UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
ESCOLA DE ENGENHARIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA
CURSO DE ENGENHARIA ELÉTRICA

LEONEL ACUNHA GUIMARÃES

IMPLEMENTAÇÃO EMBARCADA DO MÉTODO IFT

Porto Alegre
2013

LEONEL ACUNHA GUIMARÃES

IMPLEMENTAÇÃO EMBARCADA DO MÉTODO IFT

Projeto de diplomação apresentado no curso de Engenharia Elétrica, da Universidade Federal do Rio Grande do Sul, como parte dos requisitos para a obtenção do título de Engenheiro Eletricista.

Área de concentração: Sistemas de controle.

ORIENTADOR: Prof. Dr. Alexandre Sanfelice Bazanella

Porto Alegre

2013

LEONEL ACUNHA GUIMARÃES

IMPLEMENTAÇÃO EMBARCADA DO MÉTODO IFT

Esta dissertação foi julgada adequada para a obtenção do título de Engenheiro Eletricista e aprovada em sua forma final pelo Orientador e pela Banca Examinadora.

Orientador: _____

Prof. Dr. Alexandre Sanfelice Bazanella, UFRGS

Doutor pela Universidade Federal de Santa Catarina – Florianólolis, Brasil

Aprovado em: 16/12/2013

Banca Examinadora:

_____

Prof. Dr. Alexandre Sanfelice Bazanella, UFRGS

Doutor pela Universidade Federal de Santa Catarina – Florianólolis, Brasil

_____

Prof. Dr. Luis Fernando Alves Pereira, UFRGS

Doutor pelo Instituto Tecnológico de Aeronáutica– São José dos Campos, Brasil

_____

Profª. Drª. Luciola Campestrini, UFRGS

Doutora pela Universidade Federal do Rio Grande do Sul – Porto Alegre, Brasil

Porto Alegre, dezembro de 2013.

DEDICATÓRIA

Dedico este trabalho aos meus pais Leonel José Oliveira Guimarães e Ivana Maria Salum Acunha Guimarães pelo apoio incondicional ao longo de todos os últimos 23 anos. A chegada ao estágio final desse curso só foi possível graças à dedicação oferecida ao longo da graduação, permitindo que minha concentração sempre fosse toda voltada ao estudo. Os dois merecem parte do título que postulo com este trabalho.

AGRADECIMENTOS

Agradeço primeiramente aos meus pais pela ajuda nos momentos mais críticos do curso. Aos meus colegas Robertinho, Pablo, Tales, Bruno e Mario pelas noites viradas estudando ou fazendo algum trabalho e as infinitas cervejas bebidas em bares de qualidade duvidosa. Aos meus colegas na França, Otto, Jonatan, Xabi e Gustavo, cuja companhia me fez sentir em casa. Agradeço também aos amigos Guilherme, Alexande e Leandro pela parceria.

Sinceros agradecimentos aos professores Luís Fernando Alves Pereira e Alexandre Sanfelice Bazanella que me ofereceram oportunidades determinantes tanto para minha formação quanto para minha vida.

Por fim, agradeço à UFRGS por toda a estrutura oferecida, aos demais professores pelos ensinamentos e à CAPES pelo aporte financeiro e suporte ao longo da graduação.

“Não dá para controlar

Não dá!”

(Lobão)

RESUMO

Este documento descreve a implementação embarcada de um sistema de controle com auto ajuste de parâmetros baseado no método iterativo de ajuste -*iterative feedback tuning* (IFT). O trabalho consiste na análise do método, simulações computacionais e implementação embarcada de um controlador proporcional integral derivativo (PID) com auto ajuste de parâmetros num kit de desenvolvimento Arduino Due a fim de controlar uma planta definida arbitrariamente. Os resultados práticos apresentados mostram que o dispositivo utilizado de forma embarcada apresenta respostas muito semelhantes às obtidas em simulação, validando, portanto o projeto.

Palavras-chaves: Sistemas de controle. Controle digital. Controle baseado em dados.

ABSTRACT

This document aims to describe the embedded implementation of a control system with parameters auto tuning based on iterative feedback tuning method (IFT). The work consists in analysis, computational simulations and the embedded implementation of a proportional integral derivative controller (PID) with auto tuning on an Arduino Due development kit in order to control a plant chosen arbitrarily by the user. The practical results presented shows that the embedded device presents a behavior very similar to the one obtained in simulation, thus validating the project.

Keywords: Control systems. Digital control. Data-driven control.

SUMÁRIO

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

$C(z)$: Utilizado para representar a função de transferência do controlador

E: Escrita

f_s: Frequência de amostragem

$e(t)$: Utilizado para representar o sinal de erro entre a saída e a referência do sistema

$G(z)$: Utilizado para representar a função de transferência do processo

i: Índice da iteração

IFT: "*Iterative feedback tuning"* (método iterativo de auto ajuste de parâmetros)

$J(\rho)$: Utilizado para representar a função custo do sistema

L: Leitura

k_p: Utilizado para designar o ganho proporcional do controlador

k_i: Utilizado para designar o ganho integral do controlador

k_d: Utilizado para designar o ganho derivativo do controlador

LTI: *"Linear time-invariant"*(linear e invariante no tempo)

P: Polo da função transferência

P_r: Processamento

PID: Proporcional integral derivativo

$r(t)$: Utilizado para representar o sinal de referência do sistema

SISO: *"Single input, single output"* (um sinal de saída e um sinal de entrada)

t_s: Tempo de acomodação

T_s: Período de amostragem

$y(t)$: Utilizado para representar o sinal de saída do sistema

Z: Zero da função transferência

γ: Fator proporcional à norma dos parâmetros do controlador

ρ: Utilizado para designar um parâmetro do controlador

1 INTRODUÇÃO

Muitos métodos de auto ajuste de parâmetros de controladores e de controle adaptativo vêm sendo estudados ao longo dos anos. Dentre estes métodos há o método IFT – *iterative feedback tuning*-, que é um método iterativo de ajuste de parâmetros do controlador, ou seja, sistematicamente são realizados ensaios a fim de atualizar o valor destes parâmetros de maneira a aproximar a resposta do sistema a uma resposta idealizada (ÅSTRÖM; WITTENMARK, 1994. BAZANELLA;CAMPESTRINI;ECKHARD, 2012).

A vantagem na utilização do método IFT é de se obter um conjunto de parâmetros para o controlador sem necessariamente conhecer o modelo do processo que se deseja controlar. Essa abordagem pode ser utilizada também em processos cujo comportamento físico é sensível às características do meio onde esse se encontra, ou seja, de acordo com condições climáticas ou outras propriedades físicas do meio, o modelo matemático da planta poderá ser diferente do que o obtido outrora, desta forma, seria necessária a realização do projeto do controlador cada vez que houvesse uma mudança física no meio. O método IFT realiza a atualização dos valores dos parâmetros do controlador de maneira automática, mantendo o comportamento do sistema de controle tal qual o obtido nas demais condições físicas.

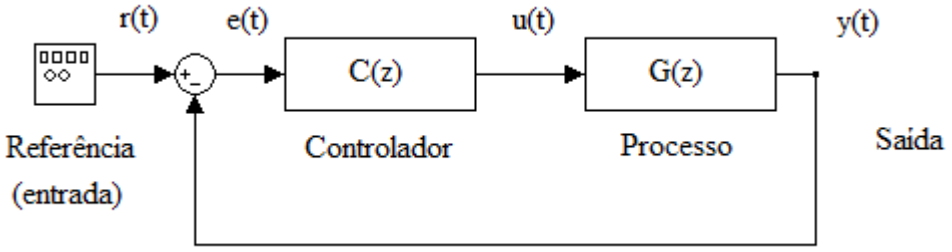
O escopo deste trabalho é a implementação embarcada do método IFT em um kit de desenvolvimento Arduino. Para isso, se faz necessário o estudo funcional do método através de simulações computacionais com objetivo de avaliar fatores que podem influenciar no funcionamento do método. Após, é possível a implementação embarcada do algoritmo no Arduino com intuito de controlar algum processo real e verificar sua eficiência.

2 ABORDAGEM TEÓRICA

2.1 TOPOLOGIA DO SISTEMA DE CONTROLE

A topologia de controle adotada ao longo do projeto visa controlar plantas lineares e invariantes no tempo (LTI) com um sinal de saída e um de entrada (SISO). A Figura 1 apresenta a topologia adotada ao longo do trabalho bem como as variáveis do sistema.

Figura 1. Topologia do sistema de controle.



Observa-se na Figura 1 que uma referência é utilizada como entrada do sistema em laço fechado. Essa referência é comparada à saída do processo, e o sinal de erro obtido $e(t) = r(t) - y(t)$ é utilizado como entrada do controlador. A saída do controlador é chamada sinal de controle e descrito por $u(t)$. Este sinal de controle $u(t)$ é utilizado como entrada do processo, gerando, portanto a saída $y(t)$.

2.2 CONTROLADOR PID

Controladores PID são amplamente utilizados na indústria, pois unem as vantagens das ações proporcional, integral e derivativa. Desta forma, este tipo de controlador age para corrigir o erro instantâneo (proporcional), futuro (derivativo) e acumulado (integral) entre a saída do sistema e a referência.

O controlador PID pode ser expresso através do somatório dos controladores (proporcional, integrador, derivativo) multiplicados pelos respectivos ganhos. O modelo

matemático mais conhecido do controlador é apresentado em (1) e o modelo utilizado durante o desenvolvimento do trabalho é apresentado em (6) onde há o mesmo denominador. Em (3), (4) e (5) é mostrado como obter os parâmetros ρ a partir dos ganhos $k_p, k_i \; e \; k_d$.

$$C(z,k) = [k_p \quad k_i \quad k_d]\begin{bmatrix} 1 \\ \frac{z}{z-1} \\ \frac{z-1}{z} \end{bmatrix} \tag{1}$$

$$= k_p + k_i\left(\frac{z}{z-1}\right) + k_d\left(\frac{z-1}{z}\right)$$

$$= \frac{k_p(z-1)z + k_i z^2 + k_d(z-1)^2}{z(z-1)}$$

$$= \frac{k_p(z^2-z) + k_i z^2 + k_d(z^2-2z+1)}{z^2-z}$$

$$= \frac{z^2(k_p + k_i + k_d) + z(-k_p - 2k_d) + k_d}{z^2-z} \tag{2}$$

$$\rho_1 \triangleq (k_p + k_i + k_d) \tag{3}$$

$$\rho_2 \triangleq (-k_p - 2k_d) \tag{4}$$

$$\rho_3 \triangleq k_d \tag{5}$$

$$\therefore$$

$$C(z,\rho) = [\rho_1 \quad \rho_2 \quad \rho_3]\begin{bmatrix} \frac{z^2}{z^2-z} \\ \frac{z}{z^2-z} \\ \frac{1}{z^2-z} \end{bmatrix} \tag{6}$$

$$= \rho^T C(z) \tag{7}$$

2.3 MÉTODO IFT

O método IFT tem por objetivo atualizar os ganhos do controlador de maneira iterativa a fim de obter a melhor aproximação possível de um sistema desejado cuja função transferência é $T_d(z)$. Para isso, busca-se minimizar a função custo $J(\rho)$, apresentado em (8), onde $y_1(t,\rho)$ é a saída do sistema controlado com os parâmetros ρ, e $y_d(t)$ é a saída desejada do sistema.

$$J(\rho) = \bar{E}[y_1(t,\rho) - y_d(t)]^2 \tag{8}$$

A atualização do valor dos parâmetros do controlador pode ser realizada buscando o ponto de mínimo da função custo $J(\rho)$. Tal procedimento pode ser realizado através do método *"Steepest Descent"*. Desta forma, o objetivo é calcular o gradiente da função custo $\nabla J(\rho)$ para obter uma minimização de $J(\rho)$ na próxima iteração. O novo parâmetro é calculado através do parâmetro anterior subtraído do valor do gradiente ponderado por um fator γ proporcional à norma do conjunto de parâmetros como mostrado em (9). Os índices $i+1$ e i denotam a ordem da iteração.

$$\rho^{i+1} = \rho^{i} - \gamma^{i}\nabla J(\rho^{i}) \tag{9}$$

O gradiente $\nabla J(\rho)$ pode ser calculado por (10) (BAZANELLA; CAMPESTRINI; ECKHARD, 2012).

$$\nabla J(\rho) = \frac{2}{N}\sum_{t=1}^{N}[y_1(t,\rho) - y_d(t)]\frac{\partial y_1(t,\rho)}{\partial \rho} \tag{10}$$

O termo $\frac{\partial y_1(t,\rho)}{\partial \rho}$ pode ser aproximado por (11), onde $y_2(t,\rho)$ é a saída do mesmo sistema controlado com os mesmos parâmetros ρ quando a entrada do sistema $r_2(t)$ é dada pelo erro obtido $(r_1(t) - y_1(t,\rho))$ quando esse sistema possui entrada $r_1(t)$ e saída $y_1(t,\rho)$.

$$\overline{\frac{\partial y_1(t,\rho)}{\partial \rho}} = \frac{1}{C(z,\rho)} \frac{\partial C(z,\rho)}{\partial \rho} y_2(t,\rho) \tag{11}$$

$$r_2(t,\rho) = r_1(t) - y_1(t,\rho) \tag{12}$$

Dessa forma, é necessário o calculo e simulação do sistema para obter os fatores $\frac{1}{C(z,\rho)} \frac{\partial C(z,\rho)}{\partial \rho}, y_1(t,\rho), y_d(t),\ y_2(t,\rho), \overline{\frac{\partial y_1(t,\rho)}{\partial \rho}}$ *e* $\nabla J(\rho)$ para atualizar o parâmetro ρ.

Considerando o controlador PID, o termo $\frac{\partial C(z,\rho)}{\partial \rho}$ é dado por (13) para o parâmetro ρ_1, (14) para ρ_2 e (15) para ρ_3.

$$\frac{\partial C(z,\rho)}{\partial \rho_1} = \frac{z^2}{z^2 - z} \tag{13}$$

$$\frac{\partial C(z,\rho)}{\partial \rho_2} = \frac{z}{z^2 - z} \tag{14}$$

$$\frac{\partial C(z,\rho)}{\partial \rho_3} = \frac{1}{z^2 - z} \tag{15}$$

Dividindo-os por $C(z,\rho)$, obtém-se (16), (17) e (18).

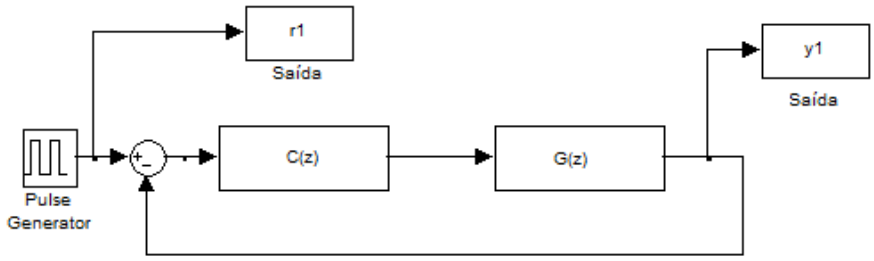
$$\frac{1}{C(z,\rho)}\frac{\partial C(z,\rho)}{\partial \rho_1} = \left[\frac{z^2 - z}{\rho_1 z^2 + \rho_2 z + \rho_3}\right]\frac{z^2}{z^2 - z} = \frac{z^2}{\rho_1 z^2 + \rho_2 z + \rho_3} \quad (16)$$

$$\frac{1}{C(z,\rho)}\frac{\partial C(z,\rho)}{\partial \rho_2} = \left[\frac{z^2 - z}{\rho_1 z^2 + \rho_2 z + \rho_3}\right]\frac{z}{z^2 - z} = \frac{z}{\rho_1 z^2 + \rho_2 z + \rho_3} \quad (17)$$

$$\frac{1}{C(z,\rho)}\frac{\partial C(z,\rho)}{\partial \rho_3} = \left[\frac{z^2 - z}{\rho_1 z^2 + \rho_2 z + \rho_3}\right]\frac{1}{z^2 - z} = \frac{1}{\rho_1 z^2 + \rho_2 z + \rho_3} \quad (18)$$

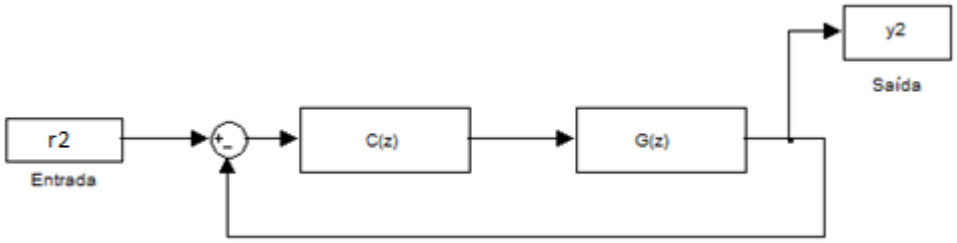
Desse modo, para cada iteração são necessários dois ensaios distintos e três filtragens (para o caso do PID). O primeiro deles tem por finalidade a obtenção do termo $y_1(t,\rho)$ que é a saída do sistema com o controlador em laço fechado. Na Figura 2 é apresentada a estrutura do sistema indicando os sinais que devem ser registrados no ensaio (r_1 e y_1).

Figura 2. Primeiro ensaio do método IFT.



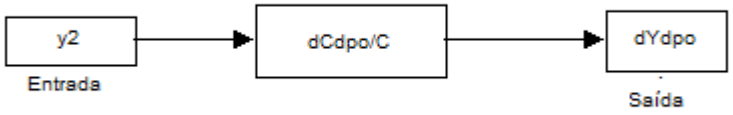
Subtraindo-se o sinal de entrada do sistema da Figura 2 da sua saída, obtém-se o sinal $e_1(t,\rho) = r_1(t) - y_1(t,\rho)$ que é utilizado como entrada no segundo ensaio que é apresentado na Figura 3. Utiliza-se o mesmo sistema (com os mesmos parâmetros de controlador) utilizado no primeiro e o objetivo é a obtenção do sinal de saída $y_2(t,\rho)$ que será utilizado na filtragem para obtenção de $\overline{\frac{\partial y(t,\rho)}{\partial \rho}}$.

Figura 3. Segundo ensaio do método IFT.



As filtragens finais utilizam o sinal $y_2(t,\rho)$ gerado no segundo ensaio como entrada, a fim de, ao final, gerar uma estimação para $\frac{\partial y_1(t,\rho)}{\partial \rho}$. Este sinal é filtrado por $\frac{1}{C(z,\rho)}\frac{\partial C(z,\rho)}{\partial \rho}$. Dessa forma, para o caso do PID, haverá três filtragens (uma relacionada a cada parâmetro do controlador) resultadas em três saídas. A filtragem é apresentada na Figura 4.

Figura 4. Filtragem do sinal $\boldsymbol{y_2(t)}$.



A partir das variáveis geradas nas simulações, é possível obter-se a estimação para o gradiente $\nabla J(\rho)$ e então é realizada a atualização do valor dos parâmetros do controlador através das equações apresentadas anteriormente.

3 SIMULAÇÕES COMPUTACIONAIS

Neste capítulo são descritos os aspectos ligados à implementação do algoritmo IFT no software $Matlab^{®}$. O objetivo é explanar detalhadamente a realização do método e apresentar os resultados obtidos, a fim de verificar e mostrar o funcionamento do mesmo. O desenvolvimento computacional servirá como base para a implementação prática, portanto, é necessário que o algoritmo implementado seja eficiente para a aplicação desejada.

3.1 IMPLEMENTAÇÃO DO ALGORITMO

Para realização da simulação é necessário, primeiramente, escolher uma função $T_d(z)$ que siga as especificações impostas pelo usuário. Da mesma maneira, deve-se, arbitrariamente, ou a partir de um projeto inicial, estabelecer os parâmetros iniciais ρ^1 do controlador que serão utilizados na primeira iteração (também chamado neste documento de "condições iniciais"). Para a simulação computacional, também se faz necessária a inclusão da função transferência do processo (que no caso da implementação prática é desconhecida).

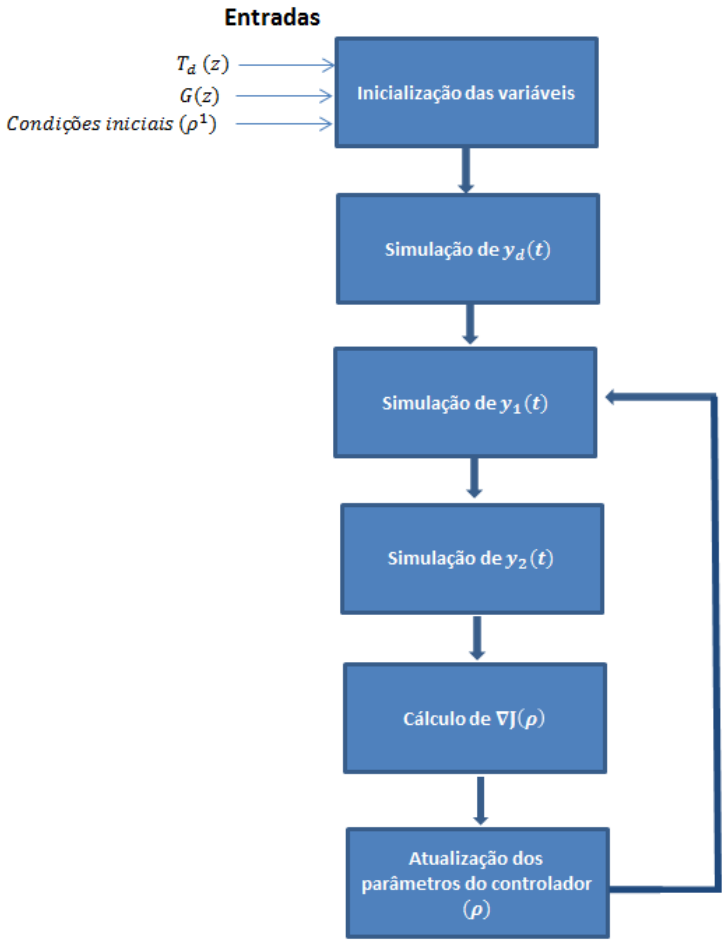
O primeiro comando a ser realizado é simular o sistema $T_d(z)$ a partir de uma entrada arbitrária $r_1(t)$. Assim, armazena-se a resposta obtida $y_d(t)$. Em seguida, realiza-se a simulação do sistema em laço fechado com o controlador $C(z, \rho^1)$ e o processo $G(z)$ utilizando a mesma entrada $r_1(t)$. Desta simulação, armazena-se o sinal de saída $y_1(t, \rho)$ e o sinal de entrada $r_1(t)$, gerando-se $e_1(t, \rho) = r_1(t) - y_1(t, \rho)$.

De posse dos dados obtidos nas duas primeiras simulações, é possível calcular a função custo $J(\rho^1)$ como o erro médio quadrático entre $y_1(t, \rho)$ e $y_d(t)$. Também se tem o sinal de entrada da terceira $r_2(t, \rho) = e_1(t, \rho)$. Portanto, é possível realizar a terceira simulação. Utiliza-se o mesmo sistema utilizado na segunda, entretanto, com entrada $r_2(t, \rho)$, obtendo-se, portanto $y_2(t, \rho)$.

Para finalizar a primeira iteração, é necessária ainda a filtragem do sinal $y_2(t,\rho)$ pelos filtros correspondentes às derivadas do controlador em relação a cada um dos parâmetros ρ. Desta forma, obtém-se três sinais distintos dados por $\frac{\partial y(t,\rho^1)}{\partial \rho_1}$, $\frac{\partial y(t,\rho^1)}{\partial \rho_2}$ e $\frac{\partial y(t,\rho^1)}{\partial \rho_3}$. De posse desses sinais (computacionalmente tratados como vetores), resta apenas aplicar (10), ou seja, multiplicar cada um desses vetores por $[y_1(t,\rho) - y_d(t)]^T$ e calcular a média aritmética de cada vetor resultante multiplicada por um fator 2. Assim, são obtidos os termos $\overline{\nabla J(\rho_1^1)}, \overline{\nabla J(\rho_2^1)}$ e $\overline{\nabla J(\rho_3^1)}$, restando apenas o cálculo dos novos parâmetros para finalizar a primeira iteração, conforme (9).

Finalizada a atualização dos parâmetros, pode ser realizada a segunda iteração, já com o novo conjunto de parâmetros ρ^2. De maneira análoga, podem-se realizar quantas iterações se queira. O fator γ^i será discutido no subcapítulo 3.2 tal como o processo de escolha das condições iniciais. O algoritmo descrito neste subcapítulo é ilustrado através do diagrama de fluxo da Figura 5 para melhor compreensão.

Figura 5. Diagrama de fluxo ilustrando o algoritmo utilizado.



3.2 TESTES E RESULTADOS DE SIMULAÇÃO

O objetivo neste subcapítulo é apresentar os resultados de simulação obtidos com o método IFT. Para essa parte do trabalho, utiliza-se o código *ift.m* (em anexo), que é a implementação do algoritmo apresentado no subcapítulo 3.1. Para melhor compreensão de cada passo, será estudado um exemplo prático.

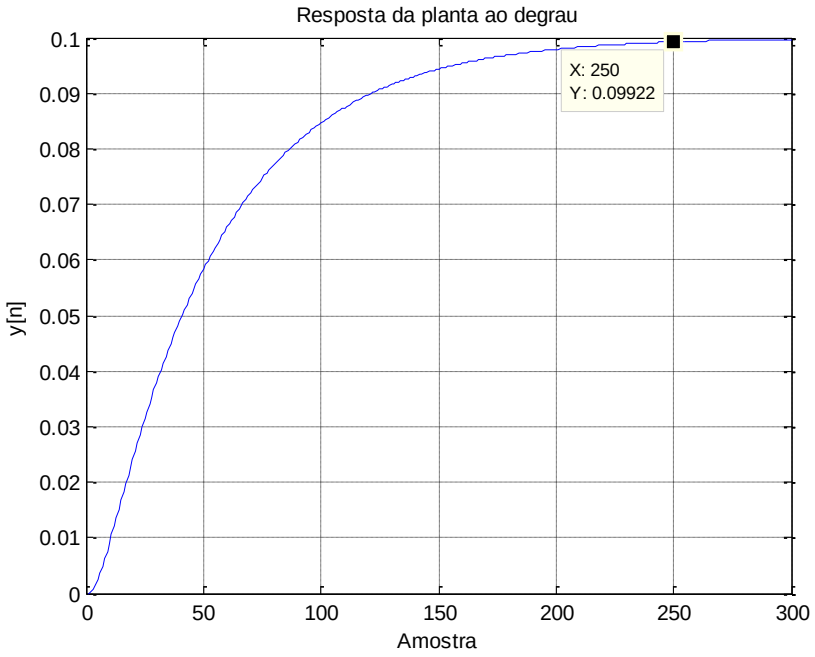
Considera-se o processo $G(z)$ dado por (19). Deseja-se obter os parâmetros do controlador PID para que o sistema em laço fechado obtenha erro nulo ao seguimento à referência e um tempo de acomodação de 50 amostras (considerando 1% de erro).

$$G(z) = 1{,}8604 \times 10^{-4} \frac{z + 0{,}9293}{(z - 0{,}8187)(z - 0{,}9802)} \tag{19}$$

Simulando-se (19) numericamente em laço aberto, considerando, como entrada, um degrau unitário, é obtido o gráfico mostrado na Figura 6. Desta forma, observa-se que o tempo de acomodação (t_s) é de aproximadamente 250 amostras.

Para o cálculo do controlador através do método IFT, primeiramente define-se uma $T_d(z)$ com o mesmo grau relativo (número de polos subtraído do numero de zeros) do processo $G(z)$ que satisfaça as especificações. Primeiramente, o polo dominante P_o da função em laço fechado deve respeitar a condição de tempo de acomodação de 50 amostras. O cálculo do polo dominante é dado por (20).

Figura 6. Resposta ao degrau unitário do sistema $\boldsymbol{G(z)}$ em laço aberto.



$$t_s = -\frac{4{,}6}{\ln(P_o)} \tag{20}$$

resultando em (21).

$$P_o = \mathrm{e}^{-\frac{4{,}6}{50}} = 0{,}9121 \tag{21}$$

Para que o sistema desejado $T_d(z)$ tenha o mesmo número de polos e zeros do sistema $G(z)$, devem ser adicionados um outro polo e um zero. Define-se arbitrariamente o polo $P_1 = 0{,}8$ e o zero $Z_1 = 0{,}85$. Dessa forma, a função $T_d(z)$ assumirá a forma (22).

$$T_d(z) = k\frac{z - 0{,}85}{(z - 0{,}8)(z - 0{,}9121)} \tag{22}$$

Para que o sistema tenha erro nulo ao seguimento à referência, deverá ser respeitado o critério estabelecido por (23).

$$\lim_{n\to\infty} y[n] = \lim_{n\to\infty} r[n] \tag{23}$$

Considerando-se a entrada como um salto unitário, tem-se (24).

$$\lim_{n\to\infty} y[n] = 1 \tag{24}$$

A partir do teorema do valor final, obtém-se o valor de ganho necessário para que a função transferência respeite o critério estabelecido em (23), gerando (25).

$$1 = \lim_{n\to\infty} y[n] = (z - 1)\lim_{z\to 1} Y(z) \tag{25}$$

$$= (z-1)\lim_{z\to 1} R(z)T_d(z) \tag{26}$$

$$= (z-1)\lim_{z\to 1} \frac{z}{(z-1)}\, k\, \frac{z-0{,}85}{(z-0{,}8)(z-0{,}9121)} \tag{27}$$

$$= \lim_{z\to 1} k\, \frac{z(z-0{,}85)}{(z-0{,}8)(z-0{,}9121)} \tag{28}$$

$$= k\, \frac{0{,}15}{(0{,}2)(\,0{,}0879)} \tag{29}$$

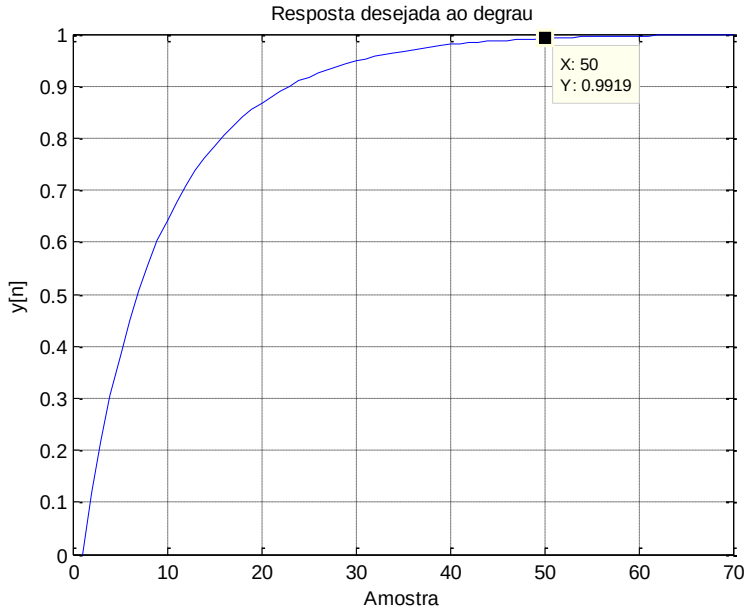
Para que (29) seja igual a 1, o ganho k deverá ser de $0{,}1172$. Logo, obtém-se a função transferência desejada, dada por (30).

$$T_d(z) = 0{,}1172\, \frac{z-0{,}85}{(z-0{,}8)(z-0{,}9121)} \tag{30}$$

Realizando-se o ensaio, considerando um salto unitário como entrada, obtém-se a resposta ao salto da função de transferência desejada $T_d(z)$. Sua curva é mostrada no gráfico da Figura 7.

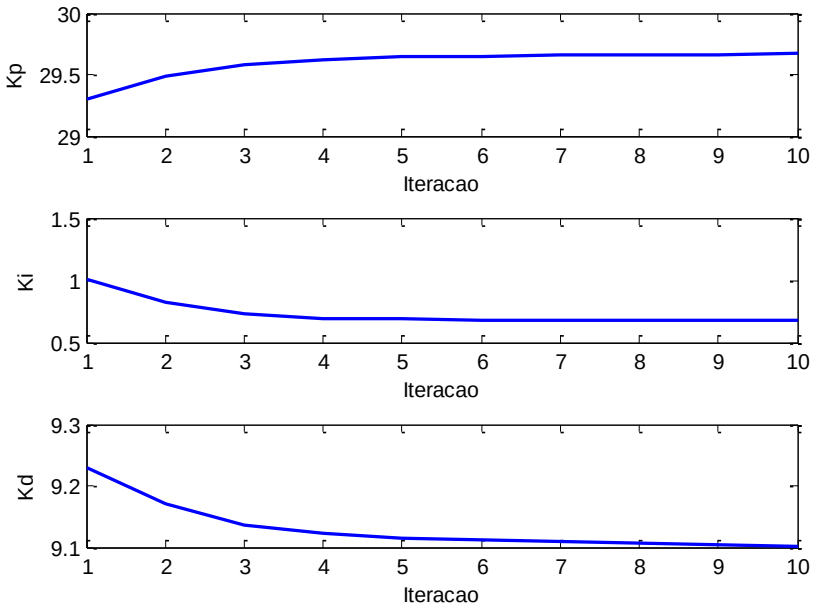
Observa-se na Figura 7 que o sistema desejado, de fato, apresenta erro nulo ao seguimento à referência e o tempo de acomodação é de aproximadamente 50 amostras, como desejado. Logo, o método IFT deverá aproximar a resposta do sistema em laço fechado o máximo possível à resposta apresentada na Figura 7.

Figura 7. Resposta ao degrau unitário da função $\boldsymbol{T_d(z)}$.



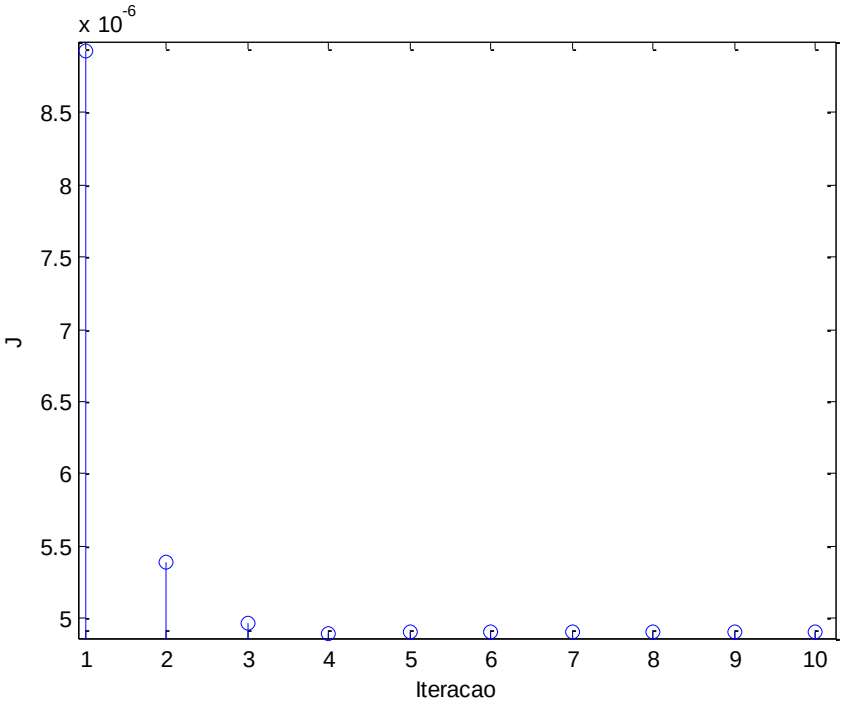
Para realizar o método IFT, basta então escolher arbitrariamente as condições iniciais (ρ). Na Figura 8 é mostrada a evolução dos parâmetros considerando 10 iterações.

Figura 8. Evolução dos parâmetros $\boldsymbol{\rho}$.



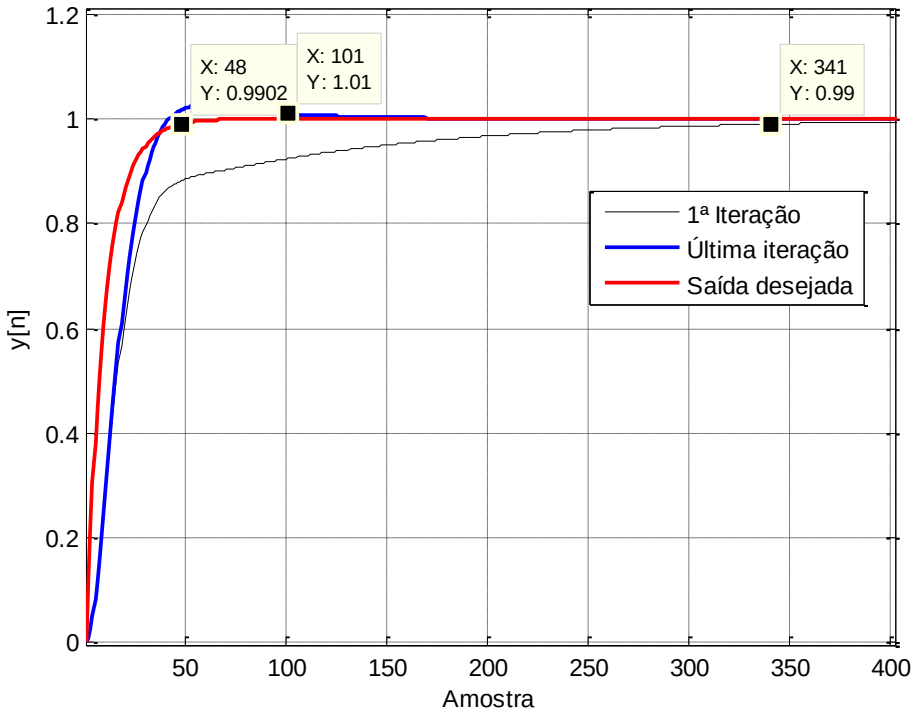
Observa-se na Figura 8 que os parâmetros convergiram cada um para um valor. Na Figura 9, observa-se a evolução da função custo em função da iteração.

Figura 9. Evolução da função custo conforme a iteração.



Pode-se observar nitidamente na Figura 9 que o método, de fato, minimiza a função custo. Na primeira iteração o valor de $J(\rho)$ é de $8{,}93 \times 10^{-6}$ e na décima esse valor decai para $4{,}9 \times 10^{-6}$. Para ilustrar o efeito que isso causa na prática, é apresentada a Figura 10, mostrando a evolução da resposta ao salto unitário quando se minimiza a função custo $J(\rho)$. Na Figura 10 são apresentadas em vermelho a curva de saída desejada, em preto a curva obtida com parâmetros aleatórios de controlador (condições iniciais) e a curva em azul representa a saída do sistema na última iteração.

Figura 10. Resposta ao salto do sistema desejado, do sistema com parâmetros arbitrários e com a função custo minimizada.



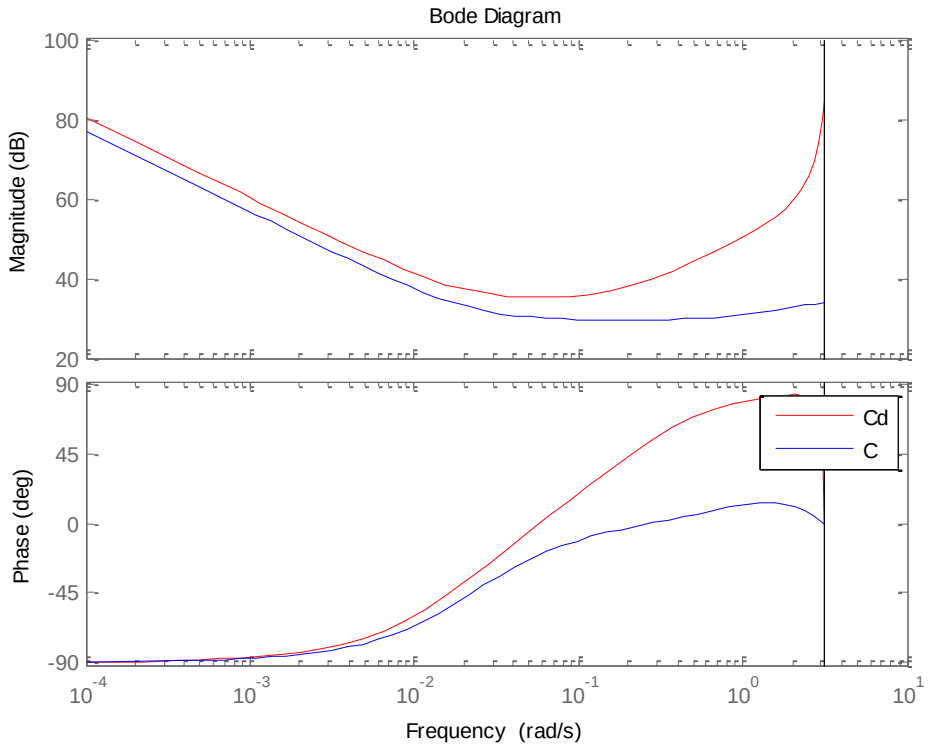
Na Figura 10 pode-se verificar, de fato, a eficácia do método, reduzindo o tempo de acomodação de 341 amostras para 100 amostras. É verdade que ainda não cumpre a especificação de realizar a acomodação em 50 amostras. Isso se deve principalmente a dois fatores. O primeiro é que a função transferência desejada pode ser inatingível com um simples controlador PID. Em (31) é apresentado o cálculo do controlador ideal que faria com que o sistema apresentasse a saída exatamente igual à desejada.

$$C_d(z) = \frac{T_d(z)}{G(z)(1 - T_d(z))} \tag{31}$$

$$C_d(z) = 629{,}972\,\frac{(z - 0{,}85)(z - 0{,}8187)(z - 0{,}9802)}{(z - 0{,}8293)\,(z - 1)\,(z + 0{,}9293)} \tag{32}$$

Observa-se em (32) que, de fato, o controlador desejado possui 3 polos e 3 zeros, enquanto o PID possui 2 polos e 2 zeros, ou seja, é impossível obter um sistema exatamente igual ao desejado com um PID. O método, por sua vez, busca um controlador com a resposta mais próxima possível do controlador desejado. Na Figura 11 é apresentado o diagrama de Bode do controlador ideal e do controlador obtido com o método IFT.

Figura 11. Comparação entre os diagramas de Bode dos controladores desejado e obtido.



Verifica-se na Figura 11 que, de fato, para altas frequências, o controlador obtido com o método IFT não representa tão bem o controle desejado. Por outro lado, para baixas frequências, o comportamento é semelhante.

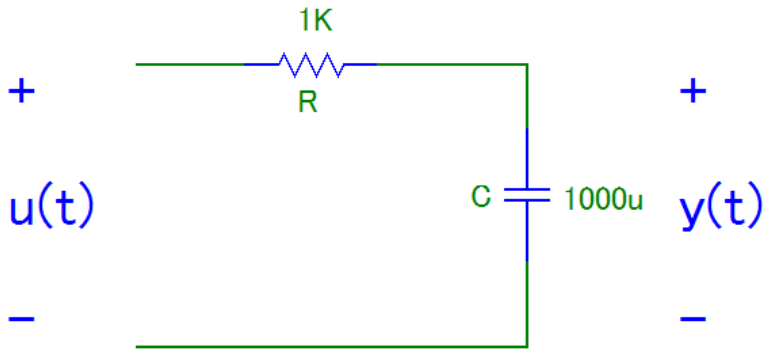
Outro problema que pode ser observado no cálculo de novos parâmetros é devido ao passo da iteração γ. Se se assume um passo muito pequeno, em uma nova iteração, os parâmetros pouco vão variar, entretanto, assumindo-se um passo muito grande, o parâmetro

pode nunca convergir para um mínimo da função $J(\rho)$. A literatura apresenta uma discussão mais detalhada acerca do tema (BAZANELLA; CAMPESTRINI; ECKHARD, 2012). Na simulação realizada, assume-se um fator γ^i inversamente proporcional ao índice da iteração. Ou seja, na primeira iteração, o fator γ^1 será elevado em relação a γ^n justamente para que ao final, o resultado convirja.

No caso de se obter qualquer tipo de informação, o método pode mostrar-se ainda mais eficaz na obtenção de parâmetros para aperfeiçoar a resposta do sistema. Deseja-se mostrar isso no próximo exemplo estudado. Este terá a finalidade de simular uma alteração física no processo seja por desgastes mecânicos, variação de parâmetros físicos do meio onde o processo se encontra ou outro fator que cause variação no seu comportamento. Desta forma, sua função transferência também sofrerá alteração. Supondo que esse processo estivesse submetido a um sistema de controle previamente projetado, esta variação na função transferência exigiria um novo projeto de controlador para mantê-lo controlado da maneira desejada. O método IFT será utilizado para reajustar os parâmetros do controlador de modo a minimizar o efeito dessa variação no processo, na saída do sistema.

A ideia é utilizar um circuito RC-série como processo (Figura 12) e projetar um controlador para ele seguindo algumas especificações. Após, varia-se o valor do resistor do circuito e aplica-se o método IFT, utilizando, como condições iniciais, os parâmetros do controlador anteriormente projetado e verifica-se se o controlador resultante do método apresenta uma resposta em laço fechado similar à obtida com o projeto do controlador inicial para o processo original.

Figura 12. Circuito RC-série.



O circuito estudado tem a função de transferência dada por (33)

$$G(s) = \frac{\left(\frac{1}{RC}\right)}{s + \left(\frac{1}{RC}\right)} \tag{33}$$

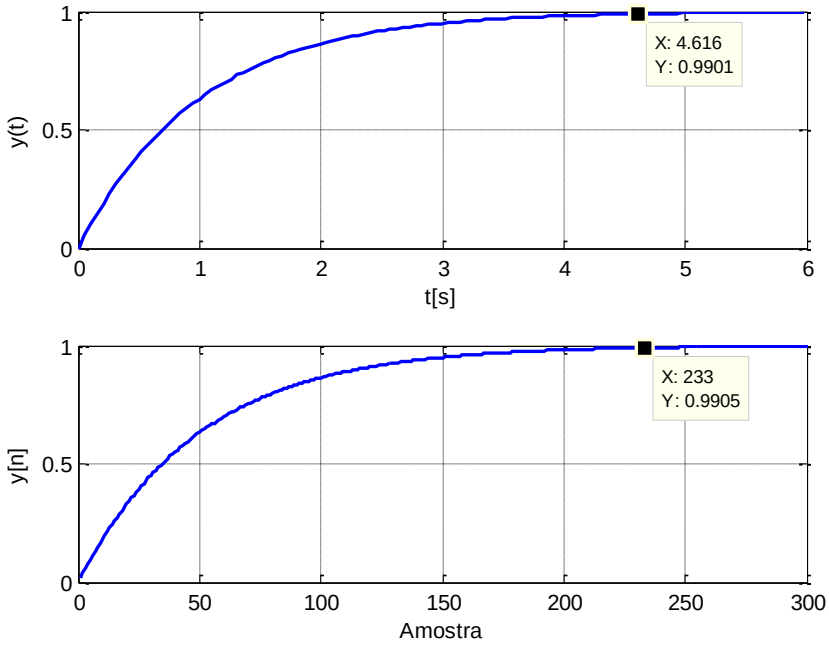
cujo polo é dado por $\frac{1}{RC}$. Desta forma, com os valores $R = 1K\Omega$ e $C = 1mH$, obtém-se a função transferência dada por (34),

$$G(s) = \frac{1}{s + 1} \tag{34}$$

que, com um período de amostragem $T_s = 0{,}02s$, gera a função de transferência apresentada em (35). Na Figura 13 é apresentada a resposta de $G(s)$ e $G(z)$ ao degrau unitário.

$$G(z) = \frac{0{,}0198z}{z - 0{,}9802} = \frac{0{,}0198}{1 - 0{,}9802\, z^{-1}} \tag{35}$$

Figura 13. Resposta do sistema ao degrau em laço aberto (contínuo e amostrado).



O interesse nessa planta é o projeto de um controlador para reduzir o tempo de carga do capacitor (que é de aproximadamente 4,6s) e obter erro nulo ao seguimento de referência. O polo (0,9802) e o zero (0) da função $G(z)$ e os polos fixos da função $C(z)$ ($0\ e\ 1$) são apresentados no diagrama da Figura 14 para que se realize a escolha dos dois zeros restantes da função $C(z)$ (que não estão apresentados na figura).

A Figura 14 é apresentada de maneira a prever o comportamento do método do lugar das raízes conforme os dois zeros restantes da função $C(z)$ são alocados. Observa-se que, ao alocar um zero em $0{,}98$ e o outro no intervalo $(0, 0{,}9802)$, obtém-se um diagrama do lugar das raízes tal qual o apresentado na Figura 15.

Figura 14. Polos e zeros da função $\boldsymbol{G(z)}$ e polos da função $\boldsymbol{C(z)}$.

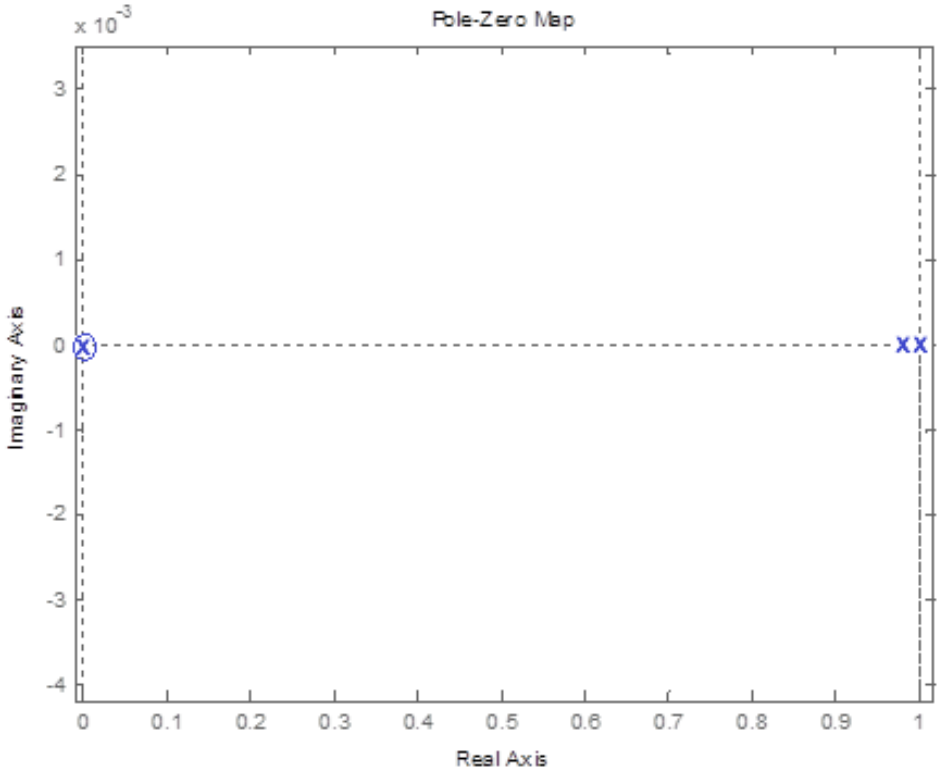
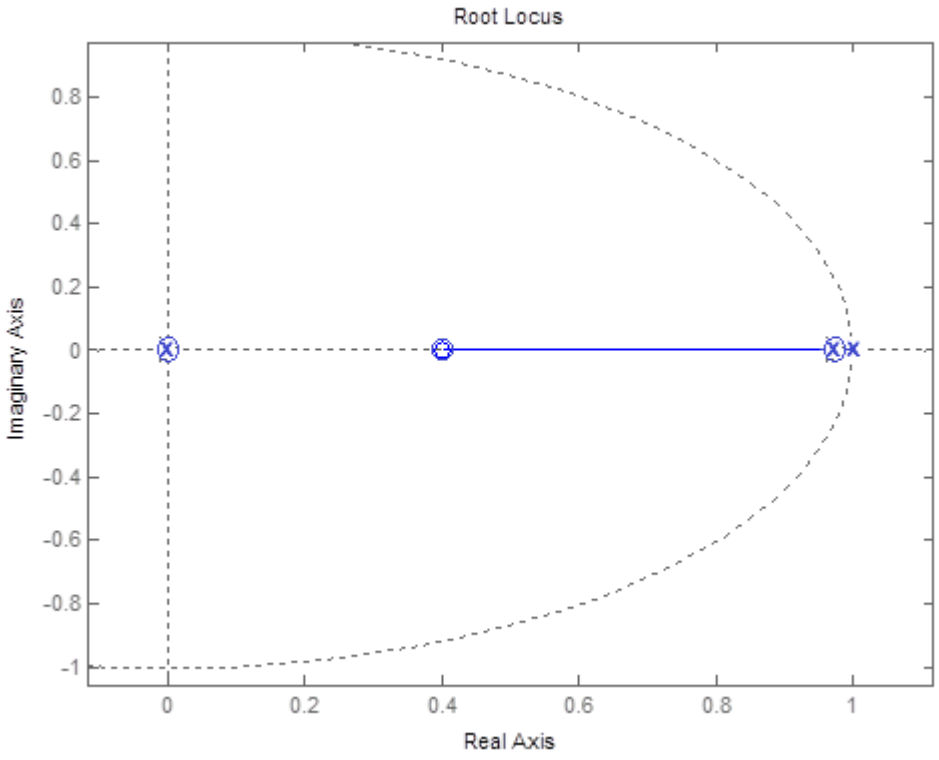


Figura 15. Diagrama lugar das raízes para o sistema com o controlador PID.

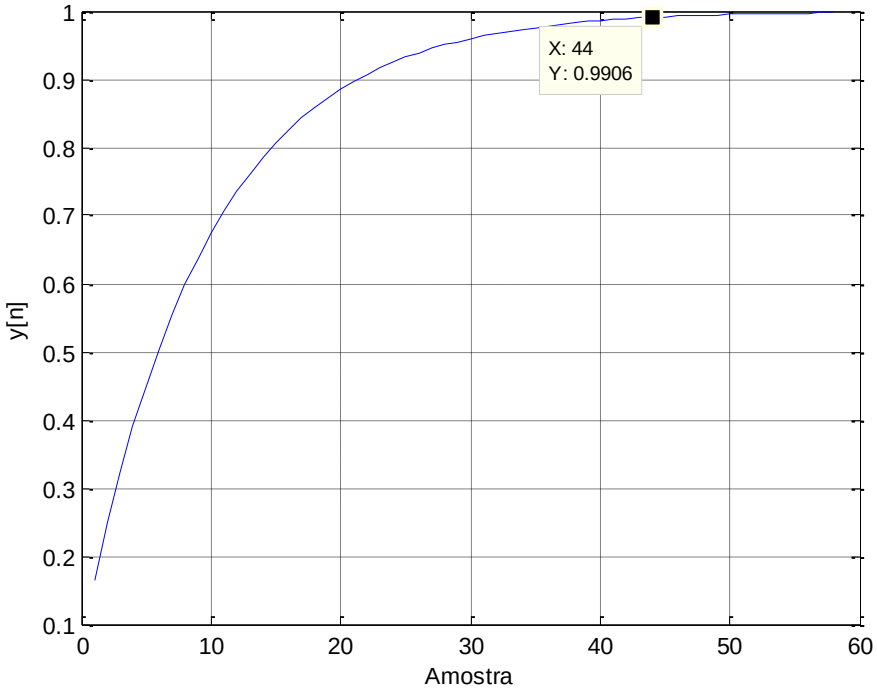


Na Figura 15 é possível inferir que quando se aumenta o ganho K, menor é o módulo do polo dominante e, portanto, mais rápido é o seguimento à referência. Dessa forma, define-se, arbitrariamente o segundo zero em $0{,}4$. Escolhendo-se o ganho $K = 10$, o tempo de acomodação do sistema será de aproximadamente 50 amostra ($1s$). Desta forma, o controlador terá a função transferência apresentada em (36).

$$C(z,\rho) = 10\frac{(z-0{,}9802)(z-0{,}4)}{z^2 - z} \tag{36}$$

O controlador em (36) possui parâmetros $k_p = 5{,}958$, $k_i = 0{,}121$ e $k_d = 3{,}921$. Simula-se então o processo com esse controlador em laço fechado. O resultado da resposta ao degrau unitário é apresentado na Figura 16.

Figura 16. Resposta do sistema controlado ao degrau unitário.



Observa-se na Figura 16 que o sistema controlado se comporta da maneira esperada. Dessa forma, tem-se um conjunto de parâmetros que controla um determinado processo (circuito RC-série) de maneira desejada. Assim sendo, pode-se definir a função transferência desejada $T_d(z)$ apresentada em (38), cujo controlador ideal é o dado por (36).

$$T_d(z,\rho) = \frac{C_d(z,\rho)G(z)}{1 + C_d(z,\rho)G(z)} \tag{37}$$

$$T_d(z,\rho) = 0{,}165\frac{(z-0{,}4)}{(z-0{,}89)(z+0{,}0889)} \tag{38}$$

A ideia é supor que o resistor do processo original, por alguma razão tenha seu funcionamento modificado e sua resistência passe a valer $1{,}2K\Omega$ sem que o usuário do sistema perceba. Assim, o método deverá adaptar um novo controlador que mantenha o funcionamento do sistema o mais próximo possível do projetado. Em (39) é apresentada a nova função transferência do sistema e em (40), a função no domínio z.

$$G_1(s) = \frac{0{,}833}{s + 0{,}833} \tag{39}$$

$$G_1(z) = \frac{0{,}01652}{1 - 0{,}9835z^{-1}} \tag{40}$$

Considerando-se tal sistema controlado com o controlador projetado em (36), obtém-se a resposta ao degrau unitário, apresentada na Figura 17.

Observa-se na Figura 17 que o sistema agora apresenta um tempo de acomodação de 83 amostras, ou seja, o controlador não cumpre mais a especificação imposta inicialmente. Neste caso, realiza-se o método IFT a fim de reestabelecer as condições desejadas. As condições iniciais utilizadas são os próprios parâmetros do controlador projetado

anteriormente. A Figura 18 mostra a evolução das respostas ao degrau unitário conforme evoluem as iterações e a Figura 19 apresenta os mesmos gráficos, entretanto com um zoom na área crítica, mostrando a diferença entre as curvas.

Figura 17. Reposta do sistema $\boldsymbol{G_1(z)}$ controlado ao degrau unitário.

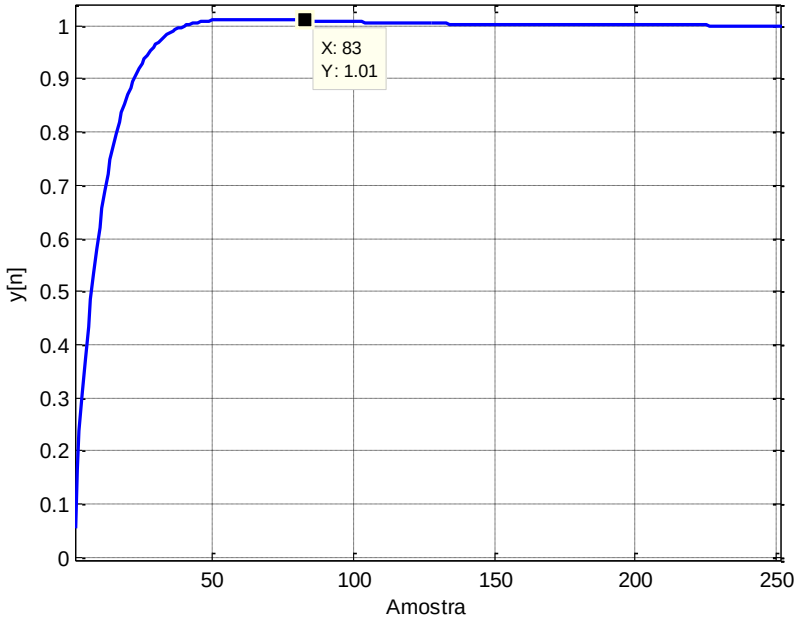


Figura 18. Resposta do sistema ao degrau unitário.

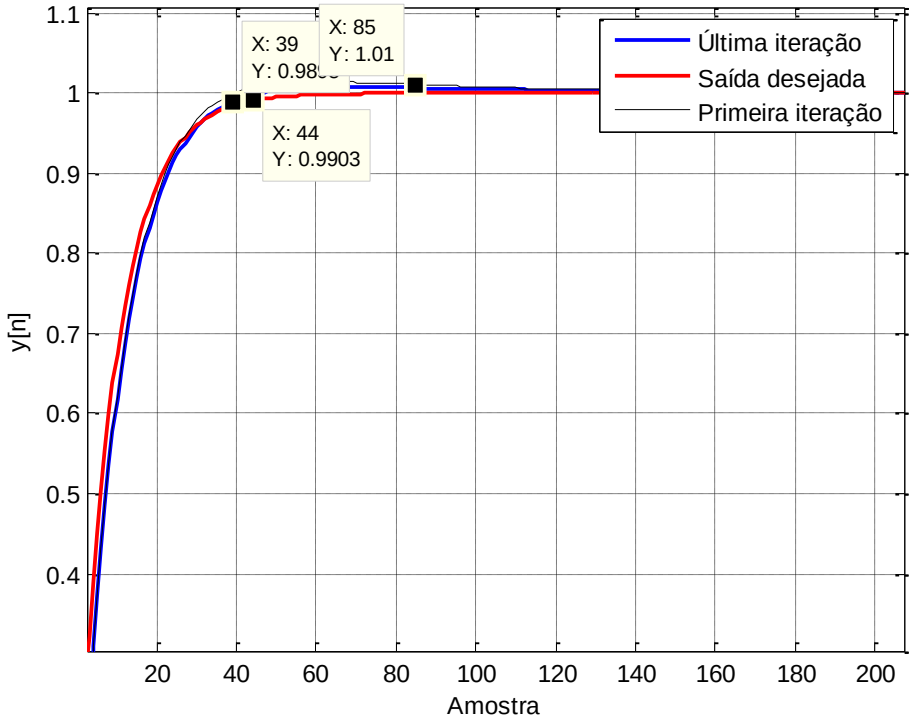
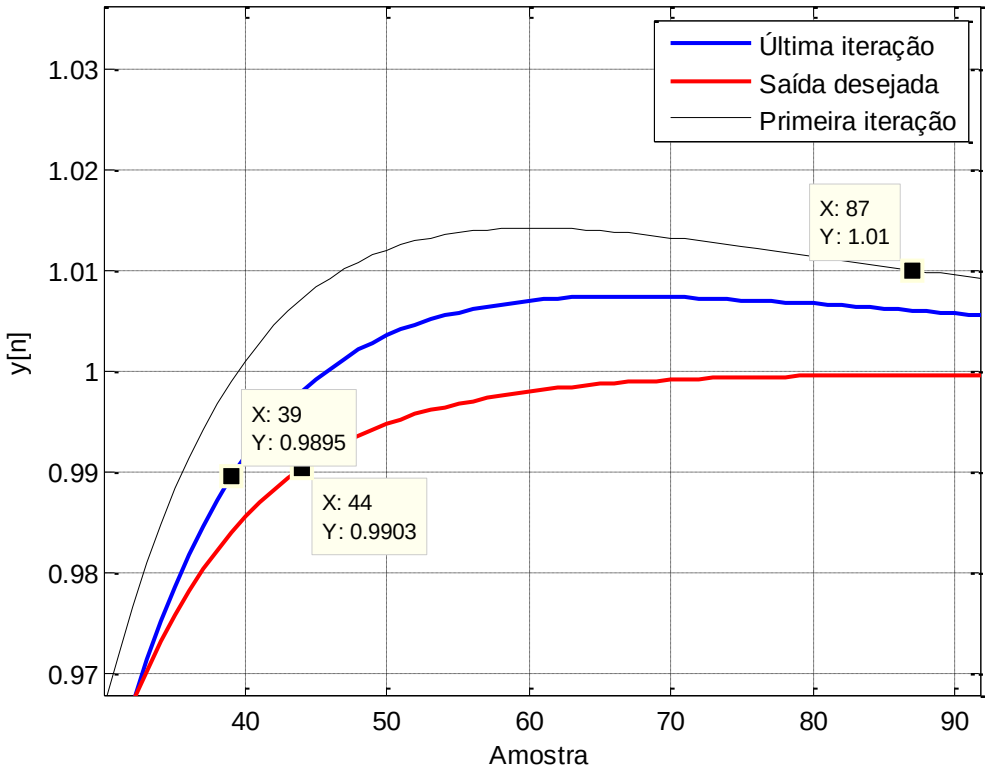


Figura 19. Resposta detalhada do sistema ao degrau.



Na Figura 19 fica claro que o método IFT reestabelece as condições desejadas para o sistema. A função custo evolui de $J(\rho^1) = 2{,}73 \times 10^{-5}$ para $J(\rho^{10}) = 2{,}585 \times 10^{-5}$. A Tabela 1 mostra a evolução dos parâmetros do controlador.

Tabela 1. Evolução dos parâmetros do controlador.

Parâmetro	**1ª Iteração**	**10ª Iteração**
k_p	5,958	5,9787
k_i	0,121	0,1117
k_d	3,921	3,9123

Observa-se na Tabela 1 que os parâmetros não apresentaram variação significativa, entretanto, foi a variação necessária para o sistema se comportar o mais próximo possível do desejado. Com este exemplo, é possível verificar que com uma boa escolha de condições iniciais, o método pode apresentar resultados muito satisfatórios.

4 IMPLEMENTAÇÃO EMBARCADA

O controlador utilizado para esta aplicação é um kit de desenvolvimento Arduino Due. A escolha pelo kit Arduino é realizada pela simplicidade da programação e a alta disseminação, resultando em um grande número de informações e fóruns de discussão online. A escolha pelo modelo Due é realizada porque este possui um processador Atmel SAM3X8E ARM Cortex-M3, de 32-bits com um *clock* de 84MHz, 512KB de memória flash e saídas e entradas analógicas. Escolhe-se esse kit (cerca de 8 vezes mais rápido que os demais Arduinos) para minimizar eventuais problemas de processamento. Em trabalhos futuros pode ser estudada a otimização do controlador em relação ao custo-benefício.

4.1 TOPOLOGIA DO SISTEMA

4.1.1 PROCESSAMENTO DIGITAL

A função transferência do controlador PID estudada anteriormente pode ser dada por (42).

$$C(z,\rho) = [\rho_1 \quad \rho_2 \quad \rho_3] \begin{bmatrix} \frac{z^2}{z^2 - z} \\ \frac{z}{z^2 - z} \\ \frac{1}{z^2 - z} \end{bmatrix} \tag{41}$$

$$= \frac{\rho_1 z^2 + \rho_2 z + \rho_3}{z^2 - z} \tag{42}$$

Multiplicando-se o numerador e o denominador da função $C(z,\rho)$ por z^{-2}, obtém-se (43).

$$C(z,\rho) = \frac{\rho_1 + \rho_2 z^{-1} + \rho_3 z^{-2}}{1 - z^{-1}} \tag{43}$$

Supondo-se que tal sistema é excitado por uma entrada e(t) (cuja transformada z é dada por $E(z)$) gerando a saída $u(t)$ (cuja transformada z é dada por $U(z)$), pode-se obter (44) que gera (45).

$$U(z) = E(z)\frac{\rho_1 + \rho_2 z^{-1} + \rho_3 z^{-2}}{1 - z^{-1}} \tag{44}$$

$$\therefore$$

$$[1 - z^{-1}]U(z) = E(z)[\rho_1 + \rho_2 z^{-1} + \rho_3 z^{-2}] \tag{45}$$

Realizando-se a transformada z inversa, obtém-se (46), resultando em (47).

$$u[n] - u[n-1] = \rho_1 e[n] + \rho_2 e[n-1] + \rho_3 e[n-2] \tag{46}$$

$$\therefore$$

$$u[n] = u[n-1] + \rho_1 e[n] + \rho_2 e[n-1] + \rho_3 e[n-2] \tag{47}$$

Dessa forma, pode-se aplicar o processamento desejado. No caso do controlador PID, é necessário armazenar em um *buffer* $u[n-1], e[n-1]$ *e* $e[n-2]$ além do termo atual $e[n]$ para obter-se o termo $u[n]$.

4.1.2 IMPLEMENTAÇÃO PRÁTICA

Dado o sistema $T_d(z)$ projetado na seção 4.2, deseja-se implementá-lo no Arduino para simulá-lo e comparar com o resultado obtido no $Matlab^{®}$ anteriormente. A adequação para um modelo da mesma forma que (47) é apresentada entre (48) e **(53)**.

$$T_d(z) = 0{,}1172\,\frac{z - 0{,}85}{(z - 0{,}8)(z - 0{,}9121)} \tag{48}$$

$$= \frac{0{,}1172\,\mathrm{z} - \ 0{,}09962}{z^2 - 1{,}712z + 0{,}7297} \tag{49}$$

$$= \frac{0{,}1172\ - \ 0{,}09962\,\mathrm{z}^{-1}}{1 - 1{,}712\,z^{-1} + 0{,}7297\,z^{-2}} \tag{50}$$

Considerando que esse sistema excitado por uma entrada $X(z)$ gera uma saída $Y_d(z)$:

$$Y_d(z) = X(z)\,\frac{0{,}1172\ - \ 0{,}09962\,\mathrm{z}^{-1}}{1 - 1{,}712\,z^{-1} + 0{,}7297\,z^{-2}} \tag{51}$$

$$Y_d(z)[1 - 1{,}712\,z^{-1} + 0{,}7297\,z^{-2}] = X(z)[0{,}1172\ - \ 0{,}09962\,\mathrm{z}^{-1}] \tag{52}$$

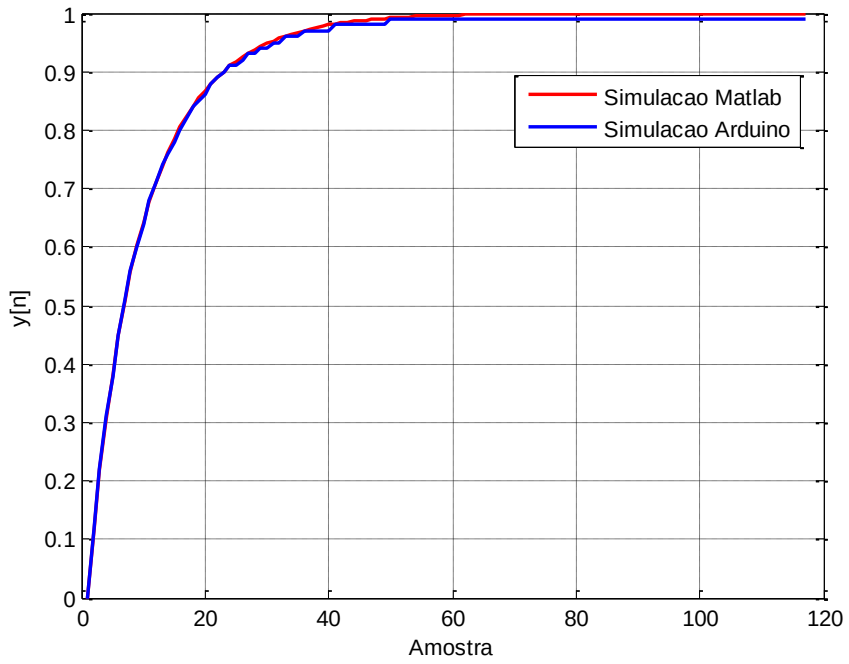
$$\therefore$$

$$y[n] = 1{,}712y[n-1] - 0{,}7297y[n-2] + 0{,}1172x[n] - 0{,}09962x[n-1] \tag{53}$$

Utilizando-se como entrada um salto unitário, pode-se implementar essa equação de diferenças no Arduino. O código gerado para essas simulação é apresentado em anexo *(filtro.ino)* e o resultado obtido, é mostrado no gráfico azul da Figura 20. O sistema simulado através do $Matlab^{®}$ é apresentado em vermelho.

Observa-se que, graças a arredondamentos internos do processador, a função em azul não representa exatamente o comportamento esperado (o da curva em vermelho). O erro médio quadrático obtido na aproximação é de aproximadamente $6{,}65 \times 10^{-5}$ com relação à simulação gerada computacionalmente.

Figura 20. Resposta ao salto do sistema $\boldsymbol{T_d(z)}$



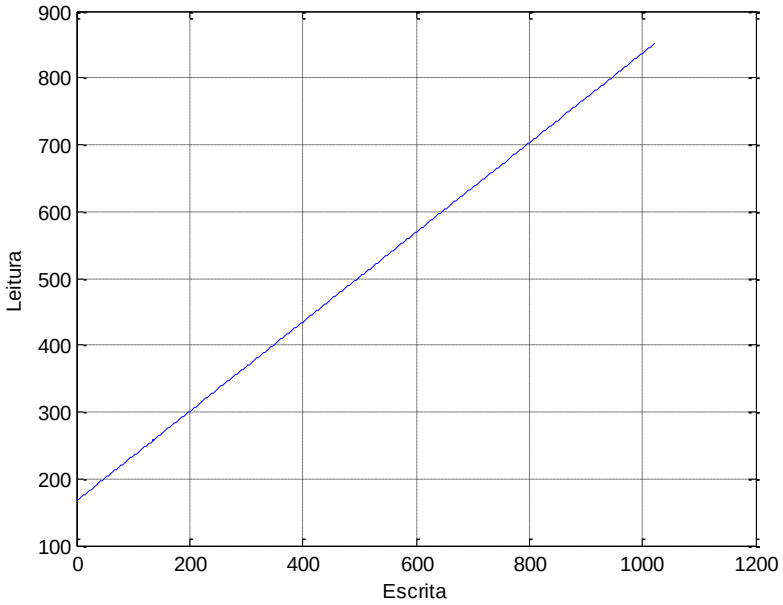
4.1.3 PECULIARIDADES TÉCNICAS

O Arduino Due apresenta entradas e saídas analógicas que podem ser utilizadas. A tensão de alimentação é de 3,3V e as tensões mínimas e máximas de saída (DAC) são de $\frac{1}{6}$ e $\frac{5}{6}$ da tensão de alimentação respectivamente, resultando em 0,55V e 2,75V. Entretanto, para as entradas analógicas, o intervalo de leitura é de 0V a 3,3V. Portanto, deve ser realizada no *firmware* uma rotina para uniformizar os níveis de tensão gerados e lidos (Atmel Corporation).

No ambiente de programação, utiliza-se o valor codificado da tensão que se deseja escrever ou ler, por exemplo, se o Arduino está configurado para operar com uma resolução de 10 bits, há 1024 níveis distintos, onde o nível 0 representa 0,55V e o nível 1023 representa 2,75V em escrita. Portanto, por exemplo, quando se escreve 1023, fisicamente é aplicada uma tensão de 2,75V na porta DAC, mas ao se ler essa mesma tensão, se obtém um nível de 853. Para avaliarem-se as entradas e saídas analógicas do Arduino, realiza-se uma simulação para

cada nível de escrita e lê-se o próprio valor no Arduino. Desta forma, o código *calibragem.ino* (em anexo) aplica o nível 0 na saída e o lê através da entrada analógica do Arduino, imprimindo o valor lido. Após, aplica-se o nível 1, e efetua-se a leitura novamente. Analogamente são feitas simulações até o último nível (1023). O gráfico da Figura 21 apresenta a comparação entre os valores lidos e escritos. Para realizá-lo, se conecta o pino de saída de tensão (DAC0) diretamente no pino de entrada de tensão (A0).

Figura 21. Calibração da interface A/D do Arduino.



Desta forma, obtém-se a equação (54) que relacionada a leitura "L" com a escrita "E" e em (55), com a variável "E" isolada

$$L = 0.6699E + 166.4254 \tag{54}$$

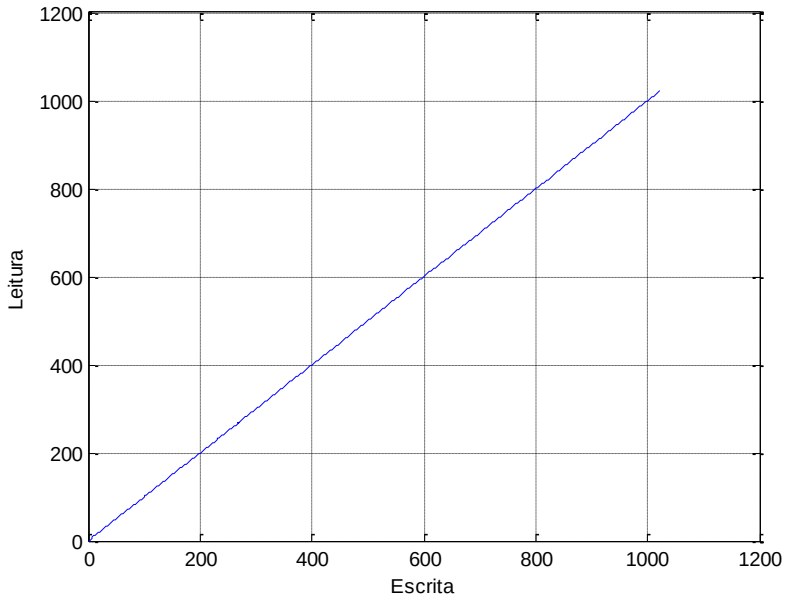
$$\therefore$$

$$E = 1.4927L - 248.4289 \tag{55}$$

onde “E” é o valor escrito e “L” é o valor lido. Para simplificar o desenvolvimento, a notação “leitura” nunca será utilizada no código. Sempre que uma leitura for realizada na forma “L”, seu valor deverá ser imediatamente convertido para a forma “E”.

Desta forma, aplica-se (55) no mesmo código para obter o valor “real” escrito obtido na leitura e realiza-se o mesmo procedimento realizado anteriormente, obtendo-se o gráfico obtido na Figura 22.

Figura 22. Leitura e escrita calibradas.



Observa-se na Figura 22 que a aproximação (55), de fato, resulta na leitura correta em relação à escrita.

Outra peculiaridade técnica do sistema de leitura e escrita é a ausência de valores de tensão negativas no Arduino. Desta forma, se, por vezes, o algoritmo de controle devesse gerar um sinal negativo, o Arduino não seria capaz de gerá-lo com relação a sua própria referência. Portanto, propõe-se a utilização de uma referência de tensão que seja exatamente a média entre a tensão máxima e mínima. Por exemplo, considerando 10 bits de resolução

(1024 níveis), a referência do circuito se dará no nível 512 (1,1V). Deste modo, se é aplicada uma tensão equivalente ao nível 500 (1,074V), como a referência é 512 (1,1V), a tensão aplicada em relação à referência do circuito será equivalente a -12 (-0,026V). Para tanto, é necessária a obtenção de mais uma interface. Essa deverá converter o valor na forma de escrita (entre 0 e 1023) para o valor a ser processado (entre -511 e 512). Assim, obtém-se a fórmula de conversão dada por (56) que é descrita com a variável "processamento" (P_r) isolada em (57).

$$E = P_r + 511 \tag{56}$$

$$\therefore$$

$$\boldsymbol{P_r = E - 511} \tag{57}$$

Desta forma, sempre após uma leitura deve-se converter os dados para a forma "E" e, se esses dados forem futuramente processados, deve-se realizar uma nova conversão para a forma "P_r". Se se deseja escrever algum dado processado numa saída analógica do Arduino, realiza-se a conversão de "P_r" para "E".

Como exemplo, o código *teste_interfaces.ino* (em anexo) recebe um valor na forma "P_r", realiza sua escrita, após, sua leitura e então, aplica um ganho aleatório. Após, escreve o valor produzido em alguma saída que é novamente lida, processada e então mostrada ao usuário, na forma P_r. O valor final mostrado na forma P_r deve ser igual ao valor inserido inicialmente no código multiplicado pelo ganho aleatório. A Tabela 2 apresenta os valores obtidos em cada passo.

Tabela 2. Evolução dos valores nas formas "P_r", "E" e "L".

Valor Inicial	Valor Escrito	Valor Lido	Valor recuperado	Ganho	Valor processado	Valor escrito	Valor lido	Valor resultante
200	711.00	709.88	198.88	-1.00	-198.88	312.12	312.83	-198.17

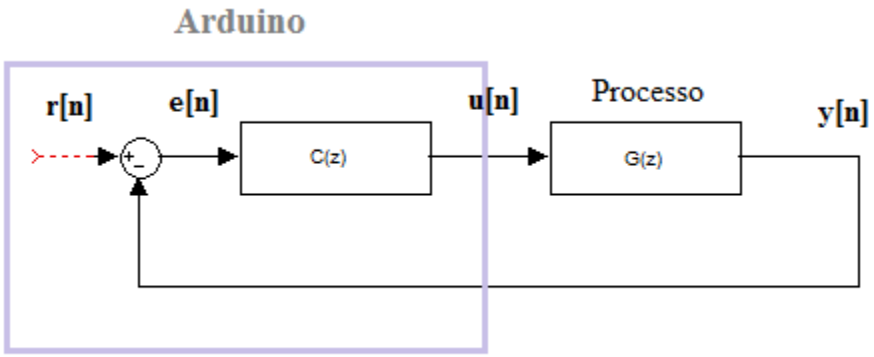
Observa-se na Tabela 2 que os valores cujo fundo é cinza estão na forma "P_r" e os valores cujo fundo é azulado estão na forma "E". Inicialmente apresenta-se o valor inicial definido arbitrariamente. Esse valor é convertido para a forma "E" e escrito no pino de saída analógica do Arduino. Este valor é então lido pelo pino de entrada, seu valor é novamente convertido para a forma "E" e então, para a forma "P_r", resultando na coluna "Valor recuperado". Observa-se que o valor recuperado não reproduz exatamente o valor inicial, resultando num erro obtido de aproximadamente 0,6%. Entretanto, após uma nova sequência de escritas e leituras, obtém-se um novo erro de 0,915% em relação ao valor inicial.

4.1.4 IMPLEMENTAÇÃO DO CONTROLADOR PID

O controlador PID estudado no subcapítulo 4.1.1 é implementado no Arduino da mesma forma ao qual o processo estudado no subcapítulo 4.1.2. A topologia adotada no projeto segue o modelo apresentado na Figura 23.

Observa-se que o Arduino irá gerar uma referência r[n] que será subtraída do valor de saída do processo y[n], resultando em e[n] que por sua vez será processado pelo controlador PID da forma apresentada em 4.1.1. A saída do controlador u[n] é a entrada do processo. Portanto, o Arduino deverá receber o valor y[n] do processo e enviar u[n].

Figura 23. Topologia do sistema de controle com Arduino.



A implementação desse sistema de controle no Arduino é semelhante a do *filtro.ino* (em anexo) estudada em 4.1. Deve-se, portanto processar o sinal $e[n] = r[n] - y[n]$ diferentemente do primeiro, onde se processava diretamente x[n].

Outras modificações são necessárias no sentido de adaptar o sistema para ser utilizado em tempo real. A primeira delas é para que a obtenção de dados se dê de maneira sistemática, a partir de uma frequência de amostragem f_s. A solução adequada para essa implementação é a partir de interrupções temporais. A cada período de tempo $T_s = \frac{1}{f_s}$, ocorre uma interrupção e o algoritmo realiza a obtenção de $y[n]$ e gera $u[n]$. Após, o sistema coloca-se em estado de espera, aguardando a próxima interrupção. Para que essa topologia seja funcional, o tempo de processamento deverá ser inferior ao período de amostragem para que a interrupção seja realizada sempre quando o sistema encontrar-se em estado de espera e nunca durante o processamento. Este controlador é implementado através do código *controlador_pid.ino* (em anexo).

Através de um ensaio em laço fechado, considerando a entrada como uma referência de amplitude 10, obtém-se a curva vermelha da Figura 24. Em azul é apresentada a curva simulada no Matlab. Na Figura 25 é apresentado o sinal de controle obtido (em vermelho) e o sinal de controle esperado (em azul).

Figura 24. Resposta do sistema ao degrau de amplitude 10**.**

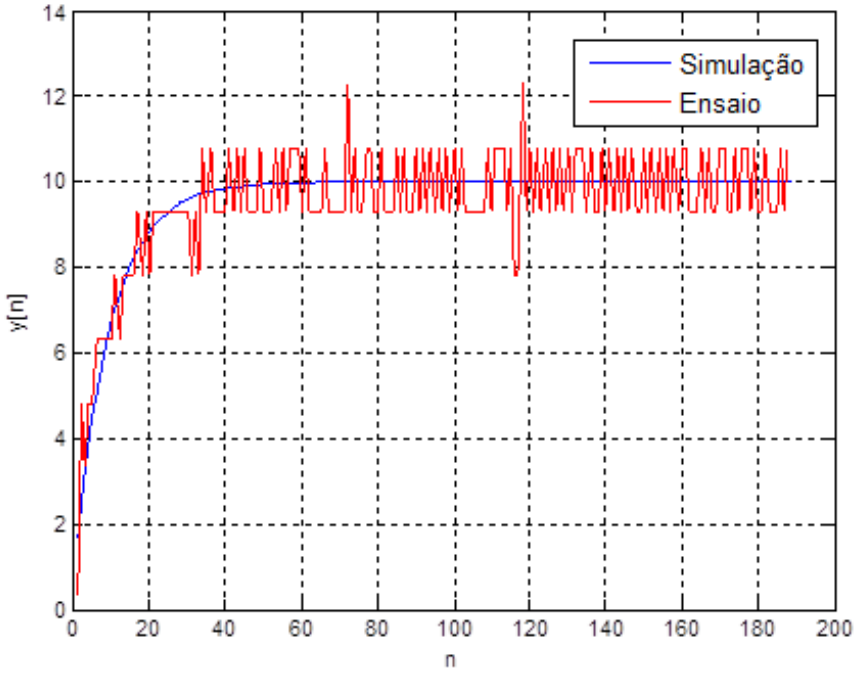
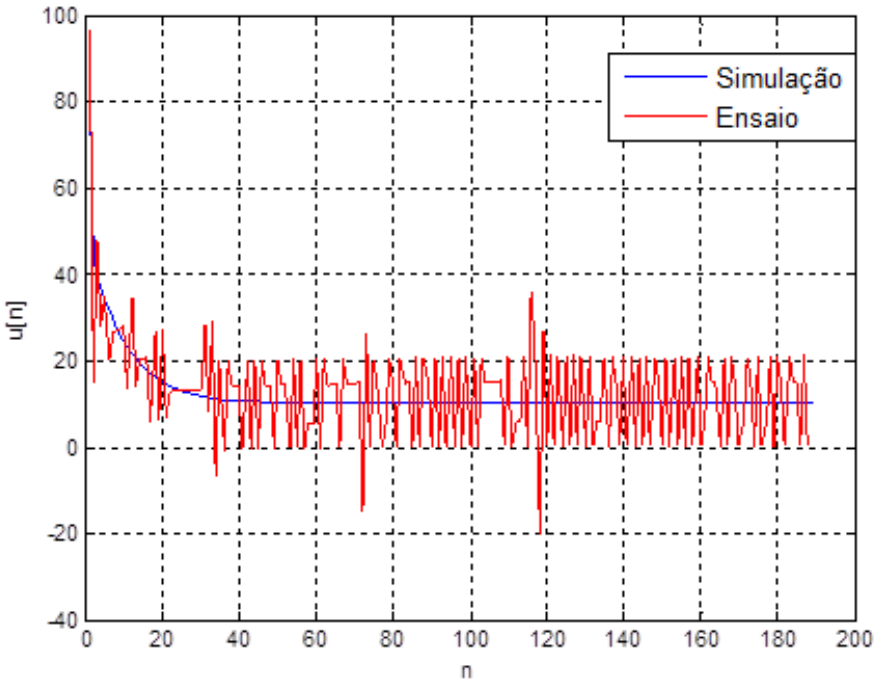
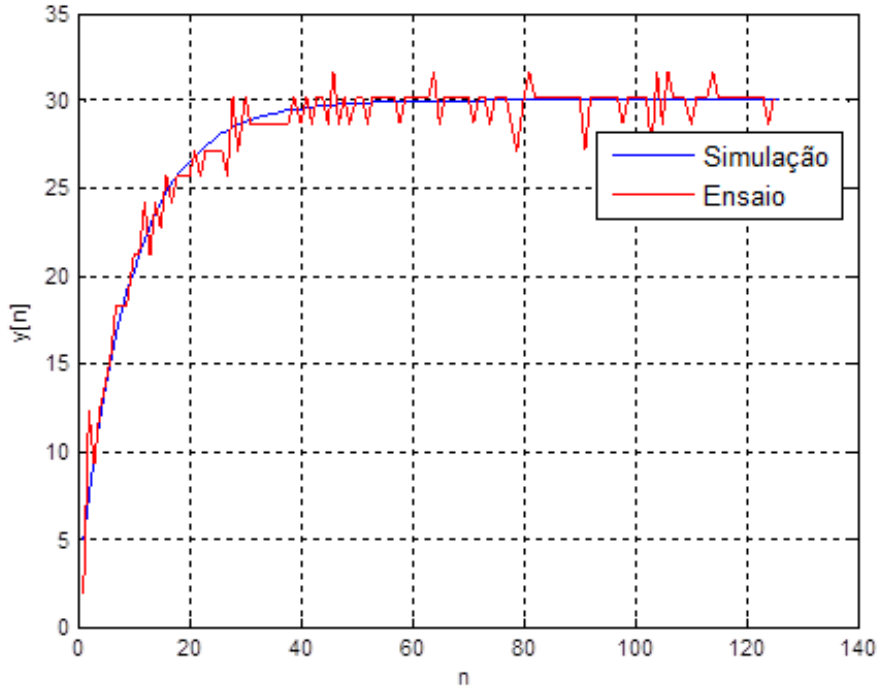


Figura 25. Sinal de controle do sistema considerando 10 como referência.



Na Figura 24 e na Figura 25, é possível verificar que o sistema realiza o controle da maneira esperada, entretanto, obtém-se um grande nível de ruído. Isto se dá porque a referência adotada (10) representa 10 níveis binários na saída do Arduino dos 1024 possíveis. Como a tensão utilizada pela placa está no intervalo $(0{,}55\text{V}, 2{,}75\text{V})$, essa referência de 10 níveis representa uma diferença de tensão de aproximadamente $2{,}1\text{mV}$, sendo, então, muito suscetível ao ruído. Aumentando-se a referência para 30 (diferença de tensão de 6,3mV) obtém-se o gráfico mostrado na Figura 26. A curva em vermelho representa o ensaio e a curva em azul, a simulação.

Figura 26. Resposta do sistema ao degrau de amplitude 30.



Observa-se na Figura 26 que o nível de ruído, em comparação com a Figura 24 é reduzido. Na Figura 27 é apresentado o sinal de controle obtido nesse ensaio.

É possível verificar na Figura 27 que o sinal de controle chega ao valor máximo de cerca de 270 que está ainda abaixo do sinal máximo possível 512 (lembrando que os 1024 níveis foram divididos entre valores positivos e negativos). Caso a referência fosse aumentada

demasiadamente, o Arduino seria incapaz de suprir o nível de tensão desejado, ocasionando *overflow* na saída. Na Figura 28 e na Figura 29, observa-se os gráficos da saída do sistema controlado e o sinal de controle considerando-se 50 como referência.

Figura 27. Sinal de controle do sistema considerando 30 como referência.

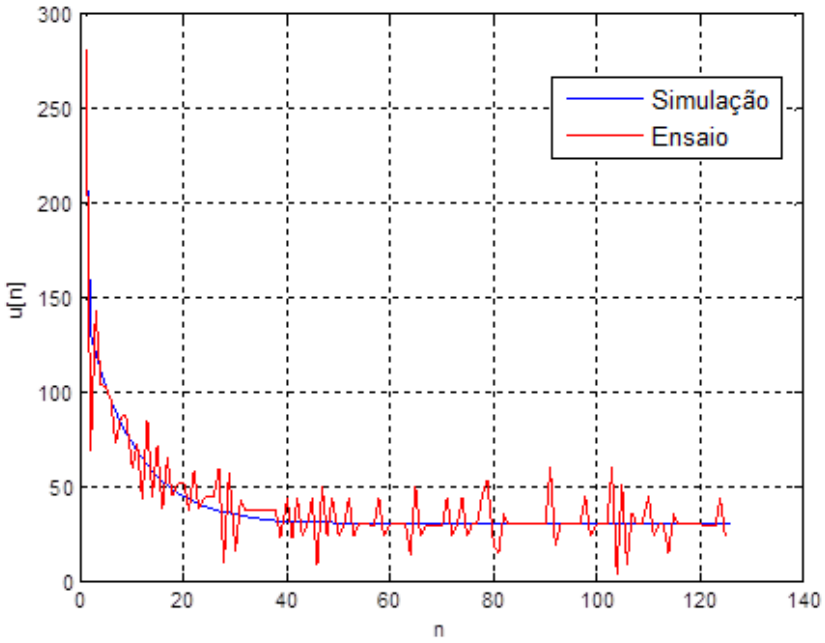


Figura 28. Resposta do sistema ao degrau de amplitude 50.

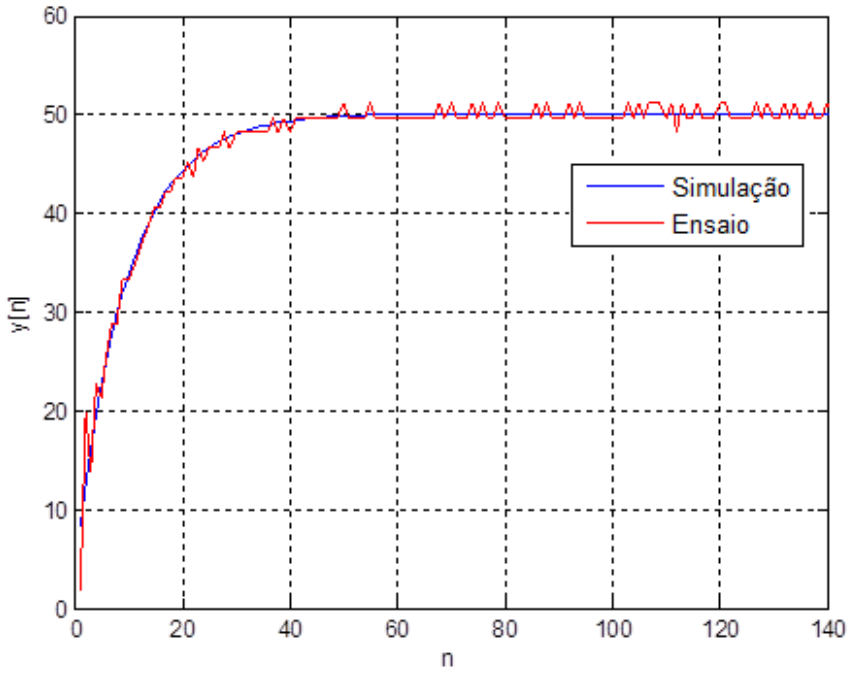
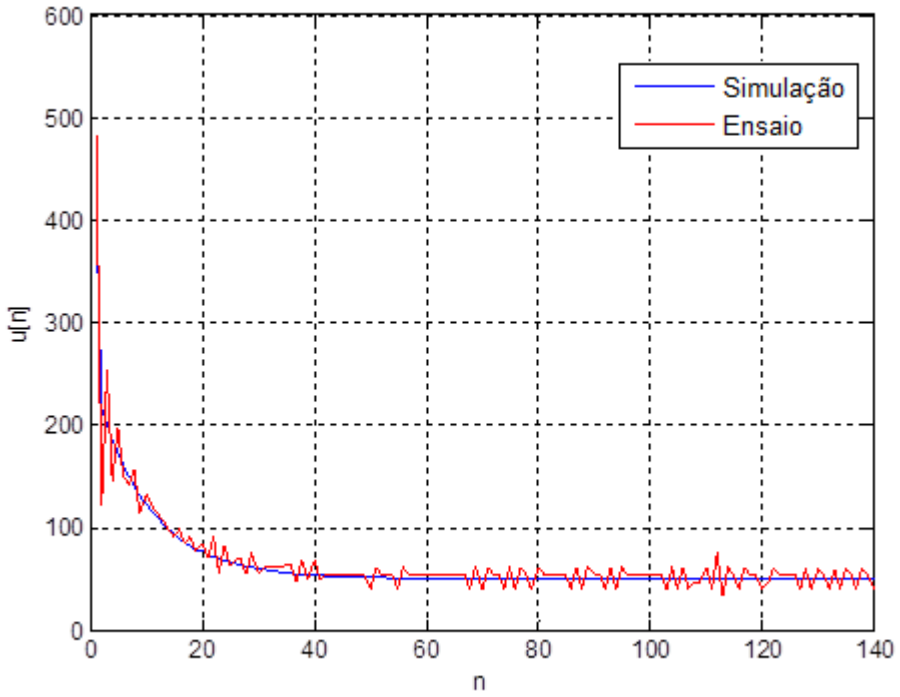


Figura 29. Sinal de controle do sistema considerando 50 como referência.



Através da Figura 28 e da Figura 29, percebe-se que o nível de ruído é muito reduzido em relação aos outros ensaios e o sinal de controle chega ao nível máximo de aproximadamente 500, ainda abaixo do nível máximo possível (512). Entretanto, ao selecionar-se uma referência de 100, se obtém o gráfico apresentado na Figura 30. Observa-se então que o sistema não está mais controlado da maneira desejada. Isso ocorre porque o sinal de controle necessário para controlá-lo excede o nível 512, ou seja, ocorre *overflow* e o Arduino não é capaz de suprir a tensão necessária para controlar o sistema. O gráfico da Figura 31 mostra a discrepância entre o sinal esperado e o sinal obtido no Arduino. Para aperfeiçoar o método neste sentido, adota-se uma condição de saturação no *firmware* para a saída: se o sinal instantâneo na forma "P_r" for menor do que -511, adota-se -511 como saída, da mesma maneira, se este sinal for maior do que 512, adota-se 512 como saída.

Figura 30. Resposta do sistema ao degrau de amplitude 100 (simulado e ensaiado).

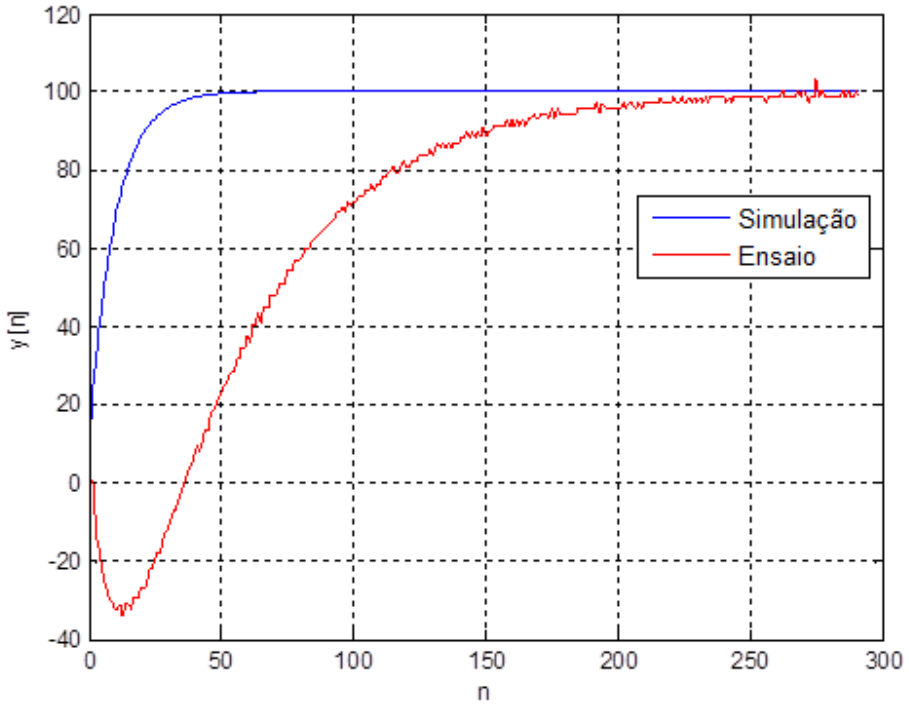
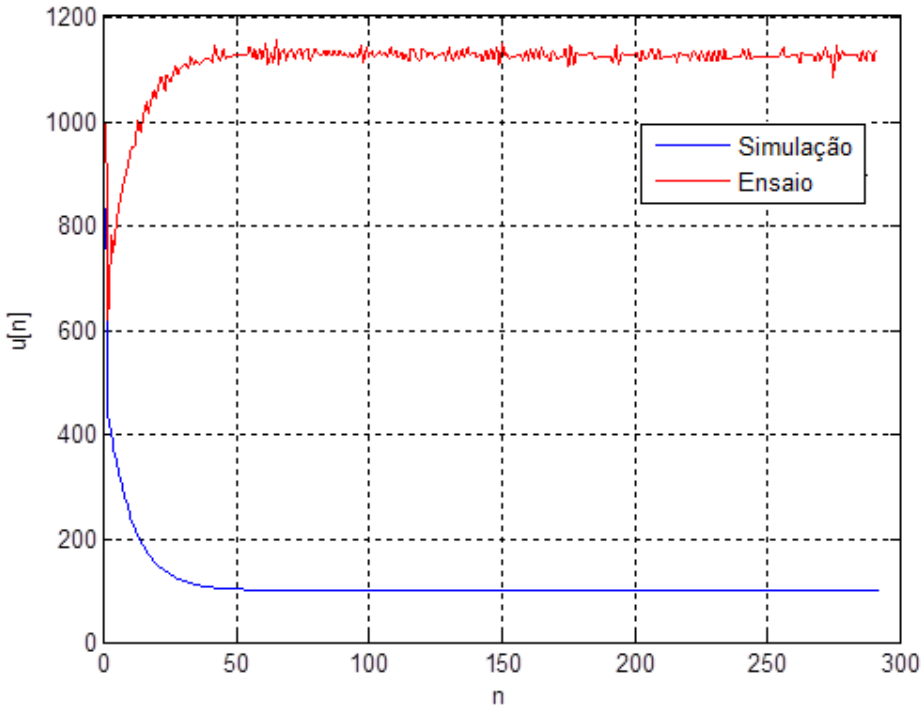


Figura 31. Sinal de controle do sistema considerando 100 como referência.

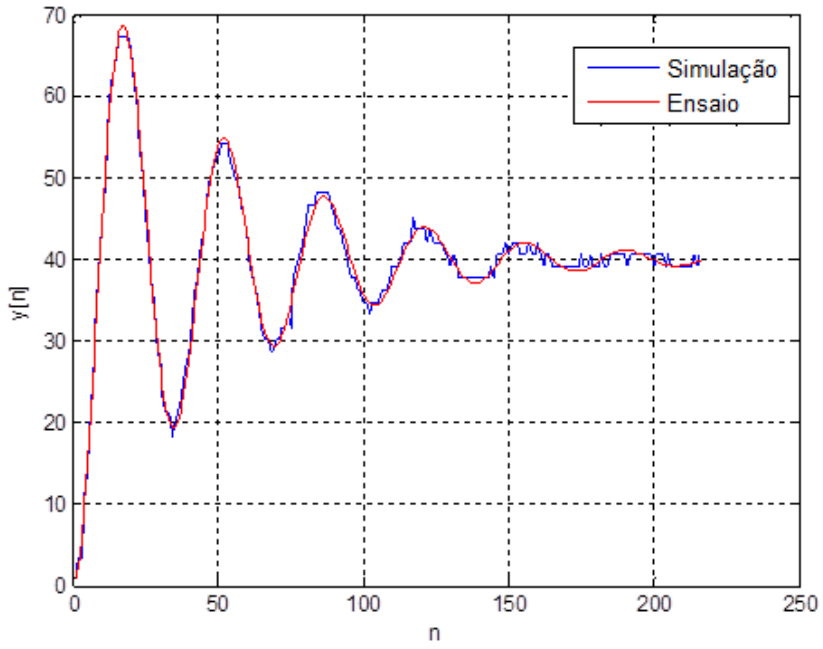


Para comprovar a eficácia do sistema de controle adotado para valores positivos e negativos, define-se um novo controlador arbitrariamente em (58).

$$C(z) = \frac{(z + 0{,}2)(z + 0{,}4)}{z(z - 1)} \tag{58}$$

O sistema dado por (58) é simulado e ensaiado com um degrau como entrada e sua resposta temporal é apresentada na Figura 32. Nesta figura, o ensaio é apresentado em azul e a simulação, em vermelho.

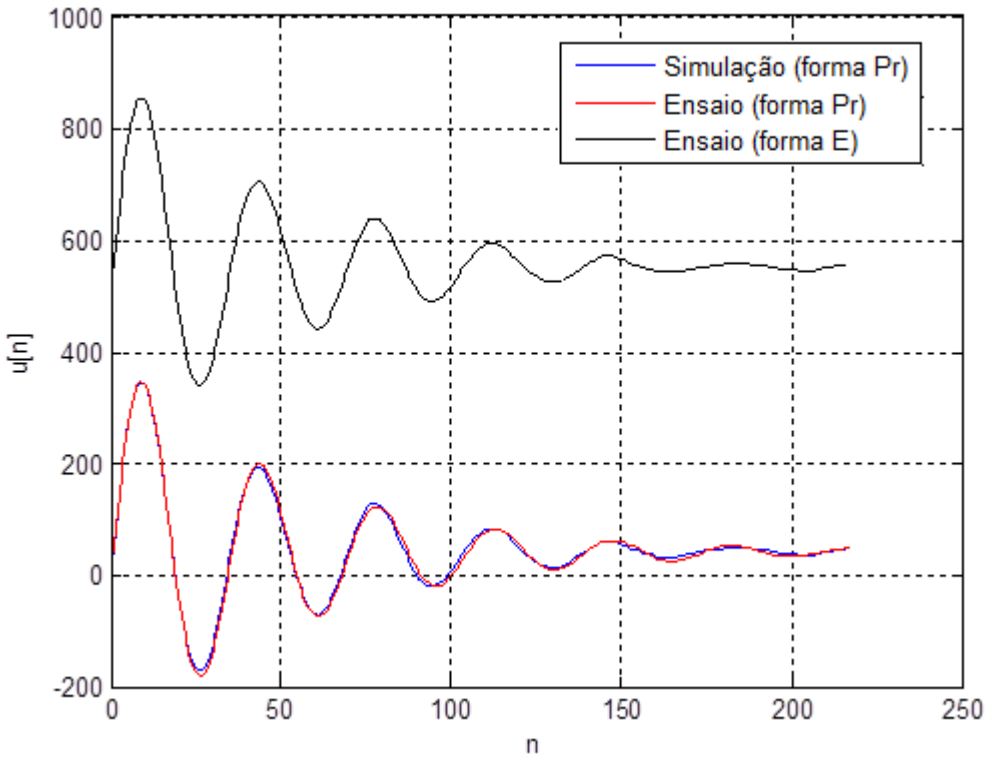
Figura 32. Resposta ao degrau do sistema com o controlador arbitrário.



Observa-se na Figura 32 que o sistema simulado representa, de fato, o sistema ensaiado. Na Figura 33 é apresentado o sinal de controle. Em vermelho a curva simulada, em azul a curva ensaiada e em preto, a curva ensaiada na forma “escrita”, ou seja, os valores que o Arduino escreve no sistema com relação à referência (nível 511). Na Figura 33, é possível

comprovar a eficácia do sistema de controle, mesmo quando se aplicam níveis negativos de tensão.

Figura 33. Sinal de controle do sistema simulado e ensaiado.

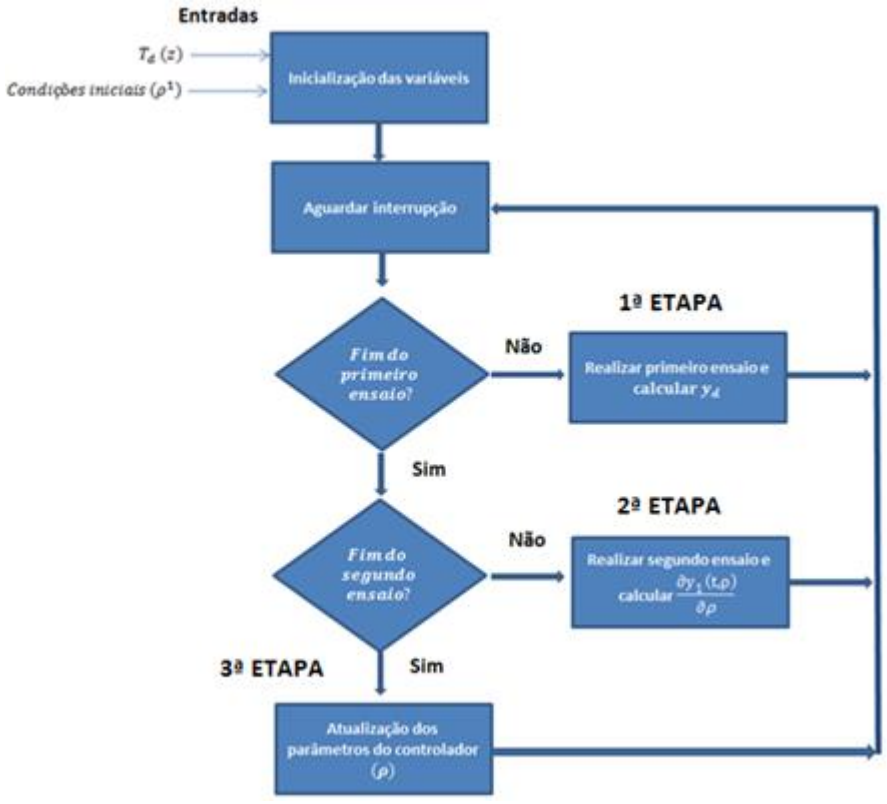


4.2 IMPLEMENTAÇÃO EMBARCADA DO MÉTODO IFT

Para a realização embarcada do método IFT se faz necessário utilizar os conceitos discutidos até então. O método deverá seguir o algoritmo apresentado na Figura 5 com algumas pequenas adaptações. Quando a implementação era realizada de maneira numérica, não havia preocupações com memória e tampouco com o tempo de realização de cada simulação. Os valores resultantes em cada ensaio simulado eram armazenados em vetores e após, eram tratados. No caso da implementação embarcada, a memória é muito reduzida em relação à memória disponível no computador onde a simulação foi realizada, desta forma, o uso de memória deve ser otimizado. Analisando o método IFT, verifica-se que é necessário

armazenar, ao final do primeiro ensaio, pelo menos dois vetores, um contendo o sinal $e_1(t,\rho)$ que será utilizado como entrada no segundo ensaio e outro contendo o erro $y_1(t,\rho) - y_d(t)$ que será utilizado posteriormente para o cálculo de $\nabla J(\rho)$. Desta maneira, pode-se realizar simultaneamente o cálculo de $y_d(t)$ e o ensaio de $y_1(t,\rho)$, com intuito de armazenar diretamente o vetor contendo $[y_1(t,\rho) - y_d(t)]$. Após a finalização da primeira etapa, realiza-se a segunda. Da mesma maneira, simultaneamente ao ensaio para obtenção de $y_2(t,\rho)$ pode ser realizada a filtragem instantânea deste pelo filtro $\frac{1}{C(z,\rho)}\frac{\partial C(z,\rho)}{\partial \rho}$. Dessa forma, ao final da segunda etapa, obtém-se três vetores contendo os sinais referentes a $\frac{\partial y_1(t,\rho)}{\partial \rho_1}, \frac{\partial y_1(t,\rho)}{\partial \rho_2}$ e $\frac{\partial y_1(t,\rho)}{\partial \rho_3}$, nesta mesma etapa, calcula-se já o gradiente $\nabla J(\rho)$. Ao final dessa etapa, resta a atualização de cada parâmetro do controlador. Assim, pode-se dizer que o algoritmo utilizado é divido em três etapas e é sintetizado pelo diagrama de fluxo da Figura 34.

Figura 34. Diagrama de fluxo do algoritmo embarcado.

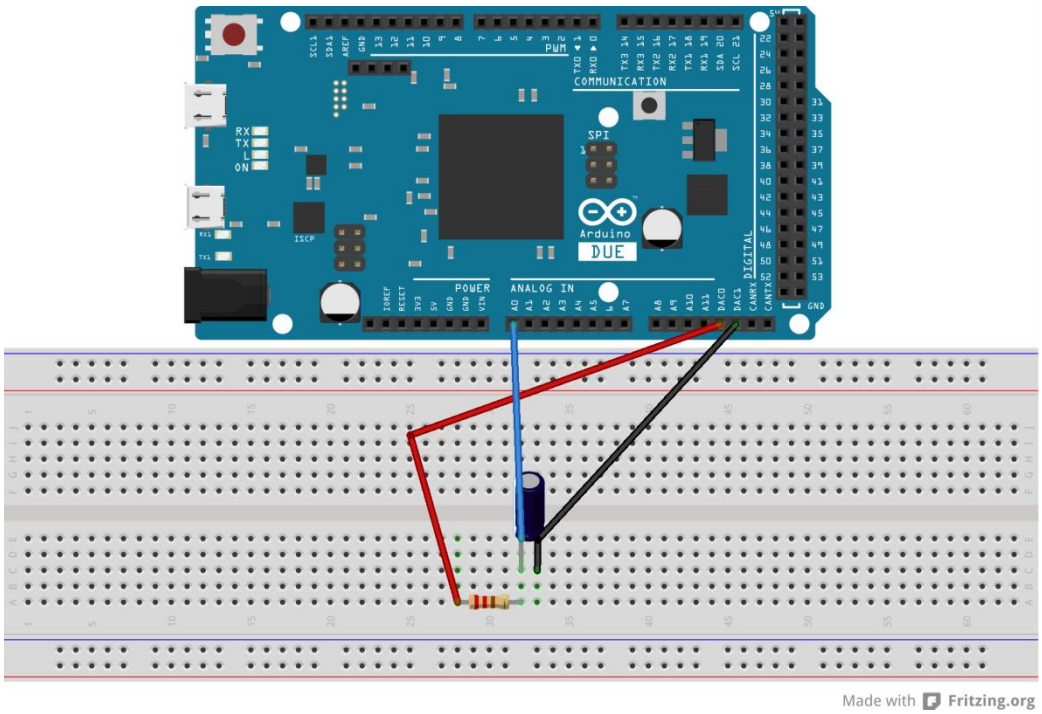


O diagrama da Figura 34 mostra a topologia inicial adotada para o desenvolvimento do código *ift.ino* (em anexo) que é o responsável pela aplicação do método IFT**.** Os dados obtidos são recuperados para o computador através da interface serial do Arduino para que se possa realizar a análise dos resultados.

4.3 RESULTADOS E TESTES

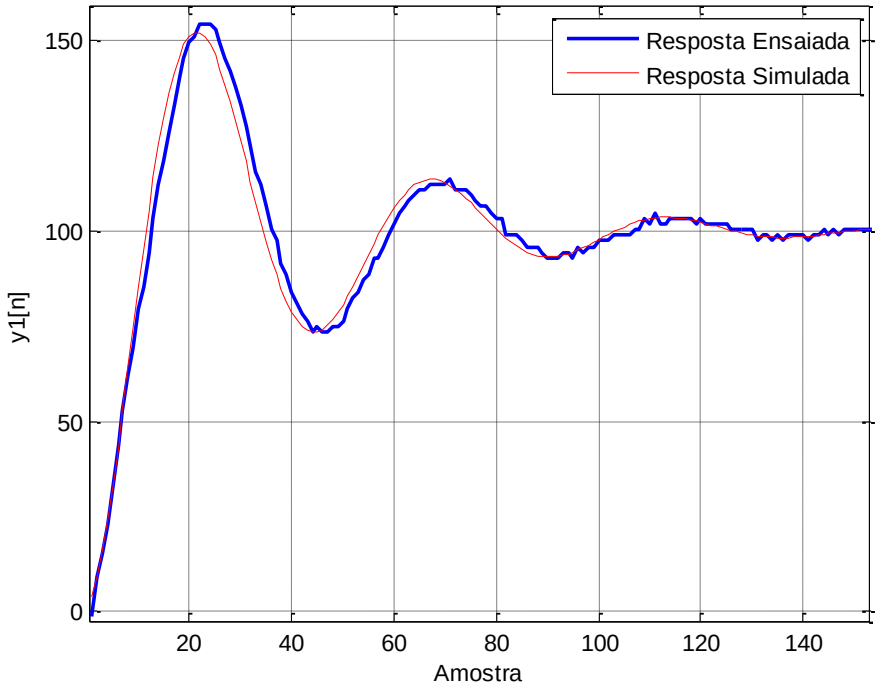
A partir do código *ift.ino* (em anexo)**,** serão estudados dois exemplos de implementação embarcada do método IFT visando apresentar resultados similares aos obtidos computacionalmente. O primeiro deles é a implementação no processo da Figura 12 (circuito RC-série), tal qual o apresentado em 4.2. O *layout* do sistema embarcado é apresentado na Figura 35.

Figura 35. Layout do sistema embarcado com circuito RC-série.



O *layout* apresentado na Figura 35 mostra que através das saídas analógicas do Arduino DAC0 e DAC1 é aplicada uma tensão no sistema. A tensão no capacitor, que é a variável que se deseja controlar, é medida através da entrada analógica A0. Na aplicação, tem-se $u(t,\rho) = [sinal(DAC0) - sinal(DAC1)]$ e $y(t,\rho) = [sinal(A0) - sinal(DAC1)]$. Submetendo-se esse sistema aos dois ensaios do método IFT, ou seja, considerando apenas uma iteração, recuperam-se os valores de $y_1(t,\rho), y_2(t,\rho)$, $\nabla J(\rho)$ e do novo conjunto de parâmetros gerado, considerando a mesma $T_d(z)$ apresentada em (38) e assim, compara-se com os obtidos no $Matlab^{®}$. As condições iniciais adotadas são $k_p = 1, k_i = 1\ e\ k_d = 0{,}1$. A resposta do primeiro ensaio ao degrau é apresentada na Figura 36.

Figura 36. Resposta do sistema ao primeiro ensaio.

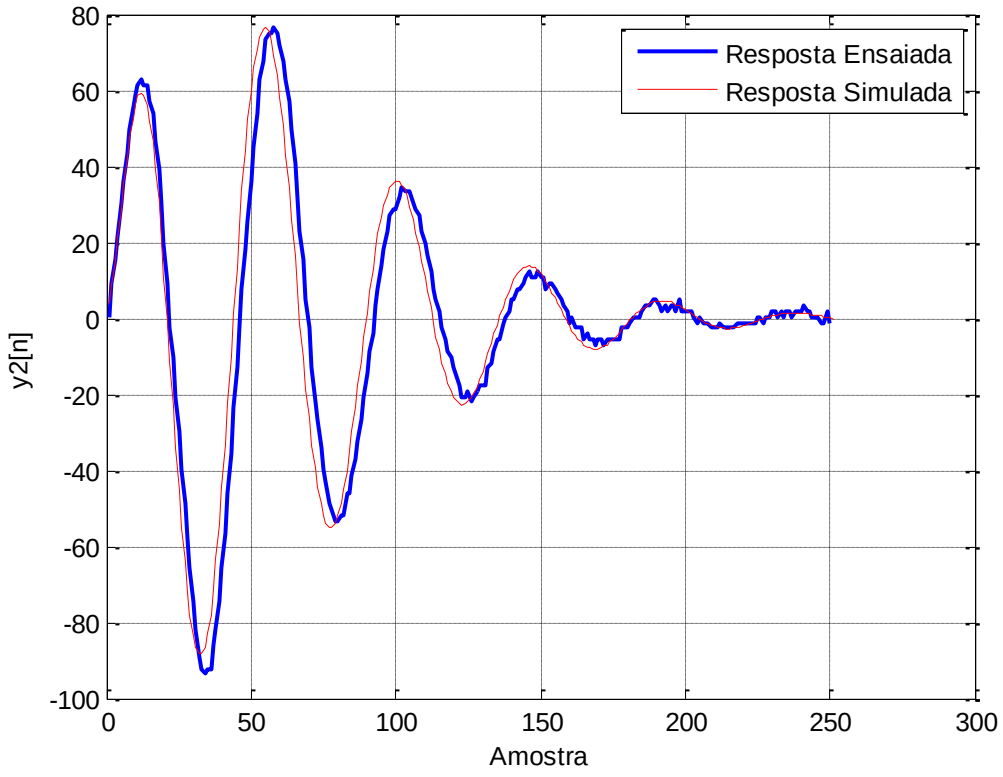


Observa-se na Figura 36 que o resultado do primeiro ensaio realizado é similar ao obtido computacionalmente. O tempo de execução foi de aproximadamente $80\mu s$ para cada aquisição de dado e processamento. Esse tempo é medido via *firmware* e indica que poderia

se realizar o processamento até para frequências de amostragens da ordem de 10kHz. Muito superiores aos $50Hz$ utilizados.

Na Figura 37 é apresentado o resultado do segundo ensaio. Tal qual o primeiro, a curva ensaiada é similar a curva ensaiada. O tempo de execução do processamento é de aproximadamente $107\mu s$. O tempo observado é maior do que no primeiro ensaio porque neste segundo são realizadas três filtragens para estimar os termos $\frac{\partial y_1(t,\rho)}{\partial \rho_1}$, $\frac{\partial y_1(t,\rho)}{\partial \rho_2}$ e $\frac{\partial y_1(t,\rho)}{\partial \rho_3}$ e também calcula-se o gradiente $\nabla J(\rho)$.

Figura 37. Resposta do sistema ao segundo ensaio.



Para verificar a eficiência da terceira etapa do algoritmo (atualização dos parâmetros), são mostrados na Tabela 3 os resultados obtidos no cálculo do novo conjunto de parâmetros, comparando-os com os obtidos computacionalmente.

Tabela 3. Relação entre resultados obtidos (Matlab e Arduino).

Parâmetro	Matlab	Ensaio
$\frac{\partial J(\rho)}{\partial \rho_1}$	0,0056	0,01
$\frac{\partial J(\rho)}{\partial \rho_2}$	0,0163	0,02
$\frac{\partial J(\rho)}{\partial \rho_3}$	0,0264	0,03
ρ_1	2,067	2,06
ρ_2	-1,296	-1,3
ρ_3	-0,05484	-0,06

Analisando-se a Figura 36, a Figura 37 e a Tabela 3**,** observa-se que o resultado da implementação embarcada reproduz a computacional. Assim, pode-se realizar o método iterativo e observar seu resultado para um número maior de iterações. Na Figura 38 observa-se a resposta à referência ($\mathrm{r}_1(\mathrm{t}) = 100$) do sistema com o controlador adotado na primeira iteração e o controlador gerado após 43 iterações. O tempo de realização das 43 iterações foi de aproximadamente 8 minutos.

Observa-se na Figura 38 que, após 43 iterações, a resposta do sistema de fato é mais similar à resposta desejada em relação à primeira, reduzindo o erro médio quadrático de $J^1(\rho) = 329{,}69$ para $J^{43}(\rho) = 11{,}72$. A evolução da função custo é apresentada na Figura 39.

Realizando-se uma simulação com o $Matlab^{®}$, considerando-se as mesmas condições iniciais, número de iterações e ruído, é possível comparar os resultados. Na Figura 40, apresenta-se a resposta do sistema à referência ($r_1(t) = 100$)) com o controlador obtido no Arduino e computacionalmente.

Figura 38. Resposta do sistema após a aplicação do método IFT.

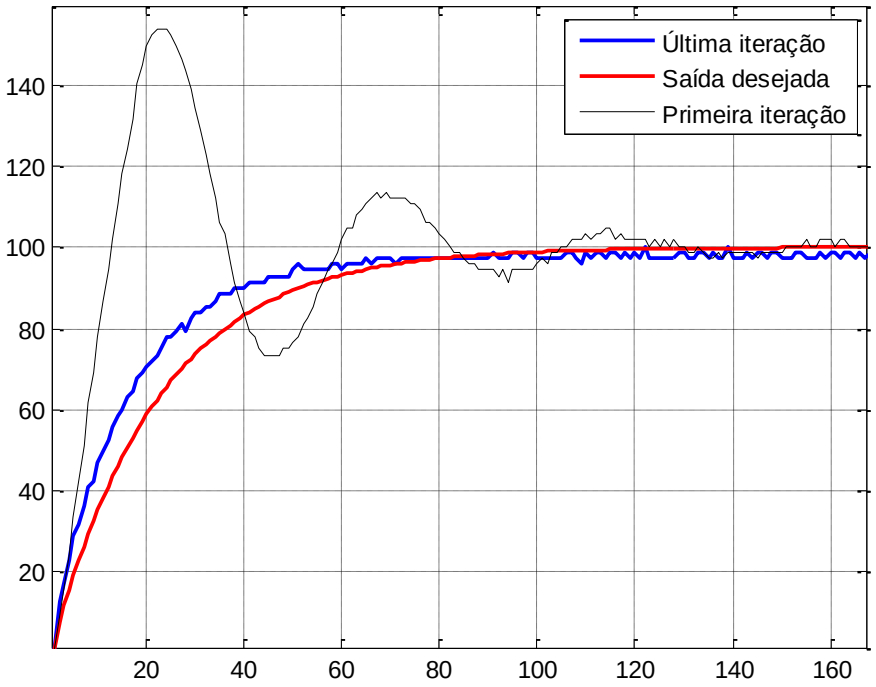
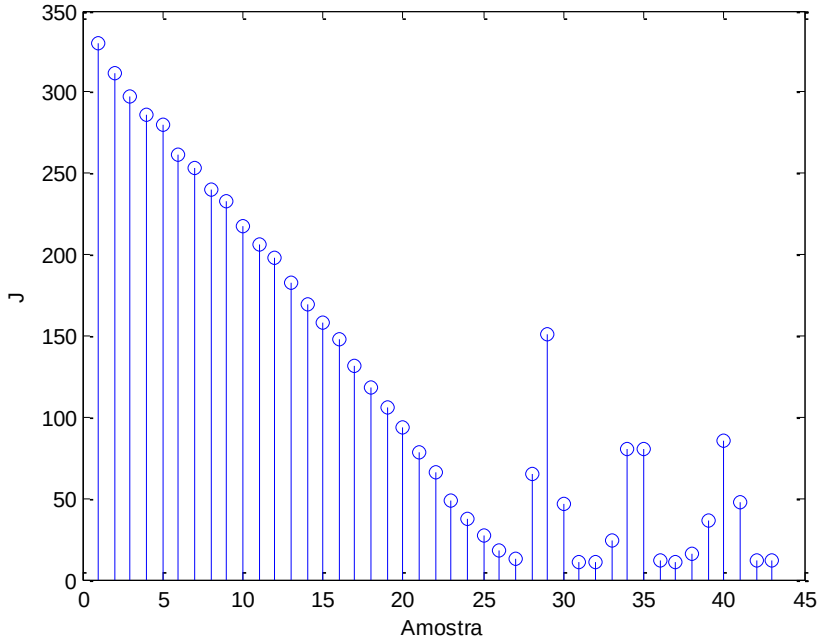


Figura 39. Evolução da função custo conforme a iteração.



É possível observar na Figura 40 que o controlador obtido no sistema embarcado resulta em uma resposta mais lenta e mais ruidosa do que a do sistema simulado. Verifica-se a

diferença entre os controladores obtidos na Figura 41 e entre o controlador obtido no método embarcado e o controlador ideal na Figura 42.

Figura 40. Resposta do sistema após realização do método IFT.

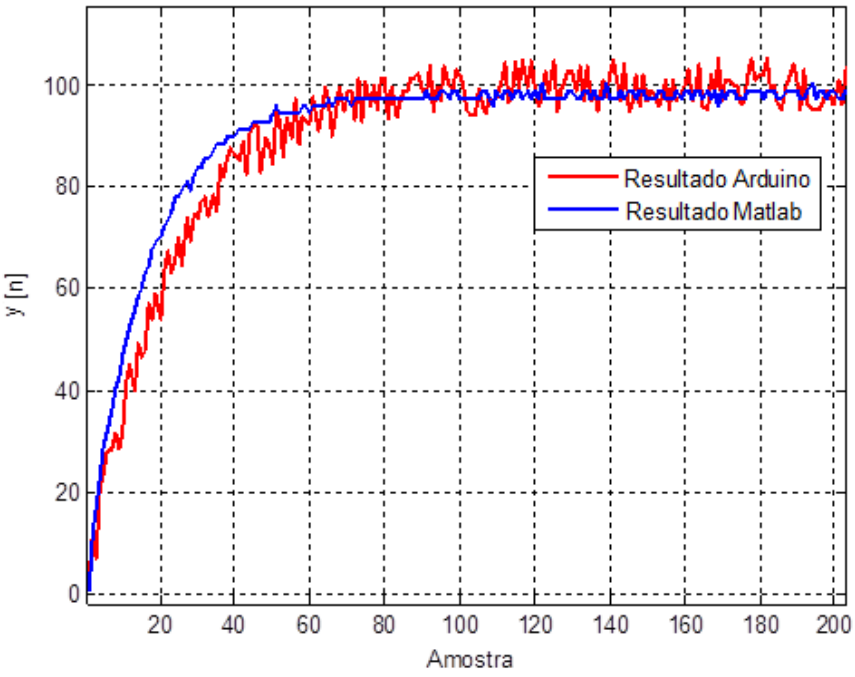


Figura 41. Relação entre o controlador obtido e o controlador simulado.

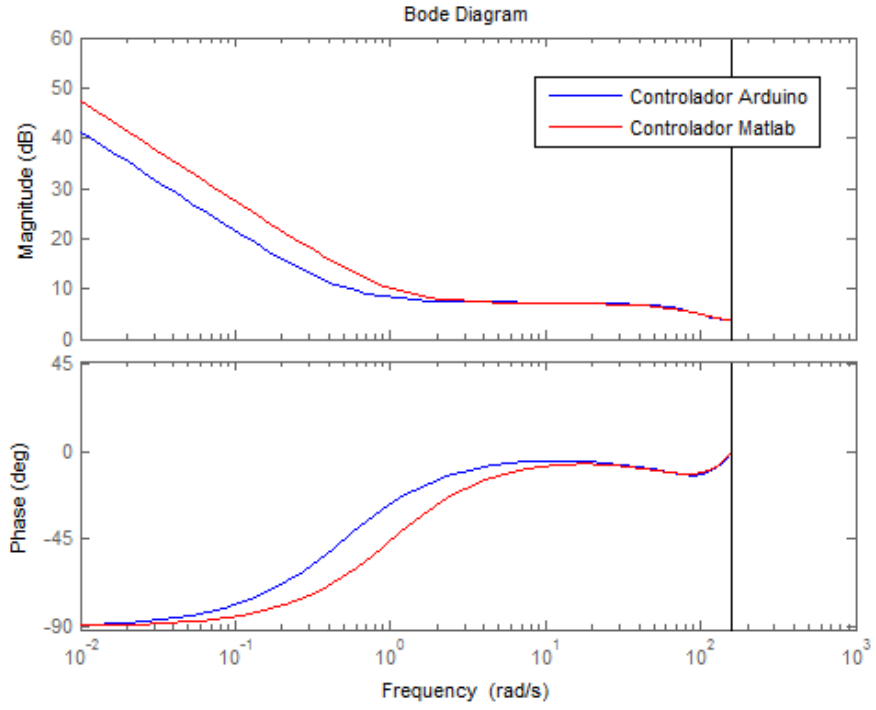
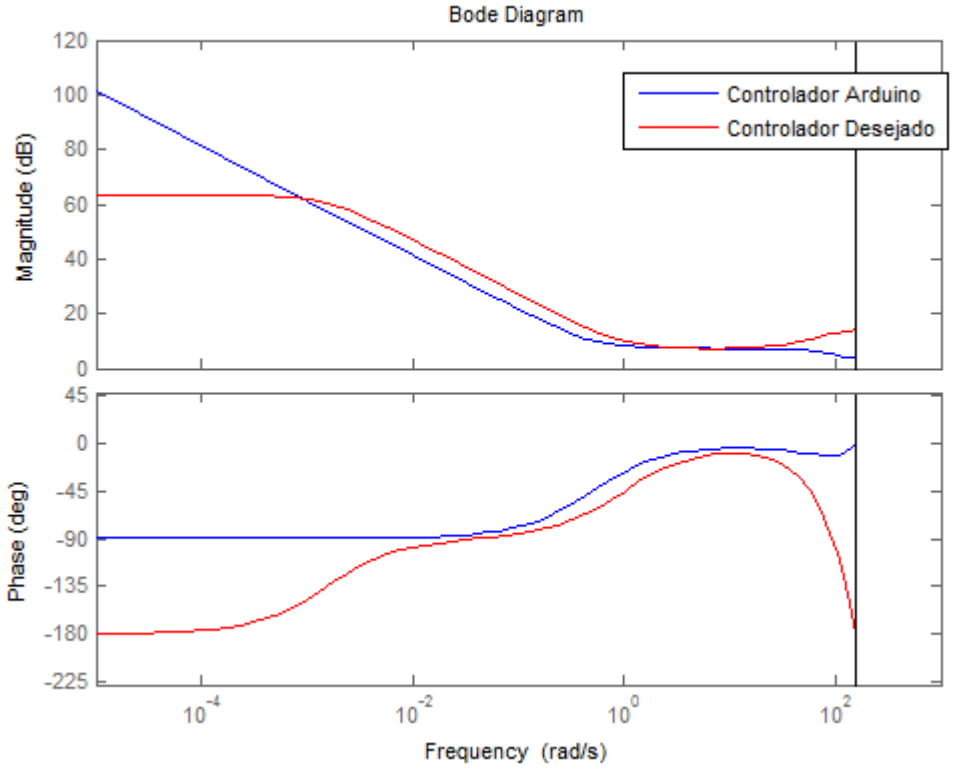


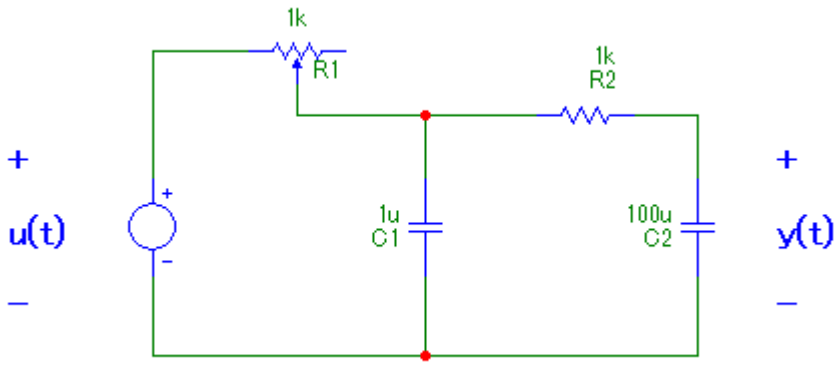
Figura 42. Relação entre o controlador obtido e o controlador ideal.



Observa-se na Figura 41 e na Figura 42 que, para baixas frequências (ordem de algumas unidades de rad/s) os diagramas são similares. Como o sistema opera nessa ordem de frequência, as discrepâncias para altas frequências não alteram significativamente o comportamento do sistema em relação ao desejado.

O próximo exemplo a ser estudado é o processo dado pela Figura 43. O circuito ainda possui um resistor variável (*trimpot*) para simular uma variação comportamental da planta.

Figura 43. Circuito com dois polos.



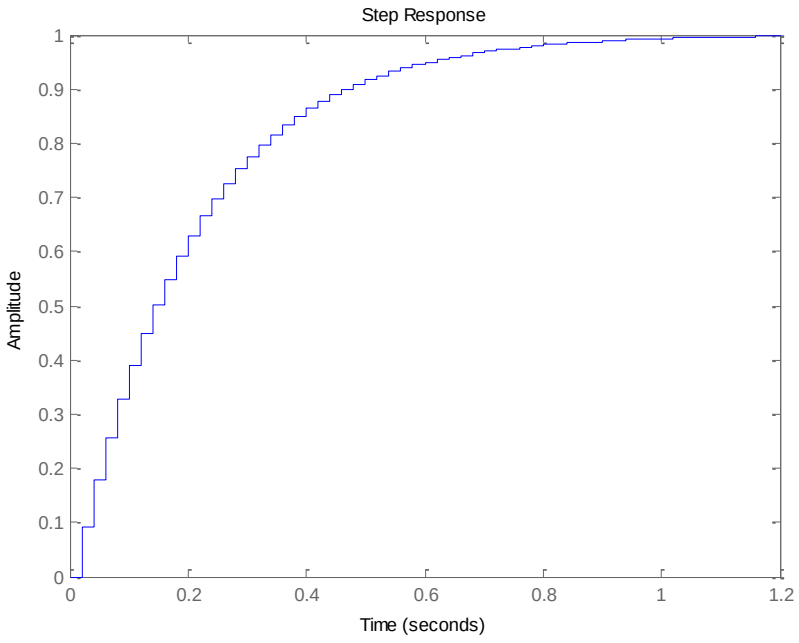
Discretizando-se o circuito da Figura 43 a uma frequência de amostragem $f_s =$ 50Hz, obtém-se (60).

$$G(s) = \frac{1000}{(s + 2005)(s + 4{,}988)} \tag{59}$$

$$G(z) = \frac{0{,}0927z + 0{,}00226}{z^2 - 0{,}9051z} \tag{60}$$

A resposta ao degrau unitário deste sistema é mostrada na Figura 44.

Figura 44. Resposta do sistema G(z) ao degrau unitário.

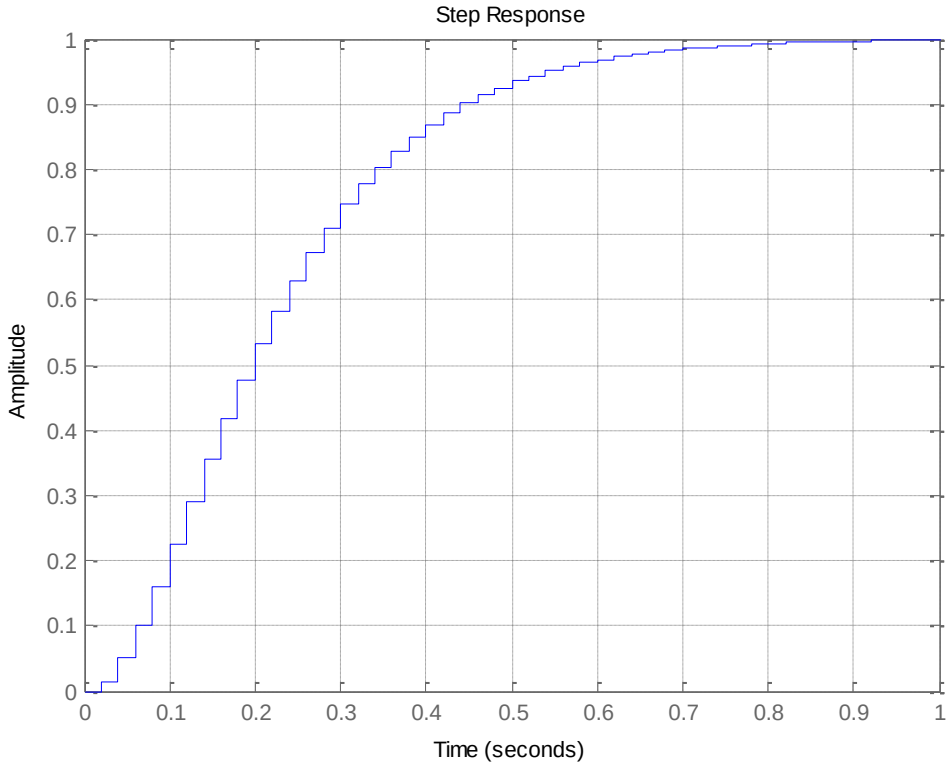


Observa-se na Figura 44 que o sistema tem um tempo de acomodação de aproximadamente 1s. Portanto, define-se que o objetivo neste exemplo é de realizar um controle que mantenha esse tempo de acomodação, qualquer que seja a posição do trimpot $R1$ e que apresente erro nulo ao seguimento à referência. Com a função de transferência desejada apresentada em (61) obtém-se o desempenho desejado, conforme pode ser visto no gráfico da Figura 46 que é a resposta de $T_d(s)$ ao degrau unitário. A função discretizada $T_d(z)$ é apresentada em (62).

$$T_d(s) = \frac{80}{(s+10)(s+8)} \tag{61}$$

$$Td(z) = \frac{0{,}0142z + 0{,}0126}{z^2 - 1{,}671z + 0{,}6977} \tag{62}$$

Figura 45. Resposta de $\boldsymbol{T_d(z)}$ ao degrau unitário.



Observa-se, portanto, que a função de transferência desejada possui tempo de acomodação de aproximadamente 1s, como desejado. Desta forma, pode-se realizar o método IFT neste sistema, considerando o *trimpot* $R1$ em uma posição aleatória, ou seja, não se sabe o valor exato da sua resistência.

A Figura 46 ilustra o sistema ensaiado. Desta forma, é implementado o método IFT. As condições iniciais adotadas são $k_p = k_i = k_d = 0{,}01$.

O gráfico da Figura 47 mostra a resposta do sistema ao degrau considerando o controlador com as condições inicias (primeira iteração), a resposta desejada e a resposta do sistema com o controlador obtido na décima iteração.

A Tabela 4 apresenta a evolução da função custo com relação à iteração. A resistência do trimpot é colocada aleatoriamente em $R = 445\Omega$.

Figura 46. Ilustração do sistema utilizado.

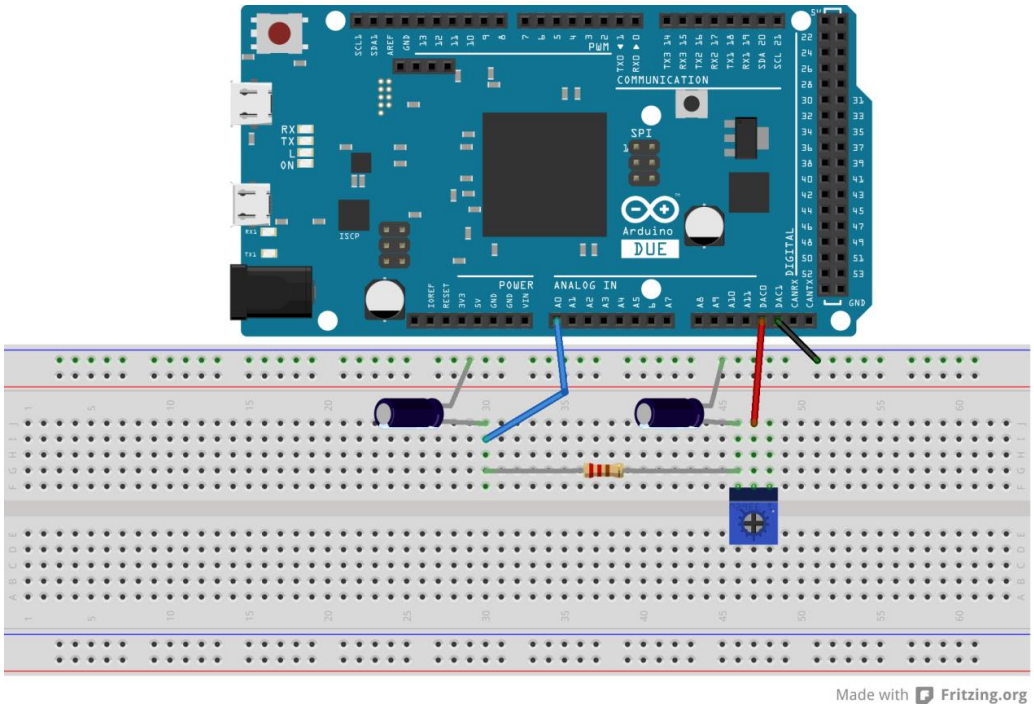


Figura 47. Resposta do sistema ao degrau.

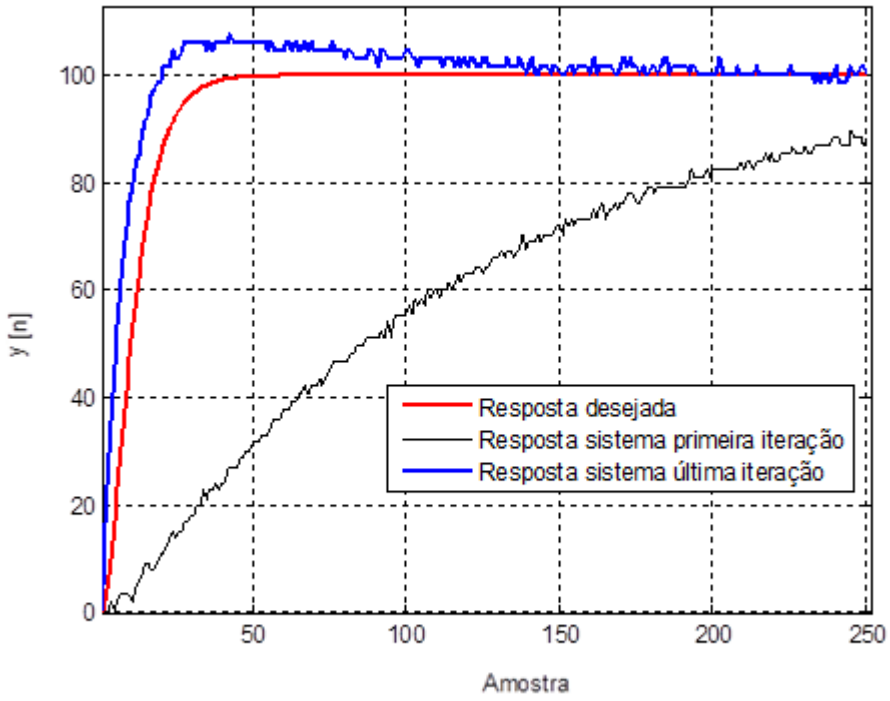


Tabela 4. Evolução da função custo com relação à iteração.

Iteração	J
1	1961,38
2	52,13
3	71,28
4	61,43
5	54,19
6	49,68
7	45,84
8	42,14
9	40,9

Observa-se na Figura 47 e na Tabela 4 que o sistema, de fato, reduz a função custo e aproxima a resposta em laço fechado à resposta desejada. O controlador obtido possui $k_p = 0{,}0115, k_i = 0{,}0084$ e $k_d = 0{,}0095$.

Reposicionando o *trimpot* R1 de maneira aleatória, obtém-se um novo valor de resistência desconhecido. Com esse valor desconhecido, realiza-se outra vez o método IFT de

maneira a obter novamente o sistema controlado. Na Figura 48 observa-se a evolução da resposta do sistema após a aplicação do método IFT. Da mesma forma, na Tabela 5, obtém-se a evolução do valor do custo.

Figura 48. Resposta do sistema ao degrau.

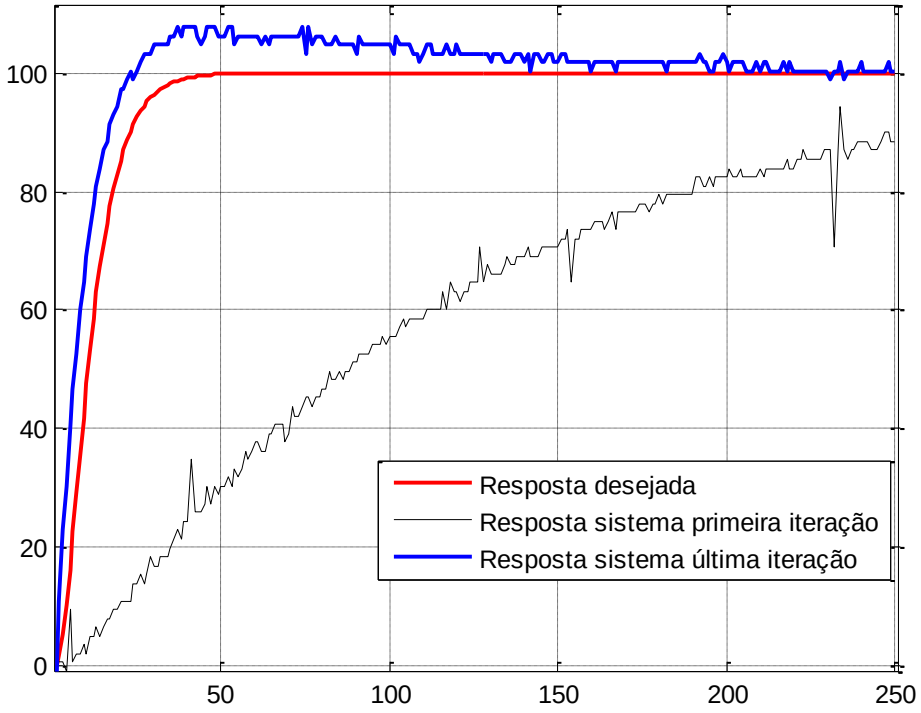


Tabela 5. Evolução da função custo com relação à iteração considerando valor aleatório de resistência.

Iteração	J
1	1993,76
2	47,62
3	63,99
4	53,58
5	47,04
6	39,47
7	39,22
8	30,97
9	29,64

Pode-se inferir, observando os resultados obtidos que o método se mostra eficiente na aplicação desejada. Desta forma, se valida o código realizado.

4.4 CONSIDERAÇÕES FINAIS

Após a aplicação prática do método, verifica-se que não há um passo γ que seja eficiente para qualquer planta, desta forma, deve-se avaliar a resposta do método para um determinado passo e, verificando a resposta em cada iteração, utilizar um novo valor de passo. Em todas as execuções realizadas neste trabalho, o passo foi dividido por um fator proporcional à iteração para forçar a convergência. Mesmo assim, em alguns ensaios realizados, a resposta do método não convergiu. Como complementação ao trabalho realizado, pode-se adicionar um método de ajuste do passo (ECKHARD, 2012).

5 CONCLUSÃO

Concretizado o projeto, pode-se verificar que, de fato, o método IFT pode apresentar-se muito eficaz para minimizar problemas de modelagem e variações inesperadas de parâmetros do processo. A realização do método de forma computacional mostrou que, de fato, os resultados minimizam a função custo e o método aplicado de forma embarcada apresenta-se de maneira muito similar. Como discutido no corpo do projeto, há melhorias que podem ser adotadas para otimizar o trabalho realizado para que este seja totalmente independente e não seja necessário um usuário com algum conhecimento técnico para utilizá-lo, sobretudo por questões de passo de iteração e saturação do sinal de controle.

O resultado do projeto é um sistema embarcado de auto ajuste de parâmetros que, graças a seus resultados, pode ser aplicado em diversos processos. A facilidade com que o método oferece ao usuário um sistema próximo ao desejado justifica o uso do sistema embarcado em larga escala, desde processos industriais até o controle de aparelhos residenciais, visando garantir o comportamento do mesmo ao longo de sua vida-útil. Desta forma, verifica-se a viabilidade de implementação embarcada do método em plataformas de baixo custo.

REFERÊNCIAS

Atmel Corporation. **SAM3X/SAM3A Series Complete** disponível por http://www.atmel.com/Images/doc11057.pdf (03 dez. 2013)

ÅSTRÖM, K.J.; WITTENMARK, B. **Adaptive control**. 2.ed. Prentice Hall, Englewood Cliffs, 1994; 580 p. ISBN: 0201558661

BAZANELLA, A.S.; CAMPESTRINI, L.; ECKHARD, D. **Data-driven Controller Design**. Springer, 2012; 241 p. ISBN: 9400722990

ECKHARD, D. **Ferramentas Para Melhoria da Convergência dos Métodos de Identificação por Erro de Predição.** 2012. 108 p. Tese (Doutorado em engenharia elétrica) – Universidade Federal do Rio Grande do Sul, Porto Alegre. 2012

KUO, B.C. **Digital Control Systems.** 2.ed. Saunders College Publishing, Orlando, 1992; 751 p. ISBN: 0030128846

ANEXO:

Códigos utilizados

IFT.M

```
close all;
clear all;
%% DEFINICOES INICIAIS
t1=12; % tempo de cada simulacao
Ts=0.02;
f=1/Ts;
t=t1*f;
%% Processo
G=tf([0.01652],[1 -0.9835],1);
%% Sistema dejado
Td=zpk([0.4],[0.9008],[0.16528],1);
%% Controlador
%condicoes iniciais
kp=5.958;
ki=0.121;
kd=3.921;
%ganhos nos termos z^2, z, z^0
ro=[];
ro(1)=kp+ki+kd; ro(2)=-kp-2*kd; ro(3)=kd;

%PID
numC=ro;
denC=[1 -1 0];
C=tf(numC,denC,1);
%% Derivadas divididas por C
numCC={[1 0 0]; [0 1 0]; [0 0 1]};
denCC=denC;
CC=tf(numCC,denCC,1); %derivada em relacao a ro
dCpC=tf(numCC,ro,1); % derivada em relacao a ro divida por C
%% Controlador na forma Cp+Ci+Cd
Cs=tf({1 [1 0] [1 -1]},{1 [1 -1] [1 0]},1);

%% Controlador IDEAL
Cd=Td/G/(1-Td);

%% Geracao da entrada / tempo de simulacao
r1=ones(t+1,1);

%% simulacoes
 gamma0=norm(ro)/5; % normalizacao para a mesma ordem de grandeza
 K=[];
 J=[];
 h = waitbar(0,'Rodando...');
 %%figure(4);hold on;
 figure(1)
 hold on
 tzz=10;
 y11=[];
for i=1:tzz
    gamma=gamma0%/(sqrt((i))); %

    %simulacao ideal
    [yd]=lsim(Td,r1);

    %simulacao 1
```

```
    sysA=feedback(C*G,1);
    [y1]=lsim(sysA,r1);
    e1=r1-y1;
    y11=[y11 y1];
    %simulacao 2
    [y2]=lsim(sysA,e1);
    y2=y2;

    % simulacao do filtro para obtencao de dY/dpo
    [dYdp]=lsim(dCpC,y2);

    %calculo do novo parametro
    erro=(y1-yd);
    grad=2*erro'*dYdp/t;
    ro=ro-gamma*grad/(norm(grad)+1);

    %parametros do controlador
    kp=-2*ro(3)-ro(2);
    ki=ro(1)+ro(2)+ro(3);
    kd=ro(3);

    k=[kp;ki;kd];
    K=[K k];

    %atualiar controlador
    C=Cs(1)*kp+Cs(2)*ki+Cs(3)*kd;
    %erro
    j=mean(erro.^2/length(erro));
    J=[J j];

   if i>20
       if ((J(i)==J(i-1)))
           break;
       end
   end
   waitbar(i/(tzz),h)
   plot(K(1,:),'b');
   plot(K(2,:),'g');
   plot(K(3,:),'r'); xlabel('Iteracao');ylabel('Parametros - Azul:Kp,
Verde:Ki, Vermelho:Kd');
end
delete(h);

figure(1)
hold on
plot(K(1,:),'b');
plot(K(2,:),'g');
plot(K(3,:),'r'); xlabel('Iteracao');ylabel('Parametros - Azul:Kp,
Verde:Ki, Vermelho:Kd');


figure(2)
stem(J); xlabel ('Iteracao'); ylabel('J');


figure(3)
subplot(2,1,1);
plot(lsim(Td,r1));grid;ylabel('Yd[n]');xlabel('Amostra');axis([0 t 0 120]);
```

```
subplot(2,1,2);
plot(lsim(G,r1));grid;ylabel('Yg[n]');xlabel('Amostra');axis([0 t 0 120]);

figure(4)
step(feedback(C*G,1));
grid;ylabel('Y[n]');xlabel('Amostra');hold;

figure(5)
bode(Cd,'r');hold;bode(C,'b');
```

FILTRO.INO

```
const int ref = 1; // referencia
int x[2]={0, 0}; //inicializacao do buffer para X
float y[3]={0,0,0}; // inicializacao do buffer para Y

void setup() {

  // initialize serial:
  Serial.begin(9600);
}
void loop() {
 x[0]=x[1]; // atualizacao do valor passado X[n-1] no buffer
 x[1]=ref; // atualizacao do valor presente de X[n] no buffer

 y[0]=y[1]; // atualizacao do valor passado Y[n-2] no buffer
 y[1]=y[2]; // atualizacao do valor passado Y[n-1] no buffer

 y[2]=-0.09962*x[0]+0.1172*x[1]-0.7297*y[0]+1.712*y[1]; // atualizacao do valor presente
Y[n] no buffer
   Serial.println(y[2]);
 delay(200);

}
```

CALIBRAGEM.INO

```
int k=0;
int L;
void setup() {
  // put your setup code here, to run once:
  Serial.begin(112500);
  analogWriteResolution(10);
  analogReadResolution(10);
}

void loop() {
  // put your main code here, to run repeatedly:
  analogWrite(DAC0,k);
  L=analogRead(A0)*1.4927-248.4289;
  Serial.print(k);
  Serial.print("\t");
  Serial.println(L);
  k++;
  while (k>1023){
  }
}
```

TESTE_INTERFACES.INO

```
int k=200;
float E;
float P;
void setup() {
  Serial.begin(112500);
  analogWriteResolution(10);
  analogReadResolution(10);
}

void loop() {
  float kE=k+511;
  analogWrite(DAC0,kE);
  E=analogRead(A0)*1.4927-248.4289;
  P=E-511;
  float a=random(-2, 2);
  float Pa=P*a;
  float Ea=Pa+511;
  analogWrite(DAC0,Ea);
  float Ee=analogRead(A0)*1.4927-248.4289;
  float Pp=Ee-511;

  Serial.print(k);
  Serial.print("\t");
  Serial.print(kE);
  Serial.print("\t");
  Serial.print(E);
  Serial.print("\t");
  Serial.print(P);
  Serial.print("\t");
    Serial.print(a);
  Serial.print("\t");
    Serial.print(Pa);
  Serial.print("\t");
      Serial.print(Ea);
  Serial.print("\t");
      Serial.print(Ee);
  Serial.print("\t");
      Serial.print(Pp);
  Serial.print("\t");

  while (true){
  }
}
```

CONTROLADOR_PID.INO

```
// =+=+=+=+=+=+  Parametro de entrada =+=+=+=+=+=+ //

const int sinalY = A0; // sinal y(t)
const int  ground = A1 ; // leitura do ground
const int sinalU = DAC0;
const int groundWrite = DAC1;
float ta=0.02; // periodo de amostragem
// funcao transferencia
float num[3]={4,-5.521, 1.568}; // parâmetros "ro" do numerador  -> [z^0 z^-1 z^-2]

// =+=+=+=+=+=+  Parametro da simulacao =+=+=+=+=+=+ //

float den[3]={1,-1,0}; // parâmetros do denominador -> [z^0 z^-1 z^-2] (fixos para o
controlador PID)
float freq=1/ta; // frequencia de amostragem
float refP; // referencia
float eP; // sinal de erro
float yP; // sinal de saída do processo

int ind=1; // indice da simulacao
const int ordNum=(sizeof(num)/sizeof(float)); // ordem do polinomio do numerador
const int ordDen=(sizeof(den)/sizeof(float)); // ordem do polinomio do denominador

float x[ordNum]; //inicializacao do buffer para X
float u[ordDen]; // inicializacao do buffer para U

float parX[10]; //inicializacao dos parametros do filtro
float parU[10]; //inicializacao dos parametros do filtro

// =+=+=+=+=+=+  Inicializacao =+=+=+=+=+=+ //

void setup() {
  Serial.begin(115200); // inicializar comunicacao
  parametros(); // gerar parametros da funcao de diferencas
  analogWriteResolution(10);
  analogReadResolution(10);
  delay(3000);
  analogWrite(sinalU,511);
  analogWrite(groundWrite,511);
  Serial.println("start"); //
  startTimer(TC1, 0, TC3_IRQn, freq); //iniciar timer para amostragem
}

void loop() {
}
```

```
// =+=+=+=+=+=+=+  Parametro da funcao de diferencas =+=+=+=+=+=+ //

void parametros(){
  for (int i=0;i<ordNum;i++){
    parX[i]=num[ordNum-i-1]; // vetor com parametros que multiplicam X
    x[i]=0; // criacao do buffer X
  }
  for (int i=0;i<ordDen-1;i++){
    parU[i]=-den[ordDen-i-1]; // vetor com parametros que multiplicam U
    u[i]=0; // criacao do buffer U
  }
}

// =+=+=+=+=+=+=+  Execucao do sistema =+=+=+=+=+=+ //
void executar(){

   float xAux=0; // resetar buffer auxiliar
   float uAux=0; // resetar buffer auxiliar

   int refP = 100; //analogRead(referencia); // leitura do pino A0
   int sinalYR = analogRead(sinalY); // leitura da saída
   yP= ((1.4927*(sinalYR) -248.4289) -511);
//   Serial.print(yP);
   eP=refP-yP; // sinal de erro
//   Serial.print("\t");
//   Serial.println(eP);

   //calcular tempo desse loop para saber qual a freq. maxima
   for (int i=0; i<ordNum-1;i++){
     x[i]=x[i+1]; // atualizacao dos valores passados de x no buffer
     xAux+=+x[i]*parX[i]; // contribuicao dos valores passados da entrada para a saída
   }
   x[ordNum-1]=eP; // atualizacao do valor atual da entrada do controlador
   xAux+=x[ordNum-1]*parX[ordNum-1]; //contribuicao do valor presente da entrada

   for (int i=0;i<ordDen-1;i++){
     u[i]=u[i+1]; // atualizacao dos valores passados de u no buffer
     uAux+=u[i]*parU[i]; // contribuicao dos valores passados de saída na saída atual
   }
   u[ordDen-1]=xAux+uAux; // saída atual = contribuicao causada pela entrada(atual e
passada) + contribuicao causada pelos valores passados de saída
   float uW=(u[ordDen-1])+511;
   analogWrite(sinalU,uW);

   Serial.print(refP); // envia o valor de x[n] para a serial
   Serial.print("\t ");  // espaco
   Serial.print(u[ordDen-1],3); // enviar valor de u[n] para a serial
   Serial.print("\t ");
   Serial.print(uW);
```

```
   Serial.print("\t");  // espaco
   Serial.print(eP); // enviar valor do erro
   Serial.print("\t ");  // espaco
   Serial.println(yP);


}

// =+=+=+=+=+=+  Interrupcao =+=+=+=+=+=+ //

void TC3_Handler(){ // a cada Ts segundos, essa funcao é chamada

  TC_GetStatus(TC1, 0); // aceitar a interrupcao -> parametros usados en startTimer
=>(Tc1,0)
  executar();  // rodar simulacao
  ind++; // atualizar indice da simulacao
}

// =+=+=+=+=+=+  Timer para a interrupcao =+=+=+=+=+=+ //

void startTimer(Tc *tc, uint32_t channel, IRQn_Type irq, uint32_t frequency) {
   pmc_set_writeprotect(false);
  pmc_enable_periph_clk((uint32_t)irq);
  TC_Configure(tc, channel, TC_CMR_WAVE | TC_CMR_WAVSEL_UP_RC |
TC_CMR_TCCLKS_TIMER_CLOCK4);
  uint32_t rc = VARIANT_MCK/128/frequency; //128 because we selected
TIMER_CLOCK4 above
  TC_SetRA(tc, channel, rc/2); //50% high, 50% low
  TC_SetRC(tc, channel, rc);
  TC_Start(tc, channel);
  tc->TC_CHANNEL[channel].TC_IER=TC_IER_CPCS;
  tc->TC_CHANNEL[channel].TC_IDR=~TC_IER_CPCS;
  NVIC_EnableIRQ(irq);
}
```

IFT.INO

```
// =+=+=+=+=+=+=+  Parametro de entrada =+=+=+=+=+=+ //

float ta=0.02; // periodo de amostragem
const int np=5*50 ; // tempo de cada ensaio *em [s]
int nIteracoes=10; // numero de iterações

// Parametros do controlador
float kp=0.0115; //ganho proporcional
float ki=0.0084; // ganho integral
float kd=0.0095; // ganho derivativo

// Funcao transferencia Desejada
float numTd[2]={0.0142,0.0126}; // parametros do numerador (multiplicam z^0, z^-1, z^-2...)
float denTd[3]={1, -1.671,0.6977}; // parametros do denomiador (multiplicam z^0, z^-1, z^-
2...)

// =+=+=+=+=+=+=+  Parametro do ensaio =+=+=+=+=+=+ //
float num[3]={kp+ki+kd,-kp-2*kd,kd}; // parâmetros "ro" do numerador  -> [z^0 z^-1 z^-2]

const int sinalY = A0; // sinal y(t)
const int sinalU = DAC0; // sinal u(t)
const int groundWrite = DAC1; // referencia

float den[3]={1,-1,0}; // parâmetros do denominador -> [z^0 z^-1 z^-2] (fixos para o
controlador PID)
float freq=1/ta; // frequencia de amostragem
float refP; // referencia
float r2P[np]; // referencia da segunda simulacao

// Parametros de entrada e saida dos sistemas
float eP; // sinal de erro
float e2P;
float yP; // sinal de saída do processo
float y2P;
float eTd; // sinal de erro Td

//Parametros de ensaio
int ind=1; // indice primeiro ensaio
int ind2=1; // indice segundo ensaio
int indIt=1;
boolean dataOk = false; // flag para verificar se a amostra já foi adquirida

// Ordem
```

```
const int ordNum=(sizeof(num)/sizeof(float)); // ordem do polinomio do numerador do
controlador
const int ordDen=(sizeof(den)/sizeof(float)); // ordem do polinomio do denominador do
controlador
const int ordNumTd=(sizeof(numTd)/sizeof(float)); // ordem do polinomio do numerador do
controlador
const int ordDenTd=(sizeof(denTd)/sizeof(float)); // ordem do polinomio do denominador do
controlador

// Buffers
float x[ordNum]; //inicializacao do buffer para X
float u[ordDen]; // inicializacao do buffer para U
float xTd[ordNumTd]; //inicializacao do buffer para XTd
float yTd[ordDenTd]; // inicializacao do buffer para YTd
float yTdBuff[np]; // memória
float x2[ordNum]; //inicializacao do buffer para X
float u2[ordDen]; // inicializacao do buffer para U
float x2Td[ordNumTd]; //inicializacao do buffer para XTd
float y2Td[ordDenTd]; // inicializacao do buffer para YTd
boolean first=true;

// Parametros
float parX[10]; //inicializacao dos parametros do controlador
float parU[10]; //inicializacao dos parametros do controlador
float parXTd[10]; //inicializacao dos parametros da FT desejada
float parYTd[10]; //inicializacao dos parametros da FT desejada

// Parametros IFT
float j=0;
float parFilter[3]={1/num[0], -num[1]/num[0], -num[2]/num[0]};
float xFilt1[3];
float xFilt2[3];
float xFilt3[3];
float yFilt1[3];
float yFilt2[3];
float yFilt3[3];
float grad1;
float grad2;
float grad3;
float erro[np]; // vetor de erro entre a saída e a FT desejada
float norma;
float gamma0=sqrt(sq(num[0])+sq(num[1])+sq(num[2]))/2;
float gammaI;
boolean itOk=false;
boolean nextIt=false;


// =+=+=+=+=+=+  Inicializacao =+=+=+=+=+=+ //
```

```
void setup() {
  Serial.begin(115200); // inicializar comunicacao
  analogWriteResolution(10);
  analogReadResolution(10);
  analogWrite(groundWrite,511); // offset no ground
  parametros(); // gerar parametros da funcao de diferencas
  delay(3000); // delay de 3s
  Serial.println("Executando"); // flag
  startTimer(TC1, 0, TC3_IRQn, freq); //iniciar timer para amostragem
}

void loop() {
  if (dataOk){ // verificar flag
    Executar();  // rodar ensaios
    dataOk=false; // esperar nova amostra
  }
  if (itOk){
    Gradiente(); // calcular gradiente
    itOk=false; // atualizar flag
    nextIt=true; // atualizar flag
  }
}

// =+=+=+=+=+=+=+  Parametro da funcao de diferencas =+=+=+=+=+=+ //

void parametros(){
// Inicializacao dos parametros
//Controlador
  for (int i=0;i<ordNum;i++){
    parX[i]=num[ordNum-i-1]; // vetor com parametros que multiplicam X
    x[i]=0; // criacao do buffer X
    x2[i]=0;
  }
  for (int i=0;i<ordDen-1;i++){
    parU[i]=-den[ordDen-i-1]; // vetor com parametros que multiplicam U
    u[i]=0; // criacao do buffer U
    u2[i]=0;
  }
  for (int i=0;i<ordNumTd;i++){
      parXTd[i]=numTd[ordNumTd-i-1]; // vetor com parametros que multiplicam X
      xTd[i]=0; // criacao do buffer X
      x2Td[i]=0; // criacao do buffer X
  }
  for (int i=0;i<ordDenTd-1;i++){
    parYTd[i]=-denTd[ordDenTd-i-1]; // vetor com parametros que multiplicam Y
    yTd[i]=0; // criacao do buffer Y
    y2Td[i]=0; // criacao do buffer Y
  }
  xFilt1={0,0,0};
```

```
  xFilt2={0,0,0};
  xFilt3={0,0,0};
  yFilt1={0,0,0};
  yFilt2={0,0,0};
  yFilt3={0,0,0};
  grad1=0;
  grad2=0;
  grad3=0;
  Serial.println("OK");
}



// =+=+=+=+=+=+=+  Execucao do sistema =+=+=+=+=+=+ //


void Ensaio1(){
// Ensaio 1
   unsigned long timeStart1=micros(); //inicializacao do timer para cronometar o tempo de
execucao de cada loop
   float xAux=0; // resetar buffer auxiliar
   float uAux=0; // resetar buffer auxiliar

   refP = 100; //analogRead(referencia); // leitura do pino A0
   int sinalYR = analogRead(sinalY); // leitura da saída
   yP= ((1.4927*(sinalYR) -248.4289) -511);
   eP=refP-yP; // sinal de erro

   for (int i=0; i<ordNum-1;i++){
     x[i]=x[i+1]; // atualizacao dos valores passados de x no buffer
     xAux+=+x[i]*parX[i]; // contribuicao dos valores passados da entrada para a saída
   }
   x[ordNum-1]=eP; // atualizacao do valor atual da entrada do controlador
   xAux+=x[ordNum-1]*parX[ordNum-1]; //contribuicao do valor presente da entrada

   for (int i=0;i<ordDen-1;i++){
     u[i]=u[i+1]; // atualizacao dos valores passados de u no buffer
     uAux+=u[i]*parU[i]; // contribuicao dos valores passados de saída na saída atual
   }
   u[ordDen-1]=xAux+uAux; // saída atual = contribuicao causada pela entrada(atual e
passada) + contribuicao causada pelos valores passados de saída
   int uW=(u[ordDen-1])+511;

   // controle de saturação
   if (uW>=0 && uW<1023){
     analogWrite(sinalU,uW);
   }
   else {
     if (uW<0){
```

```
      uW=0;
      analogWrite(sinalU,uW);
    }
    if (uW>1023){
      uW=1023;
      analogWrite(sinalU,uW);
    }
  }

 // simulacao do sinal desejado
  if (first){
    float xTdAux=0; // resetar buffer auxiliar
    float yTdAux=0; // resetar buffer auxiliar

    for (int i=0; i<ordNumTd-1;i++){
      xTd[i]=xTd[i+1]; // atualizacao dos valores passados de x no buffer
      xTdAux+=+xTd[i]*parXTd[i]; // contribuicao dos valores passados da entrada para a
saída
    }
    xTd[ordNumTd-1]=refP; // atualizacao do valor atual da entrada do sistema
    xTdAux+=xTd[ordNumTd-1]*parXTd[ordNumTd-1]; //contribuicao do valor presente da
entrada

    for (int i=0;i<ordDenTd-1;i++){
      yTd[i]=yTd[i+1]; // atualizacao dos valores passados de y no buffer
      yTdAux+=yTd[i]*parYTd[i]; // contribuicao dos valores passados de saída na saída atual
    }

    yTd[ordDenTd-1]=xTdAux+yTdAux; // saída atual = contribuicao causada pela
entrada(atual e passada) + contribuicao causada pelos valores passados de saída
    yTdBuff[ind-1]=yTd[ordDenTd-1];
  }
  r2P[ind-1]=eP;
  erro[ind-1]=(yP-yTdBuff[ind-1]);

  j+=pow(erro[ind-1],2);
   unsigned long timeEnd1=micros(); // finaliza timer
  unsigned long deltaT1=timeEnd1-timeStart1; // calcula tempo de execucao

  if (indIt==1 || indIt==nIteracoes){
    Serial.print(ind); // mostrar o valor de x[n] na serial
    Serial.print("\t ");  // espaco
 //   Serial.print(refP); // mostrar o valor de x[n] na serial
 //   Serial.print("\t ");  // espaco
 //   Serial.print(u[ordDen-1],3); // mostrar valor de u[n] na serial
 //   Serial.print("\t "); // espaco
    Serial.print(uW); // mostrar o valor que está sendo escrito no circuito
    Serial.print("\t");  // espaco
 //   Serial.print(eP); // mostrar o valor do erro
```

```
  //   Serial.print("\t ");  // espaco
     Serial.print(yP); // mostrar o valor de y na serial
     Serial.print("\t ");  // espaco
  //   Serial.print(yTdBuff[ind-1]); // mostrar o valor do y desejado na serial
  //   Serial.print("\t ");  // espaco
     Serial.print(erro[ind-1]);
     Serial.print("\t ");
     Serial.println(deltaT1);
   }
}

void Ensaio2(){
   float x2Aux=0; // resetar buffer auxiliar
   float u2Aux=0; // resetar buffer auxiliar

   int sinalY2R = analogRead(sinalY); // leitura da saída
   float r2=r2P[ind2-1];
   y2P= ((1.4927*(sinalY2R) -248.4289) -511);
   e2P=r2-y2P; // sinal de erro

   for (int i=0; i<ordNum-1;i++){
     x2[i]=x2[i+1]; // atualizacao dos valores passados de x no buffer
     x2Aux+=+x2[i]*parX[i]; // contribuicao dos valores passados da entrada para a saída
   }
   x2[ordNum-1]=e2P; // atualizacao do valor atual da entrada do controlador
   x2Aux+=x2[ordNum-1]*parX[ordNum-1]; //contribuicao do valor presente da entrada

   for (int i=0;i<ordDen-1;i++){
     u2[i]=u2[i+1]; // atualizacao dos valores passados de u no buffer
     u2Aux+=u2[i]*parU[i]; // contribuicao dos valores passados de saída na saída atual
   }
   u2[ordDen-1]=x2Aux+u2Aux; // saída atual = contribuicao causada pela entrada(atual e
passada) + contribuicao causada pelos valores passados de saída

   int u2W=(u2[ordDen-1])+511;
   analogWrite(sinalU,u2W);

   // controle de saturacao
   if (u2W>=0 && u2W<1023){
     analogWrite(sinalU,u2W);
   }
   else {
     if (u2W<0){
       u2W=0;
       analogWrite(sinalU,u2W);
     }
     if (u2W>1023){
       u2W=1023;
       analogWrite(sinalU,u2W);
```

```
    }
  }

  // simulacao
  float x2TdAux=0; // resetar buffer auxiliar
  float y2TdAux=0; // resetar buffer auxiliar

  for (int i=0; i<ordNumTd-1;i++){
    x2Td[i]=x2Td[i+1]; // atualizacao dos valores passados de x no buffer
    x2TdAux+=+x2Td[i]*parXTd[i]; // contribuicao dos valores passados da entrada para a
saída
  }
  x2Td[ordNumTd-1]=r2; // atualizacao do valor atual da entrada do sistema
  x2TdAux+=x2Td[ordNumTd-1]*parXTd[ordNumTd-1]; //contribuicao do valor presente da
entrada

  for (int i=0;i<ordDenTd-1;i++){
    y2Td[i]=y2Td[i+1]; // atualizacao dos valores passados de y no buffer
    y2TdAux+=y2Td[i]*parYTd[i]; // contribuicao dos valores passados de saída na saída
atual
  }

  y2Td[ordDenTd-1]=x2TdAux+y2TdAux; // saída atual = contribuicao causada pela
entrada(atual e passada) + contribuicao causada pelos valores passados de saída

  // Simulacao filtro IFT
  xFilt1[0]=xFilt1[1];
  xFilt2[0]=xFilt2[1];
  xFilt3[0]=xFilt3[1];
  xFilt1[1]=xFilt1[2];
  xFilt2[1]=xFilt2[2];
  xFilt3[1]=xFilt3[2];
  xFilt1[2]=y2P/refP;
  xFilt2[2]=y2P/refP;
  xFilt3[2]=y2P/refP;

  yFilt1[0]=yFilt1[1];
  yFilt2[0]=yFilt2[1];
  yFilt3[0]=yFilt3[1];
  yFilt1[1]=yFilt1[2];
  yFilt2[1]=yFilt2[2];
  yFilt3[1]=yFilt3[2];

  yFilt1[2]=parFilter[0]*xFilt1[2]+parFilter[1]*yFilt1[1]+parFilter[2]*yFilt1[0];
  yFilt2[2]=parFilter[0]*xFilt2[1]+parFilter[1]*yFilt2[1]+parFilter[2]*yFilt2[0];
  yFilt3[2]=parFilter[0]*xFilt3[0]+parFilter[1]*yFilt3[1]+parFilter[2]*yFilt3[0];

  grad1+=erro[ind2-1]/refP*yFilt1[2];
  grad2+=erro[ind2-1]/refP*yFilt2[2];
```

```
   grad3+=erro[ind2-1]/refP*yFilt3[2];


//   Serial.print(ind2); // mostrar valor da referencia
//   Serial.print("\t ");  // espaco
//   Serial.print(r2P[ind2-1]); // mostrar valor da referencia
//   Serial.print("\t ");  // espaco
//   Serial.print(u2[ordDen-1],3); // mostrar valor de u[n] na serial
//   Serial.print("\t "); // espaco
//   Serial.print(u2W); // mostrar o valor que está sendo escrito no circuito
//   Serial.print("\t");  // espaco
//   Serial.print(e2P); // mostrar o valor do erro
//   Serial.print("\t ");  // espaco
//   Serial.println(y2P/refP); // mostrar o valor de y na serial
//   Serial.print("\t ");  // espaco
//   Serial.print(y2Td[ordDenTd-1]);
//   Serial.print("\t ");  // espaco
//   Serial.print(yFilt1[2]); // mostrar o valor de y na serial
//   Serial.print("\t ");  // espaco
//   Serial.print(yFilt2[2]); // mostrar o valor de y na serial
//   Serial.print("\t ");  // espaco
//   Serial.println(yFilt3[2]);

}

// =+=+=+=+=+=+  Interrupcao =+=+=+=+=+=+ //

// a cada Ts segundos, essa funcao é chamada
void TC3_Handler(){
  TC_GetStatus(TC1, 0); // aceitar a interrupcao -> parametros usados en startTimer
=>(Tc1,0)
  dataOk=true;
}

void Executar(){
  if (indIt<=nIteracoes){
    if (nextIt){ // verifica se é para rodar a próxima iteracao
      nextIt=false; // zera o buffer
      ind=1; // "zera" o indice
      ind2=1; // "zera" o indice
      atualizaControlador(); // atualiza controlador
      indIt++; // atualizar indice de iteracao
    }
    if (ind<=np&&indIt<=nIteracoes){ // verifica se o ensaio ja foi finalizado
      if (ind==1){ // zerar buffers e gerar parametros antes de realizar a execucao
        analogWrite(sinalU,511); // zerar a referencia (offset)
        delay (3000); // tempo para descarga do capacitor
      }
      Ensaio1();
```

```
      ind++; // atualizar indice
    }
    else {
      if (ind2<=np&&indIt<=nIteracoes){ // verifica se o ensaio ja foi finalizado
        if (ind2==1){ // zerar buffers e gerar parametros antes de realizar a segunda execucao
          first=false;
          j=j/np;
          Serial.println(j);
          j=0;
          analogWrite(sinalU,511); // zerar a referencia (offset)
          delay (8000); // tempo para descarga do capacitor
        }
        Ensaio2();
        ind2++; // atualizar indice
        if (ind2==np+1){
          itOk=true;
        }
      }
    }
  }
  else{
    Serial.println("Voila");
    while(true){
    }
  }
}

void Gradiente(){
  grad1=2*grad1/np;
  grad2=2*grad2/np;
  grad3=2*grad3/np;
  Serial.print(grad1); // mostrar o valor de y na serial
  Serial.print("\t ");  // espaco
  Serial.print(grad2); // mostrar o valor de y na serial
  Serial.print("\t ");  // espaco
  Serial.print(grad3);
  Serial.print("\t ");  // espaco
  norma=sqrt(sq(grad1)+sq(grad2)+sq(grad3));
  Serial.println(norma);
  gammaI=gamma0/(indIt);
  //Serial.println(gammaI);
  grad1=gammaI*grad1/(norma+1);
  grad2=gammaI*grad2/(norma+1);
  grad3=gammaI*grad3/(norma+1);



}
void atualizaControlador(){
```

```
//    Serial.print(num[0]); // mostrar o valor de y na serial
//    Serial.print("\t ");  // espaco
//    Serial.print(num[1]); // mostrar o valor de y na serial
//    Serial.print("\t ");  // espaco
//    Serial.println(num[2]);

    num[0]=num[0]-grad1;
    num[1]=num[1]-grad2;
    num[2]=num[2]-grad3;

    Serial.print(num[0],6); // mostrar o valor de y na serial
    Serial.print("\t ");  // espaco
    Serial.print(num[1],6); // mostrar o valor de y na serial
    Serial.print("\t ");  // espaco
    Serial.println(num[2],6);
    parametros();
}
// =+=+=+=+=+=+=+  Timer para a interrupcao =+=+=+=+=+=+=+ //
void startTimer(Tc *tc, uint32_t channel, IRQn_Type irq, uint32_t frequency) {
  pmc_set_writeprotect(false);
  pmc_enable_periph_clk((uint32_t)irq);
  TC_Configure(tc, channel, TC_CMR_WAVE | TC_CMR_WAVSEL_UP_RC |
TC_CMR_TCCLKS_TIMER_CLOCK4);
  uint32_t rc = VARIANT_MCK/128/frequency;
  TC_SetRA(tc, channel, rc/2);
  TC_SetRC(tc, channel, rc);
  TC_Start(tc, channel);
  tc->TC_CHANNEL[channel].TC_IER=TC_IER_CPCS;
  tc->TC_CHANNEL[channel].TC_IDR=~TC_IER_CPCS;
  NVIC_EnableIRQ(irq);
}
```