Universidade de Brasília

Depto. de Eng. Elétrica, Lab. de Controle Dinâmico (111929) - Turma A

Laboratório de Controle Dinâmico

Arthur R. Costa, 16/0024277
Caio E. Coelho de Oliveira, 16/0003679
Lucas E. Silva, 14/0062891

Geovany G. A. Borges

Brasília
28 de Maio de 2019

Resumo

Este experimento realiza a identificação e o controle através de duas abordagens (controlador PID e imposição de pólos no espaço de estados) de uma planta rotatória da empresa *QUARC*.

Objetivo

O experimento desenvolvido objetiva modelar, simular e implementar um servo-controlador de velocidade com dinâmica de atuador. O projeto foi desenvolvido em 3 fases a primeira de identificação e modelagem do kit Quanser a segunda fase do projeto a modelagem de um controlador PID com anti-windup utilizando a sisotool do Matlab e finalizando com terceira fase com modelagem de um controlador em espaço de estados .

1 Introdução

Introdução da teoria utilizada no experimento.

Este relatório está organizado de forma que todas as fórmulas e provas matemáticas necessárias encontram-se na Introdução; em Procedimentos, aplicam-se estas fórmulas e explicam-se os propósitos de suas aplicações na ordem utilizada; em Análise estão os resultados obtidos e comentários a respeito do que foi observado. As figuras e códigos mencionados ao longo do texto podem ser encontrados no apêndice.

1.1 Identificação do sistema

Para que tenhamos as constantes do sistema, será realizada a abordagem do seguinte algoritmo de identificação.

1.1.1 Sistema de 1^a ordem

Considerando-se um sistema cuja função de transferência é dada por:

$$G(s) = \frac{Y(s)}{R(s)} = \frac{b}{s+a}$$
$$Y(s)s + aY(s) = bR(s)$$

Transformando para o domínio do tempo:

$$\dot{y}(t) + ay(t) = br(t) \tag{1}$$

Como

$$\dot{y}(t) \approx \frac{y_K - y_{K-1}}{K - (K-T)} = \frac{y_K - y_{K-1}}{T}$$

Teremos, para o domínio discreto:

$$y_K = (1 - aT)y_{K-1} + br_{K-1} \tag{2}$$

Vamos, então, definir as seguintes variáveis:

$$\theta_1 = 1 - aT \tag{3}$$
$$\theta_2 = b \tag{4}$$

Onde

T: tempo de amostragem

a: pólo simples do sistema

b: constante de ganho

E montando várias equações de recorrência para diferentes tempos na forma matricial:

$$Y = \begin{bmatrix} y_2 \\ y_3 \\ ... \\ y_n \end{bmatrix} = \begin{bmatrix} y_1 & r_1 \\ y_2 & r_2 \\ ... & ... \\ y_{n-1} & r_{n-1} \end{bmatrix} * \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix} \tag{5}$$

$$Y = A\theta$$
$$A^tY = A^tA\theta$$
$$(A^tA)^{-1}A^tY = (A^tA)^{-1}A^tA\theta$$

$$\theta = (A^tA)^{-1}A^tY \tag{6}$$

Obtemos o algoritmo de identificação para o atuador do sistema.

Neste caso, a entrada será a onda quadrada aplicada ao sistema e a saída, o sinal que sai do atuador. Assim, será possível obter os parâmetros do atuador do sistema a partir das componentes do vetor θ encontrado utilizando a pseudo-inversa acima.

1.1.2 Identificação Zona morta

Em geral, é comum em motores uma não-linearidade do tipo zona-morta, isto é, uma faixa de valores de excitação para os quais não possuímos resposta. Isso ocorre devido a uma variedade de fatores, entre eles o atrito estático. Seja essa faixa compreendida entre um certo δ_- e um δ_+, torna-se necessário identificar esses dois valores, isto é, o bordo do intervalo no qual a planta é irresponsiva. Assim, torna-se necessário um procedimento que permita identificar os valores de excitação de entrada a partir dos quais o sistema em repouso passa a responder (o valor da saída, posição muda, a velocidade passa a ser não nula).
Podemos modelar uma não-linearidade do tipo zona morta por:

$$f(x) = \begin{cases} x - \delta_+, & \text{se } x > \delta_+ \\ 0, & \text{se } \delta_- < x < \delta_+ \\ x - \delta_-, & \text{se } x < \delta_- \end{cases} \tag{7}$$

onde o sistema com a não-linearidade é formado pela função f composta com a função de transferência da planta (desconsiderando a não-linearidade). Sabemos que a planta servo-linear permanecerá em repouso enquanto a entrada estiver entre δ_- e δ_+. Para identificar a zona morta, podemos excitar o sistema com uma entrada do tipo onda triangular, e então observar os valores nos semiciclos positivo e negativo a partir dos quais a entrada provoca uma resposta do sistema. Esses dois valores serão, respectivamente, o δ_+ e o δ_- identificados.

1.1.3 Sistema de 2^a ordem

A identificação para o sistema de 2^a ordem é bastante análoga ao do de 1^a, com exceção de que o sistema é modelado por

$$G(s) = \frac{Y(s)}{U(s)} = \frac{c}{s^2 + sa + b}$$

Assim, chegamos a

$$\begin{aligned} s^2Y(s) + saY(s) + bY(s) &= cU(s) \\ \ddot{y}(t) + a\dot{y}(t) + by(t) &= cu(t) \end{aligned}$$

Transformando para o domínio do tempo discreto, teremos:

$$\begin{aligned} y_k =& y_{K-1}(2 - aT) + \\ & y_{K-2}(aT - bT^2 - 1) + u_{K-2}(cT^2) \end{aligned} \tag{8}$$

$$\theta = \begin{bmatrix} 2 - aT \\ aT - bT^2 - 1 \\ cT^2 \end{bmatrix} \tag{9}$$

Semelhantemente, montamos várias equações de recorrência e com o sistema na forma matricial utilizamos a pseudo-inversa para determinar a matriz dos parâmetros θ

1.2 Controle no Espaço de Estados

Observando um sistema genérico de espaço de estados, temos

$$\dot{x}(t) = Ax(t) + Bu(t)$$
$$y(t) = Cx(t)$$

Transformando para o domínio de Laplace:

$$sX(s) = AX(s) + BU(s)$$
$$Y(s) = CX(s)$$

Após manipulação simples de substituição, pode-se chegar à expressão da função de transferência,

$$\frac{Y(s)}{U(s)} = C(sI - A)^{-1}B$$

Caso seja necessário obter uma realização em espaço de estados a partir de uma função de transferência de um sistema, podemos utilizar a forma canônica controlável a partir dos coeficientes da função de transferência. Por exemplo, seja a função de transferência genérica de ordem 3

$$T(s) = \frac{b_0 s^3 + b_1 s^2 + b_2 s + b_3}{s^3 + a_1 s^2 + a_2 s + a_3} \tag{10}$$

A realização na forma canônica controlável será dada por

$$\begin{pmatrix} \dot{x_1} \\ \dot{x_2} \\ \dot{x_3} \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -a_3 & -a_2 & -a_1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} u \tag{11}$$

$$y = \begin{pmatrix} b_3 - a_3 b_0 & b_2 - a_2 b_0 & b_1 - a_1 b_0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \tag{12}$$

Por último, caso seja necessário controlar um sistema qualquer no espaço de estados, através de uma lei de controle $u = -kx$, o vetor k pode ser determinado através das funções place e acker do matlab. Caso não se queira utilizar o matlab, pode-se usar ainda a fórmula de Bass-Gura caso o sistema seja de estado completamente controlável (aqui mostrada para um sistema de ordem 3):

$$k = \begin{bmatrix} \alpha_3 - a_3 & \alpha_2 - a_2 & \alpha_1 - a_1 \end{bmatrix} T^{-}1 \tag{13}$$

onde

$$T = M_c W \tag{14}$$

$$M_c = \begin{bmatrix} B & AB & A^2B \end{bmatrix} \tag{15}$$

$$W = \begin{bmatrix} a2 & a1 & 1 \\ a1 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \tag{16}$$

Os coeficientes α são os coeficientes do polinômio desejado, obtido a partir dos pólos que se quer impor.

2 Procedimentos

Dos procedimentos:

1 Identificação Realizamos a identificação da dinâmica do atuador, considerando-o um sistema de primeira ordem e analisando a entrada do sistema e a saída do amplificador (entrada para a planta). Depois, identificamos os parâmetros da não-linearidade do tipo zona morta. Munidos dos valores δ_- e δ_+ da não-linearidade, estimamos os valores fornecidos para a "parte linear"da planta e estimamos seus parâmetros como um sistema de segunda ordem. Em todos os procedimentos de identificação o time step foi adotado como $T = 0.002s$

1. Sistema 1^a ordem
   Fornecemos como entrada para o sistema uma onda quadrada, como pode ser visto na figura 1. O procedimento discutido na seção anterior para identificação do sistema de primeira ordem foi realizado automaticamente através do script IDfirstorder.m, disponível no apêndice. Seja um sistema da forma $\frac{b}{s+a}$ Os resultados obtidos podem ser vistos na tabela a seguir:

Tabela 1: Tabela dos parâmetros encontrados na identificação de primeira ordem

a:	20
b:	20

2. Zona morta
   Excitamos o sistema com uma onda triangular e observamos os valores nos semiciclos positivo e negativo a partir dos quais o sistema passou a responder, a excitação (onda triangular) e a resposta do sistema podem ser vistas no apêndice na figura 2.
   Os valores identificados para δ_- e δ_+ foram, respectivamente, -0.75 e 0.968.

   Sistema 2^a ordem
   Ainda analisando a resposta do sistema a uma entrada do tipo onda quadrada, utilizamos o script IDsecondorder.m (disponível no apêndice no final do documento para consulta) para realizar a análise discutida na introdução no procedimento de identificação dos parâmetros da planta combinada com o filtro passa-baixas na saída do sistema. Seja um sistema na forma $\frac{c}{s^2+s*a+b}$. Os resultados da identificação estão listados na tabela abaixo:

Tabela 2: Tabela dos parâmetros encontrados na identificação de segunda ordem

a:	652.7447
b:	7555.5563
c:	181707.3116

2 Controle com PID e *anti-windup*
O controlador PID em questão foi projetado utilizando-se o sisotool para obter um controlador PID padrão e adicionando uma realimentação anti-windup para evitar o windup do integrador quando há saturação. Isso é feito simplesmente adicionando um ganho Kw que multiplica a diferença entre a saída antes e depois do bloco de saturação, cujo resultado é somado ao valor $Ki * Erro$ e passado para o integrador. Além disso, como o sistema possui uma não-linearidade do tipo zona morta, projetamos uma função cuja imagem é tal que se a entrada não é nula ela sempre está fora do intervalo da zona morta e corresponde a excitação necessária caso não houvesse a zona morta (de modo que se colocarmos essa nova função em cascata com a planta, ela ajustará a saída do controlador de modo que o efeito sobre a planta seja o desejado) para que a planta se comporte como um sistema puramente linear. A função desejada satisfaz a seguinte expressão:

$$h(x) = \begin{cases} x + \delta_+, & \text{se } x > 0 \\ 0, & \text{se x} = 0 \\ x + \delta_-, & \text{se } x < 0 \end{cases} \tag{17}$$

Assim, pode-se facilmente observar que a relação entre a a variável de entrada e a saída depende do sinal da entrada, como se houvesse 3 imagens distintas. Além disso, o núcleo da função possui apenas o elemento nulo (isto é, a saída é nula somente se a entrada também for nula). Assim, podemos considerar que para $x = 0$ a função corresponde à identidade, e para $x > 0$ e $x < 0$ corresponde à translações de δ_+ e δ_-, respectivamente. Considerando a função sinal definida como a seguir:

$$sgn(x) = \begin{cases} 1, & \text{se } x > 0 \\ 0, & \text{se } x = 0 \\ -1, & \text{se } x < 0 \end{cases} \tag{18}$$

pode-se utilizar a função sinal para compor os 3 casos, notando que: se somarmos a função identidade com algum produto da função sinal, sempre que x=0 o resultado será necessariamente nulo, e quando a função sinal for diferente de zero, a função corresponderá a uma translação. Seja $I(x) = x$ a função identidade, definindo $h(x) = I(x) + sgn(x) * g(x)$, e assumindo $g(x)$ linear ou linear por partes em x, satisfazemos a propriedade descrita acima. Ainda, como sabemos que $h(x)$ deve satisfazer as relações dadas em (17) e seja $sgn(x)$ definida como informado, obtemos para g(x):

$$g(x) = \begin{cases} \delta_+, & \text{se } x > 0 \\ -\delta_-, & \text{se } x < 0 \end{cases} \tag{19}$$

Existem muitas formas de se encontrar uma g(x) que satisfaça as condições acima, neste trabalho adotaremos o seguinte procedimento. Adotemos a seguinte expressão para g(x):

$$g(x) = g_1(sgn(x)) + g_2(sgn(x)) \tag{20}$$

onde g_1 e g_2 são lineares em relação à $sgn(x)$ e estão definidas para o domínio $D = \{-1, 1\}$, não sendo necessário as definir para $sgn(x) = 0$, e satisfazem as seguintes propriedades:

$$g_1(n) = \begin{cases} \delta_+, & \text{se } n = 1 \\ 0, & \text{se } n = -1 \end{cases} \tag{21}$$

$$g_2(n) = \begin{cases} 0, & \text{se } n = 1 \\ -\delta_-, & \text{se } n = -1 \end{cases} \tag{22}$$

Ora, se escolhermos g_1 e g_2 como as retas com raízes em -1 e 1 e coeficientes angulares $\frac{\delta_+}{2}$ e $-\frac{\delta_-}{2}$, respectivamente, satisfaremos as propriedades acima, isto é, faremos:

$$\begin{cases} g_1(n) = (1+n)\frac{\delta_+}{2} \\ g_2(n) = -(1-n)\frac{\delta_-}{2} \end{cases} \tag{23}$$

de modo que

$$g(x) = (1 + sgn(x))\frac{\delta_+}{2} - (1 - sgn(x))\frac{\delta_-}{2} \tag{24}$$

Assim, finalmente obtemos a função h(x) que atua como inversa da não-linearidade fora do intervalo de zona morta

$$h(x) = x + sgn(x)[(1 + sgn(x))\frac{\delta_+}{2} - (1 - sgn(x))\frac{\delta_-}{2}] \tag{25}$$

Essa função foi implementada no simulink utilizando os blocos disponíveis na ferramenta, conforme mostrado na figura 4 no apêndice.

1. Determinação dos parâmetros do controlador PID (kp, ki e kd):
   Na determinação dos parâmetros do controlador, foi montado um modelo do sistema com base nos valores encontrados na etapa de identificação junto com um controlador PID, como pode ser visto na figura 3 no apêndice. Munidos das funções de transferência do atuador e do sistema rotativo obtidos na identificação (e desconsiderando a não-linearidade do tipo zona morta), obtemos a função de transferência de malha aberta da planta e rodamos o sisotool (passando a função de transferência de malha aberta do sistema para a ferramenta). Ajustando o tempo de acomodação para 0.3s e o critério de resposta transiente do sisotool para 0.8, obtivemos um controlador PID com os seguintes valores para as constantes:
   - (i) K_p: 0.0311
   - (ii) K_i: 0.2992
   - (iii) K_d: 8.0898e-04
2. Determinação do K_w
   Ajustando os valores dos ganhos na simulação de acordo com os valores obtidos pelo sisotool, restava ajustar o ganho Kw para que o sistema se comportasse adequadamente na presença da saturação (ajustada entre uma faixa de valores limítrofes de -10 e 10. O ganho kw foi incrementado gradualmente (começando do zero) até o sistema responder adequadamente na simulação a uma entrada do tipo onda quadrada de período 2s (frequência 0.5Hz) e amplitude 45. O valor de Kw obtido através desse procedimento foi
   $$K_w = 0.6 \tag{26}$$
3. Implementação prática
   Uma vez comprovado o funcionamento do controlador através da simulação, o mesmo foi implementado em um diagrama do simulink junto com os blocos de hardware in the loop (HIL) da

Quarc para controlar o sistema servo rotativo real, como pode ser visto na figura 5 do apêndice. Os resultados podem ser vistos na imagem 6 no apêndice. É possível observar que o sistema apresenta overshoot até estabilizar, no entanto o resultado foi apresentado para o professor da disciplina que confirmou que o sobrevalor apresentado pelo sistema era aceitável, estando dentro dos requisitos. Os parâmetros utilizados no controlador podem ser vistos na tabela a seguir:

Tabela 3: Tabela dos parâmetros utilizados no controlador PID com anti-windup

K_p:	0.0311
K_i:	0.2992
K_d:	8.0898e-04
K_w:	0.6

3 Controle em espaço de estados

De posse do modelo do sistema obtido a partir da identificação, procedemos à elaboração de um controlador baseado em espaço de estados. Para tal, obtivemos uma realização do sistema a partir da função de transferência, implementamos um observador de estados a partir dessa realização e implementamos uma lei de controle com acompanhamento de referência a partir do estado estimado. Tudo isso foi realizado através do uso do script ssGera.m, que pode ser encontrado no apêndice de scripts do matlab ao final do documento.

1. Obtenção de uma realização do sistema em espaço de estados a partir da função de transferência

   Como foi visto na introdução, seja um sistema cuja função de transferência (relação entrada-saída) é conhecida, existem diversas realizações em espaço de estados relacionadas por mudanças de variáveis que satisfazem essa mesma função de transferência, em particular as formas canônicas controlável e observável. No código ssGera.m, a realizoação é obtida usando a função ssdata do matlab para obter as matrizes A, B, C, D da representação genérica no espaço de estados

$$\begin{cases} \dot{x} = Ax + Bu \\ y = Cx + Du \end{cases} \tag{27}$$

2. Implementação do observador

O observador de estados pode ser descrito por:

$$\begin{cases} \dot{\hat{x}} = A\hat{x} + Bu + L(y - \hat{y}) \\ \hat{y} = C\hat{x} + Du \end{cases} \tag{28}$$

onde no nosso caso particular $D = 0$. Definindo o erro de observação como

$$\widetilde{x} = x - \hat{x} \tag{29}$$

o que implica

$$\dot{\widetilde{x}} = (A - LC)\widetilde{x} \tag{30}$$

ou seja, o erro de observação é governado por um sistema dinâmico autônomo cuja dinâmica depende dos autovalores da matriz $A - LC$, i.e., as raízes da equação

$$det(sI - A + LC) = 0 \tag{31}$$

Caso sejam conhecidos os pólos desejados para a dinâmica do erro de observação (dinâmica do observador), pode-se usar um dos métodos discutidos na introdução para determinar a matriz L, haja visto que como o determinante é invariante à transposição, os autovalores de $A-LC$ são também os autovalores de $A^t-C^tL^t$. A determinação de L^t é igual a determinação de um vetor k para imposição de pólos a um sistema genérico no espaço de estados. No código ssGera.m, L é determinado na linha 56 através do comando L = (a c k e r (A' , C' , p o l o s O b s)) ';

3. Controle e acompanhamento de referência
Como desejamos que o erro de acompanhamento em regime estacionário do sistema para uma entrada do tipo degrau seja nulo, precisamos garantir que o sistema seja no mínimo de tipo 1. Para isso, usaremos uma lei de controle da forma

$$u = -kx(t) - k_i \int_0^t (r - Cx(t))d\tau \tag{32}$$

portanto para um sistema genérico no espaço de estados com equação dinâmica

$$\dot{x} = Ax + Bu \tag{33}$$

substituindo a lei de controle fornecida acima obtemos

$$\dot{x} = (A - Bk)x - Bk_i \int_0^t (r - Cx)d\tau \tag{34}$$

Matematicamente, a abordagem acima é equivalente a considerarmos um sistema expandido com um estado adicional x_i dado por

$$x_i = \int_0^t (r - Cx)d\tau \tag{35}$$

e uma lei de controle

$$u = -\begin{bmatrix} k & k_i \end{bmatrix} \begin{bmatrix} x \\ x_i \end{bmatrix} \tag{36}$$

onde chamaremos

$$k_e = \begin{bmatrix} k & k_i \end{bmatrix} \tag{37}$$

Na forma matricial, a equação dinâmica do sistema expandido é dada por

$$\begin{bmatrix} \dot{x} \\ \dot{x_i} \end{bmatrix} = \begin{bmatrix} A & 0 \\ -C & 0 \end{bmatrix} \begin{bmatrix} x \\ x_i \end{bmatrix} + \begin{bmatrix} B \\ 0 \end{bmatrix} u + \begin{bmatrix} 0 \\ 1 \end{bmatrix} r \tag{38}$$

chamando

$$x_e = \begin{bmatrix} \dot{x} \\ \dot{x_i} \end{bmatrix} \tag{39}$$

$$A_e = \begin{bmatrix} A & 0 \\ -C & 0 \end{bmatrix} \tag{40}$$

$$B_e = \begin{bmatrix} B \\ 0 \end{bmatrix} \tag{41}$$

e substituindo a lei de controle, obtemos a equação dinâmica do sistema expandido em malha fechada

$$\dot{x_e} = (A_e - B_e k_e)x_e + \begin{bmatrix} 0 \\ 1 \end{bmatrix} r \tag{42}$$

de modo que os pólos do sistema em malha fechada são os autovalores da matriz $A_e - B_e k_e$. Para a imposição de pólos do sistema, pode-se determinar k_e a partir de um dos métodos discutidos na introdução, e consequentemente determinar k e k_i.

4. Implementação prática

   Todos os procedimentos descritos anteriormente para obter as matrizes A, B, C da realização, k, k_i da realimentação e L do observador são realizados automaticamente pelo script ssGera.m disponível para consulta no apêndice, sendo necessário apenas escolher os pólos do observador e os pólos do sistema. Além disso, na nossa implementação, optamos por desconsiderar um dos pólos obtidos na identificação (ele estava muito distante da origem e era rápido demais para ser implementado no hardware disponível) e obtivemos uma realização de segunda ordem para o sistema. Os pólos do sistema e do observador foram escolhidos conforme a tabela abaixo:

   Nota: o sistema possui um pólo a mais pois considerou-se o sistema expandido com um estado para acomodar o erro.

Tabela 4: Tabela dos pólos do sistema e do observador escolhidos

Pólos do observador	-20	-21	-
Pólos do sistema	-5	-6	-7

Uma vez obtidos os valores necessários a partir da execução do script, montou-se um diagrama no simulink implementando o controlador para a planta a partir dos valores publicados pelo script no workspace do matlab. O diagrama pode ser visto na figura 7 do apêndice. Executando o sistema com uma onda quadrada como entrada, observou-se que o sistema acompanhou muito bem a referência, como pode ser visto na figura 8 do apêndice.

3 Análise dos resultados

3.1 Identificação

O sistema foi testado com duas entradas distintas: uma onda quadrada de 0.5 Hz e amplitude 3V; e uma onda triangular. A primeira foi aplicada com o intuito de identificar o sistema tanto de primeira ordem quanto de segunda. A entrada triangular foi aplicada para observar a tensão necessária para o sistema começar a responder, ou seja, a zona morta.

Para o caso da onda triangular, a saída do atuador gera um atraso no sinal, sem qualquer outra alteração observada. Também não é difícil notar observando o gráfico da figura 2 que a não-linearidade do tipo zona morta é assimétrica, isto é, as magnitudes dos valores positivo e negativo que provocam resposta do sistema são diferentes, o que também pode ser confirmado observando os resultados da identificação da zona morta na seção anterior.

3.2 Controladores

Ambos o controlador PID e o controlador baseado em realimentação de estados funcionaram como o esperado e promoveram estabilização e seguimento de referência. No entanto, o desempenho do controlador no espaço de estados foi mais satisfatório, mesmo este último não possuindo um mecanismo de anti-windup para lidar com a saturação como o compensador PID. Possivelmente, os sobrevalores consideráveis observados no comportamento do sistema compensado com o controlador PID, como se pode observar na figura 6, foram ocasionados por uma decisão de projeto irrealista, com um tempo de acomodação muito pequeno, necessitando de um valor de k_w muito alto para conter os efeitos de windup na presença da saturação.

4 Conclusão

No decorrer do presente trabalho foram realizados a identificação e o controle através de duas alternativas (controle PID e realimentação de estados) da planta servo rotativo da Quanser disponível no laboratório. Em especial, notou-se a grande importância da etapa de identificação, pois parâmetros errôneos advindos da identificação podem comprometer todo o desenvolvimento posterior que se segue. Superadas as dificuldades relacionadas à identificação, observou-se um comportamento mais robusto do controlador baseado em espaço de estados, mesmo este não possuindo um mecanismo de anti-windup. Assim, confirmou-se a robustez da abordagem de controle baseado em observador-controlador de estados e suas vantagens em relação à abordagem clássica.

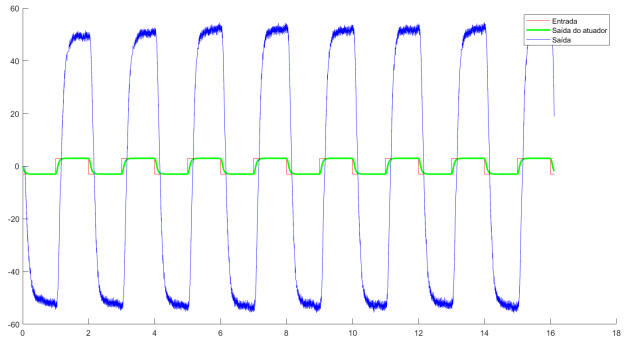
Referências

[1] Nise, N. S. *Control System Engineering*, 6^{th} ed. John Wiley & Sons, Inc., 2011.

[2] Ogata, K. *Modern Control Engineering*, 5^{th} ed. Pearson, 2010.

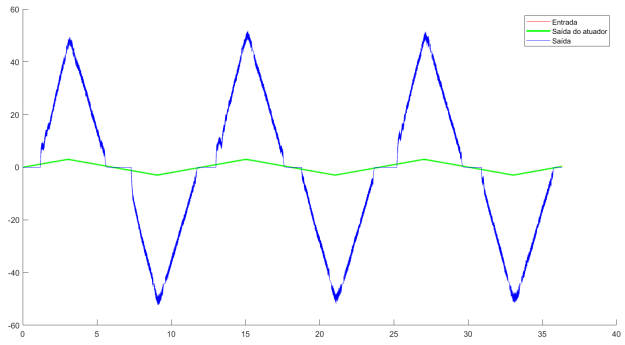
5 Apêndice de figuras referenciadas ao longo das seções

Figura 1: Dados obtidos com entrada Quadrada



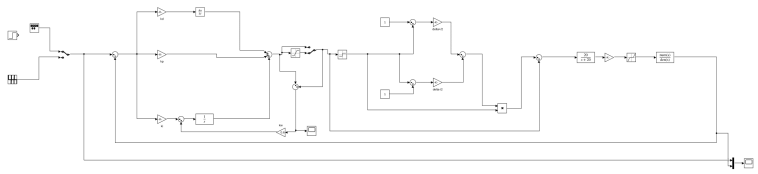
Fonte: arquivo pessoal.

Figura 2: Dados obtidos com entrada Triangular



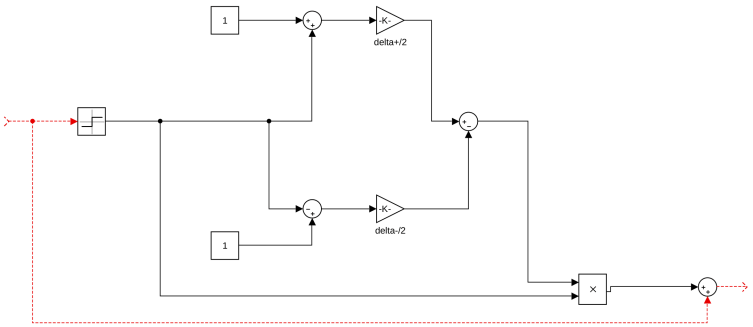
Fonte: arquivo pessoal.

Figura 3: Simulação do sistema com controlador PID no simulink



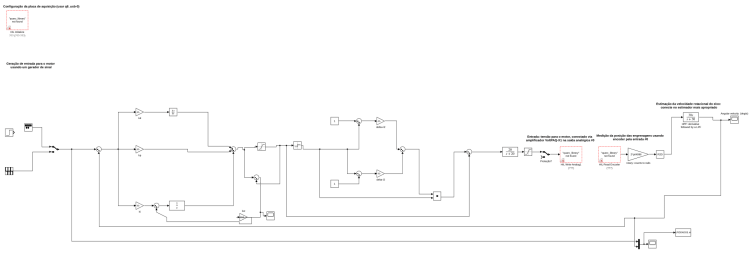
Fonte: arquivo pessoal.

Figura 4: Implementação da função para compensar a não-linearidade no simulink



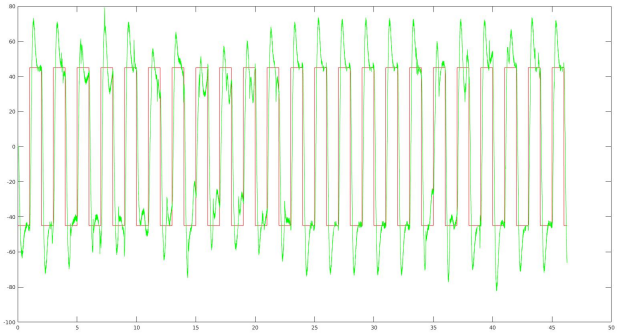
Fonte: arquivo pessoal.

Figura 5: Diagrama de controle do sistema servo rotativo com controlador PID



Fonte: arquivo pessoal.

Figura 6: Gráfico comparativo entre a entrada e a saída do sistema rotativo com o compensador PID

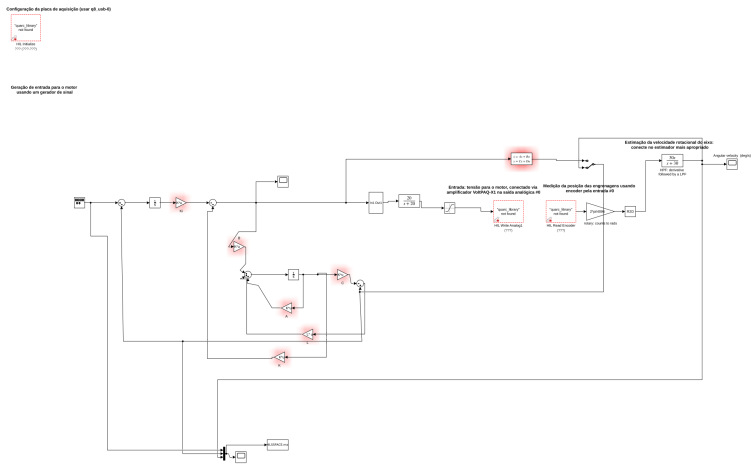


Fonte: arquivo pessoal.

6 Apêndice de scripts do Matlab mencionados ao longo do texto

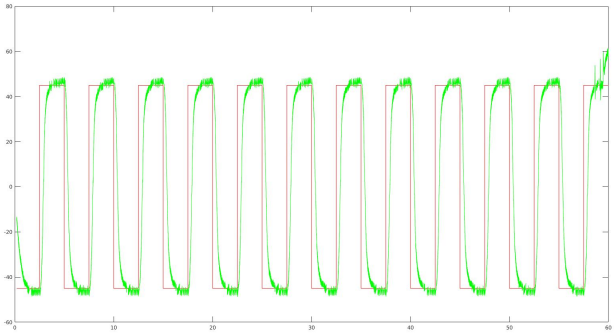
IDfirstorder.m

Figura 7: Diagrama do sistema com controlador baseado em espaço de estados



Fonte: arquivo pessoal.

Figura 8: Gráfico comparativo entre a entrada e a saída do sistema rotativo com o compensador baseado em espaço de estados



Fonte: arquivo pessoal.

```
%% Calculate the parameters of the first order tranfer
    function of the actuator
% author: Caio E C Oliveira
% date:   June 2, 2019.
% Uses the function plotDadosQT.m for showing plots of
```

```
    data and also for loading it into the script.
5
6  %clearvars;
7  close all;
8  clc;
9
10 file = 'q';
11 if (strcmp(file, 'Q') || strcmp(file, 'q'))
12     file = 'dadosQ';
13 elseif(strcmp(file, 'T') || strcmp(file, 't'))
14     file = 'dadosT';
15 end
16 dados = plotDadosQT(file); close all;
17
18 datasize = length(dados);
19 time_step= dados(1,2) - dados(1,1);              %
    discrete time step
20 disp(['Data size: ', num2str(size(dados)), ';', ' Time
    Step: ', num2str(time_step), ';']);
21
22 %% Defining Matrices
23 % sys = tf([20],[1 20]);
24 % dados(4, :) = lsim(sys,dados(5, :), dados(1, :));
25
26 if(strcmp(file, 'dadosQ') || strcmp(file, 'dadosQ.mat'
    ))
27     entrada = (dados(4, 1:datasize-1))';      % system
        input from r1 to rn-1 as a column vector
28     saida   = (dados(3, 1:datasize-1))';      %
       actuator output from y1 to yn-1 as a column
       vector
29     Y       = (dados(3, 2:datasize))';        % output
        Y as a column vector from y2 to yn
30 elseif (strcmp(file, 'dadosT') || strcmp(file, 'dadosT
    .mat'))
31     entrada = (dados(4, 1:datasize-1))';      % input
       R from r1 to rn-1 as a column vector
32     saida   = (dados(3, 1:datasize-1))';      %
       actuator output Y from y1 to yn-1 as a column
       vector
33     Y       = (dados(3, 2:datasize))';        % output
        Y as a column vector from y2 to yn
34 end
35
```

```
36 A       = [saida, entrada];                    % column
       matrix
37 disp(['A matrix size: ', num2str(size(A)), ' First
      column: actuator output; Second column: system
      input']);
38 disp(['Y matrix size: ', num2str(size(Y))]);
39
40 %% Theta calculation
41 theta=inv(A'*A)*A'*Y;
42
43 a = (1 - theta(1))/time_step;
44 b = theta(2)/time_step;
45
46 disp ('System type: b/(s + a)');
47 disp (['a value: ', num2str(a), ';', ' b value: ',
      num2str(b), ';']);
48
49 %% Simulacao do sistema
50 sys = tf([b],[1 a]);
51 saida_atuador = dados(3, :);
52 entrada_sistema = dados(4, :);
53
54 t = dados(1,:);
55 ye = lsim(sys, entrada_sistema, t);
56
57 figure;
58 hold on;
59 plot(t, saida_atuador, '--k', 'LineWidth', 2);
60 plot(t, ye, '-g', 'LineWidth', 1);
61 legend ('Sa da do Atuador', 'Sa da estimada');
```

IDsecondorder.m

```
1 %% Calculate the parameters of the system dynamics
      plus filter,
2 % Needs to be run
3 % after dead_zone.m because it uses delta_plus and
      delta_minus values
4 % author: Arthur R Costa, Caio E C Oliveira & Lucas E
      Silva
5 % date:   June 2, 2019.
6 % Perform a system identification.
7
8 %clearvars;
```

```
close all;
clc;

file = 'q';
dados = plotDadosQT(file); close all;

datasize = length(dados);
T= dados(1,2) - dados(1,1);                % discrete
    time step
disp(['Data size: ', num2str(size(dados)), ';', ' Time
     Step: ', num2str(T), ';']);

%% Matrices definition
startpoint = 500;                          % Ignoring
    noisy initial signal
delta = [0.968, -0.75];                    % dead zone
    threshold
input =  (dados(3, startpoint:end))';      % actuator
    output
output = (dados(2, startpoint:end))';      % system
    output

% Dead zone output estimation.
ddzone = zeros(length(input), 1);
i=1;
while i <= length(input)
     if ((input(i) < delta(2)))
          ddzone(i) = input(i) - delta(2);
     elseif((input(i) > delta(1)))
          ddzone(i) = input(i) - delta(1);
     end
     i=i+1;
end

figure; hold on;
subplot(1, 2, 1);
zoom = startpoint:int16(length(dados)/4);
plot(dados(1, zoom), input(zoom), 'k', 'LineWidth', 2)
    ;
title('Dead Zone Input (zoom in)');
ylabel('Dead Zone Input (V)');
xlabel('time (s)');

subplot(1,2,2);
```

```
zoom = startpoint:int16(length(dados)/8);
plot(dados(1, zoom), ddzone(zoom), 'g', 'LineWidth',
    2);
title('Dead Zone Output (zoom in)');
ylabel('Dead Zone Output (V)');
xlabel('time (s)');

clear zoom i;

%% Calculate theta(1:3)
A = [output(2:length(output) - 1), output(1:length(
    output)-2)];
A = [A, ddzone(1:(length(ddzone)-2))];
Y = output(3:length(output));

theta = inv(A'*A)*A'*Y;

a = (2 - theta(1))/T;
b = (a*T - 1 - theta(2)) / T^2;
c = theta(3)/T^2;

disp ('System type: c/(s^2 + s*a + b)');
disp(['a = ', num2str(a), ';', 'b = ', num2str(b), ';'
    , 'c = ', num2str(c)])
close all;
%% Simulation
time  = dados(1, 1:length(ddzone));
g = tf([c], [1 a b]);
sys2order = lsim (g, ddzone, time);

figure;
plot(time, output, 'k', time, sys2order, 'g', '
    LineWidth', 2);
legend('Saida Medida', 'Resposta Simulada');
```

ssGera.m

```
format longG

clearvars; close all; clc; clear all;

% load IDmodelo.mat
% num = sys.Numerator;
% den = sys.Denominator;
```

```
8  % num = num{:};
9  % den = den{:};                                          %
       third order OLTF denominator
10 % root3order = roots(den);
11 % disp('todas as raizes: ');
12 % disp(root3order);
13 % root2order = [root3order(2), root3order(3)];
14 % disp('raizes aproximadas: ');
15 % disp(root2order);
16
17 % sys_motor = tf([181707.316],[1 652.7 7555.55]);
18 %           -640.911238076149
19 %                          -20
20 %           -11.7887619238505
21 sys_motor = tf([181707.316/640.911238076149],[1
       11.7887619238505]);
22 sys_atuador = tf([20],[1 20]);
23 sys = series(sys_motor,sys_atuador);
24 %den         = poly(root2order)*root3order(1);  %
       second order OLTF denominator
25 %% Define State Space Matrices of the system
26 [A, B, C, D] = ssdata(sys);       % ss matrices using
       open loop transfer function
27 disp('A:');
28 disp(A);
29 disp('B:');
30 disp(B);
31 disp('C:');
32 disp(C);
33 disp('D:');
34 disp(D);
35
36 OLpoles = eig(A);                          % p los do
       sistema
37 disp ('P los dos sistema de malha aberta: ');
38 disp (OLpoles);
39
40 disp('Assumindo que o sistema     control vel, iremos
       realocar os p los do processo aonde for
       conveniente.')
41 %% Define K so as to place poles of the process
42 polosProc = [-5, -6, -7]
43 Aext = [A, [0; 0]; -C, 0];
44 Bext = [B; 0];
```

```
45 Kext  = acker(Aext, Bext, polosProc); % realocar os
       p los para que a planta possa responder.
46 Kproc = [Kext(1), Kext(2)];
47 Ki    = Kext(3);
48 Ac = Aext - Bext*Kext;
49 disp('P los realocados do sistema. Matriz Ac = Aext -
       BextK')
50 disp(eig(Ac))
51
52 % Define poles of the observer
53 % They should be allocated at least 3x further from
       the origin than the
54 % furthest pole of the process.
55 polosObs = [-20, -21];
56 L = (acker(A', C', polosObs))';
57 disp('ki: ')
58 disp(Ki);
59 disp('k: ')
60 disp(Kproc)
61 disp('L:');
62 disp(L);
63 save('dadosSS.mat', 'Kext', 'Kproc', 'Ki', 'L', 'A', '
       B', 'C', 'D', 'Ac', 'Aext', 'Bext');
64
65 % Generate Simulink model
66 %openmodelsim();
```