UNIVERSIDADE FEDERAL DA PARAÍBA

CENTRO DE CIÊNCIAS E TECNOLOGIA

COORDENAÇÃO DE PÓS-GRADUAÇÃO EM INFORMÁTICA

Escalonador Inteligente de Tarefas para Aplicações Robóticas

José Ricardo da Silva Ferreira

Campina Grande – PB

Julho de 1999

José Ricardo da Silva Ferreira

Escalonador Inteligente de Tarefas para Aplicações Robóticas

Dissertação apresentada à Coordenação de Pós-graduação em Informática – COPIN – da Universidade Federal da Paraíba – UFPB, como parte dos requisitos necessários à obtenção do grau de Mestre em Informática.

Área de Concentração: Ciência da Computação
Linha de pesquisa: Teoria da Computação e Inteligência Artificial

Orientador: Prof. Dr. José Homero Feitosa Cavalcanti

Campina Grande – PB
Julho de 1999

F383e Ferreira, Jose Ricardo da Silva
Escalonador inteligente de tarefas para aplicacoes roboticas / Jose Ricardo da Silva Ferreira. - Campina Grande, 1999.
101 f.

Dissertacao (Mestrado em Informatica) - Universidade Federal da Paraiba, Centro de Ciencias e Tecnologia.

1. Inteligencia Artificial 2. Escalonamento de Tarefas 3. Controle Inteligente 4. Robotica 5. Dissertacao - Informatica I. Cavalcanti, Jose Homero Feitosa II. Universidade Federal da Paraiba - Campina Grande (PB) III. Título

CDU 004.8(043)

ESCALONADOR INTELIGENTE DE TAREFAS PARA APLICAÇÕES ROBÓTICAS

JOSÉ RICARDO DA SILVA FERREIRA

DISSERTAÇÃO APROVADA EM 26.07.1999

Homero Cavalcanti

PROF. JOSÉ HOMERO FEITOSA CACALCANTI, Dr.
Orientador

Pablo Javier Alsina

PROF. PABLO JAVIER ALSINA, D.Sc
Orientador

PROF. EDILSON FERNEDA, Dr.
Examinador

Raimundo Nazareno C. Alves

PROF. RAIMUNDO NAZARENO CUNHA ALVES, D.Sc
Examinador

CAMPINA GRANDE – PB

A minha mãe Hildinair!

Agradecimentos

Este trabalho só foi realizado devido as inúmeras formas de ajuda que eu recebi de muitas pessoas durante toda a minha caminhada até aqui! Após, a conclusão do mesmo é com grande satisfação que eu agradeço principalmente ao meu orientador Prof. José Homero Feitosa Cavalcanti, que mesmo sem me conhecer acreditou em mim, me ofertando tão grandiosa tarefa e permitindo que eu chegasse onde estou hoje. Agradeço também ao Prof. Edilson Ferneda pelo apoio no inicio dessa jornada, ao Coordenador da COPIN Prof. Marcus Sampaio pela orientação inicial no mestrado, aos funcionários da COPIN/DSC e ao meus companheiros do mestrado, principalmente ao Cláudio Monteiro que me incentivou a fazer o mestrado, ao Mário Ernesto pelas inúmeras conversas elucidativas, ao Everaldo, Alexandre, Aleksandra, Giovanni e Juracy pelo companheirismo e amizade que tornaram Campina Grande e o nordeste muito mais agradável.

Não posso esquecer neste momento de agradecer também ao eterno carinho da minha família que sempre me apoiou "mesmo distante" e aos bons exemplos que me nortearam na direção do conhecimento, destacando aqui a minha eterna admiração por minha irmã Célia, a qual sempre terá o meu respeito e gratidão e por meu irmão Sabá que com suas atitudes me ajudaram a entender melhor a vida. É claro que o meu maior agradecimento vai para a minha esposa Cristiany que esteve ao meu lado durante todo o curso, ao meu pai Osvaldo e a minha mãe Hildinair, mulher de fibra e de luta, por tudo o que sou (mãe, este título também é seu!).

Quero registrar aqui também o meu agradecimento à Universidade da Amazônia – UNAMA – por ter permitido que eu realizasse este trabalho, com a dispensa das atividades acadêmicas e pelo auxílio financeiro através da sua Fundação Instituto para o Desenvolvimento da Amazônia – FIDESA, à Fundação Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – CAPES – pelo apoio financeiro e à Universidade Federal da Paraíba por manter o Curso de Pós-Graduação em Informática.

Resumo

Nesta dissertação de mestrado apresenta-se um Escalonador Inteligente de Tarefas em tempo real, desenvolvido para aplicações robóticas que implementam controle através de microcomputadores. O escalonador proposto foi idealizado a partir de observações feitas nos sistemas de controle de robôs, no Laboratório de Redes Neurais e Automação Inteligente (NEURO-LAB) ligado à Coordenação de Pós-graduação em Informática (COPIN) da Universidade Federal da Paraíba (UFPB). Nesses sistemas cada etapa do controle dos robôs era executada através de tarefas em tempo real. Para que um robô executasse todos os movimentos necessários à realização de uma meta preestabelecida, várias tarefas eram ordenadas pelo especialista e executadas seqüencialmente. Observou-se uma constante interferência do especialista na ordenação e temporização de cada tarefa até que se chegasse a uma seqüência ótima para o controle do robô. Utilizando-se o escalonador inteligente, proposto neste trabalho, o escalonamento passou a ser feito através de regras e inferências nebulosas. A utilização dessa regras juntamente com as restrições temporais, permitiu que a seleção e a execução das tarefas fossem realizadas pelo próprio software, dando mais autonomia ao sistema e conseqüentemente tornando-o mais inteligente.

Abstract

This dissertation presents a Real-time Intelligent Tasks Scheduler developed to robotics application that implement microcomputer control. The proposed scheduler was idealized after observations realized in robots' control system utilized in Intelligent Automation and Neural Networks Laboratory (NEUROLAB) linked to Coordination of Computer Science Graduation (COPIN) of Federal University of Paraíba (UFPB). In these systems, each control stage was executed through real-time tasks. Several tasks were ordered by an expert and executed sequentially to a robot executed all necessary motions to realize an established goal. Was observed a constant interference of expert in ordination and time definition of each task until was to reach an optimal sequence of tasks to robot's control. With the Intelligent Tasks Scheduler, that uses rules and inference fuzzy, the ordination and time definition of each task now is realized by own system, permitting more autonomy to the system and consequentially becomes more intelligent.

Sumário

LISTA DE FIGURAS V

LISTA DE TABELAS VII

LISTA DE QUADROS VIII

LISTA DE EQUAÇÕES X

LISTA DE REGRAS XII

LISTA DE SÍMBOLOS E ABREVIAÇÕES XIV

CAPÍTULO 1 1

INTRODUÇÃO 1

1.1. Os sistemas robóticos 1

1.2. Os sistemas de controle 1

1.3. O escalonador de tarefas em tempo real 2

1.4. O escalonador inteligente de tarefas 2

1.5. Objetivos do trabalho 3

1.6. Estrutura da dissertação 3

CAPÍTULO 2 5

ESCALONAMENTO DE TAREFAS 5

2.1. Resumo 5

2.2. Histórico 5

2.3. Tarefas e processos 6

2.4. Estados das tarefas 7

2.5. Bloco de controle das tarefas 9

2.6. Conceitos básicos sobre escalonamento de processos e tarefas 9
2.7. Algoritmos de escalonamento 11
2.8. Características de escalonamento 14
2.8.1. Quanto à utilização 15
2.8.2. Quanto à restrição de tempo 15
2.8.3. Quanto à reorganização 16
2.8.4. Quanto ao conhecimento das informações sobre os processos 16
2.8.5. Quanto ao compartilhamento da CPU 17
2.8.6. Quanto à localização 17
2.8.7. Quanto à adaptabilidade 18
2.9. O escalonador em tempo real 18
2.10. Conclusão 21
CAPÍTULO 3 22
LÓGICA NEBULOSA 22
3.1. Resumo 22
3.2. Introdução 22
3.3. Fundamentos da lógica nebulosa 23
3.4. Operações com conjuntos nebulosos 26
3.4.1. Complemento 26
3.4.2. União 27
3.4.3. Interseção 27
3.4.4. Comparação 28
3.4.5. Contido 28
3.5. Variáveis lingüísticas 29
3.6. Regras nebulosas 30
3.7. Utilização da lógica nebulosa 31

3.8. Conclusão 34
CAPÍTULO 4 35
A APLICAÇÃO ROBÓTICA 35
4.1. Resumo 35
4.2. Robôs cooperantes 35
4.3. Os manipuladores robóticos 36
4.4. Especificação do problema 38
4.5. Especificação do plano de ação 38
4.5.1. A estratégia *Pesa* 40
4.5.2. A estratégia *Doa* 40
4.5.3. A estratégia *Pega* 41
4.5.4. A estratégia *Solta* 42
4.5.5. A estratégia *Recebe* 42
4.6. O Sistema inteligente 43
4.6.1. O Sistema de Controle Inteligente (SCI) 43
4.6.2. Gerência da transferência de carga 46
4.6.3. A transferência de carga 47
4.7. Resultados experimentais 51
4.8. Conclusão 52
CAPÍTULO 5 53
O ESCALONADOR INTELIGENTE DE TAREFAS 53
5.1. Resumo 53
5.2. Introdução 53
5.3. Desenvolvimento do EIT 54
5.4. A transferência de carga 57
5.4.1. A estratégia *Pesa* 58

5.4.2. A estratégia *Doa* 60
5.4.3. A estratégia *Pega* 62
5.4.4. A estratégia *Solta* 63
5.4.5. A estratégia *Recebe* 65
5.5. Resultados experimentais 66
5.6. Conclusão 67
CAPÍTULO 6 68
CONCLUSÃO 68
6.1. Resultados obtidos 68
6.2. Trabalhos futuros 69
REFERÊNCIAS BIBLIOGRÁFICAS 70
APÊNDICES 75
APÊNDICE A – INTERFACE MOTORES CC 75
APÊNDICE B O DETECTOR DE POSIÇÃO NA ESTRATÉGIA *PESA* 79
APÊNDICE C CONTROLADOR NEURAL 82
APÊNDICE D TREINAMENTO DA RNMC 85
APÊNDICE E ALGORITMOS DAS TAREFAS DO ETR 89
APÊNDICE F O SISTEMA DE TRANSFERÊNCIA DE CARGA 93
APÊNDICE G ALGORITMOS DAS TAREFAS DO EIT 98

Lista de Figuras

Figura 2.1 – Estados das tarefas e transições entre estados. 8
Figura 2.2 – Escalonamento em fila. 11
Figura 2.3 – Processos compartilhando a CPU em um escalonamento FIFO. 12
Figura 2.4 – Processos compartilhando a CPU. 12
Figura 2.5 – Escalonamento por revezamento 13
Figura 2.6 – Escalonamento com prioridade. 14
Figura 3.1 – Representação de valores na lógica nebulosa. 23
Figura 3.2 – Conjuntos nebulosos dos tamanhos das árvores de uma loja. 25
Figura 3.3 – Representações de funções de pertinência. 26
Figura 3.4 – Representação gráfica da operação complemento. 27
Figura 3.5 – Representação gráfica da operação união. 27
Figura 3.6 – Representação gráfica da operação interseção. 28
Figura 3.7 – Representação gráfica da operação comparação. 28
Figura 3.8 – Representação gráfica da operação contido. 29
Figura 3.9 – Variáveis lingüísticas. 30
Figura 3.10 – Gráfico nebulosos da altura normalizada. 31
Figura 3.11 – Gráfico nebuloso da altura normalizada. 33
Figura 4.1 – Esquema geral de montagem da estrutura robótica 36
Figura 4.2 – Representação do manipulador em forma de pêndulo. 37
Figura 4.3 – Seqüência de estratégias de movimentação. 39
Figura 4.4 – Diagrama de estado da seqüência de movimentos. 39

Figura 4.5 – Gráfico nebuloso da estratégia *Pesa*. 40
Figura 4.6 – Travas mecânicas do motor livre. 41
Figura 4.7 – O Sistema de Controle Inteligente (SCI). 44
Figura 4.8 – O controlador neural adaptativo direto. 44
Figura 4.9 – ETR dependente do tempo. 46
Figura 4.10 – Representação da transferência da carga do manipulador 1 para o 2. 48
Figura 4.11 – Resultados experimentais da transferência de carga com o ETR. 52
Figura 5.1 – Escalonador Inteligente. 55
Figura 5.2 – Execução das tarefas e regras no tempo. 56
Figura 5.3 – Resultados experimentais da transferência de carga com o EIT. 66
Figura A.1 – Conector DB25/Fêmea. 75
Figura A.2 – Diagrama esquemático da interface controladora de motores CC. 76
Figura A.3 – Circuito de acionamento na forma de ponte monofásica completa. 78
Figura B.1 – Esquema de montagem de um detector de posição. 79
Figura B.2 – Análise gráfica de um detector de posição duplo. 80
Figura B.3 – Exemplo de movimentação durante a estratégia *Pesa*. 80
Figura B.4 – Gráfico nebuloso utilizado na estratégia Pesa. 81
Figura C.1 – Configurações de controles. 83
Figura C.2 – Controle neural adaptativo direto. 83
Figura C.3 – Sistema de controle neural adaptativo direto. 84
Figura D.1 – Valores iniciais das entradas e representação dos pesos. 86
Figura F.1 – Janela "sobre" do sistema. 93
Figura F.2 – Janela principal do sistema. 94
Figura F.3 – Janela de diálogo para confirmação da transferência. 95
Figura F.5 – Janela de visualização e alteração dos dados da RNMC. 96
Figura F.6 – Janela de visualização dos pesos e índices da RNMC. 97

Lista de Tabelas

Tabela 2.1 – Tempo de execução dos processos. 12

Tabela 3.1 – Tempo de execução dos processos. 33

Tabela A.1 – Relação dos componentes utilizados na interface 78

Lista de Quadros

Quadro 2.1 – Descritor de tarefas. 19

Quadro 2.2 – Algoritmo simplificado do escalonador em tempo real 20

Quadro 3.1 – Regras nebulosas. 30

Quadro 3.2 – Regras nebulosas. 32

Quadro 4.1 – Algoritmo simplificado do ETR. 46

Quadro 4.2 – Relação das tarefas do sistema. 47

Quadro 4.3 – Tarefas da estratégia *Pesa*. 48

Quadro 4.4 – Tarefa 9 ativada após 2000 ms. 49

Quadro 4.5 – Tarefa 1 para a transferência de carga do manipulador 1 para o 2. 49

Quadro 4.6 – Tarefas 10 para a transferência do manipulador 2 para o 1. 50

Quadro 4.7 – Descritores das tarefas da estratégia *Doa*. 50

Quadro 4.8 – Descritores das tarefas da estratégia *Pega*. 51

Quadro 4.9 – Descritor da tarefa da estratégia *Solta*. 51

Quadro 4.10 – Descritor da tarefa da estratégia *Recebe*. 51

Quadro 5.1 – Descritor de tarefas. 55

Quadro 5.2 – Algoritmo simplificado do EIT. 56

Quadro 5.3 – Relação das tarefas do sistema com o EIT. 58

Quadro 5.4 – Tarefas da estratégia *Pesa*. 58

Quadro 5.5 – Tarefas da estratégia *Doa* a partir da tarefa 1. 60

Quadro 5.6 – Tarefas da estratégia *Doa* a partir da tarefa 10. 61

Quadro 5.7 – Tarefas da estratégia *Pega* a partir da tarefa 18 62

Quadro 5.8 – Tarefas da estratégia *Pega* a partir da tarefa 17..... 63

Quadro 5.9 – Tarefas da estratégia *Solta* a partir da tarefa 12..... 64

Quadro 5.10 – Tarefas da estratégia *Solta* a partir da tarefa 13..... 64

Quadro 5.11 – Tarefas da estratégia *Recebe* a partir da tarefa 18. 65

Quadro 5.12 – Tarefas da estratégia *Recebe* a partir da tarefa 17. 66

Quadro D.1 – Características de um neurônio..... 85

Quadro D.2 – Algoritmo de propagação das entradas na RNMC. 87

Quadro D.3 – Algoritmo de propagação retroativa do erro..... 88

Lista de equações

(3.1): $\mu_X(x): \Re \rightarrow [0,1]$ 24

(3.2): $X = \{x, \mu_X(x)\}$ 25

(3.3): $\mu_{A'}(x) = 1 - \mu_A(x) \ \forall x \in X$ 26

(3.4): $A \cup B \rightarrow \mu_A(x) \cup \mu_B(x) = \max(\mu_A(x), \mu_B(x)) \ \forall x \in X$ 27

(3.5): $A \cap B \rightarrow \mu_A(x) \cap \mu_B(x) = \min(\mu_A(x), \mu_B(x)) \ \forall x \in X$ 28

(3.6): $A = B \rightarrow \mu_A(x) = \mu_B(x) \ \forall x \in X$ 28

(3.7): $A \subset B \rightarrow \mu_A(x) \leq \mu_B(x) \ \forall x \in X$ 28

(3.8): $Saida_q = \dfrac{\sum_{j=1}^{k} \mu_X(x) VD_k}{\sum_{j=1}^{k} \mu_X(x)}$ 33

(4.1): $Tl_{(t)} = ML^2 \dfrac{d^2\theta_{(t)}}{dt^2} + PL\,\mathrm{sen}(\theta_{(t)}) = ML^2\theta'' + PL\,\mathrm{sen}(\theta_{(t)})$ 38

(4.2): $Tl_{(t)} = PLSen(\theta_{(t)})$ 38

(4.3): $Tm_{(t)} = \mathrm{k_2 U = k_1} Sen(\theta)$ 38

(4.4): $\theta = \dfrac{\mathrm{k_2}}{\mathrm{k_1}} \mathrm{U} = \mathrm{KU}$ 38

(4.5): $E = \theta_{r(t+1)} - \theta_{(t+1)}$ 45

(4.6): $\dfrac{\partial I}{\partial U} = -\dfrac{\partial \theta_{(t+1)}}{\partial U_{(t+1)}} E$ 45

(4.7): $U_{(t+1)} = U_{(t)} + \left(-\mu \nabla_{I_{(U)}}\right) = U_{(t)} + \left(-\mu \frac{\partial I}{\partial U}\right) = U_{(t)} + \left(\mu E \frac{\partial \theta_{(t+1)}}{\partial U_{(t+1)}}\right)$ 45

(4.8): $I_{(U)} = \frac{1}{2} E^2 = \frac{1}{2} (\theta_{r(t+1)} - \theta_{(t+1)})^2$ 45

(D.1): $Z^* * \left(1 - \frac{Z}{\Theta}\right) * \left(Z^* - Z\right) * \beta$ 88

(D.2): $Z^* * \left(1 - \frac{Z}{\Theta}\right) * \left(W * \varepsilon\right) * \beta$ 88

(D.3): $W = W + \delta$ 88

(D.4): $\delta = \varepsilon * T * \rho$ 88

Lista de regras

Regra (4.1): Se θ_{1f} == G e θ_{3f} == P então ha_carga1 = 1, ha_carga2 = 0; 40

Regra (4.2): Se θ_{3f} == G e θ_{1f} == P então ha_carga1 = 0, ha_carga2 = 1; 40

Regra (4.3): Se ha_carga1 == 1 então $\theta_{1r} = \pi/6$ e $\theta_{2r} = \pi/4$ 41

Regra (4.4): Se ha_carga2 == 1 então $\theta_{3r} = -\pi/6$ e $\theta_{4r} = -\pi/4$ 41

Regra (4.5): Se ha_carga1 == 1 então $\theta_{3r} = -\pi/6$ 41

Regra (4.6): Se ha_carga1 == 1 e $\theta_{3r} == -\pi/6$ então $\theta_{4r} = -\pi/4$ 41

Regra (4.7): Se ha_carga2 == 1 então $\theta_{1r} = \pi/6$ 42

Regra (4.8): Se ha_carga2 == 1 e $\theta_{1r} == \pi/6$ então $\theta_{2r} = \pi/4$ 42

Regra (4.9): Se ha_carga1 == 1 então θ_{2f} = "ZE" 42

Regra (4.10): Se ha_carga1 == 1 e θ_{2r} == 0 então $\theta_{1r} = 0$ 42

Regra (4.11): Se ha_carga2 == 1 então $\theta_{4r} = 0$ 42

Regra (4.12): Se ha_carga2 == 1 e θ_{4r} == 0 então $\theta_{3r} = 0$ 42

Regra (4.13): Se ha_carga1 == 1 então $\theta_{1r} = 0$ 42

Regra (4.14): Se ha_carga1 == 1 e θ_{1r} == 0 então $\theta_{2r} = 0$ 42

Regra (4.15): Se ha_carga2 == 1 então $\theta_{3r} = 0$ 42

Regra (4.16): Se ha_carga2 == 1 e θ_{3r} == 0 então $\theta_{4r} = 0$ 42

Regra (5.1): **Se** $\theta_{1f} == \theta_{1rf}$ **então** desativa as tarefas 2, 3 e 9 e ativa a tarefa 1; 59

Regra (5.2): **Se** $\theta_{3f} == \theta_{3rf}$ **então** desativa as tarefas 2, 3 e 9 e ativa a tarefa 10; 59

Regra (5.3): **Se** $\theta_{1f} == \theta_{1rf}$ **ou** θ_{1f} =="ZE" **então** ativa a tarefa 7; 60

Regra (5.4): **Se** $\theta_{1f} == \theta_{1rf}$ **ou** θ_{1f} =="ZE" **então** ativa a tarefa 18; 61

Regra (5.5): **Se** θ_{3f} == θ_{3rf} **ou** θ_{3f} == "ZE" **então** ativa a tarefa 18; 61

Regra (5.6): **Se** θ_{3f} == θ_{3rf} **ou** θ_{3f} == "ZE" **então** ativa a tarefa 7; 62

Regra (5.7): **Se** θ_{3f} == θ_{3rf} **então** θ_{3f} = 1 **senão se** θ_{3rf} == "ZE" **então** ativa a tarefa 11; 62

Regra (5.8): **Se** θ_{3f} == θ_{3rf} **ou** θ_{3f} == "ZE" **então** desativa a tarefa 12, θ_{2rf} = "ZE"; 63

Regra (5.9): **Se** θ_{1f} == θ_{1rf} **então** ativa tarefa 12 **senão se** θ_{1f} == "ZE" **então** ativa tarefa 14; 63

Regra (5.10): **Se** θ_{1f} == θ_{1rf} **ou** θ_{1f} == "ZE" **então** desativa a tarefa 13, θ_{4rf} = "ZE"; 63

Lista de símbolos e abreviações

Símbolos

$\mathcal{R}$	-	Conjunto universal
(t)	-	Representação do tempo na forma discreta
$=$	-	Representa a operação de atribuição
$==$	-	Representa a operação de igualdade
a	-	Elemento do conjunto de árvores
A	-	Subconjunto nebuloso de X
A'	-	Complemento do subconjunto nebuloso A
B	-	Subconjunto nebuloso de X
E	-	Erro obtido na saída da planta
$f(y)$	-	Função de transferência do neurônio
freq	-	Intervalo em que uma tarefa é novamente ativada
G	-	Conjunto nebuloso das árvores grandes
id	-	Identificação de uma tarefa
L	-	Comprimento do pêndulo
M	-	Conjunto nebuloso das árvores médias
MP	-	Conjunto nebuloso das árvores muito pequenas
NG	-	Subconjunto nebuloso Negativo Grande do controlador
NM	-	Subconjunto nebuloso Negativo Médio do controlador
NP	-	Subconjunto nebuloso Negativo Pequeno do controlador
nvf	-	Número do motor relacionada a uma regra
P	-	Conjunto nebuloso das árvores pequenas
P	-	Peso do pêndulo
PG	-	Subconjunto nebuloso Positivo Grande do controlador
PM	-	Subconjunto nebuloso Positivo Médio do controlador
PP	-	Subconjunto nebuloso Positivo Pequeno do controlador

st	-	Estado de uma tarefa
t	-	Tempo na forma discreta
$tempo$	-	Tempo de espera para execução de uma tarefa
$Tl_{(t)}$	-	Torque do pêndulo
U	-	Tensão do motor
$U_{(t)}$	-	Tensão do motor informada ao controlador neural
$U_{(t+1)}$	-	Tensão fornecida pelo controlador neural
U_f	-	Tensão *fuzzyficada* do motor
VD	-	Valor desejado
vf	-	Variável lingüística do ângulo "*fuzzyficado*" do motor
W_i	-	Pesos da RNMC
x	-	Elemento do subconjunto X
X	-	Subconjunto de $\Re$ ou o conjunto universo
X_i	-	Entradas da RNMC
ZE	-	Subconjunto nebuloso Zero do controlador

Símbolos gregos

ΔU_f	-	Variação da tensão *fuzzyficada*
Ω	-	Aceleração angular do pêndulo
χ_f	-	Fator de treinamento da rede neural
μ	-	Constante da função $\mu\text{sigm}(\theta)$
θ	-	Ângulo do pêndulo
$\theta_{(t)}$	-	Ângulo informado ao controlador neural
$\theta_{(t+1)}$	-	Ângulo lido na saída da planta
$\theta_{(t-1)}$	-	Ângulo obtido no controle anterior
θ_1	-	Ângulo do motor 1
θ_{1f}	-	Ângulo *fuzzyficado* do motor 1
θ_{1r}	-	Ângulo de referência do motor 1
θ_2	-	Ângulo do motor 2
θ_{2f}	-	Ângulo *fuzzyficado* do motor 2
θ_{2r}	-	Ângulo de referência do motor 2
θ_3	-	Ângulo do motor 3

θ_4	-	Ângulo do motor 4
θ_{3f}	-	Ângulo *fuzzyficado* do motor 3
θ_{3r}	-	Ângulo de referência do motor 3
θ_{4f}	-	Ângulo *fuzzyficado* do motor 4
θ_{4r}	-	Ângulo de referência do motor 4
θ_f	-	Ângulo *fuzzyficado*
θ_i	-	Ângulo do motor *i*
θ_{if}	-	Ângulo do motor *i fuzzyficado*
$\theta_{r(t+1)}$	-	Ângulo de referência informado ao controlador neural

Abreviações

APR	-	Algoritmo de propagação retroativa do erro
CC	-	Corrente contínua
COPIN	-	Coordenação de Pós-Graduação em Informática
CPU	-	Unidade Central de Processamento (do inglês *Central Processing Unit*)
E/S	-	Entrada e Saída
EIT	-	Escalonador Inteligente de Tarefas em Tempo Real
ETR	-	Escalonador de Tarefas em Tempo Real
FIFO	-	*First In First Out* (definição de fila em inglês)
IA	-	Inteligência Artificial
IBM	-	*International Business Machine*
IBM-PC	-	Definição padrão dos microcomputadores baseados na família de processadores 8086
Motor CC	-	Motor de corrente contínua
ms	-	Milissegundo
NEUROLAB	-	Laboratório de Redes Neurais e Automação Inteligente
PC	-	Contador de programa (do inglês *Program Counter*)
pu	-	Por unidade (medida normalizada, explicada no capítulo 3 e 4)
PWM	-	Modulação de Largura de Pulso (do inglês, *Pulse Width Modulation*)
RNA	-	Rede Neural Artificial
RNMC	-	Rede Neural Artificial Multicamadas
RTC	-	Relógio de tempo real (do inglês *Real-Time Clock*)
SCI	-	Sistema de Controle Inteligente

SI	-	Sistema Inteligente
SP	-	Apontador de pilha (do inglês *Stack Pointer*)
TCB	-	*Task Control Block* (bloco de controle de tarefa em inglês)
UFPB	-	Universidade Federal da Paraíba

"Não é possível observar ou medir um objeto sem interferir nele, sem o alterar, e a tal ponto que o objeto que sai de um processo de medição não é o mesmo que lá entrou." (Heisemberg & Bohr, citado em Araújo, 1998).

Capítulo 1

Introdução

1.1. Os sistemas robóticos

O gerenciamento de sistemas robóticos através de microcomputadores envolve diversas áreas do conhecimento tais como Engenharia Elétrica, Engenharia Mecânica, Computação, entre outras. Várias ferramentas e tecnologias foram propostas e estão sendo utilizadas para se ter robôs mais autônomos e eficazes. Pode-se citar: processamento digital de imagem, aquisição de conhecimento, redes neurais artificiais, lógica nebulosa, algoritmos genéticos e controladores adaptativos. Essas são as principais razões que tornaram os sistemas robóticos amplamente utilizados por pesquisadores do mundo todo como plataformas de teste para novas técnicas e pesquisas científicas.

Cavalcanti et al. (1994) apresentaram um Sistema de Controle Inteligente (SCI), baseado em Rede Neural Artificial Multicamada e Lógica Nebulosa, capaz de controlar uma estrutura robótica elementar (planta) em tempo real. Posteriormente Alsina & Cavalcanti (1997) utilizaram esse SCI aplicado na transferência de carga entre dois robôs cooperantes. Todo o gerenciamento dos módulos dinâmicos do SCI foi realizado por um escalonador de tarefas em tempo real. O SCI se mostrou adequado ao controle dinâmico da planta, principalmente devido as propriedades adaptativos da rede neural (Alsina, 1996, p.29). Porém, como os módulos dinâmicos do sistemas foram implementados em forma de tarefas, a ordenação e a temporização destas foram feitas heuristicamente pelo especialista, limitando o escalonador à responsabilidade de simplesmente autorizar a execução de cada tarefa seqüencialmente, trabalhando como um simples despachador (Comer, 1988).

1.2. Os sistemas de controle

Um sistema de controle pode ser classificado em duas categorias: os sistemas de controle dinâmicos e os sistemas de controle de processos. O controle dinâmico envolve sempre o controle de uma planta, enquanto o controle de processo tem como objetivo a temporização, seqüencialização, sincronização e intertravamento de eventos (Kovács, 1996, p.111). A tempo-

rização pode ser vista como o tempo de espera para a execução de cada evento; a seqüencialização é a ordenação temporal dos eventos; a sincronização pode ser vista como a necessidade de comunicação entre os eventos, preferencialmente de uma forma estruturada e; o intertravamento é a competição entre os eventos por recursos. Todos esses itens estão relacionados com o escalonamento dos eventos ou tarefas que representam o processo. Portanto, em um sistema robótico existe sempre a presença de um sistema de controle de processos e por conseqüência, a necessidade de um algoritmo de escalonamento adequado, o que justifica o presente estudo de um "Escalonador Inteligente de Tarefas para Aplicações Robótica".

1.3. O escalonador de tarefas em tempo real

Alsina & Cavalcanti (1997) apresentaram um sistema robótico composto de dois manipuladores cooperantes utilizados na transferência de carga entre eles. Nesse sistema todo o controle dos manipuladores e da transferência de carga foi implementado em forma de tarefas e essas gerenciadas por um Escalonador de Tarefas em Tempo Real (ETR), baseado no escalonamento em fila (Shay, 1996, p.264). Observou-se nesse sistema que o escalonador poderia tomar algumas decisões, ditas inteligentes, para substituir as constantes interferências do especialista, necessárias no ETR para a ordenação e temporização das tarefas, e ao mesmo tempo ter o seu código reduzido o máximo possível para garantir o escalonamento de todas as tarefas em tempo real.

As características do escalonamento em tempo real e da aplicação (sistema robótico) impuseram restrições de tempo e de pendências de posição dos manipuladores que precisavam ser rigorosamente cumpridas, pois tanto o controle dinâmico da planta (os motores dos manipuladores) quanto o controle do processo (a transferência de carga) dependiam diretamente do fator "tempo" e da posição dos manipuladores em cada etapa da transferência.

1.4. O escalonador inteligente de tarefas

A partir da análise dos problemas verificados na utilização do ETR e na relação de controle de processos com o escalonamento de tarefas, idealizou-se o Escalonador Inteligente de Tarefas em Tempo Real (EIT). O EIT utiliza a Lógica Nebulosa para representar o conhecimento do especialista e por conseqüência permitir que o escalonador proceda de uma forma inteligente na ordenação e execução das tarefas, substituindo o especialista.

A solução proposta (Ferreira et al., 1998, 1999a, 1999b) foi associar a cada tarefa uma regra nebulosa do tipo *se-então* com inferências sobre variáveis nebulosas. Para isso, associou-se essas variáveis a cada um dos atuadores (motores de corrente contínua) dos manipuladores, de tal forma que fosse possível, que o próprio escalonador fizesse inferências, sobre essas variáveis, detectando quando o movimento em cada etapa foi ou não bem sucedido. O relacionamento entre regras e variáveis nebulosas possibilitou mais autonomia ao sistema e conseqüentemente permitiu que o mesmo pudesse ser considerado inteligente, visto que o mesmo, além de ser temporizado, passou a tomar suas próprias decisões a partir de informações adquiridas do ambiente em tempo real.

1.5. Objetivos do trabalho

Esta dissertação tem como principal objetivo propor e descrever o projeto e a implementação de um escalonador inteligente de tarefas para aplicações robóticas. Esse escalonador é utilizado no gerenciamento das tarefas de um sistema de controle inteligente que controla a mudança de estado e os atuadores de dois manipuladores robóticos durante a transferência de carga entre eles. O escalonador proposto utiliza inferências e variáveis nebulosas na otimização da ordenação das tarefas para garantir o escalonamento de cada uma delas em tempo real.

1.6. Estrutura da dissertação

Esta dissertação está dividida em seis capítulos, sendo que o primeiro capítulo é esta própria introdução, onde apresenta-se os motivos que conduziram ao estudo proposto, os objetivos da dissertação e um resumo geral de todo o conteúdo da mesma.

Faz-se no Capítulo 2 um estudo sobre o escalonamento de tarefas, definindo-se os conceitos básicos sobre escalonamento, os algoritmos de escalonamento mais utilizados, as principais características de um escalonamento, bem como as taxinomias sugeridas por alguns autores. Encerra-se o capítulo, com uma breve análise das características de operação necessárias à implementação de um escalonador em tempo real para o gerenciamento do controle de tarefas.

Apresenta-se no Capítulo 3 um breve estudo sobre a Lógica Nebulosa, descrevendo-se os seus fundamentos e destacando-se a definição de conjuntos nebulosos e a sua normalização. Apresenta-se também as principais operações com conjuntos nebulosos, a definição de

variáveis nebulosas e de regras nebulosas. Finaliza-se o capítulo com um estudo de utilização da lógica nebulosa, evidenciando-se a seqüência na qual cada conceito deve ser empregado para que se tire o melhor proveito dessa lógica.

Apresenta-se no Capítulo 4 uma descrição detalhada da aplicação robótica, destacando-se o projeto e desenvolvimento do protótipo utilizado para a implementação do escalonador proposto. Descreve-se também o sistema inteligente dividido em um sistema de controle inteligente, utilizado no controle dos motores elétricos de corrente contínua e um sistema de controle de processos, utilizado no controle das etapas da transferência de carga. Finaliza-se o capítulo com uma breve análise da estrutura robótica e do sistema inteligente.

Finalmente, apresenta-se no Capítulo 5 o "Escalonador Inteligente de Tarefas para Aplicações Robóticas" proposto nesta dissertação. Inicialmente, descreve-se os motivos que conduziram ao desenvolvimento desse escalonador, apresentando-se uma breve introdução, contendo as principais características de escalonamento que o mesmo deve apresentar. Em seguida descreve-se como esse escalonador foi desenvolvido, apresentando-se as estratégias de movimento, definidas no Capítulo 4, com as devidas alterações para serem utilizadas nessa nova abordagem. Encerra-se o capítulo com uma análise do novo escalonador apresentado.

Conclui-se a dissertação no capítulo 6, no qual é feita uma análise das principais vantagens obtidas com o escalonador inteligente de tarefas, destacando-se principalmente a tomada de decisão pelo próprio sistema, a utilização de regras sem a necessidade de se criar uma base de conhecimento e a maior independência do sistema em relação ao especialista. Encerra-se a conclusão com a apresentação de algumas sugestões de trabalhos futuros.

Capítulo 2

Escalonamento de tarefas

2.1. Resumo

Neste capítulo inicialmente apresenta-se um breve histórico sobre o escalonamento de tarefas, os conceitos básicos sobre escalonamento e os principais algoritmos de escalonamento. Apresenta-se também um estudo sucinto sobre "*job*"[1], processos e tarefas, destacando-se os diferentes conceitos apresentados por alguns autores e os possíveis estados que uma tarefa ou processo pode assumir. Finaliza-se o capítulo apresentando-se o escalonador de tarefas em tempo real, destacando-se as principais características que o mesmo deve ter para ser utilizado no gerenciamento do controle de processo e na implementação do escalonador inteligente de tarefas proposto no capítulo 5.

2.2. Histórico

O escalonamento de processos tem sua origem relacionada com os primeiros sistemas operacionais. Quando surgiram os antigos sistemas em lote (do inglês *Batch Systems*) – os sistemas 360 da IBM com processamento em lote – o escalonamento de processos se resumia em fazer com que o próximo *job* da fila fosse posto para executar (Tanenbaum, 1995, p.44). Nesta época os *jobs* representavam processos que ficavam armazenados em fita magnética à espera da execução. O escalonador, parte do sistema operacional responsável por fazer o escalonamento, era uma rotina muito simples, responsável pela ativação e execução dos *jobs*. Posteriormente, quando os processos passaram a depender do tempo, os algoritmos de escalonamento foram aperfeiçoados e alguns autores passaram a tratar os processos como tarefas.

---

[1] Este termo não foi traduzido pois nenhum autor da língua portuguesa consultado fez a tradução do mesmo. O autor desta dissertação acredita que a tradução do termo, que no caso seria "trabalho", não refletiria o conceito exato dentro do contexto em estudo.

Com a melhoria dos computadores e por conseqüência dos sistemas operacionais, impulsionados pelas suas utilizações no setor comercial, o escalonamento de tarefas também evoluiu, tendo seus algoritmos uma importância mais destacada no estudo e implementação dos sistemas operacionais. O computador podia ser utilizado por vários usuários ao mesmo tempo e a sua Unidade Central de Processamento (CPU, do inglês *Central Processing Unit*) precisava ser compartilhada entre esses usuários. Era necessário tirar o máximo proveito do computador, visto que o custo do mesmo era muito elevado, sendo de vital importância que a sua CPU ficasse ocupada 100% do tempo. Para isso, era necessário implementar uma rotina de escalonamento que garantisse essa eficiência na utilização da CPU. Entretanto, após a implementação de rotinas 100% eficientes, certos tipos de usuários foram prejudicados, então percebeu-se que somente a eficiência não garantia a melhor utilização da CPU, pois um sistema com eficiência máxima, não representava a melhor característica para determinados tipos de usuários.

Isso conduziu ao desenvolvimento de diferentes algoritmos de escalonamento, onde cada um procurava melhorar o desempenho do sistema para cada tipo de usuário. Entretanto, muitos sistemas precisavam atender as necessidade de diferentes usuários ao mesmo tempo e conseqüentemente, precisavam apresentar um balanceamento das inúmeras características existentes, sendo conhecidos como escalonamento de propósito geral. Os escalonamentos feitos para atender uma determinada característica foram denominados de escalonamento específico, sendo o mais conhecido o escalonamento em tempo real que precisa atender a rigorosas restrições de tempo.

2.3. Tarefas e processos

O estudo sobre escalonamento induz ao estudo dos conceitos de *jobs*, tarefas e processos. Alguns autores (Shay, 1996) (Comer, 1988) os tratam como se eles representassem o mesmo conceito. Outros autores fazem distinção entre eles (Mullender, 1995). Atualmente o conceito de *job* é pouco utilizado, visto que o mesmo é quase sempre relacionado com os antigos sistemas em lote, onde um *job* representava um programa ou um conjunto de programas (Tanenbaum, 1995, p.44) postos para executar. Os conceitos de tarefa e processo são interrelacionados, chegando a variar de sistema para sistema, segundo Comer (1988, p.7).

Um processo representa um programa em execução sendo constituído do código executável, dos dados referentes ao código, da pilha de execução, do valor do contador de pro-

grama (registrador PC, do inglês *Program Counter*), do valor do apontador de pilha (registrador SP, do inglês *Stack Pointer*) e outras informações necessárias para a execução do programa (Tanenbaum, 1995, p.10). Mullender (1995, p.392) apresenta esse mesmo conceito, destacando que um processo é uma abstração de um programa sendo executado. Segundo Mullender (1995, p.392), a definição confusa de processo e tarefa é ressaltada com a nova abordagem de sistemas distribuídos, pois diferentes grupos de pesquisa usam diferentes e conflitantes conceitos relacionados a tarefas e processos. Para Shay (1996, p.17), o conceito de processo é equivalente ao de tarefa. Então, é possível concluir que a maioria dos procedimentos adotados para tratamento dos processos pode ser aplicado também para tratamento das tarefas.

A definição entendida pelo autor desta dissertação é a seguinte. Um processo é o conjunto de informações de baixo nível manipuladas pelo escalonador do sistema operacional, enquanto uma tarefa é a representação de uma ação definida por um conjunto de instruções dependentes do tempo, manipuladas por um escalonador em alto nível. Uma tarefa pode dar origem a vários processos. Todo o estudo de escalonamento discutido a partir deste ponto nesta dissertação, está relacionado com o escalonamento de tarefas.

2.4. Estados das tarefas

As tarefas possuem estados bem definidos, onde cada tarefa pode estar em um determinado estado a cada instante. O conceito de estado é importante para se entender como o escalonador pode manipular as tarefas. Supondo-se que uma tarefa executa uma operação de entrada e saída (E/S), então ela deve ser interrompida para permitir que outras tarefas utilizem a CPU enquanto a sua operação de E/S é realizada. Assim, a tarefa que estava no estado "*executando*" passa para o estado "*espera por E/S*". Supondo-se também que uma outra tarefa esgotou o seu tempo de uso da CPU antes mesmo que o seu processamento chegasse ao final, assim ela é interrompida e enviada para o final da lista de tarefas em espera, a fim de receber uma nova fatia de tempo para terminar o seu processamento. Neste caso, a tarefa passa do estado "*executando*" para o estado "*espera por execução*". Em ambos os casos, assim que as tarefas são interrompidas uma outra tarefa recebe o controle da CPU e inicia o seu processamento, passando do estado "*espera por execução*" para o estado "*executando*".

Os exemplos evidenciam que uma tarefa pode passar por vários estados diferentes, como: *pronta* (do inglês *ready*) para assumir o controle da CPU, *executando* (do inglês *running*) usando a CPU, *inativa* (do inglês *inactive;* O termo *terminated*, também é utilizado) por

ter encerrado o seu processamento, *bloqueada* (do inglês *blocked*) por ter gerada uma operação de E/S ou *pronta* novamente, por ter esgotado o tempo de uso da CPU com retorno à lista de tarefas em espera (Guimarães, 1989). Neste caso diz-se que a tarefa sofreu uma preempção. Os estados das tarefas permitem ao escalonador um controle mais eficiente sobre todas as tarefas, pois as tarefas bloqueadas não precisam concorrer pelo uso da CPU, diminuindo assim o tamanho da lista de tarefas esperando para o uso da CPU, e diminuindo também o tempo de manipulação dessa lista, além de evitar o processamento desnecessário de tarefas bloqueadas. Os estados que uma tarefa pode assumir são *pronta*, *executando*, *bloqueada* e *inativa* e estão representados na Figura 2.1 pelas elipses. As possíveis transições de um estado para o outro também são ilustradas nessa mesma figura e estão representadas pelas setas em forma de arco com as respectivas descrições.

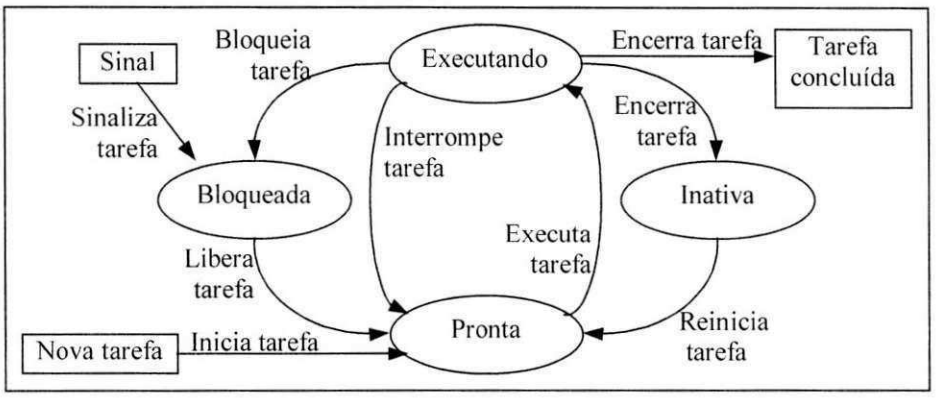


Figura 2.1 – Estados das tarefas e transições entre estados.

Os eventos esternos, que não representam transições entre estados, mas que modificam o estado de uma tarefas, estão representados pelos retângulos. Quando uma nova tarefa é criada, ela já recebe o estado *pronta,* entrando diretamente na lista de tarefas em espera para execução. Ao final da execução de uma tarefa, ela passa para o estado *inativa* ou é simplesmente removida do sistema (tarefa concluída). As tarefas no estado *bloqueada* precisam de um sinal para avisá-las que o evento que elas aguardavam já foi concluído e as mesmas já podem voltar para o estado *pronta* para concorrem ao uso da CPU.

Alguns autores apresentam o estado *bloqueada* dividido em outros estados como *espera ocupada*. Isso indica que uma tarefa foi bloqueada por ter ocorrido um erro por estar esperando por um evento gerado por outra tarefa ou por ela Ter gerado um evento de solicitação de E/S. Essa divisão é justificada por Shay (1996, p.250), para que as tarefas no estado *bloqueada* por solicitação de E/S podem entrar em uma nova concorrência pelo recurso solicita-

do, diferente das tarefas no estado *bloqueada* por erro, que somente dificultariam a gerência das outras tarefas. Porém, a maioria dos autores apresenta um único estado para essa situação, devido a sua subdivisão ser empregada somente em alguns casos particulares.

2.5. Bloco de controle das tarefas

A lista de tarefas, até o momento apresentada como sendo o conjunto de tarefas do sistema prontas para serem escalonadas, na verdade representa uma lista de descritores ou blocos de controle das tarefas (TCB, do inglês *Task Control Block*) (Shay, 1996, p.254). O descritor ou TCB de uma tarefa contém algumas informações que o sistema precisa saber sobre a tarefa, tais como: número de identificação, estado, tempo máximo de execução, prioridade, localização do código da tarefa, apontador para a entrada na lista de tarefas, valor do contador de programa, valor do apontador de pilha e valor dos registradores, entre outros.

O conteúdo do descritor varia de um sistema operacional para outro, pois muitas informações podem ser desnecessárias, dependendo do tipo de escalonamento adotado. Por exemplo, em um escalonamento onde é permitido interromper a execução de uma tarefa antes do seu final, é necessário saber todas as informações sobre o apontador de pilha, contador de programa e valor dos registradores, para que a tarefa que foi interrompida possa ser reiniciada com todos os valores desses itens restaurados. Em um escalonamento onde a interrupção não é permitida, esses itens são desnecessários.

2.6. Conceitos básicos sobre escalonamento de processos e tarefas

A busca pela eficiência na utilização da CPU (otimização no uso do computador), juntamente com a meta de se implementar o melhor algoritmo de escalonamento possível, deram origem a alguns conceitos básicos que devem ser considerados na implementação de um escalonador. Tanenbaum (1995, p.45) e Shay (1996, p.241) relacionaram alguns desses conceitos que devem estar presentes em um bom algoritmo de escalonamento, a saber:

- Eficiência – Segundo Tanenbaum (1995, p.45), eficiência é manter o processador ocupado durante 100% do tempo, pois esse tipo de equipamento é relativamente caro e se o mesmo fica parte do tempo ocioso, torna questionável o investimento feito nele. Por sua vez, Shay (1996, p.242) define eficiência pela medida de produção de processamento, afirmando que, a produção é o número de processos que são executados por unidade de

tempo. Então, a produção é baixa quando são completados poucos processos e alta quando muitos processos são completados concluindo que, deve-se tentar conseguir a produção mais alta possível. Esta definição de produção é apresentada por Tanenbaum como "*Throughput*";

- Capacidade de resposta – O mesmo algoritmo que garante a eficiência deve garantir também a resposta (resultado do processamento) dentro de uma faixa de tempo aceitável. Em determinadas situações, a garantia de resposta dentro de um tempo pequeno se opõe à eficiência, visto que um escalonador que é totalmente eficiente mas que demora excessivamente a dar respostas para "usuários *batch*" (usuários que executam processos em lote), não terá muita aceitação;

- Tempo de resposta – O sistema deve responder rapidamente às solicitações dos usuários interativos[2], dentro de um tempo razoavelmente aceitável. Esses usuários exigem mais rigidez (tempo de resposta mínimo) no tempo de resposta pois os mesmos estão interagindo diretamente com o sistema, esperando uma resposta do sistema para cada solicitação enviada. Em oposição aos usuários interativos, pode-se citar os usuários *batch*, que normalmente enviam um conjunto de *jobs* e não esperam por uma resposta imediata. O mais importante para esses usuários é o tempo total de execução de todo o conjunto de *jobs* e não a execução de cada *job* isolado;

- Consistência – O tempo que o sistema leva para executar uma determinada tarefa deve ser sempre o mesmo, independente do horário que a tarefa for posta para execução. Isto é, se um usuário executa regularmente a mesma tarefa, a resposta do sistema deve ser dentro de um tempo aproximadamente igual, mesmo que o usuário apresente a tarefa em momentos diferentes;

- Justiça – Todos os processos devem ter chances iguais no uso do processador. Este conceito pode ser visto como um balanceamento dos outros conceitos apresentados de tal forma que todos os usuários tenham o seu processamento garantido. Um bom exemplo é a comparação entre usuários *batch* e interativos que fazem solicitações diferentes e conflitantes. Ou seja, se o sistema beneficiar os usuários interativos, certamente ele prejudi-

---

[2] Usuários que operam diretamente em um terminal onde comandos curtos são enviados e respostas imediatas são aguardadas para que outros comandos sejam enviados.

cará os usuários *batch* e vice-versa. A melhor saída então é tentar atender as necessidades de todos os usuários distribuindo perdas e ganhos entre todos.

2.7. Algoritmos de escalonamento

Vários algoritmos de escalonamento de tarefas que apresentam uma combinação dos conceitos básicos sobre escalonamento foram propostos ao longo dos anos. Os mais conhecidos estabeleceram padrões e atualmente são referências para qualquer estudo sobre escalonamento. O algoritmo considerado mais simples de implementar é o escalonamento em fila ou escalonamento FIFO (Shay, 1996, p.264), que utiliza uma lista de processos ou tarefas representadas pelos quadrados com a letra P, conforme ilustrado na Figura 2.2. A lista utilizada obedece o conceito de fila, isto é, o primeiro processo a entrar na fila será o primeiro a sair. A sigla FIFO é uma abreviação em inglês do conceito de fila que significa "*First In First Out*" ("Primeiro que entra, primeiro que sai").

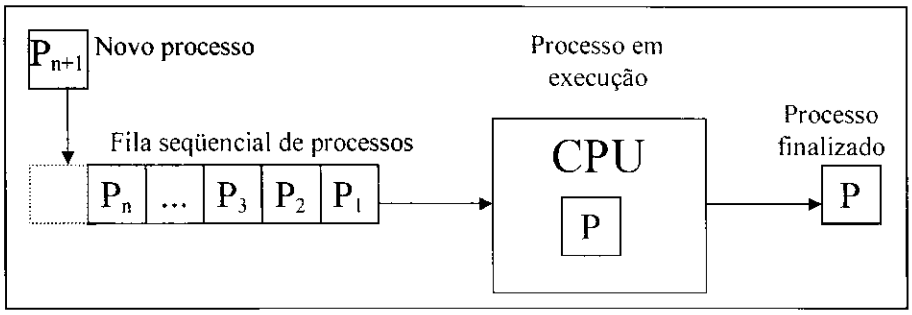


Figura 2.2 – Escalonamento em fila.

Nesse tipo de escalonamento, um processo que recebe o controle da CPU (processo em execução na Figura 2.2) só a libera quando o seu processamento chega ao final (processo finalizado na Figura 2.2). Ou seja, um processo não pode ser interrompido antes que ele termine completamente o seu processamento. Este escalonamento é apropriado para casos particulares, em que se faz muito processamento e poucas operações de E/S. Shay (1996, p.265) apresenta um bom exemplo para a utilização do escalonamento em fila, descrito a seguir:

Suponha que três processos A, B e C, entrem no sistema simultaneamente. Suponha também que cada um requeira exatamente uma hora de tempo da CPU. Considere um escalonador FIFO e que o processo A foi escalonado em primeiro lugar (Figura 2.3). Como o processo A precisa de uma hora de CPU e apresenta pouca ou nenhuma E/S, ele terminará em aproximadamente uma hora. Depois disso, suponha que o sistema operacional passe o con-

trole da CPU ao processo B, que também terminará em aproximadamente uma hora, ou seja, mais ou menos duas horas depois de ter entrado no sistema. Finalmente o processo C é executado e também termina depois de uma hora aproximadamente.

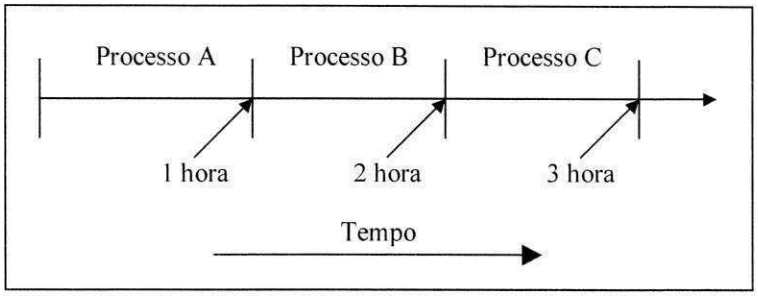


Figura 2.3 – Processos compartilhando a CPU em um escalonamento FIFO.

Analisando-se esse exemplo, conforme os informações resumidas na Tabela 2.1, verifica-se que o processo A termina no seu tempo previsto, já o processo B termina com o seu tempo previsto e mais o tempo do processo A, enquanto o processo C não se beneficia de forma alguma e termina no pior tempo.

Tabela 2.1 – Tempo de execução dos processos.

	FIFO
Tempo para o processo A	1 hora
Tempo para o processo B	2 horas
Tempo para o processo C	3 horas
Tempo médio	2 horas

Entretanto se eles tivessem compartilhando a CPU, todos terminariam em aproximadamente 3 horas, piorando o tempo de resposta para todos os processos, conforme ilustrado na Figura 2.4. Esse tipo de escalonamento que permite o compartilhamento da CPU é apresentado a seguir.

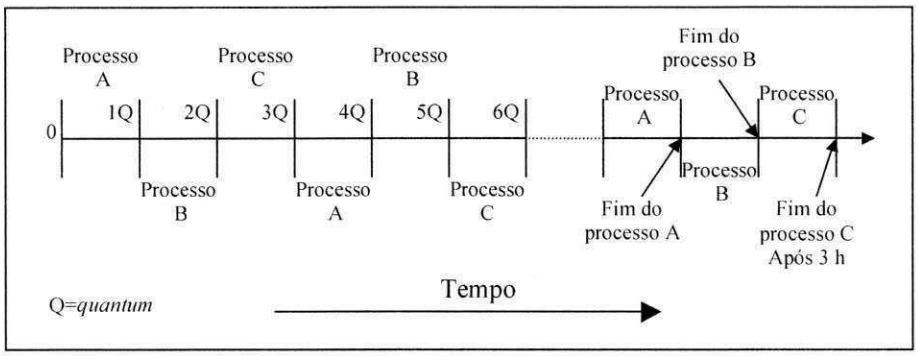


Figura 2.4 – Processos compartilhando a CPU.

Um outro tipo de escalonamento e o mais conhecido é o escalonamento por revezamento (Shay, 1996, p.257) ou "escalonamento *robin round*" (Tanenbaum, 1995, p.45), que implementa uma lista seqüencial de processos na memória, ilustrado na Figura 2.5, de onde os processos ou tarefas, representados pelos quadrados com a letra P, são ativados seqüencialmente para execução. A lista de processos neste algoritmo também é implementada através do conceito de fila (Fila seqüencial de processos na Figura 2.5). Cada processo possui um intervalo de tempo, conhecido como "*quantum*"[3] para a utilização da CPU (Processo em execução na Figura 2.5), se esse tempo se esgota antes do final da execução do processo, o mesmo é interrompido e enviado para o final da fila para ser escalonado novamente com um novo *quantum*. A essa interrupção de um processo antes do final de sua execução dá-se o nome de preempção (do inglês *preemption*). Um processo também pode ser interrompido quando o mesmo executa uma operação de E/S, liberando a CPU para o próximo processo em espera na fila seqüencial, evitando assim que a CPU fique ociosa.

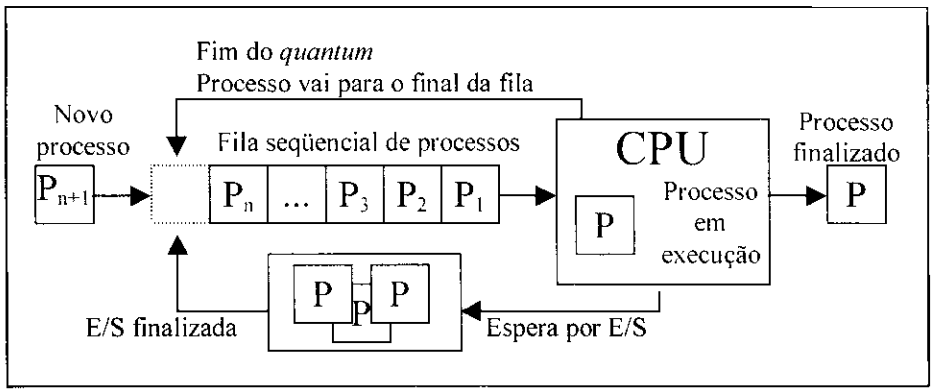


Figura 2.5 – Escalonamento por revezamento.

Um outro tipo é o escalonamento com prioridade que visa atribuir prioridade aos processos, permitindo que um processo possa passar a frente de outro na lista de espera. Este conceito anula a idéia de lista seqüencial proposta pelos escalonamentos por revezamento e em fila, pois cada processo é escalonado de acordo com a sua prioridade. Essa prioridade é definida através de números atribuídos a cada processo quando o mesmo é criado. Em algumas situações, é necessário que um processo que acabou de ser criado seja imediatamente executado, como é o caso de processos em um computador de uma universidade, onde os processos criados pela reitoria devem ter prioridade sobre os processos criados pelos professores,

[3] *Quantum* é a denominação normalmente utilizada para designar o intervalo de tempo que um processo pode usar a CPU.

enquanto estes últimos devem ter prioridade sobre os processos criados pelos alunos. O escalonamento com prioridade pode ser visto como um conjunto de escalonamentos em fila seqüencial, pois os processos com a mesma prioridade são executados de acordo com a ordem de chegada, conforme ilustrado na Figura 2.6.

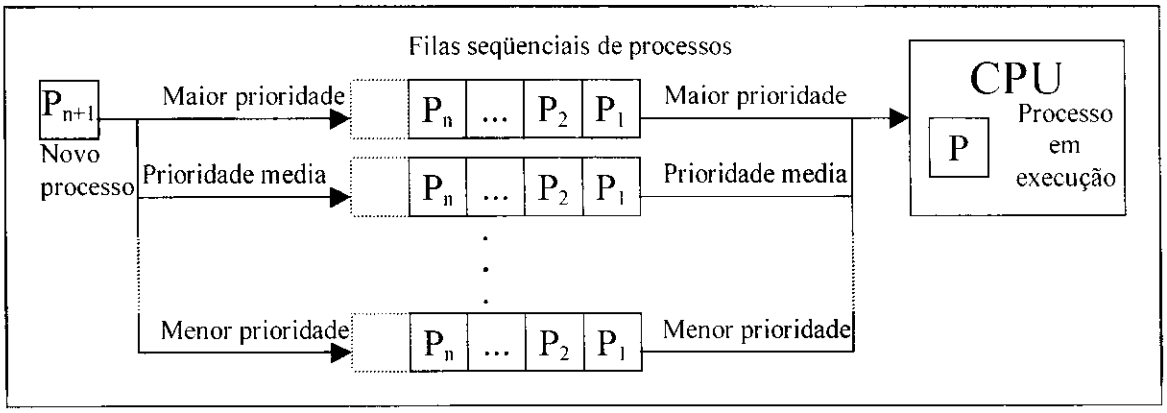


Figura 2.6 – Escalonamento com prioridade.

Muitas outras formas de escalonamento, também de igual importância, foram propostas, tais como as filas múltiplas com várias listas com prioridade, mas com um *quantum* diferente em cada lista, de tal forma que um processo que precisasse de um tempo muito grande receberia um *quantum* cada vez maior a cada reescalonamento[4], entretanto a sua prioridade iria diminuindo. Já o escalonamento "menor *job* primeiro" procurava dar prioridade ao *job* com menor tempo de processamento, de tal forma que um *job* que consumia pouco tempo de processamento era processado primeiro para aumentar o desempenho do sistema, enquanto um *job* com um tempo de processamento muito grande era prejudicado neste tipo de escalonamento. Esses escalonamentos e muitos outros existentes e que não foram apresentados aqui, possuem suas vantagens e desvantagens, sendo que a utilização de cada um deles depende do tipo de problema que se pretende resolver.

2.8. Características de escalonamento

Os diferentes algoritmos de escalonamento deram origem a uma grande variedade de tipo de escalonadores. Algumas taxinomias foram apresentadas (Kopetz, 1995, p.491) na tentativa de

[4] Este termo é utilizado para indicar a troca de contexto de processo na memória, ou seja, é o procedimento de liberação da CPU por um processo (guardando as informações deste) e a entrada de um outro processo para execução (carregando as suas informações).

estabelecer uma classificação para os diferentes tipos de escalonadores existentes. As classificações mais comuns são relacionadas a seguir, sendo que cada uma delas apresenta suas vantagens e desvantagens, ficando a cargo do sistema operacional ou do projetista a escolha pela utilização delas. Mullender (1995) relaciona as características listadas a seguir através de hierarquias, ou seja, uma dependente de outra ou como subdivisão de outra. Entretanto as taxonomias mais utilizadas são explicadas a seguir.

2.8.1. Quanto à utilização

- Propósito geral – o escalonador é utilizado em diferentes aplicações e precisa aplicar um bom algoritmo de escalonamento que beneficie todas as aplicações. A maioria dos escalonadores presentes nos sistemas operacionais é de propósito geral;

- Específico – o escalonador trabalha para apenas uma aplicação e precisa atender as necessidades específicas dessa aplicação. Alguns sistemas são desenvolvidos para gerenciar um problema específico, como o caso dos sistemas em tempo real que devem atender às rigorosas restrições de tempo necessárias em controle de processos, por exemplo;

2.8.2. Quanto à restrição de tempo

- Tempo real flexível – o escalonamento não é totalmente preso às restrições de tempo, ou seja, é tolerável que em certas circunstâncias nem todas as tarefas sejam executadas dentro do tempo predefinido. A grande maioria dos sistemas apresenta essa característica, pois as solicitações de grande parte dos usuários não exigem precisão no tempo de execução. Normalmente, os processos possuem um tempo limite (do inglês *deadline*) bem flexível para a execução;

- Tempo real inflexível – as tarefas precisam ter seu escalonamento garantido e as restrições de tempo precisam ser rigorosamente satisfeitas. Alguns sistemas específicos apresentam essa característica quando utilizados em controle de processos, controle de tráfego aéreo, controle de usinas nucleares e outros, que impõem rigorosas restrições de tempo. Como exemplo, pode-se afirmar que um atraso de alguns milissegundos é totalmente tolerável em um sistema bancário, mas esse mesmo atraso pode ser extremamente prejudicial em um sistema de controle de tráfego aéreo;

2.8.3. Quanto à reorganização

- Estático – o escalonamento é feito "*of-line*", ou seja, todas as informações sobre as tarefas devem ser conhecidas para que o escalonador monte a lista de processos escalonáveis antes do processamento. A execução de cada processo é feita a partir dessa lista sem ser permitida alterações durante o escalonamento. Os sistemas que implementam essa característica normalmente fazem periodicamente uma reorganização da lista de processos para inserir os novos processos que foram criados após o escalonamento. A lista de processo é reorganizada de acordo com o tipo de escalonamento utilizado (prioridade, menor *job* primeiro etc.). O termo reescalonamento foi evitado para não confundir com a preempção de um processo. Este termo foi utilizado anteriormente para indicar que um processo foi interrompido e enviado para o final da fila de processos em espera.

- Dinâmico – o escalonamento é feito "*on-line*" no momento que cada processo é criado, mesmo que já existam processos em execução, ou seja, as decisões de escalonamento são tomadas em tempo de execução baseadas nos requerimentos correntes. Os processos vão sendo posicionados dinamicamente na lista de processos de acordo com o tipo de escalonamento utilizado (prioridade, menor *job* primeiro etc.). Essa característica é vantajosa, pois não há a necessidade de interromper o escalonamento para a reorganização da lista de processos. Entretanto, é gasto algum tempo para a verificação de novos processos e a sua inserção na fila de processos;

2.8.4. Quanto ao conhecimento das informações sobre os processos

- Determinístico – o escalonamento é determinado antes da execução dos processos. É possível conhecer todas as informações sobre as tarefas antes de suas execuções e inferir como elas vão se comportar durante o tempo em que terão o controle da CPU. A maioria dos sistemas apresenta essa característica, mesmo sabendo-se que uma vez que o processo assume o controle da CPU, não é garantido que tudo ocorrerá conforme o previsto, principalmente com relação ao tempo de processamento;

- Não determinístico – as informações sobre os processos não podem ser conhecidas antes de suas execuções. O escalonamento neste caso precisa ser dinâmico para que os novos processos sejam escalonados em tempo de execução. Alguns sistemas específicos como sistemas de controle de processos em tempo real com realimentação, precisa apresentar

essa característica, pois dependendo da realimentação, podem ser gerados novos processos ou alteradas as informações dos processos existentes, o que exige uma nova reorganização da lista de processos;

2.8.5. Quanto ao compartilhamento da CPU

- Preemptivo – Um processo pode ter seu processamento interrompido antes mesmo do seu processamento ter terminado, permitindo assim que outro processo utilize a CPU. Isso pode ocorrer por vários motivos, tais como fim do *quantum* e requisição de serviço. A grande maioria dos sistemas implementam essa característica, pois é ela que permite que uma CPU seja compartilhada por vários processos, tirando o máximo proveito da máquina;

- Não preemptivo – os processos não podem ter seu processamento interrompido antes do seu final. Um processo, após receber o controle da CPU, só a libera quando o seu processamento chega ao final, mesmo que o processo em execução faça uma solicitação de serviço, como um acesso a disco, e seja necessário esperar pela resposta dessa solicitação. Isso mantém a CPU ociosa, impedido-a de realizar o processamento de um outro processo. Segundo Kopetz (1995, p.492), o escalonamento não preemptivo é razoável em um cenário onde muitas tarefas curtas têm de ser executadas;

2.8.6. Quanto à localização

- Centralizado – o escalonamento é feito em um processador dedicado em sistemas distribuídos. A centralização do escalonamento facilita o controle dos processos, evitando redundância ou desatualização;

- Descentralizado – o escalonamento é feito em vários processadores. A principal finalidade dessa característica é implementar a tolerância a falhas no sistema. Com o escalonamento descentralizado, se ocorrer algum problema com uma das máquinas que faz o escalonamento, nem todo o sistema é afetado, pois ainda existem outros processos em outras máquinas prontos para serem escalonados. Uma outra vantagem é o ganho de tempo de escalonamento, pois como os processos estão distribuídos em porções menores e escalonados paralelamente, o sistema se torna mais rápido;

2.8.7. Quanto à adaptabilidade

- Adaptável – o escalonamento pode se auto modificar de acordo com resultados anteriores. Isto significa que em determinada situação, o escalonamento pode ser feito por prioridade ou por revezamento, de acordo com informações obtidas previamente. Em algumas situações onde a consistência não é garantida ou não é desejada, um escalonamento adaptável passa a ser interessante, pois em um determinado momento pode haver uma incidência muito grande de processos pequenos com muitas operações de E/S, enquanto num outro momento esses processos são inexistentes, dando lugar a uma grande quantidade de processos grandes sem operações de E/S mas com muito processamento e;

- Não adaptável – o escalonamento é sempre o mesmo, mantendo sua estrutura inalterável. O escalonamento é realizado de acordo com um determinado tipo, mesmo que a demanda de diferentes processos varie de um período para outro. A vantagem deste escalonamento é a garantia de um mesmo mecanismo, visto que se um usuário interativo precisar, por exemplo, utilizar o sistema por várias horas do dia e vários dias seguidos, ele deve querer que o sistema seja consistente, pois é inaceitável que um processo que foi executado em segundos passe a ser executado em minutos ou horas.

2.9. O escalonador em tempo real

Ao ser desenvolver um escalonador deve-se considerar os conceitos básicos e os algoritmos de escalonamento existentes, além de se poder utilizar as características de escalonamento de acordo com sua finalidade. O desenvolvimento do escalonador proposto nesta dissertação foi definido a partir de estudos realizados sobre o escalonador em tempo real utilizado por Alsina & Cavalcanti (1997). Eles utilizaram com sucesso um escalonador em tempo real, o qual deu origem à idéia de se desenvolver um escalonador com procedimentos inteligentes. O algoritmo de escalonamento base foi o escalonamento em fila, devido a não haver necessidade de preempção de tarefas, reduzindo bastante a complexidade do mesmo. As características mais importantes incorporadas foram: escalonamento específico, tempo real inflexível, dinâmico, determinístico, não preemptivo, centralizado e não adaptável.

O escalonamento deve ser específico, pois é evidente que ele não vai tratar de tarefas de usuários interativos ou usuários *batch*; ele deve tratar de tarefas relacionadas com acionamento de motores, leitura de detectores de posição, entre outras específicas da aplicação ro-

bótica. O escalonamento em tempo real inflexível é fundamental para garantir que todas as tarefas sejam executadas dentro de uma rigorosa restrição de tempo, garantindo o sucessos do controle e acionamento dos motores.

O escalonamento deve ser dinâmico para permitir que durante o escalonamento em tempo real, seja possível alterar a ordem de execução das tarefas a partir de informações adquiridas do ambiente. O escalonamento deve também ser determinístico visto que as tarefas são todas conhecidas. A garantia de que o escalonamento não será preemptivo elimina muitos problemas tradicionais dos sistemas operacionais ("*deadlock*", condição de corrida e outros), pois uma tarefa jamais pode ser interrompida depois que a mesma assume o controle do processador.

O fato do escalonamento não ser preemptivo teve sua importância evidenciada depois que análises exaustivas na utilização de aplicações similares demostraram que o tempo de execução das tarefas é pequeno (Alsina & Cavalcanti, 1997), não havendo problema de tempo para a execução do conjunto de tarefas a cada interrupção. É evidente que o escalonamento deve ser centralizado pois esse tipo de aplicação é característico de sistemas centralizados. Por último, é considerado o escalonamento não-adaptável pois, como o escalonamento é específico, o tipo das tarefas é conhecido sem a necessidade de mudança no tipo de escalonamento.

Considerando-se todas essas informações sobre o escalonamento, definiu-se as informações necessárias que devem constar nos descritores das tarefas, ilustradas no Quadro 2.1. O fato do escalonamento ser não preemptivo, permitiu reduzir significativamente a quantidade de informações no descritor de tarefas: *id*, identifica a tarefa. É usado também para localizar o código da tarefa a ser executado; *st* guarda o estado da tarefa. Os possíveis estados que as tarefas podem assumir são: *pronta*, *bloqueada* e *executando*; *tempo* indica o intervalo de tempo que a tarefa deve esperar para ser executada; *freq* indica a freqüência de reativação[5] da tarefa, isto é, a tarefa pode ser ativada várias vezes com o intervalo de ativação igual a *freq*.

Quadro 2.1 – Descritor de tarefas.

[5] Termo utilizado para indicar que uma tarefa é novamente acionada, diferindo de reescalonamento que representa a mudança de contexto, ambos definidos nesta dissertação.

Descritor(*id, st, tempo, freq*)

id - identificação da tarefa
st - estado atual da tarefa
tempo - intervalo de tempo para acionamento
freq - freqüência de reacionamento

O Escalonador de tarefas em tempo real, utilizado em (Cavalcanti et al., 1994), (Lima et al., 1994) e (Alsina & Cavalcanti, 1997), foi projetado para manipular as tarefas de acordo com as informações do descritor apresentado no Quadro 2.1. O algoritmo simplificado desse escalonador é descrito no Quadro 2.2. A cada interrupção de tempo que ocorre a cada milissegundo, o escalonador verifica o *tempo* de espera de cada tarefa; se esse for igual a zero, o escalonador muda o estado (*st*) da tarefa para 1 e atualiza o *tempo* com o conteúdo de *freq*, que define a freqüência de reativação.

Quadro 2.2 – Algoritmo simplificado do escalonador em tempo real.

```
Faça {
    Para i = 1 até total_tarefas
        Se (tempo == 0)
            st = 1;
            tempo = freq;
        Senão
            tempo = tempo - 1;
        Se (st == 1)
            Localiza código da tarefa;
            st = -1;
            Executa tarefa;
    Fim para
}(a cada interrupção)
```

Nesse sistema, todas as tarefas são criadas com *st* igual a 0 (*bloqueadas*); durante o escalonamento, elas podem passar para o estado *pronta* (*st == 1*)[6] ou para o estado *executando* (*st == -1*). Caso o *tempo* seja diferente de zero, então o escalonador decrementa o *tempo* de uma unidade. Cada unidade representa 1 ms, visto que o escalonador é executado a cada 1 ms. Isso permite que as tarefas sejam programadas para serem ativadas de acordo com um tempo predefinido em ms. Em seguida, o escalonador verifica o estado da tarefa (*st*); se esse for

[6] O sinal == é utilizado nesta dissertação para representar a operação de igualdade em oposição ao sinal = que representa a operação de atribuição.

igual a 1, ou seja *pronta*, ele localiza o código da tarefa para execução, muda o estado da mesma para -1 (*executando*) e executa a tarefa.

Todas as tarefas são sempre mantidas em uma única lista de tarefas, independente do estado de cada uma delas. Este procedimento foi adotado devido ao número total de tarefas do sistema ser pequeno e a mudança dos descritores das tarefas de uma lista para outra consumir muito tempo de processamento.

2.10. Conclusão

Apresentou-se neste capítulo a definição de um escalonador de tarefas para uma aplicação robótica. Evidenciou-se que os conceitos de tarefa e processos são tidos por muitos autores como conceitos similares. Entretanto, para o autor desta dissertação, é possível diferenciá-los, destacando que o conceito de tarefa é mais importante para este trabalho.

Capítulo 3

Lógica nebulosa

3.1. Resumo

Apresenta-se neste capítulo uma breve introdução sobre a utilização da lógica nebulosa na representação do conhecimento impreciso. Em seguida, descreve-se os seus fundamentos, destacando-se a definição de conjuntos nebulosos e a sua normalização. Apresenta-se também as principais operações com conjuntos nebulosos, a definição de variáveis nebulosas e de regras nebulosas. Finaliza-se o capítulo com o estudo da utilização da lógica nebulosa, evidenciando-se a seqüência na qual cada conceito deve ser empregado para que dela se tire o melhor proveito.

3.2. Introdução

A utilização da lógica *booleana* foi de vital importância no desenvolvimento dos computadores. A sua principal característica de operar com apenas dois valores lógicos (falso ou verdadeiro) possibilitou a sua utilização na concepção dos sistemas elétricos digitais e por conseqüência no desenvolvimento de programas computacionais. Entretanto, com o avanço das pesquisas na área de Inteligência Artificial (IA) sobre a utilização de mecanismos computacionais para permitir que a máquina se comporte de uma maneira inteligente, percebeu-se que uma grande parte dos problemas a serem solucionados através de computador não poderia ser representado por apenas dois valores lógicos, pois se a máquina deve imitar a inteligência humana, então ela deve ser capaz de operar com valores imprecisos ou vagos, como os seres humanos o fazem naturalmente. Por exemplo, um ser humano é capaz de definir a altura de uma pessoa, através de medidas imprecisas como alto, baixo, mediano, baixinho e outros. Esse tipo de raciocínio humano pode ser definido como um raciocínio aproximado, o qual pode ser representado através de uma lógica nebulosa (Zadeh, 1994, p.xvii). Uma representação gráfica convencional de valores na lógica nebulosa é ilustrada na Figura 3.1, em que a

altura das pessoas é representada na abscissa e três funções (baixo, mediano e alto) representam a classificação das pessoas quanto à altura.

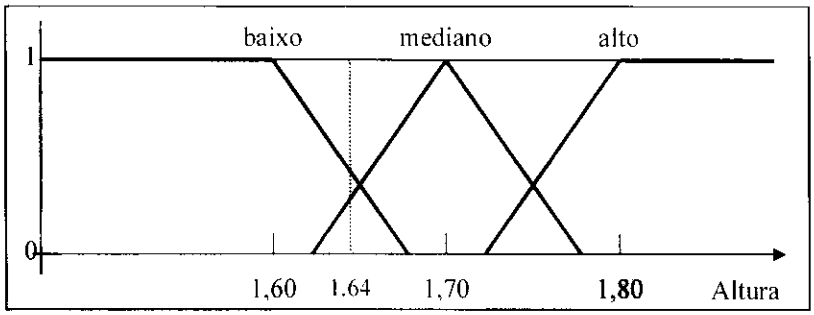


Figura 3.1 – Representação de valores na lógica nebulosa.

Uma pessoa medindo 1,64 m de altura é considerada baixa, de acordo com o gráfico da Figura 3.1, mas está muito próxima de ser considerada de altura mediana. O gráfico acima contém elementos que serão discutidos nas seções seguintes. Entretanto, é possível observar que uma pessoa é totalmente baixa de 0 até 1,60 m (a curva "baixo" indica valor 1 e as demais valor 0 na faixa de valores indicada). A partir de 1,60 m a reta que define o valor baixo começa a decrescer, enquanto a reta que define a altura mediana começa a crescer. Uma pessoa com 1,70 m é considerada de altura mediana e acima de 1,80 m é considerada alta. Essas medidas são totalmente imprecisas, variando de acordo com os conceitos de cada pessoa, região, cidade, país e outros. Em muitas situações porém, os valores inexatos são mais importantes e possuem significados mais expressivos do que os valores exatos.

Dada as suas características, a lógica nebulosa é atualmente considerada uma poderosa ferramenta de IA, sendo utilizada em classificações de padrões, sistemas baseados em conhecimento e sistemas de controle de processos, entre outros.

3.3. Fundamentos da lógica nebulosa

A lógica nebulosa foi proposta por Lotfi A. Zadeh em 1965 como uma matemática que podia representar as incertezas do cotidiano (Bezdek, 1994, p.3); é basicamente uma linguagem que serve para descrever e analisar dependências imprecisas (Zadeh, 1994, p.xvii). Diante dos problemas da lógica booleana e dos recursos oferecidos pela lógica nebulosa, muitos pesquisadores passaram a utiliza-la como ferramenta para o desenvolvimento de sistemas inteligentes. Atualmente, há uma grande variedade de pesquisas envolvendo a lógica nebulosa.

A lógica nebulosa é fundamentada na teoria dos conjuntos, absorvendo as principais características desta última, mas apresentando as suas próprias características que lhe permite ser vista como uma nova abordagem no tratamento de problemas inexatos. Enquanto na teoria dos conjuntos um determinado elemento pertence ou não a um conjunto e a cada elemento é associado o valor 1 ou 0, descriminando-o como membro ou não membro do conjunto, respectivamente (Klir & Folger, 1988, p.10), na lógica nebulosa um elemento pode assumir qualquer valor dentro do intervalo fechado de 0 a 1, de tal forma que esse número indica o grau de pertinência do elemento dentro de um conjunto. Isto é, na lógica nebulosa, um elemento pode não pertencer a um conjunto (valor 0, nenhuma pertinência), pode pertencer ao conjunto (valor 1, pertinência total) ou pode pertencer parcialmente ao conjunto (0< valor < 1, pertinência parcial). O grau de pertinência de um elemento é obtido através de uma função de pertinência (do inglês *membership function*), que pode ser definida formalmente. Seja $\mathcal{U}$ o conjunto universal e X ($X = \{x \mid x \in \mathcal{U}\}$) um subconjunto de $\mathcal{U}$ ($X \supset \mathcal{U}$). A função de pertinência do subconjunto X é definida pela Equação 3.1, na qual [0,1] representa o intervalo fechado dos números reais de 0 a 1 (Kartalopoulos, 1996, p.122) (Klir & Folger, 1988, p.10) e x associado a cada valor no intervalo [0, 1], indica a possibilidade de x pertencer a X.

$$\mu_X(x): \Re \rightarrow [0,1] \tag{3.1}$$

O exemplo a seguir é utilizado para expressar essa definição. Considere que existe uma loja que vende árvores de todos os tipos. Então, seja a o tamanho de uma árvore e G o conjuntos das arvores grandes no universo $\mathcal{U}$ de todas as árvores da loja, a função de pertinência $\mu_G(a)$ define o quanto a árvore a é grande, ou seja, o quanto a pertence a G. Se $\mu_G(a)=0$, então a não é grande (nenhuma pertinência). Se $\mu_G(a)=1$, então a é grande (pertinência total). Se $0 < \mu_G(a) < 1$, então a tem uma pertinência relativa no conjunto grande.

Definindo-se os conjuntos das árvores muito pequenas (MP), pequenas (P) e médias (M) para o exemplo do parágrafo anterior, tem-se quatro funções de pertinência para o conjunto de todas as árvores da loja. Assim, de acordo com o gráfico ilustrado na Figura 3.2, é possível ter uma mesma árvore a pertencendo a mais de um conjunto. A árvore de 160 cm, indicada pela linha pontilhada na Figura 3.2, pertence parcialmente ao conjunto das árvores médias e das árvores grandes, sendo a sua pertinência maior no conjunto das árvores médias (0,6) do que no conjunto das árvores grandes (0,4). Essa pertinência parcial não é admissível

na teoria dos conjuntos, na qual um elemento pertence ou não pertence a um determinado conjunto.

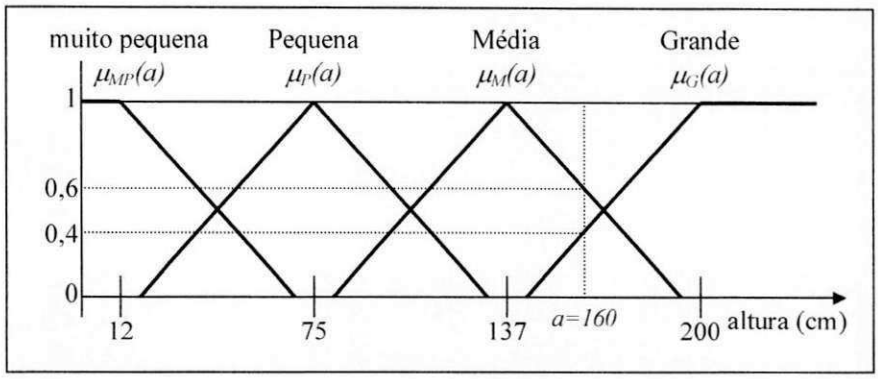


Figura 3.2 – Conjuntos nebulosos dos tamanhos das árvores de uma loja.

Os conjuntos definidos no parágrafo anterior e ilustrados na Figura 3.2, são chamados de conjuntos nebulosos. As funções relacionadas com cada conjunto são chamadas de funções de pertinência, o valor *a* é normalmente chamado de alvo e o gráfico completo apresentado na Figura 3.2 é chamado de gráfico nebuloso. Uma outra definição formal de um conjunto nebuloso pode ser obtida pelo par ordenado, descrito na Equação 3.2, na qual X é o conjunto nebuloso, x é um valor pertencente a X e $\mu_X(x)$ a função de pertinência de X. Cada função só pode assumir valores no intervalo fechado [0, 1]. A restrição das funções a esse intervalo permite a normalização da lógica nebulosa.

$$X = \{x, \mu_X(x)\} \tag{3.2}$$

A representação das funções de pertinência depende do tipo de equação escolhida ou das equações escolhidas para cada função. A representação mais utilizada é a função triangular ilustrada na Figura 3.3a. Entretanto, essa representação depende do tipo de problema a ser tratado, sendo necessário empregar outros tipos de representação de acordo com cada tipo de problema. As representações mais utilizadas são ilustradas na Figura 3.3. Observa-se que algumas funções (triangular e trapezoidal) necessitam de mais de uma equação. A representação triangular utiliza duas equações (Figura 3.3a) uma para cada reta e é a representações mais simples devido a sua derivada ser constante. Já a representação trapezoidal utiliza três equações (Figura 3.3b) e é uma variação da triangular. As outras duas representações, sigmóide (Figura 3.3c) e parabólica (Figura 3.3d), utilizam apenas uma equação, sendo a sigmóide adequada para controle neuro-nebuloso e a parabólica ideal para certos tipos de problemas que possam tirar proveito da sua característica de apresentar a segunda derivada constante.

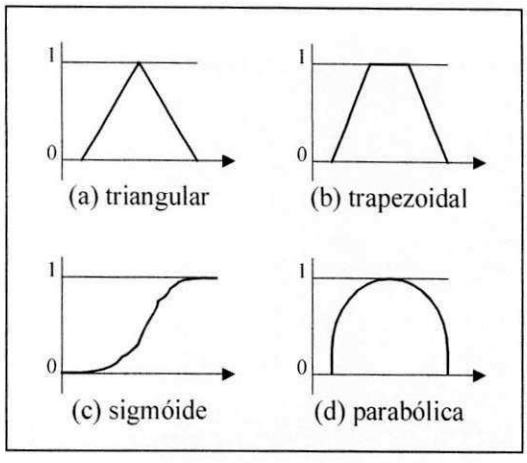


Figura 3.3 – Representações de funções de pertinência.

3.4. Operações com conjuntos nebulosos

A utilização dos conjuntos nebulosos a partir das definições apresentadas até aqui, é feita através de operações, herdadas da teoria dos conjuntos, as quais formam a lógica nebulosa. Segundo Klir & Folger (1988, p.37), a teoria original dos conjuntos nebulosos foi fundamentada nas termos das três operações "*união*", "*interseção*" e "*complemento*" que são equivalentes às operações "*ou*", "*e*" e "*negação*" da lógica *booleana*, respectivamente. Essas operações são apresentadas a seguir. Outras operações, também de grande importância para a lógica nebulosa, foram propostas como a operação "*comparação*" e "*contido*", também apresentadas a seguir. Assume-se para cada operação as seguintes premissas: X é um conjunto nebuloso universal, x é um elemento de X ($x \in X$) e A e B são dois subconjuntos nebulosos de X, ou seja A e $B \supset X$.

3.4.1. Complemento

A operação complemento é utilizada para definir a função de pertinência oposta de um subconjunto, ou seja, o complemento do subconjunto A, definido como A', é formado pelos pontos opostos de A de dentro do intervalo [0, 1]. Essa operação, quando tratada nos extremos desse intervalo, é equivalente à operação "negação" da lógica *booleana*. A representação formal da operação complemento é descrita na Equação 3.3. A sua representação gráfica é ilustrada na Figura 3.4.

$$\mu_{A'}(x) = 1 - \mu_A(x) \quad \forall x \in X \tag{3.3}$$

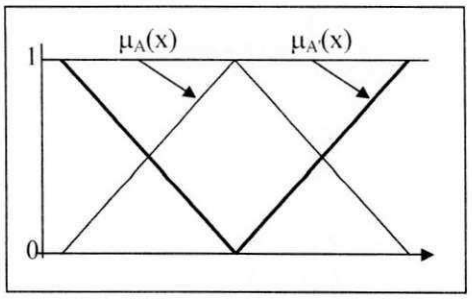


Figura 3.4 – Representação gráfica da operação complemento.

3.4.2. União

A operação união é utilizada para associar dois subconjuntos, ou seja, a união do subconjunto *A* com *B* resulta em um subconjunto abrangendo os pontos máximos dos dois subconjuntos unidos. Essa operação, quando tratada nos extremos do intervalo [0, 1], é equivalente à operação "ou" da lógica *booleana*. A representação formal da operação união é descrita na Equação 3.4. A sua representação gráfica é ilustrada na Figura 3.5.

$$A \cup B \rightarrow \mu_A(x) \cup \mu_B(x) = \max(\mu_A(x), \mu_B(x)) \ \forall x \in X \tag{3.4}$$

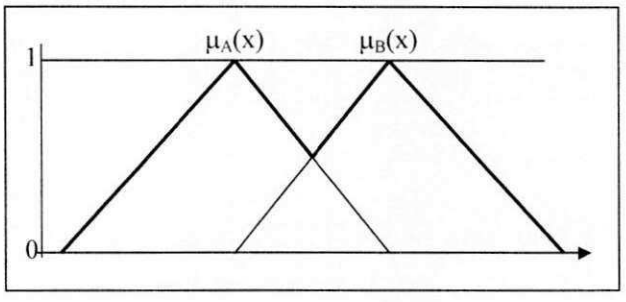


Figura 3.5 – Representação gráfica da operação união.

3.4.3. Interseção

A operação interseção é utilizada para definir a região comum entre dois subconjuntos, ou seja, a interseção do subconjunto *A* com *B* resulta em um subconjunto abrangendo os pontos que pertencem tanto ao subconjunto *A* quanto ao subconjunto *B*. Essa operação, quando tratada nos extremos do intervalo [0, 1], é equivalente à operação "e" da lógica *booleana*. A representação formal da operação interseção é descrita na Equação 3.5. A sua representação gráfica é ilustrada na Figura 3.6.

$$A \cap B \rightarrow \mu_A(x) \cap \mu_B(x) = \min(\mu_A(x), \mu_B(x)) \ \forall x \in X \tag{3.5}$$

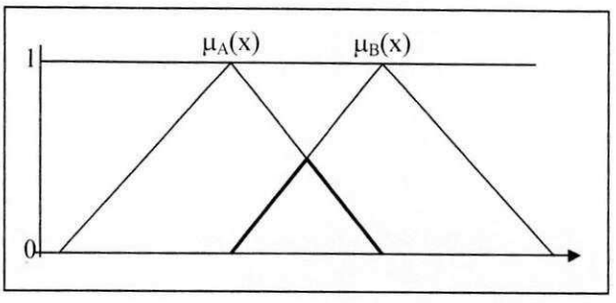


Figura 3.6 – Representação gráfica da operação interseção.

3.4.4. Comparação

A operação comparação é utilizada para definir se dois subconjuntos são iguais, ou seja, se as funções de pertinência dos dois subconjuntos são equivalentes. A representação formal da operação comparação é descrita na Equação 3.6. A sua representação gráfica é ilustrada na Figura 3.7. Esta figura representa os dois conjuntos iguais.

$$A = B \rightarrow \mu_A(x) = \mu_B(x) \ \forall x \in X \tag{3.6}$$

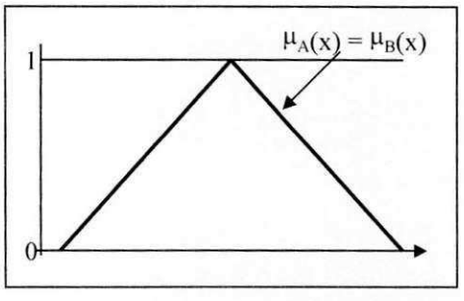


Figura 3.7 – Representação gráfica da operação comparação.

3.4.5. Contido

A operação contido é utilizada para definir se um subconjunto está contido em outro subconjunto, ou seja, A está contido em B se para qualquer x, o resultado da função de pertinência de A é sempre menor o igual ao resultado da função de pertinência de B. A representação formal da operação contido é descrita na Equação 3.7. A sua representação gráfica é ilustrada na Figura 3.8.

$$A \subset B \rightarrow \mu_A(x) \leq \mu_B(x) \ \forall x \in X \tag{3.7}$$

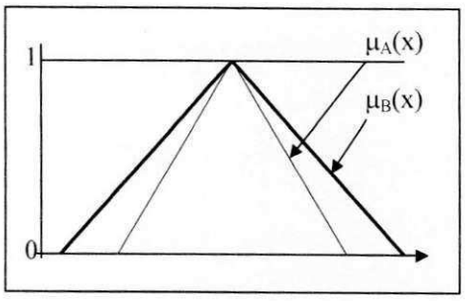


Figura 3.8 – Representação gráfica da operação contido.

3.5. Variáveis lingüísticas

A associação de valores lingüísticos a um conjunto nebuloso (pequeno, médio, grande) permite que um problema seja tratado através de uma representação mais flexível, evitando a rigidez de um tratamento preciso, expresso através de valores quantitativos. A lógica nebulosa pode ser usada para construir um vocabulário ou conjunto de termos lingüísticos (Yen et al., 1995, p.17), que pode ser usado na avaliação de algum problema.

As variáveis utilizadas para receber valores lingüísticos são definidas como variáveis lingüísticas em oposição às variáveis que armazenam valores quantitativos (valores numéricos) definidas como variáveis quantitativas ou simplesmente variáveis. A diferença entre uma variável lingüística e uma variável quantitativa pode ser evidenciada pelo exemplo a seguir.

Considere que um instrutor deve ensinar para um aluno como dirigir um carro e que, na hora de acelerar, ele deve dizer o quanto o aluno precisa pisar no pedal do acelerador. Neste caso, o instrutor jamais utilizará qualquer sistema de medida preciso para dizer o quanto o aluno deve acelerar (acelere 2,5 Pascal – medida de pressão, por exemplo). Ele deve utilizar um sistema de medida impreciso como acelere pouco, médio ou muito. Uma variável lingüística tem como característica assumir valores dentro de um conjunto de termos lingüísticos, ou seja frases ou palavras (Sales Jr., 1997, p.29). Assim, considerando-se a aceleração uma variável lingüística, ela pode assumir os valores "pouca", "media" ou "muita", conforme ilustrado no gráfico da Figura 3.9.

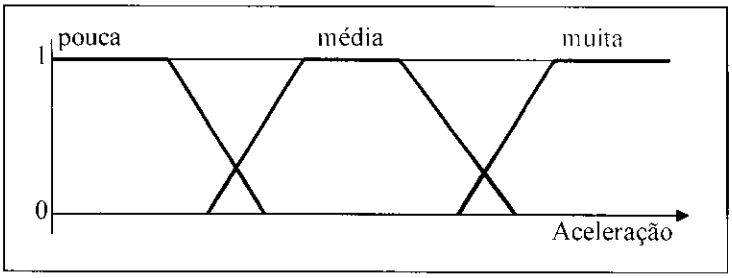


Figura 3.9 – Variáveis lingüísticas.

O conhecimento armazenado ou representado por cada uma das funções de pertinência, associadas a valores lingüísticos, é de grande importância para que o instrutor possa transmitir informações para o aluno e para que este último, aprenda mais rápido, pois caso essa informação fosse transmitida através de valores precisos haveria dificuldade, tanto para o instrutor ensinar quanto para o aluno aprender. Considerando-se que neste caso o instrutor fosse um computador, o problema de comunicação aumentaria, pois enquanto o aluno processa informações imprecisas com grande facilidade, o computador já não o faz com tanta destreza. Entretanto, com a utilização da lógica nebulosa em computadores, esse problema diminui consideravelmente.

3.6. Regras nebulosas

A representação de conhecimento na lógica nebulosa é fundamentada principalmente através de regras do tipo *se-então-senão* que permitem o relacionamento entre variáveis lingüísticas e os valores de entrada e saída de um problema. Considerando-se o problema da aceleração apresentado na seção com um computador como instrutor, as instruções para o aluno poderiam ser inferidas a partir das regras nebulosas que relacionam as variáveis nebulosas com as informações atuais sobre o veículo. A utilização de regras dá à lógica nebulosa uma capacidade ainda maior de representação do conhecimento impreciso. Alguns exemplos de regras nebulosas são descritos no Quadro 3.1.

Quadro 3.1 – Regras nebulosas.

Regra1: ***Se*** *velocidade == baixa e aceleração == media* ***então*** *aceleração = muita*
Regra2: ***Se*** *velocidade == alta e aceleração == muita* ***então*** *aceleração = media*
Regra3: ***Se*** *velocidade == alta e aceleração == media* ***então*** *aceleração = baixa*
Regra4: ***Se*** *velocidade == baixa e aceleração == baixa* ***então*** *aceleração = muita*
Regra5: ***Se*** *aceleração == muita* ***então*** *aceleração = baixa*

Uma característica muito importante nas regras nebulosas é que na maioria dos problemas elas são geradas a partir de experiências passadas (Kartalopoulos, 1996, p.125) ou de resultados antecedentes e conseqüentes de variáveis nebulosas (Yen, 1995, p.22). Isso permite que a lógica nebulosa seja utilizada com sucesso na resolução de problemas que utilizam realimentação, como no caso de controle de processos. Os exemplos de regras nebulosas descritos no Quadro 3.1 apresentam relacionamentos com variáveis nebulosas e valores anteriores a inferência em questão. Por exemplo, a regra 1 é avaliada para orientar o aluno a aumentar a aceleração se a velocidade é baixa e se a aceleração é média.

3.7. Utilização da lógica nebulosa

Os principais conceitos sobre a lógica nebulosa apresentados até aqui, precisam ser utilizados em conjunto para que se tire o melhor proveito dos recursos desta lógica, permitindo que cada problema seja representado de acordo com as suas características. Entretanto, há uma certa padronização no agrupamento de todos os conceitos que envolvem a lógica nebulosa.

Primeiro, o domínio do problema tem que ser bem definido e distribuído através das funções de pertinência. Em algumas situações é necessário normalizar as funções de pertinência, que devem cobrir todo o domínio. Pedrycz & Gomide (1998, p.13) apresentam a normalização como uma operação dos conjuntos nebulosos. A definição da altura de uma pessoa é utilizada como exemplo da operação de normalização. Inicialmente, divide-se todas as alturas pelo maior valor possível que a altura pode assumir. Neste caso, considera-se que a maior altura seja 3 m, dividindo-se então todas as altura por 3, tem-se o domínio normalizado ou por unidade (pu). Esse exemplo é ilustrado na Figura 3.10. Uma altura igual a 1,7 m é normalizada para 0,56 (1,7/3). A cada função de pertinência é associado um valor lingüístico (baixa, mediana e alta).

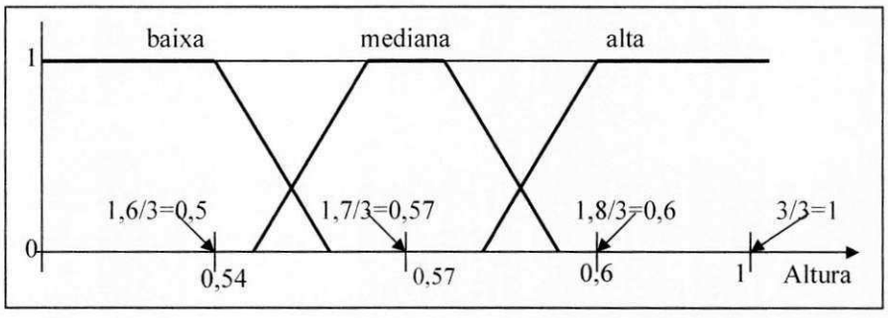


Figura 3.10 – Gráfico nebulosos da altura normalizada.

Segundo, deve ser feita a transformação de uma entrada quantitativa (por exemplo: altura = 0,56) em um dos valores lingüísticos (baixo, mediano ou alto). Este processo normalmente é conhecido como "*fuzzyficação*"[7]. A *fuzzyficação* permite maior flexibilidade para tratar os problemas do mundo real, especialmente em análise de decisão, teoria de probabilidade, domínio de controle e outros (Gupta & Sinha, 1996). Depois da *fuzzyficação* tem-se a variável nebulosa com o valor da entrada classificado em termos de valores lingüísticos.

Terceiro, a variável lingüística deve passar por um processo de inferência, através de regras nebulosas associadas ao problema em questão. Supõe-se como exemplo um sistema que deve fazer uma análise de um determinado perfil de uma pessoa para uma colocação profissional, então o sistema deve possuir regras que avaliem a altura da pessoa de acordo com requisitos preestabelecidos, como no caso das regras descritas no Quadro 3.2.

Quadro 3.2 – Regras nebulosas.

Regra1: ***Se*** *emprego == "atendente" e altura == "baixa"* ***então*** *perfil = "fraco"*
Regra2: ***Se*** *emprego == "atendente" e altura == "mediana"* ***então*** *perfil = "bom"*
Regra3: ***Se*** *emprego == "atendente" e altura == "alta"* ***então*** *perfil = "razoável"*

Quarto e último, os valores lingüísticos devem ser transformados novamente para valores quantitativos. No caso do exemplo acima, este passo não seria necessário, pois o resultado final dado através dos valores lingüísticos (fraco, razoável e bom) é o ideal para o tipo de problema em questão. Entretanto, quando a lógica nebulosa é utilizada em controle de processos, por exemplo, a saída deve ser um valor quantitativo, ou seja, deve-se transformar um saída do tipo (fraco, razoável, bom) em um valor do tipo (0,3; 0,6; 0,9). Este processo é conhecido com "*defuzzyficação*"[8].

A *defuzzyficação* pode ser feita de várias maneiras como estabelecendo-se uma tabela de valores correspondentes a cada função de pertinência, verificando-se qual o valor máximo da variável lingüística, e adotando-se esse valor como o resultado quantitativo. No exemplo citado acima, se a resposta nebulosa fosse *razoável*, o seu valor quantitativo seria então 0,6. Uma outra maneira de implementar a *defuzzyficação*, e a mais satisfatória, é utilizar o método de centro de gravidade explicado a seguir.

---

[7] Este termo foi utilizado por falta de um termo mais adequado para a língua portuguesa.

Por exemplo, supondo-se um controlador nebuloso (Cavalcanti, 1998, p.70) que pode ter a saída controlada definida de acordo com o gráfico nebuloso ilustrado na Figura 3.11, com sete valores lingüísticos possíveis (NG – Negativo Grande, NM – Negativo Médio, NP – Negativo Pequeno, ZE – Zero, PP – Positivo Pequeno, PM – Positivo Médio e PG – Positivo Grande), uma saída com valor quantitativo de 0,7 foi *fuzzyficada* no valor lingüístico PM.

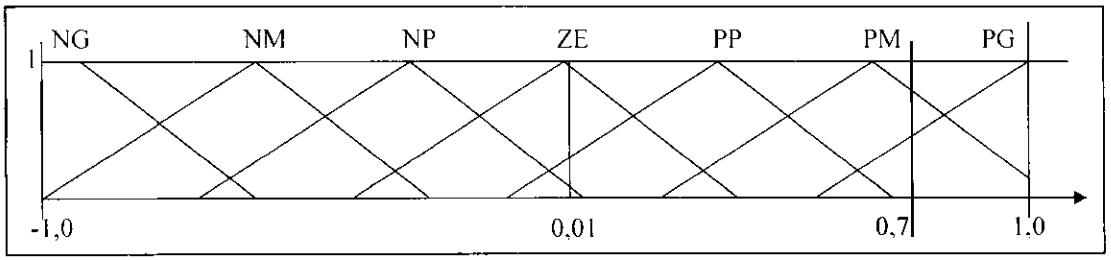


Figura 3.21 – Gráfico nebuloso da altura normalizada.

Esse valor pode ser transformado em uma saída quantitativa (*defuzzyficada*) de acordo com a Tabela 3.1. Nesse caso, 0,6 ($Saída_q$ – saída quantitativa) seria o valor correspondente a PM ($Saída_f$ – saída nebulosa). Entretanto, a saída que era 0,7 teve o seu valor alterado para 0,6 na saída do controlador nebuloso. Esse tipo de *defuzzyficação* é inadequado para determinados tipos de aplicação devido os valores de saída serem fornecidos sempre no extremo.

Tabela 3.1 – Tempo de execução dos processos.

$Saída_f$	NG	NM	NP	ZE	PP	PM	PG
$Saída_q$	-0,9	-0,6	-0,4	0,01	0,4	0,6	0,9

O método de centro de gravidade definido pela Equação 3.8 soluciona o problema dos extremos, tirando uma média de todas as funções de pertinência utilizados para cobrir todo o domínio. O objetivo da Equação 3.8 é somar todos os produtos entre o valor máximo (VD_k) de cada função de pertinência com o valor obtido na *fuzzyficação* do valor quantitativo ($\mu_X(x)$) e dividir pela soma de todos os valores obtidos na *fuzzyficação* do valor quantitativo ($\mu_X(x)$).

$$Saida_q = \frac{\sum_{j=1}^{k} \mu_X(x) VD_k}{\sum_{j=1}^{k} \mu_X(x)} \tag{3.8}$$

[8] Este termo também foi utilizado por falta de um termo mais adequado para a língua portuguesa.

Usando-se essa equação para a *defuzzyficação* do valor PM, observa-se que como o valor 0,7 tem pertinência somente em PM e PG, então somente essas duas funções de pertinência terão influência sobre o valor final, pois os outros produtos resultaram em zero, conforme exemplificado abaixo. O valor da *defuzzyficação* obtido é 0,73 que representa uma saída mais próxima da entrada sob controle.

$$Saida_q = \frac{0+0+0+0+0+\mu_{PM}(0{,}7)*0{,}6+\mu_{PG}(0{,}7)*0{,}9}{0+0+0+0+0+\mu_{PM}(0{,}7)+\mu_{PG}(0{,}7)} = \frac{0{,}75*0{,}6+0{,}6*0{,}9}{0{,}75+0{,}6} = 0{,}73$$

3.8. Conclusão

A lógica nebulosa foi descrita neste capítulo de forma resumida para introduzir os principais conceitos utilizados na implementação do Escalonador Inteligentes de Tarefas proposto no Capítulo 5. Observou-se que esta lógica possui características fundamentais para a solução de determinados tipos de problemas, principalmente aqueles relacionados com tomada de decisão sobre valores imprecisos.

Capítulo 4

A aplicação robótica

4.1. Resumo

Neste capítulo apresenta-se o sistema robótico utilizado na implementação do escalonador de tarefas em tempo real (ETR). Inicialmente, faz-se uma sucinta descrição de robôs cooperantes que foram utilizados para testes experimentais do ETR. Em seguida descreve-se a análise da operação dos manipuladores e as características básicas do protótipo. Apresenta-se também o sistema inteligente que controla o processo de transferência de carga, os atuadores e os sensores da estrutura robótica. Encerra-se o capítulo com uma análise dos resultados experimentais obtidos com o sistema robótico apresentado.

4.2. Robôs cooperantes

A cooperação entre robôs vem sendo estudada nos últimos anos por vários pesquisadores. Isso se deve a uma grande quantidade de aplicações que requerem dois ou mais robôs projetados independentemente ou localizados separadamente, mas que mutuamente influenciam um mesmo subsistema (Cui & Shin, 1996, p.206). Entre as diversas aplicações estudadas pode-se destacar a transferência de carga entre dois manipuladores (Osumi & Arai, 1994) (Alsina & Cavalcanti, 1997), o acoplamento de robôs (Handelman & Lane, 1996) e o futebol de robôs (Veloso et al., 1997) (Blythe & Veloso, 1997) (Stone & Veloso, 1998), entre outros.

Os robôs que interagem em um mesmo ambiente devem ser capazes de coordenar seus movimentos de tal forma que cada movimento seja realizado de acordo com o tempo e o estado atual do ambiente percebido pelo próprio robô. A realização de tarefas por mais de um robô depende diretamente da capacidade que os mesmos possuem de realizar tarefas coletivas. Por exemplo, a transferência de carga entre dois manipuladores exige que os manipuladores, além de realizarem suas tarefas de manipulação de objeto, interajam com o ambiente para identificarem o tipo de movimento que cada um deles deve realizar em determinados momentos, de tal forma que a meta estabelecida seja realizada da melhor maneira possível.

4.3. Os manipuladores robóticos

Um sistema robótico com dois manipuladores capazes de realizar transferência de carga entre eles foi proposto por Alsina & Cavalcanti (1997) para demostrar a utilização de um Sistema Inteligente (SI) no controle de uma estrutura robótica. O esquema geral desse sistema é ilustrado na Figura 4.1. O desenho do computador nesta figura representa o SI e a "Interface Motor CC" representa o circuito de acionamento dos motores (um estudo sobre essa interface é apresentado no Apêndice A). Cada manipulador é composto por dois elos, dois motores de corrente contínua (Motor CC) e um gancho utilizado para sustentar a carga. O primeiro motor (motor fixo) possui sensores ópticos para detecção de posição e pode ser posicionado entre os ângulos $-\frac{\pi}{2} \leq \theta_1 \leq \frac{\pi}{2}$. O segundo motor (motor livre) não possui detector óptico de posição, mas sim travas mecânicas que o forçam a permanecer na faixa de ângulos $-\frac{\pi}{4} \leq \theta_2 \leq \frac{\pi}{4}$.

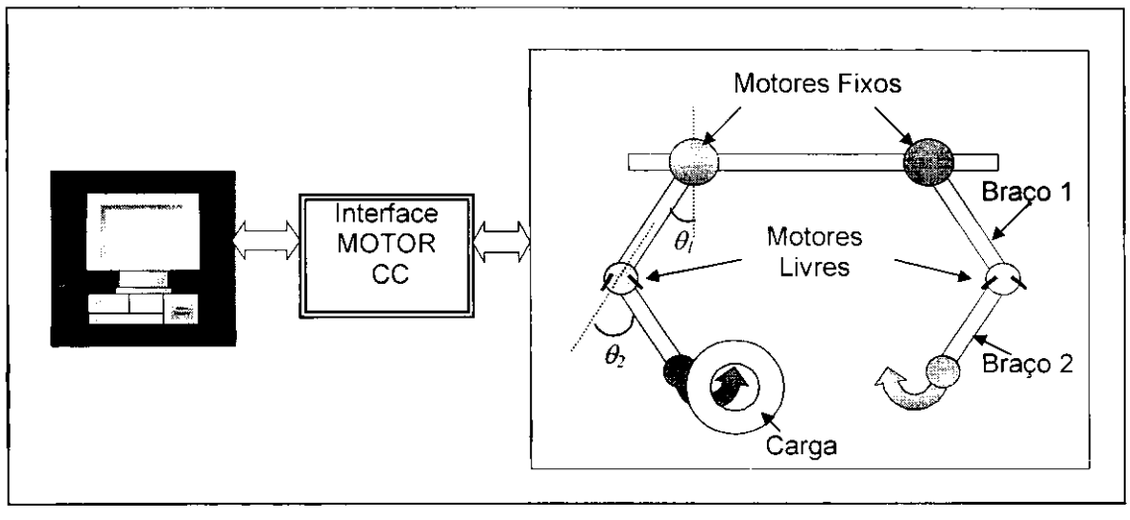


Figura 4.1 – Esquema geral de montagem da estrutura robótica.

Conforme ilustrado na Figura 4.1, os elos dos manipuladores foram montados na forma de pêndulos. Isso permitiu o estudo dos movimentos de cada manipulador através das leis físicas do pêndulo. Cavalcanti (1994) apresentou um estudo sobre o controle de um pêndulo e definiu todo o formalismo matemático necessário para controlá-lo através de um sistema inteligente. Ele demonstrou que esses movimentos estão relacionados com a tensão U aplicada sobre o motor e a posição angular θ do pêndulo gerada por essa tensão. Nesse mesmo estudo, Cavalcanti (1994) utilizou o conceito de estado passivo proposto por Cavalcanti et al. (1994) e Lima et al. (1994). Esse conceito foi fundamental para o controle do Motor CC através de um

controlador neural, pois permitiu a definição do torque do pêndulo com a carga (Tl)[9], conforme descrito nas Equações 4.1 e 4.2. O torque do pêndulo com a carga é descrito na Equação 4.1 e esse mesmo torque com o pêndulo em repouso é descrito na Equação 4.2. A representação de um pêndulo e seus componentes são ilustrados na Figura 4.2, na qual L representa o comprimento do pêndulo, P representa o peso (carga l) do pêndulo, θ representa o ângulo do pêndulo, θ'' representa o aceleração angular do pêndulo e g representa a aceleração da gravidade.

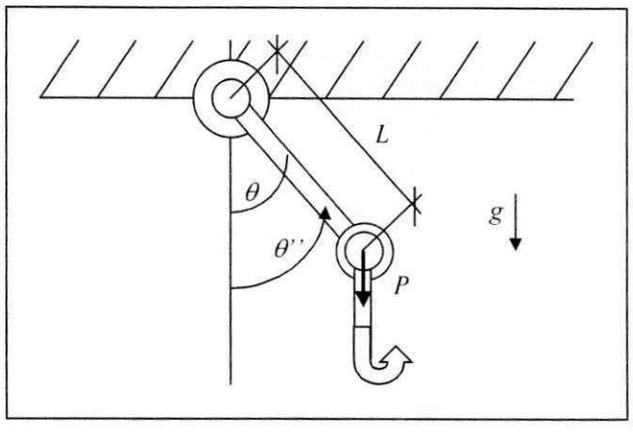


Figura 4.2 – Representação do manipulador em forma de pêndulo.

Cavalcanti et al. (1999a) apresentaram um pêndulo em repouso tal que para posicionar o pêndulo em um ângulo θ, o torque Tm gerado pelo motor deveria ser igual ao torque Tl do pêndulo ($Tm == Tl$), então o Motor CC deveria gerar um torque Tm para contrabalancear o torque Tl do pêndulo, definido na Equação 4.1. Utilizando-se o conceito de estado passivo que permite considerar a aceleração θ'' igual a zero, para movimentos pequenos do pêndulo com $\theta \cong 0$, Cavalcanti (1994) deduziu a Equação 4.2 para o torque Tl.

Assim, a Equação 4.2 pode ser substituída pela Equação 4.3, na qual PL é substituído pela constante k_1. Ainda para pequenos movimentos do pêndulo próximo de $\theta \cong 0$, pode-se considerar o torque do motor proporcional à tensão da armadura U, ou $Tm_{(t)} \cong k_2 U$. Considerando-se θ pequeno, pode-se aproximar $sen(\theta) = \theta$. O deslocamento do pêndulo em função da tensão de armadura é descrito na Equação 4.4.

---

[9] Tl = Torque do pêndulo com carga (l = carga, do inglês *load*). Esta denominação foi mantida devido haver uma certa padronização na utilização da mesma).

$$Tl_{(t)} = ML^2 \frac{d^2\theta_{(t)}}{dt^2} + PL\,\mathrm{sen}(\theta_{(t)}) = ML^2\theta'' + PL\,\mathrm{sen}(\theta_{(t)}) \tag{4.1}$$

$$Tl_{(t)} = PLSen(\theta_{(t)}) \tag{4.2}$$

$$Tm_{(t)} = \mathrm{k}_2\mathrm{U}=\mathrm{k}_1 Sen(\theta) \tag{4.3}$$

$$\theta = \frac{\mathrm{k}_2}{\mathrm{k}_1}\mathrm{U} = \mathrm{KU} \tag{4.4}$$

4.4. Especificação do problema

Alsina & Cavalcanti (1997) desenvolveram as estratégias de movimentação dos manipuladores robóticos, ilustrados na Figura 4.1, a partir de uma analogia com uma situação real. Eles observaram como um casal de cegos que pedia dinheiro na rua, sentados numa calçada, procedia para passar uma moeda de um para o outro. A seguir apresenta-se o procedimento de troca de moeda observado quando um dos cegos recebia uma esmola. Esse procedimento é realizado considerando-se somente informações "*proprioceptive*", isto é não há nenhum tipo de percepção visual.

a) Uma mulher e seu marido, ambos cegos e pobres, cantam e pedem dinheiro numa calçada de uma rua central da cidade;
b) Um dos passantes doa uma moeda à mulher;
c) Ela determina o valor da moeda, verificando o peso da mesma em sua mão, jogando-a para cima;
d) Ela informa ao marido que ganhou uma moeda de valor;
e) Ele pede a moeda;
f) Ela entrega a moeda a ele;
g) O marido recebe e guarda a moeda.

4.5. Especificação do plano de ação

A meta do sistema, baseada na analogia descrita na seção anterior, foi planejada a partir de uma seqüência de movimentos bem definidos para a realização da transferência de carga entre os dois manipuladores. Com isso diminuiu-se significativamente a quantidade de posições ou estados a serem analisados. A seqüência de movimentos foi definida em cinco etapas ou es-

tratégias de movimentação, conforme ilustrado na Figura 4.3, onde cada estratégia visa posicionar um dos manipuladores em uma posição previamente definida.

A transferência de carga parte de uma posição inicial definida na Figura 4.3a (procedimentos *a* e *b* da seção 4.4). As cinco estratégias de posicionamento estão ilustradas na Figura 4.3 a partir da parte *b* até a parte *f*.

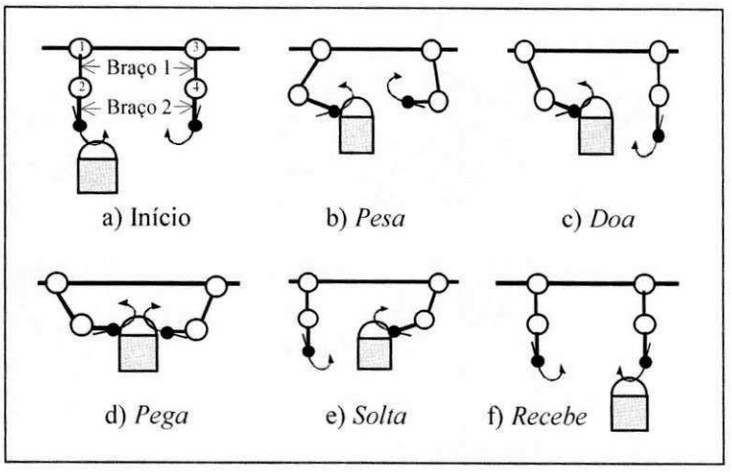


Figura 4.3 – Seqüência de estratégias de movimentação.

Considerando-se que qualquer um dos dois manipuladores pode estar de posse da carga, um diagrama de estados foi elaborado para definir a seqüência de ativação das estratégias para realizar a transferência de carga entre os manipuladores. Esse diagrama é ilustrado na Figura 4.4, com o primeiro manipulador de posse da carga e o segundo manipulador sem a carga. A seqüência de ativação das estratégias está representada pelos círculos sombreados.

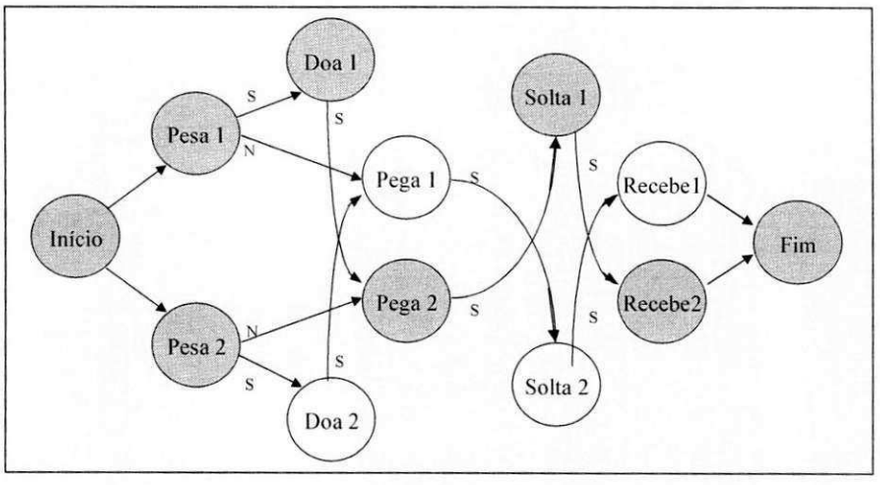


Figura 4.4 – Diagrama de estado da seqüência de movimentos.

4.5.1. A estratégia *Pesa*

A primeira estratégia definida como *Pesa* (procedimentos *c*, *d* e *e* da seção 4.4), identifica qual manipulador possui a carga (Figura 4.3b). Nessa estratégia os motores livres dos dois manipuladores (motores 2 e 4) são acionados com $\theta_{2r} = \frac{\pi}{4}$ e $\theta_{4r} = -\frac{\pi}{4}$ de tal forma que o eixo do motor fixo do manipulador que estiver com a carga terá um deslocamento, percebido pelo detector de posição, bem maior do que se estivesse sem a carga. O subscrito $_r$ indica o de referência ser alcançado pelo motor correspondente ao subscrito numérico.

O controle dessa estratégia foi definido "*off-line*" através de um gráfico nebuloso com duas funções de pertinência, relacionadas com o deslocamento do eixo do motor fixo. Foram atribuídos os valores lingüísticos pequeno (P) e grande (G) para cada uma das duas funções de pertinência, indicando que o manipulador está sem a carga (*ha_carga == 0*) ou com a carga (*ha_carga == 1*), respectivamente. O gráfico nebuloso é ilustrado na Figura 4.5. As informações relacionadas com a definição desse gráfico (posicionamento do eixo do motor) são apresentadas no Apêndice B, juntamente com um estudo sobre o detector de posição. As regras nebulosas 4.1 e 4.2 representam a estratégia *Pesa*. O subscrito $_f$ representa o ângulo θ fuzzyficado.

Regra (4.1): Se θ_{1f} *== G e* θ_{3f} *== P então ha_carga1 = 1, ha_carga2 = 0;*

Regra (4.2): Se θ_{3f} *== G e* θ_{1f} *== P então ha_carga1 = 0, ha_carga2 = 1;*

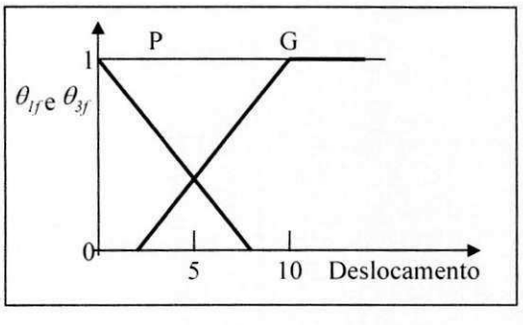


Figura 4.5 – Gráfico nebuloso da estratégia *Pesa*.

4.5.2. A estratégia *Doa*

A estratégia *Doa* (procedimento *f* da seção 4.4), representa o movimento realizado pelo manipulador com a carga, designado "doador", para entregá-la ao outro manipulador livre, desi-

gnado "receptor" (Figura 4.3c). Nessa estratégia, o Doador (manipulador 1) deve movimentar o seu elo 1 para a posição $\pi/6$ e o seu elo 2 para a posição $\pi/4$. O elo 2 dos dois manipuladores não possui detector de posição. Assim, ele pode ser posicionado em repouso (motor livre desligado), na posição $\pi/4$ (motor livre girando no sentido anti-horário) ou na posição $-\pi/4$ (motor livre girando no sentido horário). As posições $\pi/4$ e $-\pi/4$ são obtidas través de travas mecânicas que mantêm o elo 2 nessas posições, conforme ilustrado na Figura 4.6.

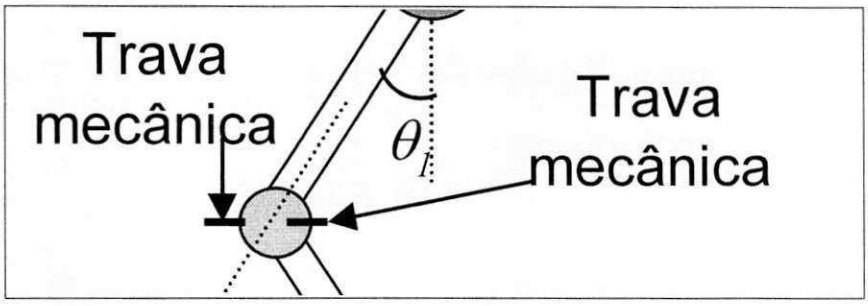


Figura 4.6 – Travas mecânicas do motor livre.

Já o elo 1 deve ser submetido a um sistema de controle de tal forma que ele alcance a posição $\pi/6$ e permaneça nela até que uma outra estratégia o modifique. As regras nebulosas 4.3 e 4.4 representam a estratégia *Doa*. O Sistema de controle se encarrega do posicionamento dos elos dos manipuladores nas posições indicadas.

Regra (4.3): Se ha_carga1 == 1 então $\theta_{1r} = \pi/6$ *e* $\theta_{2r} = \pi/4$

Regra (4.4): Se ha_carga2 == 1 então $\theta_{3r} = -\pi/6$ *e* $\theta_{4r} = -\pi/4$

4.5.3. A estratégia *Pega*

A estratégia *Pega* (procedimento *g* da seção 4.4), representa o movimento realizado pelo receptor (manipulador 2) para receber a carga (Figura 4.3d). Nessa estratégia, o receptor deve movimentar o elo 1 para a posição $-\pi/6$ e em seguida movimentar o elo 2 para a posição $-\pi/4$. As regras nebulosas 4.5, 4.6, 4.7 e 4.8 representam a estratégia *Pega*. O sistema de controle se encarrega do posicionamento dos elos dos manipuladores nas posições indicadas.

Regra (4.5): Se ha_carga1 == 1 então $\theta_{3r} = -\pi/6$

Regra (4.6): Se ha_carga1 == 1 e $\theta_{3r} == -\pi/6$ *então* $\theta_{4r} = -\pi/4$

Regra (4.7): Se ha_carga2 == 1 então $\theta_{1r} = \pi/6$

Regra (4.8): Se ha_carga2 == 1 e $\theta_{1r} == \pi/6$ *então* $\theta_{2r} = \pi/4$

4.5.4. A estratégia *Solta*

A estratégia *Solta*, representa o movimento (procedimento *f* da seção 4.4), realizado pelo doador, para deixar a carga com o receptor (Figura 4.3e). Nessa estratégia, o doador deve primeiro movimentar o seu elo 2 para a posição de repouso e depois movimentar o seu elo 1 também para a posição de repouso. As regras nebulosas 4.9, 4.10, 4.11 e 4.12 representam a estratégia *Pega*. O Sistema de controle se encarrega do posicionamento dos elos dos manipuladores nas posições indicadas.

Regra (4.9): Se ha_carga1 == 1 então θ_{2f} *= "ZE"*

Regra (4.10): Se ha_carga1 == 1 e $\theta_{2r} == 0$ *então* $\theta_{1r} = 0$

Regra (4.11): Se ha_carga2 == 1 então $\theta_{4r} = 0$

Regra (4.12): Se ha_carga2 == 1 e $\theta_{4r} == 0$ *então* $\theta_{3r} = 0$

4.5.5. A estratégia *Recebe*

A estratégia *Recebe* (procedimento *g* da seção 4.4), representa o movimento realizado pelo receptor após receber a carga do doador (Figura 4.3f). Nessa estratégia, o receptor deve primeiro movimentar o seu elo 1 para a posição de repouso e depois o seu elo 2 também para a posição de repouso. As regras nebulosas 4.13, 4.14, 4.15 e 4.16 representam a estratégia *Recebe*. O Sistema de controle se encarrega do posicionamento dos elos dos manipuladores nas posições indicadas.

Regra (4.13): Se ha_carga1 == 1 então $\theta_{1r} = 0$

Regra (4.14): Se ha_carga1 == 1 e $\theta_{1r} == 0$ *então* $\theta_{2r} = 0$

Regra (4.15): Se ha_carga2 == 1 então $\theta_{3r} = 0$

Regra (4.16): Se ha_carga2 == 1 e $\theta_{3r} == 0$ *então* $\theta_{4r} = 0$

4.6. O Sistema inteligente

O processo de transferência de carga, realizado de acordo com as estratégias definidas na seção anterior, é gerenciado por um sistema inteligente. Esse sistema é responsável pelo controle das operações de troca e pelo controle da planta representada pelos motores fixos e livres. O controle da planta é realizado por um Sistema de Controle Inteligente (SCI) através de uma Rede Neural Artificial (RNA) utilizando lógica nebulosa para geração de diferentes pontos de referência.

4.6.1. O Sistema de Controle Inteligente (SCI)

O controle e posicionamento dos manipuladores em uma determinada posição é garantido pelo SCI proposto por Cavalcanti & Ferneda (1995) e Alsina & Cavalcanti (1997), ilustrado na Figura 4.6. O SCI, composto por um controlador neural, um módulo nebuloso ("*fuzzyficador*"[10] e inferência nebulosa) e uma base de conhecimento, fornece os valores das referências a serem seguidas pelos motores dos dois manipuladores robóticos e garante a tensão de armadura dos motores.

O significado dos símbolos presentes na Figura 4.7 são descritos a seguir. O subscrito $_{(t)}$ representa o tempo na forma discreta. $U_{(t)}$ representa a tensão de armadura do motor sob controle, $\theta_{(t)}$ representa o ângulo formado pelo movimento gerado pelo motor, $\theta_{(t-1)}$ representa o ângulo formado pelo movimento gerado pelo motor durante o controle no tempo anterior, $\theta_{r(t+1)}$ representa o ângulo de referência desejado, $U_{(t+1)}$ representa a tensão de armadura fornecida pelo controlador neural a ser aplicada sobre o motor, $\theta_{(t+1)}$ representa o ângulo formado pelo movimento gerado pelo motor sob a tensão $U_{(t+1)}$, *Motor CC* representa a planta sob controle, o símbolo Σ dentro de um círculo, representa o cálculo do erro encontrado na saída da planta, E representa o erro da saída na planta, θ_f representa o valor "*fuzzyficado*"[11] na saída da planta, APR representa o algoritmo de propagação retroativa do erro, U_f representa a tensão *fuzzyficada* na armadura, ΔU_f representa a variação da tensão *fuzzyficada* na armadura e χ_f representa o fator de treinamento da rede neural (Cavalcanti et al., 1999a).

---

[10] Processo responsável por transformar uma variável quantitativa em qualitativa.

[11] Indica que uma variável foi submetida ao fuzzyficador.

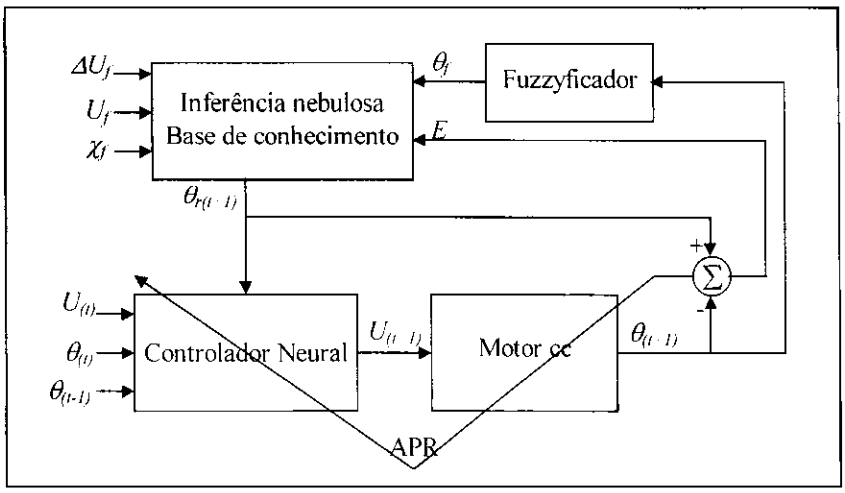


Figura 4.7 – O Sistema de Controle Inteligente (SCI).

O controlador neural, detalhado na Figura 4.8, é um controlador neural do tipo adaptativo direto, conforme o estudo sucinto sobre controladores apresentado no Apêndice C. Esse controlador, que utiliza uma Rede Neural Artificial Multicamadas (RNMC), Recebe nas suas entradas o valor da tensão atual ($U_{(t)}$) aplicada sobre o motor, o valor do ângulo atual ($\theta_{(t)}$), o valor do ângulo obtido durante o controle anterior ($\theta_{(t-1)}$) e o valor do ângulo de referência desejado ($\theta_{r(t+1)}$). A RNMC gera a nova tensão ($U_{(t+1)}$) que será utilizada no acionamento do Motor CC. Após o acionamento do motor um novo ângulo $\theta_{(t+1)}$ é gerado e o erro é calculado e propagado através da rede pelo APR. As características básicas do treinamento da RNMC do controlador neural são apresentadas no Apêndice D.

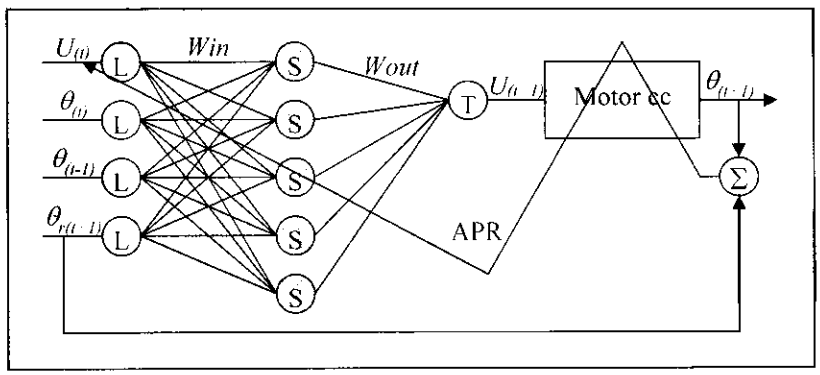


Figura 4.8 – O controlador neural adaptativo direto.

Os neurônios utilizados na rede neural são: linear (L) na camada de entrada, sigmóide (S) na camada oculta e tangente hiperbólico (T) na camada de saída. A definição da quantidade de neurônios na camada oculta foi proposta por Cavalcanti (1994, p36). Após testes exaustivos de desempenho, ele conclui que um número de quatro a seis neurônios na camada oculta

é o ideal para o controle de um Motor CC, quando forem utilizados os parâmetros reajustáveis dos neurônios propostos por Rangwala & Dornfeld (1989), apresentados no Apêndice D.

O APR é responsável pelo treinamento do controlador neural a partir do erro encontrado na saída da planta. Assim, a saída $\theta_{(t+1)}$ deve seguir um sinal desejado $\theta_{r(t+1)}$, modificando o valor de $U_{(t)}$ através do APR, de forma a minimizar o erro na saída da planta, definido na Equação 4.5. Entretanto, para que o erro encontrado na saída da planta seja propagado até a saída da RNMC (entrada da planta), ou seja, para propagar retroativamente o erro pela planta, é necessário conhecer o *jacobiano* da mesma. Nesse caso, como a planta é um pêndulo movimentado por um Motor CC e de acordo com a Equação 4.4, deduzida a partir das Equações 4.1, 4.2 e 4.3, todas apresentada na Seção 4.3, o jacobiano é dado pela Equação 4.6.

A partir do *jacobiano*, é possível definir o valor do erro na saída do controlador e então propagá-lo em toda a rede de acordo com o APR apresentado em detalhes no Apêndice D, juntamente com o algoritmo de propagação das entrada na RNMC.

Utilizou-se a regra delta, descrita na Equação 4.7, e o índice de desempenho, descrito na Equação 4.8, no desenvolvimento do valor de controle para treinamento da RNMC. A partir de Equação 4.6, obtém-se o valor para treinamento. Alguns autores classificam essa necessidade de se conhecer o *jacobiano* da planta como uma desvantagem do controle neural adaptativo direto. Porém, sob certas condições, o *jacobiano* pode ser aproximado e o uso de parâmetros de aprendizado variáveis permitem melhorar consideravelmente o desempenho do controlador e, com isso tornar o controlador neural adaptativo direto perfeitamente utilizável.

$$E = \theta_{r(t+1)} - \theta_{(t+1)} \tag{4.5}$$

$$\frac{\partial I}{\partial U} = -\frac{\partial \theta_{(t+1)}}{\partial U_{(t+1)}} E \tag{4.6}$$

$$U_{(t+1)} = U_{(t)} + \left(-\mu \nabla_{I_{(t)}}\right) = U_{(t)} + \left(-\mu \frac{\partial I}{\partial U}\right) = U_{(t)} + \left(\mu E \frac{\partial \theta_{(t+1)}}{\partial U_{(t+1)}}\right) \tag{4.7}$$

$$I_{(t)} = \frac{1}{2} E^2 = \frac{1}{2} (\theta_{r(t+1)} - \theta_{(t+1)})^2 \tag{4.8}$$

4.6.2. Gerência da transferência de carga

A transferência de carga entre os dois manipuladores, realizada através das cinco estratégias de movimento estudadas na seção 4.5, utiliza o Escalonador de Tarefas em Tempo Real (ETR) proposto por Alsina & Cavalcanti (1997) e descrito no Capítulo 2 desta dissertação. O ETR é dependente exclusivamente do tempo, conforme ilustrado na Figura 4.9. O ETR é ativado por interrupções que ocorrem de 1 em 1 ms, geradas por um relógio de tempo real (RTC, do inglês *Real-Time Clock*).

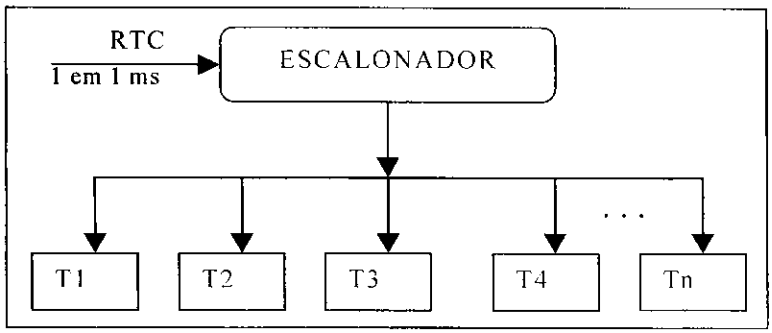


Figura 4.9 – ETR dependente do tempo.

Todas as tarefas possuem um descritor com quatro campos (*id*, *st*, *tempo*, *freq*) (apresentados no Quadro 2.1). A cada interrupção gerada pelo RTC, o valor do campo *tempo* é decrementado (ver Quadro 2.2). Quando o valor do campo *tempo* de uma tarefa chega a zero, o estado (*st*) da mesma é alterado para o valor 1, ou seja, a tarefa passa para o estado *pronta*. Em seguida, o escalonador verifica o estado de todas as tarefas, aquela que tiver *st* == *1* (estado *pronta*), será ativada. O algoritmo simplificado do ETR é descrito no Quadro 4.1.

Quadro 4.1 – Algoritmo simplificado do ETR.

```
Faça {
   Avalia todos os descritores;
   Aciona todas as tarefas com status == 1;
} (para cada interrupção)
```

A seqüência de ativação das tarefas foi definida heuristicamente pelo especialista após testes exaustivos para temporizar cada tarefa. Nesta dissertação, temporizar uma tarefa significa atribuir valores inteiros positivos aos campos *tempo* e *freq* dos descritores das tarefas. Por exemplo, ao se definir tarefa (9, 0, 2000, 0), o valor *tempo* == *2000*, significa que após 2000

ms será ativada a tarefa 9 (o valor do seu campo *st* passa de 0 para 1), como pode ser visto no Algoritmo do ETR apresentado na Seção 2.9.

Seguindo-se o diagrama de estados definido na Figura 4.5 e conforme a análise de cada estratégia de movimentação feita na Seção 4.5, e das regras nebulosas, define-se as tarefas necessárias para cada estratégia.

Todas as tarefas utilizadas na transferência de carga entre os dois manipuladores estão listadas no Quadro 4.2. Os valores dos parâmetros informados para cada uma das tarefas são ilustrativos e a definição do valor de cada um deles é de total responsabilidade do especialista, com exceção do primeiro campo, que é o número identificador da tarefa. Os detalhes da implementação (pseudo código) de cada uma das tarefas é descrito no Apêndice E.

Quadro 4.2 – Relação das tarefas do sistema.

```
tarefa (1,0,1,1);      // Tarefa para transferir a carga do manipulador 1 para o 2.
tarefa (2,0,1,1);      // Tarefa para acionamento do motor 2.
tarefa (3,0,1,1);      // Tarefa para acionamento do motor 4.
tarefa (4,0,20,20);    // Tarefa para o treinamento do motor 1 e 3 através do SCI.
tarefa (5,0,1,0);      // Tarefa para desativar os motores 1 e 2.
tarefa (6,0,1,0);      // Tarefa para desativar os motores 1 e 2.
tarefa (7,0,20,20);    // Tarefa para o treinamento do motor 1.
tarefa (8,0,20,20);    //Tarefa para o treinamento do motor 3.
tarefa (9,0,2000,0);   //Tarefa para verificar se há carga nos manipuladores.
tarefa (10,0,1,1);     // Tarefa para transferir a carga do manipulador 2 para o 1.
```

4.6.3. A transferência de carga

Inicialmente, considera-se que o peso esteja acoplado ao manipulador 1, conforme ilustrado na Figura 4.10. Inicialmente os dois manipuladores verificam se possuem o peso, conforme o diagrama de estado ilustrado na Figura 4.5 (indicados pelos círculos *Pesa 1 e Pesa 2* sombreados) utilizando a estratégia *Pesa*.

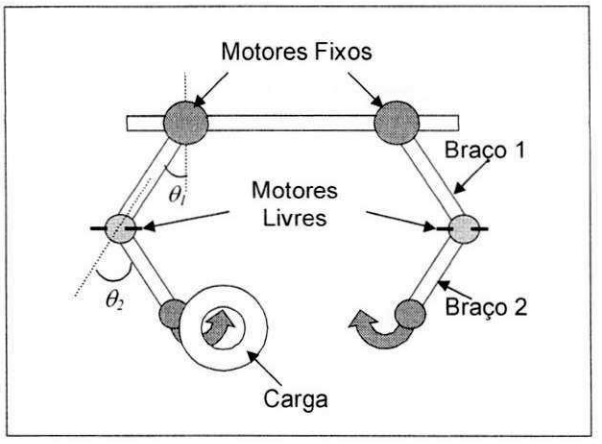


Figura 4.10 – Representação da transferência da carga do manipulador 1 para o 2.

4.6.3.1 A estratégia *Pesa*

A estratégia Pesa, representada pelas regras 4.1 e 4.2, foi implementada com 3 tarefas (tarefas 2, 3 e 9). As tarefas 2 e 3 são ativadas para posicionar o elo 2 dos dois manipuladores (motores 2 e 4) nas posições $\pi/4$ e $-\pi/4$, respectivamente (atribuições $\theta_{2r} = \frac{\pi}{4}$ e $\theta_{4r} = -\frac{\pi}{4}$). Após 2000 ms, tempo calculado heuristicamente para que os dois elos livres se posicionem nas referências, a tarefa 9 é acionada. A tarefa 9 implementa as regras nebulosas 4.1 e 4.2.

Os campos das tarefas 2, 3 e 9 são inicializados com os valores apresentados no Quadro 4.3. As tarefas 2 e 3 acionam os motores 2 e 4 com freqüência de 1 ms (*freq == 1*, quarto campo do descritor). Isto significa que os dois Motores CC serão acionados com tensão de armadura máxima (corrente contínua pulsada). O acionamento dos motores através de tarefas permite a modulação da largura de pulso da tensão de cada motor.

Quadro 4.3 – Tarefas da estratégia *Pesa*.

```
tarefa (2,1,1,1);        //Aciona o motor 2 de 1 em 1 ms.
tarefa (3,1,1,1);        //Aciona o motor 4 de 1 em 1 ms.
tarefa (9,0,2000,0);     //Após 2000ms Aciona a tarefa 9.
```

A tarefa 9 é ativada após 2000 ms para identificar qual dos manipuladores está com a carga. Os parâmetros da tarefas 9 são: *id == 9*, identificação da tarefa; *st == 1*, tarefa pronta para executar (*ready*); *tempo == 2000*, espera 2000 ms para ser executada; *freq == 0*, a tarefa 9 só será executada uma vez. A tarefa 9 é apresentada no Quadro 4.4.

Inicialmente no corpo da tarefa 9 é feita a *fuzzyficação* dos deslocamentos (ângulos θ_1 e θ_3) dos eixos dos motores fixos (motores 1 e 3), obtendo os valores nebulosos (ângulos θ_{1f} e θ_{3f}). Os ângulos θ_{1f} e θ_{3f} são utilizados pelas regras (4.1) e (4.2) para determinar qual dos manipuladores está com a carga. A seguir, a tarefa 9 desliga os motores 2 e 4. Supondo-se que o manipulador 1 está com a carga, representado pela variável *booleana ha_carga == 1*, a tarefa 1 é ativada (se a carga estivesse com o manipulador 2, a tarefa 10 seria ativada). As tarefas 1 e 10 são ativadas uma única vez, pois elas são utilizadas para ativar o conjunto de tarefas que compõem a transferência de cada manipulador.

Quadro 4.4 – Tarefa 9 ativada após 2000 ms.

```
Ativa_tarefa (9,0,2000,0);   //Verifica qual manipulador está com a carga.
{
   θ1f = Fuzzyfica (θ1);
   θ3f = Fuzzyfica (θ3);
   Se θ1f == G e θ3f == P então há_carga1 = 1,  há_carga2 = 0;
   Se θ3f == G e θ1f == P então há_carga1 = 0,  há_carga2 = 1;
   Ativa_tarefa (2,0,0,0);   //Desliga o motor 2.
   Ativa_tarefa (3,0,0,0);   //Desliga o motor 4.
   Se ha_carga1 == 1
      Ativa_tarefa (1,1,0,0);   //Ativa a transferência do manipulador 1 para o 2.
   Senão Se  ha_carga2 == 1
      Ativa_tarefa (10,1,0,0);  // Ativa a transferência do manipulador 2 para o 1.
   Fimse
}
```

Caso a carga esteja com o manipulador 1, a tarefa 1 será ativada. Essa tarefa é responsável pela seqüência de ativação das tarefas das estratégias *Doa*, *Pega*, *Solta* e *Recebe* (ver Quadro 4.5). A temporização das tarefas foi feita pelo especialista após exaustivos testes.

Quadro 4.5 – Tarefa 1 para a transferência de carga do manipulador 1 para o 2.

```
tarefa(4,0,1,20);       // doa - Traça gráficos.
tarefa(2,1,1,1);        // doa - Aciona o motor 2.
tarefa(7,0,500,20);     //doa – Espera 500 ms e Aciona o motor 1.
tarefa(8,0,1000,20);    //pega - Espera 1000 ms e posiciona Motor 3.
tarefa(3,0,1500,1);     //pega - Espera 1500 ms e posiciona Motor 4.
tarefa(5,0,2000,0);     //Solta – Espera 2000 ms e desliga os motores 1 e 2.
tarefa(6,0,2500,0);     //Recebe – Espera 2500 ms e desliga os motores 3 e 4.
```

Caso a carga esteja com o manipulador 2, a tarefa 10 será ativada. Essa tarefa ativa as tarefas que executam a transferência da carga do manipulador 2 para o manipulador 1. A temporização também foi feita pelo especialista como explicado no parágrafo anterior. A relação das tarefas que executam a transferência do manipulador 2 para o 1 é apresentada no Quadro 4.6.

Quadro 4.6 – Tarefas 10 para a transferência do manipulador 2 para o 1.

```
tarefa(4,0,1,20);       // doa - Traça gráficos.
tarefa(3,1,1,1);        // doa - Aciona o motor 4.
tarefa(8,0,500,20);     //doa – Espera 500 ms e Aciona o motor 3.
tarefa(7,0,1000,20);    //pega - Espera 1000 ms e posiciona Motor 1.
tarefa(2,0,1500,1);     //pega - Espera 1500 ms e posiciona Motor 2.
tarefa(6,0,2000,0);     //Solta – Espera 2000 ms e desliga os motores 3 e 4.
tarefa(5,0,2500,0);     //Recebe – Espera 2500 ms e desliga os motores 1 e 2.
```

4.6.3.2 A estratégia *Doa*

A estratégia *Doa* e as estratégias seguintes são explicadas considerando-se a transferência da carga do manipulador 1 para o manipulador 2. Nessa estratégia, o elo 2 é movimentado para a posição $\pi/4$ utilizando $\theta_{2r} = \pi/4$ e ativando a tarefa 2. O elo 1 do manipulador 1 é movimentado para a posição $\pi/6$ utilizando $\theta_{1r} = \pi/6$ e com a tarefa 4 ativada. Após 500 ms, é ativada a tarefa 7, completando o ciclo da estratégia *Doa*. Os descritores das tarefas que compõem a estratégia *Doa* são apresentados no Quadro 4.7.

Quadro 4.7 – Descritores das tarefas da estratégia *Doa*.

```
tarefa(4,0,1,20);       // doa - Traça gráficos.
tarefa(2,1,1,1);        // doa - Aciona o motor 2.
tarefa(7,0,500,20);     //doa – Espera 500 ms e Aciona o motor 1.
```

4.6.3.3 A estratégia *Pega*

A estratégia *Pega* inicia com o movimento do elo 1 do manipulador 2 para a posição $-\pi/6$ utilizando $\theta_{3r} = -\pi/6$ e ativando a tarefa 8. O elo 2 do manipulador 2 é movimentado para a posi-

ção $-\pi/4$ utilizando $\theta_{4r} = -\pi/4$ e ativando a tarefa 3. Os descritores das tarefas que compõem a estratégia *Pega* são apresentados no Quadro 4.8.

Quadro 4.8 – Descritores das tarefas da estratégia *Pega*.

```
tarefa(8,0,1000,20);  //pega - Espera 1000 ms e posiciona Motor 3.
tarefa(3,0,1500,1);   //pega - Espera 1500 ms e posiciona Motor 4.
```

4.6.3.4 A estratégia *Solta*

A estratégia *Solta* inicia com o movimento do elo 2 do manipulador 1 para a posição 0 utilizando $\theta_{2r} = 0$ e o elo 1 do manipulador 1 também é movimentado para a posição 0 utilizando $\theta_{1r} = 0$ ativando a tarefa 5. O descritor da tarefa 5 que compõe a estratégia *Solta* é apresentado no Quadro 4.9.

Quadro 4.9 – Descritor da tarefa da estratégia *Solta*.

```
tarefa(5,0,2000,0);   //Solta – Espera 2000 ms e desliga os motores 1 e 2.
```

4.6.3.5 A estratégia *Recebe*

A estratégia *Recebe* inicia com o movimento do elo 1 do manipulador 2 para a posição 0 utilizando $\theta_{3r} = 0$ e o elo 2 do manipulador 2 também é movimentado para a posição 0 utilizando $\theta_{4r} = 0$ ativando a tarefa 6. O descritor da tarefa 6 que compõe a estratégia *Recebe* é apresentado no Quadro 4.10.

Quadro 4.10 – Descritor da tarefa da estratégia *Recebe*.

```
tarefa(6,0,2500,0);   //Recebe – Espera 2500 ms e desliga os motores 3 e 4.
```

4.7. Resultados experimentais

A transferência de carga entre as dois manipuladores utilizando o ETR foi realizada com sucesso e alguns resultados experimentais foram obtidos. As curvas obtidas com o ETR durante a transferência entre os dois manipuladores no intervalo de tempo de 4000 ms é ilustrada na

Figura 4.11. Inicialmente a carga estava com o manipulador 1. A estratégia *Pesa* não é ilustrada nesta figura. A posição angular θ_1 do manipulador 1 durante todo o intervalo de tempo da transferência é ilustrada na Figura 4.11a. A posição angular θ_3 do manipulador 3 durante todo o intervalo de tempo da transferência é ilustrada na Figura 4.11b. Inicialmente os dois manipuladores estão na posição de repouso. Durante o tempo $0 < t < 1000$ ms o manipulador 1 *Doa* a carga (*Doa* na Figura 4.11a). Durante o tempo $1000 < t < 2000$ ms, o manipulador 2 *Pega* a carga (*Pega* na Figura 4.11b). Durante o tempo $2000 < t < 2500$ ms, o manipulador 1 *Solta* a carga (*Solta* na Figura 4.11a). Durante o tempo $2500 < t < 3000$ ms, o manipulador 1 *Recebe* a carga (*Recebe* na Figura 4.11b). Os valores de θ_1 e θ_3 na ordenada são dados em quantidade de furos do detector de posição (ver Apêndice B).

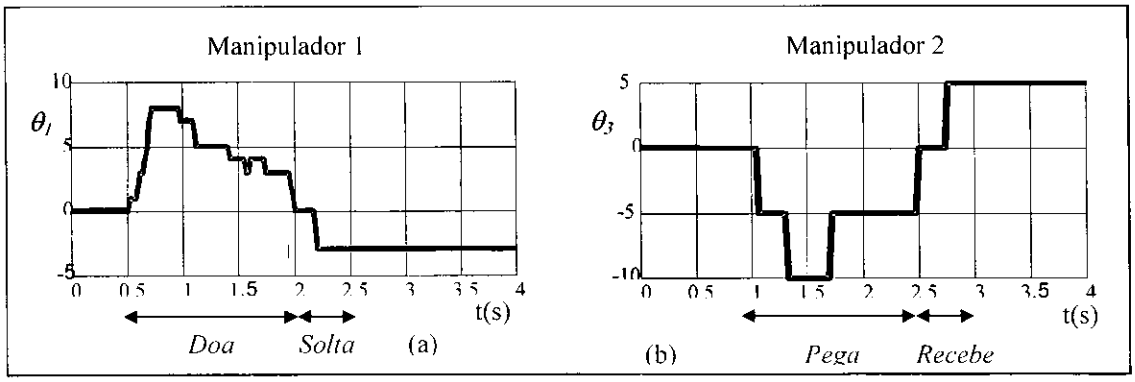


Figura 4.11 – Resultados experimentais da transferência de carga com o ETR.

4.8. Conclusão

A aplicação robótica apresentada neste capítulo se mostrou muito interessante para o estudo de cooperação entre robôs, principalmente pela utilização de tarefas no desenvolvimento de cada etapa do problema abordado e no emprego de um Sistema Controle Inteligente. Entretanto, o controle do processo de transferência através das tarefas escalonadas é totalmente dependente do especialista, visto que a ordenação no tempo das tarefas é feita *off-line* pelo especialista, o que torna o sistema um processo não adaptável, seqüencial e com quase nenhum procedimento inteligente na rotina de escalonamento.

Capítulo 5

O escalonador inteligente de tarefas

5.1. Resumo

Neste capítulo, apresenta-se o "Escalonador Inteligente de Tarefas para Aplicações Robóticas". Inicialmente, descreve-se os motivos que conduziram ao desenvolvimento desse escalonador, apresentando-se uma breve introdução, destacando-se as principais características de escalonamento que o mesmo deve apresentar. Em seguida, descreve-se como esse escalonador foi desenvolvido, apresentando-se as estratégias de movimento, definidas no Capítulo 4, com as devidas alterações para serem utilizadas nessa nova abordagem. Encerra-se o capítulo com uma análise do escalonador apresentado.

5.2. Introdução

O Escalonador Inteligente de Tarefas em Tempo Real (EIT) proposto a seguir foi desenvolvido a partir de observações feitas no Escalonador de Tarefas em Tempo Real (ETR) proposto por Alsina & Cavalcanti (1997), apresentado no Capítulo 2 e 4, destinado à aplicação robótica. Os detalhes do sistema, desenvolvido no Borland C++ Builder, são apresentados no Apêndice F. O EIT foi desenvolvido utilizando-se como plataforma de teste o mesmo problema de transferência de carga entre dois manipuladores robóticos, abordado no Capítulo 4. Para o EIT foi proposta uma alteração no algoritmo do ETR e nos descritores das tarefas para que o novo escalonador fosse capaz de realizar a ordenação das tarefas de acordo com o tempo e com informações percebidas pelos detectores de posição durante o processo de transferência de carga entre os dois manipuladores. Essas alterações permitiram que após a execução de uma tarefa o escalonador avalie a regra associada a ela. As regras foram construídas com base nas posições dos manipuladores, percebidas pelos detectores de posição de tal forma que após um movimento o escalonador pode ativar imediatamente a próxima regra. Porém, qualquer alteração no núcleo de um escalonador deve ser muito bem definida, pois o código executável do mesmo deve ter o menor tamanho possível, visto que a cada interrupção ele é executado para

permitir que a próxima tarefa faça uso da CPU. Caso esse código seja muito grande, a CPU passará a maior parte do tempo executando o código do escalonador, enquanto as tarefas aguardam a liberação da mesma.

Esse problema se agrava mais ainda, quando o escalonador é utilizado para o controle de processos em tempo real, onde as restrições de tempo devem ser rigorosamente atendidas. Assim, a idéia de fornecer inteligência ao escalonador foi desenvolvida levando-se em consideração essas restrições e adotando-se as seguintes características: escalonamento específico, tempo real inflexível, dinâmico, determinístico, não preemptivo e centralizado. O núcleo do EIT é baseado no algoritmo de escalonamento em fila, o que reduz bastante a complexidade do mesmo. O escalonamento específico é evidente pois o escalonador deve tratar de tarefas relacionadas com o acionamento de motores, leitura dos detectores de posição e outras, específicas de aplicações robóticas com o escalonamento em tempo real inflexível, que todas as tarefas serão executadas dentro de uma rigorosa restrição de tempo, garantindo o sucesso do controle e acionamento dos motores. O escalonamento dinâmico foi utilizado devido a ordem de escalonamento ser definida durante a execução das tarefas a partir de informações adquiridas do ambiente. O escalonamento é determinístico pois as tarefas são todas conhecidas. O escalonamento centralizado também é evidente pois esta aplicação foi projetada para execução em microcomputadores do tipo IBM-PC, que possui uma arquitetura centralizada (mono-usuário).

5.3. Desenvolvimento do EIT

Associou-se a cada tarefa uma regra nebulosa que é avaliada pelo próprio EIT logo após a execução da tarefa. Acrescentou-se ao núcleo do EIT somente a rotina de inferência nebulosa das regras. Isso foi possível devido às regras estarem relacionadas diretamente com as tarefas, sem a necessidade de uma base de conhecimento como normalmente as regras nebulosas são utilizadas, o que gastaria muito tempo com pesquisas durante o escalonamento. Um esquema gráfico do EIT é ilustrado na Figura 5.1. O EIT é dependente do tempo e das inferências nebulosas, ou seja, a cada 1 ms o RTC envia uma interrupção de tempo na qual o EIT executa as tarefas no estado *pronta* e avalia as regras associadas a cada uma das tarefas executadas.

A base de conhecimento existe e é formada por todas as regras associadas as tarefas do sistema. Entretanto, não há a necessidade de procedimentos de busca ou seleção de regras, devido as regras estarem distribuídas nas tarefas. Isso elimina um problema clássico na utili-

zação de sistemas baseados em conhecimento que é a busca por um determinado conhecimento na base de conhecimento.

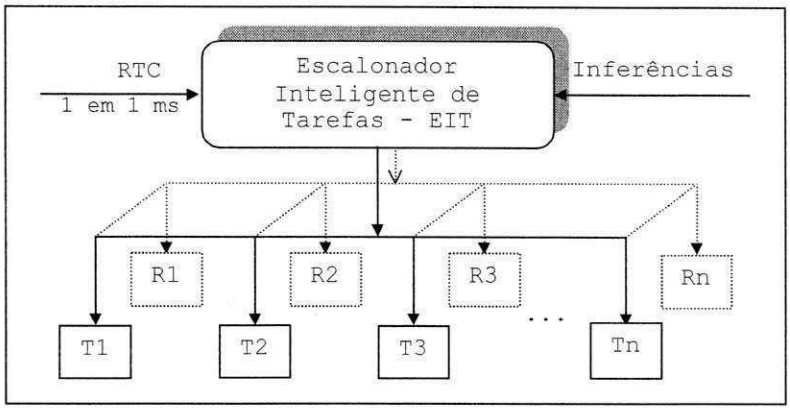


Figura 5.1 – Escalonador Inteligente.

O desenvolvimento do EIT baseou-se no ETR com o acréscimo de uma rotina que lhe permite verificar os valores dos detectores de posição e avaliar as regras. O EIT teve o acréscimo de dois novos campos nos descritores das suas tarefas. No EIT, os descritores possuem seis parâmetros descritos no Quadro 5.1. Os quatro primeiros parâmetros são os mesmos apresentados no ETR e os dois últimos *nvf* e *vf* representam o número do motor e um valor lingüístico associado ao ângulo do motor respectivamente, onde *nvf* identifica o motor com *nvf* == *1* para o motor 1, *nvf* == *2* para o motor 2, *nvf* ==*3* para o motor 3 e *nvf* == *4* para o motor 4. *vf* é uma variável nebulosa que pode assumir qualquer um dos valores lingüísticos NG, NM, ZE, PM e PG, que indicam a intensidade do deslocamento do elo associado ao motor, medido através do ângulo θ_i e *fuzzyficado* em um dos valores lingüísticos (θ_{if}).

Quadro 5.1 – Descritor de tarefas.

tarefa(*id*, *st*, *tempo*, *freq*, *nvf*, *vf*)

id – identificação da tarefa
st – estado atual da tarefa
tempo – intervalo de tempo para acionamento
freq – freqüência de reacionamento
nvf – número do motor
vf – valor nebuloso associado ao motor (θ_{if})

As regras nebulosas associadas às tarefas são avaliadas através do valor nebuloso θ_{if}, do ângulo θ_i, formado em cada acionamento de motor durante um movimento e do valor nebuloso desejado, informado em *vf*. A ativação das tarefas e regras dependentes do tempo e da posição do manipulador é ilustrado na Figura 5.2. As tarefas são ativadas no tempo (eixo *t*), a cada tarefa é associada uma regra nebulosa (R, no plano $t\text{x}\theta$) e *fuzzyficada* no plano $t\text{x}\theta\text{x}\theta_f$.

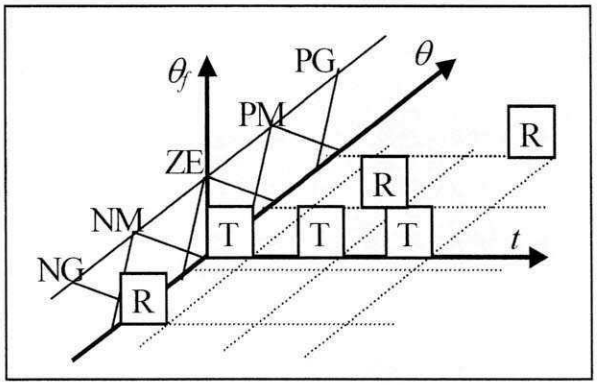


Figura 5.2 – Execução das tarefas e regras no tempo.

O algoritmo simplificado do EIT é descrito no Quadro 5.2. Esse algoritmo, executado a cada interrupção, é composto basicamente de um laço que avalia todos os descritores das tarefas.

Quadro 5.2 – Algoritmo simplificado do EIT.

```
A cada interrupção {
    Faça (Para todos os descritores){
        Se (tempo != 0) {
            tempo --;
            Se (tempo == 0) {
                tempo = freq;
                st = 1;
            }
        }
        Se (st == 1) {
            st = -1;
            Aciona tarefa;
            st = 0;
            Se (θr==θ) Avalia regra;
    }
    Aciona motores;
}
```

No algoritmo do Quadro 5.2, inicialmente o *tempo* de todas as tarefas é avaliado para saber se alguma tarefa está com o *tempo* esgotado. Note-se que para uma tarefa ser ativada através do *tempo*, este precisa ser superior ou igual a 1, pois se o *tempo* for igual a zero, a tarefa jamais será executada porque o teste para passá-la para o estado *pronta* (*st == 1*) está dentro do teste do *tempo* das tarefas (*tempo != 0*)[12]. Se uma tarefa tem o seu *tempo* esgotado (*tempo == 0*), então o escalonador passa o seu estado para *pronta* (*st == 1*) e o *tempo* recebe o valor de *freq* para definir o novo *tempo* de espera da tarefa. Caso *freq* seja igual a 0, então a tarefa não será mais ativada. O parâmetro *freq* recebe o valor 1, quando é necessário que a tarefa seja ativada em todas as interrupções. Em seguida, o estado das tarefas é avaliado e aquela tarefa que tiver *st == 1* passará para o estado *executando* (*st == -1*), será executada e logo após a sua execução, passará para o estado *inativa* (*st == 0*). A seguir, a regra associada a tarefa será avaliada, se $\theta_{if} == \theta_{ifr}$.

A última instrução do escalonador é o comando para acionar todos os motores. Esse acionamento dos motores é necessário devido o controle de velocidade de um Motor CC ser feito através de PWM[13]. Isto é, se a cada 1 ms for enviado um sinal de acionamento para um motor, então ele funcionará com a tensão máxima, mas se for enviado um sinal de acionamento a cada 2 ms então ele funcionará com a tensão aproximadamente pela metade. Assim, é possível controlar um Motor CC através de tarefas em tempo real inflexível.

As tarefas que controlam o acionamento dos motores atualizam os valores de variáveis associadas aos motores (roda[i], ver as tarefas 2 e 3 no Apêndice G). O escalonador simplesmente envia o valor dessa variável via porta paralela para que a interface controladora dos motores acione o motor que receber sinal de acionamento diferente de 0. Um estudo sobre essa interface é apresentado no Apêndice A.

5.4. A transferência de carga

A implementação da transferência de carga entre os dois manipuladores robóticos com o EIT segue a mesma análise feita no capítulo 4 com o problema dividido nas cinco estratégias de movimento. Ou seja, a transferência inicia com a estratégia *Pesa* e segue através das outras

---

[12] O símbolo != é utilizado para representar a operação de diferença.

[13] PWM (Pulse Width Modulation) que significa Modulação de Largura de Pulso, é utilizado para controlar a corrente elétrica de um motor elétrico, neste caso para controlar o Motor CC.

estratégias até que a transferência seja realizada. As tarefas que compõem todas as estratégias utilizadas com o EIT são descritas no Quadro 5.3. Os algoritmos de todas as tarefas são apresentados no Apêndice G.

Quadro 5.3 – Relação das tarefas do sistema com o EIT.

```
tarefa (1,1,0,0,0,0);      //Inicia a estratégia Doa do manipulador 1 para o 2.
tarefa (2,0,1,1,1,PM);     //Aciona o motor 2 e avalia a regra (5.1) a cada 1 ms.
tarefa (3,0,1,1,3,NM);     //Aciona o motor 4 e avalia a regra (5.2) a cada 1 ms.
tarefa (4,0,1,20,0,0);     //Treinamento inicial dos motores 1 e 3 a cada 20 ms.
tarefa (7,0,0,20,1,PM);    // aciona o motor 1 sob controle e avalia a regra 5.4 a cada 20 ms.
tarefa (8,0,0,20,3,NM);    // aciona o motor 3 sob controle e avalia a regra 5.6 a cada 20 ms.
tarefa (9,0,2000,0,0,0);   //Aciona a tarefa 9 para interromper a estratégia Pesa.
tarefa (10,1,0,0,0,0);     //Inicia a estratégia Doa do manipulador 2 para o 1.
tarefa (11,0,0,0,0,0);     // Encerra todo o processo de transferência.
tarefa (12,0,1,1,1,PM);    //Aciona o motor 2 e avalia a regra 5.3 a cada 1 ms.
tarefa (13,0,1,1,3,NM);    //Aciona o motor 4 e avalia a regra 5.5 a cada 1 ms.
tarefa (14,0,0,1,1,NM);    //Aciona o motor 2 e avalia a regra 5.10 a cada 1 ms.
tarefa (15,0,0,1,1,NM);    //Aciona o motor 4 e avalia a regra 5.8 a cada 1 ms.
tarefa (17,0,1,20,1,NM);       // Aciona o motor 1 sob controle e avalia a regra 5.9 a cada 20 ms.
tarefa (18,0,1,20,1,NM);       // Aciona o motor 3 sob controle e avalia a regra 5.7 a cada 20 ms.
```

5.4.1. A estratégia *Pesa*

A transferência inicia com o estratégia *Pesa*, da mesma forma como no ETR. Essa estratégia é composta por três tarefas (2, 3 e 9), conforme descrito no Quadro 5.4. Entretanto, agora as tarefas 2 e 3 possuem regras associadas a elas que permitem ao próprio EIT verificar qual manipulador está com a carga e ativar a próxima tarefa imediatamente após a carga ser detectada sem a necessidade de esperar por um *tempo* preestabelecido.

Quadro 5.4 – Tarefas da estratégia *Pesa*.

```
tarefa (2,0,1,1,1,PM);     //Aciona o motor 2 e avalia a regra (5.1) a cada 1 ms.
tarefa (3,0,1,1,3,NM);     //Aciona o motor 4 e avalia a regra (5.2) a cada 1 ms.
tarefa (9,0,2000,0,0,0);   //Aciona a tarefa 9 para interromper a estratégia Pesa.
```

A tarefa 2, responsável pelo acionamento do motor 2, é ativada a cada 1 ms, ou seja, se o EIT levar 20 ms para descobrir qual manipulador está com a carga, então a tarefa 2 será executada 20 vezes e conseqüentemente, a regra (5.1), associada à tarefa 2, também será avaliada 20 vezes. Cada vez que a regra (5.1) é avaliada, o EIT verifica se o valor lingüístico informado no descritor da tarefa 2 é igual ao valor percebido no detector de posição do motor 1.

Regra (5.1): ***Se*** $\theta_{1f} == \theta_{1rf}$ ***então*** *desativa as tarefas 2, 3 e 9 e ativa a tarefa 1;*

Na regra (5.1), θ_{1f} representa o valor lingüístico (*fuzzyficado*) do ângulo do motor 1 percebido pelo detector de posição, θ_{1rf} representa o valor lingüístico informado para o ângulo do motor 1 (sexto parâmetro da tarefa 2 no Quadro 5.2). Nesse caso, o valor informado foi PM, pois o motor 2 gira no sentido anti-horário, gerando um ângulo positivo e provocando um deslocamento médio no detector do motor 1. Quando a condição da regra (5.1) é verdadeira, as tarefas 2, 3 e 9 são desativadas (*freq == 0*, terceiro parâmetro) e a tarefa 1 passa para o estado *pronta* (*st == 1*, segundo parâmetro). A tarefa no estado *pronta* é executada automaticamente pelo escalonador.

Seguindo-se essa mesma análise para a tarefa 3, responsável pelo acionamento do motor 4, tem-se a regra (5.2) também avaliada a cada 1 ms pelo EIT. Observe-se que o valor lingüístico informado na tarefa 3 é NM, pois esta tarefa aciona o motor 4 no sentido horário, gerando um valor negativo. Na regra (5.2), θ_{3f} representa o valor lingüístico (*fuzzyficado*) do ângulo do motor 3 percebido pelo detector de posição (sexto parâmetro da tarefa 3). Quando a condição da regra (5.2) é verdadeira, as tarefas 2, 3 e 9 são desativadas e a tarefa 10 passa para o estado *pronta* (*st == 1*, segundo parâmetro).

Regra (5.2): ***Se*** $\theta_{3f} == \theta_{3rf}$ ***então*** *desativa as tarefas 2, 3 e 9 e ativa a tarefa 10;*

A tarefa 9 foi mantida para definir um tempo máximo de duração para a estratégia *Pesa* no caso 2000 ms. Entretanto, a verificação da presença de carga em um dos manipuladores é feita pelo EIT através das regras (5.1) e (5.2), podendo ocorrer dentro do intervalo de *tempo* de 1 a 2000 ms. Se a carga for detectada em 100 ms, por exemplo, automaticamente o EIT interrompe a estratégia *Pesa* e executa a próxima tarefa da transferência, que no caso pode ser a tarefa 1 ou a tarefa 10. No caso de não ser detectada nenhuma carga nos manipuladores, o EIT pode decidir por executar uma tarefa de erro, ativada toda vez que houver um erro no sistema, como o estouro de tempo, por exemplo.

5.4.2. A estratégia *Doa*

A transferência segue com a estratégia *Doa* após a identificação da carga através da estratégia *Pesa*. A estratégia *Doa* pode ser realizada através da tarefa 1 (transferência do primeiro manipulador para o segundo manipulador) ou da tarefa 10 (transferência do segundo manipulador para o primeiro manipulador). Caso a tarefa 1 seja ativada, a estratégia *Doa* será realizada através das três tarefas descritas no Quadro 5.5.

Quadro 5.5 – Tarefas da estratégia *Doa* a partir da tarefa 1.

tarefa (12,0,1,1,1,PM);	//Aciona o motor 2 e avalia a regra 5.3 a cada 1 ms.
tarefa (4,0,1,20,0,0);	//Treinamento inicial dos motores 1 e 3 a cada 20 ms.
tarefa (7,0,0,20,1,PM);	//Tarefa criada mas com o estado desativada, quando ela for
	//ativada, sua finalidade é acionar o motor 1 sob controle e
	//avaliar a regra 5.4 a cada 20 ms.

A tarefa 12, responsável pelo acionamento do motor 2, é ativada a cada 1 ms. Essa tarefa também possui uma regra (5.3) que também será avaliada a cada 1 ms. O EIT verifica se o valor lingüístico informado no descritor da tarefa 12 é igual ao valor percebido no detector de posição do motor 1, para saber se o movimento de responsabilidade dessa tarefa já foi realizado com sucesso e então ativar a próxima tarefa. Na regra (5.3), θ_{1f} representa o valor lingüístico (*fuzzyficado*) do ângulo do motor 1 percebido pelo detector de posição (sexto parâmetro da tarefa 12). Quando a condição da regra (5.3) for verdadeira, a tarefa 7 recebe um *tempo* de espera igual a 1 ms (*tempo == 1*, terceiro parâmetro da tarefa 7). Existem duas formas de ativar uma tarefa neste sistema, a tarefa pode receber diretamente o estado *pronta* (*st == 1*) ou a tarefa pode receber um *tempo* para ativação através do terceiro parâmetro (*tempo == 1*), o que foi feito, neste caso, com a tarefa 7. A segunda parte (θ_{1f} == *"ZE"*) da condição da regra (5.3) é utilizada para parar o motor 1 na estratégia *Solta*.

Regra (5.3): ***Se*** $\theta_{1f} == \theta_{1rf}$ ***ou*** θ_{1f} == *"ZE"* ***então*** *ativa a tarefa 7;*

A tarefa 4, responsável pelo treinamento inicial do controle dos motores 1 e 3, é ativada a cada 20 ms. Essa tarefa não possui nenhuma regra associada a ela (parâmetros 5 e 6 da tarefa 4 iguais a 0). A sua finalidade é atualizar as variáveis de treinamento dos motores 1 e 3 para que os novos valores de controle impostos pelo Sistema de Controle Inteligente (SCI) sejam utilizados.

A tarefa 7, responsável pelo acionamento do motor 1 sob o controle do SCI, é ativada a cada 20 ms. Esse *tempo* de 20 ms é necessário para que a RNMC do SCI seja treinada e para que o trem de pulso que define o sinal PWM para cada motor sob controle (1 e 3) seja aplicado. Essa tarefa também possui uma regra (5.4) que também será avaliada a cada 20 ms, assim como para as outras tarefas. Quando a condição da regra (5.4) é verdadeira, a tarefa 18 é passada para o estado *pronta* (*st* == *1*, segundo parâmetro da tarefa 18), para iniciar a estratégia *Pega*. A segunda parte (θ_{1f} == *"ZE"*) da condição da regra (5.4) é utilizada na estratégia *Solta*.

Regra (5.4): ***Se*** $\theta_{1f} == \theta_{1rf}$ ***ou*** θ_{1f} == *"ZE"* ***então*** *ativa a tarefa 18;*

A descrição feita até aqui considerou a transferência da carga do manipulador 1 para o manipulador 2. Entretanto, se a estratégia *Pesa* tivesse ativado a tarefa 10, a estratégia *Doa* seria realizada através das três tarefas descritas no Quadro 5.6 e a transferência da carga seria do manipulador 2 para o manipulador 1, conforme descrito a seguir.

Quadro 5.6 – Tarefas da estratégia *Doa* a partir da tarefa 10.

```
tarefa (13,0,1,1,3,NM); //Aciona o motor 4 e avalia a regra 5.5 a cada 1 ms.
tarefa (4,0,1,20,0,0);  //Treinamento inicial dos motores 1 e 3 a cada 20 ms.
tarefa (8,0,0,20,3,NM); //Tarefa criada mas com o estado desativada, quando ela for
                        //ativada sua finalidade é acionar o motor 3 sob controle e
                        //avaliar a regra 5.6 a cada 20 ms.
```

A tarefa 13, responsável pelo acionamento do motor 4, será ativada a cada 1 ms. A regra (5.5) associada a essa tarefa também será avaliada a cada 1 ms. Quando a condição da regra (5.5) for verdadeira, a tarefa 8 receberá um *tempo* de espera igual a 1 ms (*tempo* == *1*, terceiro parâmetro da tarefa 8). A segunda parte (θ_{3f} == *"ZE"*) da condição da regra (5.5) é utilizada para parar o motor 3 na estratégia *Solta*.

Regra (5.5): ***Se*** $\theta_{3f} == \theta_{3rf}$ ***ou*** θ_{3f} == *"ZE"* ***então*** *ativa a tarefa 18;*

A tarefa 4 é utilizada da mesma forma que na análise anterior. Assim, a tarefa 8 é semelhante à tarefa 7, mudando apenas o número do motor. A regra (5.6) da tarefa 8 também será avaliada a cada 20 ms. Quando a condição da regra 5.6 for verdadeira, então a tarefa 17

será passada para o estado *pronta* para iniciar a estratégia *Pega*. A segunda parte (θ_{3f} == *"ZE"*) da condição da regra (5.6) é utilizada na estratégia *Solta*.

Regra (5.6): ***Se*** $\theta_{3f} == \theta_{3rf}$ ***ou*** $\theta_{3f} ==$ *"ZE"* ***então*** *ativa a tarefa 7;*

5.4.3. A estratégia *Pega*

A estratégia *Pega* é muito parecida com a estratégia *Doa*, sendo que a ordem de acionamento dos dois elos dos manipuladores é invertida. Ou seja, primeiro é posicionado o elo 1, controlado pelo motor 1 ou 3 e depois é posicionado o elo 2, controlado pelo motor 2 ou 4. A estratégia *Pega* inicia com a tarefa 18 (transferência do primeiro manipulador para o segundo manipulador) ou com a tarefa 17 (transferência do segundo manipulador para o primeiro manipulador). Caso a tarefa 18 seja ativada, a estratégia *Pega* é realizada através das duas tarefas descritas no Quadro 5.7.

Quadro 5.7 – Tarefas da estratégia *Pega* a partir da tarefa 18.

```
Ativa_tarefa (18,0,1,20,1,NM); //Aciona o motor 3 sob controle e avalia a regra 5.7 a cada 20 ms.
Ativa_tarefa (15,0,0,1,1,NM);  //Tarefa criada mas não acionada. Sua finalidade é
                               // acionar o motor 4 e avaliar a regra 5.8 a cada 1 ms.
```

A tarefa 18, responsável pelo acionamento do motor 3 sob o controle do SCI, será ativada a cada 20 ms, assim como sua regra (5.7). Da mesma forma que as outras tarefas com regra, o ângulo associado à regra será avaliado para permitir que a próxima tarefa seja ativada. Quando a condição da regra (5.7) for verdadeira, a tarefa 15 receberá um *tempo* de espera igual a 1 ms para ser ativada. A segunda parte (θ_{3f} == *"ZE"*) da condição da regra (5.7) é utilizada na estratégia *Recebe*.

Regra (5.7): ***Se*** $\theta_{3f} == \theta_{3rf}$ ***então*** $\theta_{3f} = 1$ ***senão se*** $\theta_{3rf} ==$ *"ZE"* ***então*** *ativa a tarefa 11;*

A tarefa 15, responsável pelo acionamento do motor 4, será ativada a cada 1 ms. Essa tarefa é ativada quando o elo 1 já está posicionado e é inferida através da regra (5.8). Quando a condição da regra (5.8) for verdadeira, a tarefa 12 será desativada (*freq* == *0*, quarto parâmetro da tarefa 12), para iniciar a estratégia *Solta*. A segunda parte (θ_{3f} == *"ZE"*) da condição regra (5.8) é utilizada na estratégia *Recebe*.

Regra (5.8): ***Se*** $\theta_{3f} == \theta_{3rf}$ ***ou*** $\theta_{3f} ==$ *"ZE"* ***então*** *desativa a tarefa 12,* θ_{2rf} = *"ZE";*

Considerando-se agora a estratégia *Pega* iniciada através da tarefa 17 (transferência do segundo manipulador para o primeiro manipulador), a estratégia *Pega* é realizada através das duas tarefas descritas no Quadro 5.8.

Quadro 5.8 – Tarefas da estratégia *Pega* a partir da tarefa 17.

```
Ativa_tarefa (17,0,1,20,1,NM); // Aciona o motor 1 sob controle e avalia a regra 5.9 a cada 20 ms.
Ativa_tarefa (14,0,0,1,1,NM);  //Tarefa criada mas não acionada, sua finalidade é
                               //acionar o motor 2 e avaliar a regra 5.10 a cada 1 ms.
```

A tarefa 17, responsável pelo acionamento do motor 1 sob o controle do SCI, será ativada a cada 20 ms, com a regra (5.9) associada a ela. Quando a condição da regra (5.9) for verdadeira, então a tarefa 14 receberá um *tempo* de espera igual a 1 ms para ser ativada. A segunda parte (θ_{1f} == *"ZE"*) da condição regra (5.9) é utilizada na estratégia *Recebe*.

Regra (5.9): ***Se*** $\theta_{1f} == \theta_{1rf}$ ***então*** *ativa tarefa 12* ***senão se*** $\theta_{1f} ==$ *"ZE"* ***então*** *ativa tarefa 14;*

A tarefa 14, responsável pelo acionamento do motor 2, será ativada a cada 1 ms. Essa tarefa é ativada quando o elo 1 já está posicionado e é inferida através da regra (5.10), que também será avaliada a cada 1 ms. Quando a condição da regra (5.10) for verdadeira, então a tarefa 13 será desativada (*freq == 0*, quarto parâmetro da tarefa 13) para iniciar a estratégia *Solta*. A segunda parte (θ_{1f} == *"ZE"*) da condição regra (5.9) é utilizada na estratégia *Recebe*.

Regra (5.10): ***Se*** $\theta_{1f} == \theta_{1rf}$ ***ou*** $\theta_{1f} ==$ *"ZE"* ***então*** *desativa a tarefa 13,* θ_{4rf} = *"ZE";*

5.4.4. A estratégia *Solta*

A estratégia *Solta* é utilizada para desligar os motores do manipulador que está doando a carga. Essa estratégia inicia com a tarefa 12 (transferência do primeiro manipulador para o segundo manipulador) ou com a tarefa 13 (transferência do segundo manipulador para o primeiro manipulador). Caso a tarefa 12 seja ativada, a estratégia *Solta* é realizada através das duas tarefas descritas no Quadro 5.9.

Quadro 5.9 – Tarefas da estratégia *Solta* a partir da tarefa 12.

```
Ativa_tarefa (12,1,0,0,1,ZE);  //Desliga o motor 2 e avalia a regra 5.3 explicada na estratégia Doa.
Ativa_tarefa (7,0,0,20,1,ZE);  // Tarefa criada mas não acionada, sua finalidade é
                               // acionar o motor 1 sob controle e avaliar a regra 5.4 a cada 20 ms.
```

A tarefa 12, responsável pelo acionamento do motor 2, que neste caso será desativada e conseqüentemente o motor 2 é desligado, será ativada uma única vez para que a regra (5.3) seja avaliada. Essa regra foi apresentada na estratégia *Doa* (Seção 5.4.2) e tem a finalidade de permitir a ativação da tarefa 7.

A tarefa 7 agora é utilizada para movimentar o manipulador doador para a posição de repouso com θ_{1rf} == *"ZE"*. O motor 1 não é simplesmente desligado para evitar um movimento muito brusco. Por isso, o ângulo de referência recebe "ZE" para que o controlador conduza o elo para a posição de repouso. A regra (5.4), associada a tarefa 7 e apresentada na estratégia *Doa* (Seção 5.4.2), é utilizada para ativar a tarefa 18. Quando a condição da regra (5.4) for verdadeira, então a tarefa 18 será passada para o estado *pronta* para iniciar a estratégia *Recebe*.

Considerando-se agora que a estratégia *Solta* inicia com a regra 13 (transferência do segundo manipulador para o primeiro manipulador), a estratégia *Solta* é realizada através das duas tarefas descritas no Quadro 5.10.

Quadro 5.10 – Tarefas da estratégia *Solta* a partir da tarefa 13.

```
Ativa_tarefa (13,1,0,0,1,ZE);  //Desliga o motor 4 e avalia a regra 5.5 explicada na estratégia Doa.
Ativa_tarefa (8,0,0,20,1,ZE);  // Tarefa criada mas não acionada. A sua finalidade é
                               // Acionar o motor 3 sob controle e avalia a regrar 5.6 de 20 em 20 ms.
```

A tarefa 13, responsável pelo acionamento do motor 4, que neste caso será desativada e conseqüentemente o motor 4 é desligado, será ativada uma única vez para que a regra (5.5) seja avaliada. Essa regra foi apresentada na estratégia *Doa* (Seção 5.4.2) e tem a finalidade de permitir a ativação da tarefa 8.

A tarefa 8 agora é utilizada para movimentar o manipulador Doador para a posição de repouso com θ_{3r} == *"ZE"*. A regra (5.6), associada a tarefa 8 e apresentada na estratégia *Doa*

(Seção 5.4.2), é utilizada para ativar a tarefa 17. Quando a condição da regra (5.6) for verdadeira, então a tarefa 17 será passada para o estado *pronta* para iniciar a estratégia *Recebe*.

5.4.5. A estratégia *Recebe*

Esta é a ultima estratégia da transferência de carga entre os dois manipuladores e tem por finalidade mover o manipulador receptor para a posição de repouso. A estratégia *Recebe* inicia com a tarefa 18 (transferência do primeiro manipulador para o segundo manipulador) ou com a tarefa 17 (transferência do segundo manipulador para o primeiro manipulador). Caso a tarefa 18 seja ativada, então a estratégia *Recebe* é realizada através das duas tarefas descritas no Quadro 5.11.

Quadro 5.11 – Tarefas da estratégia *Recebe* a partir da tarefa 18.

```
Ativa_tarefa (18,0,1,20,1,ZE); // Aciona o motor 3 sob controle e avalia a regra 5.7 a cada 20 ms.
Ativa_tarefa (11,0,0,0,0,0);   // Tarefa criada mas não acionada. A sua finalidade é
                               // encerrar todo o processo de transferência de carga.
```

A tarefa 18 agora é utilizada para movimentar o manipulador receptor para a posição de repouso com $\theta_{3rf} ==$ *"ZE"*. Conforme já explicado na seção anterior, o motor 3 não é simplesmente desligado para evitar um movimento muito brusco. A regra (5.7), associada à tarefa 18 e apresentada na estratégia *Pega* (Seção 5.4.3), é utilizada para ativar a tarefa 11. Quando a condição da regra (5.7) for verdadeira, então a tarefa 11 será passada para o estado *pronta* para encerrar todo o processo de transferência de carga.

A tarefa 11 é utilizada para desativar todas as tarefas e encerrar a transferência com todos os motores em seus estados de repouso. Essa tarefa não possuí nenhuma regra; todas as vezes que ela é ativada, ela desliga todos os motores e interrompe o executivo em tempo real que implementa o Escalonador Inteligente de Tarefas.

Considerando-se que a estratégia *Recebe* inicia com a tarefa 17 (transferência do segundo manipulador para o primeiro manipulador), a estratégia *Recebe* é realizada através das duas tarefas descritas no Quadro 5.12.

Quadro 5.22 – Tarefas da estratégia *Recebe* a partir da tarefa 17.

```
Ativa_tarefa (17,0,1,20,1,ZE); // Aciona o motor 1 sob controle e avalia a regra 5.9 a cada 20 ms.
Ativa_tarefa (11,0,0,0,0,0); // Encerra todo o processo de transferência de carga.
```

A tarefa 17 agora é utilizada para movimentar o manipulador receptor para a posição de repouso com $\theta_{1rf} == "ZE"$. Assim como nas outras tarefas que desligam os motores 1 e 3, essa tarefa também não desliga imediatamente o motor 1 para evitar um movimento muito brusco. A regra (5.9), associada à tarefa 17 e apresentada na estratégia *Pega* (Seção 5.4.3), é utilizada para ativar a tarefa 11, que encerra toda a transferência.

5.5. Resultados experimentais

Os resultados experimentais obtidos com o EIT demostraram a otimização do tempo durante a transferência da carga entre os dois manipuladores. As curvas obtidas com o EIT durante a transferência no intervalo de tempo de 4000 ms é ilustrada na Figura 5.3. Inicialmente a carga estava com o manipulador 1. A estratégia *Pesa* não é ilustrada nesta figura. A posição angular θ_1 do manipulador 1 durante todo o intervalo de tempo da transferência é ilustrada na Figura 5.3a. A posição angular θ_3 do manipulador 2 durante todo o intervalo de tempo da transferência é ilustrada na Figura 5.3b. Inicialmente os dois manipuladores estão na posição de repouso. As estratégias foram executadas seqüencialmente, sem no entanto um tempo predefinido para a sua execução. Os valores de θ_1 e θ_3 na ordenada são dados em quantidade de furos do detector de posição (ver Apêndice B).

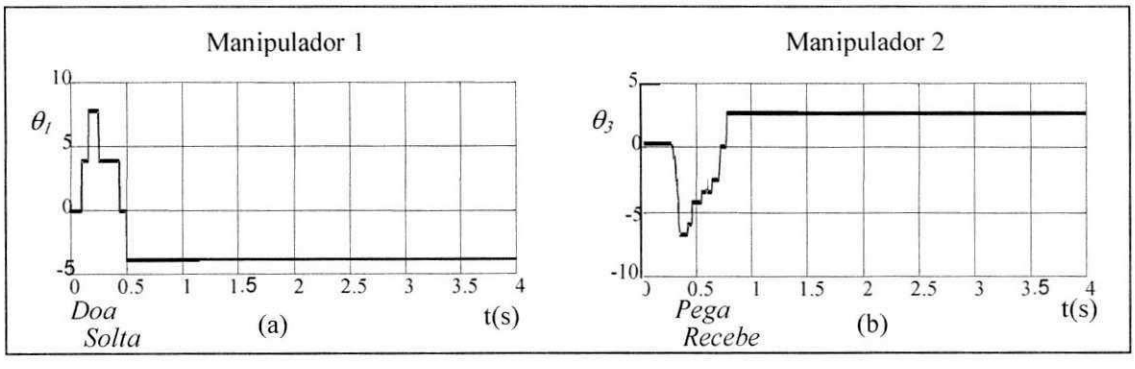


Figura 5.3 – Resultados experimentais da transferência de carga com o EIT.

5.6. Conclusão

O escalonamento descrito neste capítulo apresentou duas vantagens fundamentais sobre o escalonamento tradicional descrito no Capítulo 4. Em primeiro, lugar o controle do tempo passou a ser feito totalmente pelo próprio sistema, sem a necessidade de interferências por parte do especialista. Em segundo lugar, todo o controle foi feito com base em informações percebidas no ambiente pelo próprio sistema, o que o coloca na classe dos sistemas inteligentes com adaptação em tempo real. Portanto, conclui-se que o EIT é superior ao ETR, além de ser aplicável em sistemas reais com prototipagem.

Esse tipo de controle inteligente, conhecido como sistemas neuro-nebulosos, representa um grande avanço nos sistemas de controle, permitindo que o modelo da planta sob controle seja aprendido dinamicamente, dispensando grande parte do formalismo matemático que seria necessário para realizar tal controle. Entretanto, a distribuição das regras nas tarefas limita a generalização da idéia proposta para ser utilizada em outras aplicações diferentes das aplicações robóticas, como no caso dos sistemas especialistas. Ela é, no entanto, perfeitamente aplicável em sistemas baseados em autômatos, ou seja, sistemas com mudança de estado e com estados finitos.

Capítulo 6

Conclusão

6.1. Resultados obtidos

Os resultados obtidos na implementação final do Escalonador Inteligente de Tarefas (EIT) permitiu uma representação fiel da seqüência de ativação das tarefas (tempo) e avaliação das regras (posição) de cada estratégia. A inclusão das regras possibilitou a verificação do posicionamento de cada manipulador durante a execução das estratégias. A especificação de um tempo máximo de execução de cada estratégia permitiu a verificação de falhas no movimento da carga entre os manipuladores.

Embora todas as tarefas tenham sido criadas pelo especialista, o sistema depende muito menos do conhecimento deste, pois a execução das tarefas é definida pelo próprio EIT de acordo com as informações percebidas no ambiente pelos próprios manipuladores e avaliadas através de inferências nebulosas.

O EIT mesmo sendo baseado em regras nebulosas, não necessita de uma base de conhecimento tradicional para o armazenamento das regras pois elas são associadas diretamente às tarefas, sendo avaliadas imediatamente após a execução das tarefas. Esta característica do EIT tornou desnecessária a realização de pesquisas de regras, sendo a avaliação de cada regra imediata, sem perda de tempo com pesquisas na base de conhecimento.

Outra vantagem do EIT é a sua adaptação a diferentes ambientes, pois a sua aquisição de informações do ambiente é dinâmica, dependente das mudanças no seu ambiente. Essa característica além de permitir a adaptação mais rápida, exige pouca informação por parte do usuário sobre o ambiente, pois o próprio sistema pode realizar as suas próprias inferências sobre ele.

6.2. Trabalhos futuros

Baseado nos resultados obtidos com o EIT, sugere-se alguns estudos que podem dar continuidade a este trabalho no desenvolvimento de um sistema robótico que, acredita-se, tornar-se-á cada vez mais autônomo:

- Aprendizagem automática – implementando-se o sistema robótico com aquisição de conhecimento proposto por Ferreira et al. (1999b) através da percepção das informações do ambiente com o agrupamento dessas informações em classes através de semelhanças seguido da classificação das informações com o objetivo de geração de regras pelo próprio sistema. As regras podem ser geradas ou modificadas pelo sistema em tempo real, melhorando o EIT onde as regras são construídas previamente pelo especialista, talvez utilizando algoritmos genéticos;

- Planejamento hierárquico – a definição de uma estrutura hierárquica, utilizando os conceitos de escalonamento inteligente, pode permitir o desenvolvimento de uma ferramenta para a estruturação de sistemas robóticos mais adaptativos, seguindo a direção de sistemas inteligentes baseados em aprendizagem a partir de experiências. Tal método é proposto por Albus (1996), Albus & Proctor (1996) e Meystel & Lacaze (1997);

- Escalonamento preemptivo – permitir que o número de tarefas seja aumentado sem restrições para que o escalonador possa ser utilizado no gerenciamento de sistemas mais robustos. Essa robustez está relacionada com o fato de que o controle é feito independentemente da variação dos parâmetros da planta;

- Escalonamento tolerante a falhas – com a introdução do conceito de inteligência para o escalonador, e por conseqüência tornando-o mais independente, há a necessidade de se definir módulos de tolerância a falhas no sistema para que o escalonamento seja mais confiável e aumente o seu desempenho;

- Concepção baseada em agentes – seria interessante realizar um estudo de um supervisor inteligente para agentes, onde estes substituirão as tarefas do sistema, aproveitando todos os conceitos existentes sobre agentes e sistemas inteligentes (Silva, 1998).

Referências bibliográficas

Albus, J. S. (1996). The engineering of mind. In: International Conference on Simulation of Adaptive Behavior: From Animals to Animates 4, 4th, 1996. Proceedings. Cape Code, MA, September 1996.

Albus, J. S. and Proctor, F. G. (1996). A reference model architecture for intelligent hybrid control systems. In: Triennial World Congress, International Federation of Automatic Control (IFAC), 1996. Proceedings. San Francisco, CA, July 1996.

Alsina, P. J. (1996) Controle neuroadptativo modular de manipuladores robóticos. Campina Grande: Novembro de 1996. 154p. Tese (Doutorado em Engenharia Elétrica; Processamento da informação) – Universidade Federal da Paraíba.

Alsina, P. J. e Cavalcanti, J. H. F. (1997) Real time intelligent control system for load exchange between two robots. In: WORKSHOP ON INTELLIGENT ROBOTICS, 2°, 1997. Anais. Brasília. 1991. 1v., p.80-88.

Alsina, P. J. e Gehlot, N. S. (1996a) Controle de motor de indução usando redes neurais. Campina Grande: 1996. 51p. Relatório técnico N.° RT00072/97 (Biblioteca setorial do Departamento de Engenharia Elétrica) – Universidade Federal da Paraíba.

Alsina, P. J. e Gehlot, N. S. (1996b) Controle e identificação de manipuladores usando redes neurais. Campina Grande: 1996. 57p. Relatório técnico N.° RT00085/97 (Biblioteca setorial do Departamento de Engenharia Elétrica) – Universidade Federal da Paraíba.

Araújo, E. A. (1998) Questões metodológicas na pesquisa contemporânea. Campina Grande: 1996. 16p. Curso de Elaboração de Projeto de Pesquisa – Universidade Federal da Paraíba.

Bezdek, J. (1994) Editorial; Fuzzy models – What are they, and why?. In: Marks II, Robert J. Fuzzy logic technology and applications. New York: IEEE Technical activities board, 1994. 575p., editorial, p.3-7.

Blythe, J. e Veloso, M. M. (1997) In: INNOVATIVE APPLICATIONS OF ARTIFICIAL INTELLIGENCE. Proceedings of IAAI-97. Rhode Island, 1997. pp. 668-673.

Cavalcanti, J. H. F. (1994) Controladores neurais adaptativos. Campina Grande: Outubro de 1994. 122p. Tese (Doutorado em Engenharia Elétrica; Processamento da informação) – Universidade Federal da Paraíba.

Cavalcanti, J. H. F. (1998) Sistemas inteligentes; Redes neurais e lógica fuzzy. Campina Grande: (Não publicado), 1998. 135p. (Universidade Federal da Paraíba – Departamento de Engenharia Elétrica).

Cavalcanti, J. H. F. e Ferneda, E. (1995) Intelligent control of an inverted pendulum – training and evolution. In: SEMISH'95 – Conferência Latino-americana de Informática, 21ª, 1995. Anais. Canelas RS, 1995. 2v., v.1, p.627-637.

Cavalcanti, J. H. F., Alsina, P. J. e Ferneda, E. (1999a) Posicionamento de um pêndulo invertido usando algoritmos genéticos. SBA Controle & Automação, São Paulo, v.10, n.1. p.31-38, 1999.

Cavalcanti, J. H. F., Lima, A. M. N. e Deep, G. S. (1994) Posicionamento de um pêndulo invertido usando redes neurais. In: JORNADA ARGENTINA DE INFORMATICA E INVESTIGACION OPERATIVA, 23ª, 1994. Anais. Buenos Aires: Centro Cultural Gral. San Martin, 1994, 1v., p.307-317.

Comer, D. (1988) Projeto de sistema operacional; O enfoque XINU. Rio de Janeiro: Ed. Campus, 1988. 451p.

Cui, X. e Shin, K. G. (1996) Intelligent coordination of multiple systems with neural networks. In: Gupta, Madan M. and Sinha, Naresh K. Intelligent control systems. New Jersey: IEEE Press, 1996. 820p., cap.9, p. 206-233.

Ferreira, J. R. S. e Cavalcanti, J. H. F. (1999a) Interfaces controladoras de motores. Campina Grande: 1999. 25p. Relatório técnico N.º DSC/001/99 (Biblioteca setorial do Departamento de Sistemas e Computação) – Universidade Federal da Paraíba.

Ferreira, J. R. S. e Cavalcanti, J. H. F. (1999b) Interface controladora de motores de corrente contínua. Revista Interação (Centro de Ciências e Tecnologia da UFPB), Campina Grande, 1999. (submetido).

Ferreira, J. R. S., Alsina, P. J. e Cavalcanti, J. H. F. (1998) Escalonador inteligente de tarefas aplicado à robótica. In: INDUSCON'98 – CONFERÊNCIA DE APLICAÇÕES INDUSTRIAIS, 3º, 1998. Anais. São Paulo: Escola Politécnica da USP, 1998. 1v., p.117-120.

Ferreira, J. R. S., Cavalcanti, J. H. F. e Alsina, P. J. (1999a) Intelligent tasks scheduler. In: IJCNN'99 – INTERNATIONAL JOINT CONFERENCE ON NEURAL NETWORKS, 1999. Proceedings. Washington, D.C. USA: Renascence Hotel, (artigo aceito a ser publicado).

Ferreira, J. R. S., Cavalcanti, J. H. F. e Alsina, P. J. (1999b) An application using intelligent tasks scheduling and knowledge acquisition. In: CARS&FPF'99 – INTERNATIONAL CONFERENCE ON CAD/CAN, ROBOTICS AND FACTORIES OF THE FUTURE, 15º, 1999. Proceedings. Águas de Lindóia, São Paulo: Vacance Hotel, (artigo aceito a ser publicado).

Guimarões, C. C. (1989) Princípios de sistemas operacionais. Rio de Janeiro: Editora Campus, 1989. 222p.

Gupta, M. M. e Sinha, N. K. (1996) Intelligent control systems. New Jersey: IEEE Press, 1996. 820p.

Handelman, D. A. e Lane, S. H. (1996) Human-to-machine skill transfer through cooperative learning. In: Gupta, M. M. and Sinha, N. K. Intelligent control systems. New Jersey: IEEE Press, 820p., cap.8, p.187-205.

Kamm, L. J. (1996) Understanding Electro-mechanical engineering; An introduction to mechatronics. Piscatawa, NJ, USA: IEEE Press, 1996.

Kartalopoulos, S. V. (1996) Understanding neural networks and fuzzy logic; Basic concepts and applications. New Jersey: IEEE Press, 1996. 493p.

Klir, G. J. e Folger, T. A. (1988) Fuzzy sets, uncertainty, and information. New Jersey: Prentice Hall, 1988. 355p.

Kopetz, H. (1995) Scheduling. In: Mullender, S. Distributed Systems. New York: ACM Press, 1995. 601p., cap.18, p.491-509.

Kovács, Z. L. (1996) Redes neurais artificiais; fundamentos e aplicações. São Paulo: Edição Universitária, 1996. 164p.

Lima, A. M. N., Cavalcanti, J. H. F. e Deep, G. S. (1994) On-line Training of Adaptive Neural Network Controller. In: IEEE INDUSTRIAL ELECTRONICS SOCIETY IECON, Proceedings Bologna, 1994, 1v, p.1392-1395.

Meystel, A. e Lacaze, A. (1997) Unified Learning/Planning Automaton: Generating and Using Multigranular Knowledge Hierarchies. In: International conference on intelligent systems and semiotcs. 1997. Proceedings. Gaithersburg, MD. pp117-123.

Mullender, S. (1995) Kernel support for distributed systems. In: Mullender, S. Distributed Systems. New York: ACM Press, 1995. 601p., cap.15, p.385-409.

Narendra, K. S. e Pathasaranthy, K. (1990) Identification and control of Dynamical Systems Using Neural Networks. IEEE Transactions On Neural Networks, New Jersey, v.1, n.1. p.4-27, 1990.

Osumi, H. e Arai, T. (1994) Cooperative control between two position-controlled manipulators. In: INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION, 1994. Proceedings. San Diego, California. 1994. 2v., v.2, p.1509-1514.

Pedrycz, W. e Gomide, F. A. (1998) An introduction to fuzzy sets; analysis and design. Massachusetts, USA: A Bradford Book, 1998. 465p.

Proctor, F., Michalosk, J. e Kramer, T. (1992) A methodology for integrating sensor feedback in machine tool controllers. In: International conference on Flexible automation and Information Management, 2nd, 1992. Washington, DC. 1992.

Rangwala, S. S. e Dornfeld, D. A. (1989) Learning and optimization of machining operation using abilities of neural networks. IEEE Transactions On Systems, New Jersey, v.1, n.2. March/April. p.229-314, 1999.

Rumelhart, D. E., Hinton, G. H. e Wiliams, R. J. (1986) Learning Internal Representations by Error Propagation. In: Parallel Distributed Processing Explorations in the Microstructures Cognition, Vol. 1, MIT Pess. 1986. p.318-362.

Sales Jr., Esdras F. (1997) Sistema de controle inteligente para um elo robótico. Campina Grande: Dezembro de 1997. 70p. Dissertação (Mestrado em Informática; Inteligência Artificial) – Universidade Federal da Paraíba.

Shay, Willian A. (1996) Sistemas Operacionais. São Paulo: Makron Books, 1996. 758p.

Silva, M. E. S. (1998) De agentes racionais a agentes semióticos. Campina Grande: Abril de 1998. 130p. Dissertação (Mestrado em Informática; Inteligência Artificial) – Universidade Federal da Paraíba.

Stone, P. and Veloso, M. M. (1998) Towards collaborative and adversarial learning: A case study in robotic soccer. In: International Journal of Human-Computer Systems, 1998.

Tanenbaum, A. S. (1995) Sistemas Operacionais Modernos. Rio de Janeiro: Prentice-Hall do Brasil, 1995. 493p.

Veloso, M. M., Mulvehill, A. e Cox, M. (1997) Rationale-supported mixed-initiative case-based planning. In: INNOVATIVE APPLICATIONS OF ARTIFICIAL INTELLIGENCE. Proceedings of IAAI-97. Rhode Island, 1997. pp. 1072-1077.

Yen, J., Langari, R. e Zadeh, L. A. (1995) Industrial applications of fuzzy logic and intelligent systems. New York: IEEE Press, 1995. 365p.

Zadeh, L. A. (1994) Preface. In: Marks II, R. Fuzzy logic technology and applications. New York: IEEE Technical activities board, 1994. 575p., preface, p.xvii-xviii.

Apêndices

Apêndice A – Interface motores CC

Os motores elétricos de corrente contínua são os dispositivos eletromecânicos amplamente utilizados (Kamm et al., 1996) e por isso o controle destes dispositivos é muito importante para as mais variadas áreas. Esses motores são atuadores que possuem apenas dois estados – ligado ou desligado. Entretanto, podem ter sua velocidade controlada através de sinal PWM. A seguir apresenta-se a descrição de uma interface para o controle de até quatro motores de corrente contínua por microcomputadores do tipo IBM-PC através da porta paralela.

A porta paralela de um microcomputador pessoal pode ser utilizada como um canal de entrada e saída do microcomputador. A distribuição dos sinais dessa porta num conector DB-25, conforme ilustrado na Figura A.1, é a seguinte: 8 sinais de saída (para dados, pinos 2 a 9), 4 sinais de saída (para controle, pinos 10 a 13), 5 sinais de entrada (para controle, pinos 1, 14, 15, 16 e 17) e 8 pinos ligados ao terra (pinos 18 a 25).

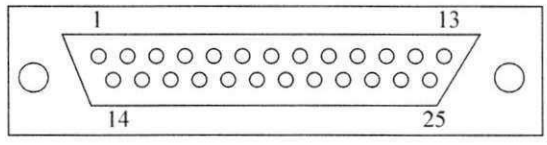


Figura A.1 – Conector DB25/Fêmea.

A interface controladora apresentada a seguir (Ferreira & Cavalcanti, 1999a) (Ferreira & Cavalcanti, 1999b) é utilizada para controlar quatro motores de corrente contínua com excitação de campo separada (ímã permanente) de baixa potência. O diagrama esquemático dessa interface é ilustrado na Figura A.2. O controle de cada motor é feito através de dois sinais de dados, sendo os oitos sinais de dados da porta paralela utilizados para controlar os quatro motores. A representação do conector *Centronic*[14], na Figura A.2, rotulado no diagrama esquemático dessa mesma figura como CN1, identifica a ligação desse oitos sinais de dados com os respectivos circuitos dos quatro motores. O sinal CN1:2 representado pelo símbolo ▭

[14] Conector utilizado nos cabos de impressoras para conexão com impressoras compatíveis com o IBM-PC.

na Figura A.2 (primeira parte do diagrama esquemático), representa o sinal de acionamento do motor 1 (tensão de armadura). Os outros sinais seguem essa mesma nomenclatura.

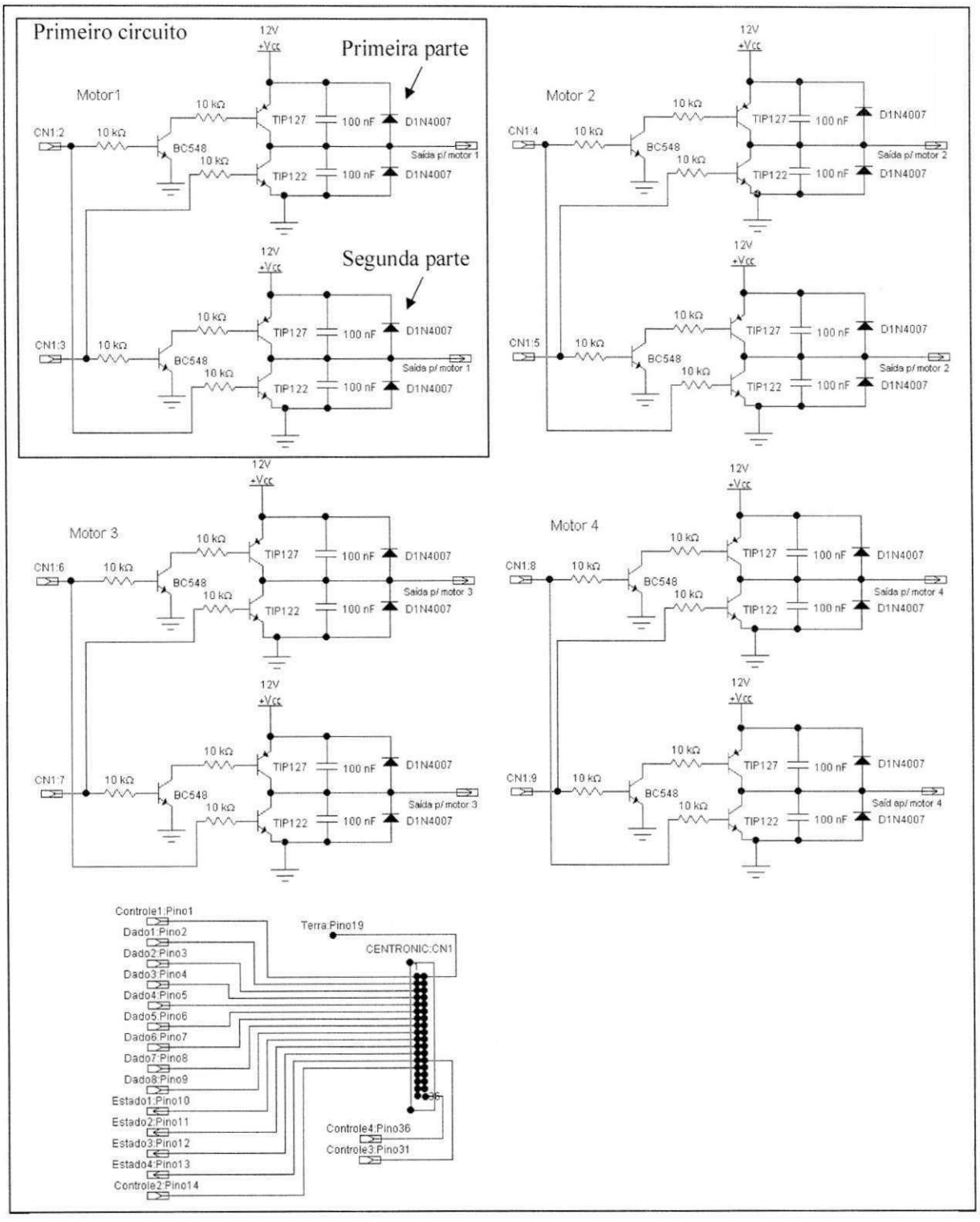


Figura A.2 – Diagrama esquemático da interface controladora de motores CC.

O diagrama esquemático ilustrado na Figura A.2, está divido em cinco partes, sendo as quatro primeiras a representação dos circuitos que controlam os quatro motores desenvolvidos

no NEUROLAB, e a última a representação da ligação dos pinos do conector CN1. Os diagramas dos circuitos dos quatro motores são idênticos, diferindo apenas nos sinais que chegam e saem de cada um deles.

Nesse circuito chegam dois sinais (CN1:2 e CN1:3) provenientes do microcomputador. Eles controlam o acionamento do motor 1 e são ligados as duas partes do primeiro circuito, sendo que na primeira parte o sinal CN1:2 está ligado à base de um transistor (BC 548), com um resistor de 10 KΩ para proteção, enquanto o outro sinal está ligado direto à base de um transistor de potência (TIP 122), também com um resistor de 10 KΩ para proteção. Essas ligações se invertem na outra parte do circuito, ou seja, o sinal CN1:2 é ligado à base do outro TIP 122 e o sinal CN1:3 é ligado à base do outro BC 548.

Analisando a primeira parte com valores arbitrários de 5V e 0V para os dois sinais CN1:2 e CN1:3 respectivamente, tem-se a base do BC 548 saturada e o sinal que chega em seu coletor passa para o emissor. A base do transistor de potência (TIP 127) está ligada ao coletor do BC 548, com um resistor de 10 KΩ para proteção. Esse TIP 127, que tem 12V ligado ao seu emissor, passa então a conduzir para o seu coletor. O coletor deste TIP 127 está ligado ao sinal de saída CN2:2 que vai direto para o motor. Um capacitor (100 nF) e um diodo (1N4007) estão ligados em paralelo com o TIP 127 para reduzir ruídos e evitar que o pulso de tensão gerado pela indutância do motor danifique o circuito. O TIP 122 que recebe o segundo sinal (0V) tem seu coletor ligado também ao sinal de saída (CN2:2). Nesse caso, esse TIP não conduz do coletor para o emissor porque a sua base não está saturada, visto que o segundo sinal é 0V. Portanto, esta parte do circuito garante uma voltagem de 12 V no sinal de saída CN2:2, que está ligado diretamente a uma das duas fases do Motor CC.

Analisando a outra parte do primeiro circuito, ainda com valores de 5V e 0V para os dois sinais CN1:2 e CN1:3 respectivamente, tem-se o inverso da primeira parte. A base do BC 548, que recebe o segundo sinal (0V), não está saturada e portanto o sinal que chega em seu coletor não passa para o emissor. Assim, o TIP 127 com 12V aplicados ao seu emissor, não conduz para o coletor pois não há corrente na sua base. O coletor deste TIP 127 está ligado ao sinal de saída (CN2:3) que vai direto para o motor. Um capacitor (100 nF) e um diodo (1N4007) funcionam da mesma forma como explicado no parágrafo anterior. O TIP 122 que recebe o primeiro sinal de 5V tem seu coletor ligado também ao sinal de saída. Nesse caso, esse TIP 122 conduz do coletor para o emissor porque a sua base está saturada, visto que o primeiro sinal é de 5V. Isso faz com que este TIP 122 conduza o sinal CN2:3 para o terra.

Como esse sinal está ligado diretamente a outra fase do Motor CC e o mesmo recebeu 12V do primeiro sinal analisado acima, o motor entra em funcionamento. Invertendo os sinais de entrada, ou seja, 0V e 5V, o sentido de rotação do motor será invertido. A interface controladora de motores de corrente contínua utiliza os oito sinais de dados. Os demais sinais são utilizados para a leitura dos detectores de posição. Portanto, esses sinais passam pela interface, mas somente para serem transferidos do circuito dos detectores de posição para o microcomputador. Os componentes utilizados na interface são listados na Tabela A.1.

Tabela A.1 – Relação dos componentes utilizados na interface.

Componente	**Valor**	**Qtde**
Resistor	10K	24
Transistor	BC 548	8
Transistor de potência	TIP 127	8
Transistor de potência	TIP 122	8
Capacitor poliester	100 nF	16
Diodo	1N4007	16
Conector *Centronic*	Fêmea	1

O diagrama esquemático do circuito de acionamento de um motor em forma de ponte monofásica completa é ilustrado na Figura A.3. Observa-se que os sinais CN1:2 e CN1:3 não podem assumir o nível lógico 1 simultaneamente.

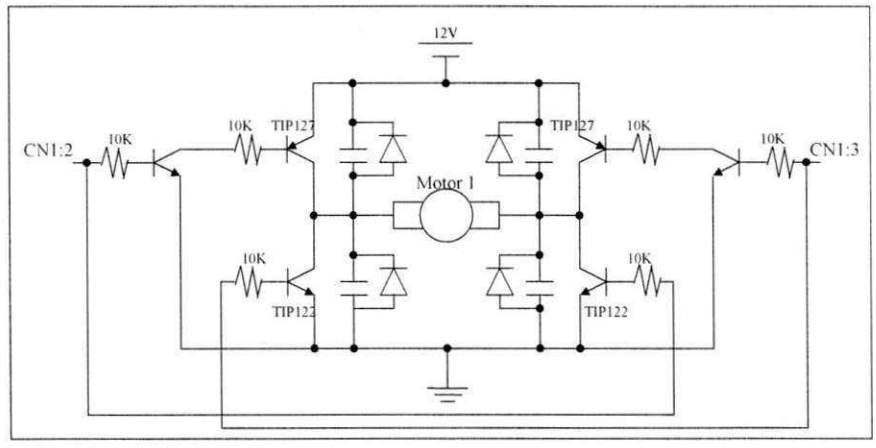


Figura A.3 – Circuito de acionamento na forma de ponte monofásica completa.

Apêndice B

O detector de posição na estratégia *Pesa*

Todos os movimentos gerados pelos motores são percebidos através dos detectores de posição, também conhecidos como "*encoders*". Os detectores atuam como sensores e informam o sentido e a intensidade do movimento do motor. O esquema de montagem de um detector de posição juntamente com o motor e o disco com furos é ilustrado na Figura B.1. Um estudo da codificação dos sinais através de detectores de posição é apresentado a seguir.

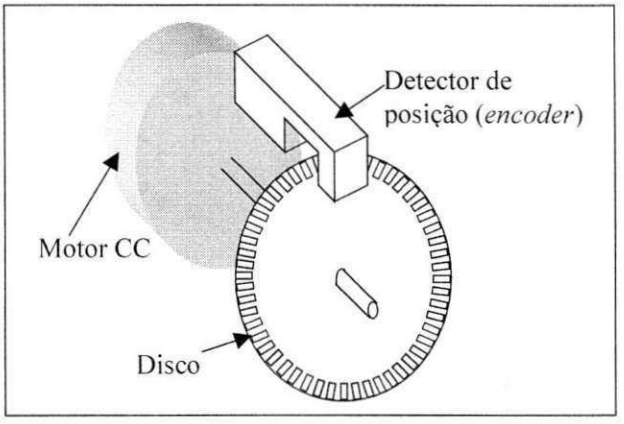


Figura B.1 – Esquema de montagem de um detector de posição.

Os detectores de posição, acoplados aos motores, precisam ser duplos, ou seja possuírem dois sinais para permitir a identificação do sentido da rotação. Um detector de posição com um único sinal pode simplesmente detectar se houve ou não movimento. Já um detector de posição duplo, além de verificar se houve movimento, pode também informar em que sentido ocorreu o movimento e medir esse movimento. Isto é possível porque a presença de dois sinais defasados, gerados pelo par de detectores de posição. Por exemplo, quando um dos sinais está adiantado (atrasado) em relação ao outro, têm-se o motor girando no sentido horário (anti-horário). Uma análise gráfica detalhada de um detector de posição duplo é feita na Figura B.2. O esquema de um detector de posição duplo é ilustrado na Figura B.2a. Os quatro possíveis posicionamentos do disco sobre o detector são ilustrados na Figura B.2b. A análise dos valores dos dois sinais do detector é feita na Figura B.2c e B.2d.

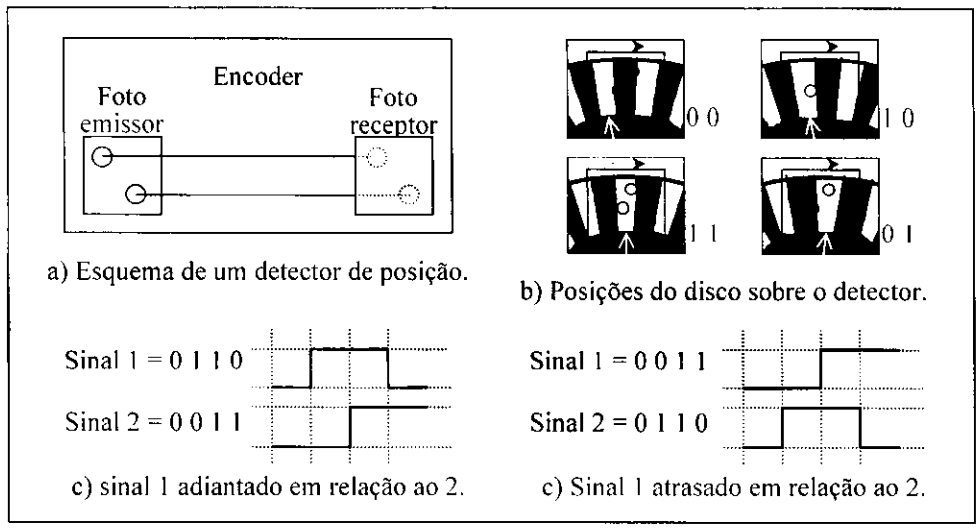


Figura B.2 – Análise gráfica de um detector de posição duplo.

Os discos utilizados no protótipo possuem 72 furos cada um. Como a cada buraco o detector de posição duplo detecta até quatro combinações diferentes de sinais, em um giro de 360° do motor é percebido um deslocamento de até 288 posições. Entretanto, definido-se que o elo 1 do manipulador (Figura B.3) ao girar no sentido anti-horário obtém-se um ângulo positivo e, ao girar no sentido horário, obtém-se um ângulo negativo. Isso conduz a um intervalo máximo de -144 a 144 sinais. Transformando-se os sinais em ângulo tem-se: 1 sinal é equivalente a um giro de 1,25°. Então, se o objetivo é posicionar o elo 1 em 30°, o motor deve girar até ser percebido pelo detector de posição 24 sinais.

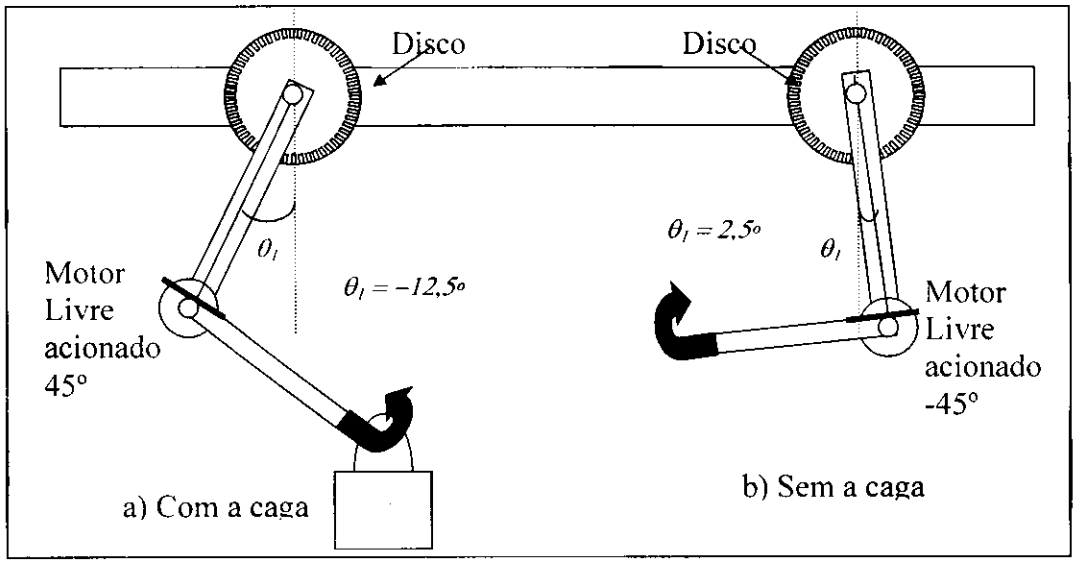


Figura B.3 – Exemplo de movimentação durante a estratégia *Pesa*.

Essa relação entre os sinais percebidos e o ângulo formado pelo elo 1 foi utilizada para definir o gráfico nebuloso da estratégia *Pesa*, apresentada no capítulo 4. Após vários testes "*off-line*", percebeu-se que o manipulador com a carga sempre provocava um deslocamento superior a 10°, ou seja, um deslocamento superior a 8 posições (Figura B.3a), enquanto o manipulador sem a carga nunca provocava um deslocamento superior a 5°, ou seja, o um deslocamento máximo de 4 posições (Figura B.3b). Esses valores dependem do peso da carga.

A partir dessa análise definiu-se os valores para o gráfico nebuloso utilizado na estratégia *Pesa*. Assim, sempre que esta estratégia é executada, a sua tarefa verifica o deslocamento provocado pelos dois manipuladores através das funções de pertinência pequeno (P) e grande (G), conforme ilustrado na Figura B.4. A função de pertinência associada ao valor lingüístico P, indica que o manipulador está sem a carga, enquanto a função de pertinência associada ao valor lingüístico G indica que o manipulador está com a carga. Os valores de 5 e 10 sinais indicados na Figura B.4 podem variar de acordo com o peso da carga.

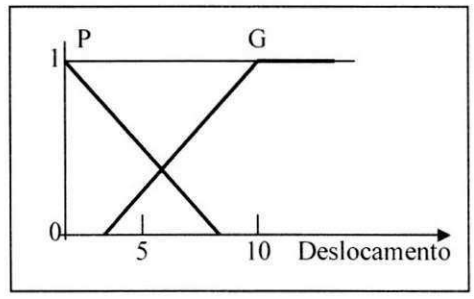


Figura B.4 – Gráfico nebuloso utilizado na estratégia Pesa.

Apêndice C
Controlador neural

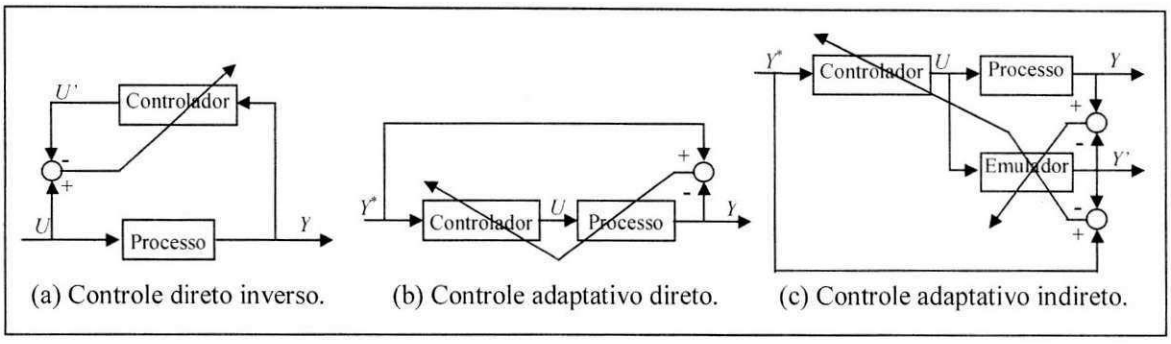
As Redes Neurais Artificiais são consideradas sistemas distribuídos, não lineares e robustos, dotadas da capacidade de aprender complexos mapeamentos não lineares. Sua aplicação em controle de processos dinâmicos é uma conseqüência natural dessas propriedades (Alsina 1996, p.29). Um controlador neural implementado através de uma Rede Neural Multicamadas (RNMC) executa uma forma específica de controle adaptativo (Cavalcanti, 1994, p.34). Para facilitar o entendimento do controlador neural uma breve descrição sobre controle de processos é apresentada a seguir.

A necessidade de controlar determinados processos (acionamento de motores, controle de temperaturas e outros) de forma automática vem sendo estudada há vários anos. Muitos conceitos foram definidos sobre a teoria dos controles em engenharia. Esses conceitos tratam basicamente de duas diretrizes em controle: controle de processos lineares e controle de processos não lineares. A linearidade ou não linearidade de um processo é definida de acordo com a dinâmica da planta a ser controlada.

As técnicas usuais de controle relacionam basicamente o controle proporcional, integral e derivativo (PID) utilizado no controle de processos lineares e o controle adaptativo, utilizado no controle de processos não lineares. Vários problemas tais como modificações no ambiente, mudanças no modelo de referência e diferentes critérios de desempenho, entre outros, caracterizam um problema não linear (Alsina & Gehlot, 1996a) (Alsina & Gehlot, 1996b). Alguns tipos de controle específico foram propostos levando em consideração as várias técnicas de controle e a dinâmica da planta a ser controlada como: controle direto inverso, controle adaptativo direto e controle adaptativo indireto (Narendra & Pathasaranthy, 1990) (Alsina, 1996), ilustrados na Figura C.1.

O controle direto inverso (Figura C.1a) caracteriza-se por aprender a dinâmica inversa do processo, utilizando como padrão de treinamento os valores da entrada U e saída Y, obtidos diretamente através de medições na entrada e saída do processo. U' é a saída gerada pelo con-

trolador. O controle adaptativo direto (Figura C.1b) caracteriza-se por aprender a dinâmica do processo, utilizando como padrão de treinamento o valor do erro obtido diretamente entre a saída Y e a saída desejada Y^*. U é a entrada gerada pelo controlador.



(a) Controle direto inverso. (b) Controle adaptativo direto. (c) Controle adaptativo indireto.

Figura C.1 – Configurações de controles.

O controle adaptativo indireto (Figura C.1c) caracteriza-se por aprender a dinâmica do processo, utilizando como padrão de treinamento o valor do erro obtido através de um emulador, que gera uma saída Y' a partir da saída Y e da saída desejada Y^*. U é a entrada gerada pelo controlador. Um controle neural adaptativo direto, ou seja, um controle adaptativo direto utilizando RNA, torna possível a otimização do desempenho do controlador diretamente em relação ao seu objetivo, que é o controle da saída Y. A configuração de um controle adaptativo direto utilizando um controlador neural é ilustrada na Figura C.2.

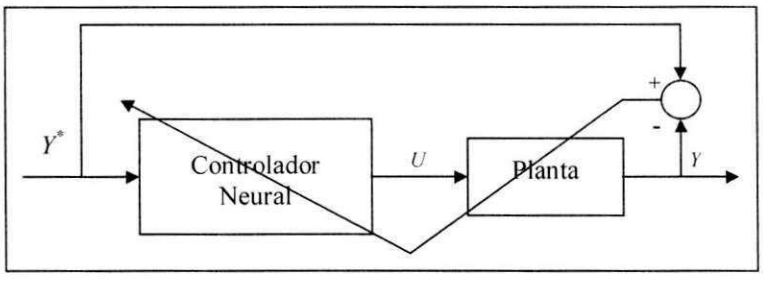


Figura C.2 – Controle neural adaptativo direto.

O controle neural adaptativo direto apresentado na Figura C.2 pode ser utilizado para controlar o Motor CC que movimenta um pêndulo. Durante o controle, o controlador neural deve adaptar a tensão aplicada sobre o Motor CC para que o movimento gerado seja o melhor possível para a estabilização do pêndulo no ângulo desejado. A adaptação do controlador neural deve ser feita preferencialmente em tempo real para garantir que mudanças no ambiente ou variações na dinâmica da planta não prejudiquem o desempenho do controle. Um esquema mais detalhado do sistema do controlador neural para o Motor CC é ilustrado na Figura C.3.

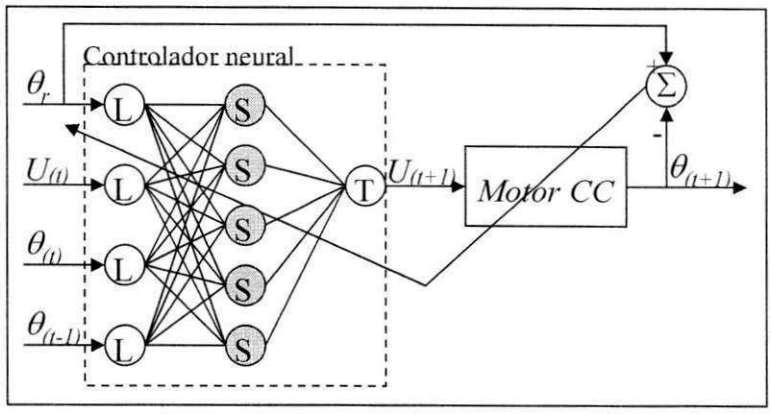


Figura C.3 – Sistema de controle neural adaptativo direto.

As quatro entradas da rede representam o ângulo desejado θ_r, a tensão aplicada sobre o Motor CC $U_{(t)}$, o ângulo atual $\theta_{(t)}$ do rotor do Motor CC em relação a normal e o ângulo $\theta_{(t)}$ obtido no controle anterior. $_{(t)}$ representa o tempo na forma discreta. A saída do controlador neural é dada por $U_{(t+1)}$ que representa a tensão a ser aplicada sobre o Motor CC. $\theta_{(t+1)}$ representa o ângulo gerado pelo Motor CC sob a tensão $U_{(t+1)}$. O símbolo Σ dentro do círculo com os sinais + e – a sua volta, representa o cálculo do erro encontrado na saída do Motor CC e a seta que sai desse símbolo até a RNMC representa o algoritmo de propagação retroativa do erro utilizado para treinar a RNMC.

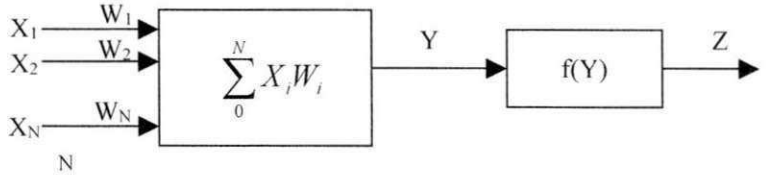
A RNMC do controlador neural é composta por três camadas de neurônios. A primeira camada (entrada) é composta por quatro neurônios lineares (L), onde cada neurônio é responsável por propagar as entradas na rede. A segunda camada (oculta) é composta por cinco neurônios sigmóides (S), responsáveis pelo processamento das entradas. A terceira e última camada (saída) é composta por um único neurônio tangente hiperbólico (T), responsável por fornecer o valor da tensão a ser aplicada ao Motor CC. A utilização de neurônios lineares na camada de entrada é apropriada, pois os neurônios lineares são cabem perfeitamente na função de propagação da informação de entrada. Já a utilização de neurônios sigmóides na camada oculta se fez necessária devido a característica deste tipo de neurônio em ter um grande poder de processamento. A quantidade de neurônios nesta camada foi proposta por Cavalcanti (1994, p.36). Após vários testes de desempenho Cavalcanti verificou que um bom número de neurônios para este tipo de controle está na faixa de três a seis neurônios utilizando-se os parâmetros de treinamento ajustáveis propostos por Rangwala & Dornfeld (1989). A camada de saída utiliza um neurônio do tipo tangente hiperbólico devido a essa função ser a derivada da função que define o jacobiano do Motor CC no acionamento de um pêndulo.

Apêndice D

Treinamento da RNMC

Uma Rede Neural Artificial (RNA) é composta por neurônios artificiais que possuem características básicas como entradas (X_i), pesos (W_i), função soma (Σ), função de transferência ($f(y)$) e saída (Z), conforme descrito no Quadro D.1. Esses neurônios são interligados para forma a RNA, que pode ser de vários tipos (Multicamadas, recorrente etc.). O funcionamento da RNA na solução de um problema deve obedecer alguns critérios predefinidos de acordo com o tipo da rede em uso. Entretanto, toda RNA deve passar por algum tipo de treinamento definido como o processo de aprendizagem da rede.

Quadro D.1 – Características de um neurônio.



Função soma: $Y = \sum_{0}^{N} X_i W_i$, utilizada em todos os neurônios.

Neurônio linear.....: Função de transferênica: $Z = \sigma Y$, onde σ é a inclinação da reta.

Neurônio sigmóide.....: Função de transferênica: $Z = 1 / (1+e^{-Y})$.

Neurônio tangente hiperbólico.: Função de transferênica: $Z = (1-e^{-Y}) / (1+e^{-Y})$.

Considera-se o treinamento de uma Rede Neural Artificial Multicamadas (RNMC) composto pela propagação das entradas na rede e a propagação retroativa do erro na rede. A seguir apresenta-se o treinamento de uma RNMC com quatro entradas, onde cada entrada possui um valor predefinido, de tal forma que o processo de treinamento da mesma possa ser entendido com mais facilidade. Inicialmente são definidos valores aleatórios para os pesos da RNMC, ilustrada na Figura D.1. A representação dos pesos e das entradas, com os seus respectivos valores também são ilustrados na Figura D.1.

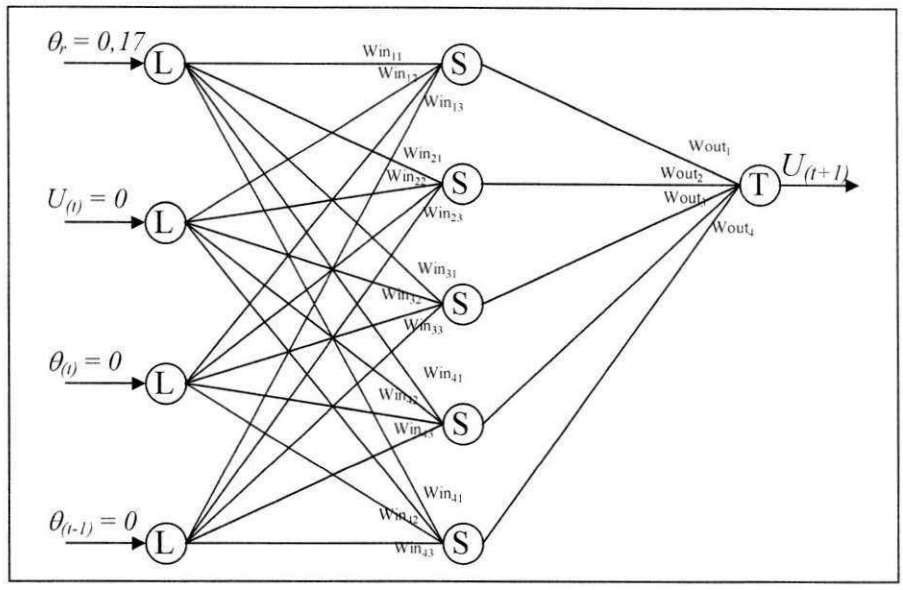


Figura D.1 – Valores iniciais das entradas e representação dos pesos.

As quatro entradas utilizadas para o controle do Motor CC são informadas à RNMC. θ_r com valor de referência fixo em 30° (30/180=0,17 em pu), $U_{(t)}$ com valor inicial de 0V, $\theta_{(t)}$ com valor inicial de 0° e $\theta_{(t-1)}$ com valor inicial de 0°. O objetivo é atingir os 30° do ângulo de referência (θ_r). A tensão e o ângulo inicial começam em zero (estado inicial do motor) e são modificados até que seja atingido o ângulo desejado. Os valores das entradas devem ser informados em pu, ou seja, no intervalo [-1, 1]. Isso é necessário devido às características dos neurônios em trabalharem com entradas e saídas nessa faixa de valores. O ângulo de 30° deve ser dividido por 180° para se obter o seu valor em pu (0,17) e assim pode-se informá-lo a RNMC. O valor 180° representa a rotação máxima que o Motor CC pode atingir, pois uma rotação completa de 360° é dividida em duas parte de 180° e –180°. A propagação das entradas na RNMC é feita pelo algoritmo de propagação das entradas descrito no Quadro D.2.

Inicialmente os valores das entradas são processados pelos neurônios lineares e transferidos para as entradas dos neurônios sigmóides na camada oculta. Nesta camada os valores oriundos dos neurônios lineares são processados juntos com os pesos das respectivas conexões e então os resultados desses processamentos são transferidos para a camada de saída. Nesta última camada, os valores que chegam também são processados juntos com os seus pesos e o resultado final é transformado em tensão a ser aplicada ao Motor CC.

O movimento efetuado pelo pêndulo sob o novo valor da tensão gerado pela rede é medido através do ângulo formado pelo pêndulo e a normal ao rotor do Motor CC. Essa medição é feita através de detectores de posição acoplados ao Motor CC. O erro é calculado en-

tre esse ângulo lido e o ângulo desejado. Caso esse erro seja igual a zero ou esteja dentro de uma faixa de valores aceitáveis, a rede é considerada treinada. Caso contrário, esse erro deve ser propagado na rede para que todos os pesos sejam ajustados, forçando a rede a apresentar um novo valor, mais próximo do desejado, na próxima tentativa de controle.

Quadro D.2 – Algoritmo de propagação das entradas na RNMC.

```
Faça i variar 1 até C;                    // Onde C representa o número total de camadas.
    Faça j variar de 1 até N;             // Onde N é a qtde de neurônios na camada i.
        Faça k variar de N-1;             // Onde N-1 é a qtde de neurônios na camada i-1.
            S[i,j] = S[i,j] + W[i,j,k] * T [i-1,k] ;  // Calcula o valor da função soma S.
        Fim do laço k;
        T [i,j] = f(S [i,j]) ;            // Calcular a função de transferência do neurônio j.
    Fim do laço j;
Fim do laço i;
```

Entretanto, o treinamento com o APR proposto por Rumelhart et al. (1986) apresenta uma curva de aprendizagem muito lenta para um controle em tempo real. Utilizando-se os parâmetros adaptáveis (Θ e β) propostos por Rangwala & Dornfeld (1989) é possível diminuir consideravelmente o tempo de aprendizagem bem como reduzir o número de neurônios na camada oculta (Cavalcanti, 1994, p.36). O algoritmo de propagação retroativa do erro é ilustrado no Quadro D.3. Este algoritmo é estruturado por três laços, sendo o primeiro controlado pela variável *i*, responsável pelo controle das camadas da rede – a rede possui um total de *C* camadas. O segundo laço, controlado pela variável *j*, é responsável pelo controle dos neurônios de cada camada – uma cada pode ter até *N* neurônios. Por último, o terceiro laço, controlado pela variável *k*, é responsável pelo controle dos pesos de cada neurônio – um neurônio pode ter até *N-1* pesos. O parâmetro "*N-1*" representa a quantidade de neurônios da camada anterior, ou seja, um neurônio terá três pesos se a camada anterior tiver três neurônios.

A condição com "$i == C$" é necessária para o cálculo do erro da última camada que difere do cálculo do erro das demais camadas. Na última camada o erro é calculado através da diferença entre a saída desejada e a saída obtida multiplicada pelos parâmetros de treinamento ajustáveis, conforme descrito na Equação D.1. Nas demais camadas o erro é calculado através do produto entre o erro anterior e o peso também multiplicado pelos parâmetros de treinamento ajustáveis, conforme descrito na Equação D.2

$$Z^* * \left(1 - \frac{Z}{\Theta}\right) * \left(Z^* - Z\right) * \beta \tag{D.1}$$

$$Z^* * \left(1 - \frac{Z}{\Theta}\right) * \left(W * \varepsilon\right) * \beta \tag{D.2}$$

O ajuste de cada peso é feito somando-se ao peso a variação do erro descrito na Equação D.3. Essa variação é calculada através do produto entre o erro anterior, o valor da entrada e a constante de aprendizagem ou fator de convergência, conforme descrito na Equação D.4. Essa constante de aprendizagem é definida a *priori* com um valor dentro do intervalo fechado de 0 a 1. O melhor valor para essa constante normalmente é definido após vários testes de desempenho de acordo com a planta sob controle.

$$W = W + \delta \tag{D.3}$$

$$\delta = \varepsilon * T * \rho \tag{D.4}$$

Quadro D.3 – Algoritmo de propagação retroativa do erro.

```
Faça i variar C até 1            // Onde C representa o número total de camadas.
   Faça j variar de 1 até N      // Onde N é a qtde de neurônios na camada i.
      Se (i==C)    // Necessário para diferenciar o cálculo do erro na última camada.
         ε[k,j] = Z[k,j] * (1-Z[k,j]/Θ[i]) * (Z*[k,j] - Z[k,j]) * β[i]; // Onde Z* é a saída desejada, Z a saída obtida e
                                                                        // Θ e β são os parâmetros ajustáveis.
      Senão
         ε[i-1,j] = Z[k,j] * (1- Z[k,j]/Θ[i]) * W[ij] * ε[i,j] * β[i];    // Propagação do erro da camada anterior.
      Fim do se;
      Faça k variar de 1 até N-1    // Onde N-1 é a qtde de neurônios na camada i-1.
         δ[i,j] = ε[i,j] * T[i,j] * ρ;    // Calcular a variação δ do erro, onde ρ é a constante de aprendizagem.
         W[i,j,k] = W[i,j,k] + δ[i,j] :    // Ajusta o peso W do neurônio j da camada i.
      Fim do laço k;
      β[i] = ρ * δ[i,j] * (S[i,j] + t[i] ) / β[i];    // Calcula o novo valor de β (treinamento do parâmetro ajustável).
      β[i] = (β[i] < 1 ? 1 : (β[i] > 3 ? 3 : β[i]));    // Força β a ficar no intervalo de 1 a 3.
                                                        // A operação (sentença?decisão1:decisão2) simula o se/então/senão.
      Θ[i] = 0.1 * (ρ * δ[i,j] * T[i,j]) / Θ[i];    // Calcula o novo valor de Θ (treinamento do parâmetro ajustável).
      Θ[i] = (Θ[i] < 1 ? 1 : (Θ [i] > 3 ? 3 : Θ[i]));    // Força Θ a ficar no intervalo de 1 a 3.
      t[i] = ρ * δ[i,j];    // Calcula o novo valor de t.
      t[i] = (t[i] < -1 ? -1 : (t[i] > 1 ? 1 : t[i]));    // Força t a ficar no intervalo de -1 a 1.
   Fim do laço j;
```

Apêndice E

Algoritmos das Tarefas do ETR

O conteúdo de cada tarefa utilizada na transferência de carga entre os dois manipuladores robóticos é listada a seguir. Algumas tarefas aparecem com todos os parâmetros dos descritores zerados. Isso significa que a tarefa está sendo desativada, ou seja, ela não será mais escalonada. Tomando como exemplo a tarefa 2 com parâmetros *(2,0,0,0)*, significa que o motor 2 está sendo desligado, pois cada motor é acionado através de pulsos enviados a cada 1 ms.

- Tarefa 1 : Executa a transferência do manipulador 1 para o manipulador 2.

```
Meta1()
{
    tarefa(2,1,1,1);              //Aciona o motor 2 para entregar a carga.
    tarefa(4,0,1,20);       // Treinamento inicial dos motores 1 e 3.
    tarefa(7,0,500,20);     // Aciona o motor 1 (sob controle) para entregar a carga.
    tarefa(8,0,1000,20);    // Posiciona Motor 3 (sob controle) para receber a carga.
    tarefa(3,0,1500,1);     // Espera 1,5s e posiciona Motor 4 para receber a carga.
    tarefa(5,0,2000,0);     // Doa a carga ao Manipulador 2 (desliga o motor 1 e 2).
    tarefa(6,0,4000,0);     // recebe a carga (desliga motores 3 e 4).
}
```

- Tarefa 2 : Aciona o motor 2.

```
Motor2()
{   se (direita)            //Variável de controle do sentido da rotação.
        roda[2]=1;          //Motor 2 acionado para a direita (sentido anti-horário).
    senão
        roda[2]=2;          //Motor 2 acionado para a esquerda (sentido horário).
}
```

- Tarefa 3 : Aciona o motor 4.

```
Motor4()
{
    se (direita)            //Variável de controle do sentido da rotação.
        roda[4]=1;          //Motor 4 acionado para a direita (sentido anti-horário).
    senão
        roda[4]=2;          //Motor 4 acionado para a esquerda (sentido horário).
}
```

- Tarefa 4 : Controla a velocidade dos motores 1 e 3 através do SCI[15] .

```
Velocidade()
{
    contador1 = ler_posição(1);          //Faz a leitura do detector de posição do motor 1.
    contador3 = ler_posição(3);          //Faz a leitura do detector de posição do motor 3.
    deslocamento1=contador1/4;           //Calcula o deslocamento do motor 1.
    deslocamento3=contador3/4;           //Calcula o deslocamento do motor 3.
    Executa_RNMC1;              //Rede neural do controlador do motor 1.
    Executa_RNMC3;              //Rede neural do controlador do motor 3.
}
```

- Tarefa 5 : Tarefa para desligar os motores 1 e 2.

```
Doa1()
{
    roda[1]=0;    // Variável de controle do sentido de rotação do motor 1 (0 == parar).
    roda[2]=0;    // Variável de controle do sentido de rotação do motor 2 (0 == parar).
    repete(7,0,0,0);        //Desativa a tarefa de acionamento do motor 1.
    repete(2,0,0,0);        //Desativa a tarefa de acionamento do motor 2.
}
```

---

[15] Sistema de Controle Inteligente definido no Capítulo 4.

- Tarefa 6 : Tarefa para desligar os motores 3 e 4.

```
Doa2()
{
    roda[3]=0;      // Variável de controle do sentido de rotação do motor 3 (0 == parar).
    roda[4]=0;      // Variável de controle do sentido de rotação do motor 4 (0 == parar).
    repete(8,0,0,0);          //Desativa a tarefa de acionamento do motor 3.
    repete(3,0,0,0);          //Desativa a tarefa de acionamento do motor 4.
}
```

- Tarefa 7 : Tarefa para o treinamento do motor 1.

```
Treinar1()
{
    se (deslocamento1 != saida_desejada1) //Verifica se o ângulo desejado foi obtido.
        Executa_RNMC1;          //Rede neural do controlador do motor 1.
        PWM(1);                 //Calcula o novo trem de pulso para o motor 1.
}
```

- Tarefa 8 : Tarefa para o treinamento do motor 3.

```
Treinar3()
{
    se (deslocamento3 != saida_desejada3) //Verifica se o ângulo desejado foi obtido.
        Executa_RNMC3;          //Rede neural do controlador do motor 3.
        PWM(3);                 //Calcula o novo trem de pulso para o motor 3.
    }
}
```

- Tarefa 9 : Tarefa para verificar se há carga nos manipuladores.

```
Ver-o-peso()        //Homenagem a minha querida Belém do Pará!
{
    ha_carga1 = Fuzzyfica (θ1);  // Verifica se o manipulador 1 possui a carga.
    ha_carga2 = Fuzzyfica (θ2);  // Verifica se o manipulador 3 possui a carga.
    tarefa(2,0,0,0);             // Desliga o motor 2.
    tarefa(3,0,0,0);             // Desliga o motor 4.
    Se (ha_carga1 == 1)
        tarefa (1,1,0,0);   //Ativa a transferência do manipulador 1 para o 2.
    Senão
        Se (ha_carga2 == 1)
            tarefa (10,1,0,0); // Ativa a transferência do manipulador 2 para o 1.
        Fimse
    Fimse
}
```

- Tarefa 10 : Executa a transferência do manipulador 2 para o manipulador 1.

```
Meta2()
{
    tarefa(3,1,1,1);              //Aciona o motor 4 para entregar a carga.
    tarefa(4,0,1,20);       // Treinamento inicial dos motores 1 e 3.
    tarefa(8,0,500,20);     // Aciona o motor 3 (sob controle) para entregar a carga.
    tarefa(7,0,1000,20);    // Posiciona Motor 1 (sob controle) para receber a carga.
    tarefa(2,0,2000,1);     // Espera 2s e posiciona Motor 2 para receber a carga.
    tarefa(6,0,3000,0);     // Doa a carga ao Manipulador 1 (desliga o motor 3 e 4).
    tarefa(5,0,4000,0);     // recebe a carga (desliga motores 1 e 2).
}
```

Apêndice F

O sistema de transferência de carga

A implementação do Escalonador Inteligente de Tarefas foi baseada na estrutura robótica apresentada no Capítulo 4 e foi desenvolvido no Borland C++ Builder. Através desse sistema tem-se o controle total de parâmetros do SCI e das variáveis de configuração do ambiente. O sistema é composto basicamente por uma janela principal de onde é possível visualizar e controlar todo o sistema. Essa janela funciona como um painel de controle. As outras duas janelas são utilizadas para alterar e consultar os parâmetros da RNMC do controlador e para visualizar os valores dos pesos e índices da RNMC, respectivamente.

O sistema inicia com uma janela, normalmente rotulada como "sobre", devido ela conter as informações técnicas sobre o sistema e ser acessada através da opção sobre a partir do "*menu*" principal do sistema. Essa janela é ilustrada na Figura F.1. Ela fica ativa por 10 s e depois é apresentada a janela principal do sistema. O usuário pode fechá-la antes dos 10 s pressionando o botão "OK".

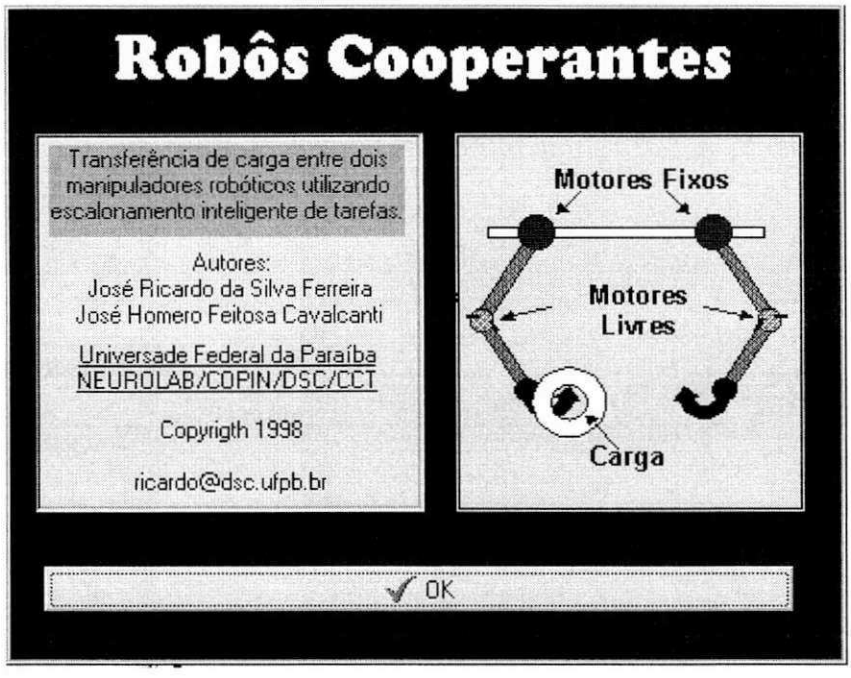


Figura F.1 – Janela "sobre" do sistema.

A janela principal, ilustrada na Figura F.2, é dividida em 5 partes. A primeira parte ("Teste dos motores") é utilizada para testar o funcionamento dos 4 motores, apresentando um botão de acionamento para cada sentido de rotação de cada motor e um para parar cada motor. Nesta parte é possível também definir o endereço da porta paralela a ser utilizada ("Porta Valor") e testar os 2 detectores de posição. Para isso, basta pressionar o botão "Teste" ("Detector 1" ou "Detector 2"). O campo Valor ao lado da caixa de alteração do endereço da porta paralela é utilizado para apresentar o valor atual do sinal enviado para esta porta e conseqüentemente para o acionamento dos motores.

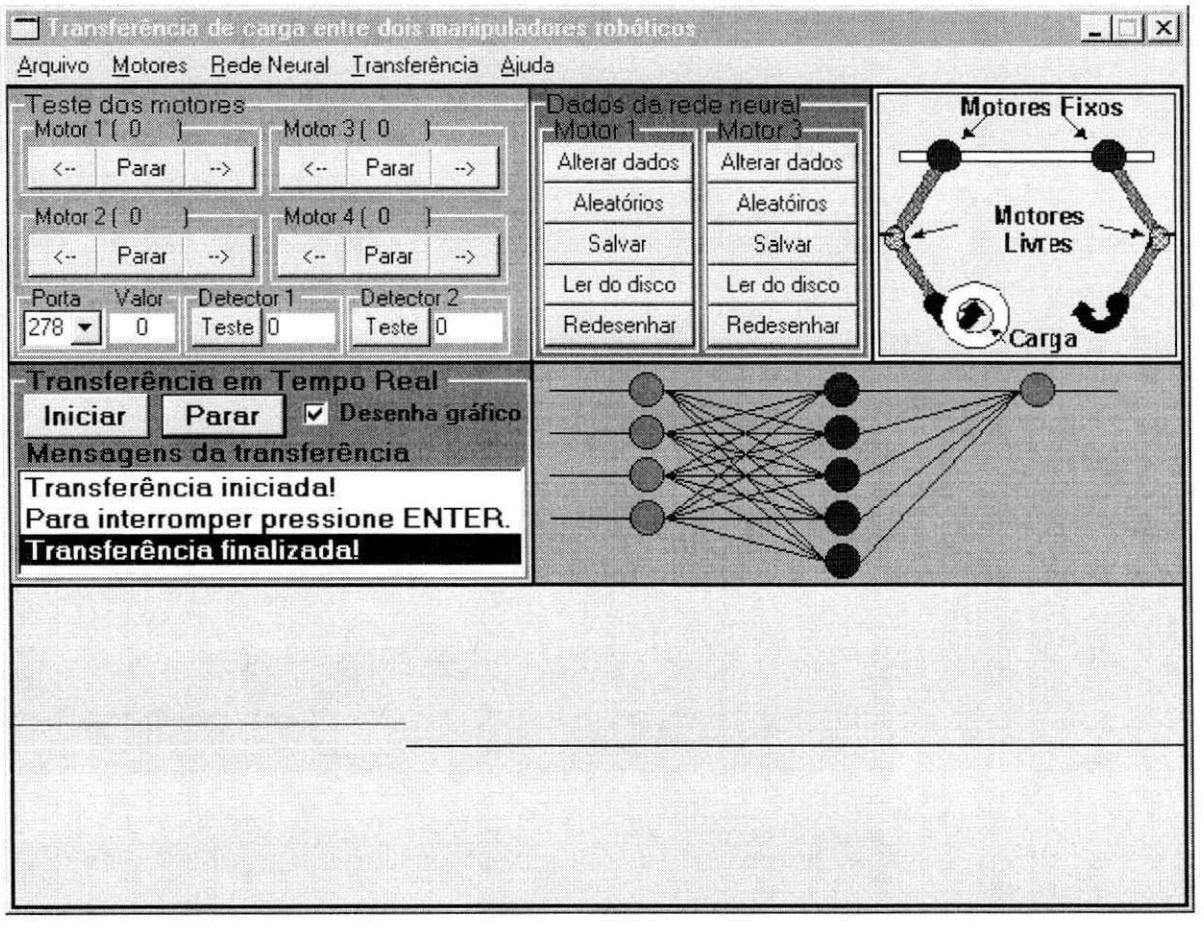


Figura F.2 – Janela principal do sistema.

A segunda parte, rotulada como "Transferência em Tempo Real", é responsável pelo controle da transferência de carga. Ela possui 2 botões para iniciar e para a transferência e um campo "Desenha gráfico" que permite controlar se o gráfico de desempenho (quinta parte) será ou não desenhado na tela durante o processo de transferência. Quando o botão Iniciar é pressionado, uma janela de diálogo se abre para confirmação do início da transferência, conforme ilustrado na Figura F.3. Aí está localizada também a caixa de mensagens do sistema

("Mensagens da transferência") onde são impressas todas as mensagens geradas pelo sistema, conforme ilustrada na Figura F.3.

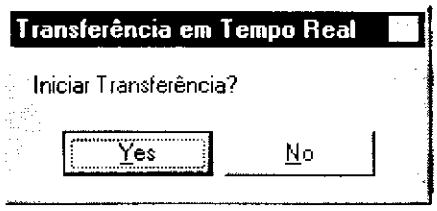


Figura F.3 – Janela de diálogo para confirmação da transferência.

A terceira parte apresenta as opções de controle das RNMC dos motores 1 e 3 e um gráfico ilustrativo do processo de transferência. O controle da RNMC ("Dados da Rede Neural") é feito através dos 5 botões presentes no campo para cada motor ("Motor 1" ou "Motor 3"), rotulados como: Alterar dados, Aleatório, Salvar, Ler do Disco e Redesenhar. O botão "Alterar dados" dá acesso a segunda janela do sistema, onde todos os dados da RNMC estão dispostos para consulta e alteração. Os outros quatro botões executam tarefas internas onde a única interação com o usuário é uma janela de diálogo solicitando confirmação da operação, conforme ilustrado na Figura F.4.

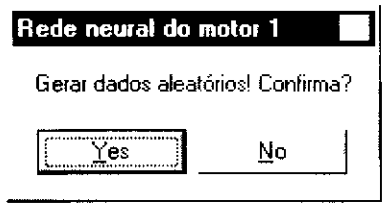


Figura F.4 – Janela de diálogo para confirmação de dados aleatórios.

A quarta parte da janela principal é utilizada para ilustrar graficamente a característica da RNMC utilizada no controlador neural. Qualquer alteração feita na estrutura da RNMC através do botão "Alterar dados" é refletida automaticamente nessa ilustração.

A quinta e última parte também é utilizada com finalidade ilustrativa, neste caso para permitir que o desempenho de cada manipulador seja acompanhado graficamente na tela, durante o processo de transferência. Esse processo consome muito tempo de processamento, sendo que em alguns testes de transferência foi necessário desativá-lo através do campo de seleção apresentado no estudo da segunda parte da janela principal.

A segunda janela do sistema permite total controle sobre os parâmetro da RNMC do controlador neural, conforme ilustrado na Figura F.5. Os campos que podem ser alterados

nesta janela são: "Qtde camadas", que define quantas camadas a RNMC pode ter (de 2 a 4 camadas); "Fator de convergência", que define o grau de adaptabilidade; "Qtde de Neurônios", que define a quantidade de neurônios em cada camada da rede. Os campos estarão disponíveis de acordo com o a quantidade de camadas defina anteriormente; "Função de Ativação", que define o tipo de função de ativação para os neurônios de cada camada; "Valores", em Planta/Referência, no qual deve ser informado o valor de referência a ser seguido pela planta.

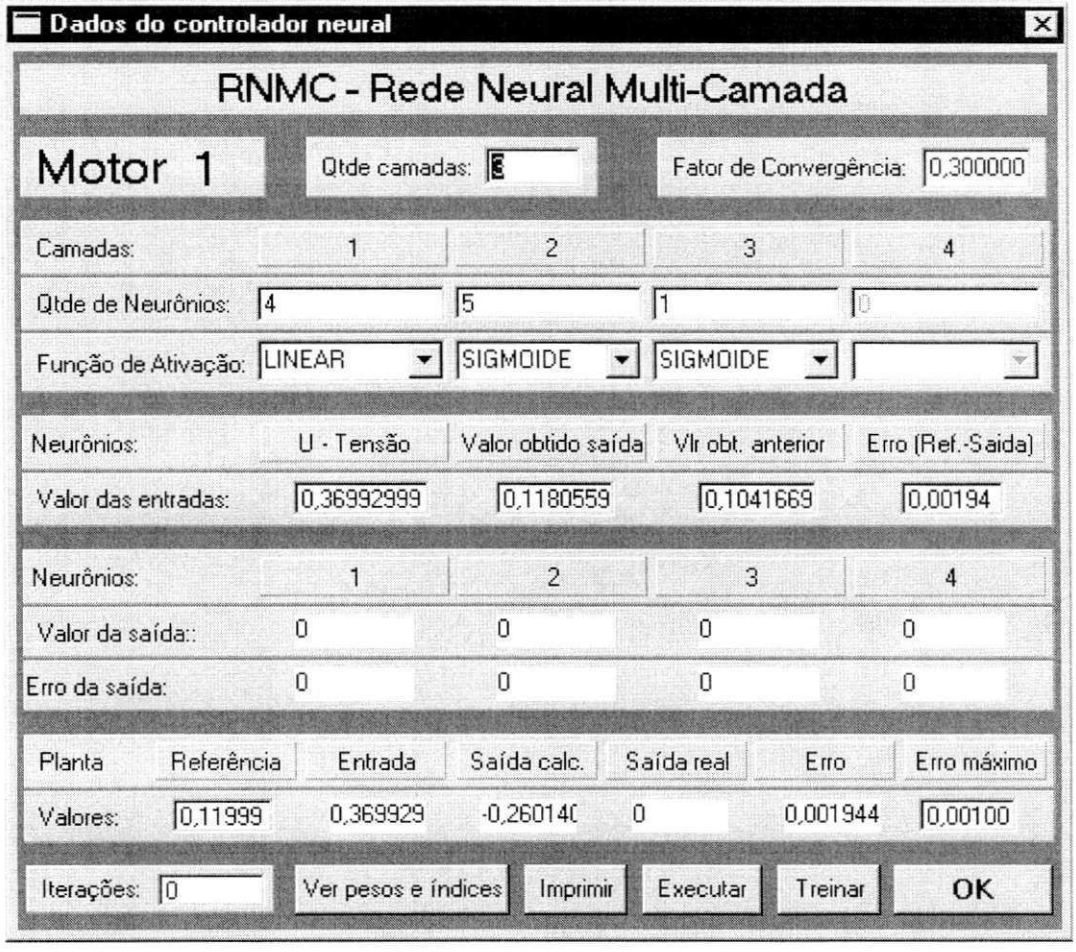


Figura F.5 – Janela de visualização e alteração dos dados da RNMC.

Os outros campos descriminados a seguir são todos para consulta, não sendo permitida a alteração dos mesmos. "Valor das entradas" apresenta o valor atual de cada entrada da rede; "Valor da saída" apresenta o valor atual de cada saída da rede; "Erro da saída" apresenta o valor atual do erro encontrado em cada saída da rede; "Valores", em Planta/Entrada apresenta o valor da entrada da planta; em Planta/Saída Calc. apresenta o valor calculado pelo controlador para a saída da planta; em Planta/Saída real apresenta o valor real obtido na saída da planta; em Planta/Erro apresenta o erro real obtido na saída da planta; em Planta/Erro máximo

apresenta o valor de tolerância para o erro da planta; "Iteração" apresenta o número de iterações que já foram realizadas na rede.

Os botões presentes no rodapé da janela ilustrada na Figura F.5 permitem mais controles sobre a RNMC. O botão "Ver pesos e índices" apresenta uma nova janela com as informações dos pesos e índices ajustáveis da rede; o botão "Imprimir" permite a impressão das informações da rede (esta rotina não foi implementada); o botão "Executar" permite que o algoritmo de propagação das entradas na rede seja executado; o botão "Treinar" permite que o algoritmo de propagação retroativa do erro na rede seja executado; finalizando, o botão "OK" fecha a janela e retorna o controle para a janela principal.

A terceira e última janela do sistema permite que sejam visualizados os valores dos pesos de todos os neurônios da rede e dos índices de treinamento (Θ, β e t), conforme ilustrado na Figura F.6. O erro em cada neurônio também é visualizado. O botão "Imprimir" também não foi implementado nesta janela. O botão "OK" fecha a janela e retorna o controle para a janela de alteração e visualização dos dados da RNMC.

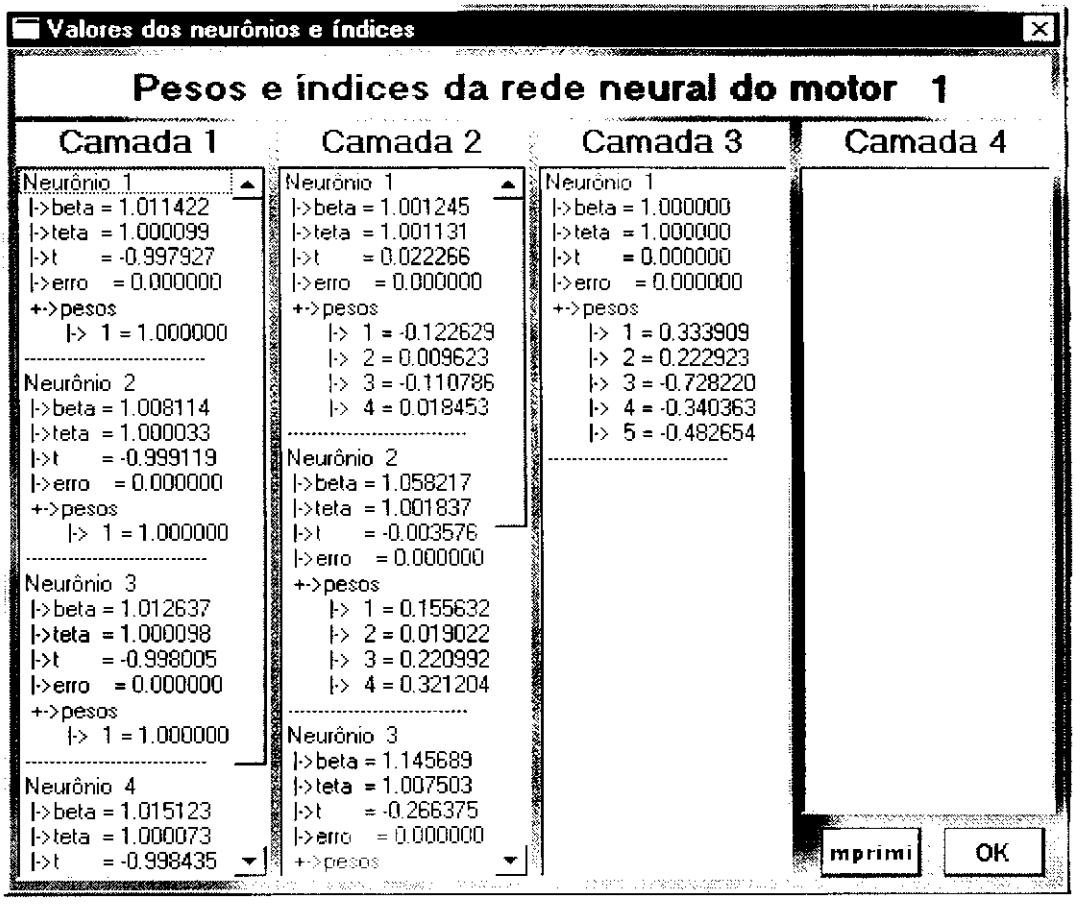


Figura F.6 – Janela de visualização dos pesos e índices da RNMC.

Apêndice G

Algoritmos das Tarefas do EIT

O conteúdo de cada tarefa utilizada na transferência de carga entre os dois manipuladores robóticos com o Escalonador Inteligente de Tarefa (EIT) é listada a seguir.

- Tarefa 1 : Executa a transferência do manipulador 1 para o manipulador 2.

```
Meta1()
{   tarefa (12,0,1,1,1,PM); //Aciona o motor 2 e avalia a regra 5.3 de 1 em 1 ms.
    tarefa (4,0,1,20,0,0); //Treinamento inicial dos motores 1 e 3 de 20 em 20 ms.
    tarefa (7,0,0,20,1,PM); //Tarefa criada mas com o estado desativada, sua
    // finalidade é acionar o motor 1 sob controle e avaliar a regra 5.4 de 20 em 20 ms.
}
```

- Tarefa 2 : Aciona o motor 2.

```
Motor2()
{   se (direita)            //Variável de controle do sentido da rotação.
        roda[2]=1;          //Motor 2 acionado para a direita (sentido anti-horário).
    senão
        roda[2]=2;          //Motor 2 acionado para a esquerda (sentido horário).
}
```

- Tarefa 3 : Aciona o motor 4.

```
Motor4()
{   se (direita)            //Variável de controle do sentido da rotação.
        roda[4]=1;          //Motor 4 acionado para a direita (sentido anti-horário).
    senão
        roda[4]=2;          //Motor 4 acionado para a esquerda (sentido horário).
}
```

- Tarefa 4 : Controla a velocidade dos motores 1 e 3 através do SCI[16] .

```
Velocidade()
{  contador1 = ler_posição(1);          //Faz a leitura do detector de posição do motor 1.
   contador3 = ler_posição(3);          //Faz a leitura do detector de posição do motor 3.
   deslocamento1=contador1/4;           //Calcula o deslocamento do motor 1.
   deslocamento3=contador3/4;           //Calcula o deslocamento do motor 3.
   Executa_RNMC1;              //Rede neural do controlador do motor 1.
   Executa_RNMC3;              //Rede neural do controlador do motor 3.
}
```

- Tarefa 7 : Tarefa para o treinamento do motor 1.

```
Treinar1()
{  se (deslocamento1 != saida_desejada1) //Verifica se o ângulo desejado foi obtido.
       Executa_RNMC1;          //Rede neural do controlador do motor 1.
        PWM(1);          //Calcula o novo trem de pulso para o motor 1.
}
```

- Tarefa 8 : Tarefa para o treinamento do motor 3.

```
Treinar3()
{  se (deslocamento3 != saida_desejada3) //Verifica se o ângulo desejado foi obtido.
       Executa_RNMC3;          //Rede neural do controlador do motor 3.
        PWM(3);          //Calcula o novo trem de pulso para o motor 3.
   }
}
```

- Tarefa 9 : Tarefa para verificar encerrar a estratégia *Pesa*.

```
Finalisa_Pesa()
{  tarefa(2,0,0,0);                  // Desliga o motor 2.
   tarefa(3,0,0,0);                  // Desliga o motor 4.
   tarefa(11,1,0,0);          // Ativa a tarefa 11 para encerrar a transferência.
 }
```

---

[16] Sistema de Controle Inteligente definido no Capítulo 4.

- Tarefa 10 : Executa a transferência do manipulador 2 para o manipulador 1.

```
Meta2()
{
   tarefa (13,0,1,1,3,NM); //Aciona o motor 4 e avalia a regra 5.5 de 1 em 1 ms.
   tarefa (4,0,1,20,0,0);  //Treinamento inicial dos motores 1 e 3 de 20 em 20 ms.
   tarefa (8,0,0,20,3,NM); //Tarefa criada mas com o estado desativada
                                    //sua finalidade é acionar o motor 3 sob controle
                                   // e avaliar a regra 5.6 de 20 em 20 ms.
}
```

- Tarefa 12 : Aciona o motor 2.

```
Motor2()
{  se (direita)          //Variável de controle do sentido da rotação.
       roda[2]=1;        //Motor 2 acionado para a direita (sentido anti-horário).
   senão
      roda[2]=2;         //Motor 2 acionado para a esquerda (sentido horário).
}
```

- Tarefa 13 : Aciona o motor 4.

```
Motor4()
{  se (direita)          //Variável de controle do sentido da rotação.
       roda[4]=1;        //Motor 4 acionado para a direita (sentido anti-horário).
   senão
      roda[4]=2;         //Motor 4 acionado para a esquerda (sentido horário).
}
```

- Tarefa 14 : Aciona o motor 2.

```
Motor2()
{  se (direita)          //Variável de controle do sentido da rotação.
       roda[2]=1;        //Motor 2 acionado para a direita (sentido anti-horário).
   senão
      roda[2]=2;         //Motor 2 acionado para a esquerda (sentido horário).
}
```

- Tarefa 15 : Aciona o motor 4.

```
Motor4()
{
    se (direita)            //Variável de controle do sentido da rotação.
        roda[4]=1;          //Motor 4 acionado para a direita (sentido anti-horário).
    senão
        roda[4]=2;          //Motor 4 acionado para a esquerda (sentido horário).
}
```

- Tarefa 17 : Tarefa para o treinamento do motor 1.

```
Treinar1()
{
    se (deslocamento1 != saida_desejada1) //Verifica se o ângulo desejado foi obtido.
        Executa_RNMC1;          //Rede neural do controlador do motor 1.
        PWM(1);                 //Calcula o novo trem de pulso para o motor 1.
}
```

- Tarefa 18 : Tarefa para o treinamento do motor 3.

```
Treinar3()
{
    se (deslocamento3 != saida_desejada3) //Verifica se o ângulo desejado foi obtido.
        Executa_RNMC3;          //Rede neural do controlador do motor 3.
        PWM(3);                 //Calcula o novo trem de pulso para o motor 3.
    }
}
```