TESE DE DOUTORADO

# ARQUITETURAS DE HARDWARE PARA ACELERAÇÃO DE ALGORITMOS DE CONTROLE PREDITIVO NÃO-LINEAR

**Renato Coral Sampaio**

**Brasília, outubro de 2018**

**UNIVERSIDADE DE BRASÍLIA**

FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASÍLIA
Faculdade de Tecnologia

TESE DE DOUTORADO

# ARQUITETURAS DE HARDWARE PARA ACELERAÇÃO DE ALGORITMOS DE CONTROLE PREDITIVO NÃO-LINEAR

**Renato Coral Sampaio**

*Tese de Doutorado submetida ao Departamento de Engenharia Mecânica como requisito parcial para obtenção do grau de Doutor em Sistemas Mecatrônicos*

Banca Examinadora

Prof. Dr. Ricardo Pezzuol Jacobi, CIC/UnB _____
*Orientador*

Prof. Dr. Carlos H. Llanos Quintero, ENM/FT/UnB _____
*Examinador interno*

Prof. Dr. Adolfo Bauchspiess, ENE/FT/UnB _____
*Examinador externo*

Prof. Dr. Roberto Zanetti Freire, Escola Politécnica/PUCPR _____
*Examinador externo*

**FICHA CATALOGRÁFICA**

SAMPAIO, RENATO
ARQUITETURAS DE HARDWARE PARA ACELERAÇÃODE ALGORITMOS DE CONTROLE PREDITIVO NÃO-LINEAR [Distrito Federal] 2018.
xvi, 120 p., 210 x 297 mm (ENM/FT/UnB, Doutor, Engenharia Mecânica, 2018).
Tese de Doutorado - Universidade de Brasília, Faculdade de Tecnologia.
Departamento de Engenharia Mecânica

1. Controle Preditivo Não-linear
2. Optimização por enxame de partículas
3. FPGA
4. Sistemas Embarcados
I. ENM/FT/UnB
II. Título (série)

**REFERÊNCIA BIBLIOGRÁFICA**

SAMPAIO, R. (2018). *ARQUITETURAS DE HARDWARE PARA ACELERAÇÃODE ALGORITMOS DE CONTROLE PREDITIVO NÃO-LINEAR*. Tese de Doutorado, Departamento de Engenharia Mecânica, Universidade de Brasília, Brasília, DF, 120 p.

**CESSÃO DE DIREITOS**

AUTOR: Renato Coral Sampaio
TÍTULO: ARQUITETURAS DE HARDWARE PARA ACELERAÇÃODE ALGORITMOS DE CONTROLE PREDITIVO NÃO-LINEAR.
GRAU: Doutor em Sistemas Mecatrônicos ANO: 2018



---

Renato Coral Sampaio
Depto. de Engenharia Mecânica (ENM) - FT
Universidade de Brasília (UnB)
Campus Darcy Ribeiro
CEP 70919-970 - Brasília - DF - Brasil

## Agradecimentos

*À minha querida esposa, Graça, pelo companheirismo evolutivo, amor e imensa paciência ao longo destes últimas anos.*
*Aos meus pais, irmãos e à minha família de modo geral, por todo o aporte e compreensão ao longo da vida.*
*Ao meu orientador, Prof. Ricardo Jacobi pela amizade, confiança e liberdade de pesquisa oferecida ao longo destes anos.*
*Ao Prof. Carlos Llanos pela valiosa amizade, ensinamentos e companheirismo.*
*Aos demais professores que contribuíram com seus ensinamentos sobre os temas das pesquisas aqui desenvolvidas. Em especial ao Prof. André Murilo pelo compartilhamento dos fundamentos de controle preditivo e ao Prof. Leandro Coelho pelas inúmeras referências sobre algoritmos bio-inspirados.*
*Aos meus colegas e amigos de trabalho e pesquisa que contribuíram imensamente no desenvolvimento desta pesquisa, em especial a André Braga, Carlos Eduardo, Daniel Muñoz, Eduardo Mesquita, Fabián Barrera, Guillermo Alvarez, Helon Ayala, Janier Arias, Jones Yudi, Marlon Marques, Oscar Anacona, Sérgio Cruz, Sérgio Pertuz, e a tantos outros que participaram do LEIA - Laboratório de Sistemas Embarcados e Aplicações de Circuitos Integrados da Universidade de Brasília.*
*Aos demais amigos evolutivos intra e extrafísicos pelo companheirismo ao longo das vidas.*
*Ao CNPq e DPI-UnB pelo suporte financeiro.*

*Renato Coral Sampaio*

## RESUMO

O Controle Preditivo Baseado em Modelos (MPC) é uma técnica avançada de controle que vem ganhando espaço tanto na academia quanto na indústria ao longo das últimas décadas. O fato de incorporar restrições em sua lei de controle e de poder ser aplicada tanto para sistemas lineares simples quanto para sistemas não-lineares complexos com múltiplas entradas e múltiplas saídas tornam seu emprego bastante atraente. Porém, seu alto custo computacional muitas vezes impede sua aplicação a sistemas com dinâmicas rápidas, principalmente a sistemas não-lineares embarcados onde há restrições computacionais e de consumo de energia. Baseado nisso, este trabalho se propõe a desenvolver algoritmos e arquiteturas em hardware capazes de viabilizar a aplicação do Controle Preditivo Não-Linear (NMPC) para sistemas embarcados.

Duas abordagens são desenvolvidas ao longo do trabalho. A primeira aplica técnicas de aprendizado de máquina utilizando Redes Neurais Artificiais (RNAs) e Máquinas de Vetor de Suporte (SVMs) para criar soluções que aproximam o comportamento do NMPC em hardware. Neste caso, técnicas para o treinamento das RNAs e SVMs são exploradas com o intuito de generalizar uma solução capaz de lidar com uma ampla faixa de referências de controle. Em seguida, arquiteturas de hardware em ponto-flutuante para a implementação de RNAs do tipo RBF (*Radial Basis Functions*) e SVMs são desenvolvidas juntamente com configurador automático capaz de gerar os códigos VHDL (*VHSIC Hardware Description Language*) das respectivas arquiteturas baseado nos resultados de treinamento e sua topologia. As arquiteturas resultantes são testadas em um FPGA (*Field-Programmable Gate Array*) de baixo custo e são capazes de computar soluções em menos de 1 $\mu$s.

Na segunda abordagem, o algoritmo heurístico de Otimização por Enxame de Partículas (PSO), é estudado e adaptado para etapa de busca da sequência de controle ótima do NMPC. Dentre as modificações estão incluídas a adição de funções de penalização para obedecer às restrições de estados do sistema, o aprimoramento da técnica KPSO (*Knowledge-Based PSO*), denominada KPSO+SS, onde resultados de períodos de soluções de períodos amostragem anteriores são combinados com informações sobre o sinal de controle em estado estacionário e seus valores máximos e mínimos para agilizar a busca pela solução ótima. Mais uma vez, arquiteturas de hardware em ponto-flutuante são desenvolvidas para viabilizar a aplicação do controlador NMPC-PSO a sistemas embarcados. Um gerador de códigos da solução NMPC-PSO é proposto para permitir a aplicação da mesma arquitetura a outros sistemas. Em seguida, a solução é testada para o procedimento de *swing-up* do pêndulo invertido utilizando uma plataforma *hardware-in-the-loop* (HIL) e apresentou bom desempenho em tempo-real calculando a solução em menos de 3 ms. Finalmente, a solução NMPC-PSO é validada em um sistema de pêndulos gêmeos e outro sistema de controle de atitudes de um satélite.



**Palavras-chave:** NMPC. Sistemas não-lineares. PSO. FPGA. RNA. SVM.

## ABSTRACT

Model-based Predictive Control (MPC) is an advanced control technique that has been gaining adoption in industry and the academy along the last few decades. Its ability to incorporate system constraints in the control law and be applied from simple linear systems up to more complex nonlinear systems with multiple inputs and outputs attracts its usage. However, the high computational cost associated with this technique often hinders its use, especially in embedded nonlinear systems with fast dynamics with computational and restrictions. Based on these facts, this work aims to study and develop algorithms and hardware architectures that can enable the application of Nonlinear Model Predictive Control (NMPC) on embedded systems.

Two approaches are developed throughout this work. The first one applies machine learning techniques using Artificial Neural Networks (ANNs) and Support Vector Machines (SVMs) to create solutions that approximate the NMPC behavior in hardware. In this case, ANN and SVM training techniques are explored with the aim to generalize the control solution and work on a large range of reference control inputs. Next, floating-point hardware architectures to implement Radial Basis Function ANNs and SVM solutions are developed along with an automatic architectural configuration too, capable of generating the VHDL (VHSIC Hardware Description Language) codes based on the training results and its topology. Resulting architectures are tested on a low-cost FPGA (Field-Programmable Gate Array) and are capable of computing the solution in under 1 $\mu$s.

In a second approach, the Particle Swarm Optimization (PSO), which is a heuristic algorithm, is studied and adapted to perform the optimal control sequence search phase of the NMPC. Among the main optimizations performed are the addition of penalty functions to address the controlled system state constraints, an improved KPSO (Knowledge-Based PSO) technique named KPSO+SS, where results from previous sampling periods are combined with steady-state control information to speed-up the optimal solution search. Hardware architectures with floating-point arithmetic to enable the application of the NMPC-PSO solution on embedded systems are developed. Once again, a hardware description configuration tool is created to allow the architecture to be applied to multiple systems. Then, the solution is applied to a real-time inverted pendulum swing-up procedure tested on a hardware-in-the-loop (HIL) platform. The experiment yielding good performance and control results and was able to compute the solutions in under 3 ms. Finally, the NMPC-PSO solution is further validated performing a swing-up procedure on a Twin Pendulum system and then on a satellite control platform, a system with multiple inputs and outputs.



**Keywords:** NMPC. Nonlinear systems. PSO. FPGA. RNA. SVM.

# SUMÁRIO

# LISTA DE FIGURAS

# LISTA DE TABELAS

# LISTA DE QUADROS

# LISTA DE SÍMBOLOS

### Símbolos Gregos

| | |
|---|---|
| $\Delta$ | Variação entre duas grandezas similares |
| $\varepsilon$ | Fração muito pequena de uma certa grandeza |

### Grupos Adimensionais

| | |
|---|---|
| $e$ | Número de Euler |

### Subscritos

| | |
|---|---|
| $max$ | Máximo |
| $min$ | Mínimo |

### Sobrescritos

| | |
|---|---|
| $\cdot$ | Variação temporal |
| $\hat{}$ | Valor estimado |

### Arquiteturas de hardware

| | |
|---|---|
| Cordicsincos | Arquitetura CORDIC para as funções seno e cosseno |
| FPadd | *Floating-point Addition/subtraction unit* |
| FPCordic | *Floating-point CORDIC unit* |
| FPdiv | *Floating-point Division unit* |
| FPfrac | *Floating-point Fractional part* |
| FPmul | *Floating-point Multiplication unit* |

### Siglas Gerais

| | |
|---|---|
| ALM | *Adaptive Logic Module* |
| ASIC | *Application Specific Integrated Circuits* |
| CORDIC | *COordinate Rotation DIgital Computer* |
| DSPs | *Digital Signal Processing* |
| FPGA | *Field-Programmable Gate Array* |
| FSM | *Finite State Machine* |
| GPU | *Graphics Processing Unit* |
| HIL | *Hardware-In-the-Loop* |
| HLS | *High Level Synthesis* |
| HW/SW | Hardware/Software |
| LFSR | *Linear Feedback Shift Register* |
| LMPC | *Linear Model Predictive Control* |
| LUT | *Lookup Table* |
| MPC | *Model Predictive Control* |
| MPSoC | *Multiprocessor System-on-a-chip* |
| MSE | *Mean Square Error* |
| NMPC | *Nonlinear Model Predictive Control* |
| PID | *Proportional–Integral–Derivative controller* |
| PLL | *Phase-Locked Loop* |
| PSO | *Particle Swarm Optimization* |
| QP | Problema de programação quadrática (*Quadratic Programming*) |
| RAM | *Random Access Memory* |
| RNA | Rede Neural Artificial |
| ROM | *Read Only Memory* |
| RTL | *Register Transfer Level* |
| SoC | *System-on-a-chip* |
| SVM | *Support Vector Machine* |
| SVR | *Support Vector Regression* |
| VHDL | *VHSIC Hardware Description Language* |
| VHSIC | *Very High Speed Integrated Circuits* |

### Siglas de Equações do MPC

| | |
|---|---|
| J | Função objetivo / custo |
| N | Horizonte de predição |
| Nc | Horizonte de controle |
| Ns | Número de estados do sistema |
| $n_u$ | Número de ações de controle do sistema (atuadores) |
| **x** | Vetor de estados |
| **u** | Vetor de entradas |
| **y** | Vetor de saídas |

### Siglas de Equações do PSO

| | |
|---|---|
| S | Número de partículas do PSO |
| MaxIter | Número máximo de iterações do algoritmo |
| $p$ | Posição da partícula |
| $v$ | Velocidade da partícula |
| $c_1$ | Coeficiente cognitivo |
| $c_2$ | Coeficiente social |
| $lb$ | melhor posição local (*local best*) |
| $gb$ | melhor posição global (*global best*) |

# 1 INTRODUÇÃO

Historicamente, diversas técnicas de controle foram desenvolvidas e aplicadas na solução de problemas em engenharia. Cada vez mais um maior número de dispositivos automatizados aplicam técnicas de controle em áreas como a indústria automotiva, aeronáutica, industrial, robótica, dentre outras. Tendo em vista que um grande número destas aplicações envolve um controlador em um dispositivo embarcado, é bastante compreensível que uma das técnicas mais aplicadas seja o controle Proporcional Integral Derivativo - PID. Por ser computacionalmente muito simples, quando bem ajustada, esta técnica de controle pode atender a uma grande gama de aplicações. Porém, para sistemas não-lineares ou sistemas com múltiplas entradas e saídas, seu desempenho é limitado e o mesmo oferece soluções sub-ótimas, ou seja, soluções longe da ideal. Neste caso, diversas outras técnicas de controle mais sofisticadas foram desenvolvidas ao longo das últimas décadas, incluindo o Controle Adaptativo [1], o Controle Ótimo [2], o Controle Robusto [3], o Controle Inteligente [4] e o Controle Preditivo [5].

Dentre estas, o Controle Preditivo baseado em Modelo ou MPC do inglês *Model-Based Predictive Control* tem ganho bastante espaço nas últimas três décadas e meia [6]. Sua principal característica é o fato de se basear no modelo do sistema ou processo a ser controlado para efetuar uma predição de seu comportamento e, com isso, poder calcular ações de controle otimizadas de acordo com os objetivos estabelecidos por uma função custo.

Do ponto de vista conceitual, é uma técnica relativamente simples de se compreender pois está diretamente relacionada com o modo como os seres humanos agem em diversos contextos. Geralmente, antes de realizar uma ação tem-se em mente algum objetivo, traçam-se algumas possíveis ações e, previsões baseadas nelas são realizadas para antever qual delas mais se aproxima do objetivo. Porém, a cada vez que se realiza uma ação, esta gera consequências e leva a um novo estado, o que deve ser novamente avaliado para que uma próxima ação possa ser tomada até que se alcance o objetivo desejado.

Um exemplo prático e relativamente simples é o de um motorista dirigido um carro. Se o objetivo é seguir uma pista com trechos de retas e curvas, suas possíveis ações são acelerar, frear e girar o volante na direção desejada sem sair da faixa. Geralmente se dirige observando um horizonte de algumas dezenas ou centenas de metros à frente do veículo (horizonte de predição). Ao se aproximar de uma curva, sabe-se que é necessário reduzir a velocidade antes mesmo de chegar ao ponto de girar o volante. Portanto, este é um procedimento de otimização de ações para poder realizar o movimento desejado. Em um contra exemplo, caso haja neblina na pista, o horizonte de visão (predição) fica reduzido e a capacidade de antever as consequências das ações fica limitada. Neste caso, ou se reduz a velocidade ou será difícil evitar um acidente.

Apesar de simples, este conceito utilizado no MPC é bastante poderoso e apresenta diversas vantagens em relação a outras técnicas de controle. Dentre elas, conforme descrito por Rossiter

[5], podemos destacar: o tratamento de dinâmicas complexas (lineares e não-lineares); a habilidade para o tratamento de problemas com múltiplas entradas e/ou saídas (MIMO - *Multiple Input/ Multiple Output*); e a incorporação de restrições, ou seja, a definição de limites para as diversas variáveis envolvidas (entradas, estados, saídas) diretamente na formulação da lei de controle.

A aplicação do MPC na indústria de processos é bastante aceita, como foi evidenciado pela pesquisa de Qin e Badwell [7], pois a técnica provê melhores resultados com economia de recursos. A saída do controlador fica mais próxima do valor desejado, obedece restrições, e portanto gera uma solução de maior qualidade e eficiência.

Porém, por trás de todas estas qualidades há um preço a ser pago que é o alto custo computacional deste algoritmo. Pelo fato de se buscar a cada instante de decisão a melhor solução dentre um grande conjunto de possibilidades, há a necessidade de solucionar um problema de otimização a cada período de amostragem, o que requer do MPC um esforço computacional muito maior ao se comparar, por exemplo, a um simples controlador PID. Características do sistema como o número de estados, o número de saídas controladas, o tamanho do horizonte de predição, o número de restrições e a linearidade ou não-linearidade do problema são todos fatores que afetam esta complexidade.

Por este motivo, historicamente o MPC começou a ser aplicado na indústria química, onde a dinâmica dos processos controlados é geralmente lenta com períodos de amostragem medidos em minutos, como pode ser observado neste *survey* de 1997 [8]. Em um *survey* mais recente, Mayne [9] apresenta os grandes avanços tanto da teoria quanto da aplicação do MPC na indústria nas últimas décadas. Na área acadêmica destacam-se mais de 5 livros dedicados ao tema, publicados após os anos 2000 [5, 10, 11, 12, 13] e milhares de artigos. Dentre as variações do MPC destaca-se seu uso a sistemas lineares [14], sistemas não-lineares [12], sistemas híbridos [15], sistemas tolerantes a falhas [16] e controle robusto [17]. Além disso em [18] Ławryńczuk destaca uma lista de aplicações como em robótica móvel [19], controle de emissões de motores a diesel [20], controle de redes TCP/IP [21], veículos elétricos [22], sistemas de condicionamento de ar [23] e veículos aéreos não tripulados [24].

Em paralelo, o desenvolvimento de hardware para sistemas embarcados também passou por uma grande evolução nas últimas décadas. Além da miniaturização cada vez maior de transistores possibilitando circuitos integrados cada vez mais densos, uma grande quantidade de arquiteturas tem sido exploradas partindo de otimizações em Processadores de Propósito Geral (GPPs), passando pelo uso de Unidades de Processamento Gráfico (GPUs) aplicada à computação paralela, Processadores de Sinais Digitais (DSPs) e até arquiteturas mais complexas como *System-on-Chip* (SoC) ou até *Multiprocessor System-on-Chip* (MPSoC) sendo implementadas em dispositivos reconfiguráveis como os *Field Programmable Gate-Arrays* (FPGAs) ou fabricadas diretamente em silício, por meio da tecnologia de *Standard Cells* [25].

Mesmo com o grande avanço na capacidade computacional de processadores embarcados, implementar um controlador MPC atendendo aos requisitos de tempo-real ainda não é uma tarefa trivial. Normalmente, se sacrifica uma solução ótima por uma aproximação para conseguir res-

peitar o período de amostragem do sistema. Porém, alcançar o tempo-real em muitos casos não é o suficiente pois em muitos sistemas embarcados as restrições de consumo de energia são críticas, a exemplos de robôs móveis ou VANTs (Veículos aéreos não tripulados).

No geral, para o caso do MPC Linear (LMPC), quando se deseja trabalhar com frequências de amostragem mais rápidas, por exemplo na faixa dos kilohertz ou até megahertz, parte-se para soluções em arquiteturas dedicadas sejam elas em silício (ASIC) ou em FPGA. Para o caso do MPC não-linear (NMPC) este ainda é um problema aberto. Em termos de algoritmo, é comum adotar uma estratégia de linearização a cada iteração para que seja possível utilizar técnicas do LMPC conforme discutido por Gros [26]. Porém, estas estratégias são limitadas e requerem um palpite inicial (*initial guess*) para que o algoritmo seja capaz de obter uma solução. Outras duas abordagens que se destacam para o caso do NMPC de alta frequência são: o uso de técnicas de aprendizado de máquina com o emprego de Redes Neurais Artificiais (RNAs) ou Máquinas de Vetor de Suporte (SVMs) para o cálculo de soluções aproximadas, como abordado por Ortega e Camacho [27]; através do uso de algoritmos heurísticos de busca global, como é o caso, por exemplo, do algoritmo de otimização por enxame de partículas (PSO). Em ambos os casos, a soluções em tempo real tem maior possibilidade de serem alcançadas quando implementadas em hardware dedicado.

## 1.1 MOTIVAÇÃO DO TRABALHO

Observando o histórico tanto da pesquisa do MPC na academia quanto do seu desenvolvimento e aplicação na indústria, é possível verificar que sua complexidade computacional fez com que esta técnica fosse, inicialmente, aplicada a problemas com dinâmicas lentas como, por exemplo, em processos químicos. Mesmo assim, a maioria das aplicações se limitava a um curto horizonte de predição e, em sua maioria, a sistemas que pudessem ser resolvidos com uma abordagem linear. Ao longo dos últimos anos, tanto a teoria do MPC Linear (LMPC) quanto a teoria não-linear (NMPC) foram se aprimorando. Abordagens para acelerar o cálculo, principalmente do LMPC surgiram primeiro. Porém, de acordo com uma pesquisa de opinião publicada em 2016 [28] praticamente 3 a cada 4 aplicações na indústria precisam resolver problemas não-lineares em que controladores lineares podem se aproximar de uma solução mas não são totalmente adequados.

Além disso, uma grande parte destas aplicações é controlada por sistemas embarcadas onde há restrições de poder de processamento, uso de memória e baixo consumo de energia do processador. Neste contexto, o uso de processadores de propósito geral ou mesmo de GPUs, ambos capazes de processaor o algoritmo MPC de maneira rápida, não se aplica pelo seu alto consumo de energia na faixa de dezenas ou até centenas de watts [29, 30]. Neste âmbito, observa-se um movimento mais recente na última década não só para desenvolver melhores algoritmos para o NMPC quanto para melhorar sua implementação, alcançando um maior desempenho em aplicações embarcadas como pode ser visto pelo projeto TEMPO - (*Training in Embedded Predictive*

*Control and Optimization*) [31].

Nesta linha de pesquisa, este trabalho se propõe a explorar algoritmos e abordagens de aceleração em hardware buscando melhorar o desempenho de controladores NMPC, tornando a aplicação desta abordagem de controle viável para sistemas embarcados trabalhando com o consumo de energia na faixa dos miliwatts.

## 1.2 OBJETIVOS

### 1.2.1 Objetivo Geral

Este trabalho se propõe a desenvolver arquiteturas em hardware reconfigurável (FPGAs) capazes de aplicar técnicas de Controle Preditivo baseado em Modelos Não-Lineares (NMPC) em tempo-real com baixo consumo de energia utilizando duas abordagens. A primeira explora técnicas de aprendizado de máquinas com RNAs e SVMs para problemas que necessitam de altas frequências (períodos de amostragem na faixa dos $\mu$s). A segunda explora o algoritmo PSO, para problemas com períodos de amostragem na faixa de milissegundos.

### 1.2.2 Objetivos Específicos

Os objetivos específicos são separados em duas etapas, uma para cada abordagem descrita nos objetivos gerais. Na primeira, visando a abordagem de aprendizado de máquina temos:

- Desenvolver o algoritmo de controle preditivo não-linear em linguagem de alto nível (Matlab) para validação de modelos e estudos de caso.
- Gerar conjuntos de dados e elaborar procedimentos de treinamento de RNAs / SVMs que aproximem o comportamento do NMPC.
- Desenvolver arquiteturas em hardware para a implementação de RNAs e SVMs.
- Validar o desempenho do sistema em simulações.

Na segunda abordagem, utilizando um algoritmo heurístico de busca global, temos:

- Desenvolver o algoritmo NMPC baseado no PSO (NMPC-PSO) em linguagem de alto nível (Matlab) para validação de modelos e estudos de caso.
- Desenvolver o algoritmo NMPC-PSO em linguagem compilada de alto desempenho (C/C++) para ser utilizado em processadores embarcados verificando sua performance de controle em simulações.
- Desenvolver a arquitetura de hardware do algoritmo NMPC-PSO e verificar seu desempenho de controle na plataforma hardware-in-the-loop (HIL).

## 1.3 METODOLOGIA

No desenvolvimento deste trabalho, foi aplicada a seguinte metodologia:

1. *Revisão Bibliográfica*: Foi realizada uma revisão bibliográfica do estado da arte a respeito de algoritmos MPC Lineares e Não-lineares ressaltando os principais trabalhos que implementam estes algoritmos de modo efetivo em sistemas embarcados e as abordagens mais promissoras para NMPC. Em seguida, uma revisão bibliográfica específica sobre o uso de aprendizado de máquinas e de algoritmos heurísticos com o PSO no problema NMPC são investigadas.

2. *Desenvolvimento da solução NMPC em software:* Inicialmente foi elaborado um algoritmo NMPC em Matlab para ser utilizado como referência o qual utilizou a função *Fmincon* (*Find minimum of constrained nonlinear multivariable function*) do próprio Matlab como otimizador para problemas não-lineares com restrições. Juntamente com este algoritmo, foi adotado como estudo de caso um sistema de um pêndulo invertido apresentado por Mercieca e Fabri em [32].

3. *NMPC utilizando técnicas de aprendizado de máquina*: A implementação em hardware foi desenvolvida em linguagem VHDL (*VHSIC Hardware Description Language*) e testada tanto de maneira independente na ferramenta QuestaSim, Quartus II e ISE quanto integrada ao simulador do sistema em Matlab a partir da *Toolbox HDL Verifier* onde seu desempenho em termos de velocidade e de precisão foram analisados para várias tamanhos de palavra em ponto-flutuante.

4. *NMPC utilizando o algoritmo PSO:* A solução do NMPC foi inicialmente elaborada e validada em Matlab com as adaptações necessárias para atender às restrições das ações de controle e de estados. Em seguida, foram realizados ajustes em seus parâmetros para otimizar o tempo de cálculo do algoritmo ao mínimo possível para encontrar uma solução satisfatória. Em seguida, uma versão em C++ foi elaborada para ser embarcada em processadores ARM (*Acorn RISC Machine* ou *Advanced RISC Machine*).

5. *Arquitetura dedicada em hardware*: A implementação em hardware para ambas as abordagens foi desenvolvida utilizando a linguagem de descrição de hardware VHDL utilizando aritmética de ponto-flutuante. Cada módulo em VHDL foi testado individualmente e em conjunto a partir de *testbenches* específicos utilizando a ferramenta de simulação QuestaSim. Em seguida, testes integrados ao simulador do Matlab com a simulação do sistema foram realizados através da *Toolbox* HDL Verifier onde seu desempenho em termos de velocidade e de precisão foram analisados para várias tamanhos de palavra em ponto-flutuante. Finalmente, as arquiteturas foram sintetizadas para FPGA utilizando a ferramenta Quartus Prime da Intel.

6. *Validação das soluções*: A última etapa foi a de verificação das soluções utilizando a plataforma *Hardware-in-the-loop* (HIL) que executa o sistema em tempo-real Simulink Real-

Time (SLRT) do Matlab. Esta plataforma é responsável por simular o sistema estudado próximo ao tempo contínuo enquanto o controlador é executado na placa FPGA conectado a ela.

## 1.4 CONTRIBUIÇÕES DO TRABALHO

Este trabalho originou um conjunto de publicações listadas no Apêndice A, cujos conteúdos foram incorporados a este documento. Além disso, há ainda resultados submetidos para publicações e outros a serem publicados. A seguir, listamos as principais contribuições alcançadas:

1. *Utilização de RNAs e SVMs para aproximar o comportamento do controlador NMPC com altas taxas de amostragem em hardware:* a primeira abordagem teve o objetivo de gerar soluções NMPC para sistemas com altas frequências de amostragem (MHz). Para isto foram empregados métodos com aprendizado de máquina com o intuito de realizar um treinamento *offline* a partir de um controlador NMPC e utilizar os resultados em um sistema de controle *online*. Resultados iniciais empregando RNAs e SVMs foram alcançados, porém limitados em seu escopo onde somente trajetórias pré-definidas puderam ser replicadas na solução final. Posteriormente, a ferramenta NIOTS, desenvolvida por Santos [33] para o treinamento de SVMs, foi utilizada e seu método de treinamento foi aprimorado para as necessidades específicas do controlador NMPC. Neste caso, foi possível generalizar uma solução para o controle de posição de um pêndulo invertido onde entradas degrau de amplitudes variadas são seguidas pelo sistema. Com esta abordagem, um gerador de arquiteturas de hardware (VHDL) para construir RNAs do tipo RBF e SVMs com *kernel* do tipo RBF Gaussiano e Polinomial foi desenvolvido. O gerador é capaz de gerar código com diversas configurações com um número arbitrário de entradas, e neurônios/*kernels* na estrutura das soluções.

   Posteriormente, para além do escopo deste trabalho, o mesmo gerador foi aprimorado para gerar redes do tipo *Perceptron* com 3 camadas e aplicado na solução de um sistema de fusão sensorial para determinar a largura e profundidade de cordões de solda em tempo-real.

2. *Desenvolvimento de Algoritmo NMPC com o PSO:* a segunda abordagem do trabalho explorou o emprego de algoritmos heurísticos como parte da solução de busca da sequência de comandos ótima do NMPC. Neste caso, apresenta-se uma proposta de modificação do algoritmo PSO para se adequar aos requisitos de restrições de comando e restrições de estados utilizando funções de penalização. Em seguida se desenvolve uma nova abordagem de aplicação da técnica KPSO, denominada KPSO+SS onde, além dos resultados de períodos de amostragem anteriores, informações sobre o sinal de controle em estado estacionário e seus valores máximos e mínimos são introduzidos como possíveis soluções ao início de cada período de amostragem para agilizar a busca pela solução ótima. Finalmente, ajustes nos parâmetros coeficientes cognitivo $c1$ e social $c2$ do PSO são realizados para aprimorar o desempenho do algoritmo. Todos estes resultados são validados e empregados a um estudo

de caso de *swing-up* de um pêndulo invertido.

3. *Implementação em Hardware da abordagem NMPC-PSO:* com a solução NMPC-PSO validada, partiu-se para a implementação de sua arquitetura em hardware com o intuito viabilizar sua aplicação a sistemas com dinâmicas rápidas em tempo-real. Propostas de paralelismo do algoritmo foram elaboradas com o intuito de acelerar ao máximo sua execução e ainda manter um baixo consumo de recursos de hardware. Duas implementações são propostas em VHDL e testadas em FPGA. A primeira solução utiliza uma abordagem de *co-design* onde a função custo é implementada em hardware em paralelo e o algoritmo PSO é executado em software. Numa segunda abordagem, a solução completa do NMPC-PSO é implementada em hardware, gerando maiores ganhos de desempenho. Um gerador de códigos da solução NMPC-PSO é desenvolvido para permitir a aplicação da mesma arquitetura a outros sistemas. Finalmente, a solução é testada para o procedimento de *swing-up* do pêndulo invertido utilizando uma plataforma *hardware-in-the-loop* (HIL) e apresentou bom desempenho em tempo-real.

4. *Aplicação da solução NMPC-PSO a outros sistemas*: por fim, a solução NMPC-PSO utilizando a técnica KPSO+SS é empregada a dois outros sistemas, um de pêndulos gêmeos e outro de controle de atitudes de um satélite, sendo este último um problema de múltiplas entradas e múltiplas saídas. Ambas as soluções são testadas e validadas apresentando desempenho satisfatório na plataforma *hardware-in-the-loop* (HIL).

## 1.5 ORGANIZAÇÃO DO TRABALHO

O presente trabalho está organizado da seguinte maneira:

No Capítulo 2, uma introdução à teoria de controle preditivo tanto linear quanto não-linear é apresentada e acompanhada de uma revisão bibliográfica do estado da arte focado em aplicações do controle preditivo em sistemas embarcados, ressaltando as técnicas mais promissoras. Além disso, são discutidos os fundamentos de técnicas de aprendizado de máquina (especificamente de RNAs e SVMs) e uma introdução ao algoritmo PSO. O Capítulo 3 apresenta a abordagem de treinamento de RNAs e SVMs para aproximar o comportamento do controlador NMPC juntamente com sua implementação em hardware e desempenho. O Capítulo 4 apresenta a abordagem de adaptação do algoritmo PSO para atuar como otimizador não-linear do NMPC e sua validação com um estudo de caso. No Capítulo 5 é apresentada o processo de desenvolvimento do hardware dedicado da solução NMPC-PSO juntamente com sua validação. Já o Capítulo 6 apresenta outros estudos de caso utilizando a abordagem NMPC-PSO. Finalmente, no Capítulo 7 apresentam-se as conclusões e perspectivas futuras do trabalho.

# 2 FUNDAMENTAÇÃO TEÓRICA

## 2.1 CONTROLE PREDITIVO BASEADO EM MODELOS

A ideia de controle preditivo ou MPC (*Model Predictive Control*) é relativamente antiga. Os trabalhos de Propoi [34] e de Rafal e Stevens [35] datados de 1963 e 1968, respectivamente, são considerados por Nikolaou [36] como precursores na área. O termo MPC não se refere a um só algoritmo mas a uma família de técnicas de controle baseadas em predição a partir de modelos, como é observado por Rossiter em [5]. Segundo Allgöwer [37], as primeiras aplicações surgem na indústria em 1976 com a publicação de Richalet et al. [38] descrevendo a técnica MPHC - *Model Predictive Heuristic Control* seguida da publicação de Cutler e Ramaker [39] em 1979 descrevendo o software DMC - *Dynamic Matrix Control*, ambas ainda sem o tratamento de restrições do processo. Posteriormente, surge o algoritmo QDMC - *Quadratic Dynamic Matrix Control* (1986) [40] que já considera restrições e utiliza a programação quadrática para o cálculo da solução ótima. Na academia destacam-se os trabalhos de Keerthi e Gilbert (1988) [41] com o estudo de estabilidade em tempo discreto e o trabalho de Mayne e Michalska (1990) [42] para sistemas não-lineares em tempo contínuo. Além destas, podem ser listadas ainda as técnicas IMC - Internal Model Control (1976) [43], MUSMAR (1984) [44] e GPC - Generalized Predictive Control (1987) [45]. Outra maneira pela qual o controle preditivo é conhecido é por RHC - *Receding Horizon Control* ou, em português, Controle de Horizonte Deslizante [46].

De maneira geral o MPC se baseia na utilização do modelo dinâmico do sistema a ser controlado para prever seu comportamento futuro e com isso poder decidir qual o melhor conjunto de ações que levam o sistema ao estado desejado. Este procedimento é ilustrado na Figura 2.1.

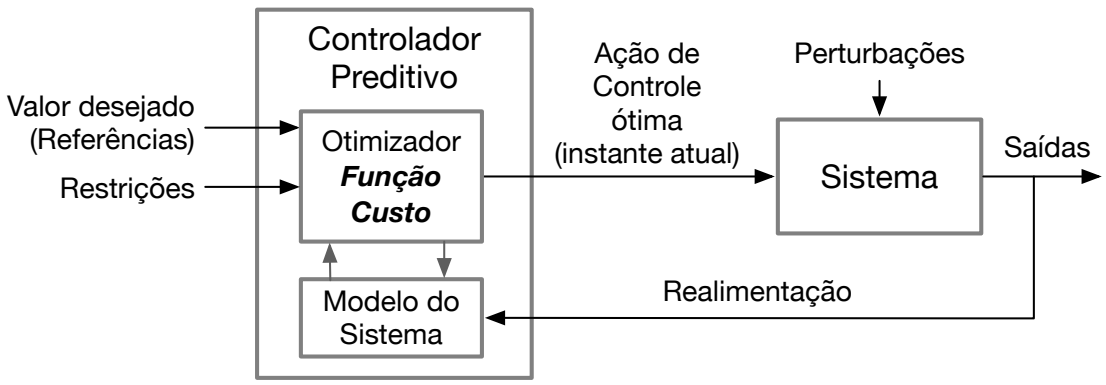



Figura 2.1: Esquema do controlador preditivo.

Segundo Rossiter [5], para realizar estas tarefas se faz necessário definir:

- O modelo de predição do sistema (linear ou não-linear) que estime o valor de estados futuros a partir da entrada ao longo de um Horizonte de Predição ($N$), ou seja, ao longo dos

próximos $N$ períodos de amostragem futuros;

- O conjunto de restrições aplicáveis ao sistema;
- A função objetivo $\boldsymbol{J}$, que sirva de referência, definindo os critérios de desempenho e objetivos a serem alcançados pelo sistema;
- A determinação da lei de controle, ou seja, a formulação matemática para definir a ação de controle.

### 2.1.1 Modelo de Predição

O modelo de predição é responsável por descrever as mudanças de estado do sistema baseado nas ações de controle. Este modelo deve representar a dinâmica do sistema. Quando uma representação matemática é possível ela geralmente é descrita por um sistema de equações diferenciais (*white-box*). Neste caso, alguns modelos utilizados na literatura incluem: Resposta Truncada ao Impulso (IRT), função de transferência e o modelo em espaço de estados. Quando o acesso ao modelo físico ou uma expressão matemática analítica não são possíveis, pode-se trabalhar com sistemas identificados experimentalmente (*black-box*) que podem ser representados, por exemplo, por redes neurais artificiais. Além disso, é possível ainda utilizar modelos mistos (*grey-box*) onde parte do funcionamento do sistema é conhecido e modelado matematicamente e outra parte é identificada experimentalmente [5]. Uma representação geral do modelo de predição em tempo discreto pode ser vista na Equação (2.1)

$$\mathbf{x}(k+1) = f(\mathbf{x}(k), \mathbf{u}(k)), \quad \mathbf{y}(k) = h(\mathbf{x}(k)) \tag{2.1}$$

onde $\mathbf{x}(k) \in \mathbb{R}^n$ é o vetor de $n$ estados do sistema no instante de amostragem $k \in \mathbb{N}$, $\mathbf{u}(k) \in \mathbb{R}^m$ é o vetor contendo as $m$ de entradas do sistema e $\mathbf{y}(k) \in \mathbb{R}^{n_y}$ representa o vetor de $n_y$ saídas do sistema. Além disso, $f : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ é a função que determina o valor do vetor de estados no instante $k+1$ a partir de $\mathbf{x}(k)$ e $\mathbf{u}(k)$, e $h$ é a função que mapeia o vetor de estados na saídas.

Com o modelo de predição descrito, é necessário definir o **horizonte de predição** ($N$), ou seja, quantos períodos de amostragem à frente serão preditos pelo controlador. Quanto maior o horizonte, maior será a capacidade de antecipar eventos futuros e otimizar as ações de controle. Porém, quanto maior o horizonte, maior o custo computacional associado e mais difícil fica a implementação prática do controlador em tempo-real. Por este motivo, geralmente se considera um tamanho de horizonte suficiente que abarque o tempo de estabilização da resposta dinâmica do sistema.

Considerando o horizonte de predição, o estado atual do sistema ($\mathbf{x}(k)$) e um conjunto de $N$ vetores de entrada $\tilde{\mathbf{u}}(k)$ representados na Equação (2.2), é possível utilizar o modelo de predição de um passo à frente (2.1) de maneira iterativa para estimar o valor tanto dos estados futuros $\tilde{\mathbf{x}}(k)$

quanto das saídas futuras do sistema $\tilde{\mathbf{y}}(k)$.

$$\tilde{\mathbf{u}}(k) = [\mathbf{u}(k), \mathbf{u}(k+1), ..., \mathbf{u}(k+N-1)] \in \mathbb{R}^{N.m} \tag{2.2}$$

$$\tilde{\mathbf{x}}(k) = [\mathbf{x}(k+1), ..., \mathbf{x}(k+N)] \in \mathbb{R}^{N.n} \tag{2.3}$$

$$\tilde{\mathbf{y}}(k) = [\mathbf{y}(k+1), ..., \mathbf{y}(k+N)] \in \mathbb{R}^{N.n_y} \tag{2.4}$$

### 2.1.2 Definição das Restrições

Como mencionado anteriormente, uma das principais vantagens do MPC é sua capacidade de lidar com restrições de maneira sistemática [5]. Esta é uma característica importante já que sistemas físicos tem restrições naturais como é o caso de atuadores que tem força limitada ou onde se deve respeitar os limites de variáveis físicas como pressão e temperatura. Esta inclusão sistemática de restrições na lei de controle permite o projeto de um controlador mais seguro e previsível que, por definição, não irá ultrapassar limites estabelecidos.

É possível definir restrições para os estados, entradas e saídas do sistema. Em muitos casos, formula-se o problema de tal modo que a saída seja uma combinação linear dos estados e, neste caso, as restrições impostas aos estados já incluem as saídas. Restrições de estados podem ser definidas por:

$$\mathbf{x}(k) \in \mathbb{X} \subset \mathbb{R}^n \tag{2.5}$$

onde $\mathbb{X}$ é um subconjunto convexo fechado definido no intervalo $[\mathbf{x}_{min}, \mathbf{x}_{max}]$ em que $\mathbf{x}_{max}$ e $\mathbf{x}_{min}$ representam os vetores de valores máximos e mínimos dos estados, respectivamente.

Já o sinal de controle **u** está restrito por:

$$\mathbf{u}(k) \in \mathbb{U} \subset \mathbb{R}^m \tag{2.6}$$

onde $\mathbb{U}$ representa é um subconjunto convexo fechado definido no intervalo $[\mathbf{u}_{min}, \mathbf{u}_{max}]$ em que $\mathbf{u}_{max}$ e $\mathbf{u}_{min}$ representam os vetores de valores máximos e mínimos que as entradas do sistema podem assumir, respectivamente.

Além disso, é comum a necessidade restringir a taxa de variação das ações de controle ($\Delta u$), conforme as Equação (2.7).

$$|\Delta\mathbf{u}(k+i-1)| \leq \Delta\mathbf{u}_{max}, i \in [1, Nc] \tag{2.7}$$

onde $\Delta\mathbf{u}_{max}$ representa o vetor com as variações máximas das entradas entre os períodos de amostragem.

Nota-se ainda a presença da contante $Nc$ que representa o Horizonte de Controle limitando o intervalo das ações de controle. Isto ocorre pois no MPC é possível definir um horizonte de controle menor ou igual ao horizonte de predição ($Nc \leq N$). Isto significa que o número de graus de liberdade para definir a sequência de ações de controle se limita a $Nc$ e que, caso $Nc < N$,

os valores do sinal de controle no intervalo $[\mathbf{u}(Nc+1), ..., \mathbf{u}(N-1)]$ se repetem e são iguais a $\mathbf{u}(Nc)$.

A viabilidade de uma solução on-line de otimização do MPC depende do tamanho e complexidade do problema. Existem casos em que não é possível resolver o problema com restrições *duras*, ou seja, obedecendo estritamente as restrições e portanto, é necessário relaxar algumas restrições para que a solução seja viável no tempo requerido. Uma maneira de fazer isto é relaxar as restrições da saída do sistema por um $\epsilon$ (constante de erro) conforme apresentado por [47]. Este caso específico não será tratado aqui.

### 2.1.3 Função Objetivo

Com a possibilidade de calcular $\tilde{\mathbf{x}}(k)$ e $\tilde{\mathbf{y}}(k)$ a partir de $\tilde{\mathbf{u}}(k)$, se faz necessário definir critérios objetivos de desempenho para que o controlador possa selecionar a melhor solução $\boldsymbol{u}^{opt}$ a ser aplicada ao sistema no próximo período de amostragem $(k+1)$. Para isto, define-se uma função custo a partir de uma combinação linear destes critérios a fim de gerar um valor escalar que possa ser minimizado. Os critérios de desempenho mais utilizados, incluindo considerações de estabilidade discutidas em [48] e [5], são: o **erro de posição** ($\mathbf{e}(k) = \tilde{\mathbf{y}}(k) - \mathbf{y}_{ref}(k)$), que considera a posição estimada de um vetor de saída (representado por uma combinação linear dos estados do sistema) em relação a um vetor de referência ($\mathbf{y}_{ref}(k)$); o **desvio da entrada** ($\hat{\mathbf{u}}(k) = \mathbf{u}(k) - \mathbf{u}_{ss}$) que calcula a diferença entre a entrada atual e a entrada necessária para manter o sistema equilibrado em estado estacionário ($\mathbf{u}_{ss}$); e a **taxa de variação da entrada** ($\Delta u = \mathbf{u}(k) - \mathbf{u}(k-1)$).

Uma possível função custo pode ser descrita pela Equação (2.8), onde uma soma ponderada dos índices ao quadrado é utilizada para a obtenção de uma função convexa, que facilita o cálculo de um valor mínimo global.

$$J(\mathbf{x}(k), \mathbf{u}(k)) \triangleq \sum_{i=1}^{N} ||\mathbf{e}(k)||_Q^2 + \sum_{i=0}^{Nc} ||\hat{\mathbf{u}}(k)||_R^2 \tag{2.8}$$

onde $Q$ e $R$ são matrizes diagonais de peso para ponderar a saída.

Porém, conforme descrito por Mayne et al. [48] a função custo descrita acima não garante a estabilidade do sistema. Neste caso, uma das maneiras de solucionar este problema reside no uso de um termo adicional chamado de custo terminal. Este custo se refere a uma ponderação específica ao estado do sistema no final do horizonte de predição $\mathbf{x}(k+N)$. Além do custo terminal, para garantir a estabilidade ainda é necessário impor restrições específicas ao valor deste estado da seguinte maneira:

$$\mathbf{x}(k+N) \in \mathbb{X}_f \subset \mathbb{R}^n. \tag{2.9}$$

Neste caso, a função custo pode ser re-escrita como:

$$J(\mathbf{x}(k), \mathbf{u}(k)) \triangleq \sum_{i=1}^{N-1} ||\mathbf{e}(k)||_Q^2 + \sum_{i=0}^{Nc-1} ||\hat{\mathbf{u}}(k)||_R^2 + ||\mathbf{x}(N) - \mathbf{x}_{ss}||_{Q_f}^2 \tag{2.10}$$

onde $Qf$ é a matriz diagonal de ponderação do custo terminal e $\mathbf{x}_{ss}$ o valor de $\mathbf{x}$ em estado estacionário.

### 2.1.4 Lei de Controle

Combinando os elementos anteriores, podemos definir a lei de controle onde o estado do sistema $\mathbf{x}(k)$ é medido e, a partir da minimização da função custo, determina-se o vetor de saída $\boldsymbol{u}^{opt}$ a ser aplicado ao sistema, conforme a Equação (2.11).

$$\begin{aligned} \boldsymbol{u}^{opt} = min_{u_{k,k+N-1}} J(\mathbf{x}(k), \mathbf{u}(k, k+N-1)), \quad \text{sujeito a:} \\ \mathbf{u}(k) \in \mathbb{U}^N,\ \mathbf{x}(k) \in \mathbb{X} \ \text{ e } \ \mathbf{x}(k+N) \in \mathbb{X}_f. \end{aligned} \tag{2.11}$$

Portanto, podemos definir os passos gerais do algoritmo MPC como:

1. Medir/Estimar os estados do processo $\mathbf{x}(k)$;
2. Calcular o vetor $\boldsymbol{u}^{opt}$ a partir da minimização da função custo $\mathbf{J}$;
3. Definir $\boldsymbol{u}(k) = \boldsymbol{u}^{opt(1)}$, ou seja, o primeiro termo da sequência de ações de controle, e aplicá-lo ao sistema;
4. Avançar um período de amostragem ($k = k + 1$) e continuar no passo 1.

Uma representação gráfica da estratégia do MPC pode ser observada na Figura 2.2. No gráfico superior é possível observar a linha contínua preta representando os estados passados do sistema, enquanto no gráfico inferior a linha contínua preta representa as ações de controle passadas. A partir do instante $k$ a referência $y^{ref}$ a ser seguida pelo sistema (linha pontilhada vermelha) sofre uma alteração e, utilizando a lei de controle, calcula-se o vetor $\boldsymbol{u}^{opt}$ com $N$ ações de controle, representado no gráfico inferior pela linha contínua azul. A partir de $\boldsymbol{u}^{opt}$ são calculado os $N$ valores de $\hat{\mathbf{y}}$ representados também em azul na parte superior, o que mostra a predição do comportamento do sistema para que a referência seja alcançada. Observa-se ainda na figura a notação $\hat{\mathbf{y}}(k+1|k)$ que significa a predição do valor de $\hat{\mathbf{y}}$ no período de amostragem $k+1$ calculado no instante $k$. Finalmente, o primeiro valor da ação de controle $\boldsymbol{u}^{opt(1)} = \mathbf{u}(k|k)$, representado em verde é aplicado ao sistema.

A seguir, detalhamos as particularidades dos casos de MPC Linear e Não-Linear, apresentando ainda o estado da arte com as abordagens para a solução de cada um dos respectivos problemas de otimização.

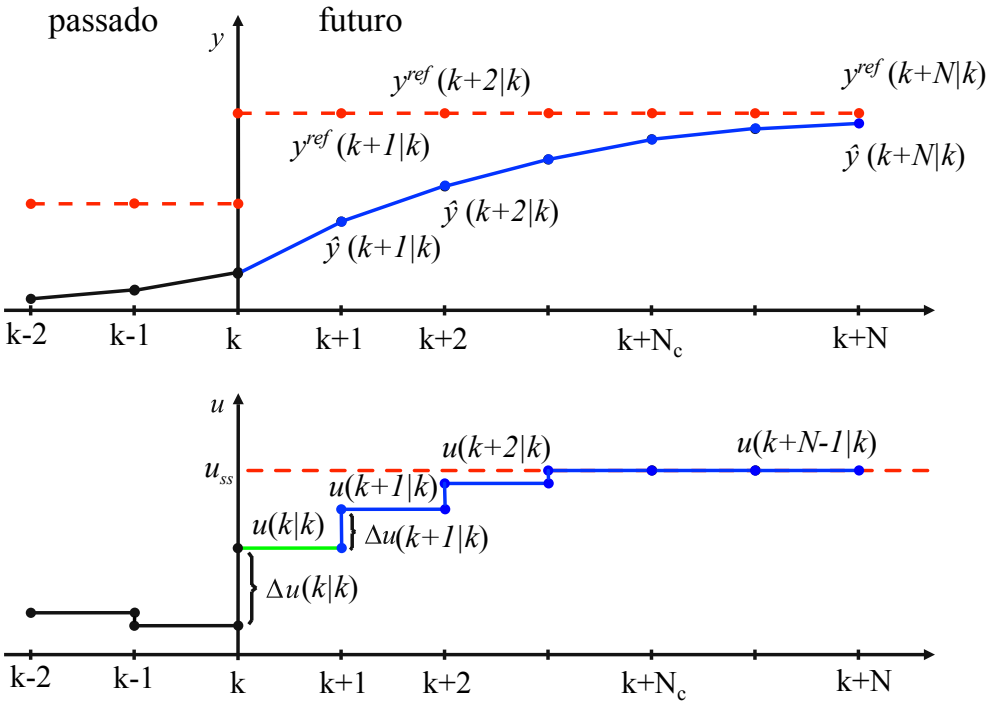



Figura 2.2: O princípio do Controle Preditivo. Adaptado de [49].

## 2.2 CONTROLE PREDITIVO LINEAR

Considerando o problema de um sistema Linear e Invariante no Tempo (LTI), é comum observar na literatura a representação em espaço de estados conforme a Equação 2.12. Neste caso, como exemplo, vamos utilizar ainda a representação em tempo discreto já que esta será a representação adotada no desenvolvimento deste trabalho.

$$\begin{aligned} \mathbf{x}(k+1) &= A\mathbf{x}(k) + B\mathbf{u}(k), \\ \mathbf{y}(k) &= C\mathbf{x}(k) + D\mathbf{u}(k) + d(k) \end{aligned} \tag{2.12}$$

onde $\mathbf{x}(k)$ e $\mathbf{x}(k+1)$ representam os vetores do estado atual e o próximo estado do sistema, respectivamente, $\mathbf{u}(k)$ é o vetor de entradas do sistema, $\mathbf{y}(k)$ o vetor de saídas, e $d(k)$ os ruídos de medida sistema. As matrizes A, B, C e D representam a relação entre as variáveis de estado, entrada e saída do sistema. Para a maioria dos sistemas, assume-se ainda que D = 0, ou seja, não há alimentação direta da entrada do sistema **u** para a saída $\mathbf{y}(k)$ e portanto, esta só vai depender da entrada $\mathbf{x}(k)$ somado os ruídos $d(k)$.

### 2.2.1 Cálculo da Predição Linear

Modelos discretos como o modelo de espaço de estados representado acima são modelos de predição de um passo à frente pois, a partir de $\mathbf{x}(k)$ e $\mathbf{u}(k)$, calculamos $\mathbf{x}(k+1)$. Conforme elaborado em [5], para encontrar $\mathbf{y}(k+1)$ basta utilizar o valor de $\mathbf{x}(k+1)$ conforme a Equação

2.13.

$$\mathbf{y}(k+1) = C\mathbf{x}(k+1) + d(k+1), \tag{2.13}$$

substituindo a primeira equação de 2.12 em 2.13 temos:

$$\mathbf{y}(k+1) = CA\mathbf{x}(k) + CB\mathbf{u}(k) + d(k+1), \tag{2.14}$$

Assumindo ainda que a perturbação tem variação pequena, é possível assumir $d(k+1) = d(k)$.

Continuando na mesma linha, descreve-se a predição para os próximos $N$ estados da seguinte maneira:

$$\begin{aligned}
&\mathbf{x}(k+1) = A\mathbf{x}(k) + B\mathbf{u}(k) \\
&\mathbf{x}(k+2) = A\mathbf{x}(k+1) + B\mathbf{u}(k+1) = A[A\mathbf{x}(k) + B\mathbf{u}(k)] + B\mathbf{u}(k+1) \\
&\mathbf{x}(k+3) = A\mathbf{x}(k+2) + B\mathbf{u}(k+2) = A[A[A\mathbf{x}(k) + B\mathbf{u}(k)] + B\mathbf{u}(k+1)] + B\mathbf{u}(k+2) \\
&... \\
&\mathbf{x}(k+N) = A^N\mathbf{x}(k) + A^{N-1}B\mathbf{u}(k) + ... + AB\mathbf{u}(k+N-2) + B\mathbf{u}(k+N-1)
\end{aligned} \tag{2.15}$$

Da mesma maneira, pode-se efetuar a predição para as próximas $N$ saídas do sistema onde:

$$\begin{aligned}
&\mathbf{y}(k+N) = C\mathbf{x}(k+N) + d(k+N) \\
&\mathbf{y}(k+N) = CA^N\mathbf{x}(k) + C(A^{N-1}B\mathbf{u}(k) + ... + B\mathbf{u}(k+N-1)) + d(k)
\end{aligned} \tag{2.16}$$

Para complementar a notação, é comum ainda o uso de um duplo índice para cada elemento da predição, onde o primeiro item do índice mostra quantos passos à frente está a predição, enquanto o segundo mostra em que instante a predição foi realizada. Neste caso, $\mathbf{x}(k+2|k)$ representa a predição do estado **x** dois passos à frente a partir do instante $k$. Da mesma maneira, $\mathbf{y}(k+5|k+1)$ representa a saída para o instante $(k+5)$ realizada no instante $(k+1)$.

Neste caso, as equações 2.15 e 2.16 se tornam:

$$\begin{aligned}
&\mathbf{x}(k+N|k) = A^N\mathbf{x}(k) + A^{N-1}B\mathbf{u}(k|k) + ... + AB\mathbf{u}(k+N-2|k) + B\mathbf{u}(k+N-1|k) \\
&\mathbf{y}(k+N|k) = CA^N\mathbf{x}(k) + C(A^{N-1}B\mathbf{u}(k|k) + ... + B\mathbf{u}(k+N-1|k)) + d(k)
\end{aligned} \tag{2.17}$$

onde pode ser notado que $\mathbf{x}(k)$ tem somente um índice já que não é uma predição e sim o valor do estado atual medido ou estimado do sistema.

Para compactar a notação, é possível ainda reescrever a lista de equações de predição de

estados e saídas em forma matricial. Sendo assim, temos:

$$\begin{bmatrix} \mathbf{x}(k+1|k) \\ \mathbf{x}(k+2|k) \\ \vdots \\ \mathbf{x}(k+N|k) \end{bmatrix} = \begin{bmatrix} A\mathbf{x}(k) + B\mathbf{u}(k|k) \\ A^2\mathbf{x}(k) + AB\mathbf{u}(k|k) + B\mathbf{u}(k+1|k) \\ \vdots \\ A^N\mathbf{x}(k) + A^{N-1}B\mathbf{u}(k|k) + ... + AB\mathbf{u}(k+N-2|k) + B\mathbf{u}(k+N-1|k) \end{bmatrix} \tag{2.18}$$

Reescrevendo a lista de estados futuros em notação vetorial dizemos que $\tilde{\mathbf{x}}(k+1)$ representa o vetor de predição de estados futuros de $(k+1)$ até $(k+n)$. Nota-se ainda que, como o estado predito é um vetor (assumindo que o sistema tenha mais de um estado), a notação compacta $\tilde{\mathbf{x}}$ representa um vetor de vetores. Separando os valores medidos do sistema ($\mathbf{x}(k)$) das variáveis a serem escolhidas ($\tilde{\mathbf{u}}(k+1)$) temos:

$$\tilde{\mathbf{x}}(k+i) = \underbrace{\begin{bmatrix} A \\ A^2 \\ \vdots \\ A^n \end{bmatrix}}_{P_x} \mathbf{x}(k) + \underbrace{\begin{bmatrix} B & 0 & \dots & 0 \\ AB & B & \dots & 0 \\ \vdots & \vdots & \ddots & \dots \\ A^{N-1}B & A^{N-2}B & \dots & B \end{bmatrix}}_{H_x} \begin{bmatrix} \mathbf{u}(k|k) \\ \mathbf{u}(k+1|k) \\ \vdots \\ \mathbf{u}(k+N-1|k) \end{bmatrix} \tag{2.19}$$

Finalmente, de maneira compacta, podemos escrever a Equação 2.19 como:

$$\tilde{\mathbf{x}}(k+i) = P_x\mathbf{x}(k) + H_x\tilde{\mathbf{u}}(k) \tag{2.20}$$

Fazendo o mesmo para as equações de predição de saídas, temos:

$$\tilde{\mathbf{y}}(k+i) = P\mathbf{x}(k) + Ld(k) + H\tilde{\mathbf{u}}(k) \tag{2.21}$$

onde,

$$P = \begin{bmatrix} CA \\ CA^2 \\ \vdots \\ CA^n \end{bmatrix}, \quad H = \begin{bmatrix} CB & 0 & \dots & 0 \\ CAB & CB & \dots & 0 \\ \vdots & \vdots & \ddots & \dots \\ CA^{N-1}B & CA^{N-2}B & \dots & CB \end{bmatrix} \quad \text{e} \quad Ld(k) = \begin{bmatrix} d(k) \\ d(k) \\ \vdots \\ d(k) \end{bmatrix} \tag{2.22}$$

### 2.2.2 Definição da Função Custo e Lei de Controle

A função custo do MPC linear segue o mesmo formato apresentado pela Equação (2.8) e pode ser reescrita de maneira extensa como:

$$J(\mathbf{x}(k), \mathbf{u}(k)) \triangleq \sum_{i=1}^{N} ||\tilde{\mathbf{y}}(k+i) - \mathbf{y}_{ref}(k+i)||_Q^2 + \sum_{i=0}^{N} ||\tilde{\mathbf{u}}(k+i) - \mathbf{u}_{ss}||_R^2 \tag{2.23}$$

onde $\mathbf{y}_{ref}$ representa as referências futuras, $\mathbf{u}_{ss}$ os valores desejados de $\mathbf{u}$ em estado estacionário e $\|v\|^2$ é a norma Euclidiana de um vetor $v$.

Para o MPC em sua forma linear apresentado aqui esta função custo, mesmo com as restrições consiste em um problema de otimização convexo podendo ser resolvido com técnicas de Programação Quadrática [50]. A seguir, apresentamos os principais métodos e trabalhos recentes na literatura abordando a solução do MPC linear.

### 2.2.3 Métodos de Cálculo para o MPC Linear

Conforme apresentado, o algoritmo MPC requer que, a cada período de amostragem, se resolva um problema de otimização. Para o caso linear cuja função custo é convexa, este problema se torna um problema de programação quadrática (QP). Porém, este processo envolve operações matriciais e, dependendo do número de entradas, estados, saídas do sistema e comprimento do horizonte de predição, este cálculo pode apresentar um custo computacional considerável.

Diversos métodos para solução de QP são explorados na literatura. A seguir, é apresentada uma compilação dos melhores resultados desde 2010 focados em aplicações para sistemas embarcados, ou seja, que apresentam algoritmos com esforço computacional reduzido e/ou implementações em hardware que viabilizem a aplicação do MPC em sistemas com restrições em consumo de energia.

#### 2.2.3.1 Métodos Explícitos

Um primeiro método para solução do MPC Linear é o método explícito que aborda o MPC como um QP parametrizado. Neste caso, soluções para o problema são calculadas previamente (*offline*), classificadas em regiões críticas do problema e organizadas de maneira parametrizada para que possam ser posteriormente acessadas em uma *lookup table*. Durante a execução do processo, basta que se identifique a região crítica a qual o estado atual pertence para resgatar a solução e calcular o valor ótimo (*on-line*). Este método é aplicado aos casos em que o poder computacional seja muito baixo ou uma solução muito rápida seja necessária (período de amostragem na ordem de microsegundos ou nanosegundos). Porém, este método só é aplicável a problemas de dimensões pequenas pois o número de soluções cresce exponencialmente de acordo com o número de estados e restrições. Um *survey* sobre aplicações de métodos explícitos ao MPC é apresentado por Alessio e Bemporad em [51].

Segundo [52], existem ainda métodos aproximados para o cálculo *offline* de soluções explícitas que diminuem o esforço computacional prévio e aumentam a dimensão dos problemas que podem ser abordados por esta técnica. Mesmo assim, o uso destes métodos continua restrito à 8 estados e a um horizonte de predição máximo de $N = 5$.

#### 2.2.3.2 Métodos Iterativos

Dentre os métodos iterativos, destacam-se os *Active-Set Methods*, *Interior-Point Methods*, *Fast Gradient Method*, *Parallel Quadratic Programming* e *ADMM - Alternating Method of Multipliers*.

Os *Active-Set Methods* podem ser classificados em Primal e Dual [53]. A essência do algoritmo consiste em definir um ponto inicial de solução viável e construir um conjunto ativo (working set) baseado nas restrições do problema. A partir disso, caso exista uma direção e um tamanho de passo sobre as restrições que minimize a solução, caminha-se nesta direção. Caso contrário a solução pode ser inviável ou encontrou-se o ponto ótimo. Uma solução que se destaca na literatura é a de Ferreau [54, 52] onde diversas melhorias incluindo uma forma otimizada de tratar o conjunto ativo e maneiras de inicializar as iterações a partir de soluções anteriores (*warm-start*) são exploradas e apresentadas em uma ferramenta de código aberto, o qpOASES[1], onde vários problemas MPC podem ser resolvidos em poucos milisegundos utilizando-se um processador comum. Em [56], Bemporad apresenta uma nova abordagem de *Active-Set Method* baseado em *Nonnegative Least Squares* (NNLS) para otimização de problemas MPC em sistemas embarcados. As principais vantagens do método são a simplicidade do algoritmo, principalmente para certificação na indústria. Seu desempenho pode chegar a uma taxa de amostragem de 800 $\mu$s para um horizonte de 30 ou 200 $\mu$s para um horizonte de 5 em problemas menos complexos, rodando em um PC a 3 GHz.

Os *Interior-Point Methods* podem ser do tipo *Primal Barrier methods* e *Primal-dual methods*. A ideia por trás do algoritmo é a substituição da definição de restrições por desigualdades pelo acréscimo de um termo logarítmico na função objetivo. Neste caso, cria-se uma barreira que impede pontos de se aproximarem da região de restrições. Em seguida, o problema é resolvido pelo método de Newton. Para problemas grandes, este método resulta em menos iterações, se comparado aos *Active-Set Methods*. Porém, o custo computacional de cada iteração é muito maior. Além disso, não há maneira fácil de utilização de *warm-start*. Na literatura, são encontrados vários exemplos de soluções rápidas com este método. Wang e Boyd [57] apresentam a otimização do algoritmo capaz de executar um problema de dimensões pequenas com um horizonte igual a 30 com um período de amostragem de 5 ms em um PC a 3 GHz. Em seguida, juntamente com Mattingley [58] os mesmos autores criaram uma ferramenta para geração de código otimizado e customizado a partir de uma descrição de alto nível do um dado problema MPC. Neste caso, se atinge uma taxa de amostragem de 500 $\mu$s em um PC a 3.5 GHz. Em [17] Zeilinger et al. criaram uma maneira de utilizar o *warm-start* neste método e de assegurar a estabilidade do sistema

[1] Ferramenta de solução QP [55] disponível em http://www.qpOASES.org

em tempo real com um período de amostragem de 300 $\mu$s em um PC a 2.6 GHz. Explorando a arquitetura de GPUs, em [59, 60] Gade-Nielsen et al. desenvolveram uma Toolbox para Matlab implementado o *Interior-Point Method*. Finalmente, em [61], Hartley et al. utilizaram a abordagem de *High-Level Synthesis* (HLS) para implementar um controlador MPC em FPGA a partir do *Interior-Point Method*. Neste caso, a solução atinge uma taxa de amostragem de (4 ms) mas a baixo consumo de energia.

O *Fast Gradient Method* acelera os métodos de gradiente clássicos utilizando informações da segunda derivada para definir a direção e o tamanho do passo a ser utilizado a cada iteração, alcançando uma convergência mais rápida. Porém, dependem da capacidade de projetar o gradiente a um conjunto viável de soluções. Segundo Ferreau [52], este custo computacional é baixo para problemas com restrições por todos os lados (*box-constrained*). Porém, se torna mais custoso para problemas gerais. Em [62, 63] Jerez et al. apresentam uma implementação em FPGA baseada no *Fast Gradient Method* onde problemas MPC podem ser resolvidos na a frequências de amostragem de 1 MHz. Em seguida, em [64], Suardi, Kerrigan e Constantinides propõem uma *toolbox* para criação de soluções em FPGA que atingem taxas de amostragem de até 500 $\mu$s.

Finalmente, ainda na área de sistemas embarcados, observa-se no trabalho de Yu e Goldsmith [65], a exploração da arquitetura de uma GPU embarcada, Tegra, onde os métodos de *Parallel Quadratic Programming* e *ADMM - Alternating Method of Multipliers* são utilizados alcançando taxas de amostragem em torno de 10 ms.

## 2.3 CONTROLE PREDITIVO NÃO-LINEAR

No caso do MPC linear, mesmo com aplicações de restrições nas leis de controle, observa-se na literatura diversos métodos e implementações confiáveis e rápidas para resolver o problema de otimização em um tempo determinado. Porém, quando se lida com sistemas dinâmicos não-lineares e suas funções custo e restrições já não se comportam como funções convexas simples, a complexidade computacional do algoritmo se torna muito maior. De acordo com Alamir [13] as matrizes que eram calculadas *off-line* no caso linear precisam ser re-calculadas a cada período de amostragem, incluindo o mapa de predições. Além disso, o problema de otimização não-linear não é mais convexo e pode apresentar vários mínimos locais tornando mais difícil a localização do mínimo global e, não encontrar uma boa solução pode tornar o sistema instável. Neste caso, entramos na área da programação não-linear onde é necessário resolver um problema de otimização não quadrático com restrições.

### 2.3.1 Cálculo da Predição Não-Linear

Por envolver equações diferenciais não-lineares, métodos numéricos se tornam necessários para o cálculo de uma solução. Dois métodos bastante difundidos são o método de Euler e os métodos de Runge-Kutta que discretizam a solução contínua em função do período de amostragem.

Em ambos os casos, partindo de um valor inicial conhecido e do período de amostragem $\Delta t$, é possível calcular, de maneira aproximada a estimativa do próximo estado.

No caso do método de Euler, expresso na Equação (2.24), é possível estimar o próximo estado em um único passo. Por ser um método de primeira ordem, seu erro é proporcional ao quadrado do período de amostragem para cada passo. Portanto, para obter erros menores, é necessário trabalhar com períodos de amostragem curtos.

$$x(k+1) = x(k) + \Delta t f(u(k), x(k)) \tag{2.24}$$

Para obter estimativas mais precisas, métodos de ordens mais elevadas podem ser utilizados. Um exemplo clássico é o método de Runge-Kutta de quarta ordem apresentado na Equação (2.25).

$$\begin{aligned} z_1 &= f(u(k), x(k)) \\ z_2 &= f(x(k) + 0,5\Delta t.z_1, u(k)) \\ z_3 &= f(x(k) + 0,5\Delta t.z_2, u(k)) \\ z_4 &= f(x(k) + \Delta t.z_3, u(k)) \\ x(k+1) &= x(k) + \Delta t.(z_1 + 2z_2 + 2z_3 + z_4)/6 \end{aligned} \tag{2.25}$$

Ambos os algoritmos podem ser paralelizados para aumentar seu desempenho. Na literatura, encontramos no trabalho de Liu et al. [66] métodos de paralelização do algoritmo Runge-Kutta de quarta ordem. Em [67], Chen et al. apresentaram uma abordagem de implementação em FPGA do método Runge-Kutta. Em [68], Rana e Kumari apresentaram problemas com equações diferenciais resolvidos em FPGA tanto com o método de Euler quanto com o método de Runge-Kutta.

### 2.3.2 Métodos de Cálculo para o NMPC

Assim como é o caso das soluções para MPC Linear, existem várias abordagens para a solução do NMPC encontradas na literatura. Neste trabalho, para este trabalho interessam as abordagens que permitem a aplicação do NMPC em sistemas embarcados e, particularmente, aquelas que tornem viável seu cálculo em tempo-real para sistemas de dinâmicas rápidas. Dentre as abordagens que se destacam na literatura estão: o uso de linearização, seja *offline* ou de maneira dinâmica, que reduz o problema de otimização a um problema QP e obtém uma solução aproximada; soluções que consideram o problema não-linear, tratando o problema de otimização não-linear de maneira aproximada através de parametrização; soluções onde redes neurais artificiais são treinadas *offline* e; soluções que utilizam algoritmos evolutivos ou bio-inspirados para efetuar a busca pelo ótimo global.

#### 2.3.2.1 Métodos incluindo linearização

Um exemplo do uso da abordagem de linearização é apresentado por Ferreau [52], onde uma revisão de métodos incluindo os métodos indiretos baseados no princípio do mínimo de Pontryagin, equações de Hamilton-Jacobi-Bellman e os métodos diretos foram discutidos. Neste caso, os primeiros dois métodos tratam as entradas $u$ como uma função e formulam um problema ótimo de dimensões infinitas que é posteriormente discretizado para encontrar a solução numérica. Em contraste, os métodos diretos primeiro discretizam o problema para, em seguida, o resolver com Programação Não-Linear (NLP), sendo os mais empregados na prática.

Nos métodos diretos o problema de otimização de dimensões infinitas é transformado em um problema finito (NLP) e a trajetória de controle $u$ é parametrizada por um número finito de parâmetros $q$ para gerar uma aproximação de $u$. Esses métodos podem ser ainda classificados nas abordagens sequencial e simultânea. Em seguida, métodos de programação não-linear são usados para a solução de problemas em tempo-real. Variações incluem a otimização de Newton (*Newton-Type Optimization*), a *Sequential Quadratic Programming* - SQP e *Interior-Point Methods*.

Em sua solução, o ACADO Toolkit, Ferreau aplica métodos para linearizar subproblemas dentro do problema não-linear. Além disso, métodos eficientes para cálculo de derivadas que possam proporcionar desempenho em tempo-real e prover soluções aproximadas mas suficientes para o bom desempenho do sistema são estudados.

Dois exemplos recentes utilizam a ferramenta ACADO Toolkit para resolver NMPC. Em um primeiro, Vukov et al. [69] implementa um controle de partida de uma turbina eólica. O problema possui 27 estados e quatro saídas e pode ser resolvido em um período de amostragem de 5 ms em um PC com processador Intel Core i7 de 2.3 GHz. Em um segundo trabalho, Quirynen et al. [70] apresentam um tutorial que utiliza as propriedades de geração de código otimizado do ACADO Toolkit para gerar soluções NMPC para problemas simples com taxa de amostragem de até 80 $\mu$s no mesmo processador Intel Core i7 de 2.3 GHz.

Em uma outra abordagem, Kapernick et al. [71] utilizaram ferramentas de *High Level Synthesis* (HLS) para implementar um controlador NMPC em FPGA. O algoritmo se baseia em métodos indiretos que calculam $u$ baseado no princípio do máximo de Pontryagin com a função hamiltoniana. A abordagem é testada para um sistema de um guindaste com pêndulo com 6 estados, uma entrada, uma saída e um horizonte de predição de 30 amostras. Os resultados mostram uma solução em FPGA capaz de calcular o valor da entrada em 26 ms.

#### 2.3.2.2 Métodos com parametrização

Segundo Alamir et al. [72], algumas das abordagens que tratam do problema não-linear incluem *multiple shooting real-time iteration* [73], o método Continuation/GMRES (*Generalize Minimum Residual*) [74] e o *Control Parametrization Approach* (CPA) [10].

A ideia por trás da estratégia de parametrização é reduzir o número de graus de liberdade do

sinal de controle diminuindo o número de variáveis a serem determinadas durante o processo de busca. Neste caso, o sinal de controle é representado por uma sequência de funções onde um número de graus de liberdade $n_p < N$ é definido. A partir disto, ao invés de definir os $N$ valores do vetor $u$, são definidos $n_p$ parâmetros a partir dos quais os $N$ valores são calculados a partir de um processo de interpolação.

Um exemplo de uso do CPA é apresentado por Murilo, Alamir e Alberer em [20] onde uma solução para sistemas embarcados de um NMPC para o controle de emissões a partir de entrada de ar de um motor à diesel é apresentada. A proposta utiliza uma função composta por uma soma de exponenciais que é capaz de aproximar o comportamento de $u$ com um número bastante reduzido de parâmetros. A solução provê um meio de implementar o NMPC de modo que o resultado da otimização é aproximado, mas satisfatório para realizar o controle do sistema seguindo as especificações. O resultado final mostra um sistema capaz de controlar uma dinâmica relativamente complexa com um horizonte de predição igual a 100 e operar em um processador embarcado de 480 MHz com um período de amostragem de 10 ms.

#### 2.3.2.3 Métodos baseados em aprendizado de máquina

Uma abordagem promissora para implementação do NMPC com frequências de amostragem altas (acima de 100 Hz ou até 1 kHz) é o uso de métodos de aprendizado de máquina, seja através de Redes Neurais Artificiais (RNAs) ou Support Vector Machines (SVMs). O desafio, nestes casos, está em definir quais entradas e quais os métodos de treinamento são necessários para replicar o comportamento do controlador e do sistema não-linear. Encontramos na literatura algumas linhas de trabalho com RNAs.

Os primeiros trabalhos relevantes na área foram realizados por Ortega e Camacho [75, 27]. Neste caso, um controlador NMPC é utilizado para o controle de trajetória de um robô móvel. A abordagem é particularmente interessante pois utiliza uma rede neural *multilayer perceptron* para implementar tanto o controle de posição quanto a rejeição de obstáculos. A dinâmica não-linear do robô consiste em 5 estados, 2 entradas e 2 saídas. A rede é construída com 3 camadas sendo que a camada de entrada possui 12 neurônios, cinco deles relativos aos estados e os demais relativos à informação dos sensores de ultrassom para evitar obstáculos. A camada de saída possui dois neurônios que fornecem a entrada do sistema (velocidades linear e angular). Neste caso, a rede é toda treinada *offline* por um controlador NMPC. Os resultados mostram um controle da RNA bastante satisfatório e aproximado em relação ao comportamento simulado. Por serem trabalhos antigos (1994 e 1996), o período de amostragem utilizado foi de 600 ms.

Em [76], Akesson e Toivonen apresentam uma RNA do tipo *feedforward* com uma camada escondida e função de ativação hiperbólica. Dois sistemas de controle, um de pH e outro de um misturador de líquidos são avaliados, sendo um SISO e um MIMO com duas entradas e duas saídas. No primeiro caso uma rede com 11 neurônios na camada escondida é treinada e resulta em um comportamento satisfatório, inclusive rejeitando perturbações. Porém, no segundo caso, o treinamento da rede se mostrou complexo pois a função custo podia fornecer uma tendência

para uma saída ou para outra. A solução encontrada foi a utilização de duas redes separadas, com as mesmas entradas mas com saídas distintas, uma para cada variável de controle. Por serem processos lentos, os períodos de amostragem são dados em segundos.

Ambas as abordagens apresentadas anteriormente utilizam RNAs para simular o comportamento completo do NMPC. Uma outra abordagem apresentada a seguir utiliza RNAs somente na etapa de predição do sistema não-linear.

Em [77], Kittisupakorn et al. apresentam um método para treinar uma RNA para realizar a predição de um passo e, de maneira recursiva, realizar a predição para todo o horizonte. Estendendo esta ideia, Ławryńczuk [49, 78, 79] apresenta uma outra abordagem que utiliza modelos neurais estruturados para evitar o acúmulo de erros de predição feita pelos modelos de um passo, principalmente para horizontes de predição longos.

Apesar da técnica, Ławryńczuk utiliza a abordagem de linearizar o problema dinamicamente, para resolver o problema de otimização convexo. Além disso, suas aplicações são voltadas para sistemas químicos com tempo de resposta lento.

#### 2.3.2.4 Métodos com algoritmos heurísticos de busca global

Uma abordagem mais recente está no uso de algoritmos bio-inspirados para a solução do NMPC de modo direto, sem linearização, a partir da aplicação do algoritmo *Particle Swarm Optimization* (PSO) [80]. O PSO é um algoritmo heurístico de busca global que utiliza o método de inteligência coletiva para realizar a busca do ótimo global com bom desempenho em problemas multimodais (não convexos), como é o caso do NMPC.

Um primeiro trabalho utilizando esta abordagem pode ser visto em [81] onde Wang e Xiao utilizam uma versão do PSO com a técnica de *simulated annealing* (SAPSO) para a solução do problema de otimização. Além disso, uma RNA do tipo RBF é utilizada para calcular a predição do sistema não-linear. No caso, a abordagem é aplicada a um sistema de controle de geração termoelétrico (temperatura e pressão) com duas entradas e duas saídas. Porém não há restrições de custo computacional pois a aplicação é lenta, taxa de amostragem de 5s.

Já em [82], Sivananaithaperumal et al. utilizam duas versões do PSO, a *Random PSO* (RPSO) e a Knowledge Based PSO (KPSO) são utilizadas e comparadas. No caso da KPSO, as entradas $u$ do sistema são inicializadas a partir das restrições e informações prévias do problema são realimentadas, havendo um ganho de desempenho na convergência do algoritmo. Além disso, uma comparação com algoritmos genéticos é feita, demonstrando o melhor desempenho do PSO. Por restrições de desempenho, um conjunto de problemas pequenos com horizonte de predição variando entre 2 e 8 e horizonte de controle variando entre 1 e 3 são utilizados. No menor problema, o tempo de cálculo é de 2,4 segundos.

Em [32], Mercieca e Fabri usam o PSO para realizar a simulação de um pêndulo invertido com modelo não-linear com quatro estados e duas entradas. O horizonte de predição usado é de

20 e o período de amostragem de 100 ms. Restrições são aplicadas tanto na entrada quanto na saída e o número máximo de iterações do PSO é de 30.

Em um trabalho mais recente, Xu et al. [83] apresentam uma solução que implementou o PSO em FPGA utilizando a abordagem de *High Level Synthesis* (HLS). A arquitetura é descrita em ponto fixo. Os testes são realizados com *hardware-in-the-loop* para um controlador de um motor em ponto morto. Aqui, mais uma vez, um problema relativamente pequeno tendo dois estados, uma entrada e uma saída. O horizonte de predição é definido em 5 enquanto o horizonte de controle é de 2. Além disso, restrições são impostas na entrada e na saída e uma perturbação externa é inserida para verificar a estabilidade do sistema. Os resultados mostram um controlador capaz de realizar o controle com um tempo de amostragem de 10 ms.

## 2.4 APRENDIZADO DE MÁQUINA

Conforme apresentado na Seção 2.3, há diversos métodos para abordar o problema de otimização não-linear do NMPC. Um deles se baseia no uso de técnicas de aprendizado de máquina para aproximar o comportamento do sistema ou até do controlador de modo geral [84]. Dentre os métodos destacamos o uso de Redes Neurais Artificiais e Máquinas de Vetor de Suporte, descritos a seguir.

### 2.4.1 Redes Neurais Artificiais

A possibilidade de uso de RNAs para representar o comportamento de sistemas não linear é corroborada por pesquisas que demonstram as redes neurais do tipo *feedforward* como aproximadores universais [85, 86]. Em [49], Ławryńczuk apresenta um *survey* sobre o uso de RNAs aplicadas ao MPC e destaca que dentre as arquiteturas mais utilizadas estão a *Multi Layer Perceptron* (MLP) e a *Radial Basis Function* (RBF).

As redes neurais com funções de base radial (*Radial basis function neural networks* - RBFNNs) são um tipo de RNA formadas por um número fixo de camadas. Cada uma destas camadas tem um propósito específico. A primeira é a camada de entrada responsável por conectar a rede com o meio externo. Sua conexão com a camada seguinte é direta não havendo qualquer tipo de fator de multiplicação. Já a segunda camada, a camada escondida, é responsável por realizar a função de ativação, neste caso com uso de funções de base radial (RBF). Estas funções de ativação possuem dois parâmetros fundamentais sendo eles o centro e a dispersão da função. Na saída desta camada, cada sinal é multiplicado por uma constante, um coeficiente de peso. O resultado final da rede é calculado na camada de saída sendo uma combinação linear do resultado de cada neurônio (da camada escondida). Uma representação gráfica da RBFNN pode ser vista da Figura 2.3.

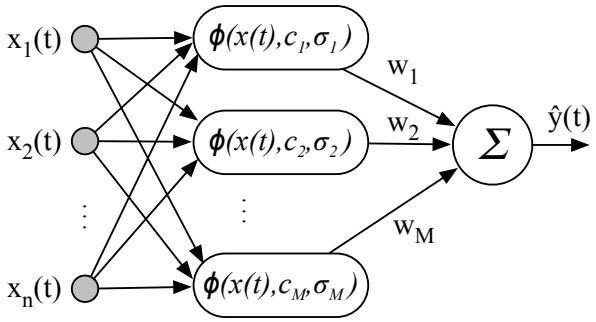



Figura 2.3: Diagrama da Rede Neural RBF.

Em termos matemáticos, podemos definir a RBFNN como:

$$y(t) = F[x(t)] = \sum_{m=1}^{M} w_m \phi(x(t), c_m, \sigma_m), \tag{2.26}$$

onde $M \in \mathbb{N}^+$ é o número de neurônios da camada escondida, $y_t$ e $x(t) \in \mathbb{R}^{n_r}$ são, respectivamente, a saída da rede, e o vetor de entradas em um dado instante $t$; $c_m \in \mathbb{R}^{n_r}$ e $\sigma_m \in \mathbb{R}^+$ são o centro e a dispersão da função do $m$-ésimo neurônio da rede (camada escondida), respectivamente. O peso de saída de cada neurônio da rede é dado por $w_m \in \mathbb{R}$. A função $\phi(\cdot)$ deve estar restrita a uma função da classe de funções de base radial [87].

Dentre as diversas funções da classe RBF podemos citar a Gaussiana, multiquadrática, inversa multiquadrática, linear, cúbica e *thin-plate spline*. Dentre elas, a função utilizada com mais frequência é a Gaussiana, que pode ser definida como:

$$\phi(x(t), c_m, \sigma_m) = \exp\left(-\frac{\|x(t) - c_m\|^2}{2\sigma_m^2}\right). \tag{2.27}$$

Uma outra maneira de descrever a mesma função, já visando a implementação de seu algoritmo em hardware é:

$$\phi(x(t), c_m, \sigma_m) = \exp\left(-\frac{1}{2\sigma_m^2}\sum_{i=1}^{n_r}(x(t)^{(i)} - c_m^{(i)})^2\right). \tag{2.28}$$

A estimação dos parâmetros da rede neural RBF pode ser descrita como um problema de otimização. Um dos métodos mais rápidos e que provê resultados bastante precisos consiste na adotação de um treinamento em 2 passos. No primeiro, os centros $c_m$ são definidos, de acordo com algumas técnicas como *clustering* ou escolhas aleatórias a partir de amostras do vetor de entradas. Uma delas é o método de *clustering* $k$-means descrito em [88]. Esta técnica pode ser utilizada para definir os centros da RBF. Em segundo lugar, define-se empiricamente o valor de $\sigma_m$ e se calcula os valores dos coeficientes de peso $w_m$ através do método de mínimos quadrados [87].

### 2.4.2 *Support Vector Machines* - SVMs

As Máquinas de Vetor de Suporte ou *Support Vector Machines* (SVMs) são um tipo de técnica de aprendizado de máquina que provê alta capacidade de generalização e robustez tanto em problemas de classificação quanto de regressão. Foram propostas por Vapnik [89] e são baseadas na busca de um hiperplano ótimo com margens largas que maximizem a distância entre as classes do conjunto de treinamento. Esta técnica garante que o erro sobre o conjunto de treinamento e sua capacidade de generalização podem ser otimizadas de maneira simultânea [90, 91].

Uma de suas vantagens está em seu processo de treinamento que é baseado em um problema de otimização convexo que provê uma solução única em contraste com o processo de treinamento de RNAs cuja solução possui vários mínimos locais, conforme descrito por Theodoridis e Koutroumbas [92]. Além disso, SVMs são capazes de condensar as informações do conjunto de treinamento podendo representar uma solução com um número bastante reduzido de dados de referência, chamados de Vetores de Suporte (SVs - *Support Vectors*).

Porém, para aplicar a teoria da SVM a aplicações com saídas contínuas, como é o caso de um controlador de um sistema dinâmico como o NMPC, uma extensão da SVM denominada de SVR (*Support Vector Regression*) foi desenvolvida por Drucker et al. [93] para induzir a função aproximada a partir de um conjunto de treinamento $X = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_3) \ldots (\mathbf{x}_n, y_n)\} \subset \mathbb{R}^d \times \mathbb{R}$, onde $\mathbb{R}^d$ denota o espaço dos padrões de entrada e $\mathbb{R}$ representa o conjunto de rótulos. A Equação 2.29, apresenta a função geral aproximada pela SVR.

$$f(x) = \sum_{i=1}^{\#SV} (\alpha_i - \alpha_i^*) K(\mathbf{x}_i, \mathbf{x}) + b, \tag{2.29}$$

onde $\alpha_i$ e $\alpha_i^*$ são as soluções ótimas do problema quadrático apresentado na Equação (2.30), conforme definido em [93], $K(\cdot, \cdot)$ é a função *kernel*, $b$ é a constante de *bias* calculada a partir de $\alpha_i$ e $\alpha_i^*$, e $\#SV$ é a cardinalidade do conjunto de vetores de suporte.

$$\max L(\alpha, \alpha^*) = -\frac{1}{2} \sum_{i,j=1}^{N} K(x_i, x_j)(\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*) - \epsilon \sum_{i=1}^{N} (\alpha_i - \alpha_i^*) + \sum_{i=1}^{N} y_i (\alpha_i - \alpha_i^*)$$

$$\text{s.a.} \sum_{i=1}^{N} (\alpha_i - \alpha_i^*) = 0$$

$$\alpha_i, \alpha_i^* \in [0, C] \qquad i = 1, ..., N. \tag{2.30}$$

O problema quadrático apresentado na Equação (2.30) possui três tipos de parâmetros, denominados de hiperparâmetros, que precisam ser ajustados sendo eles: $C$ o parâmetro de regularização, $\epsilon$ a largura do tubo $\epsilon$-*insensitive* e os parâmetros do *kernel*, que dependem de cada tipo de *kernel* como será visto a seguir. A escolha dos hiperparâmetros está diretamente relacionada tanto com a complexidade da SVR quanto com sua capacidade de generalização.

A Figura 2.4 apresenta o processo de ajuste da SVR baseado no conjunto de treinamento (pontos pretos). O ajuste da função é guiado pelo tubo $\epsilon$-*insensitive* definido a partir da constante $\epsilon$. Dados fora deste tubo são penalizados pelo parâmetro de regularização e os pontos que permanecem dentro do tubo são considerados os vetores de suporte.

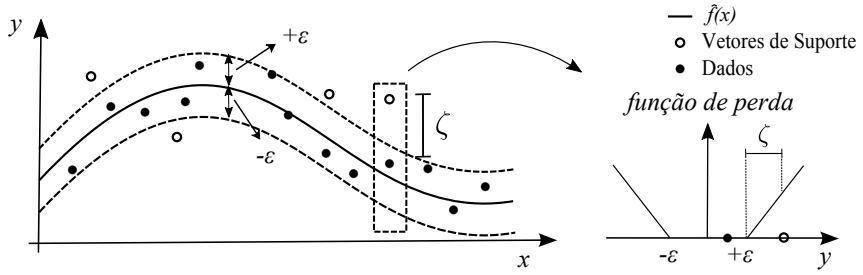



Figura 2.4: Função aproximada e o tubo $\epsilon$-*insensitive* [94].

Funções geralmente utilizadas como *kernel* são as funções de base radial (RBFs, geralmente a Gaussiana), Polinomial e Sigmoidal (90). O *kernel* RBF Gaussiano, descrito pela Equação 2.31), é de fácil treinamento já que possui somente o parâmetro $\gamma$ a ser ajustado.

$$K(\mathbf{x}_1, \mathbf{x}_2) = \exp\left(-\frac{\|\mathbf{x}_1 - \mathbf{x}_2\|^2}{\gamma}\right). \tag{2.31}$$

Já o *kernel* polinomial, descrito pela Equação 2.32 possui implementação mais simples em hardware.

$$K(\mathbf{x}_1, \mathbf{x}_2) = (\gamma \cdot \mathbf{x}_1 \cdot \mathbf{x}_2 + c)^d, \tag{2.32}$$

onde $d \in \mathbb{N}^+$ é o grau do polinômio e $c \geq 0$ é um parâmetro livre que opera para balancear a relação entre os termos de maior e menor ordem do polinômio.

## 2.5 ALGORITMOS BIOINSPIRADOS

Além do uso de métodos com aprendizado de máquina, uma segunda abordagem deste trabalho se baseia no uso de algoritmos heurísticos bioinspirados de busca global que possam atender aos requisitos do problema de otimização não-convexo do NMPC e ainda assim fornecer uma solução capaz de controlar sistemas com dinâmica rápida em tempo-real.

Os algoritmos bioinspirados são objetos de estudo da área de computação bioinspirada, um ramo da computação natural. Segundo Pereira e Tavares [80], nesta área a natureza é utilizada como fonte de inspiração para o desenvolvimento destes algoritmos com o objetivo de resolver problemas de alta complexidade cuja solução tradicional é dificilmente alcançada. Os principais grupos destes algoritmos ainda se dividem em: Algoritmos Evolucionários, Algoritmos baseados em Inteligência Coletiva, e Redes Neurais, dentre outros [95]. Estes dois primeiros grupos são baseados em populações de partículas ou potenciais soluções e são amplamente aplicados a

problemas de otimização.

Os algoritmos evolucionários se inspiram em conceitos de evolução biológica como reprodução, mutação, recombinação e seleção e os aplicam aos membros de sua população na busca por soluções. Dentre os algoritmos mais conhecidos nesta família estão os Algoritmo Genético (GA), a Programação Genética (GP) e o algoritmo de Evolução Diferencial (DE). Já na família de algoritmos baseados em inteligência coletiva ou inteligência de enxame, a busca por soluções se baseia no comportamento de grupos de animais e/ou insetos que individualmente possuem pouca inteligência, porém, ao compartilhar informações uns com os outros, apresentam uma inteligência coletiva capaz de solucionar problemas complexos [95]. Dentre os algoritmos desta família podemos citar o *Ant Colony Optimization* (ACO) inspirado no comportamento de formigas, o *Particle Swarm Optimization* (PSO) inspirado em bandos de pássaros e o algoritmo *Artifitial Bee Colony* (ABC) inspirado no comportamento de abelhas. Sendo uma área de pesquisa bastante ativa, observa-se de 2016 em diante, a proposição de dezenas de outros algoritmos, dentre eles o *Grey Wolf Optimizer* (GWO), *Moth-Flame Optimization* (MFO), *Dragon-Fly Algorithm* (DA), *Whale Optimization Algorithm* (WOA) e *Salp Swarm Algorithm* (SSA) [96].

Tanto algoritmos evolucionários quanto os baseados em inteligência coletiva tem vantagens sobre algoritmos de busca baseados em gradientes no que diz respeito ao seus métodos de busca globais e sua paralelização, já que são baseados em um grupo de indivíduos com processamento independente em boa parte. Este aspecto os torna mais adequados para aplicações em tempo-real. Porém, uma de suas limitações quanto aplicado a problemas não-convexos com restrições é o fato de nem sempre fornecerem uma solução viável, limitação esta que deve ser tratada para garantir a estabilidade no controle dos sistemas.

Dentre estes algoritmos, conforme apresentado na Seção 2.3.2.4, o PSO apresenta alguns casos de sucesso ao ser aplicado ao NMPC. Porém, estas aplicações apresentam limitações em termos de oscilações no sinal de controle e limites quanto ao comprimento do horizonte de predição. Por ser um algoritmo consolidado na literatura, foi adotado como algoritmo a ser investigado neste trabalho.

### 2.5.1 Otimização por Enxame de Partículas (PSO)

O algoritmo de Otimização por Enxame de Partículas ou *Particle Swarm Optimization* (PSO) foi desenvolvido por Eberhart e Kennedy em 1995 [97] e aprimorado por diversos pesquisadores ao longo dos anos [98]. Sua inspiração vem de sistemas biológicos sociais como um bando de pássaros ou um cardume de peixes. Em ambos os casos, ao procurar por comida, o grupo se espalha e, cada indivíduo é autônomo para fazer sua busca com certo grau de liberdade e de maneira aleatória. Ao encontrar comida, o indivíduo transmite esta informação aos outros que, aos poucos também alcançam a fonte de comida. Portanto, o algoritmo simula este comportamento através de partículas independentes que calculam a solução para um problema e, em sua descoberta, vai influenciando as demais partículas até que o enxame possa convergir para um ponto.

Em termos mais técnicos, conforme descrito por Weise [98], o PSO realiza uma busca utilizando um grupo de $S$ partículas em um espaço N-dimensional $G$ para encontrar o melhor valor possível, seja ele o valor máximo ou mínimo da função. Cada partícula possui uma posição $p \epsilon \mathbb{G} \subseteq R^n$ e uma velocidade $v \epsilon R^n$. A posição $p$ representa uma solução candidata ao problema. A velocidade $v$ da partícula determina tanto em que direção ela irá se mover para continuar sua busca quanto se ela efetua uma exploração em uma área maior (alta velocidade) ou em área menor (baixa velocidade).

A inicialização do algoritmo consiste em determinar posições e velocidades aleatórias para cada partícula. A cada passo do algoritmo são atualizadas a velocidade e em seguida a posição das partículas. Cada partícula possui uma memória guardando sua melhor posição. A interação social entre as partículas se dá pelo fato de cada uma poder conhecer a posição das partículas vizinhas, ou seja, que estejam em um raio de atuação pré-definido à sua volta. Cada partícula pode se comunicar com as vizinhas e conhecer a melhor posição da vizinhança ou a melhor posição global entre todas as partículas. Se o algoritmo utiliza a melhor posição global para atualizar a posição e velocidade das partículas, tenderá a convergir mais rapidamente, porém com menos chance de encontrar a solução ótima global. Utilizar a melhor posição da vizinhança torna a convergência mais lenta porém com maior probabilidade de encontrar a solução ótima global.

As equações básicas para o cálculo da velocidade e posição das partículas são:

$$v_{ij}^{(t+1)} = v_{ij}^{(t)} + c_1 U_{1j}(lb_{ij}^{(t)} - p_{ij}^{(t)}) + c_2 U_{2j}(gb_j^{(t)} - p_{ij}^{(t)}), \tag{2.33}$$

$$p_{ij}^{(t+1)} = p_{ij}^{(t)} + v_{ij}^{(t+1)}, \tag{2.34}$$

onde,

- $i$ : índice da partícula;
- $j$ : índice da dimensão da partícula;
- $v$ : velocidade da uma partícula;
- As velocidades $v_{ij}$ estão limitadas na faixa $[-v_{max}, v_{max}]$ evitando que as partículas abandonem o espaço de busca;
- $p$ : posição de uma partícula no espaço N-dimensional limitado na faixa $[p_{min}, p_{max}]$;
- $lb_i$ : melhor posição individual de uma partícula (*local best*);
- $gb$ : melhor posição global (*global best*) entre todas as partículas (S partículas);
- $c_1$ : coeficiente cognitivo (grau de auto-confiança);
- $c_2$ : coeficiente social (grau de confiança no enxame);
- $U_1$ e $U_2$: números aleatórios uniformemente distribuídos na faixa [0,1].

Observa-se ainda que os coeficientes cognitivo e social tem alto grau de impacto no desempenho do algoritmo. Um valor grande para o coeficiente cognitivo ($c_1$) significa um alto valor de autoconfiança da partícula enquanto um grande no coeficiente social ($c_2$) faz com que as partículas confiem mais no enxame. É conhecido que, para funções objetivo unimodais, resultados melhores são obtidos com pequenos valores de $c_1$ e valores maiores de $c_2$. Para funções multimodais o melhor desempenho é alcançado balanceando $c_1$ e $c_2$.

Uma das modificações do algoritmo básico do PSO é chamado de Fator de Inércia, proposto por Shi e Eberhart [99]. Este fator é aplicado ao cálculo da velocidade da partícula de maneira a decrescer linearmente desde seu valor inicial mais alto até seu valor final. O efeito gerado nas partículas é que no início da execução do algoritmo, o alto valor de inércia favorece uma busca global também conhecido como fase de exploração do espaço de busca. Ao longo da execução, este valor decai e faz com que as partículas priorizem uma busca mais local, conhecida como fase de refinamento da solução.

O fator de inércia é calculado a partir da seguinte Equação:

$$w = w_{max} - (w_{max} - w_{min})\frac{g}{G}. \tag{2.35}$$

onde, $g$ significa a geração ou iteração atual do algoritmo e $G$ o número máximo de iterações, pré-definida. Os valores mais comuns para $w_{max}$ e $w_{min}$ são $0,9$ e $0,4$, respectivamente [99, 100].

O uso do fator de inércia pode ser visto na Equação 2.35 representado por $w$.

$$v_{ij}^{(t+1)} = wv_{ij}^{(t)} + c_1U_{1j}(lb_{ij}^{(t)} - p_{ij}^{(t)}) + c_2U_{2j}(gb_j^{(t)} - p_{ij}^{(t)}). \tag{2.36}$$

A seguir, apresenta-se o pseudocódigo do algoritmo PSO já incorporando o fator de inércia.

### 2.5.2 Topologias do PSO

Um fator que afeta como a busca é realizada no PSO está relacionada com a topologia de como as partículas compartilham informações. De modo geral, as topologias mais utilizadas são a LBEST, onde partículas são influenciadas pelas melhores posições de suas vizinhas mais próximas, e a GBEST, onde a vinhança considerada é global, ou seja, a melhor partícula de todo o enxame influencia todas as demais. Segundo Kenedy [101], na topologia GBEST ocorre uma convergência rápida das partículas já que uma só atrai todo o enxame e isso pode gerar resultados subótimos que ficam presos a mínimos locais. Por outro lado, na topologia LBEST, a convergência é mais lenta e tem mais chance de alcançar o valor ótimo global.

Um experimento realizado por Suganthan [102] mostra ainda uma versão do PSO onde o processo de busca é iniciado com a topologia LBEST, considerando uma vizinhança pequena e, ao longo das iterações vai aumentando o tamanho da vizinhança de influencia até englobar todas as partículas e utilizar a topologia GBEST. Em [103], Zhan et al. propõem uma versão adaptativa do PSO que, dentre outros elementos, utiliza uma variação da topologia LBEST que ajusta o

**Algorithm 1:** Algoritmo PSO com fator de inércia

Inicializa o enxame de partículas $p_{1..S}$ obedecendo às restrições de $p_{min}$, $p_{max}$;
Inicializa os valores de $lb$ (Melhor Local) de cada partícula e $gb$ (Melhor Global);
`/* Laço 1 (L1) - Principal */`
**for** $iter = 1 : MaxIter$ **do**
  `/* Laço 2 (L2) - Cálculo da Função Custo */`
  **for** $i = 1 : S$ **do**
    **if** $f(p_i) \leq f(lb_i)$ **then**
      $lb_i = p_i$;
  `/* Laço 3 (L3) - Calcula a Melhor Posição Global */`
  **for** $i = 1 : S$ **do**
    **if** $f(lb_i) \leq f(gb)$ **then**
      $gb = lb_i$;
  `/* Laço 4 (L4) - Atualiza Partículas */`
  **for** $i = 1 : S$ **do**
    **for** $j = 1 : N$ **do**
      $v_{ij}^{(t+1)} = wv_{ij}^{(t)} + c_1U_{1j}(lb_{ij}^{(t)} - p_{ij}^{(t)}) + c_2U_{2j}(gb_j^{(t)} - p_{ij}^{(t)})$;
      $p_{ij}^{(t+1)} = p_{ij}^{(t)} + v_{ij}^{(t+1)}$;
      Aplicar restrições de $p_{min}$ e $p_{max}$;
  Atualiza o valor da inércia $w$;

tamanho da vizinhança de modo oscilatório alterando entre as fases de exploração e refinamento.

Apesar de oferecer uma melhor solução final em potencial, a topologia LBEST tem como desvantagens a sua lenta convergência e aumento do custo computacional. Neste caso, antes de comparar o valor da função objetivo de cada partícula com as demais, é necessário definir quais partículas estão na vizinhança de cada outra partícula a partir do cálculo das distâncias relativas. O custo computacional cresce em problemas com dimensionalidade alta.

Como a velocidade de cálculo do resultado e o baixo custo computacional são importantes na aplicação em tempo-real do NMPC, uma alternativa é o uso da topologia GBEST. Neste caso, não é necessário calcular as distâncias relativas entre partículas mas sim efetuar a comparação do valor da função objetivo de cada partícula com todas as demais. Apesar do risco do algoritmo ficar preso em um mínimo local, ajustes nos coeficientes $c_1$ e $c_2$ podem evitar este problema conforme explorado por Zhan et al. [104].

## 2.6 HARDWARE RECONFIGURÁVEL

Com o foco na aplicação do MPC em sistemas embarcados, se faz necessário explorar as arquiteturas adequadas que atendam aos requisitos de execução em tempo-real e baixo consumo de energia. Neste sentido a arquitetura do sistema computacional tem grande influência sobre

seu desempenho. As arquiteturas mais comuns são os processadores de propósito geral (GPPs), os *Digital Signal Processors* (DSPs), Unidades de Processamento Gráfico (GPUs) e arquiteturas dedicadas, sendo elas implementadas em dispositivos reconfiguráveis como os FPGAs (*Field-Programmable Gate Arrays*) ou fabricadas em silício, como é o caso de um *Application Specific Integrated Circuit* (ASIC) que é um circuito dedicado para um propósito específico, um *System-on-a-chip* (SoC) que inclui um processador ou um *Multiprocessor System-on-chip* (MPSoCs) que inclui vários processadores (geralmente heterogêneos) [25].

Processadores de propósito geral, sendo eles embarcados ou não, são baseados no modelo de *von Neumann* [105], que define uma arquitetura onde instruções e dados são armazenados em memória e interligadas a uma unidade de processamento através de um barramento. Esta arquitetura amplamente usada na indústria apresenta limitações como, por exemplo, o fato de que a frequência da unidade de processamento (CPU) é geralmente mais alta que a frequência das memórias e, portanto, em muitos casos a CPU deve ficar parada aguardando retorno de instruções ou dados da memória. Este problema, denominado de *Memory Wall* [106], limita o processamento de alto desempenho.

Visando melhorar este desempenho, a indústria de processadores vem, por muitos anos, seguindo as previsões da lei de Moore e conseguindo aumentar a quantidade de transistores em chips juntamente com sua frequência. Porém, em meados dos anos 2000 o aumento da frequência atingiu um limite onde o consumo de potência e a necessidade de dissipação de calor inviabilizavam seu aumento continuado. Este fenômeno conhecido como Power Wall [107] fez com que fabricantes tomassem uma decisão de adotar arquiteturas com múltiplas unidades de processamento (*multi-core*) para tentar viabilizar o aumento de desempenho. Porém, esta estratégia nem sempre é viável pois algoritmos nem sempre são facilmente paralelizáveis e, mesmo os que são, encontram limitações na arquitetura.

Em uma linha similar, GPUs são arquiteturas com unidades de processamento altamente paralelas e vem ganhando mercado nos últimos anos. Porém, sua arquitetura não é adequada a algoritmos naturalmente sequenciais e, apresentam ainda um alto consumo de energia, limitando sua aplicação a sistemas embarcados.

Uma opção para evitar estas limitações é a criação de arquiteturas dedicadas que implementam algoritmos diretamente em hardware, onde é possível balancear unidades de processamento eficientes tanto para o processamento sequencial quanto para o processamento paralelo. Nesta abordagem, os FPGAs oferecem uma arquitetura flexível, de prototipação rápida e baixo consumo de energia quando comparado às CPUs embarcadas, podendo ser usada tanto como tecnologia fim para a implementação do circuito desejado, quanto como tecnologia meio para a validação do projeto a ser futuramente fabricado em um chip dedicado. Por estas características foi a tecnologia computacional adotada neste projeto.

### 2.6.1 FPGAs - *Field-Programmable Gate Arrays*

Os FPGAs são circuitos integrados compostos por um arranjo de células lógicas que podem ser programadas para executar funções lógicas, armazenar dados e efetuar roteamento de sinais para outras células lógicas. Esta característica torna possível a criação de praticamente qualquer arquitetura desejada, limitada basicamente pela quantidade de elementos presentes em cada FPGA. Além disso, o fato de ser possível criar várias cópias de um módulo ou mesmo vários módulos diferentes torna o FPGA uma ótima arquitetura para paralelização. Sua programação é realizada através de linguagens de descrição de hardware HDL (*Hardware Description Language*) sendo as mais comuns o VHDL (*VHSIC Hardware Description Language*) e o Verilog [25].

Além dos elementos lógicos, multiplicadores e memórias internas, FPGAs modernos são comercializados com uma arquitetura de SoC, possuindo dentro do chip, além da área reprogramável, um processador dedicado.

Atualmente estão presentes no mercado diversos fabricantes de FPGA incluindo Achronix Semiconductor, Actel, Altera, AMI Semiconductor, Atmel, Cypress Semiconductor, Lattice Semiconductor, QuickLogic, e Xilinx. Cada fabricante oferece dispositivos com diversos tamanhos em termos do número total de portas lógicas, memórias embarcadas e elementos de processamento DSP embutidos.

Em termos de configuração de processadores embarcados internamente no FPGA observam-se exemplos dos dois maiores fabricantes a Altera e a Xilinx. A Altera oferece em seus FPGAs a possibilidade de configurar o processador Nios II que é um *Soft Core Processor* ou *softcore* (processador reconfigurável que utiliza os elementos lógicos do FPGA) e a Xilinx oferece o *Soft Processor* MicroBlaze. Também estão disponíveis processadores *hardcore* (SoC) como o ARM na família Cyclone V da Intel/Altera [108] e na família Zynq da Xilinx [109], o que torna soluções de coprojeto mais eficientes [110].

Apesar de historicamente os FPGAs oferecerem um número limitado de células lógicas, baixa frequência e maior consumo de energia comparado às soluções em hardware dedicado (ASIC), lançamentos recentes de ambos os fabricantes que utilizam tecnologia de fabricação com transistores de 16 e 14 nm fornecem soluções com um número alto de células lógicas, na casa dos milhões e altas frequências, na faixa de 1 GHz. Dois exemplos práticos são os FPGAs da família Virtex-7 da Xilinx [111] com até 2.000.000 elementos lógicos e o FPGA da família Stratix 10 da Intel/Altera [112] com até 5.510.000 elementos lógicos. Ambas estas características aliadas com as tecnologias de fabricação são capazes de fornecer uma redução no consumo de energia de até 70% em comparação com a geração anterior, tornando estes dispositivos cada vez mais atrativos para uso em larga escala.

# 3 CONTROLADOR NMPC UTILIZANDO RNA/SVM

A primeira abordagem para acelerar o cálculo da solução do NMPC foi a de utilizar Redes Neurais Artificiais (RNAs) e de Máquinas de Vetor de Suporte (SVMs), mais especificamente SVRs (*Support Vector Regressor*) que são uma versão de SVM para problemas de regressão, com o intuito de substituir o papel do controlador e obter arquiteturas para altas frequências de amostragem. Esta abordagem é inspirada no trabalho de Ortega e Camacho [27] onde uma RNA foi utilizada em um controlador preditivo aplicado ao sistema de navegação de um robô móvel.

A metodologia aplicada é representada na Figura 3.1 e consiste em:

(i) **Cálculo da solução NMPC *offline* e Treinametno da RNA/SVR**: nesta etapa, a solução do controlador NMPC é calculada a partir de simulações numéricas no Matlab. O sistema é descrito com auxílio da ferramenta Simulink que é responsável por calcular as predições, dada uma entrada. Como *solver*, utilizou-se a função *fmincon* do Matlab, que é capaz de resolver problemas de otimização não-lineares com restrições. As soluções obtidas em cada período de amostragem são armazenadas e utilizadas para o treinamento da RNA ou SVR.

(iii) **Utilização da RNA/SVR como controlador do sistema**: a solução aproximada obtida no passo anterior é implementada em *hardware* a partir de uma descrição VHDL (*VHSIC Hardware Description Language*), gerada automaticamente a partir de um gerador automático desenvolvido neste trabalho. Em seguida é utilizada no lugar do controlador NMPC.

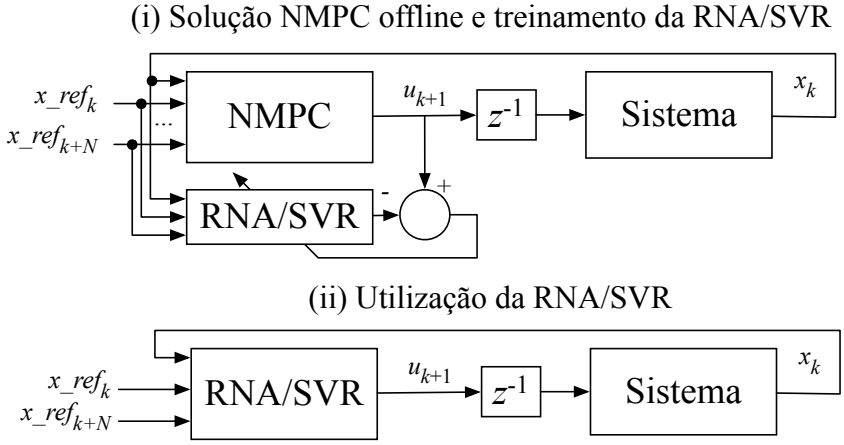



Figura 3.1: Método de treinamento da RNA/SVR.

Três abordagens foram utilizadas seguindo esta metodologia e geraram três publicações [113, 33, 94]. No primeiro trabalho [113], RNAs do tipo RBF foram utilizadas para aproximar o comportamento do controlador NMPC. Já no segundo trabalho uma ferramenta desenvolvida por Santos [33] foi utilizada para treinar SVRs utilizando além do *kernel* RBF um *kernel* polinomial mais simples que diminui o tempo de cálculo da solução bem como a utilização de recursos de hard-

ware. Um resumo da aplicação e dos resultados dos dois primeiros trabalhos é apresentado a seguir.

## 3.1 UTILIZAÇÃO DE RNA E SVM COMO CONTROLADOR NMPC DE UM BRAÇO ROBÓTICO

Nestes dois trabalhos, o caso de estudo selecionado foi o de controle de um braço robótico com uma junta de revolução elástica e um grau de liberdade como *benchmark*. Em muitas aplicações de robótica industrial a elasticidade de juntas advindas de folgas e elasticidade de engrenagens, correias de transmissão ou redutores do tipo harmônico são consideradas [114]. A junta elástica de revolução é modelada como uma mola com torção conectando a junta com o braço. O robô, pode ser visto na Figura 3.2 onde se observa que o mesmo está livre para girar no plano vertical.

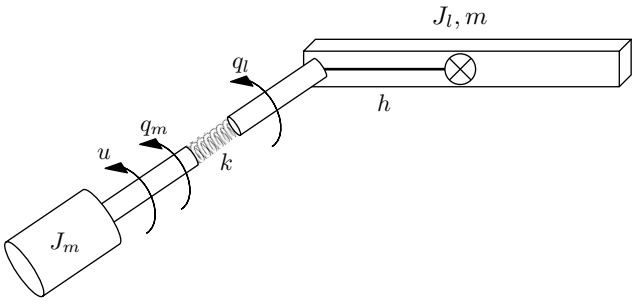



Figura 3.2: Robô manipulador com junta elástica.

A equação apresentada a seguir, descreve seu modelo e foi desenvolvida por Zhang, Polycarou e Parsini [115]:

$$J_l\ddot{q}_l + K(q_l - q_m) + mgh\sin q_l = 0 \quad (3.1a)$$

$$J_m\ddot{q}_m + F_m\dot{q}_m - K(q_l - q_m) = K_\tau u, \quad (3.1b)$$

onde $q_l$ e $q_m$ são as posições angulares do braço e do motor, respectivamente. As inércias do braço e do motor são representadas por $J_l$ e $J_m$. Para modelar o comportamento elástico da junta, uma mola de torção é utilizada, sua constante é definida como $K$. A massa e o comprimento do braço são definidos com o $m$ e $2h$, e a aceleração de gravidade por $g$. O ganho de amplificação é definido como $K_\tau$, e a viscosidade nominal do motor tem seu coeficiente representado por $F_m$. A entrada do sistema, $u$, é o torque do motor.

Este sistema (3.1) é relatado na literatura como um modelo de ensaio para métodos de detecção de falhas e isolamento (*FDI - Fault Detection and Isolation*) [115] e procedimentos de estimação não-linear [116]. Segundo [116, 115], valores adequados para cada um dos parâmetros são $9.3 \times 10^{-3}$ kg m$^2$ e $3.7 \times 10^{-3}$ kg m$^2$ para a inércia do braço e do motor , $1.8 \times 10^{-1}$ Nm/rad para a constante de torção da mola, $2.1 \times 10^{-1}$ kg para a massa do braço, $1.5 \times 10^{-1}$ m para o comprimento do braço, $8 \times 10^{-2}$ Nm/V para o ganho de de amplificação, $4.6 \times 10^{-2}$ Nm/V para

o coeficiente de viscosidade e $9.8$ m/s$^2$ para a aceleração da gravidade.

A descrição do sistema em espaço de estados discreto é apresentada na Equação (3.1).

$$\mathbf{x}(k+1) = f(\mathbf{x}(k), \mathbf{u}(k)), \quad \mathbf{y}(k) = \mathbf{x}(k), \tag{3.2}$$

onde $\mathbf{x}(k)$, $\mathbf{u}(k)$ e $\mathbf{y}(k)$ representam o vetor de estados, o vetor de entradas e o vetor de saídas do sistema, respectivamente. É importante observar que todos os estados são assumidos como mensuráveis, ou seja, não há o emprego de um estimador de estados.

O vetor de estados é definido por: $\mathbf{x} = [q_l \ \dot{q}_l \ q_m \ \dot{q}_m]^T$. As restrições para a saída do controlador (torque de entrada do sistema) são definidas pelo conjunto $\Omega = [-4, 4]$.

O problema de controle é expresso pela Equação (3.1).

$$\begin{aligned} \min_{u(k,k+N-1)} J(x(k), u(k, k+N-1)) \triangleq \sum_{k=1}^{N} \|x_{ref}(k) - x(k)\|^2, \\ \text{s.a. } u_i \in \Omega, \forall i \in [k, k+N-1], \end{aligned} \tag{3.3}$$

onde $x_{ref}(k)$ é a referência a ser seguida pelo estado $x(k)$ em um dado instante $k$, $N$ é o tamanho do horizonte de predição e o conjunto $\Omega$ define as restrições para as variáveis de controle. Utiliza-se ainda a notação $u_k^{opt}$ para representar a sequência ótima de controle dentro do horizonte de predição.

### 3.1.1 Treinamento das RNAs / SVMs

No primeiro caso, uma RNA do tipo RBF foi utilizada e treinada seguindo as etapas de *clustering* utilizando o método $k$-*means* para a definição dos centros da RBF seguido de uma etapa de tentativas e erros para a definição empírica do valor de $\sigma_m$ igual para todos os neurônios. Finalmente, calcula-se o valor dos coeficientes de peso a partir da fatoração QR. O melhor resultado foi encontrado com uma rede neural RBF com 5 neurônios.

No segundo caso, a ferramenta NIOTS (*Nature Inspired Optimization Tools for SVM*) proposta por Santos et al. [33] foi utilizada. Seu funcionamento é baseado em uma abordagem de otimização multi-objetivo através do algoritmo *Multi-Objective Particle Swarm Algorithm* (MOPSO) apresentada em [117] para o treinamento da solução. Detalhes de seu funcionamento podem ser vistos em Santos et al. [33]. As opções de configuração da ferramenta utilizadas no treinamento foram: (a) *Classificador*: regressão, (b) *Fator de Diversidade*: nenhum, (c) *Kernel*: RBF / Polinomial, (d) *Otimizador*: MOPSO, e (e) *Função Objetivo*: (Inclui a LibSVM).

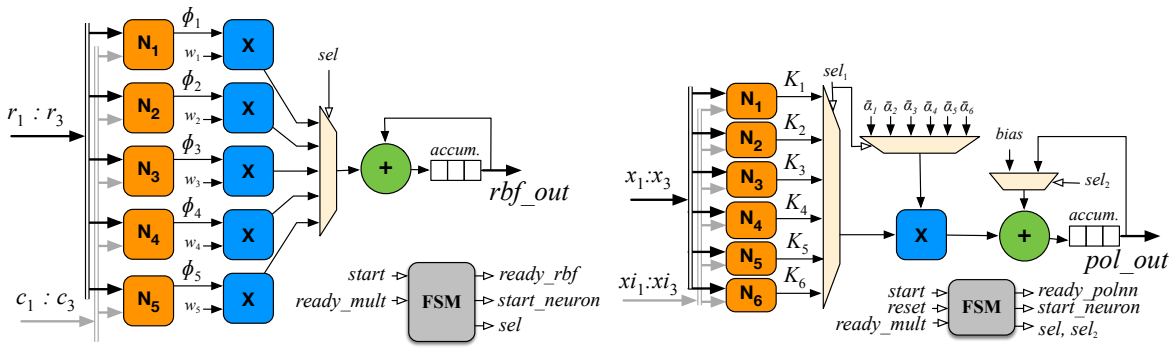
Em um primeiro resultado, utilizando o *kernel* RBF, a solução da SVR obteve 5 vetores de suporte, praticamente equivalentes aos 5 neurônios da RNA do tipo RBF. Em um segundo treinamento, utilizando um *kernel* polinomial de complexidade computacional menor, se obteve uma solução com 6 vetores de suporte, com um polinômio de grau 3.

A vantagem observada no uso do NIOTS é sua capacidade de treinamento automático, tes-

tando centenas ou até milhares de configurações entre número distinto de vetores de suporte, tipos diferentes de *kernel* e seus respectivos parâmetros.

### 3.1.2 Implementação em FPGAs

A RNA do tipo RBF e a SVR possuem estruturas semelhantes em três camadas, conforme descrito na Figura 2.3. Portanto, sua implementação em hardware é bastante similar com exceção do parâmetro de *bias* da SVR. A arquitetura em hardware foi descrita em VHDL sendo modular e com neurônios em paralelo. Esta arquitetura tira proveito do paralelismo inerente aos FPGAs para acelerar a obtenção da saída da rede. O controle de execução e sincronia dos cálculos dos neurônios é efetuado a partir de Máquinas de Estados Finitos (FSMs), tanto internamente em cada neurônio quanto externamente no controle da rede como um todo. A arquitetura foi inspirada no trabalho de Ayala et al. [118] e pode ser vista na Figura 3.3.




(a) (b)

Figura 3.3: (a) Arquitetura da RNA do tipo RBF; (b) Arquitetura da SVR.

O neurônio / *kernel* do tipo RBF utiliza a função de ativação Gaussiana conforme descrito na Equação (2.28). Já o *kernel* polinomial é expresso pela Equação (3.4).

$$K(x_1, x_2) = (x_1 \cdot x_2 + 1)^d \tag{3.4}$$

Cada neurônio possui dois somadores *FPadd*, um multiplicador *FPmul* e uma unidade para cálculo da exponencial *FPexp*. Todas as operações são implementadas utilizando unidades em ponto flutuante com tamanho de palavra customizável e bibliotecas trigonométricas baseadas no padrão IEEE 754 [119], conforme desenvolvido por Muñoz et al. em [120] e [121]. A utilização da aritmética de ponto flutuante provê uma melhor precisão e uma larga faixa de dados sendo adequada para números reais de pequena ou larga escala.

Para minimizar o número de somadores e multiplicadores necessários na implementação do módulo e manter bom desempenho, multiplexadores foram utilizados para criar um *pipeline* interno e redirecionar os dados conforme necessário. A FSM controla o processo utilizando sinais de *start* e *ready* para as unidades *FPadd*, *FPmul* e *FPexp*, e as sincroniza a partir dos seletores dos multiplexadores ($sel_1$, $sel_2$).

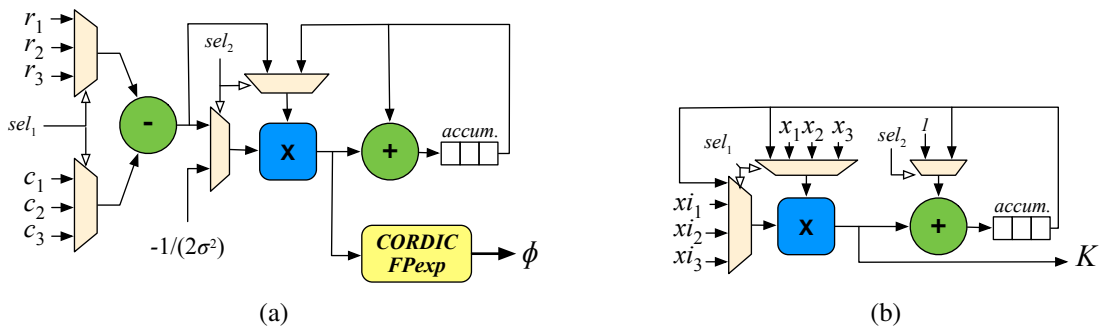



Figura 3.4: (a) Neurônio/*Kernel* Gaussiano RBF; (b) *Kernel* polinomial.

Para que fosse possível gerar e testar diversas configurações variando o número de entradas, quantidade de neurônios/*kernels* e a largura de bits das unidades de ponto flutuante, a ferramenta automatizada denominada vRBFgen em Matlab foi adaptada de [118]. Esta ferramenta recebe como entrada a topologia da rede, bem como os dados resultantes do treinamento e gera o código VHDL necessário para ser sintetizado em FPGA.

### 3.1.3 Resultados e discussões

Os resultados do desempenho dos controladores utilizando a RNA-RBF e a SVR com *kernel* polinomial implementados em FPGA são mostrados na Figura 3.5, onde o sinal de controle do motor é apresentado e na Figura 3.6, onde a posição do motor calculada pela solução SVR-POL é comparada com a solução RNA-RBF e a solução do NMPC em Matlab.

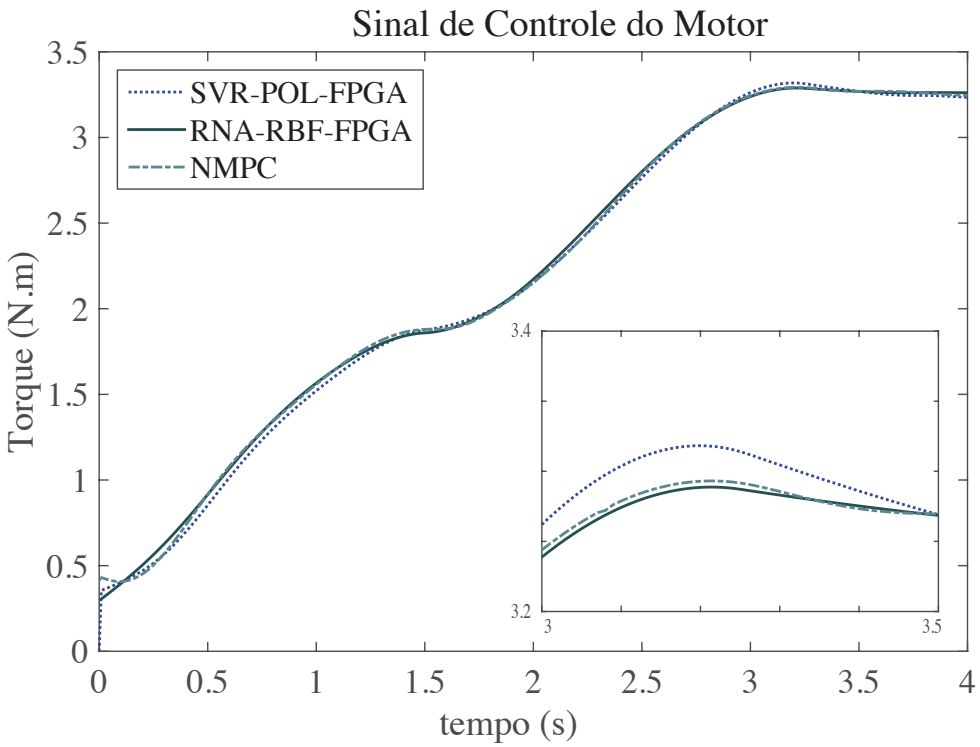



Figura 3.5: Torque do motor no tempo comparando as soluções NMPC, RBFNN e SVM-POL.

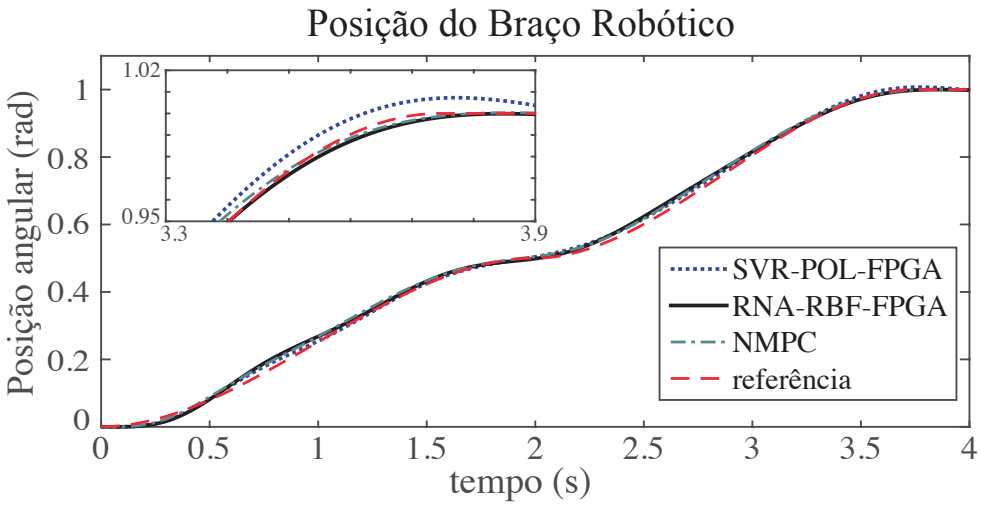



Figura 3.6: Posição do motor no tempo.

A Tabela 3.1 mostra os resultados de síntese comparando a solução RNA-RBF com a SVR-POL para o FPGA Altera Cyclone IV (EP4CE115F29C7) cujas características são apresentadas no cabeçalho da própria tabela. É possível observar que a solução SVR-POL que utiliza 6 blocos de kernel (KBs) e mantém a mesma arquitetura de rede da RNA-RBF mas somente $44.5\%$ do número de elementos lógicos (LEs), além de reduzir em $30\%$ o número de elementos DSP utilizados. Comparando os *kernels* individualmente, observamos que o polinomial utiliza $43,2\%$ dos números de elementos lógicos do *kernel* RBF. A última coluna da tabela apresenta o número total de ciclos de *clock* necessários para calcular a solução. Levando em conta a frequência máxima de cada arquitetura é possível calcular o tempo necessário para o processamento onde se observa que o a solução SVM-POL é $2.09$ vezes mais rápida que a RNA-RBF e é capaz de calcular a ação de controle em apenas 0,59 $\mu$s, viabilizando sua aplicação em sistemas com taxas de amostragem na faixa dos megahertz.

Tabela 3.1: Resultados de síntese para as arquiteturas no FPGA da Altera.

| Arquitetura | LE (114,480) | DSP 9x9 (532) | Freq. MHz | Ciclos de *clock* |
|---|---|---|---|---|
| RNA-RBF (5 Ns) | 14,638 | 70 | 85.7 | 105 |
| SVR-POL (6 KBs) | 6,952 | 49 | 69.8 | 41 |
| RBF-KERNEL | 2,649 | 7 | 94.0 | 89 |
| POLINOMIAL-KERNEL | 1,145 | 7 | 73.7 | 19 |

Apesar dos resultados e alta frequência alcançada para o cálculo da solução de controle, esta solução é limitada por não conseguir generalizar o controle do sistema para qualquer tipo de entrada.

## 3.2 APLICAÇÃO DA SVM NO CONTROLE DE UM PÊNDULO INVERTIDO

No terceiro trabalho [94], o método de treinamento das SVRs foi aprimorado para melhorar a generalização da solução. O problema de equilíbrio de um pêndulo invertido foi utilizado como estudo de caso conforme a descrição a seguir.

Para ilustrar o desenvolvimento dos algoritmos de controle com a abordagem NMPC, considera-se um sistema de um pêndulo invertido sobre um trilho. O modelo do sistema utilizado foi descrito por Alaniz [122] e mais tarde utilizado por Mercieca e Fabri [32] para aplicar técnicas de controle preditivo.

A Figura 3.7 apresenta o modelo do sistema juntamente com os digramas de corpo livre. Em seguida, as Equações (3.5) e (3.6) descrevem a dinâmica do modelo não-linear do sistema. As equações são expressas em termos das variáveis de estado $x$, $\dot{x}$, $\theta$ and $\dot{\theta}$ que são, respectivamente, a posição do carro no trilho, sua velocidade linear, o ângulo de inclinação do pêndulo e sua velocidade angular. As constantes presentes são: $M$ massa do carro, $m$ massa uniformemente distribuída do corpo do pêndulo, $l$ é a metade do comprimento do pêndulo, $b$ representa o coeficiente de atrito da superfície, $h$ é o coeficiente de atrito de rotação, $g$ é a aceleração da gravidade, e $u$ representa a ação de controle que é a força aplicada ao carro.

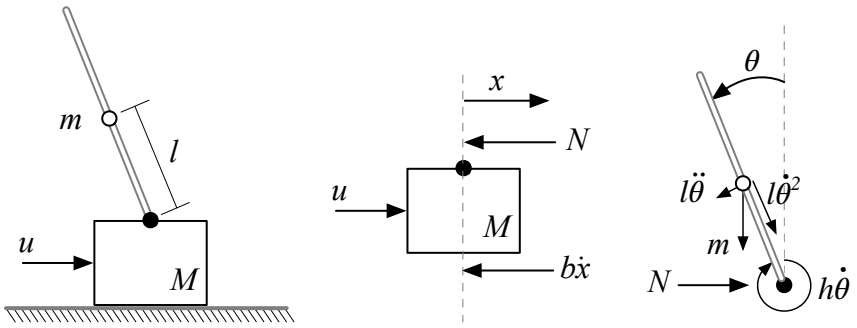



Figura 3.7: (a) Pêndulo invertido; (b) Diagrama de corpo livre 1; (c) Diagrama de corpo livre 2.

$$\ddot{x} = \frac{1}{M+m}[u - b\dot{x} - ml\ddot{\theta}\cos(\theta) + ml\dot{\theta}^2\sin(\theta)] \tag{3.5}$$

$$\ddot{\theta} = \frac{3}{4ml^2}[mgl\sin(\theta) - ml\ddot{x}\cos(\theta) - h\dot{\theta}] \tag{3.6}$$

Os valores utilizados para as constantes são $14,6$ kg para $M$, $7,3$ kg para $m$, $l$ igual a $1,2$ m, $b$ igual a $14.6$ kg/s, $h$ igual a $0,0136$ kg.m$^2$/s e, finalmente, $g$ igual a $9,81$ m/s$^2$.

A função custo utilizada se refere à Equação (2.10) e, para facilitar a visualização foi re-escrita

como:

$$J(\mathbf{x}(k), \mathbf{u}(k)) \triangleq \sum_{i=1}^{N-1} ||\tilde{\mathbf{y}}(k+i) - \mathbf{y}_{ref}(k+i)(k)||_Q^2 + \sum_{i=0}^{Nc-1} ||\tilde{\mathbf{u}}(k+i) - \mathbf{u}_{ss}||_R^2 + ||\mathbf{x}(N) - \mathbf{x}_{ss}||_{Q_f}^2. \tag{3.7}$$

Para que seja possível controlar o sistema é necessário definir os parâmetros relacionados à sua dinâmica como a taxa de amostragem ($Ta$), as restrições físicas dos estados ($x_{min}$, $x_{max}$) ou das variáveis controladas ($y_{min}$, $y_{max}$) e as restrições do atuador ($u_{min}$, $u_{max}$ e $\Delta u_{max}$). Além disso, devem definidos os parâmetros específicos do controlador MPC que são o horizonte de predição ($N$), o horizonte de controle ($Nc$), a matriz de ponderação dos estados ($Q$), a matriz de ponderação do estado estacionário ($Qf$) e a matriz de ponderação do sinal de controle ($R$). A Tabela 3.2 apresenta os parâmetros de referência utilizados em [32].

Tabela 3.2: Parâmetros dinâmicos do sistema e do controlador MPC [32].

| Parâmetro | Cenário 1 | Cenário 2 |
|---|---|---|
| $Ta$ | 100 ms | 100 ms |
| $x_{min}$ | -1,5 m | -0,5 m |
| $x_{max}$ | 1,5 m | 0,5 m |
| $u_{min}$ | -300 N | -50 N |
| $u_{max}$ | 300 N | 50 N |
| $\Delta u_{max}$ | 50 N | 50 N |
| $N$ | 20 | 20 |
| $Nc$ | 20 | 20 |
| $Q$ | diag(1, 1, 100, 1) | diag(1, 1, 100, 1) |
| $Qf$ | diag(0, 0, 0, 0) | diag(0, 0, 0, 0) |
| $R$ | 1 | 1 |

### 3.2.1 Método de treinamento multi-objetivo da SVR com a ferramenta NIOTS

Mais uma vez, aplica-se a ferramenta NIOTS para efetuar o treinamento. Conforme mencionado anteriormente, esta ferramenta utiliza uma abordagem de otimização multi-objetivo (MOOP - *Multi-Objetive Optimization Problem*) para realizar o treinamento da SVR, mais especificamente, uma versão multi-objetivo do algoritmo PSO, descrito na Seção 2.5, o MOPSO. O motivo é baseado no fato de que problemas de otimização complexos normalmente possuem vários critérios a serem atendidos e, em alguns casos estes critérios são conflitantes. No caso da SVR, por exemplo, se deseja que a mesma represente os dados de treinamento com a maior acurácia possível e com a menor complexidade possível. Porém, o aumento da acurácia geralmente implica em um maior número de vetores de suporte e vice-versa. Este fato está diretamente relacionado com o problema de definição dos hiperparâmetros que é um problema multimodal, especificamente o

problema de Seleção de Parâmetros da SVM que pertence à classe de problemas definidos como multi-objetivo [123], otimização multi-critério, conforme definido por Coello em [124].

A Equação (3.8) apresenta o problema de Seleção de Parâmentros da SVM modelado como um MOOP.

$$\begin{aligned} \min\ \mathbf{F}(C,\gamma,\epsilon) &= (MSE, \#SV, 1-R^2) \\ s.t.\ \ 2^C, 2^\gamma &\in \left[2^{LB},\ 2^{UB}\right] \\ \epsilon &\in \left[0,\ UB_\epsilon\right], \end{aligned} \tag{3.8}$$

onde MSE é o erro médio quadrático do conjunto de validação, $\#SV$ é a cardinalidade do conjunto de vetores de suporte e, $R^2$ é o coeficiente de correlação de Pearson [125]. Este dois últimos critérios são avaliados pelas Equações (3.9) e (3.10).

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2, \tag{3.9}$$

onde a variável $n$ representa a cardinalidade do conjunto de validação, $\hat{y}$ é a predição da SVR e $y$ são os rótulos do conjunto de validação.

$$R^2 = \left(\frac{\sum_{i=1}^{n}(y_i - m)(\hat{y}_i - \hat{m})}{\sqrt{\sum_{i=1}^{n}(y_i - m)^2 \sum_{i=1}^{n}(\hat{y}_i - \hat{m})^2}}\right)^2, \tag{3.10}$$

onde $m$ representa a média de $y_i$ e $\hat{m}$ representa a predição média da SVR. A métrica $R^2$ foi transformada na função objetivo para representar um problema de minimização ao invés de maximização.

Um outro conceito importante para compreender as soluções de problemas multi-objetivo é o de *dominância* pelo qual é possível identificar as melhores soluções. Este conceito pode ser definido como: um vetor $\mathbf{u} = (u_1, u_2, ..., u_k)$ é dito que domina outro vetor $\mathbf{v} = (v_1, v_2, ..., v_k)$ (denotador por $\mathbf{u} \preceq \mathbf{v}$) se e somente se $\mathbf{u}$ é parcialmente menor que $\mathbf{v}$, i.e. $\forall i \in \{1, ..., k\}, u_i \leq v_i \wedge \exists i \in \{1, ..., k\} : u_i < v_i$. Na prática, isso significa que soluções com baixo custo para um objetivo "a" e custo mais elevado para o objetivo "b" é considerada equivalente a uma solução com alto custo no objetivo "a" e baixo no objetivo "b", quando ambos os objetivos tem a mesma importância. Como resultado, obtém-se um conjunto de soluções não dominantes que, em princípio tem todas a mesma importância. Este conjunto é chamado de *Conjunto de Pareto* e sua imagem chamada *Fronteira de Pareto*.

O procedimento de otimização realizado pelo MOPSO é muito semelhante ao algoritmo do PSO descrito na Seção 2.5, ou seja, um conjunto de partículas é inicializado aleatoriamente, uma função custo multi-objetivo é avaliada, em seguida define-se os melhores valores locais e o melhor global. Por fim, a posição de cada partícula é atualizada e se inicia uma nova iteração. A diferença mais significativa é que não temos uma só melhor partícula, a melhor global, mas sim um conjunto não dominado de melhores soluções. Neste caso, a escolha da melhor global a cada iteração é realizada de maneira aleatória selecionando um dos elementos deste conjunto.

### 3.2.2 Procedimento de geração de dados de treinamento

A geração de dados de treinamento foi realizada a partir de simulações *offline* em Matlab. Foi considerado o Cenário 2, mais restrito e consequentemente mais difícil de controlar, descrito na Tabela 3.2, com exceção das restrições de excursão do carro sobre o trilho que foram modificadas para $x_{max} = 5\text{m}$ e $x_{min} = -5\text{m}$ e o os valores da matriz $Q$ que foram atualizados para $Q = diag(100, 1, 100, 1)$ para aumentar a ponderação da posição.

A trajetória de referência consiste em manter o equilíbrio do pêndulo enquanto sua posição no trilho é variada a partir de entradas do tipo degrau. A trajetória de referência é apresentada na Figura 3.8 juntamente com a solução *offline*. Os três gráficos representam de cima para baixo a posição do carro no trilho, o ângulo do pêndulo e a ação de controle (torque do motor). Para medir o desempenho de controle durante o treinamento utilizou-se a métrica do Erro Médio Quadrático (MSE - *Mean Square Error*). Neste caso o valor resultante de MSE foi calculado a partir da combinação linear de todos os estados e do sinal de controle ponderados pelas matrizes $Q$ e $R$, respectivamente. Como referência, o valor de MSE resultante da simulação apresentada na Figura 3.8 foi de $30,04$.

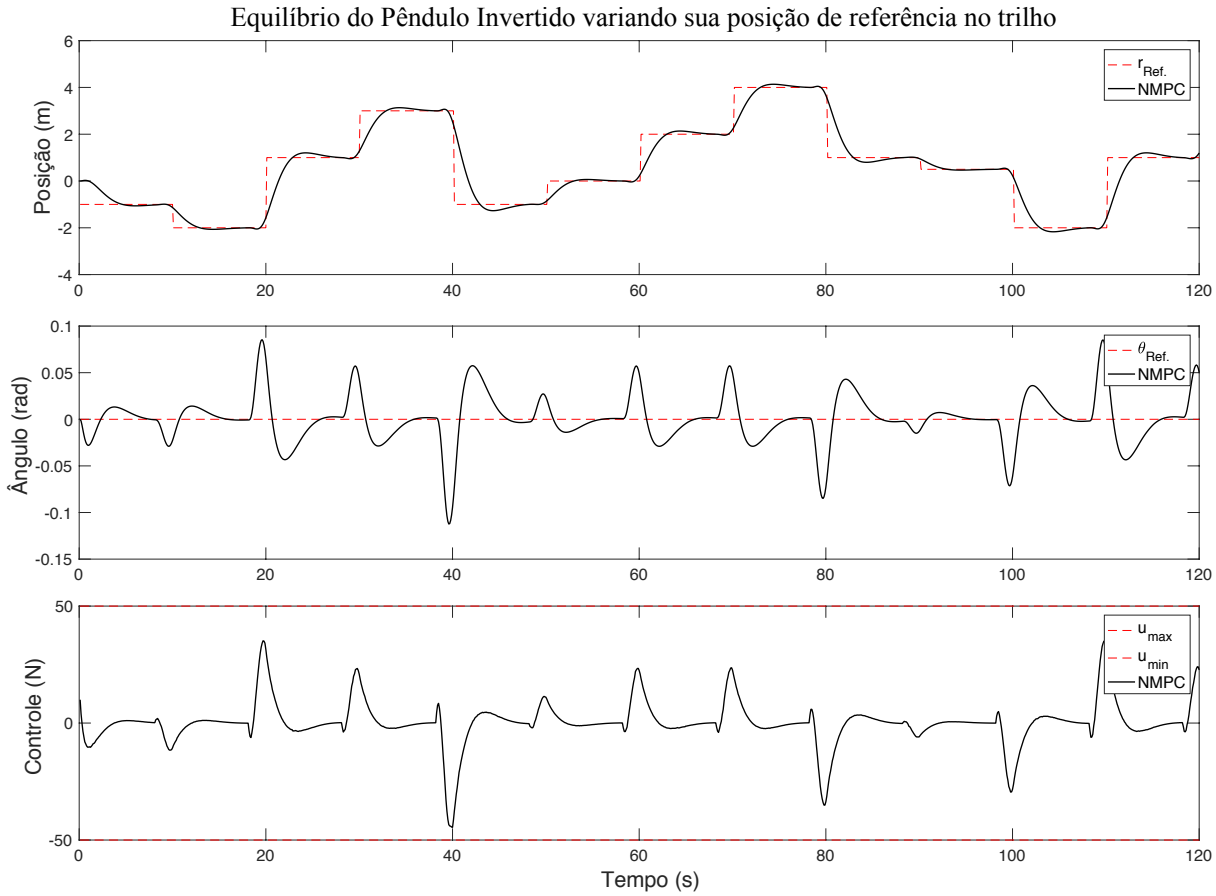
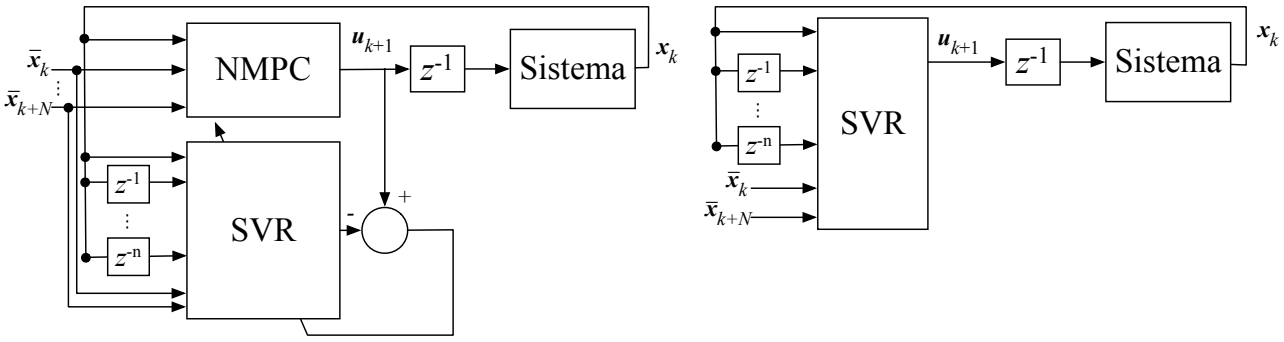



Figura 3.8: Trajetória de referência e resposta do controlador NMPC *offline*.

A Figura 3.9a representa o esquema de geração dos dados de treinamento da SVR seguido do treinamento em si. Já na in Figura 3.9b a solução da SVR é utilizada com controlador do sistema.




(a) Solução *offline* e treinamento da SVR (b) SVR como controlador

Figura 3.9: Modelo esquemático de: (a) geração de dados e treinamento da SVR. (b) SVR como controlador.

Para que a SVR pudesse capturar corretamente a dinâmica do sistema, foi necessário ainda definir o conjunto de entradas mais adequado. Como o princípio do NMPC envolve o conhecimento de estados passados e presentes bem como referências futuras, estes são as entradas candidatas. Assumindo, neste caso, que todos os estados podem ser lidos diretamente, o estado atual do sistema está disponível diretamente e estados passados podem ser obtidos a partir do armazenamento do histórico de estados em memória. Referências futuras também são conhecidas já que são entradas padrão do NMPC. A questão a ser respondida é: qual o tamanho do histórico passado e futuro necessário para obter um comportamento de controle adequado? Para isto, experimentos foram realizados utilizando os estados $x_k$, $x_{k-1}$ até $x_{k-10}$ e referências futuras iguais ao horizonte de predição. Considerando todas as possíveis entradas, uma solução com um total de 124 entradas (1 estado atual, 10 estados passados e 20 referências futuras todos multiplicados por 4 - número variáveis do sistema) seria obtida. Este é um número elevado, inclusive para uma implementação em FPGA. Portanto, uma simplificação foi realizada e após um processo de tentativas e erros, foram consideradas as 5 configurações apresentadas na Tabela 3.3.

Tabela 3.3: Configurações de entradas da SVR.

| Configuração | Núm de Entradas | Entradas |
|---|---|---|
| C1 | 8 | $\mathbf{x}_k, \bar{\mathbf{x}}_{k+1}, \bar{\mathbf{x}}_{k+N}$ |
| C2 | 14 | $\mathbf{x}_k, \mathbf{x}_{k-4}, \mathbf{x}_{k-8}, \bar{\mathbf{x}}_{k+N}$ |
| C3 | 16 | $\mathbf{x}_k, \mathbf{x}_{k-2}, \mathbf{x}_{k-4}, \bar{\mathbf{x}}_{k+1}, \bar{\mathbf{x}}_{k+N}$ |
| C4 | 20 | $\mathbf{x}_k, \mathbf{x}_{k-3}, \mathbf{x}_{k-6}, \mathbf{x}_{k-9}, \bar{\mathbf{x}}_{k+1}, \bar{\mathbf{x}}_{k+N}$ |
| C5 | 28 | $\mathbf{x}_k, \mathbf{x}_{k-2}, \mathbf{x}_{k-4}, \mathbf{x}_{k-6}, \mathbf{x}_{k-8}, \mathbf{x}_{k-10}, \bar{\mathbf{x}}_{k+1}, \bar{\mathbf{x}}_{k+N}$ |

Para melhorar a generalização da solução, é possível observar que o movimento do pêndulo em ambas as direções é simétrico. Sendo assim, sua posição relativa é suficiente para descrever sua dinâmica. Neste caso, ao invés do valor absoluto de cada estado ($x_k$), considerou-se o seu valor relativo à referência ($x_{ref_k} - x_k$). Além disso, considerando que o problema lida somente

com referências do tipo degrau, foi observado que os valores de $x_{ref_{k+1}}$ e $x_{ref_{k+N}}$ são o suficiente. Uma última otimização realizada foi a de utilizar somente os valores dos estados de posição e ângulo futuros. Portanto, o vetor $x_{ref_{k+1}}$, por exemplo, contém somente dois valores.

A partir da simulação do sistema apresentada na Figura 3.8, foi gerado um conjunto de treinamento para cada uma das configurações. Cada conjunto corresponde à uma simulação no intervalo de 0 a 120 segundos com um período de amostragem de 100 ms, gerando um total de 1201 pontos. Este conjunto foi ainda separado aleatoriamente em subconjuntos de treinamento, validação e testes com 70%, 15% e 15% das amostras em cada um, respectivamente. Esta proporção seguiu o padrão da ferramenta NIOTS. Finalmente, o *kernel* RBF Gaussiano foi selecionado.

A arquitetura de hardware segue o mesmo modelo implementado na Figura 3.3b, para a configuração geral da SVR, e o diagrama apresentado na Figura 3.4a.

### 3.2.3 Resultados e discussões

Os resultados obtidos no treinamento com a ferramenta NIOTS são apresentados a seguir. Para cada conjunto de treinamento, uma *Fronteira de Pareto* é gerada contento uma série de soluções da SVR com seus respectivos pesos e número de vetores de suporte bem como as suas métricas de desempenho. A título de exemplo, a Tabela 3.4 mostra os 10 melhores resultados alcançados para a configuração C5, com 28 entradas. As colunas de 2 a 4 apresentam os parâmetros das SVR enquanto as colunas de 5 a 7 listam as 3 métricas multi-objetivo utilizadas pela ferramenta para avaliar a qualidade do resultado. O resultado de índice 1 apresenta o menor índice de erro, porém com um valor excessivo de vetores de suporte (SV).

Tabela 3.4: Fronteira de Pareto para a SVR com 28 entradas.

| Índice | C | $\gamma$ | $\epsilon$ | Erro | SV | $1 - R^2$ | MSE 1 | MSE 2 | MSE 3 | MSE Soma |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 16,22 | -11,87 | 2,36 | 0,320 | 730 | 3,72e-3 | 91,10 | 68,87 | 78,97 | 238,94 |
| 2 | 12,33 | -8,45 | 2,57 | 0,893 | 13 | 8,83e-3 | 43,98 | 26,89 | 36,19 | 107,06 |
| 3 | 15,04 | -10,05 | 1,47 | 0,354 | 39 | 4,29e-3 | 32,92 | 19,87 | 28,62 | 81,41 |
| 4 | 15,71 | -11,51 | 2,26 | 0,778 | 20 | 5,10e-3 | 62,56 | 51,85 | 60,20 | 174,61 |
| 5 | 15,00 | -10,41 | 2,19 | 0,574 | 18 | 5,29e-3 | 111,90 | 101,76 | 110,48 | 324,15 |
| 6 | 13,57 | -8,99 | 2,51 | 0,693 | 16 | 5,85e-3 | 33,95 | 20,91 | 29,27 | 84,13 |
| 7 | 15,82 | -9,53 | 2,12 | 0,498 | 23 | 5,67e-3 | 2,47e3 | 2,49e3 | 2,53e3 | 7,49e3 |
| 8 | 12,47 | -7,66 | 1,75 | 0,450 | 26 | 6,00e-3 | 32,41 | 19,88 | 28,00 | 80,29 |
| 9 | 10,75 | -6,23 | 1,83 | 0,464 | 24 | 6,16e-3 | 31,96 | 18,33 | 27,46 | 77,75 |
| 10 | 12,03 | -6,97 | 2,22 | 0,524 | 21 | 6,39e-3 | 31,71 | 19,10 | 27,53 | 78,35 |

Porém, foi observado que mesmo a SVR sendo capaz de aproximar com um erro pequeno o sinal de controle à saída desejada, ou seja, ao sinal de controle simulado, isto não foi suficiente para garantir a estabilidade no controle do sistema. Este comportamento foi observado ao substituir o controlador NMPC pela SVR treinada conforme indicado na Figura 3.9(b). Portanto decidiu-se adicionar um novo passo de avaliação no treinamento de cada SVR que consistiu em simular a SVR treinada em trajetórias de referência diferentes da utilizada para o cenário de trei-

namento. No caso, três cenários adicionais com referências distintas foram adotados e a métrica de comparação foi o valor de MSE entre os estados do sistema e as respectivas trajetórias de referência. Estes resultados são apresentados nas colunas de 8 a 11 da Tabela 3.4 e o procedimento é ilustrado na Figura 3.10. Com esta nova métrica, é possível observar que um valor baixo do erro (coluna 5) não necessariamente se traduziu em uma boa solução. Um caso extremo é a SVR de índice 7 cujas simulações apresentaram um comportamento de instabilidade no controle levando a valores muito altos de MSE.

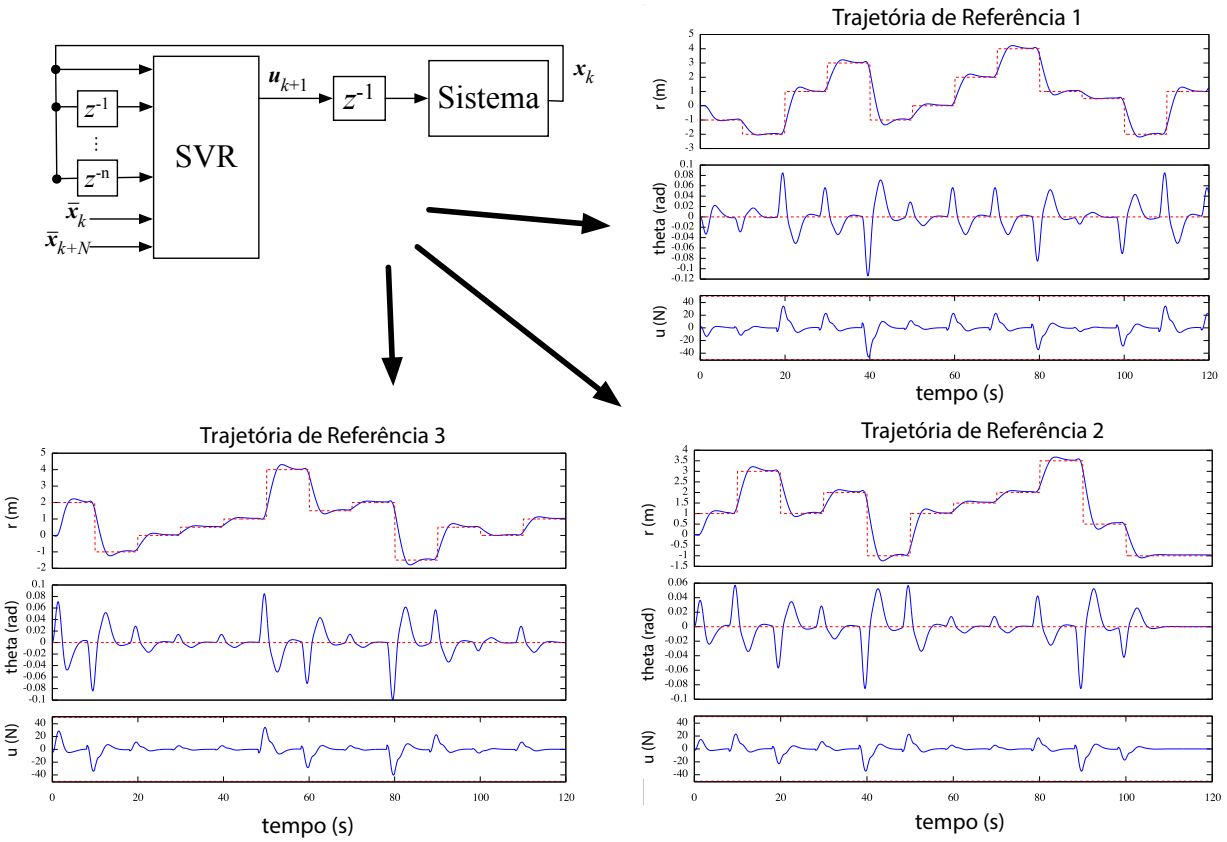



Figura 3.10: Validação do treinamento da SVR com 3 cenários de trajetórias.

Em seguida, a ferramenta NIOTS foi utilizada para gerar as soluções para as 5 configurações de entrada e os melhores resultados de cada treinamento estão apresentados na Tabela 3.5. Um critério de seleção adotada para escolher estes resultados foi o valor total de MSE (coluna 11) menor ou igual a 80, já que acima deste valor, foi observado muita oscilação no sinal de controle. Um segundo critério de seleção foi um baixo número de vetores de suporte (abaixo de 30) já que acima deste valor, começa a ser inviável sua implementação em hardware em FPGAs de baixo custo. Este último critério não foi atendido para as soluções com 14 e 16 entradas e, portanto, a solução com o menor número de vetores de suporte foi selecionada para comparação. A título de comparação, a primeira linha da Tabela 3.5 apresenta os valores de MSE da solução NMPC calculada para o treinamento.

Tabela 3.5: Melhores soluções da SVR para as 5 configurações de entradas.

| Solução | Entradas | C | $\gamma$ | $\epsilon$ | Erro | SVs | $1 - R^2$ | MSE 1 | MSE 2 | MSE 3 | MSE Soma |
|---|---|---|---|---|---|---|---|---|---|---|---|
| NMPC | - | - | - | - | - | - | - | 30,04 | 19,67 | 26,21 | 75,92 |
| SVR C1 | 8 | 15,20 | -7,97 | 1,76 | 33,24 | 18 | 0,099 | 33,22 | 19,34 | 28,14 | 80,7 |
| SVR C2 | 14 | 12,91 | -9,23 | 0,38 | 29,18 | 76 | 0,089 | 29,29 | 17,64 | 25,8 | 72,73 |
| SVR C3 | 16 | 14,79 | -11,18 | 0,80 | 29,06 | 37 | 0,089 | 29,55 | 17,53 | 25,79 | 72,87 |
| SVR C4 | 20 | 16,24 | -8,08 | 1,56 | 0,15 | 14 | 0,002 | 32,07 | 18,94 | 27,66 | 78,68 |
| SVR C5 | 28 | 10,75 | -6,23 | 1,83 | 0,46 | 24 | 0,006 | 31,96 | 18,33 | 27,46 | 77,75 |

Para demonstrar o desempenho de controle de cada solução, simulações com uma trajetória de referência diferente da utilizada no treinamento foram realizadas e são apresentadas na Figura 3.11. Na figura, são sobrepostas as soluções, juntamente com a trajetória calculada pelo controlador NMPC em Matlab e a referência em vermelho pontilhado. No terceiro gráfico, que representa o sinal de controle, é possível observar que as soluções com 8 e 28 entradas apresentam um sinal de controle com mais oscilações que as outras soluções.

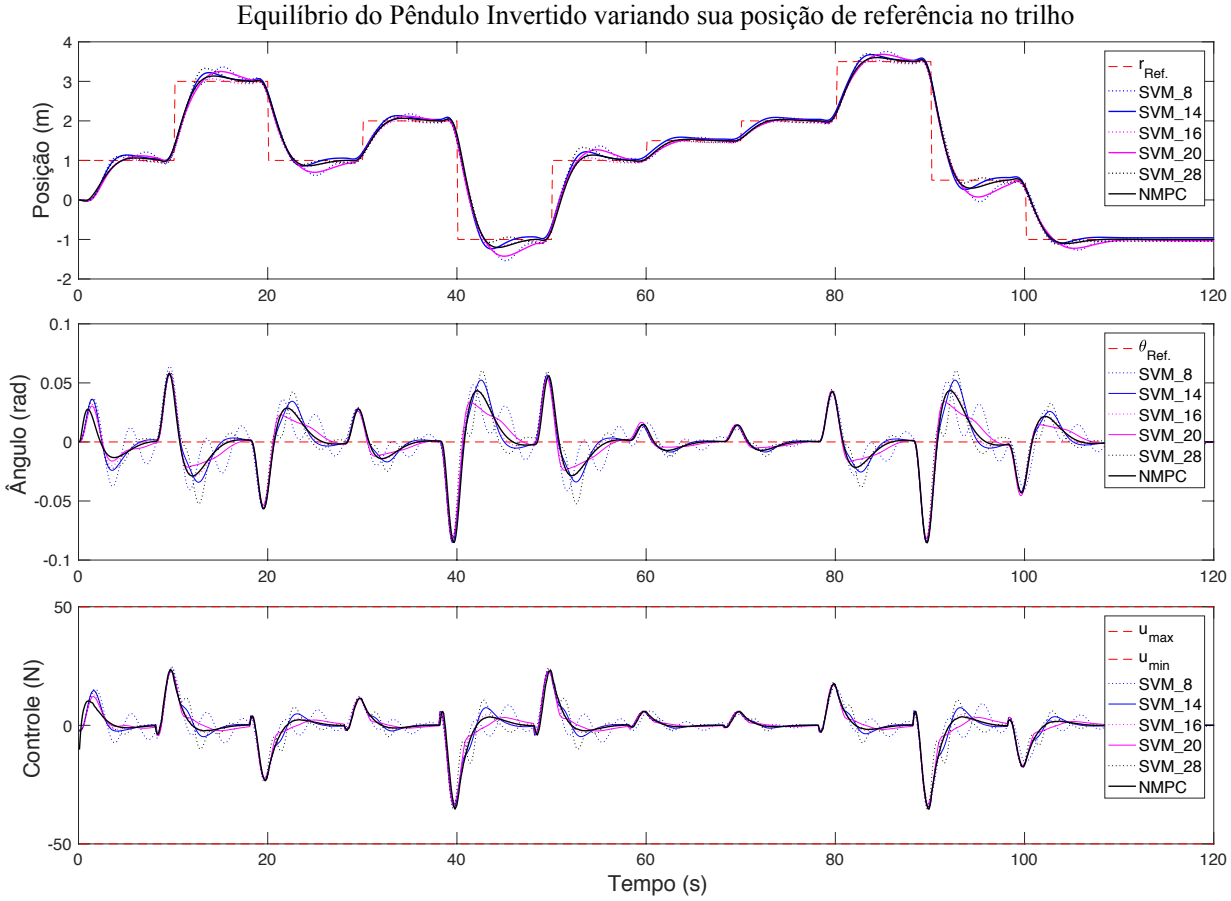



Figura 3.11: Simulação das 5 soluções de SVR seguindo a trajetória.

Finalmente, a Tabela 3.6 apresenta os resultados de síntese das arquiteturas em hardware para o FPGA Cyclone V (5CSXFC6D6F31C6N) da Altera. As soluções 2 e 3, apesar de terem atingido ótimos resultados de MSE, não puderam ser utilizadas por excederem o tamanho máximo de

recursos do FPGA utilizado. As últimas duas colunas da tabela apresentam o tempo de cálculo de cada solução no FPGA e no processador ARM Cortex-A53 at 1.2 GHz. Aqui é possível observar que em hardware, o tempo de cálculo varia entre 5 a 10 vezes mais rápido. A solução com 20 entradas foi escolhida como a melhor por apresentar bom desempenho de controle e a menor utilização de hardware.

Tabela 3.6: Resultados de síntese e desempenho no FPGA.

| SVR Config. | Entradas | SVs | LEs (41.910) | DSPs (112) | Freq. (MHz) | Ciclos | FPGA tempo ($\mu$s) | ARM time ($\mu$s) |
|---|---|---|---|---|---|---|---|---|
| 1 | 8 | 18 | 24,882 | 37 | 118,34 | 125 | 1,05 | 5,84 |
| 2 | 14 | 76 | 83,391 | 77 | - | 335 | - | 57,25 |
| 3 | 16 | 37 | 52,117 | 38 | - | 222 | - | 28,48 |
| 4 | 20 | 14 | 21,352 | 15 | 111,16 | 236 | 2,12 | 13,25 |
| 5 | 28 | 24 | 37,265 | 25 | 108,42 | 263 | 2,42 | 25,31 |

## 3.3 DISCUSSÕES FINAIS DO CAPÍTULO E CONTRIBUIÇÕES

Os resultados iniciais onde foram empregados RNAs e SVMs são bastante limitados pelo fato de não ser possível empregar trajetórias aleatórias ao sistema, somente trajetórias próximas da utilizada para o treinamento podem ser seguidas de maneira satisfatória.

Já na última abordagem, com o uso da ferramenta NIOTS, dois fatores contribuíram para a generalização da solução. O primeiro foi o uso de valores relativos para os estados e referências. O segundo fator foi incluir um simulador da SVM atuando como controlador NMPC com referências distintas aos dados de treinamento seguida da avaliação de seu desempenho com a métrica MSE

Apesar disso, é importante ressaltar que esta abordagem ainda possui limites. Um destes limites é o fato de nem todo sistema possuir estados que possam ser utilizados de maneira relativa. Um exemplo é o caso de estudo inicial da junta robótica onde seu ângulo de braço está sob a ação da gravidade e, portanto, a rotação em cada uma das direções tem efeito diferente na dinâmica do sistema. A outra limitação se dá ao fato de não ter sido possível garantir que o sistema irá obedecer as restrições de estados.

# 4 CONTROLADOR NMPC UTILIZANDO O PSO

Sendo o PSO um algoritmo de otimização, sua aplicação ao NMPC ocorre na etapa de busca da sequência de ações de controle (**u**) que minimize a função custo do NMPC. Neste caso, cada partícula do PSO representa um vetor de ações de controle (**u**), enquanto cada dimensão da partícula representa uma ação de controle ($u_k$) em um determinado instante $k$. A função custo a ser avaliada pelo PSO para cada partícula é composta pela etapa de predição ao longo do horizonte N seguida pela avaliação da função custo do próprio NMPC, tudo isto baseado no estado atual do sistema e na referência a ser seguida.

A Figura 4.1 ilustra o esquemático do controlador NMPC utilizando o PSO como otimizador não-linear.

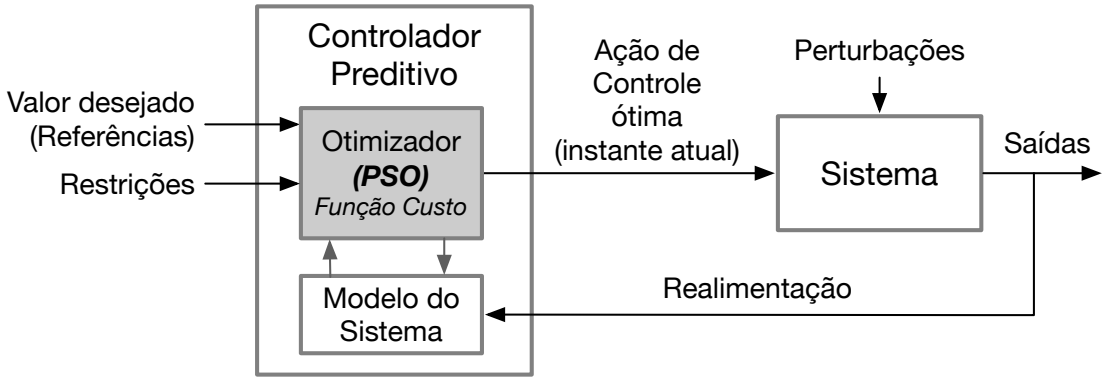



Figura 4.1: Esquemático do PSO como otimizador do controlador preditvo.

## 4.1 PROCEDIMENTO DE *SWING-UP* DE UM PÊNDULO INVERTIDO

Mais uma vez, para ilustrar o desenvolvimento do algoritmo de controle com a abordagem NMPC, utiliza-se o modelo do pêndulo invertido, descrito na Seção 3.2. Porém, ao invés do equilíbrio do mesmo no trilho, nesta seção iremos focar no procedimento de *swing-up* ou inversão e equilíbrio do pêndulo no trilho.

Conforme descrito na Tabela 3.2 da Seção 3.2, dois cenários iniciais são considerados. As Figuras 4.2 e 4.3 ilustram o procedimento de *swing-up* do pêndulo invertido para os Cenários 1 e 2, respectivamente.

Este procedimento consiste em iniciar com o pêndulo na orientação vertical virado para baixo (pendurado) e fazer com que o mesmo seja invertido até atingir a posição vertical virado para cima onde deve permanecer em uma condição de equilíbrio instável. Em ambos os casos, o estado inicial é $x = [0\ 0\ \pi\ 0]^T$ e a referência é $x_{ref} = [0\ 0\ 0\ 0]^T$.

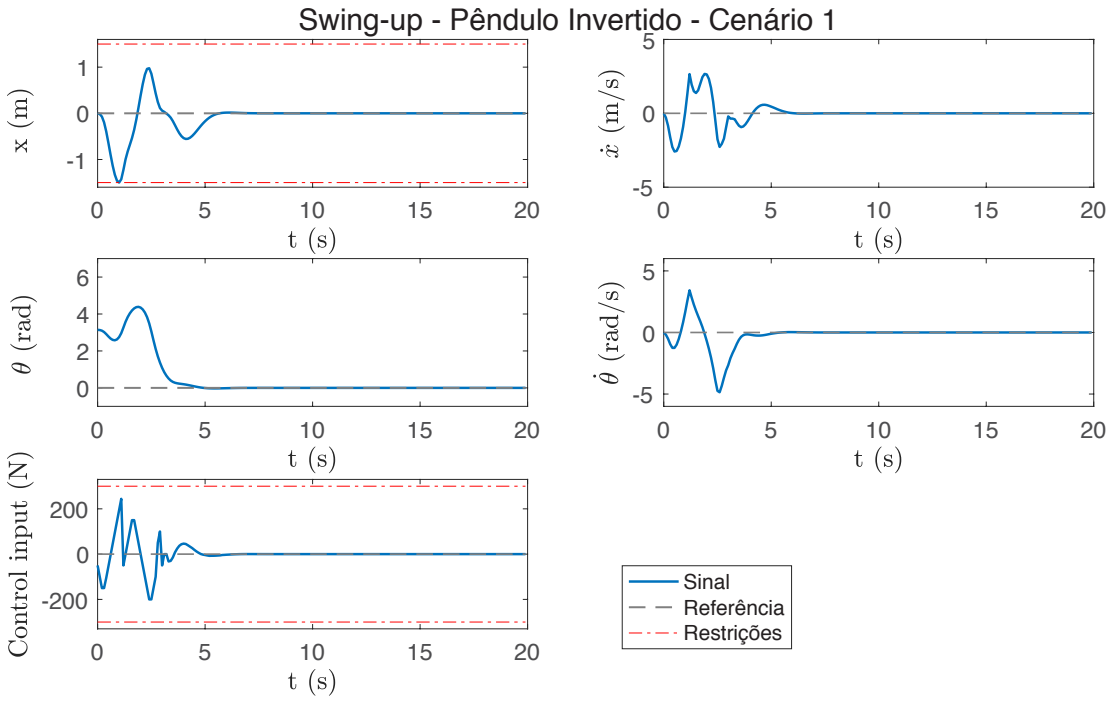



Figura 4.2: Simulação do procedimento de *swing-up* do pêndulo invertido - Cenário 1.

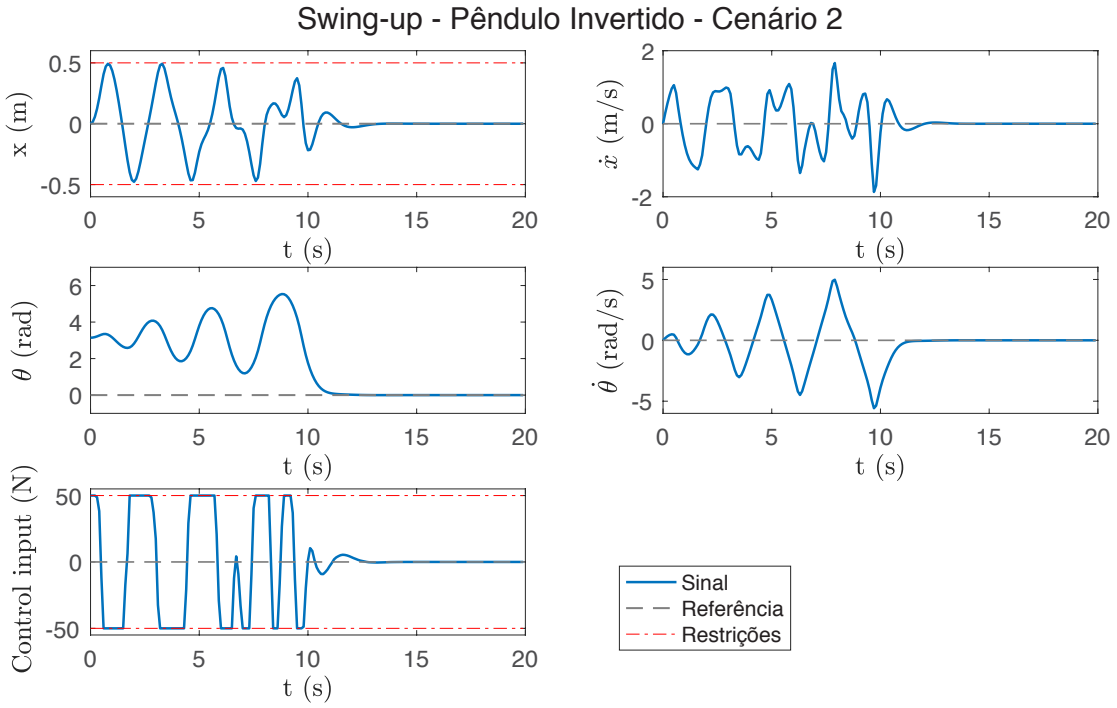



Figura 4.3: Simulação do procedimento de *swing-up* do pêndulo invertido - Cenário 2.

As simulações apresentadas foram realizadas em Matlab sem restrições de tempo de processamento e mostram o movimento desejado. No Cenário 1 as restrições são menos severas e o sistema atinge a referência em aproximadamente 5 segundos. Já no Cenário 2, tanto o sinal de controle quanto a excursão do pêndulo no trilho são mais limitados e o pêndulo tem que oscilar mais vezes até conseguir se inverter. Neste caso, o sistema atinge a referência após 13 segundos.

## 4.2 RESTRIÇÕES DO SINAL DE CONTROLE NO PSO

Para que o PSO obedeça às restrições impostas, algumas adaptações são necessárias. Em primeiro lugar, nota-se que o PSO sem modificações já aplica restrições ao espaço de busca e, já que as partículas representam as ações de controle, isto já atende aos critérios de restrição de $u_{max}$ e $u_{min}$. Porém, cada dimensão da partícula representa uma ação de controle em um instante e, neste caso, para respeitar as restrições de variação da ação de controle ($\Delta u_{max}$) se faz necessário adaptar tanto o algoritmo de inicialização das partículas quanto o algoritmo de atualização das partículas a cada iteração.

Utilizando o estudo de caso do Pêndulo Invertido e definindo os horizontes de predição e de controle como $N = 20$ e $Nc = 20$ com as restrições $u_{min}$ = -300 N, $u_{max}$ = 300 N, $\Delta u_{max}$ = 50 N, é possível demonstrar na Figura 4.4 a abordagem original do PSO para a inicialização das partículas e o resultado com a aplicação de restrições em $\Delta u$. Dois exemplos são mostrados utilizando 15 partículas cada um ($S$ = 15), no primeiro deles (parte de cima), o valor de $u(0)$, valor de u aplicado ao sistema no período de amostragem anterior, é de $u(0) = 0$, enquanto o segundo caso (parte de baixo) tem $u(0) = 250$.

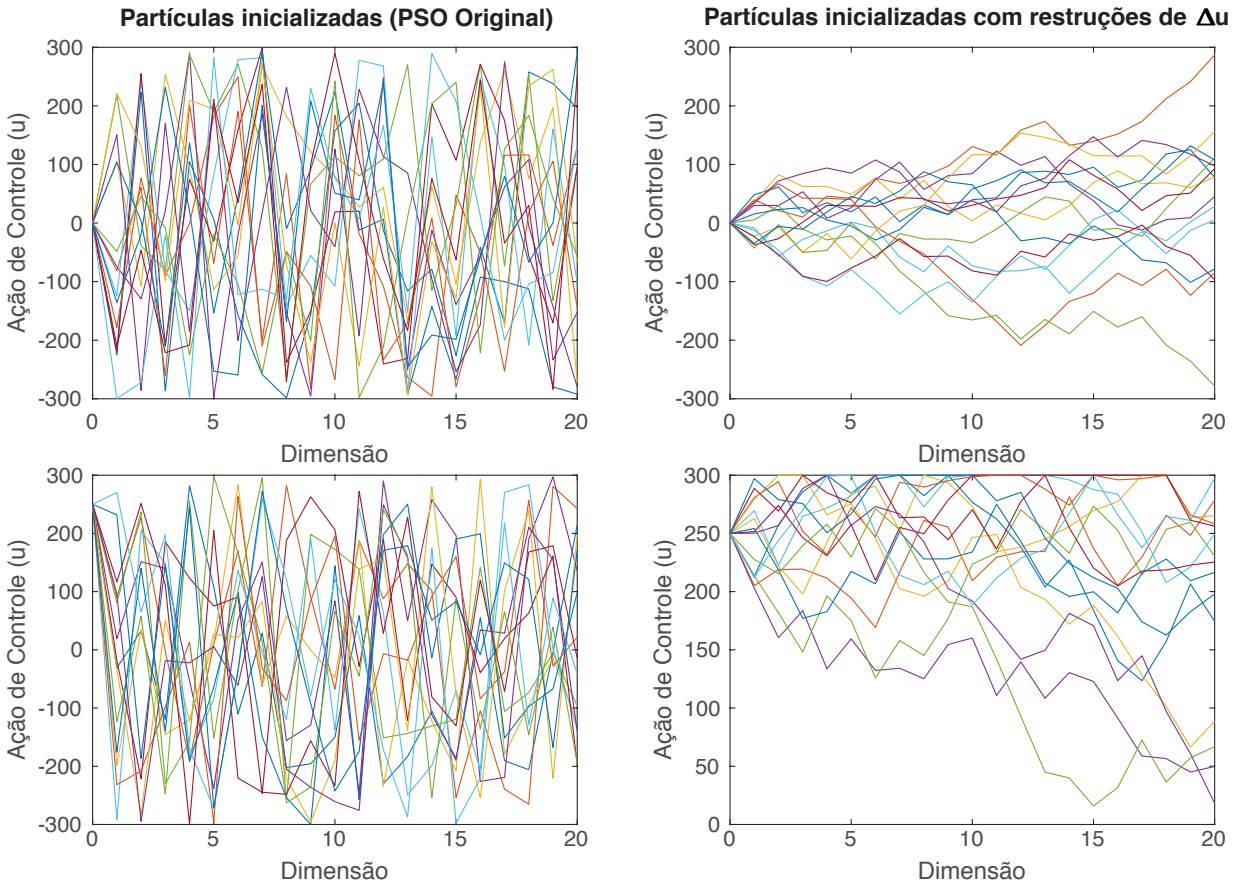



Figura 4.4: Inicialização de partículas do PSO.

## 4.3 RESTRIÇÕES DOS ESTADOS NO PSO

Além das restrições ao sinal de controle, o NMPC também possibilita restringir valores dos estados do sistema. Neste caso, como os estados futuros são consequência das ações de controle, não há uma maneira direta de restringir os mesmos à priori, ou seja, no momento em que as ações de controle estão sendo estimadas. Assim, se faz necessário verificar as ações de controle que implicam em estados futuros que venham a extrapolar restrições.

Uma abordagem para aplicar restrições nos estados do sistema controlado é a utilização de uma função de penalização associada à função custo do NMPC, conforme descrito por Boyd [126]. Neste caso, sempre que uma ação de controle incorrer em uma predição que leve o sistema a estados para fora dos limites definidos pelas restrições, haverá uma penalização no valor do custo desta ação. O trabalho de Parsopoulos e Vrahatis [127] apresenta experimentos realizados com diferentes variações de funções de penalização e mostra que a penalização pode ser fixa, ou seja, um custo adicional fixo cada vez que a restrição é extrapolada, ou variável, onde há um fator de penalização que cresce na medida em que os estados ultrapassam as restrições.

A abordagem utilizada neste trabalho é a de testar cada variável de estado estimada para cada período de amostragem ao longo do horizonte de predição e, caso a restrição seja extrapolada, aplicar uma penalização linearmente proporcional à distância entre o estado e a restrição, conforme a Equação 4.1.

$$h(x_{i,k}) = \begin{cases} 0, & \text{se } x_{min}(i) < x_{i,k} < x_{max}(i) \\ C(x_{min}(i) - x_{i,k}), & \text{se } x_i < x_{min}(i) \\ C(x_{i,k} - x_{max}(i)), & \text{se } x_i > x_{max}(i) \end{cases} \tag{4.1}$$

onde, $h(x_{i,k})$ é o fator de penalização do estado $i$ no instante estimado $k$, com $k = 1, \ldots, N$, $x_{max}$ e $x_{min}$ são os vetores de restrições máximas e mínimas de cada estado do sistema, respectivamente. A constante $C$ representa o fator de penalização que, por padrão, foi adotado com o valor de $10^6$. Este valor foi testado experimentalmente onde se observou que deveria exceder o fator máximo de penalização de cada estado limitados em $10^4$.

## 4.4 AJUSTES DOS PESOS DA FUNÇÃO CUSTO DO NMPC-PSO

A função custo utilizada pelo NMPC-PSO é uma composição da etapa de predição com a função custo do NMPC, apresentada na Equação 2.8. Na etapa de predição, adota-se o método de Runge-Kutta de 4ª ordem, conforme descrito na Equação 2.25, para realizar a predição a cada período de amostragem. Este procedimento é repetido $N$ vezes ao longo do horizonte de predição. Os dados de entrada são o estado atual do sistema ($x$), o vetor de ações de controle ao longo do horizonte de predição ($\hat{u}$), o tempo de amostragem ($Ta$) e o horizonte de predição ($N$).

O resultado é um vetor de $N$ estados estimados $\hat{x}$.

Em seguida, $\hat{u}$ e $\hat{x}$ são usados como entrada da função custo do NMPC juntamente com o vetor de referências $x_{ref}$ e os valores dos estados e sinal de controle no estado estacionário $x_{ss}$ e $u_{ss}$, respectivamente. Porém, a função custo ainda é composta por parâmetros constantes que os fatores de ponderação ou pesos, que devem ser ajustados para cada sistema. No caso da Equação 2.10 aplicada ao sistema do pêndulo invertido, temos as matriz $Q$ que pondera o desvio dos estados em relação à referência $x_{ref}$, a matriz $Qf$ que pondera os valores do estado ao final do horizonte de predição $\hat{x}_N$ em relação ao estado estacionário $x_{ss}$ e finalmente $R$ que, no caso do pêndulo que possui somente um atuador, é um escalar que pondera as ações de controle $\hat{u}$ em relação ao sinal de controle em estado estacionário $u_{ss}$.

A Equação 4.2 detalha o formato das matrizes $Q$, $Q_f$ e $R$ explicitando as nove constantes que devem ser definidas para o caso do pêndulo invertido.

$$Q = \begin{bmatrix} q_1 & 0 & 0 & 0 \\ 0 & q_2 & 0 & 0 \\ 0 & 0 & q_3 & 0 \\ 0 & 0 & 0 & q_4 \end{bmatrix}, \quad Q_f = \begin{bmatrix} q_{f1} & 0 & 0 & 0 \\ 0 & q_{f2} & 0 & 0 \\ 0 & 0 & q_{f3} & 0 \\ 0 & 0 & 0 & q_{f4} \end{bmatrix}, \quad R = r_1 \tag{4.2}$$

O procedimento de definir o valor para cada constante é geralmente realizado de modo manual pelo projetista do controlador através de um método de tentativa e erro guiado somente pelo o que se deseja que o sistema faça. Para o pêndulo invertido, deseja-se controlar o ângulo e a posição mesmo no trilho. Portanto, é natural que se escolha valores maiores para $q_1$ e $q_3$ em detrimento às demais contantes. Porém, este processo pode ser longo até que se encontre valores que levem o sistema à sua referência com rapidez e boa estabilidade.

Com isto em mente, um procedimento mais automatizado foi criado onde se utilizou do próprio algoritmo PSO, que aqui será referenciado como $PSO_{pesos}$ por diferenciação ao NMPC-PSO, para realizar uma busca pelos parâmetros $Q$, $Q_f$ e $R$. Neste caso, observando-se o desempenho desejado para o sistema ilustrado nas Figuras 4.2 e 4.3, é possível identificar que, após um determinado tempo de simulação, espera-se que o sistema atinja a referência desejada e fique estabilizado nesta posição. Portanto, para guiar a busca pelos pesos, foi definida uma função custo que utiliza o erro médio quadrático (MSE - *Mean Squared Error*) que compara o valor de cada variável de estado e do sinal de controle à sua referência ao longo do tempo de simulação. No caso dos Cenários 1 e 2, os valores dos estados são comparados a $x_{ref} = [0\ 0\ 0\ 0]^T$ enquanto o sinal de controle é comparado a $u_{ss} = 0$. Pelas mesmas figuras é possível ainda observar que, no caso do Cenário 1, o sistema se estabiliza após 5 segundos e no Cenário 2 após 13 segundos.

O procedimento de busca pelos pesos ($PSO_{pesos}$) utiliza $S$ partículas $p_{i,j}$ de dimensão 9 ($i = 1..S$, $j = 1..9$), correspondentes às nove constantes que se deseja encontrar. O espaço de busca para cada peso foi definido variando entre $10^{-2}$ e $10^4$. Para percorrer este intervalo, foi definido que cada partícula do $PSO_{pesos}$ represente o valor da potência de dez e portanto, cada dimensão da partícula variar dentro do intervalo [-2, 6]. O valor de cada partícula é então transformado nas

matrizes de peso conforme a Equação 4.3.

$$Q = \begin{bmatrix} 10^{p_{i,1}} & 0 & 0 & 0 \\ 0 & 10^{p_{i,2}} & 0 & 0 \\ 0 & 0 & 10^{p_{i,3}} & 0 \\ 0 & 0 & 0 & 10^{p_{i,4}} \end{bmatrix}, \quad Q_f = \begin{bmatrix} 10^{p_{i51}} & 0 & 0 & 0 \\ 0 & 10^{p_{i,6}} & 0 & 0 \\ 0 & 0 & 10^{p_{i,7}} & 0 \\ 0 & 0 & 0 & 10^{p_{i,8}} \end{bmatrix}, \quad R = 10^{p_{i,9}} \tag{4.3}$$

Em seguida, as matrizes $Q$, $Q_f$ e $R$ são enviadas ao simulador NMPC-PSO que realiza as simulações dos Cenários 1 e 2 ao longo de $T_{sim}$ = 20 segundos, retornando o valor dos estados $x_t$ e $u(t)$ para o período de simulação. Com estes resultados, calcula-se então o valor de MSE para cada cenário conforme apresentado na Equação 4.4.

$$MSE_{cenario} \triangleq \frac{1}{T_{sim} - T_{ref}} \left( \sum_{i=1}^{Nx} \left( \sum_{t=T_{ref}}^{T_{sim}} (x(t) - x_{ref}(t))^2 \right) + \sum_{t=T_{est}}^{T_{sim}} (u(t) - u_{ss})^2 \right) \tag{4.4}$$

, onde, $T_{ref}$ é o instante da simulação onde se espera que o sistema já tenha atingido a referência. Neste caso, para o Cenário 1 $T_{ref} = 5$s e para o Cenário 2 $T_{ref} = 13$s.

Finalmente, o problema de busca dos pesos se resume à minimização da função custo descrita pela Equação 4.5.

$$min\ J \triangleq MSE_{cenario1} + MSE_{cenario2} \tag{4.5}$$

Utilizando esta metodologia, algoritmo $PSO_{pesos}$ foi executado 20 vezes para determinar os pesos mais adequados para a referência desejada. Cada experimento utilizou $S = 30$ e um máximo de 200 iterações. Este valores foram obtidos experimentalmente onde se observou que $S > 30$ e o número de iterações acima de 200 não traziam melhores resultados. Considerano a natureza estocástica do PSO e portanto do NMPC-PSO, cada função custo de cada partícula foi ainda avaliada 10 vezes onde os valores de MSE retornados pela função foram a somas do MSE de cada rodada individual do NMPC-PSO.

Para assegurar que os pesos encontrados mantém o sistema estável, os 20 conjuntos de pesos foram testados em 1000 rodadas do NMPC-PSO para cada um dos cenários. Em seguida foram classificados na ordem do melhor desempenho para o pior. Dos 20 resultados, 8 apresentaram cenários onde o sistema não encontrou uma solução e se tornou instável. A Tabela 4.1 apresenta os 10 melhores resultados.

Para melhor avaliar e comparar, as cinco melhores soluções foram normalizadas e gerou-se o gráfico apresentado na Figura 4.5.

Em geral, se observa é que há dois grupos de resultados. Um deles (tons amarelos do gráfico) prioriza ponderar menos o valor de $q_1$ correspondente à penalização do estado da posição do pêndulo no trilho ($x$) e mais o valor de $q_{f1}$ correspondente ao custo terminal em estado es-

Tabela 4.1: Soluções para pesos ajustados

| Solução | $q_1$ | $q_2$ | $q_3$ | $q_4$ | $q_{f1}$ | $q_{f2}$ | $q_{f3}$ | $q_{f4}$ | $q_r$ |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 3,8e1 | 2,5e1 | 9,0e1 | 1,0e-2 | 1,8e2 | 5,6e1 | 1,1e-2 | 3,3e-1 | 1,9e-2 |
| 2 | 1,0e4 | 1,1e3 | 1,0e4 | 4,4e-2 | 1,0e-2 | 8,2e3 | 1,0e-2 | 5,2e2 | 2,2 |
| 3 | 1,0e6 | 1,1e5 | 1,0e6 | 1,0e4 | 1,3e4 | 1,0e6 | 6,2e1 | 1,2e1 | 1,6e2 |
| 4 | 1,7e-2 | 9,3e2 | 2,9e3 | 4,5e-1 | 9,9e3 | 6,7 | 1,0e-2 | 1,3e1 | 2,7e-1 |
| 5 | 7,2e-2 | 1,6e1 | 4,8e1 | 2,6e-2 | 1,6e2 | 7,5e-1 | 1,0e-2 | 1,5e1 | 1,0e-2 |
| 6 | 1,0e-2 | 5,7e1 | 2,4e2 | 2,5 | 6,6e2 | 2,3e2 | 1,0e-2 | 5,4e0 | 3,5e-2 |
| 7 | 1,0e4 | 2,9e3 | 1,0e4 | 6,5e-1 | 1,3 | 1,1 | 4,6e1 | 4,1 | 2,4 |
| 8 | 1,5e3 | 9,3e-2 | 1,2e3 | 2,0e-2 | 1,0e4 | 1,6e3 | 2,5 | 1,0e-2 | 2,9e-1 |
| 9 | 1,0e4 | 1,0e-2 | 6,8e3 | 1,0e-2 | 5,7e-1 | 9,6e3 | 2,8e2 | 2,9e-1 | 1,4 |
| 10 | 1,6e1 | 3,1e5 | 1,0e6 | 8,8 | 1,0e6 | 6,7e1 | 4,7 | 1,3 | 1,5e2 |

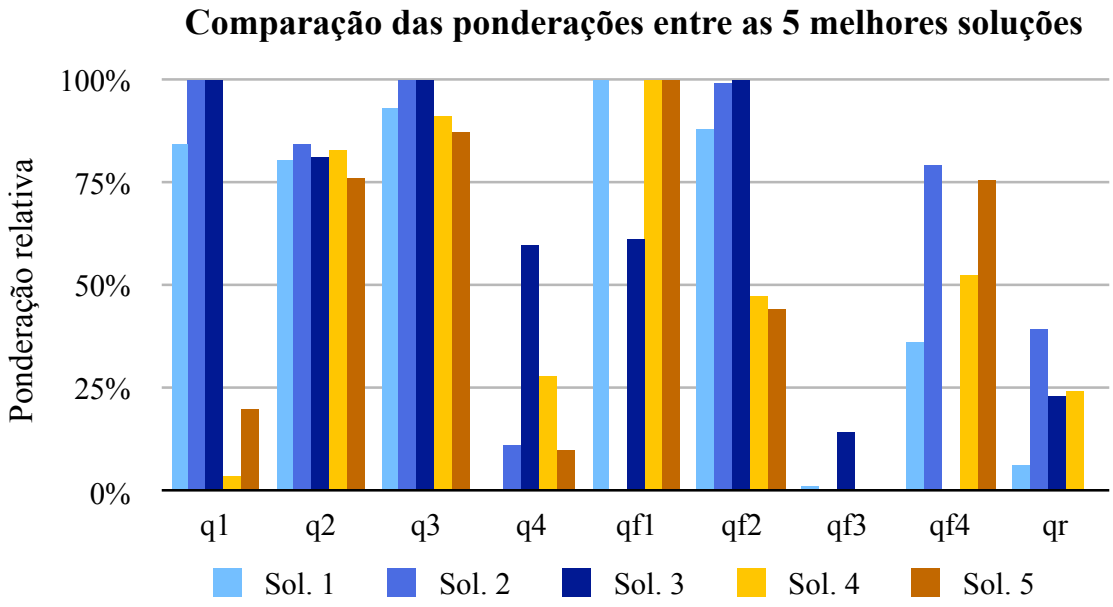



Figura 4.5: Comparação entre os pesos das cinco melhores soluções.

tacionário do mesmo estado. Um segundo grupo de resultados (tons azuis) tem um abordagem inversa. Além disso, em todos os casos, a ponderação de $q_3$ correspondente ao ângulo $\theta$ é alta e sua correspondente no custo terminal $q_f 3$ é baixa.

A partir destes resultados, escolheu-se a Solução 1 para o restante deste trabalho.

## 4.5 *KNOWLEDGE BASED* PSO (KPSO)

Dois fatores específicos do algoritmo PSO que afetam consideravelmente o tempo de cálculo da solução ótima são: o número de partículas ($S$) e o número total de iterações do algoritmo ($MaxIter$). No geral, quanto mais partículas utilizadas, mais áreas do espaço de busca são exploradas e, quanto maior o número de iterações do algoritmo, mais a solução será refinada até alcançar o resultado final.

Porém, para calcular a solução do NMPC em tempo-real com as restrições de poder de processamento impostas pelo contexto de sistemas embarcados, é necessário minimizar ao máximo estes dois parâmetros. Uma das maneiras de acelerar a busca pela solução ótima é ter um bom chute inicial. Considerando que o problema de otimização no NMPC está baseado em um sistema cujos estados não variam bruscamente entre os períodos de amostragem, e que, a cada período de amostragem, uma nova solução ótima é buscada, é razoável assumir que a solução encontrada para o período de amostragem anterior (vetor de ações de controle) possa ser uma boa candidata para a solução do próximo período de amostragem, deslocada no tempo. No caso do PSO, esta abordagem é apresentada apresentada por Sivananaithaperumal et al. [82] e nomeada de *Knowledge based* PSO (KPSO). Este conceito tem origem no algoritmo *Real-Time Iteration* (RTI) do trabalho de Diehl, Bock e Schlöder [73].

No caso do trabalho de Sivananaithaperumal et al. [82] o KPSO foi testado para sistemas não-lineares com horizontes de controle curtos (no máximo Nc = 3). Na prática, isso significa que as partículas do PSO tem dimensão igual a 3. São reportados resultados com um ganho de desempenho de aproximadamente 28% ao usar o KPSO em relação ao algoritmo regular do PSO.

Com o intuito de analisar o desempenho do PSO comparado ao KPSO, utilizou-se o Cenário 2 do modelo do pêndulo invertido para o qual foram realizados experimentos variando o número de partículas ($S$) do PSO de 10 a 40 e fixando $S$ = 10 para o KPSO. Além disso, variou-se o número de iterações dos algoritmo de 10 a 100. A métrica utilizada na comparação entre os resultados foi o MSE do valor de cada variável de estado e do sinal de controle em relação às suas respectivas referências, neste caso $x_{ref} = [0\ 0\ 0\ 0]^T$ e $u_{ss} = [0]$. O valor do MSE foi calculado entre os instantes $t = 13s$ e $t = 20s$. Cada experimento foi repetido 30 vezes e os resultados apresentados representam as médias dos valores de MSE. Neste intervalo, conforme apresentado na Figura 4.3, o sistema com bom desempenho já atingiu o estado de estabilidade. Neste caso, um sistema com desempenho ruim não atingirá a referência até este intervalo e, portanto, apresentará um maior valor de MSE.

Os resultados são apresentados na Figura 4.6, onde é possível observar que há uma relação direta entre o número de partículas e a qualidade das soluções. Além disso, quanto maior o número de iterações, menor o erro. Porém, ao comparar a solução do PSO regular com o KPSO, fica claro o salto em desempenho oferecido com esta abordagem. Neste caso, utilizando o KPSO com apenas 10 partículas é possível obter um desempenho melhor que a solução regular com 40 partículas. Quanto ao número de iterações, se observa ainda que há uma melhora significativa de

10 a 20 iterações, porém após este valor, as melhoras são menos significativas.

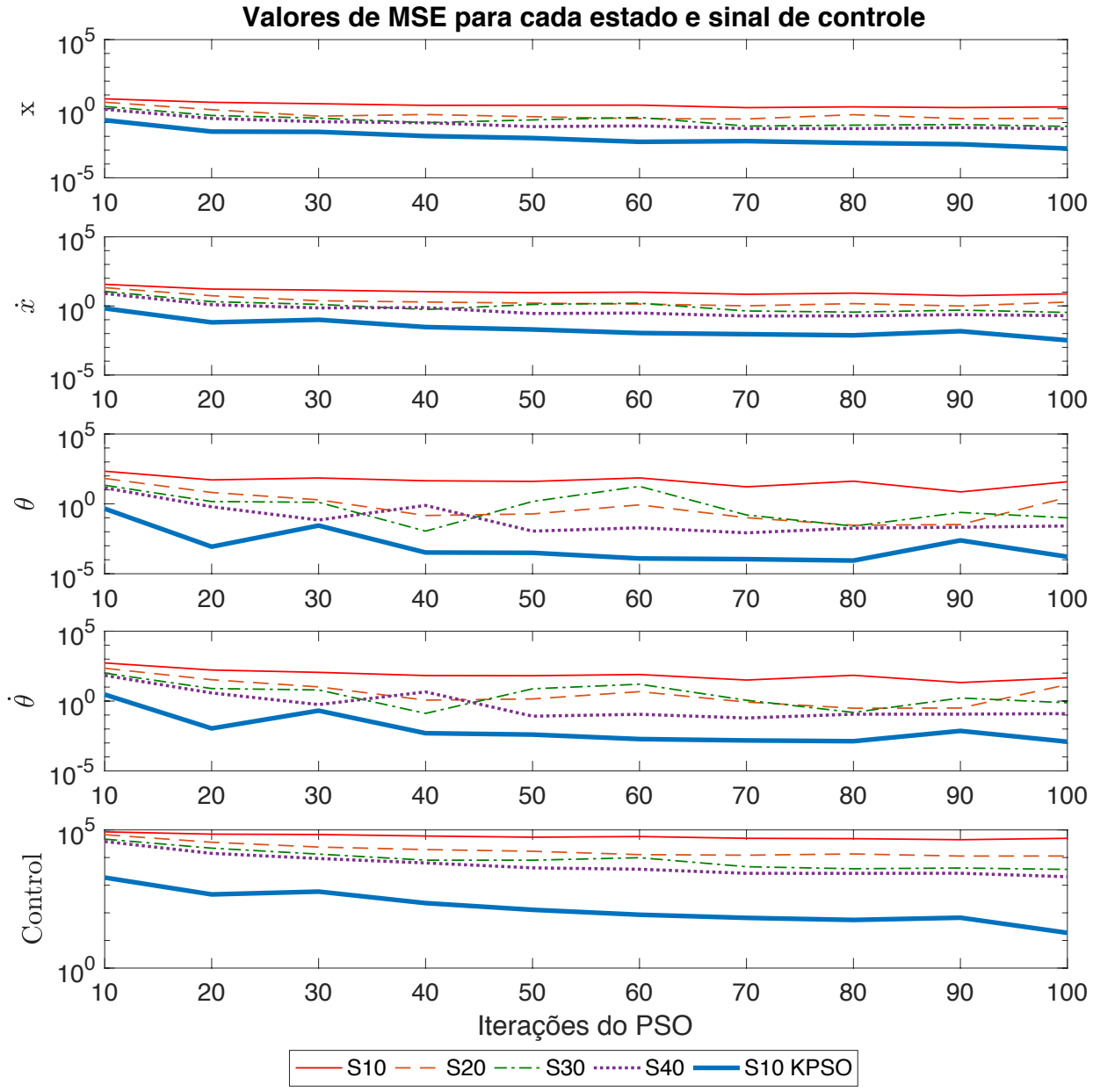
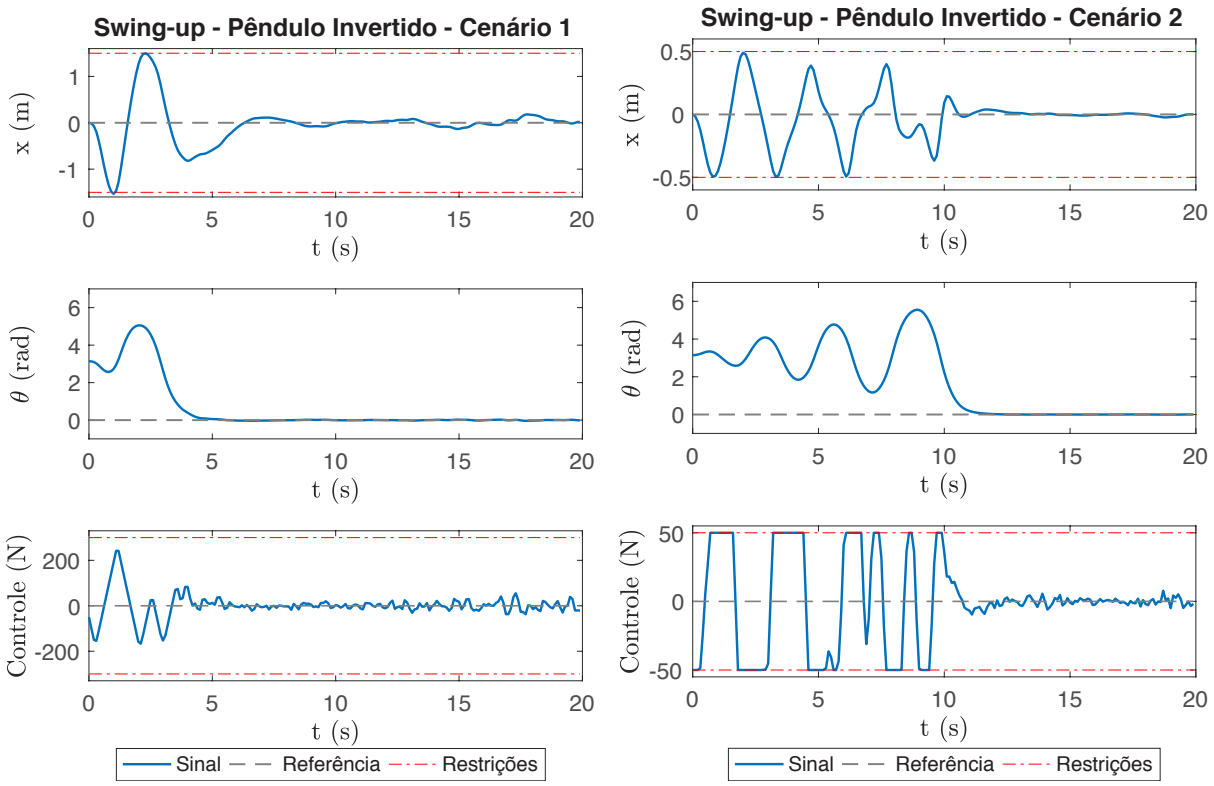



Figura 4.6: Comparação entre o número de partículas e o número de iterações do PSO.

A Figura 4.7 apresenta o desempenho do sistema com os pesos ajustados já utilizando-se da técnica do KPSO.

## 4.6 AJUSTES DOS COEFICIENTES COGNITIVO (C1) E SOCIAL (C2)

Uma característica que pode ser observada nas Figuras 4.7a e 4.7b é a oscilação no sinal de controle após o sistema atingir a referência. Isto pode ser explicado pela natureza estocástica do PSO no sentido de que numa partícula com dimensão igual a 20, pequenas variações nas posições




(a) (b)

Figura 4.7: Simulações com pesos ajustados e utilizando a técnica do KPSO (a) Cenário 1; (b) Cenário 2.

de cada dimensão não afetam significativamente o valor da função custo e que, para convergir ao valor desejado do sinal de controle no estado estacionário ($u_{ss}$) seria necessário aumentar o número de iterações. Em testes realizados, observou-se que este número deveria ser acima de 60 para minimizar as oscilações do sinal de controle, porém isto aumenta o tempo de cálculo do algoritmo.

A primeira abordagem utilizada para tratar este problema, sem ter que aumentar significativamente o número de iterações, foi ajustar os coeficientes cognitivo ($c_1$) e social ($c_2$) do PSO. Em grande parte das aplicações na literatura, observa-se o uso dos valores $c_1$ =2,0 e $c_2$ = 2,0, incluindo os trabalhos que aplicam o PSO ao MPC. Isto se deve ao fato de o trabalho inicial de Kennedy e Eberhart [97] sugerir um valor fixo igual a 2,0 para ambos os coeficientes. Trabalhos posteriores como os de Suganthan [102], Ratnaweera et al. [128] e Zhan et al. [104] sugerem que o ajuste de $c_1$ e $c_2$ para cada problema pode trazer melhor desempenho.

Com o intuito de ajustar os coeficientes, mais uma vez adotou-se a métrica MSE e foram realizados experimentos variando $c_1$, $c_2$ ao modo de um *Grid Search* e também o número de iterações do algoritmo ($MaxIter$) para verificar a influência de cada parâmetro no desempenho do algoritmo. Os melhores resultados foram encontrados quando se fixou o valor de $c_1 = 2.1$ e variou-se o valor de $c_2$ conforme apresentado na Figura 4.8.

Neste caso, é possível observar que a região próxima de $c_2 = 1{,}0$ oferece melhor desempenho e portanto, este foi o valor escolhido. Simulando novamente o sistema, com os coeficientes

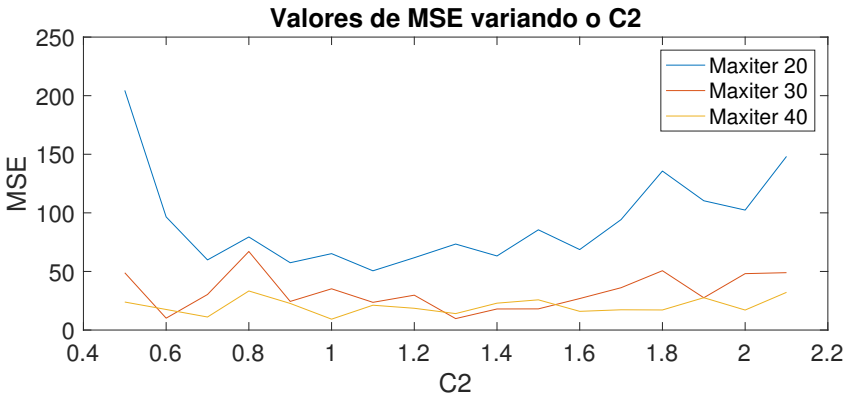
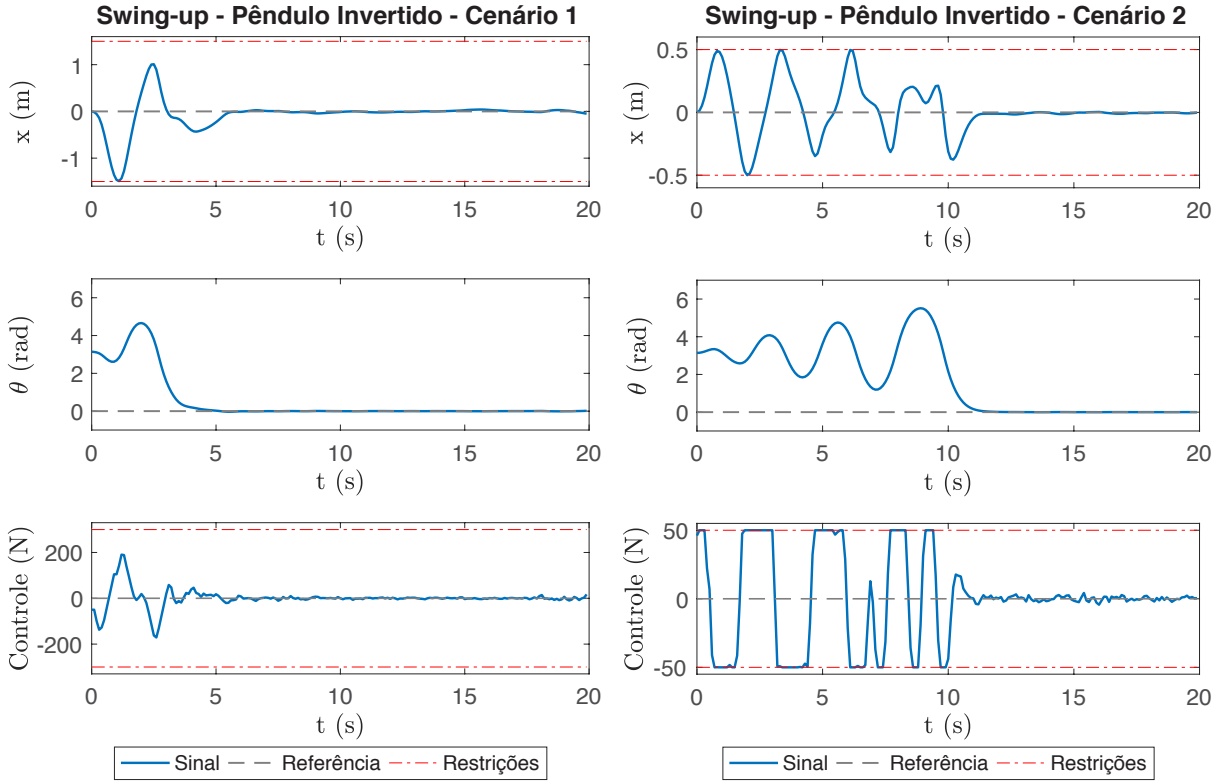



Figura 4.8: MSE da variação de C2.

ajustados, se obtém o desempenho mostrado na Figura 4.9. Nela se observa uma diminuição da oscilação do sinal de controle, porém ainda não se obteve o desempenho desejado já que trata-se de um sistema simulado sem ruídos na leitura dos estados e, neste caso, o sinal de controle deveria permanecer em zero após o sistema atingir a referência.




(a) (b)

Figura 4.9: Simulações com C1 e C2 ajustados (a) Cenário 1; (b) Cenário 2.

## 4.7 KPSO E PARTÍCULA DO ESTADO ESTACIONÁRIO

Para tratar o problema de oscilação do sinal de controle mesmo após o sistema atingir sua referência e ainda manter o número de iterações do algoritmo baixo, a segunda estratégia adotada foi a de inicializar uma das partículas com todas as dimensões iguais ao valor do sinal de controle desejado no estado estacionário ($u_{ss}$). Neste caso, ao atingir a referência, esta partícula naturalmente será avaliada como a melhor solução ($gbest$) e todas as demais irão convergir para sua posição. Caso o sistema esteja próximo mas ainda não tenha atingido a referência, esta será uma boa candidata à solução e portanto também servirá de guia para a solução final. Em todos os outros casos, ou seja, quando o sistema está longe da posição de referência, esta partícula será mal avaliada e acabará convergindo para outra posição, guiada por outra partícula que estará na condição de $gbest$.

Para demonstrar o desempenho desta abordagem, aqui denominada de KPSO+SS (KPSO mais a partícula no estado estacionário, *steady-state* $SS$), foram realizadas simulações do sistema utilizando as 3 abordagens: PSO regular, KPSO e KPSO+SS. Em todos os casos utilizou-se os pesos definidos na Seção 4.4, $S = 10$ e $MaxIter = 20$. Primeiro, simulou-se o Cenário 2 cujos resultados são apresentados na Figura 4.10 onde é possível se observar a grande diferença entre a abordagem regular do PSO em relação ao KPSO e o KPSO+SS que praticamente remove as oscilações no sinal de controle uma vez que o sistema tenha atingido a referência.

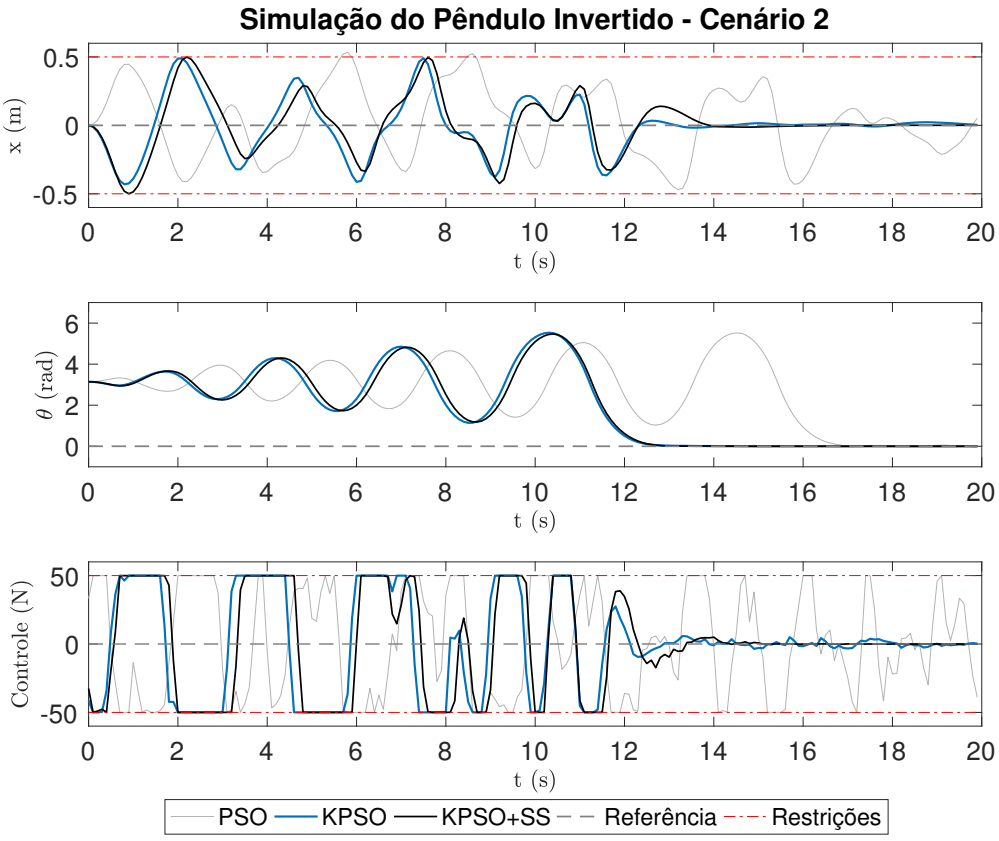



Figura 4.10: Comparação PSO regular, KPSO e KPSO+SS para $Ta = 100ms$ e $N = 20$.

Em seguida, criou-se um Cenário 3 a partir do Cenário 2, modificando o período de amostragem e consequentemente o horizonte de predição para $Ta = 20ms$ e $N = 100$. Neste caso, os resultados são apresentados na Figura 4.11 onde ficam ainda mais evidentes as diferenças entre o KPSO e o KPSO+SS.

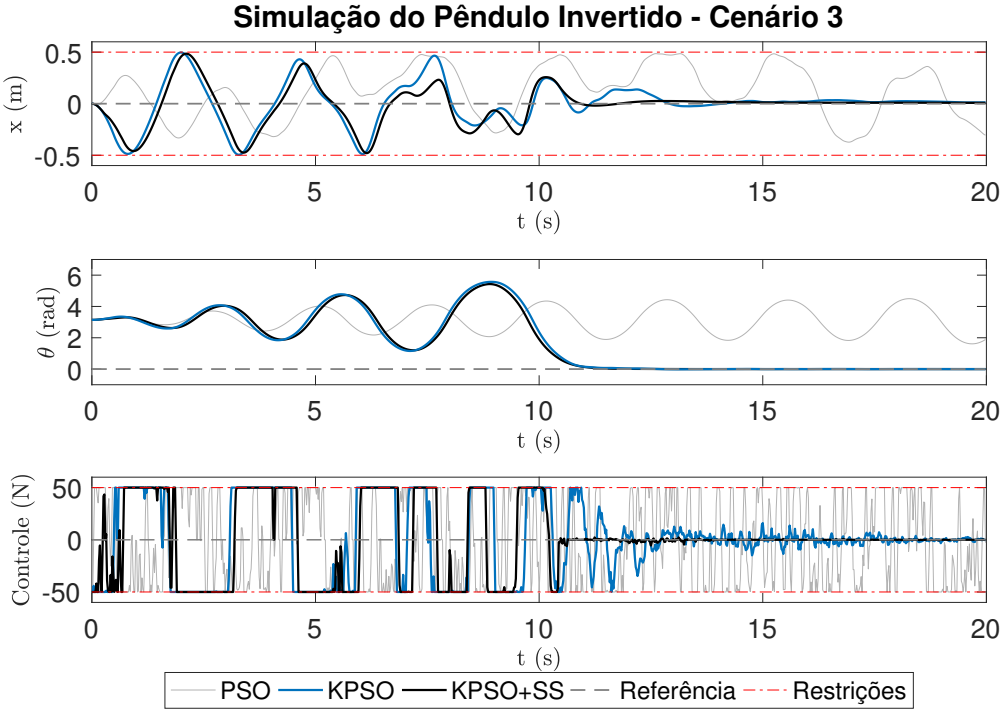



Figura 4.11: Comparação PSO regular, KPSO e KPSO+SS para $Ta = 20ms$ e $N = 100$.

## 4.8 ALGORITMO DO NMPC-KPSO+SS

Combinando a abordagem do KPSO+SS com o NMPC, é possível descrever o Algoritmo 2 executado a cada período de amostragem.

Neste caso, $\mathbf{x}(k)$ é o vetor do estado atual do sistema, $\mathbf{x}_{ref}(k, k+N)$ são os vetores de referência desde o instante atual $k$ até o final do horizonte de predição $(k+N)$ para cada estado do sistema, $\hat{u}_{1..S}$ são os $S$ vetores de controle estimados a cada iteração do algoritmo (neste caso as partículas do PSO), $\mathbf{u}_{opt}(k)$ é o vetor de controle ótimo estimado para o período de amostragem do instante $k$, $\mathbf{u}_{ss}$ é o vetor dos valores de $\mathbf{u}$ desejados em estado estacionário, $\mathbf{y}_{i..S}$ são os vetores com as melhores posições locais de cada partícula, $\mathbf{y}_s$ é o vetor da melhor posição do enxame. As demais constantes seguem a notação do algoritmo do PSO apresentado na Seção 2.5.1

**Algorithm 2:** Algoritmo NMPC-PSO+SS

Inicializa o sistema com a trajetória de referência $\mathbf{x}_{ref}(k, k+N)$;
Realiza-se a medição dos estados $\mathbf{x}(k)$;
Inicializa a partícula $\hat{\mathbf{u}}_1$ com o valor $\mathbf{u}_{opt}(k-1)$;
Inicializa a partícula $\hat{\mathbf{u}}_2$ com o valor de $\mathbf{u}_{ss}$ em todas as suas dimensões;
Inicializa o enxame de partículas $\hat{u}_{3..S}$ obedecendo às restrições de $\mathbf{u}_{min}$, $\mathbf{u}_{max}$ e $\Delta\mathbf{u}_{max}$;
Inicializa os valores de $\mathbf{y}$ (Melhor Local) de cada partícula e $\mathbf{y}_s$ (Melhor Global);
`/* Laço 1 (L1) - Principal */`
**for** $iter = 1 : MaxIter$ **do**
    `/* Laço 2 (L2) - Cálculo da Função Custo */`
    **for** $i = 1 : S$ **do**
        Estima $\hat{\mathbf{x}}_i(k, k+N)$ a partir de $\hat{\mathbf{u}}_i$; `/* Loop interno de Predição */`;
        Calcula o valor da função custo da partícula $\hat{\mathbf{u}}_i$ ;
        **if** $f(\hat{\boldsymbol{u}}_i) \leq f(\boldsymbol{y}_i)$ **then**
            $\mathbf{y}_i = \hat{\mathbf{u}}_i$;
    `/* Laço 3 (L3) - Calcula a Melhor Posição Global */`
    **for** $i = 1 : S$ **do**
        **if** $f(\boldsymbol{y}_i) \leq f(\hat{\boldsymbol{y}}_s)$ **then**
            $\mathbf{y}_s = \mathbf{y}_i$;
    `/* Laço 4 (L4) - Atualiza Partículas */`
    **for** $i = 1 : S$ **do**
        **for** $j = 1 : N$ **do**
            $v_{ij} = wv_{ij} + c_1U_1[0,1](y_{ij} - \hat{u}_{ij}) + c_2U_2[0,1](y_{sj} - \hat{u}_{ij})$;
            Aplicar restrições de $\Delta\mathbf{u}_{max}$;
            $\hat{u}_{ij} = \hat{u}_{ij} + v_{ij}$;
            Aplicar restrições de $\mathbf{u}_{min}$ e $\mathbf{u}_{max}$;
    Atualiza o valor da inércia $w$;
Aplicar o primeiro valor de $\mathbf{u}_{opt}(k)^{(1)}$ ao sistema;

## 4.9 DISCUSSÕES FINAIS DO CAPÍTULO E CONTRIBUIÇÕES

Neste capítulo foram apresentadas as modificações e ajustes necessários para adaptar o algoritmo do PSO como método de solução do problema de otimização não-linear (programação não-linear) do NMPC. Para isto foi necessário adicionar, inicialmente, um método para que a busca pela soluções obedecesse às restrições de variação do sinal de controle ($\Delta u_{max}$). Em seguida, funções de penalização foram utilizadas para que o algoritmo obedecesse às restrições dos estados do sistema que são dependentes das ações de controle.

Um método para realizar o ajuste dos pesos da função custo do NMPC, utilizando o próprio algoritmo PSO, foi apresentado. Sua métrica de avaliação foi baseada em MSE comparando os estados do sistema e o sinal de controle à referência desejada. Dois cenários do sistema do pêndulo invertido foram considerados nos experimentos. Para demonstrar a confiabilidade dos resultados, cada uma das soluções foi submetida a uma segunda rodada de testes com 1000 experimentos para

cada cenário, onde foi possível isolar somente aquelas que não apresentassem nenhum caso de instabilidade nas simulação. Este passo extra é importante, uma vez que algoritmos estocásticos são criticados na literatura por não garantirem soluções ótimas, factíveis ou mesmo estabilidade em malha fechada, conforme discutido por Khusainov et al. [129]. Além disso, este método de ajustes de pesos poderia ser facilmente modificado para incluir mais cenários conforme a necessidade.

Para reduzir o número de partículas e iterações necessárias dos algoritmo, duas técnicas foram empregadas. A primeira delas foi o KPSO, já discutido na literatura. A segunda, KPSO+SS, desenvolvida neste trabalho, contribui com a redução do número de iterações necessárias, principalmente quando o sistema já está próximo da referência desejada, diminuindo as oscilações do sinal de controle e adicionando estabilidade ao mesmo. Estas últimas técnicas foram complementadas com o ajustes dos coeficientes social e cognitivo do PSO, o que reforça trabalhos na literatura de que o ajuste destes parâmetros para cada problema pode melhorar seu desempenho.

Combinando todas as técnicas e ajustes aplicados, obteve-se um algoritmo (NMPC-KPSO+SS) com baixo número de partículas ($S = 10$) e iterações ($MaxIter = 20$) que é um bom candidato à atingir o desempenho de tempo-real exigida pelos requisitos de otimização *on-line* do NMPC.

# 5 DESENVOLVIMENTO DO CONTROLADOR NMPC-PSO EM HARDWARE RECONFIGURÁVEL

Com a solução do NMPC-KPSO+SS definida e testada em Matlab, partiu-se para a etapa seguinte de avaliação de seu desempenho em sistemas embarcados. Para isto, sendo o Matlab um ambiente de testes que geralmente não oferece o melhor desempenho em termos de processamento, criou-se uma versão do algoritmo otimizado em C++ para que fosse possível testá-lo em diferentes processadores embarcados. Foram selecionados 4 processadores para comparação listados a seguir:

1. Intel® Core i7 Quad Core (7820HQ) - 2.9GHz (3.9 GHz Turbo Boost) (MacBook Pro)
2. ARM® Cortex™ A53 - Quad Core 1.2GHz Broadcom BCM2837 (Raspberry Pi 3)
3. ARM® Cortex™-A8 AM335x 1GHz (BeagleBone Black)
4. ARM® Cortex™-A9 MPCore™ 925 MHz (Arrow SoCKit FPGA)

O processador Intel, apesar de ser um processador de alta potência e consumo de energia (o modelo em questão utiliza aproximadamente 45W) foi adicionado à lista somente a título de referência. Em seguida temos três variações do ARM. Os items 2 e 3 se referem as placas bastante populares para aplicações embarcadas, o Raspberry Pi 3 e a BeagleBone Black. Por último, temos o processador ARM embarcado no SoC FPGA utilizado neste trabalho.

Os testes de desempenho foram realizados para o sistema do pêndulo invertido utilizando o Cenário 2 (N = 20, Ta = 100 ms) e o Cenário 3 (N = 100, Ta = 20 ms) que diferem entre si no tempo de amostragem e no tamanho do horizonte de predição. A Tabela 5.1 mostra o resultado das simulações para cada processador, por ordem de desempenho, comparando o tempo médio e máximo para o cálculo da solução a cada período de amostragem, lembrando que o PSO usado tem 10 partículas, executa 20 iterações e está utilizando a estratégia KPSO+SS.

Tabela 5.1: Comparação de desempenho do PSO entre os processadores.

| | | Cenário 2 ($Ta = 100ms$) | | Cenário 3 ($Ta = 20ms$) | |
|---|---|---|---|---|---|
| | Clock | Tempo Médio (ms) | Tempo Máximo (ms) | Tempo Médio (ms) | Tempo Máximo (ms) |
| Intel Core i7 | 2,9 a 3,9 GHz | 0,74 | **1,52** | 3,28 | **5,62** |
| ARM - SoCKit | 925 MHz | 10,01 | **12,63** | 50,92 | 53,32 |
| ARM - Raspberry Pi 3 | 1,2 GHz | 23,73 | **26,05** | 119,16 | 131,27 |
| ARM - BeagleBone Black | 1 GHz | 31,90 | 110,04 | 159,51 | 550,18 |
| Intel Core i7 (PSO regular) | 2,9 a 3,9 GHz | 87,00 | 147,47 | 484,25 | 699,62 |

Observando os resultados, é possível notar uma enorme discrepância entre o desempenho de cada processador. Para realizar o controle do sistema em tempo real, é necessário que o tempo

máximo de cálculo esteja abaixo do tempo de amostragem do sistema ($Ta$). Neste caso, destacou-se em negrito as configurações que atendem a este requisito. Como esperado, o Intel Core i7, de alta potência, é capaz de controlar o sistema em ambos os cenários. Porém, dos processadores embarcados ARM, somente os presentes na SoCKit e no Raspberry Pi 3 são capazes de garantir desempenho para o Cenário 2 e nenhum deles foi capaz de atuar no Cenário 3.

A título de comparação, para evidenciar o ganho de otimização do algoritmo desenvolvido no Capítulo 4, foi inserido na última linha da tabela a simulação de um cenário de configuração do PSO regular que pudesse alcançar um desempenho similar ao que o KPSO+SS atinge. Neste caso, a configuração do PSO regular necessita de $S = 40$ e $MaxIter = 1000$. Mesmo assim, o desempenho de controle ainda fica abaixo da solução otimizada com o KPSO+SS, principalmente no Cenário 3. Neste caso, se observa que nem o processador de alto desempenho com alto consumo de energia é capaz de satisfazer os requisitos de tempo-real para nenhum dos cenários.

## 5.1 *PROFILING* DO ALGORITMO

Com o intuito de desenvolver a solução em hardware, a primeira medida adotada foi a de realizar um *profiling* no algoritmo, ou seja, mapear o custo computacional de cada uma das funções para identificar se é possível utilizar uma abordagem de co-projeto de hardware (*co-design*). Em caso afirmativo, somente as funções críticas são aceleradas em hardware e o restante do algoritmo fica em software, tonando o processo de desenvolvimento mais rápido.

O *profiling* foi realizado utilizando a ferramenta GNU *profiler* (gprof) no software em C++ rodando na placa de desenvolvimento escolhida para o trabalho, a Arrow SocKit FPGA (com o FPGA Cyclone V da Altera). O resultado do *profiling* é apresentado na Tabela 5.2 onde é possível observar que a maior parte do custo computacional está no cálculo da função custo e em suas funções internas (Predição e Equações Diferenciais do sistema) somando 88,26% do tempo.

Tabela 5.2: *Profiling* do algoritmo NMPC-PSO.

| Função | Esforço Compitacional (%) |
|---|---|
| Inicialização de Partículas | 0.26 |
| Laços do PSO | 8.6 |
| Função Custo | 24.12 |
| ⊢ Predição (RK4) | 28.42 |
| ⊢ Equações Diferenciais (Modelo do Sistema) | 35.72 |
| Outros | 2.88 |

Considerando o Cenário 3 e o tempo reportado na Tabela 5.1 é possível calcular que este percentual representa 40,62 ms do tempo total de 46,02. Neste cenário, caso seja possível acelerar consideravelmente em hardware o cálculo da função custo, é possível atingir os requisitos de tempo real somente implementando esta função em hardware.

## 5.2 ESTRATÉGIA DE PARALELISMO

Uma das qualidades do algoritmo PSO e que o torna bastante adequado para implementação em hardware é sua natureza paralela. Conforme descrito no Algoritmo 2, o PSO é composto por um laço externo (L1), o qual representa cada iteração do algoritmo, e uma sequência de laços internos, sendo eles: o laço 2 (L2) para o cálculo da função custo de cada partícula justamente com o cálculo da melhor posição local; o laço 3 (L3) que calcula a melhor posição global e; o laço 4 (L4) responsável por atualizar a velocidade e posição de cada partícula (este é um laço duplo que percorre tanto as partículas $[1..S]$ quanto as dimensões internas de cada partícula $[1..N]$.

Cada iteração do L1 depende dos resultados da iteração anterior e portanto este não é um candidato a ser paralelizado. Já no L2, o cálculo da função custo de cada partícula bem como a atualização de sua melhor posição local são independentes umas das outras, sendo um bom candidato ao paralelismo. No L3, compara-se a melhor posição local de cada partícula umas com as outras até encontrar a melhor posição global. Este é um procedimento que pode ser acelerado a partir de um topologia em árvore onde se inicia comparando partículas duas a duas, em seguida os resultados das primeiras comparações dois a dois até encontrar o melhor valor global. Finalmente, no L4, que é composto por dois níveis de laço, a atualização da velocidade e posição de cada partícula é independente uma da outra, porém, como é necessário obedecer aos critérios de variação do sinal de controle ($\Delta\mathbf{u}_{max}$), cada dimensão da partícula deve ser atualizada em sequência pois o valor de $\hat{\mathbf{u}}_{i,j}$ depende de $\hat{\mathbf{u}}_{i,j-1}$. Portanto, há dependência de dados entre as dimensões da mesma partícula e a única possibilidade de paralelismo se encontra em atualizar cada uma das partículas em paralelo.

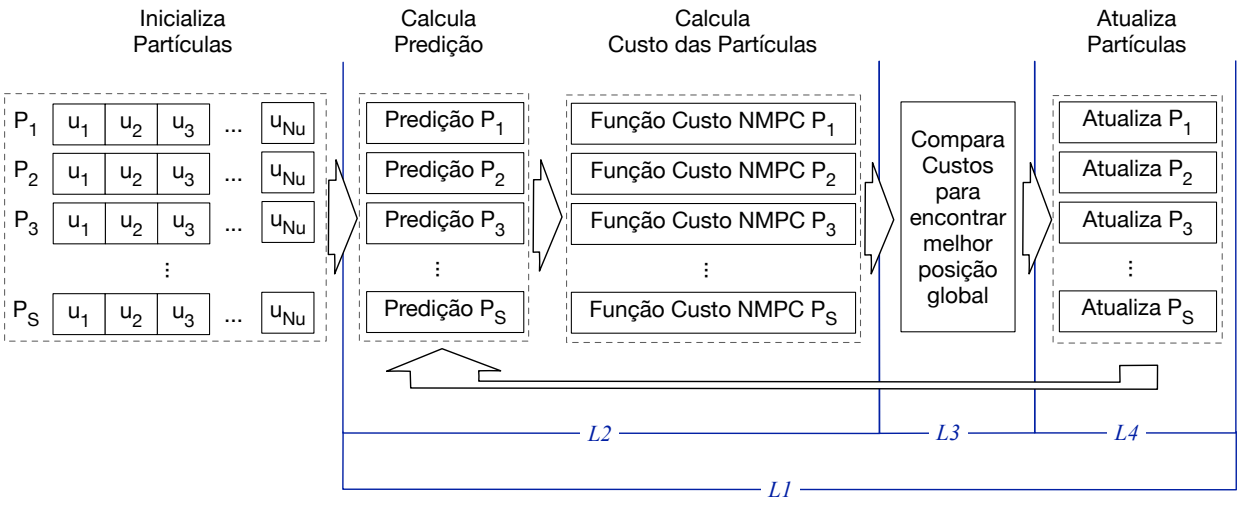



Figura 5.1: Diagrama das etapas do PSO com o NMPC.

A Figura 5.1 apresenta as etapas do algoritmo onde é possível observar que a etapa inicial de inicialização das partículas pode ser paralelizada havendo somente dependência entre as dimensões das partículas (Ex: $u_2$ depende de $u_1$, etc). O laço 1 (L1) é executado sequencialmente, ou

seja, calcula-se as etapas de predição, custo, comparação das melhores posições e atualização de partículas para uma iteração e em seguida, estes resultados são re-alimentados para a próxima iteração do L1. No L2 que compreende as etapas de predição juntamente com a avaliação do custo, cada partícula é avaliada em paralelo. O L3 que resulta na melhor posição global é executada em um só bloco onde uma árvore de comparações dois a dois é criada. Finalmente, o L4 atualiza as partículas em paralelo, havendo somente a dependência entre as dimensões como na etapa inicial.

Além dos laços explícitos do PSO, há ainda um outro nível de paralelismo a ser explorado referente ao cálculo da função custo do PSO, composta pela etapa de predição seguida da função custo do NMPC. Neste caso, é possível observar que a etapa de predição é, em si, um laço executado ao longo do horizonte de predição ($i = 1..N$) onde o valor de $\hat{\mathbf{u}}_i$ é utilizado juntamente com o valor de $\mathbf{x}(k)$ para realizar a predição de $\hat{\mathbf{x}}(k+1)$. Após a etapa de predição, um segundo laço é executado, também no intervalo $[1..N]$, para calcular o custo NMPC baseado nos valores estimados $\hat{\mathbf{u}}_{k..k+N}$, nas referências futuras $\mathbf{x}_{ref_{k..k+N}}$, no valor de $\mathbf{x}_{ss}$ e no valor de $\mathbf{u}_{ss}$. Portanto, uma primeira abordagem é a de realizar o cálculo dos dois laços em sequência, o que é mostrado na parte superior da Figura 5.2 (Estratégia S1.1). Um segunda possibilidade se baseia no fato do cálculo de cada iteração da função custo só depender das estimação do estado daquela iteração. Para melhor visualizar isto, re-escrevemos a função custo na Equação (5.1).

$$J(\mathbf{x}(k),\mathbf{u}(k)) \triangleq \underbrace{\sum_{i=1}^{N-1} ||\hat{\mathbf{x}}(k+i) - \mathbf{x}_{ref}(k+i)||_Q^2}_{\text{Custo dos Estados}} + \underbrace{||\hat{\mathbf{x}}(N) - \mathbf{x}_{ss}||_{Qf}^2}_{\text{Custo do Estado Final}} + \underbrace{\sum_{i=0}^{Nc-1} ||\hat{\mathbf{u}}(k+i) - \mathbf{u}_{ss}||_R^2}_{\text{Custo da Ação de Controle}} \tag{5.1}$$

É possível observar que o primeiro termo da equação (Custo dos Estados - CE) depende somente da predição de $\hat{\mathbf{x}}(k+i)$ e de $\mathbf{x}_{ref}(k+i)$, já conhecido. O último termo (Custo da Ação de Controle - CAC) é calculado a partir do valor atual da partícula ($\hat{\mathbf{u}}(k)$), juntamente com $\mathbf{u}_{ss}$, também já conhecido. Neste caso, uma estratégia mais otimizada é a de utilizar um *pipeline* aonde a primeira etapa de predição é calculada $\hat{\mathbf{x}}(k+1)$ e, em seguida calcula-se o custo dos estados e da ação de controle já em paralelo à estimativa da próxima etapa de predição, $\hat{\mathbf{x}}(k+2)$. A única parte da Equação (5.1) que não pode ser paralelizado é o segundo termo (Custo do Estado Final - CEF) que depende da última etapa de predição. Além de reduzir o tempo de cálculo, esta estratégia tem um segundo benefício em termos de implementação em hardware que é a economia de memória decorrente do fato de que somente a estimativa do estado da iteração atual $\hat{\mathbf{x}}_i$ precisa ser armazenada em memória para o cálculo de seu custo. Esta segunda estratégia (S1.2) é apresentada na parte inferior da Figura 5.2.

Uma segunda estratégia de *pipeline* ainda mais otimizada pode ser explorada a partir do fato da função custo estar dividida em $N$ etapas. Neste caso, se observa que a entrada para cada a etapa de predição é a dimensão $j$ da partícula $i$, ou seja $\hat{u}_{i,j}$. Invertendo a ordem dos laços internos de L4 (Linhas 16 e 17 do Algoritmo 2), fazendo com que a cada iteração uma das dimensões seja atualizada para todas as partículas, é possível criar uma terceira camada de *pipeline* além das

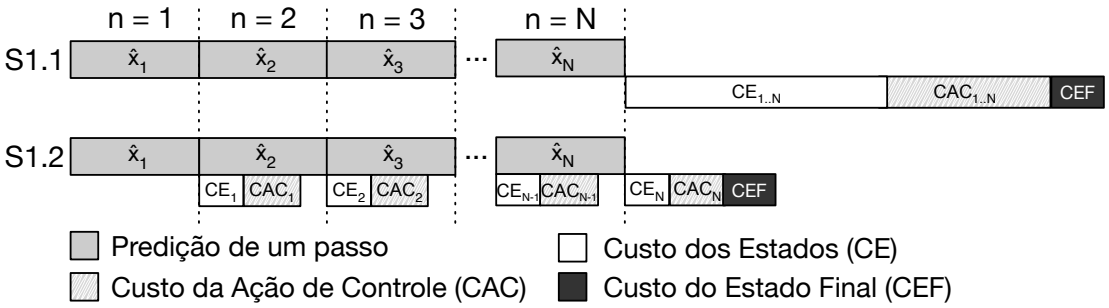



Figura 5.2: Função Custo - Estratégia de *pipeline*.

duas já existentes na função custo. A Figura 5.3 apresenta, na parte superior, o modo sequencial deste algoritmo (Estratégia S2.1) onde todas as dimensões de todas as partículas são atualizadas em primeiro lugar seguidas das $S$ funções custo executadas em paralelo. Em seguida, na parte inferior, é mostrada a estratégia com o *pipeline* (Estratégia S2.2) onde a atualização das partículas é dividida em $N$ etapas e executada em paralelo às $S$ funções custo.

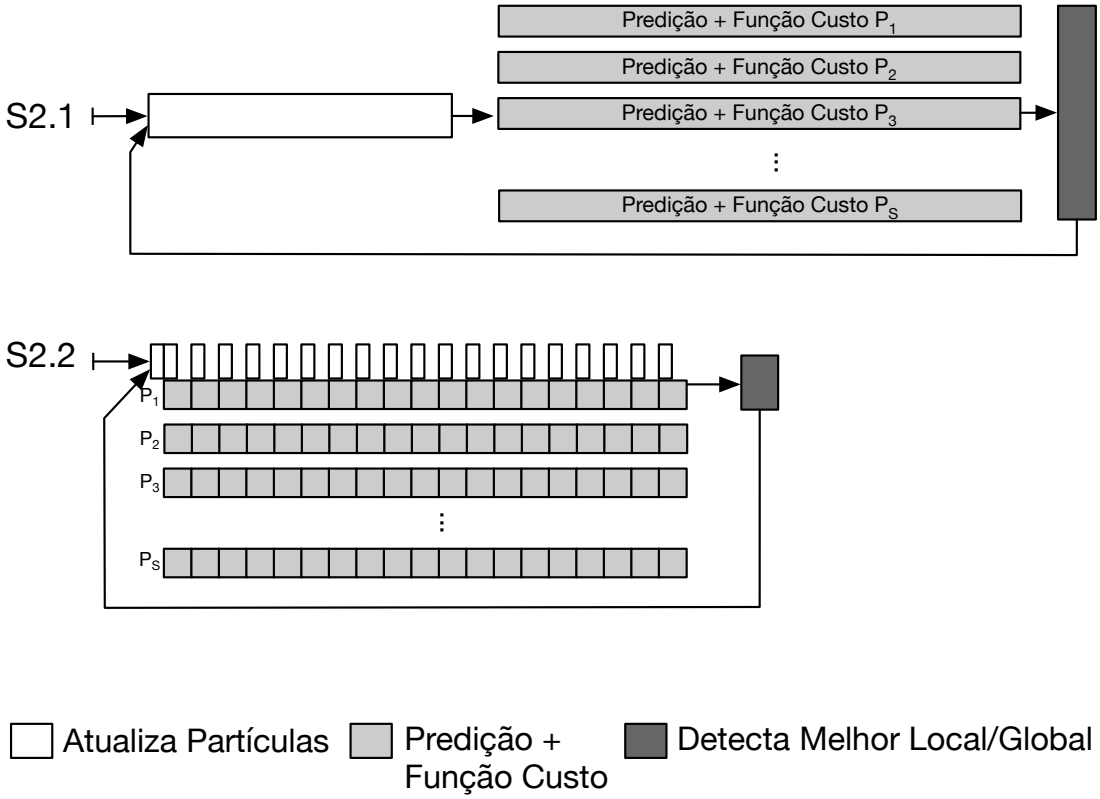



Figura 5.3: PSO - Estratégia de *pipeline*.

Em termos matemáticos, é possível formalizar as diferenças no tempo de cálculo de ambas as estratégias (sequencial e *pipeline*) a partir de equações. Para a estratégia sequencial (S1.1 + S2.1) temos:

$$tFC_1 = tPred \times N + (tCE \times Ns + tCAC \times Nu) \times N + tCEF, \tag{5.2}$$

$$tPSOiter_1 = tAP \times S + tFC_1 + tML + tMG \times S, \tag{5.3}$$

e para a estratégia em *pipeline*, temos:

$$tFC_2 = tPred \times N + (tCE \times Ns + tCAC \times Nu) + tCEF, \tag{5.4}$$

$$tPSOiter_2 = tAP + tFC_2 + tML + tMG \times S, \tag{5.5}$$

onde,

- N = horizonte de predição;
- Nc = horizonte de controle;
- Ns = número de estados do sistema;
- Nu = número de ações de controle do sistema (atuadores);
- S = número de partículas do PSO;
- tPred = tempo de cálculo da predição de um passo à frente (um período de amostragem);
- tCE = tempo para cálculo do custo de um estado;
- tCAC = tempo para cálculo da ação de controle;
- tCEF = tempo para cálculo do custo final;
- tML = tempo para cálculo do melhor local;
- tMG = tempo para cálculo do melhor global;
- tAP = tempo para cálculo da atualização de uma partícula;
- tFC = tempo para cálculo da função custo;
- tPSOiter = tempo para cálculo de cada iteração do PSO.

Como será apresentado mais à frente na seção de resultados, para o caso do pêndulo invertido, a estratégia de *pipeline* foi capaz de reduzir em 45% o tempo de processamento.

## 5.3 IMPLEMENTAÇÃO DOS MÓDULOS EM HARDWARE

Uma das características do algoritmo MPC é a sua capacidade de generalização para vários sistemas. Neste caso, basta trocar o modelo do sistema e ajustar suas constantes ($N$, $Nc$, $Ns$, $Nu$, $Ta$, restrições e pesos) que a mesma solução é capaz de controlar outros sistemas. A implementação do hardware procurou seguir esta mesma premissa, sendo genérica o suficiente ao ponto de ter todas as constantes ajustáveis. Um outro requisito arquitetural estabelecido foi o uso de aritmética de ponto-flutuante em todos os módulos do sistema pois, apesar de tornar o desenvolvimento mais complexo, provê maior precisão para ser aplicada a uma grande variedade de sistemas com diferentes demandas, incluindo aqueles com um horizonte de predição longo.

O desenvolvimento dos módulos em hardware foi realizado a partir de uma descrição do tipo *Register Transfer Level* (RTL) utilizando a linguagem de descrição de hardware VHDL. Todas as bibliotecas ponto-flutuante (FP), aritméticas e trigonométricas, são baseadas no padrão IEEE 754 [119] e foram desenvolvidas por Muñoz et al. e apresentadas em [120] e [121]. Neste trabalho, estão sendo utilizadas especificamente as bibliotecas de adição/subtração (*FPadd*), multiplicação (*FPmul*), divisão (*FPdiv*), o cálculo de senos e cossenos (*CORDICsincos*) e exponencial (*CORDICexp*).

### 5.3.1 Otimizações no módulo do *CORDICSinCos*

Antes de iniciar o desenvolvimento do sistema, observou-se que um dos módulos trigonométricos utilizados com frequência em sistemas não lineares é o cálculo de senos e cosenos. Conforme já mencionado, este cálculo é realizado a partir do módulo *CORDICSinCos* desenvolvido por Muñoz [130].

Com o intuito de acelerar os cálculos das equações diferenciais, observou-se o fato das unidades *FPadd* e *FPmul* realizarem suas respectivas operações em dois ciclos de *clock*, porém utilizam sinais de *start* e *ready* para sincronizar além registradores intermediários entre cada operação. Isto faz com que operações sequenciadas tenham uma latência total de três ciclos ao invés de dois.

Para otimizar o tempo de cálculo, foi realizada uma primeira modificação nas três unidades *FPadd*, utilizadas em sequência na *unidade de microrrotação* do CORDIC. A modificação consiste em adicionar um sinal de *ready_burst* que é acionado um ciclo antes do valor final estar disponível na saída do *FPadd*. Neste caso, é possível reduzir em um ciclo de *clock* as 25 operações em sequência realizadas pela *unidade de microrrotação*. Em seguida, foram adicionados ao módulo CORDIC multiplexadores para direcionar os resultados das somas/subtrações diretamente para os módulos seguintes sem passar pela etapa intermediária de armazenar os valores em registradores.

Após as melhorias, o novo módulo foi testado e sintetizado utilizando a ferramenta Quartus Prime 17.1 (Intel) para o FPGA Cyclone V da placa SoCKit . O resultado pode ser observado na Tabela 5.3 onde se observa redução no uso de elementos lógicos (ALMs) e registradores (REGs) bem como a redução em 27% dos ciclos de *clock*. A única característica que piorou foi a frequência máxima possível neste circuito. Porém, como o restante da arquitetura está funcionando a 100MHz não há impacto.

Tabela 5.3: Resultados de síntese em FPGA das unidades CORDICSinCos.

| Módulo (FP Precision) | ALMs (41.910) | REGs | Mult. (112) | Max. Freq. (MHz) | Ciclos |
|---|---|---|---|---|---|
| CORDICSinCos | 1.396 | 507 | 1 | 124,92 | 85 |
| CORDICSinCos Otimizado | 1.287 | 432 | 1 | 101,2 | 62 |

### 5.3.2 Função Custo do NMPC-PSO

O primeiro módulo em hardware a ser desenvolvido, conforme o resultado do *profiling*, é a função custo do NMPC-PSO. Este módulo é responsável por implementar a Equação (5.1). Internamente é composto pelo submódulo que realiza a predição de um passo (*One Step Prediction*), uma unidade *FPadd*, uma unidade *FPmul* e uma Máquina de Estados Finitos (FSM - *Finite State Machine*) responsável por controlar e sincronizar todas as etapas de cálculo.

Conforme descrito na Seção 5.2, para otimizar o algoritmo, utilizou-se a estratégia S1.2 onde a função custo é calculada incrementalmente junto a cada passo de predição ao longo do horizonte $N$. Neste caso, três registrados $J_{CE}$, $J_{CAC}$ e $J_{CEF}$ são usados para armazenar os valores intermediários de cada um dos 3 termos da Equação 5.1.

A Figura 5.4 apresenta um diagrama detalhando o funcionamento interno do módulo da *Função Custo*. No diagrama é possível observar os vetores de entrada ($\mathbf{x}_k$, $\mathbf{u}_k$, $\mathbf{x}_{ref(k+1)}$, $\mathbf{x}_{ss}$) e o resultado de saída $J$. O sinal de comando *start* que indicam o início dos cálculos do módulo. O sinal *done_step* indica o fim do cálculo para um período de amostragem. Neste momento o módulo fica no aguardo do sinal externo *next_step* indicando que as variáveis $\mathbf{x}_k$, $\mathbf{u}_k$, $\mathbf{x}_{ref(k+1)}$ foram atualizadas para $\mathbf{x}(k+1)$, $\mathbf{u}(k+1)$, $\mathbf{x}_{ref(k+2)}$, respectivamente, e o cálculo pode prosseguir. Ao atingir o final do horizonte de predição, o valor de $J$ é atualizado juntamente com o sinal *ready*, indicando o fim do processamento.

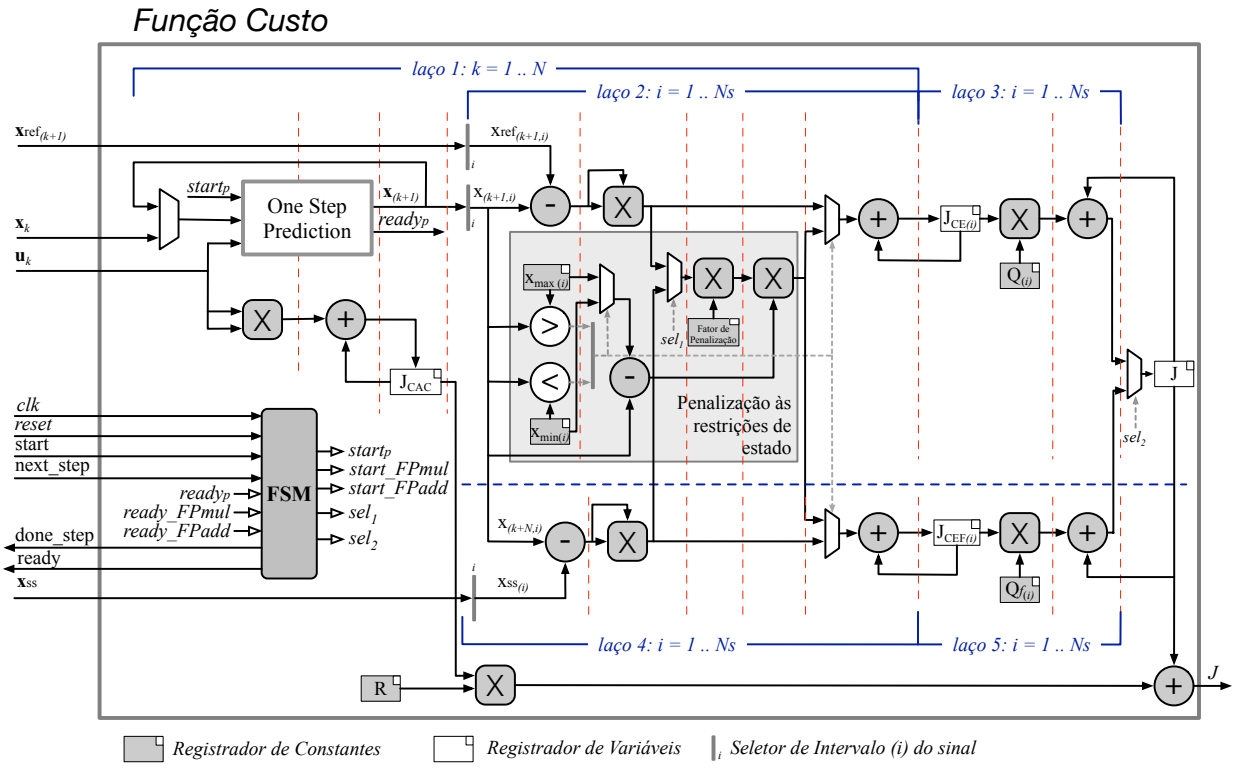



Figura 5.4: Diagrama detalhado da função custo.

No diagrama, é possível ainda observar a existência de cinco laços. Os Laços 1 e 2 são responsáveis pelo cálculo de $J_{CE}$, $J_{CAC}$ com a abordagem do *pipeline*. O registrador $J_{CAC}$ é um valor escalar, atualizado a cada ciclo do Laço 1, enquanto o registrador $J_{CE}$ é um vetor de $Ns$ registradores (número de estados do sistema) onde o custo de cada estado é atualizado no Laço 2. Em seguida, o Laço 3 é responsável por ponderar o valor do custo de cada estado do vetor $J_{CE}$ e somá-los ao valor final de $J$. A etapa posterior é composta pelo cálculo do vetor $J_{CEF}$, também composto por $Ns$ no Laço 4 e sua respectiva ponderação realizada no Laço 5 e soma ao valor de $J$. Finalmente, o valor de $J_{CAC}$ é ponderado pelo registrador $R$ e tudo é somado na saída $J$.

Ainda em relação ao diagrama, é necessário ressaltar que: os sinais de *clk* e *reset* foram omitidos para todos os módulos internos do diagrama para facilitar a leitura; as linhas pontilhadas verticais (em vermelho) apontam os estados da FSM ao controlar os procedimentos de cálculo; a cada início e fim de laço há um estado que verifica e incrementa a variável do laço; a seção do diagrama abaixo da linha pontilhada azul, representa etapas de cálculo executadas posteriormente às da parte superior mas foram posicionadas desta maneira para mostrar o uso do módulo de *Penalização às restrições de estado* que é re-aproveitado. Finalmente, em função da estratégia de *pipeline*, a maior parte do tempo de cálculo ocorre no submódulo *One Step Prediction* e, por este motivo, todos os cálculos dos custos puderam ser implementados de modo sequencial utilizando apenas uma unidade *FPmul* e uma unidade *FPadd*, economizando recursos de hardware.

#### 5.3.2.1 Predição de um passo à frente

O módulo de *Predição de uma passo à frente* (*One Step Prediction*) é responsável por calcular a predição de um período de amostragem ($Ta$) baseado nas equações diferenciais do modelo do sistema. Seu funcionamento é baseado no método numérico de Runge=Kutta de 4ª Ordem (Equação (2.25)) que discretiza o modelo dinâmico do sistema baseado em $Ta$. Sua entrada são os vetores de estado $\mathbf{x}_k$ e de controle $\mathbf{u}_k$ e sua saída é o vetor de estados um período de amostragem à frente $\mathbf{x}_{k+1}$. O módulo é composto pelo submódulo do *Modelo do Sistema*, por uma FSM que controla o processo, uma unidade *FPmul* e uma unidade *FPadd*.

Os detalhes de funcionamento do módulo são explicitados na Figura 5.5 onde é possível observar as etapas de cálculo. O Laço 1 executa as 4 etapas de predição do algoritmo que realiza em sequência a solução das equações diferenciais (*Módulo do Sistema*) e a etapa de atualizações dos coeficientes para a próxima iteração (Laço 2). O resultados de cada etapa são armazenados nos registrados $Z_{1..4}$ e utilizados no Laço 3 para calcular o valor dos estados de saída $\mathbf{x}_{k+1}$. Por ser um processo sequencial, as unidades de *FP* foram ainda compartilhadas com o *Módulo do Sistema* para reduzir o uso de hardware.

#### 5.3.2.2 Modelo do sistema - Pêndulo Invertido

Para implementar o módulo com as equações diferenciais do sistema (Equações (3.5) e (3.6)), foi necessário, inicialmente escrevê-las de modo a isolar todos os termos de $\dot{\mathbf{x}} = [\dot{x}\ \ddot{x}\ \dot{\theta}\ \ddot{\theta}]^T$

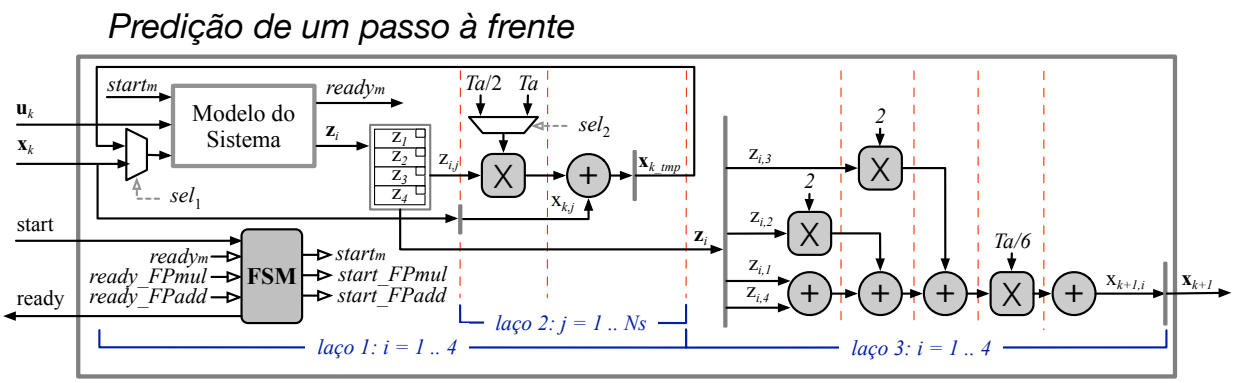



Figura 5.5: Diagrama detalhado do módulo de predição de um passo à frente

em função de $\mathbf{x} = [x\ \dot{x}\ \theta\ \dot{\theta}]^T$ e $\mathbf{u}$, colocando no formato de espaço de estados $\dot{\mathbf{x}} = A\mathbf{x} + B\mathbf{u}$. Neste caso, se observa que a Equação (3.5) está descrita em função da Equação (3.6) e portanto, é necessário substituir uma na outra para se obter os valores de $\ddot{x}$ e $\ddot{\theta}$ de maneira independente. Assim temos:

$$\ddot{\theta} = \left[u - b\dot{x} - ml\sin\theta\dot{\theta}^2 - \frac{M+m}{\cos\theta}(g\sin\theta - \frac{h\dot{\theta}}{ml})\right] \frac{1}{\left(ml\cos\theta - \frac{4l(M+m)}{3\cos\theta}\right)} \tag{5.6}$$

$$\ddot{x} = \frac{1}{M+m}[u - b\dot{x} - ml\cos\theta\ddot{\theta} + ml\sin\theta\dot{\theta}^2] \tag{5.7}$$

Com o intuito de simplificar os cálculos em hardware, observa-se os diversos termos repetidos nas equações, bem como divisões que podem ser pré-calculadas. Desse modo, foram definidas as seguintes constantes:

$$ml = m \times l;\ Mm = M + m;\ Mm\_1 = 1/(M+m);\ h\_ml = h/(m \times l);\ l43 = 4l/3$$

Além disso, termos repetidos como $ml\sin\theta$, $ml\cos\theta$ e $\dot{\theta}^2$ são calculados uma única vez, armazenados em registradores e utilizados ao longo das equações.

A Figura 5.6 apresenta o diagrama de fluxo de dados do módulo, representando as entradas (em azul), as saídas (em verde) além dos registradores intermediários. As operações estão em sequência na ordem dos estados ($s0$ a $s14$).

Mais uma vez, para otimizar o uso de recursos de hardware, foram observados os operadores que gastam mais ciclos de *clock* como o cálculo de senos, cossenos e a divisão para que fossem escalonados de modo a paralelizar os módulos com o mínimo de recursos. No total, observou-se que seriam necessários um módulo *FPmul*, um módulo *FPadd* além do módulo de divisão (*FPdiv*) e o módulo CORDICSinCos que é capaz de calcular o valor do seno e do coseno do mesmo ângulo simultaneamente. Como este último módulo já é composto internamente por 3 unidades *FPadd* e uma unidade *FPmul* estas foram compartilhadas tanto com este módulo (*Modelo do Sistema*) quanto com o módulo um nível acima (*One Step Prediction*).

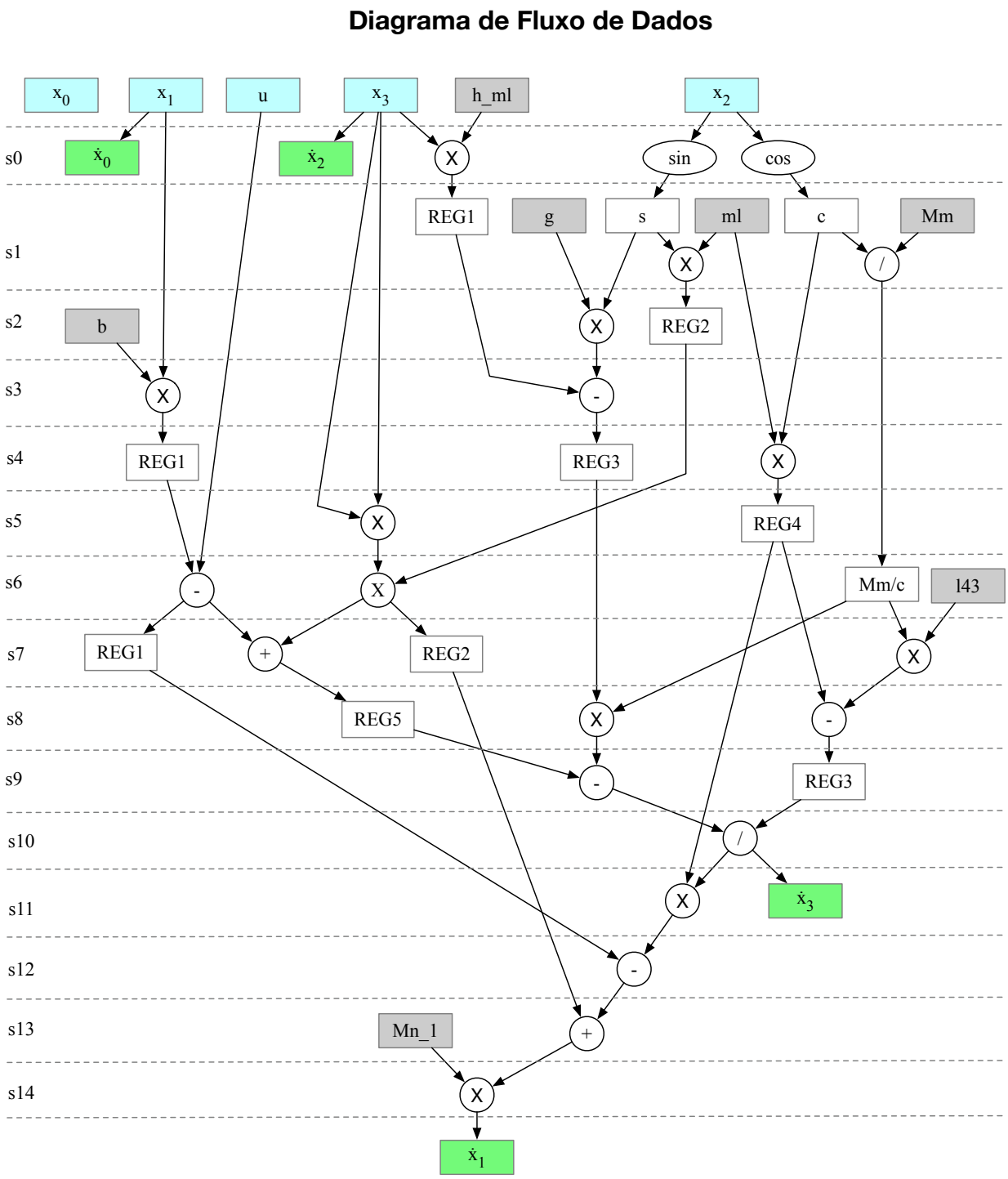



Figura 5.6: Diagrama de fluxo de dados do modelo do sistema - Pêndulo Invertido.

Finalmente, a Figura 5.7 apresenta uma visão geral do módulo da *Função Custo* juntamente com seus módulos internos onde é possível observar o compartilhamento de recursos.

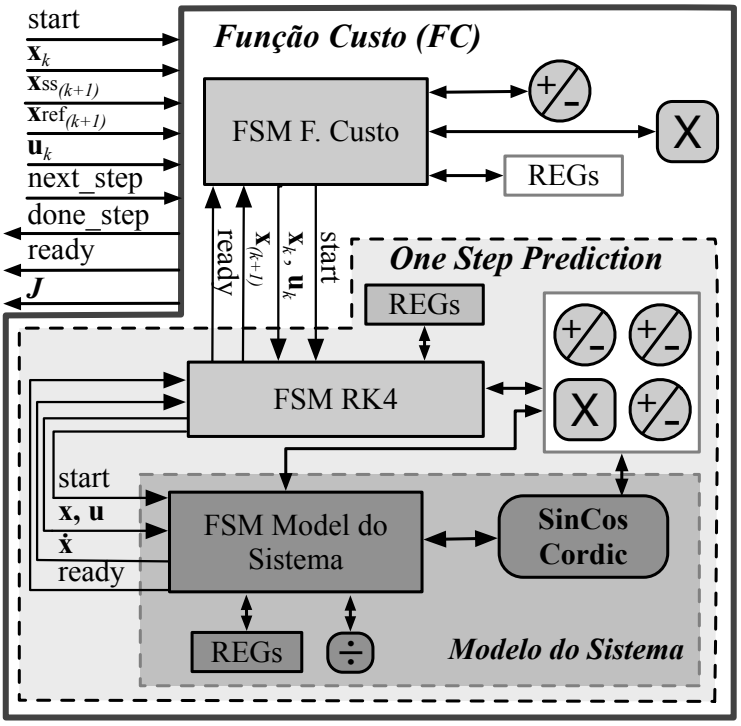



Figura 5.7: Arquitetura de hardware da Função Custo (FC).

#### 5.3.2.3 *Wrapper* AVALON da Função Custo

Com o módulo da *Função Custo* (FC) pronto, o passo final para sua utilização é a integração com o barramento para que o mesmo possa se comunicar com o processador ARM. Neste caso, optou-se pelo uso do barramento mais simples (Lightweight HPS-to-FPGA AXI bridge) que utiliza uma interface *Avalon® Memory-Mapped* (Avalon-MM). Um módulo *wrapper* no padrão *Avalon Slave* (AVS) foi desenvolvido conforme descrito na Figura 5.8. Internamente este módulo instancia $S$ módulos FC em paralelo e possui uma memória RAM para armazenar o conjunto de vetores de referências dos estados.

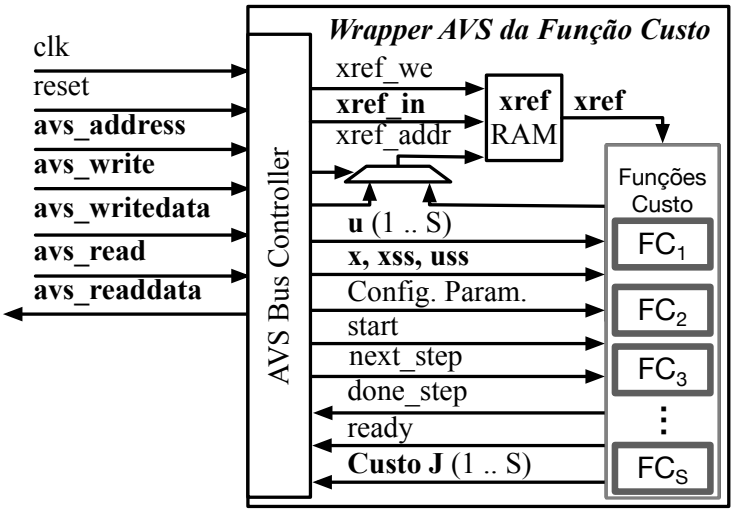



Figura 5.8: *Wrapper* AVS do módulo da Função Custo.

Neste caso, tem-se uma solução de *co-design* em que o algoritmo PSO funciona em software (C++) rodando no processador ARM e, ao atingir o estágio de avaliação da função custo, envia ao módulo em hardware o estado atual do sistema $\mathbf{x}$, os vetores de referência $\mathbf{x}_{ref}$, $\mathbf{x}_{ss}$ e $\mathbf{u}_{ss}$, e os dados das partículas, ou seja, os $S$ vetores $\mathbf{u}$ a serem avaliados.

### 5.3.3 Resultados de desempenho e síntese da solução em *co-design*

Duas soluções foram geradas utilizando 10 partículas ($S = 10$) em paralelo com precisão de 32-bits em ponto flutuante para os Cenários 2 e 3. O ganho inicial de desempenho medido foi de aproximadamente 95% quando os dados são transmitidos pelo barramento e depois se aciona o módulo em hardware a cada ciclo de avaliação da função custo. Decidiu-se então adotar uma abordagem correlata à estratégia S2.2 e enviar ao hardware os valores de uma das dimensões de todas as partículas por vez de modo a trabalhar em *pipeline*. Desta maneira, enquanto os módulos de função custo estão realizando a predição do primeiro período de amostragem e calculando seu custo, o processador vai enviando os valores da próxima dimensão das partículas pelo barramento. Esta segunda abordagem gerou um ganho de aproximadamente 200% em termos de tempo de processamento.

Os dados de desempenho do sistema são apresentados na Tabela 5.4 para os Cenários 2 e 3, onde os módulos em hardware estão funcionando a uma frequência de 100 MHz. É possível observar que a abordagem de *co-design*, para este problema do pêndulo é capaz de atender aos requisitos de tempo-real do Cenário 2, porém não do Cenário 3.

Tabela 5.4: Tempo de processamento da solução em *co-design*.

| Arquitetura | Tempo de Processamento Máximo (ms) | Fator de Aceleração |
|---|---|---|
| Cenário 2 (Ts = 100ms, N= 20) | | |
| Software no ARM | 12,63 | - |
| Co-design (Envio Sequencial) | 6,51 | 1,94 x |
| Co-design (Envio em *Pipeline*) | 4,20 | 3,01 x |
| Cenário 3 (Ts = 20ms, N= 100) | | |
| Software no ARM | 53,32 | - |
| Co-design (Envio em *Pipeline*) | 21,3 | 2,50 x |

Em seguida, a Tabela 5.5 apresenta os dados de síntese.

Tabela 5.5: Resultados de síntese no FPGA.

| Arquitetura (Precisão PF) | ALMs (41.910) | REGs | BRAM (bits) 5.662.720 | Mult. (112) | Freq. Máx (MHz) |
|---|---|---|---|---|---|
| Co-design (32 bits, Cenário 2) | 35.012 | 33.159 | 14.336 | 30 | 110,50 |

É importante ressaltar que esta solução, onde somente a função custo é implementada em hardware, é uma boa candidata a ser aplicada a sistemas onde seja necessário ter mais flexibilidade no algoritmo de busca, como por exemplo o uso de outras abordagens de algoritmos bio-inspirados.

Além disso, no caso de problemas que exijam um maior número de partículas, um ganho ainda maior pode ser alcançado com o paralelismo de um maior número de funções custo.

### 5.3.4 Módulo NMPC-KPSO+SS em hardware

O próximo passo foi desenvolver a solução completa do PSO em hardware. É importante ressaltar que uma arquitetura de hardware implementando o PSO em ponto-flutuante foi apresentada por Muñoz em [130]. Porém, esta arquitetura é genérica e não atende aos requisitos de inicialização e atualização de partículas obedecendo às restrições do NMPC. Além disso, utiliza registradores para armazenar as partículas, o que consome muitos recursos do FPGA. Por último, neste trabalho, uma estratégia de *pipeline* específica integrando a atualização das partículas com a avaliação da função custo do NMPC foi desenvolvida. Com todos estes fatores, foi necessária uma implementação totalmente nova do PSO. O único módulo aproveitado da arquitetura descrita em [130] foi o Gerador de Números Aleatórios (GNA) que foi adaptado para gerar valores com larguras variáveis de ponto-flutuante.

O módulo NMPC-PSO é responsável por implementar o Algoritmo 2 do PSO descrito na Seção 4.8 e instancia os $S$ módulos da *Função Custo* em paralelo. Suas entradas são compostas por pelo estado atual do sistema ($\mathbf{x}$) e pelos sinais de referência($\mathbf{x}_{ss}, \mathbf{u}_{ss}, \mathbf{x}_{ref}$). Como saída temos o primeiro termo da solução ótima ($\mathbf{u}^{opt}$) a ser aplicado ao sistema. Internamente, além das instâncias da *Função Custo*, temos oito submódulos que representam cada uma das etapas do PSO, sendo eles: *Inicialização das Partículas*, *Atualização das Partículas*, *Detecção do Melhor Local*, *Detecção do Melhor Global*, *Gerador de Números Aleatórios* (GNA), blocos de *RAM*, unidade *FPadd* e unidade *FPmul*. Todos estes submódulos são gerenciados por uma Máquina de Estados (FSM) responsável por executar o algoritmo com o *pipeline* definido pela estratégia (S2.2).

Uma representação de mais alto nível do módulo é apresentada no diagrama da Figura 5.9 onde se observa as interações entre as entradas e saídas externas com os respectivos módulos internos.

Em termos de funcionamento, a primeira etapa de execução controlada pela máquina de estados finitos (FSM) é a ativação do módulo *Inicializa Partículas* que utiliza as saídas do *Gerador de Números Aleatórios* (GNA) para inicializar os valores de **u**, **y** e **v** obedecendo às restrições de $\Delta u$. Conforme mencionado, o módulo GNA foi adaptado de [130] e utiliza um algoritmo do tipo *Linear Feedback Shift Register* (LFSR) de 20 bits em ponto-fixo e converte a saída para ponto-flutuante. As duas modificações realizadas foram: atualizar o intervalo de saída para $[0, 1]$ ao invés de $[-1, 1]$ e, tornar a saída de tamanho variável conforme a largura definida nas representações em ponto-flutuante. Após a inicialização das partículas, a técnica do KPSO+SS é aplicada, onde a partícula da posição 1 é inicializada com o melhor valor calculado no período de amostragem anterior ($u^{opt}_{anterior}$) e a posição 2 é inicializada com o valor de $u_{ss}$.

Em seguida, os módulos de *Função Custo* são iniciados em paralelo para calcular o custo de

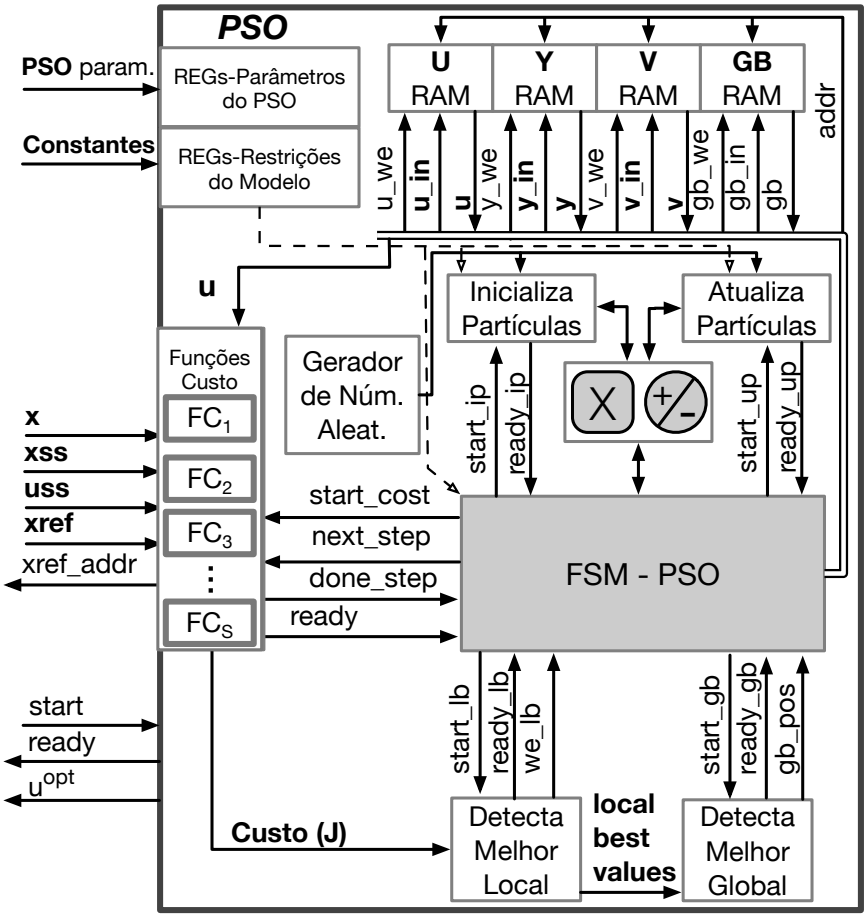



Figura 5.9: Arquitetura de hardwade do KPSO+SS.

cada partícula. Ao sinalizarem o fim dos cálculos, o módulo de *Detecção do Melhor Local* é ativado onde os resultados dos custos são comparados com registradores internos que armazenam o *Melhor Local* de cada partícula. A saída deste módulo é um vetor que indica quais posições da memória **y** devem ser atualizadas com os valores atuais de **u**. Na primeira iteração do algoritmo, os valores dos custos são simplesmente copiados para estes registradores e todas as posições **u** são copiadas para **y**.

O próximo módulo ativado é o de *Detecção do Melhor Global* que compara todos os resultados, encontra a posição do *Melhor Global* (GB - *Global Best*) e copia a os valores de **u** da melhor partícula para a respectiva área de memória (GB_RAM).

O último módulo ativado é o *Atualiza Partículas* que calcula as novas velocidades e posições de cada partícula baseado nos valores atuais de **u**, **y**, **v** e $global\_best$. Este módulo implementa as Equações (2.34) e (2.36) e utiliza saídas do GNA para os cálculos. Neste momento, o *pipeline* já é ativado e os módulos de *Função Custo* são acionados em paralelo, dando continuidade ao ciclo de iterações do algoritmo.

Ao final de cada iteração, o último procedimento executado é o de verificação do critério de parada, neste caso o número máximo de iterações (*MaxIter*) e o valor de inércia ($w$) é atualizado

(Eq. (2.35)). Quando se atinge o fim das iterações, a melhor posição, correspondente ao ($u^{opt}$), é armazenada na memória GB_RAM, é copiado para a saída e um sinal de *ready* é acionado.

A seguir, descrevemos cada um dos submódulos do PSO.

#### 5.3.4.1 Memória RAM das partículas

Para reduzir o uso de registradores, as variáveis de partículas ($u$, $y$, $v$ e *global_best* (GB)) são armazenadas em blocos de memória RAM internos do FPGA (BRAMs). Isto se torna importante, principalmente quando se considera sistemas com múltiplos atuadores e com horizontes de predição longos onde a dimensão das partículas, de tamanho $Nu \times N$, pode assumir valores acima de 300. A estrutura da memória foi organizada em quatro blocos com endereço de acesso único, porém com sinais de escrita (*write_enable*) independentes. Conforme descrito nas estratégias de *pipeline* (S1.2 e S2.2), a cada ciclo de processamento, uma das dimensões de todas as partículas é acessada e, portanto, esta foi a estrutura de memória adotada. O endereçamento de memória varia no intervalo de [1 .. N] onde cada endereço corresponde a $S$ variáveis de $u$, $y$ e $v$, correspondendo a uma dimensão específica de cada partícula, portanto o endereço de memória acessa o intervalo de [1 .. N].

A arquitetura da memória é representada na Figura 5.10.

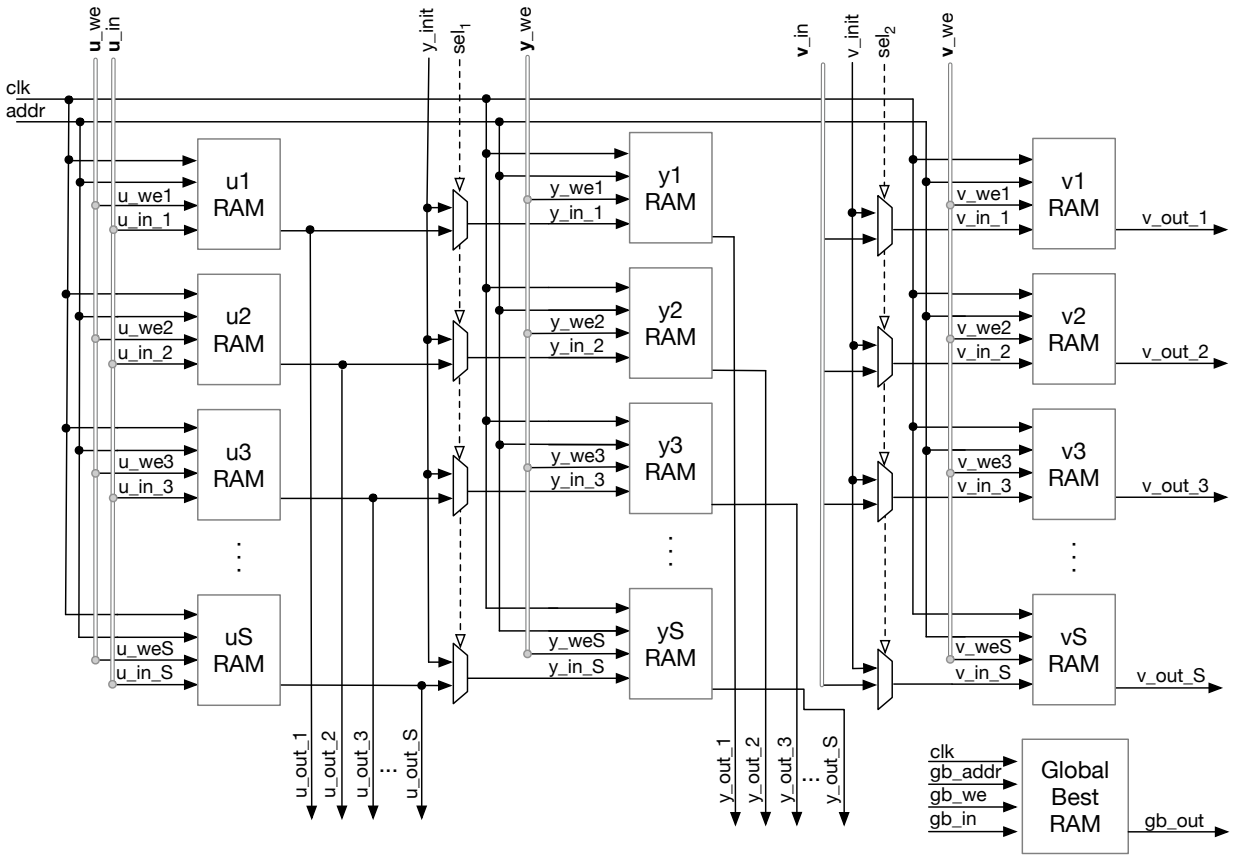



Figura 5.10: Blocos de memória RAM.

#### 5.3.4.2 Inicializa Partículas

O módulo que inicializa as partículas é responsável pelo cálculo de cada uma das dimensões de cada partícula. O laço que percorre cada uma das partículas na memória RAM é controlado pela FSM presente no módulo superior (NMPC-PSO). A Equação (5.8) descreve o procedimento de inicialização de cada dimensão de cada partícula:

$$u_{i,j} = \begin{cases} (u_{anterior} - \Delta u_{max}) + (2 \times \Delta u_{max}) \times rand, & \text{se } j = 1 \\ (u_{i,j-1} - \Delta u_{max}) + (2 \times \Delta u_{max}) \times rand & \text{se } j > 1 \end{cases} \tag{5.8}$$

onde $i$ é i índice da partícula, $j$ é a dimensão de cada partícula e $rand$ é um número aleatório no intervalo [0,1].

O valor calculado passa, em seguida pela verificação dos valores de máximo ($u_{max}$) e mínimo ($u_{min}$):

$$u_{i,j} = \begin{cases} u_{max}, & \text{se } u_{i,j} > u_{max} \\ u_{min}, & \text{se } u_{i,j} < u_{min} \\ u_{i,j} & \text{caso contrário.} \end{cases} \tag{5.9}$$

A arquitetura do módulo é apresentada na Figura 5.11.

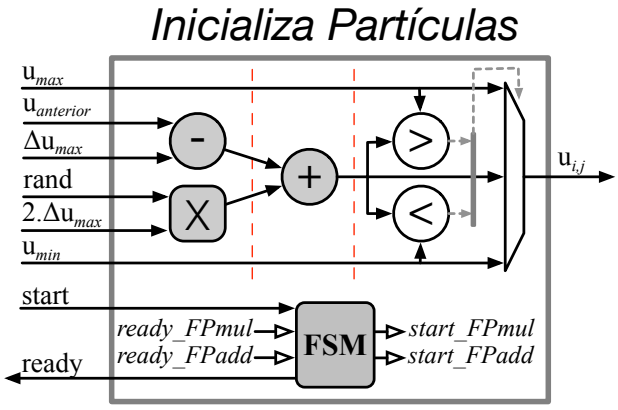



Figura 5.11: Módulo Inicializa Partículas.

#### 5.3.4.3 Atualiza Partículas

O módulo de atualização das partícula, representado na Figura 5.12, é responsável por implementar as Equações (2.36), e (2.34) para o cálculo da nova velocidade ($v$) e nova posição ($u$), respectivamente. Para respeitar as restrições do NMPC, o valor da posição de cada dimensão da partícula é subtraído da dimensão anterior e comparado ao valor de $\Delta u_{min}$ conforme a Equação

(5.10).

$$u_{i,j} = \begin{cases} u_{i,j} + \Delta u_{max}, & \text{se } (u_{i,j} - u_{i,j-1}) > \Delta u_{max} \\ u_{i,j} - \Delta u_{max}, & \text{se } (u_{i,j-1} - u_{i,j}) > \Delta u_{max} \\ u_{i,j} & \text{caso contrário.} \end{cases} \tag{5.10}$$

Em seguida, utiliza-se a Equação (5.8) para verificar as restrições de posição absoluta.

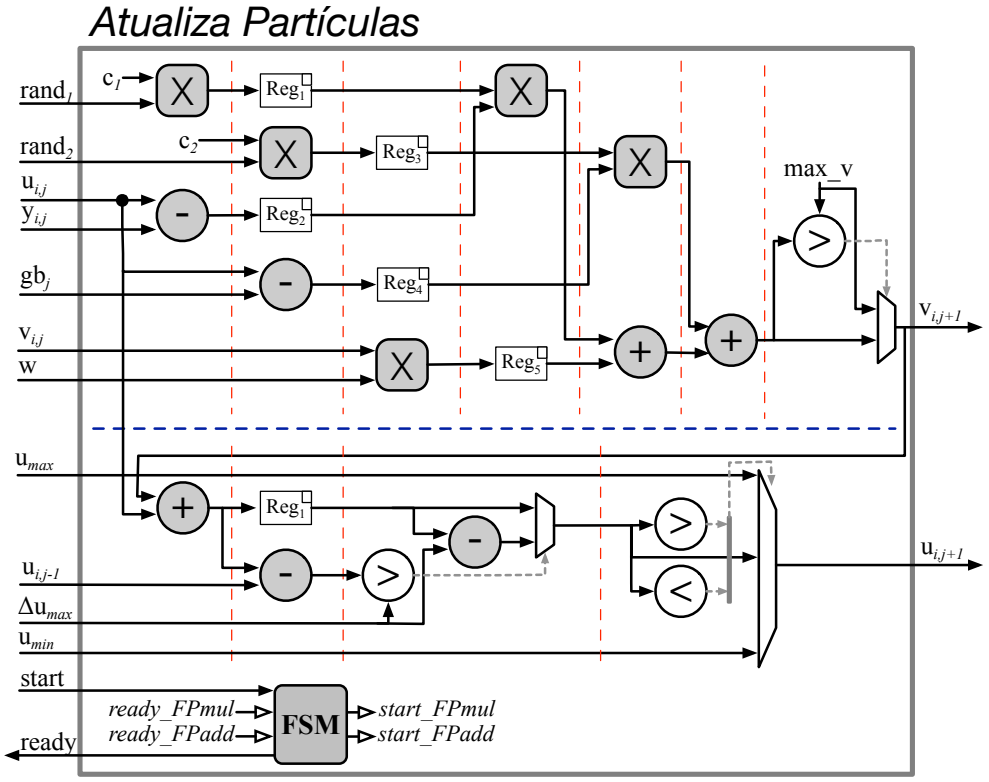



Figura 5.12: Módulo Atualiza Partículas.

#### 5.3.4.4 Detecta Melhor Local

O módulo de detecção do melhor local executa todas as comparações em paralelo em um ciclo de *clock*. Suas entradas são os valores atuais das saídas dos módulos da *Função Custo*. Internamente, registradores guardam o valor do melhor local de cada partícula e são reprogramados com o valor de infinito (*inf*) na primeira iteração do PSO. Além do valor de melhor custo local a ser utilizado na detecção do melhor global, outra saída deste módulo é um vetor de escrita (*write_enable*) para as posições de memória $y$ que serão atualizadas com os valores atuais de $u$. Este procedimento é realizado pela FSM do módulo superior (NMPC-PSO).

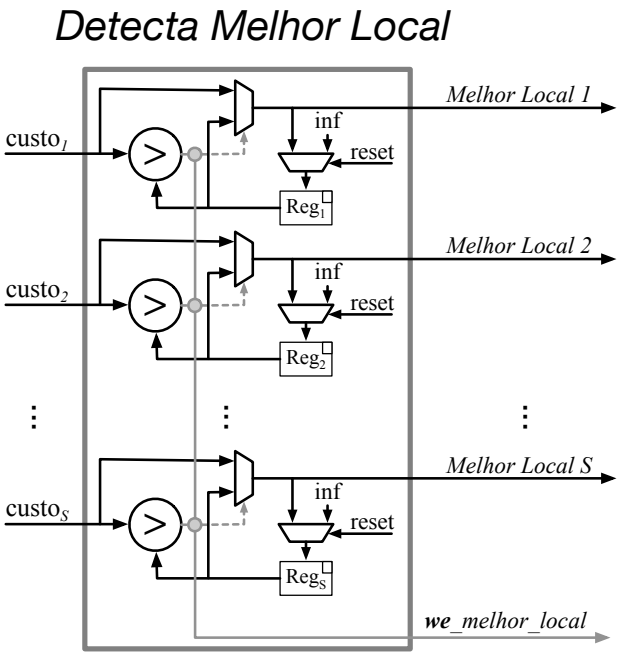



Figura 5.13: Módulo Detecta Melhor Local.

#### 5.3.4.5 Detecta Melhor Global

O último módulo interno do NMPC-PSO realiza a detecção do melhor custo global de cada iteração. Este módulo recebe do anterior, todos os valores de melhor local e detecta a posição da melhor partícula. As comparações são realizadas em uma árvore de comparação conforme apresentado na Figura 5.14 onde o valor do melhor local juntamente com sua posição são transmitidos a cada etapa e, ao final, obtém-se o valor de melhor global juntamente com sua posição.

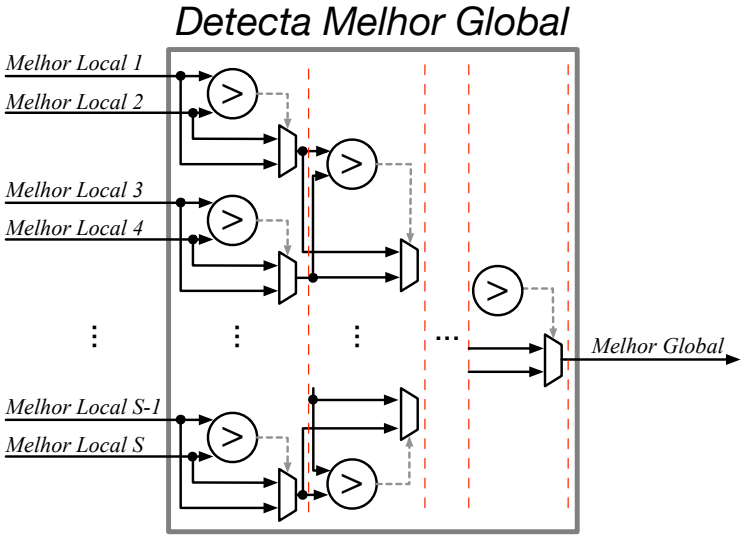



Figura 5.14: Módulo Detecta Melhor Global.

#### 5.3.4.6 *Wrapper* NMPC-PSO

Conforme apresentado no módulo da *Função Custo*, a conexão do módulo NMPC-PSO com o restante do sistema foi realizada através do barramento *Lightweight HPS-to-FPGA AXI bridge*. Neste caso, temos o módulo NMPC-PSO funcionando todo em hardware enquanto a comunicação com o sistema a ser controlado é feita pelo processador ARM. Diferente do módulo da *Função Custo*, ao invés de centenas de dados (todas as partículas e suas dimensões) sendo transmitidos pelo barramento, para o módulo NMPC-PSO funcionar somente são necessários dados atualizados do estado atual do sistema e as referências de estado ($x_{ref}$, $x_{ss}$) e de controle ($u_{ss}$). O único dado retornado é o valor ótimo do atuador a ser aplicado no próximo período de amostragem ($u^{opt_1}$). No exemplo do pêndulo invertido isso significa 4 estados, 4 valores de referência de estados e de estado estacionário e mais um valor de $u_{ss}$, ou seja, 13 valores em ponto-flutuante de entrada e 1 de retorno, ao invés dos 209 de entrada e 10 de retorno no caso de ter somente as funções custo e hardware. Isso torna o custo de tempo de transmissão dos dados pelo barramento praticamente desprezível.

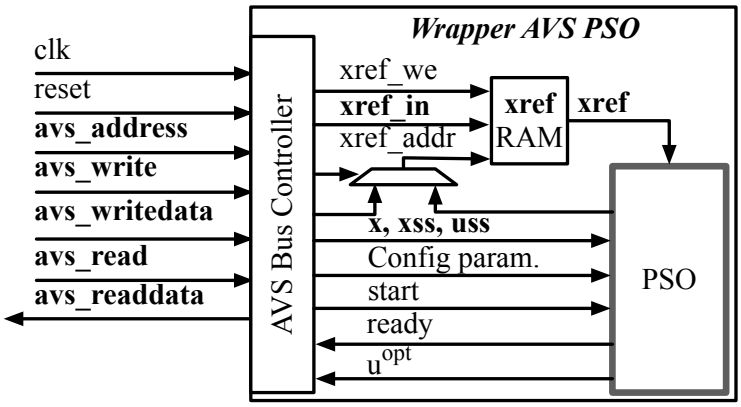



Figura 5.15: Módulo de hardware do *Wrapper* Avalon para o módulo PSO.

### 5.3.5 Gerador automático da arquitetura em hardware

Para facilitar a aplicação desta arquitetura a outros sistemas, um gerador automático capaz de customizar o código foi criado. Uma vez definidos e testados todos os módulos de hardware, os mesmos foram reescritos em formato de *template*. A partir dos *templates* foram criados *scripts* em MATLAB que utilizam com entradas os parâmetros que definem a arquitetura como o tamanho do horizonte de predição e de controle, o número de partículas, o número de estados e de ações de controle para configurar a largura das memórias, número de instâncias e comprimento dos laços em cada módulo. Outros parâmetros como os pesos da função custo, restrições, tempo de amostragem e as constantes do PSO são mapeados em registradores internos dos módulos e podem ser configurados dinamicamente via software com o sistema em funcionamento.

Em conjunto com a definição geral da arquitetura, baseado nas bibliotecas de ponto-flutuante com precisão variável, é possível configurar a largura em bits da representação dos dados reais.

Esta propriedade facilita a exploração de várias instâncias da arquitetura com precisão variada e verificar o mínimo necessário para atender aos requisitos do sistema. Este procedimento ajuda a reduzir a utilização de recursos de hardware.

A Figura 5.16 apresenta o fluxo de geração da arquitetura a partir das configurações. Neste caso, basta definir as variáveis do sistema (Pêndulo Invertido, por exemplo) e do NMPC-PSO e acionar os *scripts* de configuração para que os mesmos gerem os arquivos VHDL correspondentes. Além da arquitetura, cenários de teste podem ser utilizados para gerar automaticamente os *testbenches* de cada um dos módulos do sistema.

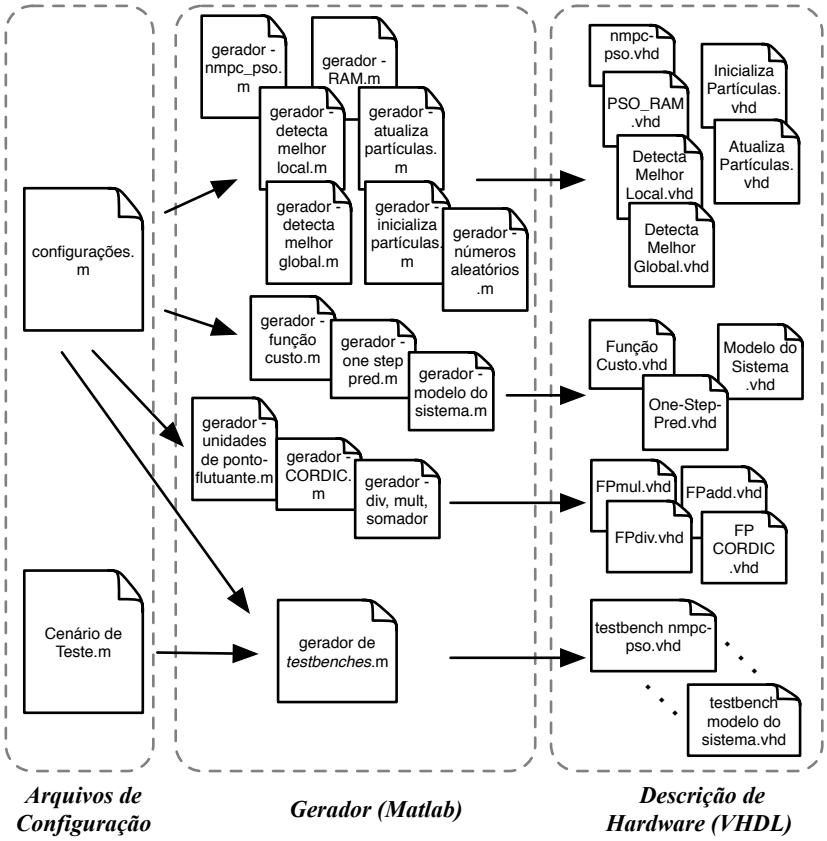



Figura 5.16: Diagrama do gerador automático da arquitetura.

### 5.3.6 Resultados de desempenho e síntese da solução NMPC-PSO

Com o objetivo de avaliar o desempenho da arquitetura em hardware no controle do sistema, dois conjuntos de experimentos foram realizados. O primeiro para medir o tempo de cálculo do controlador aplicado ao sistema em tempo-real e o segundo para otimizar o uso dos recursos de hardware explorando a flexibilidade da arquitetura para variar a precisão em ponto-flutuante e definir a mínima representação que seja suficiente para manter o controle do sistema.

Em ambos os casos, os experimentos foram realizados utilizando a abordagem *hardware-in-the-loop* (HIL) onde o kit com o FPGA foi conectado ao *Target PC* que executa a ferramenta Simulink Real-Time$^{TM}$ do MATLAB (Figura 5.17). O modelo do pêndulo invertido foi confi-

gurado para ser executado a uma frequência de atualização de 200 $\mu$s enquanto o controlador no FPGA foi executado nos Cenários 2 e 3 com uma taxa de amostragem ($Ta$) de 100 e 20 ms, respectivamente. No procedimento de *swing-up* do pêndulo, o estado inicial do sistema foi de $x = [0\ 0\ pi\ 0]^T$, representando o pêndulo na posição para baixo e o estado de referência bem como o estado estacionário desejado foram configurados como $x_{ref} = x_{ss} = [0\ 0\ 0\ 0]^T$, representando o pêndulo na posição vertical para cima.

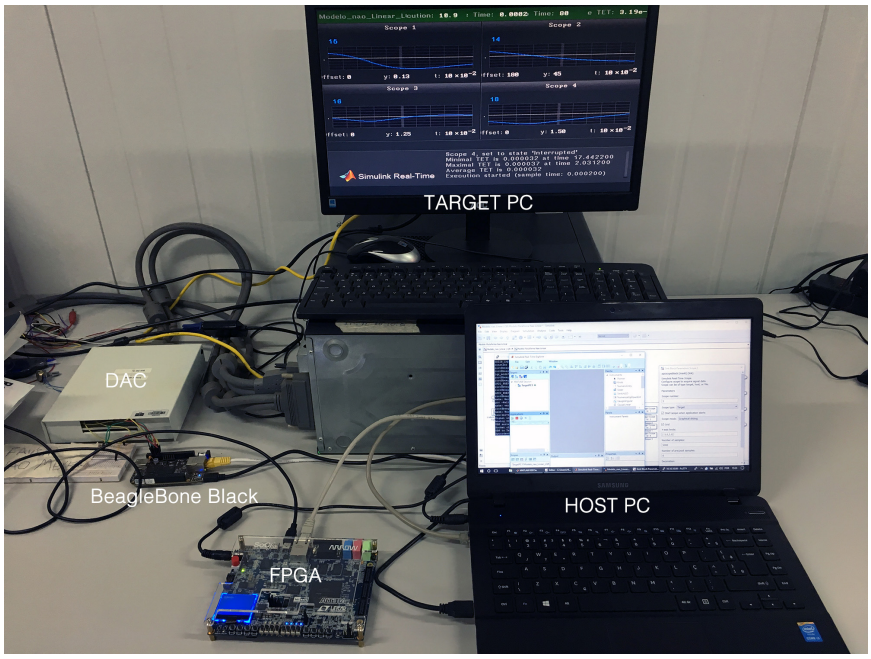



Figura 5.17: Foto da bancada com a platadorma HIL conectada ao FPGA.

O *Target PC* utiliza uma placa de conversão de sinal digital para analógico (DAC) onde são disponibilizados os valores dos estados do sistema. Como o kit FPGA utilizado não possui entradas analógicas (conversor ADC), utilizou-se uma BreagleBone Black que possui 7 conversores AD para realizar a leitura periódica dos estados. Portanto, neste caso, criou-se um software servidor de estados que funciona na placa BreagleBone Black e que se comunica com o software sendo executado no processador ARM do kit FPGA. Esta comunicação utiliza a porta Ethernet e o protocolo UDP para minimizar o atraso na transmissão dos dados. Já o envio das ações de controle é realizado digitalmente pela Ethernet, também utilizando o protocolo UDP mas, neste caso, da placa FPGA diretamente para o *Target PC*. Um diagrama ilustrando os detalhes de comunicação é apresentado na Figura 5.18. Uma descrição detalhada da plataforma HIL utilizada é apresentada por Rodrigues em [131].

Em todos os casos durante o uso da plataforma HIL se assume que todos os estados do sistema podem ser lidos diretamente, ou seja, não há necessidade de um estimador de estados. Porém, como há a presença da conversão AD/DA, ruídos aleatórios são naturalmente introduzidos no sistema na leitura dos estados, o que aproxima mais a simulação a um sistema físico real.

Uma particularidade da leitura de estados e envio de sinais de controle em uma plataforma funcionando em tempo-real é que, após ler o estado atual (instante $k$), o algoritmo de otimização é

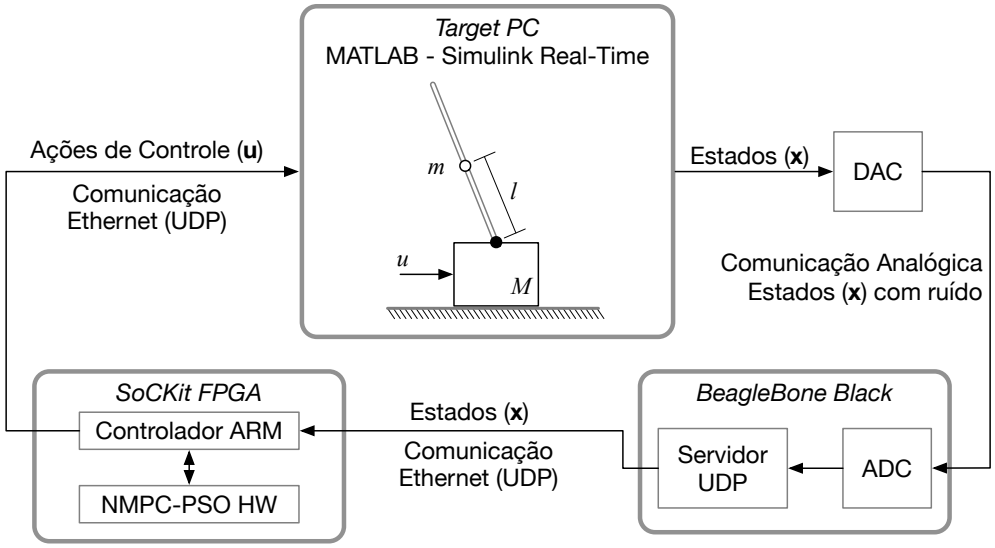



Figura 5.18: Diagrama da platadorma HIL conectada ao FPGA.

calculado para obter a resposta $u^{opt}$ e, esta ação de controle só será injetada novamente no sistema no próximo período de amostragem, ou seja, no instante $k + Ta$. Porém, isto faz com que a ação de controle seja aplicada em um estado do sistema diferente do que foi lido e pode dificultar o controle para sistemas com dinâmicas rápidas. Neste caso, adotou-se um procedimento de ajuste que melhorou o desempenho do controlador da seguinte maneira: ao ler o estado atual $\mathbf{x}_k$ no instante $k$, utiliza-se a função *one-step prediction* para calcular o estado estimado ($\hat{\mathbf{x}}$) no instante $k + Ta$. Em seguida, o procedimento de otimização é realizado em cima de $\hat{\mathbf{x}}$ ao invés de $\mathbf{x}$. Portanto, o resultado $u^{opt}$ é enviado em $k + Ta$ baseado em $\hat{\mathbf{x}}$.

#### 5.3.6.1 Desempenho computacional

Para o primeiro experimento, três implementações do algoritmo NMPC-PSO foram utilizadas. A primeira sendo a solução de software em C++ executando no processador ARM da placa SocKit FPGA em cima do sistema operacional Linux. A segunda é a solução de *codesign* com as funções custo sendo executadas paralelamente em hardware e o algoritmo PSO em software. A terceira solução utiliza a implementação do NMPC-PSO completa em hardware. Em todos os casos, a leitura dos estados e envio de ações de controle são realizadas via software via Ethernet utilizando o protocolo de comunicação UDP. Conforme apresentado, a leitura dos estados vem da placa BeagleBone Black enquanto o envio das ações de controle vai direto para o *Target PC*.

Além disso, todas as soluções utilizam 10 partículas ($S$ = 10) onde nas soluções em hardware estas partículas são executadas em instâncias paralelas do módulo da *Função Custo*. O processador ARM está sendo executado a um frequência de 800MHz enquanto as arquiteturas em FPGA estão a 100MHz.

Os resultados obtidos a partir de 30 rodadas de experimentos são apresentados na Tabela 5.6. Nela se observa os tempos máximos de execução do laço de controle para um período de

amostragem.

Tabela 5.6: Resultados de tempo computacional.

| Solução | Tempo de Cáclulo NMPC-PSO (ms) | Fator de Aceleração |
|---|---|---|
| Cenário 2 (Ts = 100ms, N= 20) | | |
| Software no ARM | 12,63 | - |
| Co-design (Envio Sequencial) | 6,51 | 1,94 x |
| Co-design (Envio em *Pipeline*) | 4,20 | 3,01 x |
| NMPC-PSO em HW | 2,92 | 4,33 x |
| Cenário 3 (Ts = 20ms, N= 100) | | |
| Software no ARM | 53,32 | - |
| Co-design (Envio em *Pipeline*) | 21,3 | 2,50 x |
| NMPC-PSO em HW | 14,2 | 4,30 x |

Assim como apresentado anteriormente na Tabela 5.4, todas as soluções foram capazes de atender aos requisitos de tempo-real do Cenário 2. Aqui se destaca a solução NMPC-PSO em hardware que é 332% mais rápida que a solução em software e 44% mais rápida que a solução de *co-design* com envio em *Pipeline*.

Já no Cenário 3, que é mais exigente, nenhuma das soluções anteriores havia sido capaz de atender aos requisitos e, portanto, a solução completa em hardware é a única adequada. Se observa ainda que seu desempenho é 50% mais rápida que a solução de *co-design*, já que é possível aqui explorar um nível ainda mais profundo de *pipeline* do algoritmo.

Um outro aspecto desempenho a ser ressaltado é o ganho de desempenho a partir das abordagens de paralelismo. A Tabela 5.7 mostra uma comparação em termos de ciclos de *clock* para o Cenário 2, considerando as diferentes combinações de estratégia de paralelismo.

Tabela 5.7: Redução de Ciclos de *clock* a partir das estratégias de paralelismo.

| Estratégia | Ciclos | Redução |
|---|---|---|
| Predição + Função Custo (Serial) - S1.1 | 139.640 | - |
| Predição + Função Custo (Pipeline) - S1.2 | 119.520 | 14,4% |
| Predição + Função Custo + Atualiza Partículas (Serial) - S1.1 + S2.1 | 226.840 | - |
| Predição + Função Custo + Atualiza Partículas (Pipeline) - S1.2 + S2.2 | 123.930 | 45,4% |

#### 5.3.6.2 Estudo da variação de precisão do hardware em ponto-flutuante

O segundo experimento explora a relação de custo-benefício entre a precisão da arquitetura em ponto-flutuante e a utilização dos recursos em hardware. Para realizar o experimento foram geradas cinco arquiteturas do NMPC-PSO em hardware variando a representação em ponto flutuante iniciando com a de 32 bits (1 bit de Sinal (S), 8 bits de Expoente (E) e 23 bits de Mantissa (M)), em seguida precisões menores de 27 bits (1S-8E-18M), 24 bits (1S-8E-15M), 21 bits (1S-8E-12M) e 18 bits (1S-8E-9M). Cada uma das arquiteturas foi sintetizada, programada no FPGA e testada na plataforma HIL.

O primeiro conjunto de resultados, apresentado na Figura 5.19, foi obtido para o Cenário 2 e mostra o desempenho do sistema para cada uma das diferentes precisões da arquitetura do NMPC-PSO em hardware. Aqui se observa que somente a arquitetura com 18 bits não é capaz de controlar o sistema. Além disso, é possível observar oscilação no sinal de controle, mesmo já no estado estacionário para todos os casos. Isso ocorre devido ao fato de haver ruído oriundo do processo de conversão do sinal digital para analógico (saída do *Target PC*) e de analógico para digital (BeagleBone Black) na leitura dos estados.

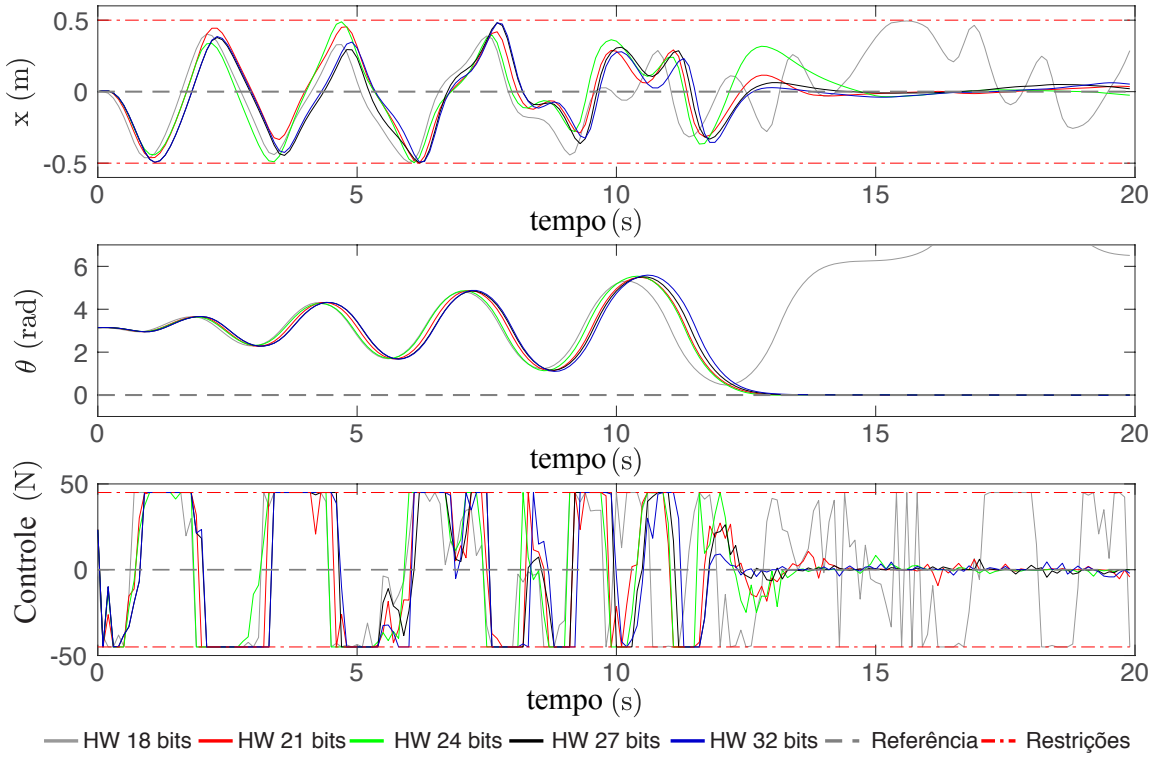



Figura 5.19: Resultados de simulação do FPGA + HIL variando a precisão em ponto-flutuante para o Cenário 2.

Em seguida, o mesmo experimento foi repetido para o Cenário 3 (Ta = 20 ms), e para uma variação do Cenário 1, onde as restrições são mais relaxadas (-1,5 m $\leq x \leq$ 1,5 m, $u_{min}$ = -300 N, $u_{max}$ = 300 N) mas com Ta = 20 ms. Os resultados são apresentados na Figura 5.20 mostrando apenas as arquiteturas de 18, 21 e 32 bits para facilitar a visualização. Nos gráficos da esquerda estão os resultados com as restrições mais relaxadas e na direita os valores do Cenário 3. Aqui se observa mais uma vez que a arquitetura com 18 bits de precisão não é capaz de controlar o sistema. Além disso, é possível observar a oscilação no sinal de controle devido aos ruídos na leitura dos estados.

A Tabela 5.8 apresenta os resultados de síntese das arquiteturas viáveis (acima de 18 bits) para o FPGA de destino (Altera Cyclone V SoC 5CSXFC6D6F31C6N FPGA). Estão discriminados os valores de Módulos Lógicos (*Adaptive Logic Module* - ALMs), quantidade de registradores (REGs), número de bits em blocos de memória RAM interna (BRAM), número de multiplicadores em hardware (Mult.) e a frequência máxima (Freq. Máx.) que a arquitetura é capaz de atingir.

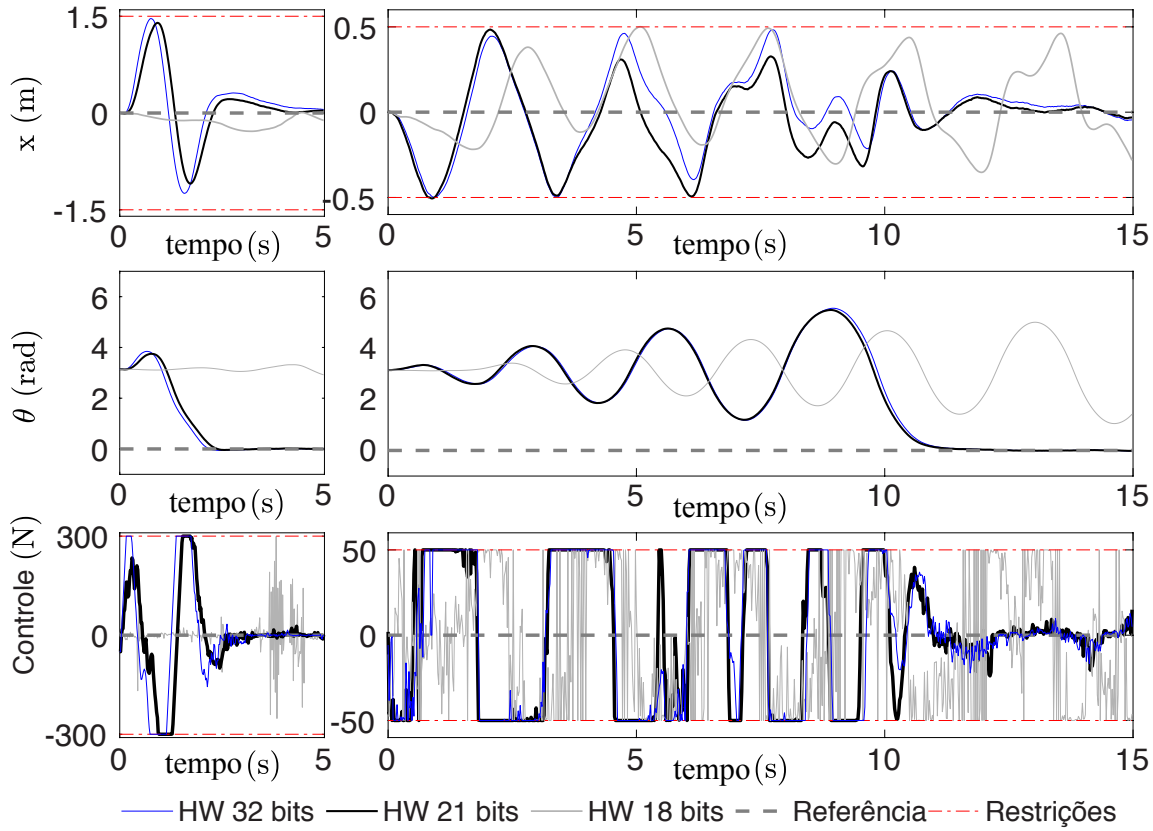



Figura 5.20: Resultados de simulação do FPGA + HIL variando a precisão em ponto-flutuante para o Cenário 3.

Tabela 5.8: Resultados de síntese em FPGA.

| Arquitetura (Precisão PF) | ALMs (41.910) | REGs | BRAM (bits) 5.662.720 | Mult. (112) | Freq. Máx. (MHz) | Potência FPGA (mW) |
|---|---|---|---|---|---|---|
| Cenário 2 (Ts=100ms N=20) | | | | | | |
| Co-design (32 bits) | 35.012 | 33.159 | 14.336 | 30 | 110.50 | 902,73 |
| NMPC-SO HW (32 bits) | 36.993 | 38.602 | 36.160 | 31 | 101,31 | 902,60 |
| NMPC-PSO HW (27 bits) | 31.450 | 32.892 | 30.510 | 31 | 101,74 | 861,80 |
| NMPC-PSO HW (24 bits) | 27.194 | 29.549 | 27.120 | 31 | 102,35 | 808,80 |
| NMPC-PSO HW (21 bits) | 23.217 | 26.299 | 23.730 | 31 | 107,33 | 782,50 |
| Cenário 3 (Ts=20ms N=100) | | | | | | |
| NMPC-PSO HW (32 bits) | 37.040 | 38.967 | 143.680 | 31 | 101,31 | 909,40 |
| NMPC-PSO HW (27 bits) | 30.964 | 32.843 | 121.230 | 31 | 101,74 | 859,41 |
| NMPC-PSO HW (24 bits) | 27.180 | 29.678 | 107.760 | 31 | 102,35 | 814,9 |
| NMPC-PSO HW (21 bits) | 23.274 | 26.233 | 94.290 | 31 | 107,33 | 783,2 |

É possível notar que, diminuindo a representação de ponto-flutuante de 32 para 21 bits, há uma redução de aproximadamente 37% no uso de ALMs, 32% no uso de registradores e 34% no uso de memória RAM. Além disso, se observa um aumento significativo no uso da memória RAM entre os Cenários 2 e 3 devido ao aumento do horizonte de predição, o que reforça a decisão

arquitetural de não utilizar somente os registradores para armazenar os dados das partículas do PSO.

Finalmente, utilizou-se a ferramenta Quartus Prime 17.1 para realizar a estimativa de consumo de potência da arquitetura do PSO completa em hardware. Os resultados de consumo de potência específicos aos módulos dedicados no FPGA são apresentados na última coluna da Tabela 5.8. Neste caso é possível notar uma queda de aproximadamente 13% do consumo de energia ao reduzir a precisão de 32 para 21 bits. Além disso, o consumo médio do processador ARM considerando um núcleo em funcionamento é estimado em 1247 mW. Portanto, a potência total de consumo incluindo o FPGA+ARM da solução de 21 bits fica em 2.176 mW. A título de comparação, o processador Intel Core i7 utilizado como referência utilizou um total de 45W.

## 5.4 SIMULAÇÕES COM PERTURBAÇÃO

Para testar a capacidade de rejeição à perturbações do sistema, um último experimento com o pêndulo invertido foi conduzido utilizando a solução de 21 bits em FPGA. Neste caso, utilizou-se a HIL e o Cenário 2 (Ta = 100 ms). Além dos ruídos na leitura de estados que já são consequência da plataforma HIL, duas outras fontes de perturbação foram adicionadas. A primeira foi uma mudança nos parâmetros do modelo do sistema em relação ao modelo implementado no controlador conforme apresentado em [32]. Estes parâmetros são apresentados na Tabela 5.9.

Tabela 5.9: Constantes do sistema.

| Parâmetro | Modelo Controlador | Modelo HIL |
|---|---|---|
| $M$ | 14,6 kg | 15,33 kg |
| $m$ | 7,3 kg | 6,935 kg |
| $l$ | 1,2 m | 1,26 m |
| $b$ | 14.6 kg/s | 15,33 kg/s |
| $h$ | 0,0136 kg.m$^2$/s | 0.0129 kg.m$^2$/s |
| $g$ | 9,81 m/s$^2$ | 9,81 m/s$^2$ |

Além disso, uma força externa com magnitude de 100 N é aplicada ao sistema no sentido contrário ao movimento do carro no trilho por 100 ms no instante t = 20 s. O resultado do experimento é apresentado na Figura 5.21 onde é possível observar a boa capacidade do sistema de realizar o procedimento de *swing-up* e manter o equilíbrio do sistema tanto com a diferença entre os parâmetros do modelo, o que gera atritos e forças inesperadas, quanto em relação à perturbação externa pontual que desloca o sistema de seu ponto de equilíbrio e é rapidamente compensada pelo controlador.

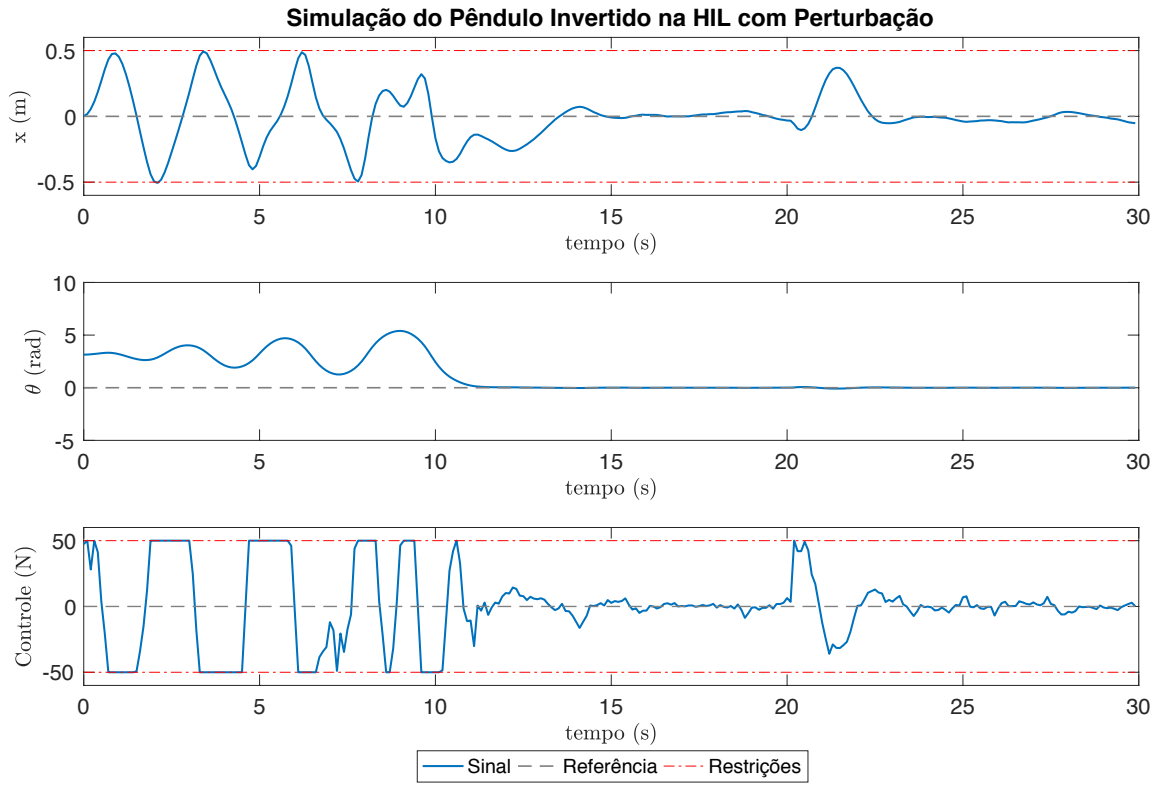



Figura 5.21: Resultados de simulação do FPGA + HIL + Perturbações para o Cenário 2.

## 5.5 DISCUSSÕES FINAIS DO CAPÍTULO E CONTRIBUIÇÕES

Para evidenciar os resultados obtidos a partir desta arquitetura, uma análise comparativa é feita com alguns dos trabalhos relacionados encontrados na literatura.

O primeiro trabalho é o de Khusainov et al. [129] que propõe uma implementação de um controlador NMPC em FPGA utilizando *Interior-Point Methods* como algoritmo de otimização. A solução é aplicada a um guindaste, cujas equações são similares ao caso do pêndulo, porém com dois atuadores (um para o movimento e outro para ajuste do comprimento do cabo). A arquitetura é desenvolvida com auxílio da ferramenta Protoip [64] e utiliza HLS para gerar a descrição de hardware para um FPGA da Xilinx (Zynq 7000) com um processador ARM embarcado, com capacidade e recursos similares ao que se utiliza neste trabalho. O período de amostragem considerado foi de $Ta = 150$ ms e o algoritmo utiliza 15 iterações para encontrar a solução. Foram testados tamanhos variados do horizonte de predição entre 2 e 6. No primeiro caso (N = 2), a melhor solução é capaz de calcular o resultado em 48 ms, no segundo caso (N = 6) somente em 191 ms, não atendendo aos requisitos de tempo-real. O trabalho critica ainda as abordagens de algoritmos estocásticos como o uso do PSO por não garantirem sempre uma solução, porém mostra que não é capaz de resolver problemas com horizontes longos em tempo-real. Em comparação, a arquitetura apresentada neste trabalho é capaz de computar uma solução para um horizonte de N=20 em 2,92 ms. A utilização de hardware não é reportada claramente a não ser por um gráfico de indica o uso de aproximadamente 30% dos recursos e a utilização de 20 unidades multiplica-

doras em ponto-flutuante.

Num segundo trabalho, citado anteriormente, Xu et al. [83] propõe um controlador NMPC em FPGA utilizando o próprio PSO como algoritmo de otimização. Apesar da aplicação ser diferente, controle de rotação de um motor em marcha lenta sem carga, a complexidade do sistema de equações é similar, porém com apenas uma entrada e uma saída controlada. Os horizontes de predição e controle são bastante curtos, N = 5 e Nc = 2. A solução do problema de otimização com o PSO é configurada com 30 partículas e 30 iterações e não utiliza estratégias como o KPSO e nem obedece às restrições de variação do sinal de controle ($\Delta u_{max}$). Quanto à implementação em hardware, o código é gerado utilizando uma ferramenta HLS e toda a arquitetura utiliza aritmética de ponto-fixo. Por conta do uso do HLS, somente uma dimensão de *pipeline* é aplicada e não há paralelismo na avaliação da função custo. A título de comparação, a solução do pêndulo invertido para o Cenário 2 (com N = 20) foi sintetizada no mesmo FPGA utilizado em [83], um Stratix III (EP3SL150F1152). Os resultados, juntamente com os dados do trabalho de Xu et al., são apresentados na Tabela 5.10 onde é possível observar que a arquitetura aqui proposta é mais rápida, mesmo com um horizonte de predição 4 vezes maior, como utiliza 6,8% menos LUTs e 71% menos multiplicadores em hardware (DSPs). Além disso, por ser descrita utilizando aritmética de ponto-flutuante, não necessita de estudo prévio de precisão em ponto fixo para customizar o tamanho da representação numérica a cada etapa de cálculo.

Tabela 5.10: Resultados de síntese em FPGA.

| Arquitetura | LUTs (113.074) | REGs (114.976) | BRAM (bits) (5.499.000) | Mult. (384) | Freq. Máx. (MHz) | N/Nc | Tempo de Cálculo (ms) |
|---|---|---|---|---|---|---|---|
| NMPC-PSO HW (21 bits) | 32.660 | 23.343 | 23.730 | 62 | 166,64 | 20/20 | 2,92 |
| Xu et al. [83] | 35.053 | 14.497 | - | 214 | 40,0 | 5/2 | 3,255 |

Em síntese, neste capítulo, foram apresentados os detalhes do desenvolvimento da arquitetura utilizando tanto a abordagem de *co-design*, onde somente as funções custo são implementadas em hardware, quanto a abordagem de implementação completa do NMPC-PSO em hardware. A primeira solução, quando viável traz mais flexibilidade para alterações futuras no algoritmo de otimização, como por exemplo o uso de outros algoritmos bio-inspirados. A segunda solução utiliza um pouco mais de recursos do FPGA sendo 44% mais rápida.

Para aumentar a eficiência da arquitetura, estratégias de paralelismo e *pipeline* diferentes do que já existe na literatura foram propostas e implementadas. Além disso, otimizações foram feitas nos módulos em ponto-flutuante desenvolvidos em [120, 121, 130] utilizados na arquitetura proposta.

Por fim, testes foram realizados em uma plataforma HIL e mostram a eficiência do sistema em controlar o pêndulo invertido. Em termos de desempenho computacional, a solução completa em hardware se mostrou adequada para ambos os cenários propostos utilizando um FPGA de baixo custo e baixo consumo de energia. Porém, é importante ressaltar que esta mesma arquitetura pode

ser sintetizada em FPGAs de mais alto desempenho e funcionar a uma frequência ainda maior quando necessário. Além disso, caso o sistema necessite de mais partículas, FPGAs maiores podem prover mais espaço e ainda aumentar o fator de aceleração do algoritmo.

# 6 ESTUDOS DE CASO COM O NMPC-PSO

Até o momento, a solução do NMPC-PSO foi apresentada somente para o estudo de caso do pêndulo invertido simples que é um sistema com uma entrada única e múltiplas saídas (SIMO). Com o intuito de demonstrar a generalidade da solução, foram definidos outros dois estudos de caso. O primeiro sendo o de Pêndulos Gêmeos ou duplos descrito por Alamir em [13], estendendo o número de estados e a complexidade do primeiro problema do pêndulo simples. Em seguida, apresentamos um segundo estudo de caso de um sistema com múltiplas entradas e múltiplas saídas (MIMO) que trata do controle de atitude de um satélite com três atuadores e seis estados, apresentado por Rodrigues em [131].

## 6.1 PÊNDULOS GÊMEOS

Seguindo o sistema do Pêndulo Invertido simples apresentado anteriormente, o experimento selecionado para dar sequência foi o do modelo de Pêndulos Gêmeos (*Twin Pendulum System*) baseado no exemplo descrito por Alamir [13]. O problema de controle em questão é semelhante ao do Pêndulo Simples, ou seja, realizar o procedimento de *swing-up*. Porém, neste caso de complexidade maior pois já que é necessário equilibrar os dois pêndulos com o mesmo atuador.

O diagrama do sistema é apresentado na Figura 6.1 onde é possível perceber que ambos os pêndulos são acoplados ao mesmo eixo e giram livremente independentes um do outro.

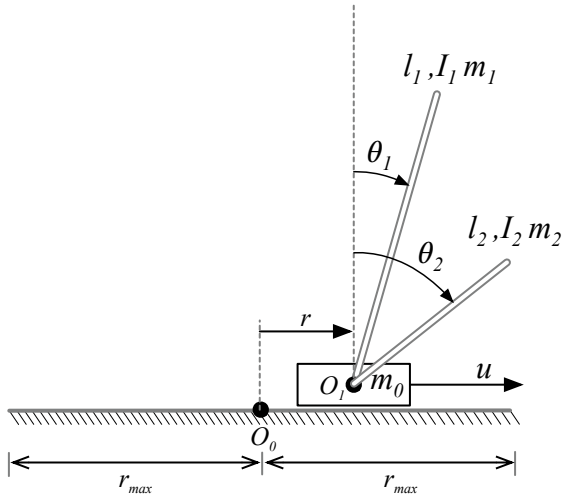



Figura 6.1: Diagrama do sistema de pêndulos gêmeos

No diagrama, $O_0$ representa a origem do sistema, enquanto $O_1$ representa o centro do eixo do carro onde os pêndulos estão acoplados. A posição do carro no trilho é medida a partir da variável $r$ e o valor máximo de excursão no trilho varia de $-r_{max}$ e $+r_{max}$ a partir da origem

$O_0$. O atuador é representado pela entrada $u$ exercendo uma força linear sobre o carro na direção paralela ao trilho. Os dois pêndulos são caracterizados pelas contantes $l_i$, $I_i$, $m_i$ e $\theta_i$ que representam seu comprimento, momento de inércia, massa e ângulo em relação à posição vertical, respectivamente. O estado do sistema possui seis variáveis e é descrito por $x = [r\ \dot{r}\ \theta_1\ \dot{\theta_1}\ \theta_2\ \dot{\theta_2}]^T$:

As equações dinâmicas que descrevem o modelo são apresentadas a seguir:

$$\ddot{r} = v, \tag{6.1}$$

$$\ddot{\theta_1} = -\alpha_1 \cos\theta_1 v + \beta_1 \sin\theta_1, \tag{6.2}$$

$$\ddot{\theta_2} = -\alpha_2 \cos\theta_1 v + \beta_2 \sin\theta_2, \tag{6.3}$$

onde $v$ é definido como:

$$v = \frac{1}{\alpha_0(\theta_1, \theta_2)}[u - \beta_0(\theta_1, \theta_2, \dot{\theta_1}, \dot{\theta_2})], \tag{6.4}$$

e os coeficientes $\alpha_i$ e $\beta_i$ são definidos da seguinte maneira:

$$\alpha_i = \frac{m_i l_i}{m_i l_i^2 + I_i}\ ; \beta_i = g\alpha_i, \tag{6.5}$$

Além disso, os coeficientes $\alpha_0$ e $\beta_0$ são definidos por:

$$\alpha_0(\theta_1, \theta_2) = m_0 + m_1 + m_2 - \sum_{i=1}^{2} \alpha_i m_i l_i \cos^2\theta_i, \tag{6.6}$$

$$\beta_0(\theta_1, \theta_2, \dot{\theta_1}, \dot{\theta_2}) = \sum_{i=1}^{2} m_i l_i \sin\theta_i(\beta_i \cos\theta_i - \dot{\theta}_i^2). \tag{6.7}$$

Os parâmetros físicos do sistema possuem os seguintes valores segundo Alamir [13]:

$$m_0 = 2{,}0 \text{ kg},\ m_1 = 0{,}2 \text{ kg},\ m_2 = 0{,}1 \text{ kg}, g = 9{,}81 \text{ m/s}^2,$$

$$l_1 = 1{,}0 \text{ m}, l_2 = 0{,}5 \text{ m},\ I_1 = 0{,}1 \text{ kg.m}^2,\ I_2 = 0{,}0125 \text{ kg.m}^2.$$

Para realizar o controle do sistema, Alamir [13] propõe uma abordagem mista onde a fase de *swing-up* é realizada a partir de um controlador NMPC não-linear. Nesta etapa, o foco do controlador é inverter ambos os pêndulos e colocá-los na posição vertical, independente de sua posição final no trilho, porém, obedecendo as restrições de excursão entre $-r_{max}$ e $+r_{max}$. Após ambos os pêndulos atingirem a posição vertical de equilíbrio, o controlador é substituído por um MPC linear que mantém a posição do conjunto no trilho. Como o foco deste trabalho é demonstrar o desempenho do controlador não-linear, ou seja, do NMPC utilizando a abordagem do PSO, as simulações são limitadas à inversão do sistema e não seu controle de posição.

Para o controle, o modelo de predição de um passo utilizado segue o método Runge Kutta de 4ª Ordem descrito pela equação 2.25. Já a função custo mede a quantidade de energia de cada

pêndulo em relação à sua posição vertical, com o pêndulo para cima, da seguinte maneira:

$$E_i = \frac{1}{2}\dot{\theta_i}^2 + \beta_i(\cos\theta_i - 1). \tag{6.8}$$

Em seguida, a Equação (6.8) é introduzida na Função custo conforme a Equação (6.9).

$$J(\mathbf{x},\mathbf{u}) = \sum_{i=1}^{N} max\left(\frac{|E_1(k+i)|}{\beta_1}, \frac{|E_2(k+i)|}{\beta_2}\right) + \sum_{i=0}^{N} ||\tilde{\mathbf{u}}(k+i) - \mathbf{u}_{ss}||_R^2, \tag{6.9}$$

onde $E_i(k+i)$ representa o valor predito de $E_i$ no instante futuro $k+i$. Além disso, o último termo da função, ausente na versão apresentada por Alamir [13] penaliza a ação de controle e é uma adição necessária para estabilizar o sinal de controle com o NMPC-PSO.

A aplicação do algoritmo NMPC-PSO neste caso foi direta, seguindo a mesma abordagem desenvolvida no Capítulo 4, ou seja, cada partícula possui $Nc$ dimensões representando o horizonte de controle, obedece às restrições do espaço de busca no intervalo $[u_{min}, u_{max}]$ e as restrições de variação do sinal de controle $\Delta u_{max}$.

Para verificar o desempenho do controlador, dois cenários propostos e utilizados para os experimentos são descritos na Tabela 6.1.

Tabela 6.1: Parâmetros dinâmicos do sistema e do controlador MPC.

| Parâmetro | Cenário 1 [13]. | Cenário 2 [13]. |
|---|---|---|
| $Ta$ | 150 ms | 150 ms |
| $x_{min}$ | -0,5 m | -0,5 m |
| $x_{max}$ | 0,5 m | 0,5 m |
| $u_{min}$ | -5 N | -2 N |
| $u_{max}$ | 5 N | 2 N |
| $\Delta u_{max}$ | 5 N | 2 N |
| $N$ | 20 | 20 |
| $Nc$ | 20 | 20 |
| $R$ | 8 $\times 10^{-3}$ | 8 $\times 10^{-3}$ |
| Estado Inicial ($\mathbf{x}_0$) | $[0\ 0\ pi\ 0\ pi\ 0]^T$ | |
| Referência degrau a partir de $k=0$ | $[0\ 0\ 0\ 0\ 0\ 0]^T$ | |

Com o intuito de verificar o número mínimo de partículas ($S$) e iterações do PSO ($MaxIter$) que consigam realizar o controle do sistema de maneira satisfatória e ainda manter um tempo de processamento baixo, o experimento realizado na Seção 4.5 foi repetido para o Cenário 1. Neste caso, variou-se o número de partículas em $S$ = [10, 20, 30] e *Maxiter* de 10 a 100 (em intervalos de 10), porém, em todos os experimentos a técnica do KPSO+SS é utilizada e os valores dos coeficientes cognitivo ($c1$) e social ($c2$) foram mantidos em 2,1 e 1,0, respectivamente, os mesmos melhores resultados encontrados na Seção 4.6 que mantiveram o melhor desempenho

do sistema. Os resultados são apresentados na Figura 6.2 onde é possível ser observar que, para $S$ = 10, os resultados de MSE são inferiores aos encontrados com $S$=20 e 30 que são semelhantes, principalmente acima de 70 iterações para o caso do sinal de controle. Neste caso, foi adotado $S = 20$ e $MaxIter = 70$.

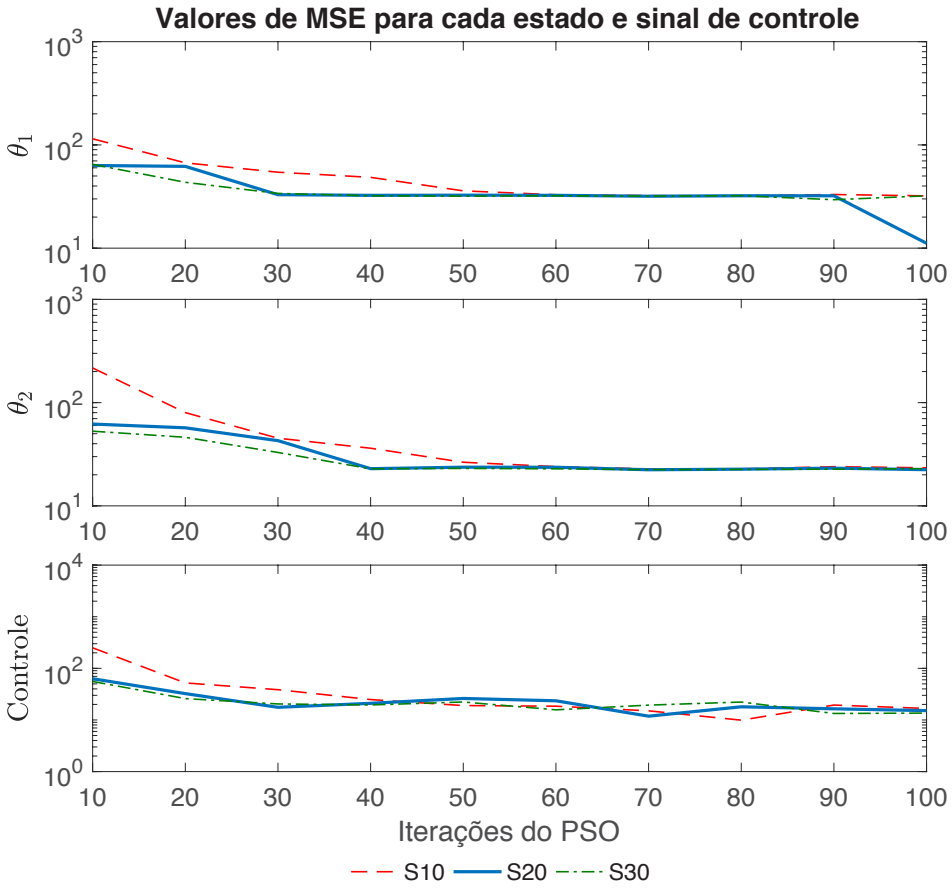



Figura 6.2: Comparação entre o número de partículas e as iterações do PSO para o sistemas de pêndulos gêmeos.

### 6.1.1 Resultados de Simulação e Desempenho

A Figura 6.3 apresenta os resultados de simulação para os Cenários 1 e 2. É possível perceber o bom desempenho do sistema com resultados de controle muito próximos dos apresentados em [13]. A Tabela 6.2 apresenta os resultados de tempo de processamento para os processadores estudados com a solução NMPC-PSO em C++. Uma diferença notável é que em [13] é reportado um tempo de cálculo da solução de aproximadamente 25 ms para o código compilado em Matlab (MEX) para o processador Intel Core i7, enquanto os resultados deste trabalho alcançam uma solução em 6,6 ms com um código em C++, no mesmo processador. É possível ainda perceber que este sistema atinge os requisitos de tempo real para o processador ARM na SocKit FPGA, porém, não para o ARM presente no Raspberry Pi3.

Finalmente, o sistema foi simulado utilizando a mesma plataforma HIL descrita na Seção 5.3.6. Neste caso, o sistema dos pêndulos gêmeos está sendo simulado a uma taxa de amostra-

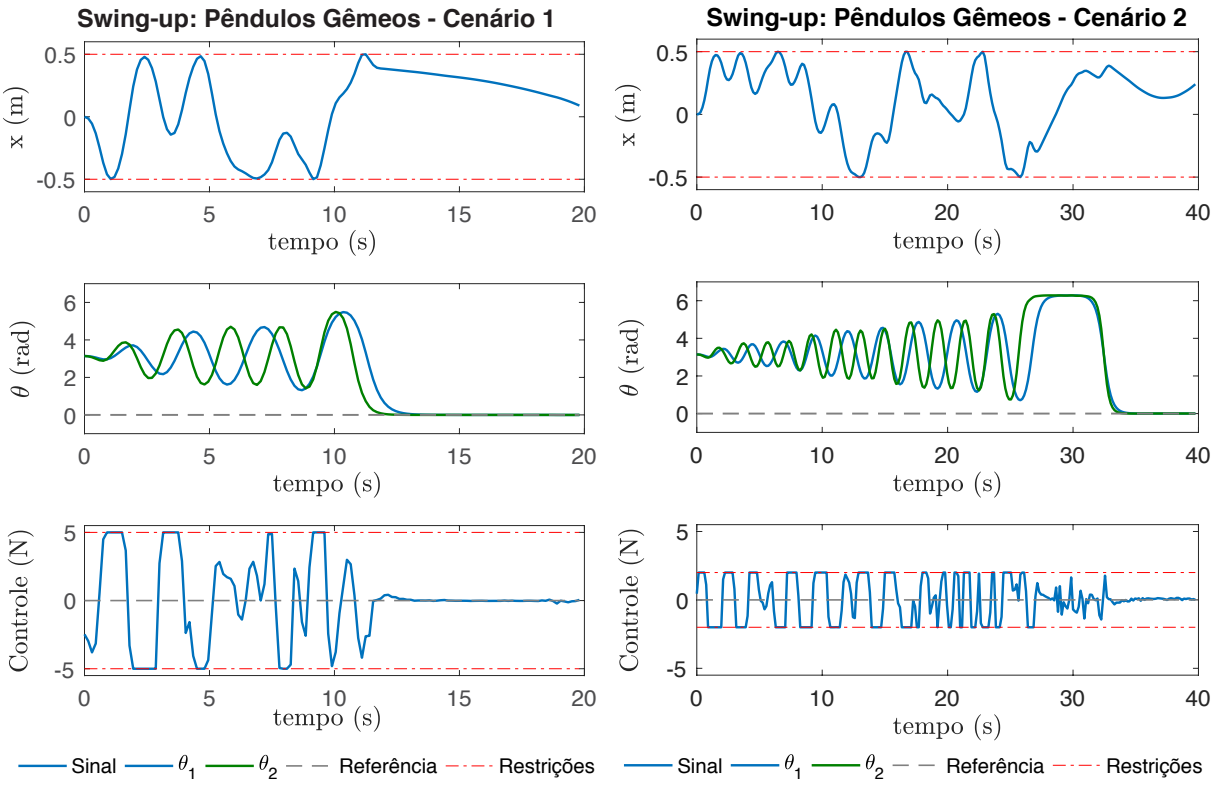
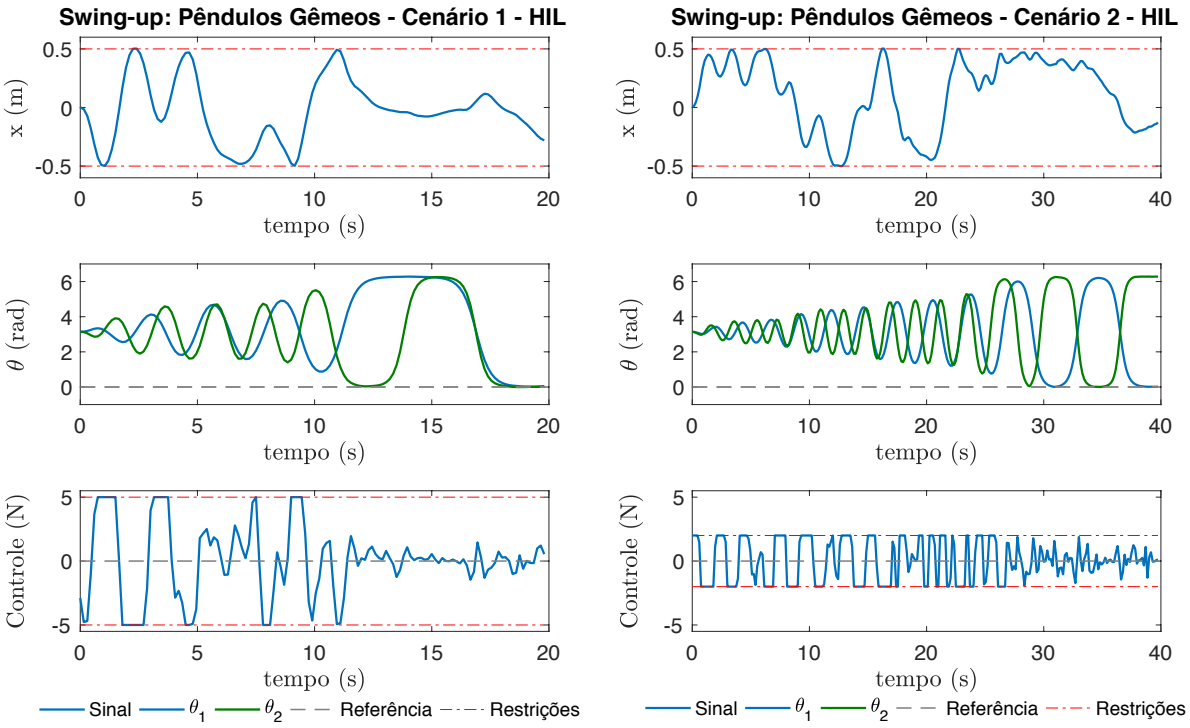



Figura 6.3: Simulações dos pêndulos gêmeos (a) Cenário 1; (b) Cenário 2.

Tabela 6.2: Desempenho de cálculo em diferentes processadores - pêndulos gêmeos

| | Intel Core i7 (2,9 a 3,9 GHz) | | ARM - SoCKit (925 MHz) | | ARM - Raspberry Pi3 (1,2 GHz) | |
|---|---|---|---|---|---|---|
| Configuração | Tempo Médio (ms) | Tempo Máximo (ms) | Tempo Médio (ms) | Tempo Máximo (ms) | Tempo Médio (ms) | Tempo Máximo (ms) |
| S=20, Iter=70 | **5,2** | **6,7** | **97,1** | **108,8** | 315,5 | 317,9 |

gem de 200 $\mu$s (taxa mais alta suportada pela plataforma HIL para este sistema) e foi utilizado o processador ARM na placa SoCKit FPGA com a solução do NMPC-PSO em software em C++. Os sinais de comando são transmitidos via protocolo UDP pela Ethernet diretamente ao *Target PC* enquanto os estados são lidos também via protocolo UDP, porém através da placa BeagleBone Black que tem suas entradas analógicas conectadas às saídas analógicas do *Target PC*. Portanto, apesar de ainda ser um sistema simulado, estão presentes os ruídos das conversões de sinal elétrico DA/AD que ficam presentes na leitura dos estados. Os resultados para os dois cenários são apresentados na Figura 6.4. Neste caso, é possível perceber mais oscilações no sinal de controle devido aos ruídos. Além disso, os pêndulos não ficam sempre estáveis após terem sido invertidos.




(a) (b)

Figura 6.4: Teste dos pêndulos gêmeos na plataforma HIL(a) Cenário 1; (b) Cenário 2.

## 6.2 CONTROLE DE ATITUDE DE SATÉLITE

A seguir, é apresentado um problema com múltiplas entradas e múltiplas saídas (MIMO). Trata-se do controle de atitude de uma plataforma de testes de satélites apresentada inicialmente por Gonzales [132]. Na ocasião, foi utilizado o método SDRE (*State Dependent Riccati Equation*) para projetar um Sistema de Controle de Atitude (SCA) para um simulador de satélites com dinâmica não-linear. Em um trabalho posterior, Rodrigues [131] utiliza a mesma plataforma para implementar um controlador NMPC e demonstra um ganho significativo de desempenho de controle em relação ao SDRE. O intuito de inserir este exemplo aqui é o fato de que em [131], o trabalho foi desenvolvido utilizando o NMPC com um *solver* do tipo *Sequential Quadratic Programming* (SQP) e deseja-se demonstrar tanto a aplicação da abordagem NMPC-PSO para sistemas MIMO quanto suas vantagens em termos de tempo de processamento.

A plataforma em questão é apresentada na Figura 6.5 e é utilizada para simular as atitudes de um satélite em órbita. Os atuadores responsáveis por manter ou modificar a orientação da plataforma são rodas de reação que consistem em motores ligados a um volante de alta inércia, livre para girar em torno de um eixo fixo. São atuadores baseados no princípio da conservação do momento angular onde, em um sistema que esteja na ausência de torques externos, a quantidade de momento angular será conservada e, por tanto, a rotação de um dos atuadores causa uma reação fazendo a plataforma girar em sentido contrário.


Figura 6.5: Plataforma de testes de satélites. [132].

Para descrever o movimento relativo da plataforma, dois sistemas de referência são utilizados. O primeiro é o sistema de referência inercial $F_i(i_1, i_2, i_3)$ localizado no centro do mancal esférico, visível na Figura 6.5. O segundo é o sistema de referência da parte móvel da plataforma, $F_b$, que possui o mesmo centro de $F_i$, porém, não pode ser visto na Figura 6.5. Esta parte móvel é acoplada a partir de um colchão de ar que propicia baixo atrito para os movimentos.

A atitude do simulador a ser controlada é, portanto, a orientação relativa entre os dois sistemas de referência onde se utilizam os ângulos de *Euler* em uma sequência 3-2-1 para descrever as orientações relativas. Neste caso, define-se o ângulo $\theta_1$ para medir a rotação entre os as referências $F_i$ e $F_b$ em torno do eixo $i_3$. Em seguida, o ângulo $\theta_2$ que mede a rotação em relação ao eixo $i_2$ e por fim o ângulo $\theta_3$ para medir a rotação em relação ao eixo $i_1$. A seguir, são descritas as equações diferenciais que representam o modelo dinâmico não-linear do satélite [132]:

$$\dot{\theta}_1 = \omega_2 \left( \frac{\sin\theta_3}{\cos\theta_2} \right) + \omega_3 \left( \frac{\cos\theta_3}{\cos\theta_2} \right), \tag{6.10}$$

$$\dot{\theta}_2 = \omega_2 \cos\theta_3 - \omega_3 \sin\theta_3, \tag{6.11}$$

$$\dot{\theta}_3 = \omega_1 + \omega_2 \left( \frac{\sin\theta_3 \sin\theta_2}{\cos\theta_2} \right) + \omega_3 \left( \frac{\cos\theta_3 \sin\theta_2}{\cos\theta_2} \right), \tag{6.12}$$

$$\dot{\omega}_1 = \omega_2 \left( \frac{I_{22}\omega_3 - I_\omega \Omega_3}{I_{11} + I_\omega} \right) + \omega_3 \left( \frac{-I_{33}\omega_2 + I_\omega \Omega_2}{I_{11} + I_\omega} \right) - \dot{\Omega}_1 \left( \frac{I_\omega}{I_{11} + I_\omega} \right), \tag{6.13}$$

$$\dot{\omega}_2 = \omega_1 \left( \frac{-I_{11}\omega_3 + I_\omega \Omega_3}{I_{22} + I_\omega} \right) + \omega_3 \left( \frac{I_{33}\omega_1 - I_\omega \Omega_1}{I_{22} + I_\omega} \right) - \dot{\Omega}_2 \left( \frac{I_\omega}{I_{22} + I_\omega} \right), \tag{6.14}$$

$$\dot{\omega}_3 = \omega_1 \left( \frac{I_{11}\omega_2 + I_\omega \Omega_2}{I_{33} + I_\omega} \right) + \omega_2 \left( \frac{-I_{22}\omega_1 + I_\omega \Omega_1}{I_{33} + I_\omega} \right) - \dot{\Omega}_3 \left( \frac{I_\omega}{I_{33} + I_\omega} \right), \tag{6.15}$$

onde as entradas do sistema são as acelerações das rodas de reação definidas por $\dot{\Omega}_1$, $\dot{\Omega}_2$ e $\dot{\Omega}_3$ enquanto $\Omega_1$, $\Omega_2$ e $\Omega_3$ são as respectivas velocidades angulares das rodas de reação. Conforme mencionado anteriormente, $\theta_1$, $\theta_2$ e $\theta_3$ são os ângulos que medem a rotação da plataforma enquanto $\omega_1$, $\omega_2$ e $\omega_3$ são as respectivas velocidades angulares e $\dot{\omega}_1$, $\dot{\omega}_2$ e $\dot{\omega}_3$ representam as respectivas acelerações angulares. Já as constantes $I_{11}$, $I_{22}$ e $I_{33}$ representam os momentos de inércia da plataforma em relação aos eixos $i_1$, $i_2$ e $i_3$, respectivamente, e $I_\omega$ é o momento de inércia das rodas de reação (igual para todas as 3). Os parâmetros físicos do sistema possuem os seguintes valores:

$$I_{11} = 1{,}17 \text{ kg.m}^2,\ I_{22} = 1{,}17 \text{ kg.m}^2,\ I_{33} = 2{,}13 \text{ kg.m}^2 \ e\ I_\omega = 1{,}8 \times 10^{-3} \text{ kg.m}^2.$$

O vetor de estados é definido como $x = [\theta_1\ \theta_2;\ \theta_3\ \omega_1\ \omega_2\ \omega_3]^T$ enquanto o vetor de controle é definido por $u = [\dot{\Omega}_1\ \dot{\Omega}_2\ \dot{\Omega}_3]$. Assume-se ainda que todos os estados do sistema são observáveis.

Seguindo para a formulação do controlador não-linear, o cálculo da predição de um passo do sistema foi mais uma vez realizado pelo emprego do método de Runge Kutta de 4ª Ordem descrito pela equação (2.25). Já a função custo segue a mesma formulação utilizada para o pêndulo simples, descrita pela Equação 2.10.

Para utilizar a abordagem NMPC-PSO, seguiu-se o mesmo procedimento anterior, porém com uma adaptação relativa às partículas. Como estamos lidando com um sistema de múltiplas entradas ($n_u = 3$), cada partícula do PSO se transforma em uma matriz $p \in \mathbb{R}^{n_u \times Nc}$. Além disso, as restrições de controle e de variação de controle se tornam os vetores $\mathbf{u}_{min}$, $\mathbf{u}_{max}$ e $\Delta\mathbf{u}_{max}$ de comprimento $n_u$.

Dois cenários de simulação foram considerados, um para ângulos grandes e outro para ângulos pequenos. Em ambos os casos foram adotados os tempos de amostragem, restrições e horizontes de controle e predição descritos em [131]. Porém, os pesos da função custo foram ajustados para o NMPC-PSO. Um resumo dos cenários é apresentado na Tabela 6.3. Além disso, para suavizar o efeito de sobressinal máximo (*overshoot*) em relação aos ângulos, a referência de degrau de entrada foi filtrada a partir da função exponencial descrita na Equação (6.16) a seguir:

$$y = A(1 - e^{\frac{-3\tau.i}{t_r}}), \tag{6.16}$$

onde A corresponde à amplitude do sinal (entrada degrau), $\tau$ é o período de amostragem do sistema e $t_r$ o tempo de resposta desejado para que o sistema atinja 95% da amplitude do degrau. Este procedimento foi adotado para as referências dos três ângulos $\theta_1$, $\theta_2$ e $\theta_3$.

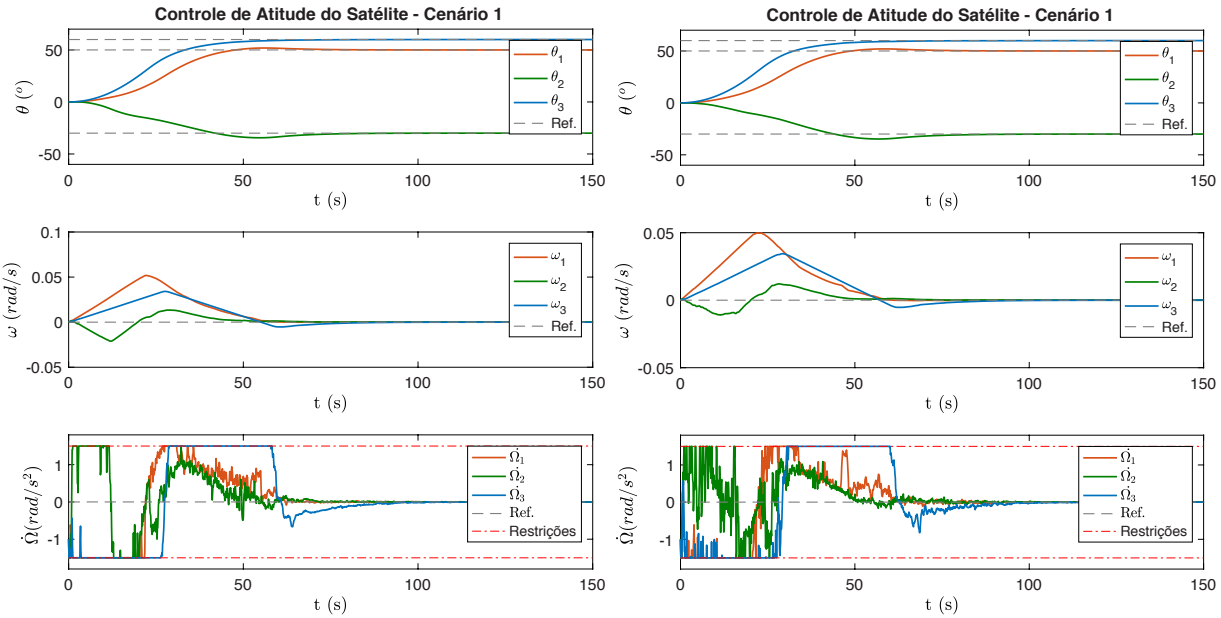
Em seguida, procedeu-se com a simulação de ambos os cenários variando a quantidade de partículas e o número de iterações do PSO. Um aspecto observado foi o de que 15 partículas são o suficiente para percorrer o espaço de buscas e controlar o sistema, porém, também foi observado que um número alto de iterações é necessário para tornar o sinal de controle mais suave. Este fato

Tabela 6.3: Cenários de simulação para o controle do satélite.

| Parâmetro | Cenário 1 [131] | Cenário 2 [131] |
|---|---|---|
| $Ta$ | 100 ms | 100 ms |
| $\mathbf{u}_{min}$ | [-1,5 -1,5 -1,5] rad/s$^2$ | [-1,5 -1,5 -1,5] rad/s$^2$ |
| $\mathbf{u}_{max}$ | [1,5 1,5 1,5] rad/s$^2$ | [1,5 1,5 1,5] rad/s$^2$ |
| $\Delta\mathbf{u}_{max}$ | [1,0 1,0 1,0] rad/s$^3$ | [1,0 1,0 1,0] rad/s$^3$ |
| $N$ | 75 | 75 |
| $Nc$ | 75 | 75 |
| $Q$ | diag($10^2$, $10^2$, $10^2$, 1, 1, 1) | diag($10^2$, $10^2$, $10^2$, 1, 1, 1) |
| $Qf$ | diag(1, 1, 1, 1, 1, 1) | diag(1, 1, 1, 1, 1, 1) |
| $R$ | diag($10^{-2}$, $10^{-2}$, $10^{-2}$) | diag($10^{-2}$, $10^{-2}$, $10^{-2}$) |
| Estado Inicial ($\mathbf{x}_0$) | $[0\ 0\ 0\ 0\ 0\ 0]^T$ | $[0\ 0\ 0\ 0\ 0\ 0]^T$ |
| Referência (degrau filtrado a parir de $k = 0$) | $[50^o\ -30^o\ 60^o\ 0\ 0\ 0]^T$ | $[11^o\ 5^o\ -11^o\ 0\ 0\ 0]^T$ |

decorre de o problema em questão ter um longo horizonte de predição e múltiplas entradas, ou seja, cada partícula em questão tem $Nc \times n_u$ ($75 \times 3 = 225$) variáveis de decisão, o que dificulta a convergência do algoritmo para um sinal de controle mais suave.

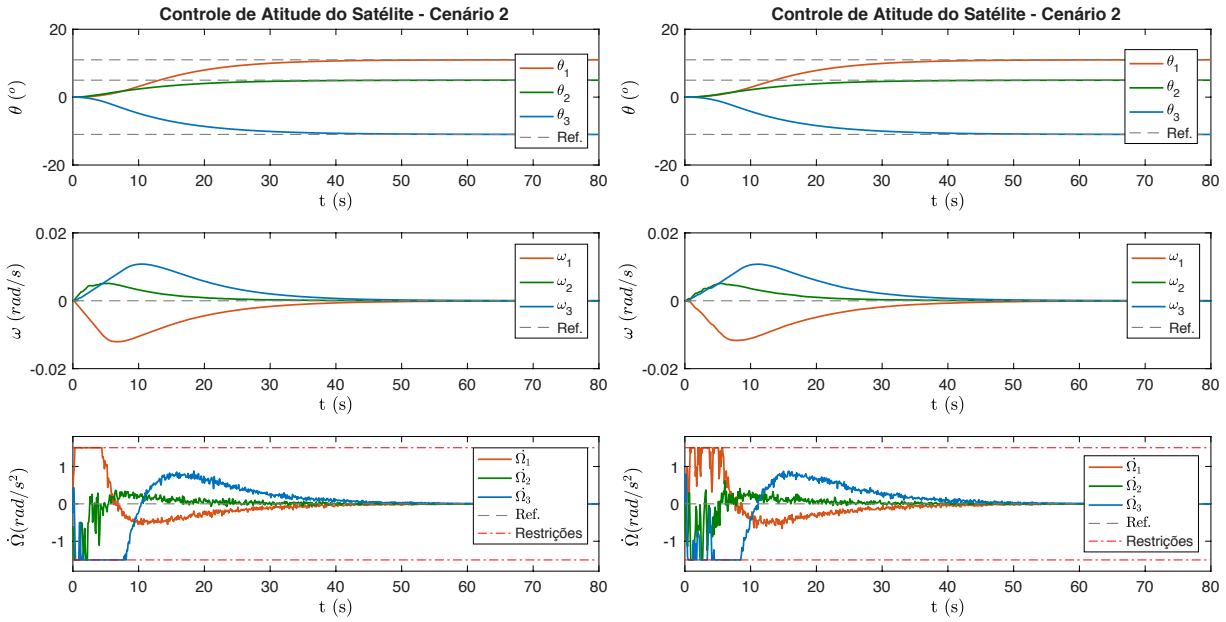
Simulações para ambos os cenários utilizando 50 e 100 iterações são apresentadas na Figuras 6.6 e 6.7.




(a) (b)

Figura 6.6: Simulações do satélite para o cenário 1 (a) $S = 15, MaxIter = 100$; (b) $S = 15, MaxIter = 50$.

Apesar de conseguir controlar o sistema em simulação, esta abordagem tem dois aspectos a serem melhorados. O primeiro, já apontado, são as oscilações no sinal de controle que necessitam um número muito alto de iterações para serem atenuadas e mesmo assim não alcançam um sinal




(a) (b)

Figura 6.7: Simulações do satélite para o cenário 2 (a) $S = 15, MaxIter = 100$; (b) $S = 15, MaxIter = 50$.

suave. O segundo é o tempo de cálculo do algoritmo que, mesmo utilizando 50 iterações ainda fica acima de 140 ms no processador embarcado ARM utilizado na placa SocKit FPGA. Para resolver estas duas questões a abordagem adotada é a de parametrização do sinal de controle apresentada a seguir.

### 6.2.1 Parametrização do Sinal de Controle

Uma abordagem para reduzir a complexidade do problema de otimização não-linear do NMPC é aplicar uma estratégia de parametrização conforme descrito na Seção 2.3.2.2. Neste caso, a formulação utilizada segue a proposta apresentada por Murilo, Alamir e Alberer [20] da utilização de uma combinação linear de funções exponenciais para modelar o sinal de controle do satélite e adotada por [131]. A estratégia é ilustrada na Figura 6.8. Em termos matemáticos é possível definir a parametrização como:

$$u_n(i\tau + t) = Sat_{u_{min}}^{u_{max}}(u_n^* + \alpha_1^{u_n}.e^{-\lambda i\tau} + \alpha_2^{u_n}.e^{-q\lambda i\tau}) \text{ para t } \in \mathbb{R}[(k-1)\tau, k\tau[, \tag{6.17}$$

onde $n \in 0, ..., n_u$ representando o índice do atuador do sistema, $i \in 0, ..., N-1$, $\tau$ é o período de amostragem, $\lambda > 0$, $q \in \mathbb{N}$ são os parâmetros a serem ajustados *offline*, $\alpha_1, \alpha_2 \in \mathbb{R}^2$ são os coeficientes a serem determinados pelo processo de busca e $Sat$ é o mapa de saturação definido

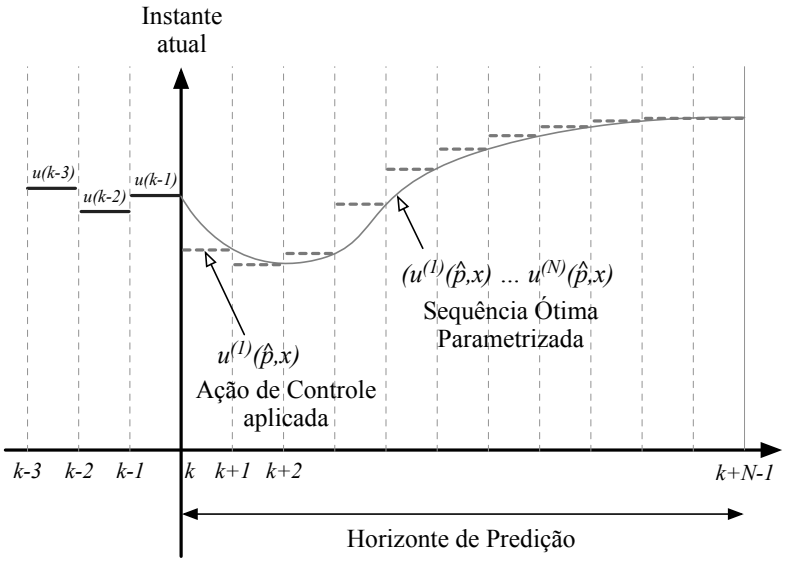



Figura 6.8: Esquemático da estratégia de parametrização aplicada ao sinal de controle do NMPC.

como:

$$Sat_{u_{min}^n}^{u_{max}^n}(u_n) = \begin{cases} u_{max}^n, & \text{se } u_n > u_{max}^n \\ u_{min}^n, & \text{se } u_n < u_{min}^n \\ u_n & \text{caso contrário.} \end{cases} \tag{6.18}$$

Para garantir a continuidade do sinal de controle no instante $k$, é possível substituir $i = 0$ na Equação (6.17) obtendo a seguinte restrição:

$$u_n^* + \alpha_1^{u_n} + \alpha_2^{u_n} = u_n(k-1), \tag{6.19}$$

onde $u_n(k-1)$ é o valor do sinal de controle utilizado no período de amostragem anterior para o atuador $n$. Neste caso, o mapa de saturação $Sat$ foi omitido da equação para simplificar a notação. Seguindo em frente, é possível ainda substituir $i = 1$ na Equação (6.17) e obter:

$$u_n^* + \alpha_1^{u_n}.e^{-\lambda\tau} + \alpha_2^{u_n}.e^{-q\lambda\tau} = u_n(k). \tag{6.20}$$

A diferença entre as Equações (6.19) e (6.20) gera a seguinte expressão:

$$u_n(k) - u_n(k-1) = \alpha_1^{u_n}.(e^{-\lambda\tau} - 1) + \alpha_2^{u_n}.(e^{-q\lambda\tau} - 1) = \delta_{max}^n, \tag{6.21}$$

onde $\delta_{max}^n = u_n(k) - u_n(k-1)$ representa a restrição de variação do sinal de controle do atuador $n$. Como a restrição de variação pode variar entre $-\delta_{max}$ e $+\delta_{max}$ é possível redefinir a Equação (6.21) em função de um parâmetro $p$ tal que $p_j \in [-1, +1]^2$ para $j \in 1, ..., n)u$. Neste caso,

re-escrevendo a Equação (6.21) já para os 3 atuadores do satélite temos:

$$\begin{aligned} p_1.\delta^1_{max} &= \alpha_1^{u_1}.(e^{-\lambda\tau}-1) + \alpha_2^{u_1}.(e^{-q\lambda\tau}-1), \\ p_2.\delta^2_{max} &= \alpha_1^{u_2}.(e^{-\lambda\tau}-1) + \alpha_2^{u_2}.(e^{-q\lambda\tau}-1), \\ p_3.\delta^3_{max} &= \alpha_1^{u_3}.(e^{-\lambda\tau}-1) + \alpha_2^{u_3}.(e^{-q\lambda\tau}-1). \end{aligned} \tag{6.22}$$

Considerando ainda o valor do sinal de controle em estado estacionário $u_n^*$ como uma variável a ser encontrada, define-se ainda $u_1^* = p_4$, $u_2^* = p_5$, $u_3^* = p_6$. Reescrevemos então a Equação (6.19) para os 3 atuadores como:

$$\begin{aligned} u_1(k-1) &= p_4 + \alpha_1^{u_1} + \alpha_2^{u_1}, \\ u_2(k-1) &= p_5 + \alpha_1^{u_2} + \alpha_2^{u_2}, \\ u_3(k-1) &= p_6 + \alpha_1^{u_3} + \alpha_2^{u_3}. \end{aligned} \tag{6.23}$$

Combinando as Equações (6.22) e (6.23) é possível montar um sistema de equações isolando as variáveis $x = \alpha_i^{u_n}$ que se deseja calcular em função das variáveis $p_j$ no formato $x = M^{-1}.b(p)$:

$$\begin{bmatrix} \alpha_1^{u_1}(p) \\ \alpha_2^{u_1}(p) \\ \alpha_1^{u_2}(p) \\ \alpha_2^{u_2}(p) \\ \alpha_1^{u_3}(p) \\ \alpha_2^{u_3}(p) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ K1 & K2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & K1 & K2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & K1 & K2 \end{bmatrix}^{-1} \times \begin{bmatrix} u_1(k-1) - p_4 \\ p_1.\delta^1_{max} \\ u_2(k-1) - p_5 \\ p_2.\delta^2_{max} \\ u_3(k-1) = p_6 \\ p_3.\delta^3_{max} \end{bmatrix}, \tag{6.24}$$

onde, $K1 = e^{-\lambda\tau} - 1$ e $K2 = e^{-q\lambda\tau} - 1$. É importante notar que a matriz M é constante e pode ser calculada *offline* em função dos parâmetros $\tau$, $\lambda$ e $q$ que são ajustados para cada sistema de acordo com o perfil de resposta do atuador.

Finalmente, a Equação (6.25) descreve o sinal de controle de cada atuador do sistema em função dos parâmetros $p_j$ a serem determinados pelo algoritmo de busca:

$$\begin{aligned} u_1(i\tau + t) &= Sat^{u_{max}}_{u_{min}}(p_4 + \alpha_1^{u_1}(p).e^{-\lambda i\tau} + \alpha_2^{u_1}(p).e^{-q\lambda i\tau}), \\ u_2(i\tau + t) &= Sat^{u_{max}}_{u_{min}}(p_5 + \alpha_1^{u_2}(p).e^{-\lambda i\tau} + \alpha_2^{u_2}(p).e^{-q\lambda i\tau}), \\ u_3(i\tau + t) &= Sat^{u_{max}}_{u_{min}}(p_6 + \alpha_1^{u_3}(p).e^{-\lambda i\tau} + \alpha_2^{u_3}(p).e^{-q\lambda i\tau}). \end{aligned} \tag{6.25}$$

Com este procedimento, se reduz o número de graus de liberdade do sistema de $n_u \times N$ ($3 \times 75 = 225$) para os 3 parâmetros $p_4$, $p_5$ e $p_6$, o que restringe o espaço de busca drasticamente. Em seguida, adaptou-se o algoritmo NMPC-PSO para substituir a busca de $\mathbf{u}_n \in [\mathbf{u}_{min}, \mathbf{u}_{max}]$ por $p_j \in [-1, +1]$ . A função custo permanece a mesma, porém, uma passo a mais é adicionado para transformar as partículas $p_j$ nos vetores de controle $\mathbf{u}_1$, $\mathbf{u}_2$ e $\mathbf{u}_3$ correspondentes aos três atuadores ao longo do horizonte de predição $N$, que são as entradas da função custo $J(\mathbf{x}, \mathbf{u})$. Nota-se ainda

que as técnicas de KPSO+SS continuam mantidas para os valores dos parâmetros $p_j$.

### 6.2.2 Resultados de Simulação e Desempenho

Utilizando-se da técnica de parametrização, que reduz o número de parâmetros de busca na etapa de otimização do NMPC-PSO, foi possível reduzir para 4 partículas ($S = 4$) e 20 iterações do algoritmo e obter soluções adequadas, inclusive melhores do que as das simulações anteriores ao uso da parametrização. Os parâmetros fixos foram experimentalmente ajustados em $\lambda = 0,5$ e $q = 8$.

Os resultados destas simulações com a técnica da parametrização para os Cenários 1 e 2 são apresentados na Figura 6.9.

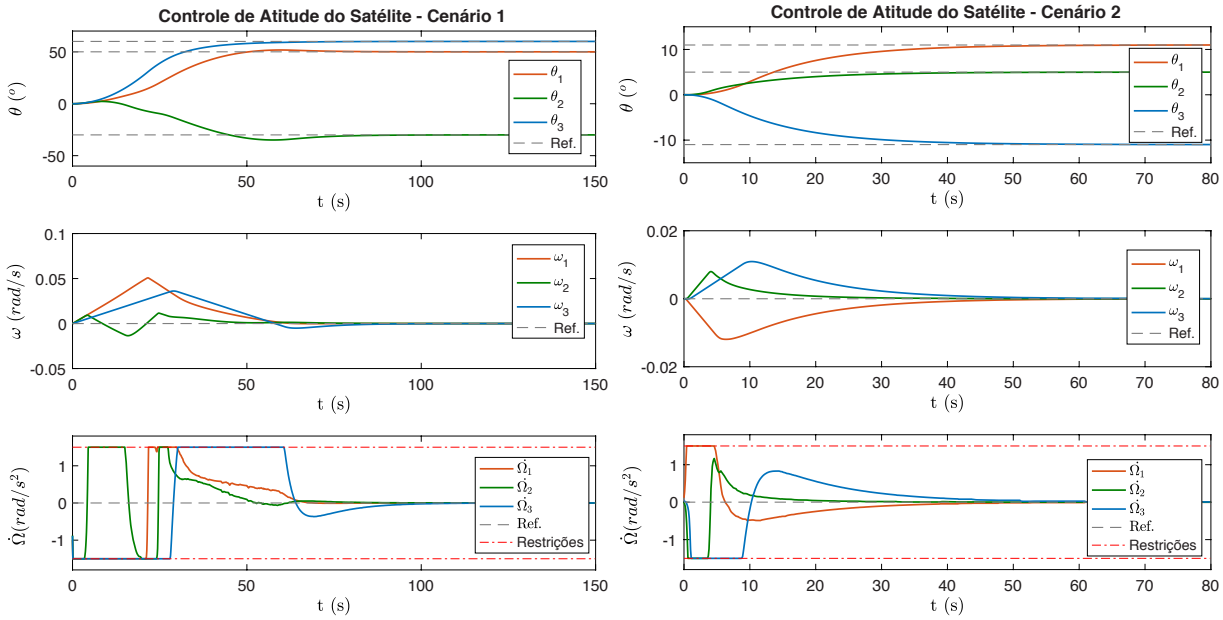


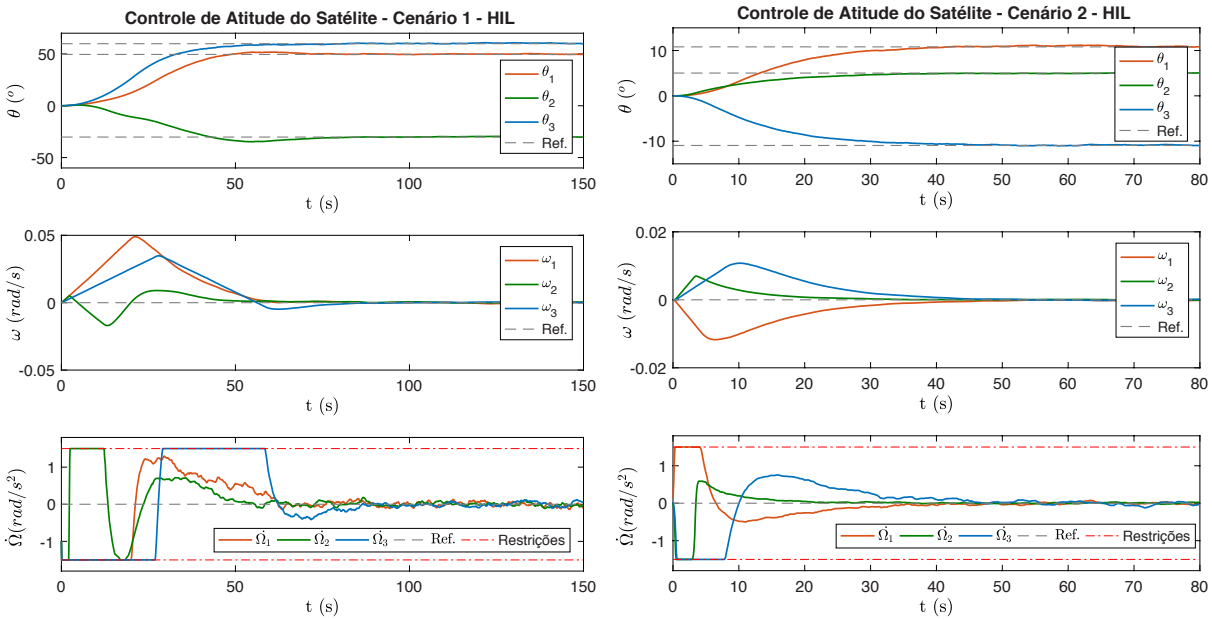

Figura 6.9: Simulações do satélite com a técnica da parametrização (a) Cenário 1; (b) Cenário 2.

Em seguida, a Tabela 6.4 apresenta os resultados de tempo de processamentos para os processadores testados com a solução NMPC-PSO em C++. É possível notar que, sem a técnica de parametrização, nenhum dos processadores ARM é capaz de controlar o sistema em tempo-real. Já com a parametrização, todos são capazes. A título de comparação, destaca-se que Rodrigues obteve um tempo de 66 ms de cálculo da solução SQP utilizando a plataforma BeagleBone Black em [131] , o que é 2,3 vezes mais lento do que a pior solução encontrada utilizando o NMPC-PSO no ARM da Raspberry Pi3.

Finalmente, o sistema do satélite é testado na plataforma HIL com a presença de ruídos na leitura dos estados. Os resultados para os dois cenários são apresentados na Figura 6.10 onde é possível notar as oscilações no sinal de controle devido aos ruídos. Apesar disso, o sistema ainda é controlado de maneira satisfatória atingindo a referência estabelecida.

Tabela 6.4: Comparação de desempenho do PSO entre os processadores - Cenários do satélite.

| Configurações | Intel Core i7 (2,9 a 3,9 GHz) | | ARM - SoCKit (925 MHz) | | ARM - Raspberry Pi3 (1,2 GHz) | |
|---|---|---|---|---|---|---|
| | Tempo Médio (ms) | Tempo Máximo (ms) | Tempo Médio (ms) | Tempo Máximo (ms) | Tempo Médio (ms) | Tempo Máximo (ms) |
| NP: S=15, Iter=50 | **10,1** | **14,2** | 135,9 | 140,5 | 217,4 | 217,6 |
| NP: S=15, Iter=100 | **18,8** | **24,8** | 269,3 | 273,2 | 431,6 | 432,1 |
| P: S=4, Iter=20 | **0,77** | **1,52** | **12,3** | **12,5** | **27,9** | **28,1** |




(a) (b)

Figura 6.10: Teste do satélite na plataforma HIL (a) Cenário 1; (b) Cenário 2.

## 6.3 DISCUSSÕES FINAIS DO CAPÍTULO DE CONTRIBUIÇÕES

Neste capítulo, apresentamos dois sistemas para validar a estratégia NMPC-PSO desenvolvida anteriormente. Em ambos os casos a estratégia KPSO+SS é empregada sem modificações. No primeiro caso o procedimento de *swing-up* de pêndulos gêmeos foi realizado, o que mostra um sistema mais complexo do que o problema original do pêndulo simples.

Em seguida um problema MIMO de controle de um satélite é explorado utilizando inicialmente a abordagem padrão do NMPC-PSO que é capaz de controlar o sistema em simulações, porém não em tempo-real ao utilizar processadores embarcados ARM. Neste caso, a estratégia de parametrização exponencial foi empregada gerando melhores resultados em termos de um sinal de controle mais suave e ainda um tempo de processamento 21,8 vezes mais no ARM da SoCKit e 15,3 vezes mais rápido ao utilizar o ARM da Raspberry Pi3. Além disso, a solução foi, no pior caso, 2,3 vezes mais rápida que a abordagem SQP apresentada por Rodrigues [131].

Ambos os problemas foram testados na plataforma HIL que simula o sistema a uma taxa de

amostragem de 200 $\mu$s, ou seja, acima de 500 vezes mais rápido que a taxa de amostragem do controlador. Além disso, ruídos na leitura dos estados estão presentes e adicionam mais um grau de complexidade ao controle. Em ambos os casos, o desempenho de controle foi satisfatório.

Finalmente, ambos os sistemas puderam ser controlados em tempo-real utilizando a implementação em software do NMPC-PSO. Caso se deseje aumentar o período de amostragem em qualquer um dos casos, sua implementação em hardware pode ser facilmente adaptada a partir da solução apresentada no Capítulo 5.

# 7 CONCLUSÕES

Neste trabalho foram apresentadas duas abordagens de solução do NMPC para aplicação em sistemas de dinâmica rápida. A primeira utilizou técnicas de aprendizado de máquina empregando RNAs e SVMs para criar soluções aproximadas ao comportamento do controlador NMPC. A segunda, utilizou o algoritmo PSO para a etapa de busca da sequência ótima de controle do NMPC.

No primeiro caso, o uso de RNAs e SVMs se mostrou promissor para a obtenção de soluções com altas frequências de amostragem. Na tentativa de generalizar a solução do controlador para um maior número de entradas e perturbações possível, foi observada a necessidade do uso de um histórico de estados bem como referências futuras como entradas do sistema. Além disso, observou-se que os métodos de treinamento que consideram sinais de controle como referências de saída não são o suficiente para obter um controlador estável. Neste caso, o processo de treinamento foi aprimorado através do uso de um simulador interno onde cada solução treinada é testada com um conjunto de referências além dos dados de treinamento para verificação de estabilidade. Soluções em hardware foram desenvolvidas juntamente com uma ferramenta automatizada para a geração das arquiteturas a partir do processo de treinamento. O método foi aplicado a um estudo de caso de controle de posição de um pêndulo invertido obtendo sucesso. A solução de controle é computada em menos de 1 $\mu$s. Apesar disto, a tentativa de aproximar o comportamento do NMPC possui limitações relacionadas à inviabilidade de incorporar e garantir as restrições dos estados do sistema. Além disso, há limitações quanto à diversidade de sinais de referência que um mesmo sistema previamente treinado consegue seguir.

A segunda abordagem, permite a implementação da solução NMPC completa incluindo todos os tipos de restrições. Neste caso, o algoritmo heurístico PSO baseado em inteligência de enxames foi utilizado para resolver o problema de otimização não-linear do NMPC. Adequações foram feitas para que o mesmo atendesse aos requisitos de restrições nos sinais de controle e sua variação no tempo bem como às restrições nos estados do sistema. Além disso, a técnica KPSO foi aplicada e aprimorada a partir da introdução de uma partícula do estado estacionário (KPSO+SS), o que diminui as oscilações do sinal de controle uma vez que o sistema se aproxima da referência desejada. Ajustes nas constantes do PSO foram realizados e um método para ajuste dos pesos da função custo do NMPC foi apresentado.

Uma versão otimizada em C++ do algoritmo NMPC-PSO foi desenvolvida com o foco em processadores embarcados. Em seguida uma versão em hardware foi implementada baseada em aritmética de ponto-flutuante, onde se explorou vários níveis de *pipeline* possíveis no algoritmo para sua aceleração. A solução em hardware foi desenhada de maneira flexível juntamente com uma ferramenta para geração de seu código (VHDL) de modo parametrizado de modo a ser facilmente adaptado a outros sistemas. Esta solução em hardware foi sintetizada em FPGA e validada para o caso de *swing-up* de um pêndulo invertido simulado através de um plataforma

*hardware-in-the-loop* conectada ao controlador em FPGA.

Validações posteriores do algoritmo NMPC-PSO foram realizadas com dois outros estudos de caso: um sistema de pêndulos gêmeos e um sistema de controle de atitude de um satélite. Ambos testados na plataforma HIL. Em todos os casos, atingiu-se um desempenho de controle adequada e tempo de processamento suficiente para o controle embarcado em tempo-real.

Finalmente, observa-se que, apesar das críticas ao fato de algoritmos heurísticos não garantirem uma solução ótima ou mesmo uma solução factível na tarefa de controle, há evidências neste trabalho de que quando bem ajustados tanto em seus parâmetros quanto nos pesos da função custo, são capazes de mitigar estes riscos e fornecer uma solução de controle estável com alto desempenho computacional. Nesta mesma linha de pesquisa, o trabalho de Mesquita [96] utiliza a técnica do KPSO+SS desenvolvida neste trabalho para investigar o emprego de diversos outros algoritmos bio-inspirados ao NMPC, incluindo técnicas algoritmos com abordagens auto-adaptativas.

## 7.1 TRABALHOS FUTUROS

Embora as técnicas e arquiteturas desenvolvidas neste trabalho tenham sido empregadas com sucesso, há muitas vertentes de melhorias e desdobramentos a serem explorados. Algumas das possibilidades são listadas a seguir:

Quanto ao emprego de RNAs e SVMs ao NMPC é possível:

- Explorar maneiras de aplicar restrições aos estados;
- Realizar treinamentos separados para tipos e faixas diferentes de referências de entrada e implementar um chaveamento dinâmico de pesos na arquitetura para aumentar a faixa de operação do sistema treinado.

Quanto ao algoritmo NMPC-PSO é possível:

- Utilizar a estimativa de RNAs/SVMs como chutes iniciais de partículas do NMPC-PSO para um dado sistema, diminuindo ainda mais o tempo de convergência do algoritmo;
- Treinar RNAs/SVMs como estimadores da predição de um passo e empregar uma combinação destas arquiteturas em hardware com a versão do NMPC-PSO em hardware;
- Explorar mais abordagens de parametrização do sinal de controle.

Quanto à arquitetura NMPC-PSO em hardware é possível:

- Estender a otimização de todos os operadores em ponto-flutuante para uma latência de 2 ciclos de *clock* assim como foi feito com o módulo CORDIC para o cálculo de senos e cossenos;

- Verificar o uso de *look-up tables* para substituir o módulo do CORDIC no cálculo de senos e cossenos diminuindo a latência de cálculo;
- Empregar a arquitetura com mais partículas em um FPGA de maior capacidade e com maior frequência para problemas mais complexos com períodos de amostragem menores;

# REFERÊNCIAS BIBLIOGRÁFICAS


1 ÅSTRÖM, K.; WITTENMARK, B. *Adaptive Control: Second Edition*. [S.l.]: Dover Publications, 2013. (Dover Books on Electrical Engineering). ISBN 9780486319148.

2 KIRK, D. *Optimal Control Theory: An Introduction*. [S.l.]: Dover Publications, 2012. (Dover Books on Electrical Engineering). ISBN 9780486135076.

3 MCFARLANE, D. C.; GLOVER, K. *Robust Controller Design Using Normalized Coprime Factor Plant Descriptions*. [S.l.]: Springer-Verlag Berlin Heidelberg, 1990. (Lecture Notes in Control and Information Sciences). ISBN 9783540468288.

4 HARRIS, C.; MOORE, C.; BROWN, M. *Intelligent Control: Aspects of Fuzzy Logic and Neural Nets*. [S.l.]: World Scientific, 1993. (Series in Robotics and Automated Systems). ISBN 9789810210427.

5 ROSSITER, J. A. *Model-Based Predictive Control: A Practical Approach*. [S.l.]: CRC Press, 2003. (Control Series). ISBN 9780203503966.

6 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.

7 QIN, J.; BADGWELL, T. A survey of industrial model predictive control technology. v. 11, p. 733–764, 07 2003.

8 QIN, S. J.; BADGWELL, T. A. An overview of industrial model predictive control technology. In: *Chemical process control—V, Fifth international conference on chemical process control*. [S.l.: s.n.], 1997. p. 232–256.

9 MAYNE, D. Q. Model predictive control: Recent developments and future promise. *Automatica*, v. 50, n. 12, p. 2967–2986, 2014.

10 ALAMIR, M. *Stabilization of Nonlinear Systems Using Receding-horizon Control Schemes: A Parametrized Approach for Fast Systems*. 1. ed. The address: Springer-Verlag London, 2006. v. 1. (10, v. 1). ISBN 978-1-84628-471-7.

11 RAWLINGS, J.; MAYNE, D. *Model Predictive Control: Theory and Design*. [S.l.]: Nob Hill Pub., 2009. ISBN 9780975937709.

12 GRüNE, L.; PANNEK, J. *Nonlinear Model Predictive Control*. [S.l.]: Springer, 2011.

13 ALAMIR, M. *A Pragmatic Story of Model Predictive Control: Self-Contained Algorithms and Case-Studies*. 1st. ed. USA: CreateSpace Independent Publishing Platform, 2013. ISBN 1489541349, 9781489541345.

14 BEMPORAD, A.; MORARI, M.; DUA, V.; PISTIKOPOULOS, E. N. The explicit linear quadratic regulator for constrained systems. *Automatica*, v. 38, n. 1, p. 3–20, 2002.

15 OBERDIECK, R.; PISTIKOPOULOS, E. N. Explicit hybrid model-predictive control: The exact solution. *Automatica*, v. 58, p. 152–159, 2015.

16 PRODAN, I.; ZIO, E.; STOICAN, F. Fault tolerant predictive control design for reliable microgrid energy management under uncertainties. *Energy*, v. 91, p. 20–34, 2015.

17 ZEILINGER, M. N.; RAIMONDO, D. M.; DOMAHIDI, A.; MORARI, M.; JONES, C. N. On real-time robust model predictive control. *Automatica*, v. 50, n. 3, p. 683–694, 2014.

18 ŁAWRYŃCZUK, M. Modelling and predictive control of a neutralisation reactor using sparse support vector machine Wiener models. *Neurocomputing*, v. 205, p. 311–328, 2016. ISSN 18728286.

19 AL-ARAJI, A. S.; ABBOD, M. F.; AL-RAWESHIDY, H. S. Applying posture identifier in designing an adaptive nonlinear predictive controller for nonholonomic mobile robot. *Neurocomputing*, Elsevier, v. 99, p. 543–554, 2013. ISSN 09252312.

20 MURILO, A.; ALAMIR, M.; ALBERER, D. A General NMPC Framework for a Diesel Engine Air Path. *International Journal of Control*, n. January 2015, p. 1–21, 2014. ISSN 0020-7179.

21 BIGDELI, N.; HAERI, M. Predictive functional control for active queue management in congested tcp/ip networks. *{ISA} Transactions*, v. 48, n. 1, p. 107 – 121, 2009. ISSN 0019-0578.

22 MOZAFFARI, A.; VAJEDI, M.; AZAD, N. L. A robust safety-oriented autonomous cruise control scheme for electric vehicles based on model predictive control and online sequential extreme learning machine with a hyper-level fault tolerance-based supervisor. *Neurocomputing*, v. 151, Part 2, p. 845 – 856, 2015. ISSN 0925-2312.

23 XI, X.-C.; POO, A.-N.; CHOU, S.-K. Support vector regression model predictive control on a {HVAC} plant. *Control Engineering Practice*, v. 15, n. 8, p. 897 – 908, 2007. ISSN 0967-0661. Special Section on Modelling and Control for Participatory Planning and Managing Water SystemsIFAC workshop on Modelling and Control for Participatory Planning and Managing Water Systems.

24 SHIN, J.; KIM, H. J.; PARK, S.; KIM, Y. Model predictive flight control using adaptive support vector regression. *Neurocomputing*, v. 73, n. 4–6, p. 1031 – 1037, 2010. ISSN 0925-2312. Bayesian Networks / Design and Application of Neural Networks and Intelligent Learning Systems (KES 2008 / Bio-inspired Computing: Theories and Applications (BIC-TA 2007).

25 GAJSKI, D. D.; ABDI, S.; GERSTLAUER, A.; SCHIRNER, G. *Embedded System Design: Modeling, Synthesis and Verification*. [S.l.]: Springer, 2009. ISBN 9781441905031.

26 GROS, S.; ZANON, M.; QUIRYNEN, R.; BEMPORAD, A.; DIEHL, M. From linear to nonlinear MPC: bridging the gap via the real-time iteration. *International Journal of Control*, v. 7179, n. October, p. 1–19, 2016. ISSN 13665820.

27 ORTEGA, J. G.; CAMACHO, E. F. Mobile robot navigation in a partially structured static environment, using neural predictive control. *Control Engineering Practice*, v. 4, n. 12, p. 1669–1679, 1996.

28 DOMAHIDI, A.; FERREAU, J.; ALMéR, S.; JEREZ, J.; HOVGAARD, T. G. Survey of industrial applications of embedded model predictive control. In: *European Control Conference*. [S.l.: s.n.], 2016.

29 DATTA, A. K.; PATEL, R. Cpu scheduling for power/energy management on multicore processors using cache miss and context switch data. *IEEE Transactions on Parallel and Distributed Systems*, v. 25, n. 5, p. 1190–1199, May 2014. ISSN 1045-9219.

30 SEN, R.; WOOD, D. A. Gpgpu footprint models to estimate per-core power. *IEEE Computer Architecture Letters*, v. 15, n. 2, p. 97–100, July 2016. ISSN 1556-6056.

31 PROJECT TEMPO - Training in Embedded Predictive Control and Optimization. <http://www.itk.ntnu.no/tempo/>. Acessado em: 2018-08-01.

32 MERCIECA, J.; FABRI, S. G. A Metaheuristic Particle Swarm Optimization Approach to Nonlinear Model Predictive Control. *International Journal on Advances in Intelligent Systems*, v. 4, n. 3, p. 357–369, 2012.

33 SANTOS, C. E.; SAMPAIO, R. C.; AYALA, H.; COELHO, L. dos S.; JACOBI, R.; LLANOS, C. H. A SVM optimization tool and FPGA system architecture applied to NMPC. In: *30th Symposium on Integrated Circuits and Systems Design*. Fortaleza, CE, Brazil: [s.n.], 2017.

34 PROPOI, A. I. Use of linear programming mehtos for synthesizing sampled-data automatic systems. *Automatic Remote Control*, v. 27, n. 7, p. 837–844, 1996.

35 RAFAL, M. D.; STEVENS, W. F. Discrete dynamic optimization applied to on- line optimal control. *AIChE Journal*, v. 14, n. 1, p. 85–91, 1968. ISSN 15475905.

36 NIKOLAOU, M. Model predictive controllers: A critical synthesis of theory and industrial needs. *Advances in Chemical Engineering*, v. 26, n. 713, p. 131–204, 2001. ISSN 00652377.

37 ALLGöWER, F.; ZHENG, A. *Nonlinear Model Predictive Control*. [S.l.]: Birkhäuser Basel, 2000. (Progress in Systems and Control Theory). ISBN 9783764362973.

38 RICHALET, J.; RAULT, A.; TESTUD, J.; PAPON, J. Model predictive heuristic control: Applications to industrial processes. *Automatica*, v. 14, n. 5, p. 413 – 428, 1978. ISSN 0005-1098. Disponível em: <http://www.sciencedirect.com/science/article/pii/0005109878900018>.

39 CUTLER, C. R.; RAMAKER, B. L. Dynamic Matrix Control: A Computer Control Algorithm. In: *The 86th National Meeting of the American Instituto of Chemical Engineers*. Houston, TX, EUA: [s.n.], 1979.

40 GARCIA, C. E.; MORSHEDI, A. Quadratic programming solution of dynamic matrix control (qdmc). *Chemical Engineering Communications*, v. 46, n. 1-3, p. 73–87, 1986.

41 KEERTHI, S.; GILBERT, E. Optimal infinite-horizon feedback laws for a general class of constrained discrete-time systems: Stability and moving-horizon approximations. *Journal of Optimization Theory and Applications*, v. 3, 1988.

42 MAYNE, D. Q.; MICHALSKA, H. Receding horizon control of nonlinear systems. *IEEE Transactions on Automatic Control*, v. 35, n. 7, p. 814–824, July 1990. ISSN 0018-9286.

43 FRANCIS, B. A.; WONHAM, W. M. The internal model principle of control theory. *Automatica*, v. 12, n. 5, p. 457–465, 1976. ISSN 00051098.

44 GRECO, C.; MENGA, G.; MOSCA, E.; ZAPPA, G. Performance improvements of self-tuning controllers by multistep horizons: The MUSMAR approach. *Automatica*, v. 20, n. 5, p. 681–699, 1984. ISSN 00051098.

45 CLARKET, D. W.; MOHTADIT, C.; TUFFS˜, P. S. Generalized Predictive Control Algorithm* Part I. The Basic. *Automatica*, v. 23, n. 2, p. 137–148, 1987. ISSN 00051098.

46 OGAWA, M. A. *Controle Preditivo Aplicado ao Seguimento de Trajetória de Robô Móvel com Rodas*. Dissertação (Mestrado) — Universidade Federal do Ceará, Fortaleza, CE, 2014.

47 NIKOLAOU, M. Model predictive controllers: A critical synthesis of theory and industrial needs. In: . [S.l.]: Academic Press, 2001, (Advances in Chemical Engineering, v. 26). p. 131 – 204.

48 MAYNE, D. Q.; RAWLINGS, J. B.; RAO, C. V.; SCOKAERT, P. O. Constrained model predictive control: Stability and optimality. *Automatica*, v. 36, n. 6, p. 789–814, 2000. ISSN 00051098.

49 ŁAWRYŃCZUK, M. Neural Networks in Model Predictive Control. In: *Intelligent Systems for Knowledge Management*. [S.l.]: Springer, 2009. v. 252, n. Cd, p. 31–63. ISBN 9780955301889.

50 BOYD, S.; VANDENBERGHE, L. *Convex Optimization*. Cambridge University Press, 2004. ISBN 9781107394001. Disponível em: <https://books.google.com.br/books?id=IUZdAAAAQBAJ>.

51 ALESSIO, A.; BEMPORAD, A. A survey on explicit model predictive control. In: ____. *Nonlinear Model Predictive Control: Towards New Challenging Applications*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009. p. 345–369. ISBN 978-3-642-01094-1.

52 FERREAU, H. J. *Model Predictive Control Algorithms for Applications with Millisecond Timescales*. Tese (Doutorado) — Katholieke Universiteit Leuven, 2011.

53 FORSGREN, A.; GILL, P. E.; WONG, E. Primal and dual active-set methods for convex quadratic programming. *Mathematical Programming*, v. 159, n. 1, p. 469–508, Sep 2016. ISSN 1436-4646. Disponível em: <https://doi.org/10.1007/s10107-015-0966-2>.

54 FERREAU, H. J. *An Online Active Set Strategy for Fast Solution of Parametric Quadratic Programs with Applications to Predictive Engine Control*. Dissertação (Mestrado) — Ruprecht-Karls-Universität Heidelberg, 2006.

55 FERREAU, H. qpOASES – An Open-Source Implementation of the Online Active Set Strategy for Fast Model Predictive Control. In: *Proceedings of the Workshop on Nonlinear Model Based Control – Software and Applications*. Loughborough, Groot-Britannië: [s.n.], 2007. p. 29–30.

56 BEMPORAD, A. A Quadratic Programming Algorithm Based on Nonnegative Least Squares with Applications to Embedded Model Predictive Control. *IEEE Transactions on Automatic Control*, v. 61, n. 4, p. 1111–1116, 2016. ISSN 00189286.

57 WANG, Y.; BOYD, S. Fast model predictive control using online optimization. *IEEE Transactions on Control Systems Technology*, v. 18, n. 2, p. 267–278, 2010.

58 MATTINGLEY, J.; WANG, Y.; BOYD, S. Code generation for receding horizon control. *2010 IEEE International Symposium on Computer-Aided Control System Design*, p. 985–992, 2010. ISSN 2165-3011.

59 GADE-NIELSEN, N. F.; JØRGENSEN, J. B.; DAMMANN, B. MPC Toolbox with GPU Accelerated Optimization Algorithms. *10th European workshop on advanced control and diagnosis (ACD 2012)*, n. Acd, 2012.

60 GADE-NIELSEN, N. F. *Interior Point Methods on GPU with application to Model Predictive Control*. 1–161 p. Tese (Doutorado) — University of Denmark, 2014.

61 HARTLEY, E.; JEREZ, J.; SUARDI, A.; MACIEJOWSKI, J.; KERRIGAN, E.; CONSTANTINIDES, G. Predictive control using an FPGA with application to aircraft control. *IEEE Transactions on Control Systems Technology*, v. 22, n. 3, p. 1006–1017, 2014.

62 JEREZ, J.; GOULART, P.; RICHTER, S.; CONSTANTINIDES, G.; KERRIGAN, E.; MORARI, M. Embedded predictive control on an FPGA using the fast gradient method. In: *2013 European Control Conference*. Zurich, Switzerland: [s.n.], 2013. p. 3614–3620.

63 JEREZ, J. L.; GOULART, P. J.; RICHTER, S.; CONSTANTINIDES, G.; KERRIGAN, E. C.; MORARI, M. Embedded online optimization for model predictive control at megahertz rates. *IEEE Transactions on Automatic Control*, v. 59, n. 12, p. 3238–3251, 2014.

64 SUARDI, A.; KERRIGAN, E.; CONSTANTINIDES, G. Fast FPGA prototyping toolbox for embedded optimization. In: *2015 European Control Conference*. Linz, Austria: [s.n.], 2015. p. 2589–2594.

65 YU, L.; GOLDSMITH, A.; Di Cairano, S. Efficient Convex Optimization on GPUs for Embedded Model Predictive Control. *Proceedings of the General Purpose GPUs on - GPGPU-10*, p. 12–21, 2017.

66 LIU, C.; WU, H.; FENG, L.; YANG, A. Parallel fourth-order runge-kutta method to solve differential equations. In: ____. *Information Computing and Applications: Second International Conference, ICICA 2011, Qinhuangdao, China, October 28-31, 2011. Proceedings*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011. p. 192–199. ISBN 978-3-642-25255-6.

67 CHEN, H.; SUN, S.; ALIPRANTIS, D. C.; ZAMBRENO, J. Dynamic simulation of electric machines on fpga boards. In: *2009 IEEE International Electric Machines and Drives Conference*. [S.l.: s.n.], 2009. p. 1523–1528.

68 RANA, R. S.; KUMARI, B. FPGA Implementation of Linear Observer. *International Journal of Future Computer and Communication*, v. 3, n. 1, p. 14–17, 2014. ISSN 20103751.

69 VUKOV, M.; GROS, S.; HORN, G.; FRISON, G.; GEEBELEN, K.; JØRGENSEN, J.; SWEVERS, J.; DIEHL, M. Real-time nonlinear MPC and MHE for a large-scale mechatronic application. *Control Engineering Practice*, Elsevier, v. 45, p. 64–78, 2015.

70 QUIRYNEN, R.; VUKOV, M.; ZANON, M.; DIEHL, M. Autogenerating microsecond solvers for nonlinear MPC: A tutorial using ACADO integrators. *Optimal Control Applications and Methods*, v. 36, n. 5, p. 685–704, 2015.

71 KAPERNICK, B.; SUSS, S.; SCHUBERT, E.; GRAICHEN, K. A synthesis strategy for nonlinear model predictive controller on FPGA. In: *2014 UKACC International Conference on Control (CONTROL)*. [S.l.]: IEEE, 2014. p. 662–667. ISBN 978-1-4799-5011-9.

72 ALAMIR, M.; MURILO, A.; AMARI, R.; TONA, P.; FÜRHAPTER, R.; ORTNER, P. On the use of parameterized NMPC in real-time automotive control. *Lecture Notes in Control and Information Sciences*, v. 402, p. 139–149, 2010. ISSN 01708643.

73 DIEHL, M.; BOCK, H. G.; SCHLÖDER, J. P. A Real-Time Iteration Scheme for Nonlinear Optimization in Optimal Feedback Control. *SIAM Journal on Control and Optimization*, v. 43, n. 5, p. 1714–1736, jan 2005. ISSN 0363-0129.

74 OHTSUKA, T. A continuation/GMRES method for fast computation of nonlinear receding horizon control. *Automatica*, v. 40, n. 4, p. 563–574, 2004. ISSN 00051098.

75 GOMEZ-ORTEGA, J.; CAMACHO, E. Neural network mbpc for mobile robot path tracking. *Robotics and Computer-Integrated Manufacturing*, v. 11, n. 4, p. 271 – 278, 1994. ISSN 0736-5845.

76 ÅKESSON, B. M.; TOIVONEN, H. T. A neural network model predictive controller. *Journal of Process Control*, v. 16, n. 9, p. 937–946, 2006. ISSN 09591524.

77 KITTISUPAKORN, P.; THITIYASOOK, P.; HUSSAIN, M. A.; DAOSUD, W. Neural network based model predictive control for a steel pickling process. *Journal of Process Control*, Elsevier Ltd, v. 19, n. 4, p. 579–590, 2009. ISSN 09591524.

78 ŁAWRYŃCZUK, M. Efficient Nonlinear Predictive Control Based on Structured Neural Models. *International Journal of Applied Mathematics and Computer Science*, v. 19, n. 2, p. 233–246, 2009. ISSN 1641-876X.

79 ŁAWRYŃCZUK, M. On improving accuracy of computationally efficient nonlinear predictive control based on neural models. *Chemical Engineering Science*, v. 66, n. 21, p. 5253–5267, 2011. ISSN 00092509.

80 PEREIRA, F. B.; TAVARES, J. *Bio-inspired Algorithms for the Vehicle Routing Problem*. [S.l.]: Springer, 2009.

81 WANG, X.; XIAO, J. Pso-based model predictive control for nonlinear processes. In: ____. *Advances in Natural Computation: First International Conference, ICNC 2005, Changsha, China, August 27-29, 2005, Proceedings, Part II*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005. p. 196–203. ISBN 978-3-540-31858-3.

82 S.SIVANANAITHAPERUMAL; S.BASKAR; G.KALIRAJ. Particle Swarm Optimization Algorithm based Nonlinear Model Predictive Control. In: *Proceedings of the International Conference on Advances in Control and Optimization of Dynamical Systems*. [S.l.: s.n.], 2007. p. 171–175.

83 XU, F.; CHEN, H.; GONG, X.; MEI, Q. 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.

84 SIDDIQUE, N.; ADELI, H. Nature Inspired Computing: An Overview and Some Future Directions. *Cognitive Computation*, Springer US, v. 7, n. 6, p. 706–714, 2015. ISSN 18669964.

85 HORNIK, K.; STINCHCOMBE, M.; WHITE, H. Multilayer feedforward networks are universal approximators. *Neural Networks*, v. 2, n. 5, p. 359–366, 1989. ISSN 08936080.

86 PARK, J.; SANDBERG, I. W. Universal approximation using radial-basis-function networks. *Neural Computation*, v. 3, n. 2, p. 246–257, 1991.

87 HAYKIN, S. *Neural Networks and Learning Machines*. 3rd. ed. [S.l.]: Prentice Hall, 2009. ISBN 9780131471399.

88 MOODY, J.; DARKEN, C. J. Fast learning in networks of locally-tuned processing units. *Neural Computation*, v. 1, n. 2, p. 281–294, 1989.

89 CORTES, C.; VAPNIK, V. Support-Vector Networks. *Machine Learning*, v. 20, n. 3, p. 273–297, 1995. ISSN 15730565.

90 BURGES, C. J. A tutorial on support vector machines for pattern recognition. *Data mining and knowledge discovery*, Springer, v. 2, n. 2, p. 121–167, 1998.

91 LIMA, C. A. M. *Comitê de máquinas: uma abordagem unificada empregando máquinas de vetores suporte*. Tese (Tese) — Universidade Estadual de Campinas-Faculdade de Engenharia Elétrica e de Computação, 2004.

92 THEODORIDIS, S.; KOUTROUMBAS, K. *Pattern Recognition (Fourth Edition)*. Fourth edition. Boston: Academic Press, 2009. 151 - 260 p.

93 DRUCKER, H.; BURGES, C. J.; KAUFMAN, L.; SMOLA, A.; VAPNIK, V. et al. Support vector regression machines. *Advances in neural information processing systems*, Morgan Kaufmann Publishers, v. 9, p. 155–161, 1997.

94 SAMPAIO, R. C.; SANTOS, C. E.; JACOBI, R.; LLANOS, C. H.; COELHO, L. dos S.; AYALA, H. Support vector regression based nonlinear model predictive controlon FPGA. In: *24th ABCM International Congress of Mechanical Engineering - COBEM*. Curitiba, PR, Brazil: [s.n.], 2017.

95 BRABAZON, A.; O'NEILL, M.; MCGARRAGHY, S. *Natural Computing Algorithms*. [S.l.]: Springer, 2015. ISBN 9783662436301.

96 MESQUITA, E. M. *Estudo Comparativo de Metaheurísticas aplicadas ao Controle Preditivo baseado em Modelo*. Dissertação (Mestrado) — Programa de Pós–Graduação em Sistemas Mecatrônicos (PPMEC) – Universidade de Brasília, Brasília, DF, Brazil, 2018.

97 KENNEDY, J.; EBERHART, R. Particle swarm optimization. In: *Proc. IEEE International Conference on Neural Networks*. [S.l.: s.n.], 1995. v. 4, p. 1942–1948.

98 WEISE, T. *Global Optimization Algorithms – Theory and Application*. [S.l.]: Germany: it-weise.de (self-published), 2009.

99 SHI, Y.; EBERHART, R. A modified particle swarm optimizer. In: *Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on*. [S.l.: s.n.], 1998. p. 69–73.

100 SHI, Y.; EBERHART, R. C. Empirical study of particle swarm optimization. In: *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406)*. [S.l.: s.n.], 1999. v. 3, p. 1945–1950.

101 KENNEDY, J. Small worlds and mega-minds: Effects of neighborhood topology on particle swarm performance. *Proceedings of the 1999 Congress on Evolutionary Computation, CEC 1999*, v. 3, p. 1931–1938, 1999.

102 SUGANTHAN, P. N. Particle swarm optimiser with neighbourhood operator. *Proceedings of the 1999 Congress on Evolutionary Computation, CEC 1999*, v. 3, p. 1958–1962, 1999.

103 ZHAN, Z.-H.; ZHANG, J.; LI, Y.; CHUNG, H. S.-H. Adaptive particle swarm optimization. *IEEE transactions on systems, man, and cybernetics. Part B, Cybernetics : a publication of the IEEE Systems, Man, and Cybernetics Society*, v. 39, n. 6, p. 1362–1381, 2009. ISSN 1941-0492.

104 ZHAN, Z. h.; XIAO, J.; ZHANG, J.; CHEN, W. neng. Adaptive control of acceleration coefficients for particle swarm optimization based on clustering analysis. In: *2007 IEEE Congress on Evolutionary Computation*. [S.l.: s.n.], 2007. p. 3276–3282. ISSN 1089-778X.

105 BURKS, A. W.; GOLDSTINE, H. H.; NEUMANN, J. von. *Preliminary discussion of the logical design of an electronic computing instrument*. [S.l.], 1946.

106 WULF, W. A.; MCKEE, S. A. Hitting the memory wall: Implications of the obvious. *SIGARCH Comput. Archit. News*, ACM, New York, NY, USA, v. 23, n. 1, p. 20–24, mar. 1995. ISSN 0163-5964. Disponível em: <http://doi.acm.org/10.1145/216585.216588>.

107 HARTENSTEIN, R. Xputer: the alternative machine paradigm for energy-efficient computing. 2016. Disponível em: <http://hartenstein.de/Energy-efficient_computing_paradigm.pdf>.

108 COORPORATION, I. *Cyclone V Device Overview*. [S.l.], 2018. Disponível em: <https://www.intel.com/content/dam/altera-www/global/en_US/pdfs/literature/hb/cyclone-v/cv_51001.pdf>.

109 XILINX. *Zynq-7000 SoC Data Sheet: Overview*. [S.l.], 2018. Disponível em: <https://www.xilinx.com/support/documentation/data_sheets/ds190-Zynq-7000-Overview.pdf>.

110 NOGUERA, J.; BADIA, R. M. HW / SW Codesign Techniques for Dynamically Reconfigurable Architectures. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, v. 10, n. 4, p. 399–415, 2002. Disponível em: <http://ieeexplore.ieee.org/xpl/login.jsp?tp={&}arnumber=1177337{&}url=http{\%}3A{\%}2F{\%}2Fieeexplore.ieee.org{\%}2Fxpls{\%}2Fabs{_}all.>

111 XILINX, INC. *Virtex-7 FPGAs*. San Jose, CA, 2012.

112 ALTERA CORPORATION. *The Breakthrough Advantage for FPGAs with Tri-Gate Technology*. San Jose, CA, 2013.

113 AYALA, H.; SAMPAIO, R.; MUÑOZ, D. M.; LLANOS, C.; COELHO, L.; JACOBI, R. Nonlinear model predictive control hardware implementation with custom-precision floating point operations. In: *2016 24th Mediterranean Conference on Control and Automation (MED)*. [S.l.: s.n.], 2016. p. 135–140.

114 KHALIL, W.; DOMBRE, E. *Modeling, Identification and Control of Robots*. 3rd. ed. Bristol, PA, USA: Taylor & Francis, Inc., 2002.

115 ZHANG, X.; POLYCARPOU, M. M.; PARISINI, T. Fault diagnosis of a class of nonlinear uncertain systems with Lipschitz nonlinearities using adaptive estimation. *Automatica*, v. 46, n. 2, p. 290–299, 2010.

116 FAN, X.; ARCAK, M. Observer design for systems with multivariable monotone nonlinearities. *Systems and Control Letters*, v. 50, n. 4, p. 319–330, 2003.

117 MIRANDA, P. B.; PRUDENCIO, R. B.; CARVALHO, A. C. P. D.; SOARES, C. Multi-objective optimization and meta-learning for svm parameter selection. In: IEEE. *The 2012 International Joint Conference on Neural Networks (IJCNN)*. Brisbane, QLD, Australia, 2012. p. 1–8.

118 AYALA, H. V. H.; MUÑOZ, D. M.; LLANOS, C. H.; COELHO, L. d. S. Efficient hardware implementation of radial basis function neural network with customized-precision floating-point operations. *Control Engineering Practice*, Elsevier, v. 60, n. September 2015, p. 124–132, 2017. ISSN 09670661.

119 IEEE. *IEEE standard for binary floating-point arithmetic*. NY, 1985.

120 MUÑOZ, D.; SANCHEZ, D.; LLANOS, C.; AYALA-RINCON, M. Tradeoff of FPGA design of floating-point transcendental functions. In: *17th IFIP International Conference on Very Large Scale Integration (VLSI-SoC)*. [S.l.: s.n.], 2009. p. 239–242.

121 MUÑOZ, D.; SANCHEZ, D.; LLANOS, C.; AYALA-RINCON, M. FPGA based floating-point library for cordic algorithms. In: *Programmable Logic Conference (SPL), VI Southern*. Ipojuca, Brazil: [s.n.], 2010. p. 55–60.

122 ALANIZ, A. *Model predictive control with application to real-time hardware and guided parafoil*. Dissertação (Mestrado) — Massachusetts Institute of Technology. Dept. of Aeronautics and Astronautics, 2004.

123 YI, W.; GERASIMOV, I.; KUZMIN, S.; HE, H. An intelligent algorithm of support vector regression parameters optimization in soft measurements. In: IEEE. *Soft Computing and Measurements (SCM), 2016 XIX IEEE International Conference on*. St. Petersburg, Russia, 2016. p. 404–406.

124 COELLO, G. B. L. C. A. C.; VELDHUIZEN, D. A. V. *Evolutionary Algorithms for Solving Multi-Objective Problems*. USA: Springer Science, 2007.

125 SPULER, M.; SARASOLA-SANZ, A.; BIRBAUMER, N.; ROSENSTIEL, W.; RAMOS-MURGUIALDAY, A. Comparing metrics to evaluate performance of regression methods for decoding of neural signals. In: *Proceedings of the Annual International Conference of the IEEE Engineering in Medicine and Biology Society, EMBS*. Milan, Italy: [s.n.], 2015. p. 1083–1086.

126 BOYD, S.; VANDENBERGHE, L. *Convex Optimization*. [S.l.: s.n.], 2010. v. 25. 487–487 p. ISSN 10556788. ISBN 9780521833783.

127 PARSOPOULOS, K.; VRAHATIS, M. Particle swarm optimization method for constrained optimization problem. In: ____. *Frontiers in Artificial Intelligence and Applications*. [S.l.: s.n.], 2002. v. 76, p. 214–220. ISBN 1-58603-256-9.

128 RATNAWEERA, A.; HALGAMUGE, S. K.; WATSON, H. C. Self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients. *IEEE Transactions on Evolutionary Computation*, v. 8, n. 3, p. 240–255, June 2004. ISSN 1089-778X.

129 KHUSAINOV, B.; KERRIGAN, E. C.; SUARDI, A.; CONSTANTINIDES, G. A. Nonlinear Predictive Control on a Heterogeneous Computing Platform. *IFAC-PapersOnLine*, Elsevier B.V., v. 50, n. 1, p. 11877–11882, 2017. ISSN 24058963.

130 MUÑOZ, D. M. *Otimização por Inteligência de enxames usando arquiteturas paralelas para aplicações embarcadas*. 1–192 p. Tese (Doutorado) — Universidade de Brasília, 2012.

131 RODRIGUES, R. S. *Desenvolvimento de Controlador Preditivo para controle de Atitude de Satélites e Validação em HIL*. Dissertação (Mestrado) — Programa de Pós–Graduação em Sistemas Mecatrônicos (PPMEC) – Universidade de Brasília, Brasília, DF, Brazil, 2018.

132 GONZALES, R. G. *Utilização dos métodos SDRE e filtro de Kalman para o controle de atitude de simuladores de satélites*. Dissertação (Mestrado) — Curso de Pós-Graduação em Engenharia e Tecnologia Espacial, INPE - Instituto Nacional de Pesquisas Espaciais, São José dos Campos, SP, Brazil, 2009.

# APÊNDICES

## I.1 TRABALHOS PUBLICADOS EM CONGRESSOS INTERNACIONAIS

- AYALA, H.; SAMPAIO, R.; MUÑOZ, D. M.; LLANOS, C.; COELHO, L.; JACOBI, R. Nonlinear Model Predictive Control Hardware Implementation with Custom-precision Floating Point Operations. 2016.
- SANTOS, C. E.; SAMPAIO, R. C.; AYALA, H.; COELHO, L. dos S.; JACOBI, R.; LLANOS, C. H. A SVM optimization tool and FPGA system architecture applied to NMPC. In: 30th Symposium on Integrated Circuits and Systems Design. Fortaleza, CE, Brazil: [s.n.], 2017.
- SAMPAIO, R. C.; SANTOS, C. E.; JACOBI, R.; LLANOS, C. H.; COELHO, L. dos S.; AYALA, H. Support vector regression based nonlinear model predictive controlon FPGA. In: 24th ABCM International Congress of Mechanical Engineering - COBEM. Curitiba, PR, Brazil: [s.n.], 2017.

## I.2 TRABALHOS SUBMETIDOS PARA JOURNAL INTERNACIONAL

- SAMPAIO, R.; AYALA, H.; COELHO, L.; LLANOS, C.; JACOBI, R. Novel Architectures for Particle Swarm Optimization Implementation on Hardware applied to Nonlinear Model Predictive Control. Submetido a: IEEE Transactions on Industrial Electronics, 2018.

## I.3 PUBLICAÇÕES RELACIONADAS

- BESTARD, GUILLERMO ; SAMPAIO, RENATO ; VARGAS, JOSÉ ; ALFARO, SADEK. Sensor Fusion to Estimate the Depth and Width of the Weld Bead in Real Time in GMAW Processes. SENSORS, v. 18, p. 962, 2018.
- SOUDRE, M. ; CARLOS H. LLANOS ; RENATO C. SAMPAIO . An experimental evaluation of the parallella platform using model-based predictive control as a case study. In: 24th ABCM International Congress of Mechanical Engineering, 2017, Curitiba. Proceedings of the 24th ABCM International Congress of Mechanical Engineering, 2017.
- MESQUITA, E. ; CARLOS H. LLANOS ; RENATO C. SAMPAIO . Bio-inspired optimation applied to the tuning of model predictive control parameters. In: 24th ABCM International Congress of Mechanical Engineering, 2017, Curitiba. Proceedings of the 24th ABCM International Congress of Mechanical Engineering, 2017.