ARTHUR HENRIQUE IASBECK

CONTROLE ADAPTATIVO EMPREGADO NO RASTREAMENTO DE TRAJETÓRIA DE UM ROBÔ AUTÔNOMO

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE ENGENHARIA MECÂNICA
2019

ARTHUR HENRIQUE IASBECK

CONTROLE ADAPTATIVO EMPREGADO NO RASTREAMENTO DE TRAJETÓRIA DE UM ROBÔ AUTÔNOMO

Projeto de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia Mecatrônica da Universidade Federal de Uberlândia, como parte dos requisitos para a obtenção do título de **BACHAREL** em **ENGENHARIA MECATRÔNICA**.

Área de concentração: Controle de Sistemas.

Orientador: Prof. Dr. Rogério Sales Gonçalves

Uberlândia - MG

2019

"Prefiro questões que não podem ser respondidas, a respostas que não podem ser questionadas"

Richard Feynman

AGRADECIMENTOS

Agradeço primeiramente a Deus, que me proveu de todas as ferramentas necessárias para que eu alcançasse meus objetivos. À minha mãe, Renata, que sempre colocou os filhos acima de si mesma e sempre me incentivou a seguir em frente e superar meus medos. Ela que mesmo em situações difíceis, fez de tudo para que pudéssemos alcançar nosso sucesso e que sempre foi, acima de tudo, uma amiga. Ao meu pai, Marco Aurélio, que não está mais conosco mas que me mostrou como um professor de verdade deve ser e me inspirou a seguir o caminho da docência, que tanto dignifica e enobrece. A todos os meus familiares, que me apoiaram quando eu precisei e a todos os amigos que me ampararam nessa jornada. À minha namorada, Ana Paula, que fez luz nos meus dias e que sempre me motivou e me inspirou. Me mostrou como podemos ser perseverantes e como podemos mudar o mundo de cada um que está a nossa volta.

Ao meu grande amigo, João Vitor, que compartilhou comigo todas as dificuldades que a graduação pode oferecer e que me ajudou a passar por tudo isso da melhor forma possível. Àqueles professores que encontrei ao longo da minha formação que me inspiraram e me motivaram a ser o que eu sou hoje e também àqueles que fizeram de tudo pra tornar a minha mais difícil, porque graças a estes últimos, me tornei mais resiliente. Ao meu amigo Murilo que foi o que mais me deu amparo durante a execução deste trabalho, sempre disposto a ajudar em tudo que fosse necessário.

À UFU (Universidade Federal de Uberlândia) e à FEMEC (Faculdade de Engenharia Mecânica), porque sem elas eu jamais seria o que sou hoje tanto do ponto de vista técnico quanto do pessoal. E por fim, à EDROM (Equipe de Desenvolvimento em Robótica Móvel) onde conheci muitos amigos, vivi alguns dos meus melhores momentos, cresci muito como pessoa e aprendi bastante, e ao tutor da equipe, Rogério, que me orientou ao longo da execução deste trabalho (porque sem o apoio dele, nada disso teria sido possível).

IASBECK, A. H., **Controle Adaptativo Empregado no Rastreamento de Trajetória de um Robô Autônomo.** 2019. Projeto de Fim de Curso, Universidade Federal de Uberlândia, Uberlândia.

RESUMO

Este trabalho tem como objetivo o controle de trajetória de um robô autônomo através da utilização de um controlador adaptativo do tipo STR (*Self Tuning Regulator*). O STR adotado foi projetado a partir do Método do Posicionamento dos Polos (*Pole Placement*) enquanto a estimação *online* dos parâmetros do processo foi realizada com base no Método dos Mínimos Quadrados (*Least squares*). O comportamento da malha de controle foi avaliado por meio de simulações e experimentos. Os resultados mostraram que a trajetória predeterminada foi rastreada. Mais ainda, os requisitos de *overshoot* e tempo de acomodação estipulados foram atendidos.

Palavras Chave: Self Tuning Regulator, Controle Adaptativo, Controle de trajetória, Posicionamento de Polos, Método dos Mínimos Quadrados

IASBECK, A. H., **Adaptative Control Applied to Trajectory Tracking of an Autonomos Robot.** 2019. Graduation Completion Project, Universidade Federal de Uberlândia, Uberlândia.

ABSTRACT

This work aims to control the trajectory of an autonomous robot through the use of an STR (Self Tuning Regulator) adaptive controller. The adopted STR was designed from the Pole Placement Method while the online estimation of the process parameters was performed with the Least Squares Method. The behavior of the control system was evaluated through simulations and experiments. The results showed that the predetermined trajectory was traced. Furthermore, the stipulated overshoot and settling time requirements have been satisfied.

Keywords: Self Tuning Regulator, Adaptative Control, Trajectory control, Pole Placement, Least Squares Method.

Lista de Figuras

Lista de Símbolos

y	-	Saída do sistema a ser controlado
u	-	Entrada do sistema controlado (ação de controle)
V	-	Função de custo dos Mínimos Quadrados
y_{est}	-	Saída do sistema estimada pelo Método dos Mínimos Quadrados
ε	-	Erro de estimação
θ	-	Matriz que contém os parâmetros de um dado modelo dinâmico
q	-	Operador de deslocamento no tempo
$Y(z)$	-	Transformada z da saída do sistema a ser controlado
$U(z)$	-	Transformada z da entrada do sistema controlado
$G(z)$	-	Função de transferência do sistema controlado
λ	-	Fator de esquecimento
$degX$	-	Grau do polinômio X
u_c	-	Referência (*setpoint*) do sistema de controle
y_m	-	Saída desejada para o sistema a ser controlado
ξ	-	Fator de amortecimento
w_n	-	Frequência natural
M_p	-	Máximo sobressinal (*overshoot*)
t_a	-	Tempo de acomodação
T	-	Tempo de amostragem
e_{avg}	-	Erro médio entre a saída do sistema $y(t)$ e a saída desejada $y_m(t)$

Lista de Abreviações e Siglas

STR	-	Self Tuning Regulator
CLP	-	Controlador Lógico Programável
SISO	-	Single Input Single Output
MISO	-	Multiple Input Single Output
USB	-	Universal Serial Bus
RAM	-	Random-access memory
AG	-	Algoritmo Genético
NASA	-	National Aeronautics and Space Administration
EDROM	-	Equipe de Desenvolvimento em Robótica Móvel
FEMEC	-	Faculdade de Engenharia Mecânica
UFU	-	Universidade Federal de Uberlândia
PID	-	Proporcional Integral Derivativo
LARC	-	Latin American and Brazilian Robotics Competition
IEEE	-	Institute of Electrical and Electronics Engineers
SEK	-	Standard Educational Kit
PP	-	Pole Placement
RLS	-	Recursive Least Square

SUMÁRIO

CAPÍTULO I

INTRODUÇÃO

1.1 Objetivo

Para que um robô móvel execute tarefas de forma autônoma, precisa se locomover em um ambiente que pode, ou não, ser previamente conhecido, onde podem haver, ou não, referências que permitam a orientação. Porém, independente do local onde o robô irá atuar, é necessário que tenha conhecimento de onde se encontra e para onde está se dirigindo. Desta forma poderá saber se está no caminho certo, se está se comportando como deveria, e se está executando suas tarefas da melhor forma possível. É por isso que o projeto de um robô móvel depende diretamente do controle de sua trajetória e de sua orientação.

Cabe ressaltar que quando um robô autônomo se move por um ambiente desconhecido que muda constantemente, deve estar apto a lidar com obstáculos e adversidades que venham a surgir em seu caminho. Além disso, as suas limitações físicas, a qualidade de seus atuadores, sensores, e materiais que compõe sua estrutura, devem também ser levadas em consideração no projeto. Então, é preciso garantir que o controle de trajetória seja robusto o suficiente para lidar com tais adversidades.

Para realizar o controle de trajetória e orientação de um dado sistema, uma das possíveis soluções é utilizar uma bússola, como foi proposto neste trabalho. Por exemplo, imagine que o piloto automático de um avião precise mantê-lo em uma determinada rota durante um dado tempo, mesmo que sob ação de perturbações, como rajadas de vento e turbulências. Uma possível

solução seria utilizar uma bússola, que fosse capaz de detectar desvios a partir da rota estipulada, e implementar um controlador para compensar o efeito das perturbações, de forma que o avião fosse mantido apontado sempre na direção correta. Este trabalho adota uma ideia semelhante aplicada ao controle de trajetória de um robô de duas rodas.

Este trabalho tem como objetivo principal a realização do controle de trajetória de um robô autônomo através da implementação de um controlador que tenha seus parâmetros ajustados de forma automática.

1.2 Justificativa

A Equipe de Desenvolvimento em Robótica Móvel (EDROM), que forneceu todo o material necessário ao desenvolvimento deste trabalho, é uma associação de discentes dos cursos de graduação da Faculdade de Engenharia Mecânica (FEMEC) da Universidade Federal de Uberlândia (UFU). Esta equipe participa de competições de robótica, como por exemplo a LARC (Latin American and Brazilian Robotics Competition), que exigem que robôs autônomos sejam desenvolvidos para executarem as mais diversas tarefas. Uma das categorias da LARC, a IEEE SEK (Standard Educational Kit), propõe que os competidores resolvam uma série de problemas utilizando robôs autônomos construídos a partir dos kits LEGO Mindstorms®. Dentre os desafios já propostos estão a manutenção de plataformas de petróleo, o resgate de indivíduos em ambientes de difícil acesso, o desenvolvimento de táxis autônomos, dentre outros. Cabe ressaltar que os problemas propostos estão em escala consideravelmente reduzida, o que não significa que deixam de ser bastante complexos.

Praticamente todos os desafios já propostos na categoria IEEE SEK requereram a implementação do controle de trajetória ou orientação. Este por sua vez pode ser realizado a partir de referências no ambiente, como linhas e paredes, ou a partir da utilização de sensores inerciais como a bússola, por exemplo. Para que o robô se mova em linha reta, é possível implementar um controlador que avalie o valor da bússola e atue na potência dos motores para fazer que este valor alcance uma determinada referência *setpoint*.

Os robôs que a EDROM desenvolve com o objetivo de participar da LARC na categoria IEEE SEK utilizam controladores PID no controle de trajetória e orientação. No entanto, o ajuste

dos parâmetros deste tipo de controlador (k_p, k_i e k_d) é realizado de forma manual, através de uma série de testes. Isso geralmente requer bastante tempo e nem sempre gera resultados satisfatórios. Ainda é preciso levar em consideração que, caso o robô sofra alguma mudança, o que é algo bastante comum, os parâmetros precisarão ser reajustados. Como resultado a implementação de um controlador no qual os parâmetros são ajustados de forma automática seria algo bastante vantajoso, por resultar em uma enorme economia de tempo.

Optou-se pela implementação de um *Self Tuning Regulator* (STR) para determinação dos parâmetros do controlador porque sistemas de controle deste tipo automatizam tanto a modelagem do processo controlado quanto a sintonização do controlador. Além disso, a partir da utilização do STR foram obtidos resultados experimentais satisfatórios, que serão apresentados ao fim deste trabalho.

1.3 Organização do trabalho

Este trabalho foi dividido em quatro partes. Na Seção 2 é realizada uma revisão bibliográfica acerca dos métodos utilizados na implementação do controlador adaptativo proposto. Na Seção 3 são apresentadas mais informações sobre o sistema a ser controlado e sobre as simulações realizadas. Na Seção 4 os resultados obtidos a partir da implementação do controle de trajetória são mostrados. Por fim, na Seção 5 são realizadas as conclusões do trabalho e algumas propostas para trabalhos futuros.

CAPÍTULO II

REVISÃO BIBLIOGRÁFICA

2.1 Controle Adaptativo

O Controle Adaptativo é uma subárea do Controle de Sistemas no qual são estudados controladores que sejam capazes de lidar com processos desconhecidos ou que se alterem ao longo do tempo. Como o próprio nome sugere, estes controladores adaptam seu comportamento frente alterações tanto na dinâmica do processo controlado quanto nas características das perturbações que atuam sobre o mesmo (Astrom; Wittenmark, 1995).

Um controlador pode ser classificado como adaptativo desde que possua parâmetros ajustáveis e algum mecanismo que possibilite o ajuste destes parâmetros (Astrom; Wittenmark, 1995). Cabe ressaltar que as alterações que o controlador sofre ao longo do tempo ocorrem para que o processo controlado se comporte como desejado. Um diagrama de blocos que exemplifica a implementação de um controlador adaptativo é mostrado na Fig. 2.1.

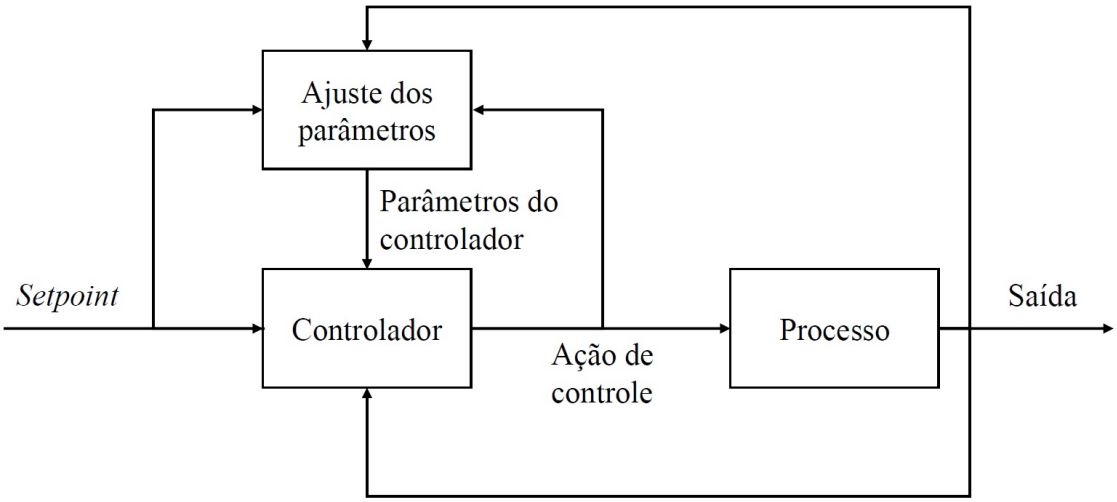

Figura 2.1 – Diagrama de blocos que exemplifica a implementação de um controlador adaptativo.

O Controle Adaptativo foi desenvolvido para lidar com processos que não podiam ser controlados por meio da implementação de controladores de ganhos constantes. As primeiras aplicações nesta área surgiram na década de 50, quando pesquisadores associados à NASA® iniciaram o desenvolvimento de *autopilots* (pilotos automáticos) para aviões de alta *performance* como, por exemplo, o X-15, mostrado na Fig. 2.2. Os pesquisadores perceberam que um controlador de ganhos constantes era suficiente para garantir que a aeronave se comportasse como desejado, porém somente quando as condições de voo (no caso a velocidade da aeronave e sua altitude) permanecessem constantes com o passar do tempo (Astrom; Wittenmark, 1995). Para que fosse possível controlar o avião mesmo em condições de voo dinâmicas, foram propostos os primeiros controladores adaptativos.

Figura 2.2 – Aeronave de alta *performance* X-15. (Fonte: `https://www.nasa.gov/centers/armstrong/multimedia/imagegallery/X-15`)

Para entender melhor a importância desta técnica de controle, considere um processo arbitrário descrito pela equação de diferenças apresentada em (2.1). A entrada e a saída do sistema, no instante de tempo t, são representada respectivamente por $u(t)$ e $y(t)$.

$$y(t) = 1,0002y(t-1) + 0,0008y(t-2) + \delta u(t-1) + 0,1503u(t-2) \tag{2.1}$$

Suponha inicialmente que o valor de δ seja igual a $-0,0244$ e que posteriormente passe a ser igual 1. Imagine que um controlador de ganhos constante tivesse sido proposto antes que o processo se modificasse. Neste caso, após a mudança do valor de δ, o sistema não mais se comportaria da maneira desejada. A Fig. 2.3 mostra a resposta do processo controlado antes e depois da alteração em δ. Neste caso uma onda quadrada foi utilizada como referência.

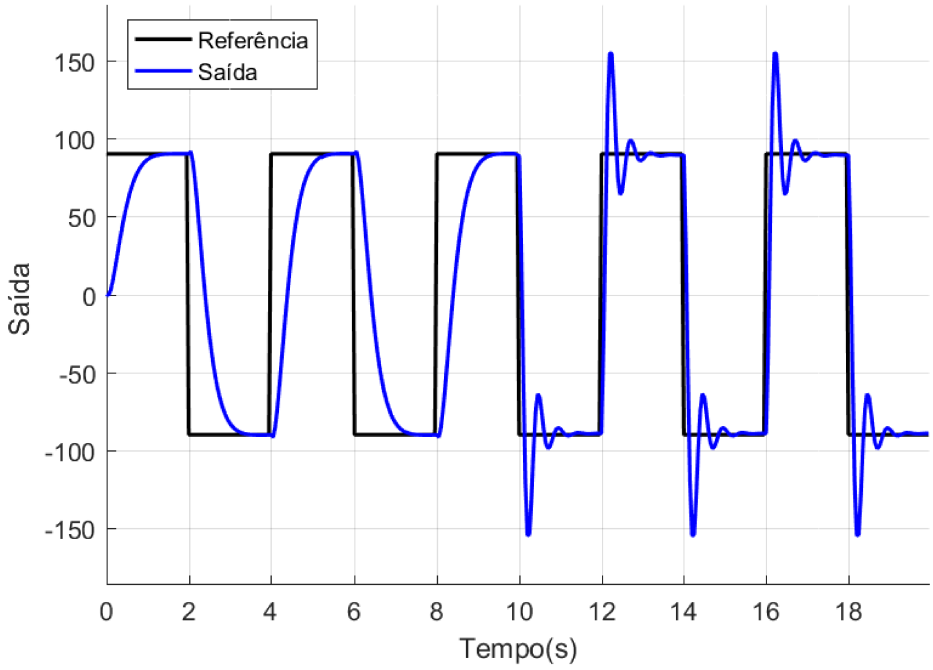

Figura 2.3 – Resposta de um processo com parâmetros variantes no tempo com um controlador de ganhos constantes.

Caso um controlador adaptativo fosse implementado as mudanças no processo resultariam em alterações no controlador para garantir que a resposta do sistema continuasse a mesma (Fig. 2.4). De fato, analisando agora a ação de controle obtida, mostrada na Fig. 2.5, é possível perceber que a mesma se adéqua às alterações sofridas pela planta de forma que a saída siga inalterada.

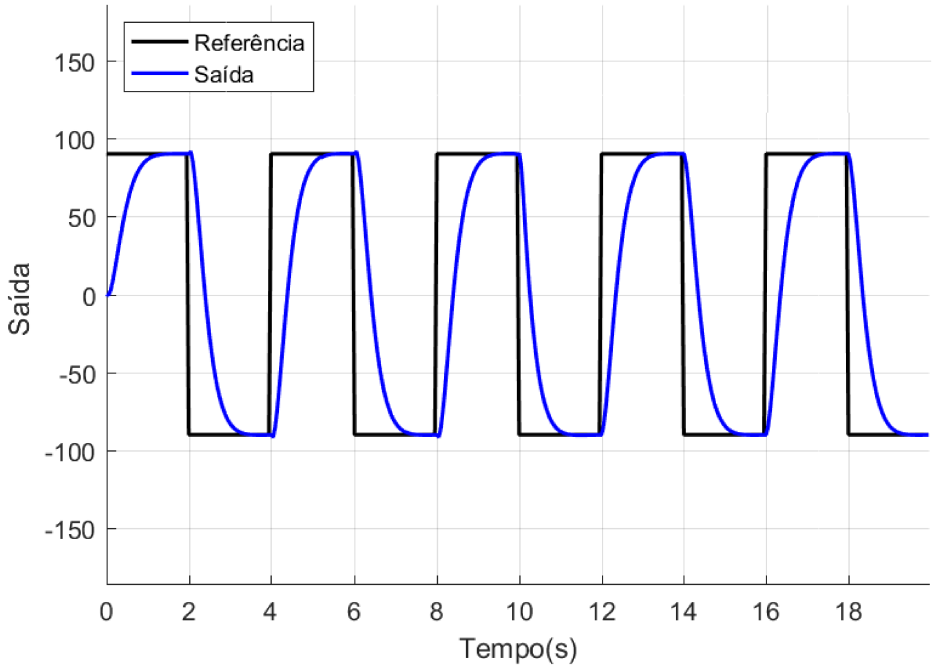

Figura 2.4 – Resposta de um processo com parâmetros variantes no tempo e um controlador adaptativo.

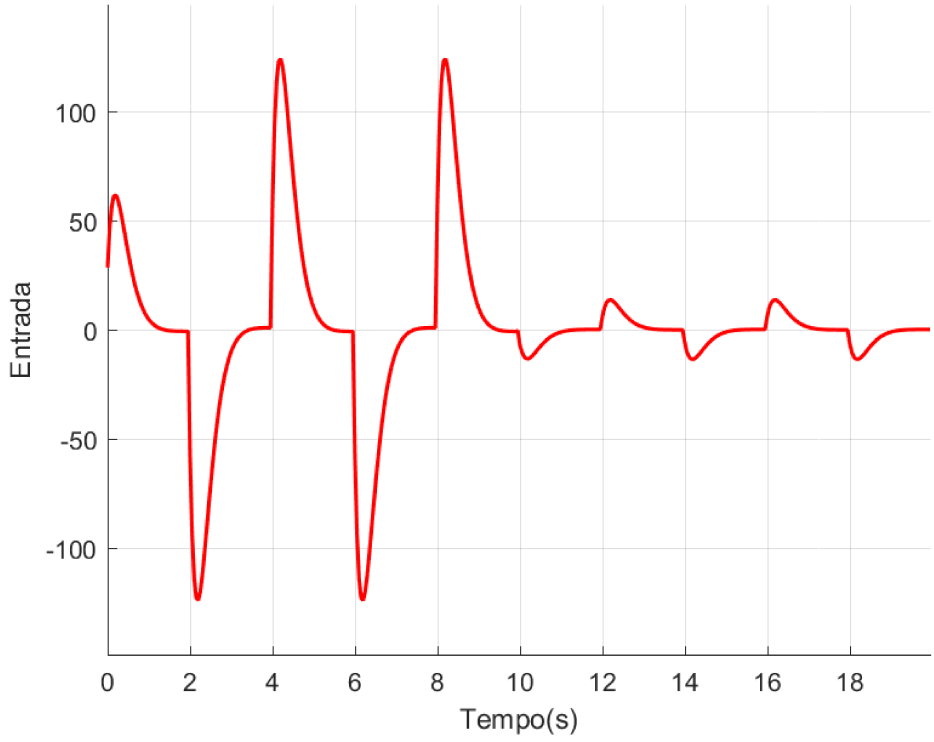

Figura 2.5 – Evolução da ação de controle (entrada) de um processo com parâmetros variantes no tempo e um controlador adaptativo.

É possível perceber que o comportamento do processo mudou de forma considerável quando foi alterado o valor de δ. Fica mais claro através deste exemplo que um controlador de ganhos constantes não é capaz de lidar com variações na dinâmica do processo controlado. A implementação deste tipo de controlador em sistemas não lineares ou em sistemas que apresentem em sua entrada ruídos desconhecidos, por exemplo, pode também apresentar resultados insatisfatórios (Astrom; Wittenmark, 1995).

Apesar dos controladores adaptativos terem sido inicialmente propostos para lidar com processos variantes no tempo e/ou que estejam sob ação de ruídos desconhecidos, diversos esquemas adaptativos podem ser também implementados na sintonização automática de controladores de ganhos constantes. Não é difícil encontrar controladores comerciais utilizados no setor industrial que implementem alguma forma de adaptação que possibilite sua sintonização automática (*auto tunning*)(Astrom; Wittenmark, 1995).

Cabe ressaltar que controladores adaptativos devem ser utilizados somente quando os resultados obtidos a partir da implementação de um controlador de ganhos constantes não forem satisfatórios. São exemplos processos variantes no tempo, não lineares, ou que apresentem em sua entrada ruídos desconhecidos. Controladores adaptativos são não lineares, complexos, de difícil computação e de difícil implementação (Astrom; Wittenmark, 1995).

De forma geral, é possível dividir os controladores adaptativos em dois grupos: diretos e indiretos.

Controladores diretos são aqueles que tem seus parâmetros obtidos diretamente, sem que a planta do processo controlado seja estimada. Como exemplo é possível citar os controladores baseados em modelos de referência, que têm seus ganhos determinados com base na comparação entre a saída do processo controlado e a saída de um modelo escolhido previamente. Outro exemplo são os controladores baseados em tabelamento de ganhos, onde são escolhidas variáveis que descrevem o comportamento do processo a ser controlado e os parâmetros do controlador são determinados a partir dos valores que estas variáveis assumem ao longo do tempo. Para realizar o controle da aeronave X-15, mostrado na Fig. 2.2, por exemplo, as variáveis escolhidas foram a altitude e a velocidade. Para cada combinação possível destas grandezas, foram atribuídos aos parâmetros do controlador os valores que garantissem que a aeronave se comportasse como desejado (Astrom; Wittenmark, 1995).

Já os controladores indiretos são aqueles em que os parâmetros do controlador dependem dos parâmetros da planta do processo a ser controlado. Neste caso, supondo que esta planta seja desconhecida ou tenha um comportamento dinâmico no tempo, é necessário estimá-la em tempo real (Astrom; Wittenmark, 1995). Um exemplo são os controladores STR, que são o foco deste trabalho e que serão detalhados nas seções subsequentes.

2.2 Estimação dos parâmetros da planta através do Método dos Mínimos Quadrados

2.2.1 Método dos Mínimos Quadrados

O **Método dos Mínimos Quadrados** foi utilizado na estimação dos parâmetros da planta. Este método propõe que o ajuste dos parâmetros de um dado modelo seja realizado de forma que este represente da melhor forma possível um conhecido conjunto de dados obtido experimentalmente. Ele foi desenvolvido por Karl Friedrich Gauss no final do século XVIII e afirma que os parâmetros de um modelo, inicialmente desconhecidos, devem ser determinados de forma que seja mínimo o quadrado da diferença entre os valores obtidos a partir deste modelo e aqueles obtidos experimentalmente (valores reais) (Astrom; Wittenmark, 1995).

O modelo nada mais é do que uma relação matemática, como em (2.2), que representa um certo conjunto de dados obtido experimentalmente e possibilita o calculo de uma aproximação $y_{est}(i)$ para a variável observada, a partir de valores conhecidos $\varphi(i)$ e parâmetros inicialmente desconhecidos θ. A determinação destes parâmetros deve ser realizada de forma que os valores estimados a partir do modelo $y_{est}(i)$ se aproximem o máximo possível dos valores reais $y(i)$.

$$y_{est}(i) = \varphi_1(i)\theta_1 + \varphi_2(i)\theta_2 + \varphi_3(i)\theta_3 + \cdots + \varphi_n(i)\theta_n \tag{2.2}$$

Também chamada de **modelo de regressão**, (2.2) pode ser escrita de forma matricial, como mostrado em (2.3).

$$y_{est}(i) = \varphi^T(i)\theta \tag{2.3}$$

em que

$$\varphi^T(i) = \begin{bmatrix} \varphi_1(i) & \varphi_2(i) & \varphi_3(i) & \cdots & \varphi_n(i) \end{bmatrix} \tag{2.4}$$

$$\theta = \begin{bmatrix} \theta_1 & \theta_2 & \theta_3 & \cdots & \theta_n \end{bmatrix}^T \tag{2.5}$$

Para avaliar quão próximos os valores estimados se encontram dos valores reais, a função de custo dos mínimos (2.6) quadrados pode ser utilizada. O método propõe que sejam

escolhidos valores para θ que minimizem $V(\theta, t)$ (Astrom; Wittenmark, 1995).

$$V(\theta, t) = \frac{1}{2} \sum_{i=1}^{t} \left(y(i) - \varphi^T(i)\theta \right)^2 \tag{2.6}$$

Considerando que $y_{est}(i)$ seja linear nos parâmetros θ, é possível encontrar uma solução analítica que permita a determinação dos parâmetros θ.

Pode-se demonstrar que, para minimizar o valor de $V(\theta, t)$, os parâmetros θ devem ser calculados por (Astrom; Wittenmark, 1995).

$$\theta(t) = \left(\Phi^T(t)\Phi(t) \right)^{-1} \Phi^T(t) Y(t) \tag{2.7}$$

Para entender melhor como o Método dos Mínimos Quadrados funciona, considere um conjunto de dados gerado por (2.8) (Astrom; Wittenmark, 1995). Para cada valor de $u(i)$, é obtido um valor de $y(i)$.

$$y(i) = 1 + 0,45u(i) + 0,1u(i)^2 + e(t) \tag{2.8}$$

Neste caso, $e(t)$ é um valor aleatório que varia entre 0 e $0,1$ e representa um ruído de medida.

Considere que (2.8) seja utilizada para construir um conjunto de dez dados e sobre este seja aplicado o Método dos Mínimos Quadrados, propostos quatro modelos, (a), (b), (c) e (d), descritos pelas equações a seguir.

(a) $y_{est}(i) = b_0$

(b) $y_{est}(i) = b_0 + b_1 u(i)$

(c) $y_{est}(i) = b_0 + b_1 u(i) + b_2 u^2(i)$

(d) $y_{est}(i) = b_0 + b_1 u(i) + b_2 u^2(i) + b_3 u^3(i)$

Cada um destes modelos pode ser representado em sua forma matricial, como em

(2.3). Por exemplo, para o modelo (c), tem-se

$$\varphi^T(i) = \begin{bmatrix} 1 & u & u^2 \end{bmatrix} \tag{2.9}$$

$$\theta = \begin{bmatrix} b_0 & b_1 & b_2 \end{bmatrix}^T \tag{2.10}$$

$$y_{est}(i) = \varphi^T(i)\theta \tag{2.11}$$

O resultado da aplicação do Método dos Mínimos Quadrados, considerando cada um dos modelos propostos, ao conjunto de dados gerado por (2.8), é mostrado na Fig. 2.6. Os círculos em cada um dos gráficos representam os dados gerados, enquanto as linhas contínuas representam as aproximações obtidas pelo Método dos Mínimos Quadrados, para cada um dos modelos propostos. O código utilizado na geração dos resultados apresentados a seguir pode ser encontrado no Apêndice I.

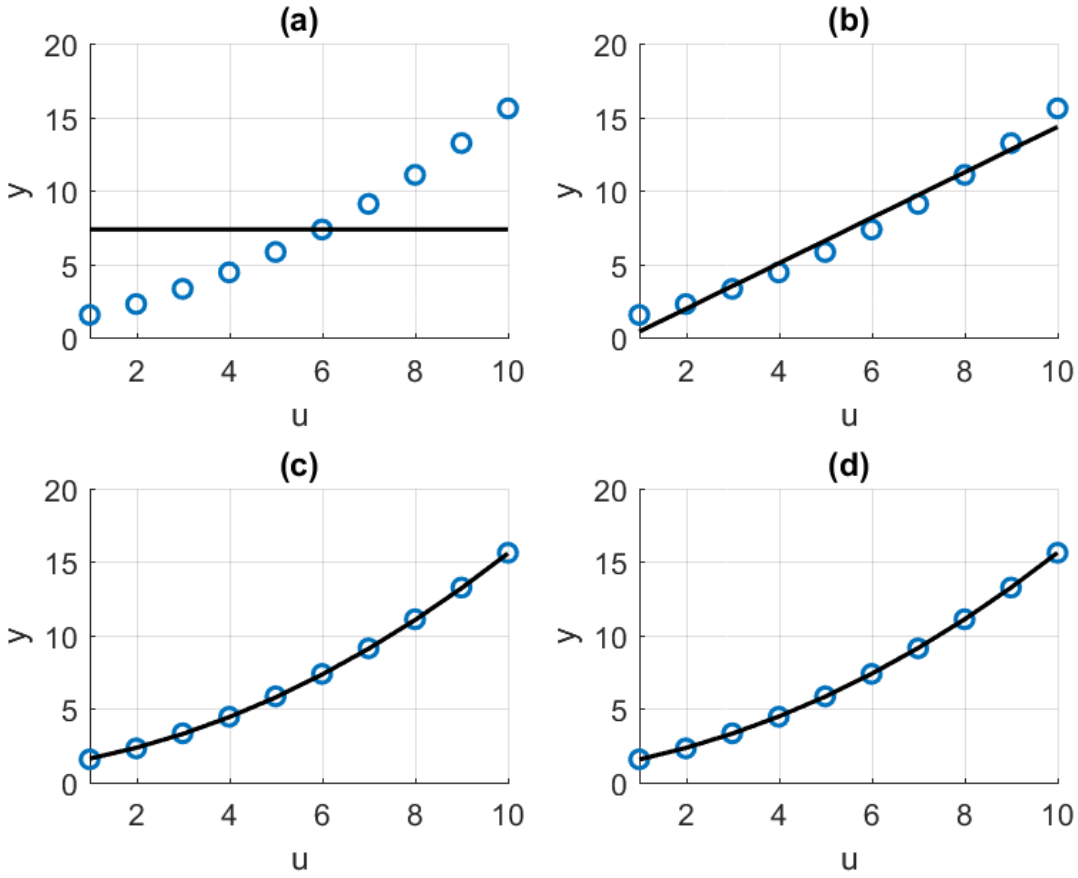

Figura 2.6 – Comparação entre o conjunto de dados gerado por (2.8) (círculos azuis) e as aproximações obtidas a partir da implementação do Método dos Mínimos Quadrados (linhas pretas), para cada um dos modelos propostos.

A partir destes resultados é possível avaliar a importância da escolha apropriada do modelo na implementação do Método dos Mínimos Quadrados. Caso o modelo seja muito mais simples do que a função que de fato descreve o conjunto de dados a ser aproximado, como foi o caso dos modelos (a) e (b) por exemplo, a estimação não representará adequadamente este conjunto de dados. E caso o modelo seja muito complexo, os dados reais podem ser muito bem aproximados, mas o gasto computacional exigido pode ser maior do que o necessário.

É interessante observar que a escolha do modelo está relacionada diretamente ao erro de estimação admissível. Se for possível tolerar pequenos erros de estimação, o modelo (b) poderia ser utilizado, o que proporcionaria inclusive uma redução no gasto computacional envolvido no cálculo de $y_{est}(i)$.

2.2.2 Implementação recursiva do Método dos Mínimos Quadrados

É importante lembrar que a implementação de um controlador adaptativo indireto exige que a estimação da planta seja realizada *online*. Como é necessário computar tanto $Y(t)$ quanto $\Phi(t)$ na estimação dos parâmetros contidos no vetor θ, é preciso ter em mente que, a cada iteração, todas as saídas $y(i)$ obtidas e todos os vetores $\varphi(i)$ construídos até o momento da estimação devem ser computados. Com o passar do tempo, maiores se tornam $Y(t)$ e $\Phi(t)$, e mais cálculos se fazem necessários para que os valores de θ sejam calculados através da implementação de (2.7).

Já que seria muito custoso computacionalmente obter os valores de θ em tempo real utilizando (2.7), uma nova forma de calculá-los deve ser proposta. Será aqui introduzida, portanto, a implementação recursiva do Método dos Mínimos Quadrados, que permite que os valores de $\theta(t)$ sejam calculados a partir de $\theta(t-1)$, de forma que o gasto computacional envolvido na implementação *online* do Método dos Mínimos Quadrados seja significativamente reduzido (Astrom; Wittenmark, 1995).

Para encontrar uma equação que permita a determinação dos valores de $\theta(t)$ a partir de $\theta(t-1)$, será utilizada uma variável auxiliar $P(t)$ definida por

$$P(t) = \left(\Phi^T(t)\Phi(t)\right)^{-1} = \left(\sum_{i=1}^{t} \varphi(i)\varphi^T(i)\right)^{-1} \tag{2.12}$$

Desta forma, segue que

$$\theta(t) = \theta(t-1) + P(t)\varphi(t)\left(y(t) - \varphi^T(t)\theta(t-1)\right) \tag{2.13}$$

É possível ainda reescrever θ na forma

$$\theta(t) = \theta(t-1) + K(t)\varepsilon(t) \tag{2.14}$$

em que

$$\begin{aligned} \varepsilon(t) &= y(t) - \varphi^T(t)\theta(t-1) \\ K(t) &= P(t)\varphi(t) \end{aligned} \tag{2.15}$$

A cada iteração, o valor de $\theta(t)$ é atualizado com base nos valores de $\theta(t-1)$. Além disso, a amplitude dos ajustes aplicados a $\theta(t)$ está diretamente relacionada ao erro de estimação $\varepsilon(t)$, e que este, por sua vez, é ponderado pela matriz $K(t)$.

Para computar $\theta(t)$ a cada iteração, utilizando (2.13), é necessário também computar $P(t)$. A relação introduzida em (2.16) será utilizada na determinação de $P(t)$, que dependerá dos valores de $P(t-1)$.

$$P(t) = P(t-1) - P(t-1)\varphi(t)\left(I + \varphi^T(t)P(t-1)\varphi(t)\right)^{-1}\varphi^T(t)P(t-1) \tag{2.16}$$

Assim sendo, a implementação Recursiva do Método dos Mínimos Quadrados (*Recursive Least Square* ou RLS) e a determinação dos valores de $\theta(t)$ são realizadas com base nas seguintes equações

$$\begin{aligned} P(t) &= P(t-1) - P(t-1)\varphi(t)\left(1 + \varphi^T(t)P(t-1)\varphi(t)\right)^{-1}\varphi^T(t)P(t-1) \\ K(t) &= P(t)\varphi(t) \\ \varepsilon(t) &= y(t) - \varphi^T(t)\theta(t-1) \\ \theta(t) &= \theta(t-1) - K(t)\varepsilon(t) \end{aligned} \tag{2.17}$$

Note que, como a computação de $\varphi^T(t)P(t-1)\varphi(t)$ resulta em um número real, a operação $\left(I + \varphi^T(t)P(t-1)\varphi(t)\right)$ pode ser substituída por $\left(1 + \varphi^T(t)P(t-1)\varphi(t)\right)$, já que a matriz I teria neste caso uma única linha e uma única coluna, e conteria apenas o número 1.

É importante ressaltar que a dimensão de $P(t)$ está diretamente relacionada à quantidade de parâmetros a serem estimados, ou seja, à dimensão de θ. Caso o modelo proposto possua n parâmetros a serem determinados, a matriz $P(t)$ terá n linhas e n colunas.

Para exemplificar a implementação do RLS considere um conjunto de dados gerado por (2.8), como foi feito na seção anterior. Lembre-se que a implementação recursiva deve ser

utilizada na determinação *online* dos parâmetros θ. Assim sendo, os dados utilizados na estimação dos parâmetros b_0, b_1 e b_2 serão gerados ao longo da simulação, através da computação de (2.8). Os resultados apresentados a seguir foram obtidos a partir de um algoritmo implementado no Matlab®, que pode ser encontrado no Apêndice I.

Considere neste caso um modelo como o apresentado em (2.18).

$$y_{est}(i) = b_0 + b_1 u(i) + b_2 u^2(i) \tag{2.18}$$

Perceba que três parâmetros precisam ser determinados, b_0, b_1 e b_2. Assim sendo, é possível concluir que a matriz $P(t)$ será uma matriz 3×3. Reescrevendo o modelo em sua forma matricial, segue que

$$\varphi^T(i) = \begin{bmatrix} 1 & u & u^2 \end{bmatrix} \tag{2.19}$$

$$\theta = \begin{bmatrix} b_0 & b_1 & b_2 \end{bmatrix}^T \tag{2.20}$$

$$y_{est}(i) = \varphi^T(i)\theta \tag{2.21}$$

Através da implementação de (2.17), os parâmetros b_0, b_1 e b_2 podem ser estimados com o passar das iterações. É importante ressaltar que as matrizes $P(i)$ e $\theta(i)$ são inicializadas com valores aleatórios entre 0 e 1. É atribuído a $u(i)$, em cada uma das iterações, um valor aleatório entre 0 e 10. Isto permite a análise do comportamento do estimador em uma estimação *online*, com um novo dado sendo adquirido a cada iteração. Em seguida, (2.18) é utilizada para que, a partir do $u(i)$ sorteado, seja obtido um valor de $y(i)$, que será então empregado na atualização dos parâmetros $\theta(i)$.

Note que, neste caso, os valores de $y(i)$ são obtidos durante a execução da simulação, enquanto que no exemplo apresentado na seção anterior o Método dos Mínimos Quadrados foi aplicado a um conjunto de dados construído previamente. Os resultados obtidos são apresentados nas Figs. 2.7, 2.8 e 2.9.

Os gráficos da Fig. 2.7 mostram a comparação entre os valores reais $y(i)$ gerados por (2.8) e os valores $y_{est}(i)$ obtidos a partir da estimação. O primeiro gráfico apresenta as primeiras trinta iterações enquanto o segundo, as últimas trinta. É interessante observar que o modelo se

ajusta, com o passar das iterações, ao conjunto de dados gerado. Inicialmente, os valores obtidos pelo modelo distam consideravelmente dos valores reais, enquanto que, após algumas iterações, o modelo já consegue prever com bastante precisão os valores de $y(i)$.

O gráfico da Fig. 2.8 mostra a evolução do erro de estimação $\varepsilon(t)$, que é a diferença entre o valor real $y(i)$ e o valor obtido a partir do modelo $y_{est}(i)$. Este erro é calculado através da equação $\varepsilon(t) = y(t) - \varphi^T(t)\theta(t-1)$, e permite que a eficácia do Método dos Mínimos Quadrados seja avaliada, já que, quando o modelo estiver devidamente ajustado e os seus parâmetros tiverem sido estimados, $\varepsilon(t)$ se aproximará de 0. Note que, no começo da estimação, os valores do erro são consideravelmente altos, enquanto que, com o passar das iterações, tendem a zero.

Os gráficos da Fig. 2.9, por fim, mostram a evolução dos parâmetros do modelo. As linhas tracejadas representam os valores que b_0, b_1 e b_2 devem assumir para que $y(i)$ seja igual a $y_{est}(i)$, que são, respectivamente, 1, 0,45, e 0,1, enquanto as linhas contínuas representam os valores que estes de fato assumem ao longo das iterações. É fácil perceber que a convergência dos parâmetros b_0, b_1 e b_2, que assumiram, ao fim da simulação, valores próximos de 1, 0,45, e 0,1, garantiu que $y(i)$ fosse igual a $y_{est}(i)$. Este resultado indica a eficácia da implementação recursiva do Método dos Mínimos Quadrados.

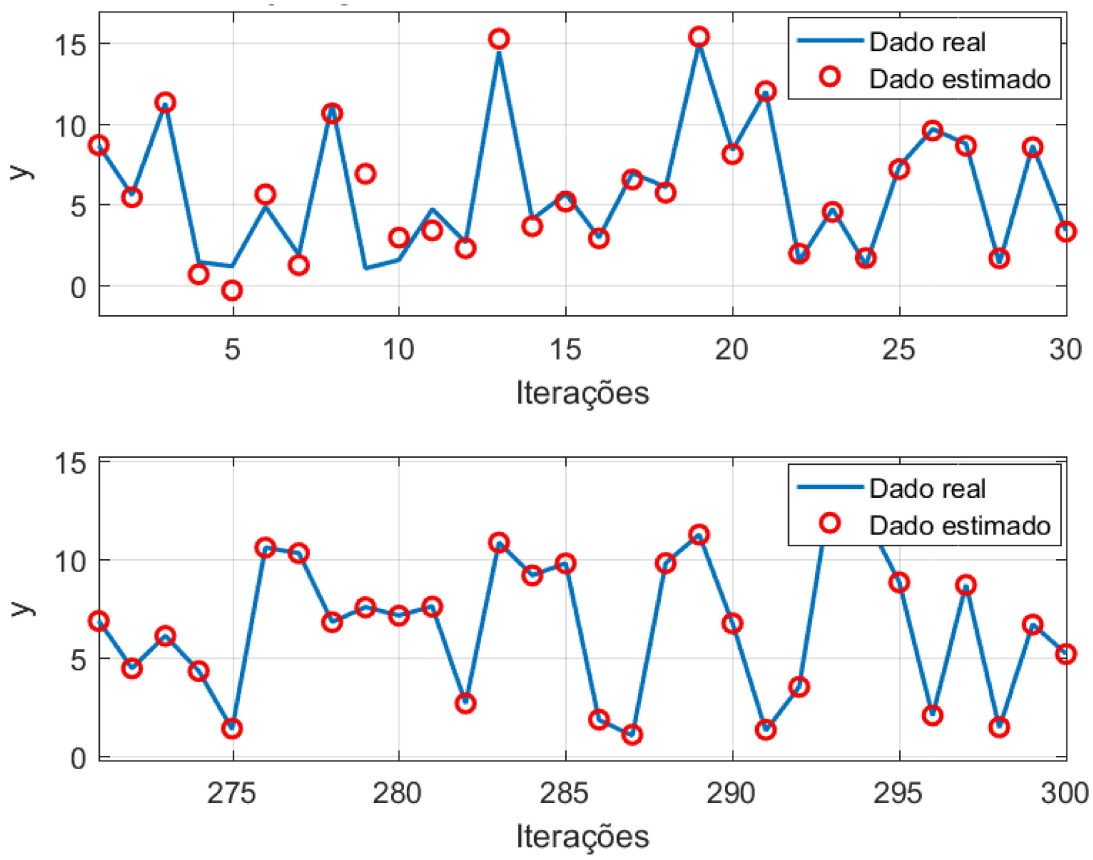

Figura 2.7 – Comparação entre os valores gerados a partir de (2.8) e os estimados através da implementação do RLS.

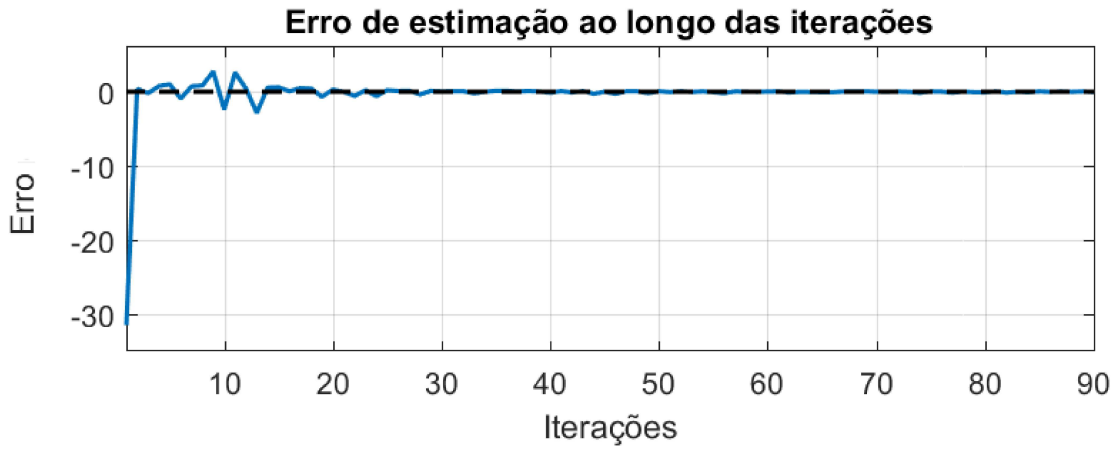

Figura 2.8 – Evolução do erro de estimação.

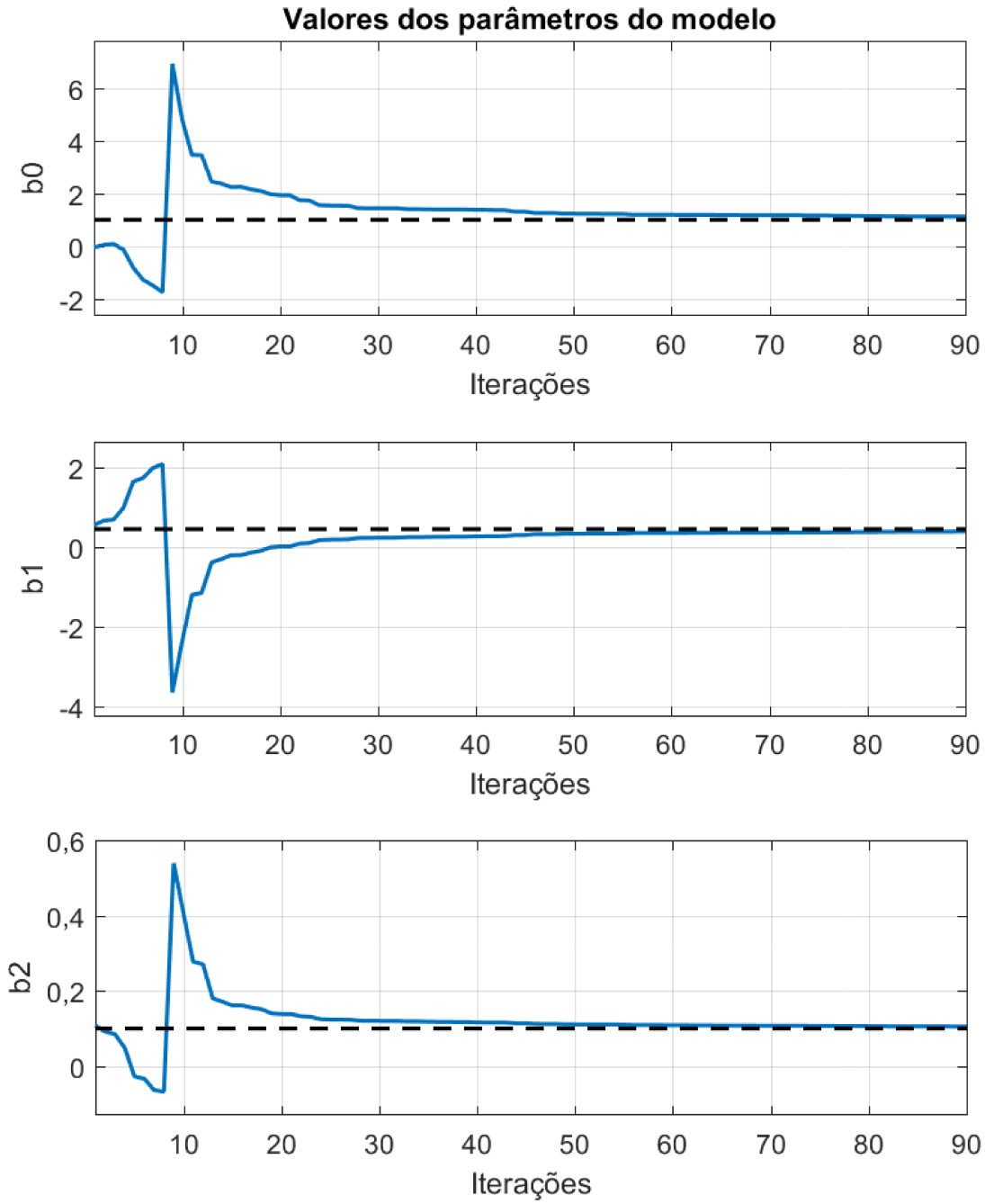

Figura 2.9 – Evolução dos parâmetros do modelo.

2.2.3 Utilização do Método dos Mínimos Quadrados na determinação dos parâmetros da planta de um sistema

Uma vez apresentada a implementação do RLS, é possível aplicá-lo na determinação dos parâmetros da planta de um dado processo a ser controlado. Será mostrado mais adiante

que, para tal, basta considerar como modelo uma equação de diferenças que represente o sistema a ser modelado e determinar os seus parâmetros. Mas antes disso, será introduzido o **operador de deslocamento no tempo** q (Astrom; Wittenmark, 1995). Este operador é responsável por promover um deslocamento discreto no tempo quando aplicado a um dado sinal. Considerando sua aplicação, por exemplo, à saída $y(t)$ de um sistema qualquer, seria obtida a relação

$$qy(t) = y(t+1) \tag{2.22}$$

De forma análoga, quando o inverso deste operador é aplicado a algum sinal, o deslocamento se dá para iterações passadas, como mostrado a seguir

$$q^{-1}y(t) = y(t-1) \tag{2.23}$$

Funções de transferência, que ditam a relação entre a entrada e a saída de processos, são sempre representadas no domínio s ou no domínio z. No entanto, o operador q permite que estas funções sejam representadas diretamente no domínio do tempo discreto. Considere por exemplo uma função de transferência arbitrária

$$G(z) = \frac{Y(z)}{U(z)} = \frac{0,65z^{-1}}{1-0,35z^{-1}} = \frac{0,65}{z-0,35} \tag{2.24}$$

Realizando algumas simplificações matemáticas e aplicando a transformada z inversa, é possível obter uma equação de diferenças que represente, no domínio do tempo, a planta $G(z)$, como mostrado a seguir

$$G(z) = \frac{Y(z)}{U(z)} = \frac{0,65z^{-1}}{1-0.35z^{-1}} \Rightarrow Y(z) = 0,35z^{-1}Y(z) + 0,65z^{-1}U(z) \Rightarrow$$
$$y(t) = 0,35y(t-1) + 0,65u(t-1) \tag{2.25}$$

Reescrevendo a equação de diferenças obtida utilizando o operador de deslocamento

q, é possível definir $g(t)$ como mostrado a seguir

$$y(t) = 0,35y(t-1) + 0,65u(t-1) = 0,35q^{-1}y(t) + 0,65q^{-1}u(t) \Rightarrow$$
$$g(t) = \frac{y(t)}{u(t)} = \frac{0,65q^{-}1}{1-0,35q^{-}1} = \frac{0,65}{q-0,35} \tag{2.26}$$

Note que (2.24) e (2.26), são bastante semelhantes. De fato, é possível concluir deste exemplo que o operador q permite a representação de uma função de transferência no domínio do tempo discreto, de forma que seja possível, ainda no domínio do tempo, avaliar o comportamento de um sistema através da análise dos seus polos e zeros. Utilizando o operador q, as funções de transferência serão representadas através de uma razão de polinômios dependentes de q.

Considere agora que o Método dos Mínimos Quadrados seja empregue na determinação dos parâmetros da planta de um sistema descrito pela relação

$$A(q)y(t) = B(q)u(t) \tag{2.27}$$

onde $A(q)$ e $B(q)$ são polinômios de ordem n e $m-1$ respectivamente.

$$A(q) = q^n + a_1q^{n-1} + \cdots + a_n \tag{2.28}$$
$$B(q) = b_1q^{m-1} + b_2q^{m-2} + \cdots + b_m \tag{2.29}$$

Reescrevendo (2.27) utilizando as definições de $A(q)$ e $B(q)$, segue que

$$\left(q^n + a_1q^{n-1} + \cdots + a_n\right)y(t) = \left(b_1q^{m-1} + b_2q^{m-2} + \cdots + b_m\right)u(t) \tag{2.30}$$

Multiplicando ambos os lados da equação por q^{-n} e desenvolvendo é possível obter

$$\left(1 + a_1q^{-1} + \cdots + a_nq^{-n}\right)y(t) = \left(b_1q^{m-n-1} + b_2q^{m-n-2} + \cdots + b_mq^{-n}\right)u(t) \Rightarrow$$
$$y(t) + a_1y(t-1) + \cdots + a_ny(t-n) = b_1u(t+m-n-1) + \cdots + b_mu(t-n) \tag{2.31}$$

Por fim, isolando o termo $y(n)$ em (2.31), (2.32) é obtida.

$$y(t) = -a_1 y(t-1) - \cdots - a_n y(t-n) + b_1 u(t+m-n-1) + \cdots + b_m u(t-n) \tag{2.32}$$

Note que (2.32) é bastante semelhante aos modelos que foram empregues até o momento para que o Método dos Mínimos Quadrados fosse implementado. Inclusive, é possível reescrever (2.32) na forma matricial

$$y(t) = \varphi^T(t-1)\theta \tag{2.33}$$

onde

$$\theta = \begin{bmatrix} a_1 & a_2 & \cdots & a_n & b_1 & b_2 & \cdots & b_m \end{bmatrix}^T \tag{2.34}$$

$$\begin{aligned} \varphi^T(t-1) = [&-y(t-1) \quad -y(t-2) \quad \cdots \quad -y(t-n) \\ &u(t+m-n-1) \quad u(t+m-n-2) \quad \cdots \quad u(t-n)] \end{aligned} \tag{2.35}$$

Assim sendo, para que o Método dos Mínimos Quadrados seja empregado na determinação dos parâmetros da planta de um dado processo a ser controlado, basta que o modelo proposto seja uma equação de diferenças que descreva o comportamento deste processo. Observe que, para que os parâmetros da planta sejam determinados, é necessário conhecimento prévio sobre a sua estrutura.

Para exemplificar a aplicação do Método dos Mínimos Quadrados na determinação dos parâmetros de um sistema, considere um processo arbitrário descrito pela equação de diferenças

$$y(t) = 1,0002y(t-1) + 0,0008y(t-2) - 0,0244u(t-1) + 0,1503u(t-2) \tag{2.36}$$

A função de transferência no domínio z associada a esta equação de diferenças é mostrada a seguir

$$\frac{Y(z)}{U(z)} = \frac{-0,0244z^{-1} + 0,1503z^{-2}}{1 - 1,0002z^{-1} - 0,0008z^{-2}} = \frac{-0,0244z + 0,1503}{z^2 - 1,0002z - 0,0008} \tag{2.37}$$

Para aplicar o Método dos Mínimos Quadrados, considere o modelo

$$y_{est}(t) = -a_1 y(t-1) - a_2 y(t-2) + b_0 u(t-1) + b_1 u(t-2) \tag{2.38}$$

É possível reescrever (2.38) em sua forma matricial

$$y_{est}(t) = \varphi^T(t-1)\theta \tag{2.39}$$

que segue

$$\theta = \begin{bmatrix} a_1 & a_2 & b_0 & b_1 \end{bmatrix}^T \tag{2.40}$$

$$\varphi^T(t-1) = \begin{bmatrix} -y(t-1) & -y(t-2) & u(t-1) & u(t-2) \end{bmatrix} \tag{2.41}$$

Considere que durante a estimação dos parâmetros do processo descrito por (2.36), uma entrada qualquer seja aplicada. A utilização de sinais periódicos é preferível uma vez que uma entrada periódica de período T permite a determinação de um modelo que contenha até T parâmetros (Astrom; Wittenmark, 1995). Além disso, simulações mostraram que a eficácia do RLS depende tanto da amplitude da entrada, que deve ser grande o suficiente para provocar alterações na saída, quanto de seu período, que deve ser maior do que o tempo de acomodação do sistema.

Foi então aplicada como entrada uma onda quadrada de amplitude igual a 10 e período de aproximadamente 2 segundos. Considere também que o tempo total de simulação seja de 10 segundos, escolhido de forma que a convergência do RLS possa ser observada. Os gráficos da Fig. 2.10 mostram a evolução temporal da entrada $u(t)$ e da saída $y(t)$.

O gráfico da Fig. 2.11 mostra como a estimação evolui ao longo do tempo. É possível perceber que no início da simulação os valores estimados y_{est} se encontram bastante distantes dos valores reais da saída y. No entanto isso muda com o passar das iterações e ao fim da simulação os valores de $y_{est}(t)$ e $y(t)$ são praticamente iguais. Essas observações podem ser confirmadas pela análise do gráfico da Fig. 2.12, que mostra a evolução do erro ao longo da simulação. Note que este assume valores elevados inicialmente. Contudo, com o passar das iterações, tende a

zero.

Por fim, os gráficos das Fig. 2.13 e 2.14 mostram a evolução de cada um dos parâmetros do modelo. As linhas tracejadas em preto representam os valores que a_1, a_2, b_0 e b_1 devem assumir para que $y_{est}(t)$ fosse igual a $y(t)$, enquanto as linhas contínuas em azul representam os valores que estes de fato assumem com o passar das interações. No título de cada gráfico se encontra o valor estimado para cada um dos parâmetros depois de 10 segundos de simulação, e entre parênteses, os valores que estes deveriam assumir para garantir que $y_{est}(t)$ fosse igual a $y(t)$. Observe que a convergência dos parâmetros mostra, mais uma vez, a eficácia da estimação.

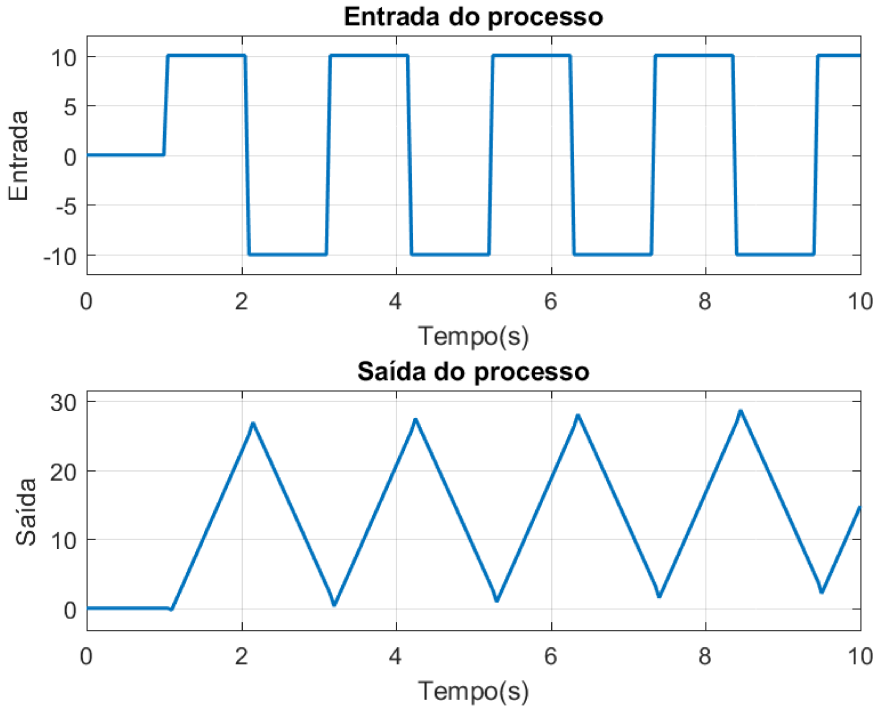

Figura 2.10 – Entrada aplicada ao processo descrito por (2.36) e sua respectiva saída ao longo do tempo.

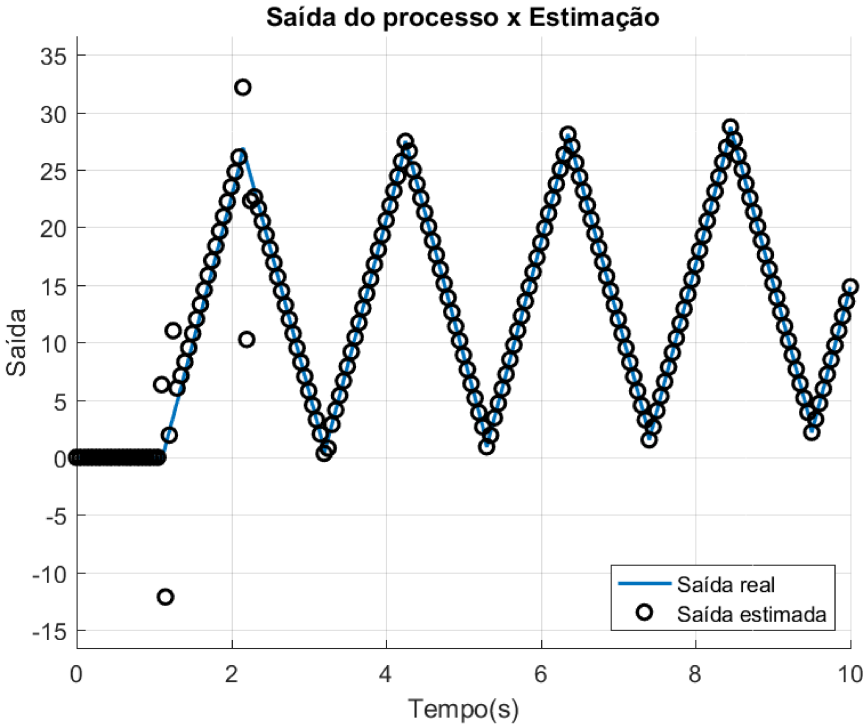

Figura 2.11 – Comparação entre a saída do sistema e a saída estimada a partir da implementação do RLS.

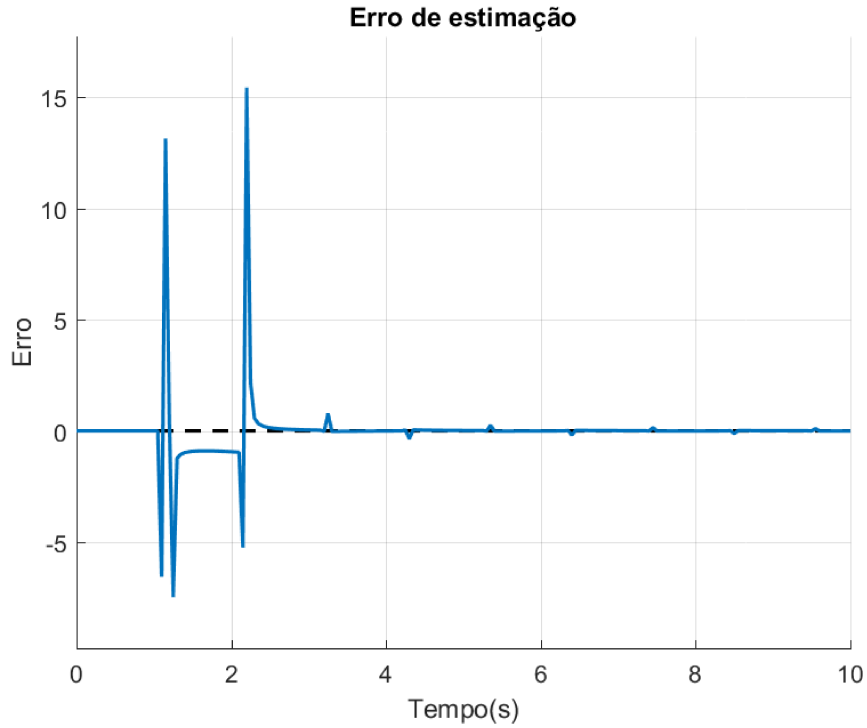

Figura 2.12 – Evolução do erro de estimação.

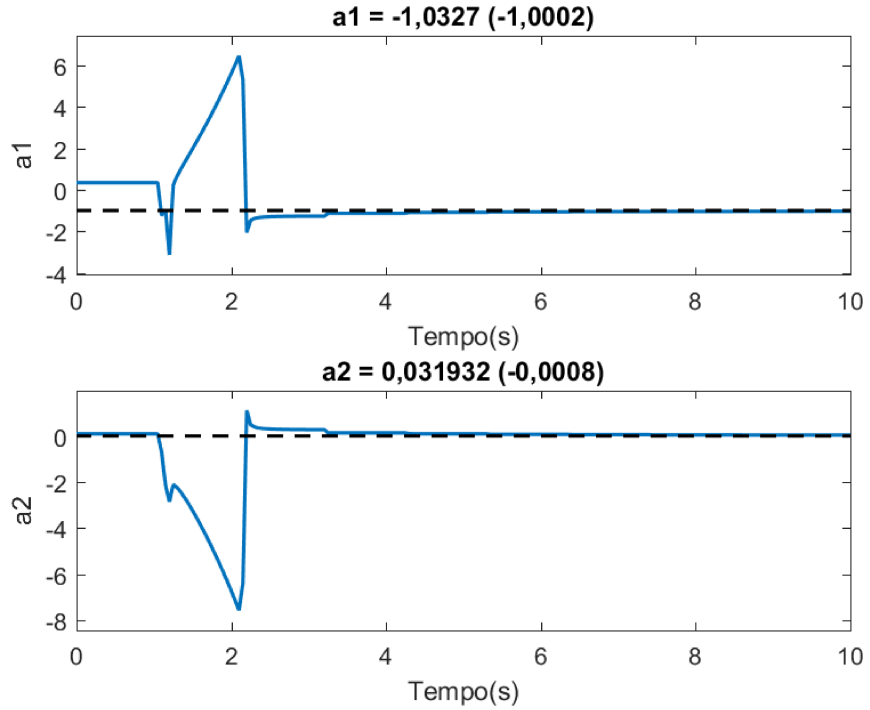

Figura 2.13 – Evolução dos parâmetros a_1 e a_2.

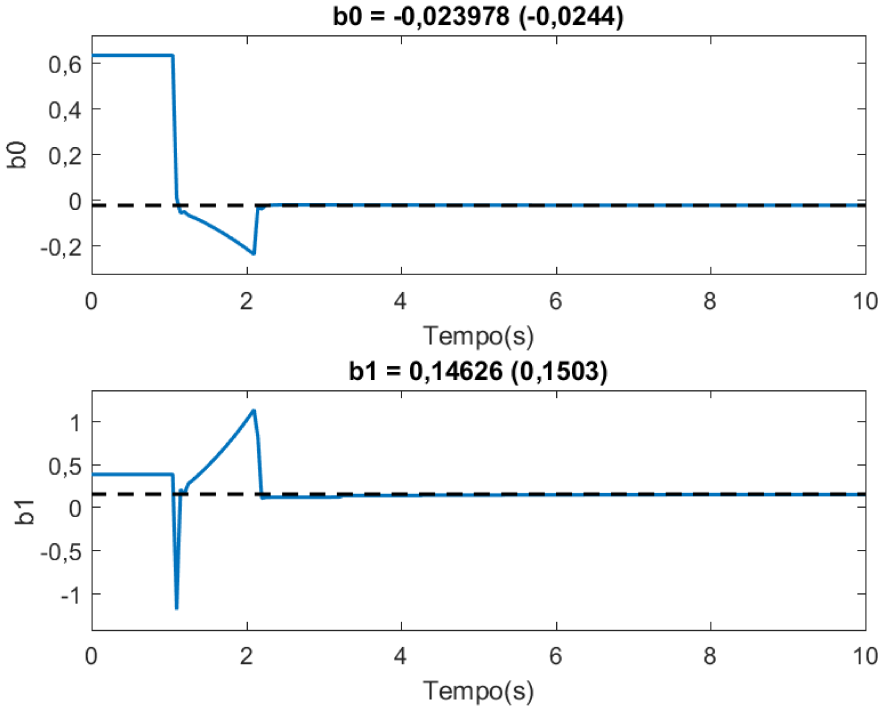

Figura 2.14 – Evolução dos parâmetros b_0 e b_1.

2.2.4 Estimando parâmetros que variam no tempo

Em algumas situações, os parâmetros a serem determinados variam ao longo do tempo. Essa variação pode estar relacionada a alterações na dinâmica do processo controlado ou à presença de perturbações externas. Para lidar com esse tipo de situação, é preciso avaliar como os parâmetros mudam com o tempo. Caso estes sofram, raramente, mudanças bruscas, basta que à matriz $P(t)$ seja atribuído, periodicamente, o valor αI, onde $\alpha >> 1$. Desta forma, o valor de $K(t)$ sofrerá ocasionalmente uma mudança brusca, que levará ao ajuste dos parâmetros θ. Este método é conhecido como *resetting* (Astrom; Wittenmark, 1995).

Em contrapartida, caso os parâmetros sofram, com frequência, mudanças menos intensas, basta considerar durante a estimação de θ o *fator de esquecimento* λ (Astrom; Wittenmark, 1995). Utilizando este fator, a função de custo dos mínimos quadrados pode ser reescrita como se segue

$$V(\theta, t) = \frac{1}{2} \sum_{i=1}^{t} \lambda^{t-i} \left(y(i) - \varphi^T(i)\theta\right)^2 \tag{2.42}$$

Esta técnica é também conhecida como *esquecimento exponencial*, graças à forma como o fator de esquecimento é inserido na equação de custo. Considerando λ um número que varie entre 0 e 1, é possível concluir que, graças a este fator, é dado peso 1 à estimação mais recentes e peso λ^n àquela que foi realizada n iterações atrás.

Para um dado valor de t, quanto menor for o valor de i, ou seja, quanto mais distante da estimação atual estiver a estimação i a ser computada, maior será o valor do expoente $t - i$ e consequentemente menor será o valor de λ^{t-i}. Desta forma, quanto mais distante da estimação atual estiver a estimação i a ser computada, menor será sua importância, uma vez que a ela será atribuído um peso menor.

Caso seja necessário utilizar a implementação recursiva do Método dos Mínimos Quadrados, basta empregar as (2.17), apresentadas na seção anterior, realizando o cálculo de $P(t)$ como segue

$$P(t) = \lambda^{-1}P(t-1) - \lambda^{-1}P(t-1)\varphi(t)\left(\lambda + \varphi^T(t)P(t-1)\varphi(t)\right)^{-1}\varphi^T(t)P(t-1) \tag{2.43}$$

2.3 Projeto do controlador utilizando o Método do Posicionamento dos Polos

O Método do Posicionamento dos Polos (*Pole Placement* ou PP), é um método que propõe o projeto de um controlador que possibilite que os polos do sistema controlado, em malha fechada, sejam realocados de forma a garantir que o mesmo se comporte como desejado. Os valores que os polos devem assumir são determinados a partir dos requisitos de desempenho (Astrom; Wittenmark, 1995).

2.3.1 Análise do processo a ser controlado

Considere que a relação entre a entrada e a saída do processo a ser controlado é dada por (2.44) (Astrom; Wittenmark, 1995).

$$A(q)y(t) = B(q)u(t) \tag{2.44}$$

onde $y(t)$ é a saída do sistema, $u(t)$ sua entrada, e $A(q)$ e $B(q)$ são polinômios em q. Os graus destes polinômios serão definidos da seguinte forma

$$degA = n \tag{2.45}$$

$$degB = degA - d_0 \tag{2.46}$$

onde $degA$ é o grau do polinômio A, e $degB$ é o grau do polinômio B.

Note que d_0, também conhecido como *delay* do processo, representa a diferença entre os graus dos polinômios $A(q)$ e $B(q)$. Esta por sua vez está diretamente relacionada à diferença entre os números de polos e zeros do sistema, uma vez que este tem sua função de transferência associada à relação $B(q)/A(q) = y(t)/u(t)$, como foi discutido na seção anterior. Além disso, é possível concluir que d_0 indica quanto tempo uma dada entrada demora para influenciar na saída do processo (Astrom; Wittenmark, 1995).

Assim como foi considerado na seção anterior, os polinômios $A(q)$ e $B(q)$ são definidos

como

$$A(q) = q^n + a_1 q^{n-1} + \cdots + a_n \tag{2.47}$$

$$B(q) = b_1 q^{m-1} + b_2 q^{m-2} + \cdots + b_m \tag{2.48}$$

É importante observar que o polinômio $A(q)$ é mônico, o que significa que o coeficiente que multiplica o fator de maior potência, neste caso q^n, é igual a 1.

2.3.2 Análise do controlador proposto

O Método do Posicionamento dos Polos será aqui aplicado a um controlador linear genérico descrito pela equação

$$R(q)u(t) = T(q)u_c(t) - S(q)y(t) \tag{2.49}$$

O objetivo desta seção é apresentar um algoritmo que possibilite a determinação dos polinômios $R(q)$, $T(q)$ e $S(q)$ de forma que o sistema em malha fechada apresente os polos estipulados pelo projetista.

É possível reescrever (2.49) da seguinte forma

$$u(t) = \frac{T(q)}{R(q)}u_c(t) - \frac{S(q)}{R(q)}y(t) \tag{2.50}$$

O diagrama de blocos que retrata a implementação do controlador aqui proposto é mostrada na Fig. 2.15. Lembre-se que o processo a ser controlado, descrito por (2.44), pode ser representado pela relação $y(t)/u(t) = B(q)/A(q)$.

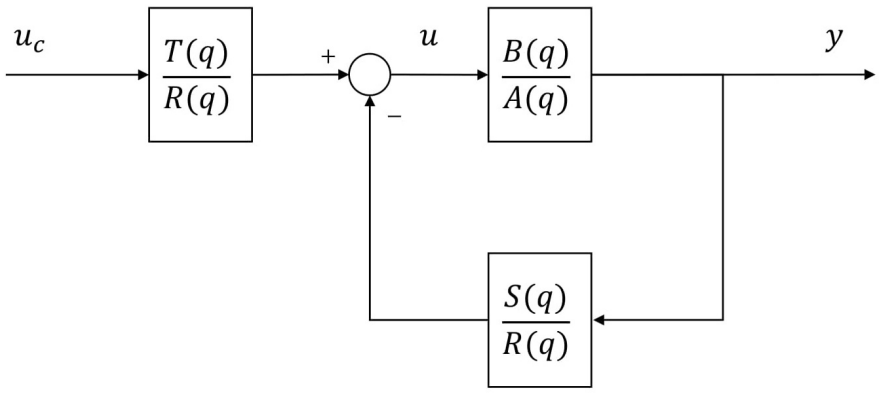

Figura 2.15 – Diagrama de blocos que ilustra a implementação do controlador linear genérico utilizado neste trabalho.

Para obter a função de transferência em malha fechada, que descreve a relação entre $u_c(t)$ e $y(t)$, basta combinar a definição de $y(t)$, de (2.44), e a de $u(t)$, de (2.50).

$$\begin{aligned} Ay(t) = Bu(t) = B\left(\frac{T}{R}u_c(t) - \frac{S}{R}y(t)\right) \Rightarrow \\ y(t) = \frac{BT}{AR + BS}u_c(t) \end{aligned} \tag{2.51}$$

A partir de agora, as notações A, B, R, S e T serão utilizadas para representar, respectivamente, os polinômios $A(q)$, $B(q)$, $R(q)$, $S(q)$ e $T(q)$. Esta mudança será executada para que fique claro que o desenvolvimento matemático apresentado nesta seção, e nas posteriores, é puramente algébrico e pode ser aplicado tanto a sistemas discretos quanto a sistemas contínuos, bastando que, no segundo caso os polinômios sejam escritos, não mais em função de q, mas sim de p, onde $p = d/dt$ é o operador diferencial (Astrom; Wittenmark, 1995)

Uma vez definida a função de transferência em malha fechada do sistema a ser controlado, é preciso lembrar que é necessário escolher R, S e T de forma que o sistema em malha fechada se comporte como desejado. Para facilitar o desenvolvimento matemático, será proposto então um modelo de referência, que descreverá a resposta desejada $y_m(t)$ para uma dada referência $u_c(t)$. Este modelo será descrito pela relação apresentada em (2.52). É importante ressaltar que os polos deste modelo, que será discutido em detalhes na Seção 2.3.3, devem ser

obtidos a partir dos requisitos de desempenho.

$$y_m(t) = \frac{B_m}{A_m} u_c(t) \tag{2.52}$$

Comparando (2.51) e (2.52), é possível concluir que, para que a resposta em malha fechada do sistema controlado $y(t)$ seja igual à resposta do modelo $y_m(t)$, é necessário que

$$\frac{BT}{AR + BS} = \frac{BT}{A_c} = \frac{B_m}{A_m} \tag{2.53}$$

onde

$$A_c = AR + BS. \tag{2.54}$$

A Equação 2.54 é também conhecida como *Equação de Diophantine* (Astrom; Wittenmark, 1995).

Perceba que será necessário realizar alguns cancelamentos entre os fatores de BT e A_c para que a igualdade (2.53) se torne verdadeira. No entanto, antes de analisar esta equação, considere que o polinômio B será fatorado da seguinte forma

$$B = B^+ B^- \tag{2.55}$$

onde B^+ é um polinômio mônico que contém os zeros que serão cancelados pelo controlador, enquanto B^- contém àqueles que não serão. Essa abordagem permite que, a partir do mesmo modelo, sejam projetados controladores distintos, uma vez que a determinação dos polinômios R, S, e T depende diretamente do cancelamento (ou não) dos zeros da planta (Astrom; Wittenmark, 1995). O efeito deste cancelamento será observado mais adiante com alguns exemplos.

Através da análise de (2.53), é possível concluir que B^+ deve ser um dos fatores de A_c, uma vez que B^+ contém os zeros a serem cancelados pelos polos do controlador. Além disso, A_m também deve ser um dos fatores de A_c, uma vez que, após efetuada a operação A_c/BT, apenas A_m restará no denominador. E por fim, é preciso considerar, para que a análise seja a mais geral possível, que há outros fatores de A_c que não serão cancelados. Estes serão representados por

A_0. Assim sendo, é possível definir A_c como (Astrom; Wittenmark, 1995)

$$A_c = A_0 A_m B^+ \tag{2.56}$$

Uma vez analisado o denominador da relação mostrada em (2.53), é preciso agora analisar o numerador. Como proposto anteriormente, B^+ será eliminado por algum dos polos do controlador, e no numerador sobrarão apenas B^- e T. Como os cancelamentos entre A_c e BT resultam em uma relação cujo numerador é B_m, é possível concluir que B^- deve ser um fator de B_m. Além disso, como feito anteriormente, é preciso considerar que há ainda outros fatores não considerados nesta análise, que não serão cancelados. Estes fatores serão representados por B'_m. Assim sendo, é possível definir B_m como (Astrom; Wittenmark, 1995)

$$B_m = B^- B'_m \tag{2.57}$$

Aplicando agora as definições propostas em (2.55) e (2.56), e observando, por meio da análise de (2.57), que $B^- = B_m/B'_m$, a relação apresentada em (2.53) pode ser reescrita da seguinte forma

$$\frac{BT}{A_c} = \frac{B^+B^-T}{A_0A_mB^+} = \frac{B^-T}{A_0A_m} = \frac{B_mT}{B'_mA_0A_m} = \frac{T}{A_0B'_m}\frac{A_m}{B_m} \tag{2.58}$$

Pela análise de (2.58), é possível concluir que para que BT/A_c seja igual a A_m/B_m, é necessário que T seja definido como

$$T = A_0 B'_m \tag{2.59}$$

Para desenvolver agora uma equação que permita o cálculo de R é preciso analisar a *Equação de Diophantine*. Reescrevendo esta equação utilizando as definições propostas em (2.55) e (2.56). Segue que

$$\begin{gathered} AR + BS = A_c \Rightarrow \\ \underbrace{AR}_{\mathsf{X}} + \underbrace{B^+B^-S}_{\mathsf{Y}} = \underbrace{A_0A_mB^+}_{\mathsf{Z}} \end{gathered} \tag{2.60}$$

Através da análise de (2.60), é possível concluir que como B^+ é um fator de Z e de Y, ele deve ser também um fator de X. Como A está relacionado à planta do processo a ser controlado, e não pode ter B^+ como um dos seus fatores, então B^+ deve ser fator de R. Como foi feito anteriormente, é preciso considerar que R é composto de outros fatores além de B^+, que serão representados por R'. Assim sendo, R pode ser definido como

$$R = R'B^+ \tag{2.61}$$

É possível então reescrever a *Equação de Diophantine* (2.54) da seguinte forma

$$AR' + B^-S = A_0A_m \tag{2.62}$$

Sendo T, R e S obtidos a partir de (2.59), (2.61) e (2.62), seria possível pensar que o controlador já foi determinado. No entanto, não basta que a implementação do mesmo possibilite que o sistema se comporte de acordo com o modelo de referência. É necessário também que o controlador seja causal (ou não antecipativo), e possua grau mínimo, de modo que o custo computacional para obtenção da ação de controle seja o menor possível. Uma das formas de satisfazer estas duas condições é fazendo com que

$$degA_m = degA \tag{2.63}$$

$$degB_m = degB \tag{2.64}$$

$$degA_0 = degA - degB^+ - 1 \tag{2.65}$$

onde o operador deg representa a ordem do polinômio a ele associado.

A relação entre o grau do controlador e o grau do processo a ser controlado é dada por

$$degR = degS = degT = degA - 1 \tag{2.66}$$

A Equação 2.66 indica que a ordem do controlador de grau mínimo, obtido a partir do Método do Posicionamento dos Polos, será sempre uma unidade menor que a ordem do processo

a ser controlado. Este, por sua vez, é representado pela função de transferência $B(q)/A(q)$, e tem sua ordem definida a partir de $degA$, uma vez que $degA > degB$.

Mais detalhes sobre as condições de causalidade e a determinação do controlador de grau mínimo podem ser encontrados no Apêndice IV.

Para exemplificar a implementação do Método do Posicionamento dos Polos, considere um processo arbitrário que tem sua dinâmica descrita pela função de transferência

$$G(z) = \frac{0,1065z + 0,0902}{z^2 - 1,6065z + 0,6065} \tag{2.67}$$

Dada a relação entre z e q já discutida anteriormente, é possível escrever, com base em $G(z)$, a função de transferência no domínio do tempo discreto $B(q)/A(q)$

$$\frac{B(q)}{A(q)} = \frac{0,1065q + 0,0902}{q^2 - 1,6065 + 0,6065} = \frac{b_0q + b_1}{q^2 + a_1q + a_2} \tag{2.68}$$

Considerando que $degA = 2$ segue, de (2.66), que

$$degR = degS = degT = 1 \tag{2.69}$$

Para garantir que $degA_m = degA$ e $degB_m = degB$ (com o intúito de satisfazer uma das condições de causalidade) será proposto o seguinte modelo (Astrom; Wittenmark, 1995)

$$\frac{B_m(q)}{A_m(q)} = \frac{b_{m0}q}{q^2 + a_{m1}q + a_{m2}} = \frac{0,1761q}{q^2 - 1,3205q + 0,4966} \tag{2.70}$$

Os polos deste modelo foram definidos a partir de um sistema de segunda ordem de frequência natural w_n igual a 1, e coeficiente de amortecimento ξ igual a $0,7$. Além disso, b_{m0} é determinado a partir da equação $b_{m0} = 1 + a_{m1} + a_{m2}$, o que garante um ganho estático unitário.

Uma vez proposto o modelo de referência, o próximo passo é fatorar o polinômio $B(q)$ como mostrado em (2.55). Para tal, é necessário escolher quais zeros serão cancelados pelo controlador. Neste caso deve-se cancelar o zero da planta, para que seja possível avaliar o efeito que o cancelamento de zeros tem na resposta do sistema. Uma vez que $B^+(q)$ deve ser um

polinômio mônico, $B(q)$ deve ser fatorado como se segue

$$B(q) = (b_0)(q + \frac{b_1}{b_0}) = B^-(q)B^+(q) \tag{2.71}$$

donde segue que

$$B^+(q) = q + \frac{b_1}{b_0} \tag{2.72}$$

$$B^-(q) = b_0 \tag{2.73}$$

Como neste exemplo $degR = 1$ e $degB^+ = 1$, então $degR' = 0$. Como $R'(q)$ é um polinômio mônico (Astrom; Wittenmark, 1995),

$$R'(q) = 1 \tag{2.74}$$

Assim sendo, considerando que $R'(q) = 1$, é possível encontrar $R(q)$ empregando (2.61).

$$R(q) = R'(q)B^+(q) = B^+(q) = q + \frac{b_1}{b_0} \tag{2.75}$$

Agora, para obter $S(q)$, é necessário resolver a *Equação de Diophantine*, introduzida em (2.62). Porém, a sua solução depende de A_0. O grau do polinômio $A_0(q)$ é dado por $degA_0 = degA - degB^+ - 1$. Como $degA = 2$ e $degB^+ = 1$, então $degA_0 = 0$. Como A_0 é um polinômio mônico (Astrom; Wittenmark, 1995), segue que

$$A_0(q) = 1 \tag{2.76}$$

Como $degR = degS = degT = 1$ e $S(q)$ não é um polinômio mônico (Astrom; Wittenmark, 1995), é possível escrever que

$$S(q) = s_0q + s_1 \tag{2.77}$$

O próximo passo é resolver a *Equação de Diophantine*.

$$A(q)R'(q) + B^{-}(q)S(q) = A_0(q)A_m(q) \Rightarrow$$
$$q^2 + (a_1 + b_0 s_0)q + (a_2 + b_0 s_1) = q^2 + a_{m1}q = a_{m_2} \tag{2.78}$$

Comparando os dois lados de (2.78) é possível concluir que

$$s_0 = \frac{a_{m1} - a_1}{b_0} \tag{2.79}$$
$$s_1 = \frac{a_{m2} - a_2}{b_0} \tag{2.80}$$

Por fim, (2.59) pode ser empregada no cálculo de $T(q)$ como mostrado a seguir

$$T(q) = A_0(q)B'_m(q) = B'_m(q) \tag{2.81}$$

De (2.57) segue que

$$B'_m(q) = \frac{B_m(q)}{B^{-}(q)} \Rightarrow B'_m(q) = \frac{b_{m0}}{b_0}q \tag{2.82}$$

o que implica que

$$T(q) = B'_m(q) = \frac{b_{m0}}{b_0}q \tag{2.83}$$

Os parâmetros do controlador aqui proposto, introduzido em (2.49), já foram determinados. Resumidamente

$$R(q) = q + \frac{b_1}{b_0} \tag{2.84}$$
$$T(q) = \frac{b_{m0}}{b_0}q \tag{2.85}$$
$$S(q) = s_0 q + s_1 \tag{2.86}$$

É possível então reescrever (2.49) da seguinte forma

$$\begin{aligned} R(q)u(t) &= T(q)u_c(t) - S(q)y(t) \Rightarrow \\ u(t) &= \frac{b_{m0}}{b_0}u_c(t) - s_0 y(t) - s_1 y(t-1) - \frac{b_1}{b_0}u(t-1) \end{aligned} \tag{2.87}$$

Antes de avaliar os resultados obtidos a partir da implementação deste controlador, é interessante avaliar a resposta do sistema em malha aberta. A Fig. 2.16 mostra a resposta a uma entrada degrau. É interessante observar que o sistema é naturalmente instável. Este fato era esperado uma vez que possui um polo em $z = 1$.

A resposta do processo em malha fechada a uma entrada degrau na referência encontra-se na Fig. 2.17. A comparação entre a saída do sistema $y(t)$ e a do modelo $y_m(t)$ mostra que, de fato, y acompanha y_m como o Método dos Posicionamento dos Polos propõe. A evolução da ação de controle é apresentada na Fig. 2.18.

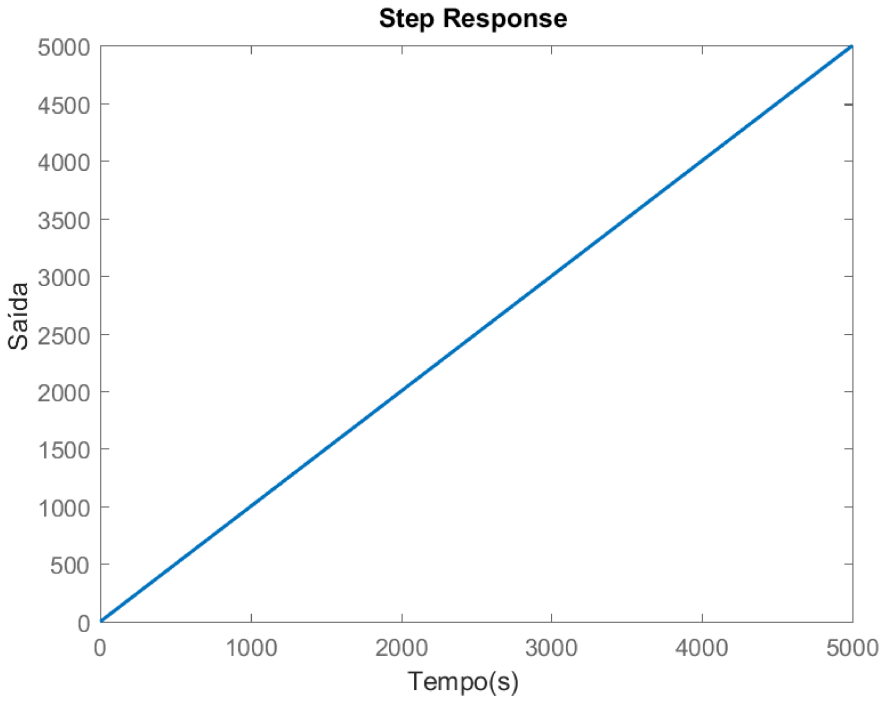

Figura 2.16 – Resposta de $G(z)$ a uma entrada degrau.

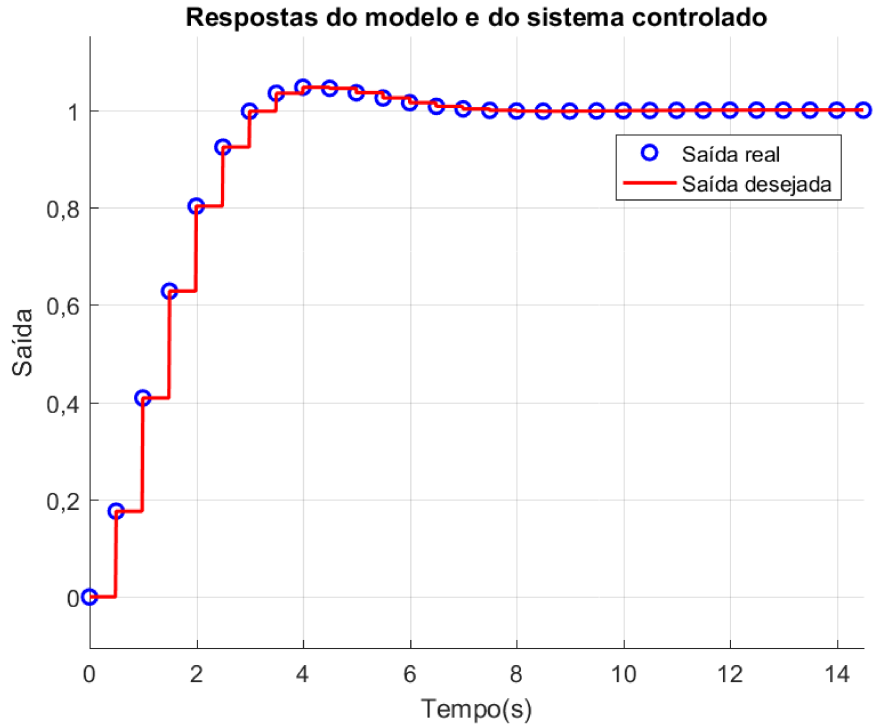

Figura 2.17 – Comparação entre a resposta ao degrau do processo controlado $y(t)$ e a resposta desejada $y_m(t)$.

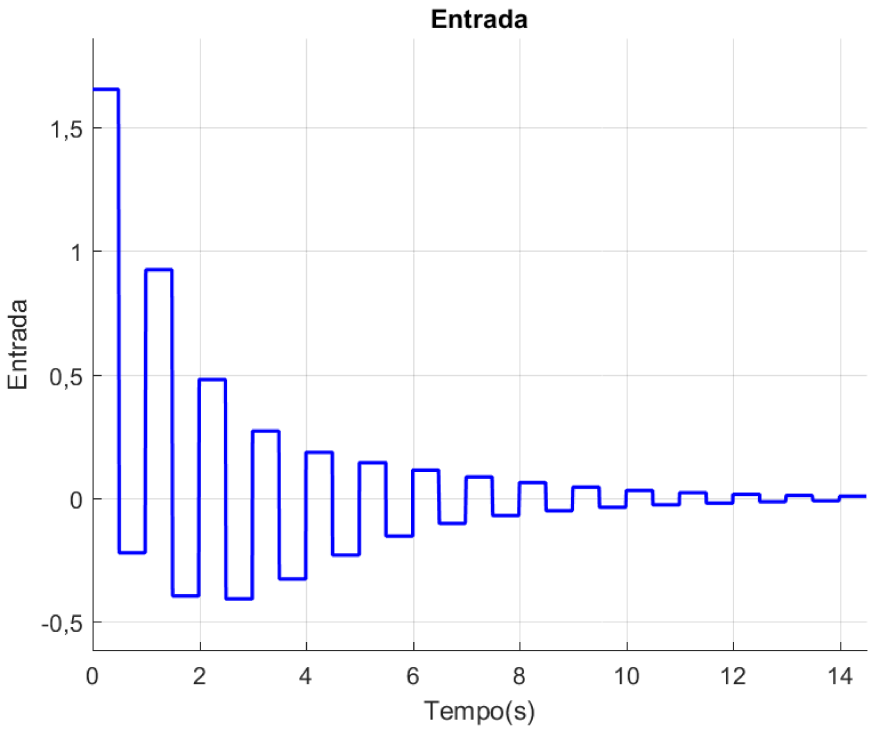

Figura 2.18 – Evolução da ação de controle.

Para entender melhor a influência do cancelamento dos zeros na resposta do sistema, considere agora que você deseja controlar o mesmo processo, porém agora sem realizar tal cancelamento. Então, levando em consideração que $B^+(q)$ é um polinômio mônico, $B(q)$ deve ser fatorado da seguinte maneira

$$B(q) = (1)(b_0 q + b_1) \tag{2.88}$$

de onde segue que

$$B^+(q) = 1 \tag{2.89}$$

$$B^-(q) = B(q) = b_0 q + b_1 \tag{2.90}$$

Considere agora que você deseja que seu modelo apresente o mesmo comportamento daquele introduzido em (2.70), que foi empregado no exemplo anterior. E além disso, deseja garantir que $degB_m = degB$, de forma que sejam satisfeitas as condições de causalidade. Assim sendo, o modelo pode ser definido como mostrado a seguir (Astrom; Wittenmark, 1995)

$$\frac{B_m}{A_m} = \beta \frac{b_0 q + b_1}{q^2 + a_{m1} q + a_{m2}} \tag{2.91}$$

sendo

$$\beta = \frac{1 + a_{m1} + a_{m2}}{b_0 + b_1} \tag{2.92}$$

Note que este modelo possui os mesmos polos do modelo apresentado em (2.70), que foram calculados com base nos requisitos de desempenho (w_n e ξ), e os mesmo zeros do processo a ser controlado. A constante β garante que o ganho estático do modelo seja unitário.

Segundo (2.61), $R = R'B^+$. Como $B^+(q) = 1$, segue que

$$R(q) = R'(q)B^+(q) = R'(q) \tag{2.93}$$

Utilizando (2.66) é possível concluir que $degR = degS = degT = 1$. Como $R'(q)$ é

um polinômio mônico, enquanto que $S(q)$ não,

$$R'(q) = R(q) = q + r_1 \tag{2.94}$$

$$S(q) = s_0 q + s_1 \tag{2.95}$$

Por fim, o grau do polinômio $A_0(q)$ é determinado.

$$degA_0 = degA - degB^+ - 1 = 1$$

Como A_0 é um polinômio mônico,

$$A_0(q) = q + a_0 \tag{2.96}$$

Agora basta resolver a *Equação de Diophantine*, introduzida em (2.62), para que $R(q)$ e $S(q)$ sejam determinados.

$$A(q)R'(q) + B^-(q)S(q) = A_0(q)A_m(q) \Rightarrow$$

$$q^3 + (a_1 + r_1 + b_0 s_0)q^2 + (a_2 + a_1 r_1 + b_0 s_1 + b_1 s_0)q + (r_1 a_2 + b_1 s_1) =$$

$$q^3 + (a_{m1} + a_0)q^2 + (a_{m2} + a_{m1} a_0)q + (a_{m2} a_0) \tag{2.97}$$

Comparando os dois lados da equação acima, é possível estabelecer as seguintes relações

$$\begin{cases} a_1 + r_1 + b_0 s_0 = a_{m1} + a_0 \\ a_2 + a_1 r_1 + b_0 s_1 + b_1 s_0 = a_{m2} + a_{m1} a_0 \\ r_1 a_2 + b_1 s_1 = a_{m2} a_0 \end{cases} \tag{2.98}$$

Isolando s_0 e s_1 na primeira e última equações do Sistema 2.98, respectivamente, segue que

$$s_0 = \frac{a_{m1} + a_0 - a_1 - r_1}{b_0} \tag{2.99}$$

$$s_1 = \frac{a_{m2} a_0 - a_2 r_1}{b_1} \tag{2.100}$$

Perceba que, de posse de r_1, já é possível determinar os valores de s_0 e s_1, uma vez que os demais parâmetros envolvidos em (2.99) e (2.100) são conhecidos. Para, por fim, obter r_1, basta substituir (2.99) e (2.100) na segunda equação do Sistema 2.98.

$$r_1 = \frac{a_{m2} + a_{m1}a_0 - a_2 - \dfrac{b_0}{b_1}a_{m2}a_0 - \dfrac{b_1}{b_0}(a_{m1} - a_1 + a_0)}{a_1 - \dfrac{b_0}{b_1}a_2 - \dfrac{b_1}{b_0}} \tag{2.101}$$

Por fim, $T(q)$ pode ser obtido através de (2.59). Porém é preciso, antes disso, determinar $B'_m(q)$ empregando (2.57) e lembrando que, neste caso, $B^-(q) = B(q) = b_0q + b_1$, como mostrado a seguir.

$$B'_m(q) = \frac{B_m(q)}{B^-(q)} = \frac{\beta(b_0q + b_1)}{b_0q + b_1} = \beta \tag{2.102}$$

donde segue que

$$T(q) = A_0(q)B'_m(q) = \beta(q + a_0) \tag{2.103}$$

Assim sendo, todos os parâmetros do controlador já foram determinados

$$R(q) = q + r_1 \tag{2.104}$$

$$S(q) = s_0q + s_1 \tag{2.105}$$

$$T(q) = \beta(q + a_0) \tag{2.106}$$

onde

$$r_1 = \frac{a_{m2} + a_{m1}a_0 - a_2 - \dfrac{b_0}{b_1}a_{m2}a_0 - \dfrac{b_1}{b_0}(a_{m1} - a_1 + a_0)}{a_1 - \dfrac{b_0}{b_1}a_2 - \dfrac{b_1}{b_0}} \tag{2.107}$$

$$s_0 = \frac{a_{m1} + a_0 - a_1 - r_1}{b_0} \tag{2.108}$$

$$s_1 = \frac{a_{m2}a_0 - a_2r_1}{b_1} \tag{2.109}$$

Assim como no caso em que o zero da planta foi cancelado, os parâmetros do controlador dependem apenas dos parâmetros da planta, supostamente conhecidos, e dos parâmetros do

modelo de referência. Neste caso, porém, os parâmetros do controlador dependem também de a_0, uma vez que $A_0(q)$ não é igual a 1. Vale ressaltar que $A_0(q)$ não depende da planta e nem do modelo, mas que, assim como o modelo, deve ser previamente fornecido para que o Método do Posicionamento dos Polos possa ser implementado. A escolha de a_0 fica a cargo do projetista. Neste trabalho, será feita a consideração de que $a_0 = 0$ (Astrom; Wittenmark, 1995).

De posse de $R(q)$, $S(q)$ e $T(q)$, é possível reescrever (2.49)

$$R(q)u(t) = T(q)u_c(t) - S(q)y(t) \Rightarrow$$
$$u(t) = \beta(u_c(t) + a_0 u_c(t-1)) - s_0 y(t) + s_1 y(t-1) - r_1 u(t-1) \tag{2.110}$$

A resposta do processo controlado a uma entrada degrau, após a implementação do controlador, pode ser observada na Fig. 2.19. A comparação entre a saída do sistema $y(t)$ e a do modelo $y_m(t)$ mostra que, de fato, $y(t)$ acompanha $y_m(t)$ como o Método dos Posicionamento dos Polos propõe. A evolução da ação de controle $u(t)$ é apresentada na Fig. 2.20.

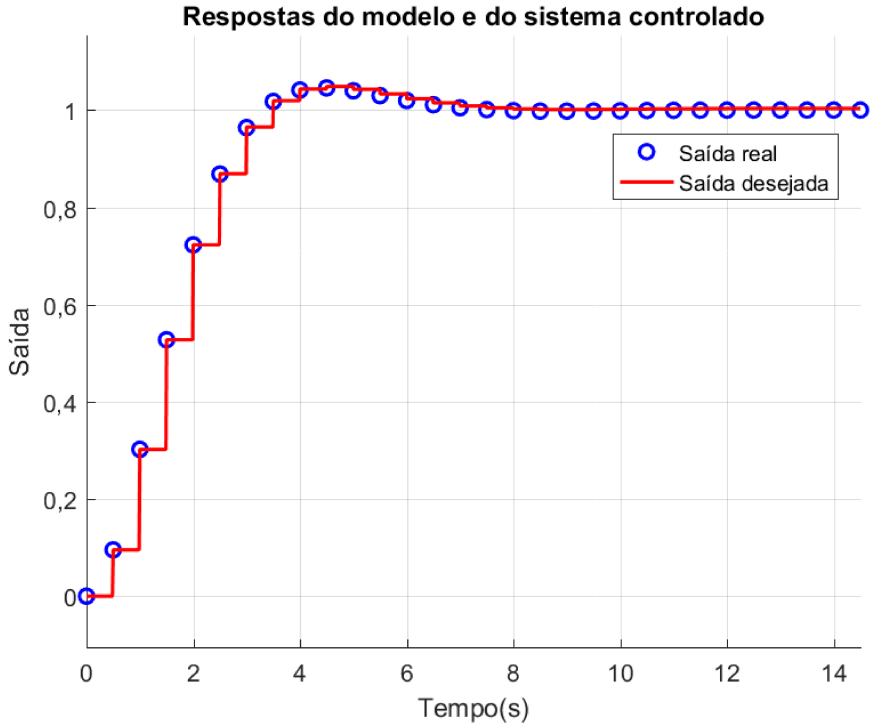

Figura 2.19 – Comparação entre a resposta ao degrau do processo controlado $y(t)$ e a resposta desejada $y_m(t)$.

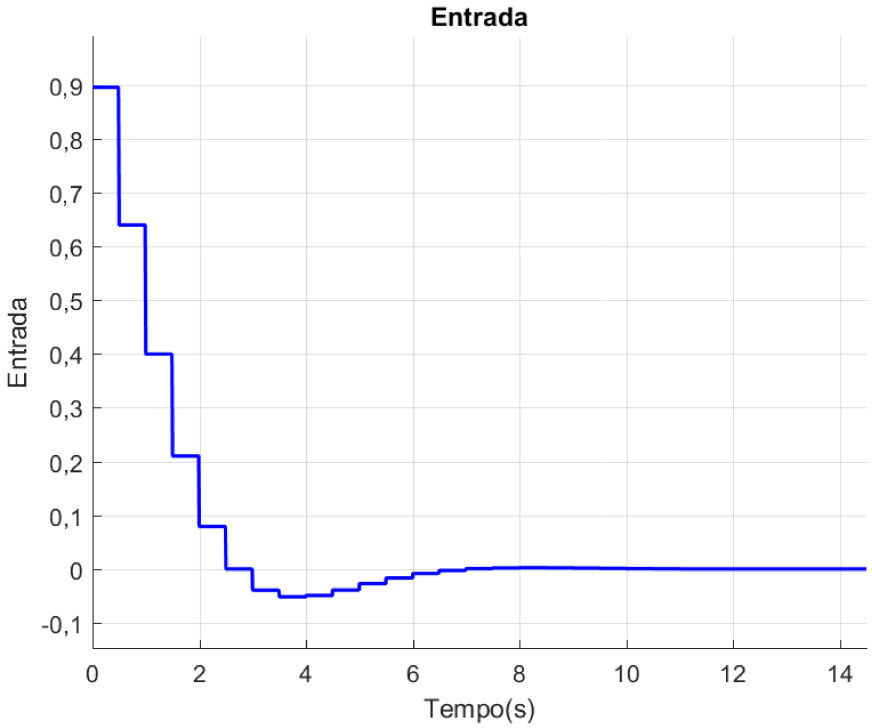

Figura 2.20 – Evolução da ação de controle.

Até o momento, dois controladores, projetados a partir do Método do Posicionamento de Polos, foram apresentados. Um deles emprega o cancelamento do zero da planta, enquanto o outro não, e é pertinente que os resultados obtidos a partir da implementação de cada um dos controladores sejam avaliados e comparados.

As Figs. 2.17 e 2.19 mostram que a implementação de qualquer um dos controladores garante que a saída do processo controlado se equipare à saída do modelo. No entanto, a ação de controle fornecida por cada um deles é bem diferente, como é possível observar nas Figs. 2.18 e 2.20.

A ação de controle obtida a partir do primeiro controlador, que foi projetado para cancelar o zero da planta, apresentou uma característica extremamente oscilatória e uma amplitude maior do que aquela observada no segundo controlador, que foi projetado para não cancelar o zero da planta.

A princípio, a oscilação que aparece na ação de controle quando os zeros da planta são cancelados poderia ser desprezada, já que, após a implementação do controlador, o sistema de fato se comporta como o modelo. No entanto, dependendo da dinâmica do sistema, essas oscilações podem levar a saída $y(t)$ à divergência, depois de poucos segundos. Assim sendo, é

recomendado que os zeros da planta não sejam cancelados (Astrom; Wittenmark, 1995).

2.3.3 Determinação do modelo com base nos requisitos de controle

Como mostrado nas seções anteriores, com o Método do Posicionamento dos Polos é possível projetar um controlador que faça com que o processo controlado apresente uma certa resposta desejada, que é especificada por um modelo da forma

$$y_m(t) = \frac{B_m}{A_m} u_c(t) \tag{2.111}$$

O controlador proposto deve ser capaz de fazer com que a saída do processo controlado $y(t)$ se equipare à saída do modelo $y_m(t)$ para uma dada referência $u_c(t)$.

Neste trabalho, foi proposto um modelo de referência de segunda ordem baseado em (2.112), também conhecida como *forma-padrão* dos sistemas de segunda ordem, que possibilita a análise do comportamento de sistemas analógicos de segunda ordem (Ogata, 2010).

$$G(s) = \frac{w_n^2}{s^2 + 2\xi w_n s + w_n^2} \tag{2.112}$$

Como dito anteriormente, a função de transferência do modelo é construída a partir da especificação dos requisitos de desempenho. Neste trabalho serão adotados o tempo de acomodação t_a e o valor do sobressinal (*overshoot*) M_p.

Os polos do modelo, z_1 e z_2, são determinados através da discretização dos polos s_1 e s_2, que são definidos a partir de (2.112), que depende dos requisitos de desempenho, ξ (fator de amortecimento) e w_n (frequência natural), especificados pelo projetista. A partir de z_1 e z_2 o polinômio A_m pode ser construído. Os zeros do modelo são então determinados a partir do processo a ser controlado. Por fim, um fator β é inserido para garantir que o ganho estático do modelo seja unitário, e a partir daí é especificado o polinômio B_m.

Os polos s_1 e s_2 de G(s) podem ser determinados a partir de (2.112)

$$s_1 = -\xi w_n + j w_n \sqrt{1 - \xi^2} = M + jN \tag{2.113}$$

$$s_2 = -\xi w_n - j w_n \sqrt{1 - \xi^2} = M - jN \tag{2.114}$$

onde

$$\begin{aligned} M &= -\xi w_n \\ N &= w_n\sqrt{1-\xi^2} \end{aligned} \tag{2.115}$$

Uma das formas de mapear os polos s_1 e s_2 para o domínio discreto é utilizando o Método de Tustin (Ogata, 1987), que propõe que

$$s = \frac{2}{T}\frac{z-1}{z+1} \Rightarrow z = \frac{1+\frac{T}{2}s}{1-\frac{T}{2}s} = \frac{1+ts}{1-ts} \tag{2.116}$$

onde $t = \frac{T}{2}$, e T é o período de amostragem.

Aplicando o Método de Tustin ao polo s_1 é possível obter seu equivalente z_1 no domínio discreto como mostrado a seguir

$$\begin{aligned} z_1 = \frac{1+ts_1}{1-ts_1} \Rightarrow z_1 = \frac{1+t(M+jN)}{1-t(M+jN)} \Rightarrow \\ z_1 = \frac{1-(tM)^2-(tN)^2}{(1-tM)^2+(tN)^2} + j\frac{2tN}{(1-tM)^2+(tN)^2} \end{aligned} \tag{2.117}$$

De forma análoga, é possível obter z_2 a partir do mapeamento do polo s_2

$$z_2 = \frac{1-(tM)^2-(tN)^2}{(1-tM)^2+(tN)^2} - j\frac{2tN}{(1-tM)^2+(tN)^2} \tag{2.118}$$

Observe que, z_1 e z_2 são polos complexos conjugados, assim como s_1 e s_2.

Para facilitar as manipulações matemáticas, (2.117) e (2.118) podem ser reescritas da seguinte forma

$$z_1 = O + jP \tag{2.119}$$

$$z_2 = O - jP \tag{2.120}$$

onde

$$\begin{aligned} O &= \frac{1-(tM)^2-(tN)^2}{(1-tM)^2+(tN)^2} \\ P &= \frac{2tN}{(1-tM)^2+(tN)^2} \end{aligned} \tag{2.121}$$

Perceba que os polos z_1 e z_2 estão indiretamente relacionados aos valores de w_n e ξ, uma vez que são definidos em função de O e P, que por sua vez dependem de M e N, que são determinado a partir de w_n e ξ. Além disso, é importante lembrar que ξ e w_n podem ser obtidos a partir dos requisitos de desempenho, como mostram (2.122) e (2.123) (Ogata, 2010).

$$\xi = \sqrt{\frac{ln^2 M_p}{ln^2 M_p + \pi^2}} \tag{2.122}$$

$$w_n = \frac{4}{\xi t_a} \tag{2.123}$$

onde M_p é o valor do sobressinal (*overshoot*) e t_a é o tempo de acomodação do sistema. As características do modelo (ξ e w_n) podem ser calculadas a partir dos requisitos de desempenho a partir de (2.122) e (2.123).

Uma vez que os polos tenham sido mapeados para o plano z, já é possível escrever a equação característica $D(z)$ do modelo (Ogata, 1987).

$$D(z) = (z - z_1)(z - z_2) = z^2 - 2Oz + (O^2 + P^2) \tag{2.124}$$

É possível ainda escrever (2.124) na forma

$$D(z) = z^2 + a_{m1}z + a_{m2}$$

de onde segue que

$$a_{m1} = -2O \tag{2.125}$$

$$a_{m2} = O^2 + P^2 \tag{2.126}$$

Dada a relação já observada entre as variáveis z e q, é possível relacionar a equação

característica $D(z)$ construída a partir dos polos z_1 e z_2 à equação característica $A_m(q)$ do modelo. Assim sendo,

$$A_m(q) = q^2 + a_{m1}q + a_{m2} \tag{2.127}$$

Uma das formas de satisfazer a condição de causalidade que exige que $degB_m = degB$ é fazendo com que $B_m = B$, ou seja, garantindo que os zeros do modelo sejam os mesmos do processo a ser controlado (Astrom; Wittenmark, 1995). Esta será a abordagem utilizada neste trabalho. Além disso, é importante lembrar que o sistema a ser controlado teve seu comportamento representado por uma função de transferência de segunda ordem

$$\frac{B(q)}{A(q)} = \frac{b_0q + b1}{q^2 + a_1q + a_2} \tag{2.128}$$

Uma constante β é então adicionada ao modelo para garantir que o ganho estático seja unitário. A função de transferência final do modelo é por fim apresentada em (2.129).

$$\frac{B_m(q)}{A_m(q)} = \beta\frac{b_0q + b_1}{q^2 + a_{m1}q + a_{m2}} \tag{2.129}$$

onde β é definido como

$$\beta = \frac{1 + a_{m1} + a_{m2}}{b_0 + b_1} \tag{2.130}$$

2.4 Self Tuning Regulator

O esquema adaptativo proposto neste trabalho (Fig. 2.21) é composto de dois *loops*. Um interno, no qual é implementado o controle em malha fechada do processo, e um externo, em que os parâmetros do processo e do controlador são determinados. Por meio de um estimador recursivo e de um bloco para sintonização do controlador, este esquema automatiza os processos de modelagem e *design* e por essa razão é conhecido como *Self Tuning Regulator*, ou STR (Astrom; Wittenmark, 1995).

Tanto os parâmetros do processo quanto os do controlador são atualizados a cada amostragem e o controlador é sintonizado de forma que o sistema apresente um certo comportamento em malha fechada (Astrom; Wittenmark, 1995).

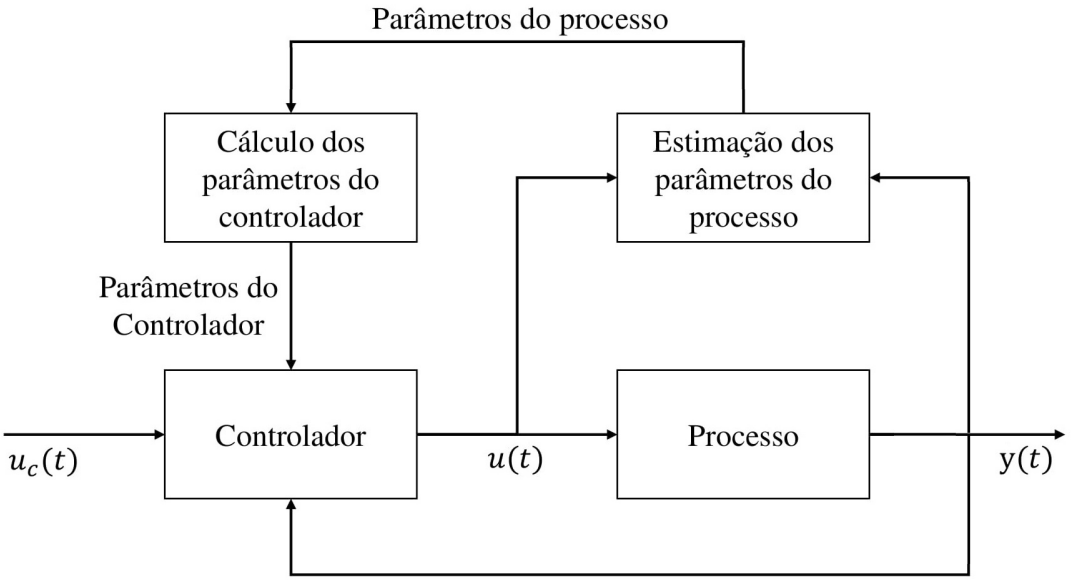

Figura 2.21 – Diagrama de blocos que ilustra o funcionamento de um STR.

Diversos métodos podem ser utilizados na obtenção de uma estimativa dos parâmetros do processo e no projeto do controlador. Neste trabalho uma combinação entre o Método Mínimos Quadrados Recursivo e o Posicionamento dos Polos será adotada.

Os controladores adaptativos são tipicamente implementados no controle de sistemas de dinâmica variante no tempo. Para simplificar este tipo problema, considera-se que a estrutura

da planta (quantidade de polos e zeros) que descreve o comportamento do processo controlado se mantém ao longo do tempo, mas que seus parâmetros se alteram à medida que a sua dinâmica muda. É por isso que os parâmetros do processo devem ser estimados em cada instante de amostragem. Note que, para que o STR seja projetado, é necessário que se tenha um conhecimento prévio da planta, já que a estimação dos seus parâmetros é realizada com base em um modelo da forma

$$y_{est}(t) = \varphi^T(t-1)\theta \tag{2.131}$$

onde

$$\theta = \begin{bmatrix} a_1 & \cdots & a_n & b_1 & \cdots & b_m \end{bmatrix}^T \tag{2.132}$$

$$\varphi^T(t-1) = \begin{bmatrix} -y(t-1) & \cdots & -y(t-n) & u(t+m-n-1) & \cdots & u(t-n) \end{bmatrix} \tag{2.133}$$

Note que para construir este modelo, é necessário que se tenha em mente a quantidade de polos e zeros que possui a função de transferência do sistema a ser controlado. Os valores de m e n estão, respectivamente, associados à quantidade de zeros e polos do sistema.

Para entender melhor a implementação e o funcionamento de um STR, considere que seu objetivo seja controlar o mesmo sistema que foi introduzido na Seção 5.2, cuja função de transferência é

$$G(z) = \frac{0,1065z + 0,0902}{z^2 - 1,6065z + 0,6065} \tag{2.134}$$

o que implica que

$$\frac{B(q)}{A(q)} = \frac{0,1065q + 0,0902}{q^2 - 1,6065 + 0,6065} = \frac{b_0 q + b_1}{q^2 + a_1 q + a_2} \tag{2.135}$$

Lembre-se que o controlador projetado a partir do Método do Posicionamento dos Polos, tem seus parâmetros determinados em função do processo e do modelo de referência. Será proposto neste exemplo o modelo introduzido em (2.129) e discutido na Seção 2.3.3

$$\frac{B_m(q)}{A_m(q)} = \beta \frac{b_0 q + b_1}{q^2 + a_{m1} q + a_{m2}} \tag{2.136}$$

em que

$$\beta = \frac{1 + a_{m1} + a_{m2}}{b_0 + b_1} \tag{2.137}$$

Considerando que nenhum zero da planta será cancelado, os seus parâmetros podem ser determinados como mostrado na Seção 5.2. Perceba que a planta e o modelo utilizados nesta seção são os mesmos que foram aqui propostos. Desta forma segue que

$$R(q) = q + r_1 \tag{2.138}$$

$$S(q) = s_0 q + s_1 \tag{2.139}$$

$$T(q) = \beta(q + a_0) \tag{2.140}$$

sendo

$$r_1 = \frac{a_{m2} + a_{m1}a_0 - a_2 - \frac{b_0}{b_1}a_{m2}a_0 - \frac{b_1}{b_0}(a_{m1} - a_1 + a_0)}{a_1 - \frac{b_0}{b_1}a_2 - \frac{b_1}{b_0}} \tag{2.141}$$

$$s_0 = \frac{a_{m1} + a_0 - a_1 - r_1}{b_0} \tag{2.142}$$

$$s_1 = \frac{a_{m2}a_0 - a_2 r_1}{b_1} \tag{2.143}$$

Ainda na Seção 5.2, foi possível concluir que a ação de controle, neste caso, pode ser calculada por meio da equação

$$u(t) = \beta(u_c(t) + a_0 u_c(t-1)) - s_0 y(t) + s_1 y(t-1) - r_1 u(t-1) \tag{2.144}$$

No entanto, é preciso lembrar que o STR propõe a estimação *online* dos parâmetros da planta, de forma que o controlador seja capaz de lidar com variações na dinâmica do sistema. Assim sendo, considere que os valores de b_0, b_1, a_1 e a_2 sejam desconhecidos, e que eles serão estimados recursivamente a partir do Métodos dos Mínimos Quadrados. Para que este método seja implementado, é preciso que um modelo, que represente o comportamento do processo a ser controlado, seja escolhido, e é neste momento que será importante que o projetista possua um conhecimento prévio acerca do comportamento do sistema. Neste caso, sabe-se que este pode ser representado por uma planta de segunda ordem. Então

$$\frac{y_{est}(t)}{u(t)} = \frac{b_0 q + b_1}{q^2 + a_1 q + a_2} \tag{2.145}$$

o que implica que (2.131) pode ser reescrita na forma

$$y_{est}(t)(q^2 + a_1 q + a_2) = u(t)(b_0 q + b_1) \Rightarrow$$
$$y_{est}(t) = -a_1 y_{est}(t-1) - a_2 y_{est}(t-2) + b_0 u(t-1) + b_1 u(t-2) \tag{2.146}$$

de onde segue que

$$\theta = \begin{bmatrix} a_1 & a_2 & b_0 & b_1 \end{bmatrix}^T \tag{2.147}$$
$$\varphi^T(t-1) = \begin{bmatrix} -y_{est}(t-1) & -y_{est}(t-2) & u(t-1) & u(t-2) \end{bmatrix} \tag{2.148}$$

Por fim, é possível obter os valores de a_{m1} e a_{m2} a partir dos requisitos de desempenho. Para tal, basta empregar as equações introduzidas na Seção 2.3.3. Considere que $M_p = 5\%$ e $t_a = 1s$, onde M_p é o valor máximo do sobressinal (*overshoot*) e t_a é o tempo de acomodação. Desta forma, é possível obter, através do emprego de (2.122) e (2.123), $\xi = 0,6901$ e $w_n = 5,7962$.

Uma vez que ξ e w_n tenham sido calculados, é possível obter os valores de a_{m1} e a_{m2} a partir das equações introduzidas na Seção 2.3.3. Do emprego destas, segue que

$$M = -\xi w_n = -0,4$$
$$N = w_n \sqrt{1 - \xi^2} = 0,4195$$
$$O = \frac{1 - (tM)^2 - (tN)^2}{(1 - tM)^2 + (tN)^2} = 0,8018$$
$$P = \frac{2tN}{(1 - tM)^2 + (tN)^2} = 0,1718$$
$$a_{m1} = -2O = -1,6036$$
$$a_{m2} = O^2 + P^2 = 0,6724$$

Assim sendo, já foram propostos

- Um modelo de referência que dita o comportamento desejado para o processo
- Uma equação que permite o cálculo de $y_{est}(t)$ e consequentemente a estimação dos parâmetros b_0, b_1, a_1 e a_2

- Equações que associam os parâmetros do processo controlado aos parâmetros do controlador $R(q)$, $S(q)$ e $T(q)$

Desta forma, já é possível implementar o STR. Como os resultados aqui obtidos serão gerados a partir de uma simulação, os valores de $y(t)$ serão fornecidos pela equação de diferenças

$$y(t) = 1.6065y(t-1) - 0.6065y(t-2) + 0.1065u(t-1) + 0.0902u(t-2); \qquad (2.149)$$

obtida a partir da transformada z inversa da função de transferência $G(z)$.

A entrada do processo deve excitar todos os seus modos de forma que seja garantida a convergência dos parâmetros do modelo de estimação para seus valores reais (Narendra; Annaswamy, 1987). A utilização de uma entrada persistentemente excitada de ordem n permite a determinação de um modelo de estimação que possua até n parâmetros. É possível mostrar que um sinal periódico de período n é persistentemente excitado de ordem n e pode neste caso ser utilizado como entrada (Astrom; Wittenmark, 1995). O conceito de excitação persistente está inclusive relacionado à robustez dos sistemas adaptativos (Narendra; Annaswamy, 1987)

Foi escolhida como referência uma onda quadrada (entrada periódica), que garante a convergência dos quatro parâmetros que precisam ser determinados. Os resultados obtidos são apresentados a seguir.

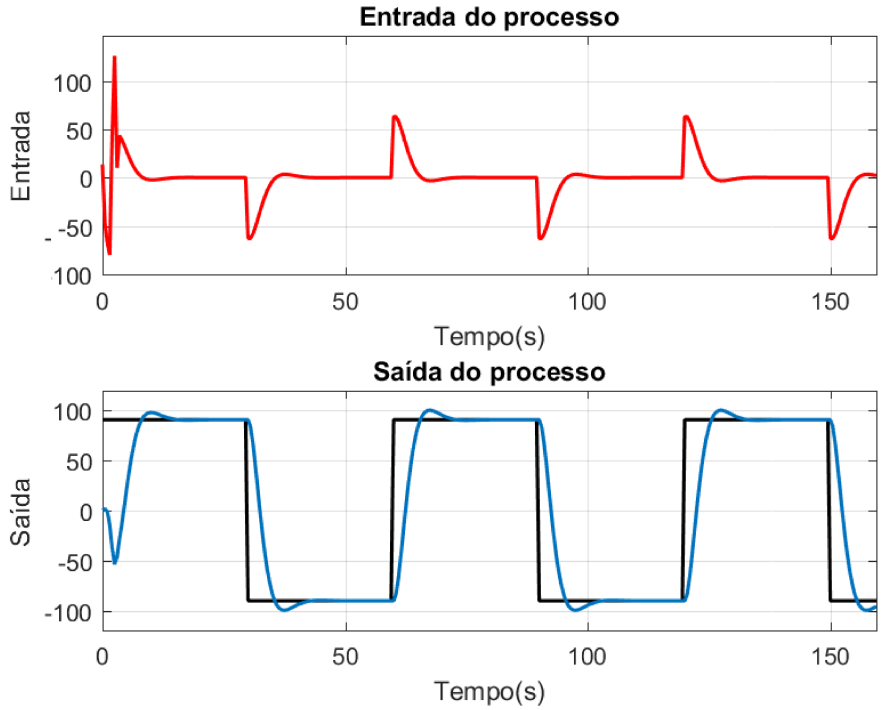

Figura 2.22 – Evolução da entrada e da saída do processo controlado.

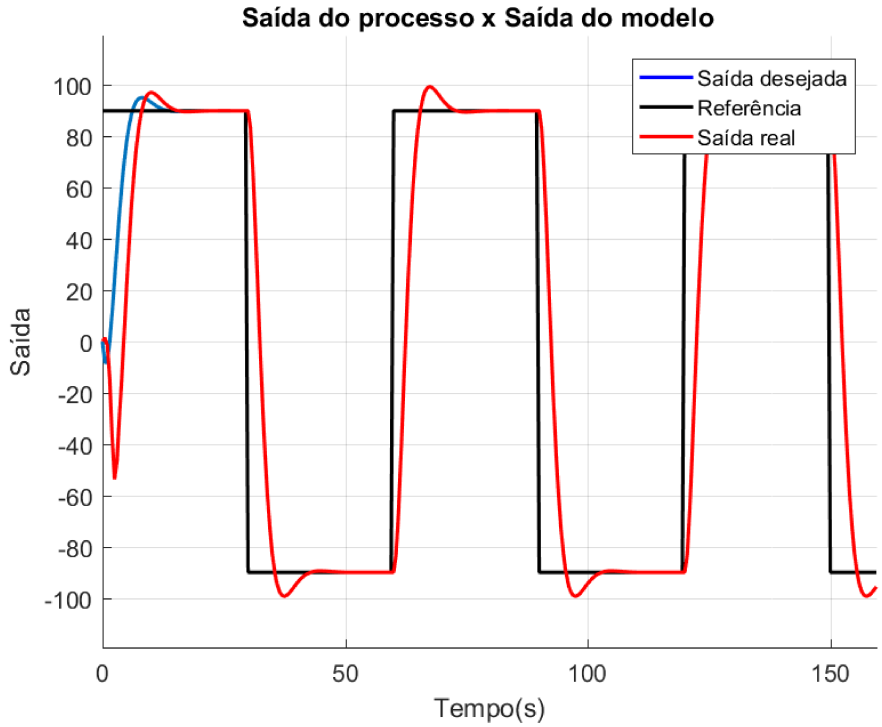

Figura 2.23 – Comparação entre a saída desejada $y_m(t)$ obtida a partir do modelo, a saída do processo controlado $y(t)$, e o sinal de referência $u_c(t)$.

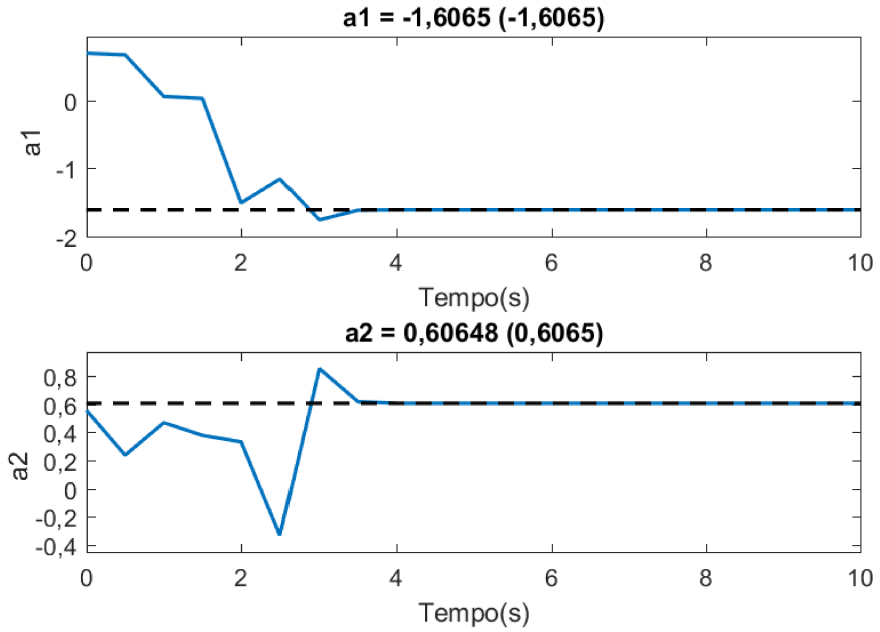

Figura 2.24 – Evolução dos parâmetros a_1 e a_2.

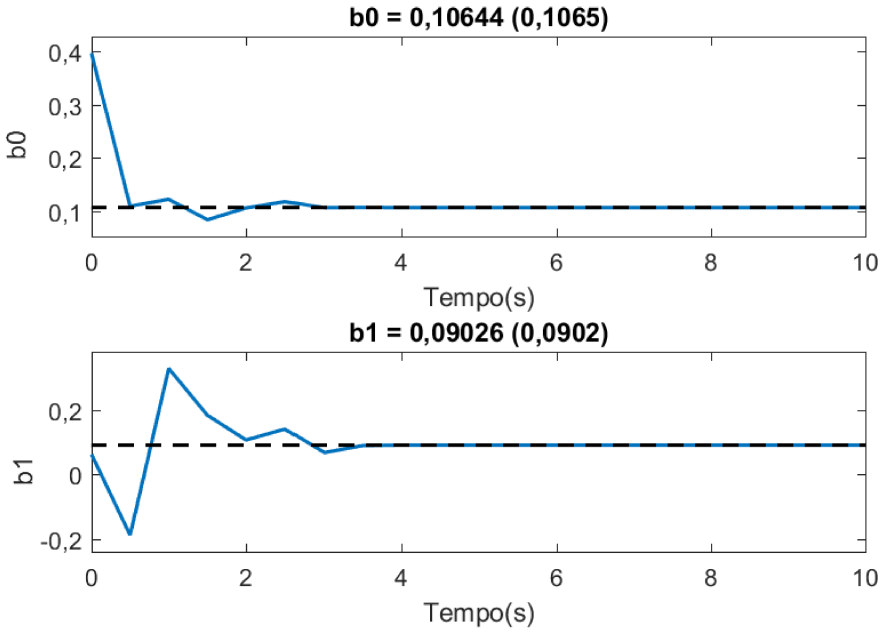

Figura 2.25 – Evolução dos parâmetros b_0 e b_1.

As Figs. 2.24 e 2.25 mostram a evolução dos parâmetros a_1, a_2, b_0 e b_1. Acima de cada um dos gráficos se encontra o valor final de cada parâmetro e, entre parênteses, o valor que o respectivo parâmetro deveria assumir para que $y(t)$ fosse igual a $y_{est}(t)$ (valor este representado em linhas tracejadas). As linhas contínuas representam os valores que a_1, a_2, b_0 e b_1 assumem ao longo da simulação. Cabe ressaltar que a estimação de todos os parâmetros convergiu de forma a garantir que $y(t)$ e $y_{est}(t)$ assumissem, ao fim da simulação, valores praticamente iguais.

A Fig. 2.23 mostra como os valores da saída $y(t)$ do sistema ficaram próximos de $y_m(t)$ depois de alguns segundos de simulação. Lembre-se que o controlador projetado a partir do Método do Posicionamento dos Polos deve garantir que isso aconteça.

Além disso, é interessante observar que no início da simulação há uma pequena discordância entre os valores de $y(t)$ e $y_m(t)$. De fato, neste momento os parâmetros do processo, estimados a cada iteração, ainda não convergiram. Como a ação de controle depende diretamente destes parâmetros, até que os seus valores convirjam, $u(t)$ e, consequentemente, $y(t)$ sofrerão variações bruscas, como pode ser observado na Fig. 2.22.

CAPÍTULO III

METODOLOGIA

Esta seção trata da implementação dos conceitos visto até o momento. O sistema a ser controlado será aqui apresentado assim como os esquemas de controle utilizados e as simulações desenvolvidas ao longo da execução deste trabalho.

O objetivo principal deste trabalho é implementar uma técnica de controle que possibilite o ajuste da trajetória de um robô autônomo de forma que este se comporte como desejado sem a necessidade de um ajuste prévio. Em outras palavras, deve ser implementado um controlador que seja capaz de ajustar seus parâmetros com base em um dado comportamento desejado. Para alcançar este objetivo adota-se um STR. A trajetória de referência escolhida aqui foi uma linha reta. Assim sendo, após a implementação do controlador, o robô deve ser capaz de se manter em uma linha reta mesmo que perturbações lhe sejam impostas.

3.1 Desenvolvimento Teórico

3.1.1 Análise do sistema a ser controlado e de seus componentes

O robô utilizado neste trabalho, que será referenciado ao longo do texto como *seguidor* (Fig. 3.1), foi construído com peças do kit **LEGO Mindstorms NXT®** especificamente para o desenvolvimento deste trabalho. É composto basicamente por elementos estruturais (barras, rodas e pinos), um CLP, responsável pela leitura dos sensores e pelo acionamento dos servomotores, e uma bússola. Cada um destes componentes será detalhado a seguir.

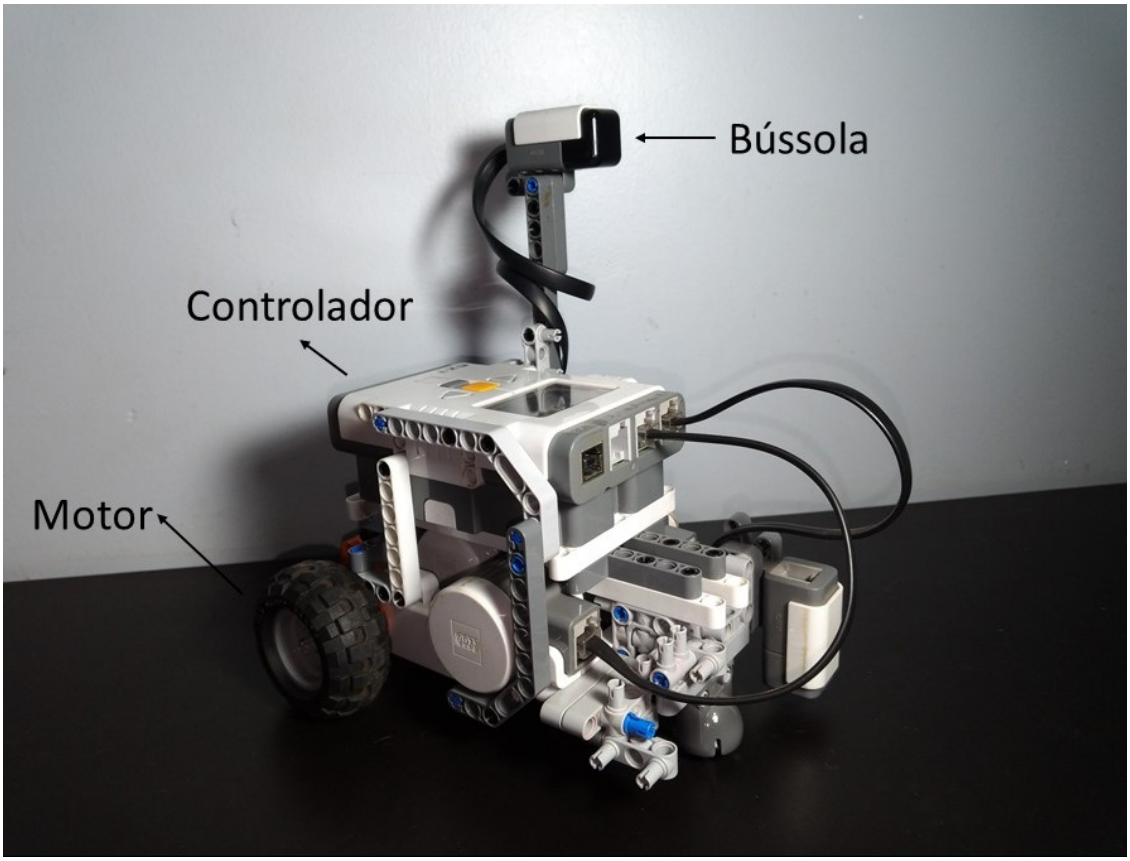

Figura 3.1 – Robô autônomo que terá sua trajetória ajustada a partir da implementação do controlador adaptativo proposto neste trabalho.

A bússola acoplada ao robô, mostrada na Fig. 3.2, permite que o mesmo se oriente. O valor lido a partir dela indica a direção do seguidor em relação ao norte. O funcionamento da bússola é exemplificado na Fig. 3.3. Quando o sensor está diretamente apontado para o norte, ele retorna um valor nulo, enquanto que, quando rotacionado um ângulo de α graus, ele retorna um valor α. Em (a) o valor retornado pela bússola seria 0, em (b) 30, em (c) 90 e em (d) 330. Note que quando a bússola dá uma volta completa o valor lido a partir dela volta a ser zero (HiTechnic, 2018).

Como dito anteriormente, a trajetória escolhida neste trabalho foi uma linha reta. Então, para garantir que o seguidor acompanhe esta trajetória, o controle deve ser realizado de forma a garantir que o valor da bússola não se altere à medida que o robô se move, mesmo na presença de perturbações. Note que, neste caso, a referência (*setpoint*) do controlador é um valor para a bússola, ou seja, uma direção.

Figura 3.2 – Bússola eletrônica compatível com o kit NXT. Produzida pela HiTechnic®.

.

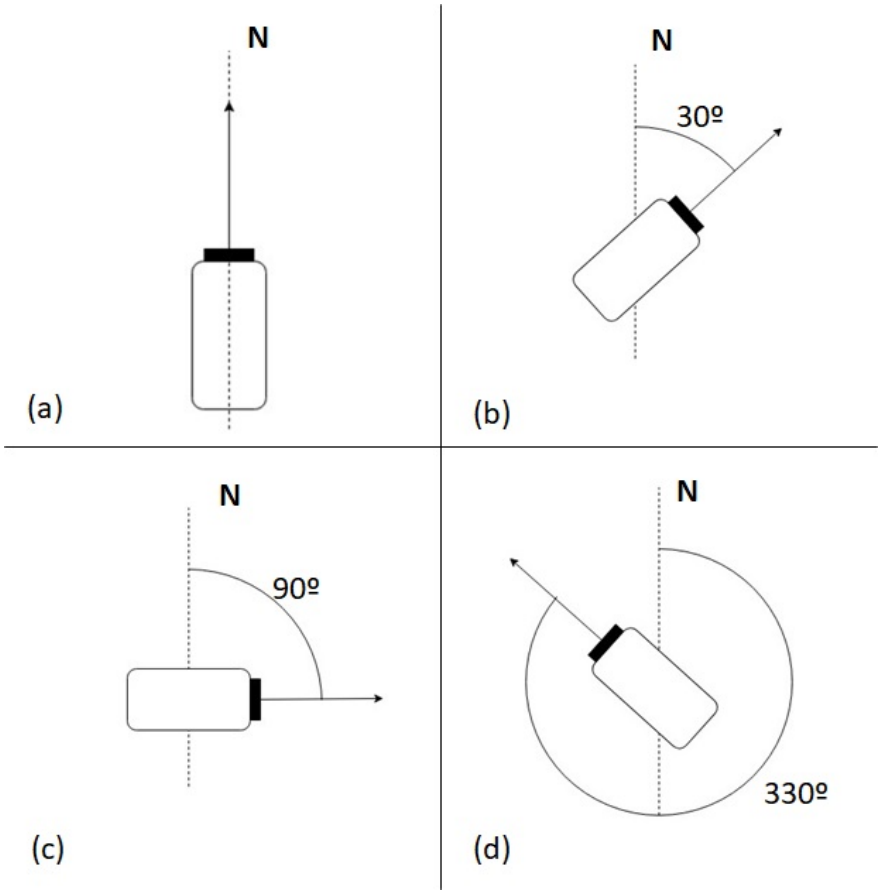

Figura 3.3 – Diagrama que ilustra o funcionamento da bússola.

Os servomotores por sua vez possibilitam que o robô se locomova. Na Fig. 3.4 é possível observar o servomotor utilizado. O seguidor apresentado na Fig. 3.1 possui dois

servomotores que são controlados pelo seu CLP. O controle adequado destes servomotores permite que o robô se mova para frente, para trás e vire para a direita e para a esquerda. A programação do CLP é feita em Java e diversos métodos permitem que a potência entregue a cada um dos servomotores seja controlada. Neste trabalho foi utilizado o método `setPower(int power)` que recebe como parâmetro a fração da potência máxima que deve ser entregue a cada servomotor. O parâmetro `power` deve ser um valor inteiro entre 0 a 100. Quando `power` vale 0 o servomotor não se move, e quando ele é igual a 100 o servomotor gira em sua velocidade máxima (leJOS, 2018).

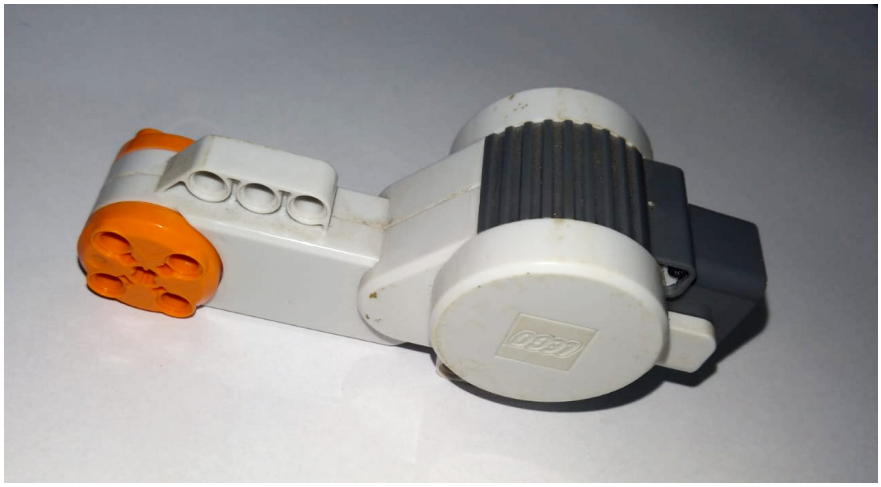

Figura 3.4 – Servomotor do kit LEGO Mindstorms NXT®.

.

Observe que a potência dos servomotores é a variável de entrada do sistema a ser controlado enquanto o valor da bússola é a variável de saída. As técnicas de controle apresentadas até o momento podem ser aplicadas somente a sistemas SISO (*Single Input Single Output*), ou seja, que possuam apenas uma entrada e uma saída. No entanto, o seguidor possui, a princípio, duas entradas, já que dois servomotores precisam ser controlados. Para resolver este problema, será considerada como entrada do sistema a diferença entre a potência dos dois servomotores. Desta forma, será atribuído a um dos servomotores uma potência de $40 + u(t)$ e ao outro uma de $40 - u(t)$, como mostrado na Fig. 3.5.

Note que, se ao invés de $40 \pm u(t)$ fosse atribuída aos servomotores a potência $50 \pm u(t)$, o seguidor se moveria mais rapidamente em linha reta, uma vez que sua direção tivesse sido ajustada. No entanto, pequenos ajustes, neste caso, alterariam sua trajetória mais

significativamente. Então, foi possível avaliar experimentalmente que a potência de $40 \pm u(t)$ garante que a resposta do seguidor não seja nem muito lenta e nem muito rápida.

Desta forma, quando a entrada do sistema é nula, os dois servomotores giram com a mesma potência e o seguidor se movimenta para frente. Se a ação de controle for positiva, o seguidor se move para a direita e se ela for negativa, ele se move para a esquerda, já que a variação de $u(t)$ fará com que um servomotor gire mais que o outro. A Fig. 3.6 ilustra como a variação da potência dos servomotores permite que o seguidor faça curvas.

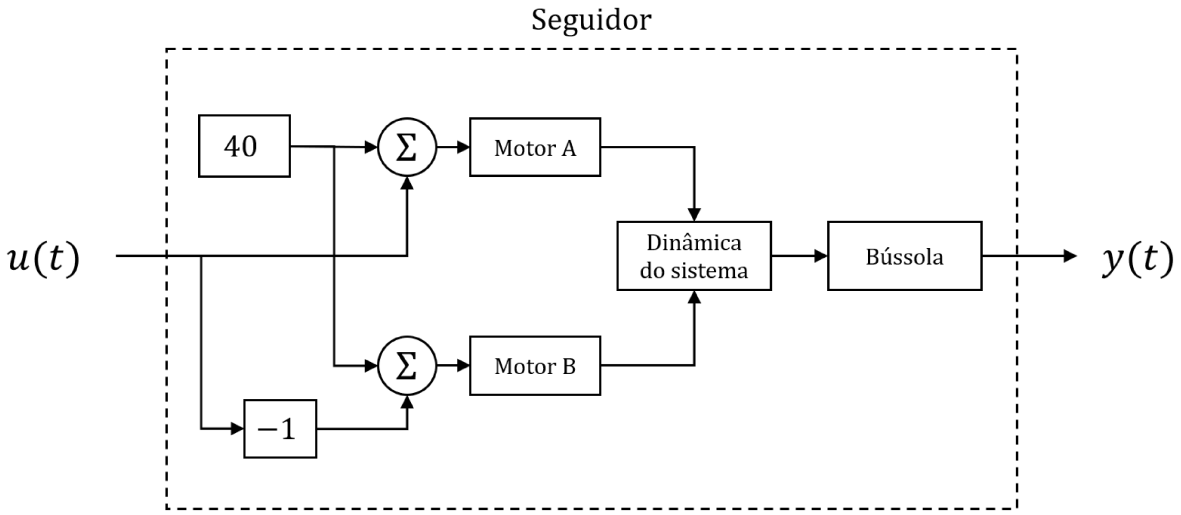

Figura 3.5 – Diagrama que ilustra o funcionamento do seguidor, evidenciando sua entrada u e sua saída y.

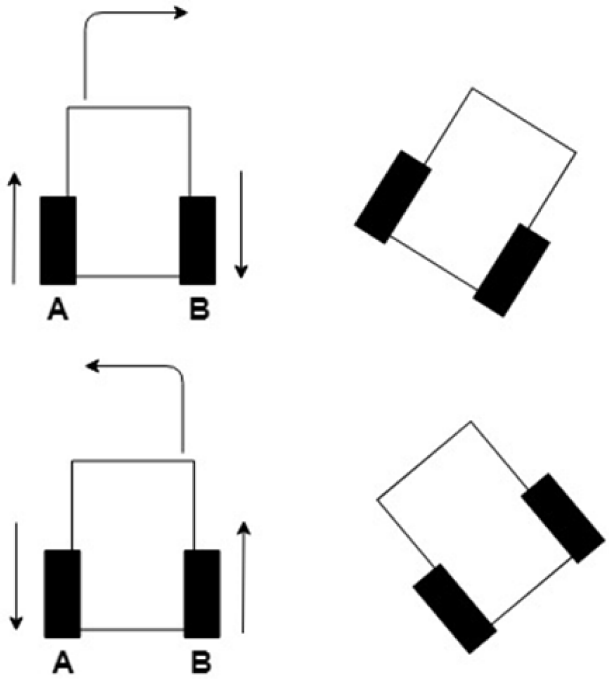

Figura 3.6 – Diagrama que ilustra como a variação da potência dos servomotores é capaz de fazer com que o seguidor se mova para a esquerda ou para direita.

Como foi dito anteriormente, o CLP (Fig. 3.7) é responsável pelo processamento dos dados advindos dos sensores e pelo acionamento e alimentação dos servomotores. Em cada CLP podem ser conectados 4 sensores e 3 servomotores. Toda a lógica de processamento, que determina o que o robô deverá fazer, é embarcada no CLP e baseada na programação em Java. Fornece também suporte para comunicação Bluetooth com outros dispositivos e se conecta ao computador (tanto para envio de programas quanto de dados) através de uma porta USB 2.0. Pode ser alimentado por uma bateria de 10 V ou por pilhas.

A maior limitação do CLP se encontra em sua capacidade de processamento. Seu componente principal é um microcontrolador de 32 bits ARM7TDMI-core Atmel AT91SAM7S256 e este possui apenas 256 KB de memória *flash*, 64 KB de memória RAM e *clock* de 48 MHz (LEGO, 2006). O baixo poder de processamento deste CLP impossibilita que ele seja utilizado em aplicações muito complexas.

Uma possibilidade, neste caso, seria utilizar a nova versão do kit LEGO Mindstorms®, o kit LEGO Mindstorms EV3®. O CLP deste novo kit, apresentado na Fig. 3.8, possui, diferente de seu antecessor, um processador ARM9 com 16 MB de memória *flash*, 64 MB de memória RAM,

e um *clock* de 300 MHz (LEGO, 2018). A princípio, seria plausível pensar que o processamento deixaria de ser um problema caso o kit LEGO Mindstorms EV3® fosse utilizado, dado a evolução considerável que seu CLP apresentou em relação ao CLP do kit LEGO Mindstorms NXT®. No entanto, o CLP EV3 roda um sistema operacional baseado em Linux, que lida com interrupções de *hardware*, escalonamento de processos, gerenciamento de memória, organização de um sistema de arquivos, dentre outras tarefas que exigem a dedicação de recursos computacionais.

Experimentos utilizando tanto o CLP NXT quanto o CLP EV3, mostraram que utilizando o CLP do kit LEGO Mindstorms EV3® foi possível implementar um sistema de controle com tempo de atraso de 80 ms, enquanto que, utilizando seu predecessor, foi possível atingir até 50 ms (que foi inclusive o tempo de amostragem adotado neste trabalho).

Figura 3.7 – CLP (ou *intelligent brick*) do kit LEGO Mindstorms NXT®.

Figura 3.8 – CLP (ou *intelligent brick*) do kit LEGO Mindstorms EV3®. (Fonte: `https://shop.lego.com/en-US/product/EV3-Intelligent-Brick-45500`. Acessado em 28 de Dezembro de 2018)

3.1.2 Estudo do controlador a ser implementado

Este trabalho propõe a implementação de um controlador STR. O diagrama apresentado na Fig. 3.9 ilustra como o controle de trajetória do seguidor foi realizado. A estimação dos parâmetros do processo foi realizada através da implementação Método RLS enquanto o cálculo dos parâmetros do controlador foi baseado no Método do PP, ambos introduzidos na Seção 2.

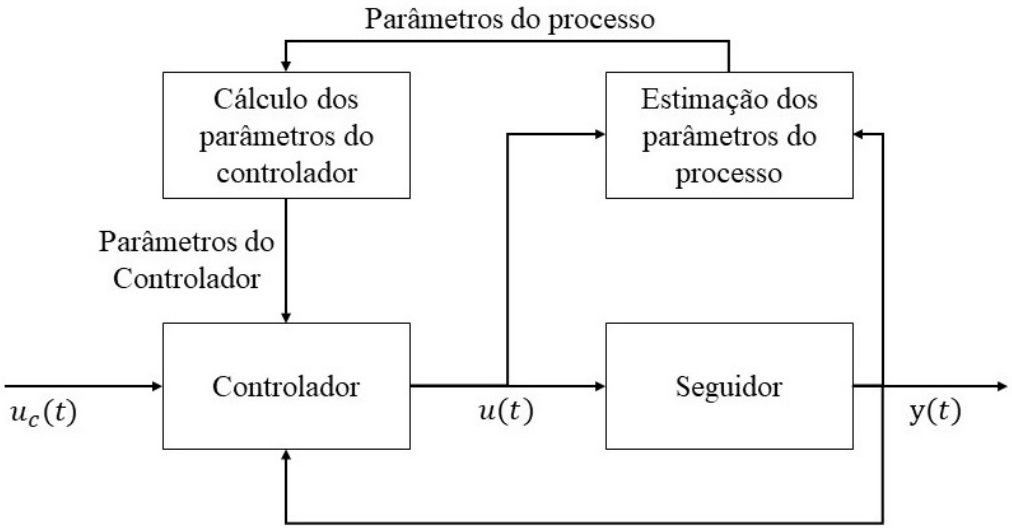

Figura 3.9 – Diagrama que ilustra como o STR pode ser implementado no controle de trajetória do seguidor.

Como foi discutido na Seção 2.3 não há apenas uma maneira de implementar o método do posicionamento dos polos. Para decidir se o controlador utilizado efetuaria ou não o cancelamento dos zeros da planta, simulações foram realizadas com o objetivo de avaliar o comportamento do seguidor em uma série de possíveis cenários sem que fosse necessário de fato utilizar o robô. Para executar as simulações, foi necessário um modelo matemático que representasse o seguidor, neste caso uma equação de diferenças, de forma que o sistema de controle pudesse ser implementado como mostrado na Fig. 3.10.

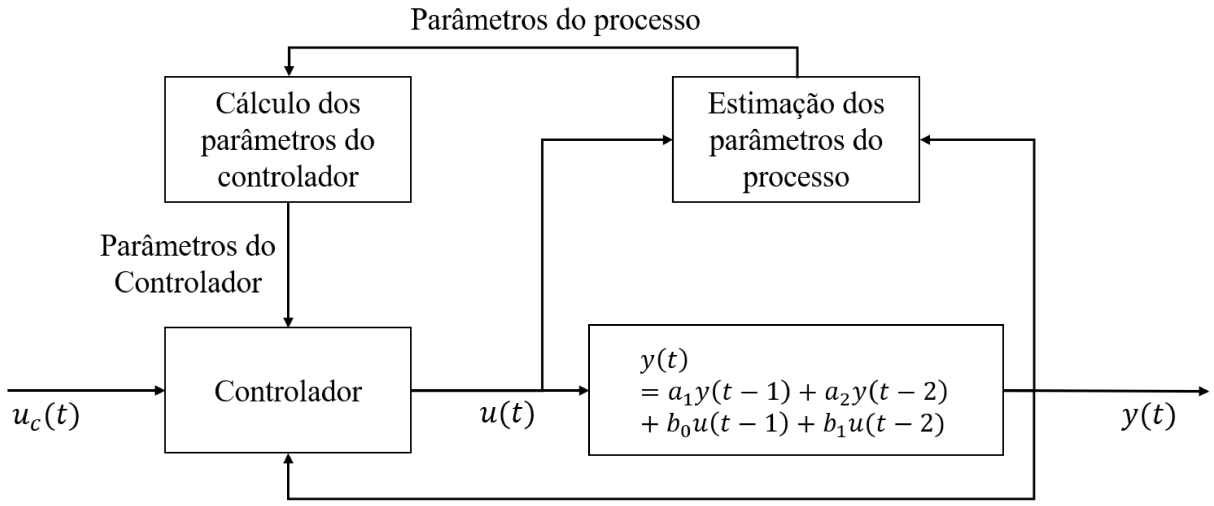

Figura 3.10 – Diagrama que ilustra como o controlador STR proposto neste trabalho foi implementado em simulação.

Note que a cada iteração, o valor da saída do sistema $y(t)$ pode ser obtido a partir da computação da equação de diferenças

$$y(t) = a_1y(t-1) + a_2y(t-2) + b_0u(t-1) + b_1u(t-2) \tag{3.1}$$

onde os valores de a_1, a_2, b_0 e b_1 ainda precisam ser determinados. Neste trabalho, o sistema a ser controlado foi representado por um modelo de segunda ordem assim como este, e a determinação dos seus parâmetros, realizada empiricamente, será discutida na seção seguinte. O modelo de segunda ordem foi escolhido porque foi possível comprovar experimentalmente que este é suficiente para representar o comportamento do seguidor.

3.2 Desenvolvimento Prático

3.2.1 Estimação dos parâmetros da planta do sistema a ser controlado

Para simular o comportamento do seguidor seja avaliado, é necessário encontrar a equação de diferenças que representa a dinâmica do sistema. Com este propósito, dois experimentos foram realizados com o intuito de gerar dados que permitissem que o modelo fosse estimado

utilizando o RLS. Em cada um dos experimentos uma onda quadrada foi aplicada na entrada do sistema e a aquisição da saída foi realizada por meio de uma conexão USB estabelecida entre o seguidor e um computador. É importante ter em mente que os dados experimentais devem ser enviados ao computador à medida que são adquiridos, uma vez que a memória limitada do CLP não permite que estes sejam armazenados.

Como o tempo de amostragem utilizado nesta aquisição foi de 50 ms, o controlador aqui proposto deverá ser implementado considerando também um tempo de amostragem de 50 ms. O diagrama da Fig. 3.11 exemplifica como os experimentos foram realizados. Dois conjuntos de dados foram coletados, um para estimação dos parâmetros do processo e outro para validação do modelo obtido.

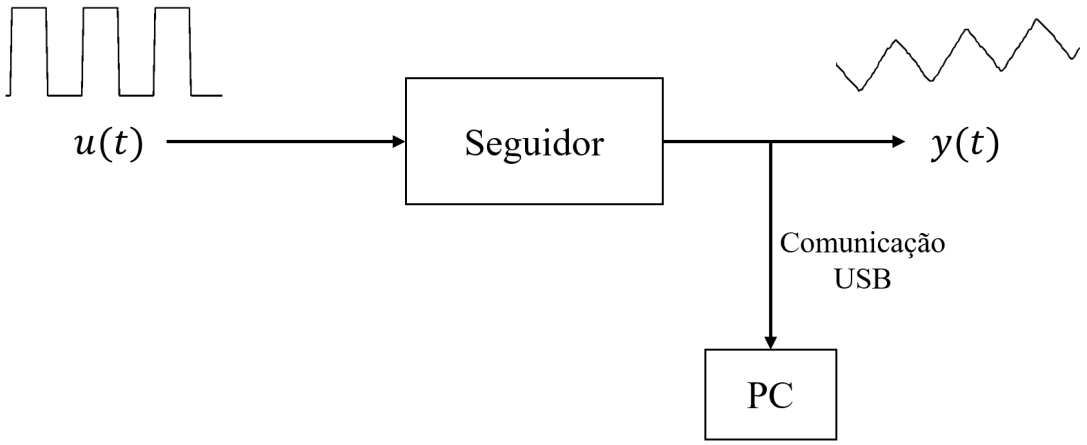

Figura 3.11 – Diagrama que ilustra como foram coletados os dados utilizados na estimação (e posterior validação) dos parâmetros da equação de diferenças que descreve o comportamento do seguidor.

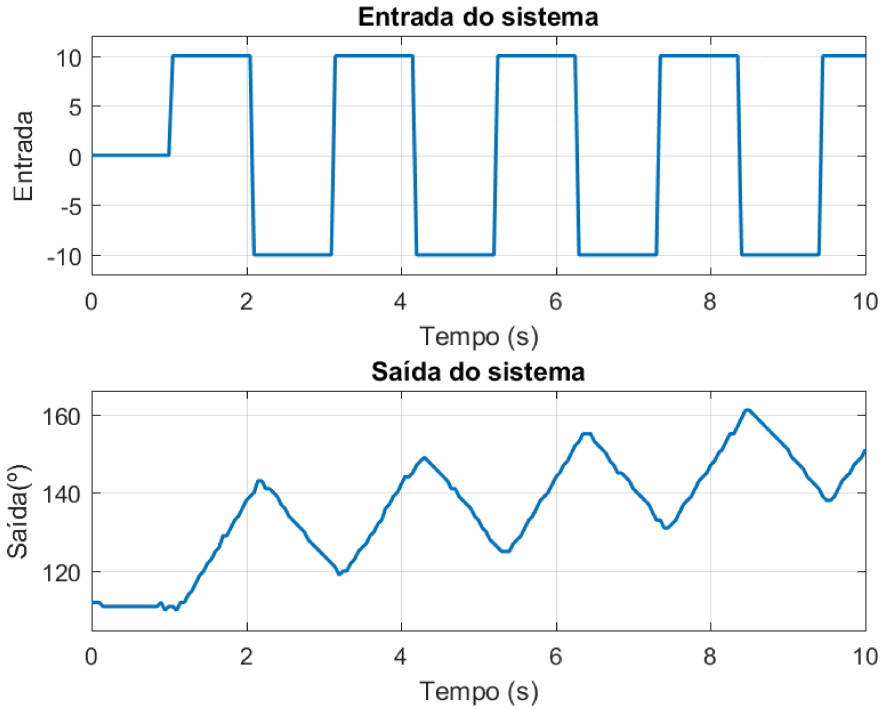

Figura 3.12 – Dados coletados para estimação do modelo matemático que descreve o comportamento do seguidor.

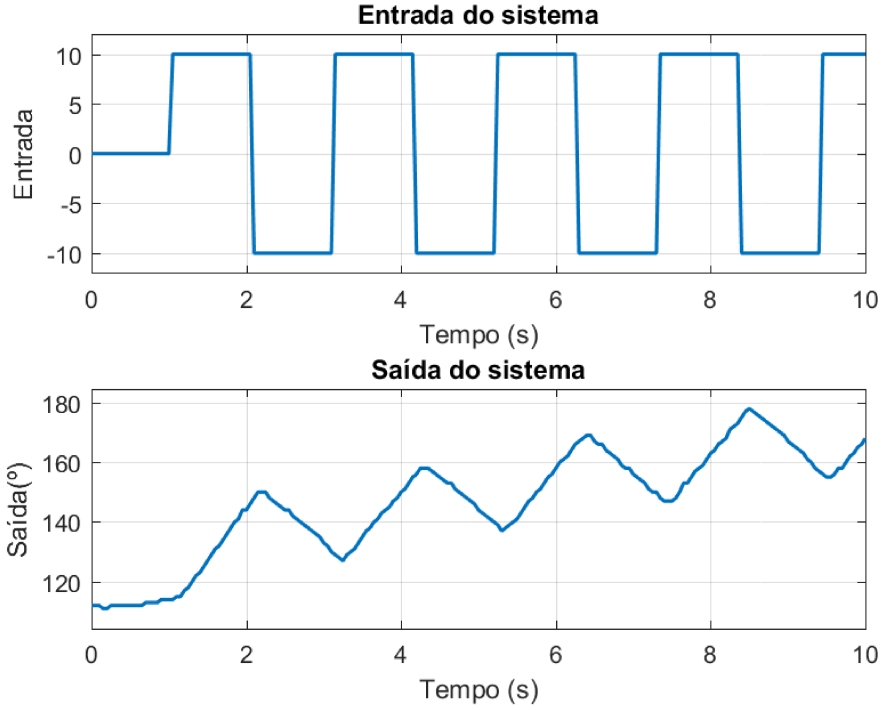

Figura 3.13 – Dados coletados para validação do modelo matemático obtido a partir da aplicação do RLS.

Os dados utilizados na estimação dos parâmetros da equação de diferenças que descreve o comportamento do seguidor, e na validação do modelo obtido, estão representados nos gráficos das Figs. 3.12 e 3.13, respectivamente.

A equação de diferenças (3.2) representa o comportamento do seguidor, e foi obtida *offline* a partir da implementação do RLS. Os resultados obtidos por meio da estimação podem ser observados nos gráficos da Fig. 3.14 e mostram que o comportamento do seguidor pode de fato ser representado por um modelo de segunda ordem. Os códigos responsáveis pela computação dos resultados apresentados a seguir podem ser encontrados no Apêndice I.

O modelo obtido foi o seguinte

$$y(t) = 1,0002y(t-1) + 0,0008y(t-2) - 0,0244u(t-1) + 0,1503u(t-2) \qquad (3.2)$$

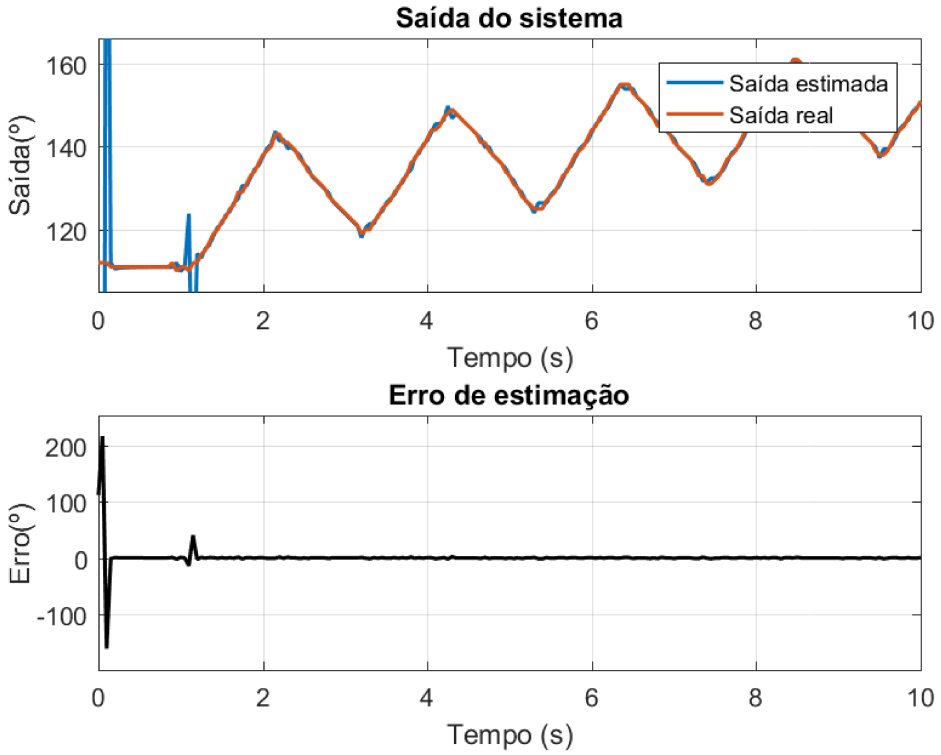

Figura 3.14 – No primeiro gráfico é apresentada uma comparação entre os valores obtidos experimentalmente e os estimados através da implementação do RLS. O segundo gráfico apresenta a evolução do erro de estimação.

A validação deste modelo foi realizada através da utilização de uma das ferramentas

de identificação do Matlab®, o *ident*, cuja interface é apresentada na Fig. 3.15. Para utilizar esta ferramenta é preciso inicialmente lembrar que a equação de diferenças (3.2) representa uma função de transferência no domínio z com polos em 1,001 e -0,0008 e zero em 6,16

$$G(z) = \frac{-0,0244z + 0,1503}{z^2 - 1,0002z - 0,0008} \tag{3.3}$$

Esta função de transferência deve então ser inicializada no *workspace* do Matlab® e importada para o *ident*. Em seguida, os dados a serem utilizados na validação devem ser também importados. Então, basta clicar na caixa de seleção *Model output* para que a comparação entre os dados de validação e a saída do modelo seja realizada.

Será exibido um valor entre 0% e 100% que representa o ajuste do modelo, que neste caso foi de $88,9\%$. Este valor, conhecido como *fit*, é calculado com base no erro médio quadrático normalizado entre as saída do modelo e os dados experimentais. É determinado a partir de (3.4), onde $\boldsymbol{y_{exp}}$ e $\boldsymbol{y}$ são vetores que contém, respectivamente, os dados experimentais e a saída do modelo e $\overline{\boldsymbol{y_{exp}}}$ é a média dos dados em $\boldsymbol{y_{exp}}$ (MathWorks, 2019)).

$$fit = 1 - \frac{\|\boldsymbol{y_{exp}} - \boldsymbol{y}\|}{\|\boldsymbol{y_{exp}} - \overline{\boldsymbol{y_{exp}}}\|} \tag{3.4}$$

A comparação entre a saída real do sistema, obtida experimentalmente, e a saída do modelo, é apresentada na Fig. 3.16.

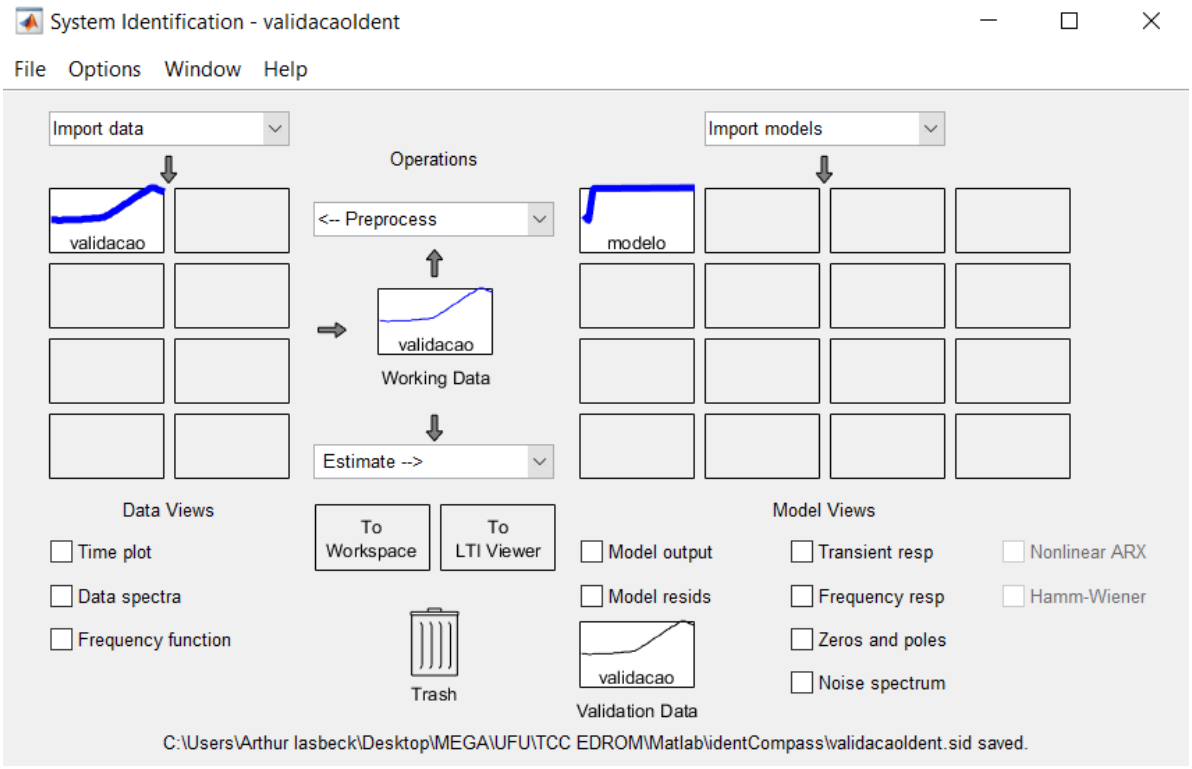

Figura 3.15 – Interface do *ident* (ferramenta de identificação de sistemas do Matlab®).

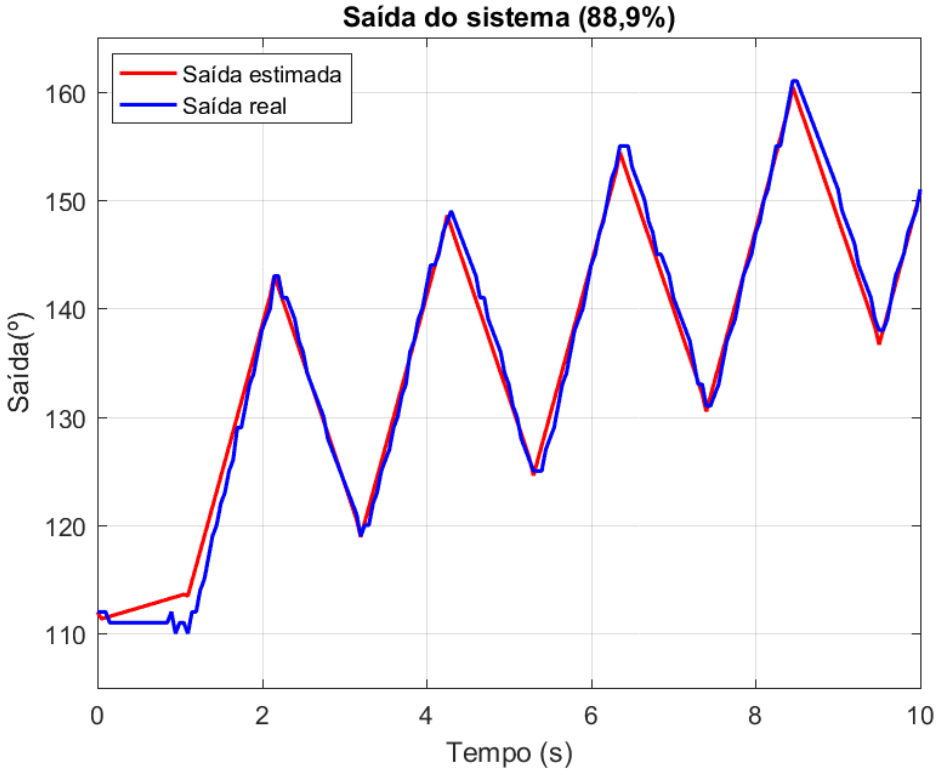

Figura 3.16 – Resultados da validação do modelo matemático obtido a partir da aplicação do RLS.

3.2.2 Implementação das simulações

Uma vez identificado um modelo é possível simular o comportamento do sistema em malha fechada.

Para decidir se o zero da planta deve ou não ser cancelado, o comportamento do sistema foi avaliado por meio de simulações computacionais, utilizando (3.2).

Observe que o mesmo procedimento para o cálculo dos parâmetros do controlador utilizado na Seção 5.2 pode ser aqui empregado, uma vez que, neste caso, o sistema a ser controlado foi também descrito por um modelo de segunda ordem

$$y(t) = -a_1 y(t-1) - a_2 y(t-2) + b_0 u(t-1) + b_1 u(t-2) \tag{3.5}$$

O controlador aqui proposto é definido pela equação

$$R(q)u(t) = T(q)u_c(t) - S(q)y(t) \tag{3.6}$$

onde os polinômios $R(q)$, $T(q)$ e $S(q)$ devem ser determinados pelo implementação do PP, como foi feito na Seção 5.2.

Para que os parâmetros do controlador sejam determinados, é necessário que seja proposto um modelo de referência $y_m(t)/u_c(t) = B_m(q)/A_m(q)$ que possibilite a obtenção da saída desejada $y_m(t)$ a partir de uma dada referência $u_c(t)$. Este modelo deve ser construído a partir da especificação dos requisitos de desempenho, ou seja, dos valores do sobressinal (*overshoot*) M_p e do tempo de acomodação t_a. Nas simulações apresentadas a seguir foram adotados

$$M_p = 0,1\ \%$$

$$t_a = 1\ s$$

O diagrama de blocos que representa o funcionamento de um STR em simulação já foi apresentado na Fig. 3.10. O fluxograma que ilustra a implementação das simulações realizadas neste trabalho, é mostrado na Fig. 3.17. Os algoritmos empregados na obtenção dos resultados

apresentados a seguir, podem ser encontrados no Apêndice I. Nos parágrafos seguintes, serão detalhadas cada uma das etapas do fluxograma da Fig. 3.17.

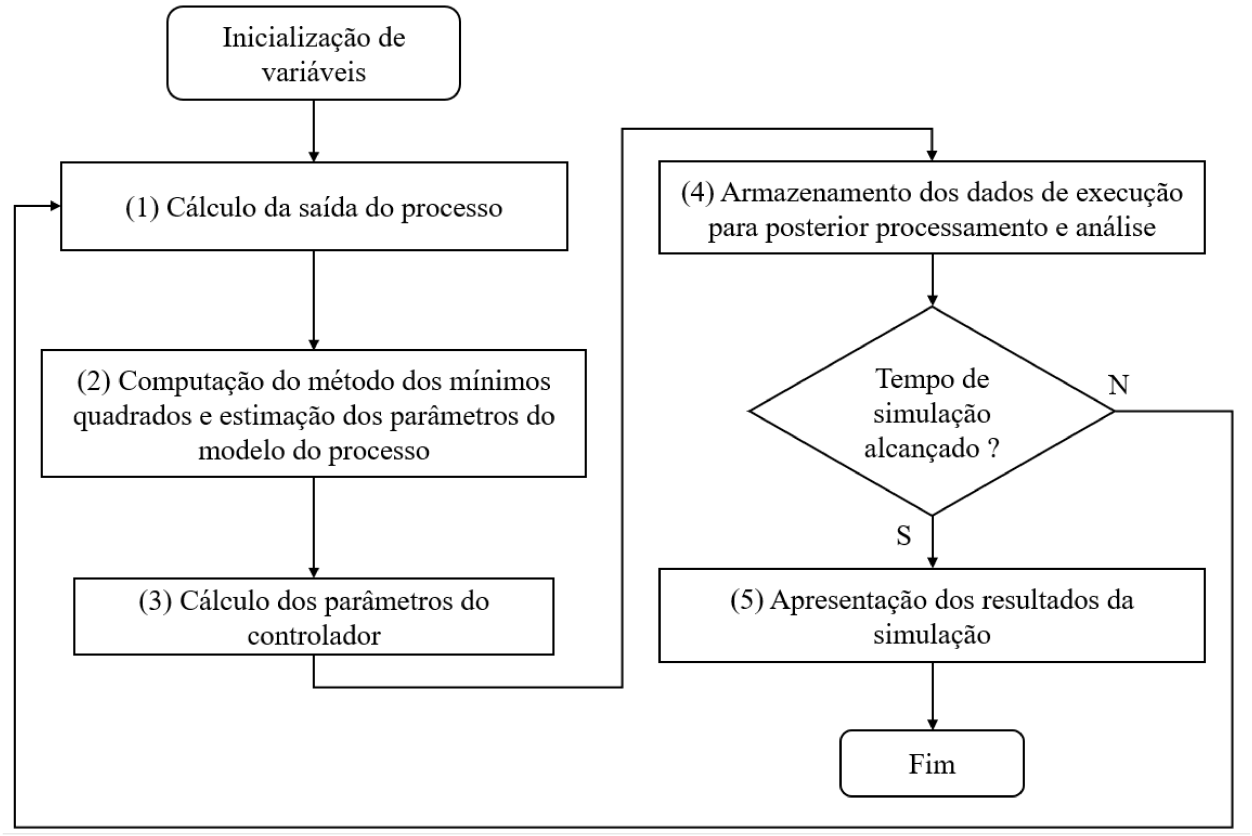

Figura 3.17 – Fluxograma que apresenta as etapas da implementação de um STR. Perceba que, neste caso, a saída é calculada através da utilização de um modelo, o que indica que este fluxograma corresponde a uma simulação e não a uma implementação real.

Na etapa (1) o valor de $y(t)$ é obtido através da computação da equação de diferenças introduzida em (3.2), obtida experimentalmente. Observe que, caso o STR fosse implementado no controle de um sistema real, esta etapa poderia ser substituída por *Aquisição da saída do processo*, onde a saída da planta seria não mais calculada através da utilização de um modelo, mas sim obtida a partir da utilização de um sensor.

Na etapa (2) um modelo que represente o funcionamento da planta é estimado *online* ao longo das iterações. Esta estimação é baseada no RLS e é a partir dos resultados aqui obtidos que os parâmetros do controlador serão posteriormente determinados. Foi possível concluir

empiricamente que o sistema a ser controlado pode ser representado por um modelo de segunda ordem, logo, deverão ser determinados, a partir da estimação, os parâmetros de (3.5).

Neste caso a_1 e a_2 são acompanhados de um sinal negativo, porém, a escolha dos sinais de a_1, a_2, b_0 e b_1 não afeta o resultado da estimação. Além disso, é possível representar esta equação de diferenças em sua forma matricial

$$y_{est}(t) = \varphi^T(t-1)\theta \tag{3.7}$$

onde

$$\begin{aligned} \theta &= \begin{bmatrix} a_1 & a_2 & b_0 & b_1 \end{bmatrix}^T \\ \varphi^T(t-1) &= \begin{bmatrix} -y(t-1) & -y(t-2) & u(t-1) & u(t-2) \end{bmatrix} \end{aligned} \tag{3.8}$$

Uma vez que os parâmetros do processo controlado tenham sido determinados na etapa (2), já é possível, com base neles, realizar o cálculo dos parâmetros do controlador. Este cálculo é realizado na etapa (3), com base no PP, de forma que o sistema se comporte como desejado. Em outras palavras, os valores de $R(q)$, $S(q)$ e $T(q)$ serão, nesta etapa, determinados. É importante ressaltar que a determinação dos parâmetros do processo controlado foi realizada a partir da implementação do RLS, o que significa que eles são atualizados a cada iteração. Isto garante que, caso a dinâmica do sistema sofra alguma alteração, esta será detectada e os parâmetros do controlador serão recalculados com base em novas estimações. É isso que faz deste controlador um controlador adaptativo.

Por fim, na etapa (4) os dados gerados ao longo da execução, como o valor da saída do sistema $y(t)$, o valor de sua referência $u_c(t)$, o valor estimado para sua saída a partir do RLS $y_{est}(t)$, os valores dos parâmetros a_1, a_2, b_0 e b_1, e a ação de controle $u(t)$, são armazenados para que sejam apresentados graficamente ao fim da simulação.

Para simular o comportamento do seguidor, será implementada na etapa (1) a equação de diferenças introduzida em (3.2), que representa o comportamento deste sistema, e foi obtida empiricamente, como descrito anteriormente. Nas Figs. 3.18, 3.19, 3.20 e 3.21 são mostrados os resultados obtidos a partir da implementação, em simulação, de um STR que foi projetado

através da utilização do método do posicionamento dos polos, e que não cancela os zeros da planta.

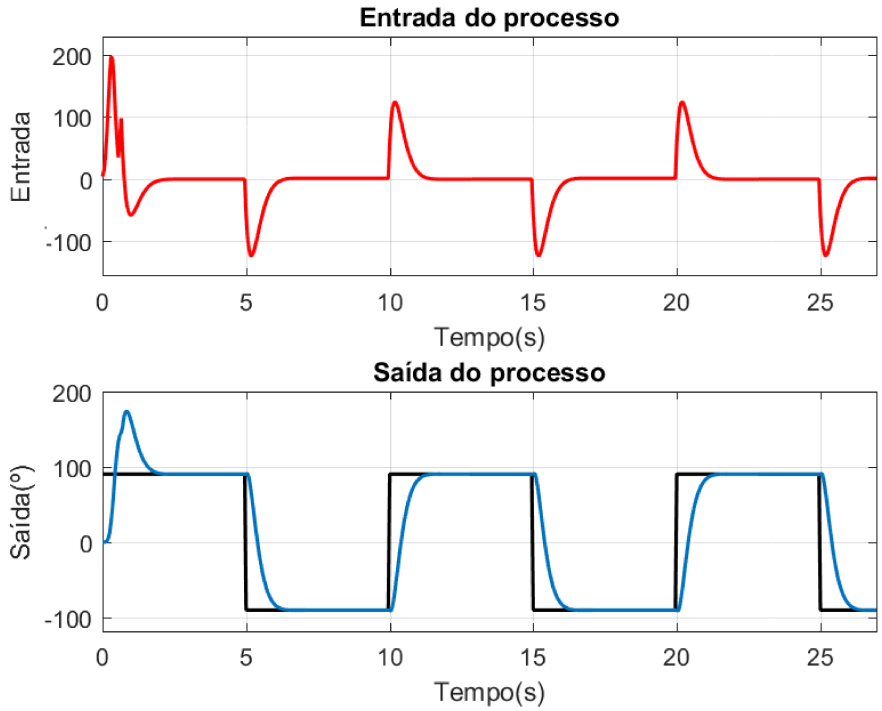

Figura 3.18 – Evolução da entrada do sistema $y(t)$ e da ação de controle $u(t)$, obtidos a partir da implementação de um controlador STR sem cancelamento dos zeros da planta.

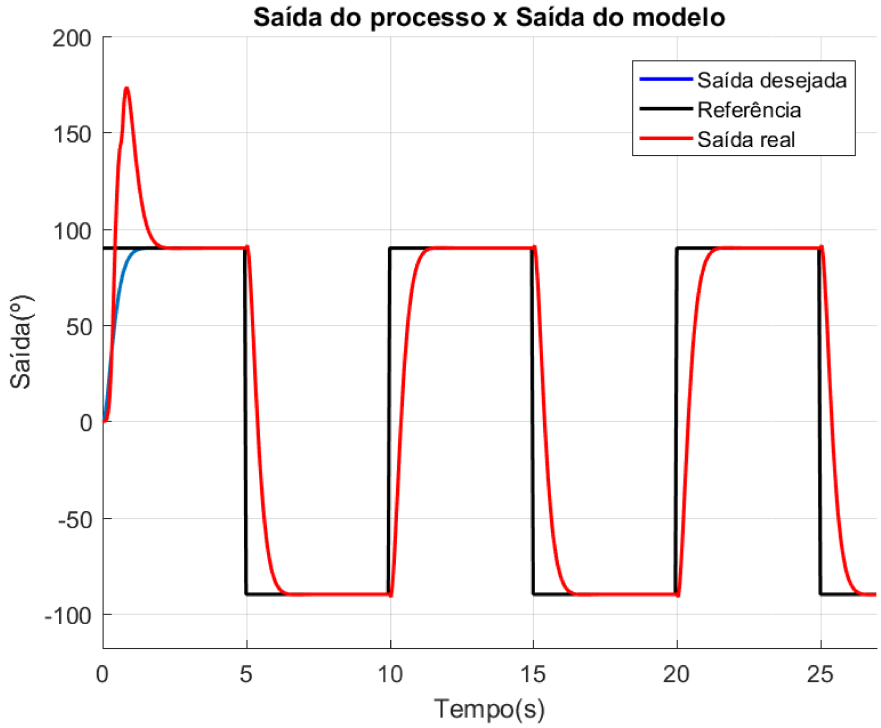

Figura 3.19 – Comparação entre a saída desejada $y_m(t)$ obtida a partir do modelo, a saída real $y(t)$ do processo controlado, e o sinal de referência $u_c(t)$.

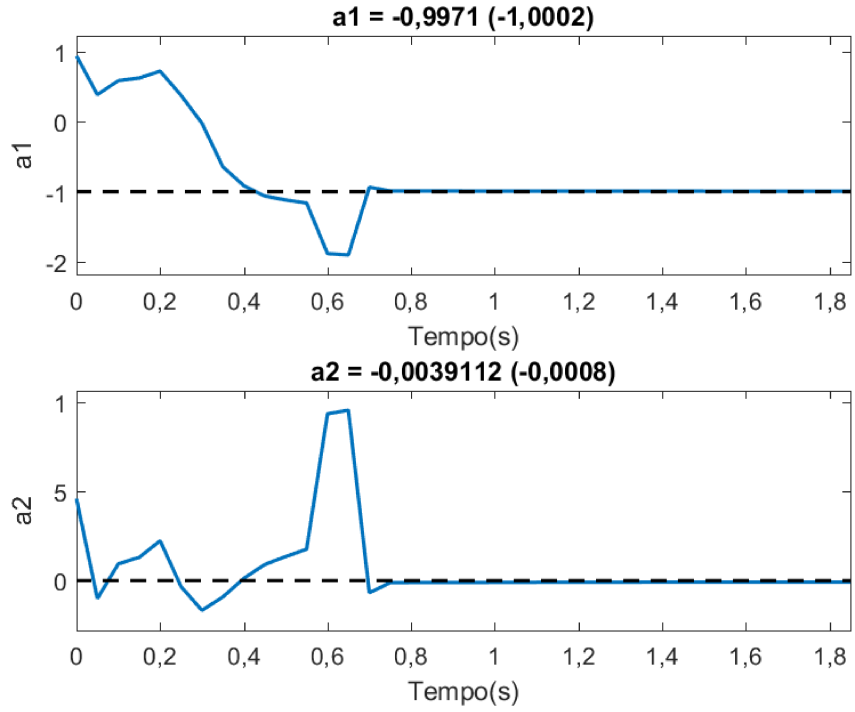

Figura 3.20 – Evolução dos parâmetros a_1 e a_2.

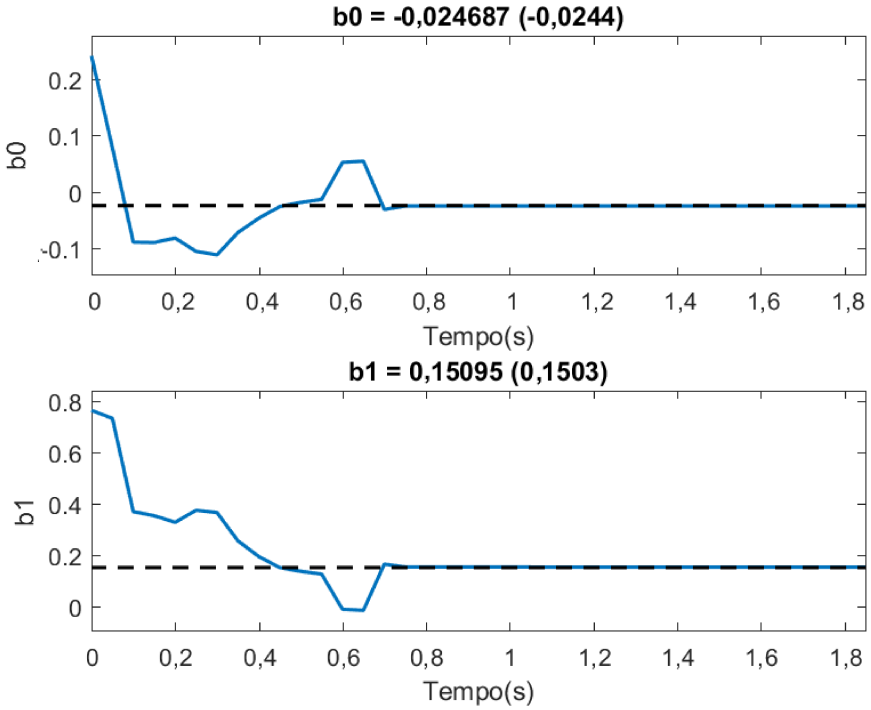

Figura 3.21 – Evolução dos parâmetros b_0 e b_1.

É interessante observar que os resultados são semelhantes àqueles apresentados na Seção 2.4. Nos primeiros momentos da simulação, em que os parâmetros do modelo ainda estão sendo ajustados, é possível observar que a ação de controle e a saída do sistema assumem valores bastante oscilatórios. No entanto, depois que os valores de a_1, a_2, b_0 e b_1 convergem a saída do sistema $y(t)$ se iguala à saída desejada $y_m(t)$ e assim permanece até o fim da simulação.

O gráfico da Fig. 3.22 mostra a saída e a entrada (ação de controle) obtidos a partir da implementação de um STR projetado através da utilização do PP, que cancela os zeros da planta. É possível observar que o sistema diverge depois de pouco menos de 1 segundo. Para que um zero seja cancelado, um polo deve ser posicionado sobre ele. Como o zero do sistema é instável, cancelá-lo requer a alocação de um polo também instável, o que explica a divergência tanto na entrada quanto na saída.

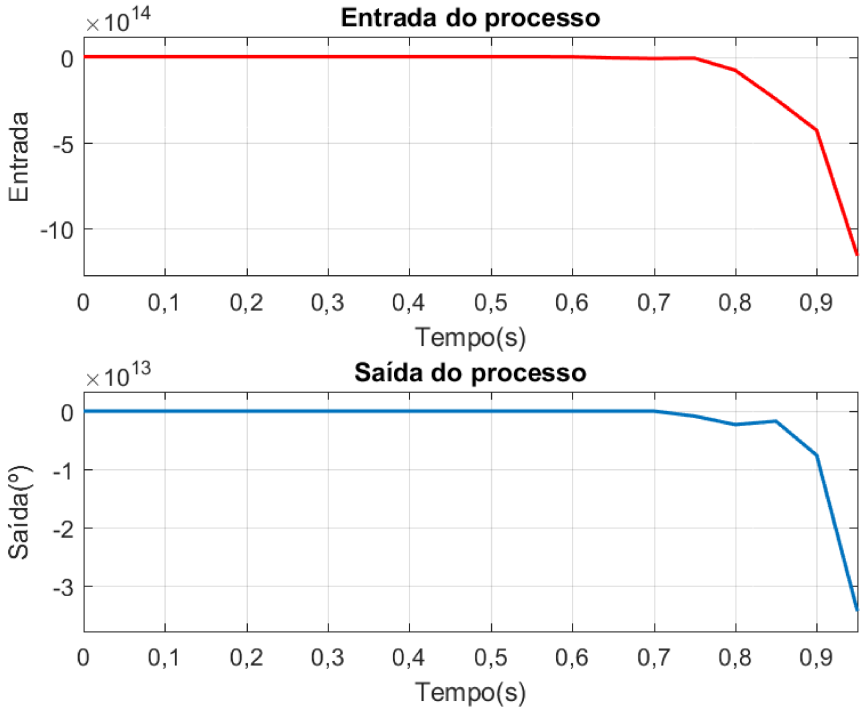

Figura 3.22 – Evolução da entrada do sistema $y(t)$ e da ação de controle $u(t)$, obtidos a partir da implementação de um controlador STR (neste caso os zeros da planta foram cancelados).

Para fazer com que a simulação fique ainda mais próxima da realidade, é possível considerar a saturação dos atuadores do sistema controlado, neste caso, os servomotores do seguidor. Lembre-se que a potência dos servomotores é determinada por um parâmetro que varia

de 0 a 100, e que para um dos servomotores este parâmetro vale $40 + u(t)$ enquanto para outro vale $40 - u(t)$. Desta forma, $u(t)$ não pode ser maior que 60, uma vez que, quando ele for igual a 60, $40 + u(t)$ valerá 100, o que corresponde à potência máxima que pode ser enviada ao servomotor. Os resultados da simulação em que a saturação dos atuadores do seguidor foi levada em consideração são apresentados nas Figs. 3.23 e 3.24.

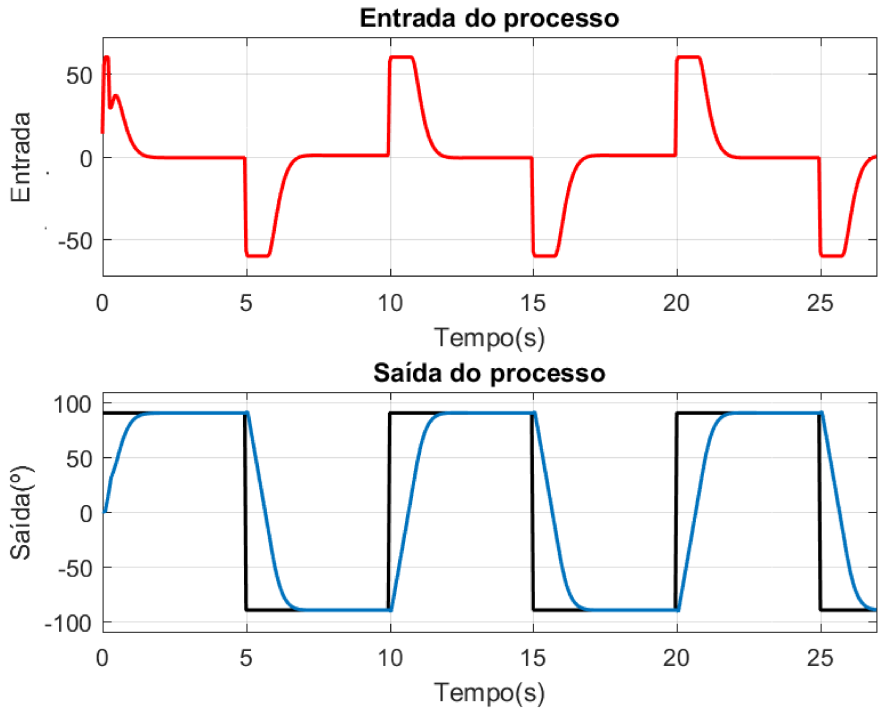

Figura 3.23 – Evolução da entrada do sistema $y(t)$ e da ação de controle $u(t)$, obtidos a partir da implementação de um controlador STR (neste caso os zeros da planta não foram cancelados), considerando um tempo de acomodação excessivamente pequeno.

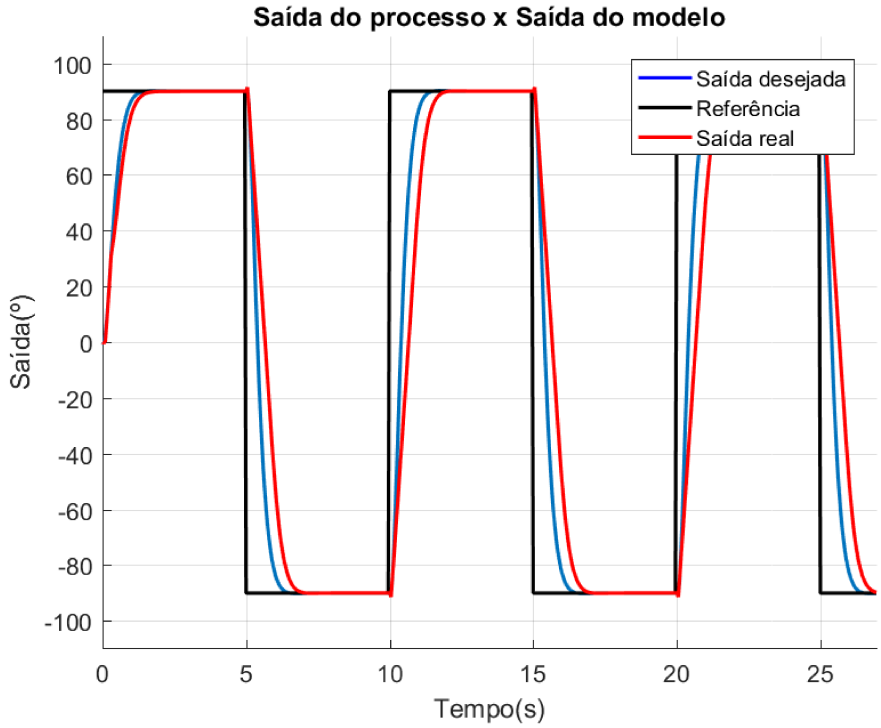

Figura 3.24 – Comparação entre a saída desejada $y_m(t)$ obtida a partir do modelo, a saída real $y(t)$ do processo controlado, e o sinal de referência $u_c(t)$.

No gráfico da Fig. 3.23 que mostra a evolução temporal da entrada do processo (ação de controle), é possível observar que o valor desta não ultrapassa o 60. Lembre-se que, para que o sistema se comporte exatamente como desejado, é necessário que a ação de controle seja bem maior que 60. Como isso não é possível, o sistema deverá apresentar um comportamento apenas próximo do desejado.

A Fig. 3.24 mostra uma comparação entre a saída do processo e a saída desejada. É possível observar que o processo apresentou um tempo de acomodação um pouco maior do que o desejado. Isso se deve às limitações físicas dos atuadores.

3.3 Implementação do STR no robô

Para empregar o STR no seguidor, as etapas do fluxograma da Fig. 3.17 foram implementadas no CLP. Cabe salientar que a etapa *Cálculo da saída do processo* deve ser substituída, neste caso, por *Aquisição da saída do processo*, uma vez que o valor de $y(t)$ é agora obtido a partir das leituras da bússola.

No entanto, durante a implementação do controlador STR utilizando o próprio seguidor, há uma limitação de processamento que não pode ser ignorada. Como foi discutido anteriormente, o poder de processamento do CLP do kit LEGO Mindstorms NXT® é bastante limitado. Lembre-se que para implementar um controlador do tipo STR é necessário realizar a estimação *online* dos parâmetros da planta do processo a ser controlado, o que exige que as matrizes $P(t)$, $K(t)$, $\theta(t)$ e $\varphi(t)$ sejam computadas. Esta computação depende do cálculo de produtos matriciais, o que demanda um processamento considerável, uma vez que, para o modelo aqui proposto (uma equação de diferenças com quatro parâmetros a serem determinados), $P(t)$ é uma matriz 4×4, enquanto $\theta(t)$ e $\varphi(t)$ são matrizes 4×1.

Ao computar, por exemplo, o produto entre $P(t)$ e $\varphi(t)$ (para a obtenção de $K(t)$), 28 operações se fazem necessárias. O cálculo dos elementos da matriz $P(t)$ a partir de $P(t-1)$ exige 88 operações, enquanto que para que $\theta(t)$ seja atualizado, são necessárias mais 16. São ao todo 132 operações (de soma e multiplicação) que precisam ser realizadas a cada iteração, com tempo de amostragem (tempo máximo que pode durar uma iteração) de 50 ms.

Considerando que o CLP seja responsável por processar somente o controle de trajetória, as seguintes etapas devem ser executadas a cada iteração

1. Aquisição do valor da bússola
2. Estimação dos parâmetros do modelo do processo
3. Cálculo dos parâmetros do controlador
4. Ajuste da potência dos servomotores

A execução de todas estas etapas leva cerca de 45 ms. A etapa 2 é responsável por aproximadamente 40% deste tempo, cerca de 18 ms. O cálculo dos parâmetros do controlador pode ser realizado em apenas 0,7 ms, enquanto as demais etapas (1 e 4) precisam de 26 ms para serem executadas. Observe que a estimação foi responsável por boa parte do gasto computacional envolvido na implementação do controlador STR e que quase não foi possível realizar todas as operações necessárias dentro do intervalo de tempo limite de 50 ms (tempo de amostragem). No entanto, é preciso ter em mente que ainda há muitas outras operações a serem executadas, sensores a serem lidos e servomotores que precisam ser controladores, sem contar

que a comunicação entre CLPs, que é quase sempre implementada, exige também um certo gasto computacional. Neste caso, provavelmente não será possível realizar as operações 1, 2, 3 e 4 em menos de 50 ms.

Para lidar com as limitações de processamento impostas pelo CLP, a implementação do controlador STR proposto neste trabalho foi dividia em duas etapas: *adaptação* e *controle*. Durante a adaptação, o CLP fica responsável somente pelo processamento das etapas 1, 2, 3 e 4, enquanto a referência do sistemas sofre mudanças bruscas em intervalos de tempo constantes, o que garante que a estimação apresente resultados satisfatórios. Uma vez que os valores dos parâmetros do modelo do processo tenham convergido e não sejam mais detectadas alterações bruscas em sua saída, a adaptação chega ao fim.

Já que a dinâmica do seguidor não se altera significativamente com o passar do tempo, não há porque prosseguir com a estimação dos parâmetros do processo uma vez que seus valores tenham convergido, já que, neste caso, eles não mais se modificariam com o passar das iterações. Assim sendo, uma vez que os valores dos parâmetros tenham convergido, a etapa 2 do fluxograma introduzido na Fig. 3.17 não se faz mais necessária.

Os parâmetros do controlador são determinados a partir dos parâmetros do processo. Uma vez que estes últimos tenham convergido, os valores dos parâmetros do controlador não mais se modificarão com o passar das iterações. Assim sendo, na etapa de *controle*, a estimação é suspensa, como ilustra a Fig. 3.25, e o controlador deixa de se comportar como um controlador adaptativo. Note que, caso mudanças ocorram na dinâmica do seguidor, e consequentemente nos parâmetros do modelo que descreve o seu comportamento, uma nova adaptação se faria necessária, o que não vem a ser um grande problema, uma vez que a etapa de adaptação dura cerca de 10 segundos apenas.

O esquema adaptativo proposto é empregado neste caso somente na sintonização dos parâmetros do controlador. Esta abordagem ainda vai de encontro ao objetivo principal deste trabalho, que é implementar um sistema de controle que tenha seus parâmetros ajustados de forma automática para satisfazer certos requisitos de desempenho.

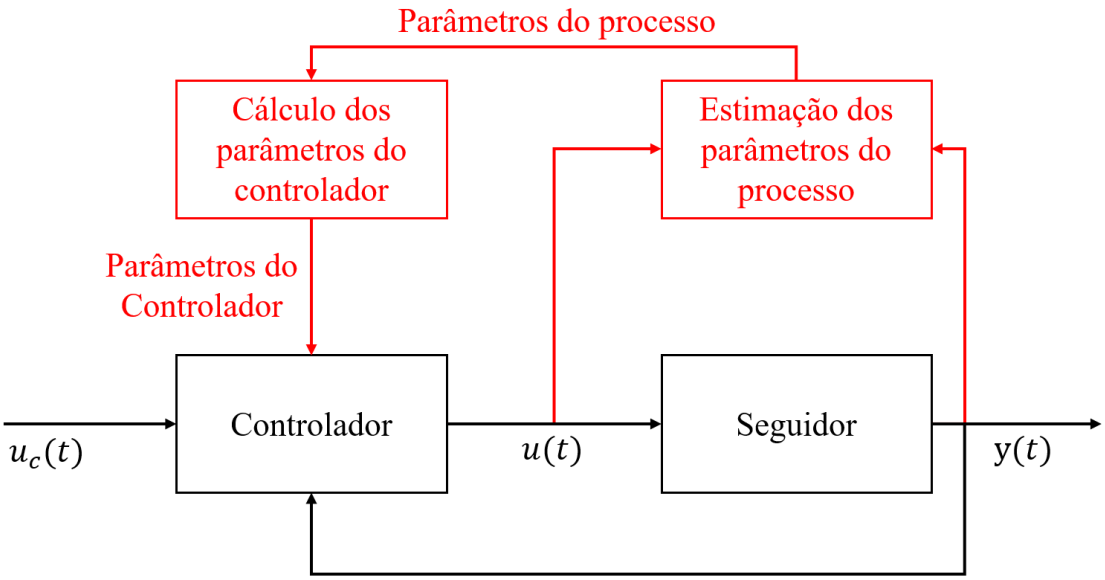

Figura 3.25 – Diagrama de blocos que destaca (em vermelho) as etapas da implementação do STR que são desconsideradas após a adaptação do controlador proposto neste trabalho.

CAPÍTULO IV

RESULTADOS E DISCUSSÕES

4.1 Discussão sobre a implementação Java do controle de trajetória

Para validar a implementação do STR proposto, diversos experimentos foram realizados e neste seção, três deles serão apresentados. Cabe ressaltar que dois experimentos realizados sob as mesmas condições apresentarão resultados distintos, uma vez que os valores dos parâmetros do processo controlado são sempre inicializados com valores aleatórios que variam entre 0 e 1, já que a princípio são desconhecidos. Esta abordagem torna mais genérica a implementação do STR.

A implementação em Java foi baseada no fluxograma da Fig. 3.17 apesar de algumas modificações terem se mostrado necessárias. É importante ter em mente que o controle de trajetória é processado dentro de um *loop*, assim como foi feito em simulação. Porém, durante sua execução, uma série de outras tarefas estão sendo simultaneamente realizadas, como já foi discutido anteriormente (comunicação com outros CLPs, leitura de sensores, controle de motores, processamento de dados, dentre outras). Logo, imaginando que o período de amostragem seja de 50 ms e o processamento das etapas necessárias ao controle de trajetória leve 30 ms, a execução não pode ser interrompida durante 20 ms, de forma que o tempo gasto no processamento de cada iteração seja de 50 ms, uma vez que isto poderia prejudicar o funcionamento de outras tarefas.

Desta forma, o método responsável pelo cálculo da ação de controle é invocado em

todas as iterações, porém, uma verificação de tempo é realizada de forma que a ação de controle só seja recalculada de fato se T segundos tiverem se passado desde sua ultima atualização. Se o período de amostragem for de 50 ms mas o processamento de cada iteração levar somente 1 ms, o método responsável pela computação da ação de controle retornará 50 vezes o mesmo valor, e somente depois de 50 ms, a ação de controle será recalculada. O objetivo desta abordagem é garantir um período de amostragem constante sem que a execução de outras tarefas seja prejudicada.

Os fluxogramas apresentados nas Figs. 4.2 e 4.1 mostram como o STR proposto neste trabalho foi de fato implementado no CLP do seguidor. Observe que o método responsável pela computação da ação de controle, cujo fluxograma é apresentado na Fig. 4.1, é também encarregado de realizar a estimação dos parâmetros do processo, uma vez que é a partir destes que os parâmetros do controlador são determinados. Cabe ressaltar que, depois de finalizada a adaptação, o mesmo método é utilizado na computação da ação de controle, porém, sem que a estimação dos parâmetros do processo seja realizada.

O fluxograma apresentado na Fig. 4.2 mostra como o *loop* responsável pelo controle de trajetória foi implementado. Observe que suas etapas são bastante semelhantes às do fluxograma introduzido na Fig. 3.17. Além disso, cabe ressaltar que a referência é alterada durante a etapa de adaptação para que a estimação dos parâmetros do processo apresente bons resultados. A mesma assume um valor constante uma vez que esta etapa tenha se encerrado, já que a trajetória a ser rastreada é uma linha reta. Os resultados apresentados a seguir mostram a evolução da etapa de adaptação e foram obtidos adotando-se como referência uma onda quadrada de amplitude igual a $80^{\underline{o}}$.

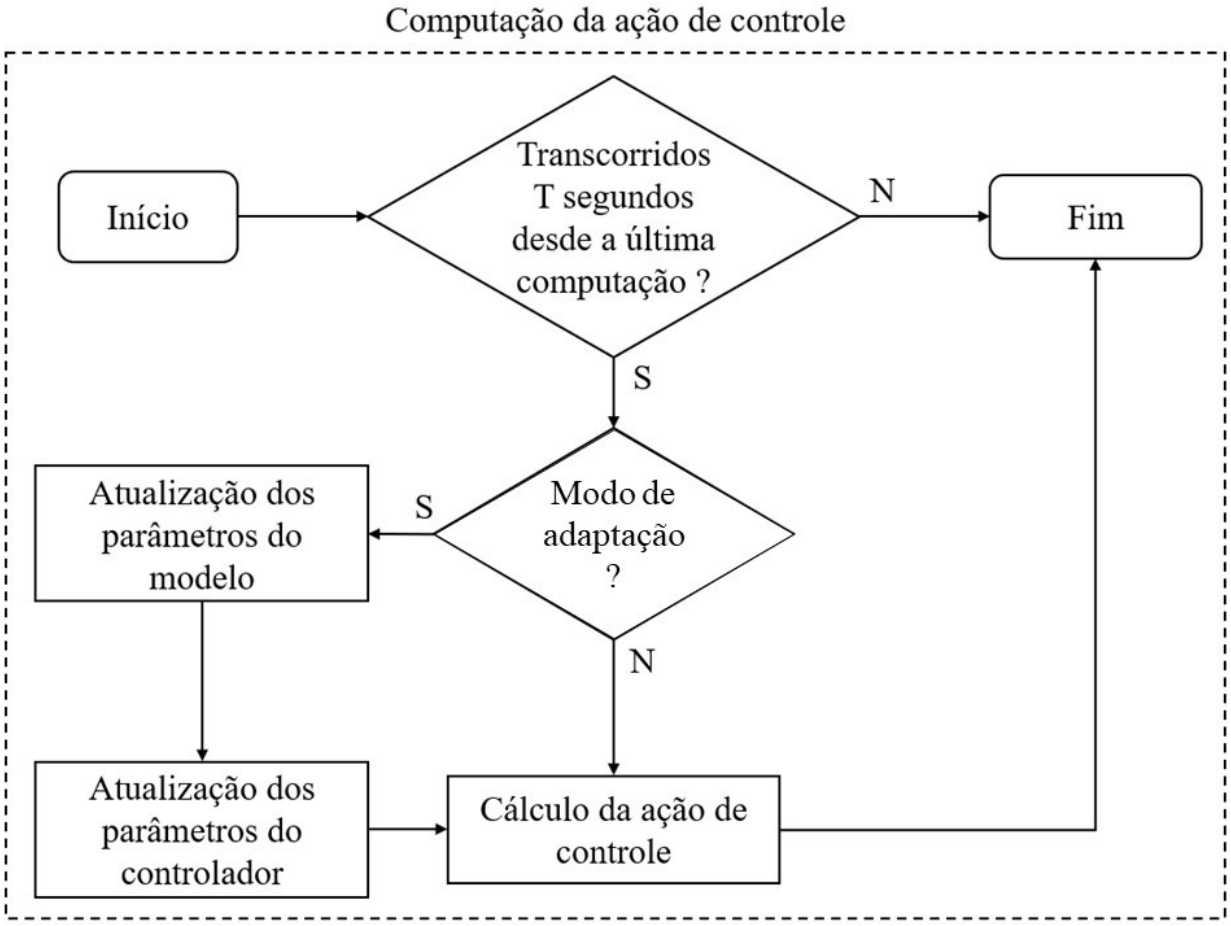

Figura 4.1 – Fluxograma que ilustra o funcionamento do método responsável pela computação da ação de controle.

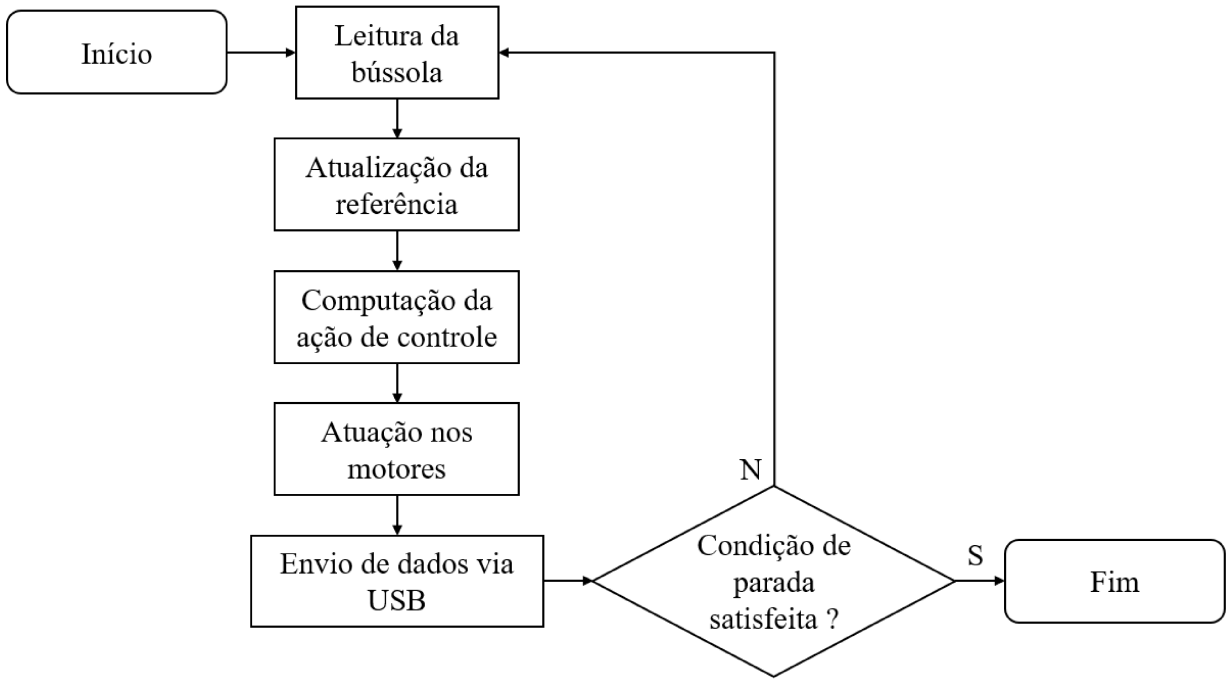

Figura 4.2 – Fluxograma que ilustra como o controlador STR proposto neste trabalho foi de fato implementado no CLP do robô cuja trajetória deseja-se controlar.

4.2 Análise dos resultados

A seguir serão apresentados alguns resultados obtidos a partir da implementação dos algoritmos discutidos na seção anterior. Para cada resultado serão avaliados tanto a saída do sistema $y(t)$, que neste caso corresponde ao valor da bússola, quanto a ação de controle $u(t)$, ao longo do tempo. Juntamente a $y(t)$ serão representados os valores de $y_m(t)$ e $u_c(t)$, para que seja possível avaliar se o sistema controlado apresenta o comportamento desejado e se sua saída acompanha de fato a referência.

O primeiro conjunto de dados que será aqui analisado se encontra representado nos gráficos das Figs. 4.3, 4.4 e 4.5. A Fig. 4.3 possibilita a comparação entre a saída do sistema $y(t)$ e a saída desejada $y_m(t)$, que foi definida a partir dos seguintes requisitos de desempenho

$$M_p = 0.1\%$$

$$t_a = 1s$$

Primeiramente, cabe ressaltar que a saída desejada não atende os requisitos de desempenho apesar dos valores de M_p e t_a serem bastante próximos do desejado. Essa divergência se deve ao método utilizado na discretização dos polos em malha fechada, a Transformada Bilinear de Tustin. Caso a expressão $z = e^{sT}$ fosse utilizada no mapeamento entre os planos s e z, os requisitos de desempenho seriam satisfeitos. No entanto, a discordância entre o comportamento do modelo de referência e o desejado é muito pequena e pode ser desconsiderada.

Observe que estes resultados mostram que a adaptação foi bem sucedida, uma vez que depois de apenas 8 segundos, $y(t)$ e $y_m(t)$ passam a assumir valores bastante próximos. Avaliando somente o intervalo de 8 a 12 segundos, apresentado na Fig. 4.5, é possível observar um erro médio de apenas 0,9377 graus (algo que nem sequer pode ser detectado pela bússola, cuja resolução é de 1 grau). O cálculo do erro foi realizado a partir da implementação da Equação 4.1.

$$e_{avg} = \frac{1}{n}\sum_{i=1}^{n} |y_m(t) - y(t)| \tag{4.1}$$

onde n é número de dados contido no intervalo analisado.

Além disso, com o intuito de validar os dados obtidos em simulação, apresentados na Fig. 4.6, é possível compará-los aqueles obtidos experimentalmente. Observe que, neste caso, as simulações devem levar em consideração as condições iniciais e as referências utilizadas no experimento a partir do qual foram obtidos os dados apresentados na Fig. 4.3. A semelhança entre os dados teóricos e experimentais mostra que as simulações representaram satisfatoriamente o comportamento do seguidor.

Por fim, vale ressaltar que em todos os experimentos a saída desejada $y_m(t)$ foi calculada a partir de condições iniciais nulas, o que facilita consideravelmente a implementação, e o que explica a diferença entre os valores iniciais de $y(t)$ e $y_m(t)$. Além disso, é interessante observar que, como foi visto em simulação, oscilações bruscas nos valores de $y(t)$ podem ser observadas durante os primeiros segundos de adaptação, uma vez que leva um certo tempo para que se verifique a convergência dos parâmetros do processo controlado.

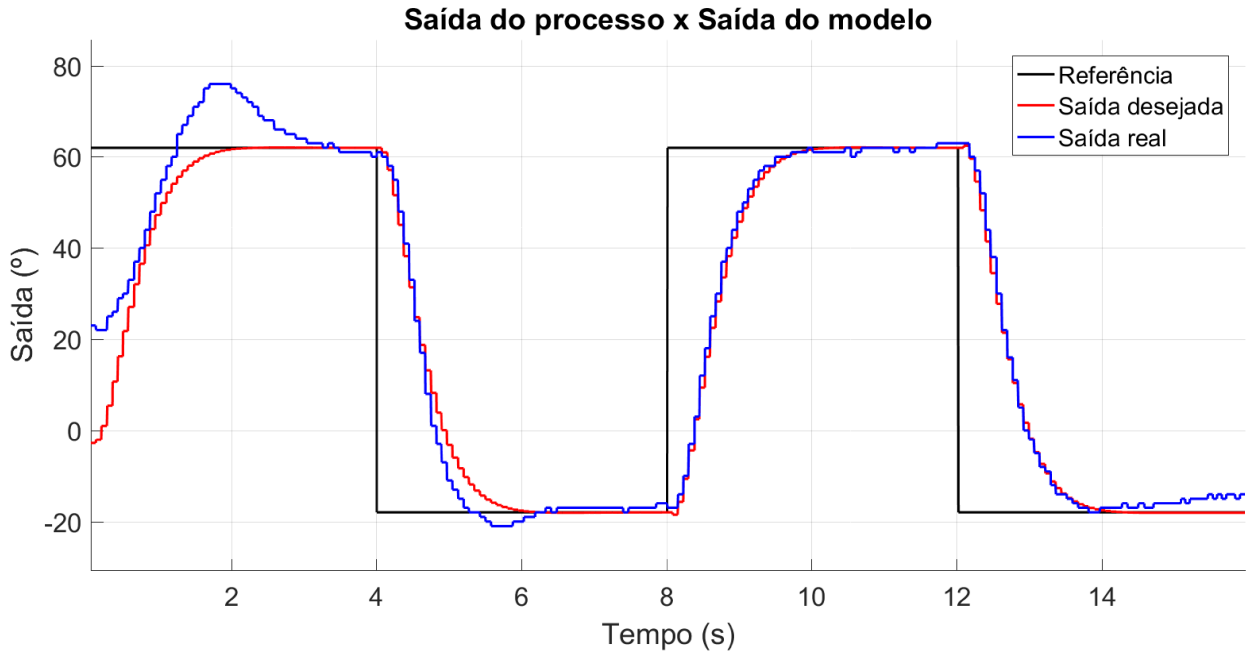

Figura 4.3 – Comparação entre a saída desejada $y_m(t)$ obtida a partir do modelo, a saída do processo controlado $y(t)$, e o sinal de referência $u_c(t)$.

Figura 4.4 – Evolução da entrada do processo (ação de controle).

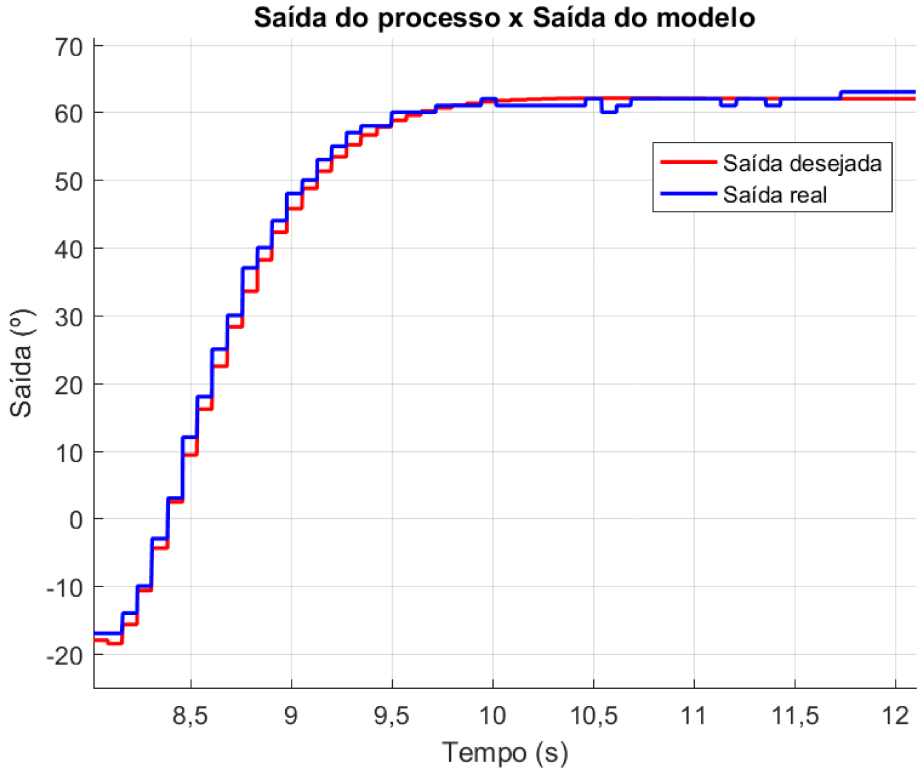

Figura 4.5 – Comparação entre a saída desejada $y_m(t)$ obtida a partir do modelo e a saída do processo controlado $y(t)$, considerando um intervalo de tempo dentro da adaptação.

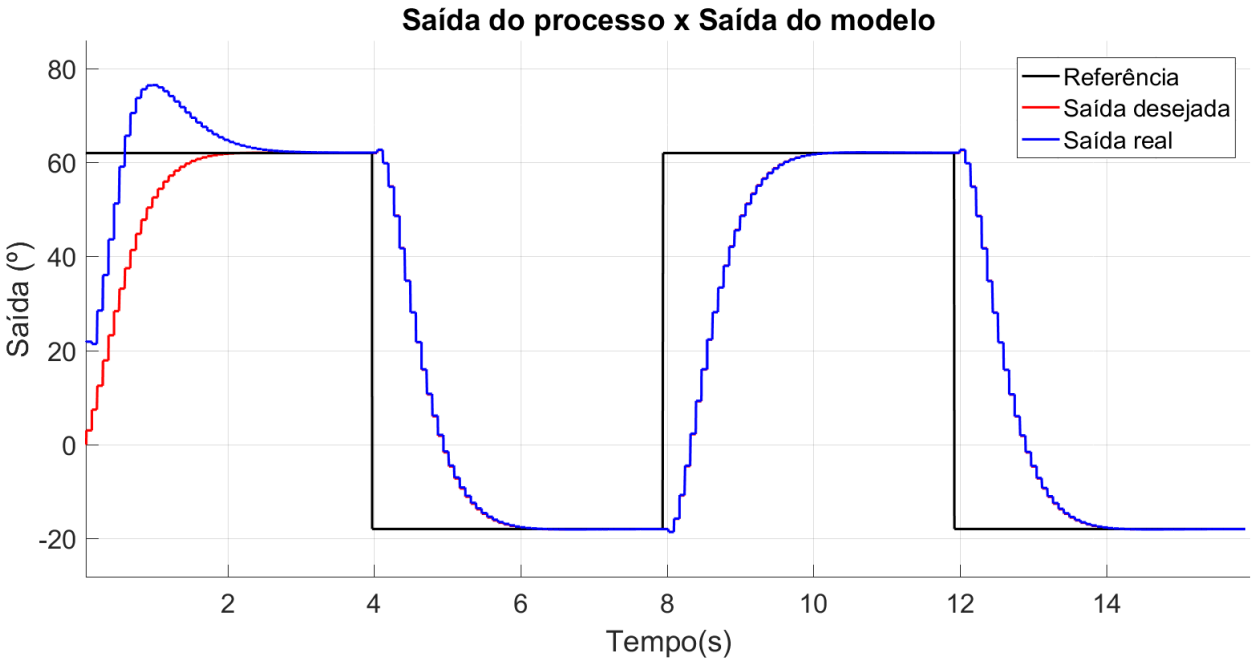

Figura 4.6 – Resultados obtidos em simulação para as mesmas condições (valor inicial de $y(t)$ e tempo de amostragem) observadas experimentalmente.

O segundo conjunto de dados (Figs. 4.7, 4.8 e 4.9) mostra o que acontece quando não é possível satisfazer os requisitos de desempenho especificados devido a limitações físicas do processo controlado. Neste caso, a saída desejada $y_m(t)$ foi definida a partir dos seguintes requisitos de desempenho

$$M_p = 0,1\%$$

$$t_a = 0,5s$$

Observe que, neste caso, o tempo de acomodação é duas vezes menor do que o proposto no último conjunto de dados apresentado. A Fig. 4.9, que mostra a saída do sistema no intervalo de 6 a 9 segundos, deixa claro que, mesmo que os valores de $y(t)$ tenham acompanhado os de $y_m(t)$ durante a maior parte do tempo, foi possível observar um *overshoot* indesejado na saída do sistema. Este comportamento pode ser explicado pela escolha equivocada dos requisitos de desempenho, que não levou em consideração as limitações físicas dos atuadores.

A especificação dos requisitos de desempenho (em especial do tempo de acomodação)

exigiu do sistema uma resposta excessivamente rápida, o que fez com que seus atuadores saturassem, como é possível observar na Fig. 4.8. Quando o valor da saída do sistema $y(t)$ se aproxima da referência $u_c(t)$, a ação de controle $u(t)$ precisa começar a diminuir para que $y(t)$ convirja suavemente para o valor desejado. Porém, quando o valor de $y(t)$ atinge $u_c(t)$ pela primeira vez, $u(t)$ já assumiu seu valor máximo, e durante o tempo que ele leva para começar a decair, uma pequena oscilação aparece na saída do sistema. Isso explica o aparecimento do *overshoot*.

Note que o tempo de acomodação foi bastante próximo do desejado, apesar do *overshoot* inesperado. Além disso, é importante dizer que, um *overshoot* como este, de cerca de $3,5\%$, é aceitável e passaria praticamente despercebido durante a execução do controle de trajetória.

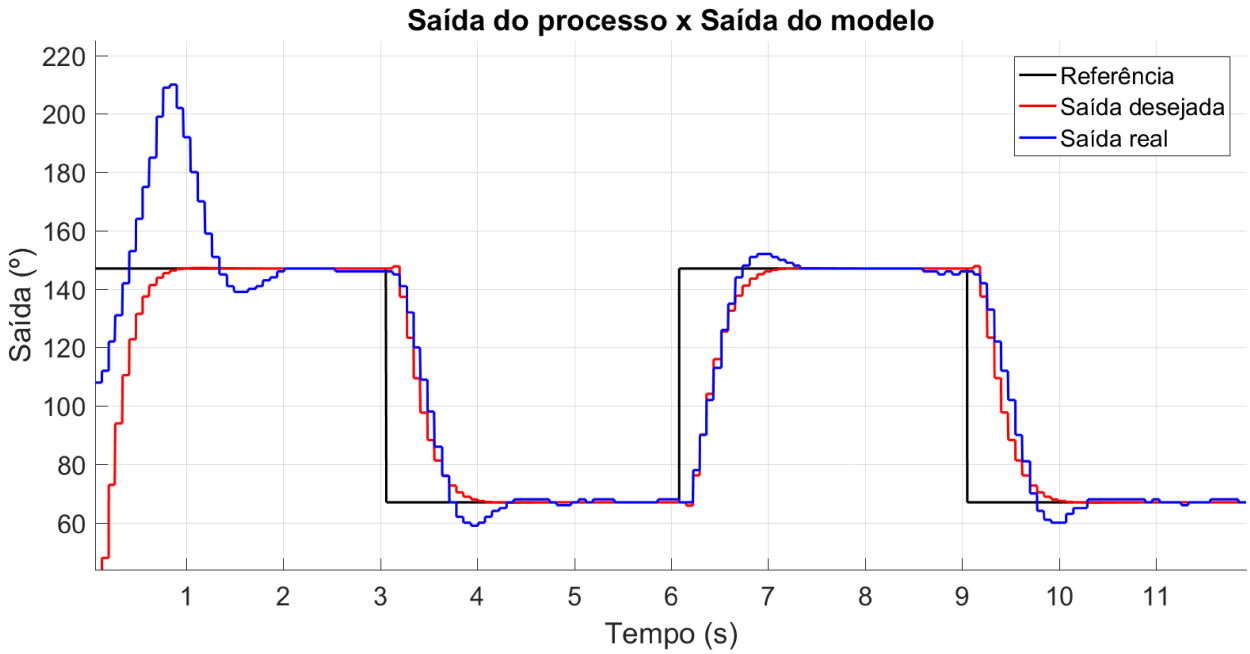

Figura 4.7 – Comparação entre a saída desejada $y_m(t)$ obtida a partir do modelo, a saída do processo controlado $y(t)$, e o sinal de referência $u_c(t)$.

O terceiro e último conjunto de dados que será analisado nesta seção, representado nos gráficos das Figs. 4.10, 4.11 e 4.12, será aqui apresentado para que fique bastante clara a importância do controle adaptativo. Neste caso, a saída desejada $y_m(t)$ foi definida a partir dos

seguintes requisitos de desempenho

$$M_p = 0,1\%$$

$$t_a = 1s$$

assim como no primeiro experimento introduzido nesta seção.

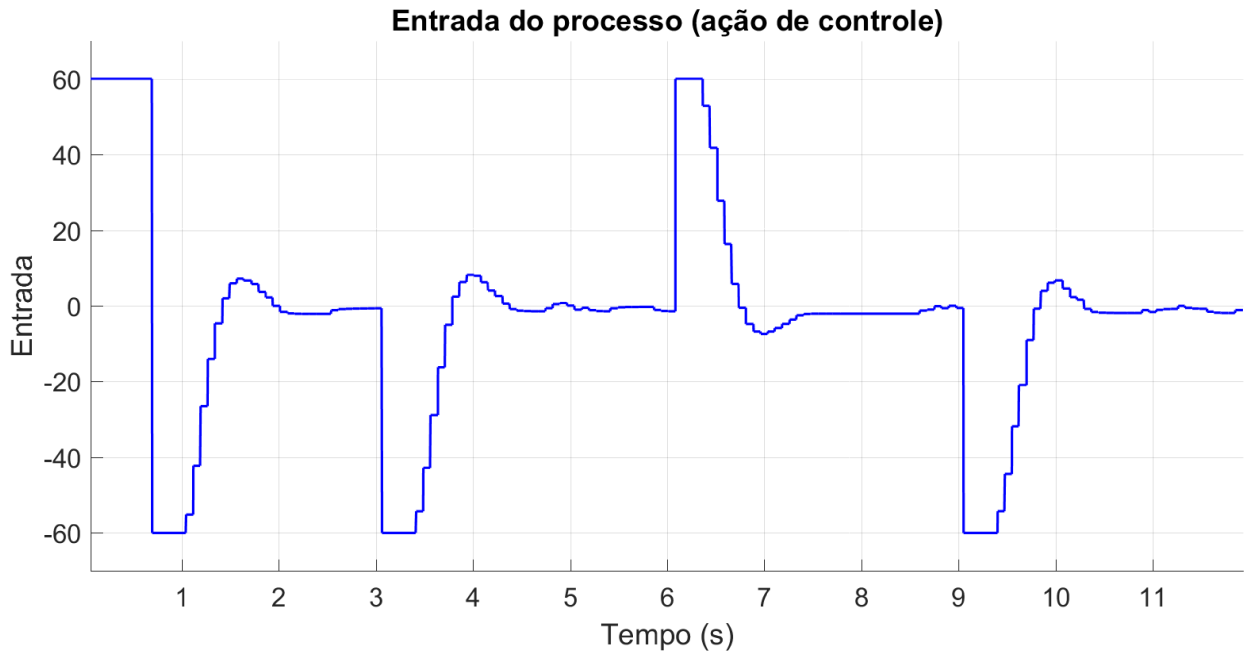

Figura 4.8 – Evolução da entrada do processo (ação de controle).

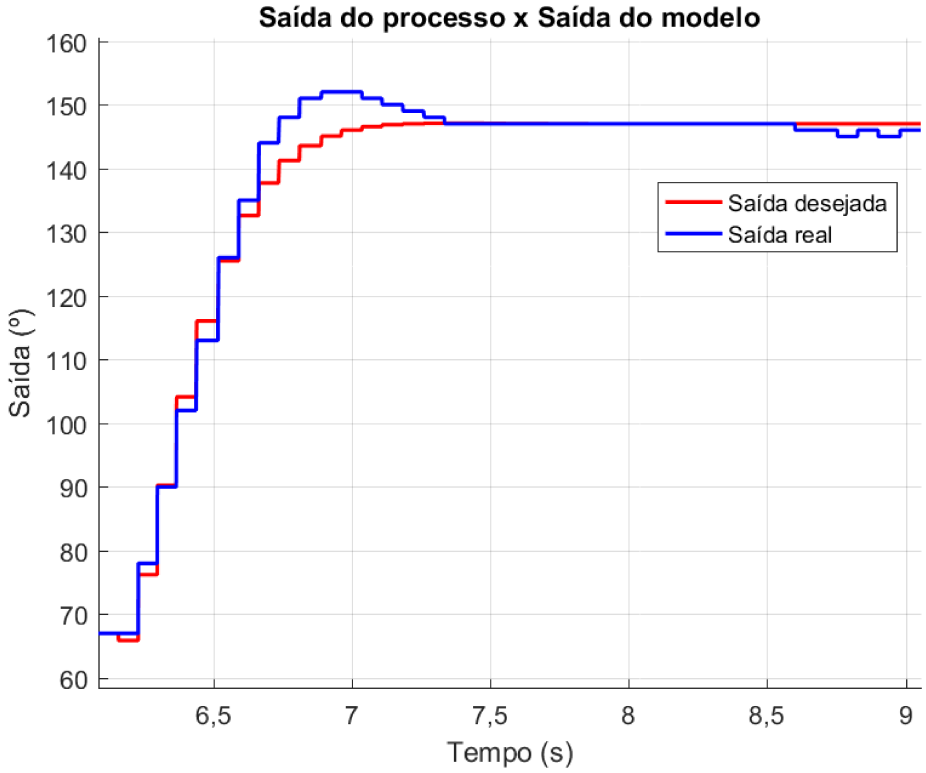

Figura 4.9 – Comparação entre a saída desejada $y_m(t)$ obtida a partir do modelo e a saída do processo controlado $y(t)$, considerando um intervalo de tempo dentro da adaptação.

Em primeira análise é possível observar que apesar do erro em regime permanente ter sido diferente de zero, a saída do sistema $y(t)$ acompanhou razoavelmente bem a saída desejada $y_m(t)$. A aplicação da Equação 4.1 ao conjunto de dados compreendido entre 6 e 9 segundos de execução, mostra que o erro médio entre $y(t)$ e $y_m(t)$, neste intervalo, foi de de $3,415°$.

Cabe ressaltar que o erro em regime permanente foi maior do que deveria e que, para corrigir este problema, seria necessária uma nova adaptação. No entanto, é interessante observar através deste experimento que nem sempre a saída do sistema se equiparará à saída desejada, uma vez que os parâmetros do processo controlador são inicializados com valores aleatórios entre 0 e 1, o que faz com que cada adaptação gere um resultado distinto.

Além disso, é importante ressaltar que esta adaptação foi realizada quando a bateria do seguidor estava quase no fim, enquanto que o primeiro experimento discutido nesta seção foi executado com ela praticamente cheia. Apesar dos valores da saída do sistema terem se mostrado bastante semelhantes nos dois casos, a ação de controle $u(t)$, que atingiu um valor máximo de apenas 20 no primeiro experimento, atingiu, no segundo, cerca de 30 (um valor 50% maior). Desta forma, é possível concluir que a carga da bateria do seguidor pode influenciar de forma significativa em seu comportamento, e é aqui que fica clara a importância do controle adaptativo. Note que, o controlador se adaptou, durante a etapa de adaptação, às condições de funcionamento do sistema, de forma que este apresentasse comportamento o mais próximo possível do desejado.

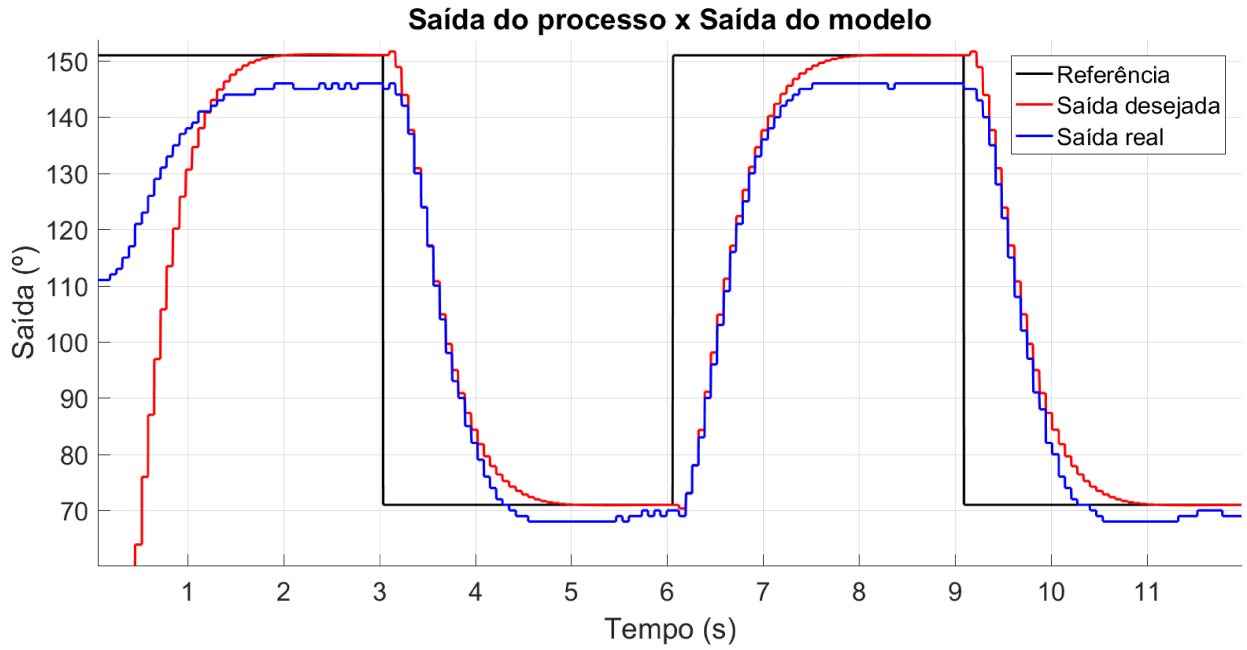

Figura 4.10 – Comparação entre a saída desejada $y_m(t)$ obtida a partir do modelo, a saída do processo controlado $y(t)$, e o sinal de referência $u_c(t)$.

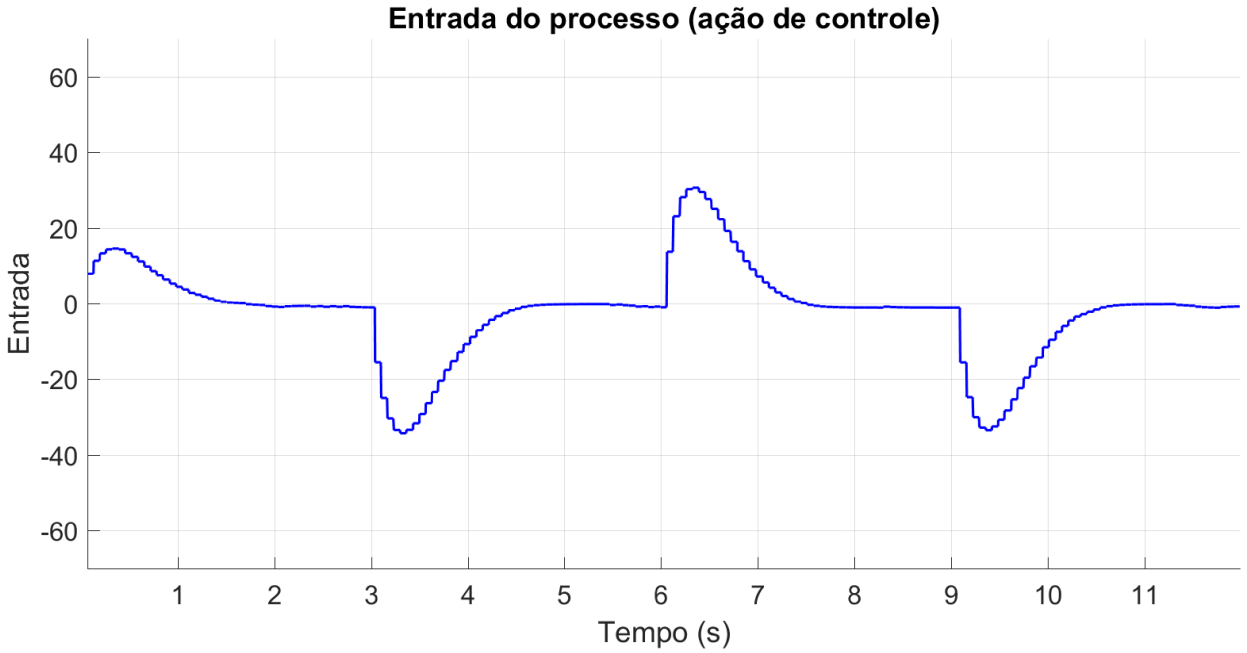

Figura 4.11 – Evolução da entrada do processo (ação de controle).

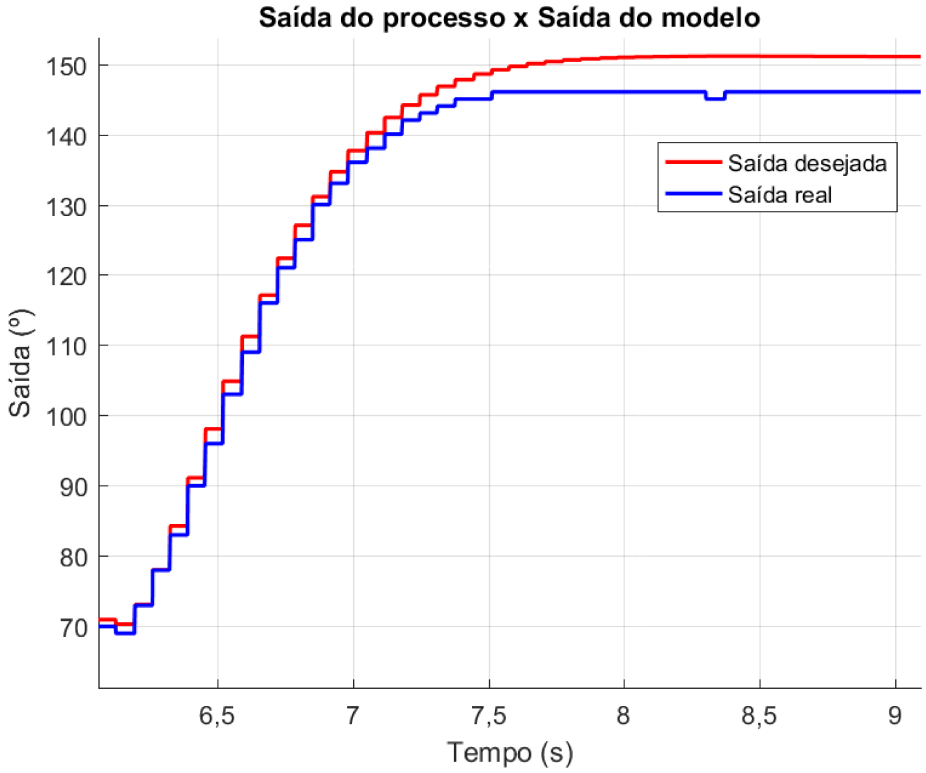

Figura 4.12 – Comparação entre a saída desejada $y_m(t)$ obtida a partir do modelo e a saída do processo controlado $y(t)$, considerando um intervalo de tempo dentro da adaptação.

Não serão apresentados os resultados referentes à fase de controle uma vez que nesta a referência se mantém constante e nenhuma alteração é observada na saída. No entanto, caso alguma perturbação seja imposta ao sistema foi possível avaliar experimentalmente que a implementação do controlador possibilita que o valor de referência seja novamente alcançado. Além disso, durante o regime transiente o tempo de acomodação e o *overshoot* atingidos foram bem próximos daqueles estipulados pelo usuário. Um vídeo que mostra o funcionamento do seguidor na fase de controle pode ser acessado em `https://youtu.be/ZVC4HjIPwM4`.

Um vídeo do seguidor durante a etapa de adaptação, considerando como requisitos de desempenho *overshoot* igual a 0,1% e tempo de acomodação igual a 1 segundo, pode ser acessado no link `https://youtu.be/HHvOO9RgzqI`. Neste vídeo, a referência é alterada ao longo do tempo para que os parâmetros do controlador possam ser determinados da melhor forma possível. Para ver o resultado desta adaptação, basta acessar o vídeo no link `https://youtu.be/bd5g05JOUfI`, onde o comportamento do seguidor pode ser avaliado à medida que a referência (direção a seguida) é alterada a cada 2 segundos.

CAPÍTULO V

CONCLUSÕES E TRABALHOS FUTUROS

5.1 Conclusões

Simulações no Matlab® permitiram que o comportamento do seguidor fosse previamente avaliado, para que em seguida o STR fosse implementado no robô e a eficácia do sistema de controle pudesse ser observada na prática.

Após as simulações e através da implementação prática, foi possível concluir que o objetivo principal deste trabalho foi alcançado e um controlador adaptativo, do tipo STR, foi implementado na sintonia dos parâmetros de um controlador que possibilitou o rastreamento de um trajetória pré determinada por um robô autônomo.

Um algoritmo em Java foi desenvolvido para determinação dos parâmetros do controlador implementado bastando somente que fossem informados os requisitos de desempenho desejados. O tempo gasto na implementação do controle de trajetória foi consideravelmente reduzido uma vez que a sintonização do controlador deixou de ser realizada manualmente.

O controlador implementado foi capaz de se adequar à dinâmica do processo controlado de forma que fossem garantidos *overshoot* e tempo de acomodação bem próximos daqueles especificados. Observou-se que eventuais modificações sofridas pelo sistema, como variações na carga da bateria por exemplo, foram compensadas pela atuação do controlador depois deste devidamente calibrado. E além disso, foi possível validar os resultados obtidos em simulação, através de sua comparação com os dados experimentais.

5.2 Trabalhos Futuros

Apesar dos bons resultados obtidos neste trabalho, muito ainda pode ser melhorado. Para tratar o sistema a ser controlado como um SISO, a ação de controle foi associada à potência dos motores de forma que, caso a rotação de um dos motores aumentasse, a potência do outro deveria diminuir. Isso impede que o controlador seja capaz de lidar com um robô que apresente um mal funcionamento em apenas um dos motores. Assim sendo, para que o controle de trajetória funcione da melhor forma possível, cada motor deve ser tratado separadamente, o que faria do sistema um MISO.

Além disso, trabalhos futuros podem abordar outras estruturas de controle adaptativo baseadas tanto no Tabelamento de Ganhos quanto em Modelos de Referência. Sistemas de controle baseados em Inteligência Artificial tem ganhado força recentemente, uma vez que podem ser aplicados na solução de problemas complexos e possibilitam a obtenção de resultados bastante satisfatórios. Assim sendo, esquemas deste tipo também serão abordados futuramente.

REFERÊNCIAS BIBLIOGRÁFICAS

ASTROM, Karl J.; WITTENMARK, Bjorn. Adaptative Control. 2. ed. [S.l.]: Addison Wesley, 1995.

HAYES, Monson H. Digital Signal Processing. 2. ed. [S.l.]: McGraw-Hill Education, 2011.

HITECHNIC. HiTechnic NXT Compass Sensor for LEGO Mindstorms NXT®. 2018. http://www.hitechnic.com/cgi-bin/commerce.cgi?preadd=action&key=NMC1034. [Acessado dia 21 de Outubro de 2018].

LANDAU, Ioan Doré; LOZANO, Rogelio; M'SAAD, Mohammed; KARIMI, Alireza. Adaptive control. [S.l.]: Springer London, 1998.

LEGO. LEGO Mindstorms NXT® : Hardware Developer Kit. 2006.

LEGO. LEGO Mindstorms EV3® Intelligent Brick. 2018. https://www.amazon.com/LEGO-Mindstorms-EV3-Intelligent-Brick/dp/B00G1L0278. [Acessado dia 28 de Dezembro de 2018].

LEJOS. Controlling the Motors. 2018. http://lejos.org/nxt/nxj/tutorial/MotorTutorial/ControllingMotors.htm. [Acessado dia 21 de Outubro de 2018].

MATHWORKS. Matlab function: goodnessOfFit. 2019. https://www.mathworks.com/help/ident/ref/goodnessoffit.html. [Acessado dia 24 de Janeiro de 2019].

OGATA, K. Discrete-Time Control Systems. [S.l.]: Prentice-Hall, 1987. (Prentice-Hall International Editions).

OGATA, K. Engenharia de controle moderno. [S.l.]: Prentice Hall, 2010. (Instrumentation and controls series).

NARENDRA, Kumpati S.; ANNASWAMY, Anuradha M. Persistent excitation in adaptative systems. International Journal of Control, [S. l.], p. 127-160, 1987.

APÊNDICES

APÊNDICE I - CÓDIGOS UTILIZADOS

Este apêndice trás os códigos utilizados na geração dos dados apresentados ao longo deste trabalho. As funções auxiliares

- `getControlPar()`
- `changeRef()`
- `plotResultsControl()`
- `getModel()`
- `plotResults()`
- `completePlot()`
- `completeSubPlot()`
- `completePlotDigital()`
- `completeSubPlotDigital()`

se encontram no Apendice II. Todos os códigos apresentados tanto no Apêndice I quanto no Apêndice II podem ser acessados no link
`https://gitlab.com/arthuriasbeck/iasbeck-tcc-adaptative-control`.

Código utilizado na reprodução dos dados apresentados nas Figuras 3, 4 e 5.

```
addpath('libraries');
addpath('../libraries/PlotData')

clear; clc; close all;

% Parametros para execucao
T = 0.05;        % Tempo de amostragem
tSim = 20;       % Tempo de simulacao
tInput = 2;      % Tempo para que ocorra uma mudanca na entrada do sistema
ucAmp = 90;      % Determina a amplitude da entrada degrau
ucSig = -1;      % Determina se o degrau sera positivo ou negativo

% Parametros do modelo
am1 = -1.6302;
am2 = 0.6700;
a0 = 0;

% Inicializacao de variaveis para armazenamento de dados
uArray = zeros(1,1);     % Vetor que armazena os valores de entrada
ucArray = zeros(1,1);     % Vetor que armazena os valores de referencia
yArray = zeros(1,1);     % Vetor que armazena os valores de saida
yeArray = zeros(1,1);    % Vetor que armazena os valores de estimacao
ymArray = zeros(1,1);    % Vetor que armazena os valores do modelo
a1Array = zeros(1,1);    % Vetor que armazena os valores de a1
a2Array = zeros(1,1);    % Vetor que armazena os valores de a2
b0Array = zeros(1,1);    % Vetor que armazena os valores de b0
b1Array = zeros(1,1);    % Vetor que armazena os valores de b1

% Inicializacao de variaveis utilizadas na simulacao da planta
y = 0;       % y(t)
y_1 = 0;     % y(t-1)
y_2 = 0;     % y(t-2)
u = 0;       % u(t)
```

```
u_1 = 0;     % u(t-1)
u_2 = 0;     % u(t-2)

% Inicializacao das variaveis utilizadas na simulacao do modelo
ym = 0;      % ym(t)
ym_1 = 0;    % ym(t-1)
ym_2 = 0;    % ym(t-2)
uc = 0;      % uc(t)
uc_1 = 0;    % uc(t-1)
uc_2 = 0;    % uc(t-2)

% Inicializacao das variaveis utilizadas na estimacao
ye = 0;                      % Saida estimada pelo minimos quadrados
fi = [-y_1 -y_2 u_1 u_2];    % Vetor de variaveis conhecidas
parNum = length(fi);         % Quantidade de parametros a serem determinados
P = 5*rand(parNum);          % Matriz de correcao
teta = rand(parNum,1);       % Vetor de parametros

% Inicializacao dos parametros da planta
a1 = 0;
a2 = 0;
b0 = 0;
b1 = 0;

% Variaveis para controle de execucao
numLoops = tSim/T;

for loops = 1:numLoops
    % Computando saida da planta
    if loops < numLoops/2
        y = 1.0002*y_1 + 0.0008*y_2 - 0.0244*u_1 + 0.1503*u_2;
    end
    if loops ≥ numLoops/2
        if loops == numLoops/2
            fi = [-y_1 -y_2 u_1 u_2];    % Vetor de variaveis conhecidas
            parNum = length(fi);         % Quantidade de parametros a ...
```

```
            serem determinados
        P = eye(parNum)*5;          % Matriz de correcao
        teta = rand(parNum,1);      % Vetor de parametros
    end
    y = 1.0002*y_1 + 0.0008*y_2 + 1*u_1 + 0.1503*u_2;
end

% Atualizacao dos parametros do modelo
fi = [-y_1 -y_2 u_1 u_2]';
P = P - P*fi*(1/(1 + fi'*P*fi))*fi'*P;
K = P*fi;
ye = fi'*teta;
errEst = y - ye;
teta = teta + K*errEst;

% Atualizando parametros da planta
a1 = teta(1);
a2 = teta(2);
b0 = teta(3);
b1 = teta(4);
if loops < numLoops/2
    a1 = 1.0003;
    a2 = -8.1854e-04;
    b0 = -0.0244;
    b1 = 0.1503;
end

% Atualizando parametros do controlador
[s0, s1, r1, beta] = getControlPar(a0, a1, a2, b0, b1, am1, am2);
if loops < numLoops/2
    s0 = 0.3222;
    s1 = -0.0031;
    r1 = -0.6221;
    beta = 0.3161;
end
```

```
    % Computando saida do modelo
    ym = -am1*ym_1 - am2*ym_2 + beta*b0*uc_1 + beta*b1*uc_2;

    % Computando as entradas
    ucSig = changeRef(loops, tInput, T, ucSig);
    uc = ucAmp*ucSig;
    u = -r1*u_1 + beta*uc + beta*a0*uc_1 - s0*y - s1*y_1;
    %u = correctInput(u);

    % Armazenando dados de execucao
    uArray(loops) = u;
    ucArray(loops) = uc;
    yArray(loops) = y;
    yeArray(loops) = ye;
    ymArray(loops) = ym;
    a1Array(loops) = a1;
    a2Array(loops) = a2;
    b0Array(loops) = b0;
    b1Array(loops) = b1;

    % Atualizando variaveis que armanezam dados de iteracoes passadas
    y_2 = y_1;
    y_1 = y;
    u_2 = u_1;
    u_1 = u;
    ym_2 = ym_1;
    ym_1 = ym;
    uc_2 = uc_1;
    uc_1 = uc;
end

% Mostrando os resultados da simulacao
U = uArray;
UC = ucArray;
Y = yArray;
YE = yeArray;
```

```
YM = ymArray;
A1 = a1Array;
A2 = a2Array;
B0 = b0Array;
B1 = b1Array;
TETA = [1 0.0008 1 0.1503];

plotResultsControl(numLoops, T, U, UC, Y, YE, YM, A1, A2, B0, B1, TETA);
```

Código utilizado na reprodução dos dados apresentados na Figuras 6.

```
clear; clc; close all;

b0 = 1;
b1 = 0.45;
b2 = 0.1;

%-------------------------------------------------------------------

U = 1:10;
Y = zeros(1,1);
FI = zeros(1,1);

for i = 1:10
    u = U(i);
    fiT = 1;
    FI(i,:) = fiT;
    y = b0 + b1*u + b2*u^2 + (rand/10);
    Y(i) = y;
end
Y = Y';

TETA0 = inv(FI'*FI)*FI'*Y;
Y0 = FI*TETA0;

%-------------------------------------------------------------------

U = 1:10;
Y = zeros(1,1);
FI = zeros(1,2);

for i = 1:10
    u = U(i);
    fiT = [1 u];
```

```
    FI(i,:) = fiT;
    y = b0 + b1*u + b2*u^2 + (rand/10);
    Y(i) = y;
end
Y = Y';

TETA1 = inv(FI'*FI)*FI'*Y;
Y1 = FI*TETA1;

%-------------------------------------------------------------------------

U = 1:10;
Y = zeros(1,1);
FI = zeros(1,3);

for i = 1:10
    u = U(i);
    fiT = [1 u u^2];
    FI(i,:) = fiT;
    y = b0 + b1*u + b2*u^2 + (rand/10);
    Y(i) = y;
end
Y = Y';

TETA2 = inv(FI'*FI)*FI'*Y;
Y2 = FI*TETA2;

%-------------------------------------------------------------------------

U = 1:10;
Y = zeros(1,1);
FI = zeros(1,4);

for i = 1:10
    u = U(i);
    fiT = [1 u u^2 u^3];
```

```
    FI(i,:) = fiT;
    y = b0 + b1*u + b2*u^2 + (rand/10);
    Y(i) = y;
end
Y = Y';

TETA3 = inv(FI'*FI)*FI'*Y;
Y3 = FI*TETA3;

thickness = 1.5;
subplot(2,2,1), hold on;
plot(U,Y,'o');
plot(U,Y0,'k');
sizeAxis = axis;
sizeAxis(1) = 1;
axis(sizeAxis);
set(findall(gca, 'Type', 'Line'),'LineWidth',thickness);
title('(a)');
xlabel('u');
ylabel('y');
grid on;

subplot(2,2,2), hold on;
plot(U,Y,'o');
plot(U,Y1,'k');
sizeAxis = axis;
sizeAxis(1) = 1;
axis(sizeAxis);
set(findall(gca, 'Type', 'Line'),'LineWidth',thickness);
title('(b)');
xlabel('u');
ylabel('y');
grid on;

subplot(2,2,3), hold on;
plot(U,Y,'o');
```

```
plot(U,Y2,'k');
sizeAxis = axis;
sizeAxis(1) = 1;
axis(sizeAxis);
set(findall(gca, 'Type', 'Line'),'LineWidth',thickness);
title('(c)');
xlabel('u');
ylabel('y');
grid on;

subplot(2,2,4), hold on;
plot(U,Y,'o');
plot(U,Y3,'k');
sizeAxis = axis;
sizeAxis(1) = 1;
axis(sizeAxis);
set(findall(gca, 'Type', 'Line'),'LineWidth',thickness);
title('(d)');
xlabel('u');
ylabel('y');
grid on;

print('lsExample','-dpng');
```

Código utilizado na reprodução dos dados apresentados nas Figuras 7 e 8.

```
clear;
clc;
close all;

nData = 300;

u = rand;
fi = [1 u u^2]';

arrayTeta = zeros(3,1);
arrayY = zeros(1,1);
arrayMod = zeros(1,1);
arrayErr = zeros(1,1);

P = rand(length(fi));
teta = rand(3,1);

% Modelo
% y(i) = b0 + b1u + b2u^2

b0 = 1;
b1 = 0.45;
b2 = 0.1;

U = 1:nData;

for i = 1:nData
    u = rand*10;
    y = b0 + b1*u + b2*u^2;
    fi = [1 u u^2]';
    P = P - P*fi*(1/(1 + fi'*P*fi))*fi'*P;
    K = P*fi;
    err = y - fi'*teta;
```

```
    teta = teta + K*err;

    arrayTeta(:,i) = teta;
    arrayY(i) = y;
    arrayMod(i) = fi'*teta;
    arrayErr(i) = err;
end

%%
% Plotando resultados
x = 1:nData;
y = arrayY;
lx = 'Iteracoes';
ly = 'y';
t = 'Comparacao entre a saida real e a saida estimada';
completeSubPlot(1,2,1,1,x,y,'',1.5,lx,ly,'','grid',0,0.1,'','');
y = arrayMod;
completeSubPlot(1,2,1,1,x,y,'ro',1.5,lx,ly,t,'grid',0,0.1,'','');
y = arrayY;
completeSubPlot(1,2,1,2,x,y,'',1.5,lx,ly,'','grid',0.9,1,'','');
y = arrayMod;
completeSubPlot(1,2,1,2,x,y,'ro',1.5,lx,ly,'','grid',0.9,1,'','');

% Plotando erro
x = 1:nData;
y = arrayErr;
lx = 'Iteracoes';
ly = 'Erro';
t = 'Erro de estimacao ao longo das iteracoes';
completeSubPlot(2,2,1,1,x,y,'',1.5,lx,ly,t,'grid',0,1,'','');
y = zeros(1,nData);
completeSubPlot(2,2,1,1,x,y,'--k',1.5,lx,ly,t,'grid',0,1,'','');

% Plotando parametros
x = 1:nData;
y = arrayTeta(1,:);
```

```
lx = 'Iteracoes';
ly = 'b0';
t = 'Valores dos parametros do modelo';
completeSubPlot(3,2,1,1,x,y,'',1.5,lx,ly,'','grid',0,1,'','');
y = ones(1,nData)*b0;
completeSubPlot(3,2,1,1,x,y,'--k',1.5,lx,ly,t,'grid',0,1,'','');

y = arrayTeta(2,:);
ly = 'b1';
completeSubPlot(3,2,1,2,x,y,'',1.5,lx,ly,'','grid',0,1,'','');
y = ones(1,nData)*b1;
completeSubPlot(3,2,1,2,x,y,'--k',1.5,lx,ly,'','grid',0,1,'','');

y = arrayTeta(3,:);
ly = 'b2';
completeSubPlot(4,2,1,1,x,y,'',1.5,lx,ly,'','grid',0,1,'','');
y = ones(1,nData)*b2;
completeSubPlot(4,2,1,1,x,y,'--k',1.5,lx,ly,'','grid',0,1,'','');
```

Código utilizado na reprodução dos dados apresentados nas Figuras 10, 11, 12, 13 e 14.

```
addpath('libraries');
addpath('../libraries/PlotData')

clear; clc; close all;

% Parametros para execucao
T = 0.05;   % Tempo de amostragem
tSim = 10;  % Tempo de simulacao
tInput = 1; % Tempo para que ocorra uma mudanca na entrada do sistema
uAmp = 10;  % Determina a amplitude da entrada degrau
uSig = 0;   % Determina se o degrau sera positivo ou negativo

% Inicializacao de variaveis para armazenamento de dados
uArray = zeros(1,1);        % Vetor que armazena os valores de entrada
yArray = zeros(1,1);        % Vetor que armazena os valores de saida
yeArray = zeros(1,1);       % Vetor que armazena os valores de estimacao
a1Array = zeros(1,1);       % Vetor que armazena os valores de a1
a2Array = zeros(1,1);       % Vetor que armazena os valores de a2
b0Array = zeros(1,1);       % Vetor que armazena os valores de b0
b1Array = zeros(1,1);       % Vetor que armazena os valores de b1
errEstArray = zeros(1,1);   % Vetor que armazena os valores do erro de ...
    estimacao

% Inicializacao de variaveis utilizadas na simulacao da planta
y = 0;      % y(t)
y_1 = 0;    % y(t-1)
y_2 = 0;    % y(t-2)
u = 0;      % u(t)
u_1 = 0;    % u(t-1)
u_2 = 0;    % u(t-2)

% Inicializacao das variaveis utilizadas na estimacao
```

```
ye = 0;                     % Saida estimada pelo minimos quadrados
fi = [-y_1 -y_2 u_1 u_2];   % Vetor de variaveis conhecidas
parNum = length(fi);        % Quantidade de parametros a serem determinados
P = rand(parNum);           % Matriz de correcao
teta = rand(parNum,1);      % Vetor de parametros

% Variaveis para controle de execucao
inputData = load('data/inputData.txt');
numLoops = length(inputData);  % Quantidade de iteracoes na simulacao

for loops = 1:numLoops
    % Computando entradas e saidas
    y = 1.0002*y_1 + 0.0008*y_2 - 0.0244*u_1 + 0.1503*u_2;
    u = inputData(loops);

    % Atualizacao dos parametros do modelo
    fi = [-y_1 -y_2 u_1 u_2]';
    P = P - P*fi*(1/(1 + fi'*P*fi))*fi'*P;
    K = P*fi;
    ye = fi'*teta;
    errEst = y - ye;
    teta = teta + K*errEst;

    % Atualizando parametros da planta/controlador
    a1 = teta(1);
    a2 = teta(2);
    b0 = teta(3);
    b1 = teta(4);

    % Armazenando dados de execucao
    uArray(loops) = u;
    yArray(loops) = y;
    yeArray(loops) = ye;
    a1Array(loops) = a1;
    a2Array(loops) = a2;
    b0Array(loops) = b0;
```

```
        b1Array(loops) = b1;
        errEstArray(loops) = errEst;

        % Atualizando variaveis que armanezam dados de iteracoes passadas
        y_2 = y_1;
        y_1 = y;
        u_2 = u_1;
        u_1 = u;
    end

    % Mostrando os resultados da simulacao
    U = uArray;
    Y = yArray;
    YE = yeArray;
    E = errEstArray;
    A1 = a1Array;
    A2 = a2Array;
    B0 = b0Array;
    B1 = b1Array;
    plotResults(numLoops, T, U, Y, YE, E, A1, A2, B0, B1);
```

Código utilizado na reprodução dos dados apresentados nas Figuras 17 e 18.

```
clear; clc; close all;
addpath('../libraries/PlotData');
% Parametros da planta e do modelo
b0 = 0.1065;
b1 = 0.0902;
a1 = -1.6065;
a2 = 0.6065;

% Parametros do modelo
bm0 = 0.1761;
am1 = -1.3205;
am2 = 0.4966;

% Parametros de execucao:}
T = 0.5;
tSim = 15;

% Variaveis para execucao
% Saidas para a planta
y = 0;
y_1 = 0;
y_2 = 0;
% Entradas para a planta
u = 0;
u_1 = 0;
u_2 = 0;
% Saidas para o modelo
ym = 0;
ym_1 = 0;
ym_2 = 0;
% Entradas para o modelo
uc = 0;
uc_1 = 0;
```

```
uc = 1;
yArray = zeros(1,1);
ymArray = zeros(1,1);
uArray = zeros(1,1);
loops = tSim/T;

for i = 1:loops
    % Computando acao de controle e saidas da planta e do modelo
    y = -a1*y_1 - a2*y_2 + b0*u_1 + b1*u_2;
    ym = -am1*ym_1 - am2*ym_2 + bm0*uc_1;
    u = (bm0/b0)*uc - (b1/b0)*u_1 + ((a1 - am1)/(b0))*y + ((a2 - ...
        am2)/(b0))*y_1;

    % Armaenando informacoes sobre execucao
    yArray(i) = y;
    ymArray(i) = ym;
    uArray(i) = u;

    % Atualilzando variaveis que armazenam entradas e saidas anteriores
    uc_1 = uc;
    ym_2 = ym_1;
    ym_1 = ym;
    u_2 = u_1;
    u_1 = u;
    y_2 = y_1;
    y_1 = y;
end

% Motrando graficamente os resultados
time = (0:i-1)*T;
x = 'Tempo(s)';
y = 'Saida';
title = 'Respostas do modelo e do sistema controlado';
f = 'zeroCancelOutput';
completePlotDigital(1, time, yArray, 'bo', 1.5, x, y, title, 'grid', 0, ...
```

```
    1, '', '', 0);
completePlotDigital(1, time, ymArray, 'r', 1.5, x, y, title, 'grid', 0, ...
    1, '', '', 1);
lgd = legend('Saida real', 'Saida desejada');
lgd.Position(2) = lgd.Position(2) - 0.1;
completePlotDigital(1, time, ymArray, 'r', 1.5, x, y, title, 'grid', 0, ...
    1, f, '', 1);

y = 'Entrada';
title = 'Entrada';
f = 'zeroCancelInput';
completePlotDigital(2, time, uArray, 'b', 1.5, x, y, title, 'grid', 0, ...
    1, f, '', 1);
```

Código utilizado na reprodução dos dados apresentados nas Figuras 19 e 20.

```
clear; clc; close all;
addpath('../libraries/PlotData');
addpath('libraries');
% Parametros de execucao
T = 0.5;
tSim = 15;

% Parametros da planta
a1 = -1.6065;
a2 = 0.6065;
b0 = 0.1065;
b1 = 0.0902;

% Parametros do modelo
am1 = -1.3205;
am2 = 0.4966;
a0 = 0;

% Variaveis para execucao
% Saidas para a planta
y = 0;
y_1 = 0;
y_2 = 0;
% Entradas para a planta
u = 0;
u_1 = 0;
u_2 = 0;
% Saidas para o modelo
ym = 0;
ym_1 = 0;
ym_2 = 0;
% Entradas para o modelo
uc = 0;
```

```
uc_1 = 0;
uc_2 = 0;

uc = 1;
yArray = zeros(1,1);
ymArray = zeros(1,1);
uArray = zeros(1,1);
loops = tSim/T;

for i = 1:loops
    % Computando acao de controle e saidas da planta e do modelo
    [s0, s1, r1, beta] = getControlPar(a0, a1, a2, b0, b1, am1, am2);
    y = -a1*y_1 - a2*y_2 + b0*u_1 + b1*u_2;
    ym = -am1*ym_1 - am2*ym_2 + beta*b0*uc_1 + beta*b1*uc_2;
    u = -r1*u_1 + beta*uc + beta*a0*uc_1 - s0*y - s1*y_1;

    % Armaenando informacoes sobre execucao
    yArray(i) = y;
    ymArray(i) = ym;
    uArray(i) = u;

    % Atualilzando variaveis que armazenam entradas e saidas anteriores
    uc_2 = uc_1;
    uc_1 = uc;
    ym_2 = ym_1;
    ym_1 = ym;
    u_2 = u_1;
    u_1 = u;
    y_2 = y_1;
    y_1 = y;
end

% Motrando graficamente os resultados
time = (0:i-1)*T;
x = 'Tempo(s)';
y = 'Saida';
```

```
title = 'Respostas do modelo e do sistema controlado';
f = 'zeroCancelOutput';
completePlotDigital(1, time, yArray, 'bo', 1.5, x, y, title, 'grid', 0, ...
    1, '', '', 0);
completePlotDigital(1, time, ymArray, 'r', 1.5, x, y, title, 'grid', 0, ...
    1, '', '', 1);
lgd = legend('Saida real','Saida desejada');
lgd.Position(2) = lgd.Position(2) - 0.1;
completePlotDigital(1, time, ymArray, 'r', 1.5, x, y, title, 'grid', 0, ...
    1, f, '', 1);

y = 'Entrada';
title = 'Entrada';
f = 'zeroCancelInput';
completePlotDigital(2, time, uArray, 'b', 1.5, x, y, title, 'grid', 0, ...
    1, f, '', 1);
```

Código utilizado na reprodução dos dados apresentados nas Figuras 22, 23, 24, 25, 43, 44, 45, 46, 47, 48 e 49.

```
addpath('libraries');
addpath('../libraries/PlotData');

clear; clc; close all;

% Parametros para execucao
T = 0.075;              % Tempo de amostragem
tSim = 15.98;           % Tempo de simulacao
tInput = 4;         % Tempo para que ocorra uma mudanca na entrada do ...
    sistema
ucAmp = 90;          % Determina a amplitude da entrada degrau
ucSig = -1;          % Determina se o degrau sera positivo ou negativo

% Parametros do modelo
ta = 1;
Mp = 5;
[am1 ,am2] = getModel(ta, Mp, T);
a0 = 0;

% Inicializacao de variaveis para armazenamento de dados
uArray = zeros(1,1);     % Vetor que armazena os valores de entrada
ucArray = zeros(1,1);    % Vetor que armazena os valores de referencia
yArray = zeros(1,1);     % Vetor que armazena os valores de saida
yeArray = zeros(1,1);    % Vetor que armazena os valores de estimacao
ymArray = zeros(1,1);    % Vetor que armazena os valores do modelo
a1Array = zeros(1,1);    % Vetor que armazena os valores de a1
a2Array = zeros(1,1);    % Vetor que armazena os valores de a2
b0Array = zeros(1,1);    % Vetor que armazena os valores de b0
b1Array = zeros(1,1);    % Vetor que armazena os valores de b1

% Inicializacao de variaveis utilizadas na simulacao da planta
y = 0;           % y(t)
```

```
y_1 = 0;          % y(t-1)
y_2 = 0;          % y(t-2)
u = 0;            % u(t)
u_1 = 0;          % u(t-1)
u_2 = 0;          % u(t-2)
initialY = 0;     % Condicao inicial para a saida do sistema

% Inicializacao das variaveis utilizadas na simulacao do modelo
ym = 0;       % ym(t)
ym_1 = 0;     % ym(t-1)
ym_2 = 0;     % ym(t-2)
uc = 0;       % uc(t)
uc_1 = 0;     % uc(t-1)
uc_2 = 0;     % uc(t-2)

% Inicializacao das variaveis utilizadas na estimacao
ye = 0;                       % Saida estimada pelo minimos quadrados
fi = [-y_1 -y_2 u_1 u_2];     % Vetor de variaveis conhecidas
parNum = length(fi);          % Quantidade de parametros a serem determinados
P = rand(parNum);             % Matriz de correcao
teta = rand(parNum,1);        % Vetor de parametros

% Inicializacao dos parametros da planta
a1 = 0;
a2 = 0;
b0 = 0;
b1 = 0;

% Variaveis para controle de execucao
numLoops = round(tSim/T);

for loops = 1:numLoops
    % Computando saida da planta
    if loops == 1
        y = initialY;
    else
```

```
    % Funcao que representa o seguidor
    y = 1.0002*y_1 + 0.0008*y_2 - 0.0244*u_1 + 0.1503*u_2;

    % Funcao que representa a planta G(s) = 1/s(s+1)
    % y = 1.6065*y_1 - 0.6065*y_2 + 0.1065*u_1 + 0.0902*u_2;
end

% Atualizacao dos parametros do modelo
fi = [-y_1 -y_2 u_1 u_2]';
P = P - P*fi*(1/(1 + fi'*P*fi))*fi'*P;
K = P*fi;
ye = fi'*teta;
errEst = y - ye;
teta = teta + K*errEst;

% Atualizando parametros da planta
a1 = teta(1);
a2 = teta(2);
b0 = teta(3);
b1 = teta(4);

% Atualizando parametros do controlador
[s0, s1, r1, beta] = getControlPar(a0, a1, a2, b0, b1, am1, am2);

% Computando saida do modelo
ym = -am1*ym_1 - am2*ym_2 + beta*b0*uc_1 + beta*b1*uc_2;

% Computando as entradas
ucSig = changeRef(loops, tInput, T, ucSig);
uc = ucAmp*ucSig;
u = -r1*u_1 + beta*uc + beta*a0*uc_1 - s0*y - s1*y_1;

% Descomente a linha abaixo caso deseje verificar a influencia da
% limitacao da saida
% u = correctInput(u);
```

```
    % Armazenando dados de execucao
    uArray(loops) = u;
    ucArray(loops) = uc;
    yArray(loops) = y;
    yeArray(loops) = ye;
    ymArray(loops) = ym;
    a1Array(loops) = a1;
    a2Array(loops) = a2;
    b0Array(loops) = b0;
    b1Array(loops) = b1;

    % Atualizando variaveis que armanezam dados de iteracoes passadas
    y_2 = y_1;
    y_1 = y;
    u_2 = u_1;
    u_1 = u;
    ym_2 = ym_1;
    ym_1 = ym;
    uc_2 = uc_1;
    uc_1 = uc;
end

% Mostrando os resultados da simulacao
U = uArray;
Uc = ucArray;
Y = yArray;
YE = yeArray;
YM = ymArray;
A1 = a1Array;
A2 = a2Array;
B0 = b0Array;
B1 = b1Array;

% Parametros relacionados ao seguidor
TETA = [-1.0002 -0.0008 -0.0244 0.1503];
```

```
% Parametros relacionados a planta G(s) = 1/s(s+1)
% TETA = [-1.6065 0.6065 0.1065 0.0902];

plotResultsControl(numLoops, T, U, Uc, Y, YE, YM, A1, A2, B0, B1, TETA);
```

Código utilizado na reprodução dos dados apresentados nas Figuras 37, 38 e 39.

```
%% Obtencao dos dados referentes a experimentacao com a bussola
clc;
clear;
close all;

F = '';

addpath('../libraries/PlotData');

% Upload dos dados
data = load('../../Java/WorkspaceTCC/PC/results/27 de Junho de 2018 - ...
    16h41.txt');
input = data(:,1);
compass = data(:,2);
time = data(:,3);
timeS = time/1000;

% Plot dos dados de estimacao
x = 'Tempo (s)';
y = 'Entrada';
t = 'Entrada do sistema';
f = 'Dados para estimacao';
completeSubPlot(1,2,1,1,timeS,input,'',1.5,x,y,t,'grid',0,1,'',F);
y = 'Saida(graus)';
t = 'Saida do sistema';
completeSubPlot(1,2,1,2,timeS,compass,'',1.5,x,y,t,'grid',0,1,f,F);

%% Estimacao da planta com minimos quadrados

% Caracteristicas do modelo
nPoles = 2;
nZeros = 2;
uCurrentBool = 0;
```

```
% Inicialiacao de variaveis para armazenar dados da execucao
arrayErr = zeros(1,1);
arrayY = zeros(1,1);
arrayMod = zeros(1,1);

% Inicializacao das variaveis utilizadas na estimacao da planta
uCurrent = 0;
yCurrent = 0;
y = zeros(1,nPoles);
u = zeros(1,nZeros);
if uCurrentBool == 1
    fi = [-y uCurrent u];
else
    fi = [-y u];
end

parNum = length(fi);
P = rand(parNum);
teta = rand(parNum,1);

% Laco para obtencao dos parametros da planta
for i = 1:length(input)
    % Obtencao da saida real do processo
    yCurrent = compass(i);
    uCurrent = input(i);

    % Atualizacao dos parametros do modelo
    if uCurrentBool == 1
        fi = [-y uCurrent u]';
    else
        fi = [-y u]';
    end

    P = P - P*fi*(1/(1 + fi'*P*fi))*fi'*P;
    K = P*fi;
```

```
    yModel = fi'*teta;
    err = yCurrent - yModel;
    teta = teta + K*err;

    % Armazenando informacoes sobre a execucao
    arrayY(i) = yCurrent;
    arrayMod(i) = yModel;
    arrayErr(i) = err;

    % Atualiacao dos vetores de entrada e saida
    shift = 1;

    uShift = zeros(size(u));
    uShift(1+shift:end) = u(1:end-shift);
    u = uShift;
    u(1) = uCurrent;

    yShift = zeros(size(y));
    yShift(1+shift:end) = y(1:end-shift);
    y = yShift;
    y(1) = yCurrent;
end

% Mostra um grafico para compararmos a saida real e a saida do modelo, ...
   e um
% para avaliarmos o erro (diferenca entre a saida real e a saida
% do modelo).
x = 'Tempo (s)';
y = 'Saida(graus)';
t = 'Saida do sistema';
f = 'Resultados da estimacao';
completeSubPlot(2,2,1,1,timeS,arrayMod,'',1.5,x,y,t,'grid',0,1,'',F);
completeSubPlot(2,2,1,1,timeS,arrayY,'',1.5,x,y,t,'grid',0,1,'',F);
legend('Saida estimada','Saida real');
t = 'Erro de estimacao';
y = 'Erro(graus)';
```

```
completeSubPlot(2,2,1,2,timeS,arrayErr,'k',1.5,x,y,t,'grid',0,1,f,F);

display(teta);

%% Validacao da planta obtida

% Upload dos dados
valData = load('../../Java/WorkspaceTCC/PC/results/27 de Junho de 2018 ...
    - 16h42.txt');

inputVal = valData(:,1);
compassVal = valData(:,2);
timeVal = valData(:,3);
timeS = timeVal/1000;

% Plot dos dados de validacao
x = 'Tempo (s)';
y = 'Entrada';
t = 'Entrada do sistema';
f = 'Dados para validacao';
completeSubPlot(3,2,1,1,timeS,inputVal,'',1.5,x,y,t,'grid',0,1,'',F);
y = 'Saida(graus)';
t = 'Saida do sistema';
completeSubPlot(3,2,1,2,timeS,compassVal,'',1.5,x,y,t,'grid',0,1,f,F);

%%
% ESTA SEcaO DO CoDIGO e EXECUTADA SOMENTE DEPOIS QUE A VALIDAcaO e FEITA
% NO IDENT E EXPORTADA PARA UMA FIGURA

figure(4);
grid on;
axis([0 10 105 165]);
set(findall(gca, 'Type', 'Line'),'LineWidth',1.5);
title('Saida do sistema (88,9%)');
xlabel('Tempo (s)');
ylabel('Amplitude');
```

```
legend('saida estimada','saida real');
print('results\Resultado da validacao', '-dpng');
```

Código utilizado na reprodução dos dados apresentados nas Figuras 53, 54, 55, 56, 57, 58, 59, 60, 61 e 62.

```
for sample = 1:3
    clearvars -except sample; clc; close all;
    addpath('../libraries/PlotData');
    dataNumber = num2str(sample);
    if strcmp(dataNumber,'1')
        gap = 107:162;
    end
    if strcmp(dataNumber,'2')
        gap = 83:123;
    end
    if strcmp(dataNumber,'3')
        gap = 92:138;
    end
    tickSize = 16;

    % Carregando os dados
    data = load(strcat(dataNumber,'.txt'));
    yArray = data(:,1);
    ucArray = data(:,2);
    ymArray = data(:,3);
    uArray = data(:,4);
    time = (data(:,5) - data(1,5))/1000;

    % Tratando os dados
    dt = zeros(1,1);
    for loop = 2:length(data)
        dt(loop) = time(loop) - time(loop-1);
    end

    sumTime = 0;
    newYmArray = zeros(1,1);
    newYArray = zeros(1,1);
```

```
newUcArray = zeros(1,1);
newUArray = zeros(1,1);
newTime = zeros(1,1);
for loop = 1:length(data)
    sumTime = sumTime + dt(loop);
    if sumTime > 0.05
        sumTime = 0;
        newYmArray(loop) = ymArray(loop);
        newYArray(loop) = yArray(loop);
        newUcArray(loop) = ucArray(loop);
        newUArray(loop) = uArray(loop);
        newTime(loop) = time(loop);
    else
        newYmArray(loop) = -999;
        newYArray(loop) = -999;
        newUcArray(loop) = -999;
        newUArray(loop) = -999;
        newTime(loop) = -999;
    end
end

newYmArray(newYmArray == -999) = [];
newYArray(newYArray == -999) = [];
newUcArray(newUcArray == -999) = [];
newUArray(newUArray == -999) = [];
newTime(newTime == -999) = [];

yArray = newYArray;
ucArray = newUcArray;
ymArray = newYmArray;
uArray = newUArray;
time = newTime;

% Mostrando a evolucao da saida do sistema
lx = 'Tempo (s)';
ly = 'Saida (graus)';
```

```
t = 'Saida do processo x Saida do modelo';
f = strcat('y_ym_uc','_',dataNumber);
completePlotDigital(1, time, ucArray, 'k', 1.5, lx, ly, t, 'grid', ...
    0, 1, '', '', 1);
completePlotDigital(1, time, ymArray, 'r', 1.5, lx, ly, t, 'grid', ...
    0, 1, '', '', 1);
completePlotDigital(1, time, yArray, 'b', 1.5, lx, ly, t, 'grid', ...
    0, 1, '', '', 1);
legend('Referencia','Saida desejada','Saida real');
set(gcf,'units','points','position',[20,100,900,400]);
get(gca, 'XTick');
set(gca, 'FontSize', tickSize);
completePlotDigital(1, time, yArray, 'b', 1.5, lx, ly, t, 'grid', ...
    0, 1, f, '', 1);

% Mostrando a evolucao da acao de controle
t = 'Entrada do processo (acao de controle)';
ly = 'Entrada';
f = strcat('u','_',dataNumber);
completePlotDigital(2, time, uArray, 'b', 1.5, lx, ly, t, 'grid', ...
    0, 1, '', '', 1);
set(gcf,'units','points','position',[20,100,900,400]);
get(gca, 'XTick');
set(gca, 'FontSize', tickSize);
sizeAxis = axis;
sizeAxis(3) = -70; sizeAxis(4) = 70;
axis(sizeAxis);
print(strcat('results\',f),'-dpng');

% Comparando a saida do sistema a saida do modelo para o ...
    controlador ja
% ajustado
t = 'Saida do processo x Saida do modelo';
ly = 'Saida (graus)';
f = strcat('y_ym','_',dataNumber);
ymArray = ymArray(gap);
```

```
    yArray = yArray(gap);
    time = time(gap);

    completePlotDigital(3, time, ymArray, 'r', 1.5, lx, ly, t, 'grid', ...
        0, 1, '', '', 1);
    completePlotDigital(3, time, yArray, 'b', 1.5, lx, ly, t, 'grid', ...
        0, 1, '', '', 1);
    lgd = legend('Saida desejada','Saida real');
    if(sample == 2)
        lgd.Position(2) = lgd.Position(2) - 0.15;
    else
        lgd.Position(2) = lgd.Position(2) - 0.1;
    end
    completePlotDigital(3, time, yArray, 'b', 1.5, lx, ly, t, 'grid', ...
        0, 1, f, '', 1);

    % Calculando a diferenca entre y e ym
    e = ymArray - yArray;
    e = abs(e);
    e = sum(e)/length(e);
    display(e);
end
```

APÊNDICE II - CÓDIGOS AUXILIARES

```
function [s0, s1, r1, beta] = getControlPar(a0, a1, a2, b0, b1, am1, am2)
    r1 = (am2 + am1*a0 - a2 - (b0/b1)*am2*a0 - (b1/b0)*(am1 - a1 + ...
       a0))/(a1 - (b0/b1)*a2 - (b1/b0));
    s0 = (am1 + a0 - a1 - r1)/b0;
    s1 = (am2*a0 - a2*r1)/b1;
    beta = (1 + am1 + am2)/(b0 + b1);
end
```

```
function refSigChange = changeRef(loops, tInput, T, refSig)
    refSigChange = refSig;
    if ¬mod(loops - 1, round(tInput/T))
        if refSig == 1
            refSigChange = -1;
        else if refSig == -1
            refSigChange = 1;
            end
        end
    end
end
```

```
function plotResultsControl(numLoops, T, uArray, ucArray, yArray, ...
   yeArray, ymArray, a1Array, a2Array, b0Array, b1Array, realTeta)
    % Mostrando resultados da simulacao da planta
    time = (0:numLoops-1)*T;
    lX = 'Tempo(s)';
    lY = 'Entrada';
```

```
title = 'Entrada do processo';
completeSubPlot(1, 2, 1, 1, time, uArray, 'r', 1.5, lX, lY, title, ...
    'grid', 0, 1, '', '');

title = 'Saida do processo';
lY = 'Saida';
file = 'Simulacao do seguidor com entrada real';
completeSubPlot(1, 2, 1, 2, time, ucArray, 'k', 1.5, lX, lY, title, ...
    'grid', 0, 1, '', '');
completeSubPlot(1, 2, 1, 2, time, yArray, '', 1.5, lX, lY, title, ...
    'grid', 0, 1, file, '');

% Mostrando comparacao entre a saida do processo e do modelo
title = 'Saida do processo x Saida do modelo';
file = 'Comparacao entre a saida do processo e a saida do modelo';
completePlotDigital(2, time, ymArray, 'b', 1.5, lX, lY, title, ...
    'grid', 0, 1, '', '', 0);
completePlotDigital(2, time, ucArray, 'k', 1.5, lX, lY, title, ...
    'grid', 0, 1, '', '', 0);
completePlotDigital(2, time, yArray, 'r', 1.5, lX, lY, title, ...
    'grid', 0, 1, '', '', 0);
legend('Saida desejada', 'Referencia', 'Saida real');
completePlotDigital(2, time, yArray, 'r', 1.5, lX, lY, title, ...
    'grid', 0, 1, file, '', 0);

% Mostrando dados da estimacao
title = 'Saida do processo x Saida estimada';
file = 'Resultados da estimacao';
completePlot(3, time, yArray, '', 1.5, lX, lY, title, 'grid', 0, 1, ...
    '', '');
completePlot(3, time, yeArray, 'k', 1.5, lX, lY, title, 'grid', 0, ...
    1, file, '');
legend('Saida do process', 'Saida estimacao');

% Mostrando evolucao dos parametros ao longo do tempo
% Parametro a1
```

```
lY = 'a1';
a1Real = ones(1, numLoops)*(realTeta(1));
a1Last = a1Array(end);
title = strcat({'a1 = '}, num2str(a1Last),{' ('}, ...
    num2str(realTeta(1)),{ ')'});
completeSubPlot(4, 2, 1, 1, time, a1Array, '', 1.5, lX, lY, title, ...
    '', 0, 0.07, '', '');
completeSubPlot(4, 2, 1, 1, time, a1Real, '--k', 1.5, lX, lY, ...
    title, '', 0, 0.07, '', '');

% Parametro a2
lY = 'a2';
a2Real = ones(1, numLoops)*(realTeta(2));
a2Last = a2Array(end);
title = strcat({'a2 = '}, num2str(a2Last),{' ('}, ...
    num2str(realTeta(2)),{ ')'});
file = 'Evolucao dos parametros a';
completeSubPlot(4, 2, 1, 2, time, a2Array, '', 1.5, lX, lY, title, ...
    '', 0, 0.07, '', '');
completeSubPlot(4, 2, 1, 2, time, a2Real, '--k', 1.5, lX, lY, ...
    title, '', 0, 0.07, file, '');

% Parametro b0
lY = 'b0';
b0Real = ones(1,numLoops)*(realTeta(3));
b0Last = b0Array(end);
title = strcat({'b0 = '}, num2str(b0Last),{' ('}, ...
    num2str(realTeta(3)),{ ')'});
completeSubPlot(5, 2, 1, 1, time, b0Array, '', 1.5, lX, lY, title, ...
    '', 0, 0.07, '', '');
completeSubPlot(5, 2, 1, 1, time, b0Real, '--k', 1.5, lX, lY, ...
    title, '', 0, 0.07, '', '');

% Parametro b1
lY = 'b1';
b1Real = ones(1,numLoops)*(realTeta(4));
```

```
    b1Last = b1Array(end);
    title = strcat({'b1 = '}, num2str(b1Last),{' ('}, ...
        num2str(realTeta(4)),{ ')'});
    file = 'Evolucao dos parametros b';
    completeSubPlot(5, 2, 1, 2, time, b1Array, '', 1.5, lX, lY, title, ...
        '', 0, 0.07, '', '');
    completeSubPlot(5, 2, 1, 2, time, b1Real, '--k', 1.5, lX, lY, ...
        title, '', 0, 0.07, file, '');
end
```

```
function [am1 ,am2] = getModel(ta, Mp, T)
    Mp = Mp/100;
    qsi = sqrt(((log(Mp))^2)/((log(Mp))^2 + pi^2));
    wn = (4)/(qsi*ta);
    t = T/2;
    A = -qsi*wn;
    B = wn*sqrt(1 - qsi^2);
    C = (1 - (t*A)^2 - (t*B)^2)/((1 - t*A)^2 + (t*B)^2);
    D = (2*t*B)/((1 - t*A)^2 + (t*B)^2);
    am1 = -2*C;
    am2 = C^2 + D^2;
end
```

```
function plotResults(numLoops, T, uArray, yArray, yeArray, errEstArray, ...
    a1Array, a2Array, b0Array, b1Array)
    % Mostrando resultados da simulacao da planta
    time = (0:numLoops-1)*T;
    lX = 'Tempo(s)';
    lY = 'Entrada';
    title = 'Entrada do processo';
    completeSubPlot(1, 2, 1, 1, time, uArray, '', 1.5, lX, lY, title, ...
```

```
    'grid', 0, 1, '', '')

lY = 'Saida';
title = 'Saida do processo';
file = 'Simulacao do seguidor com entrada real';
completeSubPlot(1, 2, 1, 2, time, yArray, '', 1.5, lX, lY, title, ...
    'grid', 0, 1, file, '')

% Mostrando dados da estimacao
title = 'Saida do processo x Estimacao';
file = 'Resultados da estimacao';
completePlot(2, time, yArray, '', 1.5, lX, lY, title, 'grid', 0, 1, ...
    '', '')
completePlot(2, time, yeArray, 'ko', 1.5, lX, lY, title, 'grid', 0, ...
    1, file, '')
legend('saida do process','estimacao');

% Mostrando evolucao dos parametros ao longo do tempo
% Parametro a1
lY = 'a1';
a1Real = ones(1,numLoops)*(-1.0002);
a1Last = a1Array(end);
title = strcat({'a1 = '}, num2str(a1Last),{' ('}, ...
    num2str(-1.0002),{ ')'});
completeSubPlot(3, 2, 1, 1, time, a1Array, '', 1.5, lX, lY, title, ...
    '', 0, 1, '', '');
completeSubPlot(3, 2, 1, 1, time, a1Real, '--k', 1.5, lX, lY, ...
    title, '', 0, 1, '', '');

% Parametro a2
lY = 'a2';
a2Real = ones(1,numLoops)*(-0.0008);
a2Last = a2Array(end);
title = strcat({'a2 = '}, num2str(a2Last),{' ('}, ...
    num2str(-0.0008),{ ')'});
file = 'Evolucao dos parametros a';
```

```
completeSubPlot(3, 2, 1, 2, time, a2Array, '', 1.5, lX, lY, title, ...
    '', 0, 1, '', '');
completeSubPlot(3, 2, 1, 2, time, a2Real, '--k', 1.5, lX, lY, ...
    title, '', 0, 1, file, '');

% Parametro b0
lY = 'b0';
b0Real = ones(1,numLoops)*(-0.0244);
b0Last = b0Array(end);
title = strcat({'b0 = '}, num2str(b0Last),{' ('}, ...
    num2str(-0.0244),{ ')'});
completeSubPlot(4, 2, 1, 1, time, b0Array, '', 1.5, lX, lY, title, ...
    '', 0, 1, '', '');
completeSubPlot(4, 2, 1, 1, time, b0Real, '--k', 1.5, lX, lY, ...
    title, '', 0, 1, '', '');

% Parametro b1
lY = 'b1';
b1Real = ones(1,numLoops)*(0.1503);
b1Last = b1Array(end);
title = strcat({'b1 = '}, num2str(b1Last),{' ('}, num2str(0.1503),{ ...
    ')'});
file = 'Evolucao dos parametros b';
completeSubPlot(4, 2, 1, 2, time, b1Array, '', 1.5, lX, lY, title, ...
    '', 0, 1, '', '');
completeSubPlot(4, 2, 1, 2, time, b1Real, '--k', 1.5, lX, lY, ...
    title, '', 0, 1, file, '');

% Mostrando a evolucao do erro de estimacao
lY = 'Erro';
title = 'Erro de estimacao';
file = 'Erro de estimacao';
e = zeros(1,numLoops);
completePlot(5, time, e, '--k', 1.5, lX, lY, title, 'grid', 0, 1, ...
    '', '')
completePlot(5, time, errEstArray, '', 1.5, lX, lY, title, 'grid', ...
```

```
            0, 1, file, '')
    end
```

```
function completePlot(figureNumber,x,y,plotConfig,thickness,lableX,lableY,
plotTitle,gridCte,xMin,xMax,saveName,fullScreen)
    % figureNumber - Numero da figura em que se encontra o plot
    % x - Valores de x
    % y - Valores de y
    % plotConfig - Configuracoes adicionais do plot. Se nao houverem
    %              configuracoes adicionais para o plot (cor da linha,
    %              tipo de linha...) o usuario deve inserir '' neste
    %              parametro
    % thickness - Espessura da linha do plot
    % lableX - Titulo do eixo X
    % lableY - Titulo do eixo Y
    % plotTitle - Titulo do plot
    % gridCte - Igual a 'grid' se o usuario desejar um grid no plot
    % xMax - Valor maximo de X, em porcentagem, a ser mostrado no plot
    % xMin - Valor minimo de X, em porcentagem, a ser mostrado no plot
    % saveName - Nome dos arquivos que conterao um print do plot. Se o
    %            usuario nao deseja salvar o plot num arquivo, deve inserir
    %            '' neste argumento.
    % fullScreen - Igual a 'full' se o usuario desejar o plot em tela
    %              inteira.

    % Instrucoes de utilizacao:
    % 1 - Esta funcao da um hold on automatico, entao se voce chama-la
    % utilizando o mesmo numero de figura 2 vezes, voce tera uma
    % sobreposicao de plots no mesmo figure.
    % 2 - Caso voce queira plotar varios graficos no mesmo plot para depois
    % salvar em um arquivo, voce deve inserir o nome do arquivo no
    % argumento saveName apenas no seu ultimo plot. Caso contrario, voce
    % salvara uma figura que nao tera todos os seus graficos ainda.
    % 3 - Caso voce queira plotar em tela inteira, com varios graficos no
    % mesmo plot, voce nao precisa necessariamente inserir o parametro
    % 'full' em todos os seus plots. Basta inserir em um deles.
```

```
% Verificacao dos limites de X.
if xMin > xMax
    display('Reveja os valores de xMin e xMax');
    return;
end

% Plot dos dados recebidos.
figure(figureNumber); hold on;
minIndex = round(xMin*length(x) + 1);
maxIndex = round(xMax*length(x));
x = x(minIndex:maxIndex);
y = y(minIndex:maxIndex);

if isempty(plotConfig)
    plot(x,y);
else
    plot(x,y,plotConfig);
end

%Configuracoes de exibicao.
xlabel(lableX);
ylabel(lableY);
title(plotTitle);
if strcmp(gridCte,'grid')
    grid on;
end
set(findall(gca, 'Type', 'Line'),'LineWidth',thickness);

% Determinacao dos limites do grafico.
sizeAxis = axis;
sizeAxis(1) = min(x);
sizeAxis(2) = max(x);
if min(y) ≠ max(y)
    sizeAxis(3) = min(y) - abs(max(y) - min(y))*0.1;
    sizeAxis(4) = max(y) + abs(max(y) - min(y))*0.1;
```

```
    end
    axis(sizeAxis);

    % Salvamento do plot em um arquivo .png e um em um arquivo .fig.
    if strcmp(fullScreen,'full')
        % Utilizado para maximizar a janela do plot
        set(gcf, 'Position', get(0, 'Screensize'));
    end
    if ¬isempty(saveName)
        if exist('results','dir') == 0
            mkdir('results');
        end
        if exist('results\figs','dir') == 0
            mkdir('results\figs');
        end
        print(strcat('results\', saveName), '-dpng');
        savefig(strcat('results\figs\', saveName));
    end

end
```

```
function ...
   completeSubPlot(figureNumber,m,n,a,x,y,plotConfig,thickness,lableX,
lableY,plotTitle,gridCte,xMin,xMax,saveName,fullScreen)
    % figureNumber - Numero da figura em que se encontra o plot
    % m - Numero de linhas do subplot
    % n - Numero de colunas do subplot
    % a - Numero do subplot no qual esta funcao inserira um grafico
    % x - Valores de x
    % y - Valores de y
    % plotConfig - Configuracoes adicionais do plot. Se nao houverem
    %              configuracoes adicionais para o plot (cor da linha,
    %              tipo de linha...) o usuario deve inserir '' neste
```

```
%              parametro
% thickness - Espessura da linha do plot
% lableX - Titulo do eixo X
% lableY - Titulo do eixo Y
% plotTitle - Titulo do plot
% gridCte - Igual a 'grid' se o usuario desejar um grid no plot
% xMax - Valor maximo de X, em porcentagem, a ser mostrado no plot
% xMin - Valor minimo de X, em porcentagem, a ser mostrado no plot
% saveName - Nome dos arquivos que conterao um print do plot. Se o
%            usuario nao deseja salvar o plot num arquivo, deve inserir
%            '' neste argumento.
% fullScreen - Igual a 'full' se o usuario desejar o plot em tela
%              inteira.

% Instrucoes de utilizacao:
% 1 - Esta funcao da um hold on automatico, entao se voce chama-la
% utilizando o mesmo numero de figura 2 vezes, voce tera uma
% sobreposicao de plots no mesmo figure.
% 2 - Caso voce queira plotar varios graficos no mesmo plot para depois
% salvar em um arquivo, voce deve inserir o nome do arquivo no
% argumento saveName apenas no seu ultimo plot. Caso contrario, voce
% salvara uma figura que nao tera todos os seus graficos ainda.
% 3 - Caso voce queira plotar em tela inteira, com varios graficos no
% mesmo plot, voce nao precisa necessariamente inserir o parametro
% 'full' em todos os seus plots. Basta inserir em um deles.

% Verificacao dos limites de X.
if xMin > xMax
    display('Reveja os valores de xMin e xMax');
    return;
end

% Plot dos dados recebidos.
figure(figureNumber); hold on;
minIndex = round(xMin*length(x) + 1);
maxIndex = round(xMax*length(x));
```

```
x = x(minIndex:maxIndex);
y = y(minIndex:maxIndex);
subplot(m,n,a);
if isempty(plotConfig)
    plot(x,y);
else
    plot(x,y,plotConfig);
end

%Configuracoes de exibicao.
xlabel(lableX);
ylabel(lableY);
title(plotTitle);
if strcmp(gridCte,'grid')
    grid on;
end
set(findall(gca, 'Type', 'Line'),'LineWidth',thickness);

% Determinacao dos limites do grafico.
sizeAxis = axis;
sizeAxis(1) = min(x);
sizeAxis(2) = max(x);
if min(y) ≠ max(y)
    sizeAxis(3) = min(y) - abs(max(y) - min(y))*0.1;
    sizeAxis(4) = max(y) + abs(max(y) - min(y))*0.1;
end
axis(sizeAxis);

% Salvamento do plot em um arquivo .png e um em um arquivo .fig.
if strcmp(fullScreen,'full')
    % Utilizado para maximizar a janela do plot
    set(gcf, 'Position', get(0, 'Screensize'));
end
if ¬isempty(saveName)
    if exist('results','dir') == 0
        mkdir('results');
```

```
        end
        if exist('results\figs','dir') == 0
            mkdir('results\figs');
        end
        print(strcat('results\', saveName), '-dpng');
        savefig(strcat('results\figs\', saveName));
    end
end
```

```
function completePlot(figureNumber,x,y,plotConfig,thickness,lableX,lableY,
plotTitle,gridCte,xMin,xMax,saveName,fullScreen,digital)
    % figureNumber - Numero da figura em que se encontra o plot
    % x - Valores de x
    % y - Valores de y
    % plotConfig - Configuracoes adicionais do plot. Se nao houverem
    %              configuracoes adicionais para o plot (cor da linha,
    %              tipo de linha...) o usuario deve inserir '' neste
    %              parametro
    % thickness - Espessura da linha do plot
    % lableX - Titulo do eixo X
    % lableY - Titulo do eixo Y
    % plotTitle - Titulo do plot
    % gridCte - Igual a 'grid' se o usuario desejar um grid no plot
    % xMax - Valor maximo de X, em porcentagem, a ser mostrado no plot
    % xMin - Valor minimo de X, em porcentagem, a ser mostrado no plot
    % saveName - Nome dos arquivos que conterao um print do plot. Se o
    %            usuario nao deseja salvar o plot num arquivo, deve inserir
    %            '' neste argumento.
    % fullScreen - Igual a 'full' se o usuario desejar o plot em tela
    %              inteira.
    % digital - Igual a 1 se o usuario quiser que os graficos sejam
    %           plotados em degraus (digital no tempo) e 0 caso contrario
```

```
% Instrucoes de utilizacao:
% 1 - Esta funcao da um hold on automatico, entao se voce chama-la
% utilizando o mesmo numero de figura 2 vezes, voce tera uma
% sobreposicao de plots no mesmo figure.
% 2 - Caso voce queira plotar varios graficos no mesmo plot para depois
% salvar em um arquivo, voce deve inserir o nome do arquivo no
% argumento saveName apenas no seu ultimo plot. Caso contrario, voce
% salvara uma figura que nao tera todos os seus graficos ainda.
% 3 - Caso voce queira plotar em tela inteira, com varios graficos no
% mesmo plot, voce nao precisa necessariamente inserir o parametro
% 'full' em todos os seus plots. Basta inserir em um deles.

% Verificacao dos limites de X.
if xMin > xMax
    display('Reveja os valores de xMin e xMax');
    return;
end

% Plot dos dados recebidos.
figure(figureNumber); hold on;
minIndex = round(xMin*length(x) + 1);
maxIndex = round(xMax*length(x));
x = x(minIndex:maxIndex);
y = y(minIndex:maxIndex);

if digital == 1
    xB = [];
    yB = [];
    for i = 1:length(x)-1
        xB = [xB linspace(x(i), x(i+1), 1000)];
        yB = [yB ones(1,1000)*y(i)];
    end

    x = xB;
    y = yB;
end
```

```
if isempty(plotConfig)
    plot(x,y);
else
    plot(x,y,plotConfig);
end

%Configuracoes de exibicao.
xlabel(lableX);
ylabel(lableY);
title(plotTitle);
if strcmp(gridCte,'grid')
    grid on;
end
set(findall(gca, 'Type', 'Line'),'LineWidth',thickness);

% Determinacao dos limites do grafico.
sizeAxis = axis;
sizeAxis(1) = min(x);
sizeAxis(2) = max(x);
if min(y) ≠ max(y)
    sizeAxis(3) = min(y) - abs(max(y) - min(y))*0.1;
    sizeAxis(4) = max(y) + abs(max(y) - min(y))*0.1;
end
axis(sizeAxis);

% Salvamento do plot em um arquivo .png e um em um arquivo .fig.
if strcmp(fullScreen,'full')
    % Utilizado para maximizar a janela do plot
    set(gcf, 'Position', get(0, 'Screensize'));
end
if ¬isempty(saveName)
    if exist('results','dir') == 0
        mkdir('results');
    end
    if exist('results\figs','dir') == 0
```

```
            mkdir('results\figs');
        end
        print(strcat('results\', saveName), '-dpng');
        savefig(strcat('results\figs\', saveName));
    end

end
```

```
function completeSubPlotDigital(figureNumber, m, n, a, x, y, ...
    plotConfig, thickness, lableX, lableY, plotTitle, gridCte, xMin, ...
    xMax, saveName, fullScreen, digital)
    % figureNumber - Numero da figura em que se encontra o plot
    % m - Numero de linhas do subplot
    % n - Numero de colunas do subplot
    % a - Numero do subplot no qual esta funcao inserira um grafico
    % x - Valores de x
    % y - Valores de y
    % plotConfig - Configuracoes adicionais do plot. Se nao houverem
    %              configuracoes adicionais para o plot (cor da linha,
    %              tipo de linha...) o usuario deve inserir '' neste
    %              parametro
    % thickness - Espessura da linha do plot
    % lableX - Titulo do eixo X
    % lableY - Titulo do eixo Y
    % plotTitle - Titulo do plot
    % gridCte - Igual a 'grid' se o usuario desejar um grid no plot
    % xMax - Valor maximo de X, em porcentagem, a ser mostrado no plot
    % xMin - Valor minimo de X, em porcentagem, a ser mostrado no plot
    % saveName - Nome dos arquivos que conterao um print do plot. Se o
    %            usuario nao deseja salvar o plot num arquivo, deve inserir
    %            '' neste argumento.
    % fullScreen - Igual a 'full' se o usuario desejar o plot em tela
    %              inteira.
```

```
% digital - Igual a 1 se o usuario quiser que os graficos sejam
%           plotados em degraus (digital no tempo) e 0 caso contrario

% Instrucoes de utilizacao:
% 1 - Esta funcao da um hold on automatico, entao se voce chama-la
% utilizando o mesmo numero de figura 2 vezes, voce tera uma
% sobreposicao de plots no mesmo figure.
% 2 - Caso voce queira plotar varios graficos no mesmo plot para depois
% salvar em um arquivo, voce deve inserir o nome do arquivo no
% argumento saveName apenas no seu ultimo plot. Caso contrario, voce
% salvara uma figura que nao tera todos os seus graficos ainda.
% 3 - Caso voce queira plotar em tela inteira, com varios graficos no
% mesmo plot, voce nao precisa necessariamente inserir o parametro
% 'full' em todos os seus plots. Basta inserir em um deles.

% Verificacao dos limites de X.
if xMin > xMax
    display('Reveja os valores de xMin e xMax');
    return;
end

% Plot dos dados recebidos.
figure(figureNumber); hold on;
minIndex = round(xMin*length(x) + 1);
maxIndex = round(xMax*length(x));
x = x(minIndex:maxIndex);
y = y(minIndex:maxIndex);

if digital == 1
    xB = [];
    yB = [];
    for i = 1:length(x)-1
        xB = [xB linspace(x(i), x(i+1), 1000)];
        yB = [yB ones(1,1000)*y(i)];
    end
```

```
    x = xB;
    y = yB;
end

subplot(m,n,a);
if isempty(plotConfig)
    plot(x,y);
else
    plot(x,y,plotConfig);
end

%Configuracoes de exibicao.
xlabel(lableX);
ylabel(lableY);
title(plotTitle);
if strcmp(gridCte,'grid')
    grid on;
end
set(findall(gca, 'Type', 'Line'),'LineWidth',thickness);

% Determinacao dos limites do grafico.

sizeAxis = axis;
sizeAxis(1) = min(x);
sizeAxis(2) = max(x);
if min(y) ≠ max(y)
    sizeAxis(3) = min(y) - abs(max(y) - min(y))*0.1;
    sizeAxis(4) = max(y) + abs(max(y) - min(y))*0.1;
end
axis(sizeAxis);

% Salvamento do plot em um arquivo .png e um em um arquivo .fig.
if strcmp(fullScreen,'full')
    % Utilizado para maximizar a janela do plot
    set(gcf, 'Position', get(0, 'Screensize'));
end
```

```
    if ¬isempty(saveName)
        if exist('results','dir') == 0
            mkdir('results');
        end
        if exist('results\figs','dir') == 0
            mkdir('results\figs');
        end
        print(strcat('results\', saveName), '-dpng');
        savefig(strcat('results\figs\', saveName));
    end
end
```

APÊNDICE III - CONSIDERAÇÕES SOBRE O MÉTODO DOS MÍNIMOS QUADRADOS

O **Método dos Mínimos Quadrados** foi utilizado na estimação dos parâmetros da planta. Este método propõe que o ajuste dos parâmetros de um dado modelo seja realizado de forma que este represente da melhor forma possível um conhecido conjunto de dados obtido experimentalmente. Ele foi desenvolvido por Karl Friedrich Gauss no final do século XVIII e afirma que os parâmetros de um modelo, inicialmente desconhecidos, devem ser determinados de forma que seja mínimo o quadrado da diferença entre os valores obtidos a partir deste modelo e aqueles obtidos experimentalmente (valores reais) (Astrom; Wittenmark, 1995).

O modelo nada mais é do que uma relação matemática, como em $(\mathrm{III}.1)$, que representa um certo conjunto de dados obtido experimentalmente e possibilita o calculo de uma aproximação $y_{est}(i)$ para a variável observada, a partir de valores conhecidos $\varphi(i)$ e parâmetros inicialmente desconhecidos θ. A determinação destes parâmetros deve ser realizada de forma que os valores estimados a partir do modelo $y_{est}(i)$ se aproximem o máximo possível dos valores reais $y(i)$.

$$y_{est}(i) = \varphi_1(i)\theta_1 + \varphi_2(i)\theta_2 + \varphi_3(i)\theta_3 + \cdots + \varphi_4(i)\theta_4 \tag{III.1}$$

Também chamada de **modelo de regressão**, $(\mathrm{III}.1)$ pode ser escrita de forma matricial, como mostrado em $(\mathrm{III}.2)$.

$$y_{est}(i) = \varphi^T(i)\theta \tag{III.2}$$

em que

$$\varphi^T(i) = \begin{bmatrix} \varphi_1(i) & \varphi_2(i) & \varphi_3(i) & \cdots & \varphi_n(i) \end{bmatrix} \tag{III.3}$$

$$\theta = \begin{bmatrix} \theta_1 & \theta_2 & \theta_3 & \cdots & \theta_n \end{bmatrix}^T \tag{III.4}$$

Para avaliar quão próximos os valores estimados se encontram dos valores reais, a função de custo dos mínimos $(\mathrm{III}.5)$ quadrados pode ser utilizada. O método propõe que sejam

escolhidos valores para θ que minimizem $V(\theta, t)$ (Astrom; Wittenmark, 1995).

$$V(\theta, t) = \frac{1}{2} \sum_{i=1}^{t} \left(y(i) - \varphi^T(i)\theta \right)^2 \tag{III.5}$$

Considerando que $y_{est}(i)$ seja linear nos parâmetros θ, é possível encontrar uma solução analítica que permita a determinação dos parâmetros θ. Para tal, considere as notações a seguir

$$\epsilon(i) = y(i) - \varphi^T(i)\theta \tag{III.6}$$

$$E(t) = \begin{bmatrix} \epsilon(1) & \epsilon(2) & \cdots & \epsilon(t) \end{bmatrix}^T \tag{III.7}$$

$$Y(t) = \begin{bmatrix} y(1) & y(2) & \cdots & y(t) \end{bmatrix}^T \tag{III.8}$$

$$E(t) = Y(t) - \Phi(t)\theta \tag{III.9}$$

$$\Phi(t) = \begin{bmatrix} \varphi^T(1) \\ \varphi^T(2) \\ \vdots \\ \varphi^T(t) \end{bmatrix} \tag{III.10}$$

Utilizando $(\mathrm{III}.9)$ é possível reescrever $(\mathrm{III}.5)$ como segue

$$V(\theta, t) = \frac{1}{2} \sum_{i=1}^{t} \epsilon^2(i) = \frac{1}{2} E(t)^T E(t) = \frac{1}{2} (Y(t) - \Phi(t)\theta)^T (Y(t) - \Phi(t)\theta) \tag{III.11}$$

Desenvolvendo $(\mathrm{III}.11)$ pode-se demonstrar que, para que o valor de $V(\theta, t)$ seja minimizado, os parâmetros θ devem ser calculados por (Astrom; Wittenmark, 1995).

$$\theta(t) = \left(\Phi^T(t)\Phi(t) \right)^{-1} \Phi^T(t) Y(t) \tag{III.12}$$

Utilizando $(\mathrm{III}.10)$ e $(\mathrm{III}.8)$ é possível reescrever $(\mathrm{III}.12)$ como se segue

$$\theta = \left(\sum_{i=1}^{t} \varphi(i)\varphi^T(i) \right)^{-1} \sum_{i=1}^{t} \varphi(i) y(i) \tag{III.13}$$

Definindo uma variável auxiliar $P(t)$

$$P(t) = \left(\Phi^T(t)\Phi(t)\right)^{-1} = \left(\sum_{i=1}^{t} \varphi(i)\varphi^T(i)\right)^{-1} \tag{III.14}$$

é possível reescrever $(\mathrm{III}.12)$ da seguinte forma

$$\theta(t) = P(t)\sum_{i=1}^{t} \varphi(i)y(i) \tag{III.15}$$

Para encontrar uma equação que permita que os valores de $\theta(t)$ sejam obtidos a partir dos valores de $\theta(t-1)$, a variável $P(t)$, introduzida em $(\mathrm{III}.14)$, será utilizada. Assim sendo, é necessário encontrar, primeiramente, uma equação que permita a computação da matriz $P(t)$ a partir dos valores de $P(t-1)$. Esta equação pode ser obtida a partir de $(\mathrm{III}.14)$ como mostrado a seguir

$$\begin{aligned}
P^{-1}(t) &= \Phi^T(t)\Phi(t) \Rightarrow \\
P^{-1}(t) &= \sum_{i=1}^{t} \varphi(i)\varphi^T(i) \Rightarrow \\
P^{-1}(t) &= \sum_{i=1}^{t-1} \varphi(i)\varphi^T(i) + \varphi(t)\varphi^T(t) \Rightarrow \\
P^{-1}(t) &= P^{-1}(t-1) + \varphi(t)\varphi^T(t)
\end{aligned} \tag{III.16}$$

Note que para determinar uma relação entre $P(t)$ e $P(t-1)$, bastou remover o último termo da somatória a partir da qual é definido $P(t)$. Aplicando o mesmo raciocínio em $(\mathrm{III}.15)$, onde é definido θ, a seguinte relação é obtida

$$\theta(t) = P(t)\left(\sum_{i=1}^{t-1} \varphi(i)y(i) + \varphi(t)y(t)\right) \tag{III.17}$$

Partindo de (III.15), e isolando o termo $\sum_{i=1}^{t} \varphi(i)y(i)$ segue que

$$\begin{aligned} \theta(t) = P(t) \sum_{i=1}^{t} \varphi(i)y(i) \Rightarrow \\ \sum_{i=1}^{t} \varphi(i)y(i) = P^{-1}(t)\theta(t) \Rightarrow \\ \sum_{i=1}^{t-1} \varphi(i)y(i) = P^{-1}(t-1)\theta(t-1) \end{aligned} \tag{III.18}$$

Isolando o termo $P(t-1)$ em (III.16)

$$P^{-1}(t-1) = P^{-1}(t) - \varphi(t)\varphi^T(t) \tag{III.19}$$

e substituindo em (III.18) o resultado obtido, é possível definir o somatório $\sum_{i=1}^{t-1} \varphi(i)y(i)$ em função de $P(t-1)$ e $\theta(t-1)$.

$$\begin{aligned} \sum_{i=1}^{t-1} \varphi(i)y(i) = \left(P^{-1}(t) - \varphi(t)\varphi^T(t)\right)\theta(t-1) \Rightarrow \\ \sum_{i=1}^{t-1} \varphi(i)y(i) = P^{-1}(t)\theta(t-1) - \varphi(t)\varphi^T(t)\theta(t-1) \end{aligned} \tag{III.20}$$

Por fim, substituindo (III.20) em (III.17), é possível definir $\theta(t)$ a partir de $\theta(t-1)$, $P(t)$ e $\varphi(t)$.

$$\begin{aligned} \theta(t) = P(t)\left(P^{-1}(t)\theta(t-1) - \varphi(t)\varphi^T(t)\theta(t-1) + \varphi(t)y(t)\right) \Rightarrow \\ \theta(t) = \theta(t-1) - P(t)\varphi(t)\varphi^T(t)\theta(t-1) + P(t)\varphi(t)y(t) \Rightarrow \\ \theta(t) = \theta(t-1) + P(t)\varphi(t)\left(y(t) - \varphi^T(t)\theta(t-1)\right) \end{aligned} \tag{III.21}$$

Definindo-se

$$\begin{aligned} \varepsilon(t) = y(t) - \varphi^T(t)\theta(t-1) \\ K(t) = P(t)\varphi(t) \end{aligned} \tag{III.22}$$

é possível reescrever $(\mathrm{III}.21)$ como segue

$$\theta(t) = \theta(t-1) + K(t)\varepsilon(t) \tag{III.23}$$

Perceba que a cada iteração, o valor de $\theta(t)$ é atualizado com base nos valores de $\theta(t-1)$. Além disso, é interessante observar que a amplitude dos ajustes aplicados a $\theta(t)$ está diretamente relacionada ao erro de estimação $\varepsilon(t)$, e que este, por sua vez, é ponderado pela matriz $K(t)$.

Para computar $\theta(t)$ a cada iteração, utilizando $(\mathrm{III}.21)$, é necessário também computar $P(t)$. Assim sendo, é preciso encontrar uma relação que permita que o cálculo de $P(t)$ seja realizado com base nos valores de $P(t-1)$. Esta por sua vez será obtida a partir de $(\mathrm{III}.16)$ como mostrado a seguir

$$\begin{gathered} P^{-1}(t) = P^{-1}(t-1) + \varphi(t)\varphi^T(t) \Rightarrow \\ P(t) = \left(P^{-1}(t-1) + \varphi(t)\varphi^T(t)\right)^{-1} = \left(P^{-1}(t-1) + \varphi(t)I\varphi^T(t)\right)^{-1} \end{gathered} \tag{III.24}$$

Note que a matriz identidade pode ser inserida no segundo termo da equação de forma que a igualdade se mantenha. Para obter a relação final de $P(t)$, será necessário utilizar o Teorema 1, introduzido a seguir (Astrom; Wittenmark, 1995).

Teorema 1 *Sejam A, C e $C^{-1} + DA^{-1}B$ matrizes quadradas não singulares, é possível afirmar que a $A + BCD$ é invertível e que sua inversa é dada por*

$$(A + BCD)^{-1} = A^{-1} - A^{-1}B(C^{-1} + DA^{-1}B)^{-1}DA^{-1} \tag{III.25}$$

Para aplicar o Teorema 1 à $(\mathrm{III}.24)$, considere que

$$\begin{aligned} A &= P^{-1}(t-1) \\ B &= \varphi(t) \\ C &= I \\ D &= \varphi^T(t) \end{aligned} \tag{III.26}$$

A aplicação do Teorema 1 permite que $(\mathrm{III}.24)$ seja reescrita como mostrado a seguir

$$P(t) = P(t-1) - P(t-1)\varphi(t)\left(I + \varphi^T(t)P(t-1)\varphi(t)\right)^{-1}\varphi^T(t)P(t-1) \qquad (\mathrm{III}.27)$$

A aplicação de $(\mathrm{III}.27)$ permite que os valores de $P(t)$ sejam calculados com base nos valores de $P(t-1)$.

Através dos desenvolvimentos matemáticos apresentados até então, é possível concluir que, para implementar o Método dos Mínimos Quadrados Recursivo (*Recursive Least Square* ou RLS) realizando a obtenção de $P(t)$ e, consequentemente, de $\theta(t)$ com base nos valores de $P(t-1)$ e $\theta(t-1)$, as seguintes equações devem ser utilizadas

$$\begin{gathered} P(t) = P(t-1) - P(t-1)\varphi(t)\left(1 + \varphi^T(t)P(t-1)\varphi(t)\right)^{-1}\varphi^T(t)P(t-1) \\ K(t) = P(t)\varphi(t) \\ \varepsilon(t) = y(t) - \varphi^T(t)\theta(t-1) \\ \theta(t) = \theta(t-1) - K(t)\varepsilon(t) \end{gathered} \qquad (\mathrm{III}.28)$$

Note que, como a computação de $\varphi^T(t)P(t-1)\varphi(t)$ resulta em um número real, a operação $\left(I + \varphi^T(t)P(t-1)\varphi(t)\right)$ pode ser substituída por $\left(1 + \varphi^T(t)P(t-1)\varphi(t)\right)$, já que a matriz I teria neste caso uma única linha e uma única coluna, e conteria apenas o número 1.

É importante ressaltar que a dimensão de $P(t)$ está diretamente relacionada à quantidade de parâmetros a serem estimados, ou seja, à dimensão de θ. Caso o modelo proposto possua n parâmetros a serem determinados, a matriz $P(t)$ terá n linhas e n colunas.

APÊNDICE IV - CONSIDERAÇÕES ADICIONAIS SOBRE O MÉTODO DO POSICIONAMENTO DOS POLOS

Condições de causalidade

Um sistema é dito **causal** ou **não antecipativo**, se o valor de sua saída for dependente apenas dos valores de suas entradas e saída passadas, e do valor atual de sua entrada (Hayes, 2011). O sistema

$$y(t) = y(t-1) + y(t-2) + u(t) + u(t-1) + u(t-2) \tag{IV.1}$$

por exemplo, é causal, uma vez que sua saída $y(t)$ depende apenas dos valores de suas saídas passadas, $y(t-1)$ e $y(t-2)$, dos valores de suas entradas passadas, $u(t-1)$ e $u(t-2)$ e do valor atual de sua entrada $u(t)$. Por outro lado, o sistema

$$y(t) = y(t-1) + u(t) + u(t+2) \tag{IV.2}$$

por exemplo, não é causal, já que depende de uma entrada futura $u(t+2)$. Assim sendo, é necessário não somente encontrar os parâmetros do controlador aqui proposto com base nas equações apresentadas na seção anterior, mas também garantir que o controlador projetado seja causal, para que seja possível implementá-lo.

Para que a lei de controle seja causal, é necessário que

$$degS \leq degR \tag{IV.3}$$

$$degT \leq degR \tag{IV.4}$$

Para entender melhor a origem destas condições, considere (2.49) reescrita da seguinte forma

$$u(t) = u_{ff}(t) - u_{fb}(t) \tag{IV.5}$$

onde

$$u_{ff}(t) = \frac{T}{R}u_c(t) \tag{IV.6}$$

$$u_{fb}(t) = \frac{S}{R}y(t) \tag{IV.7}$$

Para verificar as condições de causalidade, suponha agora um controlador em que $S = q^3 + q^2 + q + 1$ e $R = q^2 + q + 1$. Note que, neste caso, $degS > degR$, o que significa que este controlador desrespeita uma das condições de causalidade. Assim sendo, deve ser possível mostrar que este controlador não é causal. Para tal, é necessário avaliar apenas $u_{fb}(t)$.

$$\begin{aligned}
u_{fb}(t) &= -\frac{S}{R}y(t) \Rightarrow \\
Ru_{fb}(t) &= -Sy(t) \Rightarrow \\
(q^2 + q + 1)u_{fb}(t) &= -(q^3 + q^2 + q + 1)y(t) \Rightarrow \\
(1 + q^{-1} + q^{-2})u_{fb}(t) &= -(q^1 + 1 + q^{-1} + q^{-2})y(t) \Rightarrow \\
u_{fb}(t) + q^{-1}u_{fb}(t) + q^{-2}u_{fb}(t) &= -q^1y(t) - y(t) - q^{-1}y(t) - q^{-2}y(t) \Rightarrow \\
u_{fb}(t) + u_{fb}(t-1) + u_{fb}(t-2) &= -y(t+1) - y(t) - y(t-1) - y(t-2) \Rightarrow \\
u_{fb}(t) &= -u_{fb}(t-1) - u_{fb}(t-2) - y(t+1) - y(t) - y(t-1) - y(t-2)
\end{aligned} \tag{IV.8}$$

Note que, neste caso, $u_{fb}(t)$ depende de $y(t+1)$, o que significa que $u_{fb}(t)$ não é causal, e que, consequentemente, $u(t)$ também não é. Perceba que bastaria que $degR$ fosse maior que $degS$, para que $u_{fb}(t)$ fosse causal. Assim sendo, é possível verificar que, caso alguma das condições de causalidade seja desrespeitada, o controlador obtido não será causal e não poderá ser implementado. Uma análise similar pode ser realizada para $u_{ff}(t)$.

Solução de grau mínimo

Como foi discutido anteriormente, a obtenção dos parâmetros do controlador aqui proposto depende diretamente da solução da *Equação de Diphantine*. O problema é que esta equação possui infinitas soluções. Por exemplo, se R^0 e S^0 forem soluções da Equação de

Diphantine, R e S também serão, caso sejam definidas por

$$R = R^0 + QB \tag{IV.9}$$

$$S = S^0 - QA \tag{IV.10}$$

em que Q é um polinômio arbitrário. De fato, se R^0 e S^0 são soluções, é válida a igualdade $AR^0 + BS^0 = A_c$. Assim sendo

$$\begin{aligned} A(R^0 + QB) + B(S^0 - QA) &= AR^0 + ABQ + BS^0 - ABQ \\ &= AR^0 + BS^0 \\ &= A_c \end{aligned} \tag{IV.11}$$

o que mostra que também é válida a igualdade $A(R^0+QB)+B(S^0-QA) = A_c$ e que, portanto, $R = R^0 + QB$ e $S = S^0 - QA$ são soluções de *Equação de Diophantine* desde que R^0 e S^0 também sejam.

Apesar de existirem infinitas soluções possíveis para tal equação, é comumente mais vantajoso escolher a solução de menor grau, já que implementar um controlador de alta ordem é complexo e demanda alto gasto computacional. Para encontrar a solução de menor ordem para o controlador aqui proposto, é necessário avaliar o grau de todos os polinômios apresentados até o momento. Antes disso, é preciso compreender como o grau destes polinômios pode ser analisado a partir das relações vistas até agora. A título de exemplo, considere um sistema arbitrário onde

$$\begin{aligned} A &= q^3 + q^2 + q + 1 \\ B &= q^2 + q + 1 \\ R &= q + 1 \\ S &= 1 \end{aligned} \tag{IV.12}$$

É possível, com base nestas considerações, realizar o cálculo de A_c a partir de (2.54). Calculando inicialmente o produto AR segue que

$$AR = (q^3 + q^2 + q + 1)(q + 1) = q^4 + 2q^3 + 2q^2 + q + 1 \tag{IV.13}$$

Procedendo de forma análoga para BS

$$BS = (q^2 + q + 1)(1) = q^2 + q + 1 \tag{IV.14}$$

é possível observar que $degBS = degB + degS$. Perceba que, neste caso, $degS = 0$. De posse dos valores de AR e BS, já é possível calcular A_c.

$$\begin{aligned} A_c &= AR + BS \\ &= (q^4 + 2q^3 + 2q^2 + q + 1) + (q^2 + q + 1) \\ &= q^4 + 2q^3 + 3q^2 + 3q + 2 \end{aligned} \tag{IV.15}$$

Note que o grau de A_c é igual ao grau de AR, uma vez que AR possui grau maior que BS. Assim sendo, de forma mais geral, é possível afirmar que, se $Z = X + Y$, então $degZ = max\{degX, degY\}$.

Desta forma, a determinação do grau de um polinômio gerado a partir da combinação de outros polinômios pode ser realizada através da utilização do Teorema 2.

Teorema 2 *Sejam Z, X, e Y polinômios quaisquer, é possível afirmar que*

(i) $Z = XY \Rightarrow degZ = degX + degY$

(ii) $Z = X + Y \Rightarrow degZ = max\{degX, degY\}$

De posse das relações apresentadas no Teorema 2 é possível analisar agora o grau de A_c. Inicialmente, lembre-se que

$$A_c = AR + BS \tag{IV.16}$$

e que os graus dos polinômios A e B são dados por

$$degA = n \tag{IV.17}$$

$$degB = degA - d_0 \tag{IV.18}$$

o que implica que $degA > degB$. Além disso, uma das condições de causalidade exige que

$degR \geq degS$. Desta forma, segue que

$$degAR = degA + degR > degBS = degB + degS \Rightarrow$$
$$degA_c = degAR = degA + degR \Rightarrow$$
$$degR = degA_c - degA \tag{IV.19}$$

Volte agora sua atenção para o grau do polinômio S. Como há para S infinitas soluções, dadas por $S = S_0 - QA$, onde Q é um polinômio qualquer e S_0 é uma solução qualquer para a *Equação de Diophantine*, então sempre é possível escolher Q de forma que seja encontrada uma solução S tal que $degS < degA$. Por exemplo, supondo $S_0 = q^2 + q + 1$ e $A = q + 1$. Escolhendo $Q = q$ seria possível obter

$$S = S_0 - QA \Rightarrow S = (q^2 + q + 1) - q(q + 1) = 1$$

Note que, para $Q = q$, $degS = 0$ e $degA = 1$. Fica mais claro a partir deste exemplo que sempre é possível propor um valor para Q que garanta uma solução S onde $degS < degA$, ou em outros termos onde

$$degS \leq degA - 1 \tag{IV.20}$$

Lembre-se que a condição de causalidade apresentada em $(\mathrm{IV}.3)$ pede que $degS \leq degR$. Para que esta condição seja sempre satisfeita, basta garantir que

$$degR \geq degA - 1 \tag{IV.21}$$

De fato, analisando $(\mathrm{IV}.20)$ e considerando que $degR \geq degA - 1$, é possível concluir que, neste caso, $degS \leq degR$.

$$degS \leq degA - 1 \leq degR \Rightarrow degS \leq degR \tag{IV.22}$$

Combinando $(\mathrm{IV}.19)$ e $(\mathrm{IV}.21)$ é possível encontrar uma relação entre os graus dos

polinômios A e A_c, como mostrado a seguir.

$$degA - 1 \leq degR \Rightarrow \tag{IV.23}$$

$$degA - 1 \leq degAc - degA \Rightarrow \tag{IV.24}$$

$$degA_c \geq 2degA - 1 \tag{IV.25}$$

Uma vez satisfeita a primeira das condições de causalidade, é necessário satisfazer a segunda, introduzida em $(\mathrm{IV}.4)$, e garantir que $degT \leq degR$. Aplicando o Teorema 2 à (2.59), segue que

$$degT = degA_0 + degB'_m \tag{IV.26}$$

Combinando agora $(\mathrm{IV}.4)$, $(\mathrm{IV}.19)$ e $(\mathrm{IV}.26)$, segue que

$$degT \leq degR \Rightarrow$$

$$degA_0 + degB'_m \leq degA_c - degA \tag{IV.27}$$

Aplicando agora o Teorema 2 à (2.56)

$$degA_c = degA_0 + degA_m + degB^+ \Rightarrow degA_0 = degA_c - degA_m - degB^+ \tag{IV.28}$$

e substituindo o resultado obtido em $(\mathrm{IV}.27)$, segue que

$$degA_0 + degB'_m \leq degA_c - degA \Rightarrow$$

$$degA_c - degA_m - degB^+ + degB'_m \leq degA_c - degA \Rightarrow$$

$$-degA_m + degB'_m \leq -degA + degB^+ \Rightarrow$$

$$degA_m - degB'_m \geq degA - degB^+ \tag{IV.29}$$

Aplicando novamente o Teorema 2, agora à (2.55) e (2.57) segue que

$$degB = degB^+ + degB^- \tag{IV.30}$$

$$degB_m = degB^- + degB'_m \tag{IV.31}$$

Por fim, subtraindo $degB^-$ dos dois lados de $(\mathrm{IV}.29)$ e combinando o resultado obtidos àqueles apresentados em $(\mathrm{IV}.30)$ e $(\mathrm{IV}.31)$, é possível obter a relação introduzida em $(\mathrm{IV}.32)$.

$$degA_m - degB'_m \geq degA - degB^+ \Rightarrow$$
$$degA_m - degB'_m - degB^- \geq degA - degB^+ - degB^- \Rightarrow$$
$$degA_m - \left(degB'_m - degB^-\right) \geq degA - \left(degB^+ + degB^-\right) \Rightarrow$$
$$degA_m - degB_m \geq degA - degB = d_0 \tag{IV.32}$$

Então para que a condição de causalidade $(\mathrm{IV}.4)$ seja respeitada, é necessário que o *delay* do modelo de referência seja maior que o do processo a ser controlado.

Assim sendo, as condições de causalidade podem ser reescritas como mostrado a seguir

$$degS \leq degR \Rightarrow degA_c \geq 2degA - 1 \tag{IV.33}$$
$$degT \leq degR \Rightarrow degA_m - degBm \geq d_0 \tag{IV.34}$$

Caso estas relações sejam respeitadas, as condições de causalidade também o serão, e o controlador obtido pelo Método do Posicionamento dos Polos será causal e poderá ser implementado (Astrom; Wittenmark, 1995).

No entanto, é preciso ter em mente que o objetivo aqui é encontrar o controlador de menor grau possível, que tenha $degR$, $degS$ e $degT$ mínimos. Como é necessário que $degR \geq degS$ e que $degR \geq degT$, R apresenta o menor grau possível quando $degR = degS$ e $degR = degT$. Assim sendo, no projeto do controlador de grau mínimo, deve ser respeitada a relação

$$degR = degS = degT \tag{IV.35}$$

Além disso, $(\mathrm{IV}.33)$ e $(\mathrm{IV}.34)$ devem ser reformuladas

$$degA_c = 2degA - 1 \tag{IV.36}$$
$$degA_m - degB_m = degA - degB = d_0 \tag{IV.37}$$

Para satisfazer $(\mathrm{IV}.37)$ basta que

$$degA_m = degA \tag{IV.38}$$

$$degB_m = degB \tag{IV.39}$$

De $(\mathrm{IV}.28)$ segue que $degA_c = degA_0 + degA_m + degB^+$. Combinando $(\mathrm{IV}.28)$ e $(\mathrm{IV}.36)$, e considerando que $degA_m = degA$, como proposto acima

$$degA_c = 2degA - 1 \Rightarrow$$

$$degA_0 + degA_m + degB^+ = 2degA - 1 \Rightarrow$$

$$degA_0 + degA + degB^+ = 2degA - 1 \Rightarrow$$

$$degA_0 = degA - degB^+ - 1 \tag{IV.40}$$

Após todas as considerações apresentadas nesta seção, é possível concluir que, para projetar um controlador de grau mínimo, que seja causal, utilizando o Método do Posicionamento dos Polos, é necessário satisfazer as seguintes relações

$$degA_m = degA \tag{IV.41}$$

$$degB_m = degB \tag{IV.42}$$

$$degA_0 = degA - degB^+ - 1 \tag{IV.43}$$

Por fim, uma relação entre o grau do controlador e o grau do processo a ser controlado pode ser obtida através da combinação de $(\mathrm{IV}.19)$, $(\mathrm{IV}.35)$ e $(\mathrm{IV}.36)$, como segue

$$degR = degA_c - degA \Rightarrow$$

$$degR = (2degA - 1) - degA \Rightarrow$$

$$degR = degS = degT = degA - 1 \tag{IV.44}$$

A Equação 2.66 indica que a ordem do controlador de grau mínimo, obtido a partir do Método do Posicionamento dos Polos, será sempre uma unidade menor que a ordem do processo

a ser controlado. Este, por sua vez, é representado pela função de transferência $B(q)/A(q)$, e tem sua ordem definida a partir de $degA$, uma vez que $degA > degB$.