UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA MECÂNICA

IMPLEMENTAÇÃO DE CONTROLE DIFUSO DE UM ACROBOT EM FPGA

YESID ENRIQUE CASTRO CAICEDO

ORIENTADOR: CARLOS HUMBERTO LLANOS QUINTERO
CO-ORIENTADOR: WALTER DE BRITTO VIDAL FILHO

DISSERTAÇÃO DE MESTRADO EM SISTEMAS MECATRÔNICOS

PUBLICAÇÃO: ENM.DM – 29A/09
BRASÍLIA/DF: DEZEMBRO – 2009

UNIVERSIDADE DE BRASÍLIA

FACULDADE DE TECNOLOGIA

DEPARTAMENTO DE ENGENHARIA MECÂNICA

IMPLEMENTAÇÃO DE CONTROLE DIFUSO DE UM ACROBOT EM FPGA

YESID ENRIQUE CASTRO CAICEDO

DISSERTAÇÃO SUBMETIDA AO DEPARTAMENTO DE ENGENHARIA MECÂNICA DA FACULDADE DE TECNOLOGIA DA UNIVERSIDADE DE BRASÍLIA COMO PARTE DOS REQUISÍTOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM SISTEMAS MECATRÔNICOS.

APROVADA POR:

_____

Prof. Carlos Humberto Llanos Quintero, Dr. Eng. (ENM-UnB)
(Orientador)

_____

Prof. Adolfo Bauchspiess, Dr. Eng (ENE-UnB)
(Examinador Externo)

_____

Prof. Leandro dos Santos Coelho, Dr. Eng (PUCPR)
(Examinador Externo)

BRASÍLIA/DF, 10 DE DEZEMBRO DE 2009

FICHA CATALOGRÁFICA

CAICEDO, YESID ENRIQUE CASTRO	
Implementação de Controle Difuso de um Acrobot em FPGA [Distrito Federal] 2009.	
xx, 180 p., 210 x 297 mm (ENC/FT/UnB, Mestre, Sistemas Mecatrônicos, 2009).	
Dissertação de Mestrado – Universidade de Brasília. Faculdade de Tecnologia.	
Departamento de Engenharia Mecânica.	
1. Robótica Móvel	2. Controle Moderno
3. Controle Inteligente	4. FPGAs
5. Co-projeto Hardware/Software	6. Ambiente Virtual Tridimensional
I. ENM/FT/UnB	II. Título (série)

REFERÊNCIA BIBLIOGRÁFICA

CAICEDO, Y. E. (2009). Implementação de Controle Difuso de um Acrobot em FPGA, Publicação ENM.DM-29A/09, Departamento de Engenharia Mecânica, Universidade de Brasília, Brasília, DF, 180 p.

CESSÃO DE DIREITOS

AUTOR: Yesid Enrique Castro Caicedo.

TÍTULO: Implementação de Controle Difuso de um Acrobot em FPGA.

GRAU: Mestre ANO: 2009

É concedida à Universidade de Brasília permissão para reproduzir cópias desta dissertação de mestrado e para emprestar ou vender tais cópias somente para propósitos acadêmicos e científicos. O autor reserva outros direitos de publicação e nenhuma parte dessa dissertação de mestrado pode ser reproduzida sem autorização por escrito do autor.

Yesid Enrique Castro Caicedo
SQN 409 BLOCO P Apartamento 206
70.857-160 Brasília - DF - Brasil.

DEDICATÓRIA

Quero dedicar este trabalho

Aos meus pais, Sara e Pedro.
À minha familia.
À Yuli.
Ao Passarinho.
Aos meus amigos e colegas.

AGRADECIMENTOS

A Deus por iluminar sempre o meu caminho.

Aos meus pais Sara Colombia e Pedro Francisco os quais me permitiram, sem importar nem medir esforços, chegar até onde tem sido possível; os quais me ensinaram que o sacrifício recompensa; que enquanto exista o amor tudo é possível; que as coisas importantes que marcam a vida das pessoas se encontra além do material; e finalmente, que o caminho é longo e difícil mas deve ser enfrentado com humildade, com valentia e entrega.

Agradeço pela família onde me permitiram viver e onde fomos formados para a vida, para o hoje e para o amanhã, para que assim o passado reflita da melhor maneira possível o que temos sido.

Aos meus irmãos Luz Myriam, Francisco Julian e María Isabel e minhas sobrinhas Isabel e Juliana por encherem minha vida de amor e ilusão.

À Yuli por seu amor e paciência incondicionais, por ficar ali justo quando eu mais precisei, e por que foi uma luz mais que Deus colocou na minha vida.

Ao professor Carlos pela oportunidade e pela sua confiança no meu trabalho, pela orientação e disponibilidade, e pela sua amizade.

Ao professor Walter por sua orientação e por permitir-me encaminhar meu mestrado dentro de sua experiência e suas perspectivas.

Aos meus velhos amigos pela família que fomos e pela família que formamos aqui no Brasil.

Aos amigos que a vida me regalou aqui, em especial o Jones, o Syon e o Hugo, e pelo seu apoio nos momentos bons e nos momentos difíceis.

Ao Grupo de Automação e Controle (GRACO) pelos recursos e pelo espaço físico.

Aos professores do programa de Pós-Graduação em Sistemas Mecatrônicos, pelas suas orientações e pela formação que adquiri nestes dois anos.

Ao CNPQ pelo apoio financeiro que permitiu minha estada e sustento em Brasilia.

Ao Brasil pela grande oportunidade de continuar com meus estudos, de fazer possível este sonho, de crescer como pessoa e de viver esta nova experiência.

RESUMO

Este trabalho apresenta a modelagem e controle de um robô em bicicleta utilizando para isto o modelo de um *Acrobot*. Qualquer um de duas aproximações são possíveis para o robô: (a) uma configuração dedicada onde o robô é a bicicleta mesmo, e (b) o robô tem características humanas. No primeiro caso, a estabilidade do robô pode ser obtida por meio do movimento do guidão da bicicleta, com limitações de velocidade. A segunda opção permite que o robô seja controlado em baixas velocidades, inclusive quando fica parado (velocidade igual a zero). Neste caso, a modelagem do robô ciclista tem similaridade com o problema do pêndulo duplo interaturado (*Acrobot*). Neste trabalho foram desenvolvidos controladores para a segunda aproximação. A primeira implementação foi desenvolvida baseada na teoria de controle moderno, envolvendo: (a) um controle de realimentação de estados baseado na alocação de pólos apropriada, garantindo estabilidade e (b) o projeto de um controlador de LQR (*Linear Quadratic Regulator*) que minimiza um critério de custo quadrâtico. O projeto destes dois tipos de controladores foram obtidos mediante a linearização das equações dinâmicas do sistema devido as não-linearidades implícitas. A segunda implementação foi desenvolvida baseada em alguns métodos de controle inteligente, envolvendo: (a) lógica difusa, (b) redes neurais artificiais e (c) um sistema de otimização neuro-difuso o qual mistura a capacidade de aprendizado das redes neurais com o poder de interpretação lingüística dos sistemas de inferência nebulosos. É feita uma comparação entre as técnicas de controle baseadas em especificações de desempenho obtidas da resposta do sistema, e também em alguns índices de desempenho. Para testar o desempenho dos controladores foram calculadas variáveis de estado do sistema e uma solução numérica foi aplicada com o intuito de obter a simulação do sistema. O projeto dos controladores é realizado em Matlab da empresa Mathworks, e um ambiente virtual foi desenvolvido usando a ferramenta *Virtual Reality Toolbox*, a qual permite obter os testes dos controladores sobre um modelo gráfico do robô ciclista. Finalmente foi realizada a implementação do controlador nebuloso sobre arquiteturas reconfiguráveis mediante a utilização da ferramenta *xfuzzy* e utilizando as ferramentas de projeto de processadores embarcados sobre FPGAs (*Field Programmable Gate Arrays*) da Xilinx.

ABSTRACT

This works presents the modeling and control system design for a robot that rides a bicycle using the well-known Acrobot model for slow speeds. In this case, either two approaches are possible for the robot: (a) a dedicated configuration where the robot is the bicycle itself, and (b) the robot having human characteristics. In the first one, the stability of the robot can be achieved by means of the movement of the handlebar of the bicycle, with limitation of speeds. Therefore, in low speeds the centrifugal force generated by the circular or elliptical movement as a response of the handlebar movement is not enough to keep the robot balanced. On the other hand, the second option allows the robot to be controlled for low speed, even for null speed. In this case, the modeling of the robot cyclist has similarity to the problem of the underactuated inverted double pendulum (Acrobot). In this work the implementation of the controller was achieved for the second option following two ways. The first one implementation was developed based on modern control theories, involving: (a) the states feedback controller issues based on the appropriated poles allocation, guarantying stability and (b) the designs of a LQR (Linear Quadratic Regulator) type controller that minimizes the criterion of quadratic cost. The design flow of both controllers are based on the linear dynamic equations of the system due to their implicit nonlinearities. The second one implementation was achieve by means of some intelligent control methods, involving: (a) fuzzy logic, (b) artificial neural networks and (c) tuning neuro-fuzzy systems, which merge the capacity of learning of the neural networks with the power of linguistic interpretation of the fuzzy inference systems (neuro-fuzzy system ANFIS, **A**daptive **N**euro **F**uzzy **S**ystems). A comparison between the proposed techniques of control is done based on the performance specifications of the obtained system response, as well as the performance index, and the system response due to parameters variations. To test the performance of the implemented controllers, the state variables of the system were calculated and a numerical solution was applied in order to obtain the system simulation. The controllers design were developed in Matlab and a three-dimensional virtual environment was developed using the Virtual Reality Toolbox, which allow the designer to test all the controllers over an virtual graphic model of the cyclist robot. Finally, an implementation of the fuzzy controller has been achieved on reconfigurable architectures through of the Xfuzzy tool and the Xilinx embedded processor design tools for FPGAs (*Field Programmable Gate Arrays*).

SUMÁRIO

1 INTRODUÇÃO 1
1.1 MOTIVAÇÃO 1
1.2 JUSTIFICATIVA 2
1.3 OBJETIVOS 4
1.3.1 Objetivo Geral 4
1.3.2 Objetivos Específicos 4
1.4 METODOLOGIA DO TRABALHO 5
1.5 RESULTADOS OBTIDOS NO TRABALHO 6
1.6 ESTRUTURA DO DOCUMENTO 6

2 REVISÂO BIBLIOGRÁFICA E ASPECTOS DE ROBOTICA MÓVEL 7
2.1 ESTADO DA ARTE 7
2.1.1 O problema do pêndulo invertido na indústria 11
2.1.2 O problema do robô ciclista na indústria 12
2.2 ASPECTOS DE ROBÓTICA MÓVEL 13
2.2.1 Sensoriamento 14
2.2.2 Classificação de Sensores 14
2.2.3 Arquitetura de Controle 15
2.2.3.1 Estrutura 16
2.2.3.2 Aspectos de raciocínio 18
2.2.3.3 Decomposição e encapsulamento 20
2.2.4 Atuação 22
2.2.5 Tecnologias utilizadas em Arquiteturas 23
2.2.6 Simulação de robôs móveis 24
2.3 CONCLUSÃO DO CAPÍTULO 26

3 PROJETO CONCEITUAL DO ROBÔ CICLISTA 27
3.1 CONSIDERAÇÕES INICIAIS 27
3.2 TIPOS DE CONFIGURAÇÃO 28
3.3 CONTROLE DE EQUILIBRIO 28
3.3.1 Controle de Equilíbrio mediante a Movimentação do Guidão 28
3.3.2 Controle mediante a Movimentação do Ciclista 30
3.4 DESCRIÇÃO DO SISTEMA, RESTRIÇÕES E CONSIDERAÇÕES 31
3.4.1 Efeito Giroscópico 33
3.4.2 O efeito giroscópico na roda dianteira devido ao esforço no guidão 35
3.5 ARQUITETURA DE CONTROLE 37

3.6 DETALHAMENTO DO PROJETO CONCEITUAL 38
3.6.1 Projeto Mecânico 39
3.6.2 Projeto Eletro-Eletrônico 42
3.6.2.1 Baterias 43
3.6.2.2 Sensores 43
3.6.2.3 Motores 44
3.6.2.4 Unidade de Controle Baseada em FPGAs 46
3.6.3 Projeto Computacional 48
3.7 CONCLUSÃO DO CAPÍTULO 50

4 MODELAGEM DINÂMICA 51
4.1 FORMA DE CONFIGURAÇÃO 52
4.2 REPRESENTAÇÃO EM ESPAÇO DE ESTADOS 52
4.3 EQUAÇÕES DE ENTRADA E SAÍDA 54
4.4 FUNÇÃO DE TRANSFERÊNCIA 55
4.5 MÉTODO DE RUNGE KUTTA DE 4a ORDEM 55
4.5.1 Erros em aproximações numéricas 56
4.6 MODELO MATEMÁTICO DO PÊNDULO DUPLO INTERATUADO 58
4.7 ENCONTRANDO AS EQUAÇÕES DE ESTADO 61
4.8 CONCLUSÂO DO CAPÍTULO 63

5 FUNDAMENTOS DE CONTROLE 64
5.1 METODOS CLÁSSICOS DE CONTROLE 64
5.1.1 Lugar das Raízes 64
5.1.2 Reposta em Freqüência 65
5.2 MÉTODOS DE CONTROLE MODERNO 66
5.2.1 Conceitos de Controlabilidade e Observabilidade 67
5.2.2 Sistemas de Controle no Espaço de Estados 68
5.2.3 Alocação de Pólos 68
5.2.4 Controle Ótimo 70
5.2.5 Regulador Quadrático Linear (LQR) 70
5.3 CONTROLE INTELIGENTE 71
5.3.1 Controlador Difuso 72
5.3.1.1 Controladores Difusos Hierárquicos. 75
5.3.2 Redes Neurais - Aprendizado Supervisionado 76
5.3.3 Sistemas Neuro-Difusos 79
5.4 CONCLUSÕES DO CAPÍTULO 82

6 ANÁLISE, RESULTADOS E DISCUSSÃO 84
6.1 ÍNDICES DE DESEMPENHO 84

6.2 CONTROLE POR REALIMENTAÇÃO DE ESTADOS 85
6.3 Controlador LQR 91
6.4 CONTROLADOR DIFUSO 99
6.5 CONTROLADOR NEURAL 103
6.6 CONTROLADOR NEURO-D 108
6.7 RESPOSTA DOS CONTROLADORES A VARIAÇÕES DE PARÂMETROS DO SISTEMA 118
6.8 CONCLUSÂO DO CAPÍTULO 119

7 IMPLEMENTAÇÃO NO FPGA 121
7.1 CO-PROJETO DE *HARDWARE-SOFTWARSOFTWARE* 123
7.2 O MICROPROCESSADOR EMBARCADO MICROBLAZE (XILINX) 123
7.3 INTERFACES DE CONEXÃO COM O MICROBLAZE 124
7.4 COMPONENTES DO SISTEMA DIFUSO IMPLEMENTADO EM *HARDWARE* 125
7.5 O SISTEMA XFUZZY 127
7.6 O SISTEMA REALIDADE VIRTUAL USANDO MATLAB 129
7.7 IMPLEMENTAÇÃO DO CONTROLADOR E OS RESULTADOS OBTIDOS 129
7.8 CONCLUSÃO DO CAPÍTULO 136

8 CONCLUSÕES E SUGESTÕES PARA TRABALHOS FUTUROS 137
8.1 CONCLUSÕES ..... **Erro! Indicador não definido.**
8.2 SUGESTÕES PARA TRABALHOS FUTUROS 139

REFERÊNCIAS BIBLIOGRÁFICAS 142

A. APÊNDICE A – Código Fonte em Matlab para Simulador do Pêndulo Duplo Invertido Interatuado 151

B. APÊNDICE B – Código Fonte em XPS para Intercambio de Dados entre a Planta e o Processador Embarcado. 155

C. APÊNDICE C – Código Fonte de conexão entre Virtual Realm Builder, Matlab, e envío e Recepção de dados com a Spartan 3E 161

LISTA DE TABELAS

TABELA 2.1- CRITÉRIOS COMUNS UTILIZADOS PARA AVALIAÇÃO DE SENSORES 15
TABELA 2.2- QUADRO COMPARATIVO DOS DIVERSOS TIPOS DE ATUADORES USADOS PARA ROBÔS (PIERI, 2002) 22
TABELA 3.1- CARACTERÍSTICAS DOS MICRO-SERVOS *BLUEARROW*. 46
TABELA 4.1- QUADRO COMPARATIVO ENTRE REPRESENTAÇÃO DE SISTEMAS 55
TABELA 5.1- FUNCIONAMENTO DO ALGORITMO HÍBRIDO (JANG, 1993, MODIFICADO) 81
TABELA 6.1- VALORES DE GANHOS PARA DIFERENTES POSIÇÕES DE ALOCAÇÃO DOS PÓLOS 89
TABELA 6.2- RESPOSTA DO CONTROLADOR RE USANDO ÍNDICES DE DESEMPENHO. 91
TABELA 6.3- PARÂMETROS DE AJUSTE PARA ENCONTRAR O VETOR DE GANHOS DO LQR 93
TABELA 6.4- RESPOSTA DO CONTROLADOR LQR USANDO ÍNDICES DE DESEMPENHO..... 95
TABELA 6.5- RESPOSTA DO CONTROLADOR DIFUSO USANDO ÍNDICES DE DESEMPENHO 103
TABELA 6.6- RESPOSTA DO CONTROLADOR NEURAL USANDO ÍNDICES DE DESEMPENHO.... 108
TABELA 6.7- RESPOSTA DO SISTEMA NEURO-DIFUSO USANDO ÍNDICES DE DESEMPENHO ... 113
TABELA 6.8- RESUMO COMPORTAMENTO DO PROJETO DOS CONTROLADORES 118
TABELA 6.9- RESPOSTA DOS CONTROLADORES À VARIAÇÃO DE MASSA 118
TABELA 6.10- RESPOSTA DOS CONTROLADORES À VARIAÇÃO DE COMPRIMENTO 119
TABELA 7.1- RESULTADOS DA IMPLEMENTAÇÃO DOS MÓDULOS XFUZZY NA SPARTAN XC3S500E-4FG320C. 134
TABELA 7.2- COMPARAÇÃO ENTRE TEMPOS DE EXECUÇÃO DO CONTROLADOR *XFUZZY_01* PARA A *BARRA 1*. 135
TABELA 7.3- COMPARAÇÃO ENTRE TEMPOS DE EXECUÇÃO DO CONTROLADOR *XFUZZY_02* PARA A *BARRA 2*. 135

LISTA DE FIGURAS

FIGURA 1.1- CONSIDERAÇÕES SISTEMA BICICLETA-CICLISTA. 3
FIGURA 2.1- DIFERENTES TIPOS DE PÊNDULOS INVERTIDOS 10
FIGURA 2.2- *SEGWAY*, VEÍCULO DESENVOLVIDO POR *SERWAY INC.* (HTTP://WWW.SOHO.COM.CO). 11
FIGURA 2.3- MURATA BOY ROBÔ E MURATA GIRL ROBÔ (WWW.MURATABOY.COM) 12
FIGURA 2.4- *FLOSSIE*, ROBÔ DE TESTES DA EMPRESA *CASTROL* (INOVAÇÃO TECNOLÓGICA, 2009) 13
FIGURA 2.5- CLASSIFICAÇÃO DOS SENSORES EM RELAÇÃO COM SUA FUNÇÃO. 15
FIGURA 2.6- CLASSIFICAÇÃO DE ARQUITETURAS PARA ROBÔS MÓVEIS. 16
FIGURA 2.7- ARQUITETURA REATIVA 18
FIGURA 2.8- ARQUITETURA DELIBERATIVA (PIERI, 2002) 19
FIGURA 2.9- ESQUEMA DE ARQUITETURAS BASEADAS EM COMPORTAMENTOS E MÓDULOS FUNCIONAIS. 21
FIGURA 2.10- DIFERENTES ABORDAGENS DE DISTRIBUIÇÃO DE CPUS PARA UM SISTEMA 24
FIGURA 3.1- CONTROLE DE POSIÇÃO INVERTIDA DE UMA BICICLETA MEDIANTE A MOVIMENTAÇÃO DO GUIDÃO 30
FIGURA 3.2- APROXIMAÇÃO DO ROBÔ CICLISTA COM O *ACROBOT* 31
FIGURA 3.3- GRAUS DE LIBERDADE E PARÂMETROS DINÂMICOS DA BICICLETA (ÅSTROM *ET AL.*, 2005, MODIFICADA) 32
FIGURA 3.4- GERAÇÃO DO EFEITO GIROSCÓPICO 33
FIGURA 3.5- EFEITO GIROSCÓPICO NA RODA DIANTEIRA AO TOMAR UMA CURVA 33
FIGURA 3.6- ARQUITETURA DE CONTROLE PROPOSTA PARA O ROBÔ CICLISTA 38
FIGURA 3.7- ORGANOGRAMA PROJETO CONCEITUAL 38
FIGURA 3.8- PARÂMETROS DE CONFIGURAÇÃO GEOMÉTRICA DE UMA BICICLETA (ÅSTROM *ET AL.,* 2005) 39
FIGURA 3.9- PROJETO MECÂNICO DO ROBÔ CICLISTA 41
FIGURA 3.10- PROJETO ELETRÔNICO DO ROBÔ CICLISTA 42
FIGURA 3.11- PRINCIPIO DE FUNCIONAMENTO DOS ACELERÔMETROS CAPACITIVOS 44
FIGURA 3.12- ARQUITETURA BLOCOS FUNCIONAIS DA FAMÍLIA SPARTAN 3E 47

FIGURA 4.1- SISTEMA DINÂMICO E VARIÁVEIS DE ESTADO.....53
FIGURA 4.2-. ALGORITMO DE RUNGE KUTTA 4A ORDEM (BOYCE, 2001). 57
FIGURA 4.3- PARÂMETROS PARA O CÁLCULO DAS EQUAÇÕES DE MOVIMENTO DO PÊNDULO DUPLO INVERTIDO INTERATUADO. 58
FIGURA 4.4- FORÇAS DE TIPO NÃO-CONSERVATIVAS CONSIDERADAS, ATUANTES SOB O SISTEMA. 59
FIGURA 5.1- DIAGRAMA DE BLOCOS REFERENTE À REALIMENTAÇÃO DE ESTADOS. 69
FIGURA 5.2- ESTRUTURA TÍPICA DE UM CONTROLADOR DIFUSO 73
FIGURA 5.3- FUNÇÕES DE PERTINÊNCIA TÍPICAS E PARÂMETROS DE AJUSTE 74
FIGURA 5.4- FUNÇÕES DE ATIVAÇÃO DOS NEURÔNIOS..... 77

FIGURA 5.5- REPRESENTAÇÃO DE UMA REDE NEURAL 78
FIGURA 5.6- ARQUITETURA ANFIS (JANG, 1993, MODIFICADO) 80
FIGURA 6.1- COMPORTAMENTO DO SISTEMA DE EQUAÇÕES DE ESTADO LINEARIZADO DO PÊNDULO DUPLO AO REDOR DO PONTO DE OPERAÇÃO $\{X_1,X_2,X_3,X_4\}=\{0,0,0,0\}$. 87
FIGURA 6.2- RESPOSTA DO SISTEMA EM MALHA FECHADA COM DIFERENTES VALORES DE GANHOS SOBRE O SISTEMA NÃO LINEARIZADO 90
FIGURA 6.3- RESPOSTA DO CONTROLADOR DE REALIMENTAÇÃO DE ESTADOS COM CONDIÇÕES INICIAIS {X1,X2,X3,X4} = {0,1047 -0,4363 -0,2618 *0,2793*}. 92
FIGURA 6.4-. RESPOSTA DO CONTROLADOR DE REALIMENTAÇÃO DE ESTADOS COM CONDIÇÕES INICIAIS $\{X1,X2,X3,X4\}=\{-0,1047\ 0,2967\ -0,1571\ 0,2618\}$. 94
FIGURA 6.5-. RESPOSTA DO REGULADOR LQR COM CONDIÇÕES INICIAIS $\{X1,X2,X3,X4\}$ = {0,1047 -0,4363 -0,2618 0,2793} 96
FIGURA 6.6-. RESPOSTA DO REGULADOR LQR COM CONDIÇÕES INICIAIS $\{X1,X2,X3,X4\}=\{-0,1047\ 0,2967\ -0,1571\ 0,2618\}$..... 97
FIGURA 6.7- DESCRIÇÃO DA SIMULAÇÃO DA PLANTA. 99
FIGURA 6.8-. ARQUITETURA DO CONTROLADOR DIFUSO.....100
FIGURA 6.9- BASE DE REGRAS GERADAS PARA OS SUBSISTEMAS DO CONTROLADOR.....101
FIGURA 6.10- FUNÇÕES DE PERTINÊNCIA PARA CADA CONTROLADOR DIFUSO..... 101
FIGURA 6.11- ALGUMAS POSIÇÕES DE ESTUDO PARA ESCOLHA DOS PARÂMETROS DE SAÍDA DO CONTROLADOR (*SINGLETONS*). 102
FIGURA 6.12- RESPOSTA DO CONTROLADOR DIFUSO COM CONDIÇÕES INICIAIS $\{X1,X2,X3,X4\}=\{0,1047\ -0,4363\ -0,2618\ 0,2793\}$..... 104

FIGURA 6.13-. RESPOSTA DO CONTROLADOR DIFUSO COM CONDIÇÕES INICIAIS {*x1,x2,x3,x4*}={-0,1047 -0,1571 0,2967 0,2618}..... 105
FIGURA 6.14-. ARQUITETURA PERCEPTRON MULTICAMADA PARA BALANÇO DO PÊNDULO DUPLO INTERATUADO 106

FIGURA 6.15- A CONVERGÊNCIA DO TREINAMENTO DA REDE NEURAL MEDIANTE O TOOLBOX DE MATLAB. 107
FIGURA 6.16-. RESPOSTA DA REDE NEURAL COM CONDIÇÕES INICIAIS {*x1,x2,x3,x4*} ={0,1047 -0,4363 -0,2618 0,2793}. 109
FIGURA 6.17-. RESPOSTA DA REDE NEURAL COM CONDIÇÕES INICIAIS {*x1,x2,x3,x4*} ={-0,2094 0 0,3491 0,1745}. 110
FIGURA 6.18-. INTERFACE DE USUÁRIO ANFIS. 112
FIGURA 6.19-. ESTRUTURA NEURO-DIFUSA GERADA PELO ANFIS. 113
FIGURA 6.20- FUNÇÕES DE PERTINÊNCIA OTIMIZADAS PELO SISTEMA ANFIS. 113
FIGURA 6.21-. RESPOSTA DO CONTROLADOR ANFIS COM CONDIÇÕES INICIAIS {*x1,x2,x3,x4*} = {0,1047 -0,4363 -0,2618 0,2793} 115
FIGURA 6.22-. RESPOSTA DO CONTROLADOR ANFIS COM CONDIÇÕES INICIAIS {*x1,x2,x3,x4*} = {0,1745 -0,4363 0,2618 -0,2618} 116
FIGURA 7.1-. IMPLEMENTAÇÃO EM HARDWARE (XFUZZY) DE UM SISTEMA DIFUSO (MODIFICADO, SÁNCHEZ *ET AL.*, 2001). ARQUITETURA BLOCOS FUNCIONAIS DA FAMÍLIA SPARTAN 3E 126

FIGURA 7.2- REPRESENTAÇÃO TRIDIMENSIONAL DO ROBÔ EM BICICLETA UTILIZANDO *VIRTUAL REALM BUILDER 2.0* 130
FIGURA 7.3- DIAGRAMA DE BLOCOS DE CONTROLE BASEADO EM FPGA. 131
FIGURA 7.4- ARQUITETURA DE CONTROLE SOBRE A FPGA. 132
FIGURA 7.5- VISTA GERAL DO SISTEMA IMPLEMENTADO. 133

LISTA DE SÍMBOLOS, NOMENCLATURAS E ABREVIAÇÕES

ANFIS – *Adaptive Neuro-Fuzzy Systems* – Sistema Neuro-Difuso Adaptativo

ASICs – *Application Specific Integrated Circuits* – Circuito Integrado de Aplicação Específica

CAD – *Computer Aided Design* – Projeto Auxiliado por Computador

CCI – Centro de Curvatura Instantâneo

CLB – *Configurable Logic Block* - Blocos Lógicos Configuráveis

CMOS – *Complementary Metal Oxide Semiconductor*

COA – Defuzzificação por Centro de Area

CPLD – *Complex Programmable Logic Device* – Dispositivo Lógico Programável

DCM – Blocos de Manejo de Relógio Digital

DC – Direct Current – Corrente Direta

EDA – *Electronic Design Automation* – Automação de Desenho Electrônico

EDK – *Embedded Development Kit* – Kit de Desenvolvimento Embarcado

FLC – *Fuzzy Logic Controller* – Controlador Lógico Difuso

FPGA – *Field Programmable Gate Array*

FSL – *Fast Simple Link*

FSOM – *Fuzzy Self-Organized Map*

GPPs –*General Propposal Processor*

IP – *Intellectual Property* – Propriedade Intelectual

IPIF – *Intellectual Property InterFace* – Interface de Propriedade Intelectual da Xilinx

IPIC – *Intellectual Property InterConect* – Interconexão de Propriedade Intelectual

IOBs – *Input Output Blocks* - Blocos de entrada e saída

LB – *Logic Block* - Bloco Lógico

LQR – *Linear Quadratic Regulator* – Regulador Quadrático Médio

LTU – *Look Up Table* – Tabela de busca

MOM – Defuzzificação dos Máximos

NEFCALSS – Neuro Fuzzy Classification

OPB – *On Chip Peripheral Bus*

PLD – *Programmable Logic Devices* – Dispositivos Lógicos Programáveis

RAM – *Random Access Memory* – Memória de Acesso Aleatório

RC – *Reconfigurable Computing* – Computação Reconfigurável

RISC – *Reduce Instruction Set Computer* – Conjunto Reduzido de Instruções

RL – *Reinforcement Learning* – Aprendizado por Reforço

RNs – Redes Neurais

SoC – *System on Chip*

TTL – *Transistor Transistor Logic* – Lógica Transistor Transistor

XCL – *Xilinx Cache Link*

1 INTRODUÇÃO

1.1 MOTIVAÇÃO

O projeto de controle para sistemas instáveis tem sido uma área de interesse na comunidade científica de pesquisadores por desafiar constantemente as condições da natureza (Aracil e Gordillo, 2005; Cossalter, 2009). Neste caso, o projeto de controladores para este tipo de sistemas depende de vários fatores entre os quais tem-se: (a) o conhecimento e disponibilidade do modelo e conjunto de equações dinâmicas do sistema, (b) a sua complexidade, (c) o seu grau de linearidade, (d) o número de variáveis de entrada e saída e (e) a faixa de operação, entre outros. Pode-se dividir o projeto de controladores em três áreas de estudo, a saber: (a) controladores baseados em métodos clássicos e modernos, (b) controladores baseados em inteligência artificial, e (c) uma mistura entre técnicas do tipo (a) e (b), ou uma mistura entre técnicas do mesmo tipo.

Tendo em conta estas considerações e focando o projeto de controladores sobre um sistema específico, como é a instabilidade dos veículos robóticos com um número menor que três rodas, encontram-se uma inclinação (Yasuhito e Toshiyuki, 2004; Tanaka e Murakami, 2004; Åström *et al.,* 2005; Jingang *et al.*, 2006; Michini, 2006) pelo projeto de controle para estabilidade de robôs bicicletas mediante o controle do guidão. Porém, neste caso, a limitação é de que para certas velocidades **muito pequenas ou zero** não é possível manter sua postura vertical. Então, pode-se observar que além dos problemas de localização, planejamento de trajetória e sistema de navegação dos robôs móveis aparece outro, **o equilíbrio**.

Os robôs móveis são a melhor aproximação de máquinas que imitam e emulam as tarefas dos seres vivos (Floreano *et al.*, 1998; Soarez, 1999; Pieri, 2002) e constituem uma plataforma adequada para pesquisar sobre comportamentos inteligentes. Atualmente, estão sendo desenvolvidos como aplicações nas áreas de transporte, vigilância, inspeção para realizar tarefas de monitoração de ambientes, pilotagem automática, exploração (ambientes inóspitos), busca e resgate, limpeza de grandes áreas, funções de guias, etc. (Soarez, 1999). Nestes casos, os mesmos são projetados para interagir e executar tarefas específicas e obter informações relevantes relacionadas com a percepção do ambiente e com situações do

médio. Muitas vezes, estes meios de interação encontram-se afastados ou são de difícil acesso, geralmente pelas condições do terreno, por serem espaços reduzidos ou por situações que colocam em risco a integridade física das pessoas. Pelos motivos anteriores, os robôs ciclistas são considerados apropriados para as aplicações supracitadas, pelo fato de terem corpos estreitos, alta manobrabilidade e radio de giro pequeno, e podem ser mais eficientes quando comparados com veículos com três ou mais rodas para algumas destas tarefas.

Além da funcionalidade e ação dos robôs móveis, tem-se hoje em dia uma motivação pelo trabalho com novas tecnologias e na melhora do desempenho da execução dos algoritmos de controle. Isto vem sendo suportado pelos avanços da microeletrônica, especificamente na aparição dos Dispositivos Lógicos Programáveis (PLDs). O sistemas microprocessados/microcontrolados são baseados no modelo de von Neumann (Patterson *et al*., 2009), o qual possui sérias restrições, baseadas principalmente na execução seqüencial de instruções (limitações na velocidade de processamento das instruções) (Hartenstein, 2006). Estas limitações ficam dramáticas devido ao paralelismo inerente aos algoritmos de controle. Neste caso, surge então a necessidade de empregar estruturas *hardware* dedicadas que aceleram parcial ou totalmente a execução destes algoritmos.

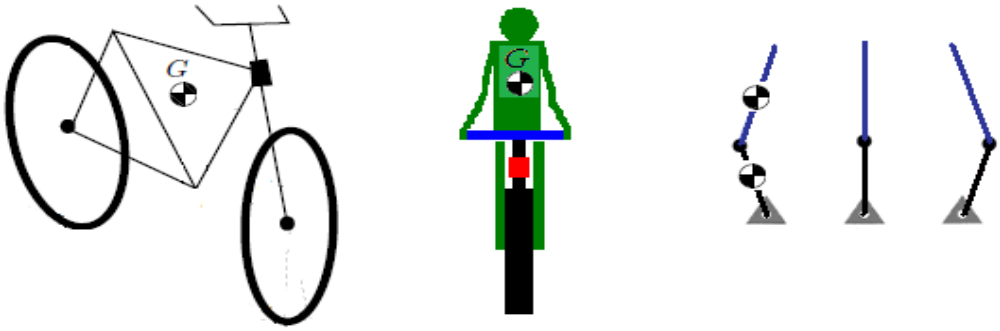
1.2 JUSTIFICATIVA

O problema de estabilidade de bicicletas tem sido estudado ao longo dos anos pelos pesquisadores de todo o mundo (Whipple, 1899; Carvallo, 1900; Klein e Sommerfield, 1910; Jones, 1970; Papadopulus e Hand, 1987; Åström *et al.*, 2005; Meijaard *et al.*, 2007). Este problema representa um sistema complexo e difícil de modelar pelos seus múltiplos graus de liberdade que possui. Adicionalmente, não se tem ainda hoje um conjunto de equações que definam de uma forma absoluta o seu comportamento (Åström *et al.,* 2005). Neste sentido, um primeiro problema surge no contexto da modelagem do sistema. Um segundo problema tem a ver com o projeto de controladores para que a bicicleta mantenha a sua posição vertical. Para tanto, é utilizado, geralmente, o controle sobre o guidão.

É conhecido o fato de que para certas condições um movimento do guidão gera um centro de curvatura instantâneo que em condições ideais produz uma trajetória circular, e que se fazendo algumas considerações adicionais (como escorregamento das rodas pode mudar

para uma trajetória elíptica), surge uma força adicional, a força centrífuga, utilizada para corrigir a postura da bicicleta. Entretanto, isto é válido para velocidades maiores que uma velocidade limite e diferentes de zero (Tanaka e Murakami, 2004).

Sob estas considerações, e tendo em conta que tanto a bicicleta como a parte superior do corpo do ciclista podem ser considerados como dos pontos de massa no espaço, é proposto o modelo do robô ciclista para baixas velocidades incluída velocidade zero, como um pêndulo duplo interatuado no médio (Figura 1.1.c). A parte superior do corpo do ciclista pode se movimentar lateralmente com ralação ao quadro da bicicleta (Figura 1.1.b), e a bicicleta pode ser considerada como um ponto de massa com dos pontos de contacto no chão (Figura 1.1.a). Esta dinâmica é amplamente estudada em Fajans (2000), Yasuhito e Toshiyuki (2004), Åström *et al.* (2005) e Jingang *et al*. (2006). Neste caso, são considerados os efeitos de mudar alguns parâmetros geométricos e tomadas em conta considerações de tipo holonômicas.



a.Configuração Dedicada b. Configuração Humanóide c. Pêndulo Duplo Invertido

Figura 1.1- Considerações do sistema bicicleta-ciclista.

O constante avanço das tecnologias de fabricação de circuitos integrados impõe novas direções ao mercado microeletrônico onde uma significativa iniciativa está na demanda de aplicações inovadoras, com elevados níveis de complexidade e que possuem tempos de desenvolvimento curtos. As tentativas para atingir ambos os requisitos simultaneamente têm levado aos pesquisadores nesta área a propor uma série de novas técnicas e estratégias de projeto, a saber: (a) a concepção dos sistemas desde a perspectiva do *System on Chip* (*SoC*), (b) a combinação de elementos de processado de propósito geral com outros de caráter específico, (c) o uso de técnicas híbridas baseadas no co-projeto

hardware/software, (d) o emprego de módulos de Propriedade Intelectual (IP) e (e) a inclusão no ciclo de desenvolvimento de etapas de prototipagem rápida, baseadas em dispositivos lógicos programáveis como os FPGAs (*Field Programmable Gates Arrays*) (Sánchez *et al.*, 2006).

Neste trabalho, apresenta-se o desenvolvimento de alguns controladores projetados inicialmente em *software* para estabilidade do robô ciclista. De outro lado, o desenvolvimento de alguns destes controladores baseados em *hardware* permitem descentralizar a unidade de controle do computador. Isto permite uma primeira aproximação visando a construção de um protótipo real que pode substituir a simulação da planta feita no computador, aproveitando características fundamentais tais como: o baixo consumo, a reconfigurabilidade, o processamento em paralelo e a capacidade de utilizar este processamento em paralelo com as características próprias dos microprocessadores embarcados.

Neste contexto, têm sido desenvolvidas ferramentas que permitem simular e experimentar virtualmente com modelos de plantas no computador, tal é o caso do Matlab da empresa *MathWorks*. Isto traz algumas vantagens tais como: (a) diminuição de custos de implementação de sistemas, (b) aproximação com o problema a tratar, (c) previsão de problemas e inconvenientes e proposta de possíveis soluções, (d) ferramentas de interconexão parcial com sistemas reais e (e) desenvolvimento de sistemas de controle e supervisórios.

1.3 OBJETIVOS

1.3.1 Objetivos Gerais

a) Estudar o problema de equilíbrio de um robô que anda de bicicleta em configuração humanóide para baixas velocidades (incluindo velocidade zero).
b) Projetar controladores que satisfaçam o critério de estabilidade, utilizando para isto ferramentas em *software* e ferramentas baseadas em arquiteturas reconfiguráveis.

1.3.2 Objetivos Específicos

Os objetivos específicos são os seguintes:

a) Propor um modelo que permita encontrar as equações dinâmicas do robô, para baixas velocidades, incluindo velocidade zero.
b) Desenvolver um simulador da planta baseado no modelo obtido.
c) Projetar controladores baseados na inteligência artificial (Lógica Difusa, Redes Neurais e Sistemas Neuro-Difusos) e em métodos de controle moderno (Realimentação de Estados e Regulador Quadrático Médio).
d) Apresentar resultados qualitativos e quantitativos relacionados com o desempenho dos controladores propostos.
e) Projetar controladores em hardware, especificamente do tipo nebuloso, utilizando um modelo de planta simulada no computador.
f) Comparar resultados de diferentes técnicas utilizadas e suas respectivas implementações.

1.4 METODOLOGIA DO TRABALHO

A metodologia do trabalho esta constituída pelas seguintes atividades:

a) Estudo das características e revisão bibliográfica do problema de estudo, estabilidade do robô ciclista.
b) Aproximações do robô com sistemas específicos (pêndulo duplo invertido interatuado) de estudo.
c) Obtenção da modelagem matemática, estudo de métodos numéricos de solução de equações diferenciais, estudo no espaço de estados de sistemas multivariáveis, estudo de técnicas de controle moderno e controle inteligente, projeto de controladores na plataforma de simulação em Matlab.
d) Realização de um projeto conceitual do robô ciclista.
e) Estudo de desenvolvimento de mundos virtuais usando Virtual Builder 2.0.
f) Desenvolvimento de um protótipo virtual do robô ciclista.
g) Estudo de projeto de sistemas embarcados em FPGAs da Xilinx.
h) Estudo e aprendizado da ferramenta computacional XFuzzy para criação e implementação de sistemas difusos.
i) Implementação de sistemas difusos baseados em FPGAs e utilização da técnica co-projeto de *hardware/software*.

j) Implementação do controlador difuso na FPGA atuando na planta simulada e o protótipo virtual no computador.

1.5 RESULTADOS OBTIDOS NO TRABALHO

O trabalho apresenta a modelagem e simulação de um pêndulo duplo invertido adaptado ao problema de equilíbrio de um robô ciclista, para isto foi necessário criar um sistema simulado baseado no modelo dinâmico e um protótipo virtual do robô, foram projetados cinco controladores: (1) controlador de realimentação de estados (RE), (2) controlador quadrático linear (LQR), (3) controlador difuso, (4) controlador neural e, finalmente, (5) controlador neuro-difuso. O desempenho dos controladores foram comparados em relação com sua resposta em diferentes condições iniciais, robustez às perturbações, variação de parâmetros e índices de desempenho. Foi implementado o controlador difuso no FPGA, usando periféricos personalizados em relação com a funcionalidade projetada (*IP cores*) e um processador Microblaze embarcado que coordenada os dados provenientes da planta simulada no computador, os periféricos criados, e o envio da variável de controle para a planta. Foram comparadas as velocidades de execução do algoritmo de controle em código de descrição de *hardware* (VHDL) usando co-projeto de *hardware-software*, e em código C usando a implementação em *software* do mesmo controlador difuso.

1.6 ESTRUTURA DO DOCUMENTO

O restante do trabalho está organizado da seguinte forma: no capítulo 2 são explicados os fundamentos da robótica móvel onde está desenvolvido o projeto do mestrado. No capítulo 3 é feito um projeto conceitual do robô ciclista. No capítulo 4 é apresentado o modelo matemático proposto e as equações dinâmicas do sistema. No capítulo 5 são apresentados os fundamentos teóricos dos controladores desenvolvidos, baseados em métodos de controle inteligente e métodos de controle modernos. No capítulo 6 apresenta-se os resultados obtidos e uma análise comparativa do desempenho dos controladores aplicados sobre a simulação da planta. No capítulo 7 apresenta-se a implementação de um controlador utilizando FPGAs, e os resultados do sistema de controle aplicado sobre um modelo tridimensional do robô ciclista utilizando para isto a ferramenta computacional *Virtual Reality Toolbox* de Matlab. Nos Anexos, são apresentadas informações sobre os algoritmos e projeto dos controladores implementados em Matlab, na plataforma de desenvolvimento da Xilinx ISE 10.1., e EDK 10.1.

2 REVISÃO BIBLIOGRÁFICA E ASPECTOS DE ROBÓTICA MÓVEL

Este capítulo trata da revisão bibliográfica referente ao tema do robô ciclista, objeto deste trabalho. Adicionalmente, no mesmo capítulo são tratados os temas de sensoriamento, arquitetura de controle, atuação, tecnologias utilizadas em arquiteturas e simulação de robôs, tendo em conta que os mesmos são fundamentais para o projeto de um robô ciclista, assim como o projeto do seu sistema de controle.

2.1 HISTÓRICO E ESTADO DA ARTE

Um dos primeiros trabalhos encontrados realizado por Macquorn Rankine em 1869, engenheiro civil e teórico termodinâmico, apresentou observações semi-quantitativas fundamentais sobre a inclinação e direção de velocípedes (primeiro nome conhecido para as bicicletas). Os veículos aqui considerados tinham eixos do guidão verticais, garfos retos e pedais que conduzem diretamente a roda dianteira. Em 1899, o meteorologista e matemático Frances Whipple escreveu sua tese sobre a dinâmica da bicicleta (Whipple, 1899). Whipple escreveu as equações de movimento não-lineares, as quais foram aceitas pela comunidade como corretas. Em 1900, Carvallo escreveu uma monografia sobre a dinâmica de monociclos (veículo com uma roda só) e bicicletas (Carvallo, 1900). Em 1910, os físicos Felix Klein e Arnold Sommerfield escreveram um livro sobre efeitos giroscópios no qual é incluído um capitulo sobre bicicletas (Klein e Sommerfeld, 1910). Em 1970, David Jones escreveu um artigo sobre sua pesquisa, principalmente sobre a geometria da parte frontal da bicicleta: guidão, garfo e roda dianteira e seus efeitos na estabilidade (Jones, 1970). Em 1972 Weir foi um dos primeiros pesquisadores em fazer uma revisão detalhada e comparar suas equações com modelos encontrados pelos pesquisadores anteriores, e escreve, baseado nos seus estudos, sua tese de doutorado (Tudelf, 2009).

Em 1987, Jim Papadópulus apresenta vários resultados com relação à dinâmica da bicicleta, e faz uma derivação compacta das equações tornadas lineares (Papadopulus, 1987), e em 1988, Mears e Kleins, estudantes de doutorado da *Urbana Champaign*, avaliam e confirmam as correções das equações feitas pelo Papadopulus e Hand. Em 2005, o trabalho de Åström, Klein e Lenarston, apresenta diferentes modelos, desde as

considerações mais simples até modelos mais realísticos utilizando *Modelica* (Zhihua *et al*., 2008) para simulação de múltiplos corpos, aonde são descritos projetos de bicicletas adaptadas para crianças com limitações físicas (Åström *et al.,* 2005; Meijaard *et al.,* 2007; Tudelf, 2009).

As bicicletas possuem um interessante comportamento dinâmico, elas são estaticamente instáveis, como os pêndulos invertidos, mas podem (baixo certas condições) serem estáveis com velocidades diferentes de zero, como foi mencionado e referenciado nos parágrafos anteriores. O objetivo deste trabalho esta dirigido ao estudo da estabilidade do robô ciclista para velocidades muito pequenas ou zero, onde é demonstrado que o controle de estabilidade da bicicleta pelo movimento do guidão para valores inferiores a uma velocidade crítica não é possível (Åström *et al.*, 2005).

Os pêndulos invertidos constituem uma família de sistemas, os quais têm sido estudados por muitos anos na engenharia de controle. Inicialmente, foram utilizados como banco de provas para aplicação de controle linear e não-linear. O caso mais estudado, sem dúvida, é o denominado controle de um pêndulo sobre um carrinho, conforme mostrado na Figura 2.1.a. O sistema consiste de uma barra que gira livremente por um de seus extremos mediante uma articulação situada no carrinho. A ação de controle é feita mediante a movimentação linear do carrinho com a qual se pretende deixar a barra na posição invertida (Drummond *et al.,*1999; Feitosa *et al.,* 1999; Miranda *et al.*, 2003; Leonor *et al.,* 2004; Yasunobu *et al.,* 2004; Morais *et al.*, 2005; Viguria *et al.*, 2006; Castro *et al.*, 2008). O pêndulo duplo invertido controlado mediante a movimentação do carrinho é uma extensão do sistema original de pêndulo invertido (Inoue *et al.*, 2006), como mostrado na Figura 2.1.b.

Outra configuração do pêndulo duplo é mostrada na Figura 2.1.c, sendo conhecida como *Arm Driver* ou *Pendubot* (Passos, 2006; Vieria e Nakahati*,* 2007). Este sistema tem um atuador no extremo de uma primeira barra (ombro), onde é aplicada diretamente a ação de controle, e no outro extremo (cotovelo) uma segunda barra que pode rotar livremente.

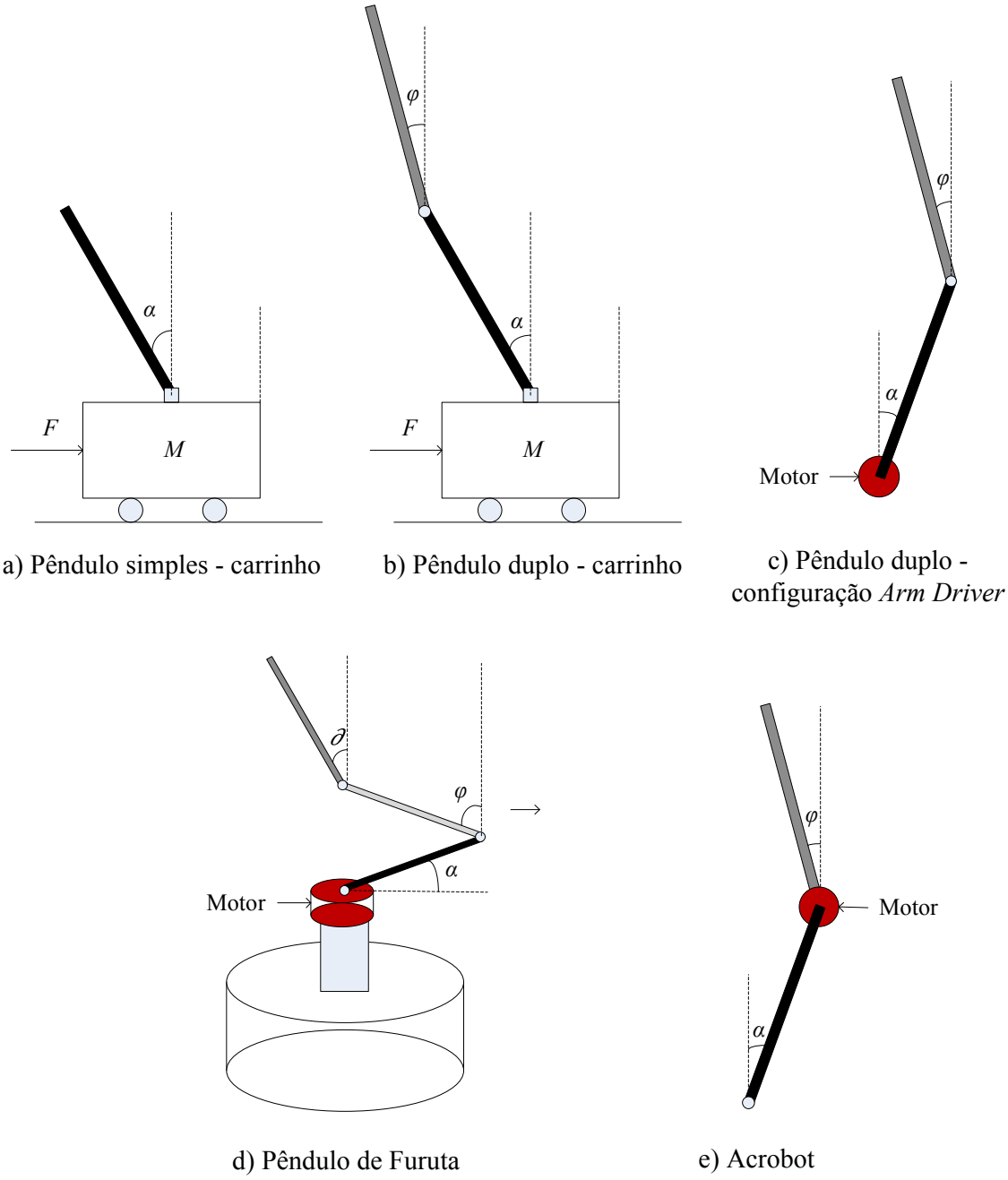
O pêndulo de Furuta mostrado na Figura 2.1.d, consiste de uma barra colocada sobre o plano horizontal *x-z* e que rota pela ação de um atuado. No outro extremo da barra é colocado o sistema do pêndulo duplo formado por outras duas barras que podem rotar

livremente no plano perpendicular com relação à primeira barra horizontal (Brockett e Hongyi, 2003).

Finalmente, é mostrado na Figura 2.1.e o sistema chamado de *Acrobot* em que a barra inferior possui a extremidade presa em um rolamento e na junção com a barra superior coloca-se o atuador. O *Acrobot* (*Acrobatic robot*) é um termo que foi criado na Universidade de Califórnia, em Berkeley, onde o primeiro estudo das propriedades de controlabilidade foi desempenhado por Murray e Hauser (Murray e Hauser, 1991). O *Acrobot* é um modelo altamente simplificado do desempenho de um ginasta humano sobre uma barra simples paralela (Murray e Hauser, 1991).

Takashima (1990) estudou problemas fundamentais sobre um robô ginasta: um ginasta pode manter um estado estável fazendo alguns truques sobre uma barra alta. Para incrementar a amplitude de balance são propostos dois métodos. O primeiro é levantar e empurrar para abaixo o centro de massas do sistema. O segundo método é o balanço oscilatório pelo movimento sinusoidal em freqüências próprias de um pêndulo considerado (Takashima, 1990).

Spong (1995) propôs uma estratégia de controle para o *swing-up* (balanço para levar o pêndulo desde a posição de equilíbrio estável até a posição invertida), baseado na escolha adequada de um conjunto de ganhos, e depois se faz um chaveamento para um controlador LQR (*Linear Quadratic Regulator*) que mantêm o pêndulo na posição invertida (Spong, 1995). Brown e Passino (1997) desenvolveram controladores inteligentes para o *swing-up* e para o balanço do *Acrobot*. São incluídos, neste caso, controladores nebulosos, nebulosos adaptativos. O mesmo trabalho inclui técnicas de otimização utilizando algoritmos genéticos (Brown e Passino, 1997). Por outro lado, Boone descreve um algoritmo de busca direta para encontrar trajetórias de *swing-up* para o *Acrobot*. O algoritmo usa uma busca antecipada que maximiza a energia total do *Acrobot* em uma janela de *N* passos (Boone, 1997).



a) Pêndulo simples - carrinho

b) Pêndulo duplo - carrinho

c) Pêndulo duplo - configuração *Arm Driver*

d) Pêndulo de Furuta

e) Acrobot

Figura 2.1- Diferentes tipos de pêndulos invertidos

Kobori *et al.* (2002), testam o problema de estabilidade do *Acrobot*, utilizando a inclusão de um algoritmo *child actor/critic* na parte do ator de um algoritmo *parent actor/critic* (Kobori *et al*., 2002). Kamio *et al.* (2004), propõem um novo método de segmentação de espaço de estados adaptativo baseados em uma rede neural *fuzzy-Art*. Em geral, o aprendizado por reforço foi utilizado para uma variedade de tarefas de controle, entre elas o balanço do *Acrobot* (Kamio *et al.*, 2004). In 2008, Lukasz apresenta os resultados de uma

simulação computacional, onde utiliza uma combinação entre redes neurais e um controlador LQR para resolver os problemas de *swing up* e balanço para o *Acrobot* (Lukasz *et al*., 2008).

Parte dos trabalhos propostos, anteriormente, considera o *Acrobot* aplicado ao sistema de braços robóticos manipuladores. Baseado na observação e na experiência dos humanos na hora de dirigir uma bicicleta, e em alguns casos sem a necessidade de assegurar o guidão com as mãos, é observado que um leve movimento do quadril pode permitir ao ciclista manter o equilíbrio. Neste trabalho, e considerando esta forma de manter o equilíbrio, é possível levar o problema de estabilidade do robô ciclista em baixas velocidades ou zero para o problema de estabilidade do pêndulo duplo interatuado, conhecido como *Acrobot.*

2.1.1 O problema do pêndulo invertido na indústria

Segway, um veículo eletrônico produzido pela fábrica *Segway Inc.*, usa duas rodas posicionadas uma ao lado da outra para avançar pela rua (Figura 2.2). A pessoa que dirige deve ir parada e o veículo arranca, freia ou retrocede dependendo de pequenas mudanças na posição de quem esta dirigindo, em quanto pode girar e fazer curvas usando a direção. *Segway* têm cinco sensores giroscópicos na sua estrutura o que permite identificar quando ocorrem mudanças na inclinação de quem dirige.

Funciona com motores elétricos que alcançam uma velocidade de até 20 km/h, utilizam baterias de lítio especiais as quais podem ser carregadas em qualquer uma tomada doméstica. Atualmente (http://www.soho.com.co), seu preço varia entre UD$5.000 e UD$7.000.

Figura 2.2- *Segway*, veículo desenvolvido por *Segway Inc.* (http://www.soho.com.co).

2.1.2 O problema do robô ciclista na indústria

Diferentes empresas tem se visto envolvidas na criação de robôs para testar os diferentes tipos de produtos que comercializam, entre as quais a empresa *Murata Boy manufacturing* e *Castrol*.

A *Murata Manufacturing Co. Ltd.*, criou os populares robôs *Murata Boy* e *Murata Girl* (Figura 2.3), conhecidos pelos japoneses como *Murata Seisaku Kun* e *Murata Seiko-chan* respectivamente. Eles não só conseguem-se equilibrar quando se deslocam para frente e para trás, mas também são capazes de detectar os obstáculos com os seus sensores e se desviar deles. Eles usam um giroscópio para se manter em equilíbrio além de sensores de proximidade, de colisão, velocidade angular e de inclinação. Com 50 centímetros de altura e 5 kg de peso, os robôs são equipados com *bluetooth* e têm uma câmera embutida que transmite vídeos em tempo real.

Figura 2.3- *Murata Girl* robô e *Murata Boy* robô (http://www.murataboy.com)

A fabricante de óleos lubrificantes Castrol precisava de um piloto para testar seus produtos para motos esportivas de grandes cilindradas. Os candidatos precisavam preencher alguns pré-requisitos, como a capacidade de manter acelerações constantes por períodos precisos, mudar as marchas tão rapidamente quanto o câmbio agüentasse e fazer isto por longos períodos e repetidas vezes. Ou seja, um piloto de testes virtualmente incansável.

Flossie (Figura 2.4), um robô magrelo atingiu os requerimentos, o mesmo consegue repetir o mesmo comportamento inúmeras vezes, permitindo que os engenheiros avaliem e comparem cada uma das alterações feitas nos produtos que estão desenvolvendo. Flossie ainda não consegue se equilibrar na moto, operando sempre ancorado. O robô possui um modo de auto-aprendizado que permite que ele aprenda rapidamente a pilotar novas motos, que podem ser desde as supermáquinas da moto *GP*, até pequenas motonetas. Um programa de inteligência artificial avalia a combinação de marchas de cada câmbio, detecta a sensibilidade da embreagem e a resposta do motor, permitindo que ele seja um piloto quase perfeito para qualquer máquina (Inovação Tecnológica, 2009).

Figura 2.4- *Flossie*, robô de testes da empresa Castrol (Inovação Tecnológica, 2009)

2.2 ASPECTOS DE ROBÓTICA MÓVEL

O projeto de um robô móvel é uma tarefa complexa, que envolve a unificação de várias tarefas mecânicas e eletromecânicas. De forma oposta aos robôs manipuladores utilizados na manufatura, onde o ambiente é altamente estruturado, os robôs móveis devem ser projetados e programados para realizar atividades predefinidas em um ambiente sobre o qual o agente só possui um conhecimento incerto e parcial (Floreano *et al.*, 1998). Desta forma, os aspectos de sensoriamentos, controle e atuação são importantes para garantir ao robô a autonomia e mobilidade que ele necessita para a execução da tarefa para o qual ele foi programado.

2.2.1 Sensoriamento

O sensor é um dispositivo capaz de monitorar a variação de uma grandeza física e transmitir esta informação a um sistema de indicação que seja inteligível para o elemento de controle do sistema (García e Alvarez, 2005). Os sensores estão vinculados aos sistemas de controle e os sistemas de controle com um processo acionado por um dispositivo de controle, que determina o resultado desejado e, ao longo do tempo, indicam o resultado obtido e corrige sua ação para atingir, o mais rápido possível, o valor desejado.

Desta maneira, para que um robô móvel possa se deslocar é preciso que o sistema de controle acione os atuadores de forma adequada, fazendo com que o robô se desloque no ambiente de uma forma robusta e segura. Para isto, o robô precisa perceber o ambiente, o que pode ser obtido a través do uso de diversos tipos de sensores (Heinen, 2007).

2.2.2 Classificação de Sensores

É preciso que o robô conheça tanto seu próprio estado como o estado do seu entorno. A informação que esta relacionada com o estado do robô (especialmente com as informações de suas articulações) é especificada pelos denominados sensores internos, em quanto á informação relacionada com o estado de seu entorno é especificada pelos denominados sensores externos. O esquema da Figura 2.5 apresenta a classificação dos sensores e algumas das grandezas medidas.

Os sensores observam de forma parcial, o estado do sistema robô-ambiente. Estes tipos de sensores podem ser ativos ou passivos, os ativos consomem energia de forma propositada (pulsos) de modo a captar (por reflexão) a informação de interesse. Os sensores passivos não utilizam energia para obter reflexão detectável, mas podem consumi-la para se manterem operacionais (García e Alvarez, 2005).

A Tabela 2.1 mostra os critérios e as definições para escolha dos sensores que devem ser tidas em conta de acordo com as necessidades e as variáveis físicas á serem medidas.

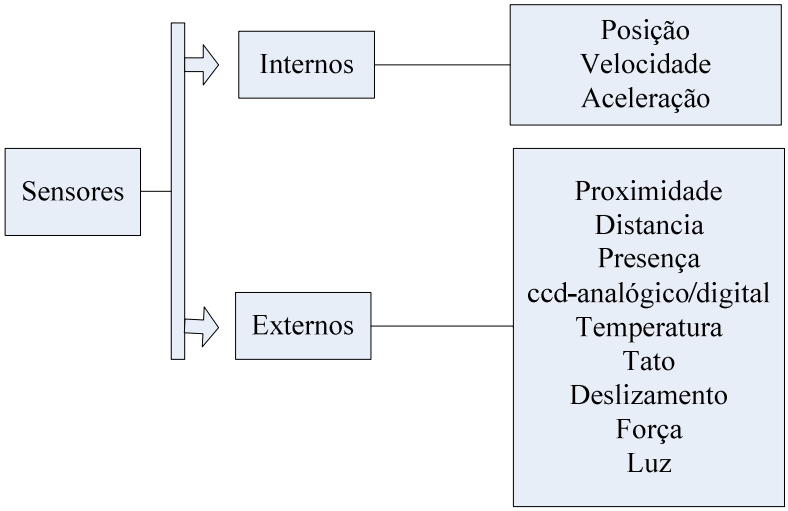


Figura 2.5- Classificação dos sensores em relação com sua função.

2.2.3 Arquitetura de Controle

O controle de um robô se refere ao jeito no qual o sensoriamento e a ação do robô são coordenados. Para integrar sensores e atuadores há a necessidade de uma arquitetura que seja responsável pelo controle das ações realizadas pelo robô.

Uma arquitetura é um sistema composto pelo arranjo, pela comunicação e pelo fluxo de dados entre componentes cuja estrutura e integração permitem realizar funções que os componentes individualmente não poderiam efetivamente realizar (Dias *et al.,* 2006).

Tabela 2.1- Critérios comuns utilizados para avaliação de sensores

Critério	Definição
Sensibilidade	Razão entre a taxa de mudança dos valores de saída pela mudança dos valores de entrada.
Linearidade	Medição da constância da taxa de saída com relação à entrada.
Faixa	Medição entre o valor mínimo e máximo
Tempo de resposta	Tempo decorrido para que uma mudança nas entradas seja percebida como mudança estável nas saídas.
Precisão	Medição da diferença entre valores medidos e reais.
Repetibilidade	Medição da diferença entre duas medidas sucessivas sob as mesmas condições.
Resolução	Número de medidas de valores diferentes possíveis dentro de uma faixa.
Tipo de Saída	Movimento mecânico, tensão, corrente, pressão, nível hidráulico, intensidade luminosa, etc.

Uma arquitetura deve facilitar o desenvolvimento de sistemas robóticos provendo restrições benéficas no projeto e na implementação da aplicação desejada, servindo inclusive para a integração de módulos que sejam desenvolvidos independentemente. As arquiteturas de controle para robôs móveis podem se classificar em relação com sua estrutura, com o raciocínio, e com a decomposição e encapsulamento de comportamentos e módulos funcionais, conforme mostrado no esquema da Figura 2.6 (Pieri, 2002).

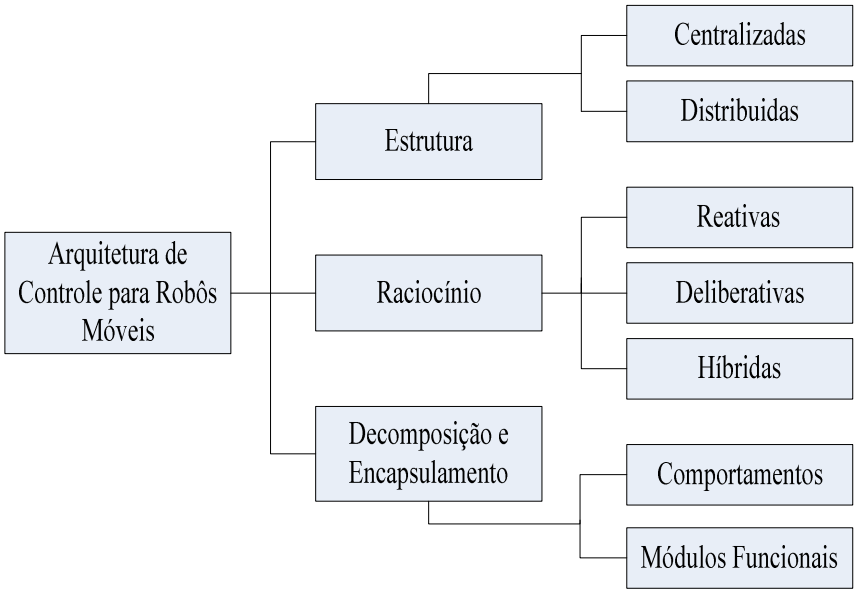


Figura 2.6- Classificação de arquiteturas para robôs móveis.

2.2.3.1 Estrutura

As arquiteturas podem ser classificadas de acordo com a estrutura em que são organizados os módulos interconectados e pela forma de processamento desses módulos em centralizadas e distribuídas (ver Figura 2.6). No caso da arquitetura centralizada, um processador central único executa todo o processamento, agregando, portanto, um melhor custo benefício que a arquitetura distribuída, por utilizar um único processador ao invés de vários. Por outro lado, a arquitetura centralizada oferece uma confiabilidade menor que a arquitetura distribuída, uma vez que uma falha no processador central compromete todo o sistema. A arquitetura distribuída é baseada em sistemas sensoriais e controladores distribuídos simples, interligados em uma rede de sensores e atuadores, na qual nenhum sistema individual está em nível hierárquico superior a qualquer outro (Dias *et al.,* 2006).

a) Estrutura Centralizada

Os robôs com uma estrutura centralizada apresentam as seguintes características (Soarez, 1999):

- A tomada de decisões é de âmbito local, ou seja, as informações sensoriais para processamento estão no próprio robô.
- O processamento da informação do robô pode ser baseado em técnicas de Inteligência Artificial (IA), tais como: Redes Neurais, Lógica Difusa, Sistemas Híbridos, etc.
- O robô age de forma isolada no ambiente.
- Baseia-se unicamente nas informações colhidas durante sua incursão no ambiente para agir.
- Não há nenhum tipo de comunicação com outras entidades do meio.
- As ações processadas surgem da interação direta com os objetos do ambiente.

b) Estrutura Distribuída

Os robôs com uma estrutura distribuída apresentam as seguintes características (Soarez, 1999):

- O robô não age de forma isolada no ambiente.
- Ele faz parte de uma sociedade de agentes onde cada entidade tem uma tarefa determinada.
- Esta abordagem surge da Inteligência Artificial Distribuída (IAD) que estuda o conhecimento e os métodos de raciocínio que podem ser necessários ou úteis para que agentes/robôs participem de sociedades de agentes robóticos.
- Para problemas realmente complexos a única possibilidade de solução é a solução distribuída.
- A idéia de distribuição na execução de tarefas se dividiu em dois enfoques: (a) Solução Distribuída de Problemas (SDP) e (b) Sistemas Multiagentes (SMA).

2.2.3.2 Aspectos de raciocínio

Segundo o método de raciocínio as arquiteturas podem ser divididas em reativas, deliberativas e híbridas.

a) Arquitetura Reativa

Uma arquitetura reativa é composta por uma série de comportamentos que relacionam certas condições sensoriais a um conjunto de ações do robô, organizados em camadas, além de ser dotada de métodos que garantam a devida coordenação destes comportamentos. Esta arquitetura produz robôs adequados para operação em tempo real, já que a simplicidade dos comportamentos reativos favorece uma alta velocidade de processamento computacional. Esta arquitetura baseia-se na decomposição da inteligência em comportamentos individuais, gerando módulos que coexistem e cooperam para a emergência de comportamentos mais complexos (Pieri, 2002; Heinen 2007). As saídas sugeridas pelo comportamento com a mais alta prioridade são então utilizadas para controlar os atuadores do robô.

Um exemplo da arquitetura reativa é a arquitetura subsunção (*subsumption*) (Brooks, 1986), na qual o robô age baseando-se na leitura de seus sensores, o sistema de controle é constituído de diversos comportamentos executados em paralelo (ver Figura 2.7).

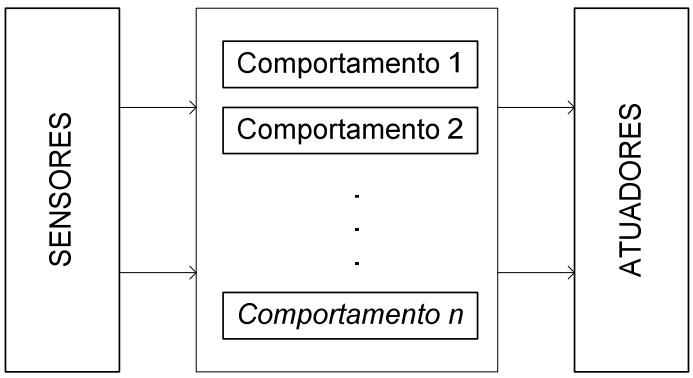


Figura 2.7- Arquitetura Reativa

b) Arquitetura Deliberativa

Planejar (ou deliberar) uma tarefa consiste em determinar as ações e seqüências necessárias para conseguir sua execução no contexto de certo modelo de mundo. Assim um robô inteligente pode tentar construir um modelo interno representando o ambiente no qual está

inserido. Para o desempenho de uma atividade do robô, ele segue as seguintes etapas: (a) explora as soluções possíveis para a execução da tarefa de acordo com o modelo de ambiente interno, (b) baseado nesta representação um plano ao longo prazo é elaborado, isto resulta em uma serie de ações que o robô deve executar para alcançar o seu objetivo, e (c) executa as ações por médio dos atuadores (Pieri, 2002).

Uma tarefa pode ser descomposta em sub-tarefas seguindo uma arquitetura hierárquica, o qual significa que o planejamento é subdividido em módulos funcionais que dependem de informações espaciais e de restrições temporais (Figura 2.8).

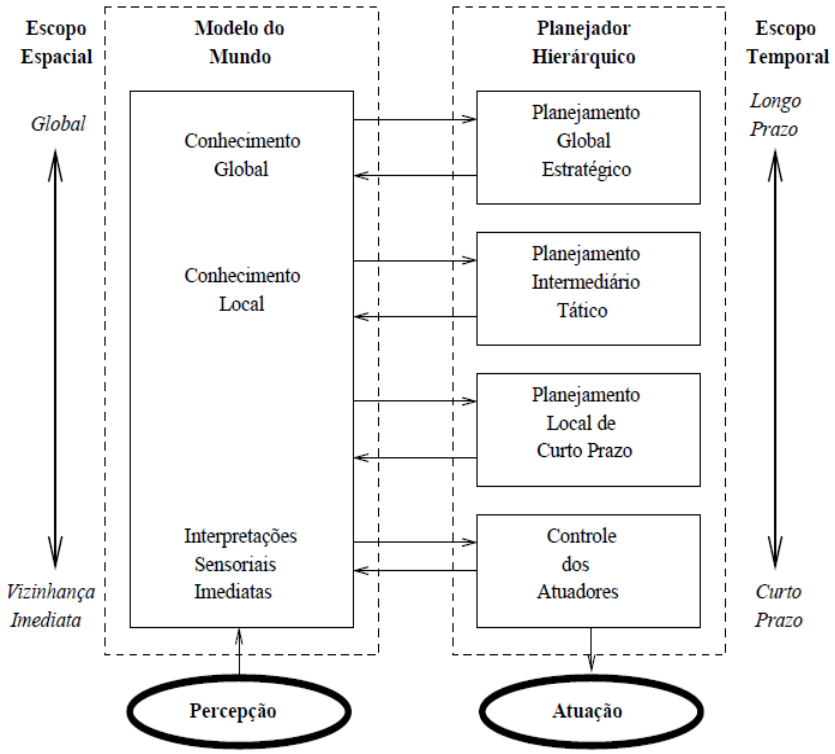


Figura 2.8- Arquitetura Deliberativa (Pieri, 2002)

O planejador é dividido em quatro níveis hierárquicos: (a) nível de planejamento global estratégico, (b) nível de planejamento intermediário tático, (c) nível de planejamento local de curto prazo e (d) nível de controle do atuador. Todos eles têm acesso ao modelo do mundo de acordo com as informações úteis para a geração de ações. Note que do nível superior da hierarquia ao inferior, cresce a restrição de tempo na resposta e diminui o espaço físico de interesse (Pieri, 2002; Heinen, 2007).

c) Arquitetura Híbrida

Nenhuma estratégia é satisfatória isoladamente, e devem ser levadas em consideração para produzir um sistema flexível, robusto e inteligente. Como é usual, as configurações híbridas buscam aproveitar as melhores características de cada arquitetura, para isto incorpora-se um elemento de planejamento sobre a definição e seleção de comportamentos reativos individuais.

A organização de uma arquitetura híbrida deliberativo-reativa pode ser descrita na forma planejar, sentir-agir, ou seja, primeiro planejar, depois sentir e agir em um único passo. O robô primeiro planeja como cumprir uma missão (usando um modelo de mundo) ou uma tarefa, e então dispara diferentes comportamentos (sentir-agir) pertencentes a um conjunto pré-definido para executar o plano. Os comportamentos são executados até que o plano se complete. Após isso, o planejador gera um novo conjunto de comportamentos e o procedimento recomeça (Britto, 2008). Isto significa que por médio da incorporação da habilidade de raciocínio baseado em modelos internos do mundo (deliberação), estas arquiteturas permitem a reconfiguração dinâmica de sistemas de controle reativo.

2.2.3.3 Decomposição e encapsulamento

As arquiteturas podem ser decompostas em comportamentos ou módulos funcionais, em sistemas baseados em comportamentos, respostas-comportamentais as quais estão explícitas no projeto, e não existe qualquer meta representada explicitamente, ou seja, a meta emerge do funcionamento normal do agente. Em arquiteturas baseadas em módulos funcionais e respostas-comportamentais não estão explícitas na arquitetura, ao invés disso, elas emergem do planejador com as dadas metas e o modelo de mundo particular que foi construído a partir dos dados sensoriais.

A decomposição em comportamentos encapsula em um único módulo várias funções (percepção, planejamento e execução) como mostrado na Figura 2.9. Um comportamento pode-se caracterizar por (Pieri, 2002):

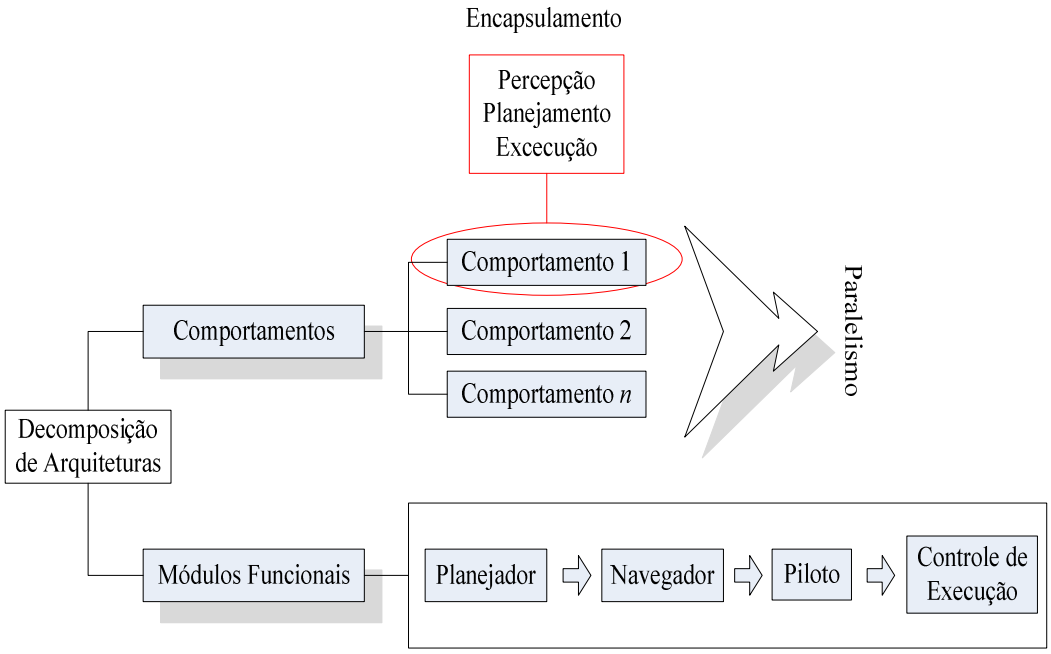


Figura 2.9- Esquema de arquiteturas baseadas em comportamentos e módulos funcionais.

- Receber entradas dos sensores.
- Gerar saídas para os atuadores.
- Resolver uma atividade concreta.
- Resolver uma tarefa completa, mesclando vários componentes simples. A mescla dos comportamentos admite muitas variações: ponderar os comportamentos, inibir comportamentos, alternar comportamentos.

No caso da decomposição por módulos funcionais é freqüente estruturar os robôs nos seguintes níveis (Pieri, 2002), como mostrado na Figura 2.9.

- Planejador: planeja um caminho que cumpra a missão encomendada pelo operador.
- Navegador: a partir das informações do ambiente e do planejador, refina-se o caminho reduzindo erros.
- Piloto: a partir da trajetória gerada pelo navegador e da informação disponível neste nível geram-se seqüências de comandos de controle.

- Controle de execução: executa os comandos enviados pelo piloto transformando-os em ações que administram os atuadores.

2.2.4 Atuação

Os atuadores são elementos ativos dentro de um sistema de automação, cuja função é transformar sinais de controles em energia mecânica, a fim de que se realize algum trabalho. Os atuadores são dispositivos responsáveis pelo movimento e articulação das partes móveis do robô. Os atuadores utilizados em robótica são classificados em função da energia que utilizam para seu funcionamento.

Os três principais tipos de atuadores são: pneumáticos, hidráulicos e elétricos. Os acionamentos pneumáticos e hidráulicos se baseiam no uso de energia liberada por fluidos em movimento, em ambos os casos, será necessária a instalação de bombas e compressores, filtros, acumuladores, equipamentos para refrigeração, válvulas, etc.

Os atuadores baseados em motores elétricos são sem dúvida os mais utilizados em projetos de robótica, isto devido ao fato de existirem uma gama grande de motores com as mais diversas características, tornando-os flexíveis o suficiente para atender quase todas as aplicações. A classificação dos atuadores é mostrada na Tabela 2.2.

Tabela 2.2- Quadro comparativo dos diversos tipos de atuadores usados para robôs (Pieri, 2002)

Tipo / **Tópico**	**Pneumático**	**Hidráulico**	**Elétrico**
Energia	Ar comprimido	Óleo Mineral	Corrente elétrica
Opções	Cilindros Motor de aletas Motor de pistão	Cilindros Motor de aletas Motor de pistão axial	Corrente contínua Corrente alternada Motor de passo
Vantagens	Baratos Rápidos Sensíveis Robustos	Rápidos Alta relação peso-potência Autolubrificantes Alta capacidade de carga Estabilidade frente à cargas estáticas	Precisos Confiáveis Fácil controle Fácil instalação Silenciosos
Desvantagens	Dificuldade de controle contínuo Instalação especial (filtros, compressores, etc.) Ruidoso	Difícil manutenção Instalação especial (filtros, eliminadores, de ar) Freqüentes fugas Caros	Potência limitada

2.2.5 Tecnologias utilizadas em Arquiteturas

Além das arquiteturas de controle e as metodologias existentes para desenvolver as estratégias de controle dos robôs móveis, existem diferentes formas de implementação para conformar a unidade de central de processamento (CPU, *Central Processing Unit*) onde vai ser desenvolvido o algoritmo de controle, ou seja, é preciso saber a disposição do elemento físico que vai se encarregar de receber os sinais de percepção do ambiente, executar as tarefas correspondentes de acordo com os objetivos para os quais foi projetado o robô, e gerar os sinais necessários para os atuadores modificarem o estado atual do robô (Britto, 2008).

Diversas tecnologias e abordagens devem ser levadas em consideração na implementação de uma arquitetura de *hardware* e *software* para controle de um robô móvel. Uma arquitetura deste tipo especifica o ambiente e fornece os meios para o desenvolvimento de diversas aplicações em uma plataforma robótica.

Existem duas abordagens utilizadas para organização dos processadores de um sistema. Elas são chamadas de Processamento Centralizado e Processamento Distribuído.

a) Processamento Centralizado

Os sistemas de processamento centralizado (Sistemas Monoprocessados) concentram toda a computação em apenas uma CPU. Em tais sistemas, a execução paralela real de processos não é possível, porém consegue-se atingir um pseudo-paralelismo através do chaveamento de processos efetivado pelos sistemas operacionais (Tanenbaum, 2006).

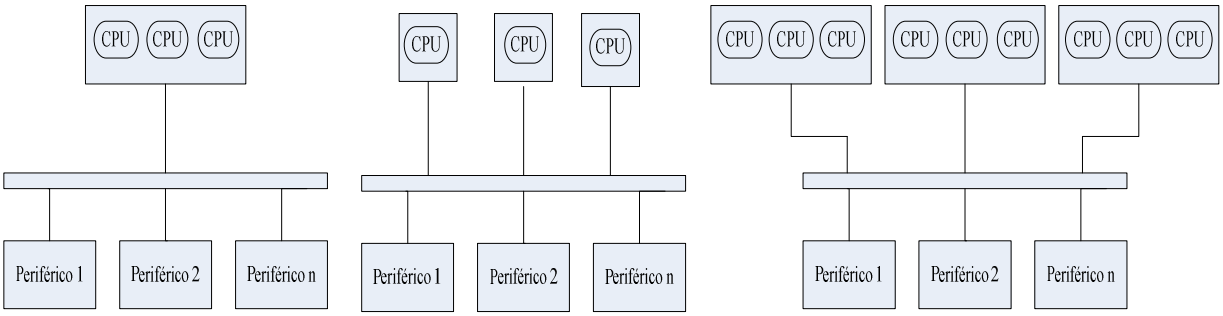
Estes sistemas possuem como vantagem a simplicidade de implementação e manutenção. Como desvantagens podem-se citar a impossibilidade de utilizar paralelismo real, bem como a inoperância de todo o sistema caso a unidade de processamento pare de funcionar.

b) Processamento Distribuído

Os sistemas de processamento distribuído (Sistemas Multiprocessados) se caracterizam pela existência de duas ou mais CPUs. A existência de múltiplos processadores em um sistema permite a execução de processos em paralelismo real. Esses sistemas podem executar processos independentes em processadores diferentes, bem como podem utilizar

os vários processadores para melhorarem o tempo de execução de um único processo. (Britto, 2008).

Existem diferentes formas de distribuir CPUs para um sistema, esta distribuição pode ser feita a través de: (a) uma mesma plataforma (sistema multiprocessado utilizando mais de uma CPU em plataforma única) (Figura 2.10.a.), (b) em plataformas distintas onde cada CPU possui memória distinta (sistema multiprocessado utilizando uma CPU por plataforma única) (Figura 2.10.b.) e (c) uma combinação das duas abordagens (a) e (b) (sistema multiprocessado utilizando mais de uma CPU por plataforma) (Figura 2.10.c.).



a. Utiliza mais de uma CPU em plataforma única

b. Utiliza uma CPU por plataforma única

c. Utiliza mais de uma CPU por plataforma única

Figura 2.10- Diferentes abordagens de distribuição de CPUs para um sistema

2.2.6 Simulação de robôs móveis

A simulação pode ser definida como a técnica de construir um modelo de um sistema real ou proposto afim do que o comportamento do sistema possa ser estudado. Ela objetiva a visualização e verificação do desempenho de um robô, determinando características importantes e funcionamento. Mas simulações no computador são complexas, levam tempo e não são simples de se criar, já que sempre é possível deixar de fora algum parâmetro que depois, mostra-se importante no mundo real (Zheng, 1993).

O papel da simulação de um robô móvel pode ser apreciado de forma adequada dando uma olha nas dificuldades do projeto de sistemas inteligentes baseados em sensores.

Um sistema típico real obtém informações do ambiente através de um sistema de percepção e gera ações para um sistema de locomoção. Inicialmente, encontra-se uma

primeira dificuldade relacionada com a obtenção de informação precisa dos sensores porque eles são imprecisos, têm um número de erros e são acumuláveis com a movimentação do robô. Além disto, a precisão do controle do sistema depende das características do sistema de locomoção que apresenta outro número de dificuldades relacionadas, por exemplo, com o escorregamento das rodas, valor ótimo de período de amostragem e problemas de estabilidade com o algoritmo de controle. Finalmente, o estado de percepção do ambiente é estocástico por natureza. Assim, no processo de programação de algoritmos de controle não é possível predizer o estado exato do robô e o ambiente de operação (Zheng, 1993).

Outros problemas estão relacionados com as seguintes considerações:

- É geralmente perigoso executar um programa o qual está incompleto ou possivelmente com erros sobre um robô real.

- A tarefa de desenvolvimento de um ambiente de programação para facilitar testes experimentais de robôs móveis não é uma tarefa trivial. Por exemplo, o ambiente de programação deve suportar a definição de muitos padrões de ações para superar diversas situações e condições do ambiente.

- É difícil reconfigurar o *software* para diferentes robôs e ambientes de operação.

- Um programa deve ser testado para confirmar que trabalha bem em ambientes de complexidade razoável.

O desenvolvimento de programas usando um simulador não é perigoso porque os testes realizados não representam situações de risco reais diminuindo custos de operação. Além disto, os algoritmos podem ser aperfeiçoados, depurados, e depois, usados no robô. O software pode ser facilmente mudado para um robô com características diferentes simplesmente especificando arquivos ou bibliotecas de definição de parâmetros apropriados, onde a geometria e as características dos sensores podem ser inicializadas. Igualmente, é possível definir uma base de dados de ambientes de operação e escolher quando adicionar ou remover um deles.

2.3 CONCLUSÃO DO CAPÍTULO

Este capítulo apresentou uma revisão bibliográfica acerca dos trabalhos desenvolvidos para modelagem e controle de robôs bicicletas, o que se tem feito em relação aos pêndulos invertidos, algumas aplicações dos robôs e dos pêndulos na indústria. Também são apresentados fundamentos conceituais necessários para o projeto de um robô móvel e suas unidades constitutivas. São tratados alguns aspectos importantes em quanto às tecnologias e às vantagens e desvantagens da construção e simulação de robôs. Estes conceitos suportam a base teórica para a construção conceitual do robô ciclista e da implementação da arquitetura de controle que será tratada no próximo capítulo.

Os tipos de arquitetura de controle misturam aspectos importantes (paralelismo e processamento seqüencial) que justificam a utilização de dispositivos que satisfaçam os requerimentos próprios de cada uma, em geral todas as arquiteturas são constituídas de blocos funcionais que são executados de forma seqüencial e/ou paralela, por exemplo, a arquitetura reativa já apresenta a divisão de comportamentos associados com tarefas que podem trabalhar em forma paralela para aplicações em tempo real.

É observado também que por meio da utilização de arquiteturas robóticas híbridas, ou seja, a incorporação da habilidade de raciocínio baseado em modelos internos do mundo é possível a reconfiguração dinâmica de sistemas de controle reativo, todos estes aspectos importantes atuais que são objeto de estudo de dispositivos microeletrônicos tais como as FPGAs.

3 PROJETO CONCEITUAL DO ROBÔ CICLISTA

3.1 CONSIDERAÇÕES INICIAIS

O projeto conceitual é a fase inicial do processo de projeto do robô, onde se analisam as necessidades que devem ser supridas pelo projeto e as transformam em parâmetros de projeto, gerando um conceito de solução que será detalhada ou modificada nas demais fases do projeto. Atualmente, busca-se sistematizar esta fase, caracterizada pela síntese da idéia foco da solução, para emprego de ferramentas de auxilio computacional, buscando assim integrar com as demais fases do projeto (Almeida, 2000).

Um protótipo virtual é um aspecto da tecnologia de informação que permite a análise da forma, do movimento e de fatores humanos no projeto conceitual com auxílio do computador. Isto facilita a comunicação entre diferentes disciplinas da engenharia durante o projeto preliminar, e também proporciona visualizações gráficas que ajudam na concepção final do robô.

Analisando a aplicação de robôs ciclistas observa-se que os mesmos podem ser usados para trabalhos de inspeção, monitoramento e vigilância. Além disto, seus corpos estreitos permitem deslocamentos ao longo de superfícies bem delgadas e alcançarem lugares de difícil acesso. Uma plataforma robótica como a proposta neste trabalho vai permitir o teste de diferentes configurações em quanto a projeto de controladores de equilíbrio, controladores de locomoção e planejamento de trajetória, detecção de obstáculos, sistemas de navegação e diferentes arquiteturas de controle desenvolvidas sobre sistemas reconfiguráveis.

O projeto conceitual do robô ciclista tem como objetivo propor um sistema flexível dotado de atuadores e sensores inicialmente tratados sobre um problema específico como o controle de equilíbrio para velocidades nulas ou próximas de zero, mas com a possibilidade e os meios estruturais necessários para testar outras possibilidades como o controle mediante a movimentação do guidão quando as velocidades são diferentes de zero.

3.2 TIPOS DE CONFIGURAÇÃO

Dois tipos de configuração podem ser desenvolvidas para o robô, um robô ciclista antropomórfico (humanóide), ou um robô bicicleta em configuração dedicada. Uma configuração dedicada refere-se ao caso no qual a bicicleta é o próprio robô. Esta utiliza geralmente como meio de controle do equilíbrio a movimentação do guidão, a outra utiliza a movimentação do corpo do robô como contra-peso (Tanaka e Murakami, 2004).

A vantagem principal dos robôs humanóides é que podem trabalhar diretamente no mesmo ambiente que os humanos sem que devam ser feitas modificações no ambiente. Em terrenos sem obstáculos abruptos o emprego de robôs com rodas é viável, porem quando ocorrem desníveis acentuados no terreno o emprego de robôs com pernas é mais indicado.

Uma primeira razão para usar a configuração humanóide é encontrada no ponto de vista prático, estético e funcional. Um corpo humano é uma forma ideal para se desenvolver atividades em um ambiente feito por humanos. A vantagem de robôs humanóides são mais obvias quando se tem uma interação direta com os humanos, os humanos utilizam diferentes formas de se comunicar, por exemplo palavras, a direção do olhar, expressões faciais, gestos e linguagem corporal desde a infância. Uma outra vantagem esta relacionada com os movimentos dos robôs humanóides que são fáceis de predizer pelos humanos o que facilita a cooperação baseada em formas diferentes de comunicação, diferentes a oral.

Uma segunda razão vem dos pesquisadores da IA (Inteligência Artificial). Neste caso, a construção de protótipos inteligentes é um método importante para entender e desenvolver tecnologias inspiradas na inteligência humana (Miura *et al.*, 2008).

3.3 CONTROLE DE EQUILIBRIO

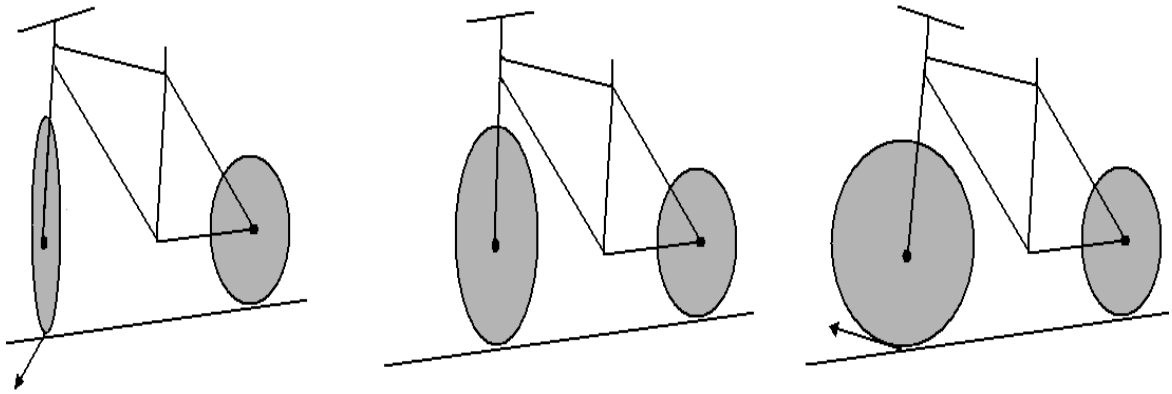
3.3.1 Controle de equilíbrio mediante a movimentação do guidão

O manejo de uma bicicleta envolve uma complicada interação entre a força centrífuga, a força gravitacional e o torque aplicado no guidão, todos regidos pela geometria da bicicleta.

A força centrífuga pode contrabalançar a bicicleta para um lado quando se movimentar o guidão na direção do tombamento. O giro produzido pelo guidão permite que a força gravitacional se equilibre a força centrífuga, evitando o tombamento.

O processo de contrabalancear um giro é ilustrado na Figura 3.1, e pode ser dividido em cinco etapas:

- Inicia um giro aplicando um torque ao guidão, assim o guidão é levado inicialmente para a esquerda (Figura 3.1.a.).
- A roda dianteira é dirigida para esquerda, a taxa na qual o ângulo de manejo aumenta se fixa principalmente pelo momento de inércia da roda I_s, o garfo e o guidão ao redor do eixo de giro do garfo, e pelo rastro (ou fuga). Como agora a bicicleta está girando para esquerda, um torque da força centrífuga inclina o ciclista e a bicicleta para a direita. A ação giroscópica também inclina a bicicleta para a direita, mas este efeito pode ser considerado depreciável (Figura 3.1.b.).
- Transmitido pelo garfo, o incremento da inclinação tenta inclinar a roda dianteira também, aqui a ação giroscópica ganha importância porque a roda responde a este efeito de torção tentando dirigir-la para a direita e contrabalanceando o torque do guidão. O ângulo do guidão para de se incrementar.
- O torque de inclinação supera o torque do guidão, e o ângulo de manejo da roda diminui. Note que a inclinação continua se incrementando porque a bicicleta ainda gira para esquerda (Figura 3.1.c.).
- Como a bicicleta tem adquirido uma velocidade de inclinação substancial, o incremento da inclinação não pode parar imediatamente. Conduzido pela inclinação cada vez maior, o ângulo de manejo da roda passa suavemente a través de zero e então vai para a direita. Um torque centrífugo em direção contraria é gerado, parando o aumento da inclinação e balançando eventualmente os esforços de torção gravitacionais (Fajans, 2000).



a. Movimento do guidão em sentido antihorário

b. Bicicleta em posição vertical

c. Movimento do guidão em sentido horário

Figura 3.1- Controle de posição invertida de uma bicicleta mediante a movimentação do guidão

3.3.2 Controle mediante a movimentação do ciclista

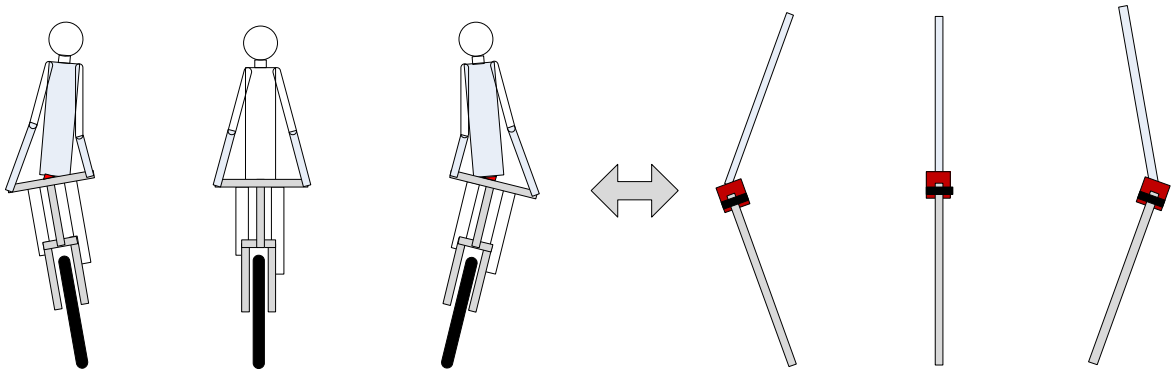
Alternativamente ao caso anterior, a inclinação da bicicleta pode ser contrabalanceada levando o quadril na direção contraria do giro da bicicleta, assim através do movimento do quadril é possível evitar o tombamento da bicicleta sem a ação das mãos.

Esta aproximação é o objeto de estudo deste trabalho, o objetivo é entender e projetar controladores que permitam manter o sistema ao redor da posição vertical invertida. Este mesmo objetivo é trabalhado no sistema denominado *Acrobot* (*Acrobat Robot*). O *Acrobot* é um sistema altamente simplificado de um ginasta humano sobre uma barra paralela simples. O sistema consiste de um manipulador atuando sobre duas barras no plano vertical. Este sistema pode representar de uma forma adequada o comportamento mostrado na Figura 3.2, associando a bicicleta e a parte inferior do robô ciclista como uma primeira barra e a parte superior do robô ciclista com uma segunda barra. Neste caso, o atuador corresponderia ao movimento do quadril.

O processo de controle pode ser descrito em duas fases:

- Se o sistema esta caindo em sentido anti-horário, a movimentação do tronco do ciclista mediante um torque aplicado no quadril em sentido anti-horário deve levar o sistema para a posição invertida.

- Se o sistema esta caindo para em sentido horário, a movimentação do tronco do ciclista mediante um torque aplicado no quadril em sentido horário deve levar o sistema para a posição invertida.



a. Robô Ciclista b. Acrobot

Figura 3.2- Aproximação do robô ciclista com o *Acrobot*

3.4 DESCRIÇÃO DO SISTEMA, RESTRIÇÕES E CONSIDERAÇÕES

Um modelo detalhado de um robô bicicleta é complexo porque o sistema tem muitos graus de liberdade, e a geometria é restringida. Alguns aspectos importantes para considerar são: (a) quais partes da bicicleta e do robô vão ser consideradas no modelo, (b) como tratar a elasticidade das partes da bicicleta, (c) quão complicado vai ser o modelo do ciclista que vai ser considerado, e (d) como tratar a interação das rodas com o chão. A maior dificuldade de um robô que usa rodas como meio de locomoção está nas irregularidades do terreno quando estas são maiores que o raio das rodas. A aceleração e forças longitudinais requeridas na freada, balanço e giro dependem das forças laterais. Um bom entendimento dessas forças é necessário para fazer considerações apropriadas com relação á modelos validos em condições de movimento (Åström *et al.*, 2005).

Alguns autores (Fajans, 2000; Åström *et al.*, 2005; Jingang *et al.*, 2006) têm encontrado as equações dinâmicas de movimento para o robô bicicleta assumindo todo tipo de considerações e impondo restrições que simplificam o modelo dinâmico do mesmo. As considerações mais freqüentes são mostradas como segue:

- È considerada a bicicleta como um ponto de massa com dos pontos de contato com o chão (ver Figura 3.3).

- Os pneumáticos das rodas são considerados delgados, não são considerados na análise.
- Os momentos de massa das rodas frontal e traseira são desprezados.
- Os momentos de massa da bicicleta são desprezados, ou seja, a bicicleta é considerada como um ponto de massa no centro de massa.
- No controle mediante a movimentação do guidão o ciclista não se movimenta.
- No controle mediante a movimentação do ciclista o guidão não se movimenta.

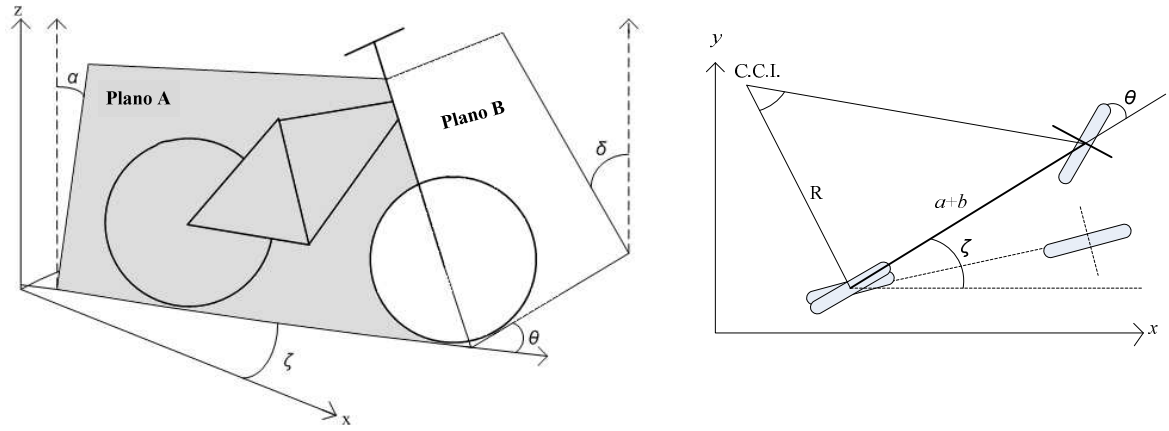


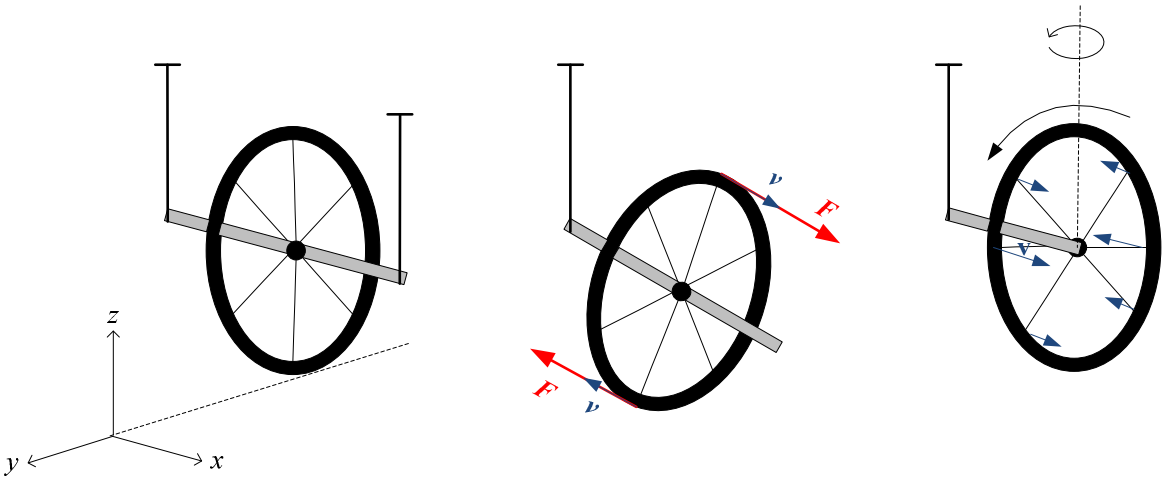
Figura 3.3- Graus de liberdade e parâmetros dinâmicos da bicicleta (Åström *et al.*, 2005, modificada)

Na Figura 3.3 é apresentada a bicicleta e as considerações necessárias para o estudo da dinâmica do sistema. No plano A tem-se a roda traseira e o quadro da bicicleta, no plano B a roda dianteira, o garfo e o guidão. O ângulo α é o ângulo de inclinação do plano A com relação à vertical, e o ângulo δ é o ângulo de inclinação do plano B com relação à vertical quando o plano B este virado um ângulo θ com relação ao plano A. Os planos A e B coincidem quando θ=0.

Quando no guidão é proporcionado um ângulo θ para produzir uma curva, é gerado o denominado Centro de Curvatura Instantâneo (CCI), normalmente utilizado no estudo cinemático do robô. O CCI é encontrado pela interseção entre o eixo perpendicular da roda traseira e o eixo perpendicular á roda dianteira quando girada um ângulo θ (ver Figura 3.3) assumindo que as rodas não sofrem escorregamento. No caso de uma trajetória reta, o CCI está no infinito.

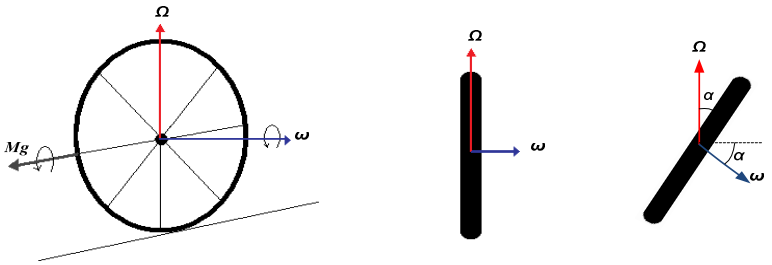
Se uma roda que segue um movimento circular ao redor do próprio eixo com uma velocidade angular $\boldsymbol{\omega}$ é obrigada a girar sobre outro eixo, perpendicular também ao anterior, com uma velocidade angular $\boldsymbol{\Omega}$, como mostrado na Figura 3.4, aparece o efeito denominado giroscópico.

3.4.1 Efeito Giroscópico



a. Roda pendurada de duas cordas | b. Roda parada | c. Roda girando

Figura 3.4- Geração do efeito giroscópico



a. Vista lateral | b. Vista frontal

Figura 3.5- Efeito giroscópico na roda dianteira ao tomar uma curva

O efeito giroscópico pode ser ilustrado como mostrado na Figura 3.4. Considere a roda pendurada por duas cordas (ver Figura 3.4.a), neste caso se a corda de um dos extremos é liberada, quando a gravidade faz cair a roda que não gira, os raios empurram a parte superior da roda (Força **F**, Figura 3.4.b) no sentido horário no plano *x*-*z*. Isto cria uma

componente de velocidade na direção de tombamento como mostrado. O mesmo efeito acontece na parte inferior da roda.

Quando a roda gira, ocorre a mesma coisa na parte superior e inferior da roda, embora quando a componente de velocidade aumenta no plano *x-z*, a velocidade angular do giro da roda traslada o efeito sobre o costado (Figura 3.4.c.), no plano *y-z*. Cada raio agrega um componente de força na sua parte superior o qual gera uma contribuição de força do lado da roda. Desta maneira, o resultado final não é o tombamento da roda, senão um movimento lateral.

O momento induzido $\boldsymbol{M_g}$ (ver Figura 3.5) pelo efeito giroscópico pode ser encontrado mediante a Equação 3.1.

$$\boldsymbol{M}_g = I_o(\boldsymbol{\Omega} \times \boldsymbol{\omega}), \tag{3.1}$$

onde I_o é o momento de inércia da roda dianteira.

Mesmo que para as bicicletas este efeito seja desprezado em relação às considerações indicadas anteriormente e assumindo um deslocamento em linha reta. A continuação é apresentada uma breve explicação do surgimento deste efeito quando a bicicleta inclina-se na realização de uma curva.

Considere-se uma roda girando ao redor de seu próprio eixo com uma velocidade angular ω em quanto a bicicleta toma uma curva de radio *R*, com uma velocidade angular Ω. Imagine a bicicleta dando voltas em uma trajetória circular, com velocidade constante; isto faz com que a bicicleta adote um ângulo de inclinação α como mostrado na Figura 3.3.

A rotação natural da roda ω e o giro da bicicleta ao redor do centro da curva Ω produzem um momento giroscópico ao redor do eixo horizontal, que ajuda a levantar a bicicleta. Tomando os valores escalares, obtêm-se a Equação (3.2), tal que

$$M_g = I_o * \Omega * \omega * \cos(\alpha), \tag{3.2}$$

onde I_o é o momento de inércia da roda em relação ao seu próprio eixo, ω é a velocidade angular de rotação com relação ao mesmo eixo, e Ω é a velocidade angular com que a bicicleta toma a curva.

Tomando agora o momento de inércia das duas rodas o efeito giroscópico fica expressado pela Equação (3.3), tal que

$$\boldsymbol{M}_g = (I_f + I_r)(\boldsymbol{\Omega} \times \boldsymbol{\omega}), \tag{3.3}$$

sendo $I_f + I_r$ os momentos polares de inércia das rodas dianteira e traseira.

Supondo que as rodas têm um peso desprezível então os momentos polares de inércia tenderiam á zero, e o efeito giroscópico será nulo.

Desprezando também a deformação dos pneumáticos, as condições de equilíbrio dinâmico para aquelas condições de movimento circular uniforme (velocidade linear constante e radio de curvatura constante) impõem-se que a resultante do peso e a força centrífuga intersectam a línea que une os pontos de contato de ambas as rodas com o chão.

Neste caso ideal, o ângulo de inclinação da bicicleta para tomar a curva vem dado pela Equação 3.4, tal que

$$tg(\alpha) = \frac{x_g}{y_g} = R\frac{\Omega^2}{g}, \tag{3.4}$$

sendo g a aceleração da gravidade, R o radio da curva, e os pontos (x_g, y_g) as coordenadas do centro de gravidade do sistema. Esta equação mostra a relação entre o ângulo de inclinação do sistema e a velocidade angular e radio de curvatura (Cossalter, 2009).

Existem dois efeitos em conseqüência do efeito giroscopico com relação ao movimento curvilíneo do sistema bicicleta-ciclista. O fato de decrementar no máximo possível o peso dos pneus dos veículos de duas rodas reduz o efeito giroscópico e aumenta a manobrabilidade do veículo. Por sua parte, se o que se requer é se manter o sistema andando em linha reta e manter-se o equilíbrio, o efeito giroscópio ajuda a levar o sistema para a posição vertical (Cossalter , 2009).

3.4.2 O efeito giroscópico na roda dianteira devido ao esforço no guidão

O efeito giroscópico é explicado o como a surgimento de um momento que tenta girar a roda quando se aplica um momento de giro nos outros dos eixos diferentes ao de rotação. Este momento giroscópico será de direção perpendicular aos outros dois momentos.

Os dois momentos indutores são o próprio eixo da roda e o giro de acordo com o eixo do garfo transmitido a través da movimentação do guidão. O resultado é um momento de rotação da bicicleta que ajuda a inclinar a mesma na curva. Neste caso, a direção é perpendicular aos outros dos momentos e seu módulo esta dado pela Equação 3.5, tal que

$$M_1 = -I * \Omega * \omega * \cos(\theta), \tag{3.5}$$

onde θ é o ângulo de giro do guidão. Esta equação indica que quanto maior o ângulo de giro, menor a sensibilidade do veículo a este efeito.

Um efeito inverso pode aparecer, se o veículo é inclinado por uma perturbação externa (por exemplo, o vento), aparecerá no guidão um momento de giro contra o qual deve-se aplicar uma força nas mãos do ciclista para se manter a verticalidade.

A outra componente do momento giroscópico induzido, pouco importante com geometrias de guidão comuns, faria girar a roda em relação ao eixo vertical. O módulo deste momento está dado pela Equação 3.6, tal que

$$M_2 = -I * \Omega * \omega * \sin(\theta). \tag{3.6}$$

Um momento de giro do guidão como conseqüência de outro momento giroscópico surge pela inclinação inesperada do veículo a causa de uma perturbação externa (por exemplo, o vento). A mistura dos momentos de giro de ambas as rodas com o momento de giro do veículo quando se inclina, produz um momento de giro em torno ao eixo vertical do veículo que tenta desalinhar as rodas. Este momento giroscópico induzido tenta provocar um derrape da roda traseira, embora as bicicletas não sejam sistemas rígidos, o que faz possível que o efeito seja absorvido pelo guidão. È importante salientar que a ação sobre o guidão provoca um momento de giro (inclina o veículo), que também origina outro momento no guidão que facilita a manobra (Cossalter, 2009).

3.5 ARQUITETURA DE CONTROLE

O controle do robô ciclista desenvolvido neste trabalho é realizado para o controle do equilíbrio. Não são geradas tarefas de trajetória nem implementação de rotinas de locomoção, por tanto, o trabalho baseia-se no desenvolvimento de controladores para o problema de equilíbrio do robô. Para isto é proposta uma arquitetura híbrida no sentido de combinar sistemas de processamento seqüencial com sistemas de processamento de código em paralelo. Além disto, a arquitetura proposta esta composta por três módulos (ver Figura 3.6), onde cada módulo supõe uma arquitetura de controle independente partindo do fato de que independentemente do processo de locomoção do robô ele sempre tem que manter o equilíbrio.

Em relação ao módulo de equilíbrio é proposta uma arquitetura reativa, os sensores (acelerômetros) definem o comportamento de cada parte funcional com relação ao modelo proposto. Neste contexto, os sinais são entregues para o controlador e este gera o sinal de controle para o atuador. O módulo de planejamento de trajetória deve incorporar um elemento deliberativo, ou seja, uma representação do ambiente (modelo de mundo) para cumprir com as tarefas requeridas, o que supõe a utilização de uma arquitetura deliberativa, utilizado basicamente para a navegação segura do robô.

O módulo de navegação é uma das tarefas complexas na problemática da locomoção de robôs móveis. Tal complexidade está relacionada ao fato de que a navegação deve integrar sensoriamento, atuação, planejamento, arquitetura, hardware, eficiência e computação. Assim, a integração de todos esses pontos é inerente à obtenção de uma boa navegação robótica. Nesse sentido, o planejamento de trajetória em robótica móvel objetiva prover aos robôs a capacidade de planejar seus próprios movimentos, sem a necessidade de interferência humana (Bastos, 2008).

A tecnologia utilizada esta relacionada com a geração de um sistema multiprocessado em plataforma única. Um FPGA só permite a geração de múltiplos processadores que podem intercambiar informações entre si, além disto, a implementação da arquitetura de controle tem vantagens de funcionamento que não são possíveis em comparação com processadores comuns. FPGAs permitem a geração de arquiteturas que contêm módulos executáveis em paralelo e a construção de sistemas que permitem a utilização destes recursos em conjunto.

A Figura 3.6 apresenta a arquitetura híbrida do robô ciclista, este projeto foca-se no módulo de controle, o qual contem um controlador que dependendo dos comportamentos definidos pelo estado dos sensores determina o sinal de controle que vai atuar sobre a planta.

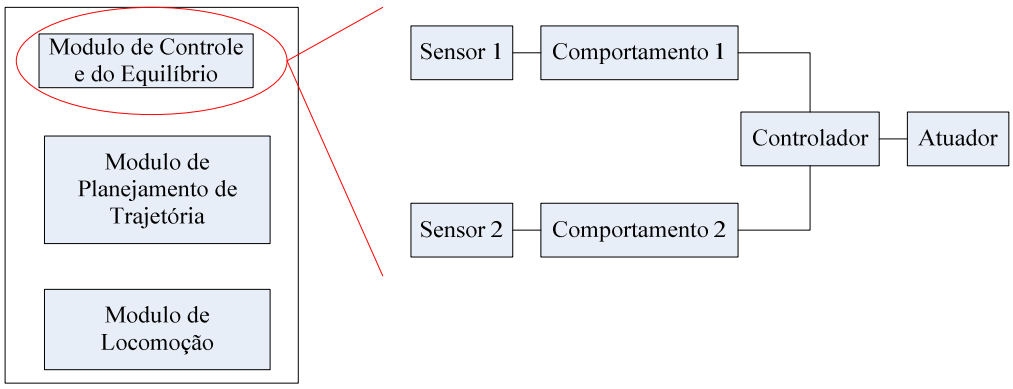


Figura 3.6- Arquitetura de controle proposta para o robô ciclista

A tecnologia utilizada permite a implementação e o estudo dos demais módulos funcionais, assim como a incorporação de novos módulos executáveis em paralelo com os propostos.

3.6 DETALHAMENTO DO PROJETO CONCEITUAL

O esquema apresentado na Figura 3.7 mostra o organograma das funções desenvolvidas.

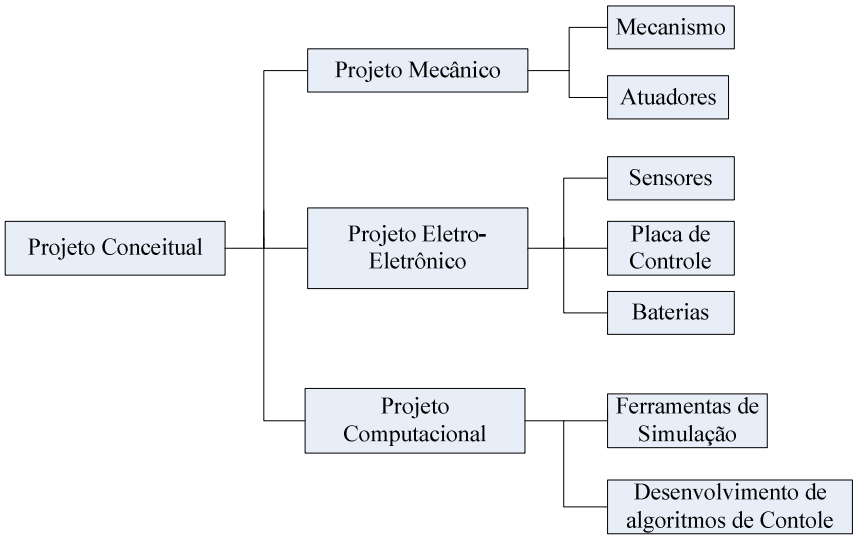


Figura 3.7- Organograma do projeto conceitual

3.6.1 Projeto Mecânico

O projeto do robô ciclista pode ser divido em duas partes, a primeira está relacionada com a construção da bicicleta e a segunda com a construção do ciclista.

Em relação à construção da bicicleta são feitas as seguintes considerações:

- O sistema vai ser um protótipo de provas pelo que não precisa ser de grande tamanho.
- Entre os materiais mais adequados para a construção do marco da bicicleta encontra-se o alumínio, por ser leve e pouco maleable.
- Os parâmetros em relação à construção da bicicleta contribuem de uma forma decisiva á estabilidade da bicicleta.

Os parâmetros que descrevem a geometria da bicicleta são definidos na Figura 3.8. Os parâmetros fundamentais são: distância entre os centros das rodas *b,* o ângulo de cabeceira λ e a fuga *c*. O garfo frontal é angulado de tal forma que o ponto de contacto da roda dianteira com o chão fique atrás do eixo de extensão do eixo do guidão. A fuga é definida como a distância horizontal *c* entre o ponto de contacto e o eixo do guidão, quando a bicicleta está na configuração de referencia vertical com ângulo do guidão zero, e *h* e altura desde o chão até o centro de massa da bicicleta. As propriedades em movimento da bicicleta são fortemente afetadas pela fuga. Uma adequada fuga melhora a estabilidade, mas faz a direção menos ágil.

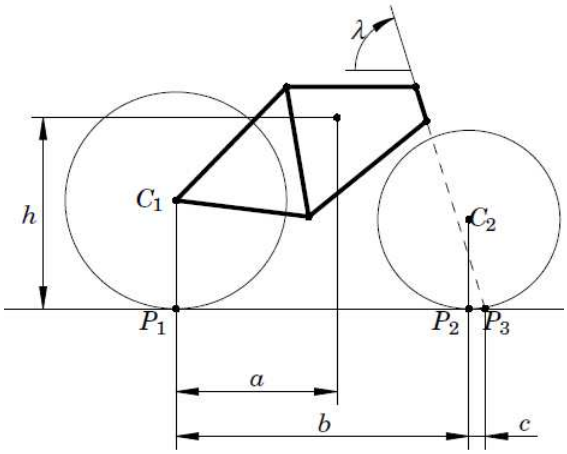


Figura 3.8- Parâmetros de configuração geométrica de uma bicicleta (Åström *et al.,* 2005)

Neste projeto é assumido que a bicicleta consiste de quatro partes rígidas, duas rodas, o quadro, e o garfo frontal com o guidão. Esta simplificação é intuitivamente justificada como um comportamento da bicicleta essencialmente da mesma forma quando a bicicleta e o ciclista estão encostados e ficam parados (sem pedalar) (Åström *et al.,* 2005). O ciclista pode aplicar um torque no guidão.

Uma apropriada coordenação entre o torque aplicado ao guidão e o movimento do tronco do robô ciclista da solução ao problema de restrição de velocidade encontrado no controle mediante a movimentação do guidão.

Em relação à construção do ciclista são tidas as conta as seguintes considerações:

- Para estes casos, onde o ciclista é incluído na análise, a parte superior do corpo do ciclista é modelada como um ponto de massa que pode se movimentar lateralmente com ralação ao quadro da bicicleta. O movimento do tronco do robô ciclista é controlado por um servomotor fixo no quadril como mostrado na Figura 3.9, os braços do robô têm articulações livres no ombro e nos colos. Isto permitirá que, por exemplo, uma movimentação no guidão da bicicleta seja absorvida pelo deslocamento dos braços diminuindo assim a geração de esforços sobre o tronco do ciclista.

- O movimento de pedalo e avanço da bicicleta vai ser controlado por um motor de corrente contínua. O movimento é transmitido ao platô, o movimento do platô é transmitido para as bielas, pedais e roda traseira mediante a cadeia, os pedais giram sobre o seu próprio eixo. As pernas do robô têm articulações livres em tornozelos, joelhos e coxas, portanto, o movimento dos pedais é absorvido pelo deslocamento das pernas.

A escolha dos materiais é um ponto fundamental na construção do robô, deve ser estabelecido um peso limite para o robô, materiais densos (ferro 7.8 g/cm^3) devem ser usados quando se constata a melhor opção, assim como materiais mais leves (alumínio 2.7 g/cm^3) não deve ser usado demasiadamente sem um cálculo prévio, a ponto de ultrapassar o limite e de dimensão e peso (Villa, 2007). Os projetos feitos sem análise de pesos podem resultar em decisões drásticas para aliviar peso, tais como alteração de baterias, motores e estrutura, muitas vezes prejudicando a funcionalidade do robô.

O veículo utilizado no projeto possui 2 rodas de borracha, uma dianteira e uma traseira. Para selecionar as rodas devesse levar em consideração o tipo de veículo e o ambiente onde ira se locomover. Para tanto, é considerada uma roda ideal quando não há deslizamento na direção ortogonal da rolagem (sem escorregamento), não pode ocorrer deslizamento de translação entre a roda e o chão (rolagem pura).

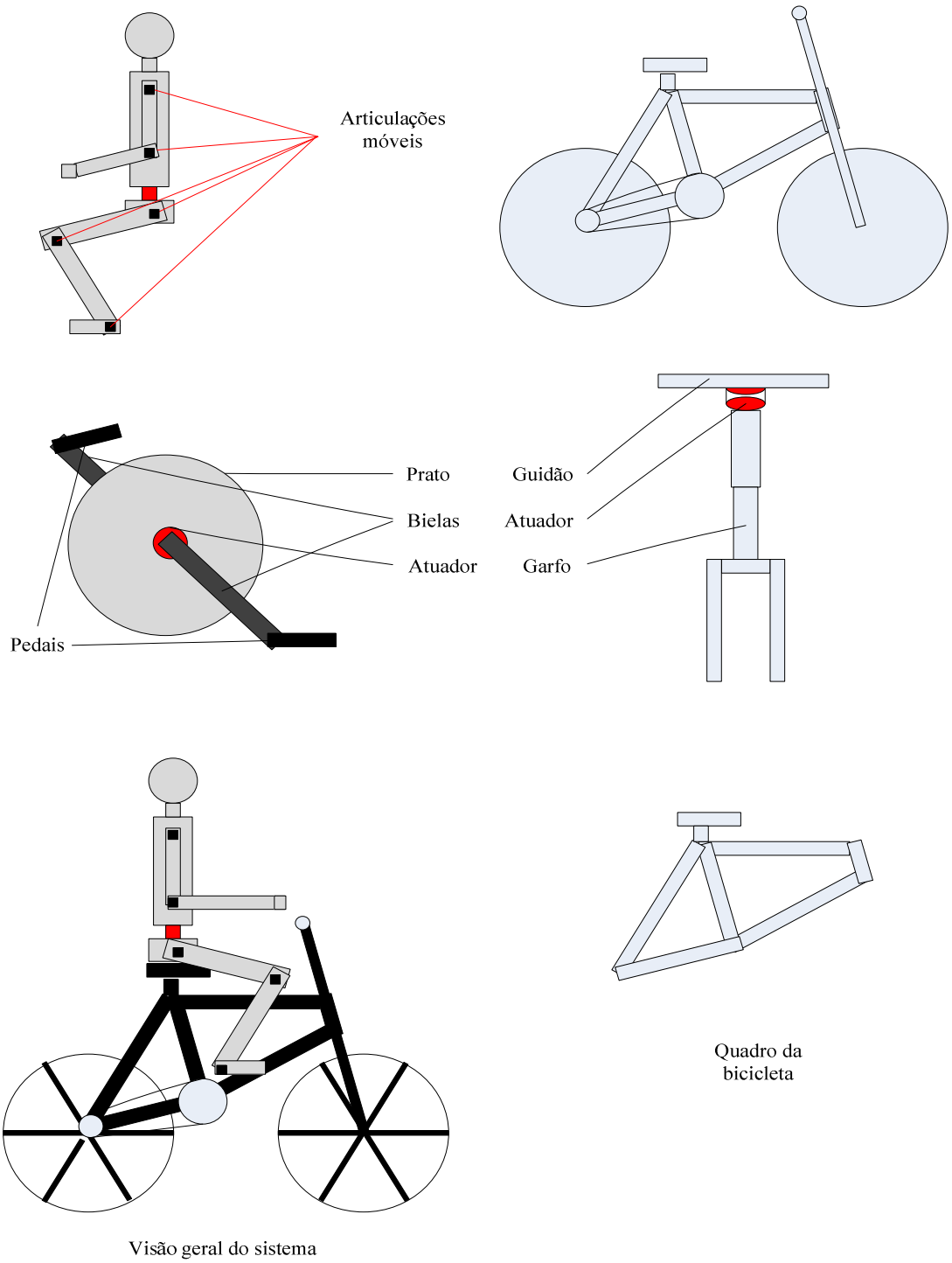


Figura 3.9- Projeto mecânico do robô ciclista

3.6.2 Projeto Eletro-Eletrônico

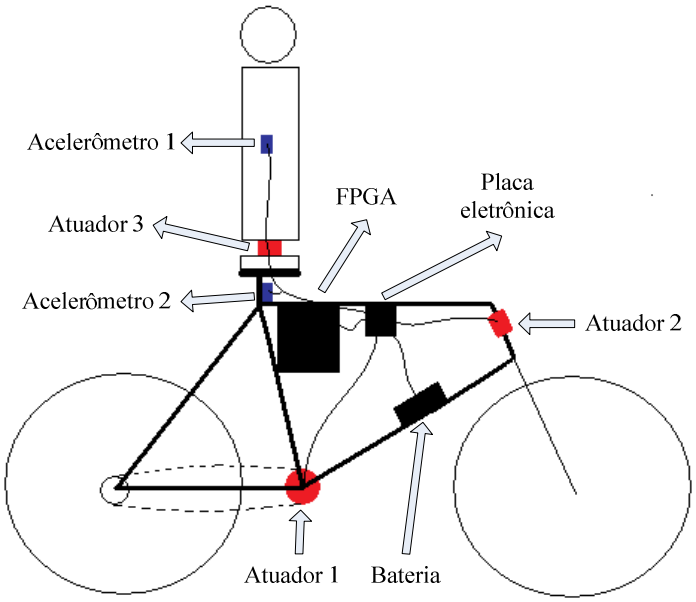


Figura 3.10- Projeto eletrônico do robô ciclista

A Figura 3.10 representa um esquema conceitual do protótipo a desenvolver. Ele contém três motores, o atuador 1 que impulsiona e comanda o movimento da bicicleta para o frente, o atuador 2 que gera a variável de controle quando o controle de equilíbrio é efetuado mediante o guidão, e o atuador 3 que gera a variável de controle quando o controle de equilíbrio é efetuado mediante a movimentação do tronco do ciclista. Dois acelerômetros são necessários para medir o grau de inclinação da bicicleta e do tronco do robô. A CPU é implementada sobre um FPGA, o circuito é alimentado por uma bateria que deve ser estrategicamente posicionada para definir o centro de gravidade do sistema. Finalmente, o circuito é ligado a uma placa eletrônica que reúne e distribui os sinais de todos os componentes eletrônicos do sistema.

O atuador 1 pode ser constituído por um motor DC, e os atuadores 2 e 3 por servomotores, já que são úteis em quanto ao posicionamento preciso do guidão ou do tronco do ciclista, respectivamente.

3.6.2.1 Baterias

Para prover energia elétrica no nível de tensão adequado para cada componente eletrônico do sistema, fornecer a configuração eletrônica de alimentação necessária para o funcionamento de circuitos mais complexos como processadores, transformar sinais eletrônicos de comandos em níveis elétricos que acionam atuadores, e transformar sinais elétricos disponibilizados por sensores em dados digitais que podem ser processados, devem ser tidas em conta as considerações:

- Dispositivos robóticos e mecatrônicos que usem motores para sua mobilidade, normalmente, têm um consumo elevado. Isso quer dizer que baterias comuns tendem a durar pouco.
- O problema maior, entretanto, não está na necessidade de se substituir constantemente essas fontes de energia, mas sim, no seu elevado custo. No entanto, para que uma bateria recarregável seja realmente a melhor solução para a alimentação de um projeto é preciso saber escolhê-la apropriadamente.
- O robô deve ter acoplado a sua estrutura sua fonte de alimentação que fornecerá energia elétrica à sua placa e permitirá a ativação de todos os dispositivos que serão utilizados, como os motores e os sensores, entre outros que podem se acrescentar.

Tendo em conta as considerações anteriores será utilizada uma bateria que durante um intervalo de tempo fornecerá a energia necessária para o robô funcionar. Uma bateria que poderia ser utilizada é a selada UP 1223 de 12V e 2,3Ah, que é fácil de ser encontrada, pesa 1 kg e as dimensões são 178 x 35 x 67 mm.

3.6.2.2 Sensores

Os giroscópios e acelerômetros apresentam em sua saída uma variação de tensão entre 0 e 5V. Os conversores analógicos digitais são utilizados para fazerem a leitura. Em particular tem-se experiência no trabalho dos acelerômetros *MMA1260 – MMA6260Q*. A série *MMA* são acelerômetros capacitivos micro-maquinados, os quais contem dois filtros de Bessel de segunda ordem passa baixas, compensação por temperatura, saída analógica linear, alta sensitividade e um desenho robusto (Freescale Semiconductor, 2006). Esta série é usada em aplicações típicas, tais como medição de vibrações, dispositivos de controle,

sensoriamento de movimento e posição, monitoramento mecânico e proteção de discos duros.

O acelerômetro pode ser modelado como duas placas estacionárias com uma placa móvel entre elas. A placa do centro pode se movimentar de sua posição de equilíbrio aplicando ao sistema uma aceleração. Quando a placa do centro se movimenta, a distância dela a uma placa fixa aumentará pela mesma quantidade em que a distância à outra placa diminui (ver Figura 3.11). A mudança na distância é uma medida da aceleração. Em quanto a placa do centro se move com aceleração, a distância entre as placas muda e o valor de cada capacitor mudará, ($C = A\varepsilon/D$). Onde A é a área da placa, o ε é a constante dielétrica e D é a distância entre as placas.

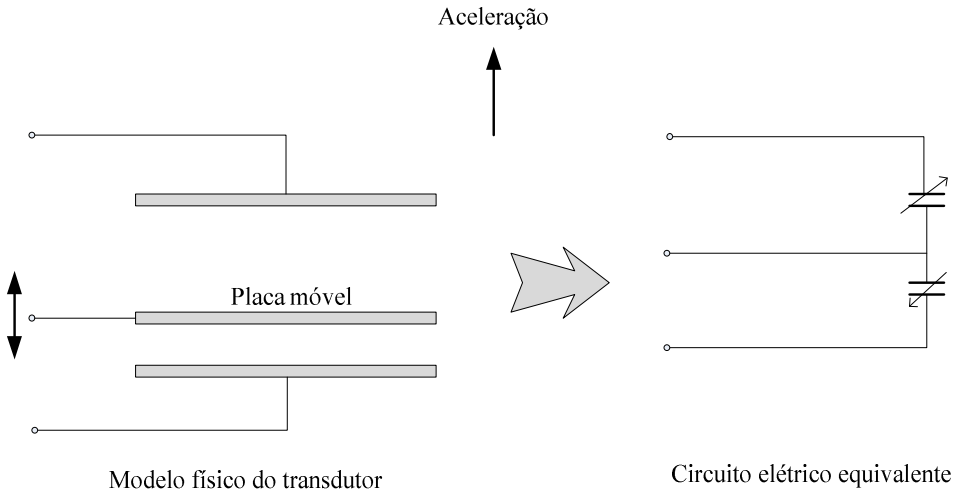


Figura 3.11- Principio de funcionamento dos acelerômetros capacitivos

3.6.2.3 Motores

Os sinais de controles enviados aos motores (servo-motores e o motor DC) devem possuir os níveis de tensão e serem capazes de suprir a corrente elétrica exigida. Além disso, deve haver uma proteção para que perturbações elétricas produzidas não afetem os circuitos de controle. Devem ser utilizados transistores ou optoacopladores, os quais são ligados com outra fonte cada vez que o sinal de controle atua sobre eles. Um exemplo no controle de funcionamento do motor DC é utilizar um transistor do tipo NPN (BC 139). A corrente produzida pela tensão de ativação do circuito de controle na base, libera o fluxo de corrente entre emissor e terra, aterrando a entrada do motor.

Os motores de equipamentos eletrônicos são abundantes no mercado em função da ampla gama de utilização, conseqüentemente, existem em varias dimensões, tensões, pesos e características.

a) Motores DC

Os motores DC automotivos são todos de tensão nominal 12 volts, são robustos e normalmente usados para condições extremas, tais como poeira, calor, variações de tensão e corrente, entre outros. Os motores mais usados em projetos de protótipos didáticos pequenos, com peso de aproximadamente de 5 kg, são os usados em trava elétrica de portas, bomba do limpador de pára-brisa e de gasolina, bomba de combustível, motor do vidro-elétrico, motor de ventoinha, motor de ventilador interno, limpador de pára-brisa dianteiro e traseiro, e bomba hidráulica do freio ABS.

Alguns modelos são incorporados com a caixa de redução (a maioria do tipo rosca sem fim), como o caso do motor de vidro elétrico e limpador de pára-brisa. Em função disto, a rotação final no eixo é baixa (não mais que 150 rpm), tornado o robô mais lento, porém com maior torque.

Em quanto ao peso dos motores, por exemplo, o motor de vidro elétrico (sem a redução elétrica), pesa aproximadamente 300 gramas, com alta rotação e excelente nível de torque. Em quanto aos motores de freio ABS pesam normalmente entre 900 e 1350 gramas, com menos rotação comparado ao motor de vidro elétrico mas com um torque muito superior, ambos inviáveis ao projeto. Outra alternativa interessante é um compressor de 12 V conectado ao acendedor de cigarros, entretanto, o motor é pequeno, pesando pouco mais de 200 gramas, destaca-se como uma excelente opção para tração do robô (Batista *et al.*, 2008).

b) Motores de passo

São indicados para aplicações que necessitam precisão nos movimentos e com baixa rotação. O acionamento do motor de passo necessita de um controlador eletrônico para energizar seqüencialmente cada bobina, fazendo-o andar um passo a cada energização. Muito usados em impressoras, para movimentar o carro dos cartuchos e para avançar o

papel, comumente usam tensões entre 5 e 12 V, e são também usados circuitos integrados para gerenciar a distribuição de passos tal como o ULN2003 (STMicroelectronics, 2002).

c) Servomotores

Da mesma forma que os motores de passo, os servomotores são aplicados em equipamentos que necessitam de controle de movimentos, tais como receptores de antena parabólica e, principalmente, usados em aeromodelos. Entretanto, eles são fabricados para girar em uma determinada faixa de ângulos (geralmente 0 º a 270º) e também necessitam de um controlador para operá-los, sendo a través de largura de pulso (modulação ***P**ulse **W**idth **M**odulation* PWM). A voltagem de operação geralmente está entre 4.8 – 6.0 V, com velocidade de 0.25sg/60 º, peso de 1.3-4.4 oz (tipo Futaba). Para microservos usados em aeromodelismo as voltagens de operação basicamente são os mesmos, mas os pesos variam entre 0.2 - 0.6 oz, com velocidade de 0.10 - 0.15 sg/60º, e as dimensões diminuem para a metade dos convencionais. A Tabela 3.1 mostra as características do microservo BA-TS-9.0 de *BlueArrow*.

Tabela 3.1- Características dos micro-servos *BlueArrow.*

1. Fonte de alimentação	4.8V	6.0V
2. Velocidade	0.12 sec/60º sem carga	
3. Torque	1.2kg.cm (17oz.in)	1.3kg.cm (18oz.in)
4. Ângulo de operação	40º / 400usec	
5. Direção	Sentido horário/ largura de pulso entre 1500 e 1900 us	
6. Corrente de operação	1mA/ corrente estática 100mA/ corrente de operação	
7. Cumprimento do conector	160mm	
8. Dimensões	22.4x22.0x11.2mm	
9. Peso	8.5g (0.30oz.) com JR, FUTABA plug	

3.6.2.4 Unidade de Controle baseada em FPGAs

A unidade de controle é desenvolvida usando FPGAs, a continuação a descrição física da Spartan 3E starter XC3S500E-4FG320C.

A família Spartan 3E de FPGAs foi projetada para reunir as necessidades de aplicações de alto volume conservando uma relação de baixo consumo e alta sensibilidade. As cinco famílias existentes oferecem um intervalo entre 100.000 e 1.6 milhões sistemas de portas

lógicas. A Spartan 3E incrementou a quantidade de I/O lógicas, reduzindo de forma significativa o custo por célula lógica.

A arquitetura da família Spartan 3E (ver Figura 3.12) está conformada por cinco elementos funcionais programáveis fundamentais, a saber:

- *Blocos lógicos Configuráveis (CLBs)*: contem LUTs (*Look Up Table*) de implementação lógica além de elementos de armazenamento usados como *flip-flops* ou *latches*. Os *CLBs* executam uma variedade de funções lógicas, assim como armazenamento de dados.
- *Blocos de entrada e saída (IOBs):* controlam o fluxo de dados entre os pinos I/O e a lógica interna dos dispositivos. Cada IOB suporta fluxo de dados bidirecionais além de operações tri-estados. Adicionalmente, suporta uma variedade de sinais padrão.
- *Blocos RAM:* Proporciona armazenamento de dados na forma de blocos de dobre porta de 18-Kbit.
- *Blocos multiplicadores:* Aceitam dois números binários de 18 bits como entradas e calculam o produto.
- *Blocos de Manejo de Relógio Digital (DCM):* proporcionam autocalibração, soluções digitais completas para distribuição, retardos, multiplicação, divisão e sinais de relógio para deslocamento de fase.

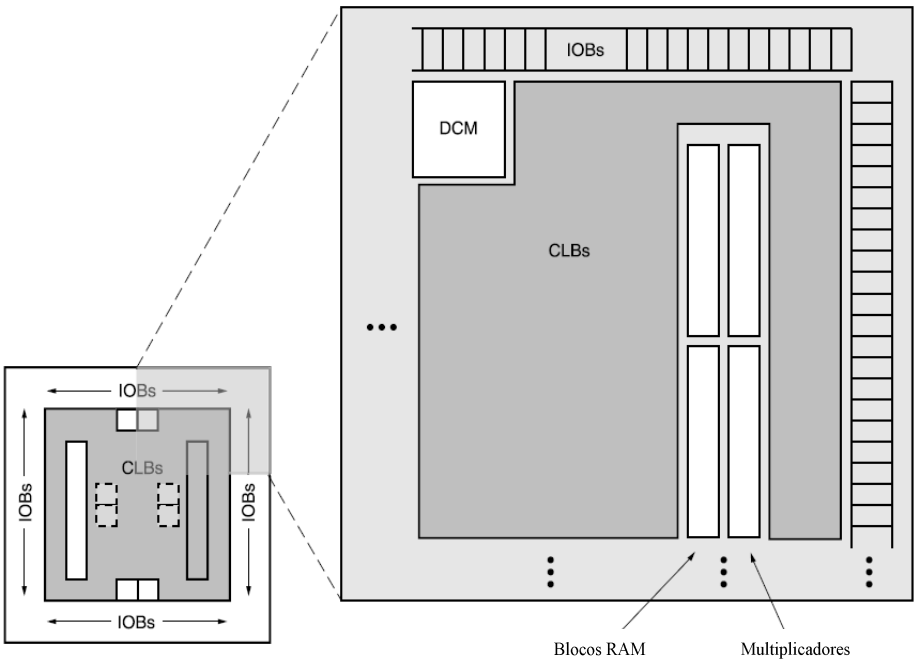


Figura 3.12- Arquitetura blocos funcionais da família Spartan 3E

Existe uma série de *IOBs* ao redor de um arreglo regular de *CLBs*. Cada dispositivo tem duas colunas de blocos *RAM* com exceção da *XC3S100E* que tem uma só. Cada coluna *RAM* está constituída de vários blocos *RAM* de 18 Kbits. Cada bloco *RAM* é associado com um multiplicador dedicado. Dois *DCMs* são posicionados no centro da parte superior e no centro da parte inferior do dispositivo. A *XC3S100E* tem um *DCMs* só na parte superior e outro na parte inferior, em quanto a *XC3S1200E* e *XC3S1600E* mais dois *DCMs* do lado esquerdo e do lado direito (linha ponteada na Figura 3.12).

A Spartan 3E tem uma extensa rede de trilhas que interconectam os cinco elementos funcionais descritos, transmitindo sinais entre eles. Cada elemento funcional tem associada uma matriz de interruptores flexíveis e configuráveis que permitem múltiplas conexões para o roteamento (Xilinx Corporation, 2005).

A Spartan 3E suporta as interfaces de pares entrada-saída seguintes:

- 3.3 V, baixa tensão TTL, LVTTL,
- Baixa tensão CMOS (LVCMOS) em 3.3, 2.5, 1.8, 1.5 ou 1.2 V,
- V PCI em 33 MHz e 66 MHz,
- HSTL I e III em 1.8 V, tipicamente para aplicações de memória,
- SSTL I em 1.8 V e 2.5 V, tipicamente para aplicações de memória.

3.6.3 Projeto Computacional

O desenvolvimento do projeto computacional tem os seguintes objetivos:

- Modelar matematicamente o sistema e simular o comportamento do sistema,
- Projetar controladores de estabilidade do pêndulo duplo invertido interatuado (*Acrobot*),
- Adaptar graficamente os controladores sobre um protótipo virtual do robô ciclista.

O sistema é composto por três *softwares*:

- *Software* de desenvolvimento de plataformas de simulação, projeto de controladores sobre a plataforma desenvolvida e testes de viabilidade. É proposta a utilização de Matlab pela experiência desenvolvida no manejo do *software*.
- *Software* de desenvolvimento da unidade central de processo e projeto de controladores em hardware, desenvolvimento de processador embarcado, e envio e recepção de dados. O software de desenvolvimento da Xilinx para as diferentes famílias de FPGAs e CPLDs que eles fabricam é o *ISE Foundation*, o qual integra todas as necessidades em um ambiente de projeto lógico que proporciona ferramentas para uso de entradas e saídas I/O, análise de consumo e simulação de HDL (Xilinx Corporation, 2009).

  Além disso, o *software Xilinx Plataform Studio (XPS)* proporciona uma amplia variedade de ferramentas, bibliotecas, guias e geradores de *templates* para facilitar a criação de processadores embarcados de uma forma fácil e rápida (Xilinx, 2009).

- Para a simulação virtual do problema existem varias ferramentas de simulação, entre elas têm-se: OpenGL, SDL, ODE, OSG, GALib, SNNS, etc. (Cole, 2005; Leblanc e Ober, 2000). Por simplicidade e tendo em conta que os controladores são desenvolvidos em Matlab, escolheu-se a ferramenta *Virtual Realm Builder 2.0. V-Realm Builder* é um editor gráfico de arquivos VRML, sendo um pacote computacional para a criação de objetos tridimensionais e mundos virtuais para ser vistos na própria plataforma Virtual do *V-Realm* ou qualquer outro compilador de VRML. *V-Realm Builder* foi projetado para fornecer ferramentas para minimizar o tamanho dos arquivos, e para fornecer médios de modelar objetos complexos com primitivas, sem ter que sobrecarregar a rede com os arquivos gerados pelos outros pacotes mais sofisticados (V-Realm™ Builder, 2009).

A ferramenta *Virtual Reality Toolbox* permite conectar um mundo virtual com um modelo realizado no Simulink ou Matlab (Ozana, 2008). Esta ferramenta esta baseada na linguagem VRML (*Virtual Reality Modeling Language*). VRML é um formato de arquivo que descreve objetos e mundos virtuais interatuando em três dimensões. Cada arquivo VRML estabelece implicitamente um sistema de coordenadas para todos os objetos definidos e incluídos no arquivo. O mesmo pode realizar hipervínculos com outros arquivos e aplicações, além de permitir definir comportamentos para os objetos e maneja uma estrutura hierárquica para os objetos.

3.7 CONCLUSÃO DO CAPÍTULO

O projeto conceitual é a primeira etapa do processo de projeto do robô, foram fornecidos dados técnicos sobre o projeto mecânico, eletrônico e computacional do robô ciclista, materiais, componentes eletrônicos, e software para simulação e desenvolvimento hardware, que podem ser úteis na implementação de um protótipo real. O fato de o robô ciclista ser um sistema complexo, pelos seus múltiplos graus de liberdade, e o conjunto de parâmetros e complicações descritas neste capítulo, fica claro a dificuldade de se obter um modelo matemático que represente totalmente o sistema (ainda hoje é objeto de pesquisa). De acordo com isto é sugerido a implementação de controladores inteligentes para o controle e estabilidade do robô ciclista já que eles não dependem do modelo matemático e sim da experiência e do ajuste empírico dos parâmetros.

4 MODELAGEM DINÂMICA

É possível encontrar na literatura trabalhos na área como mostrado no primeiro capítulo, mas não tem sido possível encontrar um modelo padrão ou um conjunto de equações dinâmicas que sejam aceitas como certas pela comunidade da área, para controlar o guidão e o contrapeso do robô ciclista. As dificuldades são devidas, principalmente, (a) à complexidade do sistema, (b) a quantidade de parâmetros envolvidos e (c) à influência destes no comportamento do sistema. Existem alguns temas em aberto tais como (a) mudanças radicais na geometria e forma da bicicleta e seu impacto na estabilidade da bicicleta, ou (b) pelo contrário, como restrições de direção e contrapeso podem influir na geometria da bicicleta. Adicionalmente, embora as bicicletas modernas dirijam e balancem notavelmente bem, como elas fazem isto envolve ainda mistério (Papadopoulus, 1990).

Entre os sistemas mais estudados, por serem sistemas instáveis, encontram-se os pêndulos invertidos. Os pêndulos começaram ser importantes na área de robótica na medida em que apresentam semelhanças com os movimentos próprios das articulações, comportamentos humanos ou como ferramentas funcionais em suas estruturas, o que virou desafio mesmo na engenharia de controle. O controle do pêndulo invertido foi amplamente utilizado nos laboratórios de controle para demonstrar a eficácia do sistema de controle em uma analogia ao controle de lançamento de foguetes. O pêndulo invertido duplo é uma extensão do sistema original de pêndulo invertido, o qual é apropriado para pesquisar diferentes métodos de controle (Aracil e Gordillo, 2005).

O modelo dinâmico do comportamento de um pêndulo invertido interatuado, assemelhado com o comportamento de um robô em bicicleta simplificado, com certas restrições de velocidade, pode ser obtido. Por outro lado, o uso de ferramentas computacionais pode ser usado para simular numericamente o modelo determinando exigências de *hardware* críticas na implementação do robô ciclista.

Existem quatro maneiras de representação matemática mais comumente usada em sistemas dinâmicos (Hung *et al.*, 1997):

- Forma de configuração,
- Representação em espaços de estados,
- Equações de entrada e saída,

- Função de transferência.

4.1 FORMA DE CONFIGURAÇÃO

São chamadas coordenadas generalizadas ao conjunto de coordenadas independentes que descrevem completamente o sistema. Para qualquer sistema, o conjunto das coordenadas que pode ser utilizado para descrever o mesmo são as coordenadas generalizadas (ver Equação 4.1). Entretanto, este sistema de coordenadas não é único (problema de unicidade da descrição), tal que

$$q_i\left(i = 1,2,3,...,n\right) , \tag{4.1}$$

onde i são os graus de liberdade. O número de graus de liberdade é igual ao número mínimo de coordenadas independentes requerido para descrever a posição de todos os elementos de um sistema. Geometricamente falando, as coordenadas generalizadas, q_i definem um espaço cartesiano unidimensional como mostrado na Equação 4.2, tal que

$$\begin{bmatrix} \ddot{q_1} & = & f_1(q_1, q_2, \dots, q_n, \dot{q_1}, \dot{q_2}, \dots, \dot{q_n}, t) \\ \ddot{q_2} & = & f_2(q_1, q_2, \dots, q_n, \dot{q_1}, \dot{q_2}, \dots, \dot{q_n}, t) \\ . & & \\ . & & \\ . & & \\ \ddot{q_n} & = & f_n(q_1, q_2, \dots, q_n, \dot{q_1}, \dot{q_2}, \dots, \dot{q_n}, t) \end{bmatrix}, \tag{4.2}$$

onde q_i e $\dot{q}_\iota$ são as coordenadas e velocidades generalizadas. As funções f_1, f_2,...,f_n geralmente não-lineares, e representam forças, sendo funções algébricas de q_i , $\dot{q}_\iota$ e tempo t.

4.2 REPRESENTAÇÃO EM ESPAÇO DE ESTADOS

As técnicas no domínio do tempo podem ser usadas para sistemas não-lineares, variantes no tempo e multivariáveis. Um sistema de controle variante no tempo é um sistema para o qual um ou mais parâmetros do sistema podem variar em função do tempo. O conceito de espaços de estado está relacionado ao que se entende por variáveis de estado.

O estado de um sistema é um conjunto de variáveis tais que o conhecimento dos valores destas variáveis e das funções de entrada, com as equações que descrevem a dinâmica, fornece os estados futuros e a(s) saída(s) futura(s) do sistema (ver Figura 4.1). As variáveis de estado descrevem a resposta futura de um sistema, dado o estado presente, as excitações de entrada e as equações que descrevem a dinâmica (Hung *et al.*, 1997).

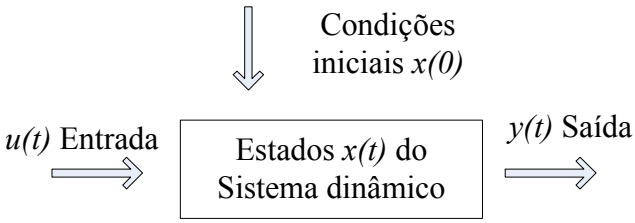


Figura 4.1- Sistema dinâmico e variáveis de estado.

O estado de um sistema é descrito por meio de um sistema de equações diferenciais de primeira ordem escrita em termos das variáveis de estado (x_1,x_2,...,x_n). Estas equações diferenciais de primeira ordem podem ser escritas na forma geral como mostrado na Equação 4.3, onde

$$\begin{bmatrix} \dot{x_1} &= a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n + b_{11}u_1 + \ldots + b_{1m}u_m \\ \dot{x_2} &= a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n + b_{21}u_1 + \ldots + b_{2m}u_m \\ . & \quad . \\ . & \quad . \\ . & \quad . \\ \dot{x_n} &= a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n + b_{n1}u_1 + \ldots + b_{nm}u_m \end{bmatrix}. \quad (4.3)$$

Assim, este sistema de equações diferenciais simultâneas pode ser escrito em forma matricial como na Equação 4.4, tal que

$$\begin{bmatrix} \dot{x_1} \\ \dot{x_2} \\ . \\ . \\ . \\ \dot{x_n} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ & . & & \\ & . & & \\ & . & & \\ a_{n1} & a_{n2} & \ldots & a_{nm} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ . \\ . \\ . \\ x_n \end{bmatrix} + \begin{bmatrix} b_{11} & \ldots & b_{1m} \\ b_{21} & \ldots & b_{2m} \\ . & & \\ . & & \\ . & & \\ b_{n1} & \ldots & b_{nm} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ . \\ . \\ . \\ u_m \end{bmatrix}. \quad (4.4)$$

A matriz coluna que contêm as variáveis de estado é chamada de vetor de estado e é escrita como mostrado na Equação 4.5.

$$x = \begin{bmatrix} x_1 \\ x_2 \\ . \\ . \\ . \\ x_n \end{bmatrix}. \tag{4.5}$$

O vetor dos sinais de entrada é definido como $\boldsymbol{u}$. O sistema pode ser representado através de notação compacta de equação diferencial de estado como mostrado na Equação 4.6.

$$\dot{\boldsymbol{x}} = A\boldsymbol{x} + B\boldsymbol{u}. \tag{4.6}$$

Esta equação diferencial e comumente chamada de equação de estado. Em geral, as saídas de um sistema linear podem ser relacionadas com as variáveis de estado e com os sinais de entrada pela Equação 4.7 de saída, onde

$$\boldsymbol{y} = C\boldsymbol{x} + D\boldsymbol{u}\,. \tag{4.7}$$

4.3 EQUAÇÕES DE ENTRADA E SAÍDA

Outra forma na qual um modelo de sistema pode ser representado é uma equação de entrada-saída, ela é uma equação diferencial simples em termos de um sistema de entradas, sistema de saídas, e suas derivadas (com relação a *t*), onde

$$y^n = a_1 y^{n-1} + \ldots + a_{n-1} y' + a_n y = b_0 u^m + b_1 u^{m-1} + \ldots + b_{m-1} u' + b_m u\,, \tag{4.8}$$

com $m \leq n$, onde $a_i (i = 1, 2, \ldots, n)$ e $b_k (k = 1, 2, \ldots, m)$ são coeficientes constantes (para um sistema linear), *y é* o sistema de saída e *u* é a entrada.

As derivadas de muitas das coordenadas generalizadas podem aparecer simultaneamente em algumas das equações diferenciais, fazendo isto bastante complexo para se obter uma equação diferencial simples e excluir as variáveis não desejadas.

A idéia é obter a transformada de Laplace de cada equação diferencial no modelo do sistema, assumindo condições iniciais zero. As variáveis não desejadas podem ser excluídas através de métodos algébricos tais como substituição ou regras de Cramer (Grossman, 2007) para produzir uma equação simples em termos da transformada de Laplace da coordenada desejada e da entrada. Finalmente, esta equação é transformada para o domínio do tempo e interpretada como uma equação diferencial.

4.4 FUNÇÃO DE TRANSFERÊNCIA

Considera-se, novamente, o sistema invariante no tempo (coeficientes constantes) descrito pela Equação 4.8, no qual *u* e *y* são o sistema entrada-saída, respectivamente. Aplicando a transformada de Laplace obtêm-se a Equação 4.9,

$$(s^n + a_1 s^{n-1} + ... + a_{n-1}s + a_n)Y(s) = (b_0 s^m + b_1 s^{m-1} + ... + b_m)U(s) \quad . \tag{4.9}$$

Assumindo condições iniciais, a função de transferência está definida como a razão entre a transformada de saída e a transformada de Laplace de entrada, conforme descrito na Equação 4.10, tal que

$$G(s) = \frac{Y(s)}{U(s)} = \frac{b_0 s^m + b_1 s^{m-1} + ... + b_m}{s^n + a_1 s^{n-1} + ... + a_{n-1}s + a_n} \quad . \tag{4.10}$$

Lembre-se que cada par de sistemas entrada-saída, corresponde a uma equação entrada-saída.

Finalmente, a Tabela 4.1 mostra uma comparação entre a representação de sistemas mediante função de transferência e espaço de estados. Isto justifica a decisão de representar o sistema de estudo neste trabalho mediante a representação em espaço de estados.

Tabela 4.1- Quadro comparativo entre a representação de sistemas (Bishop, 1997)

Itens Avaliados	**Representação de Sistemas**	
	Função de Transferência	**Espaço de Estados**
Condições Iniciais não nulas	Menos Adequado	Mais Adequado
Resposta em Freqüência	Mais Adequado	Menos Adequado
Sistema Multi-variável	Menos Adequado	Mais Adequado
Sistemas Variantes no tempo	Menos Adequado	Mais Adequado
Sistemas não-lineares	Menos Adequado	Mais Adequado

4.5 MÉTODO DE RUNGE KUTTA DE 4ª ORDEM

Os métodos numéricos são procedimentos que transformam cálculos analíticos complicados em um conjunto de cálculos simples. Os mesmos permitem encontrar uma

solução aproximada de um problema de valor inicial. Existem vários métodos, a saber: método de Euler, Euler Aprimorado, Runge–Kutta, Passos Múltiplos, entre outros (Boyce, 2001). Estes métodos se concentram, principalmente, em problemas de valor inicial para equações de primeira ordem baseados na equação diferencial $\dot{y} = f(t, y)$ com condição inicial $y_{t0} = y_0$.

O método de Runge-Kutta tem um erro de truncamento local proporcional a h^5 (h é o tamanho de passo uniforme). Assim, é duas ordens de grandeza mais preciso que o método de Euler aprimorado, e três ordens de grandeza mais preciso do que o método de Euler. Ele é relativamente simples de usar e é suficientemente preciso para tratar problemas de maneira eficiente (Boyce, 2001).

A fórmula de Runge Kutta envolve uma média ponderada de valores de *f(t,y)* em pontos diferentes no intervalo $t_n \leq t \leq t_{n+1}$, sendo dada pela Equação

$$y_{n+1} = y_n + \frac{h}{6}(k_{n1} + 2k_{n2} + 2k_{n3} + k_{n4}), \tag{4.11}$$

onde

$k_{n1} = f(t, y_n)$,

$k_{n2} = f\left(t_n + \frac{1}{2}h, y_n + \frac{1}{2}hk_{n1}\right)$,

$k_{n3} = f\left(t_n + \frac{1}{2}h, y_n + \frac{1}{2}hk_{n2}\right)$,

$k_{n4} = f\left(t_n + \frac{1}{2}h, y_n + \frac{1}{2}hk_{n3}\right)$.

O algoritmo para implementação computacional pode ser descrito em 8 passos, conforme mostrado na Figura 4.2.

A implementação do algoritmo foi desenvolvida em ambiente Matlab. O código é apresentado no Anexo A.

4.5.1 Erros em aproximações numéricas

Deve-se usar um tamanho de passos que seja suficientemente pequeno para garantir a precisão necessária, mas que não seja pequeno demais, um tamanho de passos desnecessariamente pequeno torna os cálculos mais lentos, mais caros computacionalmente e, em alguns casos, pode até causar perda de precisão.

Passo 1: defina $f(t,y)$.
Passo 2: defina os valores iniciais $t0$ e $y0$
Passo 3: defina o tamanho do passo h e o número de passos n.
Passo 4: escreva t_0 e y_0.
Passo 5: para j de 1 até n execute o passo 6.
Passo 6:

$k_1 = f(t,y)$
$k_2 = f(t+0.5*h, y+0.5*h*k_1);$
$k_3 = f(t+0.5*h, y+0.5*h*k_2);$
$k_4 = f(t+h, y + h*k_3)$
$y = h/6\ (k_1 + 2*k_2 + 2*k_3 + k_4)$
$t = t+h$

Passo 7: Escreva t e y.
Passo 8: fim do algoritmo.

Figura 4.2-. Algoritmo de Runge Kutta 4ª ordem (Boyce, 2001).

Existem duas fontes fundamentais de erro ao se resolver um problema de valor inicial numericamente. A diferença E_n entre a solução $y = \phi(t_n)$ do problema de valor inicial e sua aproximação numérica é dada pela Equação 4.12, conhecida como erro de truncamento global, tal que

$$E_n = \phi(t_n) - y_n. \tag{4.12}$$

Este erro tem duas causas, a primeira relacionada com a aproximação para determinar y_{n+1} em cada passo, e a segunda relacionada com os dados de entrada já que estão apenas aproximadamente corretos em cada etapa. Isto é devido a que em geral $\phi(t_n)$ não é igual a y_n. Assumindo que $y_n = \phi(t_n)$. Desta maneira, o único erro efetuado em cada passo é devido ao uso de uma fórmula aproximada, esse erro é conhecido como erro de truncamento local e_n.

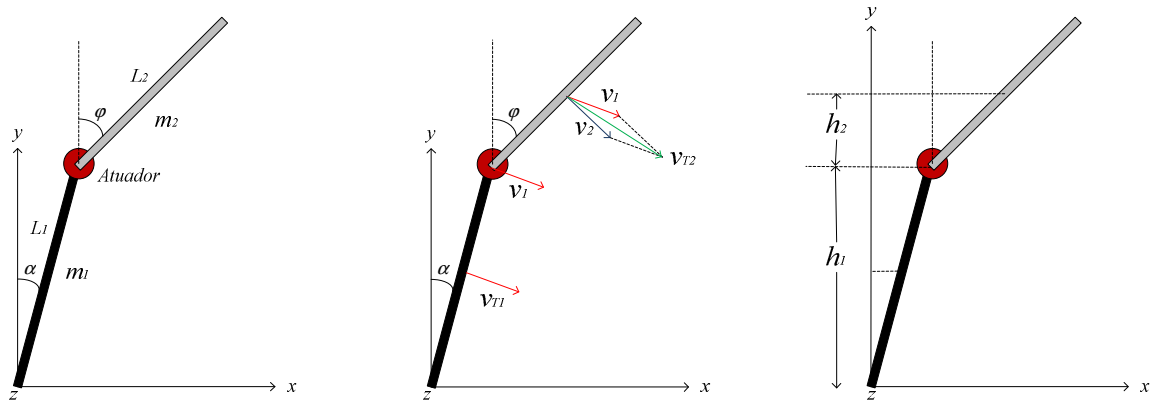
A segunda fonte fundamental de erro é que os cálculos são efetuados em aritmética com apenas um número finito de dígitos. Isto leva a um erro de arredondamento R_n definido pela Equação 4.13, tal que

$$R_n = y_n - Y_n, \tag{4.13}$$

onde Y_n é o valor computado de fato pelo método numérico usado (Boyce, 2001).

4.6 MODELO MATEMÁTICO DO PÊNDULO DUPLO INTERATUADO

Considere o pêndulo duplo interatuado no médio de duas barras, onde a primeira barra pode girar livremente sobre o plano *xy*. O atuador está fixo na extremidade da primeira barra e a segunda barra é controlada pelo torque gerado pelo atuador como mostrado na Figura 4.3.a.



a. Pêndulo duplo interatuado

b. Componentes de velocidade para cálculo da energia cinética

c. Alturas para cálculo da energia potencial

Figura 4.3- Parâmetros para o cálculo das equações de movimento do pêndulo duplo invertido interatuado.

O modelo matemático do sistema permite mediante a utilização de métodos numéricos (método de Runge-Kutta de 4ª ordem, método de Euler) a solução das equações dinâmicas do sistema para o desenvolvimento do simulador da planta no computador. Além disso, as equações encontradas tornadas lineares ao redor do ponto de equilíbrio (ponto de operação) e permitem encontrar as matrizes necessárias para o desenvolvimento dos controladores de interesse. Para isto, foi utilizado o método de energias de Langrage (Hung *et al.*, 1997) , segundo a Equação 4.14,

$$\frac{d}{dt}\left(\frac{\partial T}{\partial \dot{q}_i}\right) - \frac{\partial T}{\partial q_i} + \frac{\partial V}{\partial q_i} = Q_{nc}, \tag{4.14}$$

onde T é a energia cinética do sistema, V é a energia potencial do sistema, q_i são as coordenadas generalizadas do sistema e Q_{nc} é o trabalho realizado pelas forças de tipo não-conservativas.

Neste caso, são definidas duas coordenadas generalizadas α e φ (ver Figura 4.3.a). A energia cinética e potencial do sistema está definida pelas Equações 4.15 e 4.16, respectivamente,

$$T = \sum_i \frac{1}{2} m_i v_i^2 + \sum_i \frac{1}{2} I_c \omega_i^2 \tag{4.15}$$

e

$$V = V_e + V_g \tag{4.16}$$

onde m_i é a massa, v_i é a velocidade tangencial, I_c é o momento de inércia das barras, ω_i é a velocidade angular de acordo com a coordenada generalizada i, V_e é a energia potencial elástica do sistema e V_g é a energia potencial gravitacional do sistema.

A energia cinética esta composta pela energia cinética translacional e a energia cinética rotacional, os momentos de inércia são medidos com relação ao centro de massa como mostrado nas Equações 4.17 e 4.18, onde

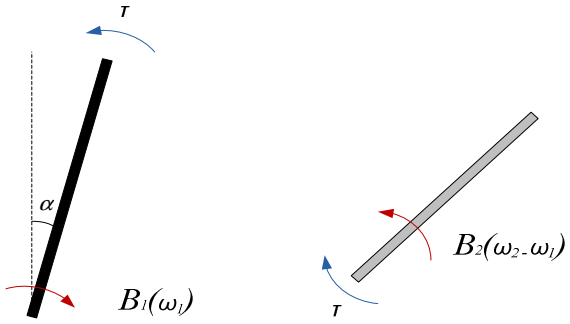
$$T = \frac{1}{2} m_1 v_{T_1}^2 + \frac{1}{2} m_2 v_{T_2}^2 + \frac{1}{2} I_{c1} \omega_1^2 + \frac{1}{2} I_{c2} \omega_2^2 \,, \tag{4.17}$$

$$T = \frac{1}{2}\left(\frac{m_1}{3} + m_2\right) \dot{\alpha}^2 L_1^2 + \frac{1}{6} m_2 L_2^2 \dot{\varphi}^2 + \frac{m_2}{2} L_1 L_2 \dot{\alpha}\dot{\varphi} \cos(\varphi - \alpha). \tag{4.18}$$

A energia potencial do sistema é toda gravitacional devido à ausência de forças elásticas segundo a Equação 4.19, onde

$$V = -\left(\frac{m_1}{2} + m_2\right) g L_1 \cos\alpha - m_2 g \frac{L_2}{2} \cos\varphi \,. \tag{4.19}$$

O trabalho devido as forças de tipo não conservativas é gerada só pelo torque τ do atuador aplicado no final da *barra 1*, sobre a *barra 2* (ver Figura 4.3.a), e pelos amortecimentos viscosos nas juntas B_1 e B_2, como mostrado na Figura 4.4.



a. Barra 1 b. Barra 2

Figura 4.4- Forças de tipo não-conservativas consideradas, atuantes sob o sistema.

Para se obter a primeira equação dinâmica de movimento relacionada com a coordenada generalizada α, são encontradas as Equações 4.20, 4.21, 4.22 e 4.23, dados por

$$\frac{\partial T}{\partial \dot{\alpha}} = (m_1 l_{c1}^2 + m_2 L_1^2 + I_{c1})\dot{\alpha} - m_2 L_1 l_{c2} cos(\varphi - \alpha)\,\dot{\varphi} \tag{4.20}$$

$$\frac{d}{dt}\left(\frac{\partial T}{\partial \dot{\alpha}}\right) = (m_1 l_{c1}^2 + m_2 L_1^2 + I_{c1})\ddot{\alpha} - m_2 L_1 l_{c2} \cos(\varphi - \alpha)\,\ddot{\varphi}$$

$$-m_2 L_1 l_{c2}\dot{\varphi}\dot{\alpha}\,\text{sen}(\varphi - \alpha) + m_2 L_1 l_{c2}\,\text{sen}(\varphi - \alpha)\,\dot{\varphi}^2 \tag{4.21}$$

$$\frac{\partial T}{\partial \alpha} = -m_2 L_1 l_{c2}\text{sen}\,(\varphi - \alpha)\dot{\alpha}\dot{\varphi} \tag{4.22}$$

$$\frac{\partial V}{\partial \alpha} = -(m_1 l_{c1} + m_2 L_1) g\text{sen}(\alpha). \tag{4.23}$$

Substituindo as Equações 4.20, 4.21, 4.22 e 4.23 em 4.14, obtém-se a Equação 4.24.

$$(m_1 l_{c1}^2 + m_2 L_1^2 + I_{c1})\ddot{\alpha} - m_2 L_1 l_{c2}\cos(\varphi - \alpha)\ddot{\varphi} + B_1\dot{\alpha}$$

$$+m_2 L_1 l_{c2}\,\text{sen}(\varphi - \alpha)\,\dot{\varphi}^2 - (m_1 l_{c1} + m_2 L_1) g\text{sen}(\alpha) = -\tau \tag{4.24}$$

Para se obter a segunda equação dinâmica de movimento com relação à segunda coordenada generalizada φ, são obtidas as Equações 4.25, 4.26, 4.27 e 4.28, onde

$$\frac{\partial T}{\partial \dot{\varphi}} = (m_2 l_{c2}^2 + I_{c2})\dot{\varphi} - m_2 L_1 l_{c2}\dot{\alpha}\cos\,(\varphi - \alpha) \tag{4.25}$$

$$\frac{d}{dt}\left(\frac{\partial T}{\partial \dot{\varphi}}\right) = (m_2 l_{c2}^2 + I_{c2})\ddot{\varphi} - m_2 L_1 l_{c2}\cos(\varphi - \alpha)\,\ddot{\alpha} +$$

$$m_2 L_1 l_{c2}\dot{\alpha}\dot{\varphi}\,\text{sen}(\varphi - \alpha) - m_2 L_1 l_{c2}\text{sen}\,(\varphi - \alpha)\,\dot{\alpha}^2 \tag{4.26}$$

$$\frac{\partial T}{\partial \varphi} = m_2 L_1 l_{c2}\dot{\varphi}\dot{\alpha}\text{sen}\,(\varphi - \alpha) \tag{4.27}$$

$$\frac{\partial V}{\partial \varphi} = -m_2 g l_{c2}\text{sen}(\varphi). \tag{4.28}$$

Substituindo as Equações 4.25, 4.26, 4.27 e 4.28 em 4.14, obtém-se a Equação dada por

$$-m_2 L_1 l_{c2}\cos(\varphi - \alpha)\,\ddot{\alpha} + (m_2 l_{c2}^2 + I_{c2})\ddot{\varphi} + B_2(\dot{\varphi} - \dot{\alpha})$$

$$-m_2 L_1 l_{c2}\dot{\alpha}^2\,\text{sen}(\varphi - \alpha) - m_2 g l_{c2}\text{sen}\,(\varphi) = \tau. \tag{4.29}$$

O sistema de equações é organizado em forma de matriz de segunda ordem segundo a Equação 4.30, onde

$$\begin{bmatrix} m_1 l_{c1}^2 + m_2 L_1^2 + I_{c1} & -m_2 L_1 l_{c2} cos(\varphi - \alpha) \\ -m_2 L_1 l_{c2} \cos(\varphi - \alpha) & (m_2 l_{c2}^2 + I_{c2}) \end{bmatrix} \begin{bmatrix} \ddot{\alpha} \\ \\ \ddot{\varphi} \end{bmatrix} +$$

$$\begin{bmatrix} B_1 & m_2 L_1 l_{c2} \operatorname{sen}(\varphi - \alpha)\, \dot{\varphi} \\ -m_2 L_1 l_{c2} \operatorname{sen}(\varphi - \alpha)\, \dot{\alpha} - B_2 & B_2 \end{bmatrix} \begin{bmatrix} \dot{\alpha} \\ \\ \dot{\varphi} \end{bmatrix} + \tag{4.30}$$

$$\begin{bmatrix} -(m_1 l_{c1} + m_2 L_1) g \operatorname{sen}(\alpha) \\ -m_2 g l_{c2} \operatorname{sen}(\varphi) \end{bmatrix} = \begin{bmatrix} -\tau \\ \\ \tau \end{bmatrix}.$$

A Equação 4.30 apresenta um sistema de segunda ordem não-linear. Para modelar a planta usando o método de Runge Kutta de 4ª ordem é necessário transforma o sistema da Equação 4.30 em um sistema de equações diferenciais de primeira ordem. Além disto, o projeto de controladores baseados em métodos convencionais requer de um sistema de equações em variáveis de estado.

4.7 ENCONTRANDO AS EQUAÇÕES DE ESTADO

Passando para equações de estado, sejam definidas as variáveis de estado na Equação 4.31.

$$\begin{array}{l} x_1 = \alpha \\ x_2 = \dot{\alpha} \\ x_3 = \varphi \\ x_4 = \dot{\varphi} \end{array}, \quad \text{então} \quad \begin{array}{lll} \dot{x_1} = \dot{\alpha} = x_2 \\ \dot{x_2} = \ddot{\alpha} \\ \dot{x_3} = \dot{\varphi} = x_4 \\ \dot{x_4} = \ddot{\varphi} \end{array} \tag{4.31}$$

$$u = \tau$$

Agora devem se procurar as relações correspondentes para $\dot{x_2} = \ddot{\alpha}$ e $\dot{x_4} = \ddot{\varphi}$.

Partindo das Equações 4.24 e 4.29, e substituindo os seguintes termos

$d_1 = m_1 l_{c1}^2 + m_2 L_1^2 + I_{c1}$; $d_2 = m_2 L_1 l_{c2} \cos(x_3 - x_1)$; $d_3 = m_2 L_1 l_{c2} \operatorname{sen}(x_3 - x_1)$;

$d_4 = (m_1 l_{c1} + m_2 L_1) g \operatorname{sen}(x_1)$; $d_6 = m_2 l_{c2}^2 + I_{c2}$; $d_8 = m_2 g l_{c2} \operatorname{sen}(x_3)$

Para tanto, obtém-se o sistema de Equações formado pelas Equações 4.32 e 4.33.

$$d_1\dot{x_2} - d_2\dot{x_4} + d_3{x_4}^2 + B_1x_2 - d_4 = -u \tag{4.32}$$

$$-d_2\dot{x_2} + d_6\dot{x_4} - d_3{x_2}^2 + B_2x_4 - B_2x_2 - d_8 = u \tag{4.33}$$

As expressões matemáticas para $\dot{x_2} = \ddot{\alpha}$ e $\dot{x_4} = \ddot{\varphi}$ agora podem ser encontradas resolvendo este sistema como um sistema de equações com duas incógnitas, por qualquer um dos métodos de substituição, igualação ou eliminação.

Aplicando o método de eliminação obtêm-se as relações (4.34) e (4.35), tal que

$$\dot{x_2} = \frac{1}{d_6d_1 - d_2^2}[-d_3d_6{x_4}^2 + d_3d_2{x_2}^2 - x_2(B_1d_6 - B_2d_2) - x_4d_2B_2 + d_4d_6 + d_8d_2 + u(d_2 - d_6)] \tag{4.34}$$

$$\dot{x_4} = \frac{1}{d_6d_1 - d_2^2}[-d_3d_2{x_4}^2 + d_3d_1{x_2}^2 - x_2(B_1d_2 - B_2d_1) - x_4d_1B_2 + d_4d_2 + d_8d_1 + u(d_1 - d_2)]. \tag{4.35}$$

O sistema de equações em variáveis de estado, substituindo as Equações 4.34 e 4.35 em 4.31 fica conformado como mostrado na Equação 4.36.

$$\begin{aligned}
\dot{x_1} &= x_2 \\
\dot{x_2} &= \tfrac{1}{d_6d_1-d_2^2}[-d_3d_6{x_4}^2 + d_3d_2{x_2}^2 - x_2(B_1d_6 - B_2d_2) \\
&\quad -x_4d_2B_2 + d_4d_6 + d_8d_2 + u(d_2 - d_6)] \\
\dot{x_3} &= x_4 \\
\dot{x_4} &= \tfrac{1}{d_6d_1-d_2^2}[-d_3d_2{x_4}^2 + d_3d_1{x_2}^2 - x_2(B_1d_2 - B_2d_1) \\
&\quad -x_4d_1B_2 + d_4d_2 + d_8d_1 + u(d_1 - d_2)].
\end{aligned} \tag{4.36}$$

Já definido completamente o sistema de equações de estado é possível desenvolver o simulador da planta no computador. Observe-se que as equações ainda não estão linearizadas, o qual é necessário para reproduzir o comportamento em qualquer estado do sistema.

Para o projeto dos controladores baseados em métodos convencionais é necessário linearizar as equações ao redor do ponto de operação $\{x_1,x_2,x_3,x_4\}=\{0,0,0,0\}$. Considerando as seguintes aproximações para ângulos pequenos:

- $sen\alpha \approx \alpha;\ sen\varphi \approx \varphi$; para os casos em que α e $\varphi << 1$.

- $\dot{\alpha}^2 \approx \dot{\varphi}^2 \approx 0$.

Ou em variáveis de estado:

- $sinx_1 \approx x_1;\ sinx_3 \approx x_3$.

- ${x_2}^2 \approx {x_4}^2 \approx 0$.

Os termos d_2, d_3, d_4, e d_8 ficam da seguinte forma

$$d_2 = m_2 L_1 l_{c2}\ ;\ d_3 = 0;\ d_4 = (m_1 l_{c1} + m_2 L_1)g;\ d_6 = m_2 l_{c2}^2 + I_{c2}\ ;\ d_8 = m_2 g l_{c2};$$

Substituindo os termos linearizados e organizando em forma de equação de estados como mostrado nas Equações 4.6 e 4.7, obtêm-se a Equação 4.37.

$$\begin{bmatrix} \dot{x_1} \\ \dot{x_2} \\ \dot{x_3} \\ \dot{x_4} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ \frac{d_4 d_6}{d_6 d_1 - d_2^2} & \frac{-(B_1 d_6 - B_2 d_2)}{d_6 d_1 - d_2^2} & \frac{d_2 d_8}{d_6 d_1 - d_2^2} & \frac{-B_2 d_2}{d_6 d_1 - d_2^2} \\ 0 & 0 & 0 & 1 \\ \frac{d_4 d_2}{d_6 d_1 - d_2^2} & \frac{-(B_1 d_2 - B_2 d_1)}{d_6 d_1 - d_2^2} & \frac{d_8 d_1}{d_6 d_1 - d_2^2} & \frac{-B_2 d_1}{d_6 d_1 - d_2^2} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{d_2 - d_6}{d_6 d_1 - d_2^2} \\ 0 \\ \frac{d_1 - d_2}{d_6 d_1 - d_2^2} \end{bmatrix} u \qquad (4.37)$$

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix};\ D = 0.$$

A Equação 4.37 vai permitir desenvolver o projeto de controladores baseados em controle moderno.

4.8 CONCLUSÂO DO CAPÍTULO

Neste capítulo apresentou diferentes formas de representação matemática usadas em sistemas dinâmicos e a representação mais adequada para o sistema utilizado neste trabalho. Foi desenvolvido o modelo matemático para o pêndulo duplo interatuado baseado no método de energias de Lagrange. O sistema de equações dinâmicas é expresso em equações diferenciais de primeira ordem para aplicar o método de Runge Kutta de 4ª ordem. Este método permite simular o comportamento da planta, necessário para o projeto de controladores inteligentes. Além disto, é expresso o sistema em equações de estado para aplicar métodos de controle moderno.

5 FUNDAMENTOS DE CONTROLE

5.1 MÉTODOS CLÁSSICOS DE CONTROLE

A característica básica da resposta transitória de um sistema de malha fechada depende essencialmente da localização dos pólos de malha fechada. Se o ganho de malha do sistema for variável, então a localização dos pólos dependerá do valor do ganho de malha escolhido.

O método do lugar das raízes permite que as raízes da equação característica sejam representados graficamente para todos os valores de um parâmetro do sistema. Este método permite prever quais os efeitos da variação do valor do ganho ou da adição de pólos e zeros de malha aberta sobre a localização de pólos em malha fechada.

Para os sistemas lineares, existem duas técnicas clássicas de projeto de controladores, a saber:

- Projeto por Lugar das Raízes (LR),
- Projeto por resposta em freqüência.

Estas técnicas permitem que, a partir do conhecimento do processo a ser controlado, se escolha um controlador de tal forma que o sistema realimentado apresente um comportamento pré-estabelecido.

5.1.1 Lugar das Raízes

O método do Lugar das Raízes foi proposto por Evans em 1947 (Chen, 1999; Ogata, 2003) e permite determinar a posição dos pólos de um sistema realimentado a partir do conhecimento dos pólos e zeros do sistema a malha aberta e em função do ganho da malha.

A função de transferência do sistema a malha fechada é apresentada na Equação

$$\frac{Y(s)}{U(s)} = \frac{KG(s)}{1+KG(s)}. \tag{5.1}$$

O método permite determinar a posição das raízes de $1+KG(s) = 0$ a partir do conhecimento de $G(s)$.

As raízes do sistema à malha fechada assumem posições diferentes no plano complexo em função do valor de K. Chama-se Lugar das Raízes (LR) o diagrama que apresenta o lugar que as raízes do sistema realimentado ocupam no plano complexo em função de K (Bishop, 1997; Franklin *et al.*, 2002).

5.1.2 Resposta em Freqüência

Os métodos por freqüência são provavelmente os mais empregados em projetos industriais. Apresentam como vantagem o fato de poderem ser empregados sem a necessidade do conhecimento dos pólos e zeros do sistema a ser controlado (conhecimento indispensável no caso do método pelo Lugar das Raízes) e de fornecerem bons resultados mesmo em face de incertezas no modelo da planta em estudo.

Dado um sistema $G(s)$ estável, linear e invariante no tempo, a resposta em regime permanente para uma excitação senoidal representada pela Equação 5.2 esta dada pela Equação 5.3.

$$u(t) = A \sin \omega t \tag{5.2}$$

$$y(t) = A|G(j\omega)| \sin[(\omega t) + < G(j\omega)], \tag{5.3}$$

Ou seja, a saída, em regime permanente (Equação 5.3), tem a mesma freqüência da excitação, porém com uma alteração de amplitude e fase que só dependem de $G(s)$ para $s = j\omega$.

A forma de representação de $G(j\omega)$, chama-se diagrama de Bode. Existe o diagrama de Bode de amplitude e o diagrama de Bode de fase. Ambos colocam as freqüências em escala logarítmica no eixo das abscissas. No diagrama de amplitude, o módulo de $G(j\omega)$ ocupa o eixo das ordenadas também em escala logarítmica, na forma da Equação 5.4 e levam a unidade de decibel (dB), onde

$$20 \log|G(j\omega)|. \quad (5.4)$$

No diagrama de fase, o ângulo de *G(jω)*, usualmente em graus, é colocado no eixo das ordenadas em uma escala linear. Esta representação facilita muito o traçado das curvas de resposta em freqüência. Da mesma forma que existem regras simples para o traçado do LR, existe também procedimentos rápidos para o traçado dos diagramas de Bode.

Os diagramas de Bode não são úteis apenas para informar as mudanças de amplitude e fase em condições de regime permanente e excitação senoidal. Eles também servem, principalmente, para determinar o comportamento dinâmico de sistemas realimentados, a partir do conhecimento da resposta em freqüência de *G*(*s*) (Bishop, 1997; Franklin *et al.*, 2002).

5.2 MÉTODOS DE CONTROLE MODERNO

Em quanto à teoria de controle convencional é fundamentada na relação entrada saída, ou função de transferência, a teoria de controle moderno é fundamentada na descrição de um sistema de equações em termos de *n* equações diferenciais de primeira ordem, que podem ser combinadas em uma equação diferencial vetorial matricial de primeira ordem, como mostrado no capitulo anterior (Ogata, 2003).

A descrição matemática do controle de processos utilizando a relação do sinal de entrada e de saída conhecido como função de transferência tem como principal vantagem de representar sistemas lineares utilizando a transformação de Laplace, de forma que equações diferenciais no tempo são convertidas em equações algébricas mais fáceis de calcular e manipular. Entretanto, não é possível observar e controlar todos os fenômenos internos que envolvem o controle de processos. A moderna teoria de controle é baseada no método do espaço de estados, que providenciam uma uniforme e poderosa representação no domínio do tempo de sistemas multi-variáveis de ordem arbitrária (Chen, 1999).

5.2.1 Conceitos de Controlabilidade e Observabilidade

Um sistema é dito controlável no instante t_0 se for possível, por meio de um vetor de controle não limitado, transferir o sistema de qualquer estado inicial $x(t_0)$ para qualquer outro estado, em um intervalo de tempo finito.

Um sistema é dito observável no instante t_0 se, com o sistema no estado $x(t_0)$, for possível determinar esse estado a partir da observação da saída durante um intervalo de tempo finito.

Os conceitos de controlabilidade e obserbabilidade forem introduzidos por Kalman. As condições de controlabilidade e observabilidade podem ditar a existência de uma solução completa para o problema de projeto do sistema de controle. Então é necessário conhecer as condições nas quais um sistema é controlável e observável.

Retomando o sistema de Equações 4.6 e 4.7 como mostrado nas Equações 5.5 e 5.6.

$$\dot{\boldsymbol{x}} = A\boldsymbol{x} + B\boldsymbol{u} \tag{5.5}$$

$$\boldsymbol{y} = C\boldsymbol{x} + D\boldsymbol{u}\,. \tag{5.6}$$

o sistema é de estado completamente controlável se e somente se, os vetores $B, AB, \ldots, A^{n-1}B$ forem linearmente independentes ou a matriz quadrada *nxn* tiver posto *n* (Equação 5.7).

$$C_x = [B\ |AB|\ \ldots |A^{n-1}B|]. \tag{5.7}$$

A matriz C_x é denominada comumente matriz de controlabilidade.

O conceito de observabilidade é útil na solução de problemas de reconstrução de variáveis de estado não mensuráveis, a partir de variáveis mensuráveis, no menor intervalo possível de tempo. Na prática, a dificuldade encontrada com o controle por realimentação de estado é que algumas das variáveis de estado não são acessíveis por medição direta, resultando ser necessário estimar a variável de estado mensurável para construir os sinais de controle.

Por outro lado, o sistema é completamente observável se o posto da matriz *n* (Equação 5.8) for *n,*

$$O_x = [C\ |CA|\ \ldots |CA^{n-1}|]. \tag{5.8}$$

5.2.2 Sistemas de Controle no Espaço de Estados

Os métodos de interesse para o projeto de sistemas de controle no espaço de estados tratados neste documento são o método de alocação de pólos e o regulador quadrático ótimo (LQR). O método de alocação de pólos é similar ao método do lugar das raízes no qual se alocam os pólos de malha fechada nas posições desejadas. A diferença básica é que no projeto pelo lugar das raízes se alocam somente os pólos dominantes de malha fechada nas posições desejadas, enquanto no projeto por alocação de pólos alocamos todos os pólos de malha fechada nas posições desejadas (Ogata, 2003).

5.2.3 Alocação de Pólos

Para este método de controle se admite que todas as variáveis de estado são mensuráveis e que estão disponíveis para realimentação. Se o sistema considerado foi de estado completamente controlável, então os pólos de malha fechada do sistema poderão ser alocados em qualquer posição desejada por meio de uma realimentação de estado, empregando uma matriz de ganho apropriada.

Para que os pólos de malha fechada possam ser alocados em posições arbitrarias no plano *s* o estado do sistema precisa ser completamente controlável (Ogata, 2003).

Na abordagem convencional para o projeto de sistemas de controle com uma entrada e uma saída, projeta-se o controlador de modo que os pólos dominantes a malha fechada apresentem uma determinada característica temporal. Nesta abordagem os efeitos na resposta temporal devido aos pólos não dominantes são desprezados. O controlador é colocado em série com o processo *G(s)* sendo alimentado pelo erro entre o valor de referencia *R(s)* e o valor de saída *Y(s)***.** Os controladores mais empregados são do tipo PID e os compensadores de avanço e/ou atraso de fase.

Considere o sistema de equações de estado descrito pelas Equações 5.5 e 5.6, aplicando uma realimentação linear de estado da forma

$$u(t) = Nr(t) - Kx(t), \tag{5.9}$$

onde *r(t)* é o nome para o sinal de entrada, *N* é o ganho de pré-compensação, *x(t)* é o vetor de estados e *K* é o ganho de realimentação.

Substituindo a Equação 5.9 em 5.5, obtêm-se a Equação 5.10

$$\dot{x(t)} = (A - BK)x(t). \tag{5.10}$$

O diagrama de blocos correspondente é apresentado na Figura 5.1.

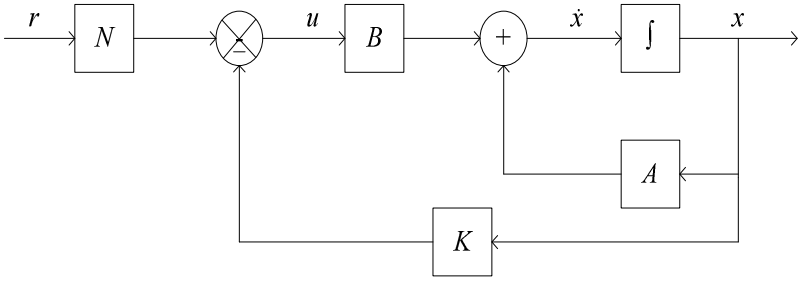


Figura 5.1- Diagrama de blocos referente à realimentação de estados.

Note que a referência foi tomada como zero (ponto de operação é zero graus). Os sistemas que incluem controladores podem ser divididos em duas categorias, sistemas reguladores onde a sinal de referência é constante incluindo o zero, e sistemas de controle onde o sinal de referência varia com o tempo.

A estabilidade e a característica da resposta temporal são determinadas pelos autovalores da matriz (*A-BK*), estes autovalores são chamados de pólos reguladores.

A técnica freqüentemente utilizada para escolher a alocação dos pólos de malha fechada desejados está baseada na escolha desses pólos com base na experiência do projeto pelo lugar das raízes, alocando um par de pólos dominantes de malha fechada e escolhendo os outros pólos de modo que eles fiquem bem distantes, a esquerda dos pólos dominantes de malha fechada (Ogata, 2003).

Existem diferentes formas de determinar a matriz de ganhos *K*, a saber: (a) determinação da matriz *K* com a utilização da matriz de transformação *T*, (b) utilização do método de substituição direta ou (c) utilização da fórmula de Ackermann (Ogata, 2003).

Neste trabalho será utilizada a fórmula de Ackermann a qual é expressa pela Equação 5.11

$$K = [0 \ 0 \ \ldots \ 0 \ 1][B \ |AB| \ldots \ |A^{n-1}B|]^{-1}\phi(A), \tag{5.11}$$

onde A e B são as matrizes da Equação 5.5, $\phi(A)$ é definida na Equação 5.12.

$$\phi(A) = A^n + q_{n-1}A^{n-1} + \cdots + q_1 A + q_0 I \tag{5.12}$$

onde q_i são os coeficientes do polinômio característico desejado (a partir dos pólos escolhidos) e I é a matriz identidade.

5.2.4 Controle Ótimo

Alocar os pólos dominantes de malha fechada distantes do eixo $j\omega$, pode fazer que a resposta do sistema se torne rápida, então os sinais no sistema se tornarão elevados, fazendo como que o sistema se torne não-linear, o que deve ser evitado. O controle ótimo determina os pólos desejados de malha fechada para que haja uma conciliação entre a resposta aceitável e o total de energia de controle requerida (Ogata, 2003).

5.2.5 Regulador Quadrático Linear (LQR)

Este método ao invés de utilizar a localização dos pólos como critério de projeto, baseia-se na minimização de um critério quadrático que está associado à energia das variáveis de estado e dos sinais de controle a serem projetados.

O objetivo é determinar a matriz K do vetor de controle ótimo da Equação 5.9, que pode se reescrever como na Equação 5.13

$$u(t) = -Kx(t), \tag{5.13}$$

de modo a minimizar o índice de desempenho

$$\int_0^\infty J(x^T Q x + u^T R u)dt, \tag{5.14}$$

onde Q é uma matriz Hermitiana ou real simétrica e definida positiva (ou semi-definida positiva), R é uma matriz hermitiana ou real simétrica e definida positiva. Note-se que o segundo termo no segundo membro da Equação 5.14 exprime o consumo de energia dos sinais de controle. As matrizes Q e R determinam a importância relativa do erro e do consumo de energia.

Quando se utiliza o critério quadrático, a seleção das matrizes peso torna-se um processo trabalhoso. Usualmente, esta seleção consiste em verificar, após várias simulações do problema, quais os valores destas matrizes que melhor satisfazem a certos critérios (como por exemplo, porcentagem de sobre-passo, máximo controle e tempo de estabilização), que quando alcançados, refletem um melhor desempenho do sistema. Isto se deve ao fato de não existir um método sistemático para tal seleção (Ferreira, 2002).

É usual adotar-se a forma diagonal para Q e R, pois esta possibilita que as componentes do estado e do controle sejam penalizadas individualmente, facilitando assim o ajuste e a interpretação física destas. A sistemática empregada neste trabalho para a seleção das matrizes pesos está baseada inicialmente de varias simulações.

Neste método o vetor de ganhos pode ser obtido a partir da Equação 5.15,

$$K = R^{-1}B'P, \tag{5.15}$$

onde P satisfaze a equação de Ricatti apontada pela Equação 5.16,

$$A'P + PA - PBR^{-1}B'P + Q = 0. \tag{5.16}$$

5.3 CONTROLE INTELIGENTE

O controle inteligente descreve a disciplina onde os métodos de controle são desenvolvidos para tentar simular ou emular características da inteligência humana. Hoje, a área do controle inteligente tende a abranger tudo que não é caracterizado como o controle convencional. Muitas vezes, mesmo sob a ausência de um modelo matemático preciso ou de algoritmos bem definidos, o operador humano é capaz de agir sobre uma planta dada, utilizando a experiência acumulada ao longo do tempo. Entre as tarefas que podem ser

realizadas por sistemas baseados em conhecimento estão a interpretação de dados, predição, diagnóstico, síntese, planejamento, monitoração, correção de falhas, treinamento e controle ativo. (Nascimento e Cairo, 2000; Luger, 2004).

Nos últimos anos, houve um interesse crescente na utilização de estratégias não convencionais do controle tais como sistemas difusos, as redes neurais artificiais (RNs), algoritmos genéticos (AGs) etc., e a conformação de sistemas híbridos entre estes tipos de controle para melhorar a extração de informações relevantes dos sistemas de estudo e encontrar formas mais eficazes de extração do conhecimento (Eksin e Erol, 1996; Santos *et al.,* 1999; Zong-Mu e Kuei-Hsiang, 2004). Estes métodos de controle derivam suas vantagens do fato de que não usam nenhum modelo matemático do sistema. Em lugar de usar relações de entrada/saída (rede neural) ou o conhecimento heurístico (lógica nebulosa) sobre o sistema.

5.3.1 Controlador Difuso

O controle difuso tem se mostrado uma alternativa viável ao controle clássico no controle de processos com parâmetros variantes no tempo, não-lineares e com informações imprecisas. Os controladores nebulosos freqüentemente oferecem um desempenho igual ou superior aos controladores convencionais (Copetti *et al.,* 2007). Técnicas de controle clássicas são baseadas em modelos linearizados dos sistemas físicos, o que representa perdas de informações que muitas vezes são importantes para o funcionamento da planta.

A teoria de controle difuso foi proposta originalmente pelo prof. Lotfi A. Zadeh (Zadeh, 1965) da Universidade de Califórnia, em Berkeley (EUA) e virou uma forma quantitativa e certamente eficaz aos problemas que envolvem incerteza e ambigüidade. A teoria, que é agora bem conhecida, foi projetada especificamente a representar matematicamente a incerteza e fornecer ferramentas formalizadas para tratar a imprecisão que é intrínseca a problemas do mundo real. A habilidade da lógica difusa para tratar a incerteza e o ruído conduziu a seu uso nos controles (Kumar *et al*., 2004).

A essência do controle e necessariamente (TS pode ser usado para tal) difuso é explorar o conhecimento do operador humano (do processo) de forma a permitir a configuração de projetos de controle com desempenho satisfatório. O controle nebuloso resulta no projeto

de sistemas de controle não-lineares. Assim, o processo de configuração e sintonização dos parâmetros próprios do controlador é uma tarefa complexa o qual consiste em examinar a influência de cada parâmetro no desempenho e robustez do projeto (Coelho *et al.*, 2003).

As entradas do controlador são de tipo real, são variáveis numéricas que são fuzzificadas. Isto significa que são associadas com os conjuntos difusos escolhidos e com os graus de pertinência em ditos conjuntos. A máquina de inferência deve receber essas informações sobre as variáveis de entrada em termos associados a variáveis lingüísticas, especificamente graus de pertinência. A máquina de inferência gera, baseada nas regras fornecidas pelo usuário, uma saída também como tipo lingüístico que deve ser convertida para uma variável numérica. Este processo de conversão para variáveis numéricas é conhecido como defuzzificação e existem principalmente dois métodos utilizados para tal fim: (a) defuzzificação por Centro de Área (COA) e (b) defuzzificação dos máximos (MOM) (Nascimento e Cairo, 2000). A Figura 5.2 mostra a estrutura típica do controlador difuso.

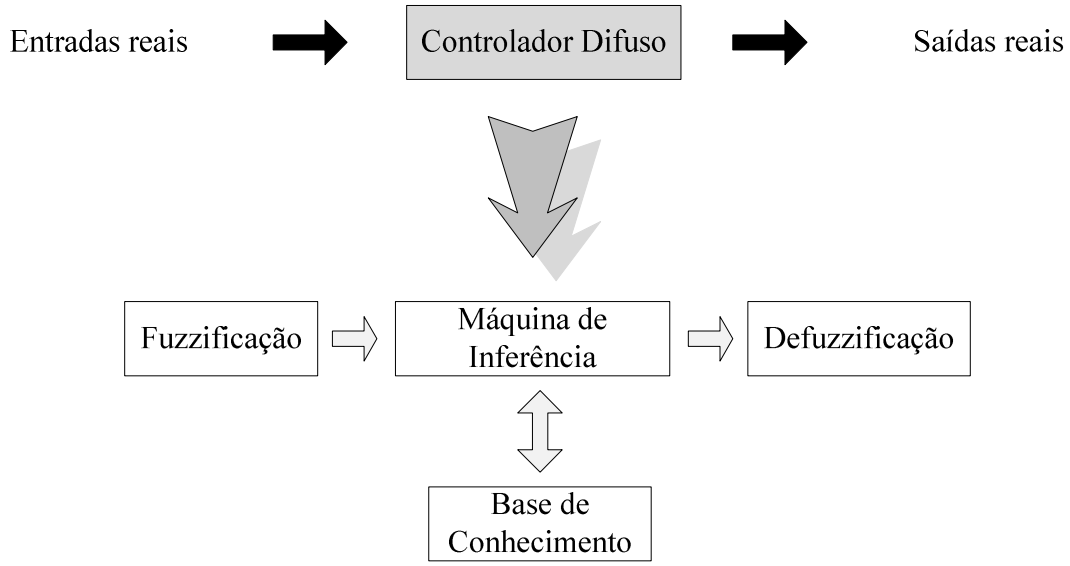


Figura 5.2- Estrutura típica de um controlador difuso

O processo de ajuste do controlador difuso consiste em ajustar os parâmetros das funções de pertinência escolhidas na fase de nebulização que depende do significado lingüístico definido para este conjunto e de sua interpretação no contexto do universo de discurso utilizado (Coelho, 2003).

A Figura 5.3 mostra algumas destas funções. Pode-se observar que uma função Gaussiana têm dois parâmetros de ajuste (σ,c);

$$f(x;\sigma,c) = e^{\frac{-(x-c)^2}{2\sigma^2}}. \tag{5.17}$$

A função triangular têm três parâmetros de ajuste (a,b,c);

$$f(x;a,b,c) = \max\left(\min\left(\frac{x-a}{b-a},\frac{c-x}{c-b}\right),0\right). \tag{5.18}$$

A função trapezoidal têm quatro parâmetros de ajuste (a,b,c,d);

$$f(x;a,b,c,d) = \max\left(\min\left(\frac{x-a}{b-a},1,\frac{d-x}{d-c}\right),0\right). \tag{5.19}$$

Finalmente a função de Bell têm dois parâmetros (a,c);

$$f(x;a,b,c) = \frac{1}{1+\left|\frac{x-c}{a}\right|^{2b}}. \tag{5.20}$$

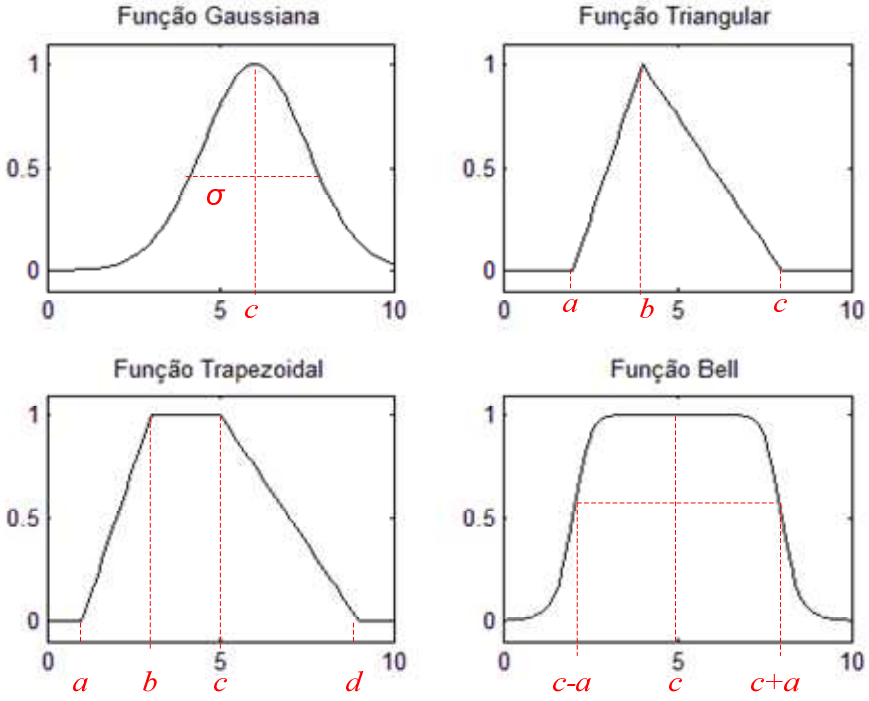


Figura 5.3- Funções de pertinência típicas e parâmetros de ajuste

5.3.1.1 Controladores Difusos Hierárquicos

Quando os sistemas são complexos, os mesmos contêm conhecimento complexo que faz com que a geração de regras difusas se torne mais elevada com o crescimento do número de entradas, do número de saídas e com as funções de pertinência. Para problemas de controle multi-variável (*Multiple Inputs, Multiple Outputs* - MIMO e *Multiple Inputs, Single Output* - MISO), é preferível descompor o sistema em subsistemas de controle para diminuir a complexidade (Zong-Mu *et al.*, 2004).

A arquitetura hierárquica propõe a geração de estados intermediários (multi-estados) onde as variáveis de entrada podem ser as variáveis de entrada do sistema ou as variáveis provenientes de outros estados, e as variáveis de saída podem ser entradas para outros subsistemas ou as saídas que vão modificar o estado da planta. O efeito é a decomposição da complexidade do sistema em cascata, o que permite identificar três tipos de variáveis lingüísticas como segue: (a) variáveis de entrada, (b) variáveis de saída e (c) variáveis intermédiáras. As variáveis geram conseqüentemente a criação de regras difusas intermediárias para estes multi-estados.

No projeto deste controlador, mesmo que não foi utilizada a geração de múltiplos estados, foi adotada a idéia de descompor o sistema em subsistemas, ajustando a saída dos subsistemas com ganhos constantes e utilizando uma operação de subtração para juntar os sinais de saída, conseguido assim um sinal de controle satisfatório para o controle do pêndulo.

As duas classes de controladores nebulosos mais utilizados são o controlador lingüístico de Mamdani (Miranda *et al.*, 2003) e o controlador do tipo interpolativo Takagi-Sugeno (Geuntaek e Wonchang, 1998). O controlador Mamdani utiliza conjuntos nebulosos como conseqüentes, em quanto os controladores Takagi-Sugeno entregam funções lineares como conseqüentes. Devido a esta diferença, as regras de controle de Mamdani são significativamente mais intuitivas linguisticamente, enquanto as regras de Takagi-Sugeno parecem apresentar mais poder de interpolação com um número reduzido de regras de controle (Geuntaek e Wonchang, 1998; Coelho *et al.*, 2003).

5.3.2 Redes Neurais - Aprendizado Supervisionado

O neurônio artificial é a unidade da arquitetura das redes neurais artificiais. Na estrutura do neurônio (Figura 5.5.a) observa-se:

- Um conjunto de entradas que recebem os sinais de entrada do neurônio,
- Um conjunto de sinapses cujas intensidades são representadas por pesos associados a cada uma delas.

Uma função de ativação relaciona as entradas e suas sinapses ao limiar da função a fim de definir se o neurônio será ativado ou não. O resultado da sinapse total de entrada é dado pelo produto interno do vetor de entrada pelo vetor de pesos, e a saída *y* é obtida pela aplicação de *f(u)*.

Assim, nas implementações de RNs que seguem um modelo similar de neurônio precisa-se computar, paralelamente, um número de produtos internos que cresce exponencialmente com a quantidade de conexões na rede.

Algumas funções de limiar usadas são apresentadas na Figura 5.4, e estão definidas pelas seguintes relações:

- Função sigmóide logística;

$$f(u) = \frac{1}{1+e^{-au}}. \tag{5.21}$$

- Função Limiar ou degrau unitário;

$f(u) = 1$ se $u > 0$, $f(u) = 0$, caso contrario. (5.22)

- Função Tangente hiperbólica;

$$f(u) = a\frac{e^{bu}-e^{-bu}}{e^{bu}+e^{-bu}}\ , \tag{5.23}$$

onde *a* é o parâmetro de inclinação da curva, *b* são os limites inferiores e superiores no gráfico, e *u* e o valor de ativação da unidade.

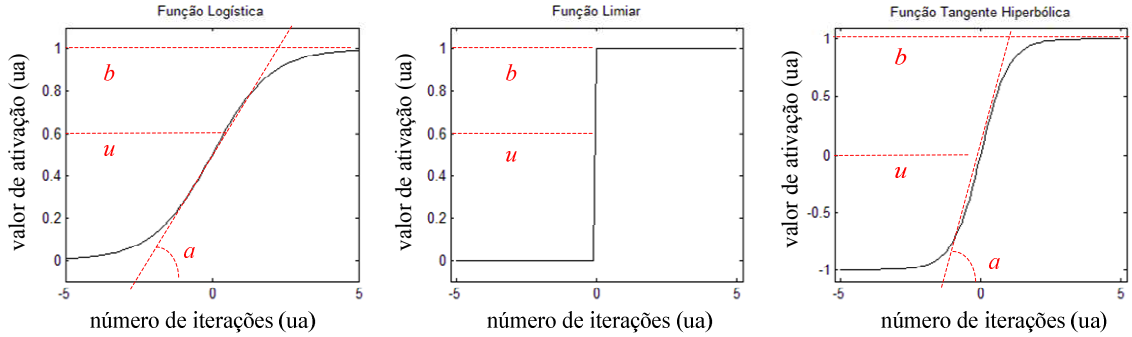


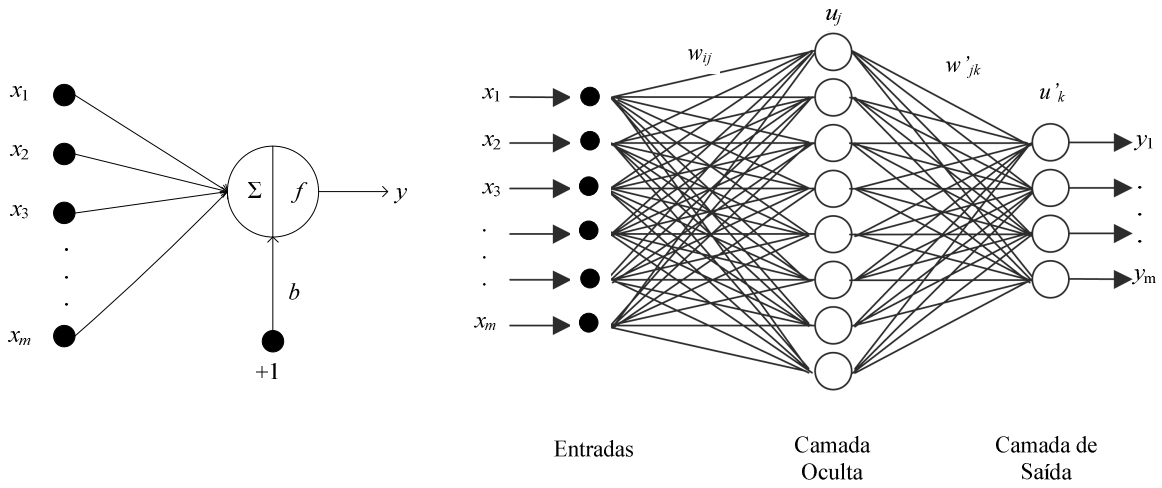
Figura 5.4- Funções de ativação dos neurônios.

Uma rede neural adaptativa é uma estrutura que consiste de um número de neurônios conectados através de links direcionados, cada neurônio representa uma unidade de processo, e as conexões entre os neurônios especificam uma relação causal entre os nós conectados. Todas as partes dos nós são adaptáveis, o qual significa que a saída desses nós depende de seus parâmetros modificáveis. A regra de aprendizado especifica como esses parâmetros devem ser atualizados para minimizar uma medida de erro pré-estabelecida.

A estrutura de conhecimento distribuído faz com que as redes neurais apresentem uma maior tolerância à incerteza e ao ruído, são freqüentemente chamadas de caixa preta, já que é difícil interpretar o conhecimento armazenado nelas ao contrario que a lógica difusa (Jyh-Shing *et al.*,1997; Kumar, 2004).

Uma rede neural adaptativa é uma estrutura que consiste de um número de nodos conectados a través de links direcionados, cada nodo representa uma unidade de processo, e as conexões entre os nodos especificam uma relação causal entre os nodos conectados. Todas as partes dos nodos são adaptáveis, o qual significa que a saída desses nodos depende de seus parâmetros modificáveis. A regra de aprendizado especifica como esses parâmetros devem ser atualizados para minimizar uma medida de erro pré-estabelecida.

A estrutura de conhecimento distribuído faz com que as redes neurais apresentem uma maior tolerância à incerteza e ao ruído, são freqüentemente chamadas de caixa preta, já que é difícil interpretar o conhecimento armazenado nelas ao contrario que a lógica difusa (Jyh-Shing *et al.*,1997; Kumar, 2004).



a. Modelo do neurônio artificial

b. Rede perceptron multicamadas

Figura 5.5- Representação de uma rede neural artificial

Entre as redes mais populares encontra se o tipo Perceptron multicamadas (PMC), a Figura 5.5.b. apresenta uma arquitetura com uma camada oculta, embora não existe uma limitante no uso de número de camadas ocultas, o teorema de aproximação universal dita que toda função contínua que mapeia números reais de entrada, e algum intervalo de números reais de saída, pode ser arbitrariamente aproximada com precisão por um Perceptron multicamada com somente uma camada oculta.

Existem diferentes técnicas para o aprendizado da rede, sendo a mais usada a denominada retropropagação do erro, este método baseia-se no princípio de correção do erro (Narendra e Parthasarathy,1990; Mohandas *et al.*, 1998; Haykin, 1999).

As redes neurais podem ser usadas de duas formas no projeto de sistemas de controle. Podem ser usadas para obter um modelo matemático de um sistema real para ser controlado ou para projetar um controlador uma vez um modelo do sistema real esteja disponível. Uma quantidade considerável de modelos de redes neurais têm sido classificados de acordo com diferentes critérios, tais como seus métodos de aprendizagem (supervisionado, não supervisionado), arquitetura (*feedforward, backforward*), tipos de saídas (binárias, contínuas), tipos de neurônios (unifomes, híbridos), implementações (*software*, *hardware*), pesos de conexão (ajustáveis, conexões fortes), etc (Jyh-Shing *et al.*,1997).

Os diferentes métodos de aprendizado podem ser classificados de acordo com a participação do supervisor no processo de aprendizado. No grau de supervisão mais forte possível, o supervisor fornece diretamente para a rede neural os valores dos pesos, num grau de supervisão menor denominada supervisão forte, o supervisor fornece para a rede neural um conjunto de treinamento, ou seja, um conjunto de entradas e suas respectivas saídas desejadas. Na supervisão denominada fraca, o supervisor faz apenas o papel de um crítico, fornecendo uma avaliação grosseira da saída da rede neural (por exemplo, certo ou errado, erro grande ou pequeno, etc.) (Nascimento e Cairo, 2000).

5.3.3 Sistemas Neuro-Difusos

Os sistemas neuro-difusos misturam a capacidade de aprendizado das redes neurais com o poder de interpretação lingüística dos sistemas de inferência difusos, obtendo-se resultados tais como aplicabilidade de algoritmos de aprendizagem desenvolvidos para as redes neurais, possibilidade de promover a integração do conhecimento e a possibilidade de extrair conhecimento para uma base de regras difusas a partir de um conjunto de dados.

O sistema neuro-difuso consiste de um sistema difuso tradicional exceto que cada etapa, pode proporcionar capacidades de aprendizado de redes neurais para aperfeiçoar o conhecimento do sistema (Jang, 1993).

O sistema neuro-difuso ANFIS (*Adapative Neuro-Fuzzy Inference Systems*) usa um conjunto de dados de entrada-saída para construir um sistema de inferência difuso cujos parâmetros de projeto são ajustados usando o algoritmo de retropropagação do erro, ou uma mistura híbrida com o método dos mínimos quadrados em batelada (*batch*).

Considere o sistema da Figura 5.6 que permitirá entender o comportamento da estrutura neuro-difusa. Neste caso, o sistema possui duas entradas e uma saída, um sistema difuso tipo Takagi-Sugeno (de primeira ordem) conformado por um conjunto de regras com 2 regras difusas pode ser expressado como:

$$Regra\ 1: \quad \textbf{Se}\ x \text{ é } A_1\ \textbf{e}\ y \text{ é } B_1\ \textbf{então}\ f_1 = p_1x + q_1y + r_1$$

$$Regra\ 2: \quad \textbf{Se}\ x \text{ é } A_2\ \textbf{e}\ y \text{ é } B_2\ \textbf{então}\ f_2 = p_2x + q_2y + r_2$$

onde os parâmetros das funções de pertinência A_1, A_2, B_1, B_2 são chamados de parâmetros do antecedente e os parâmetros das funções lineares de saída p, q e r são chamados parâmetros do conseqüente.

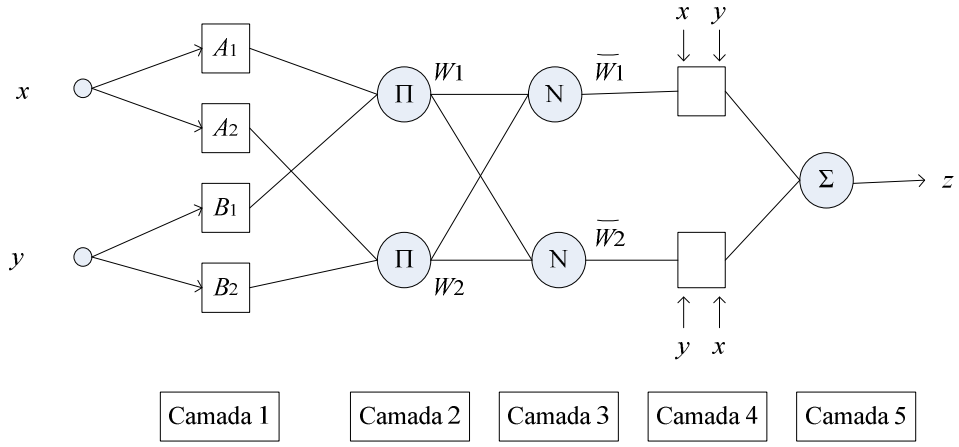


Figura 5.6- Arquitetura ANFIS (Jang, 1993, modificado)

Cada nó da *Camada 1* é um nó adaptativo com uma saída definida pelos graus de pertinência das entradas *x, y* aos conjuntos difusos definidos, como mostrado nas Equações 5.17 e 5.18.

$$saída\ Camada\ 1, i = \mu_{A_i}(x) \text{ para } i = 1,2 \tag{5.17}$$

$$saída\ Camada\ 1, i = \mu_{B_{i-2}}(y) \text{ para } i = 3,4 \tag{5.18}$$

Cada nó da *Camada 2* é um nó fixo chamado *Π* que efetua uma operação de multiplicação entre os graus de pertinência entregados pela *Camada 1* (Equação 5.19).

$$saída\ Camada\ 2, i = W_i = \mu_{A_i}(x).\mu_{B_i}(y) \text{ para } i = 1,2. \tag{5.19}$$

Cada nó da *Camada 3* é um nó fixo chamado *N* que pondera a força de acendimento da regra *i* em relação com a soma de todas as forças de acendimento das regras (Equação 5.20).

$$saída\ Camada\ 3, i = \overline{W}_i = \frac{W_i}{W_1 + W_2}, \quad i = 1,2,\ldots \tag{5.20}$$

Cada nó da *Camada 4* é um nó adaptativo, as saídas dos neurônios desta camada depende de uma função polinomial linear que esta relacionado com os parâmetros de entrada *x, y* e os parâmetros de ajuste do conseqüente Equação 5.21, tal que

$$saída\ Camada\ 4, i = \overline{W}_i f_i = \overline{W}_i(p_i x + q_i y + r_i), \quad i = 1,2,\ldots \tag{5.21}$$

A *Camada 5* tem um único nó fixo chamado *Σ*, que computa a saída geral como a soma de todos os sinais entrantes provenientes da *Camada 4* (Equação 5.22), onde

$$saída\ Camada\ 5, i = \sum \overline{W}_i f_i = \frac{\sum_i W_i f_i}{\sum_i W_i}\ , i = 1,2,\ldots \tag{5.22}$$

Assim tem se construída uma rede adaptativa que tem exatamente a mesma função que um modelo tipo Takagi Sugeno de primeira ordem (Jang, 1993). O processo de treinamento do sistema difuso pode ser feito usando o algoritmo *backpropagation* ou um algoritmo híbrido que utiliza também o método dos mínimos quadrados (Haykin, 2001).

O algoritmo de treinamento híbrido utiliza dois passos, um passo para o frente e um passo para trás. No passo para o frente as saídas do nó se propagam para o frente até a camada 4, e os parâmetros do conseqüentes (*p, q e r*) são identificados pelo método dos mínimos quadrados. No passo para trás, a retropropagação do erro atualiza os parâmetros do antecedente (os parâmetros dos conjuntos difusos escolhidos) usando o método retropropagação do erro. A Tabela 5.1 resume o funcionamento.

Tabela 5.1- Funcionamento do Algoritmo Híbrido (Jang, 1993, modificado)

	Passo para o Frente	**Passo para trás**
Parâmetros do antecedente	Fixos	Descida de encosta
Parâmetros do conseqüente	Método dos Mínimos Quadrados	Fixos
Sinais usados	Saídas dos neurônios	Sinais de erro

Um número amplo de algoritmos foi desenvolvido para direcionar o problema de aprendizagem de regras nebulosas e ajustar as funções de pertinência usando redes neurais (FSOM – ***F****uzzy* ***S****elf-****O****rganized* ***M****ap*, NEFCLASS – ***NE****uro* ***F****uzzy* ***CLASS****ification*, FUZZYTECH, etc.). O trabalho desenvolvido por Jang (1993) é um dos trabalhos que abriu o caminho neste campo. Os modelos neuro-difusos tradicionais (ANFIS) ajustam apenas seus parâmetros ou têm uma capacidade limitada de ajuste em sua estrutura. Devido ao problema de explosão do número de regras, esses modelos são geralmente utilizados em aplicações com número restrito de entradas (Kelly *et al.,* 1994). Além disso, a maioria desses sistemas neuro-fuzzy tradicionais são adequados para treinamento supervisionado. Entretanto, quando a informação sobre a saída desejada não está disponível, é necessário realizar o aprendizado dos sistemas neuro-difusos através do algoritmo de aprendizado por reforço (RL) (Junichiro *et al.*, 1999).

5.4 CONCLUSÃO DO CAPÍTULO

Neste capítulo foram apresentados aspectos importantes de métodos de controle baseados em controle convencional, controle inteligente, fundamentos teóricos em quanto a configuração, modos de funcionamento e arquiteturas, diferenças entre controladores, e algumas considerações que devem ser consideradas no projeto dos mesmos.

È claro que para o controle de sistemas multi-variáveis é mais adequado o uso de controle moderno e a representação em espaço de estados. O projeto destes controladores deve ser inicialmente baseado em especificações de desempenho para serem atingidas (sobre-passo máximo, e tempo de estabilização). Para tanto, são usadas técnicas de controle convencional para definir uma região de estabilidade, ou um critério de custo mínimo baseado na energia do sistema. Entretanto, o ajuste destes é uma medida empírica o qual apresenta similaridade com os controladores inteligentes.

A maior diferença entre os controladores de tipo convencional e inteligente é a dependência por parte dos primeiros sobre o modelo matemático e a linearização ao redor do ponto de operação.

Os controladores baseados em lógica difusa podem ser descompostos para diminuir complexidade no projeto, os controladores neurais apresentam a característica paralela do

processado distribuído, além de sua capacidade de representação de modelo de sistemas. Os sistemas neuro-difusos mostram-se como uma boa alternativa de otimização baseado no aprendizado supervisionado.

No próximo capítulo são desenvolvidos os controladores baseados em controle moderno e controle inteligente. Isto devido a que o sistema de estudo é multi-variável, e com o intuito de observar a metodologia de projeto dos mesmos e procurar semelhanças e diferenças em quanto a resposta e desempenho.

6 ANÁLISE, RESULTADOS E DISCUSSÃO

Diferentes tipos de controladores podem ser projetados de acordo com a teoria mostrada sobre controladores baseados em inteligência artificial e controle moderno. Neste capítulo apresentam-se os resultados do projeto de cinco tipos de controladores, os dois primeiros baseados em técnicas de controle moderno: (a) projeto de controlador de **R**ealimentação de **E**stados (RE) mediante a alocação de pólos e (b) a técnica do **R**egulador **L**inear **Q**uadrático (LQR) que se baseia no critério de custo. Os outros três controladores restantes foram projetados baseados em técnicas de controle inteligente. Neste caso, foram projetados (a) um controlador difuso, (b) um controlador neural e, finalmente, (c) um sistema neuro-difuso. A seguir, se descrevem detalhes e a metodologia seguida no projeto dos controladores, onde é realizada uma comparação qualitativa (comparação da resposta do sistema e o comportamento dos controladores) e quantitativa (mediante o uso de índices de desempenho) entre eles. Os resultados obtidos neste trabalho podem ajudar no desenvolvimento futuro de protótipos de robôs que andam de bicicleta (Michini, 2006).

6.1 ÍNDICES DE DESEMPENHO

Quando um sistema é proposto para uma aplicação dada, o mesmo deve satisfazer requerimentos específicos, estes requerimentos podem estar relacionados com a resposta do sistema ou a otimização do sistema em um caminho determinado. Os requerimentos que um sistema de controle deve reunir são geralmente denominados de especificações de desempenho. O desempenho de um sistema de controle pode ser considerado em três sentidos, o primeiro tem a ver diretamente com a resposta do sistema, o segundo tem a ver com o índice de desempenho que é uma função do erro ou da saída, e o terceiro tem a ver com o erro do sistema causado pela variação de parâmetros.

Dois índices de desempenho para os controladores desenvolvidos neste trabalho serão levados em conta. Estes índices permitem a análise quantitativa em relação ao comportamento dos controladores, as variações dos sinais de controle e ao comportamento do erro ao longo do tempo de estabilização. Isto indica quão rápido o controlador alcança a posição vertical e quanto esforço é obtido pelo atuador para atingir o objetivo.

Os índices de desempenho são o somatório do erro quadrático (MSE, –*Mean Square Error*) e o somatório do incremento do controle quadrático segundo as Equações 6.1 e 6.2 (Copetti *et al.*, 2007), tal que

$$J_1 = \frac{1}{N}\sum_{k=1}^{N}[e(k)]^2 \tag{6.1}$$

$$J_2 = \frac{1}{N}\sum_{k=1}^{N}[u(k) - u(k-1)]^2. \tag{6.2}$$

6.2 Controle por Realimentação de Estados

O controle por realimentação de estados consiste em obter um vetor de ganhos *K,* que permite obter o sistema em malha fechada. Este vetor de ganhos faz parte da lei de controle, a qual descreve de uma forma adequada, o comportamento das variáveis de estado ao redor de um ponto de operação pré-estabelecido, e com umas condições de projeto que são ajustadas mediante a alocação de pólos no semi-plano esquerdo do plano complexo, o qual garante a estabilidade (Franklin *et al*., 2002).

Foram assumidos os seguintes valores como parâmetros de simulação e teste do pêndulo duplo interatuado: (a) massas das duas barras $m_1 = m_2 = 1$ Kg, (b) comprimentos L_1=L_2=0,4m, (c) aceleração da gravidade g=9,8m/s^2 e (d) amortecimento das juntas B_1=B_2 = 0,05 Ns/m. O centro de gravidade das barras é considerado no centro das barras.

Para a simulação da planta foi escolhido um passo fixo de integração de *h*=0,0025, ou seja, o erro de truncamento local do algoritmo de Runge Kutta é de 9,76 x 10^{-14}.

O primeiro passo é encontrar as matrizes *A, B, C* e *D* do sistema de equações de estado como mostrado na Equação 6.3, onde

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 31{,}8247 & -0{,}0387 & 9{,}0928 & -0{,}2320 \\ 0 & 0 & 0 & 1 \\ 27{,}2784 & 0{,}5026 & 28{,}7938 & -0{,}7345 \end{bmatrix}, B = \begin{bmatrix} 0 \\ -0{,}7733 \\ 0 \\ 10{,}0515 \end{bmatrix} \tag{6.3}$$

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \text{D} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

Como dito no capítulo anterior, a estabilidade do sistema pode ser avaliada com a equação característica associada com a matriz do sistema *A*. O polinômio característico fica como mostrado na Equação 6.4, tal que

$$s^4 + 0.7732s^3 - 60.4736s^2 - 22.7320s + 668.3196 = 0. \qquad (6.4)$$

As raízes do polinômio característico (pólos em malha aberta) são encontradas para os valores (s + 6,9999)(s + 3,9925)(s – 6,5911)(s – 3,6282). A condição para estabilidade é que todos os pólos tenham parte real negativa ($R_e(s_i) < 0$, para todo i), logo pode se observar que o sistema apresenta dois pólos no semi-plano positivo localizados em (6,5911, 3,6282). Isto quer dizer que o sistema é instável, como era de se esperar. A Figura 6.1 apresenta o esboço do lugar das raízes encontrado usando o programa Matlab, da empresa MathWorks.

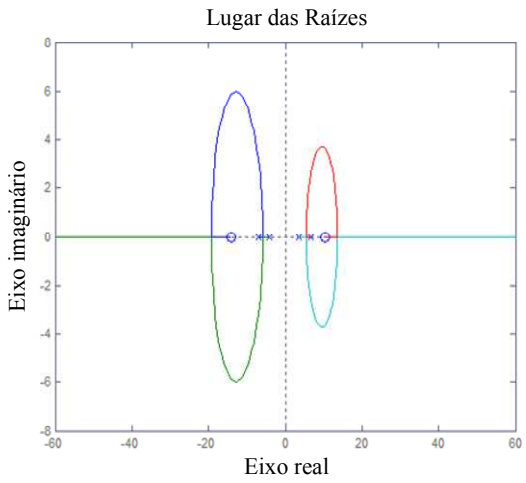
A Figura 6.1.a. mostra a localização dos pólos e zeros em malha aberta assim como a resposta do sistema com uma entrada impulso. Observa-se que o sistema diverge do ponto de operação $\{x_1,x_2,x_3,x_4\}=\{0,0,0,0\}$.

Antes de projetar os controladores deve-se comprovar que o sistema é totalmente controlável. A matriz de controlabilidade é apresentada na Equação 6.5.

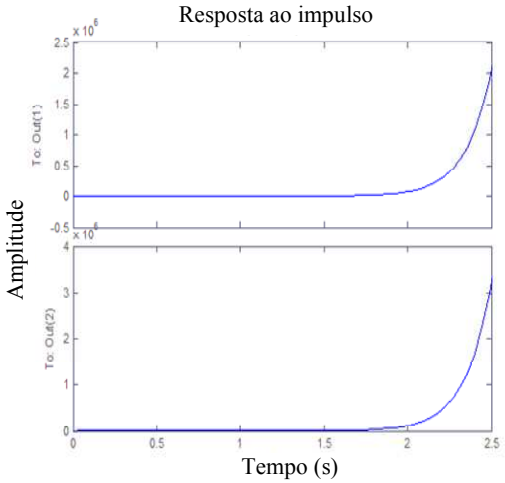
$$Cx = \begin{bmatrix} 0 & -0{,}7732 & -2{,}3017 & 68{,}6815 \\ -0{,}7732 & -2{,}3017 & 68{,}6815 & -209{,}8697 \\ 0 & 10{,}0515 & -7{,}7718 & 272{,}8828 \\ 10{,}0515 & -7{,}7718 & 272{,}8828 & -452{,}4899 \end{bmatrix} \qquad (6.5)$$

O posto da matriz é n=4, ou seja, tanto as filas como as colunas são linearmente independentes. Por tanto, o sistema é totalmente controlável. Isto significa que teoricamente qualquer alocação de pólos arbitrária no semi-plano esquerdo do plano complexo pode fazer o sistema estável em relação ao ponto de operação desejado.

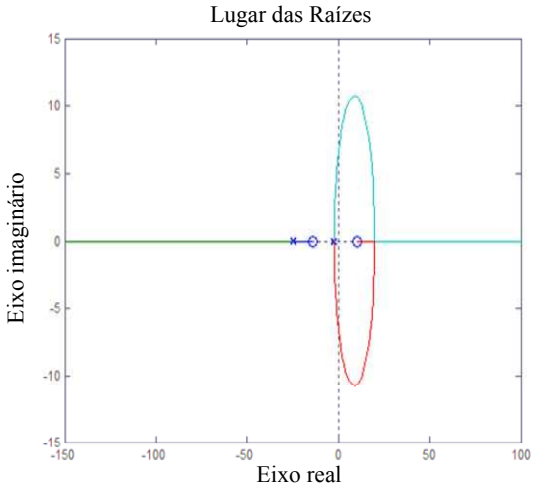
A Figura 6.1.c e Figura 6.1.e, mostram os resultados de escolher a alocação dos pólos em [-2,5, -2, -25, -25] e [-2,5–5i, -2,5+5i, -25, -25] respectivamente, assim como a resposta do sistema em malha fechada (A - KB) a uma entrada impulso. Os valores dos ganhos são obtidos usando a fórmula de Ackermann (Ogata, 2003).



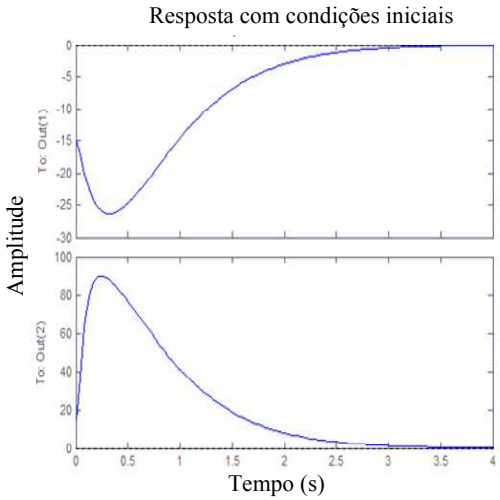
(a) Pólos em malha aberta do sistema linearizado



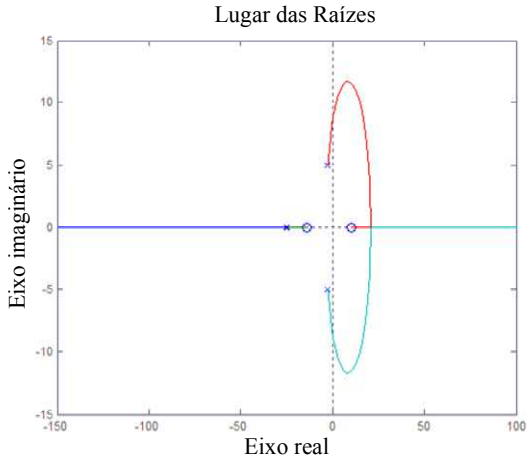
(b) Resposta do sistema a uma entrada impulso



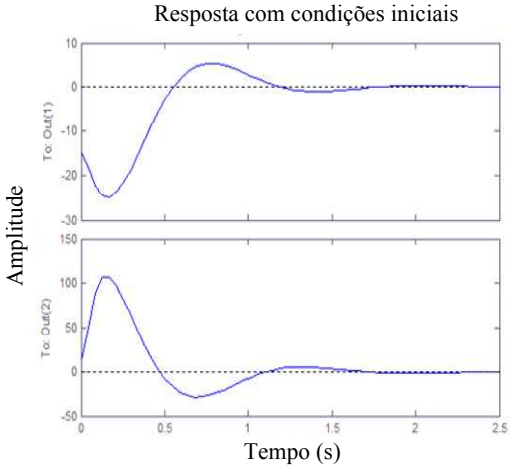
(c) Alocação de polos em [-2.5,-2,-25,-25]



(d) Resposta do sistema linearizado em malha fechada com uma entrada impulso



(e) Alocação de polos em [-2.5-5i,-2.5+5i,-25,-25]



(f) Resposta do sistema linearizado em malha fechada com uma entrada impulso

Figura 6.1- Comportamento do sistema de equações de estado linearizado do pêndulo duplo ao redor do ponto de operação $\{x_1,x_2,x_3,x_4\}=\{0,0,0,0\}$.

Os coeficientes do polinômio característico desejado são obtidos resolvendo o sistema $(s-p_n)(s - p_{n-1})...(s - p_0) = 0$, onde p_n são os pólos desejados, que no caso da Figura 6.1.c Figura 6.1.e são [-2,5 -2 -25 -25] e [-2,5-5*i* -2,5+5*i* -25 -25]. Os vetores de ganhos obtidos são K_1= [463,7550 71,5134 147,3802 10,8462] e K_2 = [701,2781 98,1706 178,4395 12,9465], respectivamente.

Pode ser observado que o deslocamento dos pólos ao longo do eixo imaginário $j\omega$ induze no sistema uma característica oscilatória, mesmo que o sistema seja de quarta ordem. Deve se lembrar que quando é analisado um sistema de segunda ordem, cuja função de transferência fica expressada em termos dos parâmetros amortecimento e freqüência natural do sistema, o fator de amortecimento indica se o sistema é sub-amortecido, criticamente amortecido ou sobre-amortecido. Portanto, valores de pólos complexos indicam o comportamento de um sistema sub-amortecido. A resposta oscilatória produzida pelos ganhos K_2 leva o sistema para a posição desejada quase na metade do tempo quando comparada com a resposta produzida com os ganhos K_1.

A idéia agora é observar o comportamento dos ganhos projetados sobre o sistema linear nas equações não lineares. Além disto, observar o efeito de alocar os pólos em posições distantes do eixo imaginário. A Tabela 6.1 mostra diferentes valores de alocação de pólos e os valores dos ganhos obtidos.

Observou-se que a escolhia dos pólos a esquerda do plano *s,* melhora a estabilidade do sistema e apresenta um sobre-passo menor. Entretanto, os ganhos de realimentação aumentam como mostrado na Tabela 6.1. Uma escolha dos pólos muito afastada do eixo complexo, gera valores de ganho muito altos, que podem por sua vez gerar valores de torque tão grandes que ao invés de estabilizar o pêndulo no ponto de operação $\{x_1,x_2,x_3,x_4\}=\{0,0,0,0\}$, podem fazer o sistema incontrolável.

A Figura 6.2 mostra a resposta do sistema com diferentes valores de ganhos, os quais são ajustados de acordo com a resposta do sistema, mesmo que um conjunto de ganhos qualquer consiga controlar o sistema. A busca dos ganhos adequados é avaliada com as especificações de desempenho do controlador, sobre-passo máximo e tempo de estabilização (também torque máximo), o que faz necessário uma saturação para não exceder o valor de torque do atuador.

Para os ganhos K = [151,7083 24,8333 50,4179 4,6687] e condições iniciais (0,1047 0 -0.2618 0), os valores de torque para estabilização variam entre (-3,2139, 3,3600) N, o qual produz uma resposta lenta em termos de tempo de estabilização (Figura 6.2.a.).

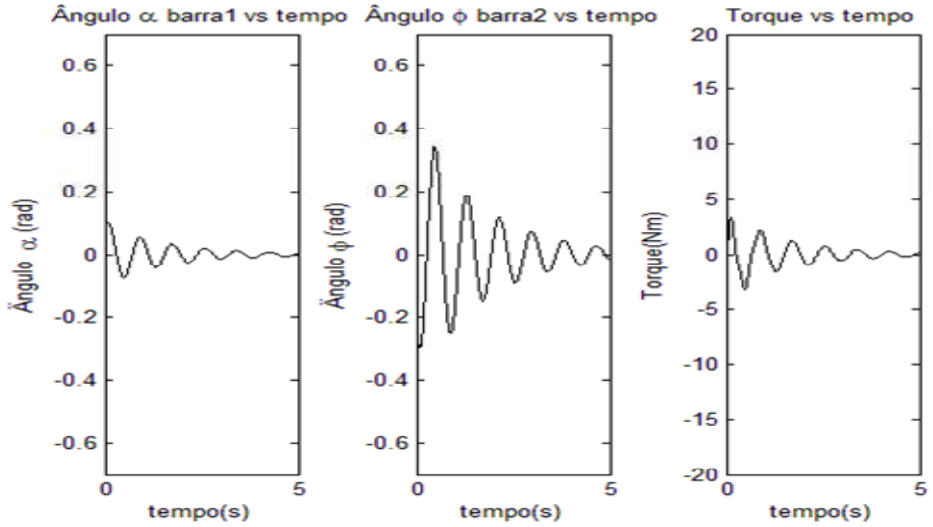
Tabela 6.1- Valores de ganhos para diferentes posições de alocação dos pólos

Posição dos pólos alocados				**Ganhos encontrados usando Ackermann**			
P_1	P_2	P_3	P_4	K_1	K_2	K_3	K_4
-3	-7	-15	-25	514,7863	88,9308	150,4602	11,7383
-2.5	-2	-25	-25	463,7550	71,5134	147,3802	10,8462
-2.5	-2	-25	-15	309,7688	49,0875	99,7174	8,1262
-2.5	-2	-25	-15	208,0414	33,5416	68,0078	5,9355
-2.5	-2	-12	-12	151,7083	24,8333	50,4179	4,6687
-2-1,46*i*	-2+1,46*i*	-12	-12	148,4378	24,0179	48,8498	4,5562
-2-7,46*i*	-2+7,46*i*	-12-7,46*i*	-12+7,46*i*	343,5875	47,7272	81,5617	6,3800
-2+3,46*i*	-2-3,46*i*	-25	-25	542,5001	76,6310	153,5181	11,1901

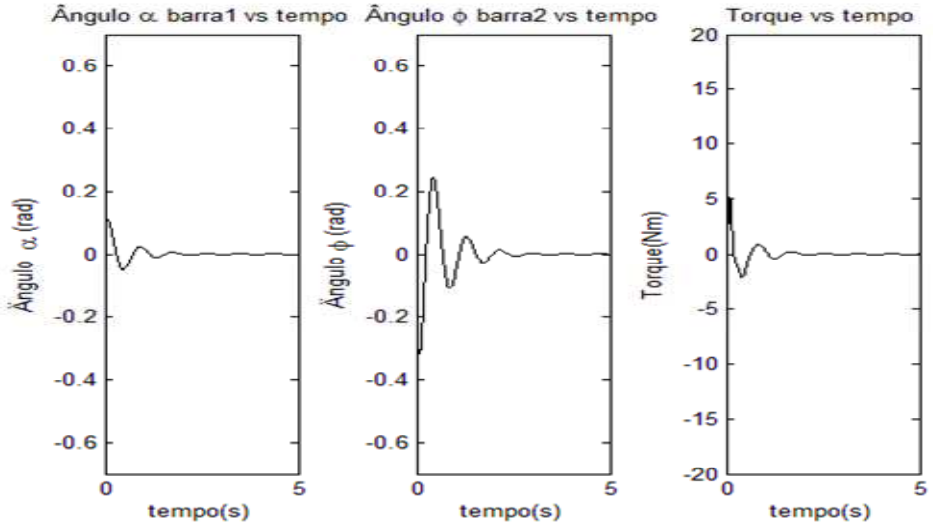
Os ganhos K=[309,7688 49,0875 99,7174 8,1262] e condições iniciais (0,1047 0 -0,2618 0) geram valores de torque entre (-6,3330, 5,0452) N, a resposta é mostrada na Figura 6.2.b. Neste caso, o tempo de estabilização melhora notavelmente. Para as mesmas condições e valores de ganhos K = [514,7863, 88,9308, 150,4602, 11,7383] os valores de torque para estabilização variam entre (-14,5179, 18,4153) N, já estes valores começam ser consideráveis, mas a resposta é melhor que no caso anterior (Figura 6.2.c).

Entre os valores de ganhos testados foram escolhidos como ótimos os valores K = [151,7083, 24,8333, 50,4179, 4,6687], com pólos alocados em [-2,5, -2, -25, -25]. O sistema apresenta um valor de estabilização menor que 2 segundos e uma variação de torque entre (-9,9803, 7,2091) N. Com estes valores é possível posicionar a *barra 1* em um ângulo máximo de 8 graus, isto depende também da posição da *barra 2* e das velocidades angulares. Com uma posição angular de 6 graus para a *barra 1* é possível alcançar uma posição angular até um valor máximo de -0,4363 graus para a *barra 2* com valores de torque variando entre 0 e 10 Nm. Este comportamento é simétrico com relação à posição angular medidas a partir da vertical.

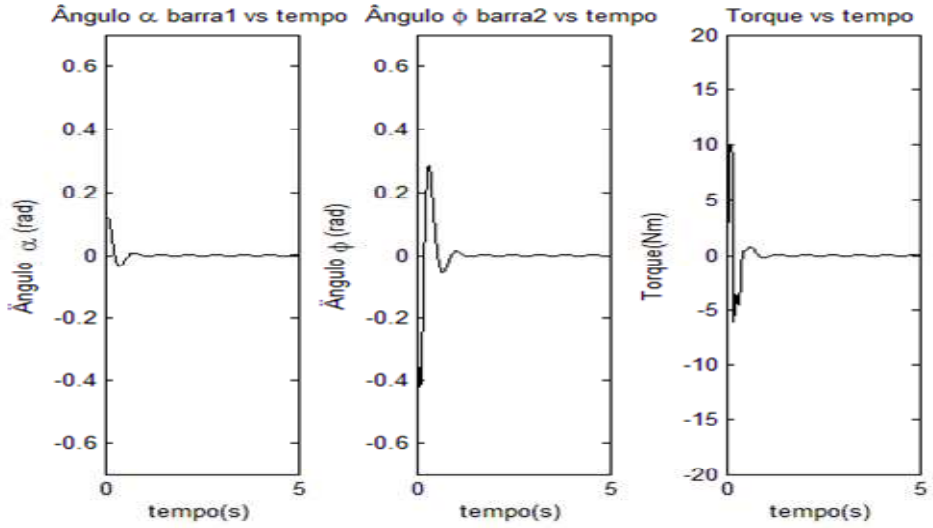
Nem todas as condições iniciais são aceitáveis, aquelas posições angulares das duas barras, ambas positivas ou negativas fazem com que o centro de gravidade do sistema fique seriamente comprometido pela ação da gravidade. Nesse caso, é possível que o torque não consiga manter o pêndulo na posição invertida.



(a) Resposta do sistema não linearizado com alocação de pólos em [-2.5,-2,-12,-12]



(b) Resposta do sistema não linearizado com alocação de pólos em [-2.5,-2,-25,-15]



(c) Resposta do sistema não linearizado com alocação de pólos em[-3,-7,-15,-25]

Figura 6.2- Resposta do sistema em malha fechada com diferentes valores de ganhos sobre o sistema não linearizado

Na Tabela 6.2 apresentam-se os resultados do controlador de realimentação de estados como os ganhos escolhidos para diferentes condições iniciais. O sistema é testado em condições normais e sob a ação de um distúrbio de força em forma de impulso com uma intensidade de 5N e uma duração de 0.05 s. O distúrbio é aplicado no sistema em 2.52 s, e para ambas situações são calculados os índices de desempenho J_1 e J_2.

A Figura 6.3 e a Figura 6.4 apresentam os resultados do sistema com condições iniciais $\{x_1,x_2,x_3,x_4\}=\{0{,}1047\ -0{,}4363\ -0{,}2618\ 0{,}2793\}$ e $\{x_1,x_2,x_3,x_4\} = \{-0{,}1047\ 0{,}2967\ -0{,}1571\ 0{,}2618\}$ respectivamente. Note que o tempo de estabilização é menor de 2 segundos, o valor do torque satura em 10 N. O segundo gráfico apresenta condições iniciais angulares negativas. Neste caso, o controlador consegue manter o sistema na posição invertida, mas para valores maiores de qualquer um das posições angulares já não é possível.

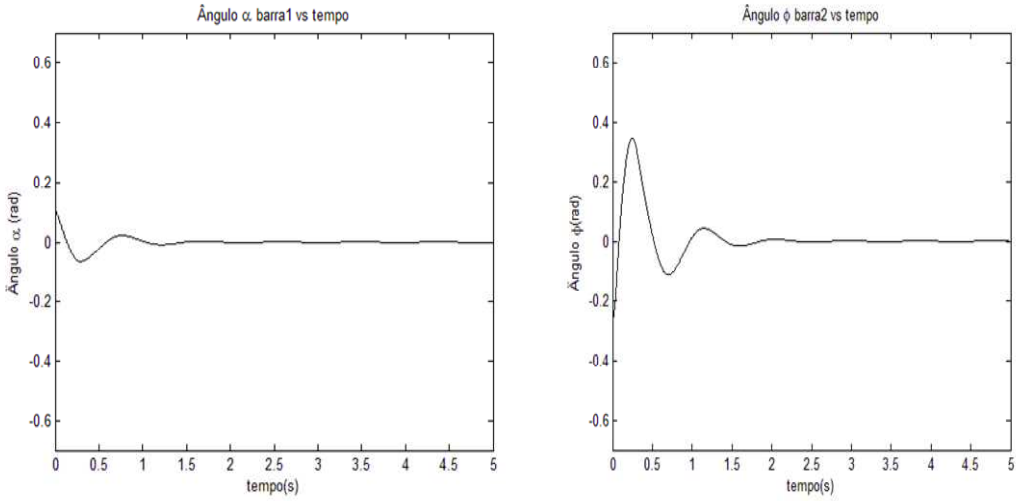
Tabela 6.2- Resposta do controlador RE usando índices de desempenho.

Condições Iniciais				**Índices sem Distúrbio**			**Índices com Distúrbio**		
α (rad)	ω_1 (rad/s)	φ (rad)	ω_2 (rad/s)	J_1		J_2	J_1		J_2
				α	φ	μ	α	φ	μ
0,1047	-0,4363	-0,2618	0,2793	0,0001	0,0016	0,0159	0,0001	0,0017	0,0527
-0,1947	-0,1571	0,2967	0,2618	0,0003	0,0042	0,0222	0,0003	0,0043	0,0590
-0,0524	-0,1745	0,2094	0,3491	0,0640 e-03	0,5607 e-03	0,1168	0,0675 e-03	0,6412 e-03	0,2189
0,0873	0,2618	-0,3840	-0,2094	0,0002	0,0014	0,2957	0,0002	0,0015	0,3984
0,1047	-0,5236	0,3491	0	0,0002	0,0022	0,4923	0,0002	0,0023	0,5951
-0,0698	0	0,5236	0	0,0001	0,0014	0,3242	0,0001	0,0015	0,4264

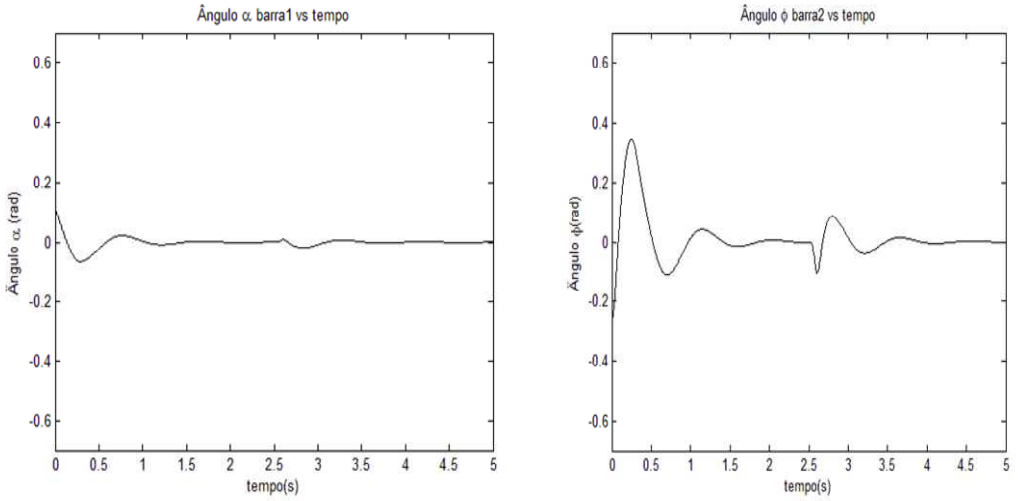
6.3 Controlador LQR

O controlador LQR pretende obter o vetor de ganho mediante a escolha adequada das matrizes Q e R da função de custo. Após condições iniciais ou uma perturbação, o controlador tenta levar as variáveis de estado o mais rápido possível para a condição de equilíbrio $\{x_1,x_2,x_3,x_4\}=\{0,0,0,0\}$ considerando a variação de energia de entrada do sistema.

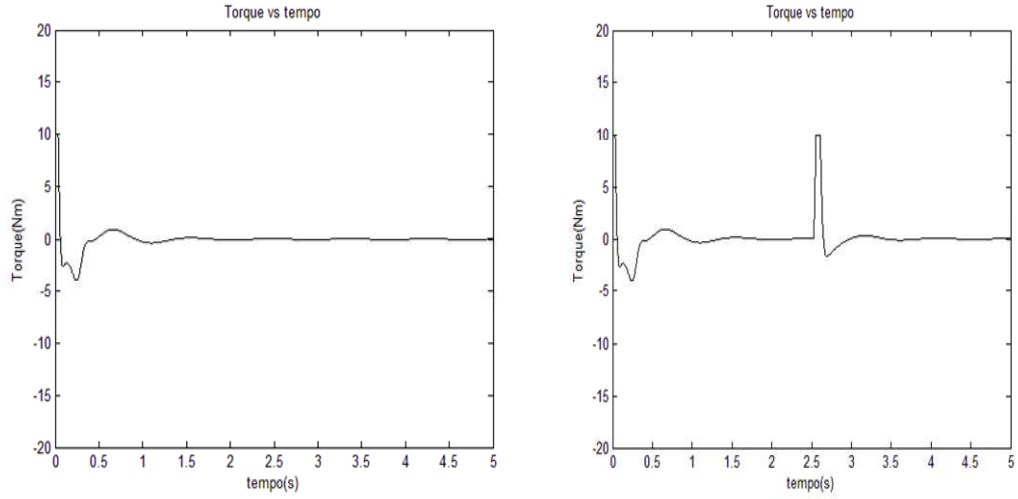
Se os valores de Q são grandes, quando comparados com os valores de R, isso significa que o controle é "barato", isto é, que a regulação é mais importante que o custo de se usar uma quantidade relativamente grande de energia na entrada. Por outro lado, se R é grande em comparação a Q, então o controle da energia é mais importante.



a. Resposta angular sem perturbação



b. Resposta angular com perturbação



c. Variações de torque sem perturbação

d. Variações de torque com perturbação

Figura 6.3- Resposta do controlador de realimentação de estados com condições iniciais $\{x_1,x_2,x_3,x_4\} = \{0{,}1047\ -0{,}4363\ -0{,}2618\ 0{,}2793\}$.

O vetor de ganhos ótimos K é obtido a partir da Equação (5.15), em que P satisfaz a equação de Riccati, apontada pela Equação (5.16). A equação de Ricatti pode ser resolvida numericamente. Alguns *softwares* como o Matlab possuem funções internas que implementam a resolução da equação de Ricatti e a obtenção dos ganhos de realimentação. A função implícita $K = lqr(A,B,Q,R)$, cujos parâmetros são as matrizes de estado A e B e as matrizes de peso Q e R, e o retorno é o vetor K, de ganhos ótimos de realimentação (Kawakami, 2005).

As matrizes Q e R devem ser obtidas por tentativa e erro, considere a matriz Q como mostrada na Equação 6.6.

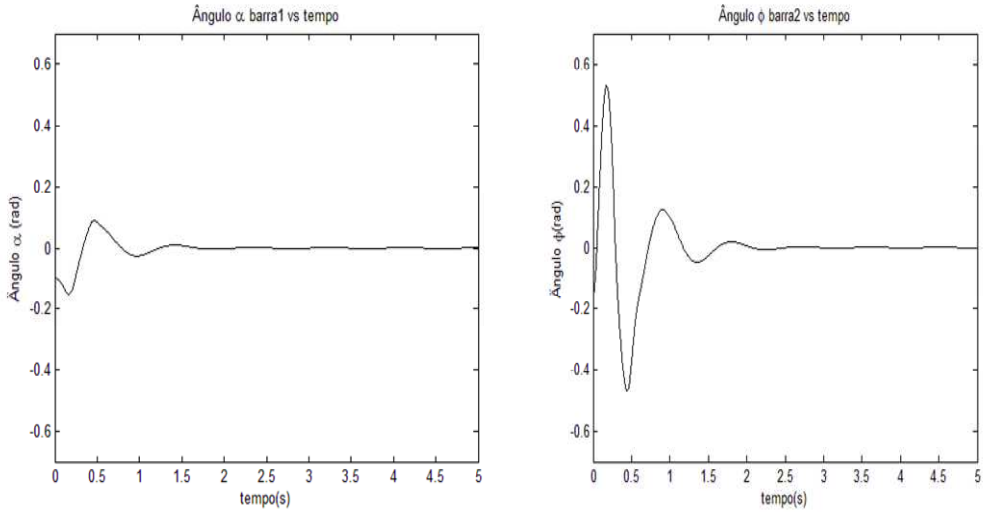
$$Q = \begin{bmatrix} x & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & y & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.6}$$

As matrizes Q e R determinam a importância relativa entre o seguimento de estados e a suscetibilidade a ruídos, os elementos das matrizes Q e R são definidos de tal forma que um elemento q_i seja maior que um elemento q_j quando se deseja dar prioridade ao estado x_i em relação ao estado x_j.

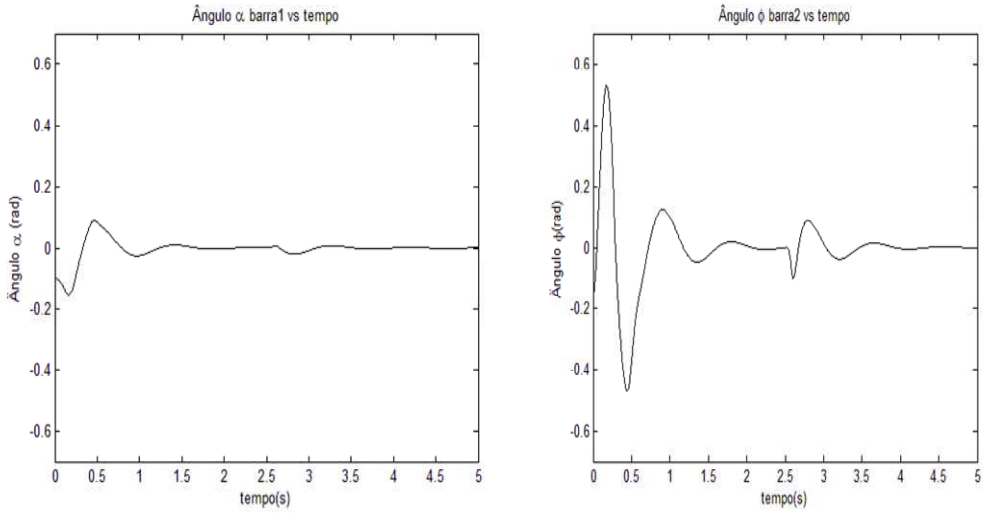
O ajuste dos parâmetros x e y leva á escolha adequada da matriz Q. Um primeiro passo é, por exemplo, deixar fixo o valor da matriz $R = 1$, e variar os parâmetros x, y observando os valores numéricos dos ganhos obtidos quando aplicado o comando *lqr* fornecido pelo toolbox de Matlab. O segundo passo consiste em testar estes valores sobre o sistema não-linear e observar a resposta do controlador com diferentes condições iniciais.

Tabela 6.3- Parâmetros de ajuste para encontrar o vetor de ganhos do LQR

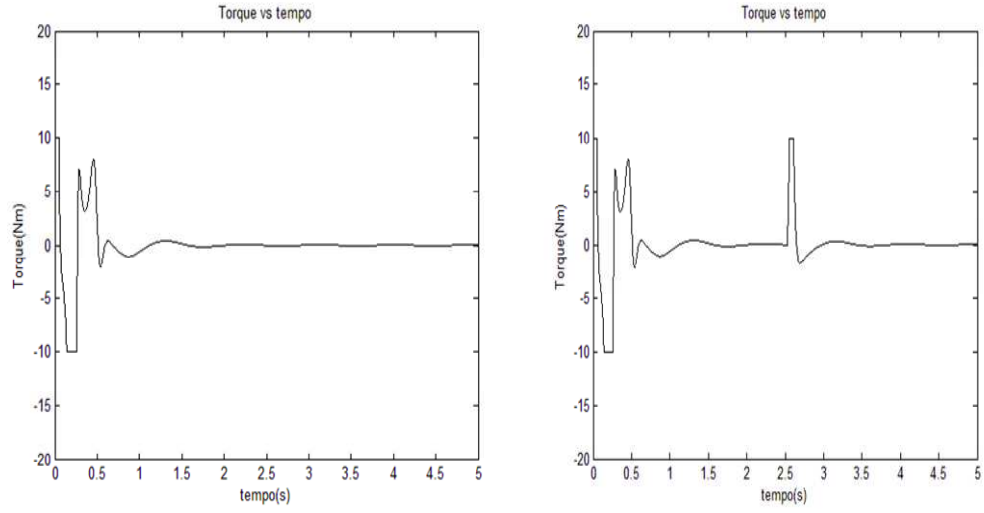
R	x	y	K_1	K_2	K_3	K_4
1	1	1	111,5279	18,7261	36,9105	3,5009
	10	10	135,82	22,89	43,37	3,97
	100	10	148,3440	24,9903	46,4127	4,1983
	10	100	224,0946	38,0528	66,4136	5,6082
	100	100	228,2931	38,7395	67,3361	5,6755
	1000	10	211,0892	35,4092	61,1448	5,2753
	10	1000	488,8528	83,5136	133,2099	10,1202
10	1	1	107,8861	18,1013	35,9339	3,4287
	10	10	111,5279	18,7261	36,9105	3,5009
	100	10	114,2596	19,1883	37,6067	3,5528
	10	100	134,3377	22,6435	43,0039	3,9467
	1000	10	133,1185	22,3688	42,3387	3,9040
	10	1000	223,6631	37,9822	66,3187	5,6013



a. Resposta angular sem perturbação



b. Resposta angular com perturbação



c. Variações de torque sem perturbação

d. Variações de torque com perturbação

Figura 6.4-. Resposta do controlador de realimentação de estados com condições iniciais $\{x_1,x_2,x_3,x_4\}=\{-0,1047\ 0,2967\ -0,1571\ 0,2618\}$.

Como mostrado na Tabela 6.3, um aumento de *R* (por exemplo, de 1 para 10) traz como conseqüência um ajuste cada vez mais fino dos ganhos em quanto *x* e *y* aumentam. A conseqüência de se aumentar o parâmetro *x* por cima de *y* gera um ajuste menor que se aumentar o valor *y* por encima de *x*.

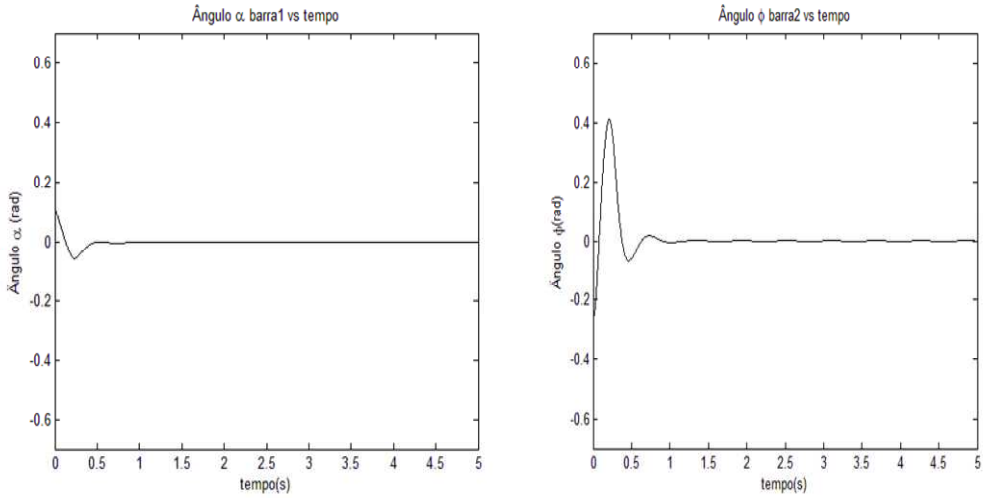
Após várias tentativas foram escolhidos os valores de ganhos *K* = [135,6042, 34,2822, 59,3933, 5,1533]. A resposta do controlador sobre as equações não-lineares pode ser observada na Figura 6.5 e na Figura 6.6. O processo de sintonização dos parâmetros mostra um comportamento similar com o processo de ajuste do controlador de realimentação de estados como mostrado na Figura 6.2.

Na Tabela 6.4 são calculados os índices de desempenho do controlador para diferentes condições iniciais.

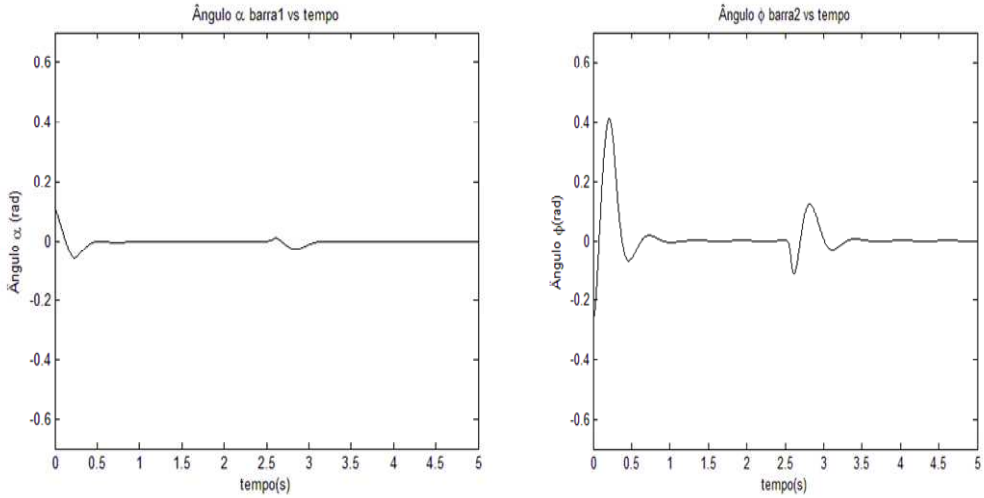
Pode ser observado que este controlador permite alcançar posições angulares para a *barra 1* e a *barra 2* de -0,3491 e 1,0472 radianos respectivamente, conforme mostrado na Tabela 8. Este comportamento é simétrico em relação ao eixo vertical.

Tabela 6.4- Resposta do controlador LQR usando índices de desempenho

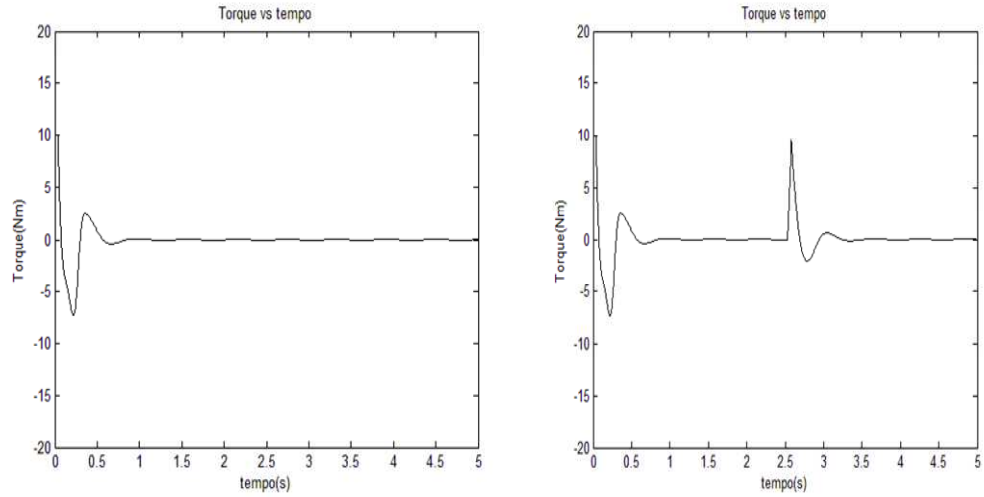
Condições Iniciais				**Índices sem Distúrbio**			**Índices com Distúrbio**		
α (rad)	ω_1 (rad/s)	φ (rad)	ω_2 (rad/s)	J_1		J_2	J_1		J_2
				α	φ	μ	α	φ	μ
0,1047	-0,4363	-0,2618	0,2793	0,0001	0,0018	0,1074	0,0001	0,0015	0,0660
-0,1047	-0,1571	0,2967	0,2618	0,1478 e -03	0,8472 e -03	1,6734 e -004	0,0002	0,0010	0,0343
0,1571	0	-0,1745	0	0,0007	0,0090	0,0451	0,0007	0,0092	0,0793
-0,1571	0,2618	0,4363	-0,2618	0,0238 e -03	0,1014 e -03	0,9347	0,0001	0,0011	0,0580
0,1396	-0,1745	-0,5236	0,3491	0,0001	0,0014	0,0440	0,0001	0,0016	0,0782
0,1396	-0,4363	0,1745	0,2443	0,0004	0,0017	0,0344	0,0004	0,0019	0,0685
0	-0,4363	0,5236	0,5236	0,0001	0,0016	0,0543	0,0543	0,0018	0,0885
-0,3491	0	1,0472	0	0,0012	0,0185	0,0423	0,0007	0,0108	0,0583



a. Resposta angular sem perturbação



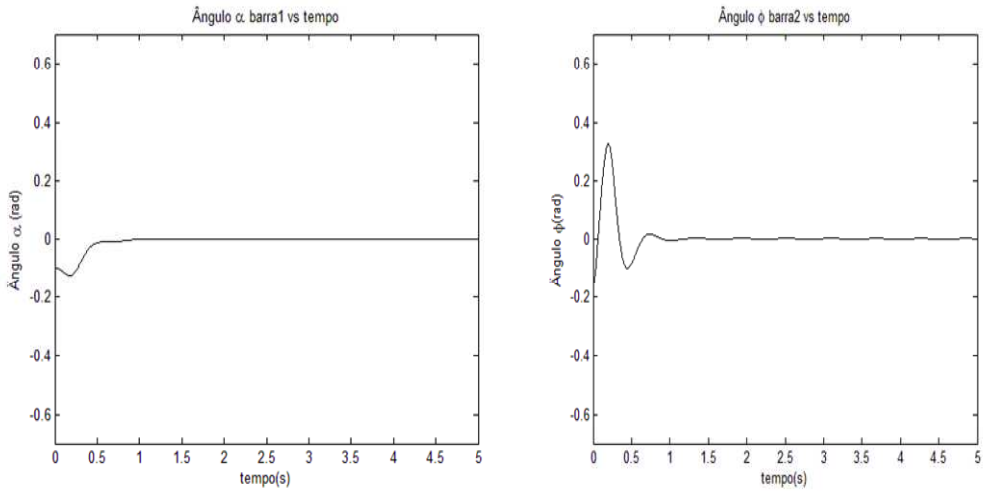
b. Resposta angular com perturbação



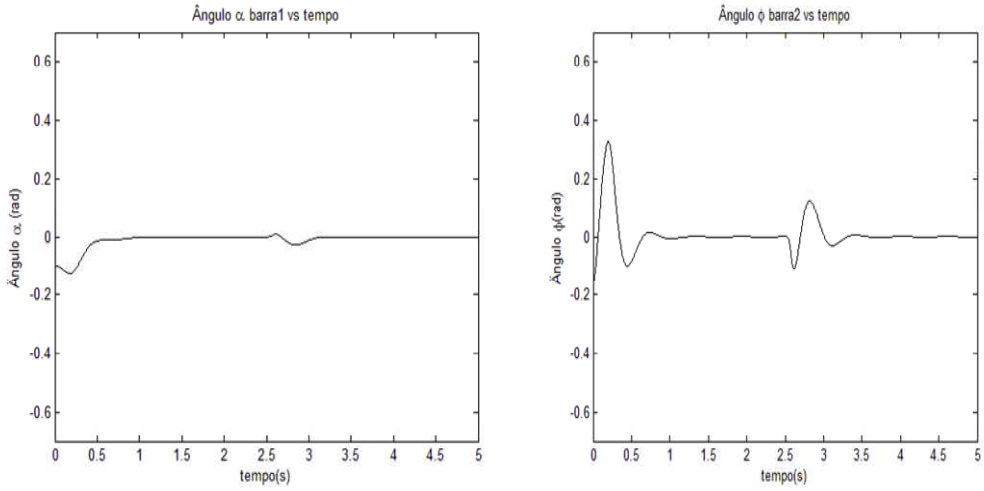
c. Variações de torque sem perturbação

d. Variações de torque com perturbação

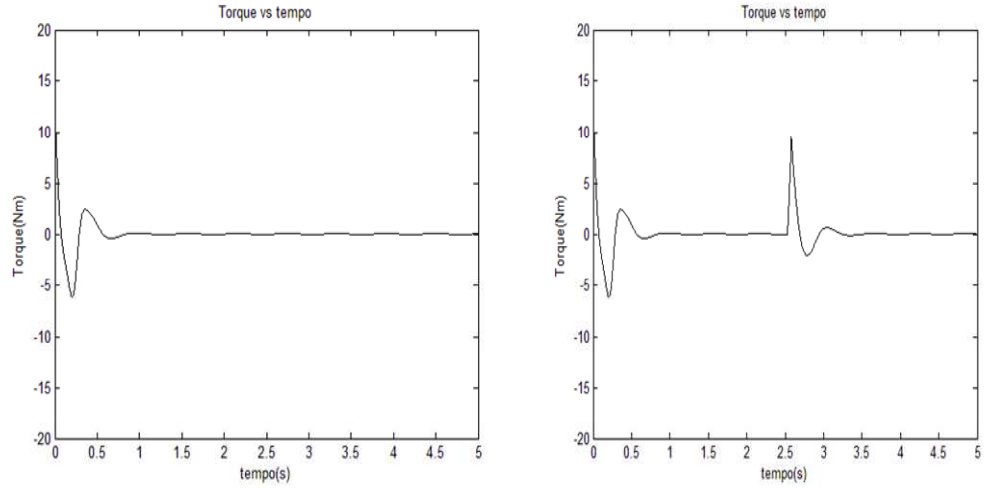
Figura 6.5-. Resposta do regulador LQR com condições iniciais $\{x_1,x_2,x_3,x_4\}$ = {0,1047 -0,4363 -0,2618 0,2793}



a. Resposta angular sem perturbação



b. Resposta angular com perturbação



c. Variações de torque sem perturbação

d. Variações de torque com perturbação

Figura 6.6-. Resposta do regulador LQR com condições iniciais $\{x_1,x_2,x_3,x_4\}$={-0,1047 0,2967 -0,1571 0,2618}.

O tempo de estabilização para condições iniciais é menor que 1 seg. para as duas barras. A resposta á perturbação é rápida e o tempo de estabilização é melhor quando comparado com o controlador RE.

Algumas considerações podem ser feitas entre estes dois tipos de controle. Os dois métodos de controle moderno são baseados na escolhia empírica dos parâmetros. O método LQR é mais adequado porque ele já envolve uma função de custo baseada na equação de Riccati. Analiticamente, é observado que para os valores de ganhos escolhidos a resposta do LQR é mais rápida, os valores de torque, ou seja, da energia de controle são fornecidos de uma maneira mais suave.

Mesmo que teoricamente a equação de Ackermann permita encontrar os ganhos correspondentes a uma alocação de pólos qualquer, existem posições que levam uma geração de ganhos com valores para o sinal de controle (torque) impossíveis de se implementarem. A mesma coisa acontece com o LQR, para isto é necessário introduzir uma saturação do controlador.

Em quanto aos índices de desempenho, para a primeira condição inicial gerada, pode-se observar que o esforço de controle é menor para o controlador RE, quanto ao índice com distúrbio quanto ao índice sem distúrbio. A *barra 1* é posicionada para ambos controladores no mesmo tempo, a *barra 2* demora um pouco a mais com o LQR que com o RE sem distúrbio. Entretanto, com distúrbio presente, o LQR posiciona mais rápido a *barra 2* que o RE. Para a segunda condição inicial os índices de desempenho J_1, J_2 são menores para o LQR que para o RE com distúrbio e sem distúrbio (Castro *et al.*, 2009b).

Em termos gerais, os dois controladores resultam satisfatórios quando aplicados sobre o sistema simulado não-linear. A faixa angular de controle é um pouco maior para o controlador LQR. O controlador RE permitiu uma faixa de operação entre [-0,1047 0,1047] radianos para a *barra 1*, e uma faixa de operação entre [-0,4363 0,4363] radianos para a *barra 2*. O controle LQR permitiu uma faixa de operação entre [-0,1571 0,1571] radianos para a *barra 1*, e uma faixa de operação entre [-0,5236 0,5236] para a *barra 2*.

Esta faixa de operação está limitada pelos valores das condições iniciais, como mostrado na Tabela 6.4. O LQR permite uma faixa de operação maior quando as velocidades angulares são consideradas iguais a zero.

6.4 CONTROLADOR DIFUSO

Inicialmente é realizado um estudo das condições e do comportamento da planta simulada no ambiente computacional Matlab: sistema de coordenadas, direção dos ângulos e das velocidades angulares de rotação das barras, quando aplicados torques positivos e negativos.

Como observado na Figura 6.7, o primeiro quadrante descreve posições angulares positivas para as duas barras, e o segundo quadrante descreve posições angulares negativas. Além disto, no sistema simulado, um torque positivo produz na *barra 2* um giro em sentido horário. Portanto, o torque refletido sobre a *barra 1* produz um giro em sentido anti-horário.

Se o movimento angular das barras é feito no sentido anti-horário a velocidade angular diminui, se o movimento angular é no sentido horário a velocidade angular aumenta. Pelo qual um valor negativo ou positivo da velocidade angular em um instante determinado não define o sentido de giro. Neste caso, precisa-se do valor anterior para saber se está aumentando ou diminuindo.

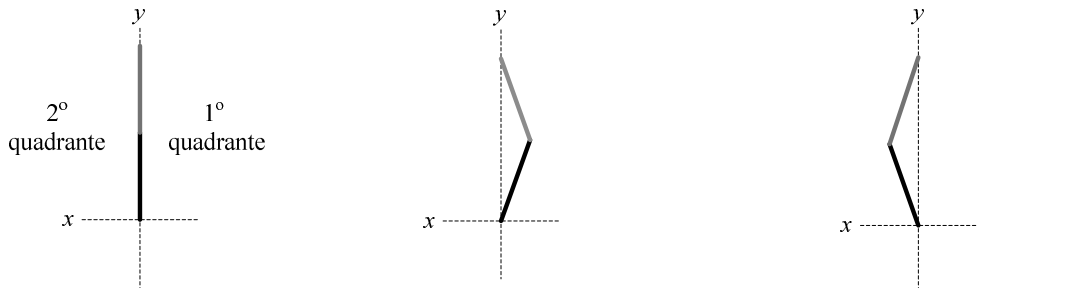


Figura 6.7- Descrição da simulação da planta.

No projeto do controlador como tal, foram adotadas os seguintes critérios: (a) descompor o sistema em subsistemas (mesmo que não sendo utilizada a geração de múltiplos estados),

(b) ajustar a saída dos subsistemas com ganhos constantes e (c) utilizar uma operação de subtração para combinar os sinais de saída dos dois subsistemas e conseguir assim um sinal de controle satisfatório para o controle do pêndulo (Castro *et al.*, 2009a).

A decomposição do controlador foi dividida em dois subsistemas, onde o primeiro subsistema está composto pelas variáveis de entrada da primeira barra (o ângulo α e velocidade angular ω_1) e a saída é multiplicada por um ganho t_1. O segundo subsistema esta composto pelas variáveis de entrada da segunda barra (o ângulo φ e velocidade angular ω_2) e a saída é multiplicada por um ganho t_2, como mostrado na Figura 6.8.

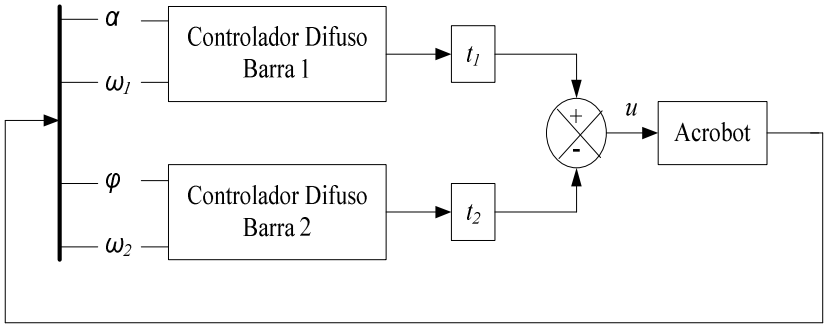


Figura 6.8-. Arquitetura do controlador difuso.

Foi utilizado o controlador difuso tipo Takagi-Sugeno com duas entradas e uma saída para as variáveis relacionadas com os parâmetros próprios de cada barra. A entrada dos controladores de cada subsistema é associada mediante três funções de pertinência triangulares (*tmrf*), cujos conjuntos difusos são denotados pelas letras *N*(Negativo), *M*(Medio) e *P*(Possitivo). Estas funções de pertinência geram uma base de conhecimento conformada por 9 regras difusas da forma "***Se*** *(*α ***é*** *...)* ***e*** *(*ω_1 ***é****...)* ***então*** *Torque* ***é*** *..."* para cada subsistema como mostrado na Figura 6.9.

O número de regras difusas pode ser estimado mediante a relação N^M, onde *M* é o número de variáveis de entrada com uma saída e *N* é o número de funções de pertinência para cada variável (Zong-Mu e Kuei-Hsiang, 2004). É observado que o número de regras difusas pode incrementar rapidamente com o número de entradas, com o número de saídas e com o número de funções de pertinência escolhidas. Para um sistema multivariável de quatro entradas, uma saída e três funções de pertinência associadas para cada entrada, precisa-se de 81 regras difusas. Isto significa em um aumento exponencial do número de regras, aumentando assim a dificuldade da sua implementação. Isto é devido a que conjunto com

um número elevado de regras pode apresentar conflitos entre elas, redundâncias, etc., que terminam degradando o desempenho do controlador.

Uma divisão do problema em subsistemas diminui a quantidade de regras para um total de 18, 9 regras para cada subsistema (Zong-Mu e Kuei-Hsiang, 2004). Os valores dos ganhos na saída dos subsistemas foram encontrados experimentalmente. Os mesmos apresentam um comportamento satisfatório para $t_1 = 1{,}2$ e $t_2 = 0{,}3$.

1. Se (α é N) e (w é N) então (torque é dez)
2. Se (α é N) e (w é M) então (torque é oito)
3. Se (α é N) e (w é P) então (torque é seis)
4. Se (α é M) e (w é N) então (torque é quatro)
5. Se (α é M) e (w é M) então (torque é zero)
6. Se (α é M) e (w é P) então (torque é nquatro)
7. Se (α é P) e (w é N) então (torque é nseis)
8. Se (α é P) e (w é M) então (torque é noito)
9. Se (α é P) e (w é P) então (torque é ndez)

a. Base de regras Controlador barra 1

1. Se (α é N) e (w é N) então (torque é mdez)
2. Se (α é N) e (w é M) então (torque é seis)
3. Se (α é N) e (w é P) então (torque é quatro)
4. Se (α é M) e (w é N) então (torque é moito)
5. Se (α é M) e (w é M) então (torque é zero)
6. Se (α é M) e (w é P) então (torque é um)
7. Se (α é P) e (w é N) então (torque é mquatro)
8. Se (α é P) e (w é M) então (torque é mseis)
9. Se (α é P) e (w é P) então (torque é oito)

b. Base de regras Controlador barra 2

Figura 6.9- Base de regras geradas para os subsistemas do controlador

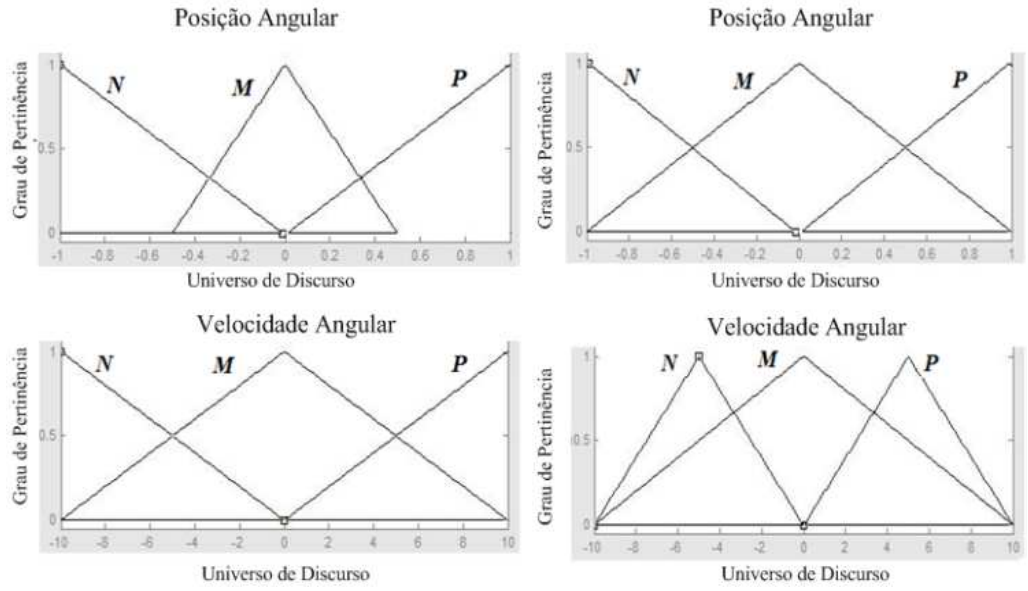


Figura 6.10- Funções de pertinência para cada controlador difuso

Os valores de saída do controlador são associados com 9 conjuntos *singletons* (*mdez, moito, mseis, mquatro, zero, quatro, seis, oito, dez*) que são valores constantes de controle de saída, usados para a interpolação dos dados gerados pelo sistema de inferência.

Posteriormente, é aplicado o método de centro de área para a desnebulização, encontrando um valor real que vai modificar o estado da planta.

A metodologia proposta para se obterem os conjuntos de saída (*singletons*) do controlador consistiu em testar diferentes valores de torque para diferentes condições iniciais da planta (ver Figura 6.11). Neste caso, foram escolhidos aqueles valores que conseguiram levar o pêndulo para a posição de equilíbrio com a menor energia possível (critério de mínima energia na posição invertida). A saída do torque depende da interpolação desses valores em quanto o grau de pertinência das entradas aos conjuntos difusos.

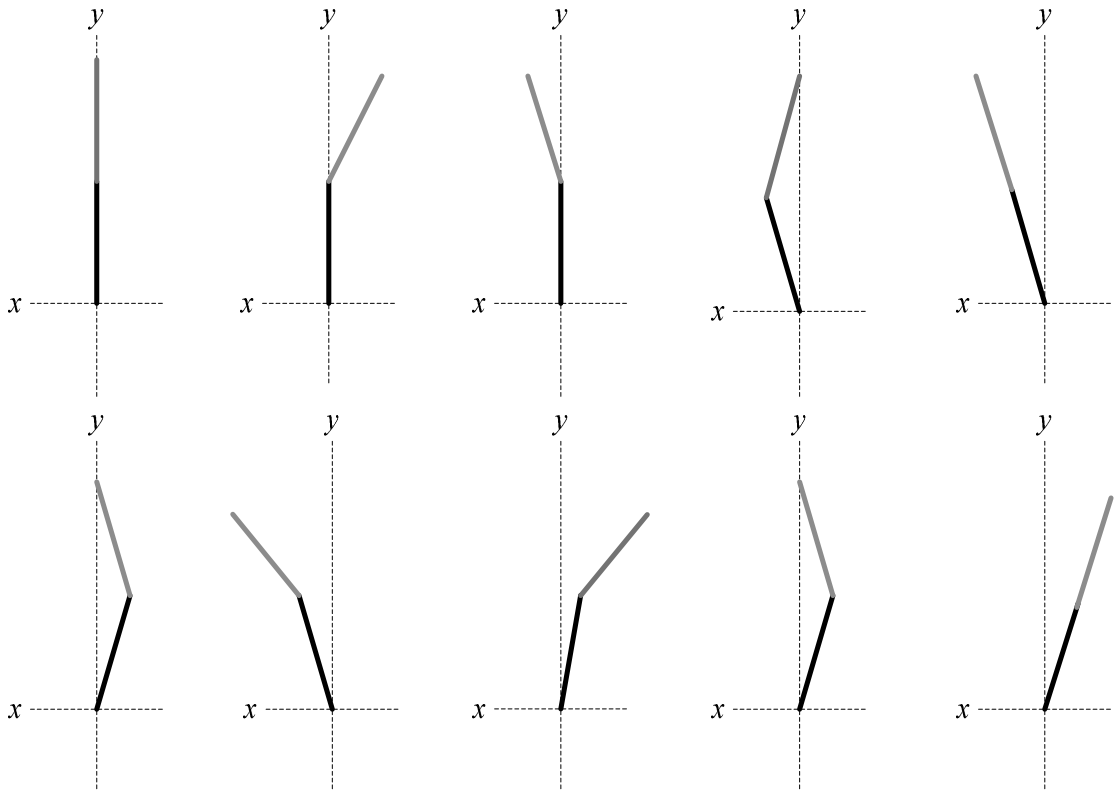


Figura 6.11- Algumas posições de estudo para escolha dos parâmetros de saída do controlador (*singletons*).

Na Tabela 6.5 apresentam-se os resultados da resposta do controlador com diferentes condições iniciais. Similarmente, também são medidos os índices de desempenho e calculados em condições normais e para uma perturbação de 5 N em forma de impulso, com uma duração de 0.05 s. Nas Figura 6.12 e 6.13 a resposta do sistema para condições iniciais $\{x_1,x_2,x_3,x_4\}=\{$ 0.1047 -0,4363 -0,2618 0,2793$\}$ e $\{x_1,x_2,x_3,x_4\}=\{$-0,1047 -0,1571 0,2967 0,4363$\}$ são apresentadas.

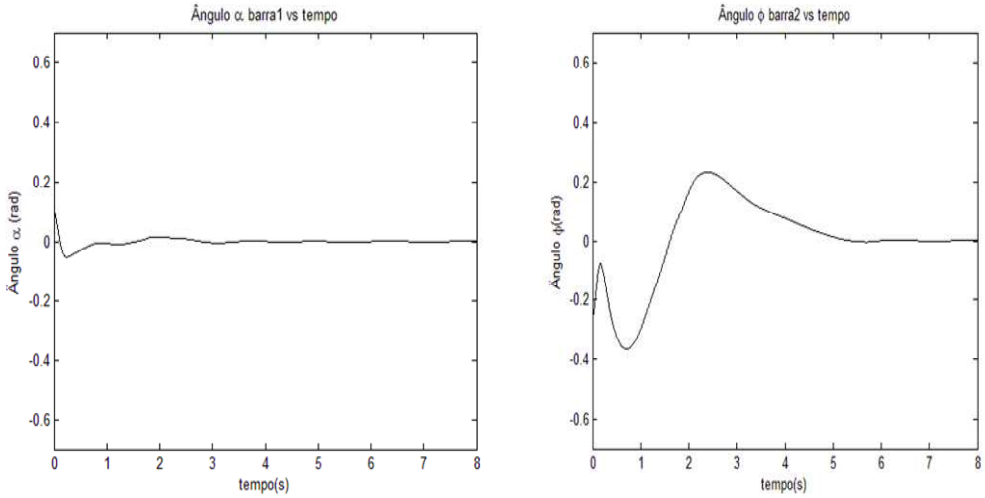
Tabela 6.5- Resposta do Controlador Difuso usando índices de desempenho

Condições Iniciais				**Índices sem Distúrbio**			**Índices com Distúrbio**		
α (rad)	ω_1 (rad/s)	φ (rad)	ω_2 (rad/s)	J_1		J_2	J_1		J_2
				α	φ	μ	α	φ	μ
0,1047	-0,4363	-0,2618	0,2793	0,0001	0,0150	1,4875 e-005	0,0002	0,0238	0,0013
-0,1047	-0,1571	0,2967	0,2618	0,0001	0,0074	1,0939 e-004	0,0006	0,0798	0,0214
0,2618	0,1745	-0,2618	-0,3840	0,0006	0,0050	0,0025	0,0006	0,0166	0,0046
-0,3491	-0,5236	0,2967	0,1396	0,0010	0,0312	0,0060	0,0011	0,0419	0,0080
0,2618	0,2269	0,4712	-0,4538	0,0007	0,0370	0,0049	0,0008	0,0489	0,0070
-0,2094	-0,3142	-0,4189	0,2793	0,0005	0,0630	0,0027	0,0005	0,0737	0,0045
-0,2618	0	0,2618	0	0,0005	0,0527	0,0022	0,0006	0,0633	0,0040
0,4363	-0,2618	0,2443	-0,3491	0,0019	0,0515	0,0136	0,0019	0,0621	0,0155

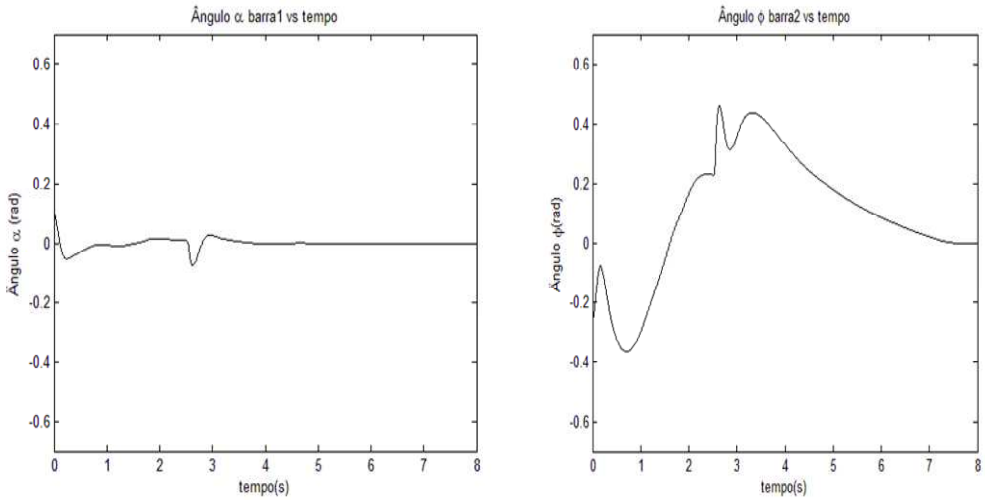
No primeiro casso, é observado que o tempo de estabilização é menor que 1 segundo para a primeira barra, a segunda barra demora quase 4 segundos em alcançar a posição desejada, sem perturbação. O sistema responde satisfatoriamente para as situações onde são apresentadas perturbações. As variações de torque são bem menores que os métodos apresentados anteriormente, mas o tempo de estabilização é maior.

6.5 CONTROLADOR NEURAL

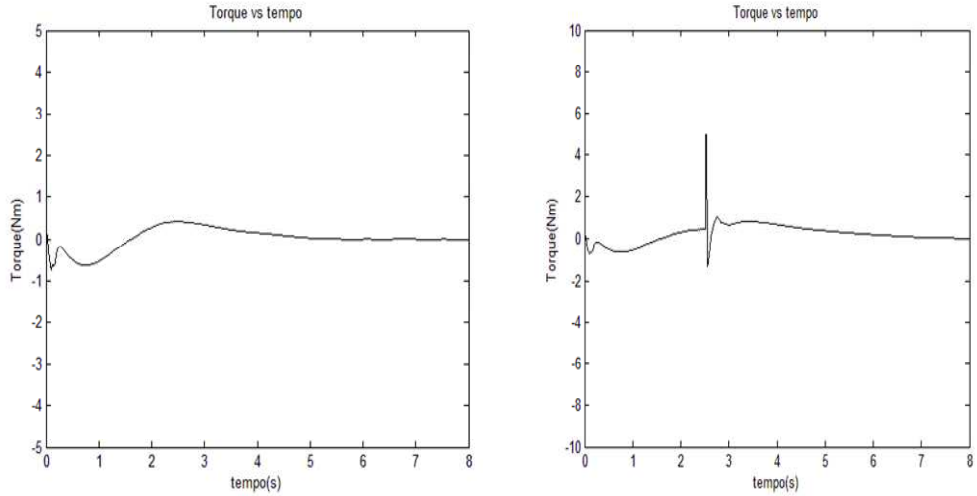
Foi utilizado para o controlador neural o sistema de aprendizado supervisionado baseado no algoritmo de retropropagação do erro. É utilizado o toolbox de Matlab para treinamento de redes neurais, a arquitetura perceptron multicamada utilizada é apresentada na Figura 6.14. Foram testadas diferentes configurações para a rede neural: 5, 10, 15, 20, 25, 30, 35, 40, 45 e 50 neurônios na camada oculta. Neste caso, a melhor resposta foi encontrada para a camada oculta com 50 neurônios.



a. Resposta angular sem perturbação



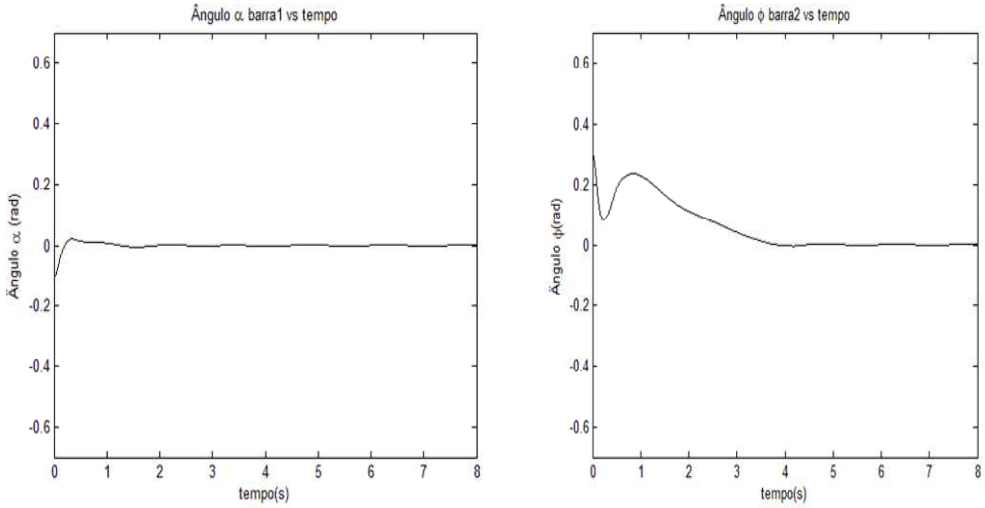
b. Resposta angular com perturbação



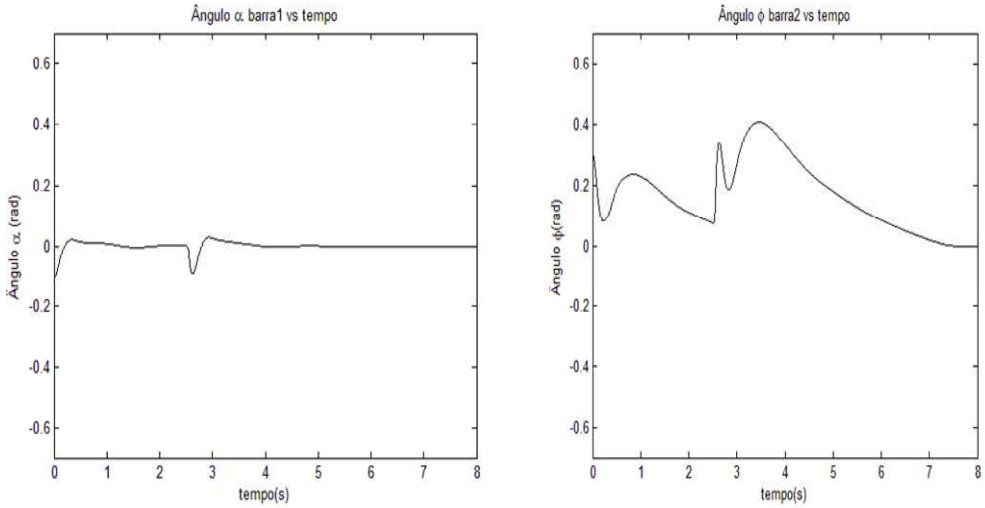
c. Variações de torque sem perturbação

d. Variações de torque com perturbação

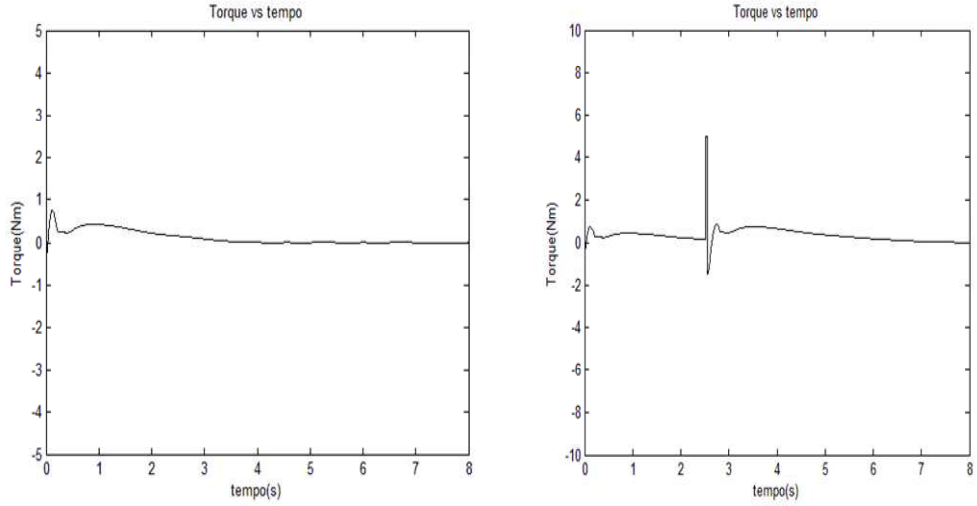
Figura 6.12- Resposta do controlador difuso com condições iniciais $\{x_1,x_2,x_3,x_4\}$={ 0,1047 -0,4363 -0,2618 0,2793}.



a. Resposta angular sem perturbação



b. Resposta angular com perturbação



c. Variações de torque sem perturbação

d. Variações de torque com perturbação

Figura 6.13-. Resposta do controlador difuso com condições iniciais $\{x_1,x_2,x_3,x_4\}=\{-0,1047\ -0,1571\ 0,2967\ 0,2618\}$.

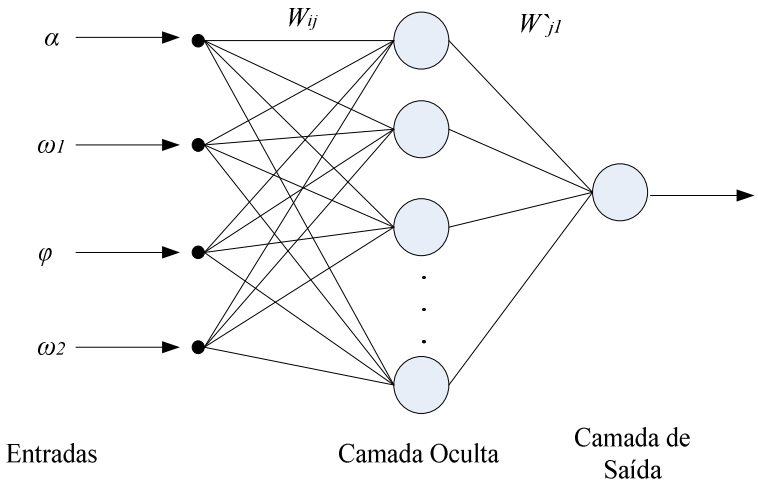


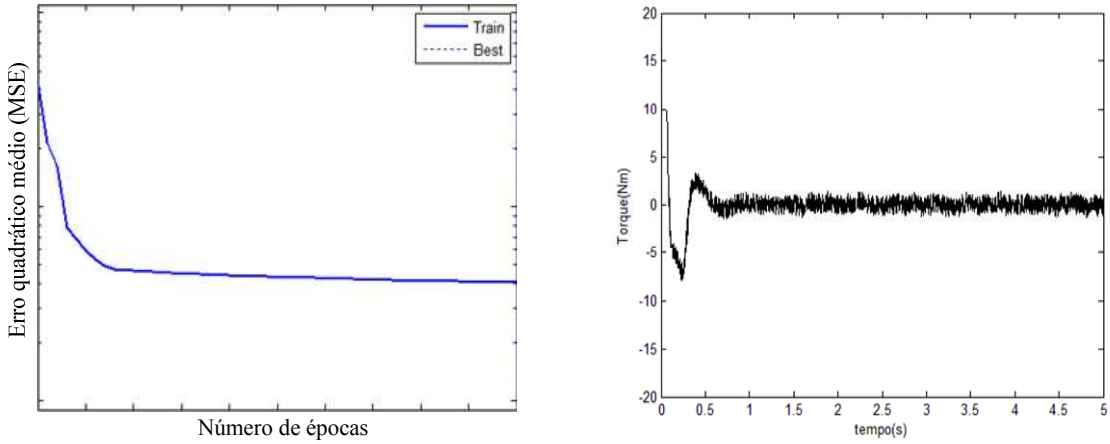
Figura 6.14-. Arquitetura perceptron multicamada para balanço do pêndulo duplo interatuado

Existe a necessidade de se utilizar um supervisor que forneça para a rede os exemplos necessários para que ela aprenda os comportamentos desejados, e assim possa reproduzir esses comportamentos em situações não previstas na sua função como controlador. Devido à ausência de uma planta real foi utilizado o controlador LQR como supervisor. Fornecidas umas condições iniciais, os valores do torque gerados pelo LQR são alterados com um valor fornecido por uma função aleatória a cada ciclo de integração do algoritmo de simulação. Isto permite que para certas situações a rede aprenda de valores diferentes aos dados pelo controlador diretamente (ver Figura 6.15.b.) (Castro *et al.*, 2009).

A aquisição de dados (posições e velocidades angulares) fornecida pelo supervisor LQR é armazenada em uma matriz. Os valores de torque são armazenados em um vetor separado (estes dados são necessários para o treinamento da rede). Para tanto, foram coletados 4000 dados para um tempo de simulação de 10 segundos e foram utilizadas 50 épocas de treinamento. As funções de ativação dos neurônios das camadas ocultas são do tipo *tansig* (não-linear). Adicionalmente, o neurônio de saída tem a função de ativação tipo *purelin* (linear) disponível em Matlab.

A Figura 6.15.a. apresenta o gráfico de treinamento da rede para 50 épocas. Pode-se observar como o erro quadrático médio (MSE) diminui na medida em que o número de

épocas aumenta. Mesmo que em 50 épocas o treinamento convirja, é observado que um número de épocas maior poderia trazer uma melhor aproximação, mas foram testadas também 100 e 150 épocas de treinamento e o desempenho do controlador não apresentou melhoria significativa quando comparado com as 50 épocas. Os valores de torque limites para o atuador saturam em 10 Nm. Os valores dos pesos inicialmente da rede são gerados aleatoriamente.



a. Treinamento da rede neural

b. Geração do conjunto de dados para treinamento da rede neural

Figura 6.15- A convergência do treinamento da rede neural mediante o toolbox de Matlab.

O sistema responde mais rápido se comparado com o controlador difuso. A *barra 2* é posicionada na metade do tempo que o controlador difuso. Portanto, o sobre-passo do sinal e o tempo de estabilização são menores. A resposta do controlador neural é bastante mais tolerável às perturbações que o controlador difuso. Entretanto, a variação de torque do controlador difuso é menor durante a perturbação (ver Figura 6.12).

Diferentes condições iniciais são aplicadas ao sistema e observadas diferentes respostas do controlador neural como mostrado na Tabela 6.6. Um distúrbio de 5 N é aplicado e os índices de desempenho J_1 e J_2 são verificados para este caso. A Figura 6.16 e a Figura 6.17 apresentam a resposta do controlador a condições iniciais $\{x_1,x_2,x_3,x_4\}$ = {0,1047 -0,4363 -0,2618 0,2793} e $\{x_1,x_2,x_3,x_4\}$ = {-0,2094 0,3491 0 0,1745} respectivamente. Pode ser observado que o tempo de estabilização é menor que 1 segundo, e o sistema converge mais rapidamente quando aplicada a perturbação.

Tabela 6.6-. Resposta do controlador neural usando índices de desempenho

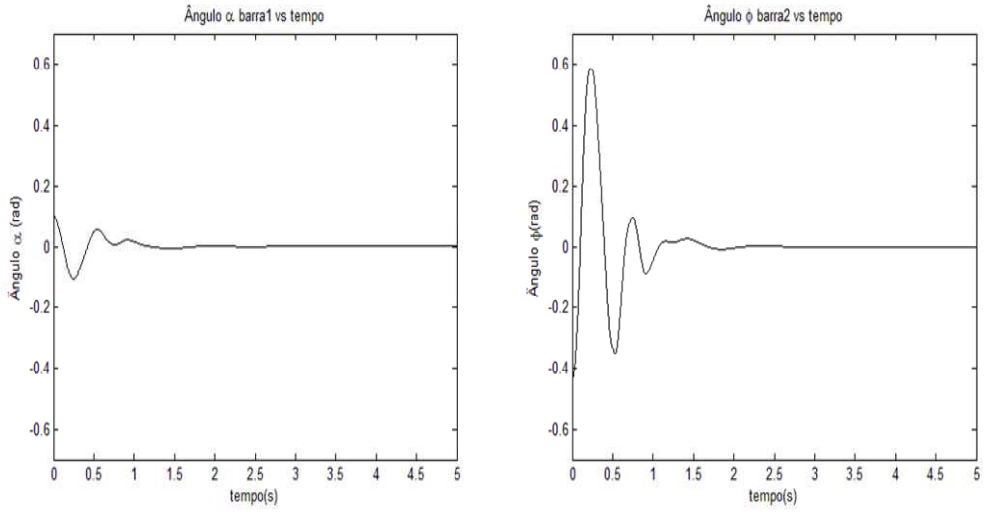
Condições Iniciais				**Índices sem Distúrbio**			**Índices com Distúrbio**		
α (rad)	ω_1 (rad/s)	φ (rad)	ω_2 (rad/s)	J_1		J_2	J_1		J_2
				α	φ	μ	α	φ	μ
0,1047	-0,4363	-0,2618	0,2793	0,0010	0,0305	0,4305	0,0010	0,0309	0,5120
-0,1047	-0,1571	0,2967	0,2618	0,0003	0,0028	0,0388	0,0003	0,0031	0,1158
0,2094	0,1920	-0,2618	-0,3491	0,0072	0,0853	0,5317	0,0072	0,0853	0,5734
-0,2094	0	0,3491	0,1745	0,0012	0,0034	0,0259	0,0012	0,0037	0,1028
-0,2618	-0,2618	0,2967	0	0,0150	0,0273	0,0618	0,0150	0,0273	0,0618
0,1396	0,1745	-0,4363	-0,3142	0,0005	0,0018	0,0463	0,0005	0,0021	0,1237
-0,3316	0	1,1345	0	0,6801	4,2729	0,358	0,0013	0,0167	0,0873

6.6 CONTROLADOR NEURO-DIFUSO

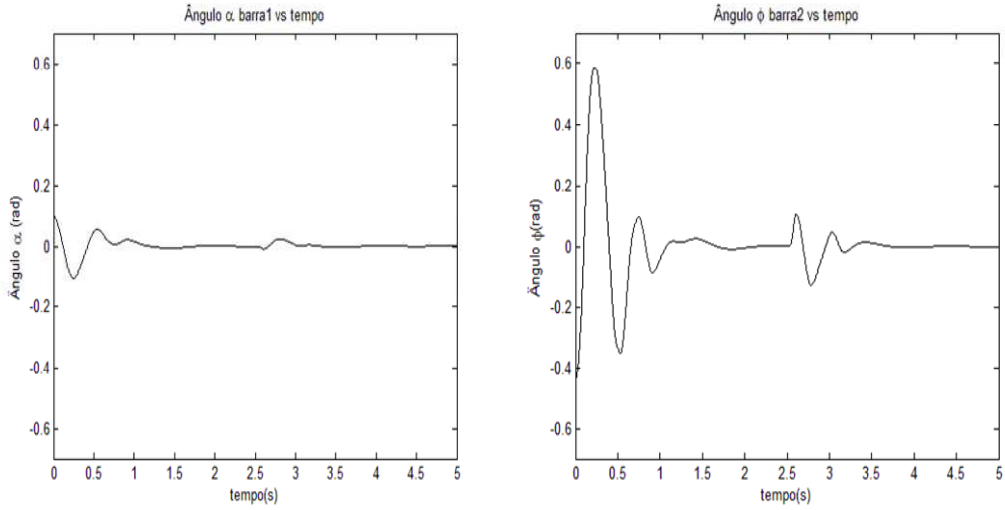
Com o intuito de otimizar o controlador difuso mediante a aplicação e treinamento das redes neurais, foi utilizado o sistema neuro-difuso ANFIS (*Adaptive Neuro-Fuzzy Inference System*). O sistema ANFIS pode otimizar as funções de pertinência de um sistema já criado, ou criar um sistema básico e ajustar os parâmetros do controlador mediante um conjunto de dados. Para este caso, é utilizada a mesma metodologia de treinamento da rede neural anterior, usando como supervisor o controlador LQR.

Foi fornecido um conjunto de 8000 dados de treinamento, assim como um número de 100 épocas de treinamento. Para este caso, os dados devem ser organizados em forma matricial. Foi criada uma matriz de cinco colunas e 8000 filas. As primeiras quatro colunas devem conter as informações relacionadas com as variáveis $\{\alpha, \omega_1, \varphi, \omega_2\}$ de entrada, e a quinta coluna deve conter as informações relacionadas com a variável de controle τ.

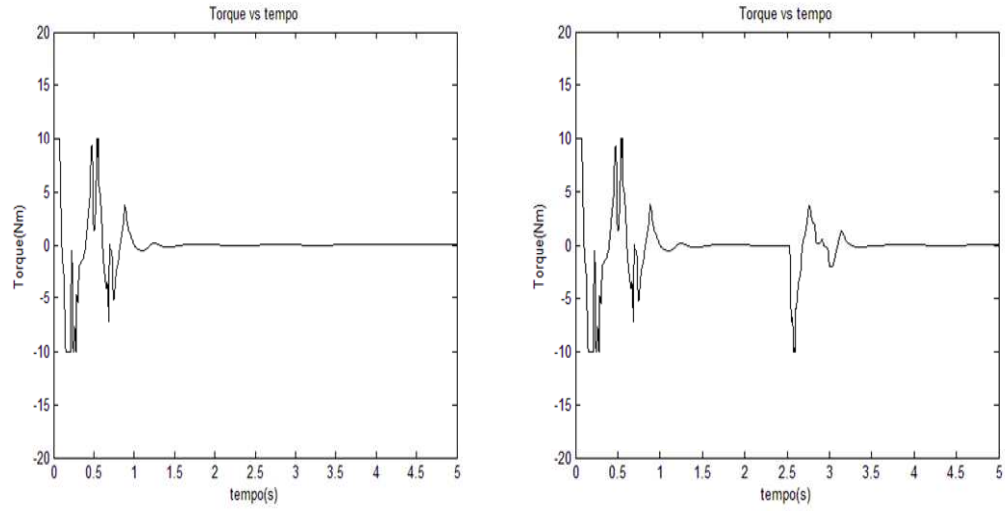
Quando a matriz de dados é carregada no ANFIS, o sistema reconhece que as primeiras colunas são entradas e a última coluna é a saída. Os passos para criar o sistema ANFIS são os seguintes: (a) primeiro carregam-se os dados de treinamento (*load data*), (b) define-se o número de funções de pertinência e o tipo, elas vão ser associadas com as entradas que o sistema já identificou (*Generate FIS*), (c) Escolhe-se o algoritmo de treinamento, entre o algoritmo híbrido (combinação entre mínimos quadrados em batelada e retropropagação do erro) ou o algoritmo retropropagação do erro. A tolerância do erro que deve ser atingida e o número de épocas de treinamento (*train now*). A interfase de usuário é mostrada na Figura 6.18.



a. Resposta angular sem perturbação



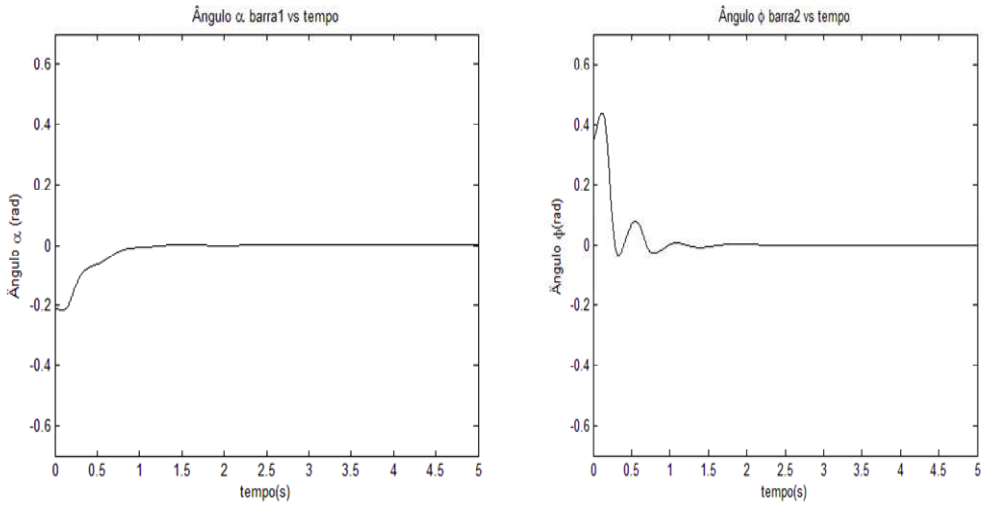
b. Resposta angular com perturbação



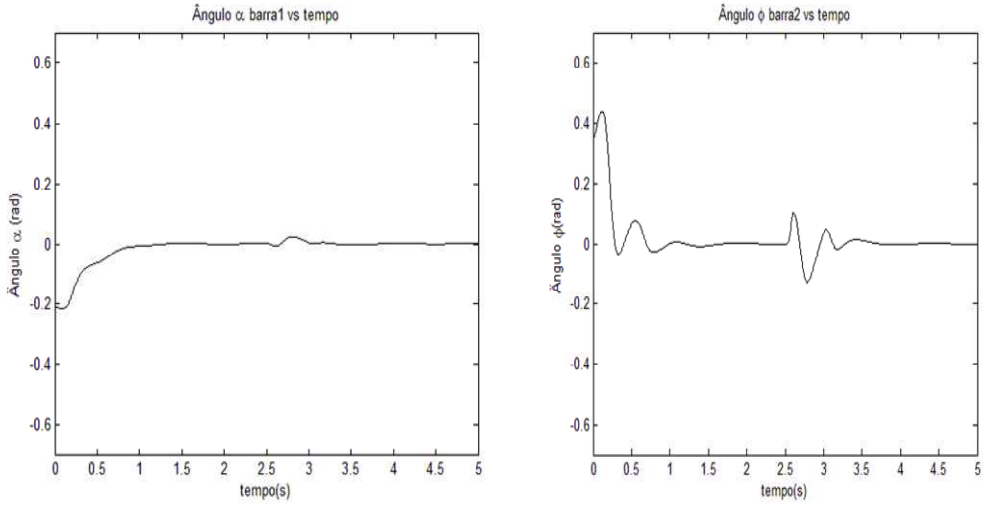
c. Variações de torque sem perturbação

d. Variações de torque com perturbação

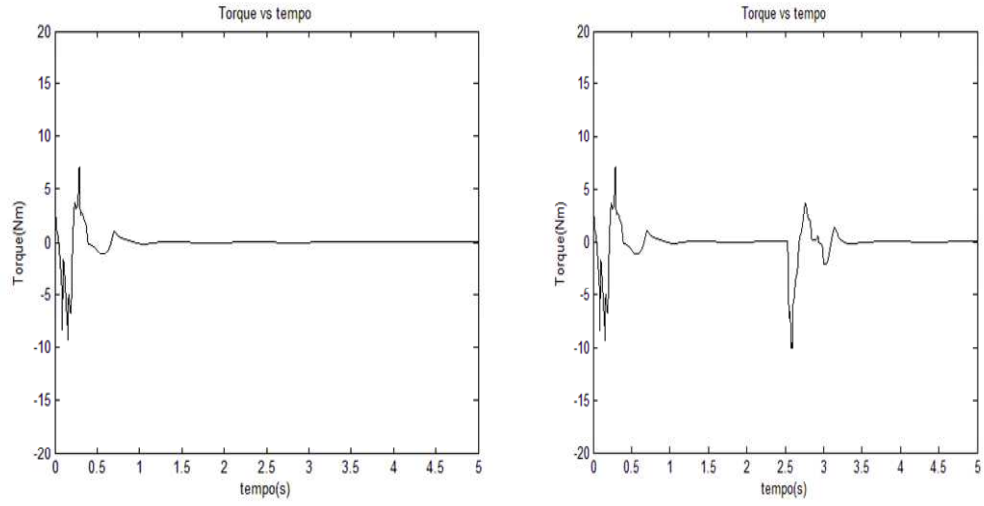
Figura 6.16-. Resposta da rede neural com condições iniciais $\{x_1,x_2,x_3,x_4\}$ ={0,1047 -0,4363 -0,2618 0,2793}.



a. Resposta angular sem perturbação



b. Resposta angular com perturbação



c. Variações de torque sem perturbação

d. Variações de torque com perturbação

Figura 6.17-. Resposta da rede neural com condições iniciais $\{x_1,x_2,x_3,x_4\}$ ={-0,2094 0 0,3491 0,1745}.

Como explicado na teoria, a estrutura do ANFIS é conformada por cinco camadas claramente diferenciáveis: em negro as entradas e as saídas (camada 1 e 5), em branco as camadas que contem parâmetros ajustáveis de acordo com as regras de aprendizado do algoritmo de treinamento (camadas 2, 4 e 5), e em azul as regras difusas geradas (ver Figura 6.19).

Observe que no casso do controlador neural foram usados 4000 dados de treinamento. Com isto a rede pode aprender a lidar com situações não previstas. No caso do sistema neuro-difuso são utilizados 8000 dados de treinamento, e o universo de discurso do controlador difuso é escolhido pelo algoritmo em relação com o valor máximo e mínino dos valores de treinamento de entrada. Isto significa que para se obter um conjunto de treinamento que satisfaça a faixa de operação do controlador é necessário testar as condições de funcionamento limite - à esquerda e à direita do ponto de operação- o qual justifica o fato de se utilizar o dobro de dados de treinamento.

Foram testadas funções de pertinência tipo Gaussiana e triangulares. As citadas funções apresentaram um desempenho satisfatório em termos de levar e manter o sistema na posição desejada. Na Figura 6.20 são mostradas as funções de pertinência ajustadas, do tipo Gaussiano.

Na Tabela 6.7 são apresentadas diferentes condições inicias que foram usadas para observar o desempenho do controlador. De maneira similar que nos outros casos, o sistema é submetido à ação de um distúrbio de 5N em forma de impulso com uma duração de 0.05 s. A Figura 6.21 e a Figura 6.22 apresentam a resposta do controlador com condições iniciais $\{x_1,x_2,x_3,x_4\}$ = {0,1047 -0,4363 -0,2618 0,2793}, e $\{x_1,x_2,x_3,x_4\}$ = {0,1745 -0,4363 0,2618, -0,2618}.

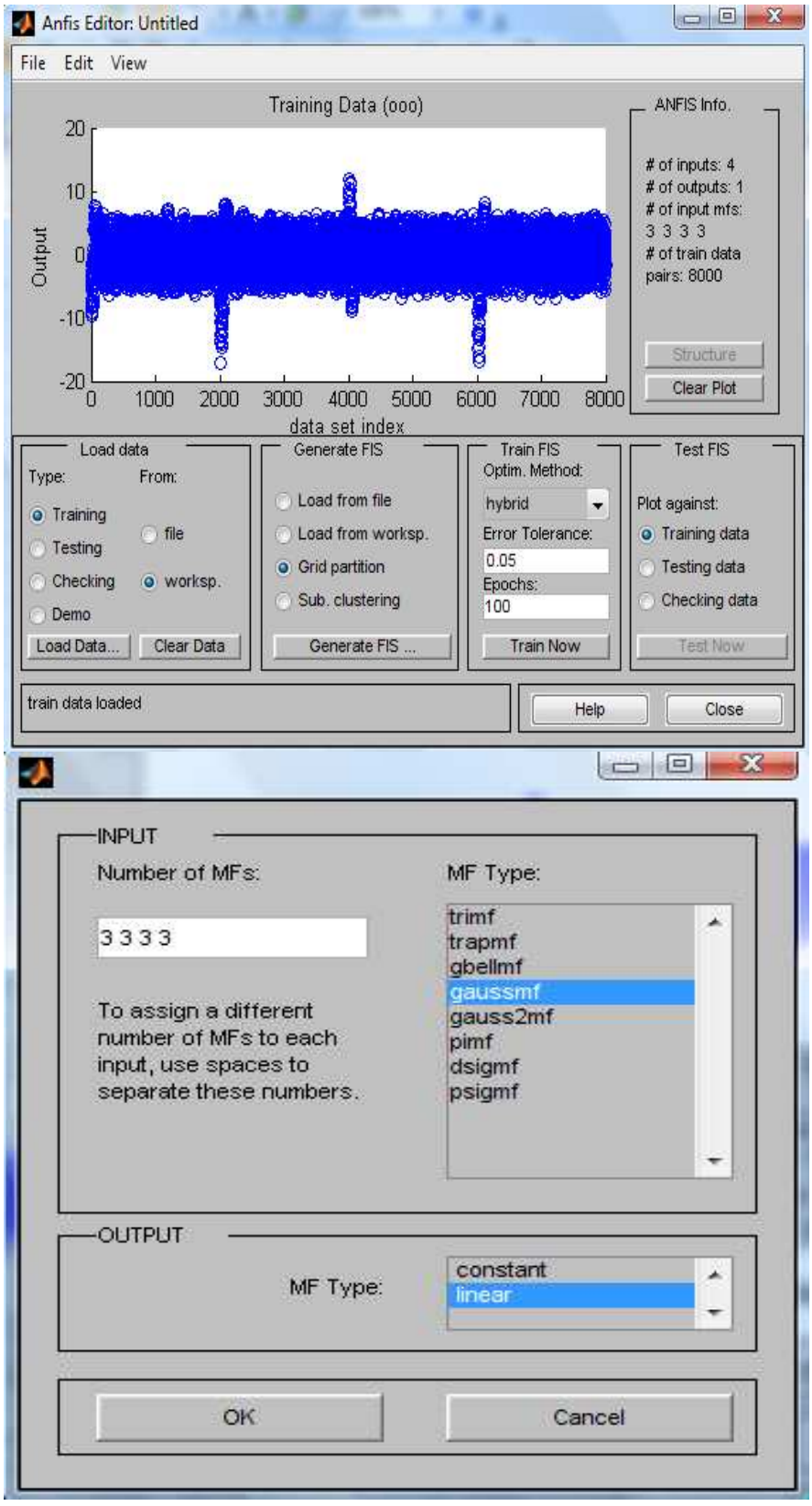


Figura 6.18-. Interface de usuário ANFIS.

Finalmente, pode-se observar a estrutura formada pelo ANFIS na Figura 6.19.

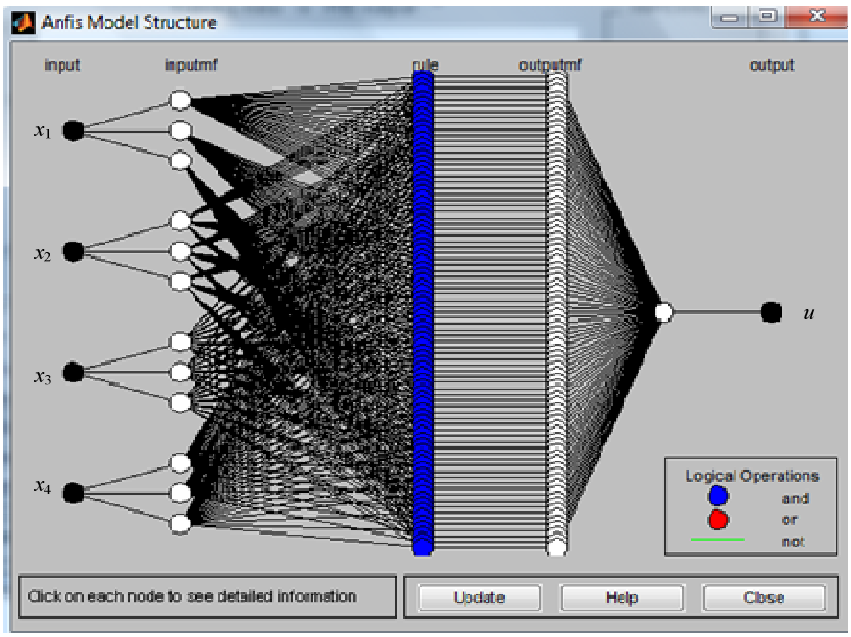


Figura 6.19-. Estrutura neuro-difusa gerada pelo ANFIS.

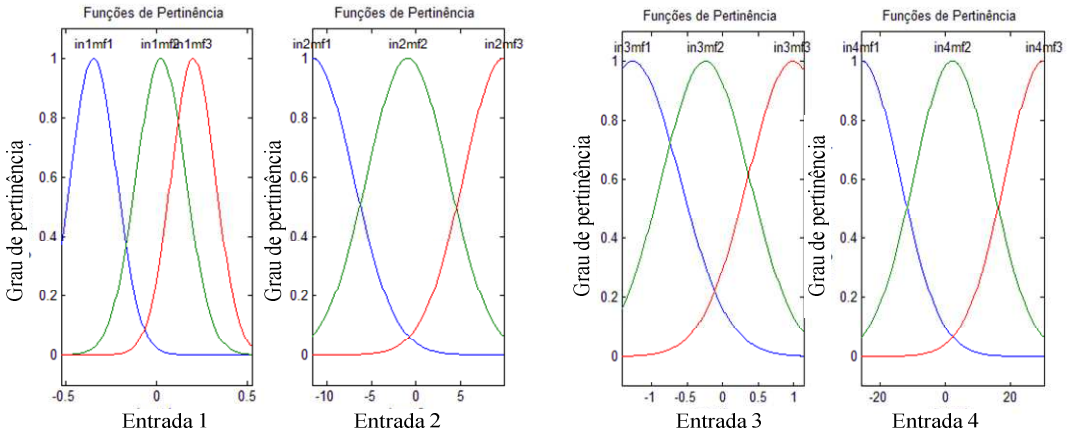


Figura 6.20- Funções de pertinência otimizadas pelo sistema ANFIS.

Tabela 6.7- Resposta do sistema neuro-difuso usando índices de desempenho

Condições Iniciais				**Índices sem Distúrbio**			**Índices com Distúrbio**		
α (rad)	ω_1 (rad/s)	φ (rad)	ω_2 (rad/s)	J_1		J_2	J_1		J_2
				α	φ	μ	α	φ	μ
0,1047	-0,4363	-0,2618	0,2793	0,0001	0,0029	2,1732	0,0001	0,0033	2,2367
-0,1047	-0,1571	0,2967	0,2618	0,0003	0,0016	0,5400	0,0003	0,0020	0,6034
0,0698	0,1745	0,1396	-0,1745	0,0011	0,0052	0,0638	0,0011	0,0056	0,1273
-0,1396	-0,1571	0,4363	0	0,0004	0,0022	0,3549	0,0004	0,0026	0,4184
0,1571	0,2094	-0,1745	-0,2967	0,0022	0,0058	0,0394	0,0022	0,0061	0,1028
0,1745	0,2618	-0,4363	-0,2618	0,0010	0,0042	0,0066	0,0010	0,0046	0,0745

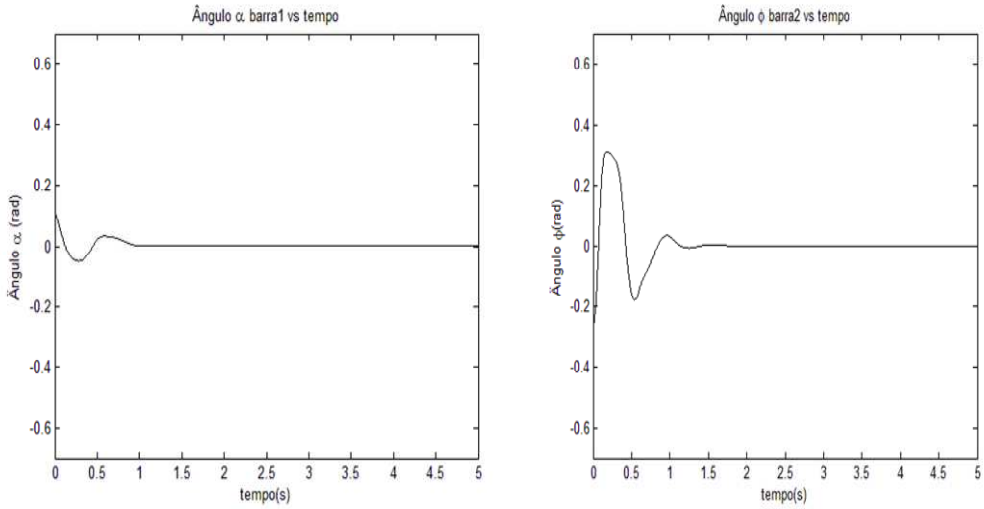
Foi discutida anteriormente a dificuldade de se utilizar sistemas difusos com mais de duas entradas, por causa da perda de sensibilidade sobre as variáveis, assim como o incremento na complexidade na geração de regras. O método neuro-difuso permite a criação de sistemas difusos de múltiplas entradas e uma saída. O sistema consegue estabilizar em um tempo maior que 1 segundo, e é tolerável à perturbação aplicada.

Pode ser observado que as redes neurais atingem uma faixa maior de operação quando comparadas com o controlador otimizado como o ANFIS. Mesmo assim, o controlador ANFIS posiciona as barras em um tempo menor que o controlador neural, como indicam os índices de desempenho J_1 na Tabela 6.6 e na Tabela 6.7 (para as duas condições iniciais das primeiras duas filas).

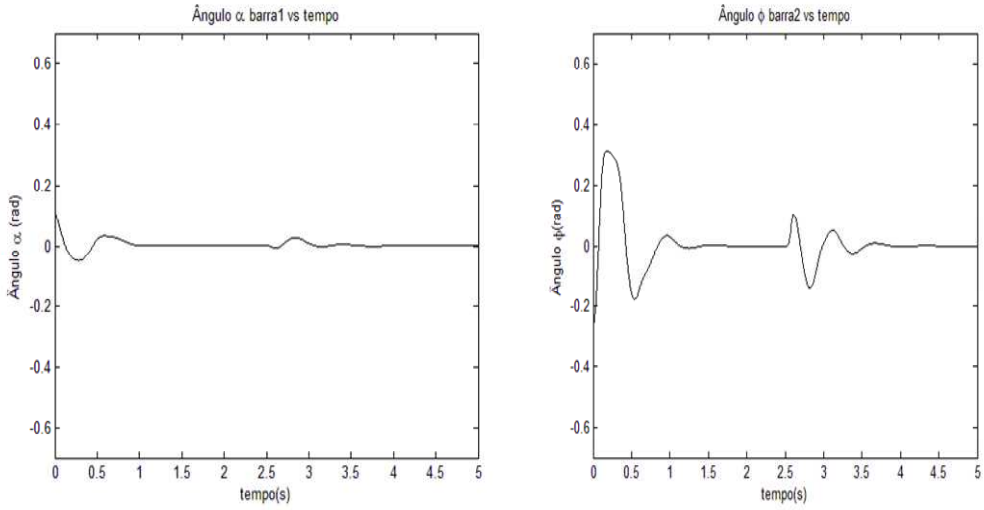
O controlador ANFIS gasta um consumo maior de energia que a rede neural, como mostrado nos índice de desempenho J_2 para as mesmas condições iniciais. Entre os três controladores projetados para esta situação é observado que o controlador difuso é mais lento que os outros dois (difuso e neural) na hora de estabilizar.

Pode ser observado que o projeto de controladores baseados em controle moderno é mais sistemático, em quanto os controladores inteligentes obtidos são mais flexíveis. O desempenho de todos os cinco controladores é satisfatório quanto ao tempo de resposta e sobre-passo. Neste sentido o LQR apresenta um menor sobre-passo. Em quanto à rapidez de posicionamento das barras, o controlador LQR apresentou a resposta mais rápida.

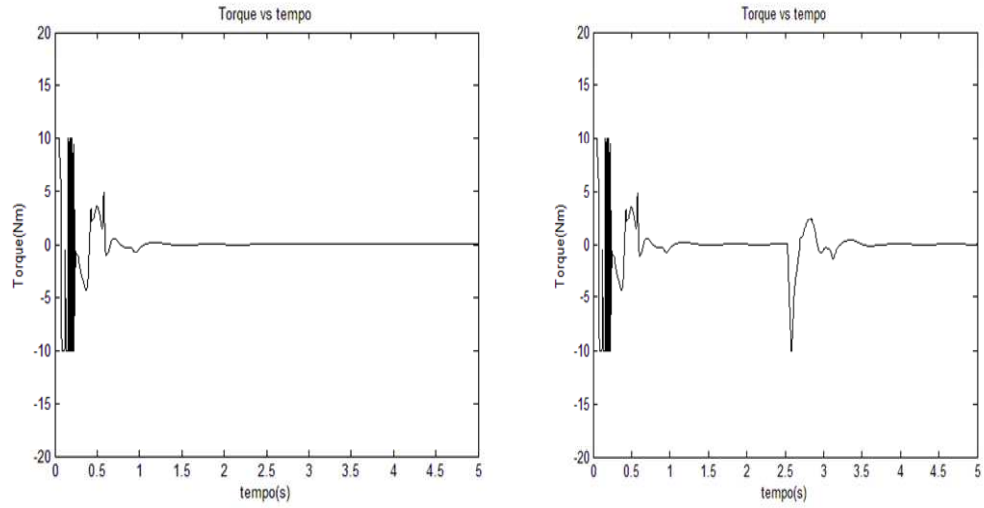
As primeiras duas filas de cada tabela de dados, usada para observar o comportamento do sistema, contêm as mesmas condições iniciais para todos os controladores. É observado que o controlador difuso é o controle que oferece menor consumo de energia para o atuador, em quanto o controlador neuro-difuso contêm o maior esforço de controle (embora o tempo de estabilização do controlador difuso é maior que para qualquer um dos outros controladores).



a. Resposta angular sem perturbação



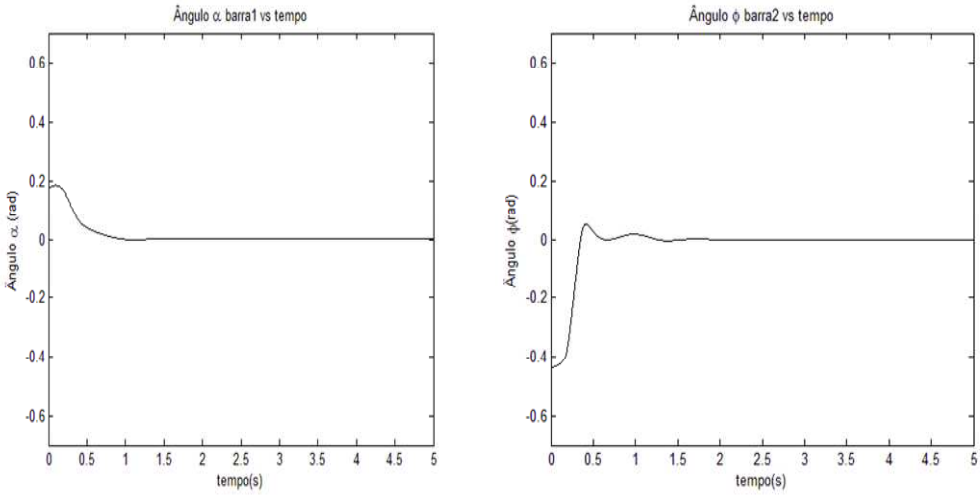
b. Resposta angular com perturbação



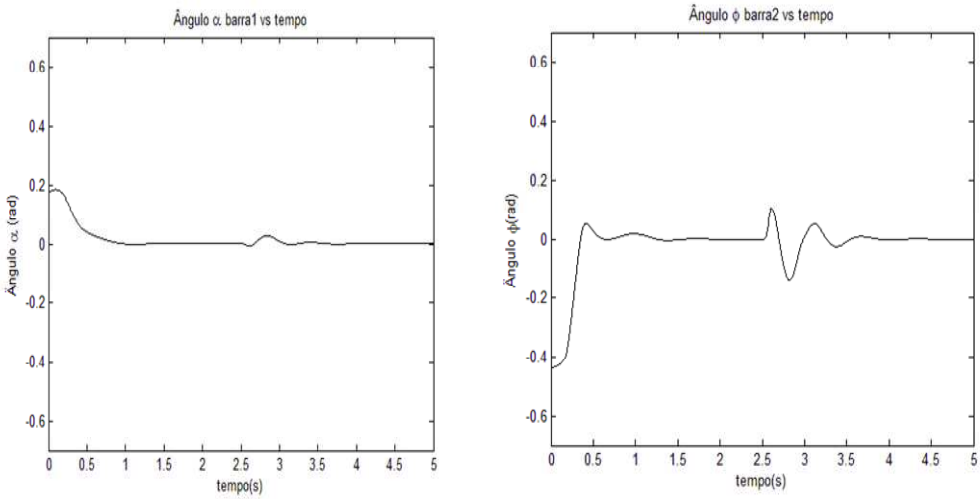
c. Variações de torque sem perturbação

d. Variações de torque com perturbação

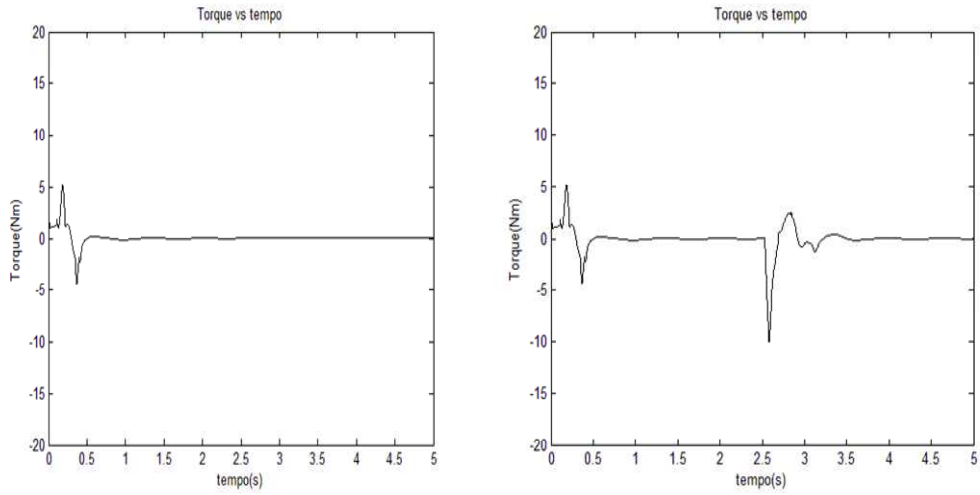
Figura 6.21-. Resposta do Controlador ANFIS com condições iniciais $\{x_1,x_2,x_3,x_4\}$ = {0,1047 -0,4363 -0,2618 0,2793}



a. Resposta angular sem perturbação



b. Resposta angular com perturbação



c. Variações de torque sem perturbação

d. Variações de torque com perturbação

Figura 6.22-. Resposta do controlador ANFIS com condições iniciais $\{x_1,x_2,x_3,x_4\}$ = {0,1745 -0,4363 0,2618 -0,2618}

A Tabela 6.8 apresenta os resultados comparativos dos controladores projetados neste capítulo. De acordo com os resultados obtidos foi dada uma nota a cada controlador entre 1 e 5 em relação com o parâmetros de resposta: a) tempo de estabilização, b) sobre-passo máximo, c) índice J_1, e d) índice J_2, como mostrado. É observado que o controlador LQR apresenta o melhor comportamento.

Devido ao uso de uma operação de subtração pelo controlador difuso para produzir um adequado sinal de controle, este gera valores de torque menores quando comparados com os outros controladores. Além disto, o uso dos ganhos fixos nas saídas dos subsistemas cria uma condição hierárquica, isto significa que quanto mais perto da posição de equilíbrio encontra-se a *barra* 1, os valores do torque para a *barra* 2 começam a ganhar importância.

Tanto os controladores modernos quanto o controlador difuso têm uma maior complexidade de projeto quando comparados com os controladores neural e neuro-difuso. As duas últimas técnicas dependem de um conjunto de exemplos e uma regra de aprendizado para o ajuste dos seus parâmetros, o problema é reduzido a atingir um valor de erro pré-estabelecido ou um número de épocas definido, ou seja: o processo é sistemático. Os controladores RE, LQR e Difuso dependem da observação do sistema e da influência de cada parâmetro sobre o desempenho do controlador, ou seja: o processo é empírico.

Assim, o controlador RE depende da sintonização de 4 parâmetros, a localização dos pólos nas posições adequadas. O controlador LQR neste caso depende da escolha da matriz R e dos elementos x e y da diagonal da matriz Q, ou seja, depende de três parâmetros de projeto. Enquanto o controlador difuso, um controlador de quatro entradas e três funções de pertinência tipo triangulares para cada entrada, supõe o uso de três parâmetros de ajuste por função, isto significa um total de nove parâmetros de ajuste por cada entrada e finalmente um total de 36 parâmetros de ajuste para o controlador.

O processo de sintonização do controlador difuso é mais complicado que qualquer outro controlador. Existem diversas formas de simplificar o problema, como por exemplo fazer coincidir vértices e parâmetros de duas funções ou mais, isto se vê refletido sobre os resultados apresentados na Tabela 6.8.

Tabela 6.8- Resumo Comportamento do Projeto dos Controladores

Tipo de Controle	Menor Tempo de Estabilização	Menor Sobre-passo	Menor Índice J_1	Menor Índice J_2
LQR	1	1	1	2
RE	2	2	2	3
Difuso	5	5	5	1
Neural	3	3	3	4
Neuro-Difuso	4	4	4	5

6.7 RESPOSTA DOS CONTROLADORES A VARIAÇÕES DE PARÂMETROS DO SISTEMA

Uma última análise dos controladores está relacionada com a variação de parâmetros do sistema e a observação do comportamento do mesmo, fora das condições nas que foram projetados. São utilizadas as seguintes condições iniciais: $\{x_1,x_2,x_3,x_4\}$ = {0,1047 -0,4363 -0,2618 0,2793}. Para as mesmas, todos os controladores conseguem controlar.

O teste consistiu em variar os parâmetros de massa (como mostrado na Tabela 6.9) em passos de 0.5 Kg, e o comprimento das barras (como mostrado na Tabela 6.10) em passos de 0.2 m. Para estas condições foi observado se o controlador conseguia ou não manter o sistema na posição invertida.

Tabela 6.9-. Resposta dos controladores à variação de massa

Controlador	Variações de Massa (Kg)				
	m_1= 0,5 m_2 = 0,5	m_1 = 1,5 m_2 = 0,5	m_1 = 0,5 m_2 = 1,5	m_1 = 1,5 m_2 = 1,5	m_1 = 2 m_2 = 2
RE	Sim	Sim	Não	Sim	Sim
LQR	Sim	Sim	Não	Sim	Não
Difuso	Sim	Sim	Não	Sim	Sim
Neural	Sim	Sim	Não	Sim	Sim
Neuro-Difuso	Sim	Não	Não	Sim	Não

Pode ser observado que o controlador RE, para uma variação de massa até um valor máximo de 2,5 Kg, não consegue controlar o sistema. Por outro lado, o controlador neural sim consegue.

Tabela 6.10- Resposta dos controladores à variação de comprimento

Controlador	Variações de Comprimento (m)				
	L_1= 0,2 L_2 = 0,2	L_1 = 0,4 L_2 = 0,2	L_1 = 0,2 L_2 = 0,4	L_1 = 0,6 L_2 = 0,6	L_1 = 0,9 L_2 = 0,9
RE	Não	Sim	Não	Sim	Sim
LQR	Não	Não	Não	Sim	Não
Fuzzy	Não	Sim	Não	Sim	Sim
Neural	Não	Não	Não	Sim	Não
Neuro-Fuzzy	Não	Não	Não	Sim	Não

As variações de comprimento resultam serem mais críticas que as variações de massa. Para este caso, os controladores não conseguem realizar o controle, exceto para variações iguais e superiores ao valor preestabelecido no projeto original. Por outro lado, todos conseguem controlar para variações iguais de comprimento entre 0,4 e 0,9 m. Neste caso, o controlador RE resulta ser mais robusto nesta consideração.

Um ajuste mais fino dos comprimentos é tolerável para todos os controladores para o caso em que os comprimentos variem entre 0,4 e 0,25 m.

Os parâmetros de tempo e estabilização, sobre-passo, são alterados. Portanto, o teste consistiu em observar como eles se comportam com a variação de parâmetros e não que tão ótimo é o desempenho dos controladores.

6.8 CONCLUSÂO DO CAPÍTULO

Neste capitulo foram apresentados os resultados experimentais dos cinco controladores projetados sobre a planta simulada em Matlab (LQR, RE, difuso, neural e neuro-difuso). Foi observado que todos os controladores conseguem manter o sistema na posição invertida, mesmo os controladores baseados em controle moderno, que são projetados sobre o sistema linearizado ao redor do ponto de operação. A resposta sobre o sistema não-linear é satisfatória e tão boa quanto as respostas que oferecem os controladores projetados usando controle inteligente.

O projeto dos controladores é baseado em métodos empíricos de escolha de parâmetros. Para tanto, podem ser executados algoritmos de busca destes parâmetros para sistematizar a escolha dos mesmos em relação aos parâmetros de desempenho.

É observada a importância dos sistemas neuro-difusos para a geração de controladores com múltiplas entradas. Partindo de um conjunto de dados experimentais (fornecidos por um supervisor) é possível gerar um sistema difuso que ajusta os seus parâmetros, baseado no aprendizado das redes neurais. A complexidade de ajuste das regras difusas é simplificada em quanto o algoritmo faz isto por meio dos dados de treinamento.

Os controladores foram submetidos a distúrbios externos e variações de parâmetros. Todos apresentaram respostas similares. Neste caso, o controlador LQR resultou ser o mais adequado em termos de sobre-passo e tempo estabilidade. Os controladores inteligentes possuem a vantagem de não precisarem para o seu projeto de modelos matemáticos.

Os índices de desempenho são úteis porque permitem observar o desempenho de um ponto de vista quantitativo. Para as mesmas condições iniciais, foi observado que o LQR é mais rápido e o difuso gasta menor energia no atuador.

A principal contribuição deste capítulo é a metodologia de projeto das diferentes técnicas de controle usadas, quando aplicadas ao projeto de estabilidade de um pêndulo duplo invertido interatuado na posição invertida.

7 IMPLEMENTAÇÃO EM FPGA

Este capítulo tem como objetivo descrever a implementação do controlador difuso aplicando as técnicas de sistemas reconfiguráveis, baseadas em FPGA.

Para propósitos de aplicação, um controlador difuso pode ser dividido em duas classes: (a) processador difuso com computação difusa especializada, ou (b) *hardware* difuso dedicado para aplicações específicas. O processador difuso de propósito geral tem sido implementado sobre varias plataformas: (a) computadores (PC ou WS), (b) processadores (microprocessadores, microcontroladores, processadores digitais de sinais), ou *transputer* (processador pioneiro em processamento paralelo), e (c) LTU (*Look Up Tables*, dispositivos de memória digital).

Os computadores atuais são baseados em processadores de propósito geral (*General Propposal Processor* - GPPs), fundamentados no modelo de von Neumann. Esse modelo permite uma grande flexibilidade na programação ao custo do desempenho. Ou seja, embora hoje a velocidade dos novos processadores comerciais seja muito elevada, há, ainda, uma degradação acentuada nessa velocidade, devido à execução seqüencial e às limitações do barramento de dados. Embora o tempo em que sinais elétricos percorram o *chip* seja bem baixos (picosegundos em tecnologias mais rápidas), a constante repetição de instruções pode tornar determinado algoritmo lento, o que faz necessária essa engenharia no circuito (Hartenstein, 2006). Neste contexto, pode-se afirmar que o modelo de von Neumann está orientado a fluxo de instruções, o que leva a se ter um contador de instruções para garantir a execução correta do programa. O conjunto de limitações deste modelo é denominado de “gargalo de von Neumann”.

Um processador difuso de propósito geral pode ser implementado rapidamente, e ser aplicado de maneira flexível. Entretanto, o mesmo tem um desempenho baixo, por ser limitado pelas restrições do modelo de von Neumann. Por outro lado, um *hardware* difuso dedicado requer de um tempo grande de desenvolvimento, podendo só ser usado restritivamente. Por outro lado, o mesmo poder oferecer um alto desempenho. No caso de um *hardware* especifico para um FLC (*Fuzzy Logic Controller*) existem dois requerimentos importantes: flexibilidade e alta velocidade (Kim, 2000).

Para tanto, a Computação Reconfigurável (RC) tem a proposta de se ter o desempenho de circuitos para execução específica de algoritmos com a flexibilidade de processadores de propósito geral (GPPs). Neste caso, os circuitos não são realmente modificados no *chip*, apenas reconfigurados (Hartenstein, 2006).

Um sistema de computação reconfigurável, baseado em chips que usam aspectos da reconfigurabilidade das FPGAs para implementar um algoritmo, é atrativo porque oferece um compromisso entre os ASICs (*Application Specific Integrated Circuits* - de propósito especial) e os GPPs. Diferentemente do hardware dedicado, os FPGAs são flexíveis porque podem ser facilmente reconfigurados (mesmo em tempo real) para os fins de implementar sistemas complexos (Kim, 2000).

O constante avanço das tecnologias de fabricação de circuitos integrados impõe novos retos ao mercado na área de microeletrônica, onde a contribuição mais significativa é a demanda de aplicações inovadoras, com elevados níveis de complexidade e tempos de desenvolvimento curtos.

As tentativas por satisfazer simultaneamente ambos os requisitos (desempenho e flexibilidade) têm levado aos pesquisadores de sistemas eletrônicos a proporem uma série de novas técnicas e estratégias de projeto entre as quais tem-se: (a) a concepção dos sistemas desde a perspectiva de *System on Chip* (SoC), (b) a mistura de elementos de processamento de propósito geral com outros de caráter específico, (c) o uso de técnicas híbridas baseadas no co-projeto de *hardware/software* (*co-design hardware/software*), (d) o emprego de módulos de Propriedade Intelectual (IP) e (e) a inclusão no ciclo de desenvolvimento de etapas de prototipagem rápido, baseadas nos dispositivos lógicos programáveis como os FPGAs.

Este capítulo descreve o desenvolvimento de módulos *xfuzzy* diretamente em *hardware*, que aceleram a execução de mecanismos de inferência. O controlador difuso foi sintetizado em código VHDL utilizando a ferramenta Xfuzzy 2.0. Esta ferramenta recebe como entrada uma descrição do sistema fuzzy (suas funções de pertinência) e gera de maneira automática o código VHDL. O código gerado pelo Xfuzzy é utilizado como um periférico na FPGA, e conectado a um processador embarcado no mesmo dispositivo, utilizando a

interface FSL (*Fast Simple Link*). O FPGA utilizado é o XC3S500E-4FG320C (Xilinx), qual recebe as variáveis de estado da planta simulada em Matlab e retorna o valor do sinal de controle. Para tanto, utilizado o kit de desenvolvimento da Xilinx, Spartan 3E starter.

7.1 CO-PROJETO DE *HARDWARE-SOFTWARE*

A alternativa por de implementação de algoritmos em *software* providencia soluções que apresentam grande flexibilidade, mas com necessidade de grande área de silício e longos tempos de execução (pelo gargalo de von Neumann). Por outro lado, a alternativa de implementação de algoritmos em *hardware* otimiza o tamanho e a velocidade do circuito, porém, limita a flexibilidade da solução. O meio termo entre ambas estratégias é a técnica de co-projeto *hardware/software*, a qual tenta obter um apropriado compromisso entre as vantagens e desvantagens destas duas abordagens.

Em processos de analise do co-projeto de *hardware-software*, todas as tarefas que o sistema deve realizar devem ser analisadas, avaliando o impacto que as opções da possível implementação podem ter nos fatores que definem a funcionalidade do sistema e seu custo. Os principais parâmetros para considerar na avaliação são a velocidade de execução e a área necessária pela implementação em *hardware*. Baseado nestes resultados, um processo de particionamento é obtido, o qual consiste em decidir qual tarefa deva ser executada pelo *software* e qual deve ser implementada pelo *hardware* embarcado (Lacerda, 2002).

7.2 O MICROPROCESSADOR EMBARCADO MICROBLAZE (XILINX)

O Microblaze é um processador virtual (*soft processor*) que é construído por uma combinação de blocos de código chamados *cores* dentro da FPGA. O interessante é a possibilidade de trabalhar sobre ele como em qualquer microprocessador, assim como poder ajustá-lo às necessidades do projeto. O Microblaze e um processador RISC de 32 bits baseado em uma arquitetura Harvard, ou seja, dados e instruções são endereçados e acessados em espaços de memória distintos. O acesso a memória é feito através de 3 tipos de barramentos: *Local Memory Bus* (LMB), *On Chip Peripheral Bus* (OPB) e o *Xilinx Cachê Link* (XCL).

O LMB proporciona um acesso de ciclo simples ao bloco RAM: (BRAM). A interface OPB proporciona uma conexão para ambos periféricos *On chip* e *Off chip* e memória. A interface *Cachê Link* é utilizada para uso de controladores de memória externa

especializada. Além das 3 interfaces de memória suportadas (LMB, OPB, XCL), o Microblaze também suporta até 8 portas do tipo *Fast Simplex Link* (FSL), cada uma com interface *máster* e *slave*. O FSL é uma simples e poderosa interface ponto a ponto que conecta aceleradores hardware desenvolvidos pelos usuários ao processador com o *Pipeline* de Microblaze, para acelerar algoritmos de tempo crítico. Todas as instruções do Microblaze são de tamanho 32 bits (Jesman *et al.,* 2008).

Microblaze é um tipo de processador de carga e armazenamento, o que significa que este pode só carregar e armazenar dados desde e para a memória. Esta arquitetura não pode fazer operações sobre dados diretamente em memória. Em vez disso, o dado em memória deve ser carregado dentro do processador Microblaze, para depois localizá-lo nos registros de propósito geral para executar alguma operação. Ambas as interfaces de dados e instruções de Microblaze são de largura de 32 bits, e usam formatos *Big-Endian* e bit reservados para representação de dados. (Jesman *et al.,* 2008)

7.3 INTERFACES DE CONEXÃO COM O MICROBLAZE

Em Microblaze existem diversas formas de interligar e utilizar códigos VHDL, que representem alguma funcionalidade pré-definida pelo usuário. Estes códigos são chamados de *IP cores* (núcleos de Propriedade Intelectual), sendo que a utilização deles junto com a implementação de um processador embarcado representam uma interessante e importante funcionalidade. Isto é devido a que se faz possível a utilização das características de um processador normal com o poder de processamento em paralelo dos FPGAs, acelerando o tempo de execução em software.

A continuação, duas formas de conexão de periféricos com o Microblaze utilizando o *On Chip Peripheral Bus (OPB)* e a *Fast Simplex Link (FSL)* serão explicadas. Se a aplicação é de tempo crítico, o usuário deve conectar o periférico utilizando a interface *FSL*.

Se o usuário utiliza a interface OPB, deve ter em conta o barramento padrão, ou seja, deve ter certeza de que o *IP core* está conectado e que satisfaz as especificações do barramento. Para isto, o Microblaze traz um *wizard* que gera uns *templates* utilizando o protocolo de comunicação de propriedade intelectual da Xilinx chamado *IPIF*. Estes *templates* facilitam e agilizam a conexão do *IP core* com o barramento. Entretanto, a principal desvantagem deste tipo de conexão é que o protocolo de comunicação toma muito tempo pra conectar o

IP core com o barramento, perdendo assim a vantagem que proporciona o processamento em paralelo.

Se o usuário utiliza a interface FSL, é possível usar funções predefinidas em C e C++ para usar o *IP core* na aplicação *software*. O usuário pode usar 8 entradas e 8 saídas para o *IP core*. Os periféricos podem ser criados como mestre ou como escravo. Um periférico conectado na porta mestre do barramento *FSL* coloca sinais de dados e controle para o *FSL*. Um periférico conectado na porta escravo do barramento *FSL* lê e pega sinais de dados e controle do barramento *FSL*. A configuração do Microblaze pode ser usada em conjunto com alguma das outras configurações de barramentos.

A Xilinx fornece um *software* para o desenvolvimento de sistemas sobre seus FPGAs que permite acessar a suas propriedades e componentes. Os componentes de propriedade intelectual podem ser adicionados aos projetos *hardware* mediante o uso de suas ferramentas. O Kit de Desenvolvimento de sistemas Embarcados (EDK) fornece a interface de usuário gráfica da Xilinx (XPS), como a principal ferramenta de *software* para desenvolvimento de sistemas embarcados.

O EDK usa uma biblioteca de propriedade intelectual (IPIF) para implementar funcionalidades comuns dentro de vários periféricos do processador. O mesmo também proporciona um protocolo para um conjunto de barramentos, simplificado assim o barramento *IP Interconect* (IPIC), o qual é muito mais fácil que operar o protocolo correspondente para os barramentos OPB ou PLB diretamente (Newcomb, 2008).

7.4 COMPONENTES DO SISTEMA DIFUSO IMPLEMENTADO EM *HARDWARE*

O diagrama de blocos básico de um sistema difuso que utiliza o método de defuzzificação da media difusa é mostrado na Figura 7.1 (baseado na arquitetura gerada pelo Xfuzzy). Os circuitos geradores de funções de pertinência (MFCs) geram os conjuntos difusos utilizados nos antecedentes das regras. Para cada valor das entradas, os MFCs fornecem tantos pares (etiqueta, valor de ativação) como grau de sobreposição tenha se previsto no sistema. O máximo número de regras ativas fica limitado ao fixar um grau de sobreposição.

A etapa de inferência encarrega-se de processar seqüencialmente cada uma de essas regras, usando para isto um arreglo de multiplexores controlados por um contador. Em cada ciclo

do contador os graus de pertinência são misturados a través do operador *MIN* ou *Produto* para calcular o grau de ativação da regra (α^i), em quanto as etiquetas dos antecedentes endereçam a posição de memória de regras que contêm o conseqüente correspondente para dita regra (c^i) (Sánchez *et al.*, 2001; Sánchez *et al.*, 2006).

Finalmente, a etapa de defuzzificação calcula a saída do sistema como a media dos conseqüentes da regra ponderada pelos seus graus de ativação, segundo a Equação 7.1

$$y = \sum_r \frac{\alpha^i . c^i}{\sum_r \alpha^i} . \tag{7.1}$$

Em um sistema de *I* variáveis de entrada com *N* bits de precisão e grau de sobreposição 2, que emprega as etapas de *pipeline* mostradas na Figura 7.1, a velocidade de inferência vem limitada pela maior das três quantidades seguintes: o número de ciclos de relógio necessários para calcular os antecedentes, o número de regras ativas (2^I) e o número de ciclos empregados na divisão.

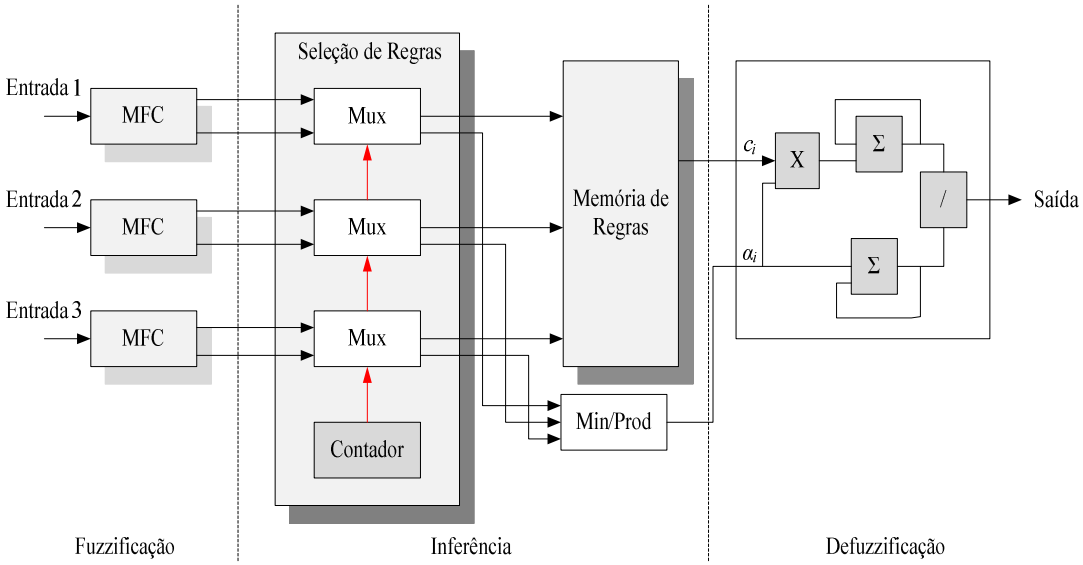


Figura 7.1-. Implementação em *hardware* (Xfuzzy) de um sistema difuso (Modificado, Sánchez *et al.*, 2001)

A operação do sistema difuso vem determinada pelo conteúdo de sua base de conhecimento, ou seja, pelo conjunto de regras que a definem. Como mostrado no diagrama de blocos a base de conhecimento encontra-se distribuída entre os MFCs e a memória de regras.

Os MFCs podem ser implementados mediante técnicas vetoriais ou aritméticas. No primeiro caso, as memórias dos antecedentes que armazenam os distintos graus de pertinência são endereçadas pelas palavras binárias correspondentes com os valores das entradas.

O tamanho de cada uma das I memórias é de 2^N palavras de $(2*D + \log_2 F)$ bits, onde D é o número de bits de precisão com que armazenam-se os graus de pertinência e F é o número de etiquetas usado. O emprego de MFCs baseados em memória permite a definição de funções de pertinência arbitrarias, mas pode limitar a realização do circuito quando se incrementa o número de entradas ou sua precisão (Sánchez *et al.*, 2001; Brox, 2006).

Desde o ponto de vista de implementação, tanto a memória de antecedentes (independentemente do tipo de MFC utilizado) como a memória de regras admitem diferentes opções: (a) memória RAM ou ROM (de acordo com o interes ou não modificar a base de conhecimento do sistema de forma concorrente com sua operação), ou (b) circuitos combinacionais, que permitam reduzir o tamanho da lógica necessária.

7.5 O SISTEMA XFUZZY

Atualmente, existem diferentes versões do ambiente Xfuzzy (2.0, 2.1., 2.2, 3.0). As versões mais antigas como a 2.x. permitem a síntese de código em VHDL do sistema difuso. Mesmo assim, o código gerado apresenta alguns erros que o programador deve corrigir. Versões posteriores, por exemplo 3.0, permitem a geração de sub-sistemas mas não tem disponibilizado ainda a biblioteca *xfvhdl* para a geração do código VHDL.

O *xfvhdl* impõe algumas limitações sobre o arquivo fonte *xfl*, derivadas da definição corrente da biblioteca de componentes, entre elas:

- O sistema deve ter uma saída só. Sistemas com múltiplas saídas podem ser sintetizados por meio de arquivos fonte *xfl* separados.
- Composição de módulos não é suportada.
- Todos os tipos de variáveis de entrada devem conter o mesmo número de funções de pertinência e não mais que duas sobreposições em algum ponto do universo de discurso.

O *xfvhdl* suporta só um subconjunto de conectividades predefinidas e operações:

- T-norms: min
- T-conorms: max
- Negação: not
- Defuzzificação: FuzzyMean, Yager, Centro de Somas.

A idéia de se dispor de um *hardware* específico, de baixo custe e alta funcionalidade em quanto a velocidade de operação e consumo, faz necessário a implementação eletrônica de sistemas difusos. Para isto é conveniente contar com uma metodologia de projeto para este tipo de sistemas, que permita percorrer as diferentes etapas desde a especificação do sistema até a implementação e testado do circuito.

Neste caso, a metodologia de projeto baseia-se no entorno de desenvolvimento de sistemas difusos Xfuzzy (Moreno *et al.*, 2001), este inclui ferramentas de descrição, simulação e simplificação que permitem definir, verificar e otimizar o sistema difuso.

A ferramenta *xfvhdl* toma como entrada a especificação XFL de um sistema difuso e gera um conjunto de arquivos VHDL que implementam a máquina de inferência. Para isto, os componentes correspondentes às distintas opções arquiteturais têm sido previamente definidos e incluídos em uma biblioteca de células parametrizáveis que satisfazem as restrições impostas pelas ferramentas de síntese disponíveis (Brox *et al.*, 2006).

A descrição VHDL constitui o ponto de partida das etapas de simulação (ModelSim de Mentor Graphics), assim como a tarefa de síntese lógica (XST de Xilinx). Com o intuito de acelerar a realização destas duas etapas de projeto, *xfvhdl* fornece duas saídas adicionais: Um arquivo de *Testbench* para facilitar a simulação do sistema difuso e um procedimento de comandos que permitem realizar em *batch* os processos de síntese e implementação da FPGA sobre dispositivos *Xilinx*.

7.6 O SISTEMA REALIDADE VIRTUAL USANDO MATLAB

Para efeitos de validação do modelo sistema-controlador, foi desenvolvido um protótipo tridimensional utilizando a linguagem de programação Virtual Realm Builder 2.0 do robô ciclista. Foi criado um objeto em Matlab para acessar as instâncias do protótipo e modificar desde o programa geral os parâmetros do protótipo.
Figura 7.2.a. apresenta o esquema do protótipo desde diferentes pontos de vista. Cada parte do robô ciclista é um objeto que têm coordenadas relativas com relação a seu próprio centro geométrico, e coordenadas absolutas com relação a sistema de coordenadas absoluto do espaço virtual. Este sistema de coordenadas varia com o utilizado em Matlab, para o qual o programador deve tomar providencia. Em geral, foram utilizados para a parte superior do ciclista 13 objetos, para a parte inferior do ciclista e para a bicicleta 45 objetos.

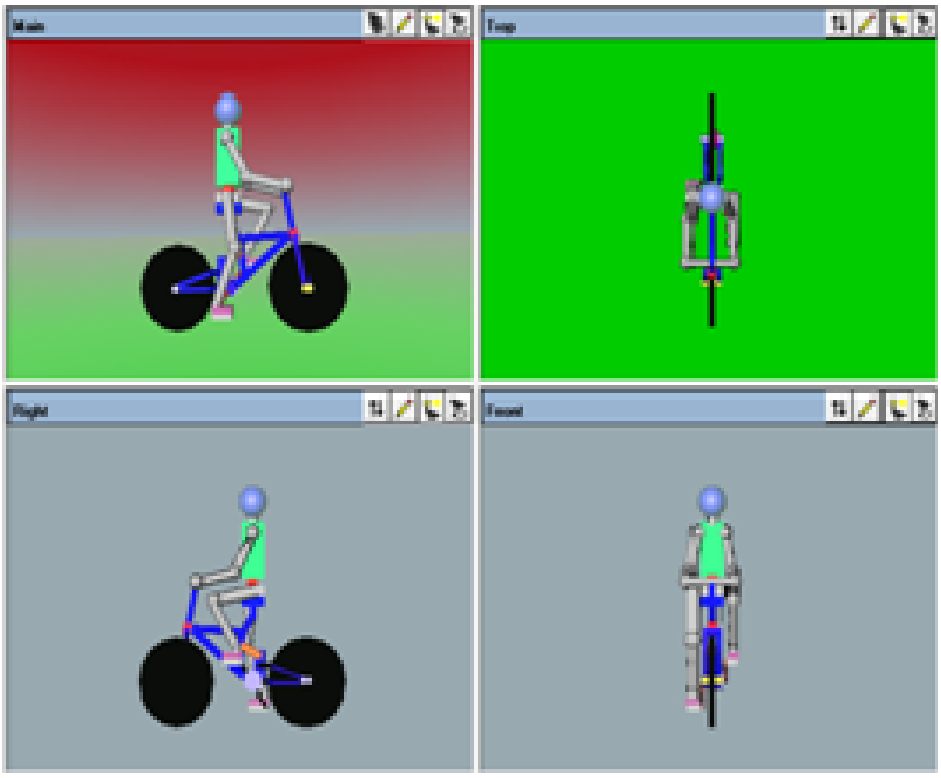
O protótipo gerado em Virtual Realm Builder 2.0 e o Matlab são ligados tendo em conta a funcionalidade do pêndulo duplo invertido interatuado (Acrobot), assumindo: (a) que a bicicleta fica em um mesmo plano, (b) que a inclinação com relação a vertical é conforme com a inclinação da primeira barra, e (c) que o tronco do ciclista vai se movimentar conforme a segunda barra. Isto é possível a través de um atuador que substitui o quadril do ciclista. As variáveis de estado são entregues para o protótipo virtual em cada instante de tempo, e o resultado é apresentado na

Figura 7.2.b.

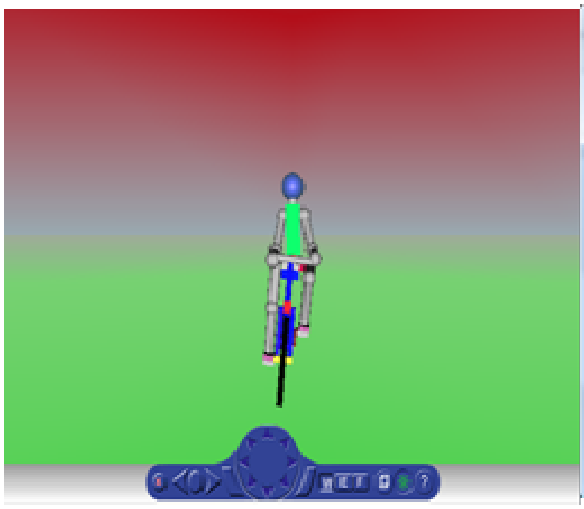
7.7 IMPLEMENTAÇÃO DO CONTROLADOR E OS RESULTADOS OBTIDOS

O trabalho realizado sobre a FPGA em conjunto com a simulação da planta e o ambiente virtual, é descrito no diagrama de blocos de controle apresentado na Figura 7.3.

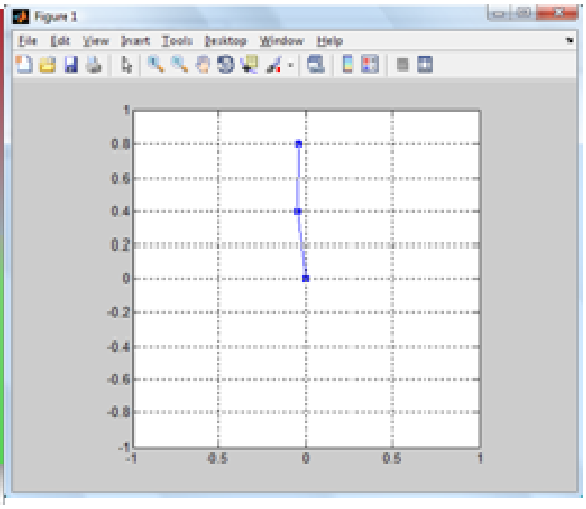
Como mostrado no capítulo anterior, o sistema difuso foi dividido em dois subsistemas. O primeiro subsistema depende das variáveis de entrada da *barra 1*, posição angular e velocidade, e o segundo depende das variáveis de entrada da *barra 2*. Inicialmente, o controlador é desenvolvido com a ferramenta de Matlab e testado sobre a simulação da planta.



a) Diferentes pontos de vista



b) *Viewer* WRL



c) *Acrobot* em Matlab

Figura 7.2- Representação tridimensional do robô em bicicleta utilizando *Virtual Realm Builder 2.0*

O seguinte passo foi a criação do sistema difuso no ambiente Xfuzzy 2.1. Este ambiente permite a geração do código VHDL do controlador (mediante a ferramenta *xfvhdl*) para utilizá-lo sobre a FPGA spartan 3E starter da Xilinx.

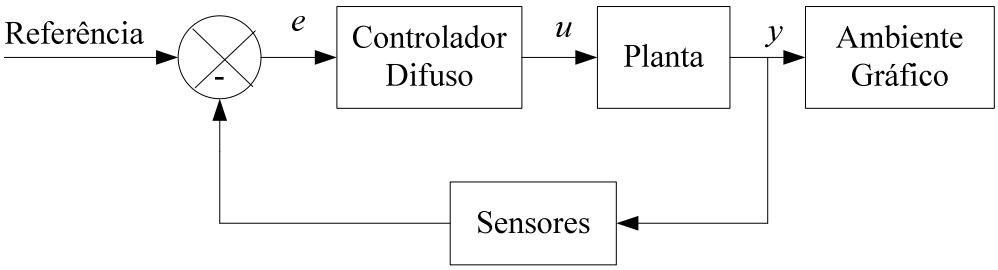


Figura 7.3- Diagrama de blocos de controle baseado em FPGA.

O código VHDL é composto por um programa principal, e por outros programas encarregados de realizar as diferentes etapas apresentadas na metodologia da Figura 7.1. Por exemplo, para o *xfuzzy_01* da primeira barra tem-se o programa principal *pendulo_barra1.vhd*, e os programas:

- *control_nc.vhd*,
- *pendulo_barra1AntecedentMem_1.vhd*,
- *pendulo_barra1_AntecedentMem_2.vhd*,
- *MF_Grade.vhd*,
- *pendulobarra1RulesMem.vhd*,
- *Minimum2.vhd*,
- *FuzzyMean.vhd*n
- *Division.vhd*.

Além disto, são geradas duas bibliotecas para definição global de constantes e entidades chamadas de *pendulobarra1Constants.vhd* e *pendulobarra1Entities.vhd*.

O código gerado pela ferramenta *xfvhdl* para sistemas difusos é um *template* cujos valores de entrada e saída devem ser previamente processados. Quando o universo de discurso contém valores negativos, o código gerado define valores de entrada entre 0x00 e o número máximo de acordo com o número de bits escolhido para a precisão do sistema. Por exemplo, para 5, 7 ou 12 bits temos um limite máximo de 0x1F, 0x7F, 0xFFF, respectivamente. Assim, o pré-processamento consiste em relacionar o valor real de entrada com uma representação adequada entre 0x00 e 0x7F para uma quantidade de 7 bits. Não é possível gerar sistemas difusos com mais de uma saída para esta versão do *Xfuzzy*.

A metodologia de projeto do controlador difuso é tipo Takagi Sugeno (Coelho *et al.*, 2003). Cada conjunto difuso contém duas entradas e 3 funções de pertinência, e utiliza 8 *singletons* de saída denominados *pdez*, *poito*, *pseis*, *pquatro*, *zero*, *nquatro*, *nseis*, *noito*, *ndez*. Em total, usou-se um número de 9 regras difusas para cada controlador. Para cada entrada e saída foi utilizada uma precisão de 12 bits. O método de defuzzificação usado é o centro de área.

A criação e implementação dos controladores difusos como periféricos têm as seguintes etapas: (1) uma vez obtidos os dos controladores em linguagem de descrição de hardware, são criados dois periféricos usando o *wizard* do XPS 10.1 e fazendo uso da interfase FSL, (2) o *wizard* cria um *template* que pode servir como exemplo para gerenciar o fluxo de dados entre o periférico e o Microblaze (o uso das bandeiras que indicam o estado de envio e recepção de dados). Nesta abordagem, criou-se um componente no *template* que liga os sinais de cada periférico VHDL com os sinais da interface FSL.

Uma pequena máquina de estados finitos (*Finite State Machine* – FSM) controla o fluxo de dados de acordo com o estado das bandeiras. Neste caso deve-se importar o periférico para o sistema e conectá-lo com o Microblaze. Para isto, devem-se conectar os periféricos com as interfaces FSL escolhidas, e as interfaces escolhidas com o relógio do sistema. A arquitetura é mostrada conforme na Figura 7.4.

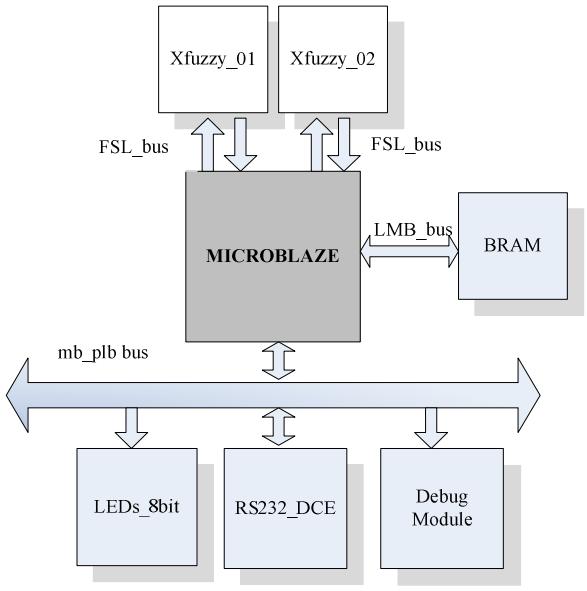


Figura 7.4- Arquitetura de controle sobre a FPGA.

A planta simulada envia para a FPGA as variáveis de estado via RS232, o processador embarcado recebe os dados, repassado para cada módulo *xfuzzy_01* e *xfuzzy_02* as variáveis correspondentes. Cada módulo *xfuzzy* retorna para o processador o valor do controlador em relação com a situação atual da planta. Finalmente, é enviada a variável de controle para a planta simulada no Matlab via RS232. O comportamento da planta está ligado ao comportamento do protótipo virtual. O funcionamento do sistema em geral é apresentado na Figura 7.5 (Castro *et al.*, 2009c).

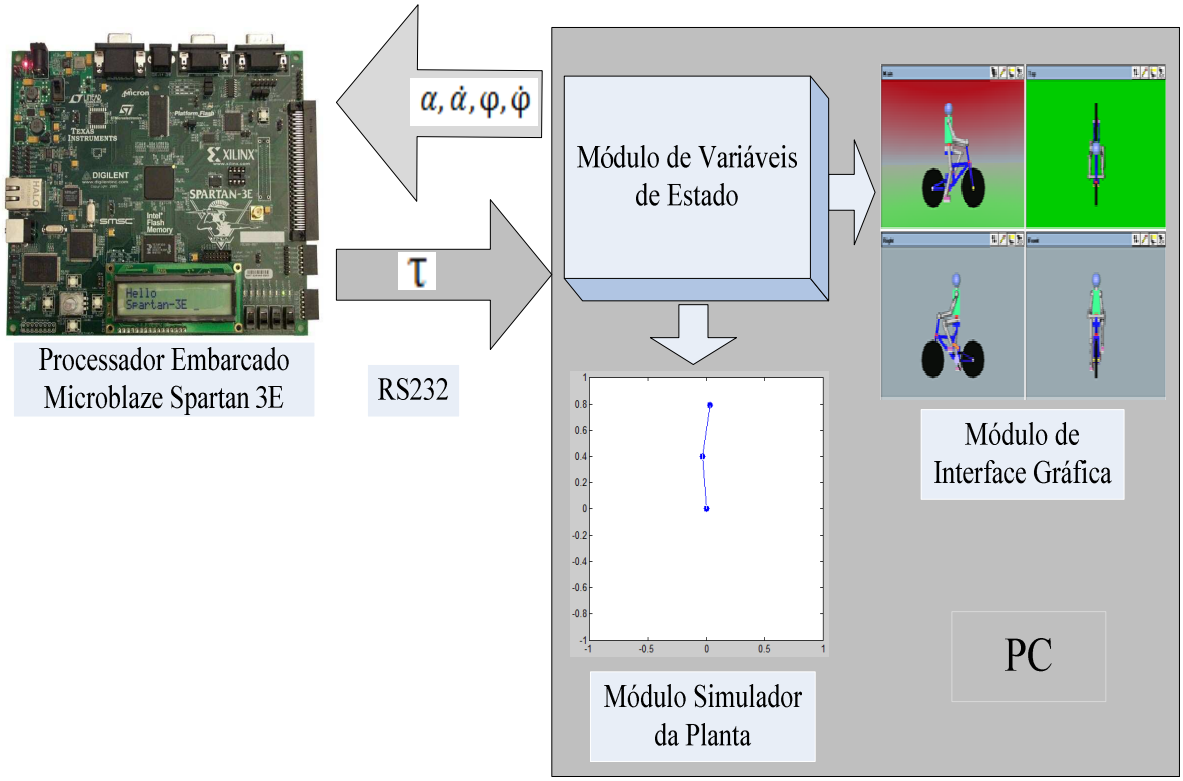


Figura 7.5- Vista geral do sistema implementado.

A Tabela 7.1 apresenta os dados de consumo de recursos da FPGA quando implementados os sistemas difusos *xfuzzy_01* e *xfuzzy_02* sobre ela. Trabalhos prévios Na área de implementação de sistemas difussos em FPGAs não apresentam resultados para esta metodologia do controlador difuso, os quais incluem uma partição do sistema (Mahmoud *et al.*, 1995; Kim, 2000; Gschwind *et al.*, 2001; Sánchez *et al.*, 2007). Por outro lado, os resultados obtidos em outros trabalhos apresentam diferentes funções de pertinências e diferentes precisões em quanto ao número de bits escolhidos para representar as entradas.

Tabela 7.1- Resultados da implementação dos módulos Xfuzzy na Spartan XC3S500E-4FG320C.

Controlador Difuso	4 input LUT *Max: 9,312*	Slice FFs *Max: 9,312*	IOBs *Max: 232*	RAMB16s *Max: 20*
Xfuzzy_01	188	148	40	8
Xfuzzy_02	156	148	40	8

Com o intuito de comparar a velocidade de execução dos algoritmos implementados em *software*, e a velocidade de execução dos algoritmos usando co-projeto de *hardware-software* foram realizados os testes mostrados nas Tabelas 15 e 16, respectivamente.

Para medir o tempo de execução foi utilizado um timer na FPGA. Assim é possível estimar o número de ciclos que gasta a função desde o momento em que são entregues os dados de posição angular e velocidade, até receber o valor do sinal de controle (torque) para cada subsistema difuso. O tempo é estimado de acordo com o relógio da placa (o relógio tem uma freqüência de 50 Mhz), ou seja, que um ciclo de instrução demora 0.02 *us*. Isto permite estimar o tempo total da execução de todas as instruções usadas pelos algoritmos de controle (implementação em *software*).

O ambiente de simulação *xfuzzy* permite a geração de código em C, C++ e Java, além do código VHDL. Isto permitiu implementar o código em C para cada subsistema *xfuzzy_01.h* e *xfuzzy_02.h* e testar o seu funcionamento no processador Microblaze.

Outro teste consistiu em utilizar as funções *tic toc* de Matlab que permite estimar o tempo que demora em se executar uma função entre as linhas de código *tic* e *toc*. Assim, foram testados os subsistemas difusos criados e projetados, inicialmente, no Matlab e reproduzidos sob Xfuzzy. Neste caso, o Matlab estava rodando sobre uma plataforma Intel Core(TM)2 Duo CPU com um clock de 1.8 GHz.

Tabela 7.2- Comparação entre tempos de execução do controlador *xfuzzy_01* para a *barra 1*.

Condições de entrada		**# Ciclos no *Software* embarcado**	**Tempo de execução no *software* embarcado (ms)**	**# Ciclos Xfuzzy (implementação em *hardware*)**	**Tempo de execução no *hardware* embarcado - fuzzy VHDL (us)**	**Tempo de execução em Matlab (ms)**
Ângulo (rad)	**Velocidade (rad/s)**					
-0.5	-1.0	283.492	5.6698	191	3.82	2.4
-1.0	-10.0	265.212	5.3042	190	3.80	1.2
-1.0	10.0	268.963	5.3792	190	3.80	1.0
1.0	10.0	272.422	5.4484	191	3.82	1.0
0.8	-2.0	285.472	5.7094	191	3.82	1.2
0.0	1.3	280.189	5.6037	191	3.82	1.0

Os resultados da Tabela 7.2 e da Tabela 7.3 apresentam resultados interessantes. É observado que o sistema em *hardware* é mais rápido que os sistemas projetados sobre processadores baseados no modelo de von Neumann (em três ordens de magnitude). Pode ser visto também que o sistema projetado em Matlab é mais rápido que o sistema projetado no Microblaze.

Os códigos fonte correspondentes às implementações estão disponibilizados no Anexo B.

Tabela 7.3- Comparação entre tempos de execução do controlador *xfuzzy_02* para a *barra 2*.

Condições de entrada		**# Ciclos no *Software* embarcado**	**Tempo de execução no *software* embarcado (ms)**	**# Ciclos Xfuzzy (implementação em *hardware*)**	**Tempo de execução no *hardware* embarcado - fuzzy VHDL (us)**	**Tempo de execução em Matlab (ms)**
Ângulo (rad)	**Velocidade (rad/s)**					
-0.5	-9.9	288.639	5.7727	191	3.82	2.0
-0.5	-4.5	287.824	5.7564	191	3.82	1.1
-0.9	9.9	288.384	5.7676	191	3.82	1.1
0.9	9.9	292.662	5.8532	191	3.82	1.1
0.8	-2.0	291.844	5.8368	191	3.82	1.2
0.0	1.3	269.786	5.3957	191	3.82	1.1

7.8 CONCLUSÃO DO CAPÍTULO

Neste capítulo foram explicados alguns aspectos relacionados com a computação reconfigurável, assim como aspectos do modelo de computação tradicional baseado no modelo de von Neumann.

Foi explicada a metodologia de criação e funcionamento de sistemas difusos sob hardware, o uso do ambiente de projeto de controladores difusos Xfuzzy 2.1, usando a ferramenta *xfvhdl*, e as limitações de implementação. Também foram tratados temas concernentes á utilização de sistemas embarcados em FPGAs, o uso de Microblaze, e as interfaces de conexão com periféricos de propriedade intelectual.

As principais contribuições deste capítulo são a metodologia de implementação de controladores difusos em VHDL, e seu uso como periféricos na implementação de sistemas de *co-projeto de hardware-software*, e suas vantagens em termos do tempo e velocidade de processamento quando comparado com o desempenho de GPPs. Além disto, a aplicação desta funcionalidade sob um problema específico como é o controle de estabilidade de um pêndulo duplo invertido interatuado atuando no controle de equilíbrio de um robô ciclista.

Foram feitas diferentes comparações de desempenho para implementação do controlador difuso, dando como resultado final que a implementação do sistema em *hardware* com um *clock* de 50 MHz é mais rápida que as implementações em *software* (mesmo rodando, no caso do Matlab, em um processador Core(TM)2 Duo CPU com um *clock* de 1.8 GHz).

8 CONCLUSÕES E SUGESTÕES PARA TRABALHOS FUTUROS

Este trabalho propõe o modelo dinâmico de um pêndulo duplo invertido interatuado (*Acrobot*), adaptado sobre a dinâmica que descreve um robô ciclista quando as velocidades de deslocamento são iguais a zero. A análise do tema foi baseada na instabilidade própria do sistema e na movimentação que o ciclista pode exercer sob o seu tronco para contrabalançar o peso.

Foi desenvolvido um projeto conceitual do robô ciclista, o qual consiste na primeira fase na metodologia de projeto do robô. O projeto conceitual permitiu definir a funcionalidade do robô ciclista, sensores e atuadores baseados na experiência de trabalhos prévios, que permitiram sugerir dados técnicos. A construção de um protótipo real do robô ciclista como protótipo de provas é uma alternativa interessante visando a geração de algoritmos de controle, e sua implementação em paralelo com o algoritmos de equilíbrio fundamental para o seu deslocamento.

Desta forma, o projeto conceitual do robô ciclista permitiu o estudo de efeitos físicos envolvidos na dinâmica da bicicleta na medida em que as velocidades começam a ser maiores, e o controle mediante a movimentação do guidão. Embora estes efeitos não sejam considerados nesta implementação, são tidos em conta na análise respectiva.

O modelo matemático encontrado para o sistema em estudo permitiu duas coisas: (a) a simulação da planta usando o método numérico de Runge Kutta de 4ª ordem e (b) o projeto dos controladores baseados em técnicas de representação de estados e controle moderno tais como LQR e RE, através da linearização das equações ao redor do ponto de operação. A simulação da planta foi fundamental na medida em que permitiu o desenvolvimento dos controladores, incluindo os difuso, neural e neuro-difuso.

O estudo do sistema e a utilização de diferentes técnicas de controle permitiram comparar técnicas e metodologias de projeto, assim como tratar aspectos de desempenho e requisitos para cada um dos controladores estudados. No contexto das implementações foram usadas várias ferramentas computacionais, tais como Matlab, Xfuzzy, Virtual Builder, XPS e Modelsim. Foi observado que todos os controladores são baseados na escolha empírica de seus parâmetros em relação com a resposta sobre o sistema. Todos eles apresentaram um

comportamento satisfatório do ponto de vista de estabilidade. O controlador LQR apresentou uma melhor resposta em termos do tempo de estabilização, e o controlador difuso apresentou o menor esforço de controle.

Mediante o desenvolvimento da metodologia para o controle dos algoritmos difusos, foi experimentada a dificuldade de trabalhar sistemas difusos com múltiplas entradas. O sistema neuro-difuso, como o apresentado neste trabalho, permitiu o desenvolvimento de um sistema de quatro entradas e uma saída. O mesmo foi treinado a partir do aprendizado das redes neurais, mediante o uso de exemplos sem a necessidade de se preocupar pelo ajuste das 81 regras difusas, como conseqüência do uso de três funções de pertinência para cada entrada.

A realização do protótipo virtual foi importante para a visualização gráfica que é um objetivo a longo prazo, e como se pretende relacionar a funcionalidade do pêndulo com o problema de estudo. O desenvolvimento de programas usando um simulador não representa situações de risco reais diminuindo custos de operação. Além disto, os algoritmos podem ser aperfeiçoados, depurados, e depois, usados no robô. O *software* pode ser facilmente modificado para um robô com características diferentes, mediante a definição de parâmetros apropriados. Igualmente, é possível definir uma base de dados de ambientes de operação e escolher quando adicionar ou remover um deles.

Foram estudadas as diferentes técnicas de arquiteturas de controle, entre elas a reativa, deliberativa e híbrida. A arquitetura do robô foi definida como reativa em quanto o robô sente pelos seus sensores e reage, mas é tido em conta que a geração de algoritmos de navegação e locomoção sobre o robô, em conjunto com a proposta para o equilíbrio, pode mudar o conceito para uma arquitetura híbrida.

A arquitetura proposta permitiu implementar o controlador difuso na FPGA, usando co-projeto *de hardware-software*. O controlador difuso resultou o método mais adequado para implementação na FPGA pelas suas múltiplas etapas de funcionamento, e por existe uma ferramenta apropriada para isto (o Xfuzzy). Isto permitiu observar em nosso caso de estudo, características importantes do funcionamento da FPGA em quanto ao tempo de processo, consumo de área e recursos.

Os métodos baseados em controle moderno são de caráter mais sistemático, tomando em conta considerações de tipo matemático. O resultado final é a geração do vetor de ganhos, que quando gerado, sua implementação se reduz a multiplicações e somas das variáveis de estado e os vetores de ganhos. A principal limitação destes métodos é que uma má projeção do sistema dinâmico afeta de maneira direta o desempenho dos controladores, levando até à inoperância dos mesmos.

O projeto de sistemas baseados em FPGAs através desta técnica permite a inclusão de outros periféricos de funcionalidade específica, trabalhando em conjunto com os existentes. Um FPGA oferece vantagens importantes na geração de múltiplas unidades de processamento, já que pode ser implementado mais de um processador embarcado.

O estudo da implementação do controlador usando lógica difusa e sua implementação em FPGAs permitiu observar que a utilização de *hardware* reconfigurável, neste tipo de sistemas, faz possível a descentralização e independência dos algoritmos de controle e unidades de processamento principal do sistema robótico, melhorando o desempenho global do sistema.

A principal contribuição deste trabalho foi propor um caso de estudo sobre o qual fossem desenvolvidos e aplicados vários conceitos relacionados com metodologia de projetos, modelagem, projeto e simulação de sistemas, projeto e simulação de controladores baseados em duas grandes áreas como são (a) o controle moderno e (b) a inteligência artificial. Foram estudados aspectos relevantes sobre robótica, implementação física sobre dispositivos reconfiguráveis e, por fim, uma exploração de múltiplas possibilidades de implementação de algoritmos de controle sobre FPGAs.

8.1 SUGESTÕES PARA TRABALHOS FUTUROS

Como trabalhos futuros são propostos:

a) algoritmos de otimização e escolha de parâmetros sistematizados.

A busca do vetor de ganhos, no caso do controlador de realimentação de estados ser feita através da alocação manual dos pólos no semi-plano complexo esquerdo, a possível implementação de um algoritmo de busca destes pólos de uma forma sistêmica.

A observação dos parâmetros sobre-passo e tempo de estabilização podem permitir a escolha dos valores de alocação de pólos apropriada. Uma forma intuitiva supõe o uso de ciclos aninhados, incrementando os valores de cada ciclo em uma quantidade determinada, tomando em conta que os valores devem ser negativos e que uma escolha afastada destes pode gerar instabilidade.

Usar outra forma de otimização que pode ser feita através da inteligência artificial mediante o uso de algoritmos genéticos (Ekisin e Erol, 1996; Kumar e Garg, 2004). Os algoritmos genéticos são comumente usados para otimização, mas também podem ser implementados como algoritmos de controle mesmo. Isto vai trazer como conseqüência um alto custo computacional. Uma forma de tentar diminuir o custo é o uso de *hardware* para encontrar esses valores, ou seja, a implementação em *hardware* destes algoritmos de busca.

Uso de algoritmos genéticos pode ser interessante também no caso do controle ótimo, na busca e sintonização dos parâmetros das matrizes Q e R para sintonização do controlador LQR.

b) A forma em que as redes neurais operam (distribuição de processamento em paralelo) apresentam uma interessante aplicação de implementação sobre os FPGAs. Pode ser feito um estudo mais afundo do uso e implementação destes algoritmos nos dispositivos reconfiguráveis, tendo em conta o alto grau de custo computacional que eles envolvem principalmente no treinamento.

c) O estudo de sistemas baseados em métodos discretos e tempo real. Outras técnicas de controle podem ser estudadas baseadas em técnicas discretas. Uma aplicação interessante tem a ver com a utilização de bibliotecas que permitem a implementação do computador como planta em tempo real. Matlab conta com um aplicativo para fazer simulações em tempo real chamado *Real Time Windows Target*. Esta ferramenta além do controle em tempo real permite simulação de plantas físicas tais como turbinas de avião e modelagem de sistemas físicos. Mas é preciso a utilização de Simulink, que acompanha o Matlab ao invés de código fonte como neste trabalho.

d) Construção do protótipo real. Foi feito um estudo das possibilidades de configuração, tipos de sensores e atuadores partindo de experiências prévias, além de baterias, desenhos e implementação de controladores em FPGAs, e metodologia de projeto especificada neste trabalho. O próximo passo supõe a construção de um protótipo real. A descentralização da unidade de controle do computador é o primeiro passo para a criação de periféricos que diligenciem as tarefas de aquisição de dados dos sensores, e gerem os sinais de controle necessários para o atuador, no caso do controle de equilíbrio. A plataforma proposta contém outros atuadores para a marcha do robô e para o controle mediante o guidão.

e) Outro trabalho pode ser orientado no sentido de estudar os efeitos de hibridizar controladores do tipo LQR e ANFIS, LQR e Neural, ou LQR e Difuso, e os seus comportamentos sob o sistema *Acrobot*. Algumas vantagens podem ser obtidas no tratamento de sistemas não-lineares ou com fortes perturbações de carga.

f) Por último, estudo de outras técnicas mais atuais de comunicação (envio e recepção de dados) com o computador, radio freqüência, Bluetooth, wireless etc., para tarefas de monitoramento, supervisão e controle.

REFERÊNCIAS

Almeida, F. J. (2000). *Estudo e Escolha de Metodología para o Projeto Conceitual.* Revista de Ciência e Tecnología, Vol. 8, No. 16, pp. 31-42.

Aracil, J., Gordillo, F. (2005). El péndulo invertido: Un desafio para el control no lineal. Revista Iberoamericana de Automática e Informática Industrial. Vol. 2, No. 2, pp. 8-19.

Åström, K. J., Klein, R. E., Lennartsson, A. (2005). *Bicycle Dynamics and Control: Adapted Bicycles for Education and Research*. The IEEE Control Systems Magazine, Vol. 25, No. 4, pp. 26-47.

Aadeca (2009). Site acessado em 06/2009.
Disponível em http://www.aadeca.org/articulos/historia_autom_4.php.

Bastos, S. C. (2008). *Planejamento de Trajetória para um Robô Móvel com Duas Rodas Utilizando um Algoritmo A-Estrela Modificado*. Tese de Mestrado em Ciências em Engenharia Elétrica, UFRJ, Rio de Janeiro, Brasil.

Bishop, R. (1997). *Modern Control Systems Analysis & Design Using Matlab,* Addison Wesley, Austin Texas, pp. 49-129.

Boone, G. (1997). *Minimum-time Control of the Acrobot.* The IEEE International Conference on Robotics and Automation, Albuquerque, New Mexico USA, Vol. 4, pp. 3281 – 3286.

Boyce, W., Diprima, R. (2001). *Equações Diferenciais Elementares e Problemas de Valores de Contorno*, sétima edição, LTC Editora, Rio de Janeiro, RJ, Brasil, pp. 225-244.

Brockett, R., Hongyi, L. (2003). *A Light Weight Rotary Double Pendulum: maximizing the Domain of Attraction.* In: The 42nd IEEE Conference on Decision and Control, Maui, Hawaii, USA, Vol. 4, No. 9, pp. 3299-3304.

Brooks, R. (1986). *A Robust Layered Control System for a Mobile Robot*. In: IEEE Journal of Robotics and Automation, Vol.2 , pp. 14–23.

Brown, S. C., Passino K. M. (1997). *Intelligent Control for an Acrobot.* Journal of Intelligent and Robotic Systems, Netherlands, Vol. 18, No. 3, pp 209 – 248.

Brox, M. and Sánchez-Solano, S. (2006). *Development of IP Modules of Fuzzy Controllers for the Design of Embedded Systems on FPGAs.* In: 16th International Conference on Field Programmable Logic and Applications (FPL2006), Madrid, Spain, pp 1-2**.**

Britto, R. (2008). *Uma Arquitetura Distribuída de Hardware e Software para Controle de um Robô Móvel Autônomo.* Dissertação de Mestrado, Programa de Pós-Graduação em Engenharia Elétrica e de Computação, Universidade Federal do Rio Grande do Norte, Natal, RN.

Carvallo, M. E. (1900). *Théorie du Mouvement du Monocycle et de la Bicyclette*. In: Journal de L'Ecole Polytechnique, Series 2, Part 1, Vol. 5, pp. 119-188, Part 2, Vol. 6, pp. 1-118.

Castro, C. Y. E., Aline, S. de P., Olavo, M. de C., Britto, W. (2008). *Estudo Comparativo de Estratégias de Controle Aplicadas a um Pêndulo Invertido*. In: *VIII Congresso Nacional de Engenharia Mecânica e Industrial,* Brasilia, DF, Brasil.

Castro, C. Y. E., Llanos, C. H., Britto, W., Coelho, L. S. (2009a). *Controle Inteligente Aplicado ao Problema do Pêndulo Duplo Interatuado*. In: IX Simpósio Brasileiro de Automação Inteligente (SBAI). University of Brasilia, Brasília, DF, Brasil.

Castro, C. Y. E., Llanos, C. H., Britto, W., Coelho, L. S. (2009b). *A Study of Three Control Approaches for the Cyclist Robot Problem*. In: 20th International Congress of Mechanical Engineering (COBEM). Gramado, RS, Brasil.

Castro, C. Y. E., Llanos, C. H., Britto, W., Coelho, L. S. (2009c). *Fuzzy Control for Cyclist Robot Stability using FPGAs.* In*:* International IEEE Conference on ReConFigurable Computing and FPGAs. Cancun, México.

Chen, C.T. (1999). *Linear System Theory and Design,* The Oxford Series in Electrical and Computer Engineering, Oxford, New York, USA, 3rd edition.

Coelho, L. S., Almeida O. M. e Coelho A. A. R. (2003). *Projeto e Estudo de Caso da Implementação de um Sistema de Controle Nebuloso.* Revista SBA Controle & Automação, Vol. 14, No. 1, pp. 20-29.

Cole, P., Seaweed Syst. Inc., Woodinville, WA. (2005). *Open GL ES SC – open standard embedded graphics API for safety critical applications*. In: 2005 IEEE/AIAA 24th Digital Avionics Systems Conference (DASC), Hyatt Regency Crystal City, Washington D.C., Vol. 2, pp. 8.

Copetti, C. T., Coelho L. S. e Coelho A. A. R. (2007). *Controle Nebuloso Adaptativo por Modelo de Referência: Projeto e Aplicação em Sistemas Não Lineares*. Revista Controle & Automação, Vol. 18, No. 4, pp. 479-489.

Cossalter, V. (2009). Site acessado em 10/05/2009.
Disponível em http://www.torremoto.com/ini.htm.

Dias, S., Medeiros, F., Alsina, P., Medeiros, A., (2006). *Arquitetura Distribuída de Hardware e Software Aplicada a uma Plataforma Robótica Autônoma*. In: XXVI Congresso da SBC, III Encontro de Robótica Inteligente ENRI, Brasil, pp. 46 – 54.

Drummond, A., Oliveira, K., Bauchspiess A. (1999). *Estudo do Controle de Pêndulo Inverso sobre carro utilizando Rede Neural de Base Radial*. The IV Brazilian Conference on Neural Networks – IV Congreso Brasileiro de Redes Neurais, São Jose dos Campos, SP, Brasil, pp. 320-325.

Eksin, I., Erol, O. K. (1996). *Design of optimum fuzzy controller using genetic Algorithm.* In: The IEEE Electrotechnical Conference, MELECON '96, 8th Mediterranean, Bari, Italy, Vol. 1, pp 186-190.

Fajans, J. (2000). *Steering in Bicycles and Motorcycles*, American Journal of Physics, Vol. 68, Issue 7, pp. 654 – 659.

Feitosa, J., Alsina, P., Ferneda E. (1999). *Posicionamento de um Pêndulo Invertido usando Algoritmos Genéticos.* In: SBA Controle e Automação, Vol. 10, No. 1, pp. 31-38.

Ferreira, da S. C. H. (2002). *Modelagem e Aplicação de Técnicas de Controle Moderno a Sistemas Reguladores de Velocidade e Tensão de Máquinas Síncronas de pequenas Centrais Hidroelétricas*. Tese de Mestrado apresentada ao Programa de Pós-Graduação em Engenharia Química da Universidade Federal de Uberlândia, Uberlândia, MG.

Floreano, D., Godjevac, J., Martinoli, A., Mondada, F. e Nicoud, J.-D. (1998). Design, Control, and Applications of Autonomous Mobile Robots, Kluwer Academic Publishers.

Franklin, G., Powell, J., Naeini, A. (2002.) *Feedback Control of Dynamic Systems*, Prentice Hall.

Freescale Semiconductor, acessado em 07/2009

Disponível em http://www.freescale.com/files/sensors/doc/data_sheet/MMA1260D.pdf?fsrch=1.

García, P. M., Alvarez, A. J. (2005). *Instrumentción Electrónica*, Thomson Editores Spain, 2da edición.

Grossman, S. (2007). *Álgebra Lineal,* McGraw Hill, México, sexta edição, pp 217.

Geuntaek, K., Wonchang, L. (1998). *Design of TKS Fuzzy Controller Based on TKS Fuzzy Model Using Pole Placement*. In: IEEE World Congress on Computational Intelligence, Vol. 1, Issue, p.p. 246 – 251.

Hartenstein, R. (2006). *Why we need Reconfigurable Computing Education*. In: 1st International Workshop on Reconfigurable Computing Education, Karlsruhe, Germany.

Haykin, S. (2001). *Neural Networks a Comprehensive Foundation*, Segunda Edição, Pearson Prentice Hall, India.

Heinen, M. R. (2007). *Controle Inteligente do Caminhar de Robôs Móveis Simulados*. Tese de Mestrado, Programa Interdisciplinar de Computação Aplicada, Ciências Exatas e Tecnológicas, Universidade do Vale do Rio dos Sinos, São Leopoldo, RS.

Heinen, F. J. (2002). *Sistema de Controle Híbrido para Robôs Móveis Autônomos.* Tese de Mestrado, Mestrado em Computação Aplicada, Ciências Exatas e Tecnológicas, Universidade do Vale do Rio dos Sinos, São Leopoldo, RS.

Hung, V. V., Esfandiari, R. S. (1997). *Dynamic Systems: Modeling and Analysis*, McGraw-Hill, Singapore, pp. 108-155.

Innovação Tecnológica, acccesado em 08/2009

Disponível em http://www.inovacaotecnologica.com.br/noticias/noticia.php?artigo=robo-vira-piloto-testes-motos-esportivas&id=010180090721.

Inoue, A., Deng M., Tanabe, T. (2006). *Practical Swing-up Control System Design of Cart-type Double Inverted Pendulum*. In: The IEEE Control Conference, Harbin, China, Vol. 7, No. 14, pp. 2141-2146.

Jang, J. S. (1993). *ANFIS: Adaptive-Network-Based Fuzzy Inference System*. In: The IEEE Transaction on Systems, Man, and Cybernetics, Vol. 23, No. 3, pp. 665-685.

Jesman, R., Martinez, F. and Saniie, J. (2008). *Microblaze Tutorial Creating a Simple Embedded System and Adding Custom Peripherals Using Xilinx EDK Software Tools.* In: Embedded Computing and Singal Processing Laboratory – Illinois Institute of Technology. Acessado em 03/2009.

Disponível em http:/ecasp.ece.iit.edu.

Jingang, Y. D. S., Levandowski A. and Jayasuriya S. (2006). *Trajectory Tracking and Balance Stabilization Control of Autonomous Motorcycles*. In: IEEE International Conference on Robotics and Automation, Orlando Fl, USA, pp. 2583-2589.

Jones, D. E. H. (1970). *The Stability of the Bicycle*, Physics Today, London, Vol. 23, Issue 3, pp. 34 – 40.

Junichiro, Y., Shin I., Masa-aki S. (1999). *Application of Reinforcement Learning to Balancing of Acrobot*. In: IEEE International Conference on Systems, Man, and Cybernetics, Tokio, Japan, Vol. 5, pp. 516-521.

Jyh-Shing, R., Jang, Chuen-Tsai S., Eiji M. (1997). *Neuro Fuzzy and Soft Computing*. Prentice Hall, pp. 199 – 250.

Kawakami, L. (2005). *Sobre a Estabilização Global de Sistemas não Lineares via Equação de Riccati Dependente do Estado*. Dissertação de Mestrado em Ciências em Engenharia Elétrica, UFRJ, Rio de Janeiro, Brasil.

Kamio, T., Soga, S., Fujisaka, H., Mitsubori, K. (2004). *An Adaptive State Space Segmentation for Reinforcement Learning Using Fuzzy-ART Neural Network*. In: The 47th IEEE International Midwest Symposium on Circuits and Systems, Hiroshima, Japan, Vol. 3, pp. iii-117-20.

Kelly, D. J., Burton, P. D., Rahman, M. A. (1994). *The Application of a Neural-Fuzzy Logic Controller to Process Control*. In: First International Joint Conference of the North American Fuzzy Information Processing Society Biannual Conference, San Antonio, TX, USA, pp. 235-236.

Klein, F. e Sommerfeld, A. (1910). *Über die Theorie des Kreisels*, ch. IX, Section 8, pp. 863-884.

Kim, D. (2000). *An Implementation of Fuzzy Logic Controller on the Reconfigurable FPGA Systems*. In: The IEEE Transactions on Industrial Electronics, Vol. 47, No. 3, pp. 703 – 715.

Kobori, N., Kenji S., Pitoyo H. and Shuji H. (2002). *Learning to Control a Joint Driven Double Inverted Pendulum Using Nested Actor Critic Algorithm*. In: The 9th International Conference on Neural Information Processing (ICONIP'O2), Singapore, Vol. 5, pp 2610 – 2614.

Kumar, M., Garg, P. (2004). *Intelligent Learning of Fuzzy Logic Controllers via Neural Network and Genetic Algorithm*. In: JUFSA, Japan-USA Symposium on Flexible Automation, Denver, Co, USA.

Lacerda, W. (2002). *Implementação de Sistemas Fuzzy em Hardware*. Programa de Pós-graduação em Engenharia Elétrica Universidade Federal de Minas Gerais, Belo Horizonte, Brasil, pp. 1- 45.

Leblanc, P., Ober, I. (2000). *Comparative Case Study in SDL and UML*. In: 33rd International Conference on Technology of Object-Oriented Languages, Mont-Saint-Michel, France, pp. 120-131.

Leonor, M., Neves, M. (2004). *Construção de um Pêndulo Invertido sobre um Robô Móvel Controlado com Executivo SHaRK*. Revista do Detua, Vol. 4., No. 2, pp 270-272.

Luger, F. G. (2004). Inteligência Artificial, Estruturas e Estratégias para a Resolução de Problemas Complexos, 4ª Edição, Bookman, P. Alegre, RS.

Lukasz W., Stephan C., Rick M. (2008). *A Small Spiking Neural Network with LQR Control Applied to the Acrobot*. The Neural Comput & Applic, Vol. 18, No. 4, pp. 369-375

Mahmoud, A., Manzoul and Jayabharathi, D. (1995). *FPGA for Fuzzy Controller.* In: The IEEE Transactions on Systems, Man and Cybernetics, Vol. 25, No. 1, pp. 213-216.

Meijaard, J. P., Papadopoulos J.M., Ruina, A., Schwab A. L. (2007). *Linearized Dynamics Equations for the Balance and Steer of a Bicycle: a Benchmark and Review*, The Royal Society A., Vol. 463, pp. 1955-1982.

Miranda, P., Barbosa M., Konbauer D. (2003). *Sistema de Controle Difuso de Mandani Aplicações: Pêndulo Invertido e outras*. Trablaho de Graduação, Departamento de Computação e Estatística, Centro de Ciências Exatas e Tecnologia, Universidade Federal de Mato Grosso do Sul.

Michini, B. (2006). Autonomous Stability Control of a Moving Bicycle, III Version, 16.621 Spring.

Miura, N., Sugiura, M., Takashi, M., Moridaira, T., Miyamoto, A., Kuroki, Y., Kawashima, R. (2008). *An Advantage of Bipedal Humanoid Robot of the Empathy generation: A neuroimaging study*. In: The IEEE/RSJ International Conference on Intelligent Robots and Systems, Nice, France, pp 2465-2470.

Morais, M. H., Muralikrishna, A., Bravo, R., Guimarães, L. N. (2005). *Um Controlador Nebuloso Aplicado ao Problema do Pêndulo Invertido*. In: XXVIII Congresso Nacional de Matemática Aplicada e Computacional, São Paulo, Brasil.

Murray, R. M., Hauser, J. (1991). *A Case Study in Approximate Linearization: The Acrobot Example.* In: American Control Conference, San Diego, Ca, USA.

Nascimento, J. e Cairo L. (2000). Inteligência Artificial em Controle e Automação, Edgar blucher Ltda, São Paulo, pp. 58.

Narendra, K.S. e Parthasarathy K. (1990). *Identification and Control Dynamical Systems Using Neural Networks.* In: The IEEE Trans. On Neural Networks, Vol. 1, pp. 4-27.

Vieira, S. E., Nakahati, Y. W. (2007). *Desenvolvimento de um sistema supervisorio para experimentos de controle e implementação de um Pendubot*. Trabalho de Graduação, Engenharia Mecatrônica Universidade de Brasília, Brasília, DF.

Ogata, K. (2003). *Engenharia de Controle Moderno*, 4ª ed, Prentice Hall of Brazil.

Ozana, S. (2008). *Visualization of Active Suspension by Robust Controller in Virtual Reality Toolbox.* In: The 19th IEEE International Conference on System Engineering, Las Vegas, NV, Vol. 19, pp. 38-42.

Papadopoulos, J. M. (1987). *Bicycle Steering Dynamics and Self-stability: A Summary Report on Work in Progress.* In: Technical Report, Cornell Bicycle Research Project, Cornell University New York, USA, pp. 1-23.

Passos, G. W. (2006). *Simulação e Controle de um Pêndulo Duplo Invertido*. Trabalho de Graduação, Engenharia Mecatrônica, Universidade de Brasília, Brasília, DF.

Patterson, D. A., Hennessy, J. L. Computer organization and design: the hardware/software interface, 4th ed. Massachusetts: Morgan Kaufmann, c2009. XXV, 703, [196] p.

Pieri, E. R. (2002). *Curso de Robótica Móvel*. Programa de Pós-graduação em Engenharia Elétrica. Universidade Federal de Santa Catarina, Florianópolis, SC.

Sánchez, S., Cabrera ,A., Brox, M. and Gonzalez, A. (2006). *Controladores Difusos Adaptativos como Modulos de Propriedade Intelectual para FPGAs.* In: XII Workshop IBERCHIP (IWS-2006), Costa Rica.

Santos, R. T., Nievola, J. C., Freitas A. A., Lopes, H. S. (1999). *Extração de Regras Neurais via Algoritmos Genéticos*. In: The IV Brazilian Conference on Neural Networks, Brazil, pp. 158-163.

Soarez, R. (1999). *Curso de Robots Móviles.* Universidad de San Carlos III de Madrid, España.

Spong, M. W. (1995). *The Swing up Control Problem for the Acrobot.* In: The IEEE Control Systems Magazine*,* Vol. 15, No. 1, pp 49 – 55.

Tudelf (2009). *History of Bicycle Steer and Dynamics Equations,* acessado em 01/2009. Disponível em http://www.tudelft.nl/live/pagina.jsp?id=25d19ab8-47e8-4427-831c-4d0680fd2c2c&lang=en.

Tanaka, Y., Murakami T. (2004). *Self Sustaining Bicycle Robot with Steering Controller.* In: The 8th IEEE International Workshop on Advance Motion Control, Kawasaki Japan, pp. 193-197.

Tanenbaum, A. S. (2006), Organização Estruturada de Computadores, 5a edição, Prentice-Hall.

Takashima, S. (1990). Dynamic *Modeling of a Gymnast on a High Bar*. In: The IEEE International Workshop on Intelligent Robots and Systems (IROS'90), Ibaraki, Japan, Vol. 2, pp 955-962.

V-Realm™ Builder (2009). *User's Guide and Reference*, acessado em 02/2009.

Disponível em http://metalab.uniten.edu.my/~farrukh/vrml/user_guide.pdf.

Viguria, A., Prieto, A., Fiacchini, M., Cano, R., Rubio, F. R., Aracil, J., Canudas-de-wit C. (2006). Desarrollo y experimentación de un Vehículo basado en péndulo invertido (PPCAR). *Revista Iberoamericana de Automática e Informática Industrial*. Vol. 3, No. 4, pp. 54-63.

Villa, P. L. (2007). *Construção de Robôs Móveis*. Tutorial conceitos básicos de construção, Cascavel, Paraná.

Whipple (1899). The Stability of the Motion of a Bycicle. *The Quarterly Journal of pure and Applied Mathematics*, London, Vol. XXX, pp. 313-384.

Xilinx Corporation, (2005). *Spartan 3E FPGA family: Complete Data Sheet*, accesado em 08/2008.

Disponível em http://www.xilinx.com/support/documentation/data_sheets/ds312.pdf

Xilinx Corporation, (2009). http://www.xilinx.com, acessado em 06/2009.

Yasunobu, S., Takayuki, I. (2004). *Swing Up Intelligent Control of Double Inverted Pendulum Based on Human Knowledge.* In: The IEEE SICE Annual Conference, Sapporo, Japan, Vol. 2, pp. 1869-1873.

Yasuhito T., Toshiyuki M. (2004). *Self Sustaining Bicycle Robot with Steering Controller*. In: The 8th IEEE International Workshop, Kawasaki, Japan, pp. 193-197.

Zadeh, L. A. (1965), Fuzzy Sets. Information and Control, Vol. 8, pp. 338-353.

Zhihua L., Ling Z., Huili Z. (2008). *Solving PDE Models in Modelica*. In: Symposium IEEE Information on Science and Engineering, Shangai, China, Vol. 1, pp 53-57.

Zheng, Y. F. (1993). *Recent Trends in Mobile Robots.* World Scientific Series in Robotics and Automated Systems, Philadelphia, U.S.A, Cap. 6, Vol. 11, pp. 215.

Zong-Mu, Y., Kuei –Hsiang, L. (2004). A systematic approach for designing multistage fuzzy control systems. *Fuzzy Sets and Systems Elsevier,* pp. 251-273.

APÊNDICES

A. APÊNDICE A – Código Fonte Matlab para Simulador do Pêndulo Duplo Invertido Interatuado

O programa principal fica representado pelas seguintes linhas de código.

```
yin(1)=th1(i);
yin(2)=w1(i);
yin(3)=th2(i);
yin(4)=w2(i);

yout=runge_kutta_4ta(t(i),yin,h,N,M1,M2,L1,L2,g,tq);

th1(i+1)= yout(1);
w1(i+1) = yout(2);
th2(i+1)= yout(3);
w2(i+1) = yout(4);
```

A função principal ***runge_kutta_4ta*** recebe um vetor de entrada com os valores das variáveis posição e velocidade para cada barra *yin*, além de outros parâmetros necessários para a encontrar a solução das equações diferencias como as massas *M1, M2,* os comprimentos *L1,L2,* a gravidade *g*, os valor presente do torque *tq*, a constante de passo *h*, o tempo atual *t*, o número de equações para ser resolvidas *N*. A função retorna um vetor de saída *yout* com os valores das variáveis no estado seguinte no tempo $t=t+h$.

O código da função é mostrado como segue:

```
function yout = runge_kutta_4ta(xin,yin,h,N,M1,M2,L1,L2,g,tq)

%Definição de arreglos ************************************************
dydx=zeros(1,N);
dydxt=zeros(1,N);
yt = zeros(1,N);
k1 = zeros(1,N);
k2 = zeros(1,N);
k3 = zeros(1,N);
k4 = zeros(1,N);
%*************************************************************************
hh = 0.5*h;
xh = xin + hh;
%*************************************************************************
%primeiro passo
dydx=derivadas(xin,yin,dydx,M1,M2,L1,L2,g,tq);
for i=1:N
   k1(i)=h*dydx(i);
   yt(i)=yin(i) + 0.5*k1(i);
end
%*************************************************************************
%segundo passo
dydxt=derivadas(xh,yt,dydxt,M1,M2,L1,L2,g,tq);
for i=1:N
   k2(i)=h*dydxt(i);
   yt(i)=yin(i) + 0.5*k2(i);
end
%*************************************************************************
%terceiro passo
dydxt=derivadas(xh,yt,dydxt,M1,M2,L1,L2,g,tq);
```

```
for i=1:N
   k3(i)=h*dydxt(i);
   yt(i)=yin(i) + k3(i);
end
%*************************************************************************
%quarto passo
dydxt=derivadas(xin+h,yt,dydxt,M1,M2,L1,L2,g,tq);
for i=1:N
   k4(i)=h*dydxt(i);
   yout(i)= yin(i) + k1(i)/6 + k2(i)/3 + k3(i)/3 + k4(i)/6;
end
%*************************************************************************
```

Como mostrado na seção 4.5 o algoritmo descreve os passos que são implementados nesta função. Esta função utiliza para o cálculo das derivadas a função *derivadas*.

```
 function [dydx,h2,fi1,fi2]=derivadas(xin,yin,dydx,M1,M2,L1,L2,g,tq)

%*************************************************************************
%alguns parâmetros

lc1 = (L1/2);
lc2 = (L2/2);
Ic1 = (1/3)*M1*(L1^2);
Ic2 = (1/3)*M2*(L2^2);
B1  = 0.05;
B2  = 0.05;

d1= M1*(lc1^2)+ M2*(L1^2)+ Ic1;
d2= M2*L1*lc2*cos(yin(3)-yin(1));
d3= M2*L1*lc2*sin(yin(3)-yin(1));
d4= (M1*lc1 + M2*L1)*g*L1*sin(yin(1));
d6= M2*(lc2^2)+Ic2;
d8= M2*g*lc2*sin(yin(3));

%*************************************************************************
dydx(1)=yin(2);

%*************************************************************************
%equação W1'

den=(d6*d1)-(d2^2);

dydx(2) = -(yin(4)^2)*d3*d6 + (yin(2)^2)*d3*d2 - yin(2)*(B1*d6-B2*d2) - B2*d2*yin(4) + d4*d6 + d8*d2
+ tq*(d2-d6);
dydx(2) = dydx(2)/den;

%*************************************************************************
dydx(3)=yin(4);

%*************************************************************************
%equação W2'

dydx(4) = -(yin(4)^2)*d2*d3 + (yin(2)^2)*d3*d1 - yin(2)*(B1*d2-B2*d1) - B2*d1*yin(4) + d4*d2 + d8*d1
+ tq*(d1-d2);
dydx(4)=  dydx(4)/den;
%*************************************************************************
```

Este código permite visualizar em Matlab uma janela com o comportamento do pêndulo duplo interatuado.

```
%Constantes fixas *********************************************************
N      = 4;                %Número de equações pra ser resolvidas
g      =  9.8;             %ação da gravidade m/s^2
L1     = 0.4;              %comprimento link 1
L2     = 0.4;              %comprimento link 2
M1     = 1;                %massa M1 link 1 Kg
M2     = 1;                %massa M2 link 2 Kg
tq     = 0;                %torque N*m
NSTEP   = 48001;           %Número de interações que define também a precisão o valor h
%Condições iniciais ********************************************************
TMIN =      0;             %Tempo de arranque em segundos
TMAX =120;                 %Tempo de final em segundos
TH10 =   -15;              %Angulo teta1 inicial em graus
TH20 =    15;              %Angulo teta2 inicial em graus
W10  =       0;            %Velocidade angular inicial 1 em graus/segundo
W20  =       0;            %Velocidade angular inicial 2 em graus/segundo

%***************************************************************************
%Definição de variáveis

yin  = zeros(1,N);
yout = zeros(1,N);
h = (TMAX - TMIN) / (NSTEP - 1);

%Tamnaho de passo pra integração ***************************************
t=zeros(1,NSTEP);

for i=1:NSTEP
   t(i)=TMIN + h*i;
end

%Conversão de ângulos pra radianos dos valores iniciais
th1(1)= (TH10)*pi/180;
w1(1) = W10*pi/180;
th2(1)= TH20*pi/180;
w2(1) = W20*pi/180;
%***************************************************************************
%Grafica as coordenadas iniciais

xc=[0,L1*sin(th1(1)),L1*sin(th1(1))+L2*sin(th1(1)-th2(1))];
yc=[0,L1*cos(th1(1)),L1*cos(th1(1))+L2*cos(th1(1)-th2(1))];
hob=plot(xc,yc,'b.-');
axis([-1 1 -1 1])
axis square
grid on
set(hob,'EraseMode','xor','MarkerSize',18);

taux(1)=t(1);

%Programa principal ********************************************************
for i=1:7005 %NSTEP

        yin(1)=th1(i);
        yin(2)=w1(i);
        yin(3)=th2(i);
        yin(4)=w2(i);
```

```
        yout=runge_kutta_4ta(t(i),yin,h,N,M1,M2,L1,L2,g,tq);

        th1(i+1)= yout(1);
        w1(i+1) = yout(2);
        th2(i+1)= yout(3);
        w2(i+1) = yout(4);

        x_l1(i)=L1*sin(yout(1));
        x_l2(i)=L1*sin(yout(1))+ L2*sin(yout(3));
        y_l1(i)=L1*cos(yout(1));
        y_l2(i)=L1*cos(yout(1))+ L2*cos(yout(3));

        v1=[0,x_l1(i),x_l2(i)];
        v2=[0,y_l1(i),y_l2(i)];

        set(hob,'XData',v1,'YData',v2);

  drawnow;
end
%*************************************************************************
```

B. APÊNDICE B – Código Fonte XPS para Intercambio de Dados entre a Planta e o Processador Embarcado.

O seguinte código recebe da planta simulada em Matlab os dados das variáveis de estado, passa esses valores para os módulos Xfuzzy 01 e Xfuzzy 02, e retorna o valor da variável de controle para a planta em Matlab.

```
#include "xparameters.h"
#include "mb_interface.h"
#include "xbasic_types.h"
#include "xuartlite_l.h"
#include "xgpio_l.h"

Xuint8 num2ascii(Xuint8);
Xuint8 compara (Xuint8);

int main(void)
{
int i=0,j=0,k=0,resp1=0,resp2=0,sw;
Xuint32 dado1=0, dado2,cons = 0xFFFF0000;
Xuint32 data_to_local_link[] = {0,0,0,0};
Xuint8 buff=0, k1, k2, k3, k4, k5, k6, k7, k8, k9;

// O nibble baixo do elemento entrada corresponde com a entrada in1 e o nibble alto corresponde
// com a entrada in2 do controlador difuso.

 xil_printf("%c[2J",27);

while(1)
 {

buff   = XUartLite_RecvByte(XPAR_RS232_DCE_BASEADDR); //Recebo o primeiro caracter enviado a
                                                      //etiqueta

if (buff==97)
{
  //Recebo o dado que vem com a etiqueta 'a' (97)

                        k5 = XUartLite_RecvByte(XPAR_RS232_DCE_BASEADDR);
                        k5 = compara (k5);

          //O seguinte caracter me indica quantos dados conformam o número que se esta enviando

                        if (k5 == 4)
                        {
                                k1  = XUartLite_RecvByte(XPAR_RS232_DCE_BASEADDR);
                                //Recebo o segundo caracter enviado
                                k2  = XUartLite_RecvByte(XPAR_RS232_DCE_BASEADDR);
                                //Recebo o terceiro caracter enviado
                                k3  = XUartLite_RecvByte(XPAR_RS232_DCE_BASEADDR);
                                //Recebo o quarto caracter enviado
                                k4  = XUartLite_RecvByte(XPAR_RS232_DCE_BASEADDR);
                                //Recebo o quarto caracter enviado
                                XUartLite_SendByte  (XPAR_RS232_DCE_BASEADDR,10);
                                //Convierto os valores ascii para os verdadeiros valores numéricos

                                k1 = compara (k1);
```

```
                k2 = compara (k2);
                k3 = compara (k3);
                k4 = compara (k4);

                resp1 = (k1)*1000 + (k2)*100 + (k3)*10 + (k4)*1;
        }

        if (k5 == 3)
        {

                k1  = XUartLite_RecvByte(XPAR_RS232_DCE_BASEADDR);
                //Recebo o segundo caracter enviado
                k2  = XUartLite_RecvByte(XPAR_RS232_DCE_BASEADDR);
                //Recebo o terceiro caracter enviado
                k3  = XUartLite_RecvByte(XPAR_RS232_DCE_BASEADDR);
                //Recebo o quarto caracter enviado
                XUartLite_SendByte  (XPAR_RS232_DCE_BASEADDR,10);
                //Convierto os valores ascii para os verdadeiros valores numér

                k1 = compara (k1);
                k2 = compara (k2);
                k3 = compara (k3);

                resp1 = (k1)*100 + (k2)*10 + (k3);
        }

        if (k5 == 2)
        {

                k1  = XUartLite_RecvByte(XPAR_RS232_DCE_BASEADDR);
                //Recebo o segundo caracter enviado
                k2  = XUartLite_RecvByte(XPAR_RS232_DCE_BASEADDR);
                //Recebo o terceiro caracter enviado
                XUartLite_SendByte  (XPAR_RS232_DCE_BASEADDR,10);
                //Convierto os valores ascii para os verdadeiros valores numéricos

                k1 = compara (k1);
                k2 = compara (k2);

                resp1 = (k1)*10 + (k2);
        }

        if (k5 == 1)
        {

                k1  = XUartLite_RecvByte(XPAR_RS232_DCE_BASEADDR);
                //Recebo o segundo caracter enviado
                XUartLite_SendByte  (XPAR_RS232_DCE_BASEADDR,10);
                //Convierto os valores ascii para os verdadeiros valores numéricos

                k1 = compara (k1);

                resp1 = k1;
        }

        sw = 1;
        buff = 0;
        XGpio_mSetDataReg(XPAR_LEDS_8BIT_BASEADDR,1,sw);

}
```

```
if (buff==98)
                {
                  k5 = XUartLite_RecvByte(XPAR_RS232_DCE_BASEADDR);
                  k5 = compara (k5);
                  k6 = XUartLite_RecvByte(XPAR_RS232_DCE_BASEADDR);
                  k6 = compara (k6);
                  //O seguinte caracter me indica quantos dados conformam o número que se esta enviando

                        if (k5 == 4)

                        {

                          k1  = XUartLite_RecvByte(XPAR_RS232_DCE_BASEADDR);
                          //Recebo o segundo caracter enviado
                          k2  = XUartLite_RecvByte(XPAR_RS232_DCE_BASEADDR);
                          //Recebo o terceiro caracter enviado
                          k3  = XUartLite_RecvByte(XPAR_RS232_DCE_BASEADDR);
                          //Recebo o quarto caracter enviado
                          k4  = XUartLite_RecvByte(XPAR_RS232_DCE_BASEADDR);
                          //Recebo o quarto caracter enviado
                          //XUartLite_SendByte  (XPAR_RS232_DCE_BASEADDR,10);
                          //Convierto os valores ascii para os verdadeiros valores numéricos

                         k1 = compara (k1);
                          k2 = compara (k2);
                          k3 = compara (k3);
                          k4 = compara (k4);

                          resp2 = (k1)*1000 + (k2)*100 + (k3)*10 + (k4);
                        }

                        if (k5 == 3)
                        {

                        k1  = XUartLite_RecvByte(XPAR_RS232_DCE_BASEADDR);
                        //Recebo o segundo caracter enviado
                        k2  = XUartLite_RecvByte(XPAR_RS232_DCE_BASEADDR);
                        //Recebo o terceiro caracter enviado
                        k3  = XUartLite_RecvByte(XPAR_RS232_DCE_BASEADDR);
                        //Recebo o quarto caracter enviado
                        //Convierto os valores ascii para os verdadeiros valores numéricos

                        k1 = compara (k1);
                        k2 = compara (k2);
                        k3 = compara (k3);

                        resp2 = (k1)*100 + (k2)*10 + (k3);
                        }

                        if (k5 == 2)
                        {

                        k1  = XUartLite_RecvByte(XPAR_RS232_DCE_BASEADDR);
                        //Recebo o segundo caracter enviado
                        k2  = XUartLite_RecvByte(XPAR_RS232_DCE_BASEADDR);
                        //Recebo o terceiro caracter enviado
                        //Convierto os valores ascii para os verdadeiros valores numéricos

                        k1 = compara (k1);
                        k2 = compara (k2);
```

```
                        resp2 = (k1)*10 + (k2);
                        }

                        if (k5 == 1)
                        {

                        k1  = XUartLite_RecvByte(XPAR_RS232_DCE_BASEADDR);
                        //Recebo o segundo caracter enviado
                        //Convierto os valores ascii para os verdadeiros valores numéricos

                        k1 = compara (k1);
                        resp2 = k1;
                        }

                        sw = 2;
                        XGpio_mSetDataReg(XPAR_LEDS_8BIT_BASEADDR,1,sw);


                        dado2 = resp1;                                          //th
                        dado2 = dado2 << 16;                                    //w
                        dado2 = dado2 & cons;
                        dado2 = dado2 + resp2;


                        for (i=0;i<3;i++)
                        {
                                if (k6==1)
                                 {
                                    // esta função espera pelos sinais  -- Interface FSL
                                      microblaze_bread_cntlfsl(dado2,0);
                                      microblaze_bread_cntlfsl(dado1,1);
                                }

        else
                                {
                                      microblaze_bwrite_cntlfsl(dado2,2);
                                      microblaze_bread_cntlfsl(dado1,3);

                                }

                        }


                                        resp1 = 0;
                                        resp2 = 0;


    }

 }

return 1;

}

//// Implementação de funções
Xuint8 compara (Xuint8 dado)
{
```

```
if (dado==48)
  {  dado=0;
            return dado;}

if (dado==49)
            {           dado=1;
              return dado;}

if (dado==50)
            {           dado=2;
              return dado;}

if (dado==51)
            {           dado=3;
              return dado;}

if (dado==52)
            {           dado=4;
              return dado;}

if (dado==53)
            {           dado=5;
              return dado;}

if (dado==54)
            {           dado=6;
              return dado;}

if (dado==55)
            {           dado=7;
              return dado;}

if (dado==56)
            {           dado=8;
              return dado;}

if (dado==57)
            {           dado=9;
              return dado;}

}

Xuint8 num2ascii(Xuint8 aux1)
{

if (aux1==0)                        // Ascci 0
  aux1 = 48;
if (aux1==1)                        // Ascci 1
  aux1 = 49;
if (aux1==2)                        // Ascci 2
  aux1 = 50;
if (aux1==3)                        // Ascci 3
  aux1 = 51;
if (aux1==4)                        // Ascci 4
  aux1 = 52;
if (aux1==5)                        // Ascci 5
  aux1 = 53;
if (aux1==6)                        // Ascci 6
  aux1 = 54;
if (aux1==7)                        // Ascci 7
```

```
   aux1 = 55;
if (aux1==8)                    // Ascci 8
   aux1 = 56;
if (aux1==9)                    // Ascci 9
   aux1 = 57;

return aux1;
}
```

C. APÊNDICE C – Código Fonte de conexão entre Virtual Realm Builder, Matlab, e envío e Recepção de dados com a Spartan 3E.

Este código configura a porta serial COM1 para transmissão e recepção de dados com o FPGA, também configura a transmissão entre os dados das variáveis de estado e o mundo virtual criado em Virtual Realm Builder 2.0.

```
%Constantes fixas ********************************************************
N       =4;             %Numero de equações pra ser resolvidas
g       =9.8;           %ação da gravidade m/s^2
L1      =0.4;           %comprimento link 1
L2      =0.4;           %comprimento link 2
M1      =1;             %massa M1 link 1 Kg
M2      =1;             %massa M2 link 2 Kg
tq      =0;             %torque N*m
NSTEP   =48001;         %Numero de interações que define tambem a precisão o valor h

%Condições iniciais *******************************************************
TMIN =      0;  %Tempo de arranque em segundos
TMAX =     120;  %Tempo de final em segundos
TH10 =     7;  %Angulo teta1 inicial em graus
TH20 =     -15;  %Angulo teta2 inicial em graus
W10  =     0;  %Velocidade angular inicial 1 em graus/segundo
W20  =     0;  %Velocidade angular inicial 2 em graus/segundo

%***************************************************************************
%Definição de variáveis

yin  = zeros(1,N);
yout = zeros(1,N);
h = (TMAX - TMIN) / (NSTEP - 1);

%Tamnaho de passo pra integração ******************************************
t=zeros(1,NSTEP);

for i=1:NSTEP
   t(i)=TMIN + h*i;
end

%Conversão de ângulos pra radianes dos valores iniciais
th1(1)= (TH10)*pi/180;
w1(1) = W10*pi/180;
th2(1)= TH20*pi/180;
w2(1) = W20*pi/180;
%***************************************************************************
%Grafica as coordenadas iniciais

xc=[0,L1*sin(th1(1)),L1*sin(th1(1))+L2*sin(th1(1)-th2(1))];
yc=[0,L1*cos(th1(1)),L1*cos(th1(1))+L2*cos(th1(1)-th2(1))];

hob=plot(xc,yc,'b.-');

axis([-1 1 -1 1])
```

```
axis square

grid on
set(hob,'EraseMode','xor','MarkerSize',18);


%*************************************************************************
%Inicializo o mundo virtual e criação de um objeto para a posterior
%manipulação

myworld = inicial();  % Inicial define os parâmetros iniciais para controlar
                      % o mundo virtual criado no VReal Builder.

%*************************************************************************
taux(1)=t(1);

%Configurando a transmissão serial ----------------------------------------
%
% Create a serial port object.
obj1 = instrfind('Type', 'serial', 'Port', 'COM1', 'Tag', '');
set(obj1, 'BaudRate',128000);
% Create the serial port object if it does not exist
% otherwise use the object that was found.
if isempty(obj1)
   obj1 = serial('COM1');

else
   fclose(obj1);
   obj1 = obj1(1)
end

% Connect to instrument object, obj1.
fopen(obj1);

%Programa principal ******************************************************
for i=1:7005 %NSTEP

        myworld = animacao(myworld,th1(i),th2(i));

        yin(1)=th1(i);
        yin(2)=w1(i);
        yin(3)=th2(i);
        yin(4)=w2(i);

        yout=runge_kutta_4ta(t(i),yin,h,N,M1,M2,L1,L2,g,tq);

        th1(i+1)= yout(1);
        w1(i+1) = yout(2);
        th2(i+1)= yout(3);
        w2(i+1) = yout(4);


        %Enviando os dados para o Microblaze via RS232 ***********************

        aux = mod2xfuzzy ('a',th1(i+1));   % Obtenho o valor correspondente para o Xfuzzy
        stg = num2str(aux);                % Obtenho o string
        num = num2str(length(stg));        % Numero de digitos que vao se enviar
        c=['a',num,stg];
        query(obj1,c);
        aux = mod2xfuzzy ('b',w1(i+1));
```

```
        stg = num2str(aux);
        num = num2str(length(stg));
        c=['b',num,'2',stg];
        data1 = query(obj1,c);
        tq1 = str2double(data1);
        tq1 = mod2xfuzzy ('t',tq1);

        aux = mod2xfuzzy ('a',th2(i+1));  % Obtenho o valor correspondente para o Xfuzzy
        stg = num2str(aux);                % Obtenho o string
        num = num2str(length(stg));        % Numero de digitos que vao se enviar
        c=['a',num,stg];
        query(obj1,c);
        aux = mod2xfuzzy ('b',w2(i+1));
        stg = num2str(aux);
        num = num2str(length(stg));
        c=['b',num,'1',stg];
        data1 = query(obj1,c);
        tq2 = str2double(data1);
        tq2 = mod2xfuzzy ('t',tq2);


         tq = 1.2*tq1 - 0.3*tq2;


        x_l1(i)=L1*sin(yout(1));
        x_l2(i)=L1*sin(yout(1))+ L2*sin(yout(3));
        y_l1(i)=L1*cos(yout(1));
        y_l2(i)=L1*cos(yout(1))+ L2*cos(yout(3));

        v1=[0,x_l1(i),x_l2(i)];
        v2=[0,y_l1(i),y_l2(i)];

        set(hob,'XData',v1,'YData',v2);

        taux(i+1)=t(i+1);
        tqs(i+1)=tq;

  drawnow;

end

%Fechando a comunicacao Serial   ++++++++++++++++++++++++++++++++++++++++++++++++
fclose(obj1);

% Clean up all objects.
delete(obj1);
```

A continuação a função inicial que configura o objeto para envio e recepção de dados entre Matlab e o Virtual Builder

```
% Funcao que receve o objeto myworld, que traz as informacoes do mundo
% virtual robo_bicicleta_2 para sua modificacao.

function myworld = inicial()

myworld=vrworld('robo_bicicleta_2'); %Definindo um objeto tipo vrml
open(myworld);                       %Abro o objeto
```

```
set(myworld,'Description','My first virtual world');
nodes(myworld);
mynodes = get(myworld,'Nodes');      %Obtenho os nodos do mundo virtual
view(myworld);                       %despliego os objetos no simulador
vrdrawnow;
```