



TRABALHO DE GRADUAÇÃO

PROJETO E IMPLEMENTAÇÃO EM CONTROLADOR INDUSTRIAL PARA POSICIONAMENTO DE RISERS COM VALIDAÇÃO EXPERIMENTAL

Por,
Ataias Pereira Reis
Emanuel Pereira Barroso Neto

Brasília, julho de 2016



**ENGENHARIA
MECATRÔNICA**
UNIVERSIDADE DE BRASÍLIA

UNIVERSIDADE DE BRASILIA
Faculdade de Tecnologia
Curso de Graduação em Engenharia de Controle e Automação

TRABALHO DE GRADUAÇÃO

PROJETO E IMPLEMENTAÇÃO EM CONTROLADOR INDUSTRIAL PARA POSICIONAMENTO DE RISERS COM VALIDAÇÃO EXPERIMENTAL

Por,
Ataias Pereira Reis
Emanuel Pereira Barroso Neto

*Relatório submetido como requisito parcial de obtenção
de grau de Engenheiro de Controle e Automação*

Banca Examinadora

Prof. Eugênio Libório Feitosa Fortaleza, _____
ENM/UnB
Orientador

Prof. Eduardo Stockler Tognetti, ENE/UnB _____
Co-orientador

Prof. Guilherme Caribé de Carvalho, _____
ENM/UnB

Prof. Geovany Araújo Borges, ENE/UnB _____

Brasília, julho de 2016

FICHA CATALOGRÁFICA

REIS, ATAIAS PEREIRA; NETO, EMANUEL PEREIRA BARROSO;
Projeto e Implementação em Controlador Industrial para Posicionamento de Risers com Validação Experimental ,
[Distrito Federal] 2016.
x, 47p., 297 mm (FT/UnB, Engenheiro, Controle e Automação, 2016). Trabalho de Graduação – Universidade de Brasília.Faculdade de Tecnologia.

1. Controle em Malha Fechada 2. Controlador Lógico Programável
3. Sistemas Offshore
I. Mecatrônica/FT/UnB

REFERÊNCIA BIBLIOGRÁFICA

REIS, A. P., NETO, E. P. B. (2015). Projeto e Implementação em Controlador Industrial para Posicionamento de Risers com Validação Experimental. Trabalho de Graduação em Engenharia de Controle e Automação, Publicação FT.TG-nº0XX, Faculdade de Tecnologia, Universidade de Brasília, Brasília, DF, 64p.

CESSÃO DE DIREITOS

AUTORES: Ataias Pereira Reis e Emanuel Pereira Barroso Neto

TÍTULO DO TRABALHO DE GRADUAÇÃO: Projeto e Implementação em Controlador Industrial para Posicionamento de Risers com Validação Experimental.

GRAU: Engenheiro

ANO: 2016

É concedida à Universidade de Brasília permissão para reproduzir cópias deste Trabalho de Graduação e para emprestar ou vender tais cópias somente para propósitos acadêmicos e científicos. O autor reserva outros direitos de publicação e nenhuma parte desse Trabalho de Graduação pode ser reproduzida sem autorização por escrito dos autores.

Ataias Pereira Reis

72926-048 Águas Lindas – GO – Brasil

Emanuel Pereira Barroso Neto

72130-450 Taguatinga – DF – Brasil

Dedicatórias

A Emanuel B., meu avô, fonte de inspiração eterna

Emanuel Pereira Barroso Neto

Dedico à minha família, que me apoiou ao longo desta jornada.

Ataias Pereira Reis

Agradecimentos

Eu agradeço a Deus, aos professores Eugênio Fortaleza e Eduardo Tognetti, orientadores deste trabalho, e a todos que realizaram trabalhos prévios que foram utilizados aqui. Agradeço também ao sólcito Rédyton Brenner que nos ajudou muito no início.

Ataias Pereira Reis

A meus orientadores, Professores Eugênio Fortaleza e Eduardo Tognetti, pela presteza, cordialidade e pelos ensinamentos que foram essenciais para o desenvolvimento do trabalho; Ao mestrando Rédyton Brenner, crucial para nos ensinar sobre o funcionamento da bancada; A meus amigos da Engenharia Mecatrônica, por mostrarem que além de aprender, ter amizades faz bem ao caráter; Às outras pessoas que conheci na vida acadêmica, por me proporcionarem a oportunidade da troca de conhecimento; À minha família, que me inspirou a nunca desistir sem lutar, por mais difíceis que fossem os obstáculos. À minha dupla, Ataias, pela paciência e ajuda, além da camaradagem e desejo de vencer, fatos de relevância fundamental para que fizéssemos este projeto.

Emanuel Pereira Barroso Neto

RESUMO

A extração de petróleo em águas profundas requer operações bastante complexas. Em especial, a operação de entrada reentrada ocorre quando um *riser* é conectado de sua plataforma a um poço de petróleo. O deslocamento do *riser* até o poço deve ser feito de forma rápida e precisa, mas, atualmente, o controle de posição é manual, o que pode ser ineficiente. O presente trabalho busca validar técnicas de controle por meio de experimentos em uma planta de laboratório representativa de um *riser* por meio de controle em malha aberta e fechada para operar o deslocamento do *riser* de forma automática, com resultados rápidos e precisos, além de oferecer resistência a perturbações causadas pelo oceano, para que a ponta do *riser* seja corretamente conectada ao poço.

Palavras Chave: *riser*, controle, deslocamento

ABSTRACT

Deep sea petroleum exploration requires very complex operations. Particularly, the re-entry operation occurs when a riser is connected from the platform to a wellhead. The riser's displacement must be done quickly and precisely but, presently, the position control is manual, which may be inefficient. The present paper seeks to validate control techniques with experiments in a *riser* representative lab plant using open and closed loop control to operate the riser's displacement automatically, with fast and precise results, in addition to offering resistance against disturbances caused by the ocean, so that riser's tip is correctly connected to the wellhead.

Keywords: *riser*, control, displacement

SUMÁRIO

1	Introdução	1
1.1	CONTEXTUALIZAÇÃO	1
1.2	OBJETIVOS DO PROJETO	1
2	Fundamentos	4
2.1	MODELAGEM	4
2.1.1	EQUAÇÕES GOVERNANTES	4
2.1.2	DISCRETIZAÇÃO	6
2.1.3	ESTRATÉGIA DE REDUÇÃO DA ORDEM DO MODELO	8
2.2	CONTROLE	10
2.2.1	TÉCNICAS SIMPLES DE CONTROLE	10
2.2.2	CONTROLE DISCRETO NO ESPAÇO DE ESTADOS	11
2.2.3	PREDITOR DE SMITH	13
2.2.4	FILTRO DE KALMAN	14
2.3	BANCADA	16
2.3.1	CONTROLADOR LÓGICO-PROGRAMÁVEL	16
2.3.2	SERCOS INTERFACE	18
2.3.3	LINE INTERFACE MODULE 2094-AL09	19
2.3.4	DRIVE KINETIX 6000 DA ALLEN BRADLEY	19
2.3.5	SERVOMOTOR	19
2.3.6	SENSORES INDUTIVOS	20
2.3.7	CÂMERA PRESENCEPLUS	21
2.4	REDES UTILIZADAS	22
2.4.1	DEVICENET	22
2.4.2	ETHERNET/IP	22
2.4.3	OPC - <i>OLE for Process Control</i>	22
2.5	PROGRAMAÇÃO DO CLP	23
2.5.1	VISÃO GERAL	23
2.5.2	LINGUAGEM <i>ladder</i>	24
2.5.3	TEXTO ESTRUTURADO	25
2.6	PARÂMETROS PARA GERAR TRAJETÓRIAS	26
3	Resultados	28

3.1	CÂMERA	28
3.1.1	CONFIGURAÇÃO DA REDE	28
3.1.2	CALIBRAÇÃO	29
3.1.3	PROGRAMAÇÃO	30
3.2	CALIBRAÇÃO DO SERVOMOTOR	30
3.3	MODELO NO ESPAÇO DE ESTADOS	31
3.3.1	DISCRETIZAÇÃO	33
3.3.2	CONTROLE	33
3.4	SIMULAÇÃO.....	34
3.5	RESULTADOS EXPERIMENTAIS	37
3.5.1	CONSIDERAÇÕES INICIAIS	37
3.5.2	TESTES COM RAMPA - MALHA ABERTA E MALHA FECHADA	38
3.5.3	TESTE COM TRAJETÓRIA PROPOSTA - MALHA FECHADA	39
4	Conclusões.....	41
4.1	PERSPECTIVAS FUTURAS.....	42
REFERÊNCIAS BIBLIOGRÁFICAS		43
Anexos.....		46
I	Programas utilizados.....	47
I.1	REDUÇÃO MODAL	47
I.2	FILTRO DE KALMAN UTILIZADO, EM LINGUAGEM MATLAB — ADAPTADO DE [1]	52
I.3	PROJETO DO CONTROLADOR COM REALIMENTAÇÃO DE ESTADOS	54
I.4	TEXTO ESTRUTURADO	54
I.4.1	INICIALIZAÇÃO DOS TESTES	54
I.4.2	INICIALIZAÇÃO DA REDE DEVICENET	54
I.4.3	PROGRAMA DE MOVIMENTAÇÃO DO MOTOR.....	55
I.5	LINGUAGEM <i>ladder</i>	56
I.5.1	EXECUÇÃO DE <i>trigger</i> DA CÂMERA	56
I.5.2	ROTINA DE PARADA DE EMERGÊNCIA	56
I.6	PROGRAMAS EM <i>Python</i> PARA O CONTROLE	56
I.6.1	EXEMPLO DE TESTE EM MALHA ABERTA	56
I.6.2	EXEMPLO DE TESTE EM MALHA FECHADA COM RAMPA	58
I.6.3	EXEMPLO DE TESTE EM MALHA FECHADA COM TRAJETÓRIAS QUAISQUER...	60
I.7	PROGRAMAS DA CÂMERA	64
I.7.1	DETECÇÃO DA POSIÇÃO HORIZONTAL DA BOLINHA.....	64

LISTA DE FIGURAS

1.1	Operação de reentrada [2]	2
1.2	Método atual para reconexão no poço [3]	2
1.3	Método proposto para reconexão no poço [3]	3
2.1	Malha aberta de controle	11
2.2	Malha fechada de controle.....	11
2.3	Malha fechada de controle, espaço de estados discreto.....	12
2.4	Estrutura básica do preditor de Smith [4].	13
2.5	Estrutura do preditor de Smith com filtro de Kalman e referências de topo e fundo [4].	14
2.6	Exemplo de estrutura de sistema discreto no tempo [5].	15
2.7	Exemplo de estrutura de filtro de Kalman [5].	16
2.8	Esquemático da Bancada Utilizada para o Experimento [3].....	17
2.9	CLP com identificação de elementos.....	17
2.10	Line Interface Module modelo 2094-AL09 da Allen Bradley [3]	19
2.11	Drive Kinetix 6000 da Allen Bradley	20
2.12	Servomotor modelo MPL-A310F-SJ22AA.....	20
2.13	Sensor indutivo 871T-R8B18 [3]	21
2.14	Câmera da Banner Engineering utilizada no projeto [3].....	21
2.15	Exemplo de programa <i>ladder</i> com instruções de movimentação.....	25
3.1	RSLinx com câmera reconhecida	29
3.2	Programa PresencePLUS para calibração da câmera	30
3.3	Resposta ao degrau para modelos reduzidos com atraso e sem atraso	32
3.4	Resposta ao degrau para modelos reduzidos com atraso e sem atraso, 25 segundos ..	33
3.5	Resposta do Sistema em Malha Aberta para Excursão de 30cm, entrada rampa	34
3.6	Resposta do Sistema em Malha Aberta para Excursão de 30cm, entrada suave calculada	34
3.7	Esquema principal para simulação em Simulink	35
3.8	Bloco de Atraso – saída é o valor antecipado menos um valor antigo.....	35
3.9	Modelo reduzido – não utiliza atraso, utilizado para predizer a saída	36
3.10	Bloco de controle com filtro de Kalman	36
3.11	Resposta do Sistema em Malha Fechada para Excursão de 30cm, entrada rampa	36

3.12 Resposta do Sistema em Malha Fechada para Excursão de 30cm, entrada suave calculada para topo e fundo	36
3.13 Sistema com um ruído branco adicionado.....	37
3.14 Resposta do Sistema em Malha Fechada para Excursão de 30cm, entrada suave, com ruído	37
3.15 Entrada do controlador antes e depois do ruído para cada instante.....	37
3.16 Resultado experimental para a trajetória rampa em malha aberta de 30 cm.	39
3.17 Resultado experimental para a trajetória rampa em malha fechada de 30 cm.	39
3.18 Gráfico comparativo das respostas em malha aberta e malha fechada com trajetória rampa.	39
3.19 Teste com a trajetória sugerida por Rafael utilizando o preditor de Smith.	40
3.20 Teste com a trajetória sugerida por Rafael, detalhando a execução da trajetória. ...	40
I.1 <i>Trigger</i> da câmera	56
I.2 Parada de emergência	56
I.3 Detecção da posição horizontal da bolinha.....	64

LISTA DE TABELAS

2.1	Constantes do barbante	5
2.2	Constantes da bolinha de isopor	6
2.3	Principais instruções <i>ladder</i>	24
2.4	Principais instruções de controle de movimento em <i>ladder</i>	25
2.5	Dados para simulação em escala real [3]	27
2.6	Dados para simulação em escala laboratorial	27
3.1	Relações mm/px para diferentes seções da barra de alumínio	30
3.2	Dados de calibração do servomotor, média obtida é de 71.32 mm/unidade	31

LISTA DE SÍMBOLOS

Símbolos Latinos

v	Velocidade linear	[m/s]
r	Referência de posição	[m]
W	Letra em negrito representa matriz	

Símbolos Gregos

Υ	Deslocamento horizontal	[m]
ϵ	Atraso	[s]

Subscritos

M	Sistema Modal
R	Sistema Reduzido
D	Sistema Reduzido considerando Atraso

Sobrescritos

.	Variação temporal
$-$	Valor médio
$\hat{\cdot}$	Estimação

Siglas

PCI	<i>Peripheral Component Interconnect</i>
CPU	Unidade Central de Processamento - <i>Central Processing Unit</i>
CAN	<i>Control Area Networking</i>
CIP	Protocolo Industrial Comum - <i>Common Industrial Protocol</i>
CLP	Controlador Lógico Programável
PWM	Modulação por Largura de Pulso - <i>Pulse Width Modulation</i>
SI	Sistema Internacional de Unidades
EDS	<i>Electronic Data Sheet</i>
RSLogix	<i>Rockwell Software Logix 5000</i>
RSLogix5000	<i>Rockwell Software Logix 5000</i>
OLE	<i>Object Linking and Embedding</i>
OPC	<i>OLE for Process Control</i>
SISO	<i>Singular Input, Singular Output</i>

Capítulo 1

Introdução

A motivação do presente trabalho aparece na área petrolífera, nas operações de reentrada de risers; o objetivo é a validação de um sistema de controle em malha fechada em escala laboratorial como uma alternativa ao modelo manual empregado atualmente.

1.1 Contextualização

O petróleo tem importância econômica global. Uma das formas de extração do mesmo é aquela feita em águas profundas, na qual o Brasil tem feito importantes avanços desde a descoberta do Pré-Sal em 2006. Em abril de 2015, chegou-se à produção de mais de 800 mil barris por dia no pré-sal, com campos situados em águas profundas e ultraprofundas [6]. Os desafios nesta área da engenharia são enormes, pois as operações são muito complexas.

Na Figura 1.1, observa-se uma das operações necessárias para a extração no Pré-Sal, especificamente a operação de reentrada. Nesta operação, um *riser* deve ser conectado da plataforma até o poço de petróleo no leito oceânico. O comprimento do *riser* chega a 2km.

Devido à complexidade inerente das diversas operações *offshore*, propulsores e sensores de localização e orientação - GPS, giroscópios, câmeras, etc - são requisitos essenciais para se poder posicionar a embarcação e os *risers* [3].

1.2 Objetivos do projeto

O foco deste trabalho está na operação de reentrada, conforme apresentada na Figura 1.1. Atualmente, é uma operação feita manualmente por um operador na plataforma que observa remotamente as imagens capturadas por *ROVs* (*Remotely Operated Vehicles*) da região do *riser* próxima ao poço e controla a plataforma através de um *joystick* com o auxílio do sistema de posicionamento dinâmico. O custo envolvido na operação é enorme e os riscos para o equipamento

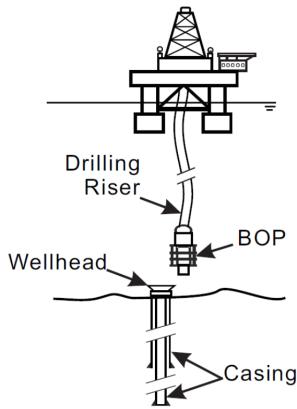


Figura 1.1: Operação de reentrada [2]

também, já que os próprios *ROVs* e o *riser* ficam sujeitos às perturbações das ondas, correntezas e variações ambientais no fundo do mar. A Figura 1.2 apresenta o esquema descrito.

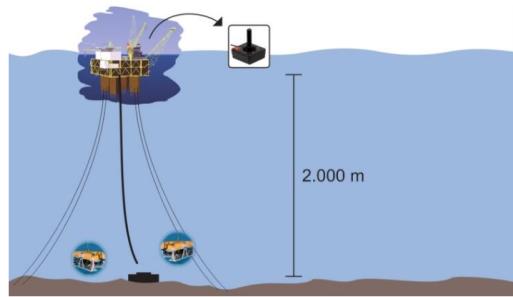


Figura 1.2: Método atual para reconexão no poço [3]

O objetivo deste trabalho é apresentar uma forma mais eficiente de realizar a operação de reentrada, economizando tempo e recursos, evitando riscos para pessoal e equipamento. Fabrício [7] validou por meio de simulação uma técnica de controle em malha aberta e fechada, enquanto Rédyton [3] validou o controle em malha aberta experimentalmente; neste presente trabalho, deseja-se projetar um controle em malha fechada para validação experimental em escala laboratorial, utilizando a técnica apresentada inicialmente por Fabrício [7] e então detalhada e um pouco modificada por Rafael [4]. A malha fechada visa a resistir perturbações, sendo uma técnica mais robusta que a malha aberta para atender aos requisitos de trajetória da operação de reentrada do *riser*.

Para fechar a malha, necessita-se de um sensor que realmente os dados de posição, o qual, na presente bancada, é uma câmera industrial. Passos necessários incluem:

- Atualização do Firmware da Câmera
- Conexão na rede Ethernet/IP da câmera e CLP;

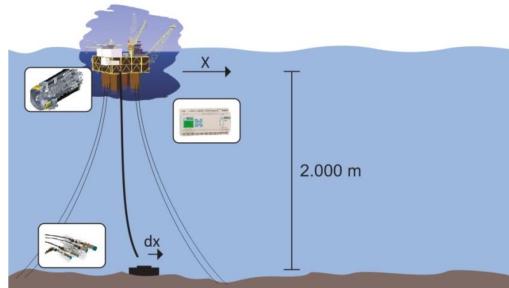


Figura 1.3: Método proposto para reconexão no poço [3]

- Configuração da câmera no software RSLogix;
- Alimentação da bancada com fontes dedicadas de 24V¹
- Configuração do OPC (Subseção 2.4.3) para comunicação entre o CLP e computador;
- Redução do modelo por meio de uma rotina escrita em Julia [8];
- Simulações em Simulink e projeto utilizando MATLAB;
- Programação de rotinas usando Python [9], OPC e Texto Estruturado para o controle.

O controle apresentado por Fabrício [7] utiliza uma estrutura conhecida como Predictor de Smith, uma técnica de controle preditivo. Tal técnica foi utilizada pelo fato do sistema ter um atraso introduzido para reduzir a transferência direta do modelo reduzido. Isso será detalhado durante este trabalho. A determinação dos polos é feita seguindo técnicas de controle digital em espaço de estados. Um filtro de Kalman é presente no sistema para compensar ruídos de medida, ponderando entre modelo e medição. A combinação de controlador, planta e modelo reduzido, em conjunção com o projeto das trajetórias de topo e de fundo do *riser*, conforme determinado em [7], e o filtro de Kalman é a base para se compreender a estrutura do sistema de controle a ser desenvolvido.

¹Anteriormente, duas fontes de tensão de saída ajustável eram utilizadas, mas foram trocadas por fontes de saída de tensão única.

Capítulo 2

Fundamentos

Este capítulo apresenta equações básicas do sistema que se deseja validar, assim como a redução modal utilizada, o projeto para obtenção dos ganhos para um dado conjunto de polos, e apresenta uma visão geral sobre o filtro de Kalman e o Preditor Smith. Também são expostas informações sobre a bancada laboratorial e sobre a programação do CLP.

2.1 Modelagem

2.1.1 Equações Governantes

Estruturas submarinas tais como os *risers* são esbeltas e tem um alto módulo de cisalhamento. Portanto, a simplificação de Euler-Bernoulli para vigas é utilizada para propósitos de modelagem. O deslocamento de interesse é o horizontal e o *riser* está sob a ação de forças hidrodinâmicas externas e de tração. A equação diferencial parcial para a variável deslocamento, Υ , é dada por

$$m_s \frac{\partial^2 \Upsilon}{\partial t^2} = -EJ \frac{\partial^4 \Upsilon}{\partial z^4} + \frac{\partial}{\partial z} \left(T(z) \frac{\partial \Upsilon}{\partial z} \right) + F_n(z, t), \quad (2.1)$$

na qual m_s é a densidade linear do tubo, E é o módulo de Young e J é o segundo momento de inércia do *riser*. $T(z)$ descreve as forças de tração ao longo do comprimento do *riser*. $F_n(z, t)$ é a força resultante externa — força linear, unidade N/m [7].

No caso do problema em escala laboratorial que foi trabalhado, o tubo é representado por um barbante e daqui em diante serão discutidas e apresentadas as variáveis necessárias para se trabalhar com esse problema:

- m_s é a massa linear do barbante (densidade linear, kg/m);
- E é o módulo de Young do barbante e ele é desconhecido;
- J é o segundo momento de área e representa a resistência do barbante à flexão – como o barbante não apresenta tal resistência, $J = 0$;

- $T(z)$ é a força de tração e é dada por

$$T(z) = (m_b + zm_s) g,$$

sendo m_b a massa da bolinha (kg), $m_s = m_{\text{barbante,kg}}/L$, sendo L o comprimento do barbante, z a posição vertical a partir do carrinho e g é a força da gravidade.

As únicas forças externas atuando no *riser* são hidrodinâmicas, exceto nas extremidades do topo e do fundo, nas quais forças de reação seguem condições de contorno. A equação de Morison descreve a força externa resultante, dada por

$$F_n(z, t) = -m_{fbar} \frac{\partial^2 \Upsilon}{\partial t^2} - \mu \left| \frac{\partial \Upsilon}{\partial t} \right| \frac{\partial \Upsilon}{\partial t}, \quad (2.2)$$

na qual μ é o coeficiente de arrasto (unidade 1/s) e m_{fbar} é a massa do fluido adicionado, que será posteriormente pormenorizada. Fazendo $m = m_s + m_{fbar}$ e substituindo a Equação 2.2 na 2.1, obtém-se

$$\frac{\partial^2 \Upsilon}{\partial t^2} = -\frac{EJ}{m} \frac{\partial^4 \Upsilon}{\partial z^4} + \frac{\partial}{\partial z} \left(\frac{T(z)}{m} \frac{\partial \Upsilon}{\partial z} \right) - \frac{\mu}{m} \left| \frac{\partial \Upsilon}{\partial t} \right| \frac{\partial \Upsilon}{\partial t}. \quad (2.3)$$

Em relação à massa do fluido adicionado, m_{fbar} é dada por

$$\begin{aligned} m_{fbar} &= 2\pi r_{bar}^2 \rho_{ar} \\ &= 0.00770 \text{ g/m}. \end{aligned} \quad (2.4)$$

Já que $m_{fbar} \ll m_s$ conforme a Tabela 2.1, consideraremos $m \approx m_s$ nos cálculos. Em relação à massa m_{fb} do fluido adicionado ao redor da bolinha de isopor, ela é dada por

$$\begin{aligned} m_{fb} &= 1.2V_b \rho_{ar} \\ &= 1.2 \left(\frac{4}{3}\pi r_b^3 \right) \rho_{ar} \\ &= 0.0220 \text{ g}, \end{aligned} \quad (2.5)$$

de onde se pode observar que $m_{fb} \ll m_b$, conforme Tabela 2.2. Assim, os cálculos consideraram $m' \approx m_b$.

Tabela 2.1: Constantes do barbante

Significado	Símbolo	Valor	Unidade
Massa	m_{bar}	0.492	g
Comprimento	L	0.82	m
Massa linear	m_s	0.6	g/m
Raio	r_{bar}	1	mm
Densidade	ρ_{bar}	191	kg/m ³

Um ponto importante de se notar é que o barbante pesa mais do que o isopor, o que faz com que a tração não seja principalmente devida pela bolinha, mas sim pelo barbante. Neste caso,

Tabela 2.2: Constantes da bolinha de isopor

Significado	Símbolo	Valor	Unidade
Massa	m_b	0.492	g
Raio	r_b	15.3	mm
Coeficiente de inércia	C_m	1.2	-
Coeficiente de arrasto	C_d	0.6	-
Volume	V_b	$\frac{4}{3}\pi r_b^3$	m^3
Área da seção transversal	A_b	πr_b^2	m^2

não se utiliza um valor médio para $T(z)$ como em [7], mas ainda se pode usar um valor médio para as constantes τ e τ' , que substituem o termo $\frac{\mu}{m} \left| \frac{\partial \Upsilon}{\partial t} \right|$ para o barbante e para a bolinha, respectivamente. Essas constantes são definidas de acordo com a trajetória prevista, uma vez que a velocidade média depende dessa trajetória. Já levando em conta um valor médio para $\left| \frac{\partial \Upsilon}{\partial t} \right|$, tem-se

$$\frac{\partial^2 \Upsilon}{\partial t^2} = -\frac{EJ}{m} \frac{\partial^4 \Upsilon}{\partial z^4} + \frac{\partial}{\partial z} \left(\frac{T(z)}{m} \frac{\partial \Upsilon}{\partial z} \right) - \tau \frac{\partial \Upsilon}{\partial t}. \quad (2.6)$$

Neste trabalho, fizeram-se experimentos com uma excursão do carrinho de 30cm. O valor de τ e τ' foi calculado por Rafael Simões [4]. Para o barbante, $\tau = 0.2426/\text{s}$. Para a bolinha, $\tau' = 0.1133/\text{s}$.

Antes de se prosseguir para a discretização e obtenção das matrizes em espaço de estados, é importante pensar nas condições de contorno. No topo, $z = L$, tem-se $\Upsilon(L, t) = u(t)$, ou seja, o carrinho se move conforme uma trajetória $u(t)$ definida. Neste mesmo ponto, $\frac{\partial \Upsilon}{\partial z}(L, t) = 0$. Para a ponta na qual a carga está situada, $z = 0$, tem-se $\frac{\partial \Upsilon}{\partial z}(0, t) = \frac{F_L}{T}$, sendo F_L a força aplicada pela ponta do riser na carga.

2.1.2 Discretização

O objetivo desta seção é representar a Equação 2.6 em um espaço de estados finito discreto. Para isso, aplica-se o método de diferenças finitas na coordenada z de maneira a se aproximar a EDP governante em um número finito de EDOs [7]. No espaço discreto, a equação do k -ésimo elemento é dada por

$$\begin{aligned} \frac{d^2 \Upsilon_k}{dt^2} &= -\frac{EJ}{ml^4} (\Upsilon_{k-2} - 4\Upsilon_{k-1} + 6\Upsilon_k - 4\Upsilon_{k+1} + \Upsilon_{k+2}) \\ &\quad + \frac{T_0 + mg(k-1)l}{ml^2} (\Upsilon_{k-1} - 2\Upsilon_k + \Upsilon_{k+1}) + g \frac{-\Upsilon_{k-1} + \Upsilon_{k+1}}{2l} - \tau \frac{d\Upsilon_k}{dt}, \end{aligned} \quad (2.7)$$

sendo N o número de pontos de discretização e l a distância entre dois pontos vizinhos ($l = L/N$).

Deve-se notar que $k \in \mathbb{N} : 2 \leq k \leq N - 1$, pois um dos extremos é a bolinha, caso no qual $k = 1$, e a equação do pêndulo rege seu movimento, enquanto que na outra ponta, $k = N$, se aplica uma condição de contorno que é a entrada da planta $u(t)$. O que aconteceria quando $k = 2$ e se precisasse de Υ_{k-2} ? Para o presente experimento, $J = 0$ e esse problema não ocorre. Caso se

façam testes com um valor de $J \neq 0$, é necessário resolver esse problema primeiro. Uma solução seria utilizar diferenças finitas unilaterais para esses pontos próximos à fronteira.

Para simplificar, definem-se as constantes

$$a = -\frac{EJ}{ml^4}, \quad (2.8)$$

$$b_k = \frac{T_0 + mg(k-1)l}{ml^2}, \quad k \geq 2, \quad (2.9)$$

$$c = \frac{g}{2l}, \quad (2.10)$$

$$d_k = b_k - c, \quad k \geq 2 \text{ e} \quad (2.11)$$

$$e_k = b_k + c, \quad k \geq 2. \quad (2.12)$$

Uma estratégia para se analisar como as matrizes do espaço de estados do sistema ficarão é escolher um número de pontos N pequeno e escrever todas as equações. Depois de compreendido o padrão desse sistema pequeno, pode-se generalizar e criar uma rotina que crie as matrizes para qualquer N . Observa-se que $a = 0$ para o barbante, pois $J = 0$, como apresentado anteriormente, o que simplifica os cálculos.

Para o caso $N = 6$, por exemplo, tem-se

$$\mathbf{x} = (\Upsilon_1 \ \Upsilon_2 \ \Upsilon_3 \ \Upsilon_4 \ \Upsilon_5 \ \Upsilon_6 \ \dot{\Upsilon}_1 \ \dot{\Upsilon}_2 \ \dot{\Upsilon}_3 \ \dot{\Upsilon}_4 \ \dot{\Upsilon}_5 \ \dot{\Upsilon}_6)^T, \quad (2.13)$$

$$u = \Upsilon(L, t) = \Upsilon_7 \text{ e} \quad (2.14)$$

$$y = \Upsilon(0, t) = \Upsilon_1. \quad (2.15)$$

Para o vetor $\dot{\mathbf{x}}$ de interesse, as equações para os estados de velocidade $\dot{\Upsilon}_i$, $1 \leq i \leq 6$, já fazem parte do vetor de estados, então não é necessário discriminá-las. Já para os pontos de aceleração fora da bolinha $\ddot{\Upsilon}_i$, $2 \leq i \leq 6$, suas equações são dadas por

$$\begin{aligned} \ddot{\Upsilon}_2 &= b_2(\Upsilon_1 - 2\Upsilon_2 + \Upsilon_3) + c(-\Upsilon_1 + \Upsilon_3) - \tau\dot{\Upsilon}_2 \\ &= d_2\Upsilon_1 - 2b_2\Upsilon_2 + e_2\Upsilon_3 - \tau\dot{\Upsilon}_2, \end{aligned} \quad (2.16)$$

$$\begin{aligned} \ddot{\Upsilon}_3 &= b_3(\Upsilon_2 - 2\Upsilon_3 + \Upsilon_4) + c(-\Upsilon_2 + \Upsilon_4) - \tau\dot{\Upsilon}_3 \\ &= d_3\Upsilon_2 - 2b_3\Upsilon_3 + e_3\Upsilon_4 - \tau\dot{\Upsilon}_3, \end{aligned} \quad (2.17)$$

$$\begin{aligned} \ddot{\Upsilon}_4 &= b_4(\Upsilon_3 - 2\Upsilon_4 + \Upsilon_5) + c(-\Upsilon_3 + \Upsilon_5) - \tau\dot{\Upsilon}_4 \\ &= d_4\Upsilon_3 - 2b_4\Upsilon_4 + e_4\Upsilon_5 - \tau\dot{\Upsilon}_4, \end{aligned} \quad (2.18)$$

$$\begin{aligned} \ddot{\Upsilon}_5 &= b_5(\Upsilon_4 - 2\Upsilon_5 + \Upsilon_6) + c(-\Upsilon_4 + \Upsilon_6) - \tau\dot{\Upsilon}_5 \\ &= d_5\Upsilon_4 - 2b_5\Upsilon_5 + e_5\Upsilon_6 - \tau\dot{\Upsilon}_5 \text{ e} \end{aligned} \quad (2.19)$$

$$\begin{aligned} \ddot{\Upsilon}_6 &= b_6(\Upsilon_5 - 2\Upsilon_6 + u) + c(-\Upsilon_5 + u) - \tau\dot{\Upsilon}_6 \\ &= d_6\Upsilon_5 - 2b_6\Upsilon_6 + e_6u - \tau\dot{\Upsilon}_6. \end{aligned} \quad (2.20)$$

A equação para a posição da carga Υ_1 leva em conta a massa da bolinha e a força de Morison e é dada por

$$m_b \ddot{\Upsilon}_1 = \frac{m_b g}{l} (\Upsilon_2 - \Upsilon_1) + \rho_{\text{ar}} C_m V_b \ddot{\Upsilon}_1 - \frac{1}{2} \rho_{\text{ar}} C_d A_b \dot{\Upsilon}_1 |\dot{\Upsilon}_1|. \quad (2.21)$$

Isolando $\ddot{\Upsilon}_1$ na Equação 2.21, tem-se

$$\ddot{\Upsilon}_1 = \frac{m_b g}{m' l} (\Upsilon_2 - \Upsilon_1) - \frac{1}{2m'} \rho C_d A_b |\dot{\Upsilon}_1|. \quad (2.22)$$

Nota-se que $m' = m_b + \rho_{\text{ar}} C_m V_b = m_b + m_{fb} \approx m_b$. Assim, assume-se $m' = m_b$ para os cálculos. Anteriormente, foi apresentada a linearização τ para o termo $\frac{1}{2m'} \rho C_d A_b |\dot{\Upsilon}_k|$ do cabo. Para o caso da bolinha de isopor, essa constante é diferente e é denotada por τ' , resultando na equação final para Υ_1 , que é

$$\ddot{\Upsilon}_1 = b_1 (-\Upsilon_1 + \Upsilon_2) - \tau' \dot{\Upsilon}_1, \quad (2.23)$$

sendo

$$b_1 = \frac{m_b g}{m' l} = \frac{g}{l}. \quad (2.24)$$

Desta forma, a partir das Equações 2.23 e 2.16-2.20, pode-se definir o sistema linear em forma matricial, resultando em

$$\dot{\mathbf{x}} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -b_1 & b_1 & 0 & 0 & 0 & 0 & -\tau' & 0 & 0 & 0 & 0 & 0 \\ d_2 & -2d_2 & e_2 & 0 & 0 & 0 & 0 & -\tau & 0 & 0 & 0 & 0 \\ 0 & d_3 & -2d_3 & e_3 & 0 & 0 & 0 & 0 & -\tau & 0 & 0 & 0 \\ 0 & 0 & d_4 & -2d_4 & e_4 & 0 & 0 & 0 & 0 & -\tau & 0 & 0 \\ 0 & 0 & 0 & d_5 & -2d_5 & e_5 & 0 & 0 & 0 & 0 & -\tau & 0 \\ 0 & 0 & 0 & 0 & d_6 & -2d_6 & 0 & 0 & 0 & 0 & 0 & -\tau \end{bmatrix} \mathbf{x} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ e_6 \end{bmatrix} u, \quad (2.25)$$

que pode ser representado concisamente como

$$\dot{\mathbf{x}} = \begin{bmatrix} \mathbf{0}_{6 \times 6} & \mathbf{I}_{6 \times 6} \\ \mathbf{M}_{6 \times 6} & \mathbf{L}_{6 \times 6} \end{bmatrix} \mathbf{x} + \begin{bmatrix} \mathbf{0}_{11 \times 1} \\ e_6 \end{bmatrix} u. \quad (2.26)$$

Para o caso de uma discretização com N pontos, tem-se

$$\dot{\mathbf{x}} = \begin{bmatrix} \mathbf{0}_{N \times N} & \mathbf{I}_{N \times N} \\ \mathbf{M}_{N \times N} & \mathbf{L}_{N \times N} \end{bmatrix} \mathbf{x} + \begin{bmatrix} \mathbf{0}_{2N-1 \times 1} \\ e_N \end{bmatrix} u, \quad (2.27)$$

$$y = \begin{bmatrix} 1 & \mathbf{0}_{1 \times 2N-1} \end{bmatrix} \mathbf{x}. \quad (2.28)$$

2.1.3 Estratégia de Redução da ordem do modelo

A maior parte da teoria clássica de controle lida com sistemas representados por um pequeno número de variáveis de estado. Portanto, uma forma de aplicar métodos clássicos de controle da

literatura para sistemas de parâmetros distribuídos discretos é por meio de uma redução da ordem do modelo [7].

Tal redução do modelo será feita em duas etapas: primeiro, uma transformação modal é aplicada nas equações originais do espaço de estados, resultando em uma nova representação em variáveis modais. Nesta forma, o sistema pode ser visto como um conjunto de subsistemas dissociados em paralelo, cuja influência na saída pode ser calculada individualmente. Então, os subsistemas com os maiores ganhos estáticos são escolhidos para criar um modelo de ordem reduzida [7].

Primeiro, deve-se obter os autovalores do espaço de estados do *riser*. Observa-se que eles são sempre distintos entre si, uma condição suficiente para a diagonalização da matriz do espaço de estados. Assim, calcula-se a matriz modal \mathbf{T} , cuja i -ésima coluna é o i -ésimo autovetor do sistema:

$$\mathbf{T} = (\mathbf{v}_1 \mid \mathbf{v}_2 \mid \dots \mid \mathbf{v}_{2N})_{1 \times 2N} \quad (2.29)$$

A matriz \mathbf{T} provavelmente tem valores complexos. Isso é um problema para a representação em espaço de estados e sua simulação. A solução é criar uma matriz $\tilde{\mathbf{T}}$ que tenha só números reais. Antes de explicar como criá-la, deve-se lembrar que os autovalores complexos sempre aparecem em pares conjugados, já que a matriz \mathbf{A} só tem valores reais. Quando a primeira coluna de um autovetor de um par complexo conjugado for encontrada, a coluna respectiva de $\tilde{\mathbf{T}}$ será sua parte real. A segunda coluna desse par complexo conjugado será a parte imaginária da coluna de \mathbf{T} .

A matriz $\tilde{\mathbf{T}}$ é utilizada para uma transformação de similaridade no sistema original:

$$\mathbf{A}_M = \tilde{\mathbf{T}}^{-1} \mathbf{A} \tilde{\mathbf{T}}, \quad (2.30)$$

$$\mathbf{x}_M = \tilde{\mathbf{T}}^{-1} \mathbf{x}, \quad (2.31)$$

$$\mathbf{B}_M = \tilde{\mathbf{T}}^{-1} \mathbf{B}, \text{ e} \quad (2.32)$$

$$\mathbf{C}_M = \mathbf{C} \tilde{\mathbf{T}}. \quad (2.33)$$

O sistema transformado, denotado pelo subscrito M , é mais adequado à análise. \mathbf{A}_M é uma matriz diagonal por blocos, com seus autovalores explícitos, e permitindo o desacoplamento do sistema original em N subsistemas formados por pares de autovalores reais ou complexo-conjugados. Nota-se que cada subsistema é de ordem 1 ou 2 dependendo se o autovetor é real ou um par complexo conjugado.

Neste estágio, procura-se determinar quais dos subsistemas são mais adequados para aproximar o modelo original por meio do cálculo do ganho estático de cada um. Este método depende da predominância de uns poucos autovalores na resposta do sistema, já que altas frequências são muito atenuadas pelas forças hidrodinâmicas e pela suavidade da entrada.

Os subsistemas selecionados são combinados em um modelo reduzido

$$\dot{\mathbf{z}} = \mathbf{A}_R \mathbf{z} + \mathbf{B}_R u \quad (2.34)$$

$$y = \mathbf{C}_R \mathbf{z} + \mathbf{D}_R u \quad (2.35)$$

cuja ordem é escolhida considerando o custo-benefício entre a acurácia da dinâmica reduzida e a simplicidade da estrutura de controle exigida. Além disso, o sistema reduzido deve compensar o ganho estático perdido nos autovalores desconsiderados. Isto é feito por meio de uma matriz de transferência direta \mathbf{D}_R , que é a diferença dos ganhos dos sistemas original e reduzido:

$$\mathbf{D}_R = \mathbf{C}\mathbf{A}^{-1}\mathbf{B} - \mathbf{C}_R\mathbf{A}_R^{-1}\mathbf{B}_R. \quad (2.36)$$

A matriz de transferência direta \mathbf{D}_R introduz novas dinâmicas: uma saída não-nula que não leva em conta o atraso de propagação da entrada e um ganho em altas frequências. Conforme mostrado por Fortaleza [10], pode-se refinar o modelo reduzido introduzindo um atraso de entrada ϵ que minimiza a transferência direta e garante dinâmica nula para $t < \epsilon$:

$$\begin{aligned} \dot{\mathbf{z}} &= \mathbf{A}_R\mathbf{z} + \mathbf{B}_D u(t - \epsilon), \\ y &= \mathbf{C}_R\mathbf{z} + \mathbf{D}_D u(t - \epsilon); \end{aligned} \quad (2.37)$$

sendo

$$\mathbf{B}_D = \mathbf{A}_R \left(e^{\epsilon \mathbf{A}_R} \right) \mathbf{A}_R^{-1} \mathbf{B}_R \text{ e} \quad (2.38)$$

$$\mathbf{D}_D = \mathbf{C}_R \left(e^{\epsilon \mathbf{A}_R} - \mathbf{I} \right) \mathbf{A}_R^{-1} \mathbf{B}_R + \mathbf{D}_R. \quad (2.39)$$

O novo modelo reduzido (2.37) é tal que, para uma entrada degrau no instante t' , a saída mantém seu valor inicial enquanto $t < t' + \epsilon$. Para $t \geq t' + \epsilon$, ambos os modelos reduzidos produzem a mesma saída. O atraso ϵ pode ser visto como uma aproximação para o atraso natural de propagação da estrutura.

2.2 Controle

2.2.1 Técnicas Simples de Controle

Técnicas de controle simples devem ser introduzidas de forma a se compreender o objetivo deste trabalho, que é do posicionamento do *riser* por meio de controle em malha fechada. Primeiramente, apresenta-se o controle em malha aberta, cujo diagrama pode ser observado na Figura 2.1. A variável $r = r(t)$ é a referência do sistema. Neste tipo de controle, a saída não é realimentada na entrada. Desta forma, este tipo de controle não requer sensores, pois somente é fornecida uma referência de entrada para a planta e espera-se que a planta reaja de acordo. Caso haja erros para seguir a trajetória, eles não poderão ser compensados. Assim, esse tipo de controle é mais recomendado quando o sistema é preciso e há pouca ou nenhuma perturbação. No entanto, este não é o caso do *riser*, pois o movimento das águas no leito oceânico perturba o tubo, causando erros na posição final desejada. O controle malha aberta foi anteriormente verificado por Rédyton [3] e também é reavaliado nesse trabalho.

Uma forma de se compensar as perturbações do ambiente é realimentando a saída na entrada, calculando a diferença entre a referência e o valor medido. Assim, um valor de erro $e = e(t)$ é obtido e o sistema calcula o sinal u conforme o erro evolui. A Figura 2.2 mostra um esquema básico deste sistema, evidenciando a presença de uma malha fechada de controle.

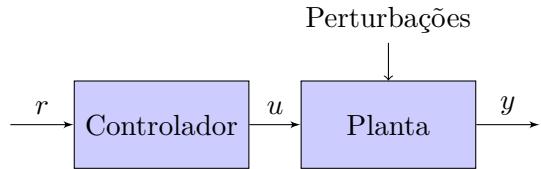


Figura 2.1: Malha aberta de controle

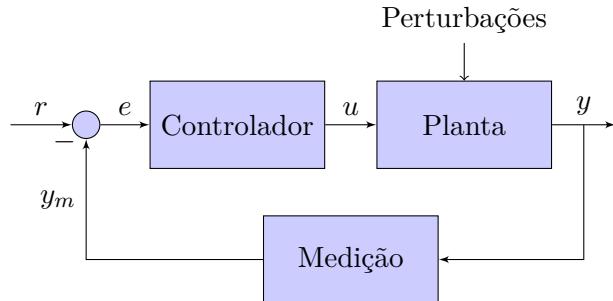


Figura 2.2: Malha fechada de controle

2.2.2 Controle Discreto no Espaço de Estados

Na seção 2.1, obtiveram-se equações do sistema contínuo no tempo, mas discretizado espacialmente. A discretização espacial transformou a equação diferencial parcial, um sistema de ordem infinita [7], em um sistema em espaço de estados finito. Mais informações sobre o espaço de estados estão disponíveis em [11] e [12].

Neste trabalho, uma câmera é utilizada para leitura da posição da bolinha, conforme é mostrado na seção 2.3. A câmera demora de 20 a 50ms para terminar uma leitura de posição, dependendo das ferramentas utilizadas no *software* da mesma. Desse modo, faz sentido trabalhar de forma discreta. Além do tempo de medição, o CLP tem um limite de processamento que deve ser respeitado. Quanto maior o tempo de amostragem, menos cálculos são realizados, o que é vantajoso para não sobrecarregar o sistema. Por outro lado, o tempo de amostragem não pode ser arbitrariamente grande, já que é importante seguir o teorema de Nyquist [11] que requer que a frequência de amostragem seja maior que duas vezes a frequência da maior oscilação. Para o presente trabalho, $T_s = 0.1\text{s}$ atende esse requerimento, o que economiza processamento se comparado com um período de 20 a 50ms que seria o mínimo possível devido à câmera.

Definido o período de amostragem, o sistema contínuo da Equação 2.37 deve ser convertido para espaço discreto. A função **c2d** do MATLAB [13] faz a conversão para o espaço discreto, bastando fornecer o período de amostragem T_s . O modo padrão é a conversão utilizando um segurador de ordem zero. Uma vez tendo o modelo discreto, projeta-se o controlador, alocando-se os polos no plano discreto z [12].

O controle escolhido utiliza realimentação de estados com um canal integral. O integrador adiciona um estado a mais no sistema em malha fechada, aumentando a ordem do sistema reduzido de 4 para 5. Considerando um sistema definido pelas matrizes **A**, **B** e **C**, o sistema aumentado é definido pelas matrizes **Â** e **Â**. O projeto considera essas duas matrizes para a definição dos polos.

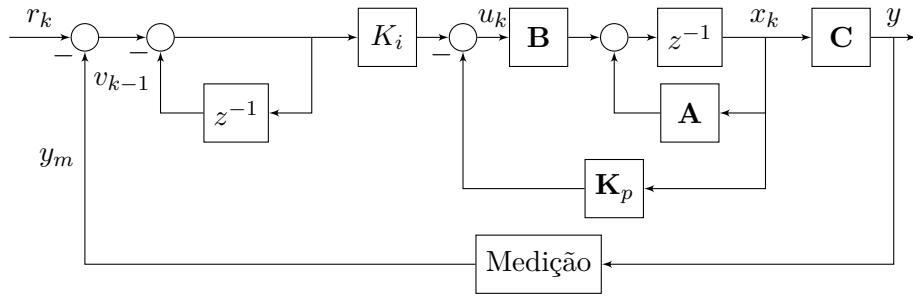


Figura 2.3: Malha fechada de controle, espaço de estados discreto

A Figure 2.3 mostra como é a malha fechada com a realimentação dos quatro estados do modelo reduzido por meio dos ganhos do vetor \mathbf{K}_p e também com a integração do erro com um ganho K_i . Considerando-se essa figura, as matrizes do sistema aumentado são dadas por

$$\hat{\mathbf{A}} = \begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \text{ e} \quad (2.40)$$

$$\hat{\mathbf{B}} = \begin{bmatrix} \mathbf{0} \\ 1 \end{bmatrix}. \quad (2.41)$$

$$(2.42)$$

O projeto do controlador resume-se a calcular o ganho $\hat{\mathbf{K}}$ a partir da fórmula de Acker, a qual é explicada em [12], que resulta em

$$\hat{\mathbf{K}} = [0, 0, 0, 0, 1] [\hat{\mathbf{B}}, \hat{\mathbf{A}}\hat{\mathbf{B}}, \hat{\mathbf{A}}^2\hat{\mathbf{B}}, \hat{\mathbf{A}}^3\hat{\mathbf{B}}, \hat{\mathbf{A}}^4\hat{\mathbf{B}}] \phi(\hat{\mathbf{A}}), \quad (2.43)$$

sendo a função $\phi(\hat{\mathbf{G}})$ dada por

$$\phi(\hat{\mathbf{A}}) = \hat{\mathbf{A}}^5 + \alpha_1 \hat{\mathbf{A}}^4 + \alpha_2 \hat{\mathbf{A}}^3 + \alpha_3 \hat{\mathbf{A}}^2 + \alpha_4 \hat{\mathbf{A}} + \alpha_5 \mathbf{I}, \quad (2.44)$$

na qual os parâmetros α_i são coeficientes do polinômio característico desejado, tal como

$$z^5 + \alpha_1 z^4 + \alpha_2 z^3 + \alpha_3 z^2 + \alpha_4 z + \alpha_5 = 0. \quad (2.45)$$

O MATLAB possui comando que permite obter $\hat{\mathbf{K}}$ a partir dos pólos do sistema:

```
1 Khat = acker(Ahat, Bhat, p);
```

As matrizes aumentadas apresentadas anteriormente consideram uma forma mais simples para obtenção do vetor de ganhos $\hat{\mathbf{K}}$, que deve ser então convertido para um vetor \mathbf{K} que pode ser utilizado no projeto, por meio de

$$\mathbf{K} = (\hat{\mathbf{K}} + [\mathbf{0} \ 1]) \begin{bmatrix} \mathbf{A} - \mathbf{I}_n & \mathbf{B} \\ \mathbf{C}\mathbf{A} & \mathbf{C}\mathbf{B} \end{bmatrix}^{-1}. \quad (2.46)$$

Na Equação 2.46, note que n é a ordem da matriz \mathbf{A} . O vetor \mathbf{K} contém os ganhos proporcionais, vetor \mathbf{K}_p , e o ganho integral, escalar K_i , que serão utilizados. Esses elementos podem ser extraídos de \mathbf{K} , sendo sua estrutura

$$\mathbf{K} = \begin{bmatrix} \mathbf{K}_p \\ K_i \end{bmatrix}. \quad (2.47)$$

Mais detalhes sobre o projeto deste controlador são fáceis de encontrar em livros de controle tais como [12].

2.2.3 Predictor de Smith

Segundo Rafael [4], não é trivial desenvolver um controlador comum com realimentação de estados caso o sistema a ser considerado possua atraso, devido aos seguintes fatos:

- Os efeitos de uma perturbação externa demoram a ser detectados pelo controlador;
- As ações de controle demoram a surtir efeito nas variáveis controladas;
- O controlador, na realidade, tenta corrigir estados que já passaram.

De forma a se lidar com os problemas envolvidos em sistemas com atraso, é possível facilitar o projeto do controlador utilizando uma estrutura conhecida como predictor de Smith. Esse predictor pode ser visto como um compensador de atraso, tornando possível projetar um controlador para o sistema equivalente sem atraso, facilitando o desenvolvimento de uma estratégia de controle. A Figura 2.4 mostra uma estrutura básica do predictor de Smith, em que \mathbf{P} é a planta, \mathbf{C} é um controlador desenvolvido com realimentação, \mathbf{RM} é o modelo obtido por redução modal, sem atraso e $e^{-\epsilon s}$ é o atraso puro.

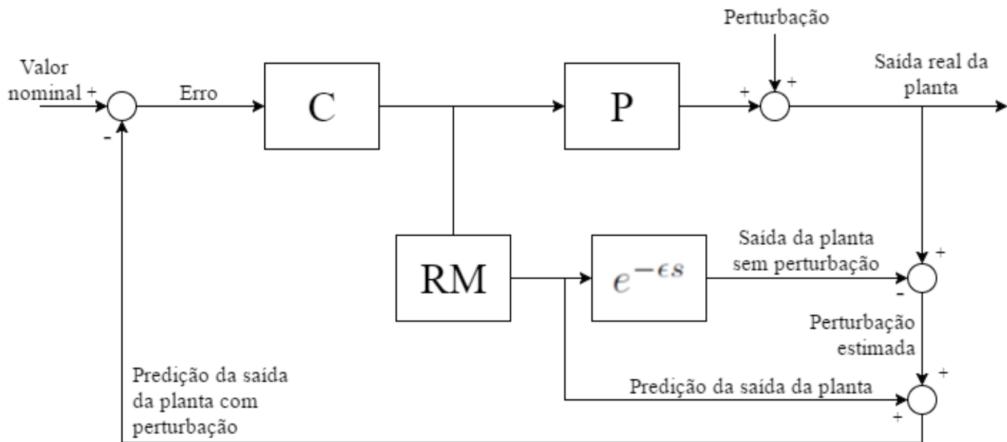


Figura 2.4: Estrutura básica do predictor de Smith [4].

A saída do modelo reduzido com atraso é uma estimativa da saída da planta desconsiderando-se efeitos de perturbação. Subtraindo-se essa saída com atraso da saída real da planta, tem-se

uma estimativa de perturbação que, somada à saída sem atraso, resulta em uma predição da saída com perturbação; assim, a realimentação do preditor considera a perturbação sem ser influenciada pelo atraso.

A grande vantagem de se utilizar o preditor de Smith é que, dado o fato que o projeto pode ser realizado sobre o modelo reduzido sem atraso, podem ser utilizadas técnicas elementares de controle em espaço de estados, apresentadas na Seção 2.2.1. Conhecendo-se o atraso ϵ , e sabendo que o modelo reduzido é um sistema SISO invariante no tempo e controlável, ele pode ser representado em forma canônica de controle. De posse desse fato, pode-se montar uma nova estrutura para o preditor de Smith, conforme a Figura 2.5. Nessa estrutura, **P** é a planta, **C** é o controlador em malha fechada, **RM** é o modelo sem atraso e **KF** é um Filtro de Kalman, que utiliza a forma canônica de controle do sistema de forma a ter sua saída simplificada. O funcionamento do filtro será explicado adiante.

Com essa nova estrutura, dada a linearidade do sistema, o princípio de superposição é válido; portanto, os problemas de planejamento e acompanhamento de trajetória podem ser tratados de forma separada. O preditor leva em consideração as referências de trajetória de topo, $u^* = \Upsilon_o(L, t)$, e de trajetória de fundo, $y^* = \Upsilon_R(0, t)$. Desta forma, garante-se que a trajetória de fundo seja seguida, mesmo com a presença de perturbações.

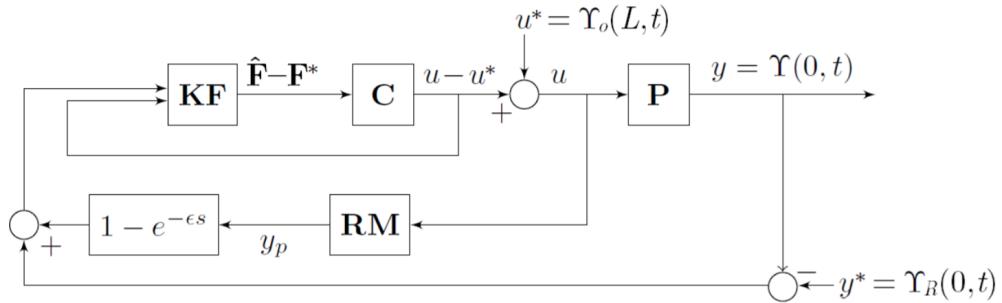


Figura 2.5: Estrutura do preditor de Smith com filtro de Kalman e referências de topo e fundo [4].

Apesar de mencionadas as trajetórias de topo e fundo, dadas por u^* e y^* , respectivamente, este trabalho não demonstra como elas foram calculadas. Foram utilizadas trajetórias determinadas por Rafael [4]. Essas trajetórias permitem um controle adequado em malha aberta na ausência de perturbações e uma referência suave para ser seguida em malha fechada. A seção de resultados irá apresentar essas trajetórias nos resultados de simulação e experimentais.

2.2.4 Filtro de Kalman

Uma estrutura essencial para um sistema de controle real é o filtro de Kalman. Esse filtro é capaz de estimar valores tanto para o processo quanto para o resultado, compensando os erros associados a ruídos de medição e à incerteza do modelo. Assumindo-se que os estados do sistema

sejam observáveis, é possível projetar o **Filtro de Kalman**, que leva o nome do seu criador, Rudolf E. Kalman.

O Filtro de Kalman, também conhecido como observador ótimo, é responsável por estimar, utilizando uma estrutura de predição e correção, variáveis de estado não medidas de forma a se utilizar os resultados obtidos em uma dada estratégia de controle [5]. O Filtro de Kalman original é aplicado, principalmente, em sistemas discretos no tempo, como o que foi obtido para este projeto por meio da redução modal. A Figura 2.6 mostra um exemplo de processo no qual o filtro pode ser aplicado.

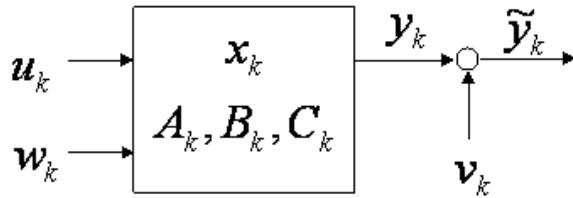


Figura 2.6: Exemplo de estrutura de sistema discreto no tempo [5].

O sistema representado pela Figura 2.6 pode então ser representado por equações lineares, como se segue:

$$x_k = \mathbf{A}_k x_{k-1} + \mathbf{B}_k u_k + w_{k-1}, \quad (2.48)$$

$$y_k = \mathbf{C}_k x_k, \quad (2.49)$$

$$\tilde{y}_k = y_k + v_k. \quad (2.50)$$

No sistema representado pelas Equações 2.48-2.50, as matrizes \mathbf{A}_k , \mathbf{B}_k e \mathbf{C}_k são as matrizes próprias do espaço de estados do sistema discreto; x_k é o vetor de variáveis de estado, y_k é a saída do processo e \tilde{y}_k é uma medida da referida saída. As variáveis w_k e v_k são ruídos associados respectivamente ao processo e à medida. Suas covariâncias podem ser representadas por \mathbf{Q}_k para o processo e \mathbf{R}_k para a medida.

Dada a estrutura do sistema, o objetivo do uso do filtro de Kalman é estimar os dados do processo e a medida das saídas, dadas as entradas e as medições das saídas. A Figura 2.7 mostra um esquema de uso do filtro, em que \hat{x}_k e \hat{y}_k são, respectivamente, as estimativas das variáveis de estado e da saída.

O Filtro de Kalman é um algoritmo de dois passos: predição, em que o estado mais recente e a estimativa do erro de covariância são projetados de forma a se computar o estado atual das variáveis de estado, e correção, onde o estado predito pelo primeiro passo é corrigido incorporando-se a medida mais recente do processo para se gerar uma nova estimativa do estado [5]. Matematicamente, a predição é dada por

$$\hat{x}_k^- = \mathbf{A}_k \hat{x}_{k-1} + \mathbf{B}_k u_k \text{ e} \quad (2.51)$$

$$\hat{\mathbf{P}}_k^- = \mathbf{A}_k \mathbf{P}_{k-1} \mathbf{A}_k^T + \mathbf{Q}_k, \quad (2.52)$$

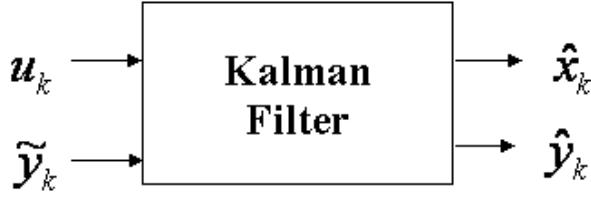


Figura 2.7: Exemplo de estrutura de filtro de Kalman [5].

enquanto que a correção pode ser descrita por

$$\mathbf{K}_k = \hat{\mathbf{P}}_k^{-} \mathbf{C}_k^T \left(\mathbf{C}_k \hat{\mathbf{P}}_k^{-} \mathbf{C}_k^T + \mathbf{R}_k \right)^{-1}, \quad (2.53)$$

$$\hat{x}_k = \hat{x}_k^- + \mathbf{K}_k (\tilde{y}_k - \mathbf{C}_k \hat{x}_k^-) \text{ e} \quad (2.54)$$

$$\mathbf{P}_k = (\mathbf{I} - \mathbf{K}_k \mathbf{C}_k) \hat{\mathbf{P}}_k^{-}. \quad (2.55)$$

Nas Equações 2.51-2.55, \mathbf{P}_k é uma estimativa da covariância de erro da medição, enquanto \mathbf{K}_k é denominado ganho de Kalman. Após os dois passos do algoritmo, \hat{x}_k e \mathbf{P}_k são preservados continuamente, de forma a serem utilizados na parte de predição no próximo instante de tempo [5].

No presente projeto, o filtro de Kalman é utilizado em conjunção com o controlador. Porém, é admitido um caso especial, em que as matrizes passadas para o filtro são discretizadas a partir do sistema contínuo escrito em forma canônica controlável. Nesse caso, \hat{y}_k é diretamente dependente de uma das variáveis de estado, tornando o conjunto controlador-filtro mais simples de ser implementado, uma vez que é tomada a integral de uma variável de estado, apenas.

2.3 Bancada

A bancada da ponte rolante presente no Laboratório de Automação e Controle está esquematizada na Figura 2.8. Serão detalhados os componentes desta bancada, de modo a se entender o papel de cada um deles. Observe que no esquemático falta a câmera, que é um sensor que fica de frente para a bancada.

2.3.1 Controlador Lógico-Programável

O controlador lógico-programável (CLP - ver Figura 2.9) é a espinha dorsal da bancada. Ele é responsável por executar os comandos de controle sobre todos os elementos que estão conectados a ele. A programação do CLP é realizada pelo computador; porém, uma vez feito o *download* do programa ao CLP, a execução ocorre independentemente do computador, contanto que o CLP esteja em modo de execução.

O CLP utilizado é fabricado pela *Allen Bradley*, modelo Logix5560M03SE. Tal modelo possui

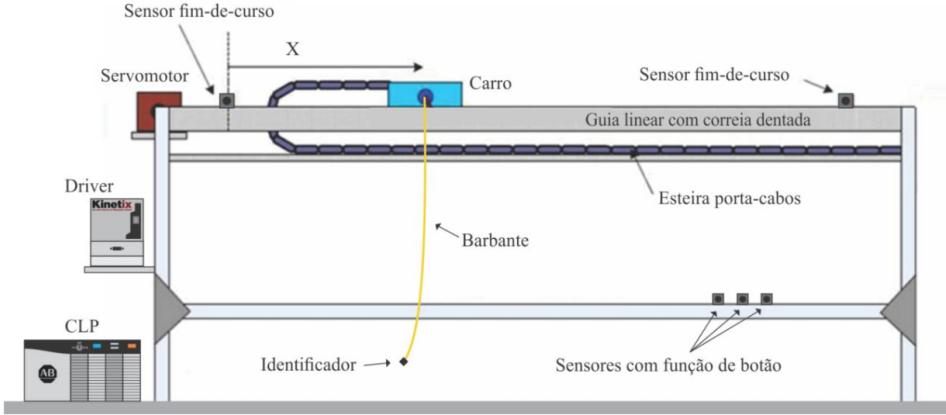


Figura 2.8: Esquemático da Bancada Utilizada para o Experimento [3]

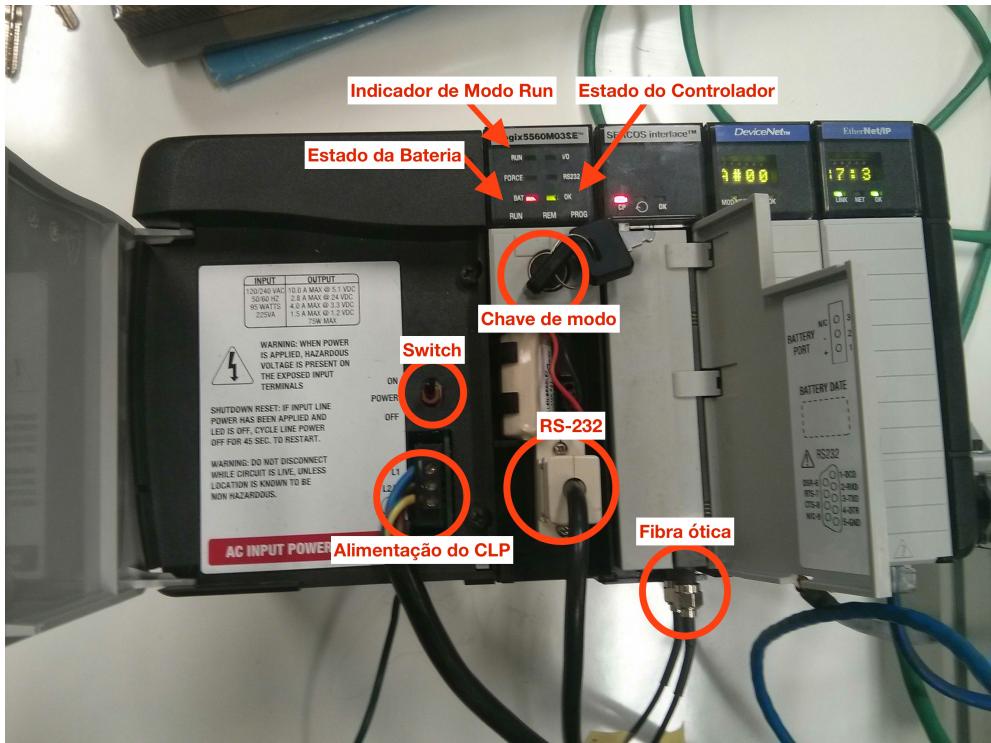


Figura 2.9: CLP com identificação de elementos

memória lógica e de dados de 750 KiB, e memória de I/O de 494 KiB. Há quatro módulos no *chassis* do controlador:

- O próprio controlador;
- *SERCOS Interface*;
- DeviceNET;
- EtherNet/IP.

Cada um desses módulos será apresentado posteriormente com mais detalhes. Além dos módulos, o controlador ainda possui um *switch* liga/desliga presente no *chassis*. No módulo Logix, há uma chave responsável por alterar o modo de funcionamento do mesmo. As posições possíveis dessa chave são:

- RUN (*Run mode*);
- REM (*Remote mode*);
- PROG (*Program mode*);

Na prática, o modo REM se divide em dois modos: REM RUN e REM PROG. A maneira de se diferenciar os dois é observar, no módulo Logix, o estado do LED indicador de modo RUN quando a chave estiver na posição REM.

No modo RUN, o controlador apenas roda o programa presente em sua memória; não há qualquer comunicação remota. No modo PROG, o controlador não roda nenhum programa; ele apenas pode receber um novo código. Nos modos REM, há a comunicação com o computador, permitindo verificar valores de variáveis de interesse e alterar, se necessário, o programa a ser rodado pelo controlador. O programa presente no controlador, em modo REM, só roda o código se estiver no modo REM RUN; se for necessário atualizar o programa, o modo deve ser o REM PROG.

Para fins de se utilizar melhor o CLP, será utilizado sempre o modo REM e suas derivações. O modo REM RUM permite a execução do programa; já o modo REM PROG permite a atualização do programa dentro da memória do CLP.

2.3.2 SERCOS Interface

Sercos é um barramento digital de automação que interconecta controladores, *drives*, dispositivos de entrada/saída e atuadores para máquinas e sistemas controlados numericamente. Foi projetado para comunicação serial de alta velocidade de dados em sistemas de tempo real por meio de fibra ótica (Sercos I & II) ou um cabo Ethernet Industrial (Sercos III). Sercos é um padrão internacional [14].

Num sistema Sercos, todas as malhas que contém servomotores são normalmente fechadas no *drive*. Isto reduz a carga computacional no CLP, permitindo-o sincronizar mais eixos do que conseguiria caso contrário. Além disso, fechar a malha dos servomotores com o *driver* ajuda a reduzir o efeito do atraso de transporte entre o controle de movimento e o *driver* [14].

Nesta bancada, o CLP deve se comunicar com o servomotor (MPL-A310F-SJ22AA) através do *drive* Kinetix (2094-AC05MP5), de forma a movimentar o carrinho segundo uma trajetória planejada ou segundo uma lei de controle em malha fechada executando no CLP. Essa comunicação se dá por um par de fibras óticas full-duplex, conforme se observa na Figura 2.9, o que caracteriza uma rede Sercos I ou II.

2.3.3 Line Interface Module 2094-AL09

Este módulo não apareceu no esquemático da Figura 2.8, mas ele tem a função essencial de interfacear a rede trifásica com o servo *drive*, permitindo o acionamento do motor. Na Figura 2.10, se observa que há três conjuntos de disjuntores nesse módulo: CB1 – liga ou desliga a rede trifásica do *drive* –, CB2 – fornece tensão monofásica ao servo *drive* – e CB3 – liga as duas fontes de tensão DC de 24V –, responsáveis pelas entradas e saídas digitais do módulo e alimentação do freio do motor [3].

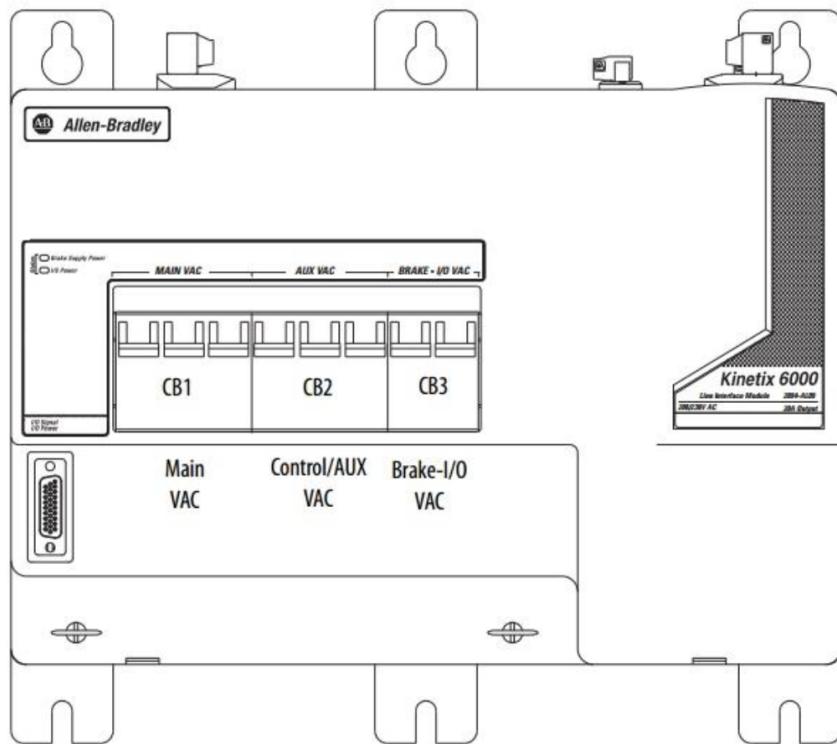


Figura 2.10: Line Interface Module modelo 2094-AL09 da Allen Bradley [3]

2.3.4 Drive Kinetix 6000 da Allen Bradley

Por meio da Interface Sercos, o CLP comanda o drive que então fornece potência ao motor. O drive controla o servomotor por meio de pulsos PWM. A Figura 2.11 apresenta um Drive Kinetix 6000 da Allen Bradley similar ao utilizado no laboratório. Mais detalhes sobre o funcionamento do drive estão disponíveis em [3] e [15].

2.3.5 Servomotor

Um servomotor é um atuador rotatório que permite controle preciso da posição angular. O motor consiste de um motor acoplado a um sensor para a realimentação de posição/velocidade. Também é necessário um drive servo para completar o sistema. O *drive* usa o sensor de realimentação para controlar precisamente a posição angular do motor, ou seja, é uma operação em



Figura 2.11: Drive Kinetix 6000 da Allen Bradley

malha fechada. Assim, usando servomotores em malha fechada, tem-se uma alternativa de alto desempenho aos motores de passo e de indução [16].

O servomotor MPL-A310F-SJ22AA da *Allen-Bradley* foi utilizado e está representado na Figura 2.12. O mesmo é composto por um motor indutivo de tensão nominal $230V_{ac}$ e um encoder do tipo StegmanHiperface, que mede posição de forma absoluta e velocidade de forma incremental [3].



Figura 2.12: Servomotor modelo MPL-A310F-SJ22AA

2.3.6 Sensores indutivos

Os sensores indutivos são elementos detectores de presença, particularmente de objetos metálicos. Eles funcionam através da variação de campo magnético ocasionada pela presença do objeto a ser identificado. Tal variação de campo magnético provoca uma variação de corrente dentro do sensor, alterando seu estado.

Na presente bancada, há 6 sensores indutivos da família 871TM, similares ao da Figura 2.13, fabricados pela *Allen-Bradley*. Eles são alimentados com tensão de 24 V, que está dentro dos

limites padrão. São sensores feitos de aço, adaptados a ambientes industriais.



Figura 2.13: Sensor indutivo 871T-R8B18 [3]

A importância desses sensores é enorme. O motivo é que a câmera às vezes falha para obter a posição atual e tem um limite de frequência devido ao processamento interno que ela tem de realizar. Com os sensores indutivos, tem-se um processamento rápido para identificar se o motor está na região próxima dos limites. Assim, a rotina de segurança que trava o motor depende diretamente desses sensores indutivos, como mostra o Anexo I.5.2.

2.3.7 Câmera PresencePlus

A câmera PresencePlus P4 GEO da Banner Engineering foi utilizada no projeto - veja Figura 2.14. Esta câmera é um sensor robusto utilizado em ambientes industriais com fácil utilização. Sua programação é feita em software próprio da Banner e é visual.

Dentre as capacidades da câmera, nota-se que ela pode capturar até 24 imagens por segundo. No entanto, além da captura das imagens há o processamento das imagens, que pode ser feito na própria câmera ou em um módulo externo da própria Banner, que também consome certo tempo. O programa utilizado no projeto está disponível no Anexo I.7.1.



Figura 2.14: Câmera da Banner Engineering utilizada no projeto [3]

2.4 Redes Utilizadas

2.4.1 DeviceNET

DeviceNET é um protocolo de baixo nível da camada de aplicação, voltado a ambientes industriais [17]. Ele suporta comunicação entre dispositivos de baixo nível, como sensores e atuadores, e dispositivos de alto nível, como o computador e o CLP [18]. Permite também uma rápida configuração entre dispositivos a *byte*, suportando tanto dispositivos analógicos quanto digitais [3]. As velocidades de transmissão são de até 500 kbps, sendo bem mais lenta que uma rede Ethernet.

No experimento, a rede DeviceNET é utilizada para a conexão entre o CLP e os sensores indutivos presentes na planta [3]. Para fins de detecção de fim-de-curso, será utilizado o modo digital dos sensores, uma vez que apenas é necessário verificar se o carrinho está na área de detecção do sensor. O módulo responsável pelo gerenciamento da rede DeviceNET é o 1756-DNB.

2.4.2 Ethernet/IP

A rede Ethernet/IP, introduzida em meados de 2001 pela ODVA (*Open DeviceNet Vendor Association*) [19], é um tipo de rede Ethernet voltada ao ambiente industrial, seguindo o CIP, assim como o DeviceNET [20]. É uma rede robusta, organizada segundo o modelo OSI de 7 camadas, que permite conexão com dispositivos conectados a redes Ethernet padrão. Ela permite a passagem de dados via pacotes TCP ou UDP e é indicada em aplicações que exigem uma transferência rápida e confiável de dados, principalmente em aplicações de tempo real [21].

No presente experimento, a rede Ethernet/IP é utilizada para receber dados de inspeção da câmera, notadamente a posição horizontal da bola de isopor presa ao barbante. Além disso, ela é utilizada na transferência de programas entre o computador e o CLP (através do módulo Ethernet 1756-ENBT/A), uma vez que ela provê uma comunicação mais rápida do que o RS-232. O computador também pode enviar e receber dados por essa rede utilizando OPC – para saber mais sobre OPC, ver subseção 2.4.3.

2.4.3 OPC - *OLE for Process Control*

A rede OPC, cuja primeira versão foi lançada em 1996, é uma rede voltada ao ambiente industrial. Essa rede é responsável, em termos gerais, pela troca confiável de dados entre o CLP, computadores e outros dispositivos. O objetivo do uso de tal rede é abstrair o protocolo de rede utilizado no CLP (seja ModBus, Profibus, Ethernet/IP, etc.) em uma interface padrão de forma que sistemas HMI/SCADA possam se comunicar com um “intermediário” que converta as operações de dados de OPC em operações específicas de dispositivo e vice-versa [22].

Uma das características da rede OPC é permitir a troca de dados entre o CLP e o computador por meio de um sistema *server/client*. No presente projeto, o *server* é disponibilizado pelo *software* RSLinx, enquanto que o cliente é um programa desenvolvido externamente ao CLP. Para esse programa, foi escolhida a linguagem Python; a razão da escolha de tal linguagem foi, além da

existência de módulos programados por terceiros para lidar com a rede OPC, o fato de Python ter outros módulos que facilitam a implementação de controladores e ser razoavelmente rápido.

Para se programar o *client* da rede OPC em Python, foi utilizado o módulo OpenOPC [23]. Esse módulo é bem simples de se utilizar; as funções de trocas de dados OPC são abstraídas de modo a utilizar elementos simples da linguagem, como o uso de dicionários, que são estruturas nas quais se relacionam chaves com valores.

A utilização do OPC permite realizar um sistema de controlador no qual o computador auxilia nos cálculos, assim como permite salvar dados do CLP para posterior desenho dos gráficos.

2.5 Programação do CLP

2.5.1 Visão geral

Programação, em termos gerais, é aplicada na resolução de problemas. Em particular, a programação de CLPs busca resolver, no âmbito industrial, problemas relacionados à automação de processos. Essa resolução de problemas segue uma metodologia, de forma a se direcionar o projeto. Antes de se proceder à programação, faz-se necessário seguir os seguintes passos [24]:

1. Descrição do problema;
2. Detalhamentos e melhoria do processo;
3. Especificação dos atuadores e sensores da planta;
4. Elaboração do algoritmo;
5. Representação gráfica do algoritmo, quando aplicável;
6. Esquema funcional, quando aplicável;
7. Seleção dos módulos do controlador;
8. Programação, utilizando linguagens suportadas.

Ao se programar o CLP, deve-se ter em conta que o controlador executa sempre três ciclos [24]:

1. *Scan* de Entrada: Ciclo em que o controlador recebe todos os dados de seus módulos de entrada;
2. *Scan* do Programa: Ciclo em que o controlador processa as entradas recebidas e gera as saídas;
3. *Scan* de Saída: Ciclo em que o controlador envia os dados de saída para os módulos de saída.

Para o presente experimento, entre as várias linguagens disponíveis para CLPs, foram selecionadas duas: uma linguagem gráfica, o ***ladder***, e uma linguagem textual, o **texto estruturado**. Ambas foram selecionadas por serem linguagens muito utilizadas, normalmente rápidas e que ocupam pouca memória, ao contrário de linguagens como SFC (*Sequential Flow Chart*) e FBD (*Function Block Diagram*), que também poderiam ser utilizadas.

2.5.2 Linguagem *ladder*

2.5.2.1 Introdução ao *ladder*

A linguagem *ladder* é uma linguagem de programação gráfica, e uma das primeiras a ser utilizada na programação de CLPs. Seu nome vem do inglês *ladder*, que significa escada; nome dado em razão dos programas, ao serem feitos, assumirem a forma de uma escada, e lidos de cima para baixo (movimento de descida). Essa linguagem foi estruturada de forma a ter uma simbologia semelhante à de um diagrama de conexão de relés, que eram utilizados em indústrias antes do CLP.

Todo programa *ladder* possui duas linhas verticais e, entre elas, uma ou mais linhas horizontais nas quais são especificados os comportamentos do programa. A linguagem possui várias instruções, em sua maioria comuns entre diferentes tipos de CLP. A Tabela 2.3 mostra as principais instruções utilizadas em *ladder*, utilizando a sintaxe presente no software RSLogix5000.

Tabela 2.3: Principais instruções *ladder*

Desenho da Instrução	Nome da Instrução	Descrição
-(-)-	OTE	Atualiza variável booleana de acordo com a condição da linha
-(L)-	OTL	Atribui valor verdadeiro à variável booleana se a linha for alimentada
-(U)-	OTU	Atribui valor falso à variável booleana se a linha for alimentada
-[]-	XIC	Examina se a variável possui valor verdadeiro
-[/]-	XIO	Examina se a variável possui valor falso

2.5.2.2 Instruções específicas

No presente trabalho, a presença do servomotor exige uma lógica e entradas que não são booleanas, ou seja, que não podem ser tratadas pelos operadores descritos na Tabela 2.3. Portanto, faz-se necessário o uso de novas instruções, diretamente relacionadas com o movimento do atuador. Tais instruções são conhecidas, dentro do RSLogix5000, como *Motion Control Instructions*, ou instruções de controle de movimento. A Tabela 2.4 mostra as instruções desse tipo utilizadas no experimento.

As funções descritas pela Tabela 2.4 lidam com outros tipos de variáveis, como números reais,

Tabela 2.4: Principais instruções de controle de movimento em *ladder*

Nome da Instrução	Sigla	Função
Motion Servo On	MSO	Inicializa o servomotor, travando a esteira.
Motion Axis Jog	MAJ	Envia comandos de velocidade e sentido de rotação ao motor, executando sua movimentação.
Motion Axis Stop	MAS	Interrompe a movimentação do motor; a esteira permanece travada.
Motion Servo Off	MSF	Desativa o servomotor, destravando a esteira e permitindo a movimentação manual do carrinho.

para a velocidade do motor, e até mesmo estruturas que o *software RSLogix* cria para armazenar informações sobre os eixos e o controle feito pelas instruções. Tais parâmetros podem ser alterados dentro dos blocos das funções. A Figura 2.15 mostra um exemplo de tais instruções.



Figura 2.15: Exemplo de programa *ladder* com instruções de movimentação.

2.5.3 Texto Estruturado

2.5.3.1 Visão geral do texto estruturado

Além de *ladder*, que é uma linguagem gráfica, o CLP utilizado no experimento também suporta uma linguagem puramente textual, o texto estruturado. Essa linguagem é uma linguagem sequencial, que lembra muito linguagens como BASIC, C ou Pascal. Cada comando é executado em sequência, salvo em caso de desvios (estruturas do tipo *if-then-else*) ou repetições (*loops*, estruturas do tipo *while*).

Em relação ao *ladder*, o texto estruturado possui vantagens e desvantagens. A principal vantagem é o fato do texto estruturado se assemelhar às linguagens de programação mais utilizadas; isso torna tal linguagem de fácil aprendizado. Além disso, por ser texto, caso seja necessário editar muitos parâmetros cuja criação seja de difícil automação, um programa auxiliar escrito em linguagens de programação genéricas pode simplificar a geração do texto estruturado. Porém, uma desvantagem do texto em relação ao *ladder* reside nas funções de movimentação, uma vez que, ao contrário do *ladder*, o texto estruturado não dá uma indicação clara associando um valor

e a variável que ele representa, além da ordem dos parâmetros. Isso torna o uso de funções como MAJ (vide Tabela 2.4) um pouco mais complicado em texto estruturado.

2.5.3.2 Instruções de movimentação do motor para o texto estruturado

Assim como o *ladder*, o texto estruturado permite o uso das funções descritas na Tabela 2.4. A sintaxe de uso dessas funções lembra muito a sintaxe da linguagem C, com o nome da função e uma lista de parâmetros válidos separados por vírgulas. A seguir, um exemplo de uso:

```
1  /**
2  * MS0: ativa o servo cujo eixo eh descrito
3  * por drive_axis; informacoes de controle
4  * sao gravadas em MS0_1
5  */
6  MS0(drive_axis,MS0_1);
7  /* Atribui o valor 0.0 ao primeiro elemento do array speed */
8  speed[0] := 0.0;
9  /* Atribui 1 para dataInitialized */
10 dataInitialized := 1;
```

Nota-se que o código apresentado, a menos da operação de atribuição, possui uma sintaxe muito semelhante à de C. A função MS0 acima recebe dois parâmetros, assim como em *ladder*; porém, a única indicação de como os parâmetros se comportam é a ordem com que eles são colocados; no caso de MS0, o 1º parâmetro é o eixo que será inicializado, e o 2º parâmetro é uma *tag* que guarda as informações de controle executadas por MS0.

2.6 Parâmetros para gerar trajetórias

Fabrício et al [7] desenvolveram um método para gerar trajetórias *offline* (controle malha aberta) assim como *online* (controle malha fechada). Para se utilizar do método, é necessário ter parâmetros do sistema. Se fosse necessário simular ou controlar o sistema real, utilizariam-se os dados da Tabela 2.5, conforme cita Rédyton [3].

Como é importante se validar o sistema, mas nem sempre isso é possível em escala real devido ao altíssimo custo e aos riscos envolvidos, é necessário um conjunto equivalente de parâmetros tais que permita validar o sistema em escala laboratorial. O *riser* é representado por um barbante e a água é trocada pelo ar. Uma massa na ponta foi adicionada e os dados resultantes estão na Tabela 2.6. Esses são os dados que serão utilizados para os experimentos, assim como as Tabelas 2.1 e 2.2.

Tabela 2.5: Dados para simulação em escala real [3]

Dados do Riser	
Diâmetro externo	0.55m
Diâmetro interno	0.5m
Comprimento	2000m
Módulo de Elasticidade	200GPa
Densidade	7860kg/m ³
Dados do fluido (Água)	
Densidade do fluido	1000kg/m ³
Viscosidade dinâmica	10^3 Pa · s

Tabela 2.6: Dados para simulação em escala laboratorial

Dados da Massa na Ponta - Isopor	
Diâmetro Externo	30.6mm
Densidade	10kg/m ³
Dados do Riser (Barbante)	
Diâmetro externo	2mm
Diâmetro interno	0mm
Comprimento	82cm
Módulo de Elasticidade	2.1MPa
Densidade	191kg/m ³
Dados do fluido (Ar)	
Densidade do fluido	1.2754kg/m ³
Viscosidade dinâmica	$17.2 \cdot 10^6$ Pa · s

Capítulo 3

Resultados

Este capítulo apresenta resultados de simulação, os procedimentos experimentais realizados e seus resultados.

3.1 Câmera

Os problemas com a câmera se resumiram à configuração da rede, à calibração e à programação. O mais difícil inicialmente foi a programação, mas se tornou bem simples depois.

3.1.1 Configuração da Rede

A câmera estava com um *firmware* antigo e foi necessário obter o arquivo 2014R1B PresencePLUS Firmware¹ que é o programa de atualização da *Banner Engineering* para esta câmera. Bastou executá-lo no computador, estando a câmera conectada ao *switch* do laboratório assim como o computador estava, ambos por cabos Ethernet. Antes desta atualização, o *software* da câmera não permitia selecionar a opção Ethernet/IP de forma a permitir utilizar o módulo Ethernet/IP do CLP.

Uma vez atualizado o *firmware*, foi obtido o arquivo EDS² da câmera que foi então integrado ao *software* da Rockwell no computador por meio do programa *EDS Hardware Installation Tool* da própria Rockwell, parte do RSLinx.

Após os procedimentos anteriores, adiciona-se a câmera como um módulo genérico usando o *software* RSLogix, conforme instruções da Banner Engineering [25]. Foi também necessário utilizar o RSLinx para verificar se a câmera estava conectada ao computador e o *software* da PresencePlus para identificar a câmera pelo endereço IP, uma vez que a rede utilizada é Ethernet/IP. A Figura 3.1 mostra a janela do RSLinx com a câmera reconhecida.

¹ Atualização de Firmware da Câmera - http://info.bannerengineering.com/_dav/cs/idcplg?IdcService=GET_FILE&RevisionSelectionMethod=LatestReleased&dDocName=B_4170042. Acesso em 29/11/2015.

² Arquivo EDS disponível em <http://www.bannerengineering.com/en-US/products/sub/78#ui-tabs-37>. Acesso em 29/11/2015.

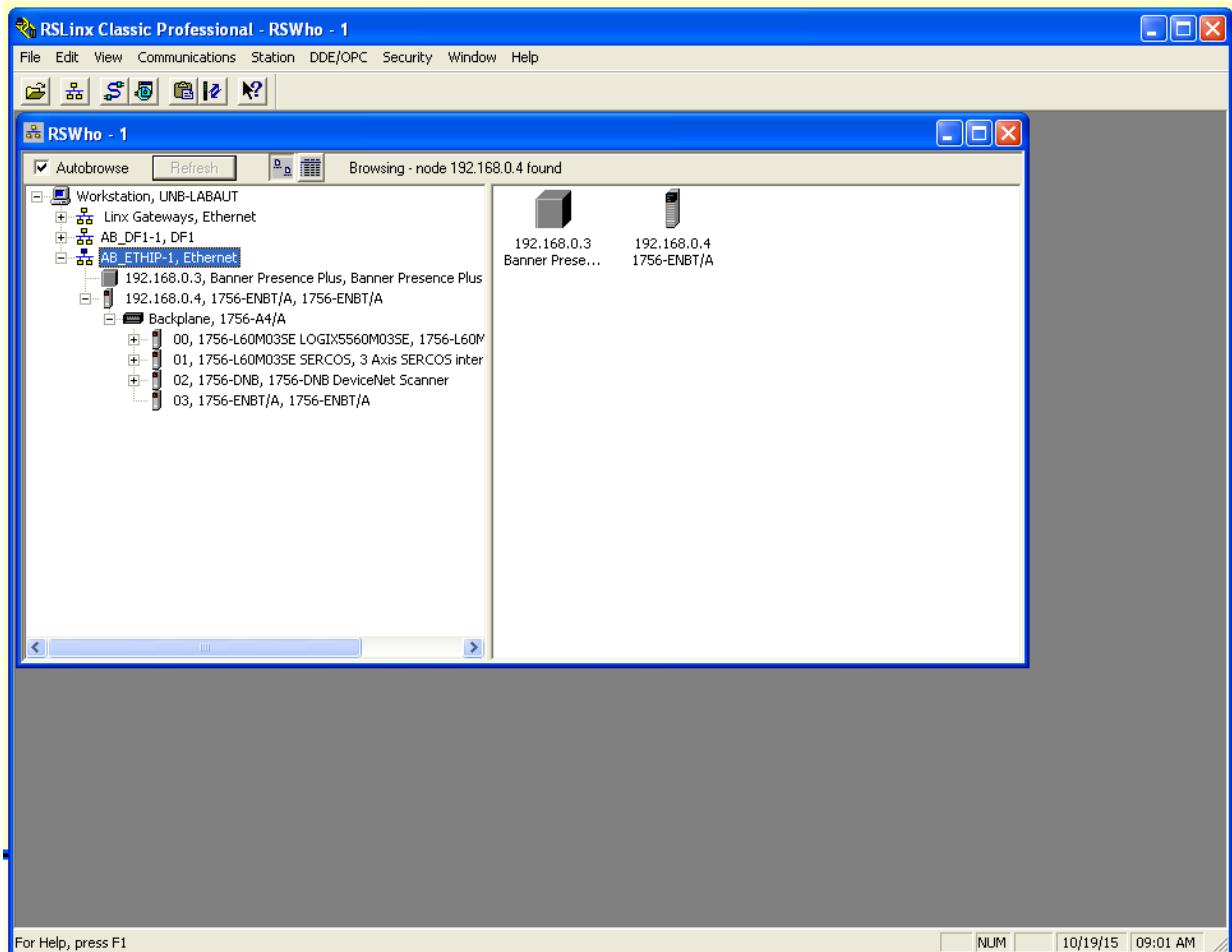


Figura 3.1: RSLinx com câmera reconhecida

3.1.2 Calibração

A câmera permite fazer medidas de distâncias em *pixels*. De forma a se converter essa distância para milímetros, uma barra de alumínio com marcas e tamanho conhecido é utilizada. É importante primeiro calibrar o sistema, para se saber se há deformação de pixels significante ao longo da distância de interesse, nomeadamente o tamanho da barra de alumínio sendo utilizado, cerca de 532mm. No PresencePlus P4 GEO 1.3, um programa é feito, com imagem de referência conforme Figura 3.2, que usa várias ferramentas de detecção de borda (ferramenta **Locate**) para identificar as posições de cada uma das marcas pretas da barra. Seis marcas foram feitas e a Tabela 3.1 apresenta os resultados para cada seção. A distância entre duas marcas é de 10cm, com exceção da distância entre P0 e PEND que é o comprimento total da barra.

O maior desvio da quantidade de milímetros por *pixel* das seções em relação à da barra inteira é de aproximadamente 4.93%. Há algumas imprecisões na maneira como os traços foram desenhados e é possível que o erro seja menor.

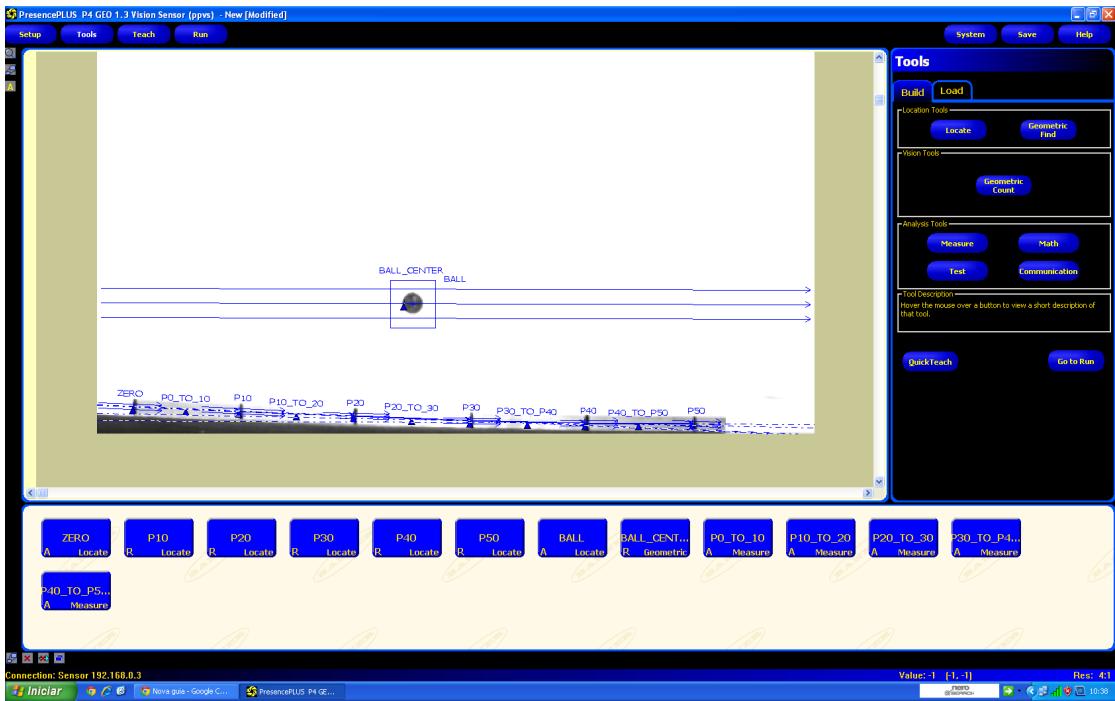


Figura 3.2: Programa PresencePLUS para calibração da câmera

Tabela 3.1: Relações mm/px para diferentes seções da barra de alumínio

Seção 1	Seção 2	Distância (px)	mm/px
P0	P10	160	0.625
P10	P20	173	0.578
P20	P30	176	0.568
P30	P40	173	0.578
P40	P50	163	0.613
P0	PEND	893	0.596

3.1.3 Programação

A programação da câmera se inicia obtendo uma imagem de referência e adicionando-se ferramentas de detecção de pontos de interesse, como mostra a Figura 3.2. A ferramenta **Locate** é responsável por detectar as bordas da bolinha, e a ferramenta **Geometric** detecta o centro da mesma. Também é possível adicionar ferramentas que fazem operações matemáticas (ferramenta **Math**), executam medições (ferramenta **Measure**), assim como as que enviam dados pela rede (ferramenta **Communication**), o que é essencial para se comunicar com o CLP.

3.2 Calibração do Servomotor

O RSLogix tem o bloco **MAJ – Motion Axis Jog** – que permite alterar a velocidade do motor enquanto ele se movimenta. No entanto, o bloco espera que a entrada tenha unidade [u/s] (unida-

des por segundo) ao invés de alguma unidade no SI tal como [mm/s]. Devido a isso, foi necessária uma calibração do sistema. O procedimento era anotar a posição inicial x_0 , definir um tempo Δt no qual o carrinho se movimenta a uma velocidade v em [u/s] e, após o movimento, registrar a posição final x_f . As posições inicial e final são medidas em milímetros. Com isso, calculava-se a velocidade em [mm/s]. Alguns ensaios são realizados seguindo esse procedimento e a média é considerada como o valor de uma unidade, resultando em aproximadamente 71.32mm. Os dados de calibração estão na Tabela 3.2.

Tabela 3.2: Dados de calibração do servomotor, média obtida é de 71.32 mm/unidade

x_0 - [mm]	x_f - [mm]	Δt - [s]	Velocidade - [u/s]	Velocidade - [mm/s]	mm/u
2	71.8	2	0.5	34.9	69.8
6	76.1	2	0.5	35.05	70.1
6	188	5	0.5	36.4	72.8
6	185	2.5	1	71.6	71.6
6	77	10	0.1	7.1	71
6	296.5	20	0.2	14.525	72.625

3.3 Modelo no Espaço de Estados

A partir da teoria apresentada na Subseção 2.1.3, foram desenvolvidas algumas rotinas em linguagem Julia [8] para obtenção das matrizes reduzidas. Na rotina, a ordem da matriz de saída é um parâmetro, mas para os propósitos deste projeto só foi testado o sistema reduzido a quatro estados. A Seção I.1 dos anexos apresenta o código completo.

As matrizes obtidas para o modelo reduzido são dadas

$$\mathbf{A}_R = \begin{bmatrix} -0.0881 & -3.8389 & 0 & 0 \\ 3.8389 & -0.0881 & 0 & 0 \\ 0 & 0 & -0.1061 & -10.8145 \\ 0 & 0 & 10.8145 & -0.1061 \end{bmatrix}, \quad (3.1)$$

$$\mathbf{B}_R = 10^3 [0.3313, 0.0066, 1.3549, 0.0163]^T,$$

$$\mathbf{C}_R = [-0.0003, 0.0148, 0.0001, -0.0029] \text{ e}$$

$$D_R = -0.0906,$$

sendo que é fácil observar que os autovalores da matriz \mathbf{A}_R são dados por $-0.0881 \pm 3.8389j$ e $-0.1061 \pm 10.8145j$, valores extraídos dos blocos diagonais dessa matriz. Esse tipo de estrutura já era esperada quando se apresentou a técnica de redução modal na subseção 2.1.3. O sistema composto por essas matrizes é dado por

$$\begin{aligned} \dot{\mathbf{z}} &= \mathbf{A}_R \mathbf{z} + \mathbf{B}_R u(t) \text{ e} \\ y &= \mathbf{C}_R \mathbf{z} + D_R u(t). \end{aligned} \quad (3.2)$$

No sistema da Equação 3.2, ainda não se compensou pela transferência direta diferente de zero. Note que ela era zero antes da redução, mas a perca do ganho estático dos outros modos do

sistema levou a esse D_R não nulo. Quando se analisa a resposta ao degrau em malha aberta para este caso, Figura 3.3, nota-se que o atraso é cerca de 0.3s. Para ser preciso, $\epsilon = 0.313s$. Com esse valor de atraso, calculam-se novas matrizes B_D e D_D para substituir B_R e D_R , respectivamente, conforme Equações 2.38 e 2.39. Utilizando exatamente esse atraso de 0.313s, a transferência direta se tornaria aproximadamente zero. No entanto, esses modelos no espaço contínuo serão discretizados com período $T_s = 0.1s$. Daí, o ϵ mais próximo realizável é 0.3s e seu uso resulta nas novas matrizes

$$\begin{aligned} \mathbf{B}_D &= 10^3 [0.1255, 0.2974, -1.3039, -0.1504]^T \text{ e} \\ D_D &= -0.0685, \end{aligned} \quad (3.3)$$

que então fazem parte do novo sistema reduzido dado por

$$\begin{aligned} \dot{\mathbf{z}} &= \mathbf{A}_R \mathbf{z} + \mathbf{B}_D u(t - \epsilon) \text{ e} \\ y &= \mathbf{C}_R \mathbf{z} + D_D u(t - \epsilon). \end{aligned} \quad (3.4)$$

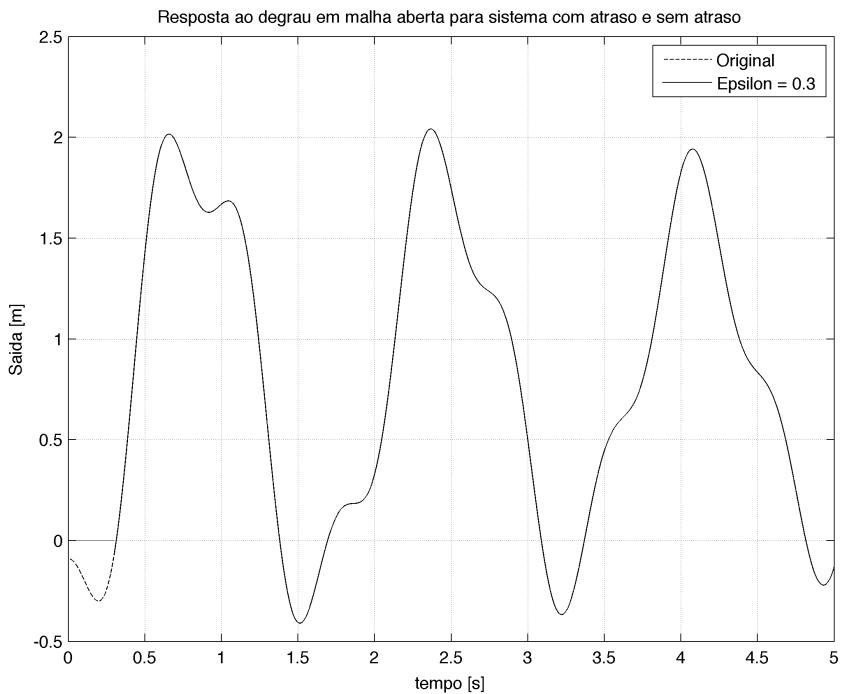


Figura 3.3: Resposta ao degrau para modelos reduzidos com atraso e sem atraso

Apesar de $\epsilon = 0.3s$ ser próximo de 0.313s, esse atraso não reduz muito a transferência direta, como se pode observar analisando D_D , Equação 3.3.

O sistema em malha aberta oscila muito, mas é estável. Observa-se que a resposta decai conforme o tempo passa, conforme Figura 3.4. O objetivo do controle é reduzir ao máximo essas oscilações, tendo uma trajetória o mais suave possível de um ponto inicial a um ponto final.

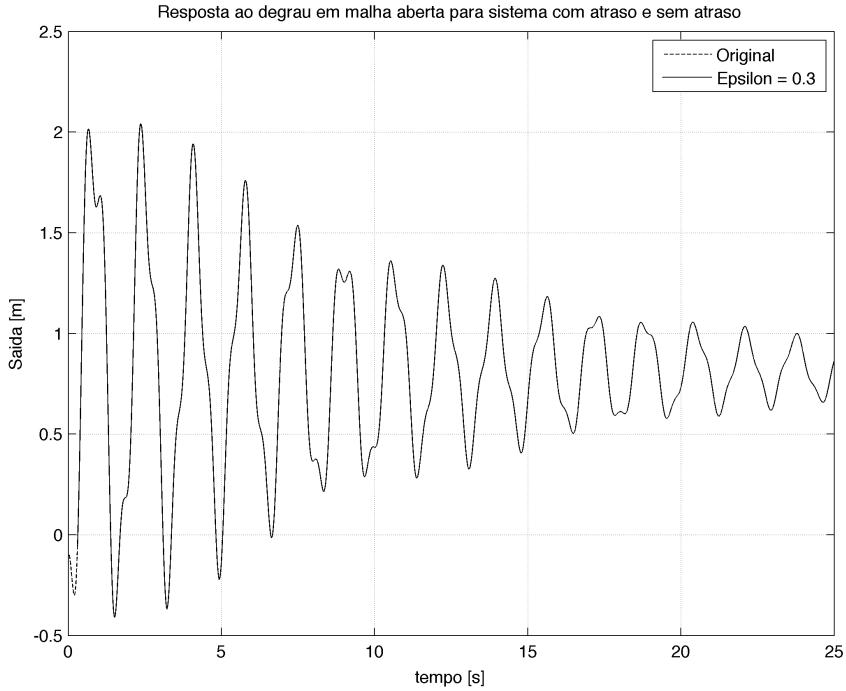


Figura 3.4: Resposta ao degrau para modelos reduzidos com atraso e sem atraso, 25 segundos

3.3.1 Discretização

O controle será feito conforme a Seção 2.2, mas primeiro deve-se discretizar o sistema. A função `c2d` [13] do MATLAB é utilizada, resultando nas matrizes

$$\begin{aligned} \mathbf{A} &= \begin{bmatrix} 0.9191 & -0.3712 & 0 & 0 \\ 0.3712 & 0.9191 & 0 & 0 \\ 0 & 0 & 0.4651 & -0.8733 \\ 0 & 0 & 0.8733 & 0.4651 \end{bmatrix}, \\ \mathbf{B} &= [6.5818, 31.2517, -98.5991, -75.6695]^T, \\ \mathbf{C} &= [-0.0003, 0.0148, 0.0001, -0.0029], \\ D &= -0.0685. \end{aligned} \quad (3.5)$$

Os polos de \mathbf{A} são $0.9191 \pm 0.3712j$ e $0.4651 \pm 0.8733j$. Como esses polos estão dentro do círculo unitário, o sistema continua estável, apesar de manter as oscilações, pelo fato dos polos terem parte imaginária. Vale notar que as matrizes \mathbf{C} e D são iguais a \mathbf{C}_R e D_D do sistema contínuo, respectivamente, conforme Equações 3.1 e 3.3.

3.3.2 Controle

A parte mais difícil do controle é a escolha dos polos. Algumas simulações foram realizadas e observava-se se o sistema era estável e se oscilava muito. O projeto final considerou 5 polos: $[0.6, 0.6, 0.6, 0.5 \pm 0.4j]$. O primeiro polo se deve ao integrador que foi adicionado ao sistema.

Os polos com parte imaginaria foram escolhidos baseados nos polos originais do sistema. A ideia é evitar forçar muito esses polos para longe, evitando que os ganhos sejam altos e economizando em atuação. Os ganhos obtidos para esse caso são dados por

$$\mathbf{K}_P = [0.0080, 0.0096, -0.0046, -0.0004], \quad K_i = 0.1953. \quad (3.6)$$

O código para calcular os ganhos está disponível na Seção I.3.

3.4 Simulação

Antes de se implementar o controle na planta, é essencial a realização de simulações. Elas são rápidas de serem realizadas e permitem averiguar se um projeto é estável ou não. As simulações aqui realizadas incluem o sistema em malha aberta com entrada tipo rampa e a entrada suave, utilizada por [4]. Depois da apresentação dos resultados em malha aberta, são apresentados os resultados em malha fechada sem considerar ruído. Por fim, os resultados com um ruído somado são apresentados.

O resultado da simulação em malha aberta é apresentado nas Figuras 3.4 e 3.4. Nota-se que a resposta à rampa teve diversas oscilações que demoram a atenuar-se. Já a entrada calculada por [4] teve uma ótima resposta. A desvantagem da implementação em malha aberta é que ela não resiste a perturbações e essas sempre ocorrem, ainda mais quando se considera o fundo do oceano.

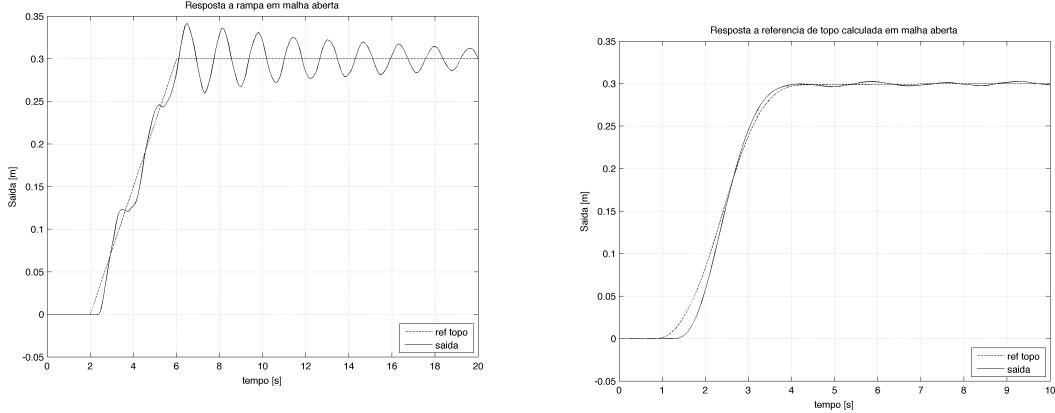


Figura 3.5: Resposta do Sistema em Malha Aberta para Excursão de 30cm, entrada rampa

g

Figura 3.6: Resposta do Sistema em Malha Aberta para Excursão de 30cm, entrada suave calculada

A solução para lidar com perturbações é fechar a malha. Para isso, uma vez obtidos os ganhos, Equação 3.6, fez-se o projeto do sistema em malha fechada em Simulink [26]. A Figura 3.7 apresenta os principais blocos e suas conexões. Nos somadores, as referências para a posição de topo, que é a posição do carrinho, e para a posição de fundo, que é a posição da bolinha, são as

entradas do sistema. O bloco **PLANTA** é o modelo sem redução modal. No caso deste projeto, a matriz do sistema sem redução de ordem é de dimensão 400×400 . Os blocos **Signal Builder** representam um sinal tipo rampa definido para comparação com as trajetórias suaves de topo e fundo [4].

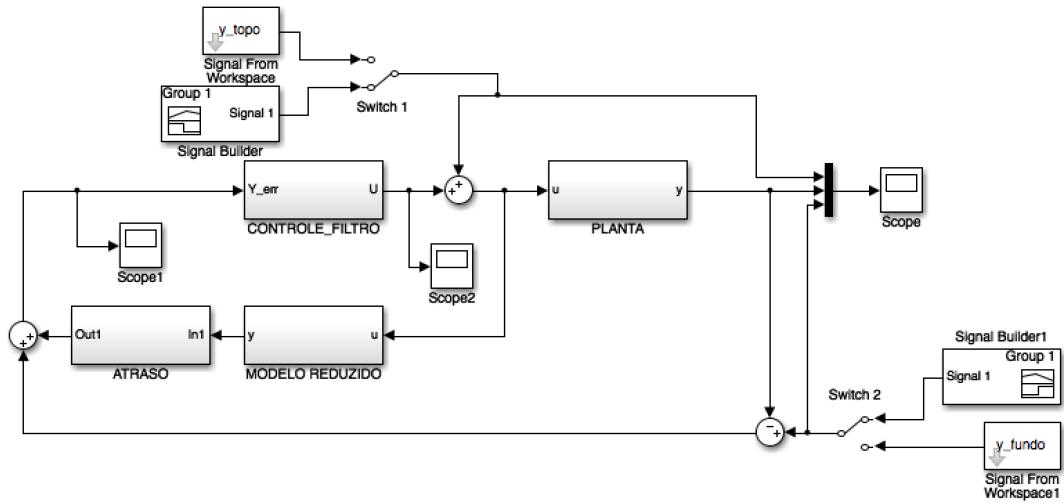


Figura 3.7: Esquema principal para simulação em Simulink

O bloco atraso, Figura 3.8, tem como saída o valor predito pelo modelo reduzido menos esse mesmo sinal atrasado em 3 períodos de amostragem, uma vez que $\epsilon \simeq 3T_s$. Esse valor é somado à diferença entre a referência de fundo e a saída da planta para então ser a entrada do bloco de controle. O bloco **CONTROLE_FILTRO** é detalhado na Figura 3.10 e utiliza os ganhos da Equação 3.6. O filtro de Kalman utilizou, como matrizes de covariância, $Q = 0.01^2 I_4$ e $R = 0.4^2$; tais valores foram obtidos de forma empírica.

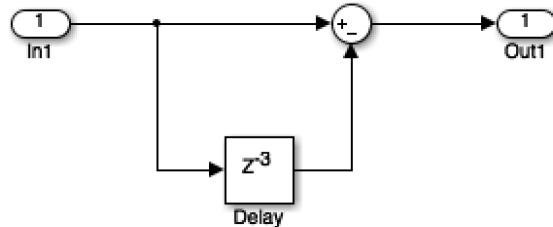


Figura 3.8: Bloco de Atraso – saída é o valor antecipado menos um valor antigo

O resultado das simulações sem ruído é apresentado nas Figuras 3.4 e 3.4. Nota-se uma ótima melhoria na resposta à rampa, enquanto a resposta à entrada suave se mantém boa.

Algo essencial é analisar a resistência ao ruído. Ela depende não só da malha fechada, mas também do filtro de Kalman que foi implementado. Os parâmetros Q e R escolhidos anteriormente são essenciais para uma boa atenuação de ruídos. O ruído foi inserido na simulação conforme a Figura 3.13. O que se nota é que o sinal de entrada do bloco de controle sempre terá algum erro,

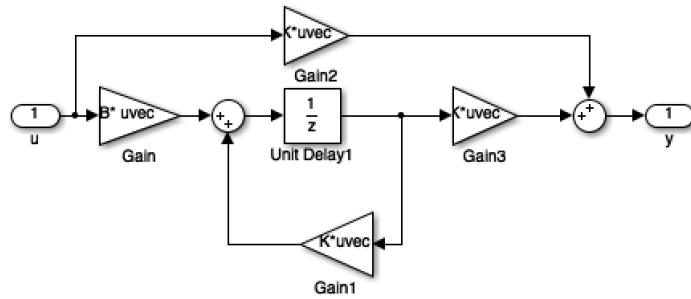


Figura 3.9: Modelo reduzido – não utiliza atraso, utilizado para predizer a saída

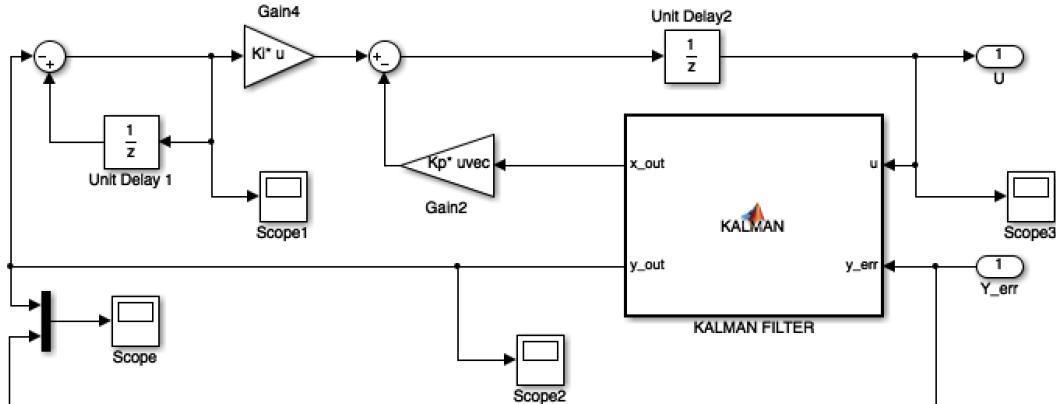


Figura 3.10: Bloco de controle com filtro de Kalman

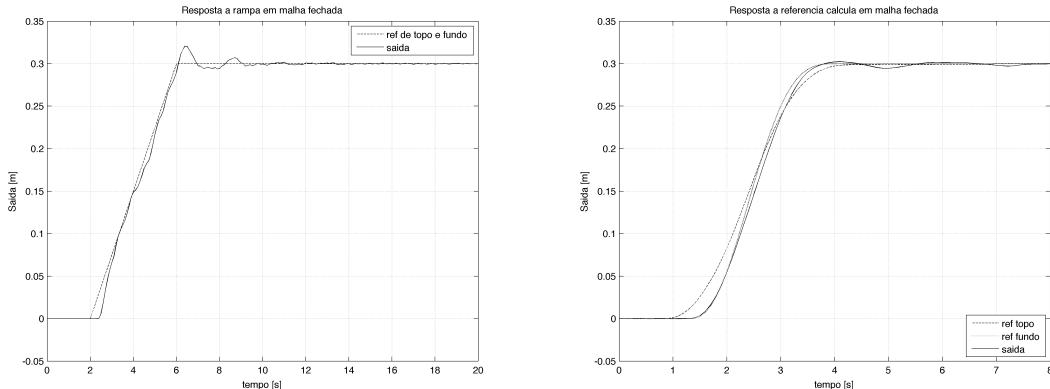


Figura 3.11: Resposta do Sistema em Malha Fechada para Excursão de 30cm, entrada rampa
Figura 3.12: Resposta do Sistema em Malha Fechada para Excursão de 30cm, entrada suave calculada para topo e fundo

o que seria um erro no sistema de medição. O resultado da simulação é apresentado na Figura 3.4. É importante observar a diferença entre o sinal de referência antes e depois de ser somado o ruído. Conforme a Figura 3.4 apresenta, esse sinal tem grandes variações comparado ao original;

mesmo assim, o resultado final está bem controlado, evidenciando a importância do controle em malha fechada com a presença do Filtro de Kalman como observador.

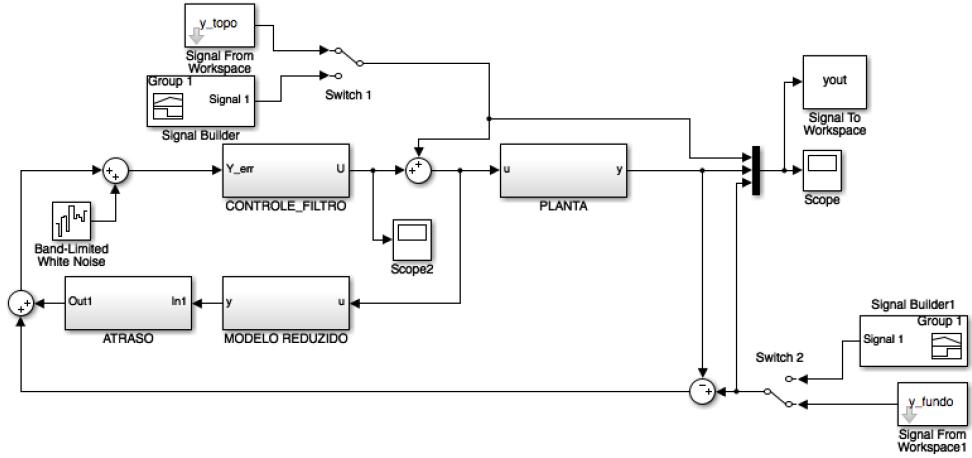


Figura 3.13: Sistema com um ruído branco adicionado

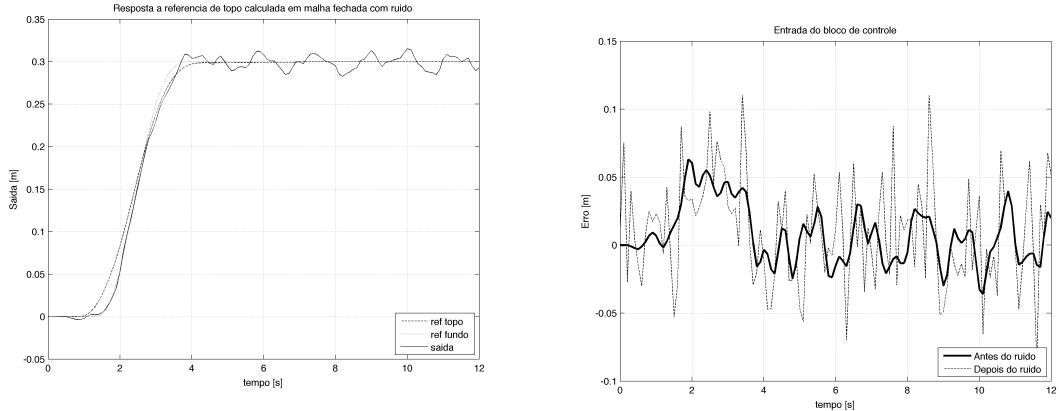


Figura 3.14: Resposta do Sistema em Malha Fechada para Excursão de 30cm, entrada suave, com ruído

Figura 3.15: Entrada do controlador antes e depois do ruído para cada instante

3.5 Resultados Experimentais

3.5.1 Considerações Iniciais

De posse das simulações feitas, assumindo-se que o sistema se comporta de forma estável e desejada com os polos escolhidos e as matrizes de covariância obtidas para o Filtro de Kalman, prosseguiu-se com a validação experimental do controle proposto. Para tal, em conjunção com os programas necessários desenvolvidos no RSLogix, foram desenvolvidos módulos e códigos em linguagem Python para implementar o controle. As três principais razões de se utilizar Python

são:

1. A linguagem Python, conforme discutido na Seção 2.4.3, possui suporte ao módulo OpenOPC [23], necessário para se efetuar a troca de informações entre o preditor de Smith e a planta por intermédio do CLP;
2. Embora haja outras linguagens e *softwares* que suportem OPC, como o próprio MATLAB, além de existirem linguagens mais rápidas e ao mesmo tempo ricas em ferramentas como o C++, o uso de Python se deve ao fato de já ter o módulo OPC e ser razoavelmente rápido, talvez não tanto quanto o C++. Porém, dada a facilidade de se programar com o OpenOPC e os tempos de comunicação e execução dos programas feitos não interferirem com o tempo de amostragem, Python se torna uma opção extremamente viável para este projeto;
3. Além do módulo OpenOPC, facilidade de programação e tempos razoáveis, Python possui também suporte à orientação a objetos. Para programar estruturas como o preditor de Smith, tal abordagem é muito importante, visto que cada componente pode ser tratado como um objeto, facilitando a implementação. Além disso, a estrutura de programação é tal que outras formas de controle podem ser facilmente adaptadas à ponte rolante por meio da interface com Python; as alterações no programa do RSLogix são mínimas, caso sejam necessárias.

Assim como nas simulações, foram considerados três experimentos a serem validados: trajetória em malha aberta; trajetória em malha fechada, considerando topo e fundo como rampas; e a trajetória considerada por Rafael [4]. Todas as trajetórias consideram excursão de 30 centímetros.

3.5.2 Testes com Rampa - Malha Aberta e Malha Fechada

O primeiro teste feito foi considerando uma trajetória em formato rampa, realizando um deslocamento de 30 centímetros em cerca de 2.5 segundos. Tal teste foi feito em malha aberta³ e o resultado se encontra na Figura 3.16.

O segundo teste realizado considerou a mesma trajetória do primeiro teste como referência tanto para o topo quanto para o fundo para um experimento com a malha fechada, utilizando-se o preditor de Smith⁴. O resultado se encontra na Figura 3.17.

Nota-se, pelos resultados apresentados, que cada um dos testes realizados com a rampa possuem vantagens e desvantagens entre si: a trajetória em malha aberta apresentou um sobressinal ligeiramente menor e melhor aproximação com o resultado final; entretanto, a trajetória em malha fechada foi melhor seguida durante o movimento. Em termos práticos, considerando a presente trajetória, o preditor de Smith realizou melhor a tarefa do acompanhamento da trajetória, mas o controle em malha fechada deixou a desejar, já que as oscilações não foram controladas. A Figura

³Teste Experimental com Rampa em Malha Aberta — <https://youtu.be/chrez0QEucU>. Acesso em 30/06/2016.

⁴Teste Experimental com Rampa utilizando Preditor de Smith — <https://youtu.be/Q0uxlsT3gBA>. Acesso em 30/06/2016.

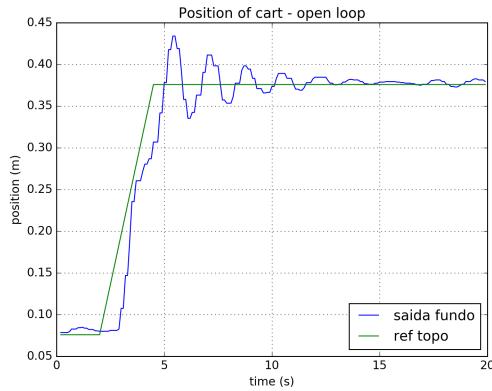


Figura 3.16: Resultado experimental para a trajetória rampa em malha aberta de 30 cm.

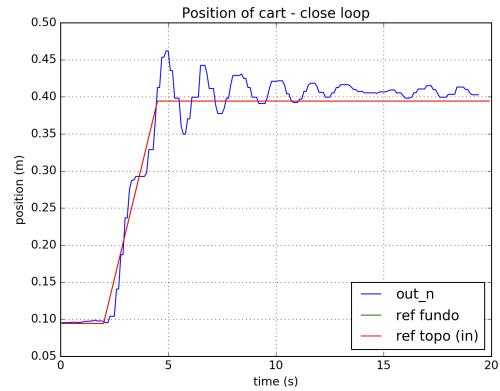


Figura 3.17: Resultado experimental para a trajetória rampa em malha fechada de 30 cm.

3.18 mostra os dois resultados comparados em um mesmo gráfico, subtraindo-se a posição inicial de cada teste.

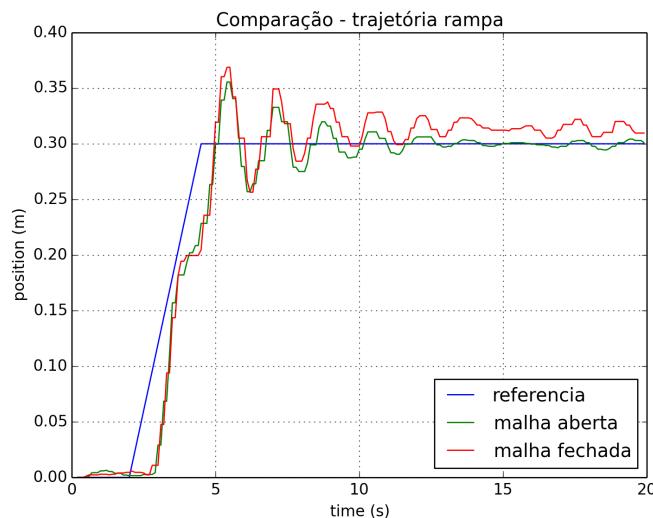


Figura 3.18: Gráfico comparativo das respostas em malha aberta e malha fechada com trajetória rampa.

Motivos para o resultado ruim, ainda mais se comparado com o experimental na Figura 3.17, incluem erros de calibração – a câmera é desconfigurada rapidamente no laboratório de automação – e o fato de termos utilizado OPC adicionava um atraso na comunicação para fechar a malha e às vezes o próximo valor medido não era lido rápido o suficiente para ser considerado no próximo passo de simulação. O filtro de Kalman possivelmente pode ter parâmetros melhores escolhidos com mais alguns testes experimentais, o que poderia também melhorar problemas com ruído.

3.5.3 Teste com Trajetória Proposta - Malha Fechada

O último teste realizado com o preditor de Smith envolve a trajetória utilizada por Rafael [4]. Nesse caso, a trajetória de topo e de fundo são ligeiramente diferentes, sugerindo uma trajetória próxima do esperado para um *riser*. O resultado do experimento⁵ se encontra na Figura 3.19. Nota-se que a saída é bem comportada, seguindo a trajetória; porém, o resultado final apresenta pequenas oscilações, além de um erro de medição. A Figura 3.20 mostra em detalhe a saída, durante o tempo em que a trajetória é executada.

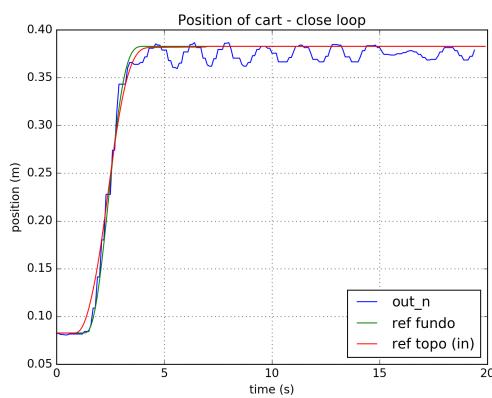


Figura 3.19: Teste com a trajetória sugerida por Rafael utilizando o preditor de Smith.

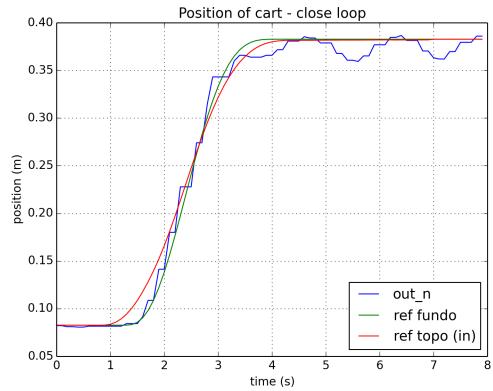


Figura 3.20: Teste com a trajetória sugerida por Rafael, detalhando a execução da trajetória.

Deve-se notar que houveram alguns resultados indesejáveis nos testes apresentados, em particular os que envolvem o preditor. Em primeiro lugar, houve a presença de um sobressinal considerável nos testes com rampa – cerca de 12.5%. Esse tipo de sobressinal pode ser extremamente prejudicial para a operação do *riser*. Além do problema do sobressinal, houve o fato de que a saída oscila e apresenta um erro estático nos testes com o preditor, representados nas Figuras 3.17 e 3.19; embora a presença desses fatores faz o resultado experimental destoar dos resultados das simulações, o resultado geral foi satisfatório. Um resultado marcante foi a atenuação da saída proporcionado pelo uso do Filtro de Kalman, considerando as matrizes de covariância escolhidas; o fato da matriz \mathbf{R} possuir valores maiores do que os elementos de \mathbf{Q} sugere que foi priorizada a correção sobre a medida da posição de fundo.

⁵Teste Experimental com Trajetória Proposta por Rafael[4] utilizando Predator de Smith — <https://youtu.be/lpPZ7H0G7hM>. Acesso em 30/06/2016.

Capítulo 4

Conclusões

O projeto desenvolvido até aqui é uma continuação do que foi desenvolvido por Rédytton [3]; porém, muitos avanços foram realizados. Os avanços e melhorias realizados foram:

- calibração da câmera, de forma a se pegar os dados da trajetória de fundo, representada pela bolinha;
- cálculo do fator de conversão entre a unidade de velocidade reconhecida pelo RSLogix e uma unidade de velocidade derivada do SI — milímetros por segundo;
- adoção o texto estruturado para se desenvolver as estratégias de controle a serem realizadas pelo CLP, uma vez que o texto é facilmente modificável para novas trajetórias, o que não ocorre com linguagens gráficas como o *ladder*;
- modificação do protocolo de comunicação do CLP com o computador. O novo protocolo, Ethernet/IP, é consideravelmente mais rápido que uma comunicação baseada em RS-232, o que resulta em um tempo menor entre um teste e outro;
- uso do OPC para comunicação entre computador e CLP. Dessa forma, uma estrutura complexa como o preditor de Smith pode ser programada em uma linguagem mais rica em termos de ferramentas do que o texto estruturado — no caso, foi utilizada a linguagem Python com o módulo OpenOPC [23].

No início do trabalho, não havia nenhum conhecimento prévio dos autores sobre programação em tempo real utilizando CLPs; os conhecimentos de programação CLP em geral eram bastante limitados. Além disso, nenhum conhecimento prévio da bancada existia também. No estágio atual, pode-se dizer que há relevante facilidade em se trabalhar com a bancada, apesar de não se ter entrado em detalhes de como funciona o servomotor, *drive* e outros componentes.

O foco do projeto era desenvolver uma estrutura de controle não clássico para lidar com um sistema de ordem infinita, representado pelo conjunto carrinho, bolinha e barbante. Dentro dessa abordagem, procurou-se validar o experimento, por meio de um extenso tratamento matemático, de forma que ele representasse confiavelmente uma operação de re-entrada com um *riser* real.

O desenvolvimento matemático e simulações demonstraram a validade da abordagem em teoria. O sistema reduzido em espaço de estados de ordem baixa permitiu desenvolver uma estratégia de controle simples, como a realimentação de estados com canal integral e permitiu integrar de forma simples um preditor de Smith com Filtro de Kalman e controlar a planta de tal forma que a trajetória de fundo real acompanhasse sua referência com erro mínimo, mesmo na presença de perturbações.

Embora os resultados experimentais não terem sido ideais, pode-se notar que o uso do preditor de Smith é um grande avanço em relação ao teste com malha aberta no que concerne ao acompanhamento da trajetória; porém, considerando todas as fontes de erro oferecidas pela planta em questão — posicionamento da câmera, perdas de comunicação entre câmera, CLP e computador, atrito na esteira do carrinho, conversão de velocidades, atraso devido à comunicação (*overhead*), entre outros fatores —, nota-se que os resultados foram satisfatórios; em outras palavras, pode-se dizer que é válido o uso do preditor de Smith como estratégia de controle para uma operação de re-entrada de *risers*, tendo como base os resultados encontrados.

4.1 Perspectivas Futuras

Há muito a ser feito ainda. Além do foco em reduzir o erro em muitos dispositivos da planta, particularmente na câmera, há outros pontos de melhoria no projeto, como por exemplo:

- Utilizar uma melhor estratégia de calibração da câmera — a atual não permite excursões maiores que 50 cm;
- Estudar outras formas de controle para o *riser*, e verificar o desempenho das mesmas em relação ao preditor de Smith;
- Procurar melhorar a programação, otimizando os algoritmos desenvolvidos e utilizando ferramentas para acelerar os programas já feitos (por exemplo, o uso de *threads*, já feito neste projeto). A razão de se acelerar os programas que fazem a comunicação por meio do OPC é impedir que seus custos de execução e *overheads* interfiram no tempo de amostragem, comprometendo os resultados;
- Fazer testes mais minuciosos com as estruturas já desenvolvidas, e utilizar Filtros de Kalman diferentes (por exemplo, filtros adaptativos). O objetivo aqui é atenuar ainda mais as perturbações que venham a ocorrer, visto que um *riser* real é constantemente atingido por forças externas ao controle, como correntes marítimas e eventuais colisões com outros corpos.

REFERÊNCIAS BIBLIOGRÁFICAS

- [1] GODDARD CONSULTING. *A Simple Kalman Filter in Simulink*. Disponível em: <http://www.goddardconsulting.ca/simulink-kalman-filter.html>. Acesso em: 29/06/2016.
- [2] FORTALEZA, E.; ALBUQUERQUE, D.; YAMAMOTO, M. An investigation about the trajectory control during the subsea equipment installation using cable. In: *Volume 1: Offshore Technology*. ASME International, 2012. Disponível em: <<http://dx.doi.org/10.1115/OMAE2012-83798>>.
- [3] SOUSA, R. B. *Implementação de Controle de Riser, Validação Experimental e Análise Através de Processamento de Imagens*. Brasília, DF: Faculdade de Tecnologia, Universidade de Brasília, 2015. Trabalho de Graduação em Engenharia de Controle e Automação.
- [4] SIMÕES, R. D. P. *Sistema de Posicionamento Dinâmico para Instalações Submarinas*. Brasília, DF: Universidade de Brasília. Faculdade de Tecnologia, Departamento de Engenharia Mecânica, 2016. Dissertação de Mestrado.
- [5] GODDARD CONSULTING. *An Introduction to the Kalman Filter*. Disponível em: <http://www.goddardconsulting.ca/kalman-filter.html>. Acesso em: 28/06/2016.
- [6] PETROBRAS. *Pré-Sal: Exploração e Produção de Petróleo*. Disponível em: <http://www.petrobras.com.br/pt/nossas-atividades/areas-de-atuacao/exploracao-e-producao-de-petroleo-e-gas/pre-sal/>. Acesso em: 27/11/2015.
- [7] MONTEIROS, F. R.; FILHO, J. O. de A. L.; FORTALEZA, E. Modal reduction based tracking control for installation of subsea equipments*. *IFAC-PapersOnLine*, v. 48, n. 6, p. 15 – 20, 2015. ISSN 2405-8963. 2nd {IFAC} Workshop on Automatic Control in Offshore Oil and Gas Production {OOGP} 2015 Florianópolis, Brazil, 27–29 May 2015. Disponível em: <<http://www.sciencedirect.com/science/article/pii/S2405896315008708>>.
- [8] NUMFOCUS FOUNDATION. *The Julia Language*. Disponível em: <http://julialang.org/>. Acesso em: 29/06/2016.
- [9] WIKIPEDIA. *Python (programming language)*. Disponível em: [https://en.wikipedia.org/wiki/Python_\(programming_language\)](https://en.wikipedia.org/wiki/Python_(programming_language)). Acesso em 30/06/2016.

- [10] FORTALEZA, E. *Active Control Applied to Offshore Structures: Positioning and Attenuation of Vortex Induced Vibrations*. Tese (Doutorado) — École Nationale Supérieure des Mines de Paris, 2009. Ph.D. thesis.
- [11] OGATA, K. *Modern Control Engineering*. 5th. ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2010. ISBN 0-13-615673-8.
- [12] OGATA, K. *Discrete-Time Control Systems*. 2nd. ed. [S.l.]: Pearson, 1995. ISBN 978-0130342812.
- [13] MATHWORKS. *Convert model from continuous to discrete time - MATLAB c2d*. Disponível em: <http://www.mathworks.com/help/control/ref/c2d.html>. Acesso em: 26/06/2016.
- [14] SERCOS INTERNATIONAL. *Introduction to Sercos*. Disponível em: <http://www.sercos.com/technology/index.htm>. Acesso em: 28/11/2015.
- [15] ALLEN-BRADLEY. *User Manual Kinetix 6000 Multi-axis Servo Drives*. July 2015. Disponível em: http://literature.rockwellautomation.com/idc/groups/literature/documents/um/2094-um001_en-p.pdf. Rockwell Automation Publication 2094-UM001I-EN-P. Acesso em: 28/11/2015.
- [16] KOLLMORGEN. *Servomotors*. Disponível em: <http://www.kollmorgen.com/en-us/products/motors/servo/servomotors/>. Acesso em: 28/11/2015.
- [17] RTA AUTOMATION. *DeviceNet™Unplugged – A View “Under the Hood” for End Users*. Disponível em: <http://www.rtautomation.com/technologies/devicenet/>. Acesso em: 30/11/2015.
- [18] ROCKWELL AUTOMATION. *DeviceNet Network*. Disponível em: <http://ab.rockwellautomation.com/Networks-and-Communications/DeviceNet-Network>. Acesso em: 30/11/2015.
- [19] ODVA. *Ethernet/IP - Quick Start for Vendors Handbook*. Disponível em: https://www.odva.org/Portals/0/Library/Publications_Numbered/PUB00213R0_EtherNetIP_Developers_Guide.pdf. Acesso em: 30/11/2015.
- [20] RINALDI, J. *The 6.5 Things You Must Know about EtherNet/IP*. Disponível em: <http://www.rtautomation.com/technologies/ethernetip/>. Acesso em: 30/11/2015.
- [21] ROCKWELL AUTOMATION. *A Single IT-Friendly Network for Enterprise and Industrial Applications*. Disponível em: <http://www.rockwellautomation.com/global/products-technologies/integrated-architecture/ethernet-ip.page>. Acesso em: 30/11/2015.
- [22] OPC FOUNDATION. *What is OPC?* Disponível em: <https://opcfoundation.org/about/what-is-opc/>. Acesso em: 26/06/2016.

- [23] SOURCEFORGE. *OpenOPC for Python*. Disponível em: <http://openopc.sourceforge.net/>. Acesso em: 26/06/2016.
- [24] EPUSP. *Automação elétrica de processos industriais*. Disponível em: http://lara.unb.br/~gaborges/disciplinas/ca/teoria_final_1_prova.pdf. Acesso em: 29/11/2015.
- [25] BANNER ENGINEERING. *PresencePLUS Industrial Ethernet User's Guide Vol 2*. September 2015. Disponível em: <http://www.bannerengineering.com/en-US/products/sub/78#ui-tabs-37>. Acesso em: 29/11/2015.
- [26] MATHWORKS. *Simulink - Simulation and Model-Based Design*. Disponível em: <http://www.mathworks.com/products/simulink/>. Acesso em: 29/06/2016.

ANEXOS

I. PROGRAMAS UTILIZADOS

I.1 Redução modal

codes/ModalReduction.jl

```
1 module ModalReduction
2     export generateA, generateB, generateC
3     export generateABC, getABC_M, getABCD_R
4     export manuscript_p48, simulation
5     export generateMATLABSimulationScript
6
7
8     function simulation(n, original=false, nout=4)
9         A, B, C = generateABC(n)
10        A_M, B_M, C_M = getABC_M(n, A, B, C)
11        A_R, B_R, C_R, D_R = getABCD_R(n, A_M, B_M, C_M, nout)
12        if original
13            generateMATLABSimulationScriptToCompare("simulacaoN" * string(n) *
14                "Compare.m", A, B, C, A_R, B_R, C_R, D_R)
15            generateMATLABSimulationScript(n, "simulacaoN" * string(n) *
16                "Reduced.m", A_R, B_R, C_R, D_R)
17        else
18            generateMATLABSimulationScript(n, "simulacaoN" * string(n) *
19                "Reduced.m", A_R, B_R, C_R, D_R)
20        end
21    end
22
23 #Gera A, B, C to sistema completo
24 function generateABC(n)
25     tau = 0.2426      # tau do barbante (1/s) para excursão de 30cm
26     tau_l = 0.1133    # tau da bolinha (1/s) para excursão de 30cm
27     ms = 0.0006       # massa linear do barbante (kg/m)
28     mb = 0.00015      # massa da bolinha (kg)
29     g = 9.80665       # aceleração da gravidade (m/s^2)
30     L = 0.82          # Comprimento total do barbante (m)
31     l = L/n           # distância entre dois pontos de discretização (m)
32     T0 = mb*g          # Tração no ponto 0 (logo acima da bolinha) - considerando
33     peso da bolinha (N)
34
35     b = zeros(n)
36     c = g/(2l)
37     d = zeros(n)
38     e = zeros(n)
```

```

36 b[1] = g/l
37 for k = 2:n
38     b[k] = (T0 + ms*g*(k-1)*l)/(ms*l^2)
39     d[k] = b[k] - c
40     e[k] = b[k] + c
41 end
42
43 A = generateA(n, b, d, e, tau, taul)
44 B = generateB(n,e[n])
45 C = generateC(n)
46
47 return A, B, C
48 end
49
50 function generateA(n, b, d, e, tau, taul)
51 M = zeros(n,n)
52 #Primeira linha de M
53 M[1,1] = -b[1]
54 M[1,2] = b[1]
55
56 #Linhas 2 ate n-1
57 for i = 2:n-1
58     M[i,i-1] = d[i]
59     M[i,i] = -2*b[i]
60     M[i,i+1] = e[i]
61 end
62
63 #Linha n
64 M[n,n-1] = d[n]
65 M[n,n] = -2*b[n]
66
67 L = eye(n)
68 for i = 1:n
69     L[i,i] = i == 1 ? -taul : -tau
70 end
71
72 #Concatenar matrizes, gerando matriz (2n,2n)
73 A = [[zeros(n,n) eye(n)]; [M L]]
74
75 return A
76 end
77
78 function generateB(n, eN)
79 B = zeros(2*n)
80 B[2*n] = eN
81
82 return B

```

```

83 end
84
85 function generateC(n)
86     C = zeros(1,2*n)
87     C[1,1] = 1
88
89     return C
90 end
91
92 function getT(n, A)
93     eig_A = eigvals(A)
94     Tcomplex = eigvecs(A)
95
96     T = zeros(2*n, 2*n)
97     i = 1
98     while i <= 2*n #será que tem algo errado? tem de testar
99         if abs(imag(eig_A[i])) > 1e-10
100             T[:,i] = real(Tcomplex[:,i])
101             T[:,i+1] = -imag(Tcomplex[:,i])
102             i = i + 2
103         else
104             T[:,i] = Tcomplex[:,i]
105             i = i + 1
106         end
107     end
108
109     return T
110 end
111
112 function getABC_M(n, A, B, C)
113     T = getT(n,A)
114
115     A_M = T \ A * T
116     B_M = T \ B
117     C_M = C * T
118
119     return A_M, B_M, C_M
120 end
121
122 function getABCD_R(n, A_M, B_M, C_M,n_out=4)
123     C_M_diag = diagm(vec(C_M)) #matriz diagonal
124     G = C_M_diag / A_M * B_M #ganhos
125     subsystems = getSubsystems(n, eigvals(A_M), G)
126     A_R = zeros(n_out,n_out)
127     B_R = zeros(n_out)
128     C_R = zeros(1,n_out)
129

```

```

130 #Construir matrizes
131 i = 1
132 j = 1
133 while i <= n_out
134     index = subsystems[j][2]
135     if length(index) == 2 #complexo
136         A_R[i:i+1,i:i+1] = A_M[index, index]
137         B_R[[i,i+1]] = B_M[index]
138         C_R[1,[i,i+1]] = C_M[index]
139         i = i + 2
140     else
141         A_R[i,i] = A_M[index[1],index[1]]
142         B_R[i] = B_M[index[1]]
143         C_R[1,i] = C_M[index[1]]
144         i = i + 1
145     end
146     j = j + 1
147 end
148
149 D_R = C_M / A_M * B_M - C_R / A_R * B_R
150 return A_R, B_R, C_R, D_R
151 end
152
153 function getSubsystems(n, eig_A, G)
154     i = 1
155     subsystems = []
156     while i <= 2*n
157         if abs(imag(eig_A[i])) > 1e-10 #complexo
158             gain = abs(G[i] + G[i+1]) #considera sinal na soma como aqui ou soma
159             os módulos?
160             append!(subsystems, [(gain, [i,i+1])])
161             i = i + 2
162         else #real
163             gain = abs(G[i])
164             append!(subsystems, [(gain, [i])])
165             i = i + 1
166         end
167     end
168     #ordena pelo primeiro elemento da tupla
169     #ordem descendente
170     sort!(subsystems, rev=true)
171     return subsystems
172 end
173
174 function manuscript_p48()
175     n = 2
176     M = [[-1 1]; [1 -3]]

```

```

176 L = -2 * eye(n)
177 A = [[zeros(n,n) eye(n)];[M L]]
178 B = generateB(2,2)
179 C = generateC(n)
180
181 A_M, B_M, C_M = getABC_M(n,A,B,C)
182 A_R, B_R, C_R, D_R = getABCD_R(n, A_M, B_M, C_M, 4)
183
184 return A_R, B_R, C_R, D_R
185 end
186
187 function generateMATLABSimulationScript(n, filename, A, B, C, D)
188 output = "A = " * string(A) * ";\n\n"
189 output = output * "B = " * string(B) * "';\n\n"
190 output = output * "C = " * string(C) * ";\n\n"
191 output = output * "D = " * string(D) * ";\n\n"
192 output = output * "sys = ss(A, B, C, D);\n"
193 output = output * "opt = stepDataOptions;" 
194 output = output * "opt.InputOffset = 0;\n"
195 output = output * "opt.StepAmplitude = 0.3;\n"
196 output = output * "t = (0:0.01:50)';\n"
197 output = output * "y = step(sys, t, opt);"
198
199 output = output * "fig = figure;\n"
200 output = output * "hold on;\n"
201 output = output * "plot(t,y,'k-');\n"
202 output = output * "xlabel('Time (s)'), ylabel('Position (m));\n"
203 output = output * "title('Sistema original N = " * string(n) * ",\n"
204     simulacao reduzida para ordem 4');\n"
205 output = output * "print('SimulationReduceN" * string(n) * "', '-dpng',\n"
206     '-r300');\n"
207 output = output * "close(fig);\n"
208
209 file = open(filename, "w")
210 write(file, output)
211 close(file)
212 end
213
214 function generateMATLABSimulationScriptToCompare(filename, A, B, C, A_R,
215     B_R, C_R, D_R)
216 output = "A_R = " * string(A_R) * ";\n\n"
217 output = output * "B_R = " * string(B_R) * "';\n\n"
218 output = output * "C_R = " * string(C_R) * ";\n\n"
219 output = output * "D_R = " * string(D_R) * ";\n\n"
220 output = output * "sysR = ss(A_R, B_R, C_R, D_R);\n\n"
221
222 output = output * "A = " * string(A) * ";\n\n"

```

```

220 output = output * "B = " * string(B) * "';\n\n"
221 output = output * "C = " * string(C) * ";\n\n"
222 output = output * "sys0 = ss(A, B, C, [0]);\n\n"
223
224 output = output * "opt = stepDataOptions;" 
225 output = output * "opt.InputOffset = 0;\n"
226 output = output * "opt.StepAmplitude = 0.3;\n\n"
227
228 output = output * "t = (0:0.01:50)';\n"
229 output = output * "yR = step(sysR, t, opt);\n"
230 output = output * "y0 = step(sys0, t, opt);\n\n"
231
232 output = output * "fig = figure;\n"
233 output = output * "hold on;\n"
234 output = output * "plot(t,yR,'k--');\n"
235 output = output * "plot(t,y0,'k');\n"
236 output = output * "legend('Reduzido', 'Original');\n"
237 output = output * "xlabel('Time (s)'), ylabel('Position (m)');\n"
238 n = round(Int,size(A)[1]/2)
239 output = output * "title('Simulacao com sistema original N = " * string(n)
240     * " e reduzido N = 4');\n"
241 output = output * "print('SimulationN" * string(n) * "', '-dpng',
242     '-r300');\n"
243 output = output * "close(fig);\n"
244
245 file = open(filename, "w")
246 write(file, output)
247 close(file)
248
249 end

```

I.2 Filtro de Kalman Utilizado, em Linguagem MATLAB — Adaptado de [1]

codes/KALMAN.m

```

1 function [xhatOut, yhatOut] = KALMAN(u,meas)
2 % This Embedded MATLAB Function implements a very simple Kalman filter.
3 %
4 % It implements a Kalman filter for estimating both the state and output
5 % of a linear, discrete-time, time-invariant, system given by the following
6 % state-space equations:
7 %
8 % x(k) = 0.914 x(k-1) + 0.25 u(k) + w(k)

```

```

9 % y(k) = 0.344 x(k-1) + v(k)
10 %
11 % where w(k) has a variance of 0.01 and v(k) has a variance of 0.1.
12 %
13 % Author: Phil Goddard (phil@goddardconsulting.ca)
14 % Date: Q2, 2011.
15 %
16 % Define storage for the variables that need to persist
17 % between time periods.
18 persistent P xhat A B C Q R
19 if isempty(P)
20 % First time through the code so do some initialization
21 xhat = [0;0;0;0];
22 P = zeros(4,4);
23 A = [0.993175847266250,0.0997676389875316,
24 0.00447446153612418,0.000154491807255262;
25 -0.266442692060039,0.989506934720158,
26 0.0794137333776907,0.00441443534842949;
27 -7.61331011046535,-0.371277877313592,
28 0.407916221277208,0.0776985503560236;
29 -134.001998512554,-9.45851595845769,
30 -10.6078657460473,0.377727256243161];
31
32 B = [0.674742463375352;3.50238796155364;
33 -32.6963528822316;-364.795682866366];
34
35 C = [1,0,0,0];
36
37 Q = 0.01^2*eye(4);
38 R = 0.4^2;
39 end
40 % Propagate the state estimate and covariance matrix:
41 xhat = A*xhat + B*u;
42 P = A*P*A' + Q;
43 % Calculate the Kalman gain
44 K = P*C'/(C*P*C' + R);
45 % Calculate the measurement residual
46 resid = meas- C*xhat;
47 % Update the state and error covariance estimate
48 xhat = xhat + K*resid;
49 P = (eye(size(K,1))-K*C)*P;
50 % Post the results
51 xhatOut = xhat;
52 yhatOut = xhat(1);

```

I.3 Projeto do Controlador com Realimentação de Estados

codes/controle.m

```
1 % Controle com realimentação de estados
2 pC = (0.6)*ones(1,5);
3 pC(4) = 0.5 + 0.4*li;
4 pC(5) = 0.5 - 0.4*li;
5
6 %Seguindo controle digital...
7 n = 4;
8 m = 1;
9
10 Ahat = [A, B; zeros(1,n), 0];
11 Bhat = [zeros(n,1); eye(m)];
12 Khat = acker(Ahat, Bhat, pC);
13 K = (Khat + [zeros(m, n), eye(m)])/([A - eye(n), B; H*A, H*B]);
14
15 % Ganhos para utilizar na realimentação
16 Ki = K(5);
17 Kp = K(1:4);
```

I.4 Texto Estruturado

I.4.1 Inicialização dos testes

codes/initP.st

```
1 /* Universidade de Brasilia
2 Trabalho de Graduacao em Engenharia Mecatronica
3 Alunos:
4 Ataias Pereira Reis 10/0093817
5 Emanuel Pereira Barroso Neto 11/0115716
6 This code must be executed only once. */
7
8 MS0(drive_axis,MS0_1);
9 Kp := 0.0075;
10
11 dataInitialized := 1;
```

I.4.2 Inicialização da rede DeviceNET

codes/InitDNetST.st

```
1 Local:2:0.CommandRegister.Run [:=] 1;
```

```
2 if NOT dataInitialized then  
3   EVENT(Initialize_speed);  
4 end_if;
```

I.4.3 Programa de movimentação do motor

codes/malhaFechadaP.st

```
1 // k is the index of the speed vector  
2  
3 if dataInitialized AND k < 210 then  
4   MAJ(drive_axis,MAJ_1,0,speed,0,50.0,1,50.0,1,0,50.0,50.0,1,0,0);  
5   k := k + 1;  
6 end_if;  
7  
8 if k >= 210 then  
9   MAS(drive_axis,MAS_1,0,0,100,1,0,100,1);  
10  if MAS_1.DN then  
11    MSF(drive_axis,MSF_0);  
12  end_if;  
13 end_if;
```

I.5 Linguagem *ladder*

I.5.1 Execução de *trigger* da câmera

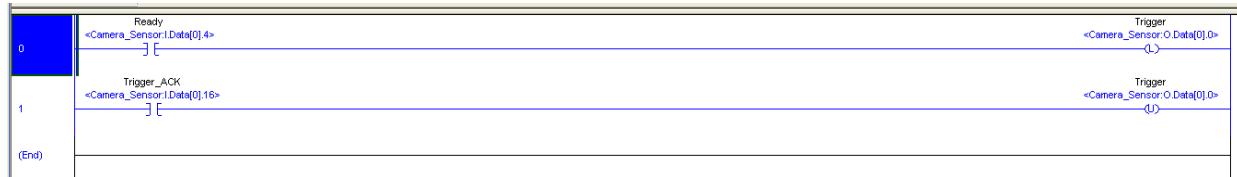


Figura I.1: Trigger da câmera

I.5.2 Rotina de parada de emergência

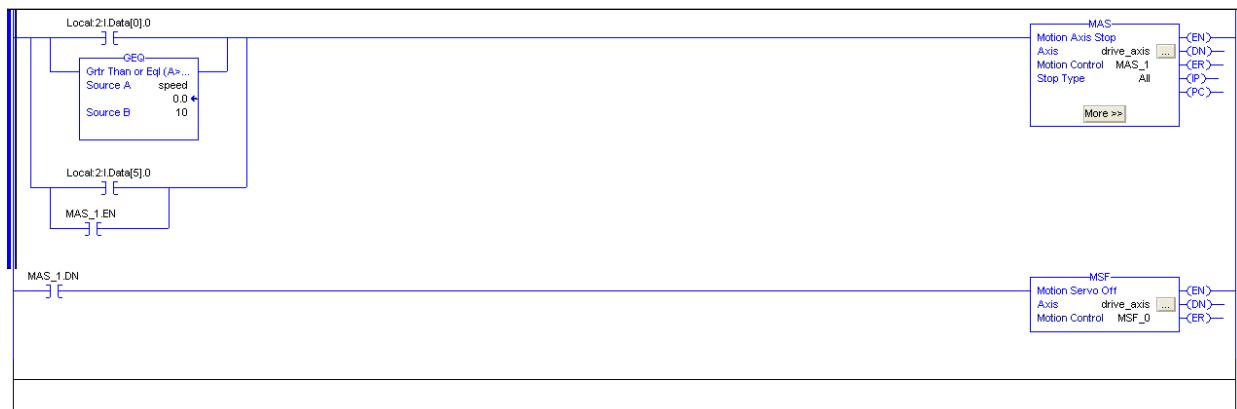


Figura I.2: Parada de emergência

I.6 Programas em *Python* para o controle

I.6.1 Exemplo de teste em Malha Aberta

codes/testeMalhaAbertaRampa.py

```
1 # -*- coding: utf-8 -*-
2
3 from __future__ import print_function
4 import time
5 from Model import Model
6 from PlantOPC import PlantOPC
7 import OpenOPC
8 import numpy
9 import matplotlib.pyplot as plt
10
11 opc = OpenOPC.client() # Cria o cliente OPC; o servidor é o RSLinx
```

```

12 opc.connect('RSLinx OPC Server') # Essa string não muda; conecta ao RSLinx
13
14 start = time.clock()
15 init_pos = (opc.read('[CLP_AB]position',update=1)[0])
16 print('Read init_pos in {:.2f}s'.format(time.clock()-start))
17 print("Initial position: {}mm".format(init_pos))
18 try:
19     input('Enter para iniciar: ')
20 except:
21     pass
22
23 opc['[CLP_AB]Program:DeviceN.pyInit'] = True
24
25
26 # criando rampa
27 Ts = 0.1
28 n_t = 200
29 y_out = numpy.zeros(n_t)
30
31 t = numpy.array(range(0, n_t)) * Ts
32 out_max = 0.3
33 y_topo= numpy.ones(n_t)*out_max + init_pos / 1000.0
34 zero_time = 2.0
35 start = int(zero_time/Ts)
36 y_topo[0:start] = 0.0 + init_pos / 1000.0
37 movement_period = 2.5
38 total_movement_time= int(movement_period/Ts)
39 for i in range(0, total_movement_time):
40     y_topo[i+start] = i * out_max / total_movement_time + init_pos /
41                     1000.0
42
43 # instantiate the plant that will be used, it should be a subclass of Plant
44 plant = PlantOPC(opc,'[CLP_AB]position','[CLP_AB]speed',init_pos)
45
46 start = time.clock()
47 t_old = start
48 times_p = []
49 for i in range(0, n_t):
50     if i > 20 and i < 45:
51         y_out[i] = plant.apply_speed(out_max/movement_period)
52     else:
53         y_out[i] = plant.apply_speed(0.0)
54
55 plant.kill()
56 print("Total simulation time: {}s".format(time.clock() - start))
57

```

```

58 ##y_out_phased = y_out[5:n_t]
59 ##t_out_phased = t[0:n_t-5]
60 ##plt.plot(t_out_phased,y_out_phased, label='out_n')
61 plt.plot(t[2:n_t], y_out[2:n_t], label='saida fundo')
62 plt.plot(t[2:n_t], y_topo[2:n_t], label='ref topo')
63 plt.legend(loc=4)
64 plt.xlabel('time (s)')
65 plt.ylabel('position (m)')
66 plt.title('Position of cart - open loop')
67 plt.grid(True)
68 ##plt.show()
69 plt.savefig('resultados/open_loop_ramp.png', format="png", dpi = 200)
70
71 File = open('resultados/open_loop_ramp.npz','wb')
72 numpy.savez(File,t=t,y_topo=y_topo,y_out=y_out)
73 File.close()
74
75 opc.close() # Encerra a sessão

```

I.6.2 Exemplo de teste em Malha Fechada com Rampa

codes/controleSmithPredictorRampa.py

```

1 # -*- coding: utf-8 -*-
2
3 from __future__ import print_function
4 import time
5 from Model import Model
6 from PlantOPC import PlantOPC
7 import OpenOPC
8 import numpy
9 import matplotlib.pyplot as plt
10
11 opc = OpenOPC.client() # Cria o cliente OPC; o servidor é o RSLinx
12 opc.connect('RSLinx OPC Server') # Essa string não muda; conecta ao RSLinx
13 init_pos = (opc.read('[CLP_AB]position',update=1)[0])
14 pC = numpy.array([0.5, 0.6, 0.7, 0.5 + 0.4j, 0.5 - 0.4j])
15 Ki = 0.183111320328469
16 Kp = numpy.array([0.007993734748865, 0.009705988539721, -0.004630469582507,
-0.000426479250745])
17
18 # Kalman Matrices
19 n = 4
20 Ak = numpy.matrix([[0.993175847266250, 0.0997676389875316,
 0.00447446153612418, 0.000154491807255262],
 [-0.266442692060039, 0.989506934720158,
 0.0794137333776907, 0.00441443534842949],

```

```

22                     [-7.61331011046535, -0.371277877313592,
23                         0.407916221277208, 0.0776985503560236],
24                     [-134.001998512554, -9.45851595845769,
25                         -10.6078657460473, 0.377727256243161]])
26
27 Bk = numpy.matrix([0.674742463375352, 3.50238796155364, -32.6963528822316,
28                   -364.795682866366]).transpose()
29 Ck = numpy.matrix([1.0, 0.0, 0.0, 0.0])
30 Q = 0.01 ** 2 * numpy.eye(4)
31 R = 0.4 ** 2
32
33 epsilon = 0.3
34 Ts = 0.1
35 A = numpy.matrix([[0.9191, -0.3712, 0, 0], [0.3712, 0.9191, 0, 0], [0, 0,
36                   0.4651, -0.8733], [0, 0, 0.8733, 0.4651]])
37 B = numpy.matrix([6.5818, 31.2517, -98.5991, -75.6695]).transpose()
38 C = numpy.matrix([-0.000339110145869, 0.014769867793814, 0.000052840348719,
39                   -0.002915328311939])
40 D = -0.068495332192496
41
42 # criando rampa
43 n_t = 200
44 y_out = numpy.zeros(n_t)
45
46 t = numpy.array(range(0, n_t)) * Ts
47 out_max = 0.3
48 y_topo = numpy.ones(n_t)*out_max + init_pos / 1000.0
49 zero_time = 2.0
50 start = int(zero_time/Ts)
51 y_topo[0:start] = 0.0 + init_pos / 1000.0
52 movement_period = 2.5
53 total_movement_time= int(movement_period/Ts)
54 for i in range(0, total_movement_time):
55     y_topo[i+start] = i * out_max / total_movement_time + init_pos /
56                           1000.0
57
58 y_fundo = y_topo
59
60 try:
61     input('Enter para iniciar: ')
62 except:
63     pass
64
65 opc['[CLP_AB]Program:DeviceN.pyInit'] = True
66
67 t = numpy.array(range(0, n_t)) * Ts
68 # time = linspace(0,10,n_t)

```

```

63 # instantiate the plant that will be used, it should be a subclass of Plant
64
65 plant = PlantOPC(opc, '[CLP_AB]position', '[CLP_AB]speed', init_pos)
66 model = Model(n, A, B, C, D, Ak, Bk, Ck, Q, R, Kp, Ki, epsilon, Ts, plant)
67
68 start = time.clock()
69 t_old = start
70 times_p = []
71 for i in range(0, n_t):
72     y_out[i] = model.closed_loop(y_topo[i], y_fundo[i])
73     #time.sleep(0.1)
74 plant.kill()
75 print("Total simulation time: {}s".format(time.clock() - start))
76
77 y_out_phased = y_out[5:n_t]
78 t_out_phased = t[0:n_t-5]
79 ##plt.plot(t, y_out[0:n_t], label='out')
80 plt.plot(t_out_phased,y_out_phased, label='out_n')
81 plt.plot(t, y_fundo[0:n_t], label='ref fundo')
82 plt.plot(t, y_topo[0:n_t], label='ref topo (in)')
83 plt.legend(loc=4)
84 plt.xlabel('time (s)')
85 plt.ylabel('position (m)')
86 plt.title('Position of cart - close loop')
87 plt.grid(True)
88 # plt.show()
89 plt.savefig("resultados/closed_loop_trajetoria_rampa.png", format='png',
90             dpi=200)
91 File = open('resultados/trajetoria_rampa.npz', 'wb')
92 numpy.savez(File, t=t, y_topo=y_topo, y_fundo=y_fundo, y_out=y_out, pC=pC,
93             Ki=Ki, Kp=Kp)
94 File.close()

94 opc.close() # Encerra a sessão

```

I.6.3 Exemplo de teste em Malha Fechada com Trajetórias Quaisquer

codes/controleSmithPredictor.py

```

1 # -*- coding: utf-8 -*-
2
3 from __future__ import print_function
4 import time
5 from Model import Model
6 from PlantOPC import PlantOPC
7 import OpenOPC
8 import numpy

```

```

9 import matplotlib.pyplot as plt
10
11 opc = OpenOPC.client() # Cria o cliente OPC; o servidor é o RSLinx
12 opc.connect('RSLinx OPC Server') # Essa string não muda; conecta ao RSLinx
13
14 pC = numpy.array([0.5, 0.6, 0.7, 0.5 + 0.4j, 0.5 - 0.4j])
15 Ki = 0.183111320328469
16 Kp = numpy.array([0.007993734748865, 0.009705988539721, -0.004630469582507,
   -0.000426479250745])
17
18 # Kalman Matrices
19 n = 4
20 Ak = numpy.matrix([[0.993175847266250, 0.0997676389875316,
   0.00447446153612418, 0.000154491807255262],
   [-0.266442692060039, 0.989506934720158,
   0.0794137333776907, 0.00441443534842949],
   [-7.61331011046535, -0.371277877313592,
   0.407916221277208, 0.0776985503560236],
   [-134.001998512554, -9.45851595845769,
   -10.6078657460473, 0.377727256243161]])
21
22
23
24
25 Bk = numpy.matrix([0.674742463375352, 3.50238796155364, -32.6963528822316,
   -364.795682866366]).transpose()
26 Ck = numpy.matrix([1.0, 0.0, 0.0, 0.0])
27 Q = 0.01 ** 2 * numpy.eye(4)
28 R = 0.4 ** 2
29
30 epsilon = 0.3
31 Ts = 0.1
32 A = numpy.matrix([[0.9191, -0.3712, 0, 0], [0.3712, 0.9191, 0, 0], [0, 0,
   0.4651, -0.8733], [0, 0, 0.8733, 0.4651]])
33 B = numpy.matrix([6.5818, 31.2517, -98.5991, -75.6695]).transpose()
34 C = numpy.matrix([-0.000339110145869, 0.014769867793814, 0.000052840348719,
   -0.002915328311939])
35 D = -0.068495332192496
36
37 n_t = 200
38 y_topo = numpy.zeros(n_t)
39 y_fundo = numpy.zeros(n_t)
40 y_out = numpy.zeros(n_t)
41
42 file_topo = open('dataWin/y_topo.txt', 'r')
43 file_fundo = open('dataWin/y_fundo.txt', 'r')
44 linha_topo = file_topo.readline()
45 linha_fundo = file_fundo.readline()
46 j = 0
47
```

```

48 start = time.clock()
49 init_pos = opc.read('[CLP_AB]position',update=1)[0]
50 print('Read init_pos in {}s'.format(time.clock()-start))
51
52 for i in linha_topo.split():
53     if j < n_t:
54         y_topo[j] = float(i) + (init_pos / 1000.0)
55         j += 1
56
57 j = 0
58 for i in linha_fundo.split():
59     if j < n_t:
60         y_fundo[j] = float(i) + (init_pos / 1000.0)
61         j += 1
62
63 file_topo.close()
64 file_fundo.close()
65
66 try:
67     input('Enter para iniciar: ')
68 except:
69     pass
70
71 opc['[CLP_AB]Program:DeviceN.pyInit'] = True
72
73 t = numpy.array(range(0, n_t)) * Ts
74 # time = linspace(0,10,n_t)
75 # instantiate the plant that will be used, it should be a subclass of Plant
76
77 plant = PlantOPC(opc, '[CLP_AB]position', '[CLP_AB]speed', init_pos)
78 model = Model(n, A, B, C, D, Ak, Bk, Ck, Q, R, Kp, Ki, epsilon, Ts, plant)
79
80 start = time.clock()
81 t_old = start
82 times_p = []
83 for i in range(0, n_t):
84     y_out[i] = model.closed_loop(y_topo[i],y_fundo[i])
85     #time.sleep(0.1)
86 plant.kill()
87 print("Total simulation time: {}s".format(time.clock() - start))
88
89 y_out_phased = y_out[5:n_t]
90 t_out_phased = t[0:n_t-5]
91 ##plt.plot(t, y_out[0:n_t], label='out')
92 plt.plot(t_out_phased,y_out_phased, label='out_n')
93 plt.plot(t, y_fundo[0:n_t], label='ref fundo')
94 plt.plot(t, y_topo[0:n_t], label='ref topo (in)')

```

```
95 plt.legend(loc=4)
96 plt.xlabel('time (s)')
97 plt.ylabel('position (m)')
98 plt.title('Position of cart - close loop')
99 plt.grid(True)
100 # plt.show()
101 plt.savefig("resultados/closed_loop_trajetoria_rafael.png", format='png',
102             dpi=200)
102 File = open('resultados/trajetoria_rafael.npz','wb')
103 numpy.savez(File, t=t, y_topo=y_topo, y_fundo=y_fundo, y_out=y_out, pC=pC,
104             Ki=Ki, Kp=Kp)
104 File.close()
105
106 opc.close() # Encerra a sessão
```

I.7 Programas da câmera

I.7.1 Detecção da posição horizontal da bolinha

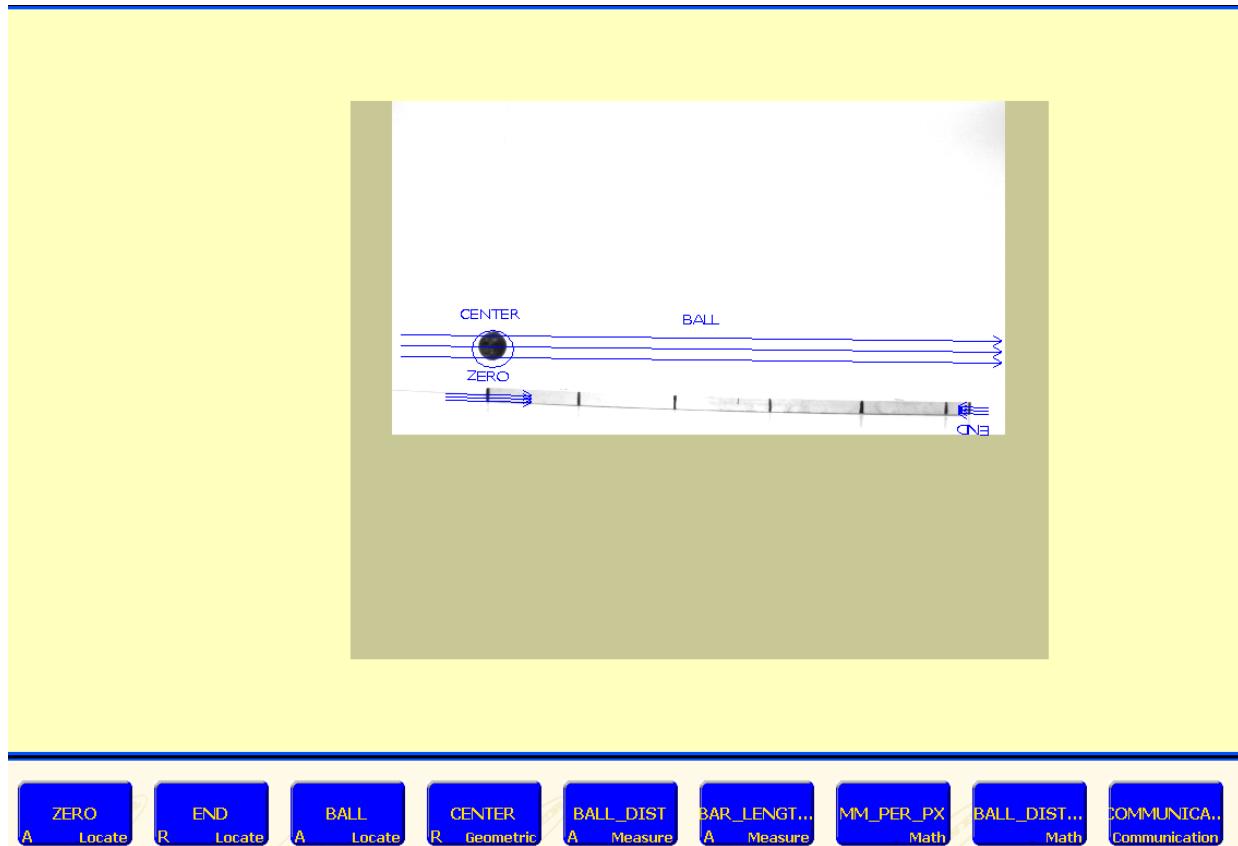


Figura I.3: Detecção da posição horizontal da bolinha