

Tiago Correa Prata

Controle Preditivo Baseado em Modelo (MPC) aplicado a uma planta didática

São Paulo

2020

Tiago Correa Prata

Controle Preditivo Baseado em Modelo (MPC) aplicado a uma planta didática

Projeto de pesquisa apresentado ao Instituto Federal de Educação, Ciência e Tecnologia de São Paulo para a qualificação no programa de mestrado em engenharia de automação e controle.

Instituto Federal de São Paulo - IFSP

Orientador: Prof. Dr. Alexandre Brincalepe Campo

São Paulo

2020

Tiago Correa Prata

Controle Preditivo Baseado em Modelo (MPC) aplicado a uma planta didática

Projeto de pesquisa apresentado ao Instituto Federal de Educação, Ciência e Tecnologia de São Paulo para a qualificação no programa de mestrado em engenharia de automação e controle.

**Prof. Dr. Alexandre Brincalepe
Campo**
Orientador

Prof. Dr. Eduardo Alves da Costa
Convidado (interno)

Prof. Dr. Diego Colón
Convidado (externo)

Prof. Dr. Ricardo Pires
Suplente (interno)

Prof. Dr. Wânderson de Oliveira Assis
Suplente (externo)

São Paulo
2020

*Dedico esse trabalho de pesquisa às minhas avós
Ana Amélia de Oliveira e Valdevina Vidigal Prata,
que na roça, com muito amor e com muito suor,
criaram as bases de duas famílias incríveis.*

Muito obrigado!

Agradecimentos

Os agradecimentos principais são direcionados aos meus pais Aurimar e Jair, por proporcionarem aos filhos o privilégio de poderem estudar sem que nada jamais faltasse; à minha irmã Thais, que apesar dos milhares de quilômetros de distância seu carinho e incentivo para a conclusão desse trabalho nunca cessaram; e a minha companheira Evelyn, pelo apoio e dedicação durante essa trajetória e por compartilhar comigo a experiência de passar por um mestrado. Amo vocês!

Agradecimentos especiais direcionados ao meu orientador Dr. Alexandre Brincalepe Campo que apesar da intensa rotina de sua vida acadêmica aceitou me orientar e me fez indicações valiosas que fizeram toda a diferença; e ao Instituto Federal de Educação, Ciência e Tecnologia de São Paulo, essencial no meu processo de formação profissional.

Resumo

O MPC é uma técnica de controle que vem ganhando atenção ao longo dos últimos 50 anos e, atualmente, ela figura entre uma das técnicas com maior destaque no controle preditivo, tendo inúmeras aplicações comerciais, desde sua implementação no controle de processos multivariáveis em indústrias químicas, até a construção de lógicas para a orientação de veículos autônomos. Esta dissertação trata da implementação de um controle preditivo baseado em modelo, aplicado a uma planta piloto, abordando todo o processo de construção e desenvolvimento do mesmo, visando proporcionar um entendimento claro de cada uma das etapas. Inicialmente é apresentado um levantamento teórico sobre otimização e controle preditivo baseado em modelo e então introduzida a planta objeto de estudo desse trabalho e suas modelagens teóricas e experimentais. A parte prática deste trabalho é desenvolvida principalmente em MATLAB® com o complemento de alguns códigos em Python, e implementada na planta piloto, visando uma fácil replicabilidade. Por fim é apresentado um comparativo entre diferentes controladores MPC utilizando os modelos teóricos e experimentais levantados da planta e a verificação do seu desempenho em comparação com um controlador PID.

Palavras-chave: controle preditivo baseado em modelo; controle preditivo; teoria de controle; identificação de sistemas

Abstract

MPC is a control technique that has been gaining attention and refinement over the last 50 years and it is currently one of the most prominent techniques in predictive control, having numerous commercial applications since its implementation in multivariate chemical process control, to the construction of logics for the orientation of autonomous vehicles. Initially, a theoretical survey on optimization and model predictive control is presented and then the plant that is the object of study in this work and its theoretical and experimental models is introduced. The practical part of this work is developed mainly in **MATLAB®** with the addition of some *Python* code, and implemented in the pilot plant, aiming at an easy replicability. Finally, a comparison is presented between different **MPC** controllers using the theoretical and experimental models raised from the plant and the verification of their performance in comparison with a PID controller.

Keywords: model predictive control; predictive contro; control theory; system identification

Listas de ilustrações

Figura 1 – Pesquisa em grade com número escalar	23
Figura 2 – Pesquisa em grade com duas variáveis e sem restrição	24
Figura 3 – Pesquisa em grade com duas variáveis e com restrição	24
Figura 4 – Influência de $f'(x_k)$ no cálculo de x_{k+1} no método de descidas mais íngremes	25
Figura 5 – Método de descida mais íngrime convergindo para mínimo global	28
Figura 6 – Método de descida mais íngrime convergindo para mínimo local	28
Figura 7 – Diagrama de blocos do quadrado da diferença entre o sistema real e o modelo	29
Figura 8 – Horizonte de tempo analisado pelo método Estimador de Horizonte Móvel	30
Figura 9 – Simulação utilizando Estimador de Horizonte Móvel	34
Figura 10 – Estrutura básica do MPC	35
Figura 11 – Ação do Estimador de Horizonte Móvel e do Controle Preditivo Baseado em Modelo	37
Figura 12 – Planta de aquecimento de ar	39
Figura 13 – Simulação utilizando Controle Preditivo Baseado em Modelo	40
Figura 14 – Laboratório de Controle de Temperatura	42
Figura 15 – Componentes principais do TCLab	43
Figura 16 – Visão geral da Planta Piloto - TCLabSP	44
Figura 17 – Componentes principais do TCLabSP	44
Figura 18 – Diagrama do ambiente de testes do modelo teórico	48
Figura 19 – Respostas real e modelada do Sensor de Temperatura 1	49
Figura 20 – Respostas real e modelada do Sensor de Temperatura 2	50
Figura 21 – Tempo de subida - Sensor de Temperatura 1	52
Figura 22 – Tempo de subida - Sensor de Temperatura 2	53
Figura 23 – Modelo <i>Simulink</i> para coleta de dados	54
Figura 24 – Sinais de entrada nos Aquecedores 1 e 2	55
Figura 25 – Entradas e saídas coletadas em ensaios no TCLabSP	59
Figura 26 – Modelos experimentais de função de transferência	62
Figura 27 – Comparativo dos modelos experimentais	69
Figura 28 – Modelo Simulink do controlador PID	70
Figura 29 – Ambiente Simulink para implementação dos controladores MPC	77
Figura 30 – Resposta do controlador MPC criado a partir do modelo teórico	78
Figura 31 – Resposta do controlador MPC criado a partir do modelo experimental (função de transferência)	78

Figura 32 – Resposta do controlador MPC criado a partir do modelo experimental (ARMAX)	79
Figura 33 – Resposta do controlador MPC criado a partir do modelo experimental (Output-Error)	79
Figura 34 – Resposta do controlador MPC criado a partir do modelo experimental (espeço de estados)	80
Figura 35 – Resposta do controlador MPC criado a partir do modelo experimental (ARX)	80
Figura 36 – Resposta do controlador PID auto-sintonizado	81
Figura 37 – Performance dos controladores MPC e PID agrupados por índice	83
Figura 38 – Resposta do controlador MPC criado empiricamente a partir do modelo experimental (função de transferência)	85
Figura 39 – Performance dos controladores MPC e PID agrupados por índice - Incluindo modelo empírico	86
Figura 40 – Funções de autocovariância dos Sensores de Temperatura 1 e 2	124
Figura 41 – Modelo: função de transferência - Teste de validação	126
Figura 42 – Modelo: função de transferência - Análise de resíduos	127
Figura 43 – Modelos experimentais de espeço de estados	127
Figura 44 – Modelo: espaço de estados - Teste de validação	128
Figura 45 – Modelos experimentais ARX	128
Figura 46 – Modelo: ARX - Teste de validação	129
Figura 47 – Modelo: ARX - Análise de resíduos	129
Figura 48 – Modelos experimentais ARMAX	130
Figura 49 – Modelo: ARMAX - Teste de validação	130
Figura 50 – Modelo: ARMAX - Análise de resíduos	131
Figura 51 – Modelos experimentais OE	131
Figura 52 – Modelo: OE - Teste de validação	132
Figura 53 – Modelo: OE - Análise de resíduos	132
Figura 54 – Modelos experimentais BJ	133
Figura 55 – Modelo: BJ - Teste de validação	133
Figura 56 – Modelo: BJ - Análise de resíduos	134
Figura 57 – MPC Design - Tela principal	135
Figura 58 – MPC Design - Limitantes	136
Figura 59 – MPC Design - Pesos	136
Figura 60 – MPC Design - Cenário	137

Lista de tabelas

Tabela 1 – Componentes principais do TCLab	43
Tabela 2 – Componentes principais do TCLabSP	44
Tabela 3 – Valores para modelagem MIMO do TCLab	45
Tabela 4 – Resultado numérico da comparação entre o modelo teórico e a planta real	49
Tabela 5 – Funções para a estimativa de parâmetros	57
Tabela 6 – Modelos experimentais do TCLabSP	61
Tabela 7 – Modelos experimentais - Função de transferência	61
Tabela 8 – Melhor modelo experimental - Função de transferência	62
Tabela 9 – Modelos experimentais - ARX	63
Tabela 10 – Melhor modelo experimental - ARX	64
Tabela 11 – Modelos experimentais - ARMAX	64
Tabela 12 – Melhor modelo experimental - ARMAX	65
Tabela 13 – Modelos experimentais - OE	66
Tabela 14 – Melhor modelo experimental - OE	66
Tabela 15 – Modelos experimentais - BJ	67
Tabela 16 – Melhor modelo experimental - BJ	68
Tabela 17 – Qualidade dos modelos experimentais	68
Tabela 18 – Sintonia dos blocos PID	71
Tabela 19 – Performance dos controladores MPC e PID - IAE	82
Tabela 20 – Performance dos controladores MPC e PID - ISE	82
Tabela 21 – Performance dos controladores MPC e PID - ITAE	82
Tabela 22 – Performance dos controladores MPC e PID - ITSE	83
Tabela 23 – Mínimos das funções de autocovariância	124
Tabela 24 – Efeito do Δ no tempo de amostragem	125
Tabela 25 – Performance dos controladores MPC e PID - IAE	138
Tabela 26 – Performance dos controladores MPC e PID - ISE	138
Tabela 27 – Performance dos controladores MPC e PID - ITAE	138
Tabela 28 – Performance dos controladores MPC e PID - ITSE	139

Listas de códigos-fonte

2.1	Busca por descidas mais íngremes	26
5.1	Criação dos conjuntos de identificação e validação para o sistema TCLabSP	59
6.1	Criação dos controladores MPC	74
A.1	Pesquisa em grade com número escalar	92
A.2	Pesquisa em grade com duas variáveis	93
A.3	Exemplo do método Estimador de Horizonte Móvel	94
A.4	Exemplo de aplicação do Controle Preditivo Baseado em Modelo	101
A.5	Criação de diversos modelos experimentais do TCLabSP	108
A.6	Calcula tempo de subida de primeira ordem do TCLabSP	116
A.7	Função auxiliar: Combina experimentos do TCLabSP	117
A.8	Função auxiliar: Calcula média de uma região	118
A.9	Plota tempo de subida	119
A.10	Encontra o Delta e o Ts	120
A.11	Função auxiliar: Encontra Delta	120
A.12	Função auxiliar: Calcula autocovariância	121
A.13	Plota autocovariância	122

Lista de abreviaturas e siglas

AIC	Critério de informação de Akaike
AR	Modelo autorregressivo
ARMAX	Modelo autorregressivo com média móvel e entrada exógena
ARX	Modelo autorregressivo com entrada exógena
BJ	Box-Jenkins
BYU	<i>Brigham Young University</i>
DMC	<i>Dynamic Matrix Control</i>
GBN	<i>Generalized Binary Noise</i>
GPC	<i>Generalized Predictive Controle</i>
IAE	<i>Integral Absolute Error</i>
ISE	<i>Integral Square Error</i>
ITAE	<i>Integral Time Absolute Error</i>
ITSE	<i>Integral Time Square Error</i>
MATLAB®	<i>Matrix Laboratory</i>
MHE	Estimador de Horizonte Móvel
MHPC	<i>Model Heuristic Predictive Control</i>
MIMO	<i>Multiple Input Multiple Output</i>
MPC	Controle Preditivo Baseado em Modelo
NCSU	<i>North Carolina State University</i>
NLP	<i>Nonlinear Programming</i>
OE	<i>Output-Error</i>
PID	Proporcional Integral Derivativo
PRBS	<i>Pseudo-Random Binary Signal</i>
s.a.	sujeito a
SI	Sistema Internacional de Unidades
SS	Espaço de estados
STR	<i>Self-Tuning Regulator</i>
TCLab	<i>Temperature Control Lab</i>
TCLabSP	<i>Temperature Control Lab Sao Paulo</i>
TF	Função de transferência

Sumário

I	APRESENTAÇÃO	15
1	INTRODUÇÃO	16
1.1	Introdução histórica ao MPC	17
1.2	Objetivos	18
1.3	Motivação e justificativas	18
II	REVISÃO DE LITERATURA	20
2	OTIMIZAÇÃO	21
2.1	O problema da otimização	21
2.2	Algoritmos de otimização	22
2.2.1	Método de pesquisa em grade	22
2.2.2	Método de busca de descidas mais íngremes	24
2.2.3	Otimizadores de programação não-linear (NLP)	28
2.3	Algumas aplicações de otimização	29
2.3.1	Estimação de parâmetros	29
2.3.2	Estimador de Horizonte Móvel (MHE)	30
2.3.2.1	Exemplo	33
3	CONTROLE PREDITIVO BASEADO EM MODELO	35
3.1	Aplicação prática	36
3.1.1	Exemplo	38
III	MATERIAIS E MÉTODOS	41
4	PLANTA PILOTO	42
4.1	TCLab	42
4.2	Planta piloto - TCLabSP	43
4.3	Modelagem teórica da planta piloto	43
4.3.1	Verificação do modelo teórico	48
5	MODELAGEM EXPERIMENTAL	51
5.1	Testes dinâmicos e coleta de dados	51
5.1.1	Período de amostragem	51
5.1.2	Sinais de excitação	53

5.1.3	Duração do experimento	54
5.2	Definição do modelo experimental	55
5.2.1	Escolha da representação matemática	55
5.2.2	Determinação da estrutura do modelos	56
5.2.3	Estimação de parâmetros	57
5.2.4	Validação do modelo	57
5.3	Modelos experimentais	58
5.3.1	Comparativo	67
6	DESENVOLVIMENTO DO CONTROLADOR	70
6.1	Controlador PID	70
6.2	Controlador MPC	71
6.2.1	Horizonte de predição	71
6.2.2	Horizonte de controle	73
6.2.3	Matrizes de peso	73
6.2.4	Desenvolvimento do controlador no MATLAB	74
IV	RESULTADOS E DISCUSSÕES	76
7	RESULTADOS	77
8	DISCUSSÕES	87
	REFERÊNCIAS	88
	APÊNDICES	91
	APÊNDICE A – CÓDIGOS-FONTE	92
A.1	Método de pesquisa em grade	92
A.2	Método de pesquisa em grade com duas variáveis	93
A.3	Exemplo do método Estimador de Horizonte Móvel	94
A.4	Exemplo de aplicação do Controle Preditivo Baseado em Modelo	101
A.5	Criação de diversos modelos experimentais do TCLabSP	108
A.6	Identificação do tempo de subida do TCLabSP	116
A.7	Encontrar Delta amostral utilizando autocovariância	120
	APÊNDICE B – ESCOLHA DA FREQUÊNCIA DE AMOSTRAGEM UTILIZANDO AUTOCOVARIÂNCIA	123
B.1	Aplicação prática	124

APÊNDICE C – GRÁFICOS DOS TESTES DE VALIDAÇÃO DOS MODELOS EXPERIMENTAIS	126
APÊNDICE D – MPC EXPERIMENTAL EMPÍRICO	135
D.1 Telas do <i>MPC Design</i>	135
D.2 Tabelas de performance	138
D.3 Sintonia	140

Parte I

Apresenta $\tilde{\text{a}}$ o

1 Introdução

O controle de processos tem fundamental importância no desenvolvimento industrial, sendo amplamente utilizado em praticamente todos os segmentos da indústria, contribuindo de maneira significativa para a maior velocidade na estabilização de sinais, aumento da qualidade de produtos, diminuição de riscos e redução de custos operacionais (OGATA, 2010). Seu objetivo, de forma simplificada, consiste em avaliar e corrigir desvios entre um valor desejado e o real valor medido na saída da planta para uma dada variável do processo (ou variáveis, nos casos de processos com múltiplas entradas e múltiplas saídas, também conhecidos por sua sigla em inglês **MIMO** (*Multiple Input Multiple Output*))). A aplicação correta de estratégias de controle acarreta numa operação eficiente da planta, mantendo suas variáveis relevantes em condições próximas às desejadas. A sintonia bem-feita do controle auxilia também na otimização do processo, possibilitando que o sistema opere com menor variabilidade, maximizando a produção e minimizando a utilização de recursos (OGATA, 2010). No [capítulo 2](#) abordaremos mais sobre otimização. Modelos matemáticos podem auxiliar a estratégia de controle uma vez que um modelo da planta ou processo pode ser utilizado para estabelecer a relação existente entre as variáveis manipuladas e variáveis controladas, assim podendo auxiliar na predição do comportamento dinâmico do sistema analisado (GARCIA, 2013). A modelagem matemática pode ser feita utilizando dados empíricos ou através da aplicação de relações físico-químicas.

A estratégia de controle predominante na indústria é o controle **PID** (Proporcional Integral Derivativo) (CAMACHO; ALBA; BORDONS, 2007) que, além de levar em consideração o efeito proporcional (P) do erro medido, também atua em desvios relativos aos efeitos integrais (I) e derivativos (D). Seu elevado número de aplicações deve-se a uma grande variedade de vantagens como: sua rápida implementação, facilidade de compreensão, disponibilidade em praticamente todas as plataformas industriais de controle e principalmente pelo fato de não requerer um modelo matemático do processo (GONÇALVES, 2012); porém apesar de poder ser aplicado com eficiência em muitos processos, o controle **PID** aparece com menos frequência em sistemas não-lineares, como em plantas de controle de pH, por exemplo (GONÇALVES, 2012). Em casos como esse, outra técnica bastante utilizada na indústria (porém em proporções bem menores que o **PID**) pode ser utilizada: o controle **MPC** (Controle Preditivo Baseado em Modelo, do inglês *Model Predictive Control*). Uma introdução histórica desta técnica é apresentada na seção a seguir e mais detalhes técnicos sobre ela serão apresentados no [capítulo 3](#).

1.1 Introdução histórica ao MPC

Segundo Lee (2011) em meados dos anos 50 as características essenciais do MPC já podiam ser observadas nas primeiras instalações de supervisórios de controle computadorizados, porém, apesar de seus potenciais benefícios, esse tipo de controle não se difundiu muito devido aos esforços necessários para mantê-lo e ao seu alto valor, até que aproximadamente na metade dos anos 70 os microprocessadores e sistemas de controle distribuídos se tornassem mais baratos e confiáveis. Não coincidentemente, por volta desta época começaram a aparecer na indústria e em seminários, publicações relatando a aplicação bem sucedida de controle baseado em modelo. (LEE, 2011)

Lee (2011) também relata que ainda durante os anos 70 a publicação das técnicas *Model Heuristic Predictive Control* (MHPC - Controle Preditivo Heurístico de Modelo) e *Dynamic Matrix Control* (DMC - Controle Dinâmico de Matriz) mostrando grande sucesso prático, impulsionaram o uso delas e de técnicas similares em refinarias de todo o mundo ocidental. No geral os algoritmos por trás dessas técnicas apresentavam uma natureza heurística, empregavam modelos baseados em resposta de domínio no tempo, eram completamente determinísticos sem nenhum modelo explícito dos distúrbios, e não possuíam garantias de estabilidade nem métodos para sintonia. Na mesma época, porém independente dos desenvolvimentos na indústria de processos, surgia da comunidade de controle adaptativo o GPC (Controle Preditivo Generalizado, do inglês *Generalized Predictive Control*), que possuía motivações bem diferentes do DMC: o GPC pretendia oferecer uma nova alternativa ao regulador de autoajuste¹, principalmente visando superar seu problema de robustez. Por causa disso ele aplicava problemas de controle multivariável, porém carecia na inclusão de restrições, que era considerada uma característica indispensável dos problemas de controle de processos.

Ao longo dos anos 80 os algoritmos MPC se espalharam comercialmente a o embasamento teórico da técnica começou a se consolidar. Nessa época muitas empresas (normalmente pequenas), que ofereciam comercialmente soluções utilizando controle MPC, foram adquiridas por empresas maiores como a Aspen Tech e a Honeywell (LEE, 2011). No campo teórico aumentaram as discussões e definições com relação à estabilidade, à robustez e ao uso de modelos não-lineares. Além disso os pesquisadores notaram que a utilização de modelos de espaço de estado poderia trazer mais benefícios e juntamente com isso estabeleceram a forma padrão do algoritmo, demonstrada na eq. (1.1) (LEE, 2011) .

$$\min_{\hat{u}(0), \dots, \hat{u}(p-1)} \left\{ V_p \triangleq \sum_{i=0}^{p-1} \left(\hat{x}^T(i) Q \hat{x}(i) + \hat{u}^T(i) R \hat{u}(i) \right) + \hat{x}^T(p) Q_t \hat{x}(p) \right\} \quad (1.1)$$

¹ O regulador de autoajuste (ou apenas STR, do inglês, *Self-Tuning Regulator*) é uma técnica composta por três partes, um estimador de parâmetros, um cálculo de projeto e um regulador com parâmetros ajustáveis (ÅSTRÖM; WITTENMARK, 1985).

Sendo:

$$\hat{x}(i+1) = A\hat{x}(i) + B\hat{u}, \quad \hat{x}(0) = x_0$$

Ao longo das últimas décadas, desde o lançamento do DMC, o MPC tem sido alvo de muito estudo teórico e prático, e passou de uma aplicação industrial heurística para uma das técnicas de controle mais influentes da atualidade (LEE, 2011). Hoje podemos considerar o MPC, segundo a definição de Seborg, Edgar e Mellichamp (2011), uma técnica de controle avançada para problemas de controle multivariável. Sendo que, mesmo para um processo de múltiplas entradas e múltiplas saídas, onde existem restrições a essas variáveis, é razoável assumir que possuindo o modelo dinâmico do processo e as medições atuais do mesmo, pode-se predizer os valores de saída futuros, e então calcular as mudanças nos valores de entrada baseando-se tanto nas previsões quanto no valor medido.

Após enormes avanços nas técnicas de resolução das equações aplicadas ao MPC, atualmente seu uso não se limita mais a sistemas onde o tempo de resposta seja relativamente lento, podendo então ser utilizado em diversas aplicações que eram consideradas impraticáveis no passado. Lee (2011) relata em seu artigo o uso do MPC no controle de tração de veículos, motorização automotiva, amortecimento de sistemas massa-mola magneticamente acionados e muitos outros.

1.2 Objetivos

Este trabalho propõe, como objetivo principal, desenvolver um controlador MPC aplicado a um sistema didático.

Além disso os seguintes objetivos específicos também serão realizados:

- Estudo do algoritmo do controle MPC
- Avaliação o desempenho do controlador MPC desenvolvido em comparação com um controlador PID
- Compilação de material teórico e experimental sobre MPC para estudantes de graduação e pós-graduação, de língua portuguesa

1.3 Motivação e justificativas

Segundo Parkinson, Balling e Hedengren (2018) o processo de determinar o melhor modelo para uma aplicação ou processo é chamado de otimização. Normalmente engenheiros costumam implementar tais técnicas em seus processos visando aumentar a eficiência e diminuindo os gastos, porém, as variáveis e limitantes do processo podem ser inúmeras, fazendo com que a tarefa de otimização se torne difícil. Para casos assim, ferramentas

computacionais de otimização são essenciais. (PARKINSON; BALLING; HEDENGREN, 2018)

Dá-se o nome de Otimização Dinâmica ao processo de otimização que é realizado dinamicamente ao longo do processo e, segundo Borrelli, Bemporad e Morari (2017), esta se tornou uma ferramenta padrão na tomada de decisões numa grande variedade de áreas. O controle MPC é um modo de implementação da otimização dinâmica e a execução deste trabalho em torno dessa técnica se deve ao fato de que ao longo das últimas quase 4 décadas (LEE, 2011) ela evoluiu para dominar a indústria de processos, onde tem sido utilizada em milhares de problemas (BORRELLI; BEMPORAD; MORARI, 2017). Além de exemplos de utilização de técnica já apresentados na seção 1.1, é importante destacar também sua utilização no controle dos processos de fabricação de cimento, torres de destilação, plantas de PVC, como descreve Camacho, Alba e Bordons (2007), e também seu crescimento em outros setores, como na indústria automobilista, onde o MPC é utilizado para o controle do sistema dinâmico de um automóvel. (YAKUB; MORI, 2013).

Parte II

Revisão de literatura

2 Otimização

2.1 O problema da otimização

Segundo Haugen (2018) normalmente problemas de otimização são apresentados como problemas de minimização, como: "Encontre o valor ótimo de x que minimize a função objetivo $f(x)$, levando em consideração qualquer restrição sobre x ou em função de x . A solução ótima é indicada por x_{opt} " (HAUGEN, 2018).

Haugen (2018) ainda mostra que há várias formas de formular matematicamente um problema de otimização (minimização), mas que, de forma geral, dado um modelo matemático M , é possível representá-lo como a minimização de x para uma função $f(x)$, ou seja:

$$\min_x f(x) \quad (2.1)$$

sujeito a (também denotado por "s.a.") restrições, que podem ser na forma de:

- Restrições de desigualdade:

$$g(x) \leq 0 \quad (2.2)$$

onde g pode ser uma função linear ou não-linear.

- Restrições de igualdade:

$$h(x) = 0 \quad (2.3)$$

onde h pode ser uma função linear ou não-linear de x .

- Limites superiores e inferiores

$$x_{li} \leq x \leq x_{ls} \quad (2.4)$$

Onde li e ls indicam 'limite inferior' e 'limite superior', respectivamente.

Sendo que as equações 2.2 e 2.3 definem restrições na relação entre as variáveis de otimização, enquanto 2.4 define as regiões limites destas mesmas variáveis.

Existem diversos métodos para encontrar a solução ótima para um problema de otimização e a seção a seguir irá mostrar exemplos e métodos numéricos simples para exemplificar, em maiores detalhes, como um problema de minimização pode ser resolvido. No capítulo 3 será feito uso das minimizações para compreender como o MPC calcula valores ótimos, dadas determinadas restrições em um dado horizonte de controle, pois

uma maior compreensão sobre problemas de minimização pode fazer grande diferença no entendimento do controle MPC em si.

2.2 Algoritmos de otimização

2.2.1 Método de pesquisa em grade

O método apresentado nesta seção não é aplicável a praticamente nenhum problema real devido à sua ineficiência computacional, porém ele ajuda a ilustrar o objetivo de todo o método numérico voltado para minimização.

Este método consiste em testar todos os valores de todas as variáveis (em um conjunto de dados definido) para verificar qual combinação minimiza a função objetivo, ou seja, testar todos os valores possíveis para $x_1, x_2, x_3, \dots, x_n$ com o objetivo de encontrar o x_{opt} , valor de x que minimiza f .

No caso de um único x , um laço condicional simples poderia testar todos os valores da função objetivo. Para ilustrar essa ideia, no [apêndice A](#) o código-fonte [A.1](#), em Python, mostra como a função $f(x)$ abaixo poderia ser computada, caso o intervalo de teste de x fosse igual 100, ou seja, $N = 100$.

$$f(x) = 0,00232x^4 - 0,111x^3 + 1,8x^2 - 11,6x + 34,4 \quad (2.5)$$

Sendo que:

$$2 \leq x \leq 22$$

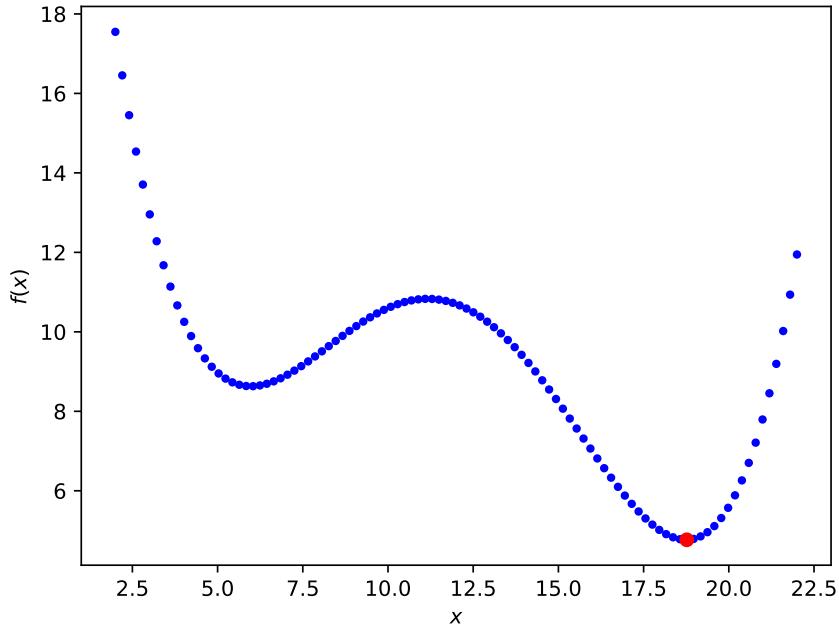
O intervalo $N = 100$ indica que serão analisados 100 valores entre 2 e 22.

A [fig. 1](#) foi construída a partir do código-fonte [A.1](#), sendo destacado em vermelho, o valor de x onde a $f(x)$ apresentava seu menor valor. Repare que o gráfico apresenta dois vales distintos: um deles é o já mencionado destaque em vermelho, onde o valor x é 18,76 e outro onde x vale aproximadamente 5,5. O vale do gráfico onde o valor de x produz o menor valor de $f(x)$ é conhecido como *mínimo global*, todos os outros são *mínimos locais*, pois são os valores mínimos da função apenas para uma região limitada.

Algoritmos que buscam encontrar o valor mínimo de uma função podem erroneamente convergir para mínimos locais. O método de pesquisa em grade não é um desses algoritmos, pois ao verificar todos os valores de x ele sempre encontrará o valor de x que minimiza a função objetivo, porém nas próximas sessões serão apresentados métodos que, apesar de serem mais eficientes computacionalmente, podem tender para mínimos locais.

Ainda neste método, casos onde há uma maior quantidade de variáveis (x_1, x_2, \dots) deve-se utilizar laços aninhados para que a varredura de todas as possibilidades possa ser

Figura 1 – Pesquisa em grade com número escalar



Fonte: Autor, adaptado de Haugen (2018)

feita.

Como exemplo, minimizemos a eq. (2.6), sendo $0 \leq x_1 \leq 2$ e $1 \leq x_2 \leq 3$.

$$f(x) = (x_1 - 1)^2 + (x_2 - 2)^2 + 0,5 \quad (2.6)$$

Neste caso, de forma direta nota-se que $f_{min} = 0,5$, $x_{1_{opt}} = 1$ e $x_{2_{opt}} = 2$.¹ Porém se a restrição da eq. (2.7) for aplicada novos valores são encontrados, descritos nas equações presentes em eq. (2.8).

$$f(x) = x_1 - x_2 + 1,5 \leq 0 \quad (2.7)$$

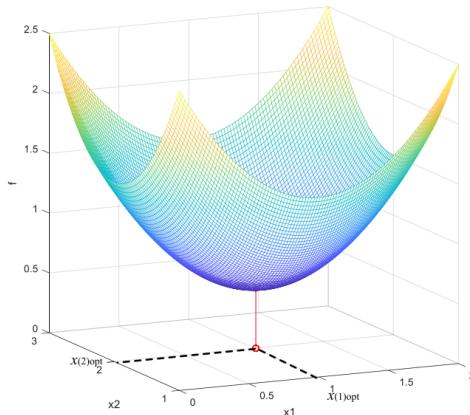
$$\begin{aligned} f_{min} &= 0,628 \\ x_{1_{opt}} &= 0,748 \\ x_{2_{opt}} &= 2,25 \end{aligned} \quad (2.8)$$

O motivo de os valores de $x_{1_{opt}}$ e de $x_{2_{opt}}$ serem diferentes quando a restrição da eq. (2.7) é aplicada pode ser visualmente observado nas figuras 2 e 3 onde elas mostram

¹ O código-fonte em Python para este cálculo pode ser encontrado no apêndice A, código-fonte A.2.

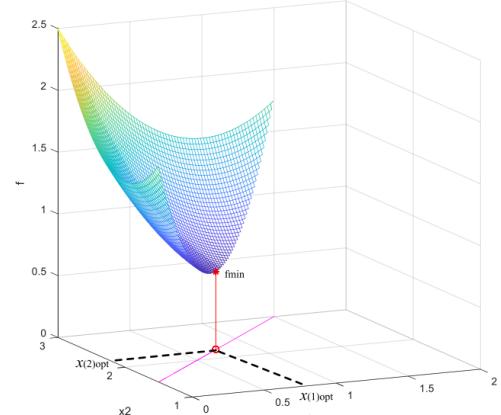
uma alteração do ponto mínimo da função custo (outra forma de chamarmos a função objetivo) devido à redução do conjunto imagem de $f(x)$.

Figura 2 – Pesquisa em grade com duas variáveis e sem restrição



Fonte: Haugen (2018)

Figura 3 – Pesquisa em grade com duas variáveis e com restrição



Fonte: Haugen (2018)

2.2.2 Método de busca de descidas mais íngremes

Tal qual o método de pesquisa em grade, a maioria dos outros algoritmos de otimização consiste em testar valores de x e indicar qual deles retorna o menor $f(x)$, porém, diferentemente do método anterior, a técnica apresentada nesta seção não testa todos os valores possíveis de x , na realidade ela calcula o próximo valor de x baseando-se na derivada da função custo calculada no ponto x .

Exemplificando para um caso escalar podemos dizer que o próximo valor de x , isto é, x_{k+1} , será dado por:

$$x_{k+1} = x_k + \Delta x_k \quad (2.9a)$$

$$\Delta x_k = -K f'(x_k) \quad (2.9b)$$

Onde:

x_k = x atual

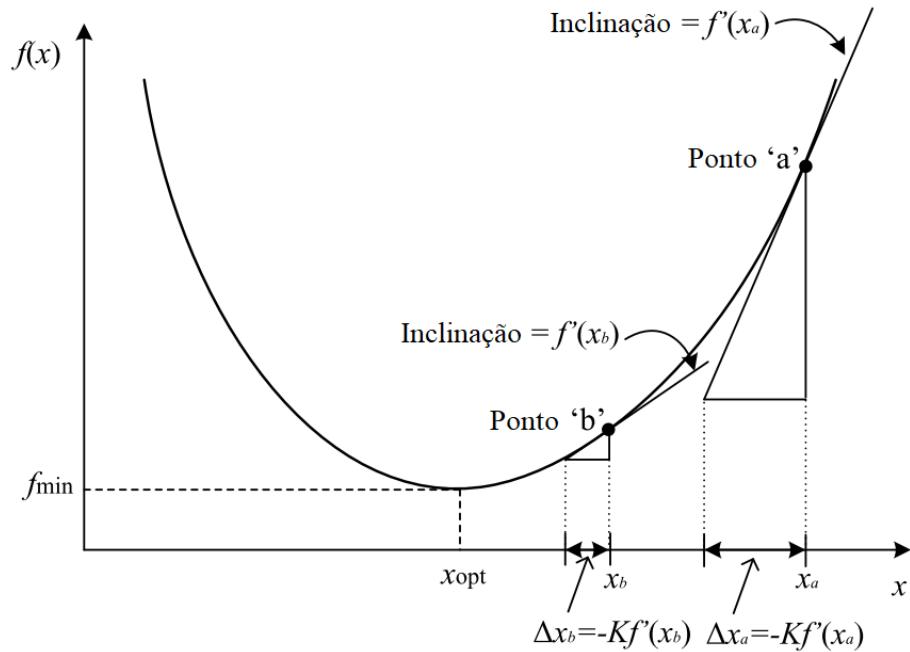
Δx_k = diferença entre x_k e x_{k+1}

K = fator multiplicador do incremento

$f'(x_k)$ = derivada da função custo calculada em x_k

A derivada $f'(x_k)$ indica quão inclinada está a função custo no ponto x_k , assim o fator K determina o peso que essa inclinação terá para o cálculo do próximo valor de x .

Figura 4 – Influência de $f'(x_k)$ no cálculo de x_{k+1} no método de descidas mais íngremes



Fonte: Autor, adaptado de Haugen (2018)

A fig. 4 mostra graficamente a influência da derivada da função custo na amplitude de Δx_k entre iterações. Na figura vemos a derivada aplicada sobre dois pontos distintos, x_a e x_b , e podemos notar que $f'(x_b)$ será menor que $f'(x_a)$ uma vez que x_b está mais próximo ao ponto mínimo de f' .

Como indicado na eq. (2.9a), o cálculo de x_{k+1} depende de x_k , ou seja, o valor do próximo x calculado depende do x anterior. Para o cálculo de x_1 é necessário fazer uma estimativa de x_0 . A maior parte das funções de optimização requerem como argumento um valor de x_0 estimado, pois é a partir dele que serão iniciadas as buscas até o ponto mínimo da função.

As mesmas equações se aplicam para casos onde x é um vetor, com a diferença que ao invés de calcularmos a derivada sobre o ponto x_k , calculamos o gradiente do vetor x_k , tal qual vemos na eq. (2.10b).

$$x_{k+1} = x_k + \Delta x_k \quad (2.10a)$$

$$\Delta x_k = -K\nabla f(x_k) \quad (2.10b)$$

Sendo que:

$$\mathbf{x}_k = \begin{bmatrix} x(1)_k \\ x(2)_k \\ \vdots \\ x(n)_k \end{bmatrix}$$

K = fator multiplicador do incremento

$\nabla f(\mathbf{x}_k)$ = gradiente da função custo calculada em \mathbf{x}_k

Além de requerer um valor estimado para x_0 , este método também apresenta um critério de parada, isto é, uma condição que quando alcançada irá parar o laço de iterações por já ter encontrado o f_{min} . Este critério é dado pela eq. (2.11) e ao alcançá-la então x_{opt} será x_{k+1} .

$$|f(x_{k+1}) - f(x_k)| \leq df \quad (2.11)$$

Onde:

df = Valor do critério de parada (ex. 0,00001)

O código-fonte 2.1 exemplifica a utilização deste método para as equações descritas em 2.12.

$$\min_x f(x) \quad (2.12a)$$

$$f(\mathbf{x}) = [x(1) - 1]^2 + [x(2) - 2]^2 + 0,5 \quad (2.12b)$$

$$|f(\mathbf{x}_{k+1}) - f(\mathbf{x}_k)| \leq df \quad (2.12c)$$

$$df = 10^{-4} \quad (2.12d)$$

Código-fonte 2.1 – Busca por descidas mais íngremes

```
# Linguagem:           Python
# Autor:              Tiago Correa Prata
# Disponível em:
# <https://github.com/TiagoPrata/MasterThesis/blob/master/4_codes/steepest_descent_
vectorial.py>

import numpy as np

# definindo a função custo descrita na eq. (2.12b)
```

```

def f_obj(x):
    return (x[0]-1)**2 + (x[1]-2)**2 + 0.5

x_guess = np.array([0., 1.]).T           # x0 estimado
x_k = x_guess
N = 10000      # limite de tentativas
abs_df = 1e-4    # valor do criterio de parada, como na eq. (2.12d)

for k in range(1, N-1):
    G_k = np.array([2*(x_k[0]-1), 2*(x_k[1]-2)]).T
    K = 0.1
    dx_k = -K * G_k
    x_kp1 = x_k + dx_k
    f_k = f_obj(x_k)
    f_kp1 = f_obj(x_kp1)

    # implementacao do criterio de parada da eq. (2.12c)
    df = f_kp1 - f_k
    x_k = x_kp1
    if abs(df) < abs_df:
        break

x_opt = x_kp1
f_min = f_obj(x_opt)

print(k)
print(x_opt)
print(f_min)

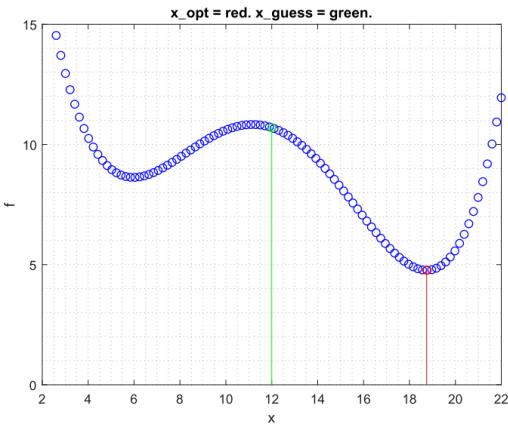
```

Fonte: Autor, adaptado de Haugen (2018)

Um problema conhecido deste método é o fato de ele poder não distinguir entre mínimos locais e mínimos globais fazendo com que o valor que é apontado como f_{min} possa ser na verdade apenas um mínimo local. O valor de x_0 estimado tem impacto direto neste resultado, pois como este método inicia o cálculo da derivada de $f(x)$ no ponto x_0 , então as iterações seguintes dependerão exclusivamente da iteração anterior, fazendo assim uma convergência para o mínimo mais próximo, seja ele local ou global. As figs. 5 e 6 mostram mais detalhadamente esta diferença.

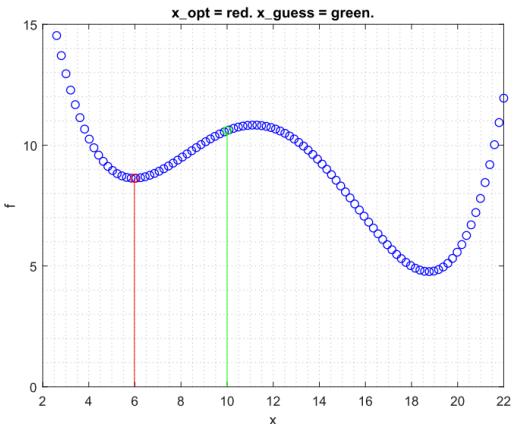
As figs. 5 e 6 apresentam a mesma função custo, porém na fig. 5 $x_0 = 12$, e as derivadas consecutivas a partir desse valor convergem para o mínimo global; já na fig. 6 estimasse $x_0 = 10$ resultando em um f_{min} completamente diferente, convergindo para um mínimo local. Em ambas as figuras a linha verde indica o valor de x_0 e a linha vermelha indica o f_{min} quando o critério de parada é atingido.

Figura 5 – Método de descida mais íngreme convergindo para mínimo global



Fonte: Haugen (2018)

Figura 6 – Método de descida mais íngreme convergindo para mínimo local



Fonte: Haugen (2018)

2.2.3 Otimizadores de programação não-linear (NLP)

Muitos métodos de otimização não suportam restrições como as descritas nas eqs. (2.2) e (2.3), este é, por exemplo, o caso do método de busca de descidas mais íngremes, porém outros métodos podem levar esse tipo de restrição em conta, e estes são conhecidos como Otimizadores de Programação não-linear (do inglês *Nonlinear Programming*, ou simplesmente **NLP**) (HAUGEN, 2018).

Otimizadores **NLP** são úteis quando há a necessidade de métodos de otimização que sejam mais robustos e mais flexíveis do que técnicas mais simples, como o método de busca de Newton, ou aqueles apresentados nas sessões anteriores deste trabalho e por isso muitos softwares como o **MATLAB®** e bibliotecas da linguagem Python dispõem de ferramentas para cálculo de métodos **NLP**. No **MATLAB®** o *Optimization Toolbox™*² possui a função *fmincon* enquanto no Python podemos encontrar a função *scipy.optimize.minimize* na biblioteca *SciPy*³.

Os autores Haugen (2018), Ruggiero e Lopes (2000) apresentam em maiores detalhes os métodos descritos nesta seção, além de muitos outros.

² O **MATLAB® Optimization Toolbox™** é um conjunto de ferramentas de otimização que provê funções para encontrar parâmetros que minimizem ou maximizem objetivos, satisfazendo as condições limites. Mais detalhes em www.mathworks.com/products/optimization.html

³ A biblioteca *SciPy* (do inglês *SciPy Library*) faz parte do conjunto de soluções do programa *SciPy*. A biblioteca *SciPy* fornece uma interface de usuário amigável para cálculos de integração numérica, interpolação, otimização, álgebra linear, estatística, entre outros. Mais detalhes em www.scipy.org/scipylib/index.html

2.3 Algumas aplicações de otimização

2.3.1 Estimação de parâmetros

Estimar parâmetros de um modelo pode ser uma das inúmeras aplicações dos métodos de otimização. Segundo Haugen (2018) um problema de estimação de parâmetros pode ser formulado como mostrado na eq. (2.13), visando encontrar os parâmetros que apresentem menor diferença entre o sistema real e o modelo sendo avaliado. A comparação do sistema real com o virtual é feita através do acúmulo da diferença das saídas dos dois sistemas elevada ao quadrado ao longo de N iterações.

$$\min_P \sum_{k=1}^N e(k)^2 \quad (2.13)$$

Onde:

P = vetor de parâmetros a ser estimado. Sendo $P = [p(1), p(2), \dots, p(r)]^T$

e = erro de predição. Sendo $e = y_{medido} - y_{modelo}$

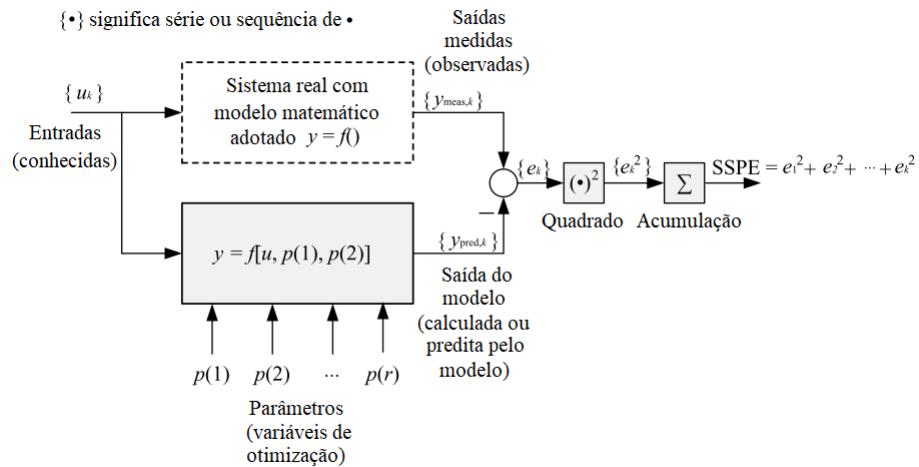
y_{medido} = medição da saída do sistema real

y_{modelo} = cálculo da saída do modelo simulado

N = quantidade de amostras

O diagrama de blocos apresentado na fig. 7 apresenta de forma visual a comparação entre o sistema real e o modelo matemático.

Figura 7 – Diagrama de blocos do quadrado da diferença entre o sistema real e o modelo



Fonte: Autor, adaptado de Haugen (2018)

Para estimar os parâmetros P de um dado modelo é necessário dividir os dados amostrais N em dois grupos: um deles destinado a adaptação do modelo, ou seja, encontrar

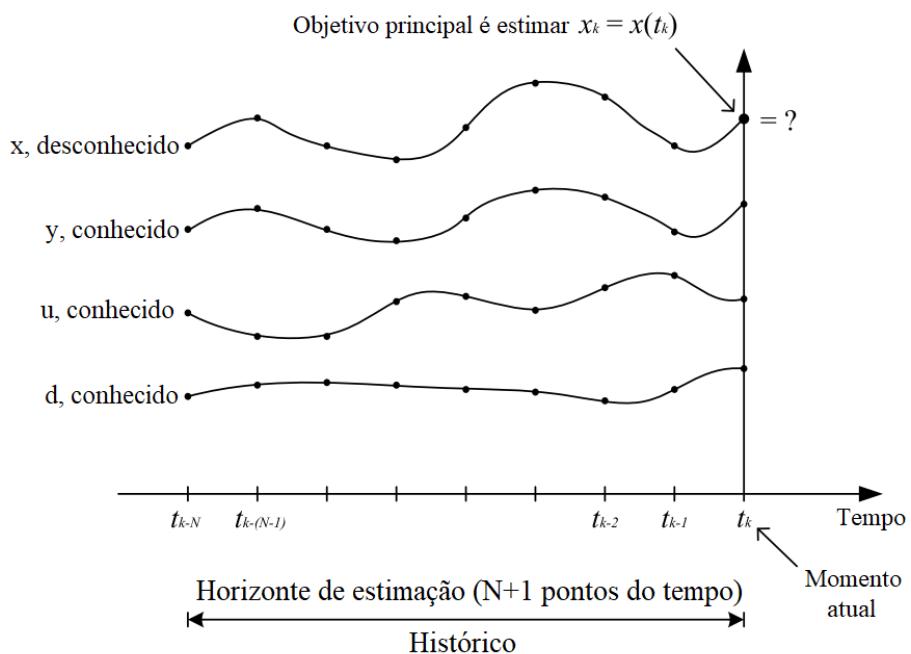
o vetor P que minimize a função custo, e o outro grupo destinado à validação, isto é, verificar se os parâmetros encontrados durante a adaptação realmente é válido e preciso.

Quando dispomos de dados previamente coletados do sistema é possível fazer uma *estimação em batelada* (também conhecida como *offline* (BOLOGNANI et al., 2009)), onde as fases de coleta e estimação acontecem em momentos distintos. Já a *estimação recursiva* (ou *online* (STADLER; POLAND; GALLESTEY, 2011)) ocorre quando a coleta dos dados e a estimação dos parâmetros acontece ao mesmo tempo. A [seção 2.3.2](#) descreve com mais detalhes a **Estimador de Horizonte Móvel** (do inglês, *Moving Horizon Estimation*, ou simplesmente **MHE**), que, assim como o filtro de Kalman, pode ser utilizado como estimador de parâmetros recursivo.

2.3.2 Estimador de Horizonte Móvel (MHE)

O método conhecido como **Estimador de Horizonte Móvel** é um método estimador recursivo de parâmetros que utiliza as medições atuais, anteriores e também as entradas do sistema sob um horizonte fixo de tempo para calcular o estado atual do sistema, assim como ilustrado na [fig. 8](#) (HAUGEN, 2018).

Figura 8 – Horizonte de tempo analisado pelo método **Estimador de Horizonte Móvel**



Fonte: Autor, adaptado de Haugen (2018)

O modelo de espaço de estado que descreve o sistema pode ser descrito como indicado na [eq. \(2.14\)](#).

$$\mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k, \dots) + \mathbf{w}_k \quad (2.14a)$$

$$\mathbf{y}_k = \mathbf{g}(\mathbf{x}_k, \dots) + \mathbf{v}_k \quad (2.14b)$$

Onde:

- \mathbf{x} é o vetor de estado a ser estimado.

$$\mathbf{x}_k = \begin{bmatrix} x(1)_k \\ x(2)_k \\ \vdots \\ x(n)_k \end{bmatrix} \quad (2.15)$$

- \mathbf{f} é um vetor de n funções lineares ou não-lineares de x_k

$$\mathbf{f} = \begin{bmatrix} f_1(x_k, \dots) \\ f_2(x_k, \dots) \\ \vdots \\ f_n(x_k, \dots) \end{bmatrix} \quad (2.16)$$

Os pontos representam possíveis argumentos adicionais de f como a variável de controle (u_k), os distúrbios do processo (d_k), e os parâmetros (p).

- \mathbf{w}_k representa os distúrbios não conhecidos ou não modelados do processo agindo sobre o estado, no instante k . Não é necessário assumir nenhuma propriedade estatística particular para este distúrbio, porém é sensato assumir um ruído, como o ruído branco, com uma matriz de covariância Q , assim como utilizado no filtro de Kalman. A matriz Q será apresentada com mais detalhes a seguir.
- \mathbf{y} é o vetor de saída com m elementos.
- \mathbf{g} é um vetor de m funções lineares ou não-lineares de \mathbf{x}_k , onde os pontos também representam possíveis argumentos adicionais.
- \mathbf{v}_k representa o erro medido, e, assim como \mathbf{w}_k , não necessita assumir nenhuma propriedade estatística particular, porém também é comum utilizar um ruído branco, com uma matriz de covariância R , também melhor apresentada a seguir.

O problema de otimização do MHE pode ser descrito como na eq. (2.17).

$$\min_{\mathbf{X}} \sum_{i=k-N}^{k-1} \|x_{i+1} - f(x_i, \dots)\|_{Q^{-1}}^2 + \sum_{i=k-N}^k \|y_i - g(x_i, \dots)\|_{R^{-1}}^2 \quad (2.17)$$

Sendo que X é a matriz contendo o estado a cada ponto do horizonte de estimação:

$$\begin{aligned} X &= [x_{k-N}, x_{k-N-1}, \dots, x_{k-1}, x_k] \\ &= \begin{bmatrix} x(1)_{k-N} & x(1)_{k-(n-1)} & \cdots & x(1)_{k-1} & x(1)_k \\ x(2)_{k-N} & x(2)_{k-(n-1)} & \cdots & x(2)_{k-1} & x(2)_k \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ x(n)_{k-N} & x(n)_{k-(n-1)} & \cdots & x(n)_{k-1} & x(n)_k \end{bmatrix} \end{aligned} \quad (2.18)$$

E com base na eq. (2.14) este problema também pode ser descrito como:

$$\min_X \sum_{i=k-N}^{k-1} \|w_i\|_{Q^{-1}}^2 + \sum_{i=k-N}^k \|v_i\|_{R^{-1}}^2 \quad (2.19)$$

Ou mesmo:

$$\min_X \sum_{i=k-N}^{k-1} w_i^T Q^{-1} w_i + \sum_{i=k-N}^k v_i^T R^{-1} v_i \quad (2.20)$$

Onde:

- w_i é o vetor de distúrbio do processo.

$$w_i = \begin{bmatrix} w(1)_i \\ w(2)_i \\ \vdots \\ w(n)_i \end{bmatrix} \quad (2.21)$$

- v_i é um vetor de erros de medição.

$$v_i = \begin{bmatrix} v(1)_i \\ v(2)_i \\ \vdots \\ v(m)_i \end{bmatrix} \quad (2.22)$$

- Q^{-1} é a matriz de ponderação de w_i

$$Q^{-1} = \begin{bmatrix} \frac{1}{Q_{11}} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \frac{1}{Q_{nn}} \end{bmatrix} \quad (2.23)$$

Assumindo que w é aleatório, então Q pode ser interpretado como a inversa da matriz de covariância do distúrbio do processo.

- R^{-1} é a matriz de ponderação de v_i

$$R^{-1} = \begin{bmatrix} \frac{1}{R_{11}} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \frac{1}{R_{nn}} \end{bmatrix} \quad (2.24)$$

Assumindo que v é aleatório, então R pode ser interpretado como a inversa da matriz de covariância do distúrbio do processo.

Desta forma a eq. (2.20) pode ser reescrita como:

$$\min_X \sum_{i=k-N}^{k-1} \left[\frac{w(1)_i^2}{Q_{11}} + \cdots + \frac{w(n)_i^2}{Q_{nn}} \right] + \sum_{i=k-N}^k \left[\frac{v(1)_i^2}{R_{11}} + \cdots + \frac{v(n)_i^2}{R_{mm}} \right] \quad (2.25)$$

Resumidamente observa-se que o MHE utiliza as medições (minimizando a influência dos erros de medição) e o modelo (minimizando os erros de modelo) para calcular o estado estimado.

2.3.2.1 Exemplo

A fig. 9 mostra o gráfico de uma simulação executada através do código-fonte A.3 exemplificando a aplicação da técnica Estimador de Horizonte Móvel para o modelo simplificado da velocidade de um equipamento de corrente contínua.

O modelo dinâmico deste equipamento é apresentado na eq. (2.26) e sua representação no espaço de estados na eq. (2.27), sendo que esta é a representação utilizada no código-fonte A.3.

$$T\dot{S} = -S + Ku + L \quad (2.26)$$

Onde:

u = sinal de controle (tensão aplicada ao motor)

S = sinal medido (velocidade do motor)

K = ganho

T = constante de tempo

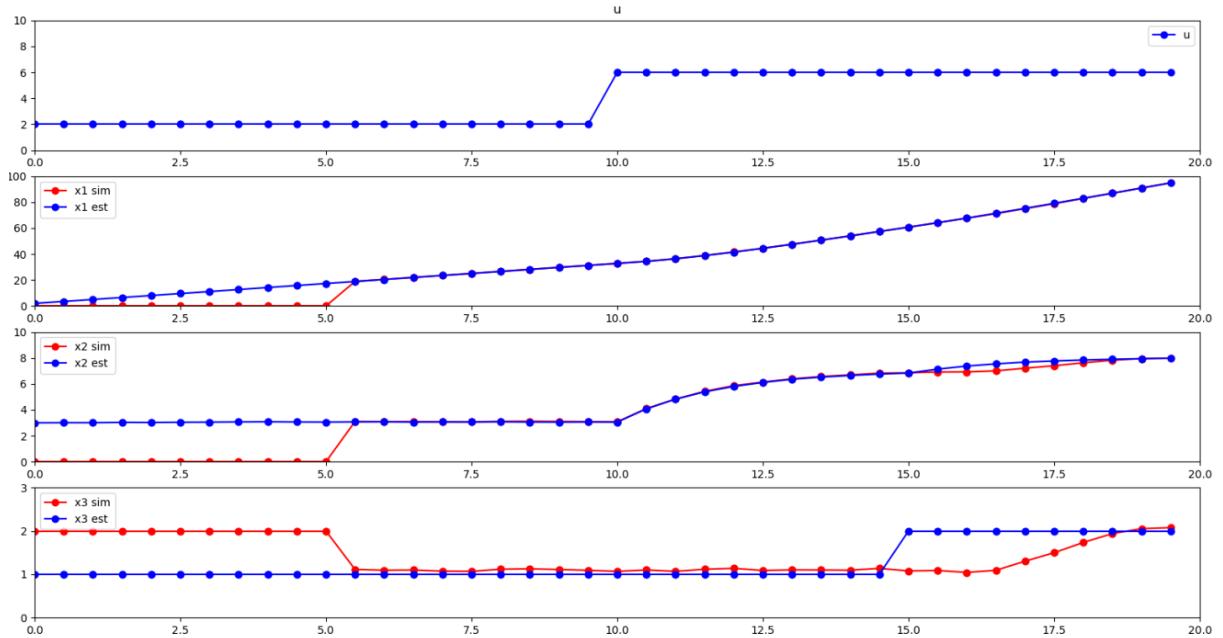
L = carga

$$\dot{x}(1) = x(2) \quad (2.27a)$$

$$\dot{x}(2) = [-x(2) + Ku]/T + d \quad (2.27b)$$

$$y = x(1) \quad (2.27c)$$

Figura 9 – Simulação utilizando Estimador de Horizonte Móvel



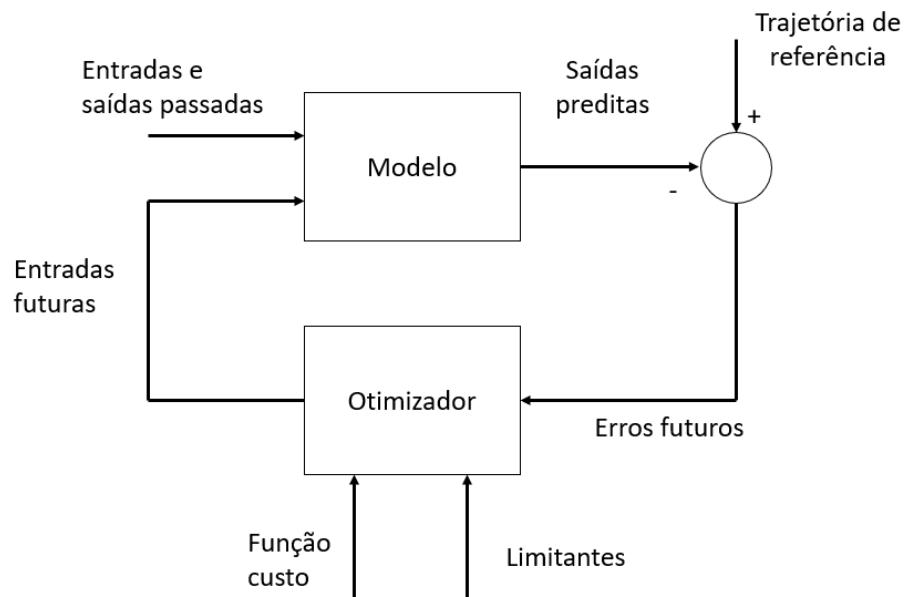
Fonte: Autor, adaptado de Haugen (2018)

A fig. 9 apresenta a aproximação do valor dos estados estimados através do MHE em comparação com o sinal simulado do equipamento. É possível observar um baixo erro relativo para todos os estados estimados, mesmo quando um degrau na entrada é aplicado. Vale ressaltar porém que a estimativa dos estados começa a ser calculada apenas quando o primeiro horizonte de predição é preenchido, nesta figura observamos isso no instante $t = 5$. Além disso, outro destaque pode ser feito para o atraso indesejado observado no estado x_3 , indicando que o modelo pode ser aprimorado caso seja necessário.

3 Controle Preditivo Baseado em Modelo

Segundo Rossiter (2003) uma das principais diferenças entre o MPC e outras técnicas de controle (como o PID, por exemplo) é que a maioria delas não leva em consideração as ações futuras do controle, porém o MPC sim, e para essas ações futuras sejam consideradas é necessário possuir um modelo do sistema que mostre as dependências das saídas e das atuais variáveis medidas e as entradas atuais e futuras. Este modelo não precisa ser linear, e nem tampouco ser extremamente fiel em descrever todas as interações físico-químicas do sistema. Na verdade, ainda segundo Rossiter (2003), a regra básica do modelo é que ele deve ser o mais simples possível, ou seja, deve ser o modelamento mínimo necessário para que possamos observar previsões com o nível de precisão necessário.

Figura 10 – Estrutura básica do MPC



Fonte: Autor, adaptado de Camacho, Alba e Bordons (2007)

Além de comparar com outros métodos de controle e de descrever os modelos utilizados no MPC, Rossiter (2003) também salienta alguns outros parâmetros e características que são importantes para a técnica:

- A **seleção das entradas** do sistema deve ser feita considerando a função custo que se deseja minimizar.
- O intervalo de tempo futuro, no qual o MPC fará o cálculo das previsões de controle (também conhecido como **horizonte retrocedente** (*Receding Horizon*) deve incluir

todas as dinâmicas significativas para o sistema, caso contrário o controle apresentará um desempenho ruim e alguns eventos não poderão ser observados.

- O controle será tão preciso quanto seu modelo permitir. Para conseguir um controle mais preciso é necessário também possuir um modelo mais preciso.
- Um dos principais pontos do MPC é a capacidade de **lidar com restrições** on-line de maneira sistemática, permitindo assim uma melhor performance.
- Ao utilizar um modelo, este método calcula a otimização do custo levando em conta as **mudanças futuras na trajetória** desejada e distúrbios mensuráveis.
- Outra característica extremamente importante do MPC é sua capacidade intrínseca de controlar sistemas **MIMO**.

3.1 Aplicação prática

Segundo Haugen (2018) tanto o MPC quanto o MHE são problemas praticamente idênticos do ponto de vista matemático, pois ambos exploram um modelo matemático sobre dado um horizonte de tempo, porém, apesar de matematicamente parecidos, eles atuam de forma inversa um ao outro, pois enquanto o MHE utiliza as variáveis de controle e os valores medidos do processo para estimar as variáveis estados (como já apresentado na [seção 2.3.2](#)), o MPC utiliza as variáveis de estado e os valores medidos do processo para estimar as variáveis de controle. Na [fig. 11](#) é possível observar o MHE atuando sob um horizonte de estimativa, utilizando dados passados, e o MPC atuando sob um horizonte de predição, predizendo os valores futuros.

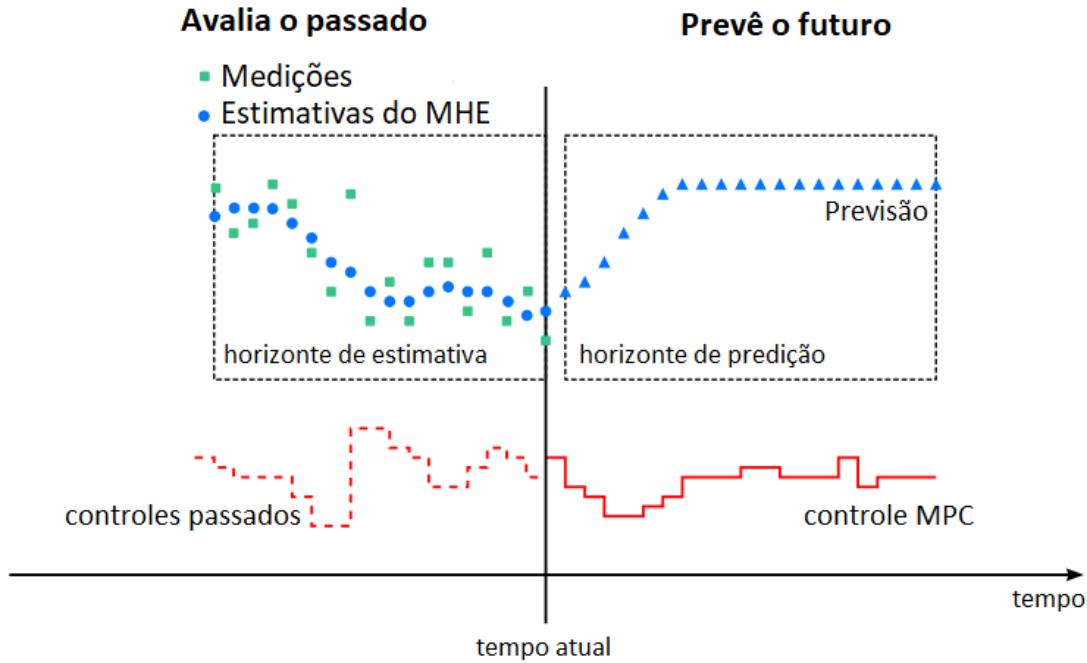
Podemos descrever o MPC através de um problema de otimização de forma análoga ao que foi feito na [seção 2.3.2](#) para descrever o MHE, porém, ao invés de minimizarmos as variáveis de estado, devemos encontrar os valores ótimos (minimizar) das variáveis de controle. As [eqs. \(3.1\)](#) a [\(3.9\)](#) descrevem este problema de otimização.

$$\min_U \sum_{i=k}^{k+N} (\|e_i\|_{C_e}^2 + \|du\|_{C_{du}}^2) \quad (3.1)$$

Onde:

- U é uma matriz contendo r variáveis de controle nos instantes k do horizonte de

Figura 11 – Ação do Estimador de Horizonte Móvel e do Controle Preditivo Baseado em Modelo



Fonte: Autor, adaptado de Vukov (2015)

predição.

$$\begin{aligned} \mathbf{U} &= [u_k, u_{k+1}, \dots, u_{k+(N-1)}, u_N] \\ &= \begin{bmatrix} u(1)_k & u(1)_{k+1} & \cdots & u(1)_{k+(N-1)} & u(1)_N \\ u(2)_k & u(2)_{k+1} & \cdots & u(2)_{k+(N-1)} & u(2)_N \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ u(n)_k & u(n)_{k+1} & \cdots & u(n)_{k+(N-1)} & u(n)_N \end{bmatrix} \end{aligned} \quad (3.2)$$

- e_i é o vetor de erro de controle

$$e_i = \begin{bmatrix} e(1)_i \\ e(2)_i \\ \vdots \\ e(m)_i \end{bmatrix} \quad (3.3)$$

Onde $e(j)_i$ é o erro de controle da saída j do processo, no instante de tempo i , sendo que:

$$e(j)_i = y(j)_{sp_i} - y(j)_i \quad (3.4)$$

- du_i é o vetor de incremento da variável de controle

$$du_i = \begin{bmatrix} du(1)_i \\ du(2)_i \\ \vdots \\ du(r)_i \end{bmatrix} \quad (3.5)$$

Onde $du(j)_i$ é o incremento da variável de controle relativo a esta mesma variável no instante de tempo anterior:

$$du(j)_i = u(j)_i - u(j)_{i-1} \quad (3.6)$$

- As matrizes C_e e C_{du} são matrizes de custo (peso) e são utilizadas como elementos de sintonia.

$$C_e = \begin{bmatrix} C_e(1, 1) & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & C_e(m, m) \end{bmatrix} \quad (3.7a)$$

$$C_{du} = \begin{bmatrix} C_{du}(1, 1) & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & C_{du}(r, r) \end{bmatrix} \quad (3.7b)$$

Substituindo as normas matriciais da eq. (3.1), temos:

$$\min_U \sum_{i=k}^{k+N} (e_i^T C_e e_i + du_i^T C_{du} du_i) \quad (3.8)$$

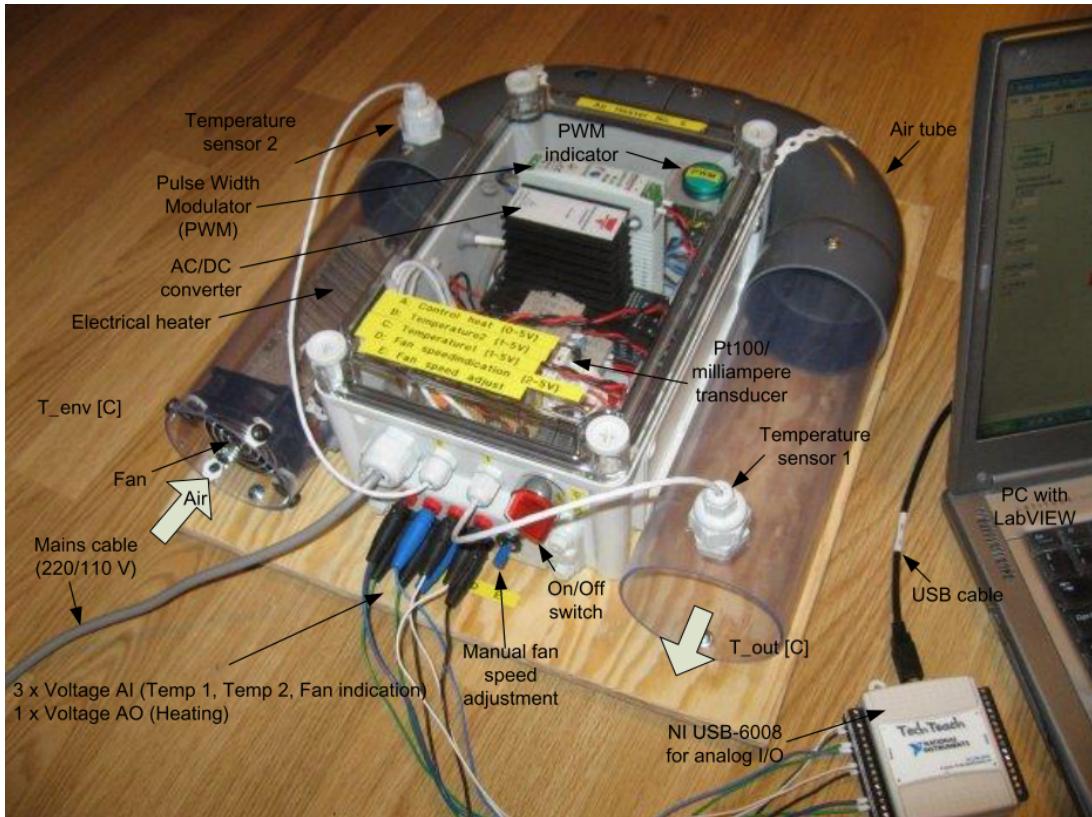
Expandindo então as eqs. (3.2) a (3.7) na eq. (3.8), obtemos:

$$\min_U \sum_{i=k}^{k+N} [C_e(1, 1)e(1)_i^2 + \cdots + C_e(m, m)e(m)_i^2] + [C_{du}(1, 1)du(1)_i^2 + \cdots + C_{du}(r, r)du(r)_i^2] \quad (3.9)$$

3.1.1 Exemplo

O exemplo descrito nesta seção apresenta a simulação do modelo de uma planta didática de aquecimento de ar utilizado na *University College of Southeast Norway*, situada em Porsgrunn, Noruega. Vale salientar que esta não é a planta objeto de estudo deste trabalho, sendo este apenas um exemplo bibliográfico de aplicação de MPC em planta didática, onde é realizado um controle de temperatura em túnel de vento, cujo modelo

Figura 12 – Planta de aquecimento de ar



Fonte: Haugen (2018)

modelo matemático está indicado na eq. (3.10) e seu detalhamento construtivo pode ser melhor compreendido na fig. 12.

$$\theta_t \dot{T}_{heat}(t) = -T_{heat}(t) + K_h[u(t - \theta_d) + d] \quad (3.10a)$$

$$T_{out}(t) = T_{heat}(t) + T_{env} \quad (3.10b)$$

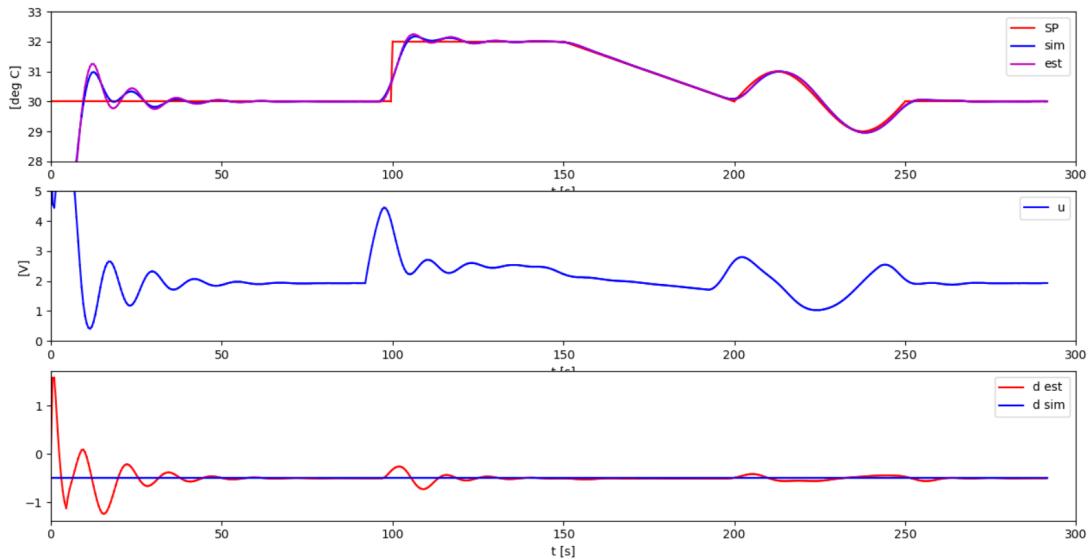
Onde:

- K_h é o ganho do aquecedor
- T_{env} é a temperatura ambiente
- T_{heat} é a contribuição para a temperatura total T_{out} devido ao aquecedor
- T_{out} é a temperatura do ar saindo da planta. Medido através de sensor
- u é o sinal de controle do aquecedor
- d é o distúrbio de entrada (adicionado ao sinal de controle)

- θ_d é o atraso que representa o transporte de ar e a dinâmica do aquecedor
- θ_t é o atraso referente à dinâmica do aquecedor

A [fig. 13](#) ilustra os dados obtidos através da execução do código-fonte [A.4](#), no apêndice A, escrito em Python.

Figura 13 – Simulação utilizando Controle Preditivo Baseado em Modelo



Fonte: Autor, adaptado de Haugen (2018)

Mais detalhes sobre o funcionamento do MPC serão abordados nos próximos capítulos, porém, utilizando a [fig. 12](#), já é possível observar algumas de suas características. No primeiro gráfico, onde são correlacionados os sinais de *set-point* (SP), de estimativa do modelo (est) e de simulação (sim), notasse que o sinal de simulação aproxima-se rapidamente do *set-point*, independentemente de a variação deste ser através de um degrau, rampa ou senoidal. Ainda neste gráfico, nota-se também que os sinais *sim* e *est* começam a ser computados apenas por volta do segundo 10, pois antes disso não há dados suficientes no horizonte selecionado para executar tal cálculo.

Parte III

Materiais e métodos

4 Planta piloto

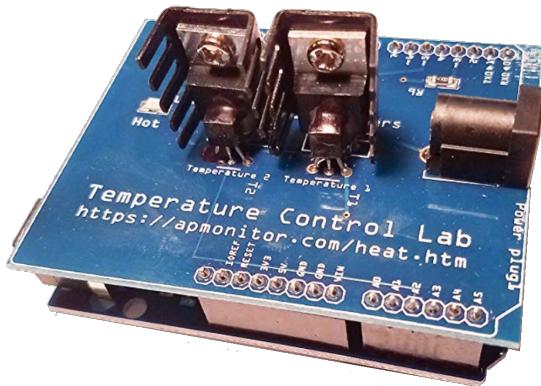
4.1 TCLab

O equipamento utilizado para o projeto de ações de controle é um mini sistema chamado *Temperature Control Lab* (do inglês, Laboratório de Controle de Temperatura), ou apenas **TCLab**¹(fig. 14). Esse sistema foi desenvolvido na *Brigham Young University* (BYU) e apresentado pela primeira vez no *2017 ASEE Summer School*, na *North Carolina State University* (NCSU). Foi desenvolvido com o propósito de facilitar o acesso de estudantes a um laboratório de testes de controle.

Esse equipamento é essencialmente um *shield* para Arduino² contendo 2 aquecedores e 2 sensores de temperatura, indicados na fig. 15. A energia dos aquecedores é transferida através de condução, convecção e radiação até os sensores de temperatura. Tanto o controle da potência dos aquecedores quanto as medições realizadas pelos sensores são efetuados através do Arduino.

O **TCLab** permite que a programação de controle do sistema possa ser feito utilizando linguagem Python, **MATLAB®** ou Simulink.

Figura 14 – Laboratório de Controle de Temperatura

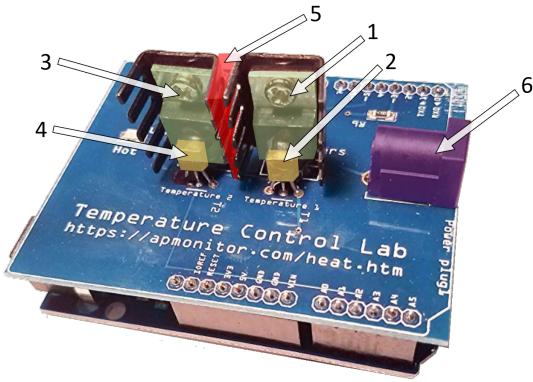


Fonte: Hedengren (2018)

¹ Maiores informações sobre o **TCLab** podem ser encontrados em <http://apmonitor.com/heat.htm>.

² Arduino é uma plataforma de prototipagem eletrônica de hardware livre e de placa única, projetada com um microcontrolador Atmel AVR com suporte de entrada/saída embutido.

Figura 15 – Componentes principais do TCLab



Fonte: Prata (2019), adaptado de Hedengren (2018)

Tabela 1 – Componentes principais do TCLab

Item	Descrição
1	Aquecedor 1 (T_{H1})
2	Sensor de temperatura 1 (T_{C1})
3	Aquecedor 2 (T_{H2})
4	Sensor de temperatura 2 (T_{C2})
5	Superfície entre dissipadores [2 cm^2]
6	Entrada para alimentação

Fonte: Prata (2019)

4.2 Planta piloto - TCLabSP

A planta piloto desenvolvida para esse projeto é chamada de *Temperature Control Lab São Paulo*, ou apenas **TCLabSP**, e consiste em um envólucro em acrílico que foi construído com o intuito de minimizar interferências externas e de possibilitar um maior controle dos distúrbios, uma vez que ventiladores foram incluídos ao sistema, juntamente com o **TCLab** apresentado na seção 4.1.

A fig. 16 apresenta uma visão geral da **TCLabSP**, enquanto a fig. 17 e a tabela 2 detalham seus componentes principais.

4.3 Modelagem teórica da planta piloto

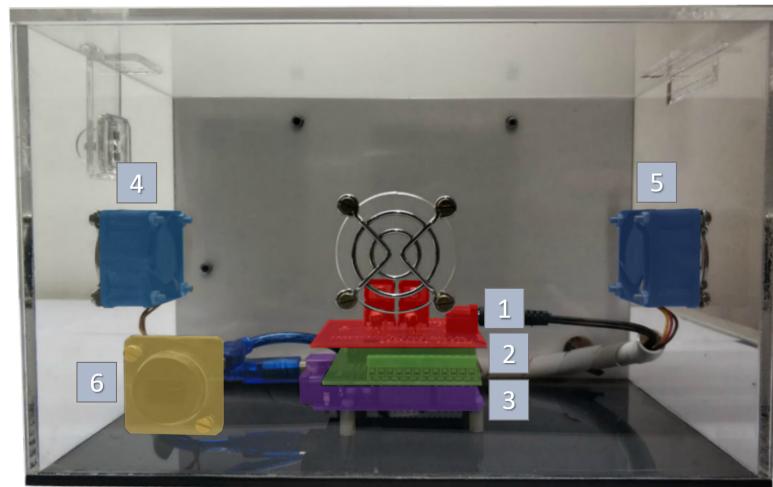
O balanço energético da **TCLab** pode ser obtido em sua documentação e a eq. (4.1) apresenta os modelos para um sistema **MIMO**, onde ambos os aquecedores e sensores são utilizados, sendo que, para isso, deve-se assumir que: os aquecedores e o sensores de temperatura estão na mesma temperatura; que o efeito da condução de calor é desprezado e

Figura 16 – Visão geral da Planta Piloto - TCLabSP



Fonte: Prata (2019)

Figura 17 – Componentes principais do TCLabSP



Fonte: Prata (2019)

Tabela 2 – Componentes principais do TCLabSP

Item	Descrição
1	TCLab
2	Shield para ligação dos ventiladores
3	Arduino Uno
4	Ventilador 1
5	Ventilador 2
6	Conector mini USB

Fonte: Prata (2019)

que as únicas trocas de calor acontecem através de radiação ou convecção; que o aquecedor inicialmente está desligado e que tanto o aquecedor quanto o sensor inicialmente estão em

temperatura ambiente.

$$mC_p \frac{dT_{H1}}{dt} = UA(T_\infty - T_{H1}) + \epsilon\sigma A(T_\infty^4 - T_{H1}^4) + Q_{c12} + Q_{r12} + \alpha_1 Q_1 \quad (4.1a)$$

$$mC_p \frac{dT_{H2}}{dt} = UA(T_\infty - T_{H2}) + \epsilon\sigma A(T_\infty^4 - T_{H2}^4) + Q_{c21} + Q_{r21} + \alpha_2 Q_2 \quad (4.1b)$$

$$\tau \frac{dT_{C1}}{dt} = T_{H1} - T_{C1} \quad (4.1c)$$

$$\tau \frac{dT_{C2}}{dt} = T_{H2} - T_{C2} \quad (4.1d)$$

$$Q_{c12} = U_s A_s (T_{H2} - T_{H1})$$

$$Q_{c21} = U_s A_s (T_{H1} - T_{H2})$$

$$Q_{r12} = \epsilon\sigma A_s (T_{H2}^4 - T_{H1}^4)$$

$$Q_{r21} = \epsilon\sigma A_s (T_{H1}^4 - T_{H2}^4)$$

A [tabela 3](#) indica a descrição de cada uma das siglas da [eq. \(4.1\)](#) e também apresenta o valor de cada um.

Tabela 3 – Valores para modelagem [MIMO](#) do [TCLab](#)

Sigla	Descrição	Valor	Valor (SI)
T_{H1}	Temperatura do aquecedor 1	-	-
T_{C1}	Temperatura do sensor 1	-	-
T_{H2}	Temperatura do aquecedor 2	-	-
T_{C2}	Temperatura do sensor 2	-	-
T_0	Temperatura inicial	20°C	293,15K
T_∞	Temperatura ambiente	20°C	293,15K
Q_1	Saída do aquecedor 1	0 à 100%	0 à 100%
α_1	Fator do aquecedor 1	0,0123 W/%aquec.	0,0123 W/%aquec.
Q_2	Saída do aquecedor 2	0 à 100%	0 à 100%
α_2	Fator do aquecedor 2	0,0062 W/%aquec.	0,0062 W/%aquec.
C_p	Capacidade de aquecimento	500 J/kg.K	500 J/kg.K
A	Área de superfície	10cm^2	$1 \times 10^{-3}\text{m}^2$
A_s	Idem A porém entre aquecedores	2cm^2	$2 \times 10^{-4}\text{m}^2$
m	Massa	4g	0,004kg
τ	Constante de tempo de condução	20,3s	20,3s
U	Coef. global de transf. de calor	4,7 W/m²K	4,7 W/m²K
U_s	Idem U_s porém entre aquecedores	15,45 W/m²K	15,45 W/m²K
ϵ	Emissividade	0,9	0,9
σ	Constante de Stefan Boltzmann	$5,67 \times 10^{-8} \text{W/m}^2\text{K}^4$	$5,67 \times 10^{-8} \text{W/m}^2\text{K}^4$

Fonte: Autor, adaptado de [Hedengren \(2018\)](#)

A partir da [eq. \(4.1\)](#) é possível elaborar o modelo de espaço de estados teórico. Para isso assume-se que os estados são T_{H1} , T_{H2} , T_{C1} e T_{C2} , todos linearizados por expansões em séries de Taylor.

O modelo de espaço de estados teórico tem a forma:

$$\dot{x} = Ax + Bu$$

$$y = Cx + Du$$

Sendo:

$$\begin{aligned} \begin{bmatrix} \dot{T}'_{H1} \\ \dot{T}'_{H2} \\ \dot{T}'_{C1} \\ \dot{T}'_{C2} \end{bmatrix} &= \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} \\ a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} \end{bmatrix} \begin{bmatrix} T'_{H1} \\ T'_{H2} \\ T'_{C1} \\ T'_{C2} \end{bmatrix} + \begin{bmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \\ b_{3,1} & b_{3,2} \\ b_{4,1} & b_{4,2} \end{bmatrix} \begin{bmatrix} Q'_1 \\ Q'_2 \end{bmatrix} \\ \begin{bmatrix} T'_{C1} \\ T'_{C2} \end{bmatrix} &= \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} T'_{H1} \\ T'_{H2} \\ T'_{C1} \\ T'_{C2} \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} Q'_1 \\ Q'_2 \end{bmatrix} \end{aligned} \quad (4.2)$$

Onde:

$$a_{1,1} = \frac{\partial f_1}{\partial T_{H1}} \Big|_{\bar{Q}, \bar{T}} = -\frac{UA + 4\epsilon\sigma AT_0^3 + U_s A_s + 4\epsilon\sigma A_s T_0^3}{mC_p} = -0.00698$$

$$a_{1,2} = \frac{\partial f_1}{\partial T_{H2}} \Big|_{\bar{Q}, \bar{T}} = \frac{U_s A_s + 4\epsilon\sigma A_s T_0^3}{mC_p} = 0.00205$$

$$a_{1,3} = \frac{\partial f_1}{\partial T_{C1}} \Big|_{\bar{Q}, \bar{T}} = 0$$

$$a_{1,4} = \frac{\partial f_1}{\partial T_{C2}} \Big|_{\bar{Q}, \bar{T}} = 0$$

$$a_{2,1} = \frac{\partial f_2}{\partial T_{H1}} \Big|_{\bar{Q}, \bar{T}} = \frac{U_s A_s + 4\epsilon\sigma A_s T_0^3}{mC_p} = 0.00205$$

$$a_{2,2} = \frac{\partial f_2}{\partial T_{H2}} \Big|_{\bar{Q}, \bar{T}} = -\frac{UA + 4\epsilon\sigma AT_0^3 + U_s A_s + 4\epsilon\sigma A_s T_0^3}{mC_p} = -0.00698$$

$$a_{2,3} = \frac{\partial f_2}{\partial T_{C1}} \Big|_{\bar{Q}, \bar{T}} = 0$$

$$a_{2,4} = \frac{\partial f_2}{\partial T_{C2}} \Big|_{\bar{Q}, \bar{T}} = 0$$

$$a_{3,1} = \frac{\partial f_3}{\partial T_{H1}} \Big|_{\bar{Q}, \bar{T}} = \frac{1}{\tau} = 0.04926$$

$$a_{3,2} = \frac{\partial f_3}{\partial T_{H2}} \Big|_{\bar{Q}, \bar{T}} = 0$$

$$a_{3,3} = \frac{\partial f_3}{\partial T_{C1}} \Big|_{\bar{Q}, \bar{T}} = -\frac{1}{\tau} = -0.04926$$

$$a_{3,4} = \frac{\partial f_3}{\partial T_{C2}} \Big|_{\bar{Q}, \bar{T}} = 0$$

$$a_{4,1} = \frac{\partial f_4}{\partial T_{H1}} \Big|_{\bar{Q}, \bar{T}} = 0$$

$$a_{4,2} = \frac{\partial f_4}{\partial T_{H2}} \Big|_{\bar{Q}, \bar{T}} = \frac{1}{\tau} = 0.04926$$

$$a_{4,3} = \frac{\partial f_4}{\partial T_{C1}} \Big|_{\bar{Q}, \bar{T}} = 0$$

$$a_{4,4} = \frac{\partial f_4}{\partial T_{C2}} \Big|_{\bar{Q}, \bar{T}} = -\frac{1}{\tau} = -0.04926$$

$$b_{1,1} = \frac{\partial f_1}{\partial T_{Q1}} \Big|_{\bar{Q}, \bar{T}} = \frac{\alpha_1}{mC_p} = 0.00615$$

$$b_{1,2} = \frac{\partial f_1}{\partial T_{Q2}} \Big|_{\bar{Q}, \bar{T}} = 0$$

$$b_{2,1} = \frac{\partial f_2}{\partial T_{Q1}} \Big|_{\bar{Q}, \bar{T}} = 0$$

$$b_{2,2} = \frac{\partial f_2}{\partial T_{Q2}} \Big|_{\bar{Q}, \bar{T}} = \frac{\alpha_2}{mC_p} = 0.00310$$

$$b_{3,1} = \frac{\partial f_3}{\partial T_{Q1}} \Big|_{\bar{Q}, \bar{T}} = 0$$

$$b_{3,2} = \frac{\partial f_3}{\partial T_{Q2}} \Big|_{\bar{Q}, \bar{T}} = 0$$

$$b_{4,1} = \frac{\partial f_4}{\partial T_{Q1}} \Big|_{\bar{Q}, \bar{T}} = 0$$

$$b_{4,2} = \frac{\partial f_4}{\partial T_{Q2}} \Big|_{\bar{Q}, \bar{T}} = 0$$

Assim temos:

$$\begin{bmatrix} \dot{T}'_{H1} \\ \dot{T}'_{H2} \\ \dot{T}'_{C1} \\ \dot{T}'_{C2} \end{bmatrix} = \begin{bmatrix} -0.00698 & 0.00205 & 0 & 0 \\ 0.00205 & -0.00698 & 0 & 0 \\ 0.04926 & 0 & -0.04926 & 0 \\ 0 & 0.04926 & 0 & -0.04926 \end{bmatrix} \begin{bmatrix} T'_{H1} \\ T'_{H2} \\ T'_{C1} \\ T'_{C2} \end{bmatrix} + \begin{bmatrix} 0.00615 & 0 \\ 0 & 0.00310 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} Q'_1 \\ Q'_2 \end{bmatrix}$$

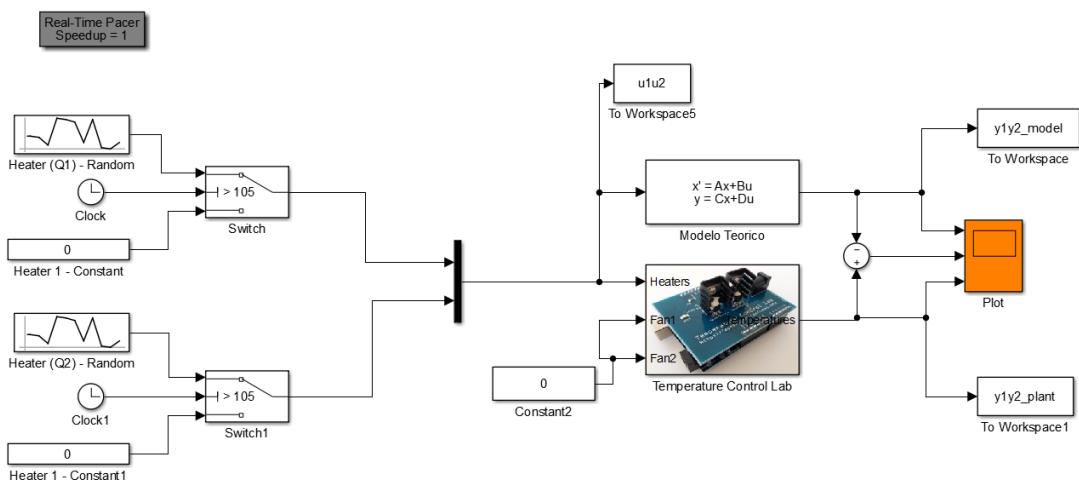
$$\begin{bmatrix} T'_{C1} \\ T'_{C2} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} T'_{H1} \\ T'_{H2} \\ T'_{C1} \\ T'_{C2} \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} Q'_1 \\ Q'_2 \end{bmatrix} \quad (4.3)$$

4.3.1 Verificação do modelo teórico

Para a verificação do modelo teórico obtido na eq. (4.3) criou-se o ambiente de teste indicado na fig. 18 onde tanto o sistema real quanto o modelo teórico eram excitados pelo mesmo sinal de entrada.

As motivações e o detalhamento dos sinais de excitação utilizados neste experimento são melhores abordados posteriormente na seção 5.1.2 - Sinais de excitação.

Figura 18 – Diagrama do ambiente de testes do modelo teórico



Fonte: Autor

A comparação entre as respostas do sistema real e do modelo teórico, quando excitados em malha aberta pelos mesmos sinais de entrada, pode ser verificada nas fig. 19 e fig. 20, respectivamente.

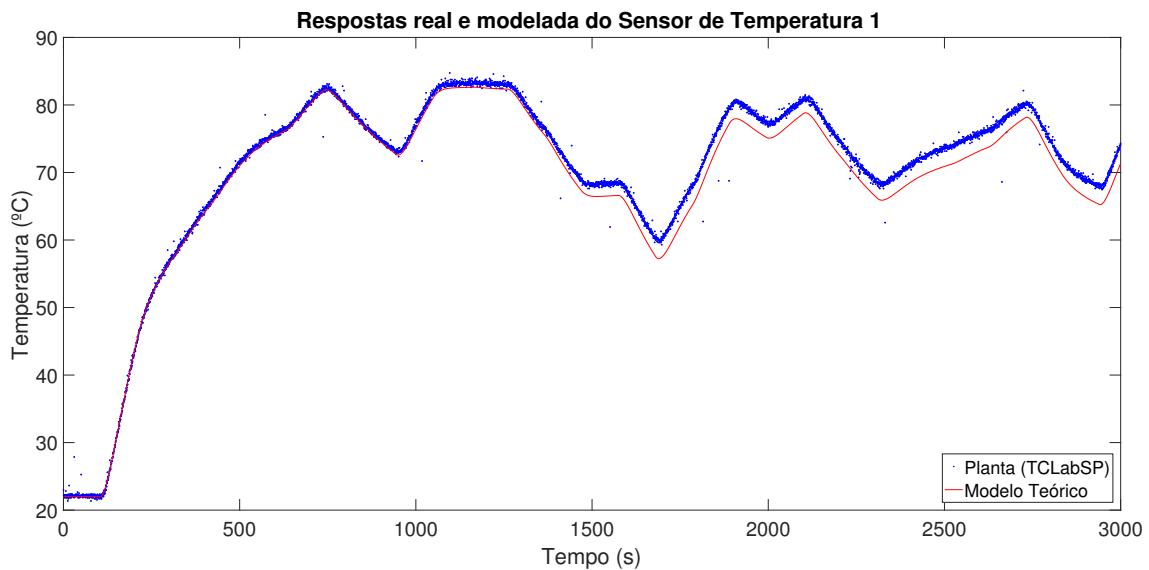
Para expressar numericamente quanto este modelo obtido se aproximou da planta real durante os experimentos, será utilizada a eq. (5.4) que também será apresentada com mais detalhes no capítulo 5. A tabela 4 apresenta esses valores.

Tabela 4 – Resultado numérico da comparação entre o modelo teórico e a planta real

Saída	fit
Sensor de temperatura 1	87.05%
Sensor de temperatura 2	67.32%

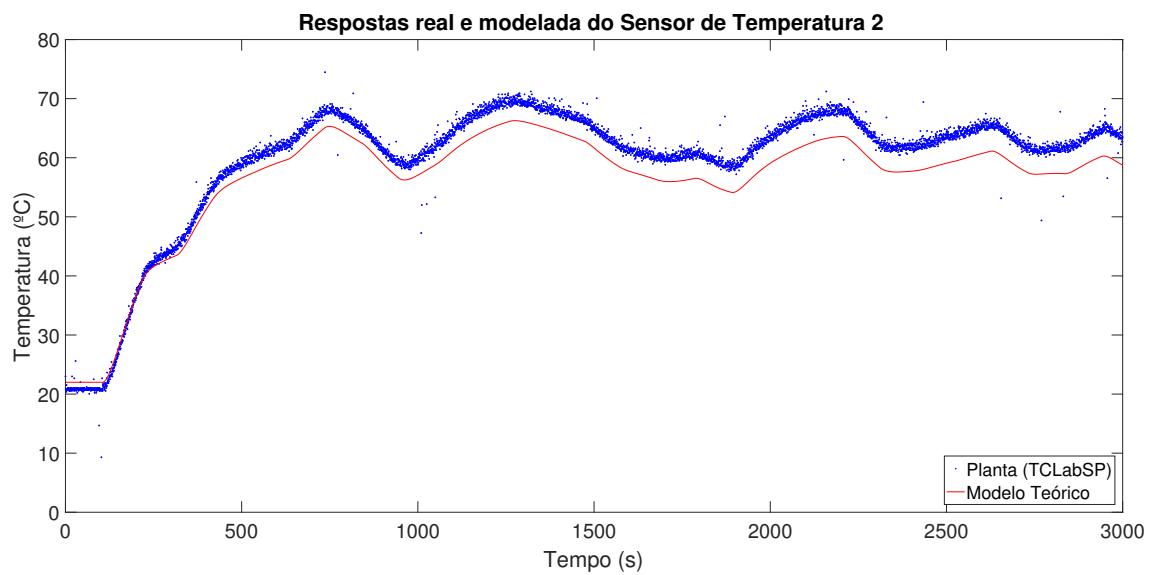
Fonte: Autor

Figura 19 – Respostas real e modelada do Sensor de Temperatura 1



Fonte: Prata (2019)

Figura 20 – Respostas real e modelada do Sensor de Temperatura 2



Fonte: Prata (2019)

5 Modelagem experimental

Segundo Gevers (2006) a teoria de identificação de sistemas data da década de 60 e as duas principais técnicas utilizadas na identificação de sistemas atualmente (método de identificação por subespaço de estados e método do erro de predição) tiveram suas bases estruturadas com os trabalhos de Ho e Kalman (1966 apud GEVERS, 2006) e de Åström e Torsten (1965 apud GEVERS, 2006).

De acordo com Dunia, Edgar e Haugen (2008 apud PRACEK et al., 2011) a identificação do sistema permite que simulações sejam desenvolvidas de modo a garantir um melhor desempenho dos sistemas de controle projetados.

As etapas para a identificação de um sistema, segundo Aguirre (2015) são:

- Testes dinâmicos e coleta de dados
- Escolha da representação matemática a ser utilizada
- Determinação da estrutura do modelos
- Estimação de parâmetros
- Validação do modelo

As seções a seguir detalham cada uma dessas etapas.

5.1 Testes dinâmicos e coleta de dados

Esta etapa abrange os procedimentos necessários para geração do conjunto de dados que serão utilizados para a identificação do sistema. Algumas das atividades realizadas nessa etapa são: escolha das variáveis, definição dos sinais de excitação, definição do período de amostragem e execução de testes. (AGUIRRE, 2015)

5.1.1 Período de amostragem

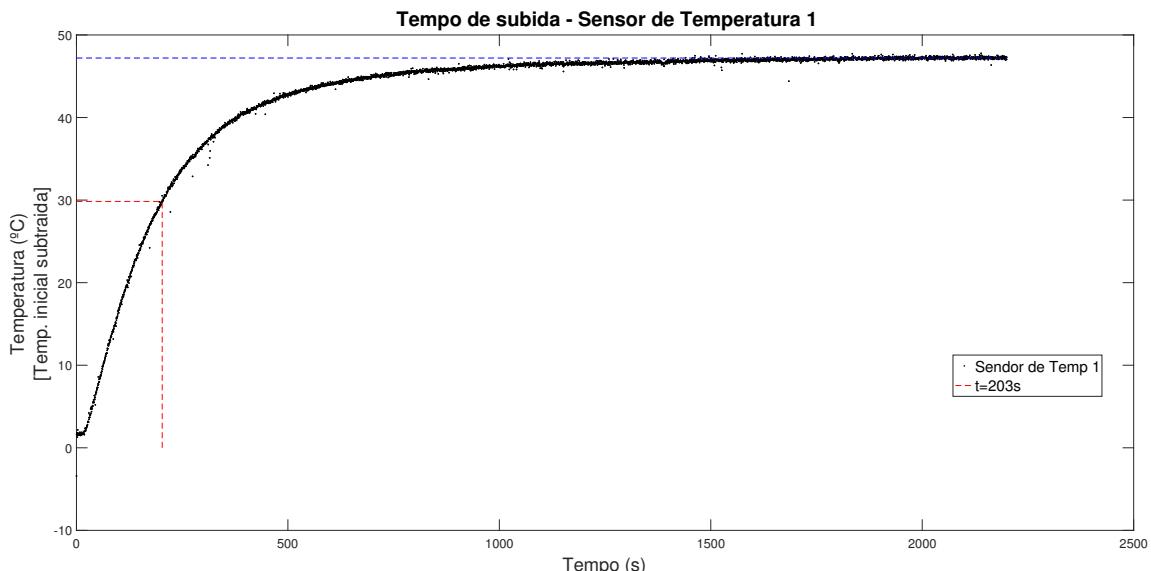
Segundo o teorema de Shannon, descrito por Aguirre (2015), "um sinal que não contenha componentes de frequência acima de $1/2T_s$ pode ser determinado unicamente a partir de amostras de tal sinal separadas por T_s ", sendo T_s o período de amostragem ou, como também é chamado, tempo de amostragem.

Na prática, contudo, um período de amostragem apenas 2 vezes maior que a frequência de interesse, como exigido pelo teorema de Shannon, nem sempre é suficiente.

Normalmente então escolhe-se uma frequência de amostragem entre 5 e 10 vezes maior do que a maior do que tal frequência desejada (AGUIRRE, 2015). Por outro lado, do ponto de vista numérico, se o intervalo de amostragem for muito curto, a estimativa de parâmetros também poderá se tornar malcondicionada (AGUIRRE, 2015).

Para a identificação da maior frequência de interesse do sistema analisado, alguns ensaios foram realizados¹ excitando o Aquecedor 1 do TCLabSP em 50% e aproximando a resposta em malha aberta dos Sensores de Temperatura 1 e 2 para um sistema de primeira ordem, a fim de obter o valor do tempo de subida (τ). Esse tempo de subida servirá de referência para o cálculo do período de amostragem.

Figura 21 – Tempo de subida - Sensor de Temperatura 1



Fonte: Prata (2019)

As figs. 21 e 22 mostram a resposta em malha aberta dos Sensores de Temperatura 1 e 2, respectivamente, a uma entrada degrau no Aquecedor 1. Este ensaio foi realizado com um período de amostragem de 0,5s, e com ele foi possível observar um tempo de subida $\tau_1 = 209\text{s}$ para o Sensor 1 e $\tau_2 = 383\text{s}$ para o Sensor 2.

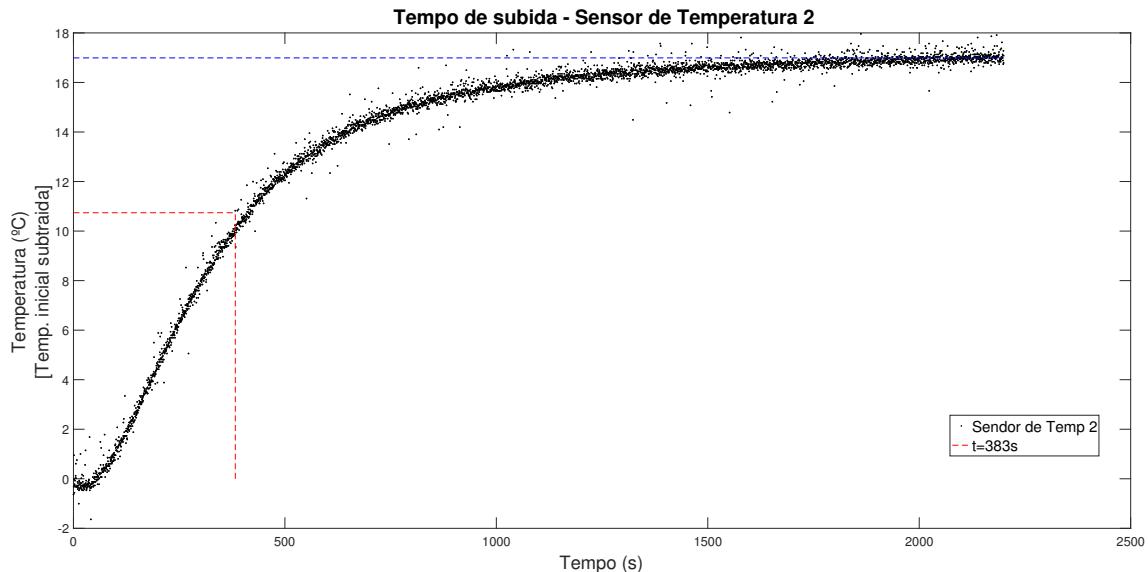
Seguindo a regra prática de Aguirre (2015) onde T_s deve ser entre 5 e 10 vezes maior que a maior frequência desejada, então temos que $20.9\text{s} > T_s > 41.8\text{s}$.

Para os ensaios realizados neste trabalho, optou-se pela maior frequência de amostragem possível, como mostrado na eq. (5.1).

$$T_s = 20.9\text{s} \quad (5.1)$$

¹ Scripts utilizados no apêndice A.6

Figura 22 – Tempo de subida - Sensor de Temperatura 2



Fonte: Prata (2019)

Outra técnica descrita por Aguirre (2015) para a escolha da frequência de amostragem também foi testada e é descrita em maiores detalhes no apêndice B, porém esta não foi escolhida pois optou-se pela técnica que apresentou menor valor de T_s .

5.1.2 Sinais de excitação

A escolha dos sinais de entrada pode ter grande impacto nos dados coletados, pois determinarão o ponto de operação do sistema e quais das suas características serão excitadas durante o experimento (AGUIRRE, 2015). Diversos tipos distintos de sinais podem ser utilizados. Entre os mais comuns destacam-se: impulsivo; degrau; PRBS - Sinal Binário Pseudo-Aleatório (do inglês, *Pseudo-Random Binary Signal*); GBN - Ruido Binário Generalizado (do inglês, *Generalized Binary Noise*); soma de senóides e etc (AGUIRRE, 2015). Porém quando objetiva-se a identificação do sistema de onde os dados foram coletados, no geral os métodos de identificação exigem que os sinais de entrada tenham um espectro de frequência branco ou quase branco. Sinais aleatórios e pseudoaleatórios satisfazem essa condição (AGUIRRE, 2015).

Sinais PRBS são comumente usados na identificação de sistemas lineares, porém para sistemas não-lineares tais sinais não são adequados, neste caso muitas vezes será desejável excitar o sistema numa larga faixa de amplitudes a fim de poder observar suas características estáticas e dinâmicas (AGUIRRE, 2015).

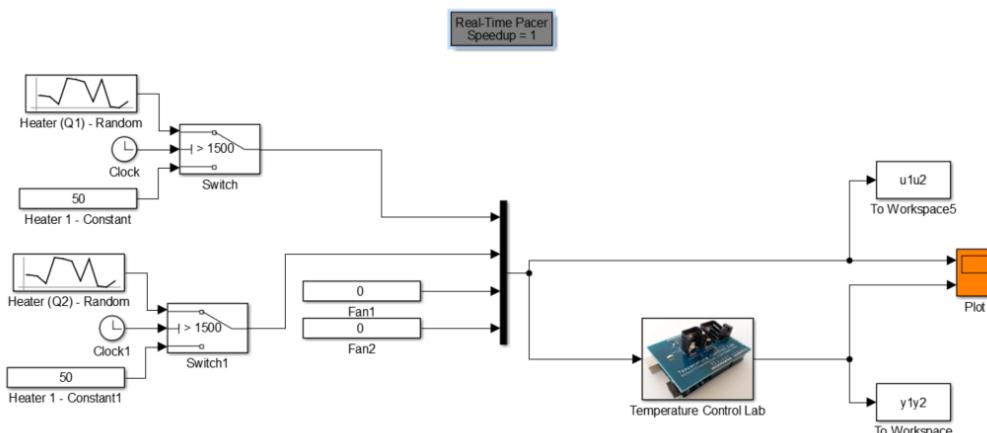
Características dinâmicas e estáticas que não forem excitadas não aparecerão nos dados. O que não estiver nos dados não pode ser identificado.

Tal princípio é utilizado, por exemplo, quando um sistema é excitado em torno de um ponto de operação. Neste caso, como as não-linearidades não foram excitadas no teste, é possível, pelo menos em princípio, ajustar um modelo linear aos dados assim obtidos. (AGUIRRE, 2015)

Segundo Aguirre (2015) uma regra prática para a aplicação do sinal de excitação é, tendo-se definido o tempo de amostragem (já discutido na seção 5.1.1), manter constante o valor escolhido aleatoriamente por um tempo, em torno de 3 a 5 intervalos de amostragem.

Para a coleta dos sinais do **TCLabSP** foram aplicados tanto no Aquecedor 1 quanto no Aquecedor 2 sinais de entrada aleatórios, mantidos constantes por um período de $3T_s$ a $5T_s$ (vide eq. (5.1)). Contudo, antes dos sinais aleatórios serem aplicados aos aquecedores, um sinal constante de 50% foi aplicado a cada um deles por 1500s com o intuito de estabilizá-los em um patamar onde as amplitudes dos sinais aleatórios aplicados tivessem maior liberdade. A fig. 23 mostra o modelo *Simulink* utilizado para a coleta dos dados e a fig. 24 mostra os sinais de entrada aplicados nos Aquecedores 1 e 2.

Figura 23 – Modelo *Simulink* para coleta de dados



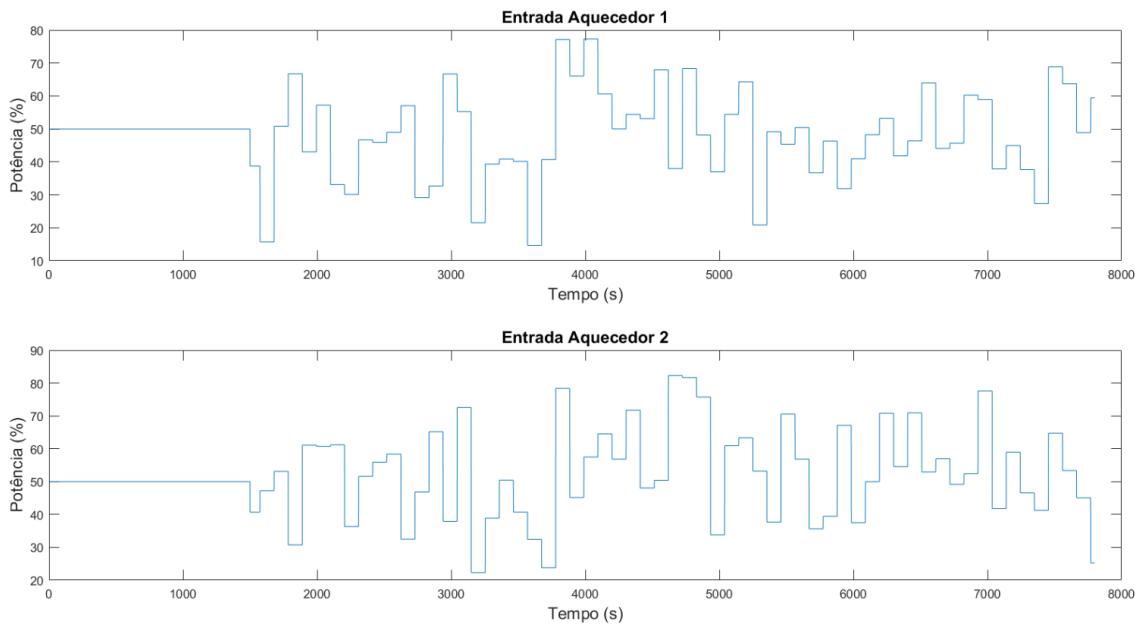
Fonte: Autor

5.1.3 Duração do experimento

A duração do experimento, segundo Garcia (2005), deveria ser a maior possível, uma vez que a variância das estimativas é proporcional ao inverso da duração do experimento, porém sob o ponto de vista prático e experimental, a duração do experimento deveria ser a menor possível para a obtenção de um modelo aceitável, pois ao longo do experimento o processo estará sujeito a perturbações extras que podem impactar na operação da planta, na qualidade dos produtos ou até na segurança do processo.

Cada um dos experimentos de coleta de dados do **TCLabSP** teve uma duração de 7800s, sendo que os 1500s iniciais foram utilizados para a estabilização do processo e os

Figura 24 – Sinais de entrada nos Aquecedores 1 e 2



Fonte: Autor

demais ($6300s$) para as coletas. O critério utilizado para a determinação deste valor foi o de estimular a dinâmica da planta aproximadamente $60x$ para cada um dos experimentos. Sabendo que os sinais de excitação podem permanecer constantes por até $105s$ ($5T_s$), como mostrado na [seção 5.1.2](#), então temos a [eq. \(5.2\)](#):

$$60 * 5T_s = 6300s \quad (5.2)$$

5.2 Definição do modelo experimental

Esta seção apresenta mais algumas características teóricas importantes na definição de um modelo que represente o sistema simulado e ao final dela, a seção 5.3 mostra a aplicação prática dessas características utilizando o conjunto de ferramentas do MATLAB® para identificação de sistemas, o *System Identification Toolbox™*.

5.2.1 Escolha da representação matemática

Existem diversas representações matemáticas distintas para modelos lineares, sendo que, segundo Aguirre (2015), a mais utilizada é a função de transferência, porém Wang (2009) destaca que nos últimos anos tem se observado um aumento na popularidade dos modelos de espaço de estados para desenvolvimento de controle preditivo. Para Aguirre (2015) é importante ainda salientar o modelo AR (Modelo autorregressivo), o

modelo ARX (Modelo autorregressivo com entrada exógena) e o modelo ARMAX (Modelo autorregressivo com média móvel e entrada exógena).

[...] quando se trata da modelagem obtida por meios fenomenológicos é comum que se adote a base de tempo contínuo, em virtude de a maioria das leis da física serem expressas nesse tempo. Por sua vez, quando se trata de identificação de sistemas por processos experimentais, trabalha-se com amostras de dados coletados a cada intervalo de tempo, nesses casos usualmente adota-se o tempo discreto. (GARCIA, 2005 apud FAVARO, 2012)

5.2.2 Determinação da estrutura do modelos

Determinar a ordem de um modelo é um dos aspectos mais importantes na determinação de sua estrutura, uma vez que, caso sua ordem seja muito menor do que a ordem efetiva do sistema real, o modelo não refletirá a completamente sua complexidade estrutural. Analogamente, escolher um modelo que a ordem seja muito maior do que a necessária, provavelmente causará uma estimativa de parâmetros mal condicionada. (AGUIRRE, 2015)

Neste trabalho será utilizado o *critério de informação de Akaike* (AKAIKE, 1974 apud AGUIRRE, 2015) (AIC), definido por:

$$AIC(n_\theta) = N \ln [\sigma_{erro}^2(n_\theta)] + 2n_\theta \quad (5.3)$$

Onde:

- N : é o número de dados
- $\sigma_{erro}^2(n_\theta)$: é a variância dos resíduos
- $n_\theta = \dim[\hat{\theta}]$: é o número de parâmetros do modelo

Segundo Aguirre (2015) a utilização deste critério na escolha da estrutura do modelo se justifica pois à medida que termos são incluídos no modelo, o número de graus de liberdade aumenta, permitindo um ajuste de dados mais exato. Resumidamente, a primeira parcela da equação quantifica a diminuição da variância dos resíduos resultante da inclusão de um termo, ao passo que a segunda parcela penaliza a inclusão de cada termo.

O índice $AIC(n_\theta)$ normalmente atinge um mínimo para um determinado número de parâmetros no modelo n_θ^* , e, ainda segundo Aguirre (2015), do ponto de vista do critério utilizado, esse número de parâmetros é ótimo.

5.2.3 Estimação de parâmetros

A estimação de parâmetros, segundo Eykhoff (1974 apud FAVARO, 2012) é a determinação experimental de valores de parâmetros que governam a dinâmica e/ou o comportamento não-linear, assumindo que a estrutura do modelo seja conhecida.

Essa etapa começa com a escolha do algoritmo a ser utilizado (AGUIRRE, 2015). Dentre eles, os mais amplamente empregados na literatura são: método da análise de frequência; método da resposta transitória e método dos mínimos quadrados (FAVARO, 2012).

Como será visto na seção 5.3, neste trabalho diversas representações matemáticas distintas foram testadas, e para cada uma delas foi utilizada uma função do MATLAB® para a estimação automática de parâmetros. A tabela 5 identifica as funções utilizadas em cada uma das representações distintas².

Tabela 5 – Funções para a estimação de parâmetros

Representação matemática	Função
Função de transferência	<code>tfest</code>
Espaço de estados	<code>ssest</code>
ARX	<code>arx</code>
ARMAX	<code>armax</code>
Erro de saída (OE, do inglês <i>Output-Error</i>)	<code>oe</code>
Box-Jenkins (BJ)	<code>bj</code>

Fonte: Autor

5.2.4 Validação do modelo

Em problemas de validação, a questão é tentar determinar se um dado modelo é válido ou não e para isso, deve-se simulá-lo sem qualquer ajuste adicional e compará-lo a dados medidos em testes diferentes daquele usado no desenvolvimento da sintonia do mesmo. A motivação para tal cuidado deve-se ao fato de desejar-se saber o quão geral é o modelo. Portanto, esta divisão entre os dados de treinamento e de validação refere-se à *capacidade de generalização do modelo* (AGUIRRE, 2015).

² Documentações das funções disponíveis em:

`tfest` - <<https://www.mathworks.com/help/ident/ref/tfest.html>>
`ssest` - <<https://www.mathworks.com/help/ident/ref/ssest.html>>
`arx` - <<https://www.mathworks.com/help/ident/ref/arx.html>>
`armax` - <<https://www.mathworks.com/help/ident/ref/armax.html>>
`oe` - <<https://www.mathworks.com/help/ident/ref/oe.html>>
`bj` - <<https://www.mathworks.com/help/ident/ref/bj.html>>

A proporção entre dados de treinamento e validação utilizada neste trabalho é de 60% para dados de treinamento e 40% para validação. Para realizar a comparação entre o modelo obtido e os dados de validação, utilizou-se a função `compare` no MATLAB®.

Esta função `compare` avalia o quanto o modelo obtido se encaixa aos dados de validação calculando o erro do quadrado médio da raiz normalizada, assim como descrito na eq. (5.4) a seguir:

$$\text{fit} = 100 \left(1 - \frac{\|y - \hat{y}\|}{\|y - \text{média}(y)\|} \right) \quad (5.4)$$

Onde:

- y : são os dados de validação
- \hat{y} : são os dados do modelo

Também neste trabalho aplica-se o teste de *análise de resíduos* dos modelos estudados e a motivação para tal pode ser melhor compreendida nas palavras de Aguirre (2015) a seguir:

[...] Do ponto de vista da validação de modelos, a motivação de se verificar quão são aleatórios os resíduos pode ser entendida lembrando que os resíduos são a parte dos dados que o modelo não consegue explicar. Exigir que o modelo explique os dados em todos os seus detalhes é, no mínimo, ingênuo. Mas quanto dos dados precisa ser explicado? O modelo deve explicar *tudo que for explicável nos dados*. Se isso ocorrer, então os resíduos conterão apenas aquilo que não é explicável e, portanto, serão brancos. Assim, se os resíduos forem brancos, não há informação útil neles, ou seja, o modelo explicou tudo que era possível explicar. [...]

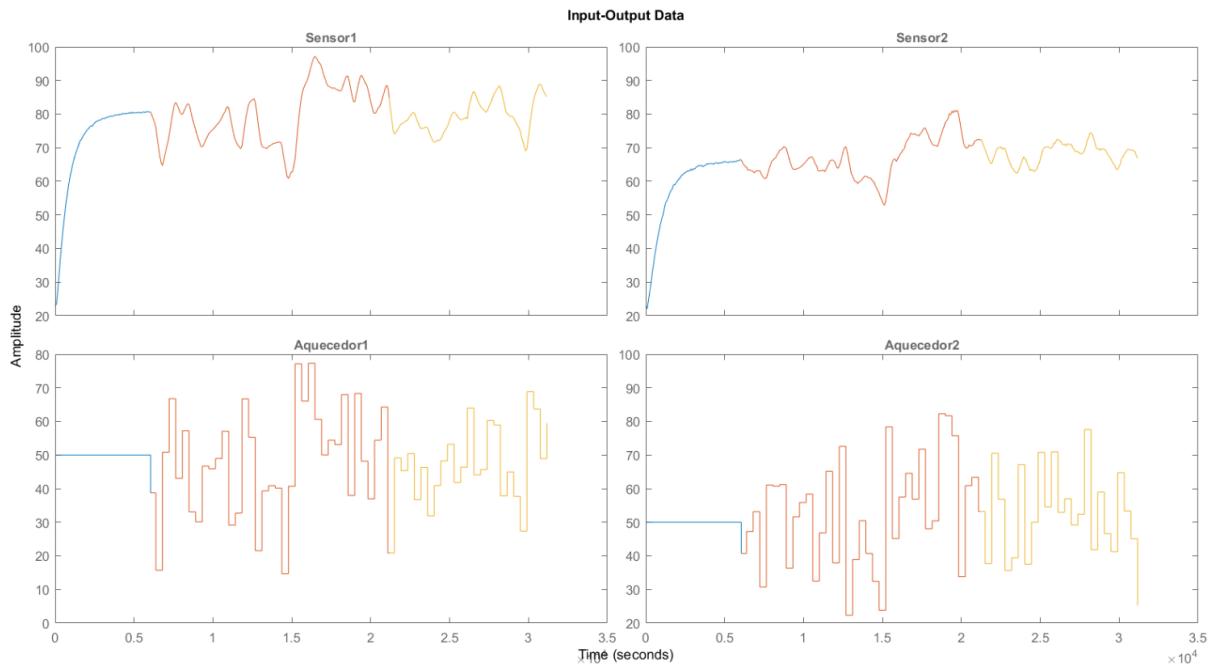
Para tal teste utiliza-se a função `resid` no MATLAB®.

5.3 Modelos experimentais

Esta seção apresenta maiores detalhes práticos dos procedimentos descritos nas seções 5.2.1 a 5.2.4 e os *scripts* de MATLAB® utilizados para a obtenção dos resultados também serão apresentados e discutidos ao longo do texto.

Após as coletas de dados descritas das seções 5.1 a 5.1.3 preparou-se então no MATLAB® um objeto do tipo `iddata` contendo os dados dos aquecedores e dos sensores de temperatura do TCLabSP. Esses dados foram divididos em 2 grupos: identificação e validação, como já visto na seção 5.2.4. As entradas e saídas do sistema, subdivididas nesses 2 grupos, foram criadas a partir do código-fonte 5.1 e podem ser observadas na fig. 25.

Figura 25 – Entradas e saídas coletadas em ensaios no TCLabSP



Fonte: Autor

Na fig. 25 é possível observar 3 conjuntos de dados distintos, indicados pelas cores azul, vermelho e laranja. Os dados em **azul** foram descartados, pois representam o período de estabilização do sistema em 50% dos aquecedores. Os dados em **vermelho** e **laranja** representam os dados de identificação e validação, respectivamente.

Código-fonte 5.1 – Criação dos conjuntos de identificação e validação para o sistema
TCLabSP

```

clear

disp(char(strcat('Inicio:', {' '}, datestr(datetime))));

%% Configuracoes
Experiments_path = './Ensaios';

Ts_actual = 0.25; % Tempo de amostragem atual
Ts_target = 21; % Tempo de amostragem desejado

estab_time = 1500; % Tempo para estabilizacao do sistema

%{
Relacao entre quant. de treinamento e validacao
(ex. 60% treinamento -> 40% validacao)
%}

```

```

%}

train_valid_rate = 0.6;

%% Carrega dados
ensaios_full = combinar_experimentos(Experiments_path);

% Ajusta tempo de amostragem para Ts = 21
Ts_rate = Ts_target/Ts_actual;
ensaios = ensaios_full(1:Ts_rate:end,:);

%% Criando modelo
tclabsp = iddata([ensaios.(‘Sensor1’), ensaios.(‘Sensor2’)], ...
                 [ensaios.(‘Aquecedor1’), ensaios.(‘Aquecedor2’)], ...
                 Ts_rate);
tclabsp.InputName = {‘Aquecedor1’; ‘Aquecedor2’};
tclabsp.InputUnit = {‘%’; ‘%’};
tclabsp.OutputName = {‘Sensor1’; ‘Sensor2’};
tclabsp.OutputUnit = {‘°C’; ‘°C’};
tclabsp.Tstart = 0;

%% Selecionando faixa de treinamento

% Quant de amostras, excluindo estabilizacao
data_length = length(find(ensaios.(‘Time’) >= estab_time));

% Encontra primeiro indice apos estabilizacao
train_start = find(ensaios.(‘Time’) >= estab_time, 1, ‘first’);
train_finish = train_start + round(data_length * train_valid_rate);

train = tclabsp(train_start:train_finish,:,:);

%% Selecionando faixa de validacao
valid_start = train_finish;
valid_finish = height(ensaios);

valid = tclabsp(valid_start:valid_finish,:,:);

%% Plotando planta + treinamento + validacao
plot(tclabsp,train,valid)

```

Fonte: Autor

Com o intuito fazer diversos testes e de avaliar muitos modelos diferentes, utilizou-se um *script* para a criação automática de modelos experimentais baseados nos dados mostrados na fig. 25. Este *script* utiliza as funções descritas na tabela 5 e seu código-fonte

pode ser visto no [apêndice A.5](#). A [tabela 6](#) apresenta a quantidade de modelos que foram gerados a partir dele.

Tabela 6 – Modelos experimentais do [TCLabSP](#)

Representação matemática	Quantidade
Função de transferência	10.000
Espaço de estados	10
ARX	9
ARMAX	36
OE	10
BJ	160

Fonte: Autor

Modelos experimentais de função de transferência

Os modelos experimentais criados na forma de função de transferência foram obtidos a partir de diversas combinações entre a quantidade de pólos e zeros nos subsistemas existentes entre os Sensores 1 e 2, e entre os Aquecedores 1 e 2. Essas combinações são descritas na [tabela 7](#).

Tabela 7 – Modelos experimentais - Função de transferência

	Aquecedor 1	Aquecedor 2
Sensor 1	pólos: de 2 a 5 zeros: de 1 a 4	pólos: de 2 a 5 zeros: de 1 a 4
Sensor 2	pólos: de 2 a 5 zeros: de 1 a 4	pólos: de 2 a 5 zeros: de 1 a 4

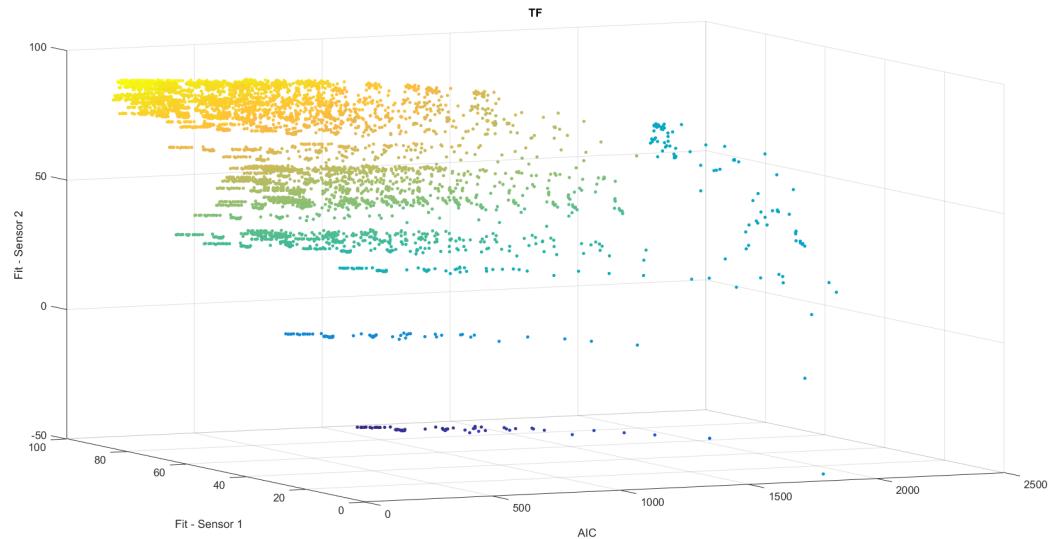
Fonte: Autor

Para cada uma das combinações possíveis descritas na [tabela 7](#) foi criado um modelo utilizando o conjunto de dados de identificação. Esses modelos foram testados com o conjunto de dados de validação através da [eq. \(5.4\)](#) e o resultado destes testes, juntamente com o critério de [AIC](#), podem ser observados na [fig. 26](#).

Cada um dos pontos presentes na [fig. 26](#) representam um modelo experimental diferente, sendo que as cores mais claras indicam aqueles com menor erro quando comparados ao conjunto de dados de validação.

A fim de escolher o melhor modelo de função de transferência frente aos 10000 gerados, os 10 primeiros modelos com menor valor segundo o critério de [AIC](#) foram submetidos à análise de resíduos (vide [seção 5.2.4](#)) e dentre eles o que mostrou menor valor na análise residual foi o modelo descrito na [tabela 8](#). O teste deste modelo junto

Figura 26 – Modelos experimentais de função de transferência



Fonte: Autor

aos dados de validação e sua análise de resíduos podem ser visualizadas nas figs. 41 e 42, respectivamente.

Tabela 8 – Melhor modelo experimental - Função de transferência

Entradas e saídas	Função de transferência
Aquecedor 1 Sensor 1	$\frac{0.1248z^{-1} - 0.01046z^{-2} - 0.1142z^{-3}}{1 - 0.8092z^{-1} - 1.03z^{-2} + 0.7405z^{-3} + 0.1342z^{-4} - 0.03503z^{-5}}$
Aquecedor 1 Sensor 2	$\frac{0.007869z^{-1} - 0.007693z^{-2}}{1 - 1.985z^{-1} + 0.2452z^{-2} + 1.659z^{-3} - 1.083z^{-4} + 0.1637z^{-5}}$
Aquecedor 2 Sensor 1	$\frac{0.00452z^{-1} + 0.005953z^{-2}}{1 - 0.2164z^{-1} - 1.301z^{-2} + 0.2197z^{-3} + 0.4412z^{-4} - 0.08218z^{-5}}$
Aquecedor 2 Sensor 2	$\frac{0.06374z^{-1} - 0.04266z^{-2}}{1 - 1.504z^{-1} + 0.4851z^{-2} + 0.2172z^{-3} - 0.2665z^{-4} + 0.12z^{-5}}$

Fonte: Autor

Modelos experimentais de espaço de estados

Para a escolha do modelo experimental em espaço de estados foram testados 10 modelos de ordens distintas, entre as ordens 1 e 10, e gráfico mostrando a comparação destes modelos frente ao critério de AIC e aos dados de validação dos sensores 1 e 2 pode ser observado na fig. 43.

Dentre todos os testados, o modelo de ordem 3 apresentou o menor valor no critério de **AIC** e foi selecionado. O formato e as matrizes deste modelo podem ser vistos na eq. (5.5) e o gráfico de teste junto aos dados de validação na fig. 44.

$$\begin{aligned} x(t + T_s) &= Ax(t) + Bu(t) + Ke(t) \\ y(t) &= Cx(t) + Du(t) + e(t) \end{aligned} \quad (5.5)$$

$$A = \begin{matrix} x_1 & x_2 & x_3 \\ \begin{bmatrix} 0.9099 & -0.02543 & -0.02673 \\ -0.06169 & 0.8193 & -0.01098 \\ -0.03506 & 0.108 & 0.9884 \end{bmatrix}, B = \begin{matrix} Aquecedor_1 & Aquecedor_2 \\ \begin{bmatrix} 0.0005512 & 7.033 \times 10^{-5} \\ 0.002047 & -0.001209 \\ -0.0009257 & 0.0009081 \end{bmatrix}, \\ C = \begin{matrix} Sensor_1 & Sensor_2 \\ \begin{bmatrix} 180.1 & 8.69 & 1.814 \\ 159.2 & -40.71 & 1.665 \end{bmatrix}, D = \begin{matrix} Aquecedor_1 & Aquecedor_2 \\ \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}, \\ K = \begin{matrix} Sensor_1 & Sensor_2 \\ \begin{bmatrix} 0.000946 & 0.0008267 \\ 0.0006276 & 0.001956 \\ -0.008009 & 0.003641 \end{bmatrix} \end{matrix} \end{matrix} \end{matrix}$$

Modelos experimentais ARX

Os modelos **ARX** foram criados a partir de combinações de diferentes números de pólos (n_a), números de zeros (n_b) e de tempo morto (n_k). As combinações utilizadas podem ser vistas na tabela 9.

Tabela 9 – Modelos experimentais - ARX

Parâmetro	Valores utilizados nas combinações
n_a	$\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \begin{bmatrix} 2 & 2 \\ 2 & 2 \end{bmatrix}, \begin{bmatrix} 3 & 3 \\ 3 & 3 \end{bmatrix}, \begin{bmatrix} 4 & 4 \\ 4 & 4 \end{bmatrix}$
n_b	$\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \begin{bmatrix} 2 & 2 \\ 2 & 2 \end{bmatrix}, \begin{bmatrix} 3 & 3 \\ 3 & 3 \end{bmatrix}$
n_k	$\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$

Fonte: Autor

O gráfico mostrando a distribuição dos modelos gerados em comparação ao critério de **AIC** pode ser visto na fig. 45 e o modelo **ARX** escolhido, seu gráfico comparativo aos

dados validação e sua análise residual podem ser vistos na tabela 10, na fig. 46 e na fig. 47, respectivamente.

Tabela 10 – Melhor modelo experimental - ARX

Saída	Parâmetros
$Sensor_1$	<p>Formato:</p> $A(z)y_1(t) = -A_i(z)y_i(t) + B(z)u(t) + e_1(t)$ <p>Onde:</p> $A(z) = 1 - 1.308z^{-1} + 0.1507z^{-2} + 0.3109z^{-3} - 0.07434z^{-4}$ $A_2(z) = -0.2768z^{-1} + 0.03432z^{-2} + 0.1842z^{-3} - 0.02748z^{-4}$ $B1(z) = 0.07487z^{-1} - 0.02z^{-2} - 0.01542z^{-3}$ $B2(z) = -0.00504z^{-1} - 0.006801z^{-2} - 0.01316z^{-3}$
$Sensor_2$	<p>Formato:</p> $A(z)y_2(t) = -A_i(z)y_i(t) + B(z)u(t) + e_2(t)$ <p>Onde:</p> $A(z) = 1 - 0.4098z^{-1} - 0.2493z^{-2} + 0.0703z^{-3} - 0.1442z^{-4}$ $A_1(z) = -0.9008z^{-1} + 0.257z^{-2} + 0.5452z^{-3} - 0.1428z^{-4}$ $B1(z) = -0.02857z^{-1} - 0.06333z^{-2} - 0.03838z^{-3}$ $B2(z) = 0.05223z^{-1} + 0.025z^{-2} + 0.01699z^{-3}$

Fonte: Autor

Modelos experimentais ARMAX

De forma análoga ao procedimento realizado para a obtenção dos modelos ARX, os modelos ARMAX também foram criados a partir de combinações de diferentes números de pólos (n_a), números de zeros mais 1 (n_b), número de coeficientes C (n_c) e de tempo morto (n_k). As combinações utilizadas podem ser vistas na tabela 11.

A partir de todos os modelos possíveis gerados a partir das combinações dos parâmetros da tabela 11, selecionaram-se os 10 modelos com menor valor segundo o critério de AIC e a partir destes foi então selecionado o modelo com menor valor residual. O gráfico mostrando a distribuição dos modelos gerados em comparação ao critério de AIC pode ser visto na fig. 48 e o modelo ARMAX escolhido, seu gráfico comparativo aos dados validação e sua análise residual podem ser vistos na tabela 12, na fig. 49 e na fig. 50, respectivamente.

Modelos experimentais OE

Os modelos OE foram criados a partir de combinações de diferentes ordens do polinômio B+1 (n_b), ordens do polinômio F (n_f) e de tempo morto (n_k). As combinações

Tabela 11 – Modelos experimentais - ARMAX

Parâmetro	Valores utilizados nas combinações
n_a	$\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \begin{bmatrix} 2 & 2 \\ 2 & 2 \end{bmatrix}, \begin{bmatrix} 3 & 3 \\ 3 & 3 \end{bmatrix}, \begin{bmatrix} 4 & 4 \\ 4 & 4 \end{bmatrix}$
n_b	$\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \begin{bmatrix} 2 & 2 \\ 2 & 2 \end{bmatrix}, \begin{bmatrix} 3 & 3 \\ 3 & 3 \end{bmatrix}$
n_c	$\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \begin{bmatrix} 2 & 2 \\ 2 & 2 \end{bmatrix}, \begin{bmatrix} 3 & 3 \\ 3 & 3 \end{bmatrix}, \begin{bmatrix} 4 & 4 \\ 4 & 4 \end{bmatrix}$
n_k	$\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$

Fonte: Autor

Tabela 12 – Melhor modelo experimental - ARMAX

Saída	Parâmetros
$Sensor_1$	<p>Formato:</p> $A(z)y_1(t) = -A_i(z)y_i(t) + B(z)u(t) + C(z)e_1(t)$ <p>Onde:</p> $A(z) = 1 - 1.804z^{-1} + 0.8137z^{-2}$ $A_2(z) = -0.09524z^{-1} + 0.08503z^{-2}$ $B1(z) = 0.1164z^{-1} - 0.1108z^{-2}$ $B2(z) = 0.004773z^{-1} - 0.008481z^{-2}$ $C(z) = 1 - 1.683z^{-1} + 0.6901z^{-2}$
$Sensor_2$	<p>Formato:</p> $A(z)y_2(t) = -A_i(z)y_i(t) + B(z)u(t) + C(z)e_2(t)$ <p>Onde:</p> $A(z) = 1 - 1.72z^{-1} + 0.7437z^{-2}$ $A_1(z) = -0.1496z^{-1} + 0.1278z^{-2}$ $B1(z) = 0.003022z^{-1} - 0.01553z^{-2}$ $B2(z) = 0.06256z^{-1} - 0.05434z^{-2}$ $C(z) = 1 - 1.67z^{-1} + 0.6744z^{-2}$

Fonte: Autor

utilizadas podem ser vistas na [tabela 13](#).

A partir de todos os modelos gerados a partir das combinações dos parâmetros da [tabela 13](#), ordenaram-se os modelos com menor valor segundo o critério de [AIC](#) e a partir destes foi então selecionado o modelo com menor valor residual. O gráfico mostrando a distribuição dos modelos gerados em comparação ao critério de [AIC](#) pode ser visto na

Tabela 13 – Modelos experimentais - OE

Parâmetro	Valores utilizados nas combinações
n_b	$\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \begin{bmatrix} 2 & 2 \\ 2 & 2 \end{bmatrix}, \begin{bmatrix} 3 & 3 \\ 3 & 3 \end{bmatrix}, \begin{bmatrix} 4 & 4 \\ 4 & 4 \end{bmatrix}$
n_f	$\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \begin{bmatrix} 2 & 2 \\ 2 & 2 \end{bmatrix}, \begin{bmatrix} 3 & 3 \\ 3 & 3 \end{bmatrix}, \begin{bmatrix} 4 & 4 \\ 4 & 4 \end{bmatrix}$
n_k	$\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$

Fonte: Autor

fig. 51 e o modelo OE escolhido, seu gráfico comparativo aos dados validação e sua análise residual podem ser vistos na tabela 14, na fig. 52 e na fig. 53, respectivamente.

Tabela 14 – Melhor modelo experimental - OE

Saída	Parâmetros
$Sensor_1$	<p>Formato:</p> $y_1(t) = \frac{B(z)}{F(z)}u(t) + e_1(t)$ <p>Onde:</p> $B1(z) = 0.08894z^{-1} + 0.04175z^{-2} - 0.09187z^{-3}$ $B2(z) = 0.008258z^{-1} + 0.001846z^{-2} - 0.01011z^{-3}$ $F1(z) = 1 - 0.912z^{-1} - 0.5633z^{-2} + 0.5215z^{-3}$ $F2(z) = 1 - 0.9537z^{-1} - 0.9783z^{-2} + 0.932z^{-3}$
$Sensor_2$	<p>Formato:</p> $y_2(t) = \frac{B(z)}{F(z)}u(t) + e_2(t)$ <p>Onde:</p> $B1(z) = -0.005385z^{-1} + 0.02174z^{-2} - 0.01629z^{-3}$ $B2(z) = 0.06252z^{-1} - 0.08437z^{-2} + 0.0602z^{-3}$ $F1(z) = 1 - 2.394z^{-1} + 1.824z^{-2} - 0.4304z^{-3}$ $F2(z) = 1 - 2.217z^{-1} + 2.138z^{-2} - 0.8374z^{-3}$

Fonte: Autor

Modelos experimentais Box-Jenkins

Os modelos BJ foram criados a partir de combinações de diferentes ordens do polinômio B+1 (n_b), ordens do polinômio C+1 (n_c), ordens do polinômio D+1 (n_d), ordens

do polinômio F+1 (n_f) e do tempo morto (n_k). As combinações utilizadas podem ser vistas na [tabela 15](#).

Tabela 15 – Modelos experimentais - BJ

Parâmetro	Valores utilizados nas combinações
n_b	$\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \begin{bmatrix} 2 & 2 \\ 2 & 2 \end{bmatrix}, \begin{bmatrix} 3 & 3 \\ 3 & 3 \end{bmatrix}, \begin{bmatrix} 4 & 4 \\ 4 & 4 \end{bmatrix}$
n_c	$\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \begin{bmatrix} 2 & 2 \\ 2 & 2 \end{bmatrix}, \begin{bmatrix} 3 & 3 \\ 3 & 3 \end{bmatrix}, \begin{bmatrix} 4 & 4 \\ 4 & 4 \end{bmatrix}$
n_d	$\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \begin{bmatrix} 2 & 2 \\ 2 & 2 \end{bmatrix}, \begin{bmatrix} 3 & 3 \\ 3 & 3 \end{bmatrix}, \begin{bmatrix} 4 & 4 \\ 4 & 4 \end{bmatrix}$
n_f	$\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \begin{bmatrix} 2 & 2 \\ 2 & 2 \end{bmatrix}, \begin{bmatrix} 3 & 3 \\ 3 & 3 \end{bmatrix}, \begin{bmatrix} 4 & 4 \\ 4 & 4 \end{bmatrix}$
n_k	$\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$

Fonte: Autor

A partir de todos os modelos possíveis gerados a partir das combinações dos parâmetros da [tabela 15](#), selecionaram-se os 10 modelos com menor valor segundo o critério de [AIC](#) e a partir destes foi então selecionado o modelo com menor valor residual. O gráfico mostrando a distribuição dos modelos gerados em comparação ao critério de [AIC](#) pode ser visto na [fig. 54](#) e o modelo **BJ** escolhido, seu gráfico comparativo aos dados validação e sua análise residual podem ser vistos na [tabela 16](#), na [fig. 55](#) e na [fig. 56](#), respectivamente.

5.3.1 Comparativo

Comparando todos os modelos experimentais escolhidos nos diferentes formatos é possível verificar que, apesar de grandes diferenças estruturais em comparação ao modelo teórico da [eq. \(4.3\)](#), os modelos experimentais apresentaram em geral resultados satisfatórios quando submetidos a testes de validação, com exceção apenas do modelo **ARX** que teve um desempenho médio aproximadamente 30% inferior aos demais modelos avaliados. Um gráfico comparativo com todos os modelos experimentais escolhidos pode ser observado na [fig. 27](#) e na [tabela 17](#) é apresentada uma consolidação dos valores de aproximação aos dados de validação para cada um dos modelos.

Tabela 16 – Melhor modelo experimental - BJ

Saída	Parâmetros
<i>Sensor</i> ₁	<p>Formato:</p> $y_1(t) = \frac{B(z)}{F(z)}u(t) + \frac{C(z)}{D(z)}e_1(t)$ <p>Onde:</p> $B1(z) = 0.1123z^{-1} - 0.07415z^{-2}$ $B2(z) = 0.007474z^{-1} - 0.005786z^{-2}$ $C(z) = 1 - 0.7216z^{-1} - 0.1974z^{-2}$ $D(z) = 1 - z^{-1}$ $F1(z) = 1 - 1.531z^{-1} + 0.5793z^{-2}$ $F2(z) = 1 - 1.866z^{-1} + 0.8761z^{-2}$
<i>Sensor</i> ₂	<p>Formato:</p> $y_2(t) = \frac{B(z)}{F(z)}u(t) + \frac{C(z)}{D(z)}e_2(t)$ <p>Onde:</p> $B1(z) = 0.00393z^{-1} + 0.001759z^{-2}$ $B2(z) = 0.06455z^{-1} - 0.02444z^{-2}$ $C(z) = 1 - 0.8508z^{-1} - 0.09047z^{-2}$ $D(z) = 1 - z^{-1}$ $F1(z) = 1 - 1.678z^{-1} + 0.6984z^{-2}$ $F2(z) = 1 - 1.218z^{-1} + 0.3003z^{-2}$

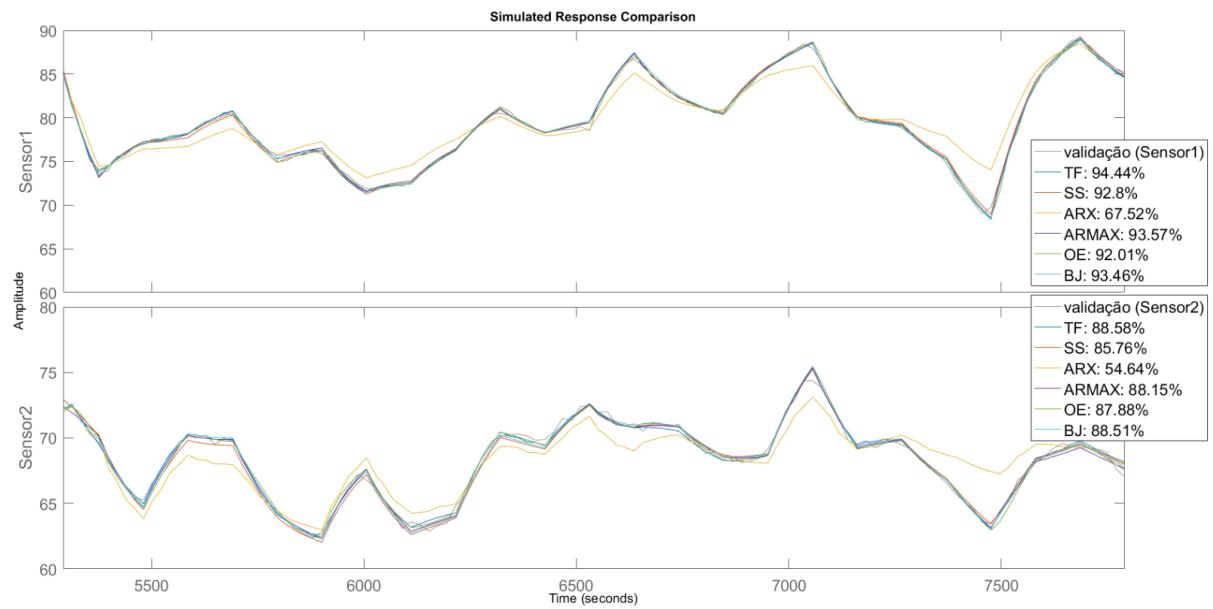
Fonte: Autor

Tabela 17 – Qualidade dos modelos experimentais

Modelo	Fit Sensor 1	Fit Sensor 2
TF (tabela 8)	94.44%	88.58%
BJ (tabela 16)	93.46%	88.51%
ARMAX (tabela 12)	93.57%	88.15%
OE (tabela 14)	92.01%	87.88%
SS (eq. (5.5))	92.80%	85.76%
ARX (tabela 10)	67.52%	54.64%

Fonte: Autor

Figura 27 – Comparativo dos modelos experimentais



Fonte: Autor

6 Desenvolvimento do controlador

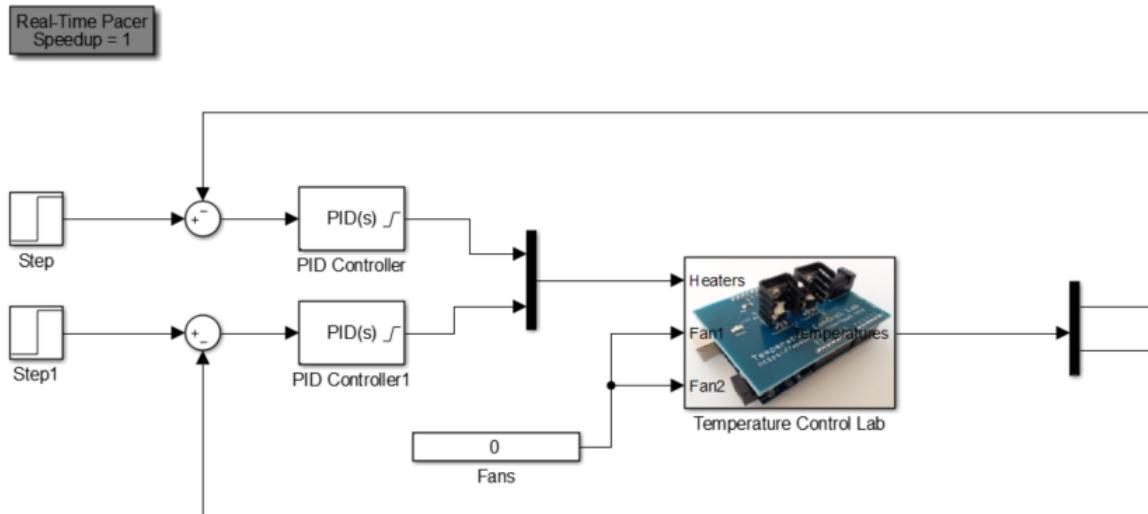
Com o intuito de controlar o **TCLabSP** aplicando o **MPC** como técnica, utilizaram-se os modelos teóricos e experimentais apresentados na [seção 4.3](#) e na [seção 5.3](#) para o desenvolvimento do bloco controlador no *Simulink*. Um controlador **PID** também foi desenvolvido a fim de servir como base comparativa ao controlador **MPC** criado. A seção a seguir descreve este controlador **PID**.

6.1 Controlador PID

O controlador **PID** desenvolvido foi criado em ambiente *Simulink*, utilizando o aplicativo *PID Tuner* do **MATLAB®**, disponível através do pacote *Control System Toolbox*.

A cada um dos Aquecedores do **TCLabSP** foi conectado a um bloco *PID Controller* retroalimentado com a saída dos Sensores de Temperatura da planta, conforme ilustrado pela [fig. 28](#).

Figura 28 – Modelo Simulink do controlador PID



Fonte: Autor

Os blocos *PID Controller* foram inicialmente configurados com os limites de saturação de saída entre 0 e 100 para que nenhum valor fora desta faixa fosse aplicado a planta controlada. Em seguida, uma vez que o desenvolvimento do controlador **PID** não

era um objetivo deste trabalho, utilizou-se a função de auto-sintonia do *PID Tuner* para encontrar valores satisfatórios de controle do sistema.

A tabela 18 apresenta os valores de sintonia encontrados para um controlador PID paralelo (eq. (6.1)).

$$P + I \frac{1}{s} + D \frac{N}{1 + N \frac{1}{s}} \quad (6.1)$$

Tabela 18 – Sintonia dos blocos PID

Parâmetro	PID Aquecedor 1	PID Aquecedor 2
Proporcional (P)	1.435	2.182
Integral (I)	0.016	0.011
Derivativo (D)	-9.119	-30.919
Filtro (N)	0.0216	0.010

Fonte: Autor

6.2 Controlador MPC

Segundo Alhajeri e Soroush (2020), o grande número de parâmetros sintonizáveis motivou diversos estudos sobre como sintonizar o MPC e nesta seção serão indicadas as técnicas utilizadas na sintonia de cada um dos parâmetros do controle MPC em questão.

6.2.1 Horizonte de predição

A janela de tempo que especifica quantos períodos de amostragem futuros sobre os quais a resposta da variável controlada y_i será predita pelo modelo da planta e então otimizada é chamado de horizonte de predição (ALHAJERI; SOROUSH, 2020). Este horizonte é indicado pela letra P e seus limites inferiores e superiores são, respectivamente, $P_{0_{n_y}}$ e P_{n_y} . Importante notar que o horizonte de predição é calculado para cada saída n_y , portanto em um sistema com 2 saídas, por exemplo, os limites inferiores serão P_{0_1} e P_{0_2} e os limites superiores P_1 e P_2 .

No seu estudo sobre as diferentes técnicas de sintonia do MPC, Alhajeri e Soroush (2020) indicam que, no geral, o horizonte de predição deve ser adequadamente grande para que as previsões das saídas controladas possam representar uma porção significativa do processo, portanto valores altos devem ser selecionados para P_1, \dots, P_{n_y} , para assegurar a estabilidade e robustez em malha fechada. Contudo, quanto maiores forem estes valores, maior será o custo computacional para a resolução do problema de otimização do MPC. Na prática, aumentar os valores de P_1, \dots, P_{n_y} aumenta a robustez e a estabilidade, porém após certos valores o aumento desta janela de predição não altera a performance

de controle significativamente, mas irá intensificar a carga computacional, o que pode deteriorar a robustez do controle.

As eqs. (6.2) e (6.3), sugeridas por Alhajeri e Soroush (2020), indicam os valores de limite inferior do horizonte de predição para plantas sem atraso e com atraso, respectivamente.

$$P_{0_1} = 1, \dots, P_{0_{n_y}} = 1 \quad (6.2)$$

$$P_{0_1} = 1 + \frac{\min_j (\sum_{j=1}^{n_u} \theta_{1_j})}{t_s}, \dots, P_{0_{n_y}} = 1 + \frac{\min_j (\sum_{j=1}^{n_u} \theta_{n_{y_j}})}{t_s} \quad (6.3)$$

Onde:

- n_u : são as entradas do sistema
- θ_i : é o tempo morto da planta
- t_s : é o período de amostragem

Os mesmos autores indicam estudos que sugerem que o limite superior seja calculados segundo as eqs. (6.4) e (6.5), sendo a primeira sem a utilização de tempo morto, e a segunda, considerando o tempo morto da planta.

$$P_1 = \frac{0.8 * T_s}{t_s}, \dots, P_{n_y} = \frac{0.8 * T_s}{t_s} \quad (6.4)$$

$$\begin{aligned} \frac{\max_j (\sum_{j=1}^{n_u} \theta_{1_j})}{t_s} + 1 &\leq P_1 \leq \frac{\max_j (\sum_{j=1}^{n_u} \theta_{1_j})}{t_s} + \max_J \left(\sum_{j=1}^{n_u} M_j \right), \dots, \\ \frac{\max_j (\sum_{j=1}^{n_u} \theta_{n_{y_j}})}{t_s} + 1 &\leq P_{n_y} \leq \frac{\max_j (\sum_{j=1}^{n_u} \theta_{n_{y_j}})}{t_s} + \max_J \left(\sum_{j=1}^{n_u} M_j \right) \end{aligned} \quad (6.5)$$

Onde:

- n_u : são as entradas do sistema
- θ_i : é o tempo morto da planta
- t_s : é o período de amostragem
- T_s : é o maior tempo de acomodação das variáveis controladas em malha aberta
- M_i : é o horizonte de controle

6.2.2 Horizonte de controle

O horizonte de controle, representado por M_i , são os valores da variável manipulada que o controlador calcula a cada instante de tempo, ou seja, $u_i(k), \dots, u_i(k + M_i - 1)$. O incremento deste horizonte aumenta também a agressividade, a habilidade para estabilizar plantas instáveis e o custo computacional, porém diminui a robustez do sistema de controle (ALHAJERI; SOROUSH, 2020).

Segundo o levantamento de Alhajeri e Soroush (2020), diferentes estudos apontaram que o ajuste recomendado para o horizonte de controle seria o indicado na eq. (6.6), pois não foi observada nenhuma melhoria expressiva da performance do controle em malha fechada com valores superiores a este.

$$\begin{aligned} M_i &\leq 2, \\ i &= 1, \dots, n_u \end{aligned} \tag{6.6}$$

6.2.3 Matrizes de peso

Além dos horizontes de predição e de controle, as matrizes de peso Q , R e Δ também são parâmetros que necessitam ser ajustados para aprimorar a sintonia do controlador MPC.

A matriz Q configura a penalidade nos erros das saídas. Os elementos dessa matriz penalizam o desvio das saídas em comparação a suas trajetórias de referência. Eles refletem o custo relativo das variáveis controladas desviando de suas trajetórias de referência e, portanto, dos seus *set-points* (ALHAJERI; SOROUSH, 2020).

As matrizes R e Δ descrevem a penalidade na taxa de variação das entradas e na magnitude das mesmas, respectivamente. O valor relativo dos elementos de Δ são um indicativo do custo relativo das entradas manipuladas (ALHAJERI; SOROUSH, 2020).

Segundo Alhajeri e Soroush (2020), normalmente assume-se que $\Delta = 0$, a menos que uma diretriz de sintonia diferente seja mencionada. Já para as matrizes Q e R , muitos estudos indicam que $Q = I$ e $R = \rho I$, onde ρ é um escalar também conhecido como *fator de supressão de movimento*. Esta abordagem, de encontrar um valor para ρ ao invés de encontrar as matrizes de peso, ao mesmo tempo que simplifica a tarefa de sintonizar o controlador MPC, retira certa flexibilidade do controle e limita o grau de qualidade que o MPC pode prover.

Um dos estudos apresentados por Alhajeri e Soroush (2020) sugere a utilização de $\rho = 0.01$ para um controle MPC sem restrições e $\rho = 0.1$ para um controle com restrições.

Dessa forma as matrizes utilizadas neste trabalho são indicadas na eq. (6.7).

$$\begin{aligned}\Delta &= 0 \\ Q &= I \\ R &= \rho I,\end{aligned}\tag{6.7}$$

sendo: $\rho = 0.01$ (MPC sem restrições)

$\rho = 0.1$ (MPC com restrições)

6.2.4 Desenvolvimento do controlador no MATLAB

Utilizando as eqs. (6.2), (6.4), (6.6) e (6.7), foi possível desenvolver um *script* no MATLAB® para a criação de um controlador MPC distinto para cada um dos modelos experimentais listados na tabela 17, e também para o modelo teórico apresentado na eq. (4.3). Este *script* pode ser visualizado no código-fonte 6.1.

Código-fonte 6.1 – Criação dos controladores MPC

```

clear

%% Carrega modelo teorico

LoadModel_FirstPrinciples
clearvars -except TCLab_Teo_ss

%% Carrega modelo experimental

LoadModel_Experimental
clearvars -except TCLab_Teo_ss TCLab_Exp_tf TCLab_Exp_ss ...
                  TCLab_Exp_bj TCLab_Exp_oe TCLab_Exp_arx
                  TCLab_Exp_armax

%% Configuracoes genericas do MPC

Ts = 7;
Max_DeadTime = 0;
Max_SettlingTime = 2000;
MV1 = struct('Min',0,'Max',100);
MV2 = struct('Min',0,'Max',100);
M = 2;
P = 0.8 * Max_SettlingTime / Ts;

% Parametros para MPC sem constraints
Weights = struct('ManipulatedVariables', [0 0], ...
                  'ManipulatedVariablesRate', [0.01 0.01], ...
                  'OutputVariables', [1 1], ...

```

```
'ECR', 100000);  
  
%% Cria MPC com modelo teorico  
  
MPC_tclab_teo_ss = mpc(TCLab_Teo_ss, Ts, P, M, Weights, [MV1 MV2]);  
  
%% Cria MPC com modelo experimental  
  
MPC_tclab_exp_tf = mpc(TCLab_Exp_tf, Ts, P, M, Weights, [MV1 MV2]);  
MPC_tclab_exp_ss = mpc(TCLab_Exp_ss, Ts, P, M, Weights, [MV1 MV2]);  
MPC_tclab_exp_oe = mpc(TCLab_Exp_oe, Ts, P, M, Weights, [MV1 MV2]);  
MPC_tclab_exp_bj = mpc(TCLab_Exp_bj, Ts, P, M, Weights, [MV1 MV2]);  
MPC_tclab_exp_arx = mpc(TCLab_Exp_arx, Ts, P, M, Weights, [MV1 MV2]);  
MPC_tclab_exp_armax = mpc(TCLab_Exp_armax, Ts, P, M, Weights, [MV1 MV2]);
```

Fonte: Autor

A aplicação de cada um dos controladores **MPC** criados, bem como suas comparações com o controlador **PID** e discussões a respeito de sua sintonia e utilização, encontram-se na [parte IV](#) a seguir.

Parte IV

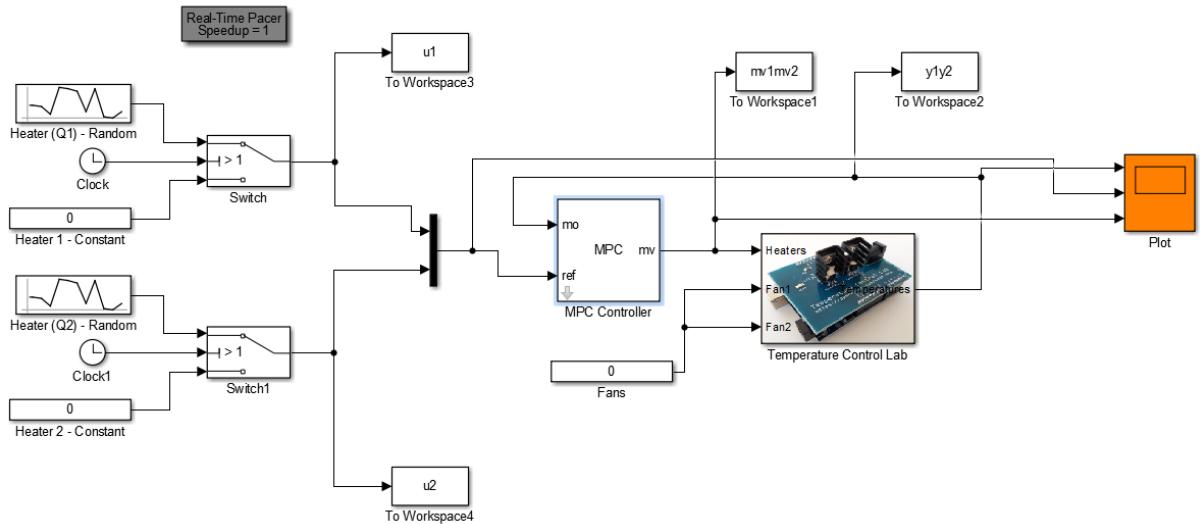
Resultados e discussões

7 Resultados

Para a aplicação dos controladores MPC criados nos capítulos anteriores, foi desenvolvido um ambiente *Simulink* (fig. 29) onde cada um dos diferentes controladores obtidos foram conectados ao **TCLabSP** para realizar seu controle.

Todos¹ os diferentes controladores foram submetidos aos mesmos sinais de referência (*set-point*) e ao mesmo tempo de experimento. Os sinais de referência utilizados foram sinais de amplitude randômica e com duração de duas vezes o tempo de acomodação da planta em malha aberta, ou seja, $2 \times T_s = 2 \times 383s = 766s$. A duração do experimento foi determinada de forma que cinco sinais de entrada distintos pudessem ser aplicados ao sistema, sendo assim o tempo dos experimentos foi de $5 \times 766s = 3830s$. Cada um dos experimentos foi repetido ao menos 3 vezes e ao final foi calculada uma média de todos os sinais coletados para atenuar possíveis erros pontuais de cada experimento.

Figura 29 – Ambiente Simulink para implementação dos controladores MPC

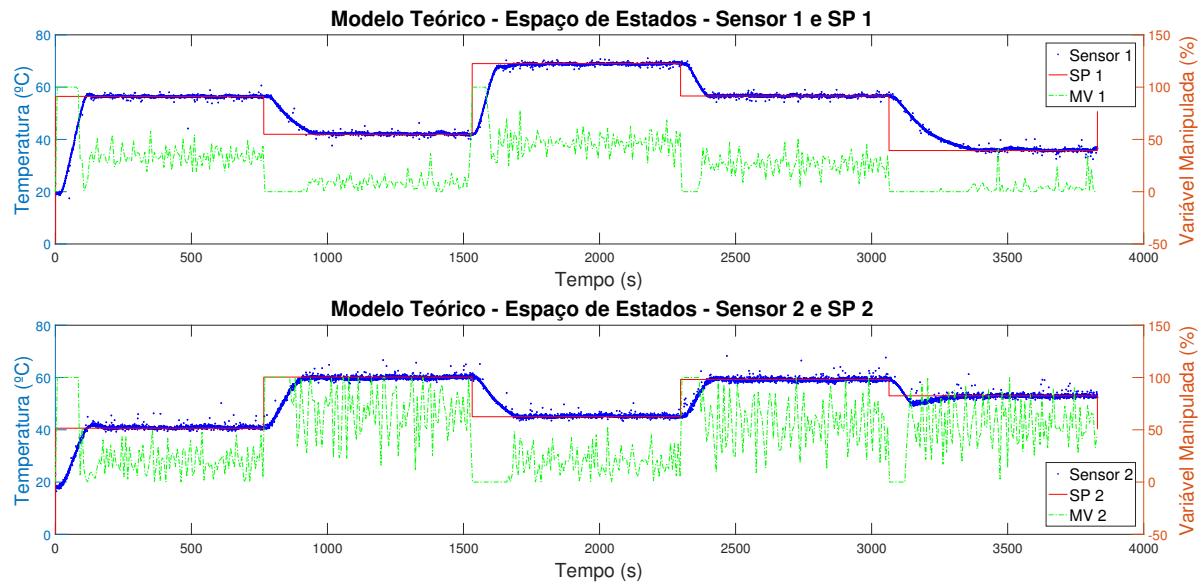


Fonte: Autor

As figs. 30 a 35 apresentam os gráficos de resposta em malha fechada para cada um dos experimentos aplicando os modelos obtidos. Para fins comparativos, a fig. 36 apresenta o gráfico de resposta do controlador PID apresentado na seção 6.1.

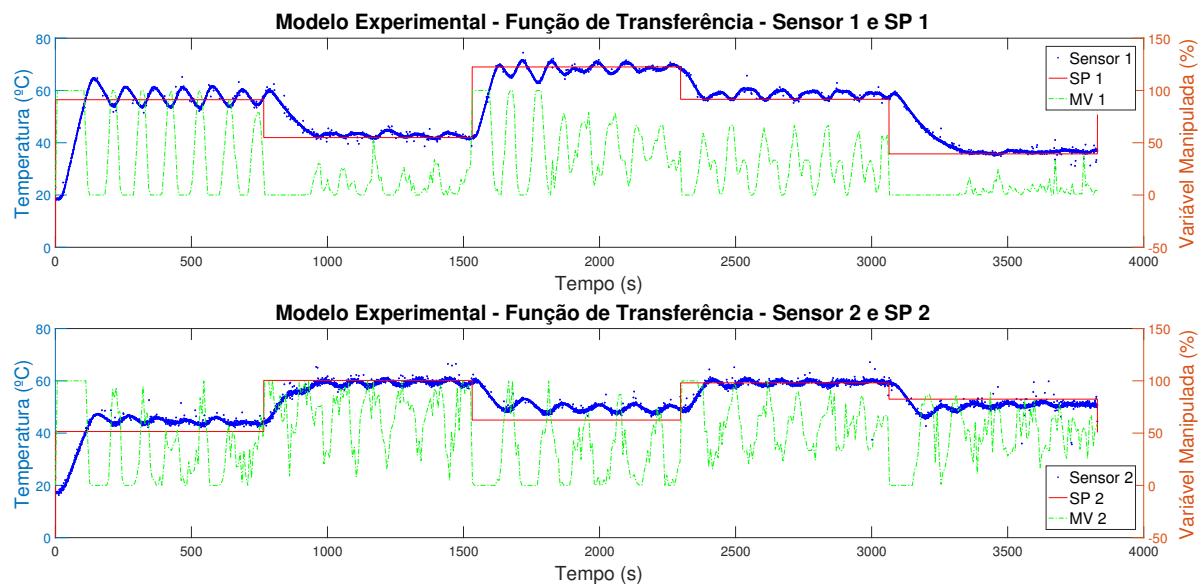
¹ O MATLAB® não conseguiu criar um controlador baseado no modelo experimental Box-Jenkins pois este modelo possui pólos discretizados próximos de $z = 0$.

Figura 30 – Resposta do controlador MPC criado a partir do modelo teórico



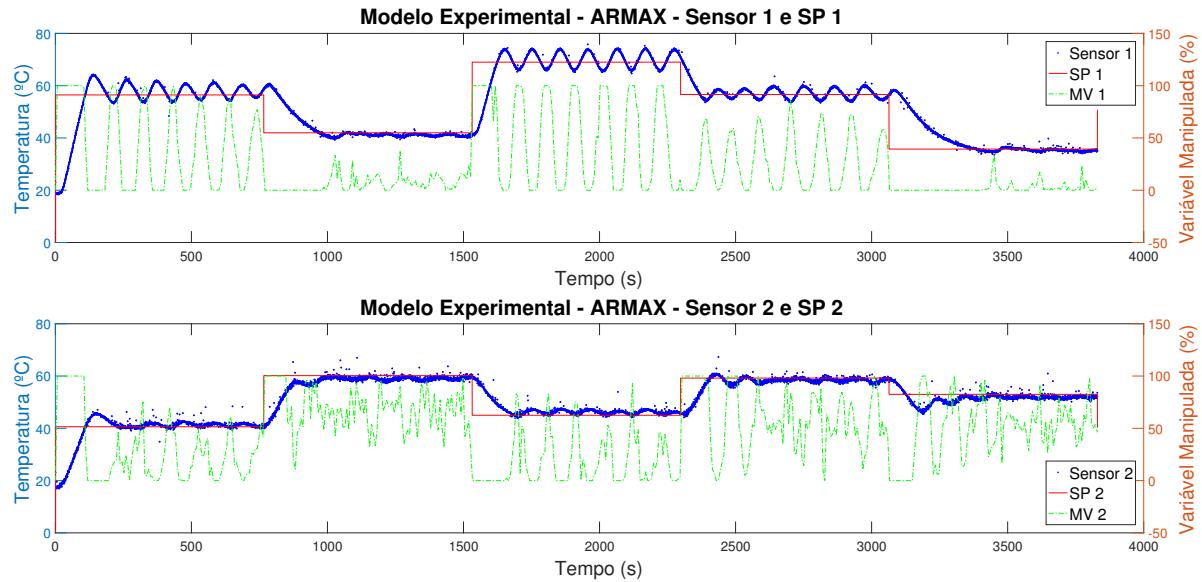
Fonte: Autor

Figura 31 – Resposta do controlador MPC criado a partir do modelo experimental (função de transferência)



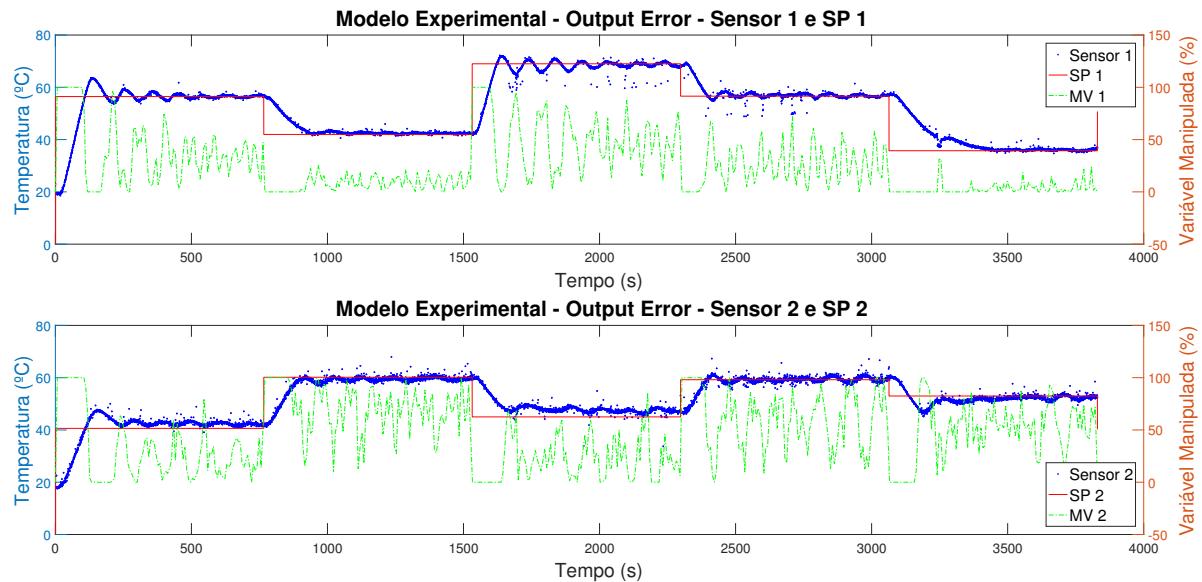
Fonte: Autor

Figura 32 – Resposta do controlador MPC criado a partir do modelo experimental (ARMAX)



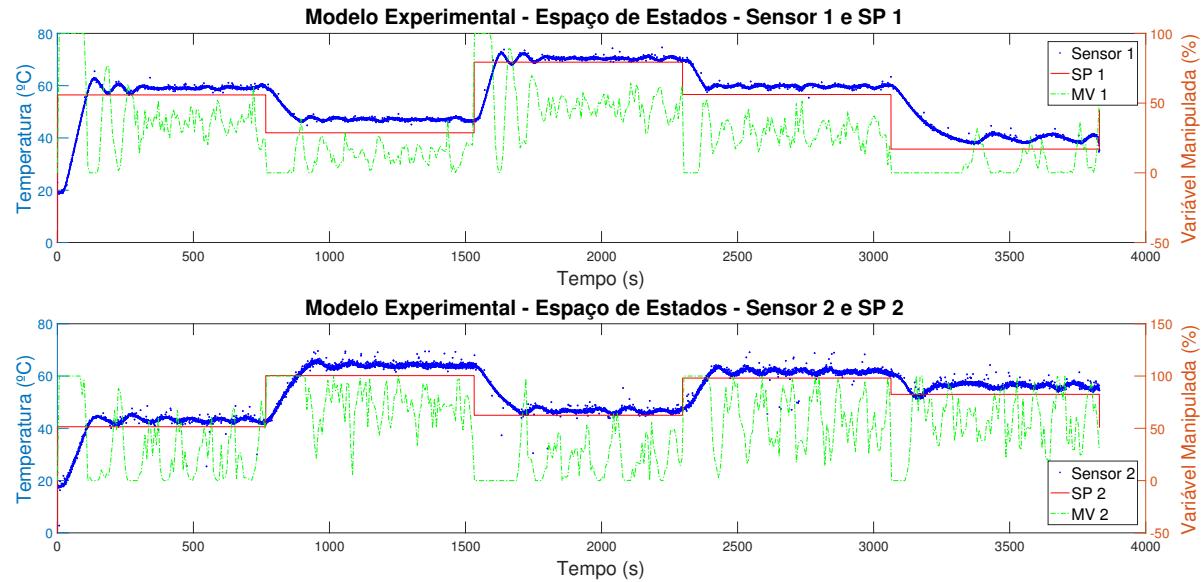
Fonte: Autor

Figura 33 – Resposta do controlador MPC criado a partir do modelo experimental (Output-Error)



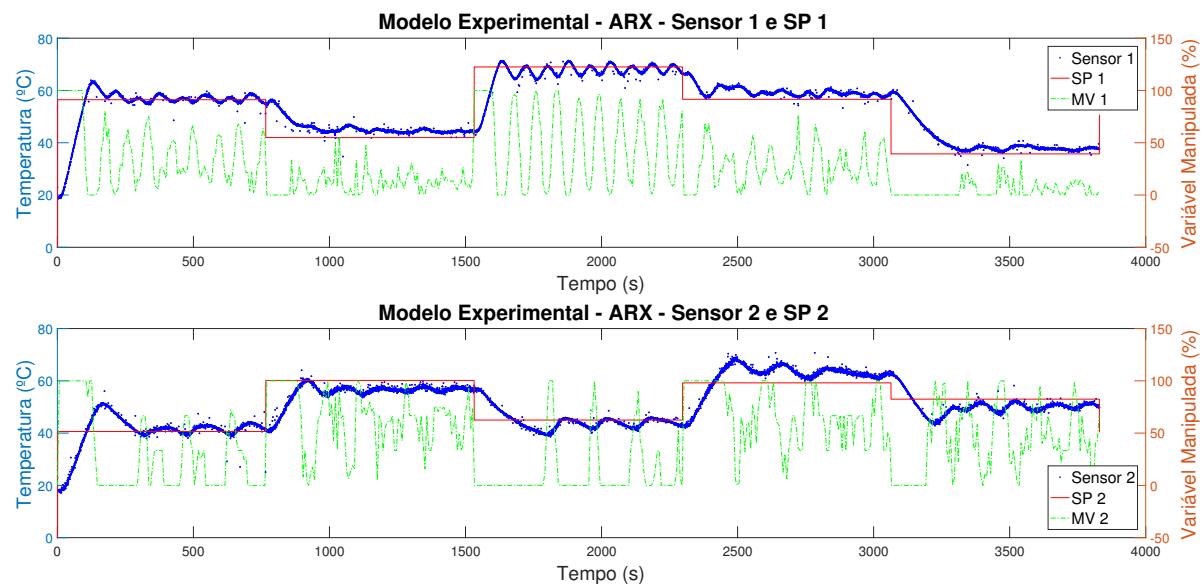
Fonte: Autor

Figura 34 – Resposta do controlador MPC criado a partir do modelo experimental (espeço de estados)



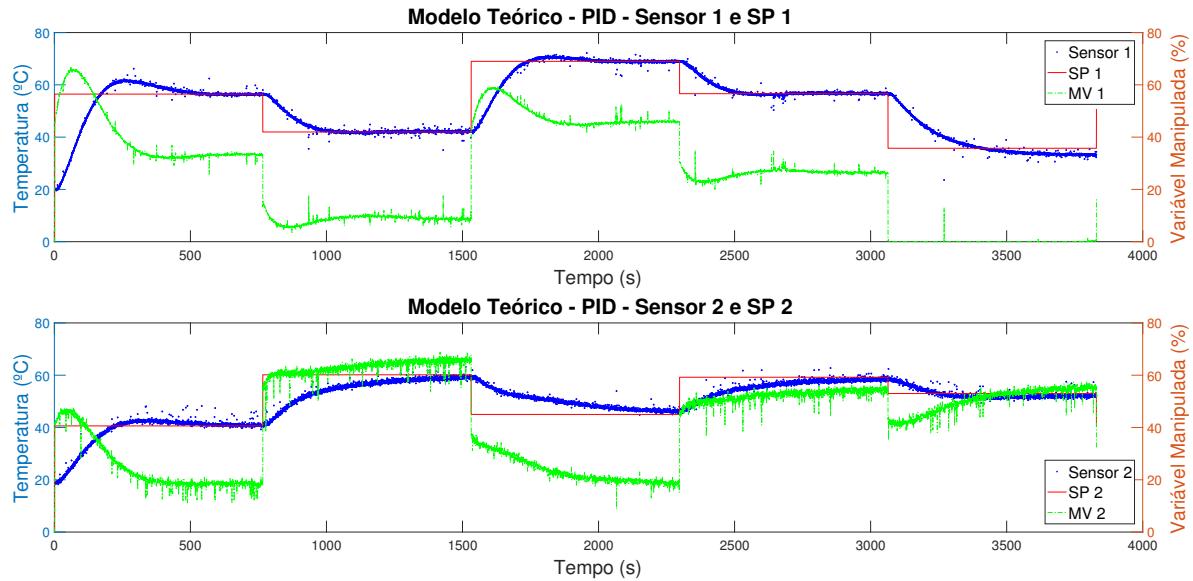
Fonte: Autor

Figura 35 – Resposta do controlador MPC criado a partir do modelo experimental (ARX)



Fonte: Autor

Figura 36 – Resposta do controlador PID auto-sintonizado



Fonte: Autor

Nas [tabelas 19 a 22](#) é possível observar diferentes indicadores de performance para cada um dos modelos testados.

Os indicadores utilizados são:

- **IAE:** Integral absoluta do erro (do inglês *Integral Absolute Error*)

$$IAE = \int_0^{\infty} |e| dt \quad (7.1)$$

- **ISE:** Integral quadrada do erro (do inglês *Integral Square Error*)

$$ISE = \int_0^{\infty} e^2 dt \quad (7.2)$$

- **ITAE:** Integral absoluta do erro no tempo (do inglês *Integral Time Absolute Error*)

$$ITAE = \int_0^{\infty} t |e| dt \quad (7.3)$$

- **ITSE:** Integral quadrada do erro no tempo (do inglês *Integral Time Square Error*)

$$ITSE = \int_0^{\infty} t \cdot e^2 dt \quad (7.4)$$

A [fig. 37](#) apresenta um resumo dos indicadores de performance mostrados nas [tabelas 19 a 22](#).

Tabela 19 – Performance dos controladores MPC e PID - IAE

Modelo utilizado no controlador	Sensor 1 (x10 ³)	Sensor 2 (x10 ³)	Média (x10 ³)
MPC Teórico SS (eq. (4.3))	38.07	27.94	33.00
MPC Experimental OE (tabela 14)	45.53	42.16	43.85
MPC Experimental ARMAX (tabela 12)	58.16	37.27	47.72
MPC Experimental TF (tabela 8)	56.90	55.08	55.99
PID (tabela 18)	53.90	59.10	56.50
MPC Experimental ARX (tabela 10)	59.13	63.35	61.24
MPC Experimental SS (eq. (5.5))	75.99	58.12	67.06

Fonte: Autor

Tabela 20 – Performance dos controladores MPC e PID - ISE

Modelo utilizado no controlador	Sensor 1 (x10 ⁴)	Sensor 2 (x10 ⁴)	Média (x10 ⁴)
MPC Teórico SS (eq. (4.3))	62.95	29.03	45.99
MPC Experimental OE (tabela 14)	66.46	32.78	49.62
MPC Experimental ARMAX (tabela 12)	77.46	30.68	54.07
MPC Experimental TF (tabela 8)	76.61	39.75	58.18
MPC Experimental ARX (tabela 10)	67.98	49.01	58.50
MPC Experimental SS (eq. (5.5))	81.04	40.95	61.00
PID (tabela 18)	76.83	50.06	63.45

Fonte: Autor

Tabela 21 – Performance dos controladores MPC e PID - ITAE

Modelo utilizado no controlador	Sensor 1 (x10 ⁶)	Sensor 2 (x10 ⁶)	Média (x10 ⁶)
MPC Teórico SS (eq. (4.3))	60.99	38.11	49.55
MPC Experimental OE (tabela 14)	73.78	61.69	67.74
MPC Experimental ARMAX (tabela 12)	91.79	57.31	74.55
PID (tabela 18)	84.97	88.90	86.94
MPC Experimental TF (tabela 8)	90.39	84.58	87.49
MPC Experimental ARX (tabela 10)	105.79	113.15	109.47
MPC Experimental SS (eq. (5.5))	136.28	97.80	117.04

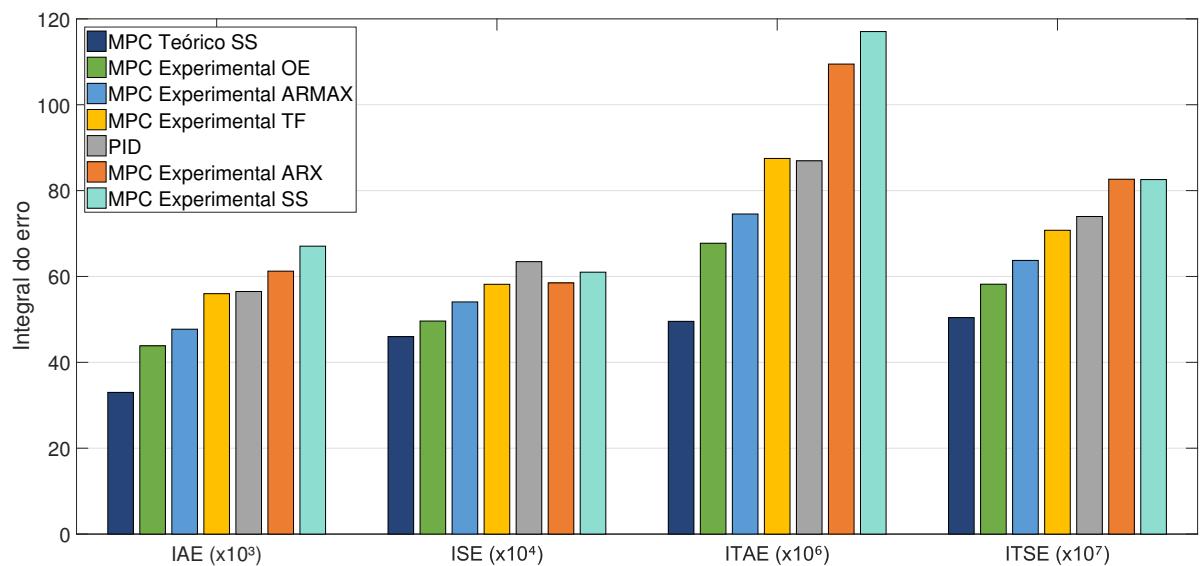
Fonte: Autor

Tabela 22 – Performance dos controladores MPC e PID - ITSE

Modelo utilizado no controlador	Sensor 1 (x10 ⁷)	Sensor 2 (x10 ⁷)	Média (x10 ⁷)
MPC Teórico SS (eq. (4.3))	74.01	26.80	50.41
MPC Experimental OE (tabela 14)	83.09	33.31	58.20
MPC Experimental ARMAX (tabela 12)	96.70	30.77	63.74
MPC Experimental TF (tabela 8)	96.51	45.02	70.76
PID (tabela 18)	92.25	55.69	73.97
MPC Experimental SS (eq. (5.5))	114.29	50.85	82.57
MPC Experimental ARX (tabela 10)	96.37	68.92	82.65

Fonte: Autor

Figura 37 – Performance dos controladores MPC e PID agrupados por índice



Fonte: Autor

A partir da [fig. 37](#) é possível observar que dentre todos os controladores analisados, o controlador **MPC** criado a partir do modelo em espaço de estados e obtido por abordagem teórica foi o aquele que mostrou melhor controle da planta estudada (menor erro relativo). Mesmo quando comparado com o controlador **PID** de referência, o controle **MPC** em questão mostrou uma maior velocidade para a estabilização dos seus múltiplos sinais de saída. Contudo, vale ressaltar que este controlador **PID** foi apenas sintonizado utilizando a função de *auto-tunning* disponível no **MATLAB®**. Um controlador **PID** melhor parametrizado poderia ter uma performance muito mais próxima das performances obtidas nos controladores **MPC**, pois segundo [Sha'aban, Lennox e Laurí \(2013\)](#) e [Taysom, Sorensen e Hedengren \(2017\)](#) a diferença na performance destes dois controles é muito pequena quando a planta possui pouco ou nenhum atraso (como a planta utilizada neste estudo), porém quando os atrasos na planta aumentam, a resposta do controlador **PID** se degrada rapidamente.

Ainda comparando os controladores **MPC** e **PID**, além das performances de ambos poderem ser diferenciadas devido ao atraso do sistema, [Taysom, Sorensen e Hedengren \(2017\)](#) também apontam em seu estudo que os controladores **MPC** geralmente apresentam uma melhor resposta para sistemas **MIMO**, além de serem mais efetivos em sistemas onde os distúrbios podem ser modelados, uma vez que o **MPC** consegue utilizar o modelo do processo e o modelo do distúrbio conjuntamente para um controle mais assertivo. Em contrapartida os controladores **PID** são relativamente mais fáceis de sintonizar e muito mais fáceis de se implementar, além de possuírem resposta similar ao **MPC** em uma grande variedade de aplicações ([TAYSOM; SORENSEN; HEDENGREN, 2017](#)).

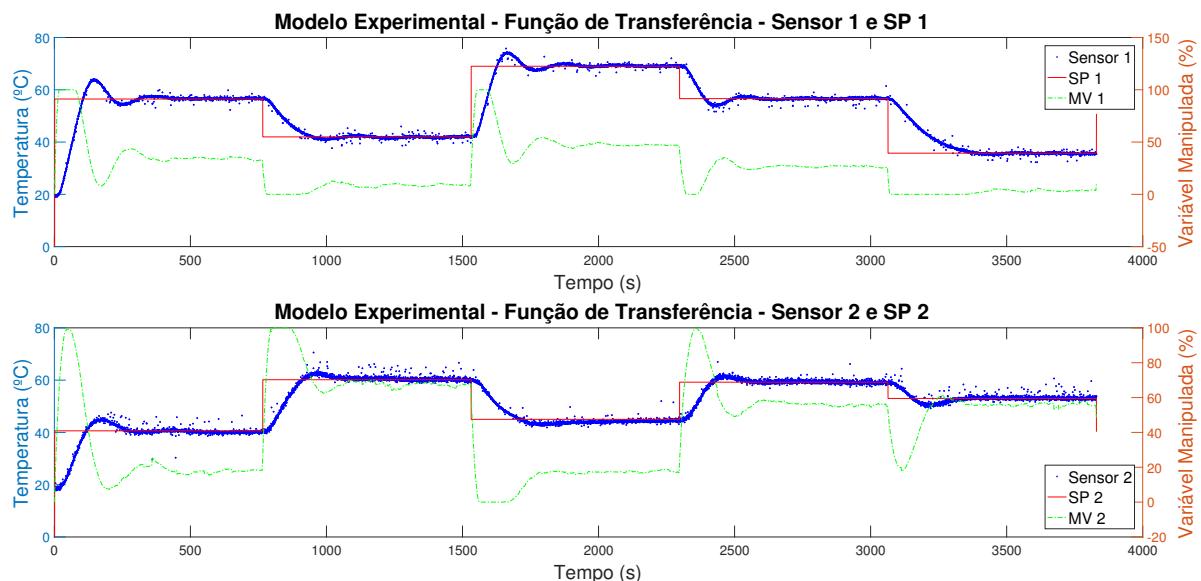
Outro ponto de destaque para os controladores **MPC** é a possibilidade da configuração de restrições de atuação, algo que é impossível de se fazer utilizando um controle **PID**. No caso do uso de restrições, o controlador **MPC** impede que a(s) saída(s) da planta controlada ultrapassem limites indesejados, fazendo com que o **MPC** seja uma opção de controle muito interessante quando se deseja maximizar ou minimizar valores de produção sem que algumas variáveis do processo atinjam ou ultrapassem limites indesejados.

Observa-se através das [figs. 31 a 35](#) que mesmo utilizando uma metodologia consolidada academicamente para a criação do controle **MPC** através de modelos experimentais, a resposta em malha fechada pode não ser satisfatória. Nestas figuras é possível observar uma oscilação indesejada nas saídas da planta, além de erro estacionário em alguns casos. Todavia, ferramentas como o *MPC design* do **MATLAB®** podem auxiliar no desenvolvimento de um controle **MPC** através de um modelo experimental que ao ser empiricamente sintonizado pode apresentar uma resposta satisfatória em malha fechada. A [fig. 38](#) apresenta o gráfico em malha fechada de um controlador **MPC** utilizando o mesmo modelo experimental em função de transferência apresentado no capítulo anterior, porém desta vez sintonizado empiricamente utilizando a ferramenta *MPC Design*. A [fig. 39](#) traz novamente

a comparação de performance dos controladores porém desta vez incluindo este novo modelo e revela que, apesar de uma sintonia empírica, este controlador apresenta uma aproximação do *set-point* bem superior em comparação aos controladores baseados em modelos experimentais que já haviam sido apresentados neste trabalho.

Algumas imagens do *MPC Design*, as tabelas de performance de cada um dos indicadores incluindo este modelo empírico e o método de sintonia podem ser encontrados no [apêndice D](#).

Figura 38 – Resposta do controlador MPC criado empiricamente a partir do modelo experimental (função de transferência)



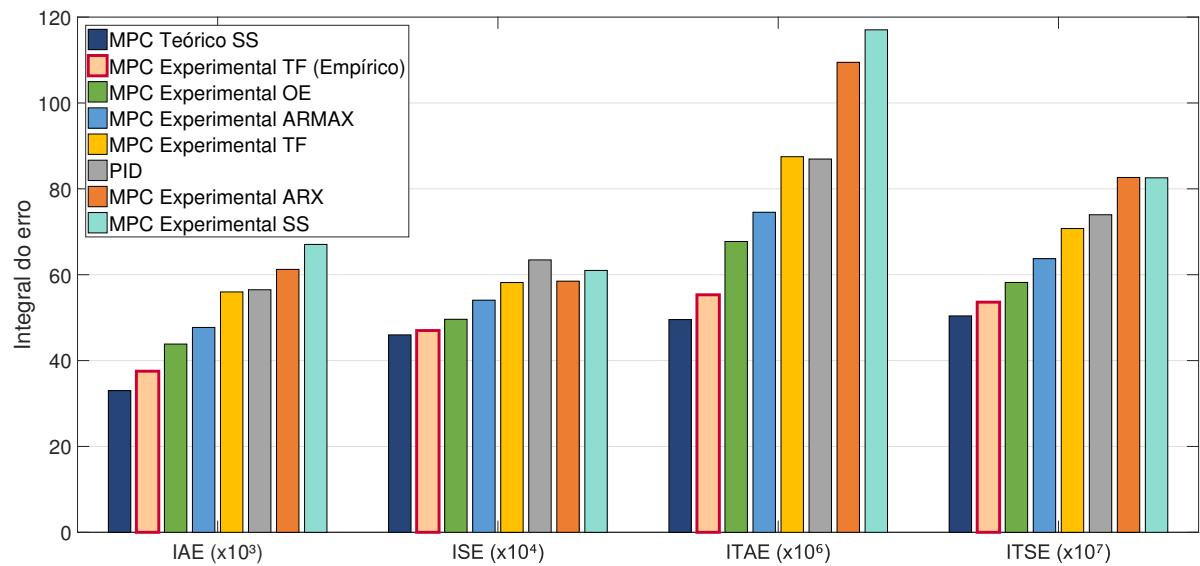
Fonte: Autor

Nota-se também neste controlador² apresentado na fig. 38 uma maior estabilidade das variáveis manipuladas, apesar de um *overshoot* maior que os demais controles.

De forma geral é possível afirmar que os objetivos deste trabalho foram alcançados com êxito, uma vez que foi possível desenvolver e implementar um controle **MPC** na planta desejada, comparar este controle com um controle **PID** convencional e ainda documentar todas as etapas de forma que essas possam ser reproduzidas por outros estudantes que estejam pesquisando sobre o [Controle Preditivo Baseado em Modelo](#).

² Parâmetros encontrados empiricamente e ajustados para este controlador: Tempo de amostragem (t_s) = 2s; Horizonte de predição = 120; Horizonte de Controle = 16; Δ = 0; R = 0,4393; Q = 0,2276.

Figura 39 – Performance dos controladores MPC e PID agrupados por índice - Incluindo modelo empírico



Fonte: Autor

8 Discussões

Como recomendação para trabalhos futuros, deixo as seguintes sugestões abaixo:

- **Realizar a sintonia fina do PID.** O controlador **PID** apresentou um desempenho bem inferior a maioria dos controladores **MPC** desenvolvidos. Essa diferença pode diminuir significativamente caso o controlador **PID** esteja bem sintonizado, tornando assim a comparação entre ambos os métodos mais próxima.
- **Modelar distúrbios.** O **TCLabSP** está equipado com ventiladores que podem atuar como distúrbios observáveis na planta. Modelar estes distúrbios e fazer com que o **MPC** atue sobre eles pode aumentar bastante a robustez do projeto, além agregar muito conhecimento de situações industriais reais para quem o desenvolver.
- **Modificar a planta para aumentar o atraso.** Como visto no capítulo de resultados, o desempenho do controle **MPC** pode ser mais expressivo, em comparação com o **PID**, principalmente em plantas com atraso. Sendo assim, seria possível fazer algumas modificações no **TCLab** para que os sensores captem a mudança de temperatura dos aquecedores mais tarde. Uma das formas de se fazer isso seria inserindo entre os aquecedores e sensores algum material de condução térmica reduzida.
- **Testar diferentes sintonias para os controladores MPC.** Ao concluir este trabalho observou-se que os modelos **MPC** experimentais sintonizados com base na bibliografia tiveram uma performance inferior ao modelo sintonizado empiricamente. Atualmente existem diversos estudos que apresentam diferentes técnicas e metodologias para a sintonia de um controlador **MPC**. Aplicar técnicas diferentes das usadas neste trabalho podem resultar em uma melhora na performance do controlador para os modelos experimentais.

Referências

- AGUIRRE, L. A. **Introdução à Identificação de Sistemas - Técnicas Lineares e Não Lineares: Teoria e Aplicação**. 4. ed. Belo Horizonte: Editora UFMG, 2015. 776 p. ISBN 978-85-423-0079-6.
- AKAIKE, H. Stochastic theory of minimal realization. **IEEE Transactions on Automatic Control**, v. 19, n. 6, p. 667–674, dec 1974. ISSN 0018-9286.
- ALHAJERI, M.; SOROUSH, M. Tuning Guidelines for Model-Predictive Control. **Industrial and Engineering Chemistry Research**, v. 59, n. 10, p. 4177–4191, 2020. ISSN 15205045.
- ÅSTRÖM, K. J.; TORSTEN, B. Numerical Identification of Linear Dynamic Systems from Normal Operating Records. **IFAC Proceedings Volumes**, v. 2, n. 2, p. 96–111, sep 1965. ISSN 14746670.
- ÅSTRÖM, K. J.; WITTENMARK, B. The Self-Tuning Regulators Revisited. **IFAC Proceedings Volumes**, v. 18, n. 5, p. xxix–xxxvii, jul 1985. ISSN 14746670.
- BOLOGNANI, S. et al. Design and Implementation of Model Predictive Control for Electrical Motor Drives. **IEEE Transactions on Industrial Electronics**, v. 56, n. 6, p. 1925–1936, 2009. ISSN 0278-0046.
- BORRELLI, F.; BEMPORAD, A.; MORARI, M. **Predictive control for linear and hybrid systems**. New York: Cambridge University Press, 2017. 440 p. ISBN 978-1107652873.
- CAMACHO, E. F.; ALBA, C. B.; BORDONS, C. **Model Predictive control**. [S.l.: s.n.], 2007. 405 p. ISSN 1098-6596. ISBN 9781852336943.
- DUNIA, R.; EDGAR, T. F.; HAUGEN, F. A. A complete programming framework for process control education. In: **2008 IEEE International Conference on Control Applications**. [S.l.]: IEEE, 2008. p. 516–521. ISBN 978-1-4244-2222-7.
- EYKHOFF, P. **System Identification: Parameter and State Estimation**. London: Wiley, 1974. ISBN 9780471249801.
- FAVARO, J. **Controle preditivo aplicado à planta piloto de neutralização de pH**. Tese (Mestrado) — Universidade de São Paulo, São Paulo, sep 2012.
- GARCIA, C. **Modelagem e Simulação de Processos Industriais e de Sistemas Eletromecânicos**. 1. ed. São Paulo: Edusp, 2005. 688 p. ISBN 978-85-314-0904-2.
- GARCIA, C. **Modelagem e Simulação de Processos Industriais e de Sistemas Eletromecânicos**. 2. ed. rev. ed. São Paulo: Edusp, 2013. 678 p. ISBN 978-85-314-0904-2.
- GEVERS, M. A Personal view of the development of system identification: A 30-year journey through an exciting field. **IEEE Control Systems**, IEEE, v. 26, n. 6, p. 93–105, 2006. ISSN 1066033X.

- GONÇALVES, D. **Implementação Prática de um Controlador Preditivo a um Processo Não-Linear.** 98 p. Tese (Mestrado) — Universidade Federal de Uberlândia, 2012.
- HAUGEN, F. A. **A brief introduction to optimization methods Contents.** 2018. Disponível em: <http://techteach.no/fag/process_control_nmbu_2018/optim/optimization_finn_haugen_2018.pdf>.
- HEDENGREN, J. D. **Temperature Control Lab B: MIMO (Multiple Input, Multiple Output) Model.** 2018. Disponível em: <https://apmonitor.com/do/uploads/Main/Lab_B_MIMO_Model.pdf>.
- HO, B. L.; KALMAN, R. E. Effective construction of linear state-variable models from input/output functions. **At-Automatisierungstechnik**, v. 14, n. 1-12, p. 545–548, jan 1966. ISSN 2196677X.
- 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, jun 2011. ISSN 1598-6446.
- OGATA, K. **Engenharia de Controle Moderno.** 5. ed. São Paulo: Pearson Prentice Hall, 2010. 809 p. ISBN 978-85-7605-810-6.
- PARKINSON, A.; BALLING, R.; HEDENGREN, J. D. **Optimization Methods for Engineering Design.** Brigham Young University, 2018. v. 2. ISBN 0521665698. Disponível em: <<http://apmonitor.com/me575/index.php/Main/BookChapters>>.
- PRACEK, B. et al. Aplicação de controle feedforward para regulação da temperatura num protótipo de túnel de vento. In: . [S.l.: s.n.], 2011. p. 532–535.
- PRATA, T. C. **TCLabSP - Temperature Control Lab Sao Paulo (Version 1.0).** Zenodo, 2019. Disponível em: <<http://doi.org/10.5281/zenodo.3534862>>.
- ROSSITER, J. A. **Model- Based Predictive Control - A practical approach.** [S.l.: s.n.], 2003. 337 p. ISBN 0849312914.
- RUGGIERO, M. A. G.; LOPES, V. L. d. R. **Cálculo Numérico: aspectos teóricos e computacionais.** 2. ed. São Paulo: Pearson Makron Books, 2000. 406 p. ISBN 978-85-346-0204-4.
- SEBORG, D. E.; EDGAR, T. F.; MELLICHAMP, D. A. Model Predictive Control. In: **Process Dynamics and Control.** [S.l.: s.n.], 2011. p. 414–438. ISBN 0471000779.
- SHA'ABAN, Y. A.; LENNOX, B.; LAURÍ, D. PID versus MPC performance for SISO dead-time dominant processes. **IFAC Proceedings Volumes (IFAC-PapersOnline)**, v. 46, n. 32 PART 1, p. 241–246, 2013. ISSN 14746670.
- STADLER, K. S.; POLAND, J.; GALLESTEY, E. Model predictive control of a rotary cement kiln. **Control Engineering Practice**, Elsevier, v. 19, n. 1, p. 1–9, 2011. ISSN 09670661.
- TAYSOM, B. S.; SORENSEN, C. D.; HEDENGREN, J. D. A comparison of model predictive control and PID temperature control in friction stir welding. **Journal of Manufacturing Processes**, The Society of Manufacturing Engineers, v. 29, p. 232–241, 2017. ISSN 15266125.

VUKOV, M. **Embedded Model Predictive Control and Moving Horizon Estimation for Mechatronics Applications**. Tese (Doutorado) — KU Leuven, 2015. Disponível em: <<https://lirias.kuleuven.be/handle/123456789/487664>https://lirias.kuleuven.be/bitstream/123456789/487664/1/thesis_final_print.pdf>.

WANG, L. **Model Predictive Control System Design and Implementation Using MATLAB®**. London: Springer London, 2009. 403 p. (Advances in Industrial Control, 9781848823303). ISSN 1430-9491. ISBN 978-1-84882-330-3.

YAKUB, F.; MORI, Y. Model predictive control for car vehicle dynamics system - Comparative study. In: **2013 IEEE Third International Conference on Information Science and Technology (ICIST)**. Yangzhou: IEEE, 2013. p. 150–155. ISBN 978-1-4673-2764-0.

Apêndices

APÊNDICE A – Códigos-fonte

A.1 Método de pesquisa em grade

Código-fonte A.1 – Pesquisa em grade com número escalar

```

# Linguagem:           Python
# Autor:               Tiago Correa Prata
# Disponivel em:
#   <https://
#     //github.com/TiagoPrata/MasterThesis/blob/master/4_codes/grid_search_scalar.py>

import numpy as np
import matplotlib.pyplot as plt

def f_obj(x):
    return x**4.0 * 0.00232 - x**3.0 * 0.111 + x**2.0 * 1.80 - x * 11.6
    + 34.4

x_min = 2
x_max = 22
N_x = 100
x_array = np.linspace(x_min, x_max, N_x)
f_array = np.dot(x_array, 0)

f_min = float('inf')
x_opt = float('-inf')

# Laco varrendo todos os valores de x
for i in range(len(x_array)):
    x = x_array[i]

    f = f_obj(x)

    # Armazena os valores caso seja a melhor solucao
    if f <= f_min:
        f_min = f
        x_opt = x

    f_array[i] = f
    x_array[i] = x

# Apresenta valores calculados
print(f_min)

```

```

print(x_opt)

# Configurações de gráfico
plt.plot(x_array, f_array, 'b.')
plt.plot(x_opt, f_min, 'ro')
plt.xlabel('$x$')
plt.ylabel('$f(x)$')
# Plota gráfico (Exibido na fig. 1)
plt.show()

```

Fonte: Autor, adaptado de Haugen (2018)

A.2 Método de pesquisa em grade com duas variáveis

Código-fonte A.2 – Pesquisa em grade com duas variáveis

```

import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

x1_min = 0
x1_max = 2
N_x1 = 100
x1_array = np.linspace(x1_min, x1_max, N_x1)

x2_min = 1
x2_max = 3
N_x2 = 100
x2_array = np.linspace(x2_min, x2_max, N_x2)

f_min = float('inf')
x1_opt = float('-inf')
x2_opt = float('-inf')

f_matrix = np.zeros([len(x1_array), len(x2_array)])

def fun_obj(x1, x2):
    return (x1-1)**2 + (x2-2)**2 + 0.5

for k_x1 in range(len(x1_array)):
    x1 = x1_array[k_x1]
    for k_x2 in range(len(x2_array)):
        x2 = x2_array[k_x2]

        f = fun_obj(x1, x2)

```

```

        if not (x1-x2+1.5 <= 0):
            f = float('inf')

        if f <= f_min:
            f_min = f
            x1_opt = x1
            x2_opt = x2

    f_matrix[k_x1, k_x2] = f

print(f_min)
print(x1_opt)
print(x2_opt)

# fig = plt.figure()
# ax = Axes3D(fig)
# x, y = np.meshgrid(x1_array, x2_array)
# ax.plot_surface(x1_array, x2_array, f_matrix, rstride=1, cstride=1)
# plt.show()

```

Fonte: Autor, adaptado de Haugen (2018)

A.3 Exemplo do método Estimador de Horizonte Móvel

Código-fonte A.3 – Exemplo do método Estimador de Horizonte Móvel

```

# Linguagem:           Python
# Autor:              Tiago Correa Prata
# Disponível em:
# <https://github.com/TiagoPrata/MasterThesis/blob/master/4\_codes/mhe\_example.py>

import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import minimize

# model param
K = 1      # Gain
T = 2      # Time constant
n = 3      # number of state variables
model_params = {'K': K, 'T': T}
cov_process_disturb_w1 = 0.001
cov_process_disturb_w2 = 0.001
cov_process_disturb_w3 = 0.001

```

```
cov_process_disturb_w = np.diag([cov_process_disturb_w1 ,  
    cov_process_disturb_w2 , cov_process_disturb_w3])  
cov_meas_noise_v1 = 0.01  
cov_meas_noise_v = np.diag([cov_meas_noise_v1])  
  
# Time settings  
Ts = 0.5          # s  
t_start = 0       # s  
t_stop = 20        # s  
t_array = np.linspace(t_start , t_stop-Ts , t_stop / Ts)  
N = len(t_array)  
t_mhe = 5  
N_mhe = int(np.floor(t_mhe/Ts))  
number_optim_vars = n*N_mhe  
  
# Preallocation of arrays for storage  
u_sim_array = t_array*0  
x1_sim_array = t_array*0  
x2_sim_array = t_array*0  
x3_sim_array = t_array*0  
y1_sim_array = t_array*0  
x1_est_optim_plot_array = t_array*0  
x2_est_optim_plot_array = t_array*0  
x3_est_optim_plot_array = t_array*0  
  
# Sim initialization  
x1_sim_init = 2  
x2_sim_init = 3  
x1_sim_k = x1_sim_init  
x2_sim_k = x2_sim_init  
  
# MHE initialization  
mhe_array = np.zeros(N_mhe)  
  
x1_est_init_guess = 0  
x2_est_init_guess = 0  
x3_est_init_guess = 2  
x1_est_optim_array = np.zeros(N_mhe) + x1_est_init_guess  
x2_est_optim_array = np.zeros(N_mhe) + x2_est_init_guess  
x3_est_optim_array = np.zeros(N_mhe) + x3_est_init_guess  
  
x_est_guess_matrix = np.concatenate(([x1_est_optim_array] , [  
    x2_est_optim_array] , [x3_est_optim_array]))  
  
u_mhe_array = mhe_array * 0  
y1_meas_mhe_array = mhe_array * 0
```

```

# Configuring continuous plotting
plt.ion()

def objective(x_est_matrix):
    # Reshaping matrix because it is flattened after minimine call
    x_est_matrix = x_est_matrix.reshape(3, N_mhe)

    K = model_params['K']
    T = model_params['T']
    Q = covars['Q']
    R = covars['R']

    J_km1 = 0

    for k in range(N_mhe-1):
        u_k = u_mhe_array[k]
        x_k = x_est_matrix[:, k]
        x1_k = x_k[0]
        x2_k = x_k[1]
        x3_k = x_k[2]
        h1_k = x1_k
        y1_meas_k = y1_meas_mhe_array[k]
        v1_k = y1_meas_k - h1_k
        v_k = v1_k

        if k <= N_mhe-1:
            x_kp1 = x_est_matrix[:, k+1]
            x1_kp1 = x_kp1[0]
            x2_kp1 = x_kp1[1]
            x3_kp1 = x_kp1[2]

            dx1_dt_k = x2_k
            dx2_dt_k = (-x2_k + K*u_k + x3_k)/T
            dx3_dt_k = 0
            f1_k = x1_k + Ts*dx1_dt_k
            f2_k = x2_k + Ts*dx2_dt_k
            f3_k = x3_k + Ts*dx3_dt_k
            w1_k = [x1_kp1 - f1_k]
            w2_k = [x2_kp1 - f2_k]
            w3_k = [x3_kp1 - f3_k]
            w_k = np.concatenate([w1_k, w2_k, w3_k]).T

            dJ_K = np.dot(np.dot(w_k.T, np.linalg.inv(Q)), w_k) + np.dot(np.
                dot(v_k.T, np.linalg.inv(R)), v_k)
            J_k = J_km1 + dJ_K

            # Time shift

```

```

J_km1 = J_k

return J_k

# Sim loop
for k in range(N):
    t_k = k*Ts

    # Process simulator
    if t_k < 2:
        u_k = 2
        d_k = 1

    # Change of u
    if t_k >=2:
        u_k = 2

    # Change of d
    if t_k >= 8:
        d_k = 1

    # Change of u
    if t_k >= 10:
        u_k = 6

    # Change of d
    if t_k >= 15:
        d_k = 2

    # derivatives
    dx1_sim_dt_k = x2_sim_k
    dx2_sim_dt_k = (-x2_sim_k + K*u_k + d_k)/T
    f1_sim_k = x1_sim_k + Ts*dx1_sim_dt_k
    f2_sim_k = x2_sim_k + Ts*dx2_sim_dt_k

    # Integration and adding disturbance
    w1_sim_k = np.sqrt(cov_process_disturb_w1) * np.random.rand()
    w2_sim_k = np.sqrt(cov_process_disturb_w2) * np.random.rand()
    x1_sim_kp1 = f1_sim_k + w1_sim_k
    x2_sim_kp1 = f2_sim_k + w2_sim_k

    # Calculating output and adding meas noise
    v1_sim_k = np.sqrt(cov_meas_noise_v1) * np.random.rand()
    y1_sim_k = x1_sim_k + v1_sim_k

    # Storage
    t_array[k] = t_k

```

```

    u_sim_array[k] = u_k
    x1_sim_array[k] = x1_sim_k
    x2_sim_array[k] = x2_sim_k
    x3_sim_array[k] = d_k
    y1_sim_array[k] = y1_sim_k

    # Preparing for time shift:
    x1_sim_k = x1_sim_kp1
    x2_sim_k = x2_sim_kp1

    # Updating u and y for use in MHE
    u_mhe_array = np.append(u_mhe_array[1:N_mhe], u_k)
    y1_meas_mhe_array = np.append(y1_meas_mhe_array[1:N_mhe], y1_sim_k)
    y_meas_mhe_array = [y1_meas_mhe_array]

    if k > N_mhe:
        Q = cov_process_disturb_w
        R = cov_meas_noise_v
        covars = { 'Q': Q, 'R': R}

        # minimize initialization
        x1_est_init_error = [0]
        x2_est_init_error = [0]
        x3_est_init_error = [0]
        x_est_init_error = np.concatenate(([x1_est_init_error], [
            x2_est_init_error], [x3_est_init_error]))

    # Guessed optim states:
    # Guessed present state ( $x_k$ ) is needed to calculate optimal
    # present meas
    # ( $y_k$ ). Model is used in prediction:
    x1_km1 = x1_est_optim_array[N_mhe-1]
    x2_km1 = x2_est_optim_array[N_mhe-1]
    x3_km1 = x3_est_optim_array[N_mhe-1]

    dx1_dt_km1 = x2_km1
    dx2_dt_km1 = (-x2_km1 + K*u_k + x3_km1)/T
    dx3_dt_km1 = 0

    x1_pred_k = x1_km1 + Ts*dx1_dt_km1
    x2_pred_k = x2_km1 + Ts*dx2_dt_km1
    x3_pred_k = x3_km1 + Ts*dx3_dt_km1

    # Now, guessed optimal states are:
    x1_est_guess_array = np.append(x1_est_optim_array[1:N_mhe],
        x1_pred_k)

```

```

x2_est_guess_array = np.append(x2_est_optim_array[1:N_mhe],
                               x2_pred_k)
x3_est_guess_array = np.append(x3_est_optim_array[1:N_mhe],
                               x3_pred_k)
# x_est_guess_matrix = np.concatenate((x1_est_guess_array,
#                                       x2_est_guess_array, x3_est_guess_array))
x_est_guess_matrix = [[x1_est_guess_array], [x2_est_guess_array],
                      [x3_est_guess_array]]

# Lower and upper limits of optim variables:
x1_est_max = 100
x2_est_max = 10
x3_est_max = 10

x1_est_min = -100
x2_est_min = -10
x3_est_min = -10

# Creating a flatten array of bounderies
# The bounderies must have the same size of x0 (
#   x_est_guess_matrix)
bnds = [(x1_est_min,x1_est_max)]*N_mhe + [(x2_est_min,x2_est_max)
                                             ]*N_mhe + [(x3_est_min,x3_est_max)]*N_mhe

x_est_optim_matrix_obj = minimize(objective, x_est_guess_matrix,
                                    method='SLSQP', bounds=bnds)
# Reshaping matrix because it is flattened after minimine call
x_est_optim_matrix = x_est_optim_matrix_obj.x.reshape(3,N_mhe)

x1_est_optim_array = x_est_optim_matrix[0]
x2_est_optim_array = x_est_optim_matrix[1]
x3_est_optim_array = x_est_optim_matrix[2]

x1_est_optim_plot_array[k] = x1_est_optim_array[-1]
x2_est_optim_plot_array[k] = x2_est_optim_array[-1]
x3_est_optim_plot_array[k] = x3_est_optim_array[-1]

# Continuous plotting
x_lim_array = [t_start, t_stop]
if (k>0 and k<N):
    if k > N_mhe:
        plt.pause(1)
    else:
        plt.pause(0.1)

plt.subplot(4,1,1)

```

```
lineU, = plt.plot([t_array[k-1], t_array[k]], [u_sim_array[k-1],  
                 u_sim_array[k]], 'b-o')  
  
if (k == 1):  
    lineU.set_label('u')  
    plt.legend()  
    plt.xlim(x_lim_array)  
    plt.ylim([0, 10])  
    plt.title('u')  
  
plt.subplot(4,1,2)  
lineX1Sim, lineX1Est, = plt.plot([t_array[k-1], t_array[k]], [  
    x1_est_optim_plot_array[k-1], x1_est_optim_plot_array[k]], 'r-o'  
, [t_array[k-1], t_array[k]], [x1_sim_array[k-1], x1_sim_array[k]]  
, 'b-o')  
  
if (k == 1):  
    lineX1Sim.set_label('x1 sim')  
    lineX1Est.set_label('x1 est')  
    plt.legend()  
    plt.xlim(x_lim_array)  
    plt.ylim([0, 100])  
  
plt.subplot(4,1,3)  
lineX2Sim, lineX2Est, = plt.plot([t_array[k-1], t_array[k]], [  
    x2_est_optim_plot_array[k-1], x2_est_optim_plot_array[k]], 'r-o'  
, [t_array[k-1], t_array[k]], [x2_sim_array[k-1], x2_sim_array[k]]  
, 'b-o')  
  
if (k == 1):  
    lineX2Sim.set_label('x2 sim')  
    lineX2Est.set_label('x2 est')  
    plt.legend()  
    plt.xlim(x_lim_array)  
    plt.ylim([0, 10])  
  
plt.subplot(4,1,4)  
lineX3Sim, lineX3Est, = plt.plot([t_array[k-1], t_array[k]], [  
    x3_est_optim_plot_array[k-1], x3_est_optim_plot_array[k]], 'r-o'  
, [t_array[k-1], t_array[k]], [x3_sim_array[k-1], x3_sim_array[k]]  
, 'b-o')  
  
if (k == 1):  
    lineX3Sim.set_label('x3 sim')  
    lineX3Est.set_label('x3 est')  
    plt.legend()  
    plt.xlim(x_lim_array)  
    plt.ylim([0, 3])  
  
plt.draw()
```

```
|| plt.show(block=True)
```

Fonte: Autor, adaptado de Haugen (2018)

A.4 Exemplo de aplicação do Controle Preditivo Baseado em Modelo

Código-fonte A.4 – Exemplo de aplicação do Controle Preditivo Baseado em Modelo

```
# Linguagem: Python
# Autor: Tiago Correia Prata
# Disponível em:
# <https://github.com/TiagoPrata/MasterThesis/blob/master/4_codes/mpc_example.py>

import numpy as np
import math
from scipy import signal
from scipy.optimize import minimize
import matplotlib.pyplot as plt

# -----
# Process params:
gain = 3.5                      #[deg C]/[V]
theta_const = 23                  #[s]
theta_delay = 3                   #[s]
model_params = {
    'gain': gain,
    'theta_const': theta_const,
    'theta_delay': theta_delay
}

Temp_env_k = 25                   #[deg C]

# -----
# Time settings:

Ts = 0.5                         #Time-step [s]
t_pred_horizon = 8

N_pred = int(t_pred_horizon/Ts)
t_start = 0
t_stop = 300
N_sim = int((t_stop-t_start)/Ts)
t = np.linspace(t_start, t_stop-Ts, t_stop / Ts)
```

```

# -----
# Storage allocations
Temp_sp_array = t*0

# -----
# MPC costs:

C_e = 1
C_du = 20
mpc_costs = {
    'C_e': C_e,
    'C_du': C_du
}

#-----
# Defining sequence for temp_out setpoint:

Temp_sp_const = 30          #[C]
Ampl_step = 2                 #[C]
Slope = -0.04                #[C/s]
Ampl_sine = 1                  #[C]
T_period = 50                  #[s]

t_const_start = t_start
t_const_stop = 100
t_step_start = t_const_stop
t_step_stop = 150
t_ramp_start = t_step_stop
t_ramp_stop = 200
t_sine_start = t_ramp_stop
t_sine_stop = 250
t_const2_start = t_sine_stop
t_const2_stop = t_stop

for k in range(N_sim):
    if (t[k] >= t_const_start and t[k] < t_const_stop):
        Temp_sp_array[k] = Temp_sp_const

    if (t[k] >= t_step_start and t[k] < t_step_stop):
        Temp_sp_array[k] = Temp_sp_const + Ampl_step

    if (t[k] >= t_ramp_start and t[k] < t_ramp_stop):
        Temp_sp_array[k] = Temp_sp_const + Ampl_step + Slope * (t[k] -
            t_ramp_start)

    if (t[k] >= t_sine_start and t[k] < t_sine_stop):

```

```
Temp_sp_array[k] = Temp_sp_const + Ampl_sine * math.sin(2*math.pi * (1/T_period) * (t[k] - t_sine_start))

if (t[k] >= t_const2_start):
    Temp_sp_array[k] = Temp_sp_const

#-----
#Initialization:

u_init = 0
N_delay = math.floor(theta_delay/Ts) + 1
delay_array = np.zeros(N_delay) + u_init

#-----
#Initial guessed optimal control sequence:

Temp_heat_sim_k = 0           #[C]
Temp_out_sim_k = 28            #[C]
d_sim_k = -0.5

#-----
#Initial values for estimator:

Temp_heat_est_k = 0           #[C]
Temp_out_est_k = 25            #[C]
d_est_k = 0                   #[V]

#-----
#Initial guessed optimal control sequence:
u_guess = np.zeros(N_pred) + u_init
u_guess = np.transpose(u_guess)

#-----
#Initial value of previous optimal value:
u_opt_km1 = u_init

# -----
# Defining arrays for plotting:

t_plot_array = np.zeros(N_sim)
Temp_out_sp_plot_array = np.zeros(N_sim)
Temp_out_sim_plot_array = np.zeros(N_sim)
u_plot_array = np.zeros(N_sim)
d_est_plot_array = np.zeros(N_sim)
d_sim_plot_array = np.zeros(N_sim)

# -----
```

```

# Matrices defining linear constraints for use in fmincon:

# A = []
# B = []
# Aeq = []
# Beq = []

# -----
# Lower and upper limits of optim variable for use in fmincon:

u_max = 5
u_min = 0
u_ub = np.zeros(N_pred) + u_max
u_lb = np.zeros(N_pred) + u_min

u_delayed_k = 2

# -----
# Calculation of observer gain for estimation of input disturbance, d:

A_est = np.array([[-1/theta_const, gain/theta_const],[0, 0]])
C_est = np.array([[1, 0]])
n_est = len(A_est)
Tr_est = 5
T_est = Tr_est/n_est
estimpoly = np.array([T_est*T_est, math.sqrt(2)*T_est, 1])
eig_est = np.roots(estimpoly)
K_est1 = signal.place_poles(A_est.transpose(), C_est.transpose(),
    eig_est)
K_est = K_est1.gain_matrix.transpose()

# Configuring continuous plotting
plt.ion()

def objective(u, state_est, Temp_sp_to_mpc_array):
    gain = model_params['gain']
    theta_const = model_params['theta_const']
    theta_delay = model_params['theta_delay']

    C_e = mpc_costs['C_e']
    C_du = mpc_costs['C_du']

    Temp_heat_k = state_est['Temp_heat_est_k']
    d_k = state_est['d_est_k']

    N_delay = math.floor(theta_delay/Ts) + 1
    delay_array = np.zeros(N_delay) + u[0]

```

```

ru_km1 = u_opt_km1
u_km1 = u[0]
J_km1 = 0

for k in range(N_pred):
    u_k = u[k]
    Temp_sp_k = Temp_sp_to_mpc_array[k]

    # Time delay
    u_delayed_k = delay_array[N_delay-1]
    u_nondelayed_k = u_k
    delay_array = np.insert(delay_array, 0, u_nondelayed_k)
    delay_array = delay_array[:-1]

    # Solving diff eq
    dTemp_heat_dt_k = (1/theta_const) * (-Temp_heat_k + gain *(
        u_delayed_k + d_k))
    Temp_heat_kp1 = Temp_heat_k + Ts*dTemp_heat_dt_k
    Temp_out_k = Temp_heat_k + Temp_env_k

    # Updating objective function
    e_k = Temp_sp_k - Temp_out_k
    du_k = (u_k - u_km1)/Ts
    J_k = J_km1 + Ts*(C_e * e_k**2 + C_du * du_k**2)

    # Time shift
    Temp_heat_k = Temp_heat_kp1
    u_km1 = u_k
    J_km1 = J_k

return J_k

def constraints(u):
    cineq = np.array([])
    ceq = np.array([])

    return (cineq, ceq)

# Creating matrix
Temp_out_est_plot_array = np.zeros(N_sim - N_pred)

for k in range(N_sim - N_pred):
    t_k = t[k]
    t_plot_array[k] = t_k

# -----

```

```

# Observer for estimating input-disturbance d using Temp_out:
# Note: The time-delayed u is used as control signal shere.
e_est_k = Temp_out_sim_k - Temp_out_est_k

dTTemp_heat_est_dt_k = (1/theta_const) * (-Temp_heat_est_k + gain*(u_delayed_k + d_est_k)) + K_est[0] * e_est_k
dd_est_dt_k = 0 + K_est[1] * e_est_k

Temp_heat_est_kp1 = Temp_heat_est_k + Ts*dTemp_heat_est_dt_k
d_est_kp1 = d_est_k + Ts*dd_est_dt_k

Temp_out_est_k = Temp_heat_est_k + Temp_env_k

# -----
# Storage for plotting
Temp_out_est_plot_array[k] = Temp_out_est_k
d_est_plot_array[k] = d_est_k

# -----
# Setpoint array to optimizer
Temp_sp_to_mpc_array = Temp_sp_array[k:k+N_pred]
Temp_out_sp_plot_array[k] = Temp_sp_array[k]

# -----
# Estimated state to optimizer
state_est = {
    'Temp_heat_est_k': Temp_heat_est_k,
    'd_est_k': d_est_k
}

# -----
# Calculating optimal control sequence
u_opt = minimize(objective, u_guess, args=(state_est,
    Temp_sp_to_mpc_array), method='SLSQP')

u_guess = u_opt.x.reshape(1,N_pred)
u_k = u_opt.x[0]
u_plot_array[k] = u_k
u_opt_km1 = u_opt.x[0]

# -----
# Applying optimal control signal to simulated process
d_sim_k = -0.5
d_sim_plot_array[k] = d_sim_k

u_delayed_k = delay_array[N_delay-1]
u_nondelayed_k = u_k

```

```

delay_array = np.insert(delay_array, 0, u_nondelayed_k)
delay_array = delay_array[:-1]

dTTemp_heat_sim_dt_k = (1/theta_const) * (-Temp_heat_sim_k + gain *(
    u_delayed_k + d_sim_k))
Temp_heat_sim_kp1 = Temp_heat_sim_k + Ts*dTemp_heat_sim_dt_k
Temp_out_sim_k = Temp_heat_sim_k + Temp_env_k

Temp_out_sim_plot_array[k] = Temp_out_sim_k

# -----
# Time shift for estimator and for simulator
Temp_heat_est_k = Temp_heat_est_kp1
d_est_k = d_est_kp1

Temp_heat_sim_k = Temp_heat_sim_kp1

# -----
# Continuous plotting
x_lim_array = t_start
x_lim_array = np.append(x_lim_array, t_stop)

if (k > 0 and k < N_sim):
    plt.pause(0.1)

    plt.subplot(3,1,1)
    lineSP, lineSim, lineEst = \
        plt.plot([t_plot_array[k-1], t_plot_array[k]], \
                 [Temp_out_sp_plot_array[k-1], Temp_out_sp_plot_array[k]], \
                 'r-', \
                 [t_plot_array[k-1], t_plot_array[k]], \
                 [Temp_out_sim_plot_array[k-1], Temp_out_sim_plot_array[k]], \
                 'b-', \
                 [t_plot_array[k-1], t_plot_array[k]], \
                 [Temp_out_est_plot_array[k-1], Temp_out_est_plot_array[k]], \
                 'm-')
    if (k == 1):
        lineSP.set_label('SP')
        lineSim.set_label('sim')
        lineEst.set_label('est')
        plt.legend()
        plt.xlim(x_lim_array)
        plt.ylim([28, 33])
        plt.xlabel('t [s]')

```

```

    plt.ylabel(' [deg C] ')

    plt.subplot(3,1,2)
    lineU, = \
        plt.plot([t_plot_array[k-1], t_plot_array[k]], \
                 [u_plot_array[k-1], u_plot_array[k]], \
                 'b-')
    if (k == 1):
        lineU.set_label('u')
        plt.legend()
        plt.xlim(x_lim_array)
        plt.ylim([0, 5])
        plt.xlabel('t [s]')
        plt.ylabel(' [V]')

    plt.subplot(3,1,3)
    line_d_est, line_d_sim = \
        plt.plot([t_plot_array[k-1], t_plot_array[k]], \
                 [d_est_plot_array[k-1], d_est_plot_array[k]], \
                 'r-', \
                 [t_plot_array[k-1], t_plot_array[k]], \
                 [d_sim_plot_array[k-1], d_sim_plot_array[k]], \
                 'b-')
    if (k == 1):
        line_d_est.set_label('d est')
        line_d_sim.set_label('d sim')
        plt.legend()
        plt.xlim(x_lim_array)
        # plt.ylim([0, 5])
        plt.xlabel('t [s]')

    plt.draw()

plt.show(block=True)

```

Fonte: Autor, adaptado de Haugen (2018)

A.5 Criação de diversos modelos experimentais do TCLabSP

Código-fonte A.5 – Criação de diversos modelos experimentais do TCLabSP

```

%% Cria modelos função de transferencia
%poles = [S1H1 S1H2; S2H1 S2H2]
%zeros = [S1H1 S1H2; S2H1 S2H2]
%ex: sys = tfest(train, [2 2; 2 2], [1 1; 1 1], 'Ts', Ts_target);

```

```
|| disp(char(strcat('TF:', {' ', }, datestr(datetime))));

models_tf = struct();
max_poles = 5;
max_zeros = 4;

i = 0;
for pS1H1 = 2:max_poles
    for pS1H2 = 2:max_poles
        for pS2H1 = 2:max_poles
            for pS2H2 = 2:max_poles
                for zS1H1 = 1:max_zeros
                    if (zS1H1 >= pS1H1), break, end
                    for zS1H2 = 1:max_zeros
                        if (zS1H2 >= pS1H2), break, end
                        for zS2H1 = 1:max_zeros
                            if (zS2H1 >= pS2H1), break, end
                            for zS2H2 = 1:max_zeros
                                if (zS2H2 >= pS2H2), break, end
                                i = i+1;
                                models_tf(i).name = strcat('tf', int2str(pS1H1), ...
                                int2str(pS1H2), ...
                                int2str(pS2H1), ...
                                int2str(pS2H2), ...
                                int2str(zS1H1), ...
                                int2str(zS1H2), ...
                                int2str(zS2H1), ...
                                int2str(zS2H2));

                                models_tf(i).model = tfest(train, [pS1H1 pS1H2; ...
                                pS2H1 pS2H2], ...
                                [zS1H1 zS1H2; ...
                                zS2H1 zS2H2], ...
                                'Ts', Ts_target);

[y,fit,x0] = compare(valid, models_tf(i).model);
models_tf(i).param_num = pS1H1+pS1H2+pS2H1+pS2H2+ ...
zS1H1+zS1H2+zS2H1+zS2H2;

models_tf(i).fit = fit;
models_tf(i).score = prod(fit,1);
models_tf(i).pS1H1 = pS1H1;
models_tf(i).pS1H2 = pS1H2;
models_tf(i).pS2H1 = pS2H1;
models_tf(i).pS2H2 = pS2H2;
models_tf(i).zS1H1 = zS1H1;
models_tf(i).zS1H2 = zS1H2;
```

```

    models_tf(i).zS2H1 = zS2H1;
    models_tf(i).zS2H2 = zS2H2;
    models_tf(i).MSE = models_tf(i).model.Report.Fit.MSE;
    models_tf(i).FPE = models_tf(i).model.Report.Fit.FPE;
    models_tf(i).AIC = models_tf(i).model.Report.Fit.AIC;
    models_tf(i).AICc = models_tf(i).model.Report.Fit.AICc;
    models_tf(i).nAIC = models_tf(i).model.Report.Fit.nAIC;
    models_tf(i).BIC = models_tf(i).model.Report.Fit.BIC;
end
end
end
end
end
end
end

%% Cria modelos de espaço de estados

% sys = ssest(train, 1:10, 'Ts', Ts_target);

disp(char(strcat('SS:', {' '}, datestr(datetime))));
models_ss = struct();
for i=1:10
    name_ss = strcat('ss', int2str(i));
    models_ss(i).model = ssest(train, i, 'Ts', Ts_target);
    [y, fit, x0] = compare(valid, models_ss(i).name);
    models_ss(i).fit = fit;
    models_ss(i).order = i;
    models_ss(i).score = prod(fit, 1);
    models_ss(i).MSE = models_ss(i).model.Report.MSE;
    models_ss(i).FPE = models_ss(i).model.Report.FPE;
    models_ss(i).AIC = models_ss(i).model.Report.AIC;
    models_ss(i).AICc = models_ss(i).model.Report.AICc;
    models_ss(i).nAIC = models_ss(i).model.Report.nAIC;
    models_ss(i).BIC = models_ss(i).model.Report.BIC;
end

%% Cria modelos ARX e OE

% arx
%sys = arx(train, [2*ones(2,2) 2*ones(2,2) 1*ones(2,2)]);
% oe
%sys = oe(train, [2*ones(2,2), 2*ones(2,2), ones(2,2)]);

disp(char(strcat('ARX e OE:', {' '}, datestr(datetime))))

```

```
na_range = 1:4;
na_size = ones(2,2);
na = bsxfun(@times,na_size,reshape(na_range,1,1,numel(na_range)));

nb_range = 1:4;
nb_size = ones(2,2);
nb = bsxfun(@times,nb_size,reshape(nb_range,1,1,numel(nb_range)));

nk_range = 0:4;
nk_size = ones(2,2);
nk = bsxfun(@times,nk_size,reshape(nk_range,1,1,numel(nk_range)));

models_arx = struct();
models_oe = struct();
i=0;

for na_i=1:size(na_range,2)
    for nb_i=1:size(nb_range,2)
        for nk_i=1:size(nk_range,2)
            i = i+1;
            models_arx(i).name = strcat('na_',int2str(na_i), ...
                '_nb_',int2str(nb_i), ...
                '_nk_',int2str(nk_i));

            models_arx(i).model = arx(train, [na(:,:,na_i) ...
                nb(:,:,nb_i) ...
                nk(:,:,nk_i)]);

            models_arx(i).aic = aic(models_arx(i).model);
            [y,fit,x0] = compare(valid, models_arx(i).model);
            models_arx(i).fit = fit;
            if (fit(1) <= 0) || (fit(2) <= 0)
                models_arx(i).score = 0;
            else
                models_arx(i).score = prod(fit,1);
            end

models_oe(i).name = strcat('nb_',int2str(na_i), ...
    '_nf_',int2str(nb_i), ...
    '_nk_',int2str(nk_i));

models_oe(i).model = oe(train, [na(:,:,na_i) ...
    nb(:,:,nb_i) ...
    nk(:,:,nk_i)]);

models_oe(i).aic = aic(models_oe(i).model);
[y,fit,x0] = compare(valid, models_oe(i).model);
```

```

        models_oe(i).fit = fit;
        if (fit(1) <= 0) || (fit(2) <= 0)
            models_oe(i).score = 0;
        else
            models_oe(i).score = prod(fit,1);
        end
    end
end

%% Cria modelos armax
%sys = armax(train, [2*ones(2,2), 2*ones(2,2), 2*ones(2,1), ones(2,2)]);

disp(char(strcat('ARMAX:', {' '}, datestr(datetime))));
na_range = 1:4;
na_size = ones(2,2);
na = bsxfun(@times,na_size,reshape(na_range,1,1,numel(na_range)));

nb_range = 1:4;
nb_size = ones(2,2);
nb = bsxfun(@times,nb_size,reshape(nb_range,1,1,numel(nb_range)));

nc_range = 1:4;
nc_size = ones(2,1);
nc = bsxfun(@times,nc_size,reshape(nc_range,1,1,numel(nc_range)));

nk_range = 0:4;
nk_size = ones(2,2);
nk = bsxfun(@times,nk_size,reshape(nk_range,1,1,numel(nk_range)));

models_armax = struct();
i=0;
for na_i=1:size(na_range,2)
    for nb_i=1:size(nb_range,2)
        for nc_i=1:size(nc_range,2)
            for nk_i=1:size(nk_range,2)
                i = i+1;
                models_armax(i).name = strcat('na_',int2str(na_i), ...
                    '_nb_',int2str(nb_i), ...
                    '_nc_',int2str(nc_i), ...
                    '_nk_',int2str(nk_i));

                models_armax(i).model = armax(train, [na(:,:,na_i) ...
                    nb(:,:,nb_i) ...
                    nc(:,:,nc_i) ...
                    nk(:,:,nk_i)]);
            end
        end
    end
end

```

```
models_arimax(i).aic = aic(models_arimax(i).model);
[y,fit,x0] = compare(valid, models_arimax(i).model);
models_arimax(i).fit = fit;
if (fit(1) <= 0) || (fit(2) <= 0)
    models_arimax(i).score = 0;
else
    models_arimax(i).score = prod(fit,1);
end
end
end
end

%% Cria modelos bj
%sys = bj(train, [2*ones(2,2), 2*ones(2,1), ...
%                 2*ones(2,1), 2*ones(2,2), ones(2,2)]);

disp(char(strcat('BJ:', {' '}, datestr(datetime))));
nb_range = 1:4;
nb_size = ones(2,2);
nb = bsxfun(@times,nb_size,reshape(nb_range,1,1,numel(nb_range)));

nc_range = 1:4;
nc_size = ones(2,1);
nc = bsxfun(@times,nc_size,reshape(nc_range,1,1,numel(nc_range)));

nd_range = 1:4;
nd_size = ones(2,1);
nd = bsxfun(@times,nd_size,reshape(nd_range,1,1,numel(nd_range)));

nf_range = 1:4;
nf_size = ones(2,2);
nf = bsxfun(@times,nf_size,reshape(nf_range,1,1,numel(nf_range)));

nk_range = 1:4;
nk_size = ones(2,2);
nk = bsxfun(@times,nk_size,reshape(nk_range,1,1,numel(nk_range)));

models_bj = struct();
i=0;
for nb_i=1:size(nb_range,2)
    for nc_i=1:size(nc_range,2)
        for nd_i=1:size(nd_range,2)
            for nf_i=1:size(nf_range,2)
                for nk_i=1:size(nk_range,2)
                    i = i+1;
```

```

models_bj(i).name = strcat('nb_',int2str(nb_i), ...
                           '_nc_',int2str(nc_i), ...
                           '_nd_',int2str(nd_i), ...
                           '_nf_',int2str(nf_i), ...
                           '_nk_',int2str(nk_i));

models_bj(i).model = bj(train, [nb(:,:,nb_i) ...
                                 nc(:,:,nc_i) ...
                                 nd(:,:,nd_i) ...
                                 nf(:,:,nf_i) ...
                                 nk(:,:,nk_i)]);

models_bj(i).aic = aic(models_bj(i).model);
[y,fit,x0] = compare(valid, models_bj(i).model);
models_bj(i).fit = fit;
if (fit(1) <= 0) || (fit(2) <= 0)
    models_bj(i).score = 0;
else
    models_bj(i).score = prod(fit,1);
end
end
end
end

%% Cria modelo não-linear

% na = [S1-TermsOutputs1 S1-TermsOutputs2; ...
%        S2-TermsOutputs1 S2-TermsOutputs2]
% nb = [S1-NumTermosH1 S1-NumTermosH2; ...
%        S2-NumTermosH1 S2-NumTermosH2]
% nk = [S1-DelayH1 S1-DelayH2; S2-DelayH1 S2-DelayH2]

% na = [2 0; 0 2];
% nb = [2 2; 2 2];
% nk = [1 1; 1 1];
% Orders = [na,nb,nk];
% sys = nlarx(train, Orders);

disp(char(strcat('NLARX:', {' '}, datestr(datetime))))
models_nlarx = struct();
i=0;
for na_1=1:2
    for na_4=1:2
        for nb_1=0:2
            for nb_2=0:2

```


Fonte: Autor

A.6 Identificação do tempo de subida do TCLabSP

Código-fonte A.6 – Calcula tempo de subida de primeira ordem do TCLabSP

```

%% Description
% Este script faz uma média de todos os resultados coletados
% e através dela obtém o tempo de subida da resposta em malha
% aberta dos Sensores 1 e 2.

%% Carregar média dos ensaios
data_folder_name = strcat(pwd, '\Ensaios');
ensaios = combinar_experimentos(data_folder_name);

%% calcula periodo de amostragem utilizado
Ts = ensaios.(‘time’)(2) - ensaios.(‘time’)(1);

%% encontrando valores iniciais das respostas dos Sensores
valor_inicial_y1 = media_regiao(ensaios.(‘y1_Media’), 1, 0, 5);
valor_inicial_y2 = media_regiao(ensaios.(‘y2_Media’), 1, 0, 5);

%% ajustando condições iniciais para zero
y1_fromZero = ensaios.(‘y1_Media’) - valor_inicial_y1;
y2_fromZero = ensaios.(‘y2_Media’) - valor_inicial_y2;

%% encontrando valores finais das respostas dos Sensores
valor_final_y1 = media_regiao(y1_fromZero, length(y1_fromZero), 5, 0);
valor_final_y2 = media_regiao(y2_fromZero, length(y2_fromZero), 5, 0);

%% encontrando valor correspondente a 63,21% do valor final das
%% respostas
y1_tau_value = 0.6321*valor_final_y1;
y1_tau_index = find(y1_fromZero >= y1_tau_value);
y1_tau_index = y1_tau_index(1);
y1_tau = y1_tau_index * Ts;
disp(char(strcat({‘Tempo de subida do sensor 1 = ’}, num2str(y1_tau), ‘’)));
y2_tau_value = 0.6321*valor_final_y2;
y2_tau_index = find(y2_fromZero >= y2_tau_value);
y2_tau_index = y2_tau_index(1);
y2_tau = y2_tau_index * Ts;
disp(char(strcat({‘Tempo de subida do sensor 2 = ’}, num2str(y2_tau), ‘’)));

```

Fonte: Autor

Código-fonte A.7 – Função auxiliar: Combina experimentos do TCLabSP

```

function [ exp ] = combinar_experimentos( folderFullPath )
%combinar_experimentos Esta função combina todos os experimentos
% de uma pasta. Esta pasta deve conter diversos ensaios de um
% mesmo tipo. Esperasse também que todos os ensaios possuam uma
% variável chamada 'y1y2' com os valores de saída e uma chamada
% 'u1u2' com os valores de entrada no sistema.

if folderFullPath == 0, return, end

localStruct = struct();

myFiles = dir(fullfile(folderFullPath, '*.mat'));
for k = 1:length(myFiles)
    baseFileName = myFiles(k).name;
    fullFileName = fullfile(folderFullPath, baseFileName);
    load(fullFileName);

    localStruct.(strcat('u1_', int2str(k))) = u1u2.Data(:,1);
    localStruct.(strcat('u2_', int2str(k))) = u1u2.Data(:,2);

    localStruct.(strcat('y1_', int2str(k))) = y1y2.Data(:,1);
    localStruct.(strcat('y2_', int2str(k))) = y1y2.Data(:,2);

try
    localStruct.( 'y1_Media' ) = localStruct.( 'y1_Media' ) + y1y2.Data
        (:,1);
catch ME
    switch ME.identifier
        case 'MATLAB:nonExistentField'
            localStruct.( 'y1_Media' ) = 0;
            localStruct.( 'y1_Media' ) = localStruct.( 'y1_Media' ) +
                y1y2.Data(:,1);
    end
end

try
    localStruct.( 'y2_Media' ) = localStruct.( 'y2_Media' ) + y1y2.Data
        (:,2);
catch ME
    switch ME.identifier
        case 'MATLAB:nonExistentField'
            localStruct.( 'y2_Media' ) = 0;

```

```

    localStruct.( 'y2_Media' ) = localStruct.( 'y2_Media' ) +
        y1y2.Data(:,2);
    end
end

localStruct.( 'y1_Media' ) = localStruct.( 'y1_Media' ) / k;
localStruct.( 'y2_Media' ) = localStruct.( 'y2_Media' ) / k;

localStruct.( 'time' ) = y1y2.Time;

try
    localArray = [localStruct.( 'time' ) localStruct.( 'u1_1' ) localStruct.
        .( 'u2_1' ) localStruct.( 'y1_Media' ) localStruct.( 'y2_Media' )];
    localTable = array2table(localArray, 'VariableNames', { 'Time', ,
        'Aquecedor1', 'Aquecedor2', 'Sensor1', 'Sensor2' });
    exp = localTable;
catch ME
    switch ME.identifier
        case 'MATLAB:catenate:dimensionMismatch'
            exp = localStruct;
    end
end
end
end

```

Fonte: Autor

Código-fonte A.8 – Função auxiliar: Calcula média de uma região

```

function [ value ] = media_regiao ( signal, index, num_of_left_values,
    num_of_right_values )
%media_regiao Calcula o valor médio dos pontos de um determinado
% sinal em torno de uma certa região.
% A média será calculada em um determinado índice, incluindo
% as amostras à esquerda e à direita do índice.

start_index = index - num_of_left_values;
end_index = index + num_of_right_values;

range = signal(start_index : end_index);

value = mean(range);

```

```
|| end
```

Fonte: Autor

Código-fonte A.9 – Plota tempo de subida

```
%% Description
% Este script plota os tempos de subida dos sensores 1 e 2.

%% plotando gráfico - Sensor 1
y1_tau_lineX = y1_tau * ones(1, 2);
y1_tau_lineY = y1_tau_value * ones(1, 2);
y1_final_lineY = valor_final_y1 * ones(1, 2);

figure(1);
plot(ensaio(:, 'time'), y1_fromZero, 'k-', ...
    y1_tau_lineX, [0 y1_tau_value], 'r--', ...
    [0 y1_tau], y1_tau_lineY, 'r--', ...
    [0 length(y1_fromZero)*Ts], y1_final_lineY, 'b--')

legend({'Sensor de Temp 1', ...
    strcat('t=', int2str(y1_tau), 's')}, ...
    'FontSize', 14, ...
    'Location', 'best');

title('Tempo de subida - Sensor de Temperatura 1', 'FontSize', 14);
xlabel('Tempo (s)', 'FontSize', 14)
ylabel({'Temperatura (°C)'; '[Temp. inicial subtraída]'}, 'FontSize', 14)
;

%% plotando gráfico - Sensor 2
y2_tau_lineX = y2_tau * ones(1, 2);
y2_tau_lineY = y2_tau_value * ones(1, 2);
y2_final_lineY = valor_final_y2 * ones(1, 2);

figure(2);
plot(ensaio(:, 'time'), y2_fromZero, 'k-', ...
    y2_tau_lineX, [0 y2_tau_value], 'r--', ...
    [0 y2_tau], y2_tau_lineY, 'r--', ...
    [0 length(y2_fromZero)*Ts], y2_final_lineY, 'b--')

legend({'Sensor de Temp 2', ...
    strcat('t=', int2str(y2_tau), 's')}, ...
    'FontSize', 14, ...
    'Location', 'best');
```

```

title('Tempo de subida - Sensor de Temperatura 2', 'FontSize', 14);
xlabel('Tempo (s)', 'FontSize', 14)
ylabel({'Temperatura (°C)'; '[Temp. inicial subtraída]'}, 'FontSize', 14)
;

```

Fonte: Autor

A.7 Encontrar Delta amostral utilizando autocovariância

Código-fonte A.10 – Encontra o Delta e o Ts

```

%% Description
% Este script calcula o menor delta para decimar a amostragem
% original

delta_y1 = find_delta(ensaios.'y1_Media');
delta_y2 = find_delta(ensaios.'y2_Media');

delta = min(delta_y1, delta_y2);
disp(char(strcat({'Delta = '}, num2str(delta))));

time_with_delta = ensaios.'time')(1:delta:end);

Ts = time_with_delta(2) - time_with_delta(1);
disp(char(strcat({'Tempo de amostragem = '}, num2str(Ts), 's')));

disp('');
disp('É possível utilizar o ensaio já realizado, aplicando delta para');
disp('evitar a superamostragem existente. Ex. sinal(1:delta:end)');
disp('Ou pode-se realizar um novo ensaio, utilizando Ts como intervalo')
;
disp('de amostragem.');

```

Fonte: Autor

Código-fonte A.11 – Função auxiliar: Encontra Delta

```

function [ delta ] = find_delta( signal )
%find_delta Função para encontrar o Delta que satisfaz 5 <= tm <= 25

%% Encontra tm para o sinal original

[ ry, ry_quad, ty, ty_quad, tm_orig ] = autocorr_min(signal);

```

```

%% Encontrar Delta para que 5 <= tm <= 25

if tm_orig > 25
    for i = 1:length(signal)
        [ ry, ry_quad, ty, ty_quad, tm ] = autocorr_min(signal(1:i:end))
        ;
        if (tm >= 5) && (tm <= 25)
            break
        end
    end
end

%% Resultado
delta = i;

end

```

Fonte: Autor

Código-fonte A.12 – Função auxiliar: Calcula autocovariância

```

function [ ry, ry_quad, ty, ty_quad, tm ] = autocorr_min ( signal )
%autocorr_with_delta Esta funcao calcula a autocovariancia linear
% e quadratica do sinal de trabalho (signal) e encontra o
% menor minimo entre elas

ry = autocorr(signal, length(signal) - 1);
ry_quad = autocorr(signal.^2, length(signal) - 1);

%% Calcula os minimos das autocovariâncias

ty = find(ry <= min(ry));
ty = ty(1);
ty_quad = find(ry_quad <= min(ry_quad));
ty_quad = ty_quad(1);

%% Encontra valor de trabalho tm

tm_y = min(ty, ty_quad);

%% Resultado

tm = tm_y;

```

```
|| end
```

Fonte: Autor

Código-fonte A.13 – Plota autocovariância

```
[ ry1, ry1_quad, ty1, ty1_quad, tm1_orig ] = autocorr_min(ensaios.(  
    'y1_Media'));  
[ ry2, ry2_quad, ty2, ty2_quad, tm2_orig ] = autocorr_min(ensaios.(  
    'y2_Media'));  
  
figure();  
subplot(2,2,1)  
area(ry1);  
title('Autocovariância linear - Sensor 1', 'FontSize', 14);  
xlabel('Atraso (\tau)', 'FontSize', 14);  
ylabel('r_{y^*}', 'FontSize', 14);  
subplot(2,2,2)  
area(ry1_quad);  
title('Autocovariância não-linear - Sensor 1', 'FontSize', 14);  
xlabel('Atraso (\tau)', 'FontSize', 14);  
ylabel('r_{y^{*2}}', 'FontSize', 14);  
subplot(2,2,3)  
area(ry2);  
title('Autocovariância linear - Sensor 2', 'FontSize', 14);  
xlabel('Atraso (\tau)', 'FontSize', 14);  
ylabel('r_{y^*}', 'FontSize', 14);  
subplot(2,2,4)  
area(ry2_quad);  
title('Autocovariância não-linear - Sensor 2', 'FontSize', 14);  
xlabel('Atraso (\tau)', 'FontSize', 14);  
ylabel('r_{y^{*2}}', 'FontSize', 14);
```

Fonte: Autor

APÊNDICE B – Escolha da frequência de amostragem utilizando autocovariância

A regra prática para a escolha da frequência de amostragem de sinal descrita por Aguirre (2015) e citada na seção 5.1.1 pode nem sempre ajudar muito, uma vez que é possível que não se tenha nenhum conhecimento prévio do sinal para poder aplicar a regra de escolher uma frequência de 5 a 10 vezes maior que a frequência de interesse (AGUIRRE, 2015). Para casos assim, Aguirre (2015) descreve uma outra técnica, que será apresentada nos parágrafos a seguir.

Assume-se que o sinal já amostrado, $y^*(k)$, tenha sido obtido utilizando-se um tempo de amostragem muito pequeno, ou seja, o sinal está superamostrado. Sendo assim, deseja-se encontrar o valor Δ que descreva a fração pela qual o $y^*(k)$ pode ser decimado a fim de obter um sinal de trabalho $y(k) = y^*(\Delta k)$, ou seja, o sinal decimado que ainda mantém as características originais do sinal $y^*(k)$.

Para fazer isso é necessário verificar o nível de correlação entre observações adjacentes do sinal $y^*(k)$. Quanto mais superamostrado estiver o sinal $y^*(k)$, maior será a redundância entre duas observações consecutivas. Este nível de correlação pode ser obtido calculando as funções de autocovariância linear e não-linear (mostradas na eq. (B.1)) do sinal original.

$$r_{y^*}(\tau) = E \left[(y^*(k) - \bar{y^*(k)})(y^*(k - \tau) - \bar{y^*(k)}) \right] \quad (\text{B.1a})$$

$$r_{y^{*2'}}(\tau) = E \left[(y^{*2}(k) - \bar{y^{*2}(k)})(y^{*2}(k - \tau) - \bar{y^{*2}(k)}) \right] \quad (\text{B.1b})$$

$E[\cdot]$ indica a esperança matemática, porém, sendo $y^*(k)$ ergótico, $E[\cdot]$ pode ser substituída pela média temporal.

Após calculados $r_{y^*}(\tau)$ e $r_{y^{*2'}}(\tau)$, determinam-se seus primeiros mínimos, τ_{y^*} e $\tau_{y^{*2'}}$, respectivamente. O menor desses mínimos passa a ser o valor de trabalho τ_m^* , ou seja, $\tau_m^* = \min [\tau_{y^*}, \tau_{y^{*2'}}]$.

Por fim, deseja-se escolha Δ de forma que as funções de autocovariância do sinal decimado $y(k) = y^*(k)$ satisfaçam

$$10 \leq \tau_m \leq 20 \quad (\text{B.2})$$

sendo que τ_m é definido pelo sinal decimado $y(k)$ de maneira análoga a τ_m^* para o sinal $y^*(k)$. Os limites inferior e superior da eq. (B.2) podem ser relaxados para 5 e 25, respectivamente.

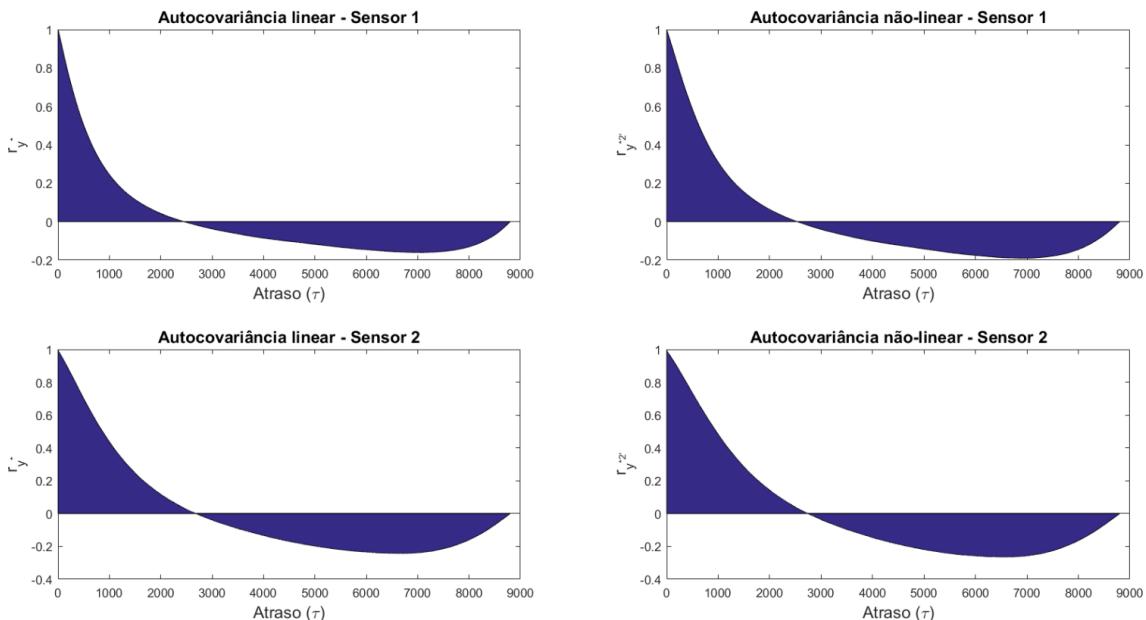
B.1 Aplicação prática

As figs. 21 e 22 da seção 5.1.1 apresentam a resposta dos sensores de temperatura do TCLabSP quando o Aquecedor 1 é excitado com um degrau de 0 à 50%.

Após alguns testes preliminares, foi observado que o sistema apresentava uma resposta lenta e que, portanto, uma coleta de dados com tempo de amostragem de $0.25s$ seria suficiente para superamostrar os dados e garantir que todas as características de resposta do sistema tenham sido amostradas.

Calcula-se então as funções de autocovariância apresentadas na eq. (B.1) para os sensores de temperatura 1 e 2, como apresentado na fig. 40.

Figura 40 – Funções de autocovariância dos Sensores de Temperatura 1 e 2



Fonte: Prata (2019)

Para cada um dos sensores, encontramos então o valor mínimo das funções de autocovariância (τ_{y^*} e $\tau_{y^{*2'}}$), e a partir delas determinamos τ_m^* ($\tau_m^* = \min[\tau_{y^*}, \tau_{y^{*2'}}]$). O resultado é apresentado na tabela 23.

Tabela 23 – Mínimos das funções de autocovariância

Sensor	τ_{y^*}	$\tau_{y^{*2'}}$	τ_m^*
Sensor 1	7015	6925	6925
Sensor 2	6735	6621	6621

Fonte: Autor

Conhecendo τ_m^* é possível encontrar um valor de Δ para obter o sinal decimado $y(k) = y^*(\Delta k)$, tal que o valor mínimo da suas funções de autocovariância satisfaçam eq. (B.2).

Uma demonstração do efeito da variação do Δ no tempo de amostragem do sinal pode ser vista na [tabela 24](#)

Tabela 24 – Efeito do Δ no tempo de amostragem

k	$\Delta = 1$	$\Delta = 2$...	$\Delta = 4$...
1	0.00	0.00		0.00	
2	0.25	0.50		1.00	
3	0.50	1.00		2.00	
4	0.75	0.50		3.00	
5	1.00	2.00		4.00	
:	:	:		:	

Fonte: Autor

Através de um algoritmo¹ foi possível testar diferentes valores de Δ a fim de encontrar aquele que primeiro satisfizesse a eq. (B.2), e por meio deste algoritmo encontrou-se os valores de Δ e de T_s (tempo de amostragem) indicados da eq. (B.3).

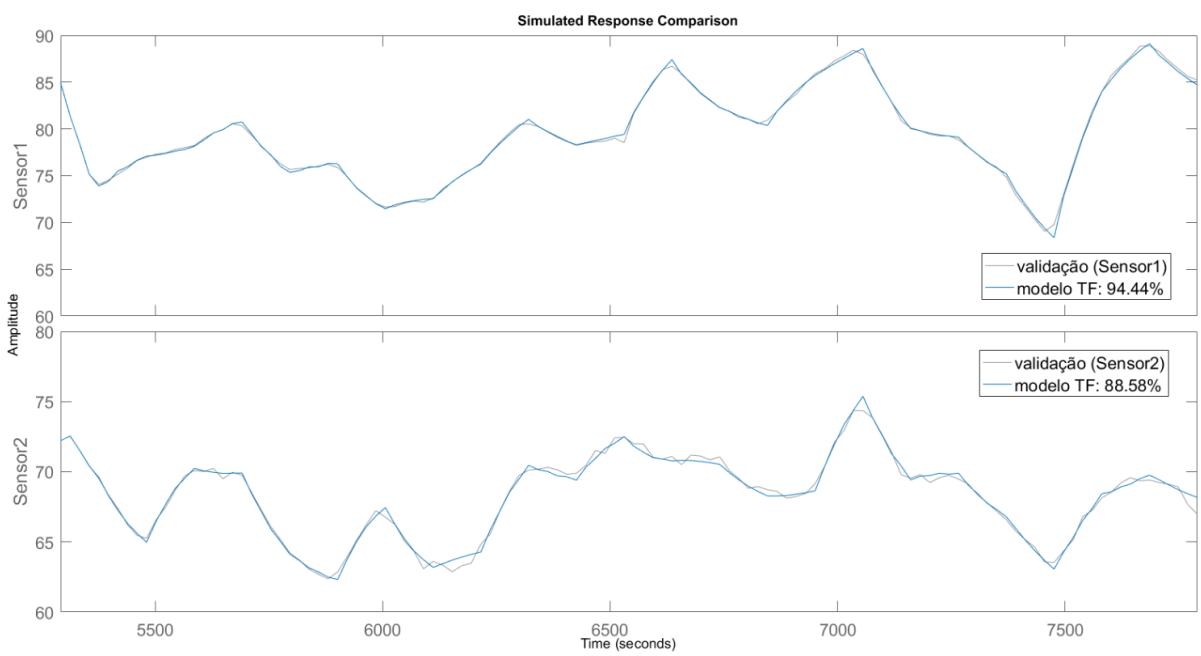
$$\Delta = 253 \quad (\text{B.3a})$$

$$T_s = 63.25s \quad (\text{B.3b})$$

¹ Scripts utilizados no apêndice A.7.

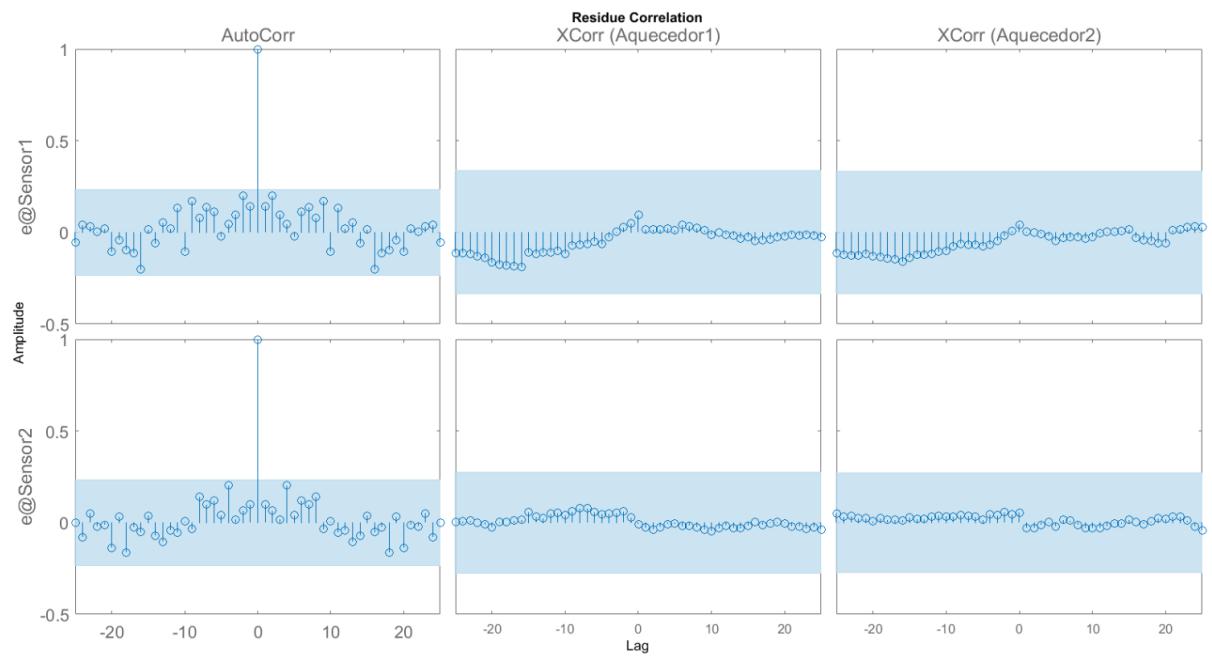
APÊNDICE C – Gráficos dos testes de validação dos modelos experimentais

Figura 41 – Modelo: função de transferência - Teste de validação



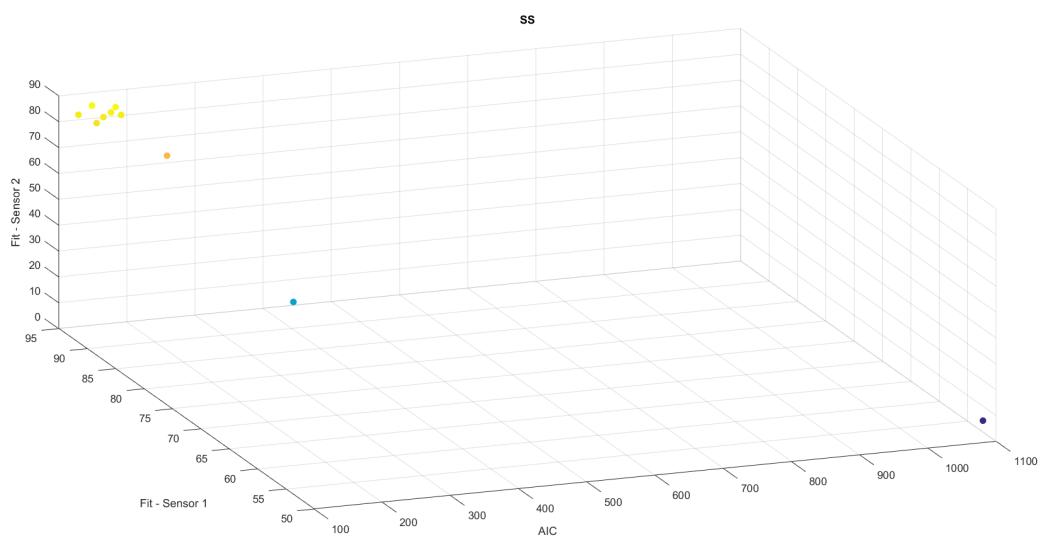
Fonte: Autor

Figura 42 – Modelo: função de transferência - Análise de resíduos



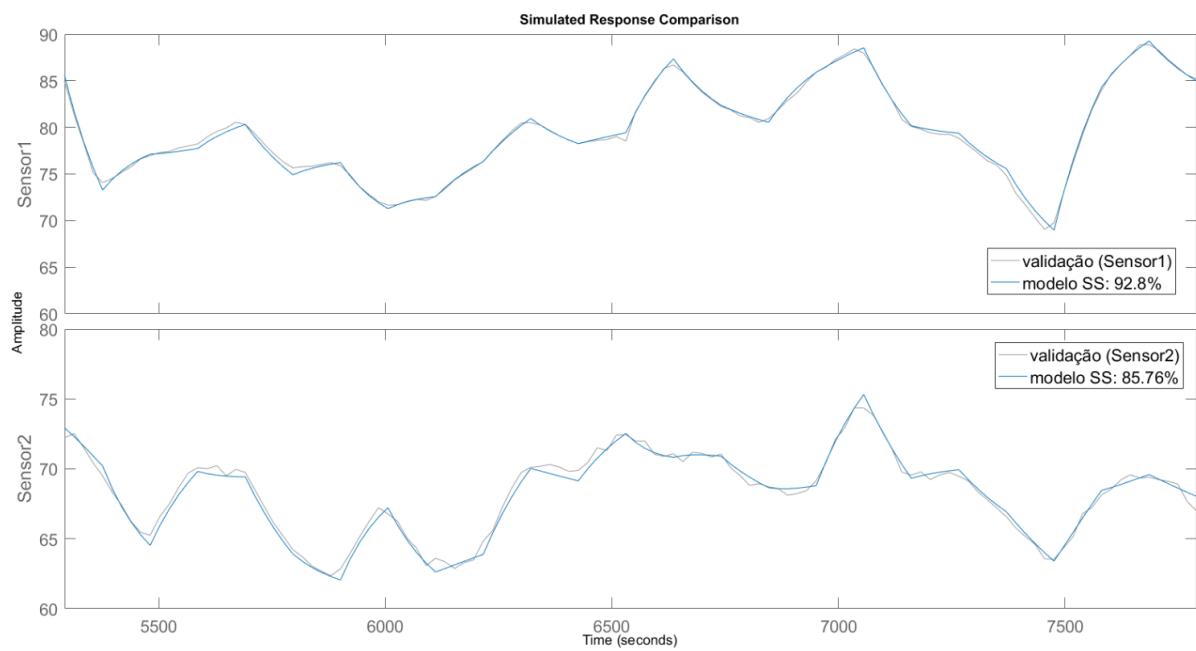
Fonte: Autor

Figura 43 – Modelos experimentais de espeço de estados



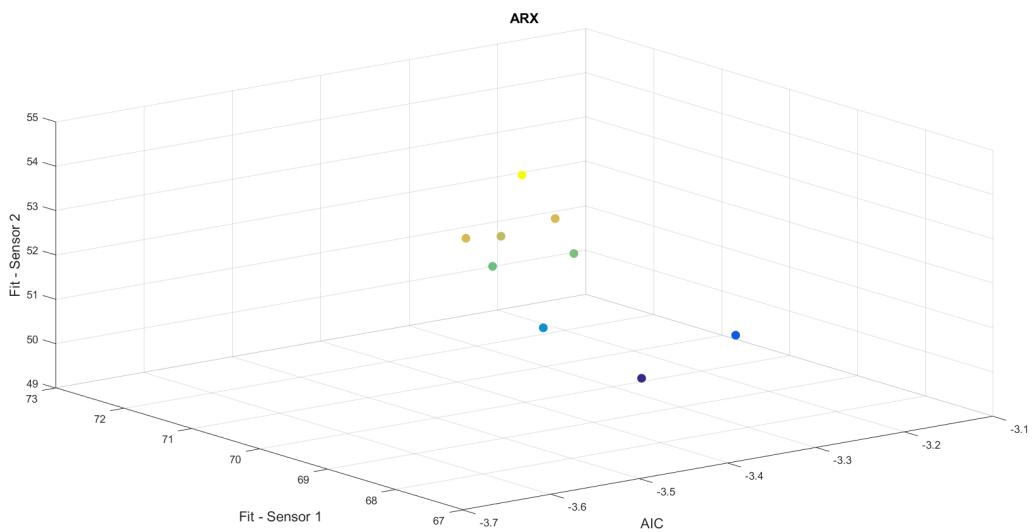
Fonte: Autor

Figura 44 – Modelo: espaço de estados - Teste de validação



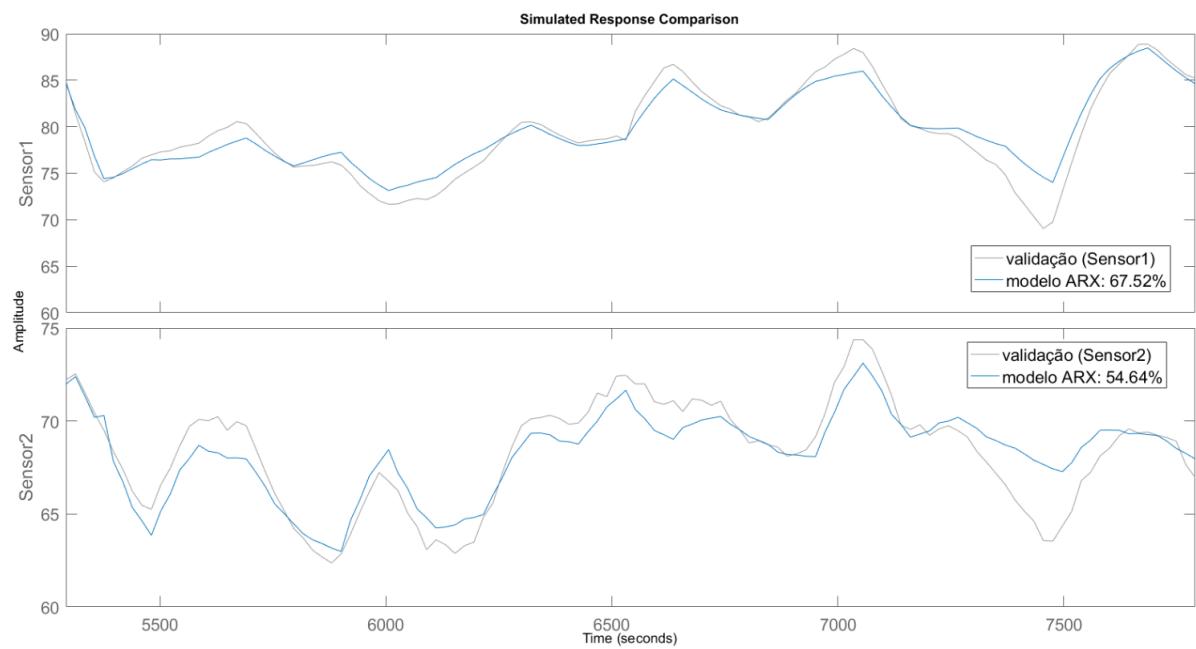
Fonte: Autor

Figura 45 – Modelos experimentais ARX



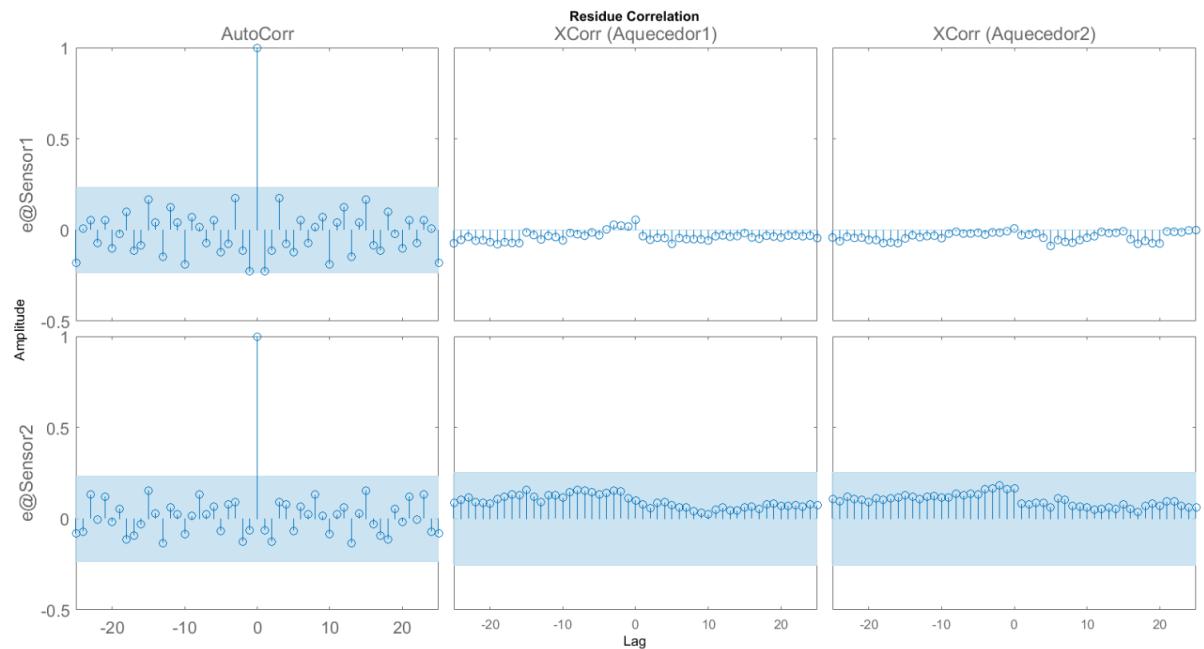
Fonte: Autor

Figura 46 – Modelo: ARX - Teste de validação



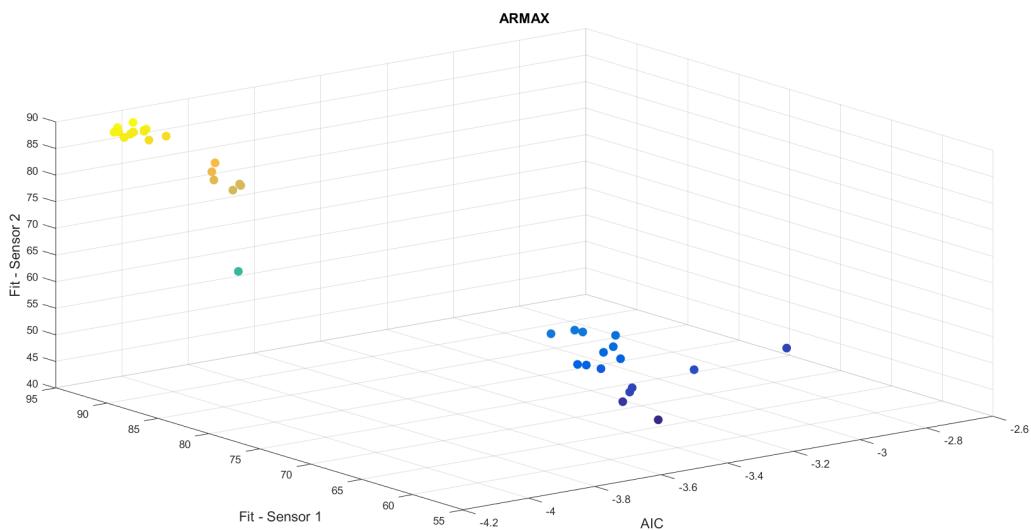
Fonte: Autor

Figura 47 – Modelo: ARX - Análise de resíduos



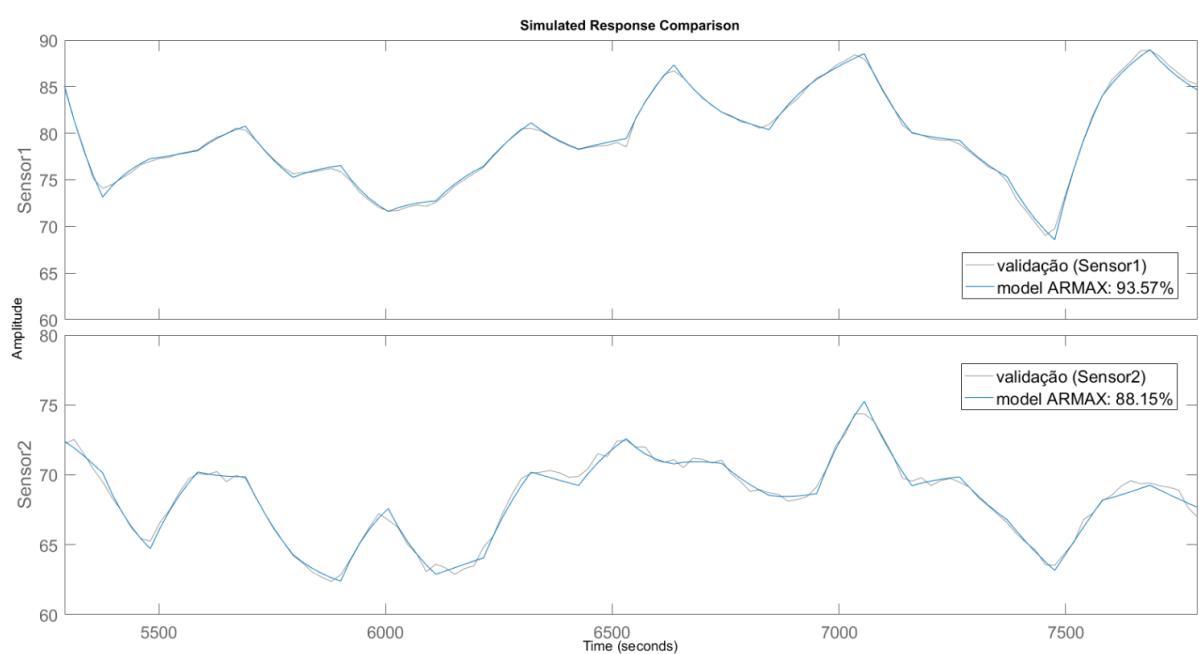
Fonte: Autor

Figura 48 – Modelos experimentais ARMAX



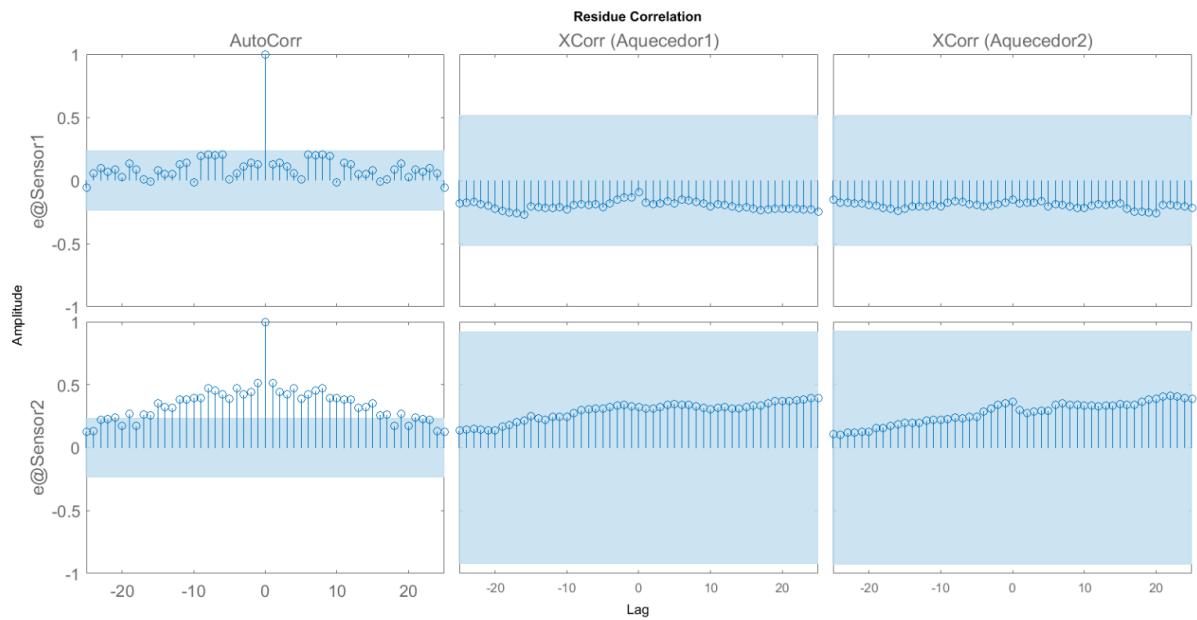
Fonte: Autor

Figura 49 – Modelo: ARMAX - Teste de validação



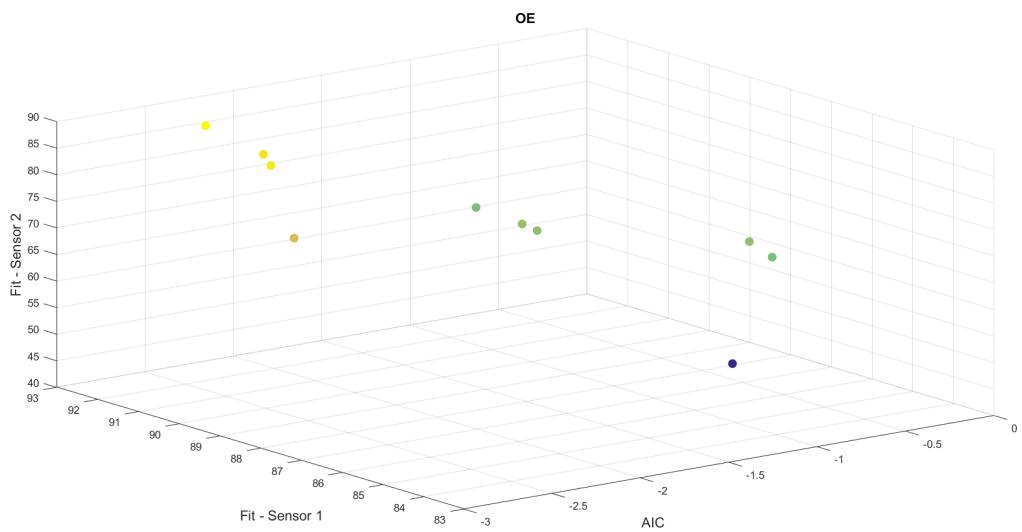
Fonte: Autor

Figura 50 – Modelo: ARMAX - Análise de resíduos



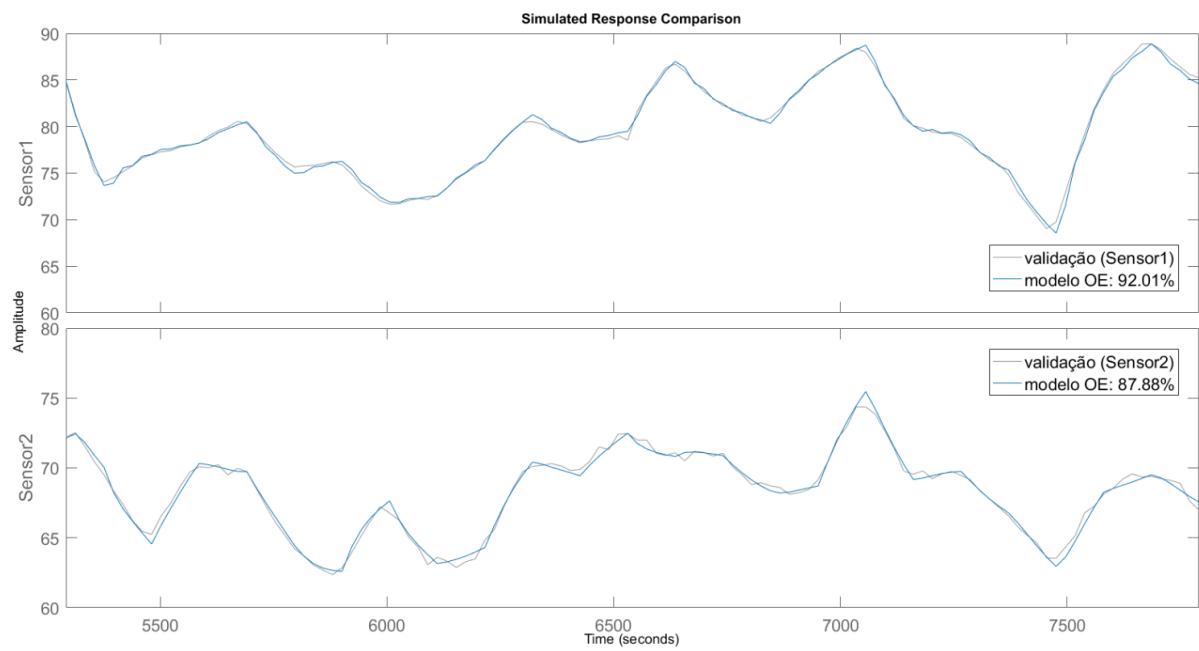
Fonte: Autor

Figura 51 – Modelos experimentais OE



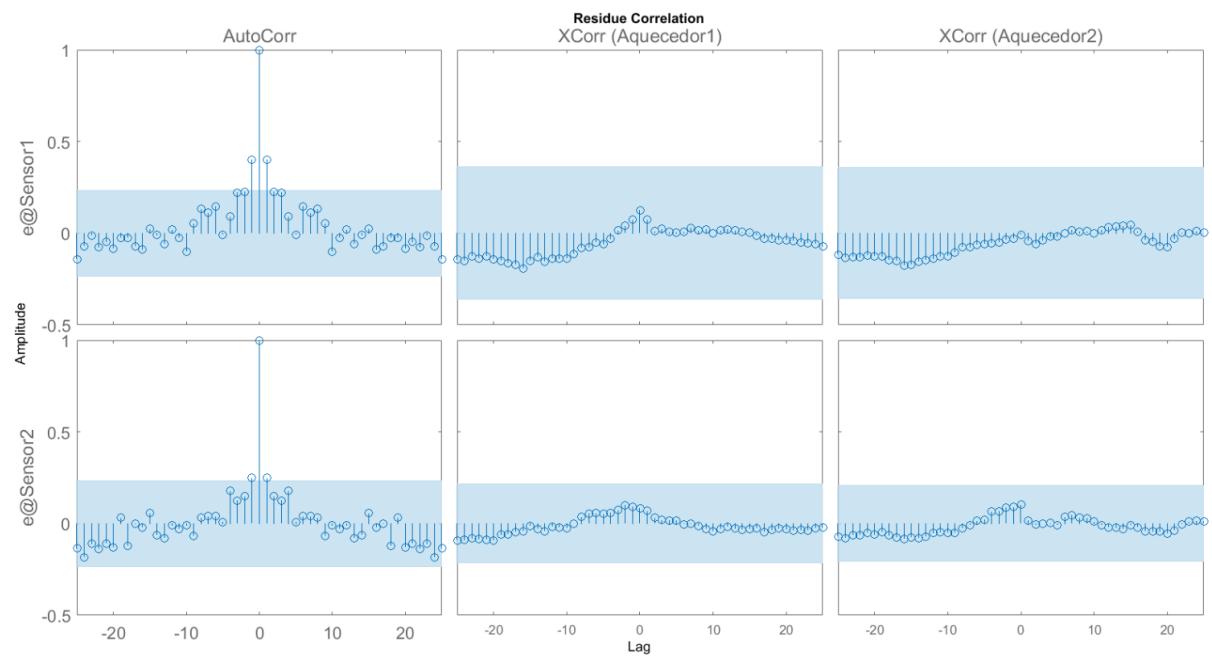
Fonte: Autor

Figura 52 – Modelo: OE - Teste de validação



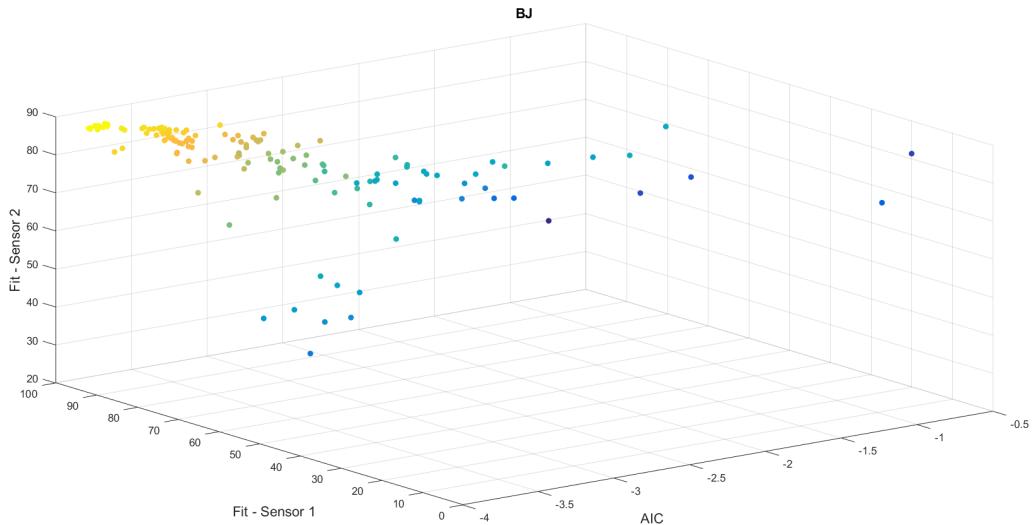
Fonte: Autor

Figura 53 – Modelo: OE - Análise de resíduos



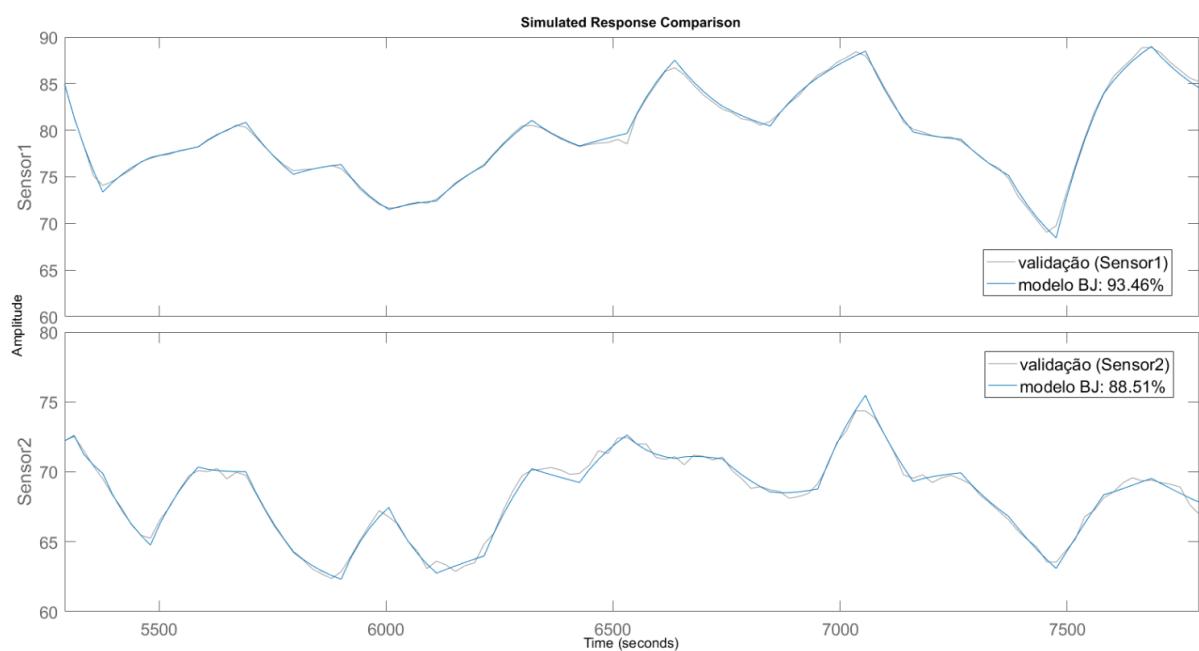
Fonte: Autor

Figura 54 – Modelos experimentais BJ



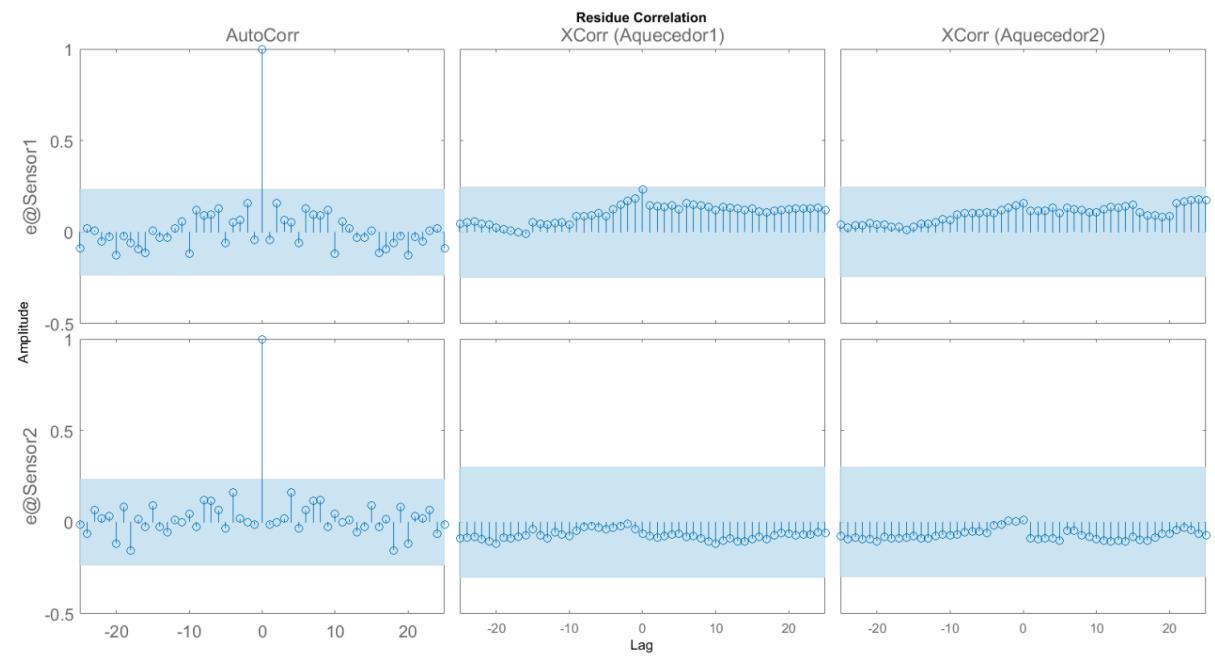
Fonte: Autor

Figura 55 – Modelo: BJ - Teste de validação



Fonte: Autor

Figura 56 – Modelo: BJ - Análise de resíduos

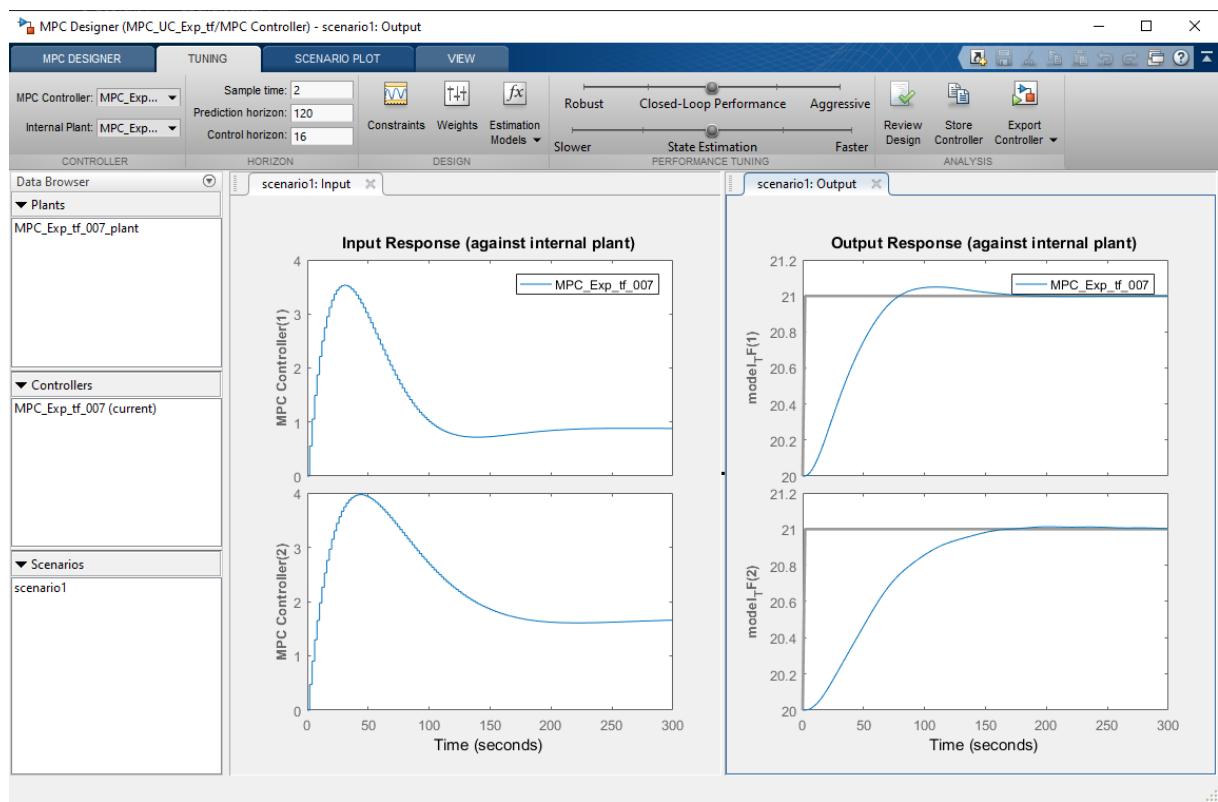


Fonte: Autor

APÊNDICE D – MPC Experimental Empírico

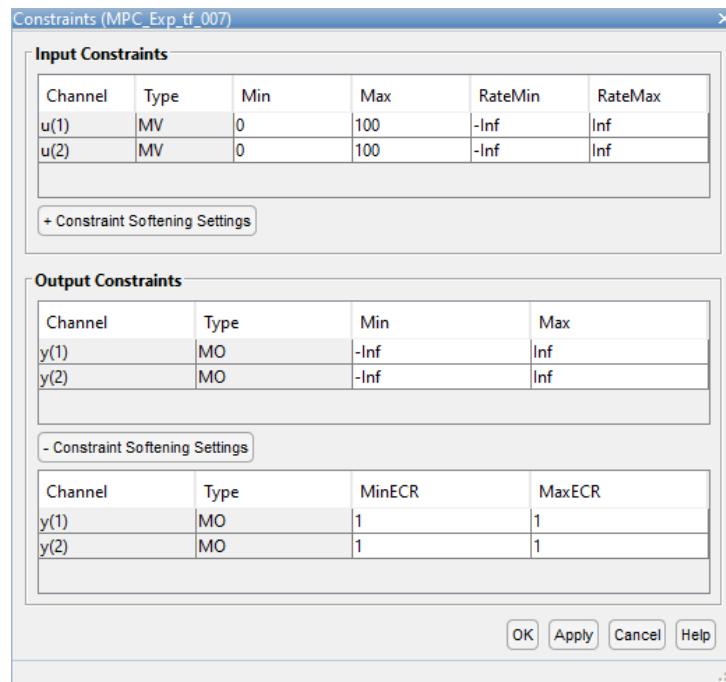
D.1 Telas do *MPC Design*

Figura 57 – MPC Design - Tela principal



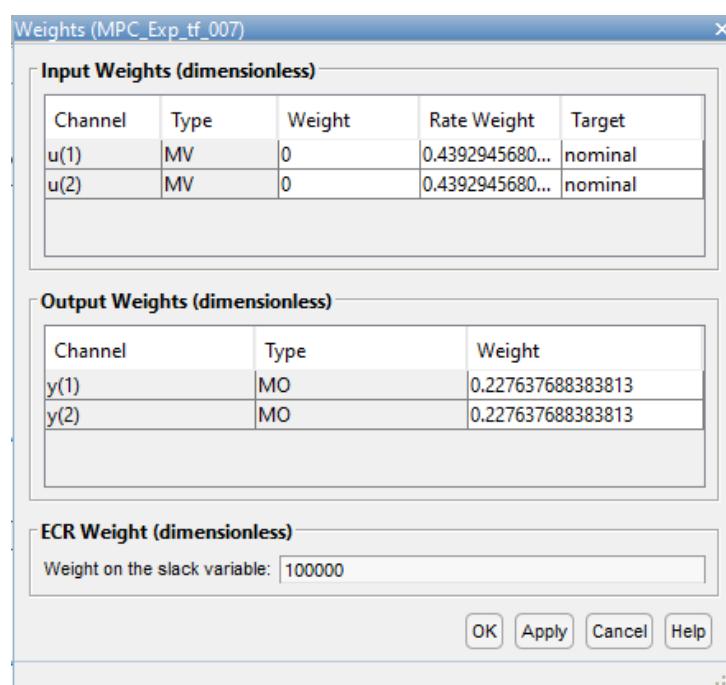
Fonte: Autor

Figura 58 – MPC Design - Limitantes



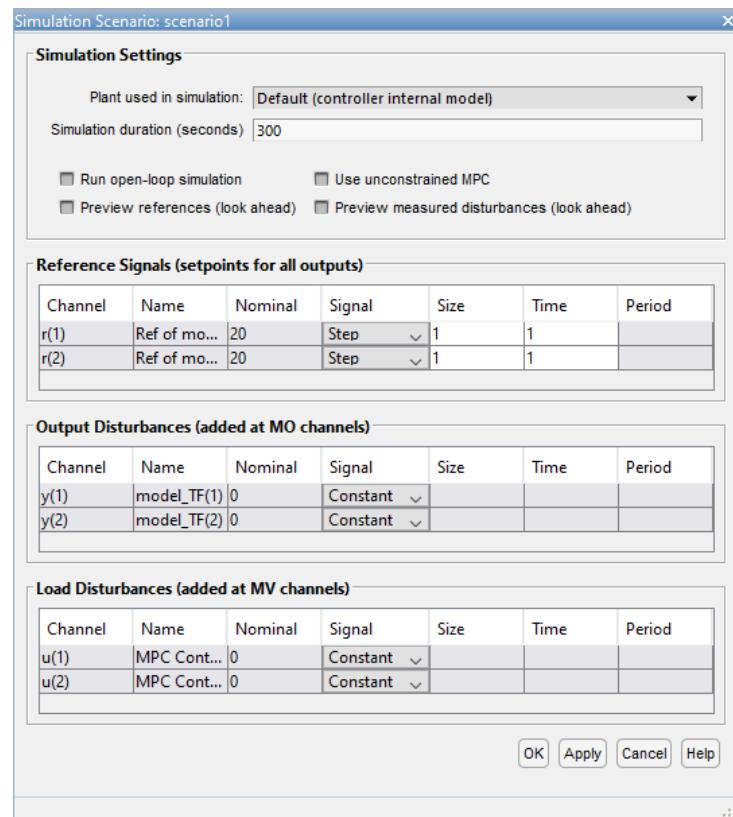
Fonte: Autor

Figura 59 – MPC Design - Pesos



Fonte: Autor

Figura 60 – MPC Design - Cenário



Fonte: Autor

D.2 Tabelas de performance

Tabela 25 – Performance dos controladores MPC e PID - IAE

Modelo utilizado no controlador	Sensor 1 (x10 ³)	Sensor 2 (x10 ³)	Média (x10 ³)
MPC Teórico SS (eq. (4.3))	38.07	27.94	33.00
MPC Exp. TF (Empírico)	41.17	33.90	37.53
MPC Experimental OE (tabela 14)	45.53	42.16	43.85
MPC Experimental ARMAX (tabela 12)	58.16	37.27	47.72
MPC Experimental TF (tabela 8)	56.90	55.08	55.99
PID (tabela 18)	53.90	59.10	56.50
MPC Experimental ARX (tabela 10)	59.13	63.35	61.24
MPC Experimental SS (eq. (5.5))	75.99	58.12	67.06

Fonte: Autor

Tabela 26 – Performance dos controladores MPC e PID - ISE

Modelo utilizado no controlador	Sensor 1 (x10 ⁴)	Sensor 2 (x10 ⁴)	Média (x10 ⁴)
MPC Teórico SS (eq. (4.3))	62.95	29.03	45.99
MPC Exp. TF (Empírico)	61.43	32.56	47.00
MPC Experimental OE (tabela 14)	66.46	32.78	49.62
MPC Experimental ARMAX (tabela 12)	77.46	30.68	54.07
MPC Experimental TF (tabela 8)	76.61	39.75	58.18
MPC Experimental ARX (tabela 10)	67.98	49.01	58.50
MPC Experimental SS (eq. (5.5))	81.04	40.95	61.00
PID (tabela 18)	76.83	50.06	63.45

Fonte: Autor

Tabela 27 – Performance dos controladores MPC e PID - ITAE

Modelo utilizado no controlador	Sensor 1 (x10 ⁶)	Sensor 2 (x10 ⁶)	Média (x10 ⁶)
MPC Teórico SS (eq. (4.3))	60.99	38.11	49.55
MPC Exp. TF (Empírico)	63.90	46.74	55.32
MPC Experimental OE (tabela 14)	73.78	61.69	67.74
MPC Experimental ARMAX (tabela 12)	91.79	57.31	74.55
PID (tabela 18)	84.97	88.90	86.94
MPC Experimental TF (tabela 8)	90.39	84.58	87.49
MPC Experimental ARX (tabela 10)	105.79	113.15	109.47
MPC Experimental SS (eq. (5.5))	136.28	97.80	117.04

Fonte: Autor

Tabela 28 – Performance dos controladores MPC e PID - ITSE

Modelo utilizado no controlador	Sensor 1 (x10 ⁷)	Sensor 2 (x10 ⁷)	Média (x10 ⁷)
MPC Teórico SS (eq. (4.3))	74.01	26.80	50.41
MPC Exp. TF (Empírico)	75.15	32.08	53.61
MPC Experimental OE (tabela 14)	83.09	33.31	58.20
MPC Experimental ARMAX (tabela 12)	96.70	30.77	63.74
MPC Experimental TF (tabela 8)	96.51	45.02	70.76
PID (tabela 18)	92.25	55.69	73.97
MPC Experimental SS (eq. (5.5))	114.29	50.85	82.57
MPC Experimental ARX (tabela 10)	96.37	68.92	82.65

Fonte: Autor

D.3 Sintonia

Visando sintonizar empiricamente o controlador em questão, os seguintes passos foram seguidos:

- Ajuste do cenário no *MPC Design* para 300s, de modo a observar o sistema até seu regime estacionário;
- Ajuste dos limitantes (*constraints*) para que os aquecedores não recebessem sinal maior que o permitido;
- Diminuição do tempo de amostragem para aumentar a velocidade entre iterações;
- Aumento gradativo do Horizonte de Predição até que nenhuma alteração perceptível fosse observada nos gráficos de resposta;
- Aumento gradativo do Horizonte de Controle até que nenhuma alteração perceptível fosse observada nos gráficos de resposta;
- Ajustar os seletores de "Robustez/Agressividade" e "Velocidade de estimativa de estados" utilizando *Backtracking* até obter uma performance satisfatória.