ e $\mathbf{B}$ do quadrirrotor (Equações 2.105 e 2.106).

Tabela 2.2: Parâmetros utilizados na modelagem das equações dinâmicas do quadrirrotor. Fonte: (LOPES et al., 2011).

Símbolo	**Parâmetro**	**Valor**
m	Massa do quadrirrotor	4.000 kg
g	Gravidade local	9.810 m/s^2
I_{xx}	Momento de inércia no eixo x	0.033 kgm^2
I_{yy}	Momento de inércia no eixo y	0.033 kgm^2
I_{zz}	Momento de inércia no eixo z	0.066 kgm^2
L	Meia envergadura do quadrirrotor	0.500 m
b	Coeficiente de empuxo do quadrirrotor	2.640×10^{-4} Ns^2
d	Coeficiente de arrasto do quadrirrotor	7.500×10^{-7} Nms^2
$\bar{u}$	Vetor de controle desejado no ponto de equilíbrio	$[192.800\ 192.800\ 192.800\ 192.800]^T$ rad/s

$$\mathbf{A} = \begin{bmatrix} 1.0000 & 0.0500 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0123 & 0.0002 & 0.0000 & 0.0000 \\ 0.0000 & 1.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.4905 & 0.0123 & 0.0000 & 0.0000 \\ 0.0000 & 0.0000 & 1.0000 & 0.0500 & 0.0000 & 0.0000 & -0.0123 & -0.0002 & 0.0000 & 0.0000 & 0.0000 & 0.0000 \\ 0.0000 & 0.0000 & 0.0000 & 1.0000 & 0.0000 & 0.0000 & -0.4905 & -0.0123 & 0.0000 & 0.0000 & 0.0000 & 0.0000 \\ 0.0000 & 0.0000 & 0.0000 & 0.0000 & 1.0000 & 0.0050 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 \\ 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 1.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 \\ 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 1.0000 & 0.0500 & 0.0000 & 0.0000 & 0.0000 & 0.0000 \\ 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 1.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 \\ 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 1.0000 & 0.0500 & 0.0000 & 0.0000 \\ 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 1.0000 & 0.0000 & 0.0000 \\ 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 1.0000 & 0.0500 \\ 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 1.0000 \end{bmatrix}, \tag{2.105}$$

$$\mathbf{B} = \begin{bmatrix} 0.0000 & 0.0000 & 0.0000 & 0.0000 \\ -0.0002 & 0.0000 & -0.0002 & 0.0000 \\ 0.0000 & 0.0000 & 0.0000 & 0.0000 \\ 0.0000 & 0.0002 & 0.0000 & -0.0002 \\ 0.0001 & 0.0001 & 0.0001 & 0.0001 \\ 0.0021 & 0.0021 & 0.0021 & 0.0021 \\ 0.0000 & -0.0012 & 0.0000 & 0.0012 \\ 0.0000 & -0.0472 & 0.0000 & 0.0472 \\ -0.0012 & 0.0000 & 0.0012 & 0.0000 \\ -0.0472 & 0.0000 & 0.0472 & 0.0000 \\ 0.0000 & 0.0000 & 0.0000 & 0.0000 \\ 0.0001 & -0.0001 & 0.0001 & -0.0001 \end{bmatrix}. \tag{2.106}$$

As saídas reguladas do sistema são definidas como sendo as variáveis de estado x, y, z e ψ (coordenadas espaciais e ângulo de arfagem): $y_r = [x\ y\ z\ \psi]^T$. Já as saídas sob as quais são aplicadas restrições, saídas restringidas, são dadas pelas variáveis de estados ϕ e θ (ângulos de rolagem e guinada), ou seja, $y_c = [\phi\ \theta]^T$. Como o modelo do quadrirrotor contêm quatro variáveis de controle, não é possível controlar todas as seis saídas definidas (incluindo tanto saídas reguladas quanto saídas restringidas) ao mesmo tempo. Portanto, restringir os ângulos θ e ϕ é importante pois mantem o quadrirrotor em sua região linear, permitindo o controle do mesmo.

Com base nisso, pode-se definir as matrizes de saída reguladas ($\mathbf{C_r}$) e saídas restringidas ($\mathbf{C_c}$) por

meio das equações 2.107 e 2.108, respectivamente.

$$\mathbf{C_r} = \begin{bmatrix} 1.0000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 \\ 0.0000 & 0.000 & 1.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 \\ 0.0000 & 0.000 & 0.000 & 0.000 & 1.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 \\ 0.0000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 1.000 & 0.000 \end{bmatrix}, \tag{2.107}$$

$$\mathbf{C_c} = \begin{bmatrix} 0.0000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 1.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 \\ 0.0000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 1.000 & 0.000 & 0.000 & 0.000 \end{bmatrix}. \tag{2.108}$$

A matriz $\mathbf{D}$ do quadrirrotor é definida como sendo nula, uma vez que não há transmissão direta das entradas para as saídas do sistema.

Assim, ao fim desta seção, têm-se o modelo em espaço de estados do quadrirrotor que é utilizado ao longo deste trabalho, nas etapas que são apresentadas nas seções de Metodologia (Capítulo 3) e Resultados e Discussões (Capítulo 4).

2.3 FPGAs

2.3.1 Introdução e desenvolvimento histórico dos FPGAs

A primeira aparição de dispositivos lógicos programáveis (do inglês, *Programmable Logic Devices* ou PLDs) na indústria se deu em meados dos anos 1970. A idéia principal por trás desse tipo de tecnologia era de desenvolver circuitos combinacionais que fossem programáveis, mas com um *hardware* que pudesse ser reconfigurado para alcançar condições específicas de acordo com a aplicação desejada. Nesse quesito, esse tipo de tecnologia difere-se de microprocessadores e microcontroladores, justamente por esses últimos possuírem *hardware* fixo, ainda que possam ser programados (PEDRONI, 2004).

Os primeiros PLDs eram denominados PAL (abreviatura do termo em inglês, *Programmable Logic Arrays*) e só eram capazes de implementar circuitos combinacionais. Posteriormente, nos anos 1980, os dispositivos passaram a contar com *flip-flops*, portas lógicas e multiplexadores, as chamadas macrocélulas, possibilitando a implementação de circuitos sequenciais. A esse tipo de dispositivo denomina-se GAL (abreviatura do termo em inglês, *Generic Logic Arrays*). Hoje, esses tipos de dispositivos são classificados como dispositivos lógicos programáveis simples (do inglês, *Simple Programmable Logic Devices*) (PEDRONI, 2004).

Com o advento das tecnologias de fabrição de circuitos integrados, vários dispositivos GAL começaram a ser fabricados em um mesmo *chip*, dando origem aos chamados dispositivos lógicos programáveis complexos ou CPLDs (do inglês, *Complex Programmable Logic Devices*), que se tornaram populares graças a sua alta densidade e desemepenho, associados a um baixo custo (PEDRONI, 2004). Justamente no meio dos anos 1980, dentro da categoria de CPLDs, surgiram os FPGAs, com o objetivo de implementar circuitos mais complexos e alcançar maiores velocidades de desempenho.

Segundo Trimberger (2015), desde a sua introdução, FPGAs passaram por diversas etapas de desenvol-

vimento. Assim, como feito no trabalho de Lee (2011) para o MPC, Trimberger (2015) divide a evolução dos FPGAs em três etapas principais: era da invenção (1984-1991), era da expansão (1992-1999) e a era da acumulação (2000-2007).

Durante a era da invenção, FPGAs eram pequenas e, portanto, voltadas a aplicações mais simples. Não era incomum encontrar sistemas com múltiplas FPGAs, como em Babb *et al.* (1997) e Vuillemin *et al.* (1996). O primeiro FPGA da empresa *Xilinx*, por exemplo, o modelo XC2064, continha somente 64 CLBs, cada um contendo LUTs (*Lookup Tables*) de três entradas e um registrador (CARTER et al., 1986). Apesar da pequena capacidade em comparação com os dias de hoje, esse sistema possuia maior capacidade de processamento do que os microprocessadores comerciais da época.

O segundo período, era da expansão, corresponde a um grande crescimento da capacidade dos FPGAs. Impulsionado pela lei de Moore, ou seja, que a quantidade de transistores em um *chip* dobra a cada 18 meses (MOORE, 1997), FPGAs começaram a contar com mais recursos de *hardware*, bem como novas tecnologias, eliminando a necessidade de múltiplos FPGAs em um único sistema. Com o aumento da popularidade dos FPGAs, empresas de EDA (do inglês, *Electronic Design Automation*) passaram a desenvolver ferramentas para esses dispositivos, facilitando sua usabilidade (MONMASSON; CIRSTEA, 2007)

Na época da acumulação, FPGAs já eram considerados componentes comuns em sistemas digitais. A capacidade desses dispositivos continuava aumentando, possibilitando que FPGAs alcançassem uma gama maior de aplicações. Com o aumento dos FPGAs, *designs* maiores puderam ser feitos, sendo possível encontrar susbsistemas completos dentro de um único FPGA. Novos tipos de tecnologia e protocolos continuaram a ser desenvolvidos para FPGAs, uma vez que esse tipo de sistema atraia bastante atenção por sua aplicação em atividades computacionalmente intensivas (MONMASSON; CIRSTEA, 2007).

Após a era da acumulação, estendendo-se aos dias atuais, FPGAs evoluíram de tal forma que não são mais simplesmente compostos por blocos lógicos configuráveis (CLBs) que conectam-se entre si por meio de interfaces reprogramáveis (MUÑOZ, 2012); mas são também compostos por blocos integrados com uma lógica programável. Os mesmos possuem diversas funcionalidades integradas com o objetivo de otimizar os recursos de *hardware*, como blocos de DSP (do inglês, *Digital Signal Processors*), incluindo multiplicadores, somadores e acumuladores, implementados diretamente no FPGA; blocos de memória (RAM, ROM, *Flash*), blocos que gerenciam o sinal de *clock* do sistema e, consequentemete, a velocidade de operação do FPGA (geralmente baseados em técnicas de *Phase-Locked-Loop* ou PLL); blocos de comunicação USB, *Ethernet* e muitos outros (MONMASSON et al., 2011).

A Figura 2.7 mostra um diagrama de blocos do *kit* de FPGA utilizado neste trabalho: o *kit* de desenvolvimento KC-705 da Xilinx. É possível perceber pela figura abaixo que o mesmo possui um FPGA Kintex 7 XC7K325T-2FFG900C, além de diversos blocos integrados, para as mais variadas funcionalidades: desde blocos de memória (Flash, DDR3, etc.) até diferentes tipos de blocos de comunicação (*Ethernet*, USB, JTAG, entre outros), além de diversos periféricos que permitem as mais diversas funcionalidades, por meio de chaves (*switches*), LEDs, interface HDMI, display de LCD, etc.

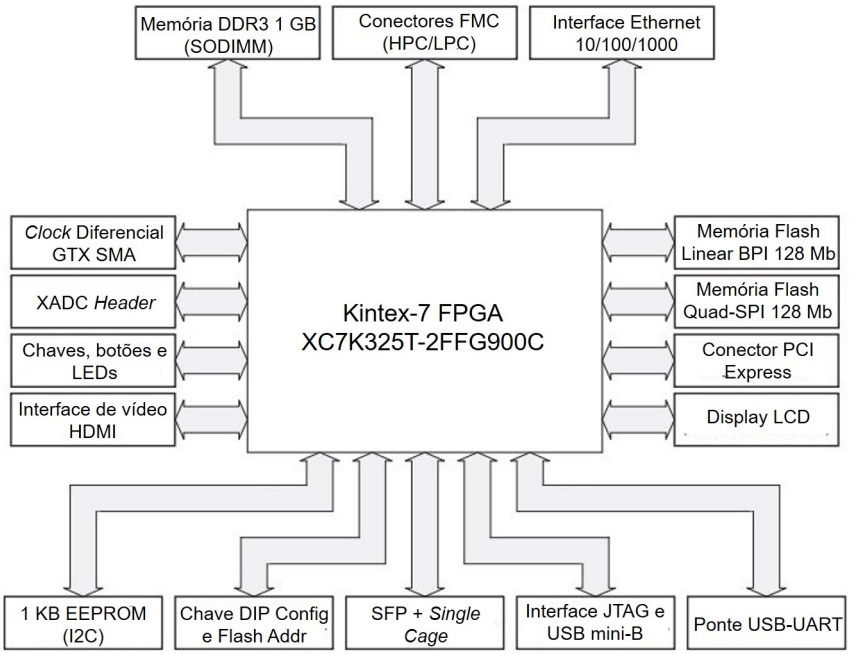

Figura 2.7: Diagrama de blocos do *kit* de desenvolvimento KC-705 da Xilinx. Adaptado de: (XILINX, 2019).

Com toda a evolução ao longo dos anos e a quantidade de recursos disponíveis nos FPGAs atuais, esse tipo de dispositivo apresenta inúmeras vantagens. As principais, de acordo com Joost e Salomon (2005) e ainda de acordo com Rodrigues-Andina, Moure e Valdes (2007) são:

- Alta flexibilidade;
- Reusabilidade;
- Fácil manutenção/atualização (graças à utilização de linguagens de descrição de *hardware* abstratas);
- Processadores embarcados, juntamente com periféricos que permitem que um único *kit* possa implementar soluções com coprocessamento em *software* (soluções conhecidas como *System On Chip* ou SoCs), possibilitando a implementação de *designs* mais complexos de maneira mais simples;
- Alto desempenho e capacidade de paralelismo;
- Implementação de sistemas reconfiguráveis de maneira mais simples;
- Facilidade de prototipagem de *hardware*.

É possível perceber, portanto, que os FPGAs evoluiram bastante desde sua introdução no mercado, com aumento de capacidade e recursos que permitem a sua utilização em aplicações que exigem alta demanda computacional, explorando o paralelismo de soluções em *hardware*. Nessa categoria encaixa-se perfeitamente o MPC: uma estratégia de controle que demanda uma alta carga computacional para sua implementação em sistemas reais, principalmente sistemas complexos e de dinâmicas rápidas, como no caso do quadrirrotor.

Por todas essas razões é possível encontrar na literatura várias implementações não somente do MPC, mas de diversas outras técnicas de controle utilizando FPGAs. Isso é abordado com mais detalhes na próxima subseção.

2.3.2 Aplicação de FPGAs em sistemas de controle e MPC

Atualmente, sistemas de controle industriais buscam alta capacidade de desempenho, flexibilidade e confiabilidade (MONMASSON et al., 2011). Por isso, a velocidade de desempenho de novos componentes e a flexibilidade proveniente de soluções programáveis e/ou reconfiguráveis possibilitam muitas oportunidades para a implementação digital de controladores (MONMASSON; CIRSTEA, 2007).

Levando em consideração tudo que foi discutido até aqui sobre o funcionamento e o desenvolvimento histórico de FPGAs, é possível perceber que esse tipo de dispositivo se encaixa nas características descritas no último parágrafo. De fato, FPGAs são reconhecidos como um tipo de plataforma de desenvolvimento de controladores capazes de mitigar de maneira eficiente alguns problemas e desafios recorrentes no campo de sistemas de controle, tais como (MONMASSON; CIRSTEA, 2007):

1. Diminuição do custo de projeto por conta da quantidade de recursos existentes em um *kit* de desenvolvimento com FPGA, como processadores de *software*, interfaces analógicas e outros em um único *chip* (SoC);
2. Sistemas de controle embarcados com restrições das mais diversas naturezas;
3. Melhora de desempenho do controlador, possibilitando, em alguns casos, a diminuição drástica do tempo de execução por meio de arquiteturas que exploram o paralelismo das soluções.

Por isso, FPGAs têm sido e continuam sendo utilizadas para a implementação de sistemas de controle nas mais diversas áreas de aplicação. Restringindo a aplicação de FPGAs para o MPC como sistema de controle, ainda assim é possível encontrar uma vasta gama de trabalhos e contribuições nas mais diversas áreas, tanto para sistemas lineares quanto sistemas não lineares. De acordo com Abughalieh e Alawneh (ABUGHALIEH; ALAWNEH, 2019), há três tipos de arquiteturas que se destacam quando deseja-se implementar o MPC de maneira paralela: processadores multinúcleos, GPUs (do inglês, *Graphics Processing Unit*) e os FPGAs propriamente ditos. A principal vantagem dos FPGAs quando comparados a esses outros tipos de arquiteturas corresponde à relação entre quantidade de recursos disponíveis nos *kits* atuais e o seu baixo custo em comparação com GPUs e processadores multinúcleos dedicados. Todas essas características, aliadas à capacidade de operar em altas frequências (chegando a faixas em MHz), tornam o FPGA um dispostivo muito utilizado para paralelizar e implementar o MPC.

A Tabela 2.3 apresenta uma combinação de alguns dos trabalhos mais relevantes encontrados na literatura, mostrando a área de aplicação do trabalho, bem como se o sistema é linear ou não linear. Além disso, também são incluidos detalhes sobre o valor do horizonte de predição *N* utilizado, a quantidade de variáveis de estado (n_x), variáveis de controle (n_u) e o período de amostragem (T_s) adotado para cada sistema. Nos parágrafos que se seguem, estes trabalhos são explicadas em maiores detalhes.

Tabela 2.3: Síntese de trabalhos encontrados na literatura referentes à implementação de diferentes tipos de controladores preditivos utilizando FPGAs.

Autores	**Aplicação**	**Classificação**	N	**FPGA**	$\mathbf{T_s}$	$\mathbf{n_x}$	$\mathbf{n_u}$
Ling, K.V *et al.* (2008)	Aeronave Cessena Citation 500	Linear	$N = 10$	*Virtex-II*	0.5s	4	1
Vouzis, P. *et al.* (2009)	Antena giratória Regulação de glicose	Linear Não linear	$N = 20$ $N = 5$	*Virtex-IV*	0.1s 5 min	2 4	1
Luo, B. *et al.* (2011)	Aeronave Cessena Citation 500	Linear	$N = 10$	*Virtex-5*	0.5s	4	1
Joos, A. *et al.* (2012)	VANT de asa fixa	Não linear	$N = 16$	*Spartan 3*	0.5s	6	2
Kapernick *et al.* (2014)	Guindaste Reator tanque agitado contínuo	Não linear Não linear	$N = 30$ $N = 30$	*Zynq-7000* *Artix-7*	1s 1200s	7 4	2 2
Hartley, E. *et al.* (2014)	Aeronave Boeing 747-200	Não linear	$N = 5$ $N = 12$	*Virtex-6*	0.2s	17	2
Xu, F. *et al.* (2016)	Sistema de ignição	Não linear	$N = 5$	*Stratix III*	10ms	4	1
Ayala, H. *et al.* (2016)	Manipulador robótico de *link* único	Não linear	$N = 50$	*Artix-7* *Cyclone IV*	0.01s	4	1

O trabalho de Ling, Wu, Maciejowski (2008) implementa em um FPGA *Virtex-II* uma abordagem linear do MPC utilizada para controlar o sistema de uma aeronave Cessena Citation 500. Os autores utilizam um *solver* de QP do tipo método do ponto interior e uma das características deste trabalho é a utilização de ferramentas de geração de código automática para programar o FPGA: no caso, geração automática convertendo os algoritmos inicialmente de MATLAB/Simulink para *Handel-C*, uma das linguagens de descrição de *hardware* disponíveis. Nesse trabalho, todo o algoritmo foi mapeado em *hardware*, ou seja, tudo é implementado no FPGA.

O modelo da aeronave Cessena Citation 500 é bastante utilizado em trabalhos mais antigos referentes à implementação de MPC em FPGAs, sendo um modelo relativamente simples (4 estados e uma única variável de controle). O horizonte de predição N utilizado também não é muito elevado ($N = 10$). Entretanto, é importante ressaltar que esse trabalho é uma das poucas implementações de MPC em FPGA encontradas na literatura durante a revisão bibliográfica deste trabalho a utilizar ponto flutuante.

Em Vouzis *et al.* (2009), trabalho que também é uma continuação de outra publicação (BLERIS et al., 2006), apresenta-se uma estratégia diferente: uma metodologia de coprocessamento consistindo em uma arquitetura de SoC, com um microprocessador de *software* implementando a estratégia do MPC e uma parte em *hardware* implementada em um FPGA *Virtex-IV* XC4VLX25-FF668-10C responsável por realizar cálculos matriciais de maior demanda computacional. Outro diferencial é a utilização de um sistema de numeração em ponto flutuante, no sistema LNS (sigla do inglês, *Logarithmic Number System*) com palavras de 16 *bits* para otimizar a implementação.

São apresentados dois sistemas diferentes: um linear (antena giratória) e um sistema não linear (regulador de glicose). Os resultados obtidos são calculados em tempos relativamente baixos (0.1s), e os sistemas são controlados adequadamente, com respeito às restrições. Os sistemas também são relativamente simples e com pequenos horizontes de predição ($N = 20$ e $N = 5$). Além disso, por mais que um dos sistemas apresentados seja não linear, trata-se de uma aplicação com período de amostragem muito alto (5 minutos), o que auxilia na implementação dos controladores em FPGA, diminuindo a quantidade de recursos de

hardware necessários.

O trabalho apresentado em Luo *et al.* (2011) também trabalha com o MPC aplicado ao modelo da aeronave Cessena Citation 500, com as mesmas características do trabalho anterior (N = 10 e tempo de amostragem igual a 0.5s). A principal inovação do trabalho é utilizar como *solver* do MPC um algoritmo de otimização bioinspirado, denominado *Particle Swarm Optimization* ou PSO. Com esse algoritmo, os autores conseguiram explorar o paralelismo do FPGA e do otimizador para controlar o sistema de maneira mais eficiente que o trabalho de Ling, Wu e Maciejowski (2008).

Neste trabalho, também foi utilizada uma ferramenta de conversão de código em MATLAB para uma linguagem de descrição de *hardware*: a ferramenta conhecida como *HDL Coder*. Entretanto, essa ferramenta é utilizada para gerar um algoritmo em VHDL do MPC, diferente da linguagem *Handel-C* adotada em Ling, Wu e Maciejowski (2008). O sistema foi testado por meio de uma interface serial (RS232), conectando um FGPA da família *Virtex-5* ao MATLAB para gerar as referências desejadas e enviadas ao sistema de controle MPC embarcado em FPGA.

O trabalho de Joss *et al.* (2012) é o primeiro dos trabalhos mostrados a trabalhar somente com sistemas não lineares, ou seja, trabalhar com o NMPC. Ainda que o foco deste trabalho seja a implementação do MPC e não do NMPC, convém a análise dessas publicações, pois elas contêm informações relevantes para analisar as contribuições deste trabalho.

O sistema estudado em Joss *et al.* (2012) é um VANT, assim como o sistema estudado neste trabalho. Entretanto, os autores optaram por trabalhar com o modelo não linear, enquanto neste trabalho, optou-se pela linearização do modelo. Além disso, o sistema estudado corresponde a um VANT de asa fixa com seis graus de liberdade, ou seja, os autores optaram por trabalhar com seis variáveis de estado: x, y, z, ϕ, θ e ψ, as coordenadas espaciais e angulares apresentadas na seção de modelagem do quadrirrotor (Seção 2.2.2). Isso difere do presente trabalho, no qual optou-se pela utilização do sistema linear completo com 12 estados (incluindo as variações espaciais e angulares $\dot{x}$,$\dot{y}$,$\dot{z}$,$\dot{\phi}$,$\dot{\theta}$ e $\dot{\psi}$) de um veículo quadrirrotor.

A quantidade de variáveis de controle também difere entre este trabalho e a publicação de Joos *et al.* (2012): o quadrirrotor estudado neste trabalho possui 4 variáveis de controle, relacionadas ao torque de cada um dos motores do sistema; já o trabalho de Joos *et al.* (2012) possui 2 variáveis de controle relacionadas à variação de *pitch* e *yaw* do VANT de asa fixa.

A grande contribuição apresentada em Joos *et al.* (2012) é a implementação do NMPC em um *kit* de FPGA modesto, uma *Xilinx Spartan 3E 1600*, com um tempo de amostragem de 0.5s, sendo capaz de calcular as variáveis de controle em aproximadamente 5.1ms. Entretanto, para alcançar esses números os autores utilizaram uma estratégia de controle auxiliar: um controlador LQR que controla as variáveis de estados e as envia para o NMPC, que deve então controlar somente as variações de *pitch* e *yaw* citadas anteriormente.

O controlador MPC foi validado por meio de uma técnica conhecida como *Hardware in the Loop* ou HIL, na qual uma parte do sistema é simulada (neste caso, o modelo do sistema é simulado em tempo real), comunicando-se com o controlador embarcado em FPGA. É uma técnica bastante utilizada na prática porque permite validar o sistema de controle desenvolvido sob condições mais próximas da realidade, sem a necessidade de operar o sistema real propriamente dito (ISERMANN; SCHAFFNIT; SINSEL, 1999).

O trabalho apresentado em Kaepernick *et al.* (2014) estuda dois modelos não lineares distintos: um guindaste, com 7 estados e 2 variáveis de controle; e um reator tanque agitado contínuo, com 4 estados e 1 variável de controle. Este artigo é mais voltado ao estudo dos efeitos da geração automática de código do MPC para programar FPGAs, utilizando tanto a ferramenta do MATLAB *HDL Coder* como a ferramenta da *Xilinx*, *Vivado HLS*, que converte um código em linguagem de programação *C* para linguagem de descrição de *hardware* (VHDL ou *Verilog*). Os sistemas estudados possuem tempos de amostragem altos comparados com a maioria das aplicações anteriores (1s e 1200s), mas são implementados em *hardware* com um horizonte de predição maior do que as aplicações anteriores ($N = 30$), o que é uma contriuição importante tendo em vista que um horizonte de predição maior acarreta um consumo também maior de recursos de *hardware* no FPGA. Foram utilizados dois *kits* diferentes: para o guindaste, um *kit* Zynq-7000 contendo um FPGA XC7Z020 CLG484-1 e para o tanque, um *kit* Artix-7 com FPGA XC7A200T-2FBG676C.

Uma das publicações que obteve melhores resultados em termos de implementação do NMPC em FGPA é o trabalho de Hartley *et al.* (2014). O sistema estudado neste trabalho é o mais complexo dentre todos os trabalhos vistos até aqui: um modelo de aeronave *Boeing* 747-200 com 12 estados. O algoritmo de *solver* de QP utilizado é do tipo método do ponto interior primal duplo ou PDIP (sigla da nomenclatura da técnica em inglês) e é capaz de implementar o NMPC em um FPGA com tempo de amostragem igual a 0.2s, e horizonte de predição igual $N = 5$ e $N = 12$, em dois testes distintos.

O sistema foi implementado com o uso da ferramenta MATLAB *HDL Coder*, para geração de código VHDL de forma automática. O trabalho conseguiu economizar recursos de *hardware* implementando o controlador NMPC utilizando aritmética de ponto fixo para armazenar os dados em um FPGA *Virtex-6* XC6VLX240T-1FFG1156. A validação experimental do controlador NMPC aqui também se dá por meio de uma simulação HIL.

Outro trabalho relevante corresponde a Xu *et al.* (2016). Nele, que é continuação de uma publicação anterior Xu *et al.* (2014), os autores utilizam o PSO como *solver* (assim como no trabalho de Luo *et al.* (2011)) para o NMPC, explorando o seu paralelismo junto com o paralelismo do *hardware* em um FPGA *Altera Stratix III*. O modelo estudado pelos autores é o de um sistema não linear de motor automotivo conhecido como *enDYNA*, composto por 2 estados e uma única variável de controle.

A grande contribuição do trabalho é a validação do sistema por meio de uma plataforma HIL construída para testar o sistema, simulando o modelo do motor em tempo real e comunicando o mesmo com o controlador NMPC embarcado em FPGA. O tempo de amostragem alcançado é bem rápido (10 ms), mas o horizonte de predição adotado pelos autores é pequeno ($N = 5$), o que auxilia na obtenção destes resultados.

Por fim, o trabalho de Ayala *et al.* (2016) também apresenta contribuições interessantes. Nele é realizado um estudo sobre a implementação do NMPC em FPGA para um sistema de manipulador robótico de *link* único (4 estados e 1 variável de controle) utilizando aritmética em ponto flutuante com precisão variável. Um dos resultados interessantes deste trabalho é que, a melhor relação entre precisão e erro obtido se dá com 27 *bits* para uma solução em FPGA utilizando ponto flutuante, para a aplicação estudada.

A solução do problema de otimização é dada por uma técnica de rede neural baseada em função radial (RBFNN, sigla do termo em inglês *Radial Basis Function Neural Network*. O sistema possui um período

de amostragem pequeno (10 ms) e um horizonte de predição N = 50, sendo este o maior horizonte de predição implementado em FPGA entre os trabalhos analisados. Foram utilizados dois *kits* distintos de FPGA: um *Xilinx Artix-7* e um *Altera Cyclone IV*, para comparação de consumo de recursos entre ambos os *kits*.

A análise de todas essas publicações mostra alguns pontos importantes que salientam as contribuições desejadas com este trabalho. Primeiramente, o sistema que serve como estudo de caso para este trabalho, o quadrirrotor, corresponde a um sistema de maior complexidade que a grande maioria dos trabalhos encontrados na literatura, com 12 estados e 4 variáveis de controle, o que torna as matrizes que descrevem o sistema em espaço de estados maiores e, consequentemente, as dimensões das matrizes que compõem o problema de controle do MPC maiores também.

Além disso, a maioria dos trabalhos utilizou algum tipo de geração automática de código para converter códigos previamente desenvolvidos do MPC ou do NMPC em linguagem de descrição de *hardware*. A abordagem mais comum se dá pela utilização do *HDL Coder* do MATLAB, convertendo um algoritmo do MATLAB em um algoritmo que pode ser embarcado em FPGA. Isso se dá por uma gama de motivos, sendo o mais comum a dificuldade e o tempo necessário para desenvolver manualmente um algoritmo em linguagem de descrição de *hardware*, por mais que existam ferramentas de EDA mais sofisticadas atualmente para tal fim.

Por fim, um outro diferencial deste trabalho consiste no fato de no mesmo ser utilizado aritmética de ponto flutuante para realizar as operações necessárias para a implementação do MPC. Com exceção, dos trabalhos de Vouzis *et al.* (2009) e Ayala *et al.* (2016), nenhum dos trabalhos analisados utiliza ponto flutuante, utilizando ponto fixo para economizar recursos de *hardware* na implementação do MPC/NMPC em FPGA. Esse tema é abordado com mais detalhes na próxima subseção.

2.3.3 Aritmética de ponto flutuante em FPGAs

Uma questão muito importante quando se trabalha com qualquer plataforma de *hardware* é a escolha de como serão representados os valores numéricos da aplicação desejada.

À medida que FPGAs foram se desenvolvendo e arquiteturas mais avançadas foram sendo desenvolvidas para as mesmas, tornou-se possível e viável implementar os dados tanto em ponto fixo quanto em ponto flutuante. Portanto, a escolha sobre o formato de representação de dados mais indicado depende da aplicação desejada. De maneira geral, implementações em ponto flutuante consomem mais recursos de *hardware* em um FPGA do que implementações em ponto fixo (URRIZA et al., 2010). Por outro lado, aplicações que requerem uma grande faixa dinâmica de valores são melhor desenvolvidas com representações em ponto flutuante, e não em ponto fixo (SALDANHA; LYSECKY, 2009).

A Figura 2.8 mostra como se dá a representação de dados em ponto fixo. Nessa representação, o número é expresso no formato <*w*,*nq*>, em que *w* corresponde à quantidade total de *bits* utilizados para representar esse número e seu valor em binário é dado por $x \times 2^{nq}$, sendo x o valor inteiro de *w* expresso na forma de complemento de 2. O número total de *bits* utilizados para representar a parte inteira do número é dado por n_i, enquanto n_q corresponde ao número de *bits* que representa a parte fracionária do número. Por fim, tém-se o *bit* mais significativo do número, dado por s (URRIZA et al., 2010).

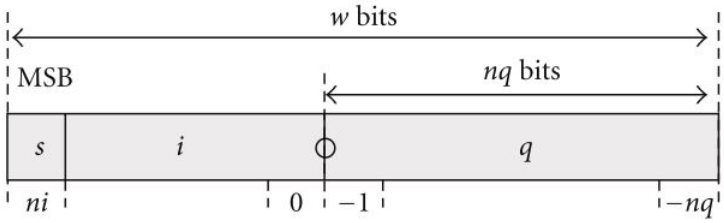

Figura 2.8: Representação numérica em ponto fixo. Adaptado de : (URRIZA et al., 2010).

A faixa dinâmica de valores totais para um número representado em ponto fixo é dada pela seguinte relação (Eq. 2.109).

$$[2^{ni}\ ,\ 2^{ni} - 2^{-nq}] \tag{2.109}$$

Já no caso de números em ponto flutuante, os dados são armazenados em um formato divido em três partes: *bit* de sinal (*s*), expoente (*e*) e mantissa (*f*), conforme mostrado na Figura 2.9.

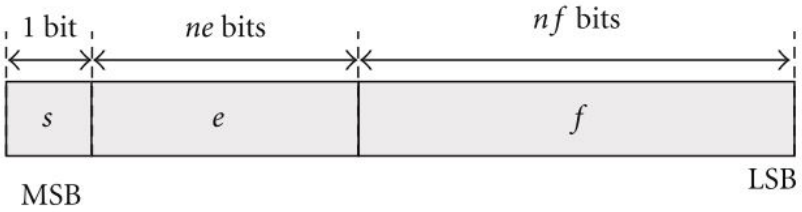

Figura 2.9: Representação numérica em ponto flutuante. Adaptado de : (URRIZA et al., 2010).

A primeira parte, *s*, indica se o número é positivo (*bit* igual a '0') ou negativo (*bit* igual a '1'). Um número em ponto flutuante é representado de acordo com a seguinte relação (Equação 2.110):

$$(-1)^s \times (f) \times 2^{2^{e-bias}}. \tag{2.110}$$

O expoente do número indica o quanto o número foi "deslocado"em relação ao termo de *bias* = $2^{ne-1} - 1$, em que n_e corresponde ao número de *bits* total da parte do expoente. O número em ponto flutuante é "normalizado", de forma de que o valor da mantissa é dado por $1 + f$.

A faixa de valores dinâmicos que um valor pode assumir em ponto flutuante é definida pela Equação 2.111 abaixo.

$$[2^{1-bias}\ ,\ (2 - 2^{-nf}) \times 2^{bias}]. \tag{2.111}$$

Comparando a faixa dinâmica de valores da representação em ponto flutuante (Equação 2.111) com a faixa dinâmica de valores da representação em ponto fixo (Equação 2.109), é possível perceber que a primeira é bem maior, ou seja, trabalhando com números em ponto flutuante é possível representar uma

variedade maior de números. Analisando as matrizes que descrevem o modelo do quadrirrotor, o estudo de caso deste trabalho, é possível notar que há uma faixa dinâmica razoável de valores, o que torna a representação em ponto flutuante mais apropriada para o controlador MPC desejado.

Além disso, o uso da representação em ponto flutuante facilita a utilização de modelos diferentes, em aplicações futuras: cada aplicação vai ter uma faixa dinâmica específica e, assim, torna-se mais fácil substituir o modelo se a representação em ponto flutuante estiver sendo utilizada. É de interesse do grupo de pesquisa com o qual este trabalho foi desenvolvido que a arquitetura do controlador MPC desenvolvida em FPGA seja aplicada a outros sistemas, o que torna a representação em ponto flutuante a opção mais interessante.

Uma representação em ponto fixo, em geral, consegue otimizar a utilização de recursos de *hardware* em um FPGA, mas em geral apresenta uma solução menos genérica: como a faixa dinâmica de valores pode mudar bastante de uma aplicação para a outra, a quantidade de *bits* para representar os dados também pode mudar de uma aplicação para a outra, podendo ser necessário realizar grandes modificações nos algoritmos desenvolvidos.

A maioria dos trabalhos analisados na subseção anterior, utilizam ponto fixo em seus controladores embarcados em FPGA, auxiliando a consumir menos recursos de *hardware*. A proposta de utilizar ponto flutuante neste trabalho, portanto, não só torna a solução mais desafiadora de ser implementada, mas também torna-a mais genérica e prova-se uma contribuição interessante para a literatura de controle preditivo embarcado em FPGAs.

Capítulo 3

Metodologia

Este capítulo apresenta a metodologia do trabalho, ou seja, as atividades que foram desenvolvidas de forma a cumprir os objetivos estabelecidos no Capítulo 1.

A divisão deste capítulo se dá de acordo com as diferentes etapas de atividades desenvolvidas:

1. Simulação do controlador MPC utilizando MATLAB;
2. Desenvolvimento de arquitetura utilizando a linguagem de descrição de *hardware* VHDL do controlador MPC;
3. Simulação da arquitetura em VHDL do controlador MPC;
4. Comparação das arquiteturas desenvolvidas em VHDL com os resultados obtidos nas simulações em MATLAB.

Todos os resultados referentes à cada subseção a seguir são apresentados no próximo capítulo deste trabalho.

3.1 Simulação do controlador MPC utilizando MATLAB

A primeira etapa necessária para desenvolver o MPC, antes de embarcar o mesmo em FPGA, corresponde à obtenção dos parâmetros do controlador, como a matriz de ponderação das saídas $\mathbf{Q_y}$, a matriz de ponderação das variáveis de controle $\mathbf{Q_u}$, número de iterações necessárias N_{iter} para que o *solver* de QP funcione adequadamente, entre outros. Além disso, esse tipo de simulação é importante pois permite estudar a estabilidade do sistema mediante a atuação do controlador MPC.

Essa é uma tarefa que exige um ambiente de simulação, que permita com que vários testes sejam efetuados e, com a análise dos resultados destes testes, escolher parâmetros que melhor cumpram os objetivos de controle adotados.

O ambiente escolhido neste trabalho é o *software* MATLAB 2016b. O uso do MATLAB se mostrou uma escolha vantajosa por conta de diversos fatores. Primeiramente, por se tratar de um *software* que

possibilita trabalhar com matrizes de maneira simples e intuitiva. Conforme visto no Capítulo de Revisão Bibliográfica, a abordagem linear do MPC apresentada realiza várias operações matriciais e, portanto, essa é uma característica importante para o ambiente de simulação adotado.

Além disso, a bibliografia principal utilizada neste trabalho para fundamentação teórica do MPC (ALAMIR, 2013), apresenta soluções já implementadas em MATLAB, que podem ser adaptadas para atender às características do sistema que deseja-se controlar. O próprio *solver* de QP utilizado neste trabalho, o algoritmo PGE explicado na Subseção 2.1.5, encontra-se disponível em Alamir (2013) já no formato de um *script* desenvolvido em MATLAB, bem como os algoritmos de cálculo das matrizes de custo e de restrição do MPC (Subseção 2.1.3) e o algoritmo de parametrização exponencial (Subseção 2.1.6).

Para realizar as simulações em MATLAB, primeiramente é necessário definir os modelos que serão utilizados. Neste trabalho, o sistema que deseja-se controlar é o sistema do veículo quadrirrotor definido na Seção 2.2. Com o sistema definido, devem ser escolhidas os tipos de controladores que serão utilizados para controlar o sistema. Neste trabalho, são utilizados tanto o controlador MPC sem restrições quanto a formulação completa do controlador MPC, que utiliza restrições do problema de controle estabelecido.

Como a lei de controle do MPC sem as restrições é bem mais simples, este é utilizado na primeira etapa de validação, uma vez que torna-se mais fácil avaliar os módulos de operação matriciais, verificando se os mesmos estão funcionando de maneira adequada e se os resultados obtidos estão corretos.

Com a definição dos controladores, é necessário definir quais os sinais de referência que serão aplicados ao sistema durante os testes, de modo a validar os controladores desenvolvidos. Para o quadrirrotor, as referências definidas foram baseadas nos testes apresentados em Lopes *et al.* (2011). Assim, são propostos dois cenários de teste distintos:

1. Aplicação de sinais de degrau filtrado como referência de posição nos eixos x, y e z, com posições finais em $x = 10m$, $y = 10m$ e $z = 10m$. Além disso, uma referência adicional é aplicada ao ângulo ψ, dada por $\psi = 0°$. O valor inicial de cada variável de estado é dado pelo vetor $x_0 = [x\ \dot{x}\ y\ \dot{y}\ z\ \dot{z}\ \theta\ \dot{\theta}\ \phi\ \dot{\phi}\ \psi\ \dot{\psi}]^T = [0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0 - 4\ \ 0 - 4\ \ 0\ \ 5\ \ 0]^T$;
2. Uma trajetória helicoidal, definida pelas referências $x_r(t) = 2cos(2t)$, $y_r(t) = 2sin(2t)$ e $z_r(t) = 0.2t$. A referência aplicada a ψ é a mesma do caso anterior, $\psi = 0°$ e o valor inicial das variáveis de estado é dado pelo vetor $x_0 = [0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0 - 4\ \ 0 - 4\ \ 0\ \ 0\ \ 0]^T$

O objetivo desses testes é avaliar se o MPC é capaz de estabilizar o quadrirrotor, fazendo com que as saídas reguladas do mesmo (x, y, z e ψ) assumam os valores de suas respectivas referências, ao mesmo tempo que as restrições impostas para as variáveis ϕ e θ sejam satisfeitas, impedindo que o quadrirrotor saia de sua região de linearização. O primeiro teste emula mudanças que são aplicadas a cada eixo de coordenada espacial do quadrirrotor, como um usuário que pilota o quadrirrotor por meio de um *joystick* e aplica comandos em cada direção dos eixos x, y e z. O desafio para o controlador nesse cenário é que as referências nos eixos x, y e z são aplicadas ao mesmo tempo.

Já no caso da trajetória helicodial, trata-se de um cenário de trajetória mais complexo, no qual a referência está constantemente mudando de valor, o que permite um bom estudo da estabilidade do quadrirrotor mediante a atuação do MPC.

Em cada um desses cenários, o conjunto de restrições adotados é o mesmo: as restrições nas saídas restringidas (ângulos θ e ϕ) são $y_c^{min} = -5^\circ$ e $y_c^{max} = 5^\circ$ e as restrições nas variáveis de controle são $u_{min} = 0$ rad/s e $u_{max} = 385.6$ rad/s.

Com a definição dos controladores, a próxima atividade necessária para realizar os testes do MPC em MATLAB corresponde à adaptação dos algoritmos apresentados em Alamir (2013) para o modelo do quadrirrotor. Isso corresponde à utilização das matrizes que descrevem esse sistema em espaço de estados, as matrizes $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C_r}$ e $\mathbf{C_c}$ definidas nas Equações 2.105 - 2.108.

Com as matrizes que descrevem ambos os sistemas em espaço de estado definidas, é possível calcular as matrizes de custo $\mathbf{F1}$, $\mathbf{F2}$, $\mathbf{F3}$, $\mathbf{H}$, $\mathbf{Fr}$ e $\mathbf{Br}$; e as matrizes de restrição $\mathbf{G1}$, $\mathbf{G2}$, $\mathbf{G3}$, $\mathbf{A_{ineq}}$ e $\mathbf{B_{ineq}}$, conforme os procedimentos apresentados na Subseção 2.1.3. Esse procedimento é efetuado pelo *script compute_ MPC_ matrices.m*, disponível em Alamir (2013).

No caso da formulação do MPC sem restrições, com base nas matrizes obtidas na atividade anterior, é possível calcular a lei de controle e realizar as simulações. Já para o caso da formulação completa do MPC, que incorpora as restrições de cada sistema, ainda é necessário realizar algumas atividades.

A primeira das atividades restantes para implementação da formulação completa do MPC corresponde à utilização da parametrização exponencial do MPC, conforme os procedimentos explicados na Subseção 2.1.6. Isso é feito por meio de uma adaptação do *script compute _Pi_ par.m* em Alamir (2013), que calcula a matriz de parametrização exponencial $\mathbf{Pi_e}$. Para essa etapa, mais alguns parâmetros devem ser determinados. Os parâmetros utilizados neste trabalho partem dos valores pré-definidos no capítulo 6 de Alamir (2013):

$$\tau_r = \begin{bmatrix} 0.1 \\ 0.1 \\ 0.1 \\ 0.1 \end{bmatrix};\ n_e = \begin{bmatrix} 2.0 \\ 2.0 \\ 2.0 \\ 2.0 \end{bmatrix};\ \alpha = 10.0. \tag{3.1}$$

Com todos esses parâmetros, pode-se obter as matrizes de custo e restrição parametrizadas $\mathbf{F_r}$ e $\mathbf{B_r}$ e enfim aplicá-las ao *solver* de QP que calcula as variáveis de controle do sistema. Esse *solver* está contido no *script* solver_ sat_ pen.m de (ALAMIR, 2013), que implementa o algoritmo PGE descrito na subseção 2.1.5.

Todas essas etapas foram incluidas em um único *script*, para automatizar a simulação do sistema do quadrirrotor sendo controlado pelo MPC em MATLAB, tanto nos cenários aplicados ao MPC sem restrições quanto para o MPC com restrições.

Após a conclusão de todas essas atividades, é possível realizar simulações para obter os parâmetros de controle que melhor cumprem o objetivo do problema proposto e observar qual o menor horizonte de predição que deve ser utilizado para estabilizar o sistema do quadrirrotor, bem como o número total de iterações necessárias para que o *solver* PGE encontre uma solução de controle adequada aos cenários propostos. Isso é de suma importância para embarcar o MPC em *hardware*, uma vez que o tamanho do horizonte de predição utilizado impacta diretamente no tamanho das matrizes de custo e restrição do MPC, e consequentemente, aumenta o custo de recursos de *hardware* necessários para sua implementação em

FPGA. Já o número de iterações do *solver*, no caso do MPC com restrições, impacta no tempo de cálculo das variáveis de controle, que deve respeitar o tempo de amostragem do sistema (50 ms para o quadrirrotor).

Para tal, as simulações serão realizadas primeiramente variando o horizonte de predição *N*. No caso do MPC com restrições, mantem-se um valor elevado de iterações do *solver* de QP, N_{iter} = 100 constante, para garantir que a influência de cada horizonte de predição possa ser analisada de maneira apropriada. Isso não ocorre no MPC sem restrições, porque o mesmo não tem como parâmetro o número de iterações do *solver* de QP, já que não utiliza um *solver* de QP.

Assim, é possível observar a partir de qual horizonte de predição, o quadrirrotor estabiliza e começa a ser controlado de maneira satisfatória. O menor valor de horizonte de predição que satisfaça essas condições, é considerado o melhor valor para ser utilizado no controlador desenvolvido em VHDL, uma vez que resultaria nas matrizes de menores dimensões mas que ainda assim são capazes de controlar o quadrirrotor.

Levantados quais os melhores valores de *N* para implementação do MPC em FPGA, são realizadas simulações variando o número de iterações do *solver* de QP. Quanto menos iterações foram necessárias para cumprir os objetivos de controle, menos tempo é gasto para calcular as variáveis de controle ótimas do sistema. Assim, busca-se o menor horizonte de predição *N* e o menor de iterações necessários para controlar o quadrirrotor, e estes serão os valores utilizados nas simulações do controlador desenvolvido em *hardware* para validação dos mesmos.

Uma vez que esses testes são concluídos, têm-se todos os parâmetros necessários para definir o controlador MPC e, portanto, pode-se iniciar o desenvolvimento de sua arquitetura em *hardware* para que o mesmo seja implementado em FPGA.

3.2 Desenvolvimento de arquitetura em VHDL do controlador MPC aplicado ao quadrirrotor

Uma vez que os parâmetros do controlador MPC foram obtidos, e tem-se o código do mesmo disponível em forma de *script* em MATLAB, é possível desenvolver o controlador MPC em VHDL.

Com base nos cenários que são descritos na seção anterior, é possível deduzir que é preciso desenvolver duas arquiteturas de controladores distintas: uma para a lei de controle do MPC sem levar em conta as restrições do problema de controle, e uma arquitetura para a formulação completa do MPC, que leva em consideração as restrições do sistema sendo controlado.

Como cada sistema que vai ser controlado possui dimensões distintas, de acordo com as dimensões das matrizes que representam esse sistema em espaço de estados e o horizonte de predição utilizado, é interessante que os controladores desenvolvidos sejam capazes de adaptar-se às dimensões das matrizes que compõem o problema de controle adotado. Isso é particularmente difícil de se realizar em *hardware*, pois uma vez que o *hardware* foi definido, o mesmo mantem suas características durante sua execução, com exceção de técnicas como reconfiguração dinâmica.

Para facilitar essa questão, a solução adotada foi de desenvolver uma arquitetura em VHDL que fosse gerada por meio de *scripts* em MATLAB. Assim, para cada sistema que deseja-se controlar, com o hori-

zonte de predição desejado, o *script* obtém as informações das matrizes do sistema e gera todos os módulos necessários em VHDL, parametrizados adequadamente para implementar o MPC, seja com ou sem restrições.

É importante notar que apesar de os códigos serem gerados por meio de *scripts*, a lógica dos mesmos foi desenvolvida manualmente, e é parametrizada pelo *script* MATLAB, também desenvolvido manualmente, para se adequar ao problema de controle que é proposto. Isso não constitui geração automática de código em VHDL da maneira que costuma ser encontrada na literatura, uma vez que esses trabalhos, de maneira geral, utilizam ferramentas específicamente para gerar a lógica dos algoritmos em linguagem de descrição de *hardware* e não somente parametrizá-los, como é feito neste trabalho. Portanto, pode-se considerar que esse tipo de abordagem é mais uma contribuição deste trabalho, uma vez que explora uma abordagem de desenvolvimento de algoritmos em VHDL pouco explorada na literatura.

Cada subseção a seguir é responsável por explicar como cada componente dos controladores foi implementado, começando pelos módulos de cálculo matricial (blocos básicos dos demais módulos), armazenamento de matrizes e a arquitetura propriamente dita de cada controlador. Após isso, as ferramentas de validação que foram utilizadas para obter os resultados deste trabalho são apresentadas, explicando as etapas de funcionamento das mesmas.

3.2.1 Módulos de cálculo matricial em ponto flutuante

Analisando os *scripts* em MATLAB e as formulações do MPC sem restrições e com restrições, apresentadas na Equação 2.34 e no algoritmo PGE, é possível perceber que a maior parte das operações efetuadas correspondem a operações matriciais, em sua grande maioria compostas por somas e multiplicações. Portanto, cada um dos módulos que compõem o MPC (com ou sem restrições) deve realizar operações matriciais, de forma que torna-se necessário desenvolver unidades de cálculo de operação matricial.

O ponto de partida para o desenvolvimento desses módulos foi através da utilização de módulos de cálculo em ponto flutuante já existentes, desenvolvidos no trabalho de Muñoz (2012). Dentre a biblioteca de unidades de aritmética em ponto flutuante apresentadas em Muñoz (2012), utilizou-se neste trabalho as unidades de soma/subtração em ponto flutuante e de multiplicação em ponto flutuante, que por sua vez foram utilizados para implementar unidades de soma/subtração matricial e unidades de multiplicação matricial, conforme descrito nas subseções a seguir.

É importante relembrar que os módulos de cálculo em ponto flutuante possuem tamanho variável, ou seja, é possível selecionar qual a quantidade de *bits* com que deseja-se utilizar. Neste trabalho, são apresentados resultados utilizando dois tamanhos de dados diferentes para que seja possível comparar os resultados obtidos com cada um: 27 *bits*, ou seja, 1 *bit* de sinal, 8 *bits* de expoente e 18 *bits* de mantissa e 32 *bits* (1 *bit* de sinal, 8 *bits* de expoente e 23 *bits* de mantissa). Dados em 27 *bits* possuem melhor relação entre precisão e consumo de recursos de *hardware* nos FPGAs da *Xilinx* (AYALA et al., 2016), porque as entradas dos DSPs dos *kits* dessa empresa costumam ser de 18 *bits*. Já dados de 32 *bits*, apesar de consumirem uma quantidade maior de recursos de *hardware*, apresentam um menor erro de arredondamento e são capazes de armazenar uma faixa de valores ainda maior.

3.2.1.1 Unidade de soma/subtração em ponto flutuante

A unidade de soma/subtração em ponto flutuante, denominada *FPadd*, apresentada em Muñoz (2012) é capaz de realizar somas ou substrações com dois operandos em ponto flutuante, de precisão variável, de acordo com o *bit* presente na entrada *op* do módulo: caso *op* = '0', é realizada uma adição; caso *bit* = '1', realiza-se uma subtração.

As etapas de operação do módulo, para realizar a operação artimética de soma/subtração correspondem a (MUÑOZ, 2012):

1. Separação de sinal, expoente e mantissa dos operandos de entrada;
2. Análise das entradas: conferir se as mesmas correspondem ao valor zero, infinito ou uma representação inválida no padrão IEEE-754;
3. Em caso de operandos válidos, adiciona-se o *bit* implícito '1' às mantissas;
4. Comparação dos expoentes e mantissas dos dois operandos;
5. Deslocamento à direita do menor dos números em *n bits*, em que *n* corresponde ao resultado da subtração dos expoentes (valor obtido na etapa anterior);
6. Somar ou subtrair as mantissas de acordo com a operação desejada (valor do *bit* de entrada *op*);
7. Deslocamento à esquerda do resultado atual da mantissa até que seja encontrado o valor de '1' no *bit* mais significativo (MSB);
8. Subtração de '1' no valor atual do expoente para cada *bit* deslocado na etapa anterior;
9. Concatenação dos resultados de sinal, expoente e mantissa.

O tempo total de operação do módulo é de dois ciclos de *clock*. No primeiro ciclo de *clock* são avaliadas possíveis exceções nos operandos, comparam-se os expoentes e mantissas, calcula-se o sinal do resultado, o resultado preliminar do expoente e o alinhamento das mantissas segundo a diferença dos expoentes dos valores de entrada. Já durante o segundo ciclo de *clock*, a mantissa do resultado é normalizada, de forma que esse mesmo valor é concatenado com o resultado do expoente e do sinal. O processo de atualização é controlado por uma máquina de estados finitos (do inglês, *Finite State Machine* ou FSM) (MUÑOZ, 2012).

3.2.1.2 Unidade de multiplicação em ponto flutuante

A unidade de multiplicação em ponto flutuante, *FPmul*, também é capaz de realizar operações aritméticas com operandos em ponto flutuante, de precisão variável, em dois ciclos de *clock*. No caso deste módulo, realiza-se a multiplicação, cujas etapas de operação são (MUÑOZ, 2012):

1. Separação de sinal, expoente e mantissa dos operandos de entrada;

2. Análise das entradas: conferir se as mesmas correspondem ao valor zero, infinito ou uma representação inválida no padrão IEEE-754;

3. Em caso de operandos válidos, adiciona-se o *bit* implícito '1' às mantissas;

4. Multiplicação das mantissas;

5. Adição dos expoentes e subtração do valor de *bias* ao resultado;

6. Determinação do sinal do resultado da multiplicação;

7. Caso o MSB do resultado da mantissa seja igual a '1', não é necessário realizar nenhuma etapa de normalização. Caso contrário, desloca-se à esquerda o resultado atual da mantissa até encontrar o valor de '1' no MSB e, para cada *bit* deslocado, subtrai-se '1' no valor do expoente;

8. Concatenação dos resultados de sinal, expoente e mantissa.

Durante a primeira etapa de funcionamento do módulo, ou seja, durante o primeiro ciclo de *clock*, avalia-se se algum dos operandos é zero ou infinito. Em paralelo, calcula-se o sinal do resultado, o resultado preliminar do expoente e a multiplicação das mantissas. Por meio de uma operação lógica XOR compara-se o sinal dos valores de entrada e determina-se o sinal do resultado. Em seguida, realiza-se uma adição de *bit* extra indicando condição de *overflow* aos expoentes, o que mostra o motivo pelo qual o valor de *bias* deve ser subtraído do resultado da soma dos expoentes. Já as mantissas, de comprimento de M *bits*, são multiplicadas, resultando em uma palavra com de $2(M+1)$ *bits*, que é truncada nos primeiros $M+2$ *bits* (MUÑOZ, 2012).

Na segunda etapa (segundo ciclo de *clock*), avalia-se o MSB do resultado da multiplicação das mantissas, que por sua vez controla um multiplexador que permite selecionar entre os M *bits* mais significativos do valor atual da mantissa (para o caso em que o MSB da mantissa é igual a '1' e a mesma já encontra-se totalmente normalizada) ou o valor da mantissa deslocado à esquerda em uma posição. No caso de o MSB do resultado da multiplicação das mantissas ser igual a '1', o resultado preliminar do expoente é incrementado em '1'. Há uma avaliação da existência ou não de *overflow* no resultado do expoente ou de operandos de entrada iguais a infinito, o que acarreta em um produto igual a infinito. Caso as avaliações não detectem nenhum desses dois cenários, concatenam-se os valores de sinal, expoente e mantissa. Assim como no módulo *FPadd*, uma FSM controla o processo de atualização do resultado final do módulo *FPmul*.

3.2.1.3 Unidade de multiplicação matricial em ponto flutuante

A operação de multiplicação matricial envolve duas etapas: multiplicação e soma. Os elementos correspondentes entre a linha da primeira matriz e a coluna da segunda matriz são multiplicados e depois somados, gerando o elemento da matriz resultante. A esse tipo de operação geralmente denomina-se acumulação, e a matriz resultante da operação de multiplicação entre duas matrizes pode apresentar dimensões diferentes do que as dimensões das matrizes sendo multiplicadas: as dimensões de uma matriz obtida mediante a multiplicação de duas matrizes são dadas pelo número de linhas da primeira matriz e o número de colunas da segunda matriz.

Levando em consideração essas informações, o módulo de multiplicação matricial foi desenvolvido buscando aproveitar o paralelismo da operação de acumulação, obtendo uma resposta mais rápida. Assim, é interessante que o módulo seja capaz de calcular o máximo possível de elementos por ciclo de operação, e esse objetivo só pode ser satisfeito escolhendo um número adequado de somadores e multiplicadores em paralelo, ou em *pipeline*.

O critério utilizado para essa escolha foi obtido analisando a dimensão das matrizes que estão envolvidas no cálculo do MPC. Como a maioria das dimensões dessas matrizes corresponde a um múltiplo do horizonte de predição *N* adotado, optou-se por utilizar *N* multiplicadores em paralelo, multiplicando os elementos correspondentes das matrizes envolvidas, e *N* somadores que são arranjados de forma a acumular todos os valores multiplicados.

Esse módulo de multiplicação matricial foi desenvolvido com dois modos de operação distintos, controlados por uma entrada denominada *mode*. O primeiro modo de operação, selecionado quando a entrada *mode* = '1', corresponde ao caso em que o número de elementos da primeira matriz sendo multiplicada é maior que *N*. Nesse caso, o resultado do módulo é armazenado, para que seja incluído no próximo ciclo de operação e somado ao resultado acumulado, até que o cálculo do novo elemento seja concluído. É o que ocorre durante a multiplicação envolvendo a matriz de custos $F2$ e o vetor de referências y_{ref}, por exemplo, no cálculo da matriz de restrições F.

Para esse cenário, o módulo deve utilizar todos os multiplicadores e somadores disponíveis, dispostos conforme mostrado na Figura 3.1.

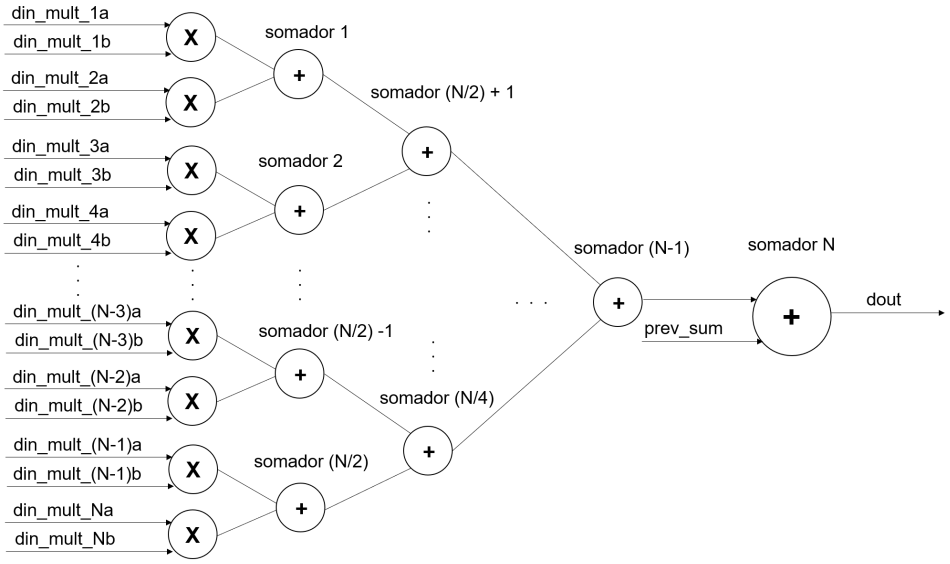

Figura 3.1: Arquitetura do módulo multiplicador de matrizes em ponto flutuante no modo de operação *mode* = '1'.

Assim, os elementos correspondentes de cada matriz, representados pelas entradas *din_ mult* na Figura 3.1 são distribuidos em *N* multiplicadores em paralelo, e o resultado das multiplicações vai sendo enviado para somadores que irão somar todos os resultados de multiplicações. Nesse modo de operação, o valor da soma de todas essas multiplicações é armazenado no sinal *prev _ sum*, que é inserido no último somador da arquitetura para que o valor acumulado da multiplicação possa ser utilizado nos próximos ciclos de

operação do módulo até que um elemento da matriz resultante seja calculado.

Neste modo de operação, a saída é atribuida ao pino *dout*, que armazena um único valor em ponto flutuante, de tamanho igual à precisão utilizada em ponto flutuante (27 ou 32 *bits*).

Já o segundo modo de operação, quando *mode* = '0', aproveita uma característica do cenário em que o número de elementos da coluna da primeira matriz sendo multiplicada é menor ou igual a *N*. Nestes casos, sempre que o sistema terminar seu ciclo de operação, ele obterá um elemento da matriz resultante, sem precisar armazenar o resultado no sinal *prev_ sum* como no modo de operação explicado anteriormente. Isso porque se o número de elementos na coluna da primeira matriz é menor que *N*, é possível atribuir todos esses elementos a multiplicadores em paralelo, assim como os elementos da linha correspondente na segunda matriz e realizar a operação de multiplicação matricial em um único ciclo de operação.

Por conta dessa caraterística, o modo de operação *mode* = '0' é mais rápido e é utilizado para multiplicar matrizes menores, permitindo até mesmo o cálculo de multiplicação entre matrizes de forma paralela, como no caso das multiplicações envolvendo $F1$ e o vetor de estados x, $G1$ e novamente o vetor de estados x e $G2$ com o vetor de variáveis de controle u no cálculo da matriz de custos F, por exemplo, que são processadas ao mesmo tempo.

A arquitetura desse modo de operação pode ser vista na Figura 3.2.

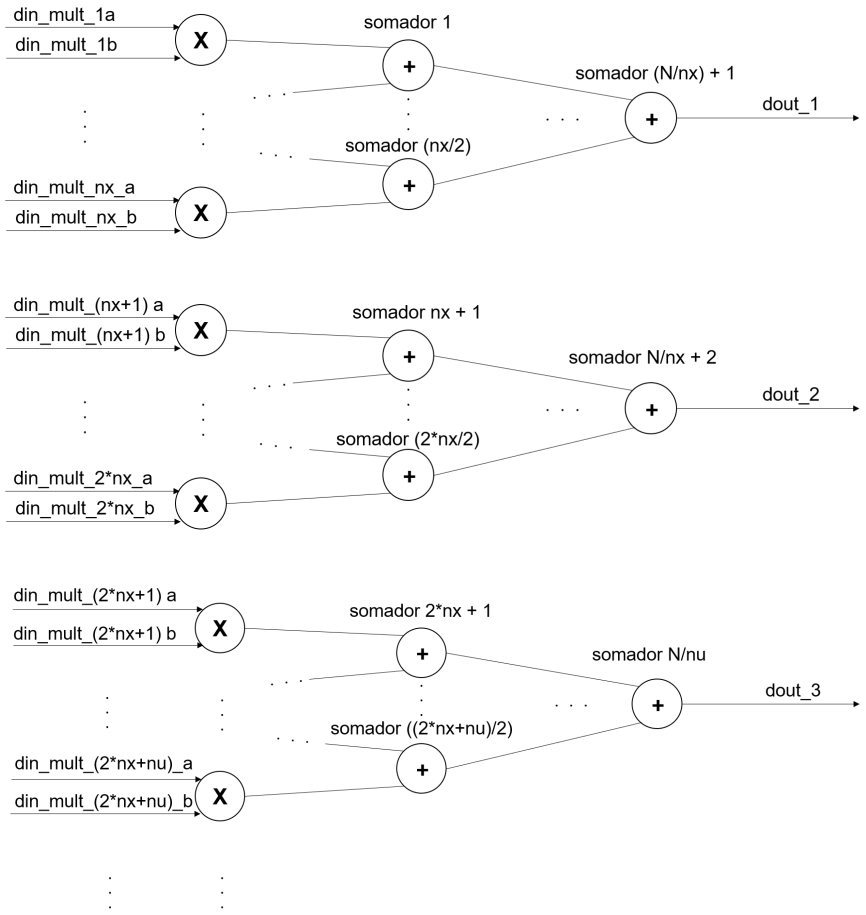

Figura 3.2: Arquitetura do módulo multiplicador de matrizes em ponto flutuante no modo de operação *mode* = '0'.

A Figura 3.2 mostra que durante o modo de operação *mode* = '0', a arquitetura arranja os multiplicadores e somadores em três regiões distintas. As duas primeiras regiões são capazes de calcular a multiplicação de matrizes onde a primeira matriz possui n_x elementos em suas colunas, em que n_x é o número de estados do sistema sendo controlado. Na última região, o número de elementos na coluna da primeira matriz sendo multiplicada deve ser igual a n_u, ou seja, o número de variáveis de controle do sistema. Com isso, é possível calcular elementos resultantes de até 3 multiplicações matriciais distintas, sendo que cada novo elemento é armazenado em uma das saídas $dout_1$, $dout_2$ ou $dout_3$, cada uma com tamanho de 27 ou 32 *bits*, mais uma vez de acordo com a precisão em ponto flutuante adotada.

Para o caso do quadrirrotor, $n_x = 12$ e $n_u = 4$.

O módulo de multiplicação matricial em ponto flutuante é um dos componentes fundamentais das arquiteturas de ambos os controladores, seja ele o MPC sem restrições ou o MPC com restrições. Todos os módulos contidos nesses controladores realizam em algum ponto multiplicação matricial, inclusive o módulo *model _simulator* que realiza a simulação dos modelos sendo controlados.

Cada módulo que realiza multiplicações matriciais foi desenvolvido para controlar qual o modo de operação mais adequado para as multiplicações matriciais que serão realizadas por meio de uma máquina de estados, garantindo o funcionamento correto dos controladores desenvolvidos.

O módulo de multiplicação matricial foi testado individualmente, comparando os resultados das operações matriciais com multiplicações matriciais previamente realizadas computacionalmente em MATLAB, em seus dois modos de operação. Somente após o correto funcionamento desse módulo em seus dois modos de operação, o mesmo foi utilizado na arquitetura dos controladores MPC (com ou sem restrições).

Outro módulo também muito importante, utilizado por vários outros módulos nas arquiteturas dos controladores é o módulo de soma/subtração matricial, que é apresentado na subseção seguinte.

3.2.1.4 Unidade de soma/subtração matricial em ponto flutuante

A operação de soma e subtração matricial é mais simples do que a operação de multiplicação matricial. Isso porque somar ou subtrair elementos de duas matrizes corresponde a realizar essa operação nos elementos correspondentes das matrizes que deseja-se somar/subtrair e a matriz resultante é do mesmo tamanho que as matrizes utilizadas para calculá-la. Portanto, a unidade de soma/subtração matricial utiliza unidades de soma/subtração em ponto flutuante em paralelo para calcular os elementos da matriz resultante também de forma paralela.

O funcionamento deste módulo desenvolvido utiliza o mesmo padrão do módulo de soma/subtração em ponto flutuante: através de uma entrada denominada *op*, o módulo realizará a soma (quando *op* = '0') ou a subtração (quando *op* = '1') de duas matrizes.

Assim como nas unidades de multiplicação matricial, o número de unidades de soma/subtração em ponto flutuante que são utilizadas em paralelo é um valor que depende da aplicação, e esse número foi definido como sendo o horizonte de predição *N* escolhido para o sistema sendo controlado, pelas mesmas razões já apresentadas na subseção anterior.

A arquitetura desse módulo é representada na Figura 3.3.

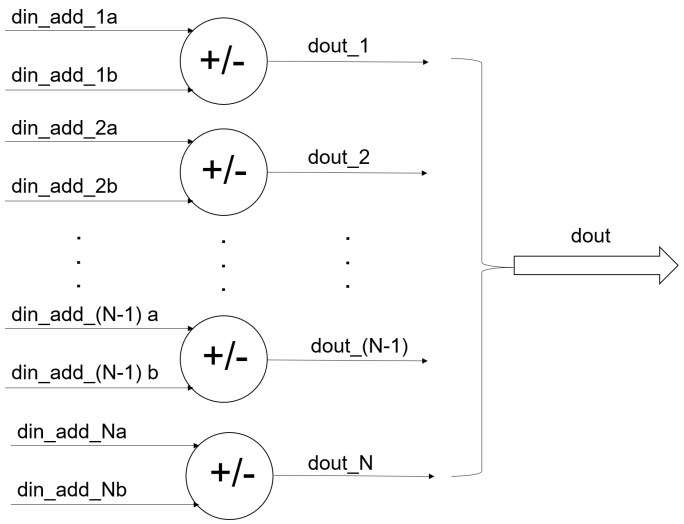

Figura 3.3: Arquitetura do módulo de soma/subtração de matrizes em ponto flutuante.

Assim, como as matrizes envolvidas nas operações do MPC possuem dimensões que são, em sua maioria, múltiplos de *N*, é possível realizar operações de soma/subtração matricial com *N* elementos das matrizes por ciclo de operação. Na Figura 3.3, os elementos que devem ser somados em cada matriz são representados pelas entradas *din_ add*. O resultado dessa soma/subtração, é atribuido às *N* saídas *dout_ i*, e são agrupados em um vetor único contendo todos esses resultados, denominado simplesmente *dout*. Dessa forma, é possível agrupar os resultados de somas/subtrações matriciais no formato adotado para armazenamentos em memórias RAM, que será explicado com mais detalhes na próxima subseção.

No caso deste módulo, cada ciclo de operação demora 2 ciclos de *clock*, que é a quantidade de ciclos exigida pelos módulos de soma/subtração em ponto flutuante para obter o resultado.

Em geral, na formulação adotada para calcular as matrizes de custo e restrições do MPC, bem como as operações matriciais utilizadas no *solver* PGE, as somas/subtrações matriciais são realizadas após multiplicações matriciais. Na maioria das vezes, isso implica que as dimensões dessas matrizes são reduzidas quando comparadas com as matrizes que são multiplicadas, o que faz com que essa escolha de *N* elementos de soma/subtração em paralelo seja satisfatória.

Na grande maioria dos casos, isso significa que a cada ciclo de operação do módulo é possível calcular uma linha da matriz, já que as dimensões das matrizes sendo somadas/subtraidas são múltiplos de *N*. No caso de matrizes menores, nem sempre é necessário utilizar todos os somadores para obter o mesmo resultado.

Como cada ciclo de operação do módulo demora 2 ciclos de *clock* para terminar, e no final, obtém-se uma linha de elementos da matriz resultante, é possível perceber que esse módulo em geral demora uma quantidade correspondente ao número de linhas da matriz resultante vezes 2 ciclos de *clock* para completar a soma/subtração da matriz.

Mais uma vez, é importante salientar que cada aplicação possui matrizes com dimensões diferentes, de acordo com as dimensões do modelo que deseja-se controlar e do horizonte de predição escolhido. Por isso, os *scripts* desenvolvidos em MATLAB são responsáveis por gerar a parametrização desse módulo, de

forma que os demais módulos possam interagir de maneira correta entre si, por meio do controle do início e do fim dos elementos que são somados/subtraidos em uma matriz através de uma FSM.

Assim, como no caso do módulo de multiplicação matricial, este módulo foi testado individualmente, comparando sua resposta com somas matriciais realizadas computacionalmente em MATLAB. Somente após seu correto funcionamento, o mesmo foi utilizado na arquitetura dos controladores MPC (sem e com restrições).

3.2.2 Armazenamento de matrizes do MPC em VHDL

Com todos os módulos definidos, o último detalhe importante a ser abordado para a implementação do MPC em VHDL é a maneira escolhida para armazenar as matrizes necessárias para realizar todas as operações demandadas pelos controladores desenvolvidos.

Algumas matrizes são definidas previamente de maneira *offline*, ou seja, são calculadas previamente e já estão disponíveis no início do funcionamento do controlador. É o caso das matrizes $\mathbf{F1}$, $\mathbf{F2}$, $\mathbf{F3}$, $\mathbf{G1}$, $\mathbf{G2}$, $\mathbf{G3}$, $\mathbf{A_r}$ e $\mathbf{H_r}$, definidas na seção 2.1.3. Outras informações importantes que estão disponíveis *offline* são as referências do sistema e o valor desejado para as variáveis de controle, y_{ref} e u_d.

As matrizes *offline* são implementadas como memórias ROM, ou seja, memórias sob as quais só podem ser executar operações de leitura. Assim, quando é necessário obter informações referentes a essas matrizes, os módulos que estiverem operando neste momento acessam as informações por meio dos endereços das memórias ROM.

Essas memórias ROM são geradas por meio do *script* em MATLAB que gera os demais módulos parametrizados já apresentados. Os elementos das matrizes são transformados em binário, seguindo o padrão de ponto flutuante adotado, e agrupados de acordo com o horizonte de predição adotado. Assim, é possível ler *N* elementos das matrizes em um único ciclo de leitura, o que permite as operações de multiplicação e soma matricial decorram de maneira apropriada e mais rápida.

Entretanto, existem matrizes que são calculadas pelo controlador MPC, de maneira *online*, ou seja, durante a execução do mesmo. Essas matrizes precisam ser armazenadas para serem lidas posteriormente, quando for necessário realizar novas operações com as mesmas. É o caso das matrizes intermediárias contidas no módulo *solver_ sat_ pen*, o *solver* de QP que minimiza e função custo do problema de controle definido e encontra as variáveis de controle ótimas do mesmo.

Para esse tipo de matriz, é necessário implementar memórias RAM, ou seja, sob as quais podem ser realizadas tanto operações de escrita quanto de leitura. Assim, as matrizes são calculadas e armazenadas em determinado endereço da memória RAM. Quando for necessário ler os elementos dessa matriz calculada anteriormente, para realizar novas operações matriciais, é possível acessá-los por meio deste mesmo endereço.

Assim como nas memórias ROM, as memórias RAM são geradas por meio de um *script* MATLAB. As mesmas são inicializadas com '0's já que elas somente receberão valores para serem armazenados quando algumas operações matriciais terminarem. A única exceção é a memória RAM que armazena a matriz parametrizada de restrições B_r, que possui alguns elementos que são calculados *offline* e já armazenados

nas suas posições correspondentes na memória RAM.

Todas as matrizes são do tipo *single port*, ou seja, há somente uma saída para realizar operações de leitura nas memórias ROM e uma entrada e uma saída nas memórias RAM, para realização de operações de escrita e leitura, respectivamente. Consequentemente, em cada um dessas memórias, também só uma entrada referente ao endereço de memória sobre o qual deseja-se realizar alguma operação. Cada uma dessas portas possui largura e profundidade de dados distintas, ou seja, a quantidade de *bits* que compõem cada entrada e quantas entradas no total compõem a memória, respectivamente. Essas duas características são parametrizadas de acordo com as dimensões do problema de controle adotado e o número de *bits* utilizados em ponto flutuante (FP). A Tabela 3.1 apresenta todas as matrizes presentes nos módulos do MPC sem restrições e com restrições, mostrando quais são implementados como memórias ROM ou memórias RAM e o tamanho da profundidade e da largura de dados de cada uma.

Tabela 3.1: Relação dos tipos de matrizes utilizadas na arquitetura dos controladores MPC.

Matriz	Tipo de memória	Largura	Profundidade	Controlador
F1	ROM	$n_x.n_u$	$N.FP$	Ambos
F2	ROM	$N.{n_u}^2$	$N.FP$	Ambos
F3	ROM	${n_u}^2$	$N.FP$	Ambos
yref	ROM	$T_s.t_{sim}.n_u$	$N.FP$	Ambos
ud	ROM	1	$n_u.FP$	Ambos
KN	ROM	n_u	$n_x.FP$	MPC sem restrições
GN	ROM	${n_u}^2$	N.FP	MPC sem restrições
LN	ROM	n_u	$n_u.FP$	MPC sem restrições
G1	ROM	${n_x}^2$	$N.FP$	MPC com restrições
G2	ROM	$n_x.n_u$	$N.FP$	MPC com restrições
G3	ROM	n_x	$N.FP$	MPC com restrições
Hr	ROM	$n_e.n_u$	$(n_e.n_u).FP$	MPC com restrições
Ar	ROM	20.*N*	$(n_e.n_u).FP$	MPC com restrições
$\mathbf{Pi_e}$	ROM	$N.n_u$	$(n_e.n_u).FP$	MPC com restrições
F	RAM	n_u	$N.FP$	MPC com restrições
Bineq	RAM	n_x	$N.FP$	MPC com restrições
Fr	RAM	1	$(n_e.n_u).FP$	MPC com restrições
Br	RAM	20	$N.FP$	MPC com restrições
G	RAM	1	$(n_e.n_u).FP$	MPC com restrições
Ap	RAM	20	$N.FP$	MPC com restrições
Hp	RAM	1	$(n_e.n_u).FP$	MPC com restrições

A tabela acima apresenta os parâmetros T_s (tempo de amostragem do sistema sendo controlado) e t_sim que corresponde ao tempo total de simulação. É possível perceber que nenhuma memória possui largura de dados maior que $N.FP$, que se dá por conta de os módulos de multiplicação e soma/subtração matriciais possuírem *N* multiplicadores e/ou somadores em paralelo, conforme explicado na subseção anterior.

3.2.3 Arquitetura em VHDL do MPC sem restrições

Para desenvolver a lei de controle do MPC sem restrições, é necessário realizar as operações matriciais descritas na Equação 2.34 na subseção 2.1.4. As etapas necessárias para implementar esse primeiro tipo

de controlador são:

1. Obtêm-se os valores atuais dos estados e das variáveis de controle (x_k e u_k, respectivamente);
2. As matrizes constantes de custo e de restrição do MPC são calculadas com base nas matrizes que descrevem o sistema em espaço de estados ($\mathbf{A}$ e $\mathbf{B}$) e ponderações de estados $\mathbf{Qx}$ e $\mathbf{Qy}$:
   - $\mathbf{F1}$;
   - $\mathbf{F2}$;
   - $\mathbf{F3}$;
3. Calcula-se a lei de controle do MPC sem restrições definida pela Equação 2.34;
4. Os valores das variáveis de estado são atualizadas (x_{k+1}) pelo modelo do sistema, de acordo com os valores atuais dos estados e o valor das variáveis de controle obtidos pelo controlador MPC;
5. As etapas 1, 2, 3, 4 e 5 são repetidas em sequência durante todo o período de simulação.

Em Alamir(2013), as etapas 1 e 2 citadas acima são realizadas pelo *script* compute_ MPC_ matrices.m; a etapa 3 é executado pelo controlador em si e a última etapa, é implementada por um simulador do modelo do sistema sendo controlado, por meio das operações matricias que descrevem o sistema em espaço de estados, definidas na Equação 2.1.

Portanto, para simular o controlador MPC sem restrições é necessário desenvolver dois módulos distintos: um módulo que calcula a lei de controle do MPC sem restrições propriamente dita, denominado *unconstrained_ MPC* e um módulo que simula o modelo do sistema sendo controlado, denominado *model_ simulator*. Este último não é parte do controlador em si, mas é necessário para realizar os testes e validar o controlador MPC desenvolvido.

A Figura 3.4 apresenta um diagrama de blocos mostrando como os módulos interagem entre si para implementar o MPC sem restrições.

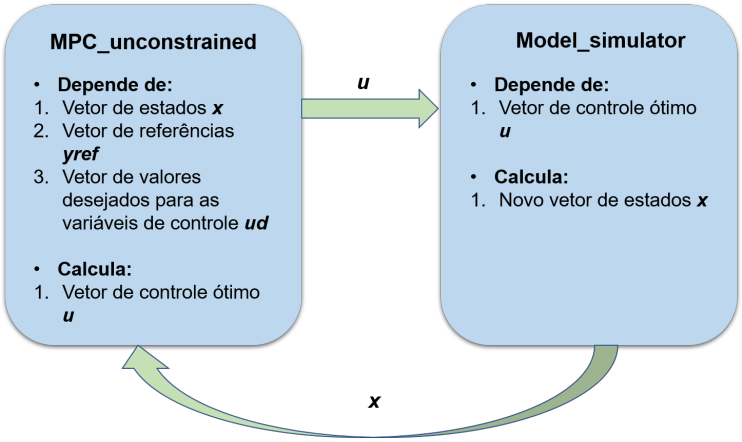

Figura 3.4: Diagrama de blocos com os módulos principais que compõem a arquitetura do MPC sem restrições

.

Portanto, conforme mostrado na Figura 3.4, o módulo MPC_ unconstrained calcula a lei de controle do MPC sem restrições dada por um vetor de controle ótimo, com base nos valores atuais das variáveis de estado, referência e valores desejados para as variáveis de controle do sistema. O vetor de controle ótimo é utilizado pelo módulo *model_ simulator* para calcular os novos valores do vetor de estados do sistema sendo controlado. A interação entre os módulos prossegue dessa forma durante todo o período de simulação do MPC sem restrições.

Cada um desses módulos foi desenvolvido e testado individualmente, antes de serem integrados para compor a arquitetura do MPC sem restrições em VHDL.

Antes de apresentar a arquitetura em *hardware* do MPC sem restrições, é importante destacar algumas informações sobre o módulo *model_ simulator* especificamente. Conforme foi citado anteriormente, esse módulo é necessário para realizar os testes e a validação dos controladores MPC (sem ou com restrições), mas ele não faz parte do controlador em si. Por conta disso, esse módulo possui uma especificidade que não é compartilhada com nenhum outro módulo desenvolvido: este possui seus próprios módulos de operação matricial em ponto flutuante (multiplicação e soma/subtração). Tal afirmação pode ser evidenciada na Figura 3.5, que mostra a arquitetura de *hardware* do módulo *model_ simulator*.

Como os módulos são implementados utilizando uma lógica pré-desenvolvida em *scripts* do MATLAB, a dimensão das entradas, saídas e sinais presentes nesta e nas demais arquiteturas que serão apresentadas variam de acordo com o problema de controle adotado. Estas são sempre dadas em função de n_x, que corresponde ao número de variáveis de estado do sistema, n_u que corresponde ao número de variáveis de controle e n_y, que corresponde ao número de saídas do sistema. Para o caso do quadrirrotor, $n_x = 12$, $n_u = 4$ e $n_y = 6$.

Esse módulo possui uma máquina de estados (*model_ simulator_ FSM*) que sequencia e controla o início e o fim das operações matricias que devem ser realizadas para implementar a simulação do modelo, por meio da Equação 2.1. Assim, o módulo de multiplicação matricial inicia sua operação por meio da entrada *start_ matrix_ mult* e realiza as operações $\mathbf{A}x$ e $\mathbf{B}u$, para calcular o novo vetor de variáveis de estado a ser utilizado no próximo instante de amostragem pelo controlador MPC. As matrizes de estado $\mathbf{A}$ e $\mathbf{B}$ estão armazenadas em memórias ROM, acessadas pela máquina de estados durante a etapa de multiplicação.

O módulo de multiplicação matricial aqui funciona somente no modo de operação *mode* = '0', pois as matrizes de estados do sistema possuem menores dimensões e podem ser calculadas de maneira mais rápida.

Ao término da operação de multiplicação, por meio de *ready_ matrix_ mult* = '1', a máquina de estados *model_ simulator _ FSM* inicia a operação de soma, para calcular $\mathbf{A}x + \mathbf{B}u$. De maneira análoga, a soma matricial é iniciada por *start_ matrix_ add*, e ao terminar, informa à máquina de estados seu encerramento por meio de *ready_ matrix_ add* = '1'.

Optou-se por incluir na arquitetura do módulo *model_ simulator* módulos de multiplicação e soma/ subtração matriciais próprios, justamente por este módulo não ser parte do controlador em si. Como o *kit* de FPGA utilizado possui uma grande quantidade de recursos disponíveis, e permite a implementação desta abordagem em *hardware*, é possível examinar a quantidade de recursos de *hardware* exigidos pelo

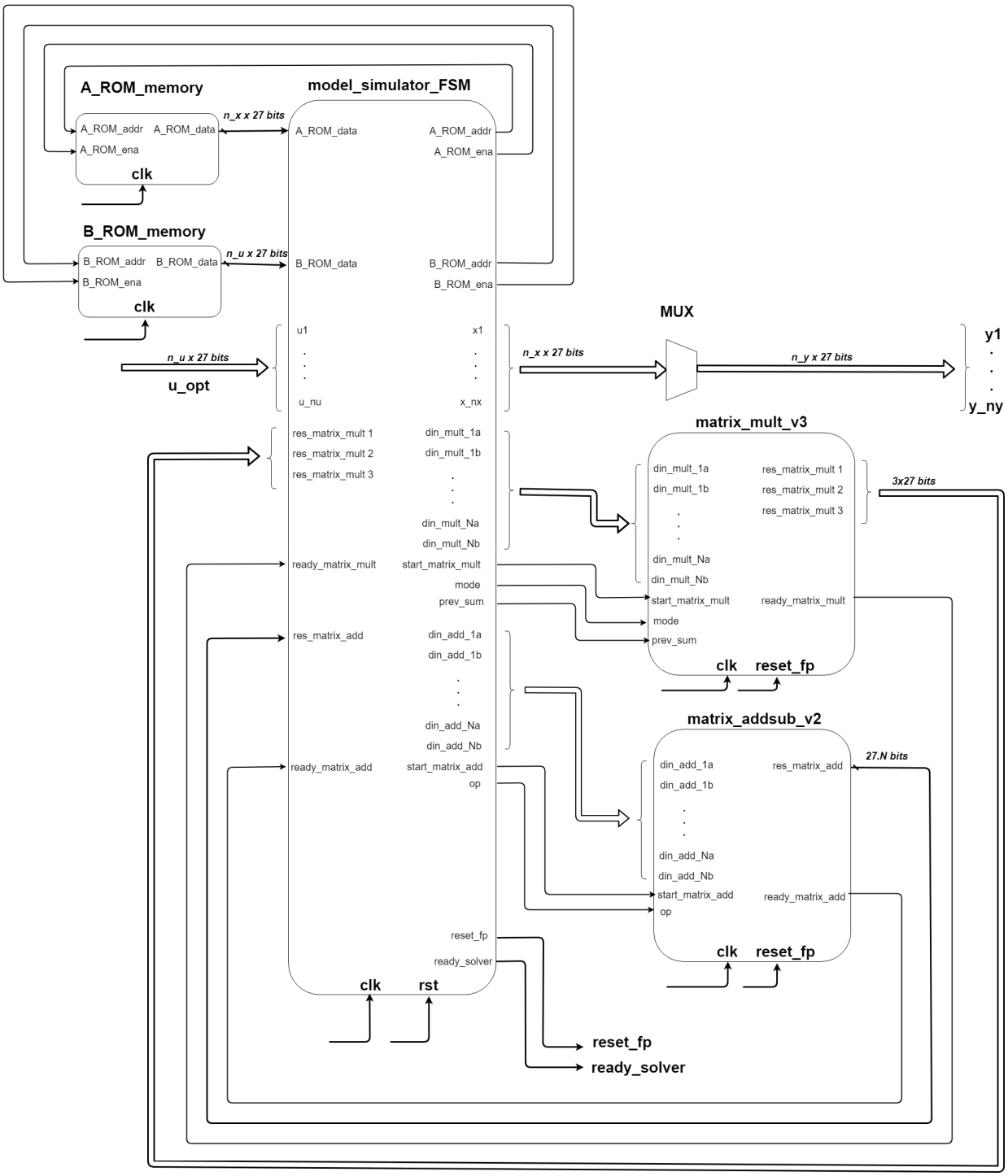

Figura 3.5: Arquitetura em *hardware* do módulo model_ simulator.

controladores de maneira mais precisa, "isolando"de certa forma controlador e simulador do modelo.

Após essa explicação referente ao módulo *model_ simulator*, é possível finalmente apresentar a arquitetura em *hardware* do controlador MPC sem restrições, dada pela Figura 3.6.

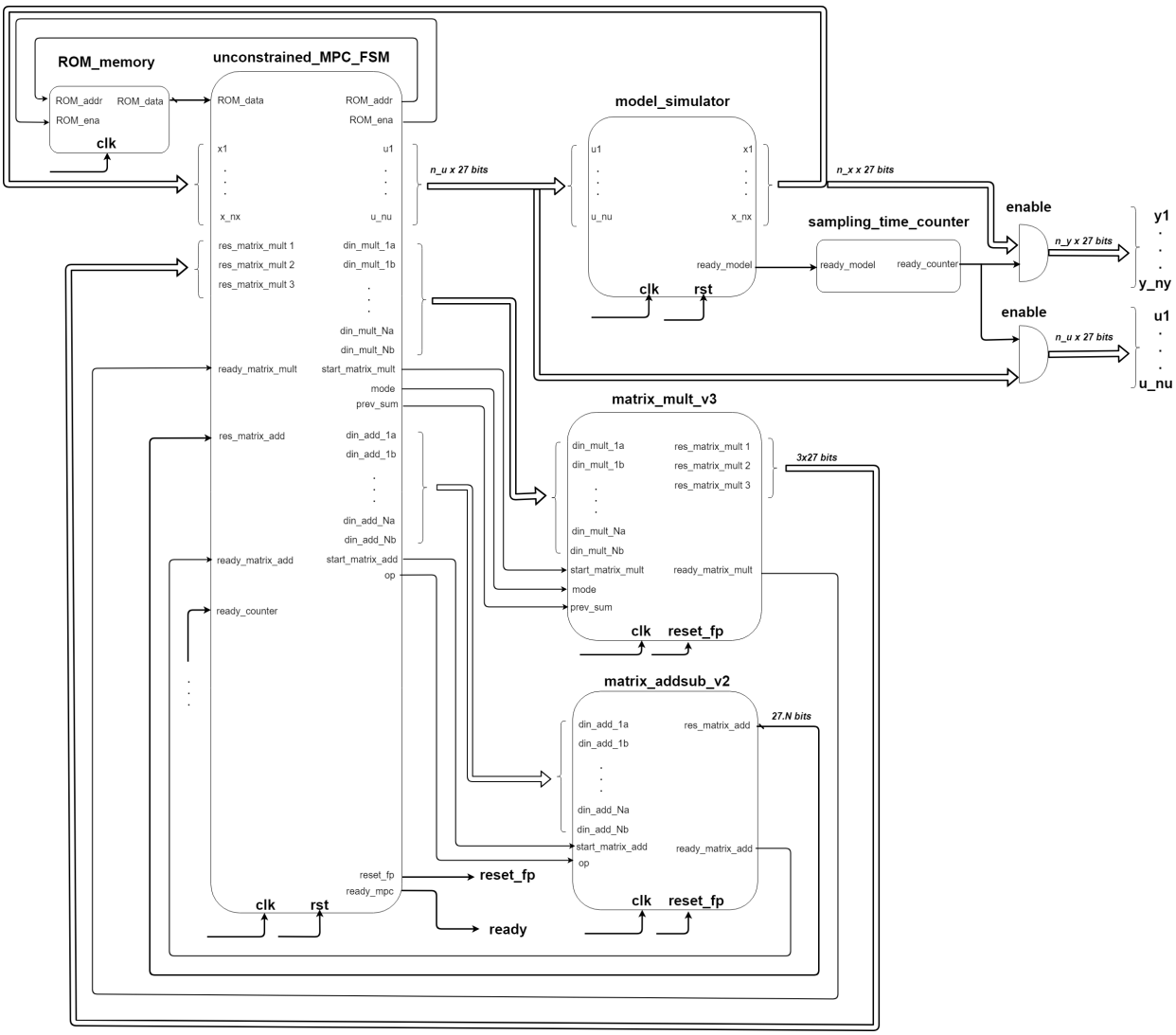

Figura 3.6: Arquitetura em *hardware* desenvolvida para o controlador MPC sem restrições.

A Figura 3.6 mostra que além do módulo *model_ simulator*, detalhado anteriormente, o MPC sem restrições é composto por uma máquina de estados *unconstrained_ MPC_ FSM*, cuja função é realizar as operações matriciais da Equação 2.34. De forma semelhante às demais máquinas de estados que foram e serão apresentadas ao longo do trabalho, a mesma acessa as matrizes necessárias para a realização dos cálculos de multiplicação e soma/subtração por meio de acesso a memórias ROM. Estas memórias armazenam as matrizes individualmente, mas na Figura 3.6 foram todas agrupadas em uma única memória para facilitar o entendimento da arquitetura. As matrizes acessadas pelo MPC sem restrições, conforme pode ser visto na Tabela 3.1 já apresentada, correspodem a $\mathbf{KN}$, $\mathbf{GN}$ e $\mathbf{LN}$.

Aqui, os módulos de multiplicação e soma estão externos ao módulo *unconstrained_ MPC_ FSM*, que coordena o momento em que cada operação deve ser realizada, quais os elementos das matrizes que devem ser utilizados como operandos e quando a resposta é calculada. Com base nos valores de x disponibilizados por *model_ simulator*, o módulo *unconstrained_ MPC_ FSM* calcula a variável de controle ótima, sem levar em consideração as restrições do problema.

Ao término da operação do módulo *model_ simulator*, as variáveis de controle e as novas variáveis

de estado estão calculadas. Entretanto, essa operação demora menos tempo do que o período de amostragem do sistema (para o quadrirrotor, 50 ms), que é um parâmetro essencial para o funcionamento correto do controlador. Por isso, o módulo *sampling_ time_ counter* implementa um contador, que com base na frequência de *clock* utilizada, espera que sejam completados os 50 ms necessários, para habilitar as saídas ($y_1 - y_6$) e as variáveis de controle calculadas ($u_1 - u_4$). Somente quando o período de 50 ms é atingido, sinalizado pela saída *ready_ counter* = '1', essas informações se tornam disponíveis e o módulo *unconstrained_ MPC_ FSM* reinicia sua operação e o ciclo se repete.

As saídas do sistema são dadas pelas variáveis de estados x_1, x_3, x_5, x_7, x_9 e x_{11}, que correspondem a x, y, z, ϕ, θ e ψ, respectivamente. Em outras palavras, as saídas do modelo do quadrirrotor são dadas por $y_1 = x_1$, $y_2 = x_3$, $y_3 = x_5$, $y_4 = x_7$, $y_5 = x_9$ e $y_6 = x_{11}$.

3.2.4 Arquitetura em VHDL da formulação completa do MPC

Já para a formulação completa do MPC, tomou-se como base os algoritmos descritos em *compute_ MPC_ matrices.m* e *solver_ sat_ pen.m* (ALAMIR, 2013). A estratégia de calculo do MPC que é implementada por esses algoritmos segue os mesmos passos introduzidos na subseção 2.1.1, ou seja:

1. Obtêm-se os valores atuais dos estados e das variáveis de controle (x_k e u_k, respectivamente);
2. As matrizes constantes de custo e de restrição do MPC são calculadas com base nas matrizes que descrevem o sistema em espaço de estados ($\mathbf{A}$ e $\mathbf{B}$) e restrições do sistema (y_c^{min}, y_c^{max}, u^{min}, u^{max}, δ^{min} e δ^{max}):
   - $\mathbf{F1}$;
   - $\mathbf{F2}$;
   - $\mathbf{F3}$;
   - $\mathbf{G1}$;
   - $\mathbf{G2}$;
   - $\mathbf{G3}$;
   - $\mathbf{H}$;
   - $\mathbf{A_{ineq}}$;
   - $\mathbf{F}$;
   - $\mathbf{B_{ineq}}$.
3. Aplica-se a ponderação exponencial para calcular as matrizes de custo e restrição reduzidas:
   - $\mathbf{H_r}$;
   - $\mathbf{A_r}$;
   - $\mathbf{F_r}$;
   - $\mathbf{B_r}$.

4. O *solver* de QP (algoritmo PGE) utiliza as matrizes reduzidas para calcular as variáveis de controle ótimas mediante às restrições do MPC;

5. Os valores das variáveis de estado são atualizadas (x_{k+1}) pelo modelo do sistema, de acordo com os valores atuais dos estados e o valor das variáveis de controle obtidos pelo controlador MPC;

6. As etapas 1, 2, 3, 4 e 5 são repetidas em sequência durante todo o período de simulação.

Cada *script* em MATLAB apresentado em Alamir (2013) realiza algumas das etapas acima. As etapas 1 e 2, por exemplo, são realizadas pelo *script compute_ MPC_ matrices.m*; a etapa 3 é realizada pelos algoritmos *Pi_ e.m* em conjunto com algumas operações matriciais adicionais. A etapa 4 é implementada pelo *script solver_ sat_ pen.m* e, por fim, a etapa 5 é implementada por um simulador do modelo, da mesma forma que no MPC sem restrições.

Assim como no caso anterior, os módulos da formulação completa do MPC foram desenvolvidos em VHDL. O módulo para calcular as matrizes de custo e restrição reduzidas e implementa as etapas 1, 2 e 3 descritas acima é denominado *compute_ MPC_ matrices*; o módulo que implementa o algoritmo PGE é denominado *solver_ sat_ pen*; e, por fim, o módulo *model_ simulator* realiza a simulação do sistema sendo controlado.

Uma vez que cada bloco é desenvolvido e validado individualmente, os mesmos podem ser conectados a fim de implementar a estratégia de controle propriamente dita.

Com essas definições, é possível desenvolver um diagrama de blocos, (assim como foi feito com o MPC sem restrições na subseção anterior) do MPC, mostrando como os módulos interagem entre si para implementar as etapas de cálculo do controlador MPC, conforme mostrado na Figura 3.7.

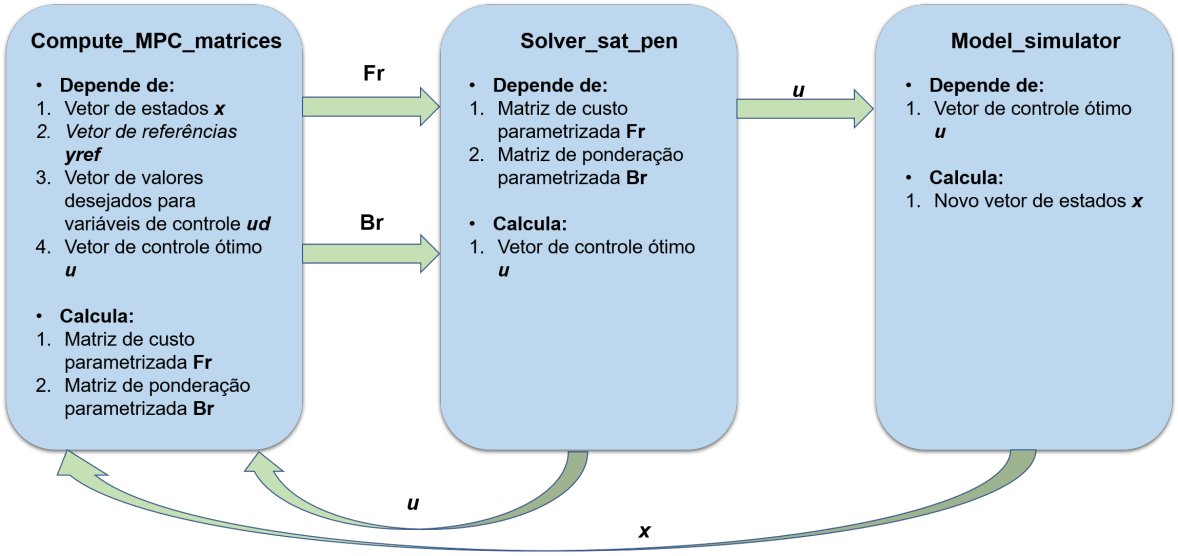

Figura 3.7: Diagrama de blocos do MPC: módulos necessários para a implementação do algoritmo utilizando o *solver* de QP PGE (*solver_ sat_ pen*) e interconexão entre os mesmos.

A Figura 3.7 mostra que, apesar de algumas semelhanças com a arquitetura do MPC sem restrições mostrada na Figura 3.4, para a formulação completa do MPC são necessários três módulos: o módulo *compute_ MPC_ matrices*, que com base no vetor de estados x, referências y_{ref}, valores desejados para as

variáveis de controle u_d e no vetor de controle ótimo u calcula as matrizes parametrizadas de custos ($\mathbf{Fr}$) e de restrições ($\mathbf{Br}$); o módulo *solver_ sat_ pen*, que utiliza as matrizes calculadas pelo módulo *compute_ MPC_ matrices* para calcular o vetor de controle ótimo u; e o módulo *model_ simulator*, que calcula os novos valores do vetor de estados x do sistema sendo controlado, com base no vetor de controle ótimo u previamente calculado pelo módulo *solver_ sat_ pen*. Esses módulos continuam interagindo dessa forma continuamente, até que a simulação do MPC termine.

A arquitetura em *hardware* do MPC com restrições é apresentada na Figura 3.8 abaixo.

Por meio da Figura 3.8, é possível notar que assim com no caso do MPC sem restrições, o módulo *model_ simulator* não compartilha recursos com os demais módulos que compõem o MPC, por não fazer parte do controlador em si e possuir internamente seu próprio multiplicador e somador/subtrator matricial, conforme mostrado na Figura 3.5. Os demais módulos (*compute_ MPC_ matrices_ FSM* e *solver_ sat_ pen_ FSM*) são máquinas de estado que realizam etapas diferentes da estratégia de controle do MPC: cálculo das matrizes parametrizadas de custo ($\mathbf{Fr}$) e restrições ($\mathbf{Br}$) e cálculo das variáveis de controle ótimas u_1, u_2, u_3 e u_4, respectivamente. Como *compute_ MPC_ matrices_ FSM* deve calcular $\mathbf{Fr}$ e $\mathbf{Br}$ para que *solver_ sat_ pen_ FSM* execute suas operações de maneira correta, ambos compartilham a utilização de multiplicador e somador/subtrator matricial, alternando qual módulo está utilizando os mesmos por meio de multiplexadores.

Assim como no caso do MPC sem restrições os valores de x são disponibilizados por *model_ simulator*, para o módulo *compute_ MPC_ matrices*. Este calcula $\mathbf{Fr}$ e $\mathbf{Br}$, armazenando essas matrizes em memórias RAM, que serão acessadas pelo módulo *solver_ sat_ pen_ FSM*, que ao realizar determinadas operações matriciais, irá salvar as matrizes resultantes nessas mesmas memórias, assim como em outras memórias RAM.

Isso é diferente do que acontecia no caso do MPC sem restrições que, por calcular o vetor de controle por meio de uma lei de controle bem mais simples e analítica, não precisa armazenar resultados intermediários em memória.

Na arquitetura do MPC com restrições, assim como feito anteriormente com a arquitetura do MPC sem restrições, as memórias ROM foram agrupadas em módulos únicos de memória ROM para *unconstrained_ MPC_ FSM* e *solver_ sat_ pen*, assim como as memórias RAM, para facilitar o entendimento da arquitetura. Na prática, cada matriz constitui uma memória distinta.

No caso das memórias RAM, matrizes intermediárias são armazenadas e, no devido momento, podem ser sobrescritas para armazenar novas matrizes resultantes do funcionamento do módulo *solver_ sat_ pen*, quando as mesmas são utilizadas no cálculo da matriz resultante, mas não serão mais utilizadas em nenhuma outra etapa do *solver* de QP.

As variáveis de controle do MPC com restrições são as saídas do módulo solver_ sat_ pen. As saídas do quadrirrotor são dadas da mesma forma que no caso do MPC sem restrições: por meio das saídas x_1, x_3, x_5, x_7, x_9 e x_{11} do módulo *model_ simulator* de forma que $y_1 = x_1$, $y_2 = x_3$, $y_3 = x_5$, $y_4 = x_7$, $y_5 = x_9$ e $y_6 = x_{11}$.

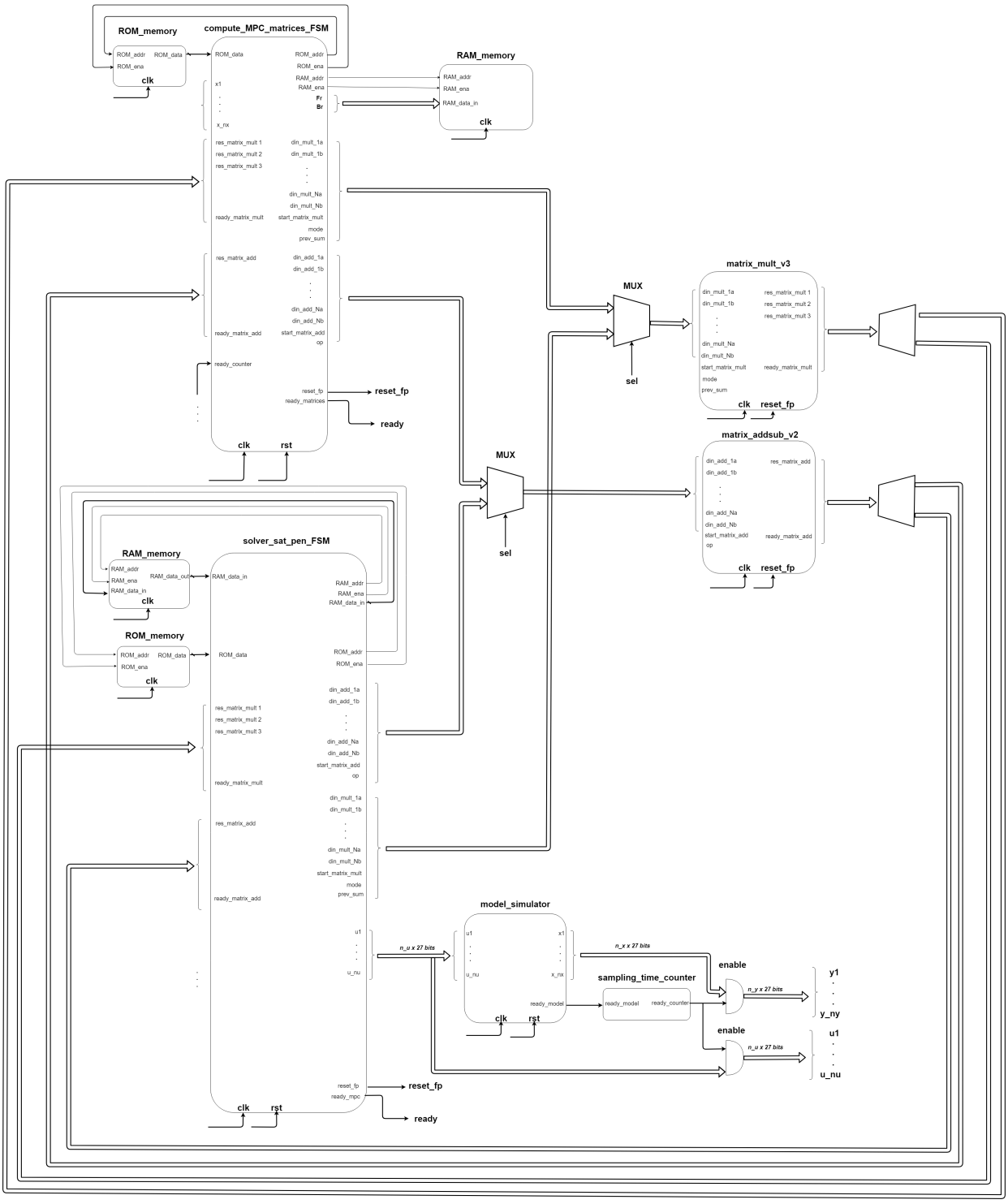

Figura 3.8: Arquitetura em *hardware* do controlador MPC com sua formulação completa: aplicando-se as restrições do problema de controle.

3.3 Simulação da arquitetura do controlador MPC em FPGA

Uma vez que todos os módulos que formam os controladores (MPC sem restrições e MPC com restrições) foram desenvolvidos e integrados, torna-se necessário simular cada controlador, mediante os cenários

explicados na Seção 3.1.

Para tal, é necessário estabelecer quais os procedimentos de validação para esses controladores. Em um módulo VHDL padrão, muitas vezes utiliza-se um padrão de verificação, composto por diversas etapas que permitem analisar o módulo, tanto do ponto vista de funcionamento (se o mesmo produz saídas corretas quando um determinado padrão de entradas é inserido no sistema) quanto do ponto de vista da arquitetura em si (consumo de recursos de *hardware* utilizados para sua implementação e o tipo de recurso utilizado).

A Figura 3.9 apresenta essas etapas do processo de desenvolvimento. De maneira geral, esse processo se aplica a linguagens de descrição de *hardware*, sejam elas VHDL, *Verilog*, ou qualquer outra que tenha sido escolhida.

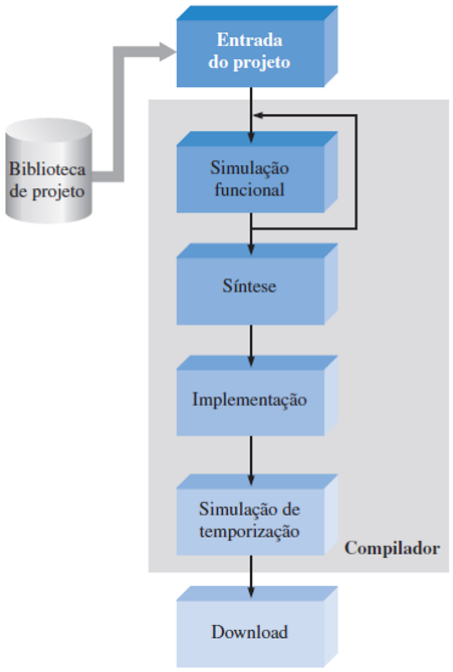

Figura 3.9: Etapas de validação de sistemas desenvolvidos em linguagem de descrição de *hardware*. Adaptado de: (TOCCI; WIDMER; MOSS, 2007)

.

A primeira das etapas apresentadas na Figura 3.9 corresponde à etapa de entrada do projeto, ou seja, a concepção do módulo em si. Esta etapa corresponde ao que já foi apresentado na subseção anterior, ou seja, é neste etapa que são desenvolvidos os algoritmos que implementam o controlador MPC.

Uma vez que esses códigos estão prontos, eles são compilados e podem ser testados na etapa de verificação funcional, na qual são realizadas simulações a nível comportamental, ou seja, verificar se saídas adequadas são geradas por um conjunto de entradas específicas aplicadas ao módulo VHDL sendo testado (TOCCI; WIDMER; MOSS, 2007).

Com base nos módulos desenvolvidos, prossegue-se para a etapa de síntese, em que é gerada uma *netlist*, ou seja, lista de componentes que relaciona todos os recursos de *hardware*, e suas quantidades, necessários para implementar a lógica desejada (TOCCI; WIDMER; MOSS, 2007).

A etapa seguinte, denominada implementação, corresponde ao mapeamento de toda a estrutura lógica

descrita pela *netlist* gerada na etapa anterior de síntese (TOCCI; WIDMER; MOSS, 2007).

A penúltima etapa consiste na realização de simulações de temporização, que confirmam que não há falhas no projeto ou problemas de temporização, relacionados à frequência de operação desejada para a aplicação, por exemplo (TOCCI; WIDMER; MOSS, 2007).

Por fim, têm-se a etapa de *download*, onde é gerado um *bitstream*. O *bitstream* corresponde a uma sequência de *bits* gerada para um dispositivo programável específico, (TOCCI; WIDMER; MOSS, 2007), que é transferido para o dispositivo e permite ao mesmo implementar o projeto desenvolvido em linguagem de descrição de *hardware*.

Apesar de se tratar de uma metodologia bastante utilizada na prática, essas etapas não são suficientes para validar a arquitetura desenvolvida neste trabalho. Isso porque o controlador MPC, seja ele com ou sem restrições, trata-se de um sistema complexo, contendo várias informações referentes a estados e variáveis de controle, cada uma com tamanho de 27 ou 32 *bits*, conforme já mencionado na Seção 3.2. Obter os dados do controlador durante os experimentos é uma tarefa que exige ferramentas específicas para isso.

A ferramenta escolhida para tal propósito corresponde ao *HDL Verifier*, ferramenta disponível no MATLAB. Esta ferramenta permite a validação de módulos desenvolvidos em VHDL ou *Verilog* por meio de uma técnica denominada FPGA *in the Loop*, ou simplesmente, FIL. Um esquemático dessa técnica é mostrado na Figura 3.10.

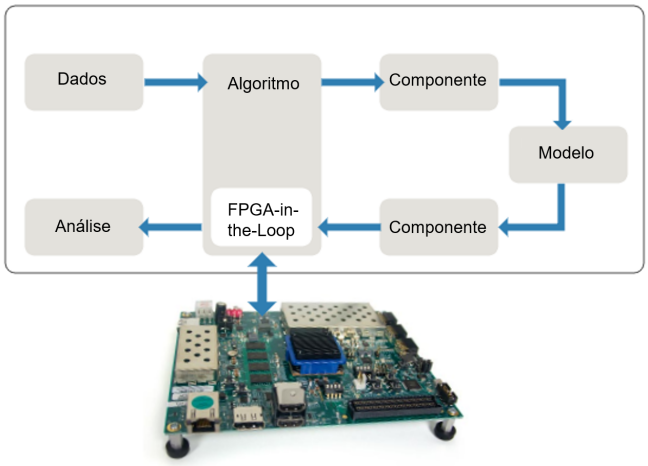

Figura 3.10: Esquemático de testes de algoritmo a nível de sistemas utilizando a técnica FPGA-*in-the-Loop* (FIL). Adaptado de: (MATHWORKS, 2019)

.

Conforme pode ser visto na Figura 3.10, o FIL é uma técnica utilizada para realizar testes com algoritmos desenvolvidos em linguagem de descrição de *hardware* a nível de sistemas, ou seja realizar testes para validar algoritmos embarcados em um FPGA. Usando o *HDL Verifier*, é possível transformar os arquivos desenvolvidos em VHDL para os controladores em um bloco no ambiente *Simulink*. Assim, é possível que dados iniciais sejam enviados do MATLAB para o FPGA, que executa o controlador desenvolvido em VHDL, enquanto os dados do algoritmo no FPGA se comunicam com componentes e modelos no MATLAB, por meio de blocos adicionais do *Simulink*. Da mesma forma, os dados tratados pelos componentes e modelos no *Simulink*, podem ser enviados de volta para o FPGA, fechando o *loop* propriamente dito,

permitindo que uma simulação completa possa ser estabelecida e, ao fim da mesma, os dados provenientes do FPGA possam ser armazenados e analisados.

O *setup* para todos os testes que são realizados neste trabalho pode ser visto na Figura 3.11. O computador *host* MATLAB é responsável por gerar estímulos ou dados de entrada por meio de um modelo desenvolvido no ambiente *Simulink* do MATLAB. Esses dados são enviados ao FPGA, no qual encontra-se implementado o controlador MPC em VHDL, por meio de um cabo *Ethernet*. Os dados gerados pelo controlador são enviados para o computador *host*, por meio do mesmo cabo *Ethernet* que é utilizado para envio de dados ao FPGA, permitindo que todas as informações sejam armazenadas no próprio MATLAB. Essa é a principal vantagem para utilização deste ferramenta: como os controladores foram validados em MATLAB antes de serem implementados em VHDL, é possível gerar gráficos com os resultados da simulação do controlador em FPGA e compará-los com a simulação computacional realizada previamente no MATLAB de maneira mais fácil, sem se preocupar com os protocolos de comunicação entre FPGA e MATLAB, que são realizadas pela ferramenta *HDL Verifier*.

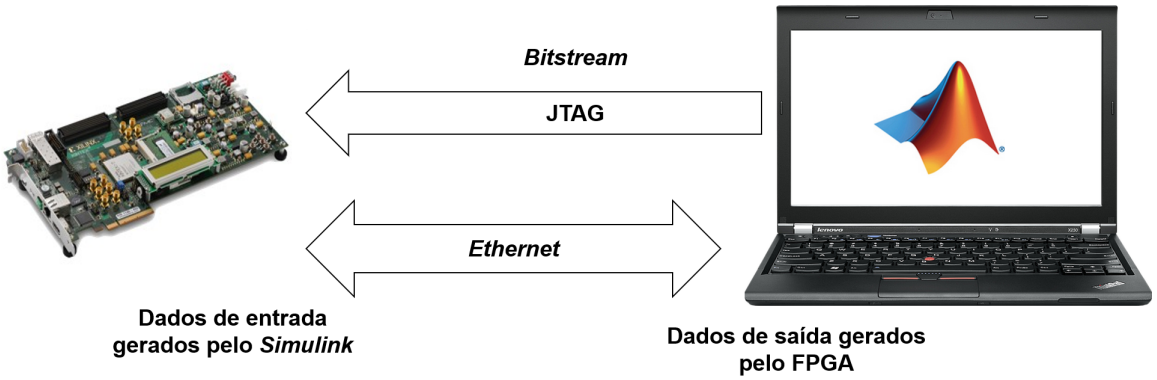

Figura 3.11: *Setup* de testes para conexão entre FPGA e Simulink utilizando o *HDL Verifier*.

O *HDL Verifier* utiliza ferramentas próprias dos fabricantes de FPGA para implementar todas as etapas descritas na Figura 3.9. No caso deste trabalho, a ferramenta utilizada corresponde ao *Vivado 2016.4*, da *Xilinx*. Portanto, em conjunto com o *Vivado 2016.4*, o próprio *HDL Verifier* realiza síntese, implementação, simulação temporal e *download* do controlador desenvolvido em VHDL para o FPGA adotado, por meio do cabo JTAG apresentado na Figura 3.11.

O FPGA escolhido para ser utilizado neste trabalho é o *kit* KC-705 da *Xilinx*, que foi escolhido principalmente por conta da sua grande quantidade de recursos para implementação da lógica dos controladores MPC apresentados. Como o MPC é uma técnica que demanda bastante processamento, optou-se por escolher um *hardware* com maior quantidade de recursos de *hardware* disponíveis, já que *a priori* a quantidade demandada para implementação do MPC não era conhecida.

A Tabela 3.2 abaixo apresenta as principais características do *kit* adotado.

Tabela 3.2: Características e funcionalidades do *kit* KC-705 da Xlinx. Fonte: (XILINX, 2019).

Frequência de *clock* máxima	200 MHz
Tipo de *clock*	Diferencial
FPGA	Xilinx Kintex-7
LUTs	203800
BRAMs	445
DSPs	840
Pinos de I/O	500

Com tudo definido, é possível finalmente realizar a simulação do MPC em FPGA, para validação dos controladores desenvolvidos. O *HDL Verifier* gera um bloco em *Simulink* correspondente ao controlador MPC, conforme pode ser visto na Figura 3.12, que é colocado no arquivo de simulação do *Simulink*.

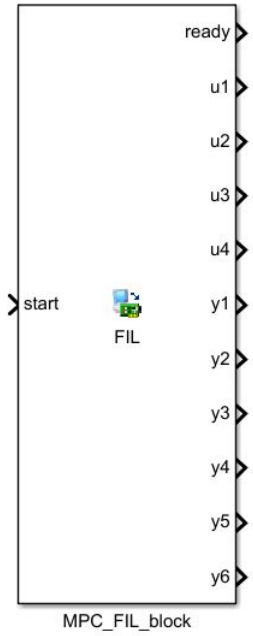

Figura 3.12: Bloco FIL referente ao algoritmo embarcado em FPGA utilizado na simulação com o HDL Verifier no Simulink.

Esse bloco é gerado com base nos arquivos VHDL que são disponibilizados para a ferramenta e ele representa o algoritmo embarcado em FPGA, possuindo a mesma quantidade de entradas e saídas para o MPC sem restrições e para o MPC com restrições, já que ambos estão controlando o mesmo modelo. A diferença está na lógica interna de cada controlador, conforme apresentado nas subseções anteriores.

Um detalhe importante a ser destacado é que, conforme pode ser visto na Figura 3.12, há uma saída denominada *ready*. Essa saída está conectada às saídas *ready* de cada um dos controladores MPC, de modo que por meio dela é possível saber o tempo de cálculo que cada controlador demanda para obter a sequência de controle ótima, armazenada nas saídas $u1$, $u2$, $u3$ e $u4$ da Figura 3.12.

Os testes utilizando o *HDL Verifier* são realizados utilizando-se uma frequência de *clock* de 50 MHz, pelo fato de esse ser um dos valores que a ferramenta utiliza como padrão.

Com base em todos esses procedimentos, é possível realizar a simulação dos controladores MPC,

sem restrições e com restrições, aplicados a todos os cenários descritos para o modelo do quadrirrotor. Os resultados desses testes são apresentados no próximo capítulo, juntamente com discussões acerca dos mesmos.

Capítulo 4

Resultados e discussões

Este capítulo apresenta os resultados das atividades descritas no capítulo anterior de Metodologia. Consequentemente, as divisões desse capítulo são dadas em subseções iguais às do capítulo anterior, ou seja, de acordo com as diferentes etapas de atividades desenvolvidas:

1. Simulação do controlador MPC utilizando MATLAB;
2. Desenvolvimento de arquitetura em VHDL do controlador MPC;
3. Simulação da arquitetura em VHDL do controlador MPC;
4. Comparação das arquiteturas desenvolvidas em VHDL com os resultados obtidos nas simulações em MATLAB.

Cada subseção também contem comentários acerca dos resultados obtidos em cada atividade.

4.1 Resultados da simulação do controlador MPC utilizando MATLAB

Após a realização das simulações em MATLAB dos controladores MPC aplicados ao quadrirrotor, tanto utilizando o MPC sem restrições quanto utilizando a formulação com restrições, foi possível obter os valores das matrizes de ponderação das variáveis de estado $\mathbf{Qy}$ e das matrizes de ponderação das variáveis de controle $\mathbf{Qu}$ que obtiveram melhores resultados para os controlar o quadrirrotor.

Estes valores são apresentados abaixo nas Equações 4.1 e 4.2, para o caso do MPC sem restrições, e nas Equações 4.3 e 4.4. O índice $\mathbf{unc}$ refere-se às matrizes de ponderação para o MPC sem restrições. As matrizes sem esse índice referem-se ao MPC com restrições.

$$\mathbf{Qy_{unc}} = \begin{pmatrix} 1.00 & 0.00 & 0.00 & 0.00 \\ 0.00 & 1.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 10.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 20.00 \end{pmatrix}, \tag{4.1}$$

$$\mathbf{Qu_{unc}} = \begin{pmatrix} 0.01 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.01 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.01 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.01 \end{pmatrix}, \tag{4.2}$$

$$\mathbf{Qx} = \begin{pmatrix} 1.00 & 0.00 & 0.00 & 0.00 \\ 0.00 & 1.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 14.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 20.00 \end{pmatrix}, \tag{4.3}$$

$$\mathbf{Qu} = \begin{pmatrix} 0.01 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.01 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.01 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.01 \end{pmatrix}. \tag{4.4}$$

Utilizando esses valores de ponderação, é possível realizar as simulações dos controladores MPC para avaliar quais os melhores valores de *N* e o número de iterações N_{iter} do *solver*, no caso do MPC com restrições, que são utilizados para implementar os controladores MPC em *hardware*. Os resultados dessa etapa são divididos em subseções para melhor organização do documento.

4.1.1 Resultados da simulação do controlador MPC sem restrições em MATLAB

Durante a realização das simulações em MATLAB, tanto para o MPC sem restrições quanto para o MPC com restrições, foi possível constatar duas informações importantes:

1. As dinâmicas que são mais difíceis de controlar são z e ψ;
2. ψ demora bastante tempo para estabilizar e, em geral, demanda um *N* alto para que isso ocorra em pouco tempo.

Para evitar uma quantidade desnecessária de imagens e facilitar a organização deste documento, serão apresentados resultados dos testes em MATLAB referentes às dinâmicas em z e ψ, que são as dinâmicas mais determinantes na escolha de *N*, além de ϕ e θ que são as variáveis restringidas no MPC com restrições.

Com base nessas considerações, são mostrados abaixo os resultados das simulações em MATLAB do MPC sem restrições, através das Figuras 4.1 - 4.2, mostrando tanto as variáveis de estados especificadas no último parágrafo quanto as variáveis de controle, para o cenário em que são aplicadas referências de degrau filtrado em x, y e z.

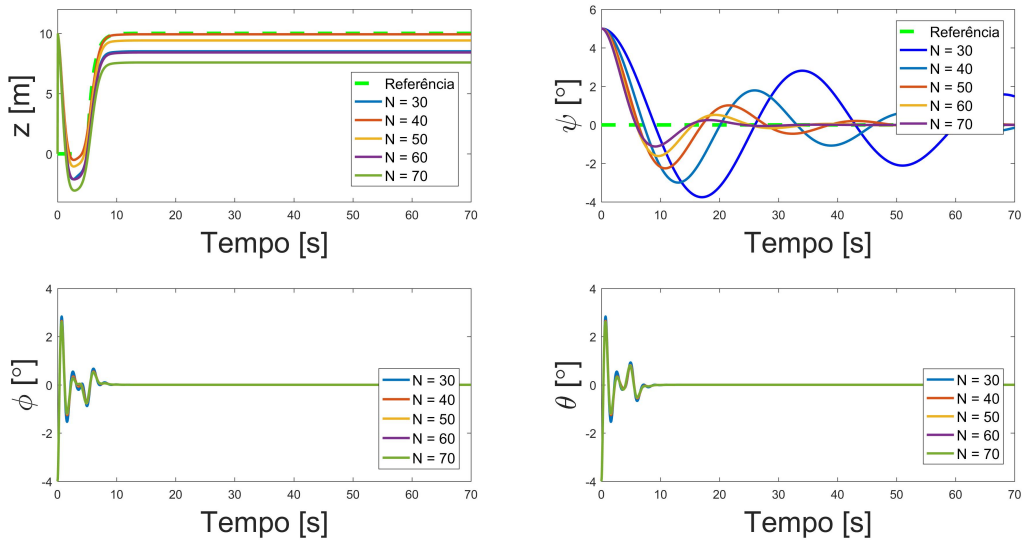

Figura 4.1: Variação das variáveis de estado z e ψ de acordo com o valor de N adotado para o controlador MPC sem restrições no cenário de referências de degrau filtrado.

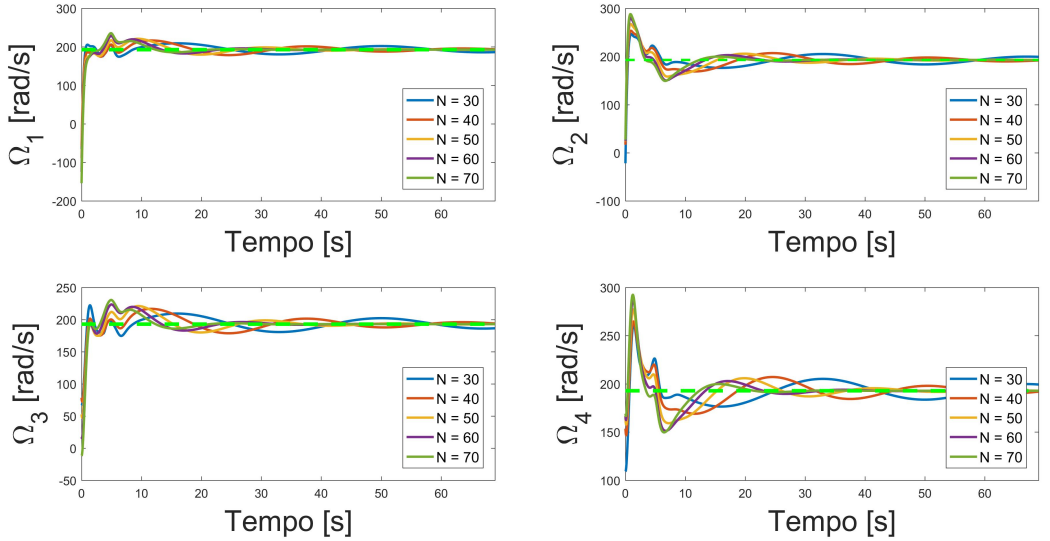

Figura 4.2: Variação das variáveis de controle de acordo com o valor de N adotado para o controlador MPC sem restrições no cenário de referências de degrau filtrado.

Analisando as imagens acima, é possível perceber que o melhor desempenho do controlador MPC sem restrições para z mediante o critério de decisão adotado para este trabalho (menor horizonte de predição que consegue controlar o sistema) ocorre com $N = 40$. Já para o controle da variável de estado ψ, o menor horizonte de predição que cumpre o objetivo é de $N = 50$. Assim, o melhor horizonte de predição para o MPC sem restrições no cenário de referências de degrau filtrado aplicadas a x, y e z seria $N = 40$, mas demandando mais do que o tempo total de simulação (70 s) para estabilização da variável de estado ψ.

Além disso, analisando as figuras é possível perceber que acima de $N = 40$ ocorre um erro no controle da dinâmica em z, o que faz com que $N = 40$ seja ainda mais adequado.

Os mesmos testes são efetuados para o cenário de uma trajetória helicoidal e são apresentados nas Figuras 4.3 - 4.4 abaixo.

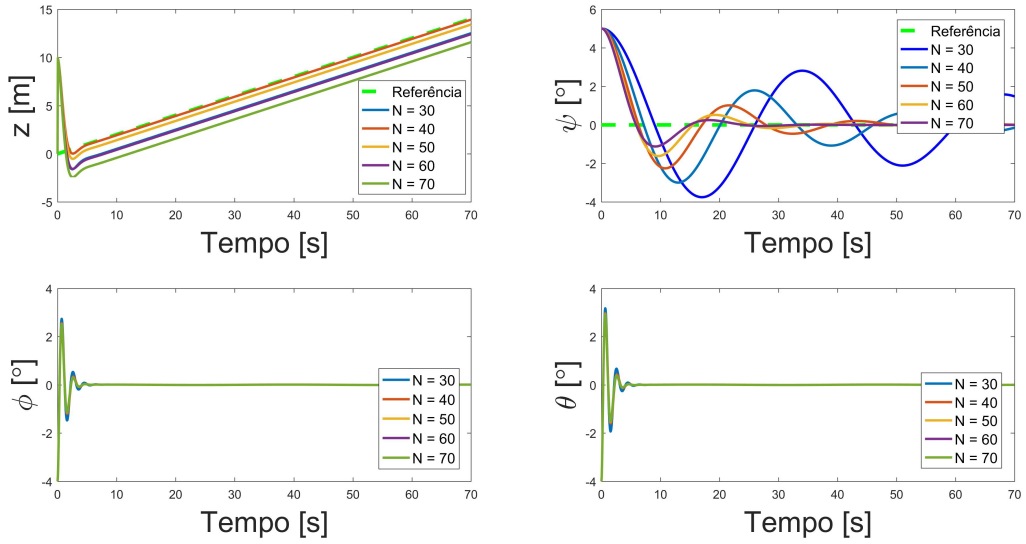

Figura 4.3: Variação das variáveis de estado z e ψ de acordo com o valor de N adotado para o controlador MPC sem restrições no cenário de trajetória helicoidal.

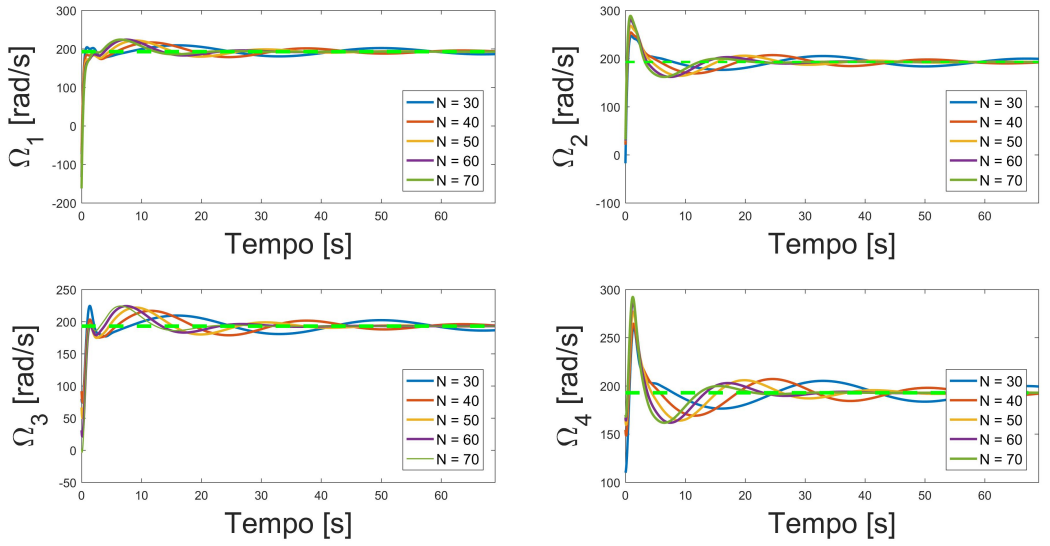

Figura 4.4: Variação das variáveis de controle de acordo com o valor de N adotado para o controlador MPC sem restrições no cenário de trajetória helicoidal.

Para o cenário de trajetória helicoidal, os mesmos resultados se mantém para o MPC sem restrições: N=50 é o menor horizonte de predição adotado que é capaz de controlar o ângulo ψ de maneira adequada, ainda que demore praticamente 50 s para isso. A melhor resposta para a referência atribuida a z é obtida com N = 40, mas esse valor não é capaz de controlar ψ dentro do tempo total de simulação adotado.

Logo, a princípio, o valor de N a ser utilizado no controlador MPC sem restrições em FPGA é N = 40, já que valores acima deste introduzem erros nos cálculos da dinâmica em z.

4.1.2 Resultados da simulação do controlador MPC com restrições em MATLAB

Agora são apresentados os resultados das simulações do MPC com restrições. As Figuras 4.5 - 4.6 apresentam os resultados das simulações em MATLAB variando o horizonte de predição e mantendo o número de iterações constante N_{iter} = 100 para o cenário de aplicação de degraus filtrados como referências para as coordenadas espaciais x, y e z. Lembrando mais uma vez, que são mostrados somente z e os ângulos ψ, ϕ e θ, já que as duas primeiras são as dinâmicas mais difíceis de controlar e as duas últimas são as dinâmicas restringidas do sistema.

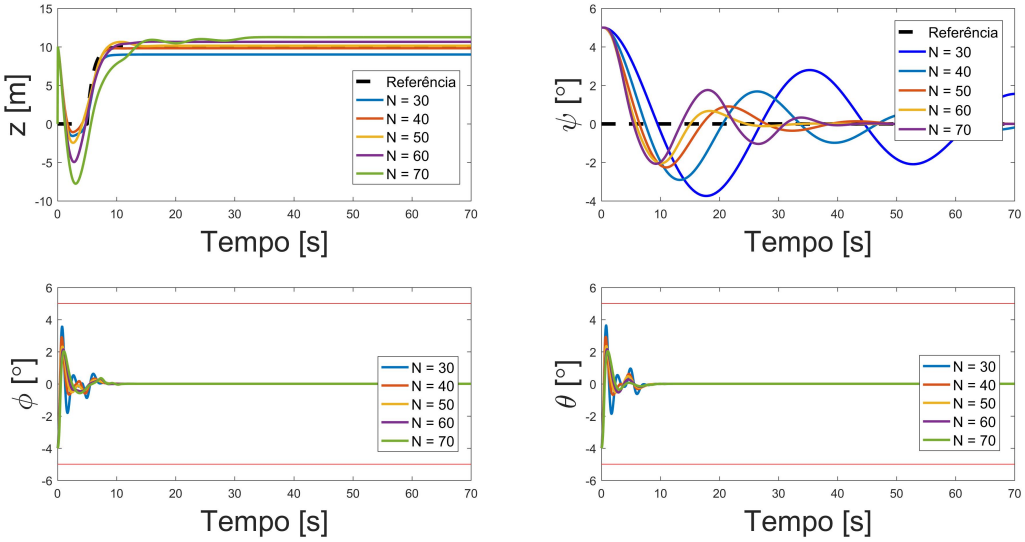

Figura 4.5: Variação das variáveis de estado z e ψ de acordo com o valor de N adotado para o controlador MPC com restrições no cenário de referências de degrau filtrado.

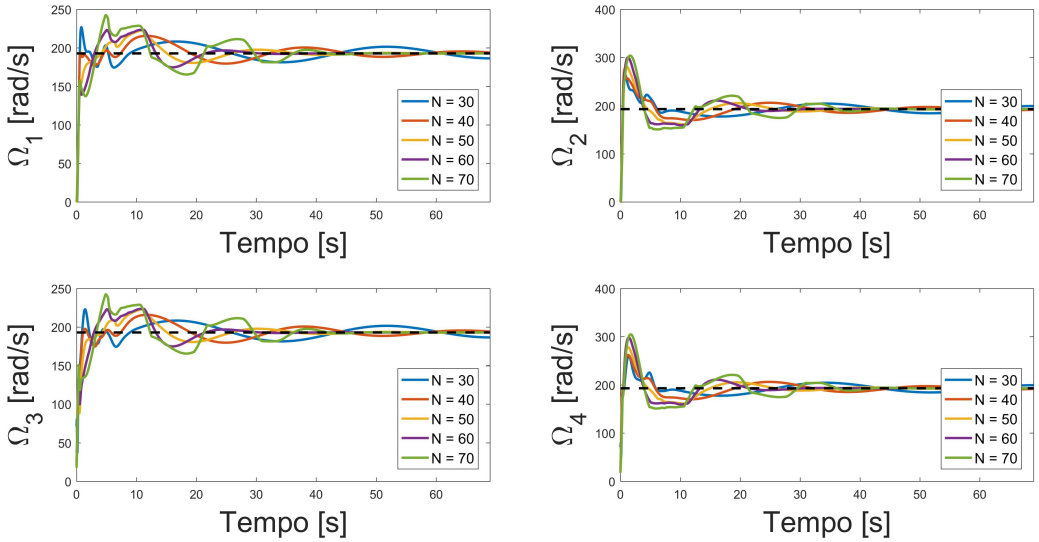

Figura 4.6: Variação das variáveis de controle de acordo com o valor de N adotado para o controlador MPC com restrições no cenário de referências de degrau filtrado.

Assim como no caso sem restrições, o menor horizonte de predição capaz de controlar a variável de estado z foi $N = 40$. Aqui, entretanto, $N = 50$ consegue controlar ambas as variáveis de estado z e ψ, mas demorando muito tempo para estabilização do ângulo ψ. Portanto, o valor de N adequado para este cenário seria $N = 50$.

O mesmo cenário é testado mais uma vez, adotando o menor horizonte de predição que foi capaz de controlar o quadrirrotor nas simulações anteriores ($N = 50$) e variando o número de iterações para verificar como cada dinâmica do quadrirrotor se comporta, resultando nas Figuras 4.7 - 4.8.

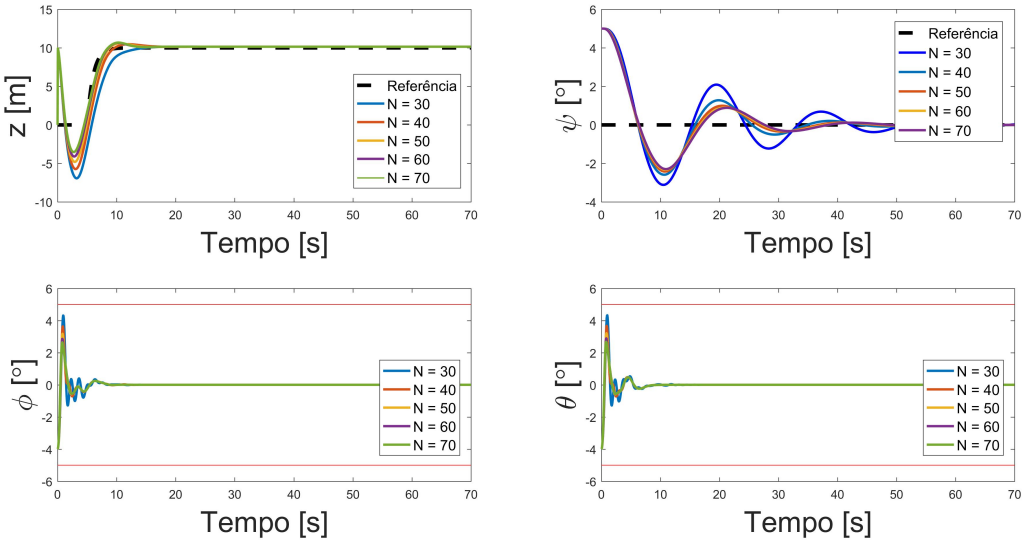

Figura 4.7: Variação das variáveis de estado z e ψ de acordo com o valor de N_{iter} adotado para o controlador MPC com restrições no cenário de referências de degrau filtrado.

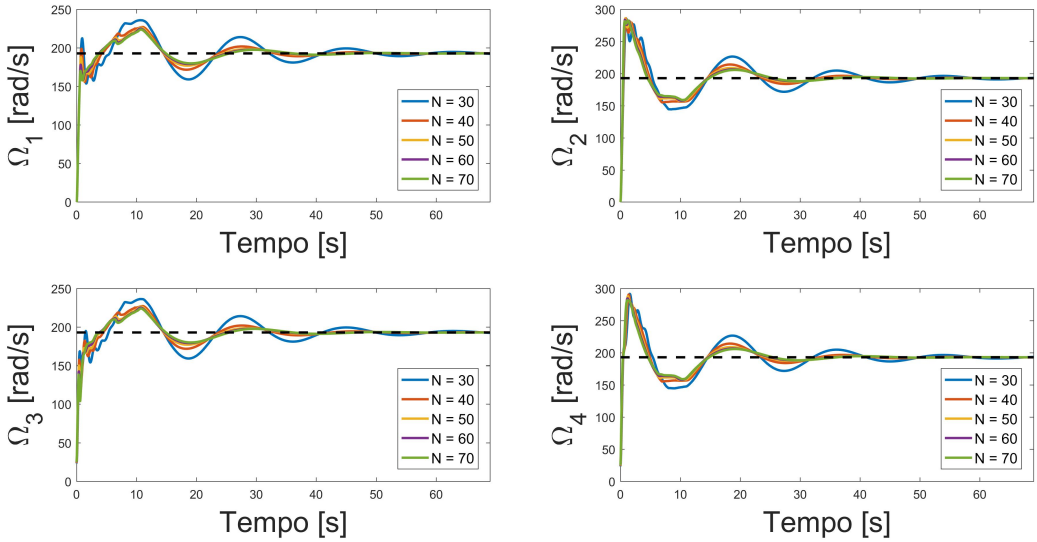

Figura 4.8: Variação das variáveis de controle de acordo com o valor de N_{iter} adotado para o controlador MPC com restrições no cenário de referências de degrau filtrado.

Com os resultados obtidos até aqui, é possível perceber que o número mínimo de iterações que o

solver precisa para conseguir controlar o sistema do quadrirrotor é de N_{iter} = 30 iterações. Portanto, esse é o melhor valor para o controlador desenvolvido em VHDL, uma vez que quanto menor o número de iterações, menor o tempo de cálculo das variáveis de controle ótimas.

Os testes acima foram repetidos para o cenário de uma trajetória helicoidal. Primeiramente, são apresentados nas Figuras 4.9 - 4.10 os resultados obtidos para cada variável de estado do quadrirrotor de acordo com o horizonte de predição adotado, mantendo N_{iter} = 100.

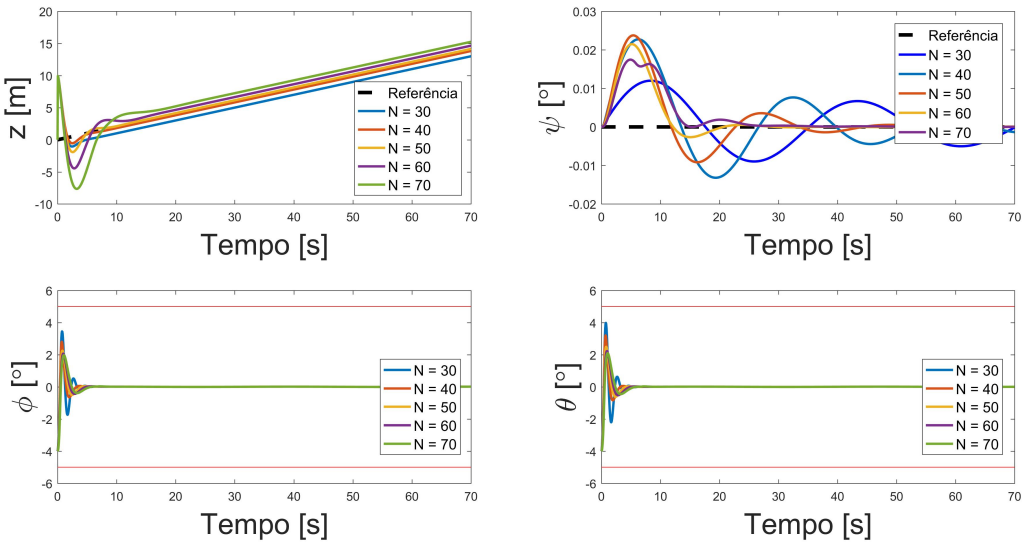

Figura 4.9: Variação das variáveis de estado z e ψ de acordo com o valor de *N* adotado para o controlador MPC com restrições no cenário de trajetória helicoidal.

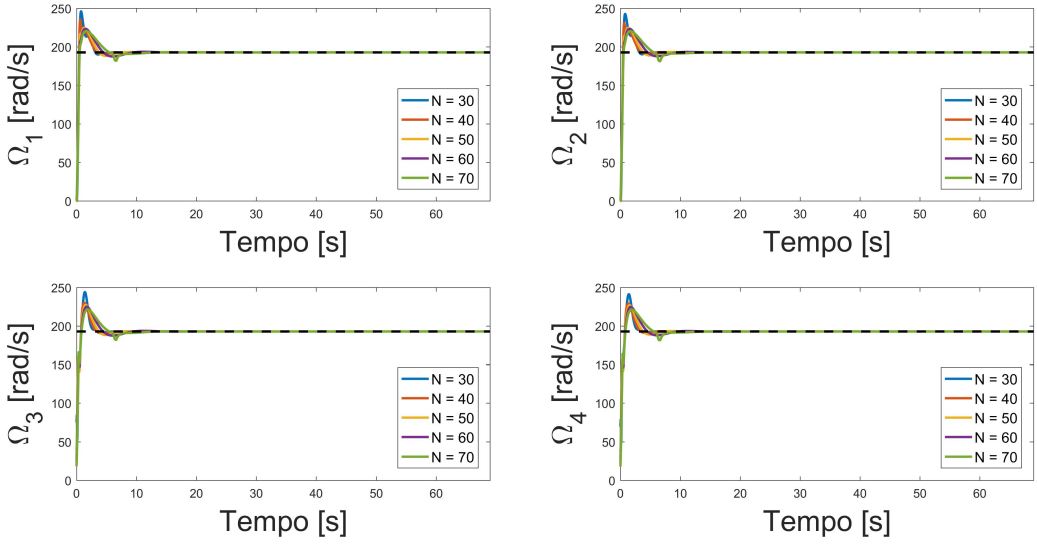

Figura 4.10: Variação das variáveis de controle de acordo com o valor de *N* adotado para o controlador MPC com restrições no cenário de trajetória helicoidal.

De maneira análoga ao outro cenário, adotou-se o menor horizonte de predição que foi capaz de controlar o quadrirrotor com referências de trajetória helicoidal (que também foi *N*=50), dessa vez variando-se

o número de iterações do *solver* de QP. Os resultados são mostrados nas Figuras 4.11 - 4.12.

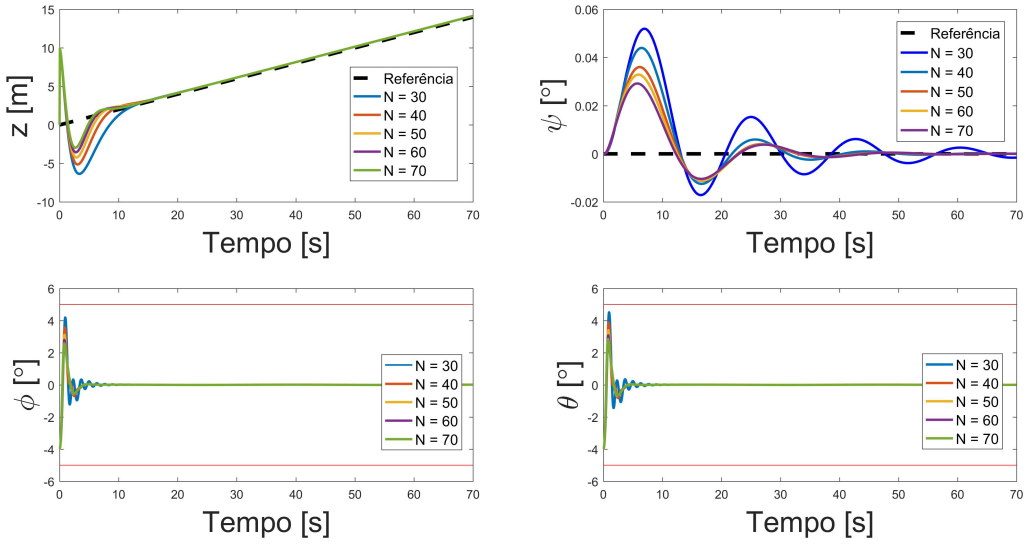

Figura 4.11: Variação das variáveis de estado z e ψ de acordo com o valor de N_{iter} adotado para o controlador MPC com restrições no cenário de referências de degrau filtrado.

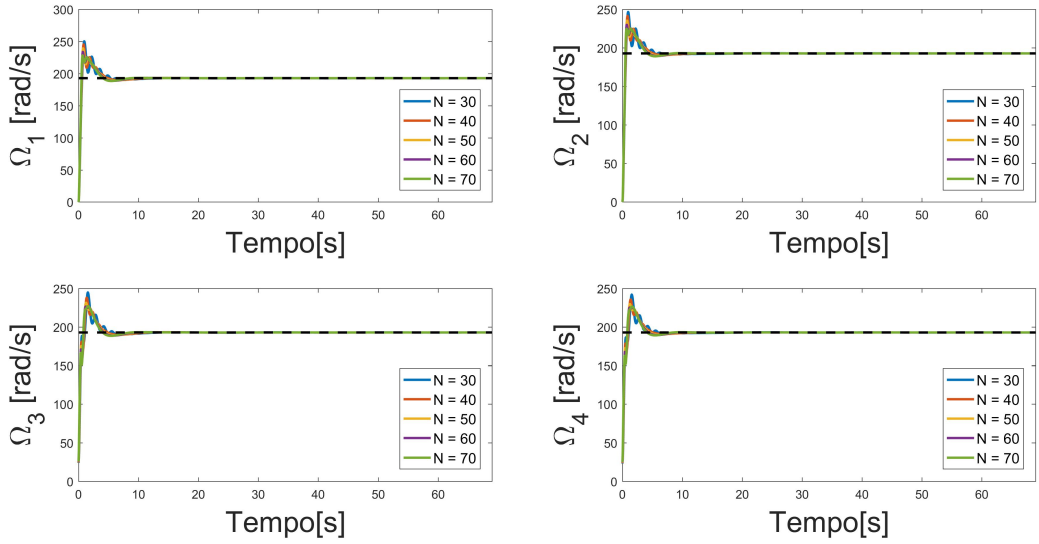

Figura 4.12: Variação das variáveis de controle de acordo com o valor de N_{iter} adotado para o controlador MPC com restrições no cenário de referências de degrau filtrado.

As conclusões dos testes do MPC com restrições mediante o cenário de trajetória helicoidal são os mesmos obtidos com as referências de degrau filtrado: o menor horizonte de predição que é capaz de controlar o quadrirrotor de maneira satisfatória é $N = 50$, com no mínimo 30 iterações do *solver* PGE como algoritmo de otimização.

Assim, pode-se concluir que, os melhores parâmetros para os controladores implementados em *hardware* são $N = 50$ e, no caso do MPC com restrições, Niter = 30. Caso não seja necessário controlar ψ, $N = 40$ é a melhor opção para utilizar no MPC com restrições em *hardware*, uma vez que é o menor horizonte

de predição capaz de controlar a variável de estado z.

4.2 Resultados do desenvolvimento de arquitetura em VHDL do controlador MPC

Os primeiros resultados apresentados neste trabalho referentes aos controladores MPC correspondem ao consumo de recursos de *hardware* necessários para a implementação de cada um no *kit* FPGA KC-705 utilizado.

A Tabela 4.1 apresenta os resultados de utilização de recursos para cada módulo que compõem o MPC, seja ele com ou sem restrições, mostrando a quantidade de recursos que são necessários para a implementação dos mesmos quando diferentes valores de *N* são adotados.

Tabela 4.1: Consumo de recursos de *hardware* do *kit* de FPGA KC-705 necessários para implementar cada módulo que compõem o MPC com e sem restrições de acordo com o valor de *N* adotado e o número total de *bits* utilizados em ponto flutuante.

N	***Bits***	**Módulo**	**LUT**	**LUTRAM**	**FF**	**BRAM**	**DSP**
40	27	*matrix_ mult*	16312 (8.00%)	–	3200 (0.79%)	–	40 (4.76%)
		matrix_ add	12208 (5.99%)	–	2000 (0.49%)	–	–
		unconstrained_ mpc	859 (0.42%)	–	904 (0.22%)	–	–
		model_ simulator	30155 (14.80%)	–	7823 (1.92%)	–	40 (4.76%)
		compute _ MPC_ matrices	18331 (8.99%)	–	13411 (3.29%)	–	–
		solver_ sat_ pen	18498 (9.08%)	–	9511 (2.33%)	–	–
40	32	*matrix_ mult*	18490 (9.07%)	–	3800 (0.93%)	–	80 (9.52%)
		matrix_ add	15648 (7.68%)	–	2400 (0.59%)	–	–
		model_ simulator	36137 (17.73%)	–	9292 (2.28%)	–	80 (9.52%)
		compute _ MPC_ matrices	16342 (8.02%)	–	15864 (3.89%)	–	–
		solver_ sat_ pen	18479 (9.07%)	–	11214 (2.75%)	–	–
50	27	*matrix_ mult*	20362 (9.99%)	–	4000 (0.98%)	–	50 (5.95%)
		matrix_ add	15260 (7.49%)	–	2500 (0.61%)	–	–
		unconstrained_ mpc	867 (0.43%)	–	908(0.23%)	–	–
		model_ simulator	37447 (18.37%)	–	9123 (2.24%)	–	50 (5.95%)
		compute _ MPC_ matrices	24288 (11.92%)	–	16628 (4.08%)	–	–
		solver_ sat_ pen	22593 (11.09%)	–	11114 (2.73%)	–	–
50	32	*matrix_ mult*	23080 (11.32%)	–	4750 (1.17%)	–	100 (11.90%)
		matrix_ add	19560 (9.60%)	–	3000 (0.74%)	–	–
		model_ simulator	44879 (22.02%)	–	10842 (2.66%)	–	100 (11.90%)
		compute _ MPC_ matrices	22139 (10.86%)	–	19681 (4.83%)	–	–
		solver_ sat_ pen	14264 (7.00%)	–	13113 (3.22%)	–	–
60	27	*matrix_ mult*	24352 (11.95%)	–	4800 (1.18%)	–	60 (7.14%)
		matrix_ add	18312 (8.99%)	–	3000 (0.74%)	–	–
		unconstrained_ mpc	867 (0.43%)	–	908(0.23%)	–	–
		model_ simulator	44739 (21.95%)	–	10423 (2.56%)	–	60 (7.14%)
		compute _ MPC_ matrices	27744 (13.61%)	–	19886 (4.88%)	–	–
		solver_ sat_ pen	24279 (11.91%)	–	12724 (3.12%)	–	–
60	32	*matrix_ mult*	27670 (13.58%)	–	5700 (1.40%)	–	120 (14.29%)
		matrix_ add	23472 (11.52%)	–	3600 (0.88%)	–	–
		model_ simulator	53617 (26.31%)	–	12392 (3.04%)	–	120 (14.29%)
		compute _ MPC_ matrices	25410 (12.47%)	–	23557 (5.78%)	–	–
		solver_ sat_ pen	16562 (8.13%)	–	15046 (3.96%)	–	–

Já a Tabela 4.2 apresenta o mesmo tipo de informação, mas dessa vez referente à arquitetura completa do MPC sem restrições e com restrições, com todos os módulos integrados.

Tabela 4.2: Consumo de recursos de *hardware* do *kit* de FPGA KC-705 necessários para implementar os controladores MPC sem restrições e com restrições de acordo com o valor de *N* adotado e o número total de *bits* utilizados em ponto flutuante.

N	***Bits***	**Controlador**	**LUT**	**LUTRAM**	**FF**	**BRAM**	**DSP**
40	27	Sem restrições	60392 (29.63 %)	–	14256 (3.50%)	6 (1.35%)	80 (9.52%)
40	27	Com restrições	105521 (51.78%)	216 (0.34%)	38754 (9.51%)	156 (35.06%)	88 (10.48%)
40	32	Com restrições	108912 (53.44%)	256 (0.40%)	45842 (11.25%)	187.5 (42.13%)	176 (20.95%)
50	27	Sem restrições	74645 (36.63%)	–	16555 (4.06%)	6 (1.35%)	100 (11.90%)
50	27	Com restrições	128109 (62.86%)	216 (0.34%)	46187 (11.33%)	195 (43.82%)	108 (12.86%)
50	32	Com restrições	135746 (66.61%)	256 (0.40%)	55600 (13.64%)	232 (52.13%)	216 (25.71%)
60	27	Sem restrições	89229 (43.78%)	–	19155 (4.70%)	6 (1.35%)	120 (14.28%)
60	27	Com restrições	151443 (74.31%)	216 (0.34%)	54883 (13.46%)	243 (54.61%)	128 (15.24%)
60	32	Com restrições	158152 (77.60%)	256 (0.40%)	65138 (15.98%)	291.5 (65.61%)	256 (30.48%)

As tabelas anteriores analisam as soluções de controladores MPC desenvolvidas com três valores distintos de *N*: 40, 50 e 60. Como apresentado na seção anterior, $N = 50$ é o valor de horizonte de predição que apresentou os melhores resultados. $N = 40$ apresentou bons resultados também, e a princípio é uma opção viável para ser implementado, por exemplo, em casos em que ψ é mantido constante ou regulado em $0°$. $N = 60$ não apresenta resultados tão bons quanto $N = 40$ e $N = 50$, mas ajuda a entender características importantes referentes ao consumo de *hardware* necessários para implementação de cada controlador.

A primeira relação que deve ser feita é de que, quanto maior o valor de *N* adotado, maior a quantidade de recursos necessários para implementar o controlador. Isso é algo que já era esperado, uma vez que quanto maior o valor de *N*, maiores também são as dimensões das matrizes que descrevem o problema de controle, tanto para o MPC sem restrições quanto para o MPC com restrições.

Além disso, quanto maior o valor de *N* adotado, mais DSPs são utilizados, pois mais multiplicadores são colocados em paralelo para a construção da arquitetura do módulo *matrix_ mult*. Esse módulo utiliza sempre *N* DSPs e também é constituido por *N* somadores, que em conjunto, realizam a operação de acumulação que é necessária para a realização de multiplicações matriciais.

Outra relação que pode ser feita é que, quanto maior o número total de *bits* utilizados para representar os dados em ponto flutuante, mais recursos de *hardware* são necessários para implementar os controladores. Isso também era esperado, uma vez que, como dito anteriormente, o valor de 27 *bits* otimiza a utilização de recursos de *hardware* em FPGAs da *Xilinx*, porque os DPSs dos *kits* mais atuais possuem 18 *bits* de entrada, mesma quantidade de *bits* que compõem a mantissa dos dados com 27 *bits* no total.

Quando utiliza-se 32 *bits* para representar os dados em pontos flutuantes, a mantissa passa a ser composta por 23 *bits*, de forma que mais DSPs são necessários para implementar cada multiplicador. Não por acaso, o consumo de DSPs dobra para cada controlador, quando comparadas as soluções de 27 *bits* e 32 *bits*.

No entanto, a grande vantagem da utilização de 32 *bits* é a diminuição de erro de arredondamento nas operações realizadas, bem como a capacidade de armazenar uma faixa dinâmica de valores maior que a solução em 27 *bits*. Isso é melhor evidenciado nas próximas seções, em que os resultados das simulações dos controladores embarcados em FPGA são apresentados.

Analisando a Tabela 4.1, é possível perceber que o módulo que mais consome recursos para sua im-

plementação é o módulo *model_ simulator*. Isso se dá porque, como já foi explicado anteriormente, este módulo foi concebido com multiplicadores e somadores de matrizes próprios, ou seja, possui em sua composição os módulos *matrix_ mult* e *matrix_ add*. Isso consome uma quantidade maior de recursos do *kit* de maneira geral, mas como isso não fez com que a capacidade do mesmo fosse excedida, foi possível evitar lógica adicional de multiplexação, que ocorre, por exemplo, quando o módulo *matrix_ mult* é compartilhado entre os módulos *compute_ MPC_ matrices* e *solver_ sat_ pen*, no caso do MPC com restrições.

É por isso que, para cada valor de *N* diferente adotado, utilizando 27 *bits* como precisão em ponto flutuante, o MPC consome 2**N* DSPs (*N* DSPs necessários para implementar o módulo *matrix_ mult* do controlador em si e mais *N* DSPs para implementar o módulo *matrix_ mult* interno ao módulo *model_ simulator*). Os 8 DSPs adicionais correspondem a 8 multiplicadores instanciados no controlador para realizarem pequenas multiplicações escalares necessárias para calcular coeficientes e fatores exigidos pelo algoritmo PGE implementado no módulo *solver_ sat_ pen*.

Para 32 *bits*, esses valores dobram e o consumo de DSPs é dado por 4**N* + 16.

Assim, como o módulo *model_ simulator* possui seus próprios recursos de operação matricial, e está de certa forma, isolado do controlador MPC, uma estimativa de consumo de recursos somente dos controladores MPC em si pode ser feita, resultando na Tabela 4.3. Para isso, os valores de cada tipo de recurso consumido por *model_ simulator* são subtraídos dos recursos correspondentes em cada módulo que compõem os controladores.

Tabela 4.3: Estimativa de consumo de recursos de *hardware* do *kit* de FPGA KC-705 necessários para implementar os controladores MPC sem restrições e com restrições de acordo com o valor de *N* adotado e o número total de *bits* utilizados em ponto flutuante, sem levar em consideração o módulo *model_ simulator*.

N	***Bits***	**Controlador**	**LUT**	**LUTRAM**	**FF**	**BRAM**	**DSP**
40	27	Sem restrições	30237 (14.70 %)	–	6433 (1.58%)	6 (1.35%)	40 (4.76%)
40	27	Com restrições	75366 (36.98%)	216 (0.34%)	30931 (7.59%)	156 (35.06%)	48 (5.71%)
40	32	Com restrições	72775 (35.71%)	256 (0.40%)	36550 (8.97%)	187.50 (42.13%)	96 (11.43%)
50	27	Sem restrições	37198 (18.25%)	–	7432 (1.83%)	6 (1.35%)	50 (5.95%)
50	27	Com restrições	90662 (44.49%)	216 (0.34%)	37064 (9.09%)	195 (43.82%)	58 (6.90%)
50	32	Com restrições	90867 (44.59%)	256 (0.40%)	44758 (10.98%)	232 (52.13%)	132 (15.70%)
60	27	Sem restrições	44490 (21.83%)	–	8732 (2.14%)	6 (1.35%)	60 (7.14%)
60	27	Com restrições	106704 (52.36%)	216 (0.34%)	44460 (10.90%)	243 (54.61%)	68 (8.09%)
60	32	Com restrições	104535 (51.29%)	256 (0.40%)	52746 (12.94%)	291.50 (65.51%)	136 (16.19%)

Essa é uma estimativa de quanto cada controlador consumiria em *hardware* do FPGA ao ser implementado, caso o simulador de modelo fosse implementado externamente ao FPGA, como em um bloco do *Simulink*, por exemplo, que se comunicasse com o FPGA utilizando a estrutura do *HDL Verifier*. Trata-se de uma estimativa porque, sem o *model_ simulator* a integração entre os demais módulos seria um pouco modificada, o que resultaria em resultados diferentes dos apresentados na tabela acima.

Como esperado, o MPC sem restrições consome bem menos recursos de *hardware* do FPGA do que o MPC com restrições. A lei de controle do MPC sem restrições é bem mais simples, exigindo pouco armazenamento de matrizes, característica que não repete no MPC com restrições. O aumento de consumo de recursos não obedece uma proporção fixa quando comparadas as soluções com e sem restrições, mas são de aproximadamente 2.44 vezes mais LUTs, 4.96 vezes mais FFs, 33 vezes mais BRAMs e 1.16 vezes mais DSPs para implementação do MPC com restrições.

O número de BRAMs utilizados corresponde ao recurso que mais cresce em utilização para a implementação do MPC com restrições, por conta da maior quantidade de matrizes e memórias que estão envolvidas nesse processo, especialmente as matrizes associadas ao *solver* de QP.

4.3 Resultados da simulação da arquitetura em VHDL do controlador MPC

Esta seção apresenta os resultados das simulações dos controladores MPC implementados em *hardware*. Assim como no caso dos testes em MATLAB, os resultados são apresentados em duas subseções: uma para o MPC sem restrições e uma para o MPC com restrições.

4.3.1 Resultados da simulação da arquitetura em VHDL do controlador MPC sem restrições

Com os resultados obtidos anteriormente, algumas considerações precisam ser feitas antes da apresentação dos resultados dos controladores em *hardware*.

O primeiro deles refere-se a algo que se mostrou importante durante os testes realizados com os controladores em FPGA. De maneira geral, o *HDL Verifier* facilita a integração dos resultados obtidos com o MATLAB, mas torna as simulações muito lentas. Na verdade, isso impacta mais nas simulações do MPC com restrições, como será abordado na próxima subseção deste trabalho, mas por conta disso, as simulações realizadas para o MPC sem restrições assumem duas condições:

1. O tempo total de simulação foi reduzido para 20 s;
2. $\psi_0 = 0°$, ou seja, ψ é regulado em $0°$ durante toda a simulação.

Isso foi feito porque o objetivo aqui é validar as arquiteturas desenvolvidas. O MPC se mostrou capaz de controlar ψ, mas para que mais testes possam ser feitos, de maneira mais eficiente, esse ângulo é mantido em $0°$, o que garante que as demais saídas reguladas do sistema estabilizem e rastream as trajetórias adequadamente.

Tendo em vista essas questões, conforme apresentado na Subseção 4.1.1, o valor de N = 40 pode ser adotado aqui, pois é capaz de controlar todas as dinâmicas de maneira satisfatória (com exceção de ψ) no tempo total de simulação adotado. Os controladores serão testados utilizando N = 40 e N = 50, para que seja possível fazer comparações entre eles.

As Figuras 4.13 - 4.16 abaixo apresentam os resultados das simulações com N = 40 para o MPC sem restrições, realizadas utilizando o *HDL Verifier* no ambiente *Simulink* do MATLAB, conforme explicado no capítulo anterior. Além disso, esses resultados são comparados com os resultados obtidos em MATLAB apresentados anteriormente.

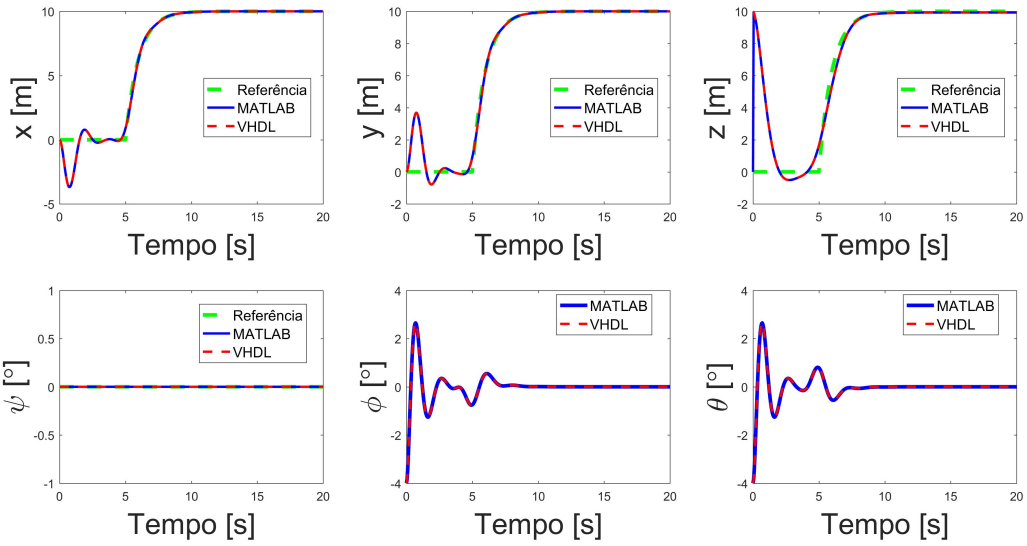

Figura 4.13: Simulação e comparação do quadrirrotor controlado utilizando o MPC sem restrições com a simulação em MATLAB: saídas do sistema para $N = 40$ e referências de degrau filtrado.

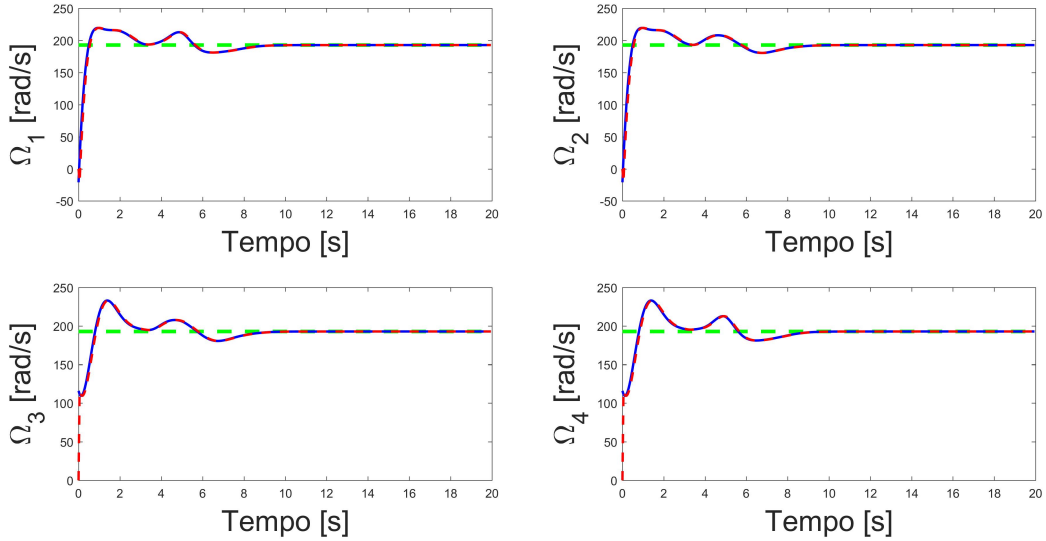

Figura 4.14: Simulação e comparação do quadrirrotor controlado utilizando o MPC sem restrições (vermelho) com a simulação em MATLAB (azul): variáveis de controle para $N = 40$ e referências de degrau filtrado.

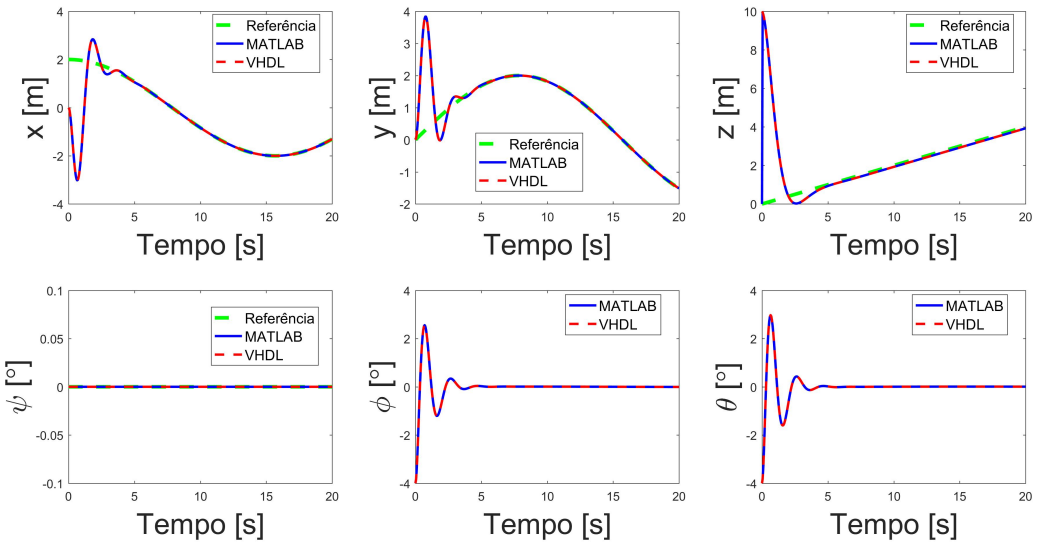

Figura 4.15: Simulação e comparação do quadrirrotor controlado utilizando o MPC sem restrições com a simulação em MATLAB: saídas do sistema para $N = 40$ e trajetória helicoidal.

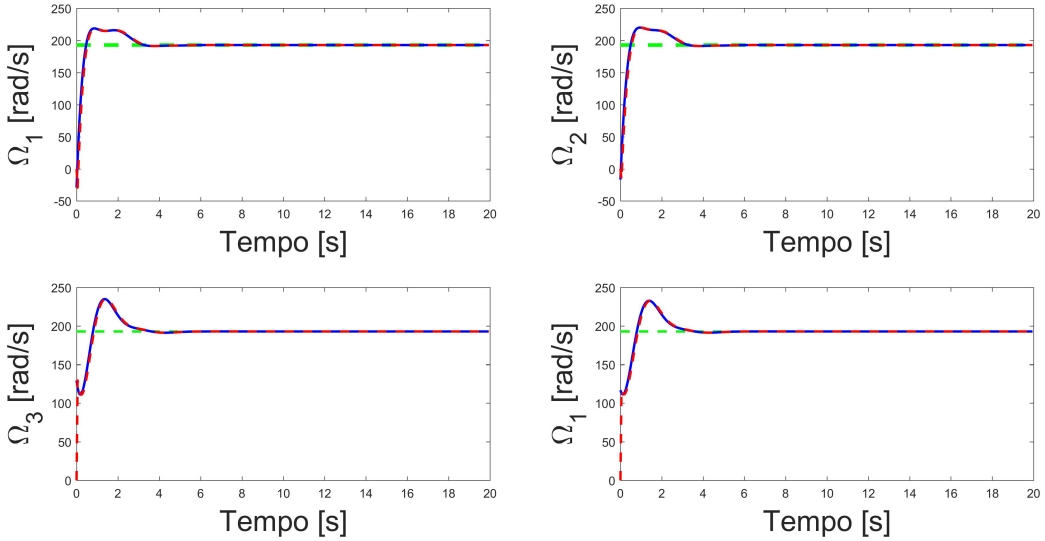

Figura 4.16: Simulação e comparação do quadrirrotor controlado utilizando o MPC sem restrições (vermelho) com a simulação em MATLAB (azul): variáveis de controle para $N = 40$ e trajetória helicoidal.

A primeira característica que pode ser observada com base nas figuras acima é de que o MPC sem restrições em VHDL se comporta de maneira muito próxima ao MPC sem restrições simulado no MATLAB. Isso valida não somente o controlador em si, mas os módulos de multiplicação e soma matricial desenvolvidos em VHDL, além do módulo *model_ simulator*.

Além disso, com base nas simulações no *HDL Verifier* é possível obter a informação de quanto tempo o sistema demora para calcular as variáveis de controle do sistema por meio da saída *ready* do sistema de controle: aproximadamente 10μs para uma frequência de *clock* do FPGA de 50MHz.

Esse é um valor bem abaixo do tempo de amostragem do sistema (50 ms), por conta da formulação sim-

ples do MPC sem restrições: como as matrizes podem ser calculadas todas *offline*, o cálculo das variáveis de controle é obtido com três multiplicações matriciais ($-\mathbf{K_N}.x$, $\mathbf{G_N}.y_{ref}$ e $\mathbf{L_N}.u_d$) e a soma entre elas. Como as referências e o valor de u_d já são conhecidos de antemão, as próprias multiplicações $\mathbf{G_N}.y_{ref}$ e $\mathbf{L_N}.u_d$ podem ser feitas *offline*, o que acelera ainda mais o cálculo das variáveis de controle do sistema.

Agora são apresentados os resultados dos mesmos testes executados anteriormente para N = 50. Estes são mostrados nas Figuras 4.17 - 4.20.

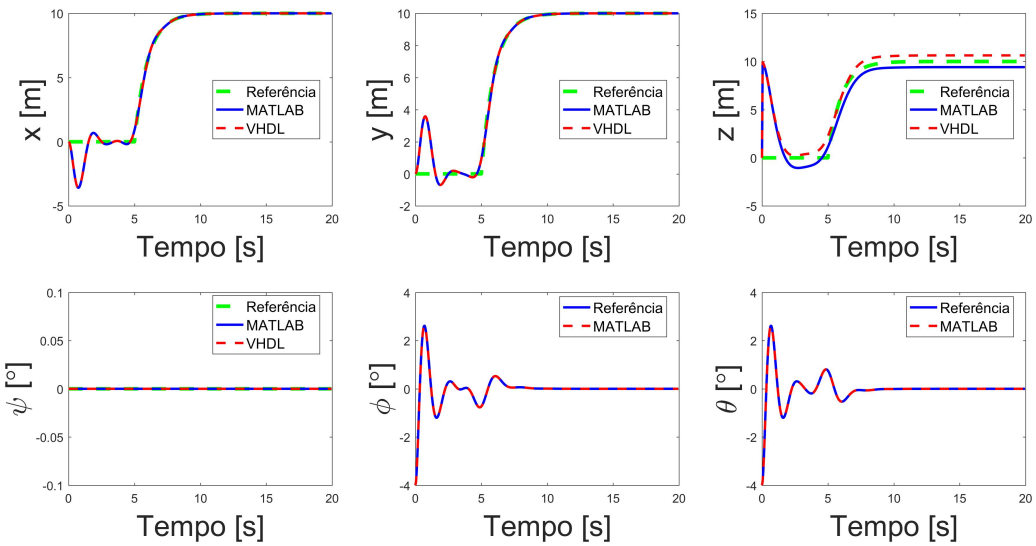

Figura 4.17: Simulação e comparação do quadrirrotor controlado utilizando o MPC sem restrições com a simulação em MATLAB: saídas do sistema para N = 50 e referências de degrau filtrado.

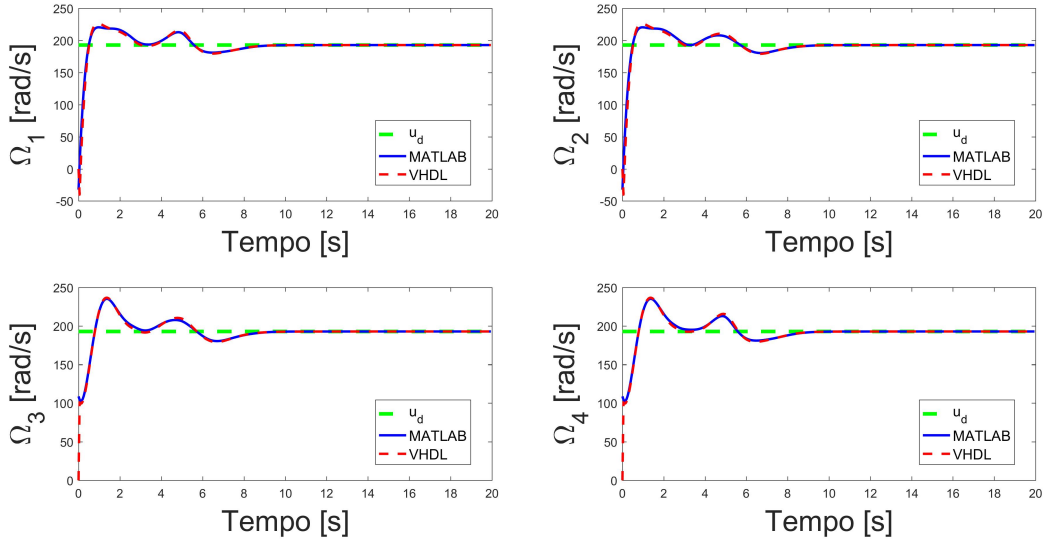

Figura 4.18: Simulação e comparação do quadrirrotor controlado utilizando o MPC sem restrições com a simulação em MATLAB: variáveis de controle para N = 50 e referências de degrau filtrado.

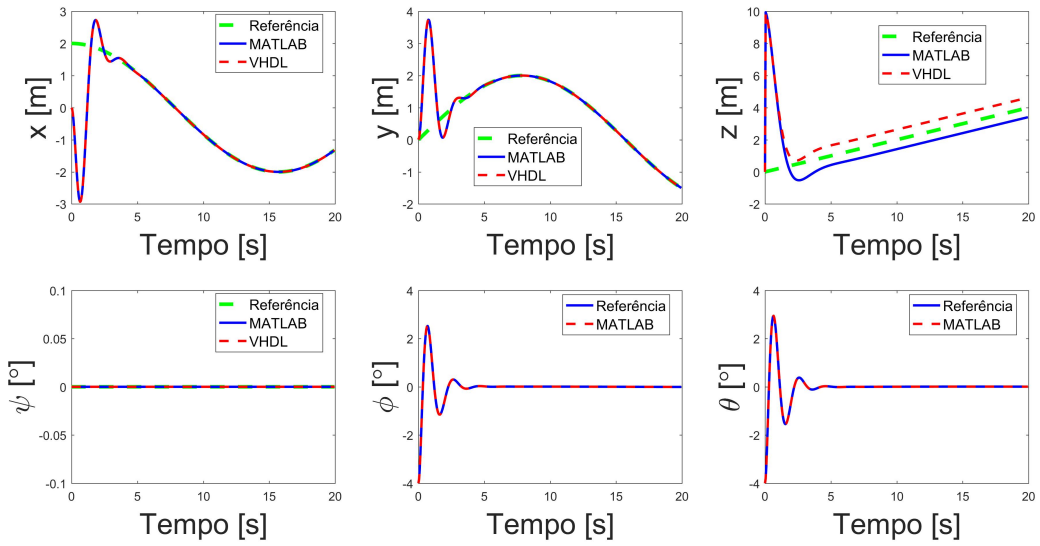

Figura 4.19: Simulação e comparação do quadrirrotor controlado utilizando o MPC sem restrições com a simulação em MATLAB: saídas do sistema para $N = 50$ e trajetória helicoidal.

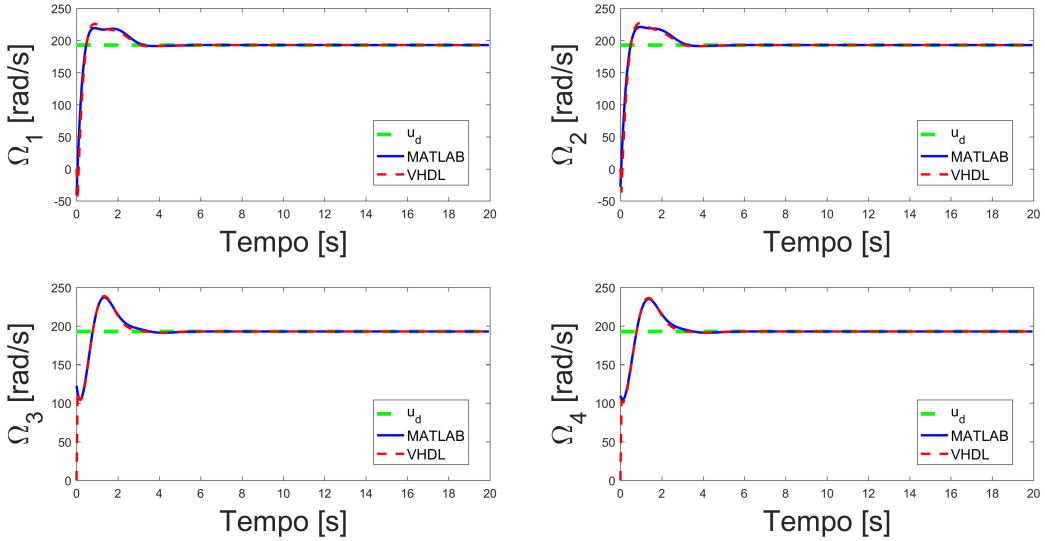

Figura 4.20: Simulação e comparação do quadrirrotor controlado utilizando o MPC sem restrições com a simulação em MATLAB: variáveis de controle para $N = 50$ e trajetória helicoidal.

Mais uma vez, o MPC sem restrições se mostrou bem próximo dos valores da simulação em MATLAB. Assim como nas simulações em MATLAB, ocorre um erro entre a variável de estado z e a sua referência quando N é maior que 40, o que comprova que os resultados obtidos são satisfatórios.

O paralelismo da solução sem restrições é de tal forma que o tempo de cálculo do controlador para $N = 50$ é o mesmo da solução $N = 40$: 10μs para uma frequência de *clock* de 50 MHz. Isso parece estranho a princípio, mas como as matrizes do sistema sem restrições possuem dimensões menores do que o cenário com restrições, o sistema é capaz de calcular as multiplicações e as somas matricias no mesmo tempo, porque de acordo com o valor de N utilizado, são instanciados N multiplicadores e N somadores

em paralelo, que sempre são capazes de calcular as matrizes resultantes no mesmo intervalo de tempo.

Assim, o melhor resultado é obtido com $N = 40$, pois consegue-se um menor erro no controle da variável de estado z e um menor consumo de recursos de *hardware* do FPGA.

Por conta do consumo de recursos de *hardware* (apresentado na Tabela 4.3) utilizados pelas soluções sem restrições, pode parecer que o MPC sem restrições é uma solução melhor. De fato, é um tipo de controlador que pode apresentar resultados muito bons para situações em que as restrições não são atingidas, como nos casos testados. Entretanto, ao adotar-se referências muito parecidas com os casos testados, mas com degrau filtrado de valor final 50, por exemplo, as restrições não são respeitadas, conforme pode ser visto nas Figuras 4.21 e 4.22.

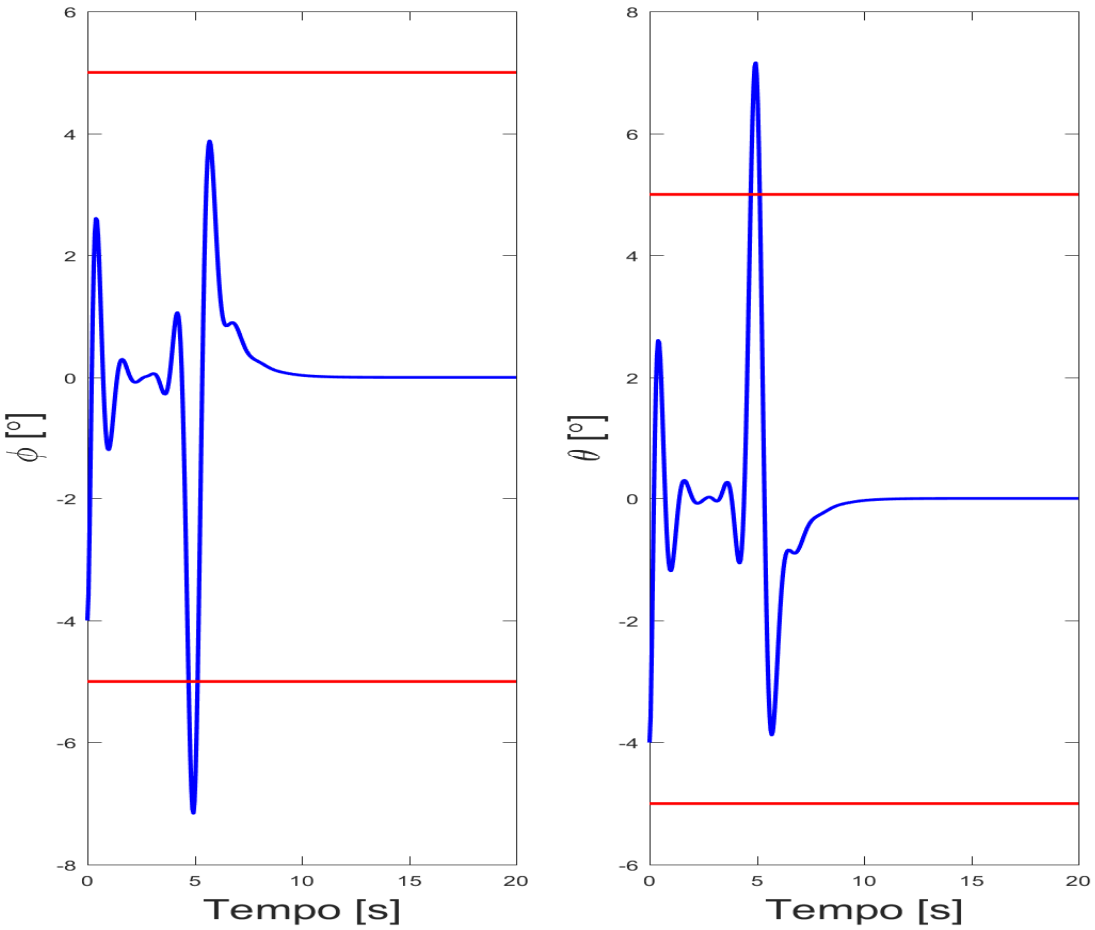

Figura 4.21: Resposta das saídas restringidas do MPC sem restrições para referências de degrau filtrado com valor final igual a 50. Restrições em vermelho.

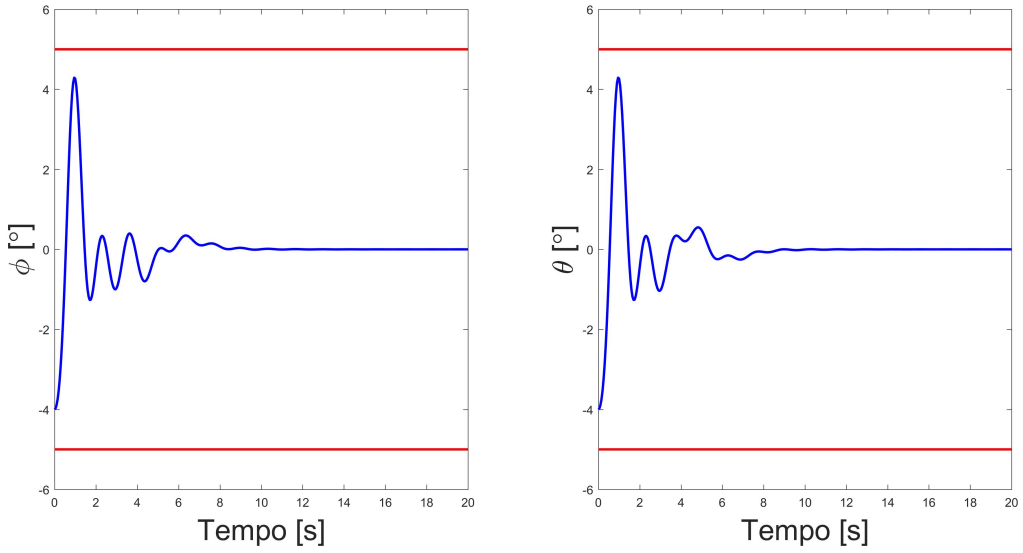

Figura 4.22: Resposta das saídas restringidas do MPC com restrições para referências de degrau filtrado com valor final igual a 50. Restrições em vermelho.

4.3.2 Resultados da simulação da arquitetura em VHDL do controlador MPC com restrições

Assim como no caso do MPC sem restrições, os parâmetros utilizados para o MPC com restrições são baseados nos resultados dos testes da Seção 4.1. O controlador será testado utilizando horizontes de predição N = 40 e N = 50, por conta das suposições que foram apresentadas na subseção anterior, para os cenários de referência de degrau filtrado e trajetória helicoidal. Além disso, o número de iterações do *solver* de QP adotado será de $N_{iter} = 30$, também em conformidade com os resultados apresentados na Seção 4.1.

Entretanto, antes de apresentar os resultados destas simulações é necessário explicar mais alguns detalhes. Foi visto na seção de resultados da simulação do MPC sem restrições que as suposições de menor tempo de simulação e manter ψ em 0° durante a simulação foram adotadas por conta da lentidão nas simulações utilizando o *HDL Verifier*. Isso se torna impactante de fato no caso do MPC com restrições: por se tratar de uma arquitetura bem mais complexa que o MPC sem restrições, a simulação no *HDL Verifier* demora ainda mais tempo para ser concluida. Por exemplo, tomando-se uma simulação com tempo total de 20s, mais de um dia é necessário para completá-la.

Por isso, e mais uma vez para tentar tornar os testes mais eficientes e rápidos, optou-se por adotar uma outra abordagem para a obtenção dos resultados apresentados aqui: realizar simulações comportamentais diretamente no *Vivado*, *software* padrão para desenvolvimento da *Xilinx*. Assim, as simulações se tornam mais rápidas, apesar de ainda demorarem bastante tempo, e o *HDL Verifier* é utilizado somente para verificar o tempo de cálculo do controlador embarcado em FPGA, informação que não está disponível na etapa de simulação comportamental no *Vivado*. No caso do tempo de execução do controlador a simulação pode ser bem menor (no mínimo, um intervalo igual a dois períodos de amostragem, que no caso do quadrirrotor resulta em um tempo mínimo de 100 ms), e portanto o *HDL Verifier* é capaz de obter essa informação com

o algoritmo embarcado em FPGA.

A simulação comportamental do *Vivado* gera resultados bem próximos aos obtidos na simulação do *Simulink*, mas, obviamente, não leva em consideração características de temporização do sistema. Para comprovar que o *Vivado* pode ser utilizado para tal fim, as Figuras 4.23 e 4.24 apresentam o resultado de uma simulação comportamental do MPC sem restrições realizada anteriormente, comparando os resultados com os dados obtidos na simulação utilizando *HDL Verifier*, mostrado na subseção anterior.

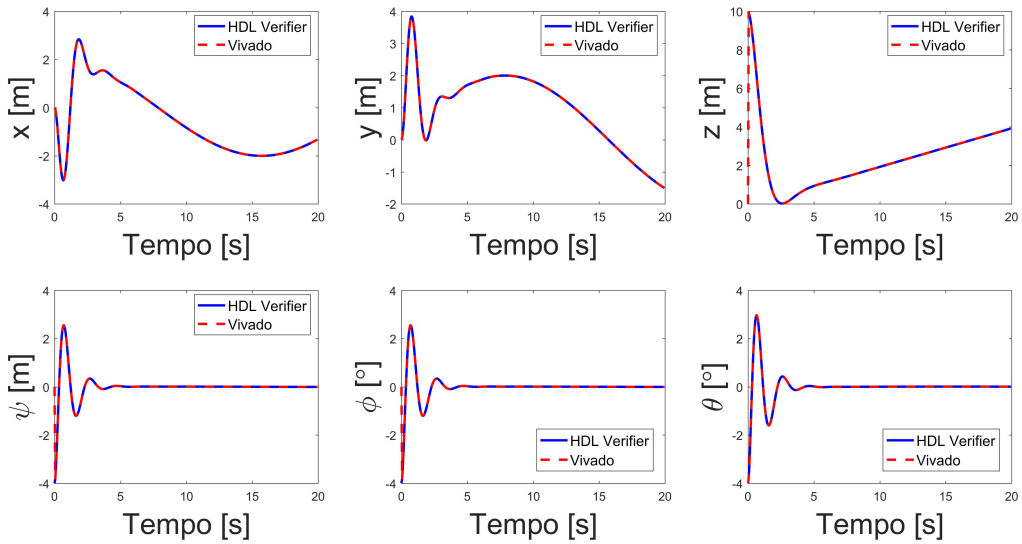

Figura 4.23: Comparação entre cálculo de saídas do quadrirrotor utilizando *HDL Verifier* e simulação comportamental no *Vivado*.

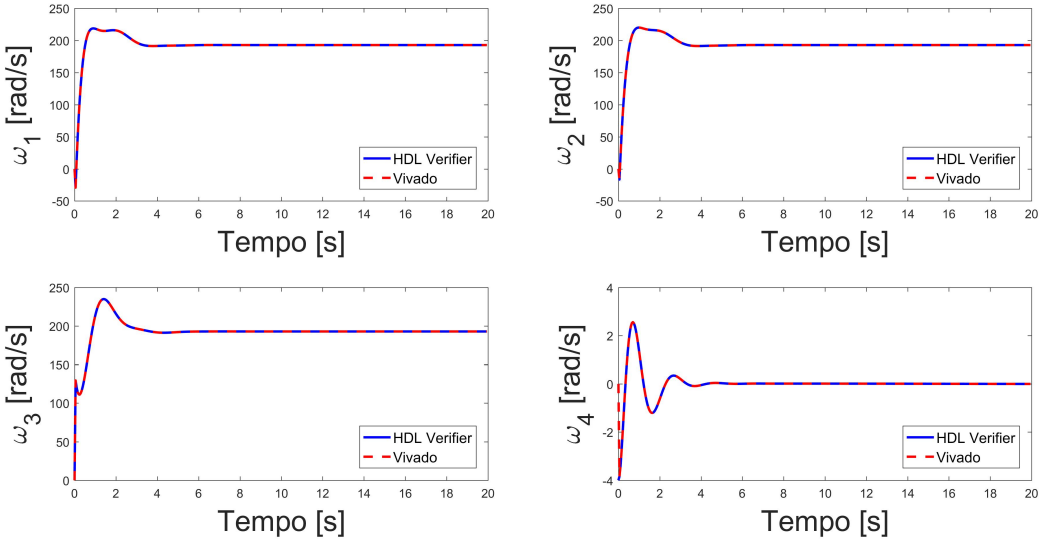

Figura 4.24: Comparação entre cálculo de variáveis de controle utilizando *HDL Verifier* e simulação comportamental no *Vivado*.

As figuras acima mostram que esses valores são bem próximos aos resultados do *HDL Verifier*. Portanto, opta-se por utilizar aqui a simulação comportamental do controlador MPC com restrições no *Vivado*,

somente para obter os dados das simulações do MPC com restrições em cada cenário desejado, de maneira mais rápida e eficiente.

Com essas características diferentes de teste explicadas, pode-se apresentar agora os resultados obtidos simulando o controlador MPC com restrições em *hardware*. Os resultados são apresentados nas Figuras 4.25 - 4.32 abaixo, tanto para N = 40 quanto para N = 50, com N_{iter} = 30. As figuras mostram resultados para simulações utilizando precisão de 27 e 32 *bits*. O tempo de cálculo das variáveis de controle, medido através da saída *ready* do controlador utilizando o HDL Verifier, é apresentado na Tabela 4.4.

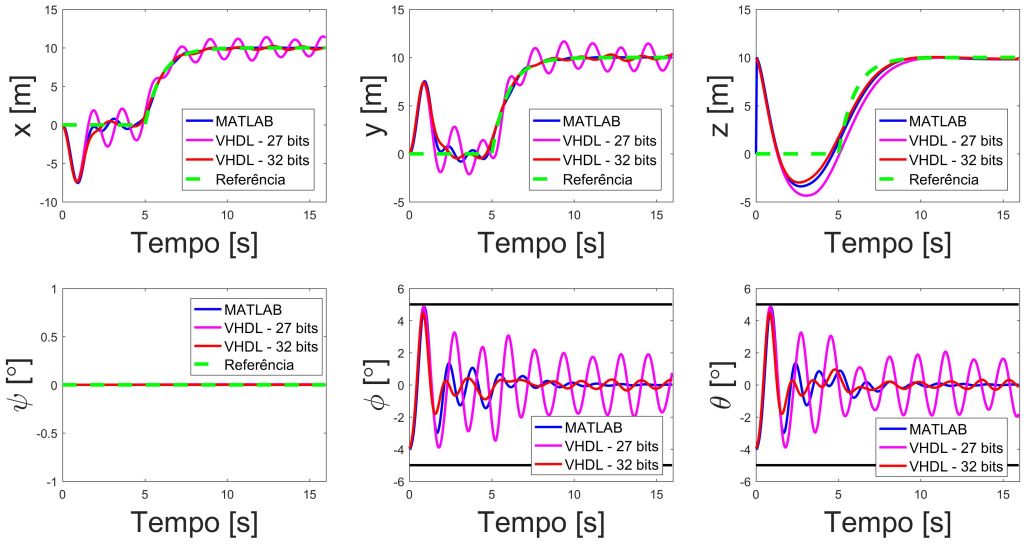

Figura 4.25: Simulação e comparação do quadrirrotor controlado utilizando o MPC com restrições com a simulação em MATLAB: saídas do sistema para N = 40, N_{iter} = 30 e referências de degrau filtrado.

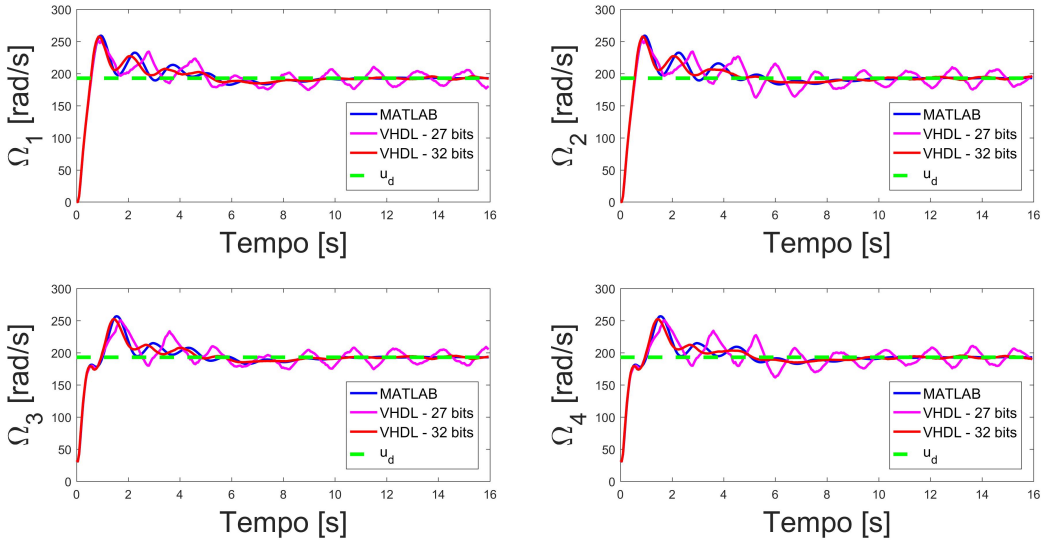

Figura 4.26: Simulação e comparação do quadrirrotor controlado utilizando o MPC com restrições com a simulação em MATLAB: variáveis de controle para N = 40, N_{iter} = 30 e referências de degrau filtrado.

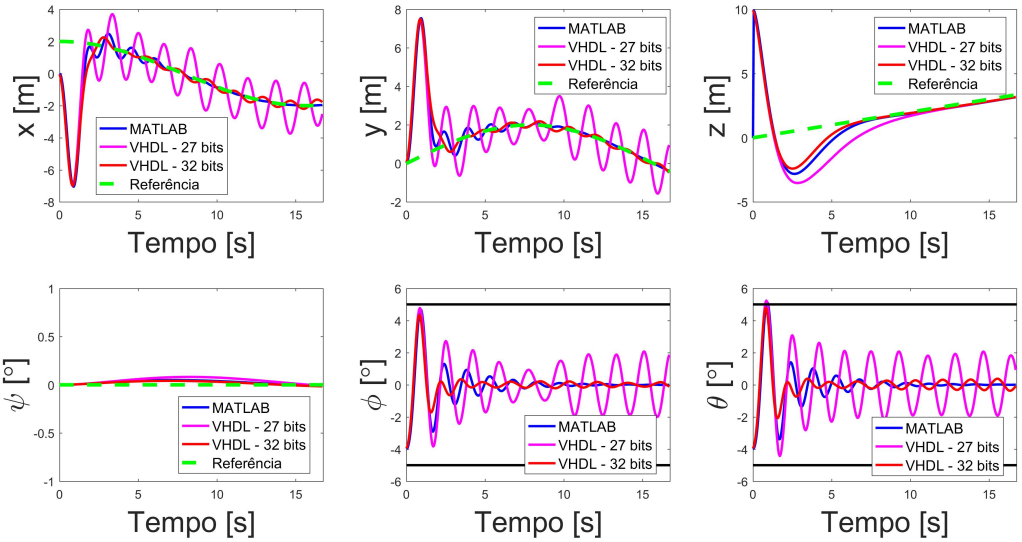

Figura 4.27: Simulação e comparação do quadrirrotor controlado utilizando o MPC sem restrições com a simulação em MATLAB: saídas do sistema para $N = 40$, $N_{iter} = 30$ e trajetória helicoidal.

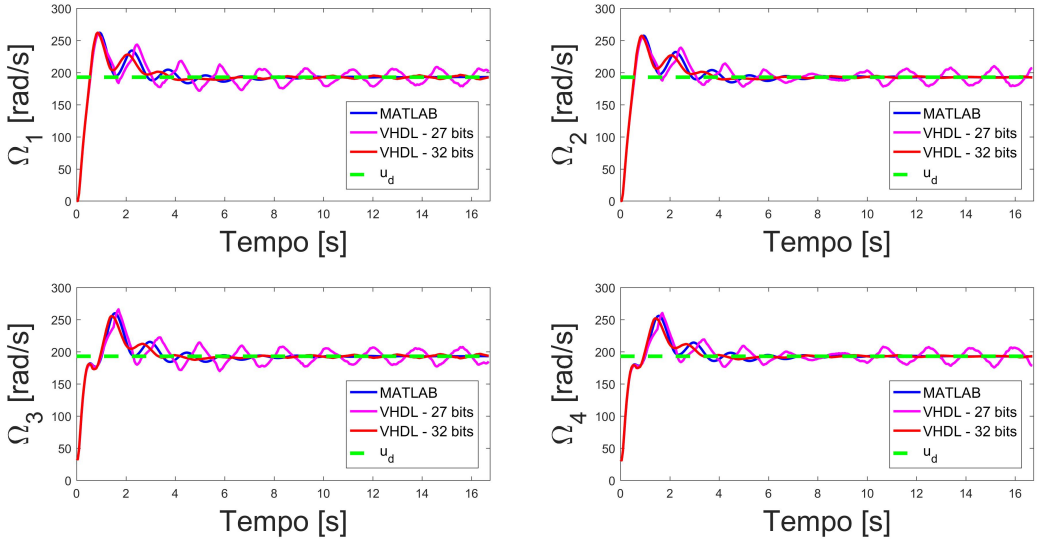

Figura 4.28: Simulação e comparação do quadrirrotor controlado utilizando o MPC sem restrições com a simulação em MATLAB: variáveis de controle para $N = 40$, $N_{iter} = 30$ e trajetória helicoidal.

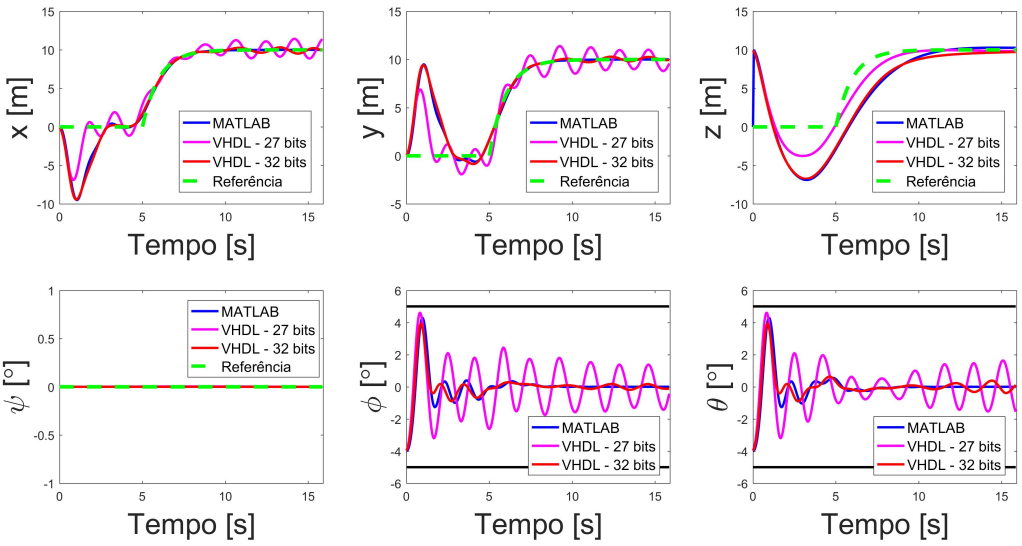

Figura 4.29: Simulação e comparação do quadrirrotor controlado utilizando o MPC com restrições com a simulação em MATLAB: saídas do sistema para $N = 50$, $N_{iter} = 30$ e referências de degrau filtrado.

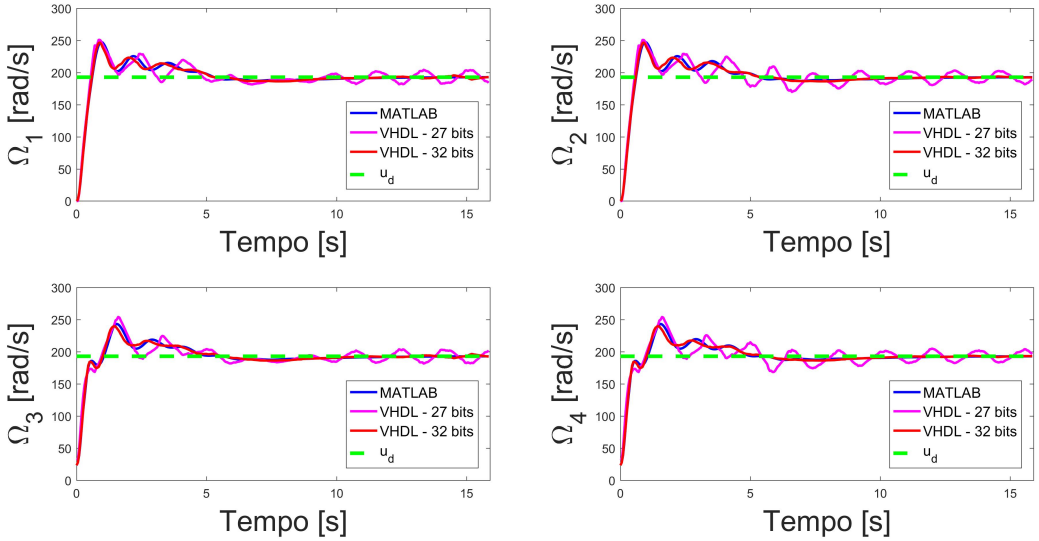

Figura 4.30: Simulação e comparação do quadrirrotor controlado utilizando o MPC com restrições com a simulação em MATLAB: variáveis de controle para $N = 50$, $N_{iter} = 30$ e referências de degrau filtrado.

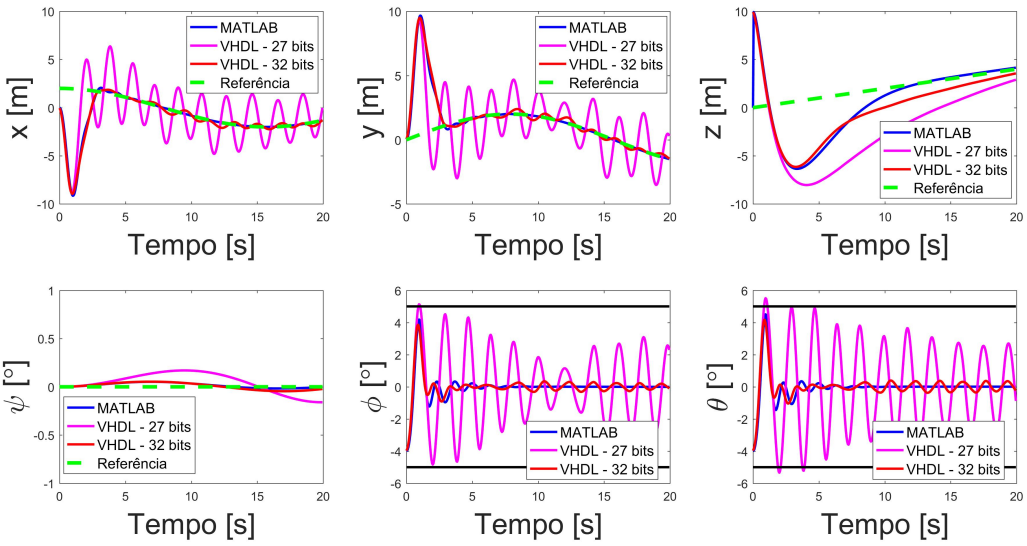

Figura 4.31: Simulação e comparação do quadrirrotor controlado utilizando o MPC com restrições com a simulação em MATLAB: saídas do sistema para N = 50, N_{iter} = 30 e trajetória helicoidal.

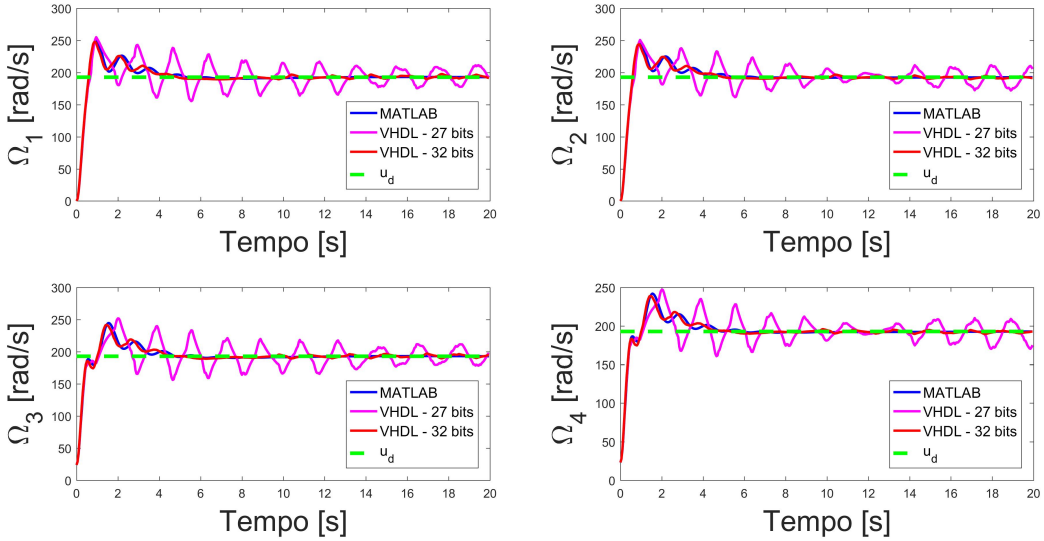

Figura 4.32: Simulação e comparação do quadrirrotor controlado utilizando o MPC com restrições com a simulação em MATLAB: variáveis de controle para N = 50, N_{iter} = 30 e trajetória helicoidal.

Tabela 4.4: Tempo de cálculo das variáveis de controle por parte dos controladores MPC com restrições implementados em FPGA.

N	**Frequência de *clock***	**Tempo de cálculo**
40	50 MHz	22.97 ms
50	50 MHz	29.34 ms

Analisando os resultados acima é possível perceber que o tempo de cálculo em ambos os casos é menor que 50 ms, o que garante que o controlador consegue calcular as variáveis de controle dentro de período de amostragem do sistema. Aqui, não ocorre o mesmo que acontece no MPC sem restrições: como a

solução não é tão paralelizável quanto a lei de controle do MPC sem restrições, o controlador com $N = 50$ demora mais tempo para calcular as variáveis de controle do que o controlador com $N = 40$. Quanto maior o valor de N, maiores as dimensões das matrizes, de forma que não é possível calcular todos os elementos resultantes das operações matriciais em um único ciclo de operação, o que causa essa diferença.

Entretanto, esse tempo de cálculo das variáveis de controle permanece constante, independente da quantidade de *bits* que são utilizados para representar os dados em ponto flutuante (27 ou 32 *bits*). Isso ocorre porque as unidades de multiplicação e soma em ponto flutuante que servem de base para os módulos desenvolvidos, apresentadas no trabalho de Muñoz (2012), são capazes de paralelizar as operações de acordo com a quantidade de *bits* totais utilizados, de forma que os mesmos sempre calculam os resultados em dois ciclos de *clock*. Assim, o tempo de cálculo do controlador não é impactado pela quantidade de *bits* adotada em ponto flutuante: isso afeta o consumo de recursos de *hardware* necessários para a implementação do controlador, conforme mostrado na Tabela 4.2.

O tempo de cálculo se mostrou rápido quando comparado com soluções de MPC implementado em FPGA encontradas na literatura e apresentadas neste trabalho. Da revisão bibliográfica acerca do tema apresentada no Capítulo 2, somente o trabalho de Ayala *et. al* (2016) consegue um tempo de cálculo menor, utilizando o NMPC também com horizonte de predição $N = 50$, mas para um sistema com 4 variáveis de estados e uma única variável de controle.

Além do tempo de cálculo ser mais rápido, nenhum dos outros trabalhos apresentados realizou testes com valores de N maiores que 30, com exceção do trabalho de Ayala *et. al* (2016), que conforme já citado, realizou testes com $N = 50$, assim como neste trabalho.

Se por um lado o tempo de cálculo é satisfatório, por outro lado, em todos os casos apresentados, o controlador apresenta um erro ao final do cálculo de cada variável de controle. No início da simulação, esse erro causa diferenças na resposta com relação às simulações computacionais em MATLAB, mas não o suficiente para fazer com que o comportamento em ambas as simulações fique muito diferente. No entanto, com o decorrer da simulação, esse erro vai aumentando e acumulando, de forma que apesar de seguir de certa forma as referências, as variáveis de estado não conseguem estabilizar nos valores dados pelas mesmas e ficam oscilando em torno delas.

O mesmo pode ser dito das variáveis de controle, que não estabilizam no valor desejado $u_d = 192.8$ rad/s: quando chegam perto desse valor, as variáveis de controle ficam oscilando em torno do valor de equilíbrio (u_d = 192.8 rad/s).

A única exceção para esse caso é a variável de estado z. É interessante que esta seja a única que consiga ser controlada, mas faz sentido quando analisa-se a física do quadrirrotor. Por conseguir estabilizar a dinâmica em z, pode-se assumir que a oscilação em torno de 192.8 rad/s para todas as variáveis de controle gera um mesmo valor médio de velocidade angular igual nos motores, já que para estabilizar a dinâmica em z, é necessário que todos os motores girem com a mesma velocidade. Assim, o valor médio de velocidade angular dos motores consegue estabilizar z, mas não consegue estabilizar as demais dinâmcias, o que gera esse comportamento oscilatório.

Contudo, os resultados mostram que ao adotar-se uma precisão maior em ponto flutuante, como no caso de 32 *bits*, essa oscilação diminui bastante, e apesar de a oscilação continuar, a amplitude das mesmas

diminui significativamente, de forma que os resultados obtidos são bem mais satisfatórios e próximos aos resultados obtidos nas simulações computacionais em MATLAB.

Esse erro é proveniente do módulo de *solver* de QP : o módulo *solver_ sat_ pen*. Esse módulo, durante suas primeiras iterações apresenta bons resultados, mas à medida que várias iterações são realizadas, o erro vai se propagando cada vez mais. Os primeiros valores das variáveis de controle calculados até controlam bem o sistema, mas eventualmente o resultado oscilatório é obtido. Como dito anteriormente, ao utilizar-se uma precisão maior em ponto flutuante, o comportamento oscilatório diminui, mas o consumo de recursos de *hardware* para implementação do controlador aumenta.

Isso indica que o *solver* apresentado em Alamir (2013) não é uma solução muito boa para ser implementada em *hardware* com o objetivo de controlar sistemas de dimensões elevadas. Ele é capaz de controlar o quadrirrotor em MATLAB, mas ao ser implementado em VHDL, não conseguiu executar de maneira eficiente os objetivos de controle. A principal dificuldade de implementar esse algoritmo em *hardware* se dá por conta da alta dependência de dados que há neste algoritmo: o módulo *solver_ sat_ pen* não é muito paralelizável, pois a maioria das operações matricias realizadas pelo mesmo têm que ser realizadas sequencialmente, ou seja, uma seguida da outra. Logo, as próximas matrizes a serem calculadas dependem das matrizes que estão sendo calculadas neste instante, e isso não permite a paralelização do algoritmo.

Segundo Soudré (SOUDRÉ, 2017), como o cálculo do próximo passo do MPC utilizando o algoritmo PGE possui dependência com o passo anterior, estendendo-se durante o cálculo de toda a sequência das próximas N variáveis de controle, verifica-se que o seu princípio de operação básica é de caráter serial e que, por conta de alta dependência de dados existente, há uma grande dificuldade na divisão do algoritmo em tarefas independentes e que possam ser executadas paralelamente.

Esse é um dos principais fatores que fazem com que o erro se propague da maneira que foi evidenciada. Os cálculos de otimização do algoritmo PGE são iterados 30 vezes, no caso das simulações apresentadas, a cada instante de amostragem. Isso signfica que, para calcular as variáveis de controle, o *solver* itera 30 vezes, acumulando a cada nova iteração um valor de erro maior, proveniente principalmente das diversas operações matricias que são realizadas ao longo dessas iterações.

Mais uma vez, é importante ressaltar que o erro não é grande o suficiente para divergir o sistema, mas impede a estabilização do mesmo.

Entretanto, por conta dos exemplos apresentados em Alamir (2013), sabe-se que esse é um *solver* que consegue controlar melhor sistemas de dimensões menores. Trata-se de um *solver* cuja implementação se mostra simples quando comparada com a implementação de outros *solvers* de QP comumente utilizados, como qpOASES, além de não apresentar dependências de bibliotecas numéricas para sua implementação. Esses foram os principais fatores que motivaram a escolha do mesmo, mas segundo Alamir (2013), este é um *solver* que pode apresentar problemas para o controle de sistemas complexos, como o quadrirrotor.

Portanto, após todas as discussões acerca dos resultados obtidos, é possível afirmar que uma arquitetura de MPC com restrições em *hardware* foi implementada. Ela realiza os cálculos em tempos menores que o tempo de amostragem do quadrirrotor (o que caracteriza um tempo de cálculo rápido quando comparado com as soluções existentes na literatura), mas que não foi capaz de estabilizar o modelo. O sistema não diverge, permanece na região de linearidade, mas não é capaz de alcançar os valores de referência deseja-

dos, oscilando em torno dos mesmos. É possível diminuir essa oscilação por meio de uma maior precisão em ponto flutuante, utilizando mais *bits* para representar os dados, mas isso também acarreta em maior consumo de recursos de *hardware* para implementação do controlador.

Logo, a solução de 32 *bits*, ainda que apresente oscilações pequenas, é a melhor opção, já que não excede a capacidade do *kit* utilizado e apresenta resultados bem mais próximos dos obtidos nas simulações computacionais em MATLAB do que a arquitetura em *hardware* do MPC com restrições utilizando 27 *bits*.

Capítulo 5

Conclusões

5.1 Conclusões do Trabalho

Após a apresentação de todos os resultados obtidos neste trabalho, é possível concluir que os objetivos estabelecidos no primeiro capítulo deste documento foram alcançados: foram implementados tanto o MPC sem restrições quanto o MPC com restrições para um quadrirrotor, utilizando arquiteturas de *hardware* desenvolvidas manualmente, com representação binária em ponto flutuante de 27 *bits* e também de 32 *bits* para o MPC com restrições. Todos os módulos são implementados por meio de um *script* em MATLAB que gera componentes parametrizados para atender às dimensões do problema de controle, de acordo com os parâmetros que foram adotados (horizonte de predição, número de iterações do *solver*, entre outros).

O controlador MPC sem restrições em *hardware* apresentou bons resultados, com um comportamento muito próximo da simulação computacional em MATLAB. Sua lei de controle é bem mais simples que a formulação completa do MPC, que incorpora as restrições do problema de controle, e por conta disso, este consome bem menos recursos de *hardware* que a solução do MPC com restrições.

O fato de a lei de controle do MPC sem restrições ser simples e analítica gera erros pequenos nos módulos de operações matriciais desenvolvidos em ponto flutuante, o que garante o bom funcionamento deste controlador. Entretanto, como esperado, este controlador não é capaz de lidar com cenários nos quais as restrições devem ser respeitadas, violando as mesmas em situações que a formulação completa do MPC é capaz de lidar com essas mesmas restrições.

No caso do MPC com restrições, os resultados variam de acordo com o número de *bits* utilizados para representar os dados em ponto flutuante. O algoritmo PGE, utilizado como *solver* de QP neste trabalho, é interessante para sistemas menos complexos, pois além de ser capaz de controlar melhor estes tipos de sistemas e ser capaz de respeitar de maneira mais adequada as restrições do problema de controle, trata-se de um *solver* sem dependências numéricas, estando completamente disponível em MATLAB. Essas foram as principais motivações para utilização do mesmo neste trabalho, pois permitiram que ele fosse implementado em VHDL de maneira manual, sendo esta inclusive, uma das principais contribuições deste trabalho.

Apesar disso, o algoritmo PGE não se mostrou uma opção muito boa para ser implementado em *hard-*

ware, pelo menos para o controle de um sistema complexo como o do quadrirrotor. Apesar de funcionar e ser capaz de controlar o modelo do quadrirrotor propriamente dito em MATLAB, há alguns fatores que resultam no comportamento obtido durante as simulações do controlador MPC em VHDL.

O fato de ser um algoritmo pouco paralelizável, com uma alta dependência de dados resulta em um controlador complexo, que consome uma grande quantidade de recursos de *hardware* quando comparado com a solução sem restrições do MPC. Trata-se de um algoritmo quase que exclusivamente sequencial, o que resulta em operações matriciais sendo realizadas de maneira serial, gerando erros de arredondamento que vão se acumulando ao longo de cada instrução. Para agravar essa situação, ainda tem-se um grande número de iterações necessários para que o *solver* funcione, o que aumenta ainda mais o erro acumulado, gerando valores subótimos de variáveis de controle que causam o comportamento oscilatório do quadrirrotor: as variáveis de estados não conseguem estabilizar nos valores de referência, oscilando em torno das mesmas.

Entretanto, é importante salientar que o erro que está sendo tratado aqui trata-se de um erro de arredondamento, que pode ser mitigado com o uso de uma maior precisão em ponto flutuante, utilizando-se mais *bits* para representação dos dados. Os resultados obtidos utilizando-se 32 *bits* para o MPC com restrições ainda apresentam oscilações, mas de amplitude bem reduzida quando comparada com os resultados de 27 *bits*, de forma que esses resultados são bem mais próximos aos obtidos nas simulações computacionais utilizando MATLAB.

Todavia, esse tipo de solução impacta no consumo de recursos de *hardware* necessários para implementar o controlador. O tempo de cálculo não é impactado aumentando-se a quantidade de *bits* utilizados em ponto flutuante, por conta do funcionamento dos multiplicadores e somadores/subtratores em ponto flutuante utilizados como base neste trabalho.

Por fim, cabe ressaltar as contribuições deste trabalho. Primeiramente, todas as arquiteturas desenvolvidas foram implementadas de maneira manual, o que difere da maioria dos trabalhos presentes na literatura, que optam pela geração automática de códigos em linguagem de descrição de *hardware*. Os controladores desenvolvidos estão todos em ponto flutuante, com precisão variável, o que garante uma faixa dinâmica de valores maior que controladores desenvolvidos em ponto fixo, que são a maioria dos trabalhos encontrados na literatura.

O problema de controle abordado neste trabalho é muito relevante. Não só o quadrirrotor é um sistema amplamente estudado na atualidade, mas é um problema de controle desafiador, com dinâmicas rápidas, acopladas e instabilidade inerente. É um problema mais complexo que a maioria dos sistemas encontrados na literatura que são controlados utilizando o MPC em *hardware*.

Tudo que foi descrito até aqui se dá em uma área ainda pouco explorada na literatura: controle de quadrirrotores utilizando o MPC em *hardware*, implementado utilizando FPGA. Portanto, considera-se que os resultados deste trabalho são importantes contribuições para a literatura, assim como as discussões apresentadas neste documento.

5.2 Sugestões de trabalhos futuros

A primeira etapa que pode ser citada como sugestão de trabalhos futuros corresponde a uma análise mais profunda das arquiteturas em VHDL desenvolvidas, analisando características que não puderem ser incluídas neste documento por não terem sido finalizadas antes do fim da escrita deste documento, como frequência máxima de operação e consumo de potência.

Este trabalho abordou uma metodologia de desenvolvimento de estratégias de controle MPC desenvolvidas em linguagem de descrição de *hardware* baseada no desenvolvimento manual dos algoritmos, ou seja sem o uso de qualquer ferramenta de geração automática de código. Este é inclusive um dos pontos considerados como contribuição deste trabalho, uma vez que a maioria dos trabalhos encontrados na literatura que implementam o MPC em FPGA utilizam algoritmos gerados automaticamente por ferramentas como o *HDL Coder* do MATLAB ou o *Vivado HLS*, da *Xilinx*.

Uma das possibilidades de trabalhos futuros seria justamente utilizar a metodologia de geração automática dos algoritmos para comparação com os resultados obtidos com os códigos desenvolvidos manualmente neste trabalho. Como os algoritmos já estão disponiveis em forma de *scripts* do MATLAB, a ferramenta *HDL Coder* do próprio MATLAB aparenta ser um bom candidato para este fim. É uma ferramenta muito utilizada por diversos trabalhos encontrados na literatura, sendo que alguns destes foram apresentados no capítulo de revisão bibliográfica deste documento.

A grande dificuldade referente a essa metodologia é que ela é uma abordagem completamente diferente: como os algoritmos que compõem o MPC realizem diversas operações matriciais, é preciso realizar modificações nos algoritmos de modo a fazer com que ferramentas de geração automática possam implementar esse tipo de operação em *hardware*: ferramentas como *HDL Coder*, mesmo sendo ferramentas do próprio MATLAB (que possui um ótimo ambiente para realização de operações matriciais), não possuem suporte direto para operações com matrizes. Isso exige que os algoritmos tenham que ser modificados, fazendo com que essa metodologia demande certo tempo para ser realizada. Essa é a principal razão pela qual tal metodologia não foi realizada neste trabalho para comparação com o MPC desenvolvido de maneira manual, e caracteriza a mesma como uma boa alternativa de trabalho futuro.

Outra metodologia que pode ser explorada é a utilização de um outro *kit* de FPGA; na verdade um SoC, que é composto por um FPGA e um processador de *software*, em um mesmo *kit*. O uso desse tipo de plataforma de desenvolvimento permite a implementação de parte da lógica do MPC em *hardware*, utilizando o FPGA do *kit*, e parte da lógica implementada em *software*, por meio da programação do processador também disponível no *kit* adotado.

O interessante dessa técnica é que, além de ser uma metodologia pouco explorada na literatura, trata-se de uma alternativa interessante para tentar lidar com alguns dos problemas encontrados neste trabalho. É possível, por exemplo, realizar somente os cálculos das matrizes de custo e restrições em *hardware*, por meio do módulo *compute_ MPC_ matrices* apresentado neste trabalho e implementar o *solver* de QP, que no caso deste trabalho foi o algoritmo de otimização PGE, em *software*. Isso poderia ajudar a reduzir a acumulação de erros que ocorre por conta das várias iterações necessárias para o cálculo de uma sequência ótima de variáveis de controle, por exemplo.

Além disso, outra possibilidade é que essa metodologia facilitaria a utilização de outros tipos de *solver*

de otimização para o MPC. Implementando os mesmos em *software*, torna-se mais fácil utilizar *solvers* mais eficientes do que o algoritmo PGE apresentado, como o qpOASES. Este é bastante utilizado tanto em aplicações industriais quanto acadêmicas, e seria um forte candidato a obter resultados melhores do que os resultados obtidos com o algoritmo PGE. O que tornou difícil sua implementação e fez com que o algoritmo PGE fosse escolhido em seu lugar é que o mesmo encontra-se disponível em linguagem C e C++, possuindo diversas bibliotecas e dependências numéricas que são difíceis de adaptar para linguagens de descrição de *hardware*, como o VHDL. Utilizando um processador de *software* essa transição seria mais simples e natural.

Por fim, é possível propor o teste com outros sistemas a serem controlados. Como dito durante a seção de revisão bibliográfica de FPGAs no Capítulo 2, é de interesse que as arquiteturas dos controladores MPC sejam capazes de se adaptar às dimensões e às características do problema de controle adotado, e os mesmos foram desenvolvidos de maneira que isso seja satisfeito, utilizando *scripts* MATLAB para parametrizar os módulos em *hardware* e fazer com que o MPC funcione adequadamente. Mediante essa característica, e por conta da utilização de aritmética em ponto flutuante (que garante uma extensa faixa dinâmica de valores que podem ser representados em binário), é possível aplicar o controlador MPC para outros tipos de sistemas. Como visto no capítulo de resultados e discussões, o *solver* PGE não é indicado para sistemas de dimensões elevadas, que geram problemas de controle mais complexos, mas isso não impede que o mesmo seja utilizado para controlar sistemas mais simples, com menos variáveis de estados e variáveis de controle.

Como as áreas abordadas nesse trabalho possuem grande espaço para contribuição, há uma grande variedade de possibilidades de trabalhos futuros. O mais importante é que este trabalho realiza contribuições para a literatura de sistemas de controle MPC embarcados em FPGA, por meio de uma arquitetura desenvolvida manualmente em ponto flutuante, utilizando o algoritmo de otimização PGE. Assim, espera-se que este trabalho possa servir como base de contribuição para as propostas que aqui foram sugeridas, auxiliando na comparação de novas metodologias e resultados obtidos com o que foi discutido neste documento.

REFERÊNCIAS BIBLIOGRÁFICAS

ABDOLHOSSEINI, M.; ZHANG, Y. M.; RABBATH, C. A. An Efficient Model Predictive Control Scheme for an Unmanned Quadrotor Helicopter. *Journal of Intelligent & Robotic Systems*, v. 70, n. 1-4, p. 27–38, apr 2013. ISSN 0921-0296. Disponível em: <http://link.springer.com/10.1007/s10846-012-9724-3>.

ABUGHALIEH, K. M.; ALAWNEH, S. G. A Survey of Parallel Implementations for Model Predictive Control. *IEEE Access*, 2019.

ALAMIR, M. *A Pragmatic Story of Model Predictive Control: Self-Contained Algorithms and Case-Studies*. [S.l.]: CreateSpace Independent Publishing Platform., 2013.

ALEXIS, K.; NIKOLAKOPOULOS, G. Experimental model predictive attitude tracking control of a quadrotor helicopter subject to wind-gusts. In: *8th Mediterranean Conference on Control & Automation Congress Palace Hotel, Marrakech, Morocco*. [s.n.], 2010. p. 1461–1466. Disponível em: <http://ieeexplore.ieee.org/abstract/document/5547844/>.

ALVARENGA, J. et al. Survey of Unmanned Helicopter Model-Based Navigation and Control Techniques. *Journal of Intelligent & Robotic Systems*, v. 80, n. 1, p. 87–138, oct 2015. ISSN 0921-0296. Disponível em: <http://link.springer.com/10.1007/s10846-014-0143-5>.

AYALA, H. et al. Nonlinear Model Predictive Control Hardware Implementation with Custom-precision Floating Point Operations. In: *IEEE 24th Mediterranean Conference on Control and Automation (MED)*. [S.l.: s.n.], 2016. p. 135–140.

BABB, J. et al. Logic emulation with virtual wires. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, v. 16, n. 6, p. 606–626, 1997.

BECKER, M.; BOUABDALLAH, S.; SIEGWART, R. Desenvolvimento de um controlador de desvio de obstáculos para um mini-helicóptero quadrirotor autônomo - 1ª fase: Simulação. In: *Congresso Brasileiro de Automática (CBA)*. [S.l.: s.n.], 2006. v. 1, p. 1201–1206.

BEMPORAD, A.; FILIPPI, C. Suboptimal Explicit Receding Horizon Control via Approximate Multiparametric Quadratic Programming. *Journal of Optimization Theory and Applications*, v. 117, n. 1, p. 9–38, apr 2003. ISSN 0022-3239. Disponível em: <http://link.springer.com/10.1023/A:1023696221899>.

BEMPORAD, A.; MORARI, M.; DUA, V. The explicit linear quadratic regulator for constrained systems. *Automatica*, v. 38, n. 1, p. 3–20, 2002. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0005109801001741>.

BLERIS, L. et al. A co-processor FPGA platform for the implementation of real-time model predictive control. *2006 American Control Conference*, p. 1912–1917, 2006. ISSN 07431619.

BOUABDALLAH, S.; SIEGWART, R. Backstepping and sliding-mode techniques applied to an indoor micro quadrotor. In: *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*. [s.n.], 2005. p. 2259–2264. Disponível em: <http://ieeexplore.ieee.org/abstract/document/1570447/>.

CAI, X. et al. Fast distributed MPC based on active set method. *Computers & Chemical Engineering*, Pergamon, v. 71, p. 158–170, dec 2014. ISSN 0098-1354. Disponível em: <https://www-sciencedirect-com.ez54.periodicos.capes.gov.br/science/article/pii/S0098135414002439>.

CARTER, W. et al. A user programmable reconfigurable gate array,. In: *Custom Integrated Circuits Conference*. [S.l.: s.n.], 1986. p. 232–235.

CASTILLO, P.; DZUL, A.; LOZANO, R. Real-time stabilization and tracking of a four rotor mini-rotorcraft. In: *IEEE Transactions on Control Systems Technology*. [s.n.], 2004. v. 12, n. 4, p. 510–516. Disponível em: <http://ieeexplore.ieee.org/abstract/document/7086519/>.

CHEN, X.; WU, X. Design and implementation of Model Predictive Control algorithms for small satellite three-axis stabilization. *Information and Automation (ICIA), 2011 IEEE International Conference*, n. June, p. 666–671, 2011.

CHIKASHA, P. N.; DUBE, C. Adaptive Model Predictive Control of a Quadrotor. *IFAC-PapersOnLine*, Elsevier B.V., v. 50, n. 2, p. 157–162, 2017. ISSN 24058963. Disponível em: <http://linkinghub.elsevier.com/retrieve/pii/S2405896317335656>.

CUTLER, C.; RAMAKER, B. Dynamic matrix control - A computer control algorithm. In: *Proc. Joint Automatic Control Conf., San Francisco*. [s.n.], 1980. Disponível em: <https://www.infona.pl/resource/bwmeta1.element.ieee-art-000004232009>.

DOMAHIDI, A. et al. Survey of Industrial Applications of Embedded Model Predictive Control. In: *European Control Conference (ECC)*. [S.l.: s.n.], 2016.

DOMAHIDI, A. et al. Efficient interior point methods for multistage problems arising in receding horizon control. In: *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*. IEEE, 2012. p. 668–674. ISBN 978-1-4673-2066-5. Disponível em: <http://ieeexplore.ieee.org/document/6426855/>.

DUA, P. et al. MPC on a chip-Recent advances on the application of multi-parametric model-based control. *Computers and Chemical Engineering*, v. 32, n. 4-5, p. 754–765, 2008. ISSN 00981354.

EKAPUTRI, C.; SYAICHU-ROHMAN, A. Model predictive control (MPC) design and implementation using algorithm-3 on board SPARTAN 6 FPGA SP605 evaluation kit. *Proceedings of 2013 3rd International Conference on Instrumentation, Control and Automation, ICA 2013*, p. 115–120, 2013.

FERREAU, H.; BOCK, H.; DIEHL, M. An online active set strategy to overcome the limitations of explicit MPC. *International Journal of Robust and Nonlinear Control*, v. 18, p. 816–830, 2008. Disponível em: <http://onlinelibrary.wiley.com/doi/10.1002/rnc.1251/full>.

FERREAU, H. J. et al. *qpOASES User's Manual*. [S.l.], 2017. Disponível em: <https://projects.coin-or.org/qpOASES/export/HEAD/releases/3.2.1/doc/manual.pdf>.

FERREAU, H. J. et al. qpOASES: A parametric active-set algorithm for quadratic programming. *Mathematical Programming Computation*, v. 6, n. 4, p. 327–363, 2014.

FRANK, P. *Advances in Control: Highlights of ECC'99*. [s.n.], 2012. Disponível em: <https://books.google.com.br/books?hl=pt-BR{&}lr={&}id=WPAICAAAQBAJ{&}oi=fnd{&}pg=PA1{&}dq=Frank,+P.M.,+1999.+Advances+in+Control+(Highlights+of+ECC{\%}2799).+Springer-Verlag,+London,+{&}ots=UMshqAfEjQ{&}sig=UyM>.

GROSMAN, B. et al. Multi-zone-MPC: Clinical inspired control algorithm for the artificial pancreas. *IFAC Proceedings Volumes (IFAC-PapersOnline)*, Elsevier, v. 18, n. PART 1, p. 7120–7125, jan 2011. ISSN 14746670. Disponível em: <http://www-sciencedirect-com.ez54.periodicos.capes.gov.br/science/article/pii/S1474667016447488>.

HARTLEY, E.; MACIEJOWSKI, J. Predictive control for spacecraft rendezvous in an elliptical orbit using an FPGA. *European Control Conference (ECC)*, p. 1359–1364, 2013. Disponível em: <http://ieeexplore.ieee.org/xpls/abs{_}all.jsp?arnumber=6669>.

HARTLEY, E. N. et al. Predictive Control Using an FPGA With Application to Aircraft Control. *IEEE Transactions on Control Systems Technology,*, v. 22, n. 3, p. 1006–1017, 2014. ISSN 1063-6536.

IRIZARRY, J.; GHEISARI, M.; WALKER, B. N. Usability assessment of drone technology as safety inspection tools. *Journal of Information Technology in Construction (ITcon)*, v. 17, n. 12, p. 194–212, 2012. Disponível em: <http://itcon.org/paper/2012/12>.

ISERMANN, R.; SCHAFFNIT, J.; SINSEL, S. Hardware-in-the-loop simulation for the design and testing of engine-control systemsNo Title. *Control Engineering Practice*, v. 7, n. 5, p. 643–653, 1999.

JEREZ, J. L. et al. Model predictive control for deeply pipelined field-programmable gate array implementation: algorithms and circuitry. *Control Theory and Applications*, v. 6, n. 8, p. 1029 – 1041, 2012. ISSN 17518644. Disponível em: <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6248369>.

JOHANSEN, T. A.; GRANCHAROVA, A. Approximate explicit constrained linear model predictive control via orthogonal search tree. *IEEE Transactions on Automatic Control*, v. 48, n. 5, p. 810–815, 2003. Disponível em: <http://ieeexplore.ieee.org/abstract/document/1198605/>.

JOOS, A. et al. *Method for Parallel FPGA Implementation of Nonlinear Model Predictive Control*. IFAC, 2012. v. 45. 73–78 p. ISSN 14746670. Disponível em: <http://linkinghub.elsevier.com/retrieve/pii/S1474667015350230>.

JOOST, R.; SALOMON, R. Advantages of FPGA-based multiprocessor systems in industrial applications. *31st Annual Conference of IEEE Industrial Electronics Society, 2005. IECON 2005.*, p. 6 pp., 2005. ISSN 1553-572X. Disponível em: <http://ieeexplore.ieee.org/document/1568946/>.

KANELLAKIS, C.; NIKOLAKOPOULOS, G. Survey on Computer Vision for UAVs: Current Developments and Trends. *Journal of Intelligent and Robotic Systems: Theory and Applications*, Journal of Intelligent & Robotic Systems, v. 87, n. 1, p. 141–168, 2017. ISSN 15730409.

KAPERNICK, B. et al. A synthesis strategy for nonlinear model predictive controller on FPGA. In: *IEEE UKAAC International Conference on Control (CONTROL)*. [s.n.], 2014. p. 662–667. Disponível em: <http://ieeexplore.ieee.org/abstract/document/6915218/>.

KENDOUL, F. Survey of advances in guidance, navigation, and control of unmanned rotorcraft systems. *Journal of Field Robotics*, v. 29, n. 2, p. 315–378, mar 2012. ISSN 15564959. Disponível em: <http://doi.wiley.com/10.1002/rob.20414>.

KHAN, B.; ROSSITER, J. A comparison of the computational efficiency of generalised function MPC using active set methods. *IFAC Proceedings Volumes*, Elsevier, v. 45, n. 15, p. 192–197, jan 2012. ISSN 1474-6670. Disponível em: <https://www-sciencedirect-com.ez54.periodicos.capes.gov.br/science/article/pii/S1474667016304414>.

KOUZOUPIS, D. et al. First-order methods in embedded nonlinear model predictive control. In: *2015 European Control Conference (ECC)*. IEEE, 2015. p. 2617–2622. ISBN 978-3-9524-2693-7. Disponível em: <http://ieeexplore.ieee.org/document/7330932/>.

KOZÁK, Š. State-of-the-art in Control Engineering. *Journal of Electrical Systems and Information Technology*, v. 1, n. 1, p. 1–9, 2014. ISSN 23147172. Disponível em: <http://linkinghub.elsevier.com/retrieve/pii/S2314717214000038>.

KUFOALOR, D. K. M. et al. Structure exploitation of practical MPC formulations for speeding up first-order methods. In: *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*. IEEE, 2017. p. 1912–1918. ISBN 978-1-5090-2873-3. Disponível em: <http://ieeexplore.ieee.org/document/8263929/>.

LEE, E.; MARKUS, L. *Foundations of optimal control theory*. NY: Wiley, 1967. 589 p. Disponível em: <http://www.dtic.mil/docs/citations/AD0663614>.

LEE, J. H. Model predictive control: Review of the three decades of development. *International Journal of Control, Automation and Systems*, v. 9, n. 3, p. 415–424, 2011. ISSN 15986446.

Limaverde Filho, J. O. D. A. Lourenço, T. S. et al. Trajectory tracking for a quadrotor system: A flatness-based nonlinear predictive control approach. *2016 IEEE Conference on Control Applications (CCA)*, p. 1380–1385, 2016. Disponível em: <http://ieeexplore.ieee.org/document/7587999/http://ieeexplore.ieee.org/abstract/document/7587999/>.

LING, K. V.; WU, B. F.; MACIEJOWSKI, J. Embedded Model Predictive Control (MPC) using a FPGA. In: *IFAC Proceedings Volumes*. [S.l.: s.n.], 2008. v. 41, n. 2, p. 15250–15255. ISBN 9783902661005. ISSN 14746670.

LOPES, R. et al. Model Predictive Control applied to tracking and attitude stabilization of a VTOL quadrotor aircraft. In: *21st International Congress of Mechanical Engineering*. [s.n.], 2011. p. 176–185. Disponível em: <https://pdfs.semanticscholar.org/a7b4/429f20b6136b948bfcfb9fe9f59afe9042d9.pdf>.

LUO, B. et al. A New Model Predictive Controller with Swarm Intelligence Implemented on FPGA. *International Symposium on Advanced Control of Industrial Processes (ADCONIP)*, n. 2, p. 427–432, 2011.

MADANI, T.; BENALLEGUE, A. Control of a quadrotor mini-helicopter via full state backstepping technique. *Proceedings of the 45th IEEE Conference on Decision & Control.*, p. 1515–1520, 2006. Disponível em: <http://ieeexplore.ieee.org/abstract/document/4177802/>.

MADANI, T.; BENALLEGUE, A. Control of a quadrotor mini-helicopter via full state backstepping technique. In: *45th IEEE Conference on Decision and Control*. [S.l.: s.n.], 2006. p. 1515–1520.

MATHWORKS. *HDL Verifier*. 2019.

MATTINGLEY, J.; BOYD, S. *Automatic code generation for real-time convex optimization*. Cambridge Press, 2009. Disponível em: <https://books.google.com.br/books?hl=pt-BR{&}lr={&}id=UOpnvPJ151gC{&}oi=fnd{&}pg=PA1{&}dq=J.+Mattingley+and+S.+Boyd,+Automatic+code+generation+for+real-time+convex+optimization,in+Convex+Optimization+in+Signal+Processing+and+Communications,{&}ots=Ze>.

MAYNE, D.; MICHALSKA, H. Receding horizon control of nonlinear systems. In: *IEEE Transactions on Automatic Control*. [s.n.], 1990. v. 35, p. 814–824. Disponível em: <http://ieeexplore.ieee.org/abstract/document/57020/>.

MAYNE, D. et al. Constrained model predictive control: Stability and optimality. *Automatica*, v. 36, n. 6, p. 789–814, jun 2000. ISSN 00051098. Disponível em: <http://linkinghub.elsevier.com/retrieve/pii/S0005109899002149>.

MISTLER, V.; BENALLEGUE, A. Exact linearization and noninteracting control of a 4 rotors helicopter via dynamic feedback. In: *Proceedings of the 2001 IEEE International Workshop on Robot and Human Interactive Communication*. [s.n.], 2001. p. 586–593. Disponível em: <http://ieeexplore.ieee.org/abstract/document/981968/>.

MONMASSON, E.; CIRSTEA, M. N. FPGA Design Methodology for Industrial Control SystemsâĂŤA Review. *IEEE TRANSACTIONS ON INDUSTRIAL ELECTRONICS*, v. 54, n. 4, p. 1824–1842, 2007.

MONMASSON, E. et al. FPGAs in industrial control applications. *Industrial Informatics, IEEE Transactions on*, v. 7, n. 2, p. 224–243, 2011. ISSN 1551-3203.

MOORE, G. E. The microprocessor: engine of the technology revolution. In: *Communications of the ACM*. [S.l.: s.n.], 1997. v. 40, n. 2, p. 112–114.

MUÑOZ, D. M. *Otimização por inteligência de enxames usando arquiteturas paralelas para aplicações embarcadas*. 192 p. Tese (Doutorado) — Universidade de Brasília, 2012.

MURILO, A.; ALAMIR, M.; ALBERER, D. A general NMPC framework for a diesel engine air path. *International Journal of Control*, v. 87, n. 10, p. 2194–2207, 2014. Disponível em: <http://www.tandfonline.com/doi/abs/10.1080/00207179.2014.905708>.

MURILO, A.; LOPES, R. V. A Low Dimensional Parameterized NMPC Scheme for Quadrotors. In: *5th International Conference on Control, Decision and Information Technologies (CoDIT)*. [S.l.: s.n.], 2018. p. 58–63.

NISSER, T.; WESTIN, C. *Human factors challenges in unmanned aerial vehicles (uavs): A literature review*. Tese (Doutorado), 2006. Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.454.2233{&}rep=rep1{&}ty>.

OGATA, K. *Modern control engineering*. 5ª. ed. Prentice Hall, 2002. Disponível em: <http://www.uvic.ca/engineering/ece/assets/docs/current/undergraduate/201709/ELEC360.pdf>.

OHHIRA, T.; SHIMADA, A. Model Predictive Control for an Inverted-pendulum Robot with Time-varying Constraints. *IFAC-PapersOnLine*, Elsevier B.V., v. 50, n. 1, p. 776–781, jul 2017. ISSN 24058963. Disponível em: <https://doi.org/10.1016/j.ifacol.2017.08.252http://www-sciencedirect-com.ez54.periodicos.capes.gov.br/science/article/pii/S2405896317305657>.

PATRASCU, A.; NECOARA, I. Complexity certifications of inexact projection primal gradient method for convex problems : application to embedded MPC. In: *IEEE 24th Mediterranean Conference on Control and Automation (MED)*. [S.l.: s.n.], 2016. p. 125–130. ISBN 9781467383455.

PEDRONI, V. A. *Circuit Design with VHDL*. [S.l.]: MIT Press, 2004.

PROPOI, A. Use of linear programming methods for synthesizing sampled-data automatic systems. *Automn. Remote Control*, v. 24, n. 7, p. 837–844, 1963. Disponível em: <https://ci.nii.ac.jp/naid/10008469285/>.

QIN, S.; BADGWELL, T. A survey of industrial model predictive control technology. *Control Engineering Practice*, v. 11, n. 7, p. 733–764, 2003. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0967066102001867>.

QIN, S. J.; BADGWELL, T. A. An overview of industrial model predictive control technology. In: *Proc. of Fifth Int. Conf. on Chemical Process Control, Tahoe City, CA, aIChE Symposium Series*. [s.n.], 1997. v. 93, p. 232–256. Disponível em: <https://pdfs.semanticscholar.org/dc68/809af6899f9ced9db111c1e9d77348880427.pdf>.

RAFFO, G. V.; ORTEGA, M. G.; RUBIO, F. R. MPC with Nonlinear âĎŃâĹđ Control for Path Tracking of a Quad-Rotor Helicopter. In: *17th World Congress International Federation of Automatic Control (IFAC-08)*. [s.n.], 2008. v. 41, n. 2, p. 8564–8569. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1474667016403277>.

RAO, C. V.; WRIGHT, S. J.; RAWLINGS, J. B. Application of Interior-Point Methods to Model Predictive Control. *Journal of Optimization Theory and Applications*, v. 99, n. 3, p. 723–757, dec 1998. ISSN 0022-3239. Disponível em: <http://link.springer.com/10.1023/A:1021711402723>.

RICHALET, J. et al. Model predictive heuristic control. *Automatica*, v. 14, n. 5, p. 413–428, 1978. Disponível em: <https://dl.acm.org/citation.cfm?id=2233257>.

RODRIGUEZ-ANDINA, J. J.; MOURE, M. J.; VALDES, M. D. Features, design tools, and application domains of FPGAs. *IEEE Transactions on Industrial Electronics*, v. 54, n. 4, p. 1810–1823, 2007. ISSN 02780046.

RODRIGUEZ, C. A.; PONCE, P.; MOLINA, A. ANFIS and MPC controllers for a reconfigurable lower limb exoskeleton. *Soft Computing*, v. 21, n. 3, p. 571–584, feb 2017. ISSN 1432-7643. Disponível em: <http://link.springer.com/10.1007/s00500-016-2321-9>.

RODRIGUEZ, J. et al. Model predictive control – a simple and powerful method to control power converters. *2009 IEEE 6th International Power Electronics and Motion Control Conference*, v. 58, n. 2, p. 41–49, may 2009. ISSN 02780046. Disponível em: <http://ieeexplore.ieee.org/document/5289335/http://link.springer.com/10.1007/s10846-009-9347-5>.

SALDANHA, L.; LYSECKY, R. Float-to-fixed and fixed-to-float hardware converters for rapid hardware/software partitioning of floating point software applications to static and dynamic fixed point coprocessors. *Design Automation for Embedded Systems*, v. 13, n. 3, p. 139–157, 2009. ISSN 09295585.

SANTANA, P. H. d. R. Q. e. A.; BORGES, G. A. Modelagem e Controle de Quadrirrotores. *Simpósio Brasileiro de Automação Inteligente*, 2009.

SANTANA, P. H. D. R. Q. E. A.; BRAGA, M. A. Concepção de um veículo aéreo não-tripulado do tipo quadrirrotor. p. 142, 2008.

SHOUKRY, Y.; EL-KHARASHI, M. W.; HAMMAD, S. MPC-On-chip: An embedded GPC coprocessor for automotive active suspension systems. *IEEE Embedded Systems Letters*, v. 2, n. 2, p. 31–34, 2010. ISSN 19430663.

SOUDRÉ, M. M. *Uma avaliação experimental da plataforma Parallela utilizando controle preditivo como um estudo de caso*. 117 p. Tese (Doutorado), 2017.

SPIVEY, B. J.; EDGAR, T. F. Dynamic modeling, simulation, and MIMO predictive control of a tubular solid oxide fuel cell. *Journal of Process Control*, Elsevier Ltd, v. 22, n. 8, p. 1502–1520, 2012. ISSN 09591524. Disponível em: <http://dx.doi.org/10.1016/j.jprocont.2012.01.015>.

STOUT, T. M.; WILLIAMS, T. J. Pioneering work in the field of computer process control. In: *IEEE Annals of the History of Computing*. [s.n.], 1995. v. 17, n. 1, p. 6–18. Disponível em: <http://ieeexplore.ieee.org/abstract/document/366507/>.

TAKÁCS, G. et al. Embedded explicit model predictive vibration control. *Mechatronics*, v. 36, p. 54–62, 2016. ISSN 09574158.

TOCCI, R. J.; WIDMER, N. S.; MOSS, G. L. *Sistemas Digitais: princípios e aplicações*. [S.l.]: Prentice-Hall, 2007.

TØNDEL, P.; JOHANSEN, T. A.; BEMPORAD, A. An algorithm for multi-parametric quadratic programming and explicit MPC solutions. *Automatica*, v. 39, n. 5, p. 489–497, 2003. ISSN 00051098. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0005109802002509>.

TRIMBERGER, S. M. Three ages of FPGAs: A retrospective on the first thirty years of FPGA technology. *Proceedings of the IEEE*, v. 103, n. 3, p. 318–331, 2015. ISSN 00189219.

ULLMANN, F. *FiOrdOs: A Matlab toolbox for C-code generation for first order methods.* Tese (Doutorado), 2011.

URRIZA, I. et al. Word length selection method for controller implementation on FPGAs using the VHDL-2008 fixed-point and floating-point packages. *Eurasip Journal on Embedded Systems*, v. 2010, 2010. ISSN 16873955.

VALAVANIS, K. Advances in unmanned aerial vehicles: state of the art and the road to autonomy. *Springer Science & Business Media*, v. 33, 2008. Disponível em: <https://books.google.com.br/books?hl=pt-BR{&}lr={&}id=EsjPyblwMdQC{&}oi=fnd{&}pg=PR11{&}dq=Valavanis,+K.P.:+Advances+in+Unmanned+Aerial+Vehicles:+State+of+the+Art+and+the+Road+to+Autonomy,+vol.+33.+Springer+Science+{\%}26+Business+Media+(2008){&}ots>.

VOUZIS, P. D. et al. A system-on-a-chip implementation for embedded real-time model predictive control. *IEEE Transactions on Control Systems Technology*, v. 17, n. 5, p. 1006–1017, 2009. ISSN 10636536.

VUILLEMIN, J. E. et al. Programmable active memories: Reconfigurable systems come of age.No Title. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, v. 4, n. 1, p. 56–69, 1996.

WORTHMANN, K. et al. Regulation of Differential Drive Robots using Continuous Time MPC without Stabilizing Constraints or Costs. *IFAC-PapersOnLine*, Elsevier B.V., v. 48, n. 23, p. 129–135, jan 2015. ISSN 24058963. Disponível em: <http://www-sciencedirect-com.ez54.periodicos.capes.gov.br/science/article/pii/S2405896315025562http://dx.doi.org/10.1016/j.ifacol.2015.11.272>.

XILINX. *KC705 Evaluation Board for the Kintex-7 FPGA User Guide UG810 v1.9*. [S.l.], 2019. 98 p.

XU, F. et al. Fast nonlinear model predictive control on FPGA using particle swarm optimization. *IEEE Transactions on Industrial Electronics*, v. 63, n. 1, p. 310–321, 2016. ISSN 02780046.

XU, F. et al. FPGA implementation of nonlinear model predictive control. *Control and Decision Conference (...*, v. 108, p. 108–113, 2014. Disponível em: <http://ieeexplore.ieee.org/xpls/abs{_}all.jsp?arnumber=6852>.

XU, R.; OZGUNER, U. Sliding mode control of a quadrotor helicopter. In: *45th IEEE Conference on Decision and Control*. [s.n.], 2006. p. 4957–4962. Disponível em: <http://ieeexplore.ieee.org/abstract/document/4177181/>.

ZHANG, X.; FERRANTI, L.; KEVICZKY, T. An improved primal-dual interior point solver for model predictive control. In: *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*. IEEE, 2017. p. 1126–1131. ISBN 978-1-5090-2873-3. Disponível em: <http://ieeexplore.ieee.org/document/8263807/>.