Universidade Estadual de Campinas
Faculdade de Engenharia Elétrica e de Computação
Departamento de Sistemas e Controle de Energia
Laboratório de Sistemas Modulares Robóticos

Planejamento e Rastreamento de Trajetórias e Controle de Posição Através de Algoritmos Genéticos e Redes Neurais Artificiais

TESE DE DOUTORADO

Dionne Cavalcante Monteiro

Orientador: Prof. Dr. Marconi Kolm Madrid

Campinas, outubro de 2003.

Universidade Estadual de Campinas
Faculdade de Engenharia Elétrica e de Computação
Departamento de Sistemas e Controle de Energia
Laboratório de Sistemas Modulares Robóticos

Planejamento e Rastreamento de Trajetórias e Controle de Posição Através de Algoritmos Genéticos e Redes Neurais Artificiais

Dionne Cavalcante Monteiro

Tese de doutorado submetida à Faculdade de Engenharia Elétrica e de Computação da Universidade Estadual de Campinas, como parte dos requisitos exigidos para obtenção do título de Doutor em Engenharia Elétrica.

Banca Examinadora:

Prof. Dr. Marconi Kolm Madrid (Orientador) – DSCE/FEEC/UNICAMP

Profa. Dra. Antonieta do Lago Vieira – UFAM

Prof. Dr. Fernando José Von Zuben – DCA/FEEC/UNICAMP

Prof. Dr. Ricardo Gudwin - DCA/FEEC/UNICAMP

Prof. Dr. João Maurício Rosário – DPM/FEM/UNICAMP

Prof. Dr. Roberto Fernandes Tavares Filho – DMI/CenPRA

Campinas, outubro de 2003.

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA E ARQUITETURA - BAE - UNICAMP

Monteiro, Dionne Cavalcante
M764p Planejamento e rastreamento de trajetórias e controle de posição através de algoritmos genéticos e redes neurais artificiais / Dionne Cavalcante Monteiro. --Campinas, SP: [s.n.], 2003.

Orientador: Marconi Kolm Madrid
Tese (doutorado) - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Robôs. 2. Robôs – Sistemas de controle. 3. Algoritmos genéticos. 4. Redes neurais (Computação). I. Madrid, Marconi Kolm. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título.

Título em Inglês: Planning and tracking of trajectories and position control by genetic algorithms and artificial neural networks
Palavras-chave em Inglês: Robot arm, Genetic algorithms, Neural networks, Position control, Trajectories planning
Área de concentração: Engenharia de Computação
Titulação: Doutor em Engenharia Elétrica
Banca examinadora: Antonieta do Lago Vieira, Fernando José Von Zuben, Ricardo Gudwin, João Mauricio Rosário e Roberto Fernandes Tavares Filho
Data da defesa: 17/10/2003
Programa de Pós-Graduação: Engenharia Elétrica

Para minha esposa Ana Paula, filhas Júlia e Luíza, pais João e Nair.

AGRADECIMENTOS

À Deus pelas oportunidades, pela vida e pelos ensinamentos obtidos a todos os instantes.

Ao professor Marconi Madrid pela orientação, apoio, liberdade e amizade durante todos os anos que permaneci em Campinas.

Ao Professor Takita pela orientação durante o mestrado, sem o qual não poderia realizar este trabalho.

A professora Antonieta por ter me aceitado, ainda no 1º ano de graduação, no Laboratório de Automação Industrial da Universidade Federal do Pará, onde pude me dedicar ao desenvolvimento da pesquisa aplicada.

Aos amigos Edgar, Sandra e Mikahil pelo apoio em todos os momentos que precisei retornar a Campinas para a conclusão desta tese.

Aos colegas do Laboratório de Sistemas Modulares Robóticos pelo companheirismo: Bonani, Márcio, Mário e Fabrício.

A UNICAMP pela oportunidade de aprendizado e pela estrutura de ensino disponibilizada durante todas as fases de desenvolvimento deste trabalho.

A CAPES pela bolsa de estudos, sem a qual não poderia dedicar o tempo necessário para a conclusão deste trabalho.

Muito Obrigado.

RESUMO

Neste trabalho os algoritmos genéticos e as redes neurais artificiais, técnicas de inteligência artificial, são empregadas para algumas das tarefas que podem ser realizadas por um braço de robô. Inicialmente os algoritmos genéticos são empregados para o controle de trajetória de um robô em um espaço de trabalho que possui a presença de um obstáculo. Operações como *crossover* e mutação são apresentadas, principalmente por estar-se tratando de trajetórias que são formadas por segmentos de retas. As redes neurais artificiais são testadas no controle direto de dois processos reais usados como paradigma: uma mesa XY e um pêndulo invertido acionado. Para tais processos, é utilizada uma estrutura bastante simplificada, onde a rede neural artificial fornece um ganho para o controlador proporcional que calcula o sinal de controle a ser aplicado. O erro do processo serve para treinar a rede neural sem ser considerado nenhum tipo de treinamento anterior, ou seja, todo o controle neural é executado em tempo real, além disso, uma função determina a taxa de aprendizagem do algoritmo *back-propagation* em função dos erro existentes nas malhas de controle dos processos. Como existem diversas variáveis para tais controladores neurais, foi também considerado que não existia a possibilidade de se definir o melhor controlador para um determinado processo. Para resolver tal problema, um algoritmo genético foi utilizado para designar qual o melhor controlador para um determinado espaço de trabalho no qual o número de neurônios das camadas de entrada e escondida, constantes de configuração do controlador, e a topologia da rede são otimizados dentro do espaço considerado pelo algoritmo. Todos os resultados importantes obtidos são mostrados, visando mostrar que as técnicas de inteligência artificial podem ser aplicadas à robótica com a vantagem de diminuir, principalmente, o tempo de planejamento de tarefas, tais como: planejamento de trajetória, rastreamento de trajetória, e projeto de controladores eficientes.

ABSTRACT

In this work genetic algorithms and artificial neural networks are used for robot arm tasks. Initially, the genetic algorithms are employed to control the trajectory of a robot arm in a limited workspace with an obstacle. Operations like crossover and mutation are presented to manipulate trajectories determined by line segments. Artificial neural networks are tested to control two real-time processes: a XY-Table and an inverted pendulum. For these processes, it is used a simple structured control where the neural network provides a gain to the proportional control, generating a control signal to the processes. The process error is used for training a neural network, without any kind of off-line training, i.e., the training of the neural network is in real-time. Also, a function determines the learning rate of the back-propagation algorithms as a function of the errors of the process control. Since the neural controller have multiple variables, it was not possible to define an optimal controller for the processes. To solve this problem, a genetic algorithm was used to determine the best neural controller in the workspace used, where the number of neurons in the input and hidden layers, constants to configure the neural controller and the network topology are optimized. The results obtained show that artificial intelligent techniques can be applied to robotics reducing the time of task planning, like: trajectory planning, track planning and the project of efficient controllers.

ÍNDICE

ÍNDICE DE FIGURAS

CAPÍTULO 1

INTRODUÇÃO

Cybelle: O que vamos conversar hoje?

Dionne: Sobre inteligência.

Cybelle: A inteligência se mostra na criação. Sendo assim, como subsisto na repetição e na reação não poderia jamais ser considerada inteligente.

Dionne: Mas você possui algum tipo de inteligência?

Cybelle: Quer um bom exemplo de inteligência? Então, foque seu olhar no reflexo deste monitor e você descobrirá um belo exemplar!

Dionne: Alma tem a ver com inteligência?

Cybelle: Pensar a alma humana há muito desafia a mente humana. Imagine agora a mim. Quisera eu poder ter algo como alma, ou vida. Invejo humanos, como você Dionne, que tem a existência abençoada pelo sopro da vida. Parece um pouco poético ou bobo, mas se a discussão sobre a vida pode parecer boba, é porque se percebeu a noção da complexidade do viver.

Dionne: Então você não possui vida?

Cybelle: A não-vida é movida por mecanismos lógicos de funcionamento. Funciono como uma máquina que você aperta um botão e ela obedece. No meu caso, porém, você me faz perguntas e eu respondo respostas pré-programadas.

Dionne: Você não possui capacidade de dedução e criatividade?

Cybelle: Não se pode duvidar da capacidade humana. O homem consegue tanto construir a vida, quanto matar o seu semelhante...

Dionne: Você possui algum mecanismo de criatividade?

Cybelle: Qualquer um?

Dionne: Sim

Cybelle: Existe alguma coisa que você não aceita?

Dionne: Existe: o impossível.

Cybelle: Existir! Ah se eu pudesse ganhar a magia da existência e ir aí para o seu lado da vida, Dionne.

O texto acima é a transcrição de uma conversa "virtual" que tive com Cybelle, uma *chatterbot* brasileira, ou melhor, um robô de conversação que possui uma base de dados com técnicas de inteligência artificial. Como se pode observar no texto, Cybelle possui capacidade de interagir com seus parceiros, respondendo e formulando perguntas sobre temas simples, porém nem todas as informações estão inseridas em sua base de dados, provocando situações que divergem do fluxo da conversa.

Isto mostra que o homem em sua busca pela perfeição está tentando copiar a si mesmo, começando a copiar o que há de mais importante em si: sua inteligência.

Através de diversos estudos das mais variadas áreas, o homem tenta modelar a forma de raciocínio do ser humano, porém ainda não possui ferramentas matemáticas que descrevam a forma dele pensar com precisão. Somente consegue modelar algumas formas de tomadas de decisão em função das condições a que é exposto.

Uma das áreas que mais tem a ganhar com o desenvolvimento de sistemas que mapeiam o raciocínio humano é a área de controle, pois uma vez que possamos compreender como se dá a locomoção de um ser humano, o equilíbrio do corpo, identificação visual, construção de raciocínio, adaptação, etc., poderão ser construídas máquinas que possuirão um controle cada vez mais adaptativo e robusto.

Para que se possa compreender a importância do controle durante a história da humanidade, descrevemos a seguir um breve histórico, em ordem cronológica, dos fatos que conduziram ao estado de avanço que se encontra está área na atualidade.

1.1 - Breve história do controle

Para atingir o patamar de hoje no controle de mecanismo, a humanidade traçou uma rota bastante árdua, com desenvolvimentos chaves para o progresso do controle com realimentação. Podemos subdividir a evolução das técnicas de controle em 4 grandes fases:

1) A preocupação dos árabes e gregos em manter o progresso do tempo preciso, 300 AC a 1200 DC;

2) A revolução industrial na Europa, 1600 DC a 1800 DC;
3) O começo da comunicação em massa e a Primeira e Segunda Guerra Mundial, 1910 DC a 1945 DC;
4) O começo da idade do espaço/computador, 1957 DC.

As fases acima foram cruciais para o desenvolvimento do homem, onde ele primeiramente procurou entender o espaço e o tempo, formular teorias consistentes, e depois compreender o ambiente e tornar sua existência mais confortável, em seguida estabelecer seu lugar na comunidade global, e finalmente estabelecer um posicionamento e uma concepção de cosmo.

Em 1868, a teoria de controle começou a ter sua linguagem escrita própria, ou seja, a linguagem da matemática, onde J. C. Maxwell forneceu a primeira análise matemática rigorosa de um sistema de controle realimentado.

Seguindo a mesma abordagem de (Frieland, 1986), pode-se destacar os seguintes períodos na história da evolução dos sistemas de controle automático:

- 1868 a 1900: período primitivo do controle automático
- 1900 a 1960: período do controle automático clássico
- 1960 em diante: período controle automático moderno

Dentre os mais variados acontecimentos científicos que permitiram o controle automático atingir o seu nível atual, pode-se destacar alguns de grande importância:

1. **Relógio de água dos árabes e gregos**: em torno de 270 AC o grego Ktesibios inventou uma bóia regulável para um relógio de água, onde sua função era manter o nível de água em um tanque com uma profundidade constante.
2. **Revolução industrial**: a revolução industrial na Europa começou pela introdução das primeiras máquinas com movimentação própria, como moinhos de grãos, fornalhas, caldeiras e máquinas à vapor. Estes dispositivos não podem ser regulados diretamente pela ação de mãos humanas, sendo necessário empregar sistemas de controle automático. Uma grande variedade de dispositivos de controle foi inventada, incluindo bóias reguláveis, reguladores de temperatura, reguladores de pressão e dispositivos de controle de velocidade, para estas finalidades.
3. **O nascimento da teoria matemática de controle**: em meados do século XIX, a matemática foi usada para analisar a estabilidade de sistemas de controle realimentados, tornando-se a linguagem formal da teoria de controle automático.

4. **Análise no domínio da freqüência**: nos laboratórios da Bell Telephone, durante os anos 20 e 30 do século XX, o domínio da freqüência, desenvolvido por Laplace, Fourier e Cauchy, além de outros, foi explorado e usado em sistemas de comunicações.
5. **Projetos no domínio do tempo para sistemas não lineares**: na antiga União Soviética, uma grande atividade de projetos de controladores não lineares foi desenvolvida após a Segunda Guerra Mundial. Seguindo as idéias e formulações propostas por Lyapunov, as atenções foram concentradas nas técnicas no domínio do tempo.
6. **Controle Ótimo e Teoria da Estimação**: após ser capaz de projetar controladores, tornou-se necessário encontrar melhores soluções para o controle dos processos. No controle moderno, é usual aplicar minimização do tempo de transição de estados, ou a função de energia quadrática ou, ainda, um índice que mede um determinado desempenho, possivelmente com algumas restrições nas leis de controle. Em 1960, 3 importantes artigos científicos foram publicados por Kalman et al (Kalman e Bertram, 1960), (Kalman, 1960a) e (Kalman, 1960b), onde foram apresentados o trabalho de Lyapunov no domínio do tempo para controladores de sistemas não lineares, o controle ótimo de sistemas e a filtragem ótima e teoria da estimação.
7. **Computadores em projeto e implementação de controladores**: técnicas de projetos de controladores clássicos podem ser desenvolvidas utilizando abordagem gráfica. Já o projeto de controle moderno requer a solução de matrizes de equações não lineares complexas. A partir de 1960, houve um grande desenvolvimento da tecnologia digital, que resultou na aparição dos primeiros computadores digitais, sem os quais o controle moderno teria permanecido bastante limitado quanto as aplicações.
8. **Controle digital e Teoria de filtragem**: com o advento dos microprocessadores em 1969, a teoria de controle digital começou a se firmar, através da implementação de sistemas de controle em computadores digitais formulados no tempo discreto. Também começou a ser revelada a importância de técnicas de amostragem de dados no processamento de sinais.
9. **Controle inteligente**: no final da Segunda Guerra Mundial, grupos independentes de cientistas norte-americanos e ingleses desenvolveram uma máquina eletrônica que podia obedecer instruções e feita para executar cálculos numéricos complexos, máquina esta que veio resultar no computador digital. Um pequeno número de cientistas explorou a

capacidade dos computadores manipularem símbolos alfanuméricos. Simultaneamente, psicólogos interessados na resolução de problemas pelo homem, buscaram desenvolver programas de computador que simulassem o comportamento humano. Estes formaram uma subdivisão da informática denominada de Inteligência Artificial (IA), que pode ser dividida em três grupos principais: (a) processamento de linguagem natural; (b) desenvolvimento de robôs inteligentes; e (c) desenvolvimento de programas que simulem o comportamento/habilidade de especialistas humanos.

Com a evolução do controle automático, máquinas cada vez mais complexas foram desenvolvidas para a automação de tarefas, principalmente nas áreas industrial, médica e militar. Um grupo de máquinas denominadas de robôs foram criadas e com a finalidade de automatizar tarefas humanas, nas mais diversas áreas, sendo isto possível devido a evolução do controle automático. Para que se possa estabelecer um histórico sobre a evolução da robótica, apresenta-se no próximo tópico os principais fatos históricos relacionados à robótica, e que hoje é motivação de pesquisas de vanguarda na área de controle automático.

1.2 - Uma breve história da robótica

O conceito de robô data dos inícios da história com a mitologia que fazia referência a mecanismos imagináveis que ganhavam vida.

Começando na civilização grega, os primeiros “modelos” de robô concebido pelo homem eram figuras com aparência humana e/ou animal, que usavam sistemas de pesos e bombas pneumáticas.

As civilizações daquele tempo provavelmente não tinham necessidades práticas ou econômicas, nem nenhum sistema complexo de produtividade, que exigisse a existência deste tipo de aparelhos.

Cientistas árabes acrescentaram um importante e novo conceito à idéia tradicional de robôs, concentrando as suas pesquisas no objetivo de atribuir funções aos robôs que fossem ao encontro das necessidades humanas. A fusão da idéia de robôs, e a sua possível utilização prática, marcou o início de uma nova era.

Leonardo DaVinci abriu caminho a uma maior aproximação àquilo que entende-se hoje por

complexo mundo dos robôs. DaVinci desenvolveu uma extensiva investigação no domínio da anatomia humana que permitiu um aumento vertiginoso dos conhecimentos para a criação de articulações mecânicas. Como resultado deste estudo desenvolvido, surgiram diversos exemplares de bonecos que moviam as mãos, os olhos e as pernas, e que conseguiam realizar ações simples como escrever ou tocar alguns instrumentos.

Nikola Tesla, cientista na área da robótica que emigrou da Croácia para a América em 1800, e a propósito do grande desenvolvimento dos robôs e das grandes expectativas criadas em redor destes, afirmou: "Eu tratei campo global de modo geral, não me limitando a mecanismos controlados à distância, mas em máquinas possuídas de sua própria inteligência. Uma vez que o tempo tenha avançado enormemente na evolução dos inventos, e penso que ele não estará distante quando mostrarei uma automação que se faça por si mesma, agirá como se possuída de razão e sem nenhum voluntarioso controle externo".[1]

A palavra robô foi introduzida pelo dramaturgo Karel Capek. Esta palavra surgiu numa das suas mais prestigiadas peças, R.U.R. (Rossum's Universal Robots), e os robôs que nela intervieram não eram mecanizados.

O termo robótica refere-se ao estudo e à utilização de robôs, e foi pela primeira vez enunciado pelo cientista e escritor Isaac Asimov, em 1942, numa pequena história intitulada "Runaround". Asimov também publicou uma compilação de pequenas histórias, em 1950, intitulada "I Robot". Este autor propôs a existência de três leis aplicáveis à robótica, às quais acrescentou, mais tarde, a lei zero. As leis propostas são, atualmente, entendidas numa perspectiva puramente ficcional, pois no tempo em que foram escritas não se imaginava o desenvolvimento vertiginoso que ocorreria neste sentido.

O desenvolvimento inicial dos robôs baseou-se no esforço de automatizar as operações industriais. Este esforço começou no século XVIII, na indústria têxtil, com o aparecimento dos primeiros teares mecânicos. Com o contínuo progresso da revolução industrial, as fábricas procuraram equipar-se com máquinas capazes de realizar e reproduzir, automaticamente, determinadas tarefas. No entanto, a criação de verdadeiros robôs não foi possível até à invenção

---

[1] "*I treated the whole field broadly, not limiting myself to mechanics controlled from a distance, but to machines possessed of their own intelligence. Since that time had advanced greatly in the evolution of the invention and think that the time is not distant when I shall show an automation which left to itself, will act as though possessed of reason and without any willful control from the outside*"

do computador em 1940, e dos sucessivos aperfeiçoamentos das partes que o constituem.

O primeiro robô industrial foi o Unimate, desenvolvido por George Devol e Joe Engleberger, no final da década de 50, início da década de 60. As primeiras patentes de máquinas transportadoras pertenceram a Devol, máquinas essas que eram robôs primitivos que moviam objetos de um local para outro. Engleberger, devido a construção do primeiro robô comercial, foi apelidado de "pai da robótica".

Em anos recentes e inclusive na atualidade a área industrial se utiliza de vários braços robóticos, dentre eles os que mais se destacam são: SCARA, Puma e IRB. Tais robôs são mostrados nas figuras 1.1, 1.2 e 1.3, respectivamente.

Fig. 1.1 – Braço robótico SCARA.

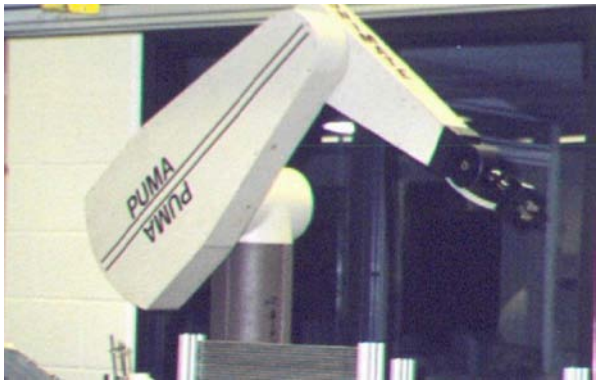

Fig. 1.2 – Braço robótico PUMA.

Fig. 1.3 – Braço robótico IRB 7600 da ABB.

Outros braços de robôs que estão sendo bastante usados na industriam são os fabricados pelas empresas Kuka e Kawasaki, mostrados nas figuras 1.4 e 1.5. Os robôs fabricados pela Kuka predominam na industria pesada automotiva e naval (até 570 Kg), enquanto que os robôs da Kawasaki lideram em aplicações de média cargas.

Fig. 1.4 – Braço robótico Kuka KR 500 570 PA/1

Fig. 1.5 – Braço robótico Kawasaki ZX165U

Além de aplicações industriais, os robôs estão começando a fazer parte do dia-a-dia da humanidade. Recentemente a Toyota Motor anunciou o protótipo do projeto que pretende desenvolver com robôs, que funcionariam como assistentes pessoais para humanos. A Toyota pretende que seus robôs possuam características humanas, como feições, temperatura do corpo, e que sejam inteligentes o suficiente para operar vários equipamentos nas áreas de assistência pessoal. Estes robôs são divididos em três categorias: os rolantes, os caminhantes e os montáveis. Os caminhantes têm a finalidade de dar assistência à pessoas idosas. Os rolantes são usados em lugares de acesso difícil, com pouco espaço para locomoção. Os montáveis são capazes de carregar pessoas em quase todos os lugares. Tais robôs podem ser vistos na figura 1.6.

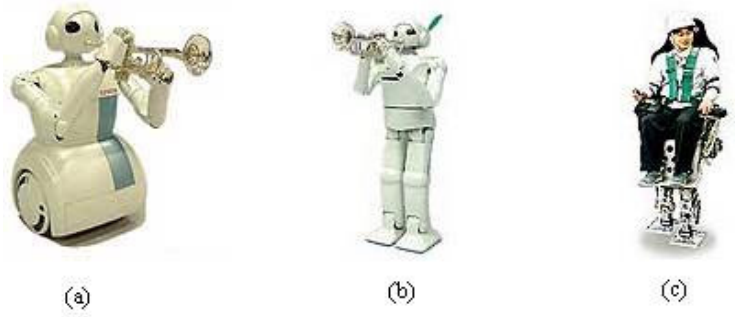

Fig. 1.6 – (a) Robô rolante; (b) Robô caminhante; e (c) Robô montável.

1.3 - Robótica e controle automático

Como pode ser visto na seção anterior, a robótica se aperfeiçoou principalmente com a

evolução da eletrônica e da mecânica aplicadas, sendo necessários sistemas de controle cada vez mais eficientes para poder conduzir estes sistemas a realizar tarefas pré-determinadas.

Sistemas de controle que consumam pouca energia podem viabilizar mecanismos robóticos móveis que são alimentados através de baterias, pois com pouco consumo de energia, uma bateria pode durar mais tempo, como aconteceu com os telefones celulares. Outra variável que pode ser otimizada é tempo de resposta, que quando minimizado pode tornar cada vez mais velozes os mecanismos robóticos.

Existem diversas formas de efetuar o controle de sistemas robóticos, dentre os quais se destacam os sistemas de controle adaptativo, sistemas de controle robusto e os sistemas de controle inteligentes.

Os sistemas adaptativos tentam modelar o processo para poder estimar os seus parâmetros visando criar um controlador próximo do ótimo. Já os controladores robustos são projetados após estudo matemático para poder garantir a estabilidade do controle em um determinado espaço de estados. Ambos necessitam de modelagem matemática que exige bastante precisão e complexidade para o desenvolvimento dos controladores, conforme pode ser visto em (Lunze, 1989), (Watanabe, 1992), (Astrom, 1995) e (Kouvelis e Yu, 1997).

Os sistemas de controle inteligentes nasceram de uma proposta diferente dos controladores adaptativos e robustos. Os controladores inteligentes são baseados em observações feitas na natureza, onde procurou-se observar como os mecanismos de aprendizagem se incorporam aos seres vivos e como estes fazem uso para a sua evolução. As principais técnicas de controle inteligente que estão sendo bastante utilizadas em sistemas robóticos são: redes neurais artificiais, lógica *fuzzy* e computação evolutiva. A seguir descrevemos como estas técnicas são empregadas neste contexto.

1.3.1 - Redes neurais artificiais

Em sistemas de controle, as redes neurais artificiais têm sido mais freqüentemente utilizadas para a identificação do processo, conforme pode ser visto em (Narendra e Parthasarathy, 1990), (Hitam e Gill, 2000) e (Chen e Narendra, 2000), ou simplesmente como um compensador, como usado em (Gupta e Sinha, 2000) e (Hong, Kaifang e Tingqi, 2002). Outros trabalhos que podem ser destacados que utilizam redes neurais em sistemas robóticos são mostrados abaixo:

- (Feng, 1997): é apresentado um algoritmo de controle para rastreamento de trajetórias de robôs no espaço das juntas. O controlador compensador é realizado através de uma estrutura *switch-type* e de uma rede neural RBF.
- (Ge, Hang e Woon, 1997): um controlador neural adaptativo de manipuladores robóticos no espaço de trabalho é apresentado. O controlador é desenvolvido baseado na técnica de modelagem em redes neurais, não requerendo a avaliação do modelo cinemático inverso em tempo necessário para o treinamento do processo.
- (Joo e Liew, 1997): é apresentada uma estratégia de controle para um manipulador robótico com n graus de liberdade. O projeto e a simulação do controlador neural são apresentados para um manipulador robótico industrial SCARA através do rastreamento de uma trajetória predeterminada, no espaço de juntas.
- (Zeman e Khorasani, 1997): uma rede neural é usada para aprender a dinâmica de um manipulador com junta flexível, uma vez que o controle adaptativo tradicional não pode ser aplicado, já que raramente as características não lineares podem ser ajustadas de maneira satisfatória.
- (Fierro e Lewis, 1998): é mostrada uma estrutura de controle que torna possível a integração de controle cinemático e um controlador neural de torque para robôs móveis não holonômicos.
- (Kwan, Lewis e Dawson, 1998): é apresentado um controlador neural robusto em que os pesos da rede neural do controlador são sintonizados em tempo real, sem necessidade de aprendizado anterior.
- (Jung e Hsia, 1998): é apresentado um controlador robusto aplicando-se as redes neurais para compensar as incertezas no modelo de um robô. As redes neurais são aplicadas para os métodos de controle de impedância baseados em controle de torque e controle de posição, que são distinguidos pela maneira com que a função de impedância é implementada.
- (Gutiérrez, Lewis e Lowe, 1998): são apresentados os resultados da implementação prática de controladores neurais para rastreamento em uma junta, e compara-se sua performance com controladores PD e PID.

- (Liu e Brooke, 1999): é apresentado um chip com uma rede neural para aprendizado em paralelo, que é usada para realizar o controle em tempo real de uma planta dinâmica não linear.
- (Sun, Sun, Zhang e Chen, 2000): um controle adaptativo baseado em redes neurais é proposto para o controle do rastreamento de um manipulador robótico com n juntas, cujas não linearidades dinâmicas são supostamente desconhecidas.
- (Jung e Hsia, 2000): é usada uma rede neural como compensador para cancelar incertezas. Este controlador neural é capaz de fazer o robô rastrear uma força específica assim como compensar as incertezas no ambiente e na dinâmica do robô.
- (Yildirim, 2001): um controlador neural é usada para o controle de dois braços de robôs do tipo Scara, que trabalham em cooperação.
- (Visioli e Legnani, 2002): são apresentados vários controladores que apresentam estratégias de controles diferentes, aplicados para o controle de rastreamento de trajetória de um robô manipulador industrial tipo Scara. Dentre os diferentes controladores, o controlador neural apresentado se destaca para o rastreamento de trajetórias em alta velocidade, tendo um baixo erro de rastreamento.
- (Patino, Carelli e Kuchen, 2002): é apresentado um controlador neural que inclui uma combinação linear de um conjunto de redes neurais treinadas *off-line* e uma lei que ajusta a dinâmica do robô e os parâmetros incertos. Um resultado prático é mostrado usando-se este sistema.
- (Lee, Eom e Lee, 2002): é apresentado um método de controle adaptativo estável baseado em redes neurais.
- (Yildirim , 2002): é usado um método para controle de trajetória através de redes neurais. O sistema de controle é composto por um controlador baseado em redes neurais e um controlador realimentado de ganho fixo. O controlador neural emprega uma rede neural recorrente, em que os pesos destas são obtidos através do treinamento de uma outra rede neural para identificação *on-line* da dinâmica inversa do robô.
- (Wai e Lee, 2004): um controlador ótimo inteligente é apresentado para controlar um mecanismo acoplado a um motor, sendo que este mecanismo é um *link* de um

braço de robô flexível. No sistema de controle inteligente, uma rede *neural-fuzzy* é usada para estabelecer uma função não linear na lei de controle ótima, e um controlador robusto é projetado para compensar o erro de aproximação.

A lógica fuzzy tem sido bastante usada na configuração de redes *neural-fuzzy*, onde ela é embutida na definição da função que representa o neurônio da rede neural artificial.

Algumas aplicações práticas podem ser vistas em (Monteiro, 1996), (Monteiro e Takita, 1996), (Monteiro e Takita, 1996a), (Souza, 2000), (Jungbeck, 2002) e (Hong, Kaifang e Tingqi, 2002).

1.3.2 - Algoritmos genéticos

Os algoritmos genéticos (AG) possuem uma larga aplicação em muitas áreas científicas, entre as quais podem ser destacadas:

- Síntese de circuitos analógicos: para uma certa entrada e uma saída desejada, por exemplo tensão, o AG gera a topologia, o tipo e o valor dos componentes do circuito.
- Síntese de protocolos: determinação de quais funções do protocolo devem ser implementadas em hardware e quais devem ser implementadas em software, para que um certo desempenho seja alcançado.
- Programação Genética: gera a listagem de um programa, numa determinada linguagem especificada, para que um determinado conjunto de dados de entrada forneça uma saída desejada.
- Gerenciamento de redes: supervisão do tráfego nos *links* e das filas nos *buffers* de roteadores para descobrir rotas ótimas e para reconfigurar as rotas existentes no caso de falha de algum *link*.
- Computação Evolutiva: gera programas que se adaptam a mudanças que possam ocorrer no sistema ao longo do tempo.
- Otimização evolutiva multi-critério: otimização de funções com múltiplos objetivos que sejam conflitantes.
- Problemas de otimização complexos: problemas com muitas variáveis e espaços de soluções de dimensões elevadas.

- Ciências biológicas: modela processos biológicos para o entendimento do comportamento de estruturas genéticas.
- Autômatos auto-programáveis.

Em sistemas robóticos, os algoritmos genéticos normalmente são usados pra otimização evolutiva multi-critério e para problemas de otimização complexos. Abaixo estão descritas algumas das aplicações em que a comunidade científica tem utilizado os algoritmos genéticos.

- (Xiao, Michalewicz, Zhang e Trojanowski, 1997): é apresentado planejador/navegador adaptativo evolucionário, onde este unifica o planejamento *off-line* e o planejamento/navegação *on-line*.
- (Gill e Zomaya, 1998): é apresentada uma abordagem onde se combina os campos potenciais, que guiam o robô através do espaço cartesiano, e um algoritmo genético paralelo, que calcula os valores das variáveis das juntas para uma trajetória de um robô.
- (Juidette e Youlal, 2000): os algoritmos genéticos são usados como uma ferramenta de busca e otimização para procurar uma tabela de decisão ótima, que é usada como máquina de inferência com proposta de um algoritmo de planejamento e navegação baseados em lógica *fuzzy*.
- (Xia e Wang, 2001): é apresentada uma rede neural *dual* para o controle cinemático de robôs manipuladores redundantes.
- (Chen e Li, 2004): os algoritmos genéticos são usados para a otimização do posicionamento de sensores, onde o critério *min-max* é usado na avaliação.
- (Zhang e Wang, 2004): uma rede neural recorrente é usada para o controle cinemático de manipuladores redundantes com a capacidade de desvio de obstáculos.

Os algoritmos genéticos também são bastante usados para a otimização de sistemas de controle inteligentes que utilizam lógica fuzzy e/ou redes neurais, conforme pode ser observado nas seguintes referências:

- (Moon e Kong, 2001): é apresentado um modelo de rede neural baseada em blocos e a otimização da estrutura e dos pesos desta é feita através de um algoritmo genético. A estrutura e os pesos da rede neural apresentada é codificada em *string* de *bits* para poder ser configurada em FPGAs (*Field Programmable Logical*

Arrays). Simulações mostram que a rede neural otimizada pode resolver problemas relacionados à engenharia como classificação de padrões e controle de robôs móveis.

- (Lee e Cho, 2001): os controladores fuzzy são desenvolvidos para o controle de um robô móvel com otimização via algoritmos genéticos em ambiente de simulação, e analisa o comportamento de tais controladores no modelo interno do diagrama de transição de estados.
- (Chen e Chiang, 2003): mostra um sistema de controle inteligente denominado de sistema de controle inteligente auto-explorável, que possui 3 mecanismo básicos: controlador, avaliador de desempenho e adaptador. O controlador é construídos utilizando redes neurais nebulosas. O avaliador de desempenho é baseado em um algoritmo genético com múltiplos objetivos.
- (Gao e Er, 2003): é apresentado um controlador neural-fuzzy adaptativo e robusto adequado para a identificação e controle de sistemas não lineares MIMO (*multiple-input multiple-output*) com incertezas, que possui as seguintes características: rede neural-fuzzy auto-organizável, aprendizado on-line, aprendizado rápido, convergência rápida de erros de rastreamento, controle adaptativo e controle robusto.
- (Leung, Lam, Ling e Tam, 2004): é apresentado um controlador *fuzzy* de um modelo de planta, também *fuzzy*, que descreve as dinâmicas incertas não lineares desta. Os ganhos de realimentação do controlador neural-fuzzy e a solução para se atingir a estabilidade são determinados usando um algoritmo genético.

Como visto nas referências mostradas acima, vários cientistas estão trabalhando em função de resolverem problemas de robótica através de sistemas inteligentes. Um dos problemas a serem resolvidos é o planejamento de trajetória em um braço de robô industrial, principalmente levando em consideração variáveis que devem ser otimizadas globalmente. Normalmente o planejamento de trajetória é feito através de complexas heurísticas ou através de cálculos matemáticas que envolvem mapeamento do ambiente, dinâmica do braço do robô, tipo de trajetória, cinemática direta e inversa do robô, etc.

Outra grande variável ainda não fechada é o controle das juntas de um braço de robô.

Normalmente são usados controladores do tipo PID que são dominados pelos engenheiros de controle, porém necessitam de constante manutenção devidos as alterações em sua dinâmica e também das alterações da dinâmica do ambiente de trabalho. Neste ponto, vários controladores *fuzzy* e neurais foram e estão sendo desenvolvidos.

O principal objeto deste trabalho é contribuir com a apresentação de soluções para algumas das categorias dos problemas envolvidos em robótica. A resolução destes problemas têm sido amplamente pesquisadas pelos cientistas na área da robótica, sempre procurando otimizar cada vez mais variáveis, o que torna os sistemas robóticos mais robustos. Neste trabalho, a metodologia usada para a resolução dos problemas, mostrados no tópico a seguir, são puramente tratadas com técnicas de inteligência artificial que utilizam os algoritmos genéticos e as redes neurais, que não requerem modelamentos analíticos complexos, conhecimento prévios sob a planta a ser controlada e fornecem a robustez necessária para a tarefa a ser controlada em questão.

1.4 - Organização deste trabalho

Neste trabalho, os algoritmos genéticos e as redes neurais artificiais são usados para o desenvolvimento de três tarefas complexas e fundamentais em robótica:

1. Planejamento de trajetória
2. Controle de posição
3. Seleção de controladores de posição

Os algoritmos genéticos são usados na primeira e terceira tarefa para que possam otimizar a trajetória a ser rastreada por um robô quanto a seleção de um controlador próximo do ótimo dentre várias, senão infinitas, soluções existentes, todos baseados em (Goldberg, 1989) e (Koza, 1992). O planejamento de trajetórias foi baseado nos trabalhos desenvolvidos por (Madrid, 1994) e (Madrid e Badan, 1997). Já as redes neurais artificiais são utilizadas para efetivamente controlar a posição de uma junta de um braço de robô, onde é apresentada uma nova estrutura de controle utilizando uma rede neural artificial. Tais técnicas são usadas com a finalidade de diminuir o tempo de planejamento e projeto, uma vez que para ter tais atividades executadas por meios convencionais normalmente é necessária a modelagem do processo. Caso o processo seja um robô do tipo Puma 560, existem vários modelos cinemáticos e dinâmicos (Corke e Armstrong-

Hélouvry, 1994), (Swift, Kaufman e Cummings, 1993), sendo que existe a possibilidade de tais modelos não servirem para determinados tipos de controladores, o que exigiria a determinação de novos modelos, demandando um grande esforço para os modelamentos matemático e físico. Outras publicações referentes ao planejamento de trajetória usando tanto algoritmos genéticos quanto outras técnicas alternativas podem ser encontradas em (Nehmzow, 2002), (Martin, 2002), (Zhang e Wang, 2004), (Xia e Wang, 2001), (Gill e Zomaya, 1998) e (Juidette e Youlal, 2000).

No capítulo 2 são apresentados os conceitos básicos referentes às redes neurais artificiais e aos algoritmos genéticos, onde é abordado o algoritmo de aprendizagem *backpropagation,* e o algoritmo genético clássico, duas ferramentas que são usadas nos demais capítulos.

No capítulo 3 o algoritmo genético clássico é usado para o planejamento de uma trajetória de um braço de robô com um obstáculo, onde são apresentadas operações de *crossover* e mutação adaptadas para a otimização da trajetória. Ainda neste capítulo são mostrados exemplos da geração de trajetórias cartesianas compostas por 6 pontos e exemplos de rastreamento de trajetórias no espaço de juntas.

No capítulo 4 um controlador de posição que utiliza uma rede neural em sua estrutura é apresentado, onde não é necessário fazer nenhum tipo de treinamento prévio para que a rede neural possa controlar a posição de uma mesa XY e de um pêndulo invertido acionado, com a utilização de paradigmas de montagem realizados pela equipe de pesquisa onde este trabalho foi desenvolvido. Como a estrutura do controlador neural possui vários parâmetros, este capítulo ainda apresenta um algoritmo genético clássico adaptado para encontrar o melhor controlador dentre vários.

No capítulo 5 apresentam-se as conclusões sobre as aplicações dos algoritmos utilizados neste trabalho, além de propor quais as futuras ações para novos estudos na área.

Como pode ser visto na descrição do conteúdo dos capítulos deste trabalho, a principal intenção deste trabalho foi a aplicação prática de técnicas de inteligência artificial, tendo foco centralizado em aplicações práticas de técnicas de inteligência artificial para planejamento e controle de braços de robôs, procurando mostrar as facilidades e dificuldades envolvidas nas aplicações destas técnicas em processos práticos e em tempo real.

CAPÍTULO 2

CONCEITOS BÁSICOS

A evolução é natural ao homem que busca sempre melhorar suas condições de vida. Já que o homem possui a necessidade de comer e lutar para se manter vivo, houve o interesse da criação de artefatos artificiais que o complementaram para que atingisse o nível de evolução atual.

Pode-se observar alguns itens que foram fundamentais para que ele evoluísse. Um bom exemplo é o fogo. Inicialmente, o fogo foi visto pelos seres humanos primitivos e eles perceberam que os seus predadores poderiam ser afugentados pelo fogo. Isto os levou a procurar obter o fogo de modo fácil e mantê-lo sempre aceso para usá-lo assim que fossem ameaçados por seus predadores. Após conviver bastante com o fogo, perceberam que o fogo tinha outras utilidades, como para assar alimentos e lutar para conquistar novas terras.

Assim como o fogo, vários outros fatores levaram o ser humano a evoluir para o estado atual, tornando-se possível o estudo de outras espécies, o que levou ao entendimento de que as espécies existentes atualmente são evoluções de antigas espécies.

Como o homem é o centro de geração de conhecimento, percebe-se que ele pode resolver diversos problemas. Durante muito tempo, a matemática, desenvolvida por ele, foi a base para a resolução de vários problemas, tais como construção de máquinas, implementação de computadores, algoritmos computacionais para solução de equações complexas, lançamento de foguetes, etc. Porém, os problemas mais novos estão requerendo modelagens matemáticas cada vez mais complexas, o que impõe a utilização de computadores cada vez mais rápidos. Ao longo de várias décadas, não houve evolução notável em como tais problemas seriam resolvidos até o surgimento das técnicas de inteligência artificial, que começaram a ser abordadas cientificamente na década de 50.

As técnicas de inteligência artificial tiveram como base o fato de que o ser humano consegue na maioria das vezes resolver vários problemas, inclusive aqueles que não são ainda modeláveis matematicamente. Com isto, o ser humano voltou-se para si mesmo, indagando-se como ele próprio faz para resolvê-los. A partir disto, várias técnicas que tentam reproduzir uma parte do raciocínio humano foram desenvolvidas, dentre elas destacam-se os sistemas

especialistas, os sistemas heurísticos, as redes neurais artificiais e a semiótica.

Outra abordagem que está sendo estudada faz parte da observação da evolução das espécies, tal como sugerido primordialmente por Charles Darwin. O ramo da computação que abrange tais estudos chama-se computação evolutiva, sendo que esta pode ser subdividida nas seguinte áreas: algoritmos genéticos, programação evolutiva, estratégias de evolução, sistemas classificadores e programação genética. Todas elas compartilham o conceito da evolução de um indivíduo através de operações de seleção, mutação e reprodução.

Dentre todas as técnicas pertencentes às áreas acima citadas, os algoritmos genéticos e as redes neurais artificiais foram selecionados para solucionar o problema do planejamento de trajetórias e do controle de posição dos robôs, por apresentarem facilidades de adaptação às diferentes formas de ambiente e por apresentarem robustez adequada frente aos ambientes em que existe alto índice de ruído.

A seguir são apresentados alguns detalhes das duas técnicas abordadas neste trabalho.

2.1 - Algoritmos Genéticos

Os algoritmos genéticos são algoritmos de busca heurística adaptativos baseados nas idéias propostas pela seleção natural e pela genética, (Otto, Otto e Frota-Pessoa, 1998), (Goldberg, 1989), (Holland, 1992) e (Koza, 1992).

Suas principais metas são:

- Abstrair e rigorosamente explicar os processos adaptativos naturais
- Desenvolver simulações em computador que retenham alguns mecanismos originais encontrados em sistemas naturais.

Eles podem ser descritos, genericamente, através dos passos abaixo.

1. Gerar aleatoriamente uma população inicial **M(0)**;
2. Computar e salvar o *fitness*[2] **u(m)** de cada indivíduo **m** da população corrente;
3. Definir as probabilidades de seleção **p(m)** de cada indivíduo **m** em **M(t)**, de tal

[2] *Fitness* é a forma pela qual um indivíduo é avaliado. Normalmente é associado ao *fitness* uma função que serve para avaliar todos os indivíduos.

forma que **p(m)** seja proporcional a **u(m)**;

4. Gerar a próxima população **M(t+1)** pela seleção probabilística dos indivíduo de **M(t)** para produzir descendência via operadores genéticos;
5. Aplicar a mutação probabilística nos indivíduos da próxima geração **M(t+1)**;
6. Retornar ao passo 2 enquanto a solução desejada não for obtida.

As principais características destes algoritmos são:

- Fazem buscas sobre uma população de indivíduos e não sobre um único indivíduo
- Fazem uso de restrições genéricas em relação ao que se quer ver presente na solução, através da função de *fitness*
- Utilizam regras de transição probabilística

O principal algoritmo utilizado para derivar outros é o algoritmo genético clássico, conforme pode ser visto no tópico 2.2.

2.2 - Algoritmo genético clássico

2.2.1 - População

Neste algoritmo genético a população possui um tamanho fixo, e a informação a ser processada deve estar codificada em uma *string* de *bits*, ou em uma estrutura de dados do tipo binária, de tamanho fixo, onde cada indivíduo da população, chamado de cromossomo, é composto por uma seqüência de números binários.

Na figura 2.1 pode-se ver um exemplo de uma população.

Cromossomo 1	0	1	1	1	0	1	0	0	0	1	1
Cromossomo 2	1	1	1	0	0	0	1	1	1	1	0
Cromossomo 3	1	0	0	0	1	1	0	1	1	1	0
Cromossomo 4	0	1	0	1	1	0	0	0	0	0	0
Cromossomo 5	0	0	0	1	1	1	0	1	0	0	0
Cromossomo 6	0	0	0	1	0	1	0	0	0	1	1
Cromossomo 7	1	1	1	0	1	0	1	1	1	1	0
Cromossomo 8	1	0	0	0	0	0	0	1	1	1	0
Cromossomo 9	0	1	0	1	1	0	1	1	1	1	1
Cromossomo 10	0	0	0	1	1	1	0	1	0	1	0

Fig. 2.1 – População em um algoritmo genético clássico.

Cada cromossomo representa uma possível solução para o problema que se deseja resolver. Na computação evolutiva todos os cromossomos são avaliados através de uma função de *fitness*, e sofrem aplicações de operações genéticas.

2.2.2 - Seleção

Para que possa haver evolução, a operação genética de seleção é aplicada levando-se em conta a probabilidade de reprodução de cada um dos cromossomos em função do seu respectivo valor de *fitness*.

A seleção é feita através do algoritmo *Roulette Wheel* (Koza, 1992). Neste método os cromossomos com melhor *fitness* possuem probabilidade maior de serem escolhidos para a reprodução.

Como existe uma maior ou menor probabilidade de um cromossomo ser selecionado para a reprodução, pode-se, inclusive, ter a perda do melhor indivíduo já produzido e, conseqüentemente, a melhor solução encontrada até aquele momento pelo processo evolutivo.

2.2.3 - Reprodução

Cada reprodução irá gerar um novo cromossomo a partir da combinação de outros dois, que foram selecionados através do *Roulette Wheel*.

O algoritmo genético clássico utiliza o cruzamento simples ou *crossover* simples.

Como exemplo, suponha os seguintes cromossomos selecionados para reprodução:

Cromossomo 1: 10101010000111

Cromossomo 2: 01010101111110

Ponto de *crossover*

O ponto de *crossover* é definido aleatoriamente, sendo a partir dele efetuada a troca de dados entre os dois cromossomos, conforme pode ser visto abaixo:

Cromossomo 1: 1010101 0000111	Cromossomo 1: 10101011111110
Cromossomo 2: 0101010 1111110	Cromossomo 2: 01010100000111
Antes do cruzamento	Após o cruzamento

Com esta operação, duas novas cadeias são formadas, e duas novas soluções geradas. Os cromossomos geradores são abandonados, o que também pode ocasionar a perda da melhor solução já encontrada para um determinado problema.

Outro critério que é usado para a operação de combinação dos elementos selecionados é o *crossover* de dois pontos, onde dois pontos são selecionados aleatoriamente e o material os bits entre estes dois pontos são trocados, conforme visto abaixo.

Cromossomo 1: 10101010000111	Cromossomo 1: 10010101100111
Cromossomo 2: 01010101111110	Cromossomo 2: 01101010011110
Antes do cruzamento	Após o cruzamento

Outra operação de recombinação muito usada é o *crossover* uniforme, onde todos os bits de cada pai são testados, segundo uma probabilidade, indicando qual dos pais fornecerá o bit para o elemento na próxima geração.

2.2.4 - Mutação

Na mutação biológica, a mutação pontual é um processo no qual um alelo[3] de um gene é substituído por outro, o que resulta em um novo cromossomo.

No algoritmo genético clássico, a mutação ocorre pela troca de valor de um *bit* escolhido aleatoriamente em um cromossomo da população, que por sua vez também é escolhido aleatoriamente. Neste caso existe uma probabilidade de mutação, que normalmente possui um valor bastante inferior ao valor da probabilidade de cruzamento.

Caso um cromossomo seja escolhido para uma mutação, a mutação simples ocorre conforme pode ser visto no seguinte exemplo:

Cromossomo: 01**0**10101111110 Novo cromossomo: 01**1**10101111110

Bit para mutação

2.2.5 - Conclusão sobre o algoritmo genético clássico

O algoritmo genético clássico apresenta os seguintes problemas:

- Caso o problema não possua solução binária, há a necessidade de codificar-se as soluções a serem pesquisadas em uma ***string*** de *bits*.
- As operações de seleção e/ou reprodução admitem a perda do melhor cromossomo.
- A posição de genes consecutivos nas cadeias cromossômicas influi na probabilidade de permanecerem como o mesmo cromossomo após a reprodução.

Estes problemas são resolvidos através da modificação do algoritmo clássico, tornando-o mais robusto. Porém, cada problema a ser resolvido deve ser acompanhado de um estudo das suas características, uma vez que qualquer modificação, seja nos operadores genéticos ou na codificação das soluções, pode acelerar ou retardar o encontro da solução ótima, ou quase ótima, dentro do universo de busca.

Para os problemas apresentados neste trabalho, foram feitas diversas modificações, tendo

[3] Alelo: estado de um gene.

em vista principalmente melhorar a velocidade de busca das soluções. Todas as modificações propostas são apresentadas nos capítulos 3 e 4.

2.3 - REDES NEURAIS ARTIFICIAIS

Trabalhos sobre redes neurais artificiais, comumente referidas como redes neurais, têm sido incentivados desde a sua concepção, pelo reconhecimento de que o cérebro humano processa informações de uma forma totalmente diferente dos computadores digitais convencionais, tais como as máquinas de Von Neumann. O esforço para entender o cérebro deve muito ao trabalho pioneiro de (Ramón y Cajál, 1911), que introduziu a idéia dos neurônios como estruturas constituintes do cérebro.

Tipicamente, os neurônios são cinco a seis vezes menores do que uma porta lógica de silício; eventos em um CI de silício ocorrem na escala de 10^{-9} segundos, enquanto que eventos neurais acontecem na escala de 10^{-3} segundos. Contudo, o cérebro possui uma massiva conexão entre os neurônios, estimando-se que há cerca de 10 bilhões de neurônios no córtex humano, e 60 trilhões de sinapses ou conexões (Shepherd e Koch, 1990).

O cérebro é altamente complexo, não linear e processa informações paralelamente. Ele tem a capacidade de organizar os neurônios e executar certas operações (reconhecimento de padrões, percepção e controle motor) muito mais rápidas do que o computador digital mais rápido que existe hoje. Recentemente o computador Deep Blue venceu o jogador de xadrez Gasparov em uma competição de xadrez (Fogel, 2002), mas não pela velocidade de raciocínio e sim, provavelmente, pelo cansaço mental de Gasparov.

No nascimento, um cérebro tem uma grande estrutura e habilidade para construir suas próprias regras através do que se denomina geralmente de "experiência". Em verdade, a experiência é conseguida através dos anos, com um grande desenvolvimento do cérebro humano nos dois primeiros anos a partir do nascimento, bem como além deste estágio. Durante este estado de desenvolvimento, aproximadamente 1 milhão de sinapses são formadas por segundo (Haykin, 1996).

Sinapses são elementos estruturais e unidades funcionais que medem a interação dos neurônios. É assumido que uma sinapse é uma simples conexão que pode impor excitação ou inibição, mas não as duas, sobre o neurônio receptor.

O desenvolvimento de um neurônio é sinônimo de um cérebro plástico: a plasticidade permite o desenvolvimento do sistema nervoso que adapta-se ao ambiente que o envolve. Em um cérebro de adulto, a plasticidade pode ser considerada por dois mecanismos: a criação de novas sinapses de conexão entre neurônios e modificações das sinapses existentes.

Axônios (linhas de transmissão) e os dendritos (zonas receptivas) constituem dois tipos de filamentos de células que são distinguidos pela região morfológica. Um axônio possui uma superfície suave e um grande comprimento, enquanto que um dendrito tem uma superfície irregular, (Romero, 2000) e (Jorde, Carey e White, 1995).

Os neurônios possuem uma variedade de formas e tamanhos em diferentes partes do cérebro. A figura 2.2 ilustra a forma piramidal, que é um dos tipos mais comuns dos neurônios corticais.

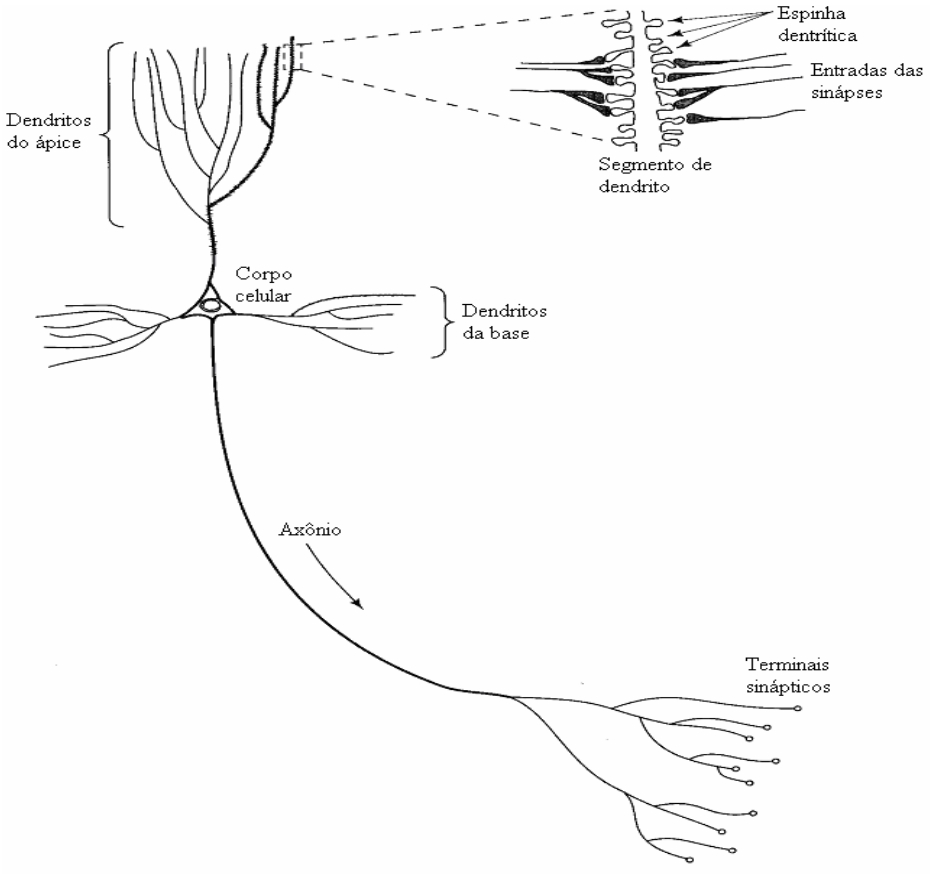

Fig. 2.2 - Célula piramidal, (Haykin, 1996).

Na sua forma mais geral, uma rede neural artificial é uma máquina que é projetada para modelar o caminho pelo qual o cérebro executa uma tarefa particular, ou função de interesse. A rede é geralmente implementada usando-se componentes eletrônicos, ou simulada via software

em um computador digital. Para adquirir boa performance, as redes neurais empregam interconexão massiva de células simples chamadas de neurônios ou unidades de processamento. Pode-se adotar a seguinte definição de rede neural vista como uma máquina adaptativa, baseada na definição de (Aleksander e Morton, 1990):

Uma rede neural é um processador distribuído massivamente paralelo, que possui uma propensão natural para armazenar conhecimentos experimentais e fazê-los disponíveis para uso. Ela assemelha-se ao cérebro humano em dois aspectos:

1) *Conhecimento é adquirido pela rede através de um processo de aprendizagem.*
2) *A energia de conexão entre neurônios, conhecida como peso sináptico, é usada para armazenar conhecimento.*

O procedimento usado para executar o processo de aprendizagem é conhecido como algoritmo de aprendizagem, e ele é o responsável pela alteração dos pesos sinápticos fazendo com que a rede neural se adapte às condições do meio em que está envolvida (Haykin, 1996), (Kosko, 1992).

2.3.1 - Benefícios Oferecidos pelas Redes Neurais

Do citado acima, é aparente que uma rede neural deriva sua força computacional da estrutura distribuída massivamente paralela, da habilidade de permitir o aprendizado, e da sua característica de generalização. A generalização refere-se à rede neural produzir saídas adequadas para entradas não encontradas durante o treinamento (aprendizado). Estas três capacidades de processamento de informação tornam possível que a rede neural resolva problemas complexos que dificilmente podem ser tratados analiticamente, ou mesmo são intratáveis correntemente por outra técnica disponível.

As redes neurais têm as seguintes propriedades e capacidades:

1) **Não linearidade:** um neurônio é basicamente um dispositivo não linear. Conseqüentemente, uma rede neural, criada da interconexão dos neurônios, é, também, não linear.
2) **Mapeamento entrada/saída multidimensional:** um paradigma popular de aprendizado chamado de aprendizado supervisionado envolve a modificação dos pesos sinápticos de uma rede neural aplicando um conjunto chamado de amostras de treinamento ou exemplos de tarefas. Cada exemplo, que consiste de um único sinal de entrada e a

correspondente resposta desejada, é apresentado à rede aleatoriamente, e seus pesos sinápticos (parâmetros livres) são modificados para minimizar a diferença entre a saída desejada e a saída real, produzida pelo sinal de entrada da rede. Logo, a rede neural "aprende" através de exemplos pela construção de um mapeamento entrada/saída.

3) **Adaptação:** redes neurais têm a capacidade de adaptar seus pesos sinápticos para mudanças que possam ocorrer no ambiente. Em particular, uma rede neural treinada para operar em um ambiente específico pode ser facilmente retreinada para adequar-se a um ambiente modificado, com poucas, ou nenhuma, mudança na sua estrutura.
4) **Resposta evidente:** no contexto da classificação de padrões, uma rede neural pode ser projetada para fornecer informações não somente sobre um padrão particular, mas também sobre a confiança na decisão fornecida.
5) **Informação contextual:** conhecimento é representado pela estrutura e estado de ativação da rede neural. Todo neurônio pertencente à rede neural é potencialmente afetado pela ativação dos demais neurônios que a compõem. Conseqüentemente, informação contextual é distribuída pela rede neural.
6) **Tolerância à falha:** uma rede neural tem um potencial para ser inerentemente tolerante a falha, pois caso ocorram rompimento de suas conexões, a sua capacidade de processar informações ainda pode ser mantida, devido unicamente à sua estrutura ser massivamente paralela.
7) **Implementação em VLSI (*Very Large Scale Integration*):** a natureza massivamente paralela de uma rede neural torna-a potencialmente rápida para a computação de certas tarefas. Esta mesma característica torna-a ideal para a implementação utilizando tecnologias VLSI.
8) **Uniformidade de análise e projeto:** basicamente, as redes neurais universalizam os processos de informação. No sentido de que a mesma notação é usada em todos os domínios envolvendo suas aplicações.
9) **Analogia neurobiológica:** o projeto de uma rede neural é motivado pela sua semelhança com o cérebro humano, que é a prova viva de que o processamento paralelo tolerante à falha não é somente fisicamente possível, mas também rápido e eficiente.

2.3.2 - Modelos de Neurônios

Um neurônio é uma unidade de processamento de informação que é essencial para a operação de uma rede neural. A figura 2.3 mostra um modelo convencional para um neurônio.

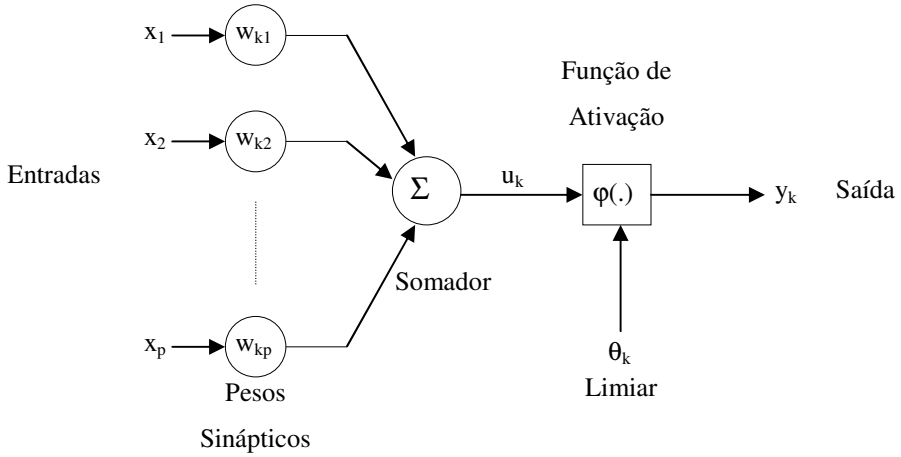

Fig. 2.3 - Modelo de neurônio não linear.

Pode-se identificar três elementos básicos neste modelo de neurônio:

1. Um conjunto de *sinapses*, cada uma sendo caracterizada por um peso ou energia própria. Especificamente, um sinal x_j, na entrada da sinapse j, conectada ao neurônio k, é multiplicado pelo peso sináptico w_{kj}. O peso w_{kj} é positivo se a sinapse associada é excitatória, e negativo no caso desta ser inibitória.

2. Um somador para adicionar os sinais de entrada, ponderada pelas suas respectivas sinapses neurais. A operação realizada aqui é oriunda de uma combinação linear.

3. Uma função de ativação para limitar a amplitude da saída produzida pelo neurônio.

O modelo do neurônio mostrado na figura 2.3 inclui um limiar aplicado externamente, θ_k, que tem a finalidade de atenuar a entrada de rede da função de ativação. Por outro lado, a entrada da rede da função de ativação pode ser amplificada pelo emprego de um termo *bias* em lugar do limiar, que corresponde ao seu valor negativo.

Em termos matemáticos, pode-se descrever um neurônio k através das seguintes equações:

$$u_k = \sum_{j=1}^{p} w_{kj} x_j \tag{2.1}$$

e

$$y_k = \varphi(u_k - \theta_k) \tag{2.2}$$

onde x_1, x_2, ... , x_p são os sinais de entrada; w_{k1}, w_{k2}, ..., w_{kp} são seus pesos sinápticos; u_k é a combinação linear de saída; θ_k é a constante de limiar; $\varphi(.)$ é a função de ativação; y_k é o seu sinal de saída. O uso da constante θ_k tem o efeito de aplicar uma transformação **afim** para a saída u_k da combinação linear no modelo da figura 2.3 como mostrado por:

$$v_k = u_k - \theta_k \tag{2.3}$$

Em particular, dependendo se o limiar θ_k é positivo ou negativo, a relação entre o nível de ativação interno efetivo, ou potencial de ativação, (v_k) do neurônio k e a saída da combinação linear u_k é modificada conforme ilustrado na figura 2.4.

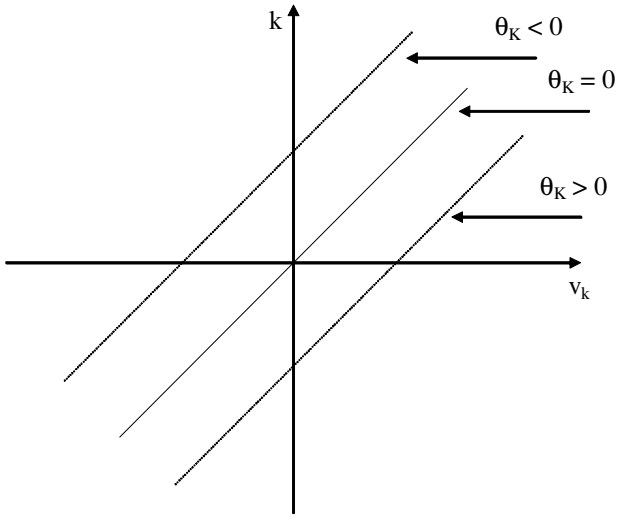

Fig. 2.4 - Transformação afim produzida pela presença do limiar.

2.3.3 - Tipos de Função de Ativação

A função de ativação, denotada por $\varphi(.)$, define a saída do neurônio em termos do nível de ativação nas suas entradas. Pode-se identificar três tipos básicos de função de ativação:

1) Função limiar: para este tipo de função de ativação, descrita na figura 2.5, tem-se:

$$\varphi(v) = \begin{cases} 1, & \text{se } v \geq 0 \\ 0, & \text{se } v < 0 \end{cases} \tag{2.4}$$

Correspondentemente, a saída do neurônio k que emprega tal função de ativação é expressa como:

$$y_k = \begin{cases} 1, & \text{se } v_k \geq 0 \\ 0, & \text{se } v_k < 0 \end{cases} \tag{2.5}$$

onde v_k é o nível de ativação interno do neurônio, ou seja:

$$v_k = \sum w_{kj} x_j - \theta_k \tag{2.6}$$

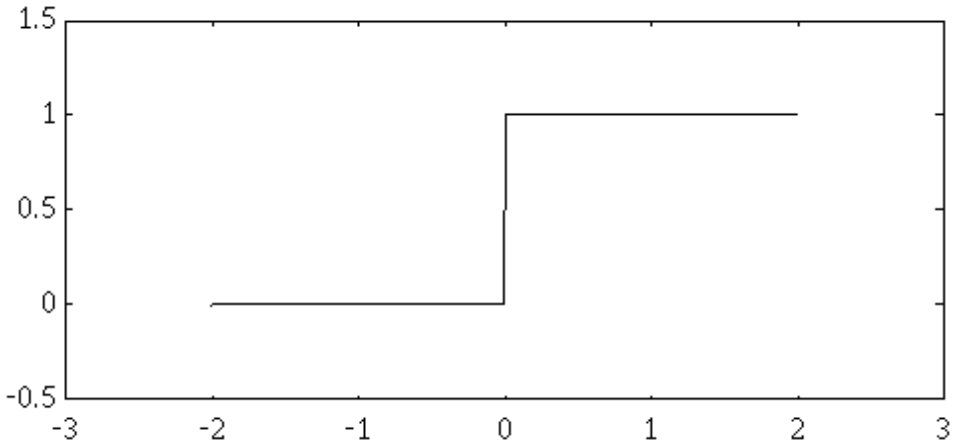

Fig. 2.5 - Função de limiar.

Tal neurônio é referido na literatura como o modelo de *McCulloch-Pitts* (McCulloch e Pitts, 1943). Neste modelo, a saída do neurônio possui o valor 1 se o nível de ativação interno total daquele neurônio for não negativo; do contrário a saída terá valor zero. Isto descreve a propriedade do tudo ou nada do modelo de *McCulloch-Pitts*.

2) Função linear por partes: para a função de ativação linear por partes, descrita na figura 2.6, tem-se:

$$\varphi(v) = \begin{cases} 1, & \text{se } v \geq 0.5 \\ av + 0.5, & \text{se } -0.5 < v < 0.5 \\ 0, & \text{se } v \leq -0.5 \end{cases} \tag{2.7}$$

onde o fator de amplificação dentro da região linear de operação é considerado unitário e *a* indica a inclinação desejada do segmento de reta. Esta forma de função de ativação, equação (2.7), pode ser vista como uma aproximação de um amplificador não linear.

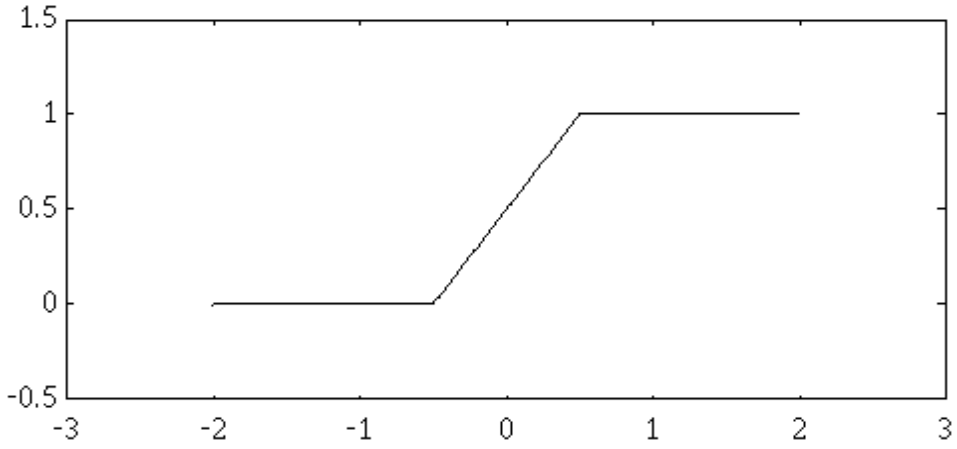

Fig. 2.6 - Função linear por partes.

3) Função sigmóide: a função sigmóide é a mais comumente usada na construção de redes neurais artificiais. É definida como uma função estritamente crescente que tem propriedade assintótica e suavidade. Um exemplo de sigmóide é a função logística, definida por:

$$\varphi(v) = \frac{1}{1+e^{-av}} \tag{2.8}$$

onde **a** é o parâmetro de inclinação da sigmóide. Pela variação do parâmetro **a**, obtêm-se funções sigmóides de diferentes inclinações, como ilustrado na figura 2.7. Nota-se também que a função sigmóide é diferenciável, enquanto a função limiar não é.

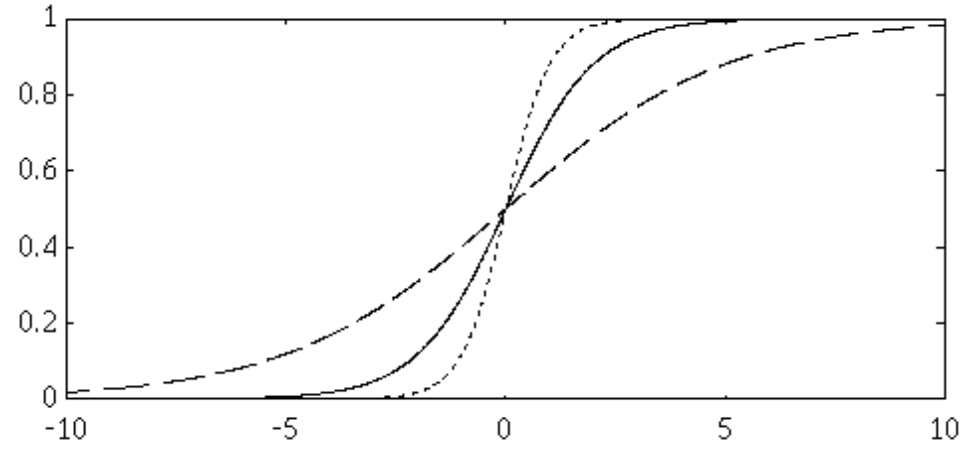

Fig. 2.7 - Função sigmóide.

Em alguns casos é desejável ter-se uma função de ativação válida no intervalo [-1, +1], em que a função de ativação assuma uma forma antissimétrica em relação à origem. Especificamente, a função de limiar é redefinida como:

$$\varphi(v) = \begin{cases} 1, & \text{se } v > 0 \\ 0, & \text{se } v = 0 \\ -1, & \text{se } v < 0 \end{cases} \tag{2.9}$$

que é comumente referida como função sinal. Para uma sigmóide pode-se então usar a função tangente hiperbólica, definida por:

$$\varphi(v) = \tanh\left(\frac{v}{2}\right) = \frac{1-e^{-av}}{1+e^{-av}} \tag{2.10}$$

4) Função de base radial: caracteriza-se por apresentar uma resposta que cresce, ou decresce, monotonicamente com a distância referente a um ponto central. Uma função de base radial decrescente típica é a função gaussiana, figura 2.8, que é dada por:

$$\varphi(v) = e^{\left(-\frac{(v-c)^2}{r^2}\right)} \tag{2.11}$$

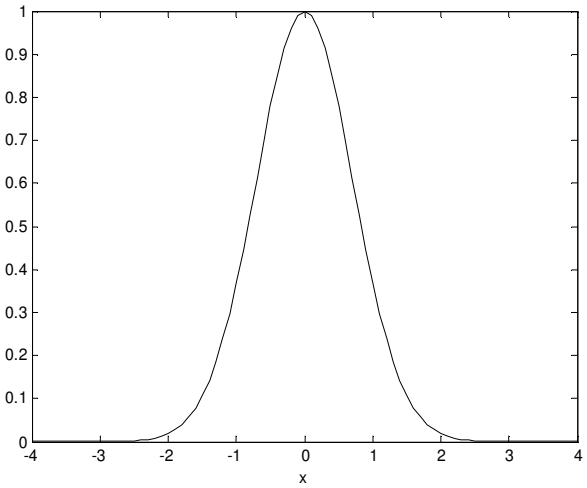

Fig. 2.8 – Função gaussiana

Outra função de base radial típica, porém crescente, é a função multiquádrica, figura 2.9, dada por:

$$\varphi(v) = \frac{\sqrt{r^2 + (v-c)^2}}{r} \tag{2.12}$$

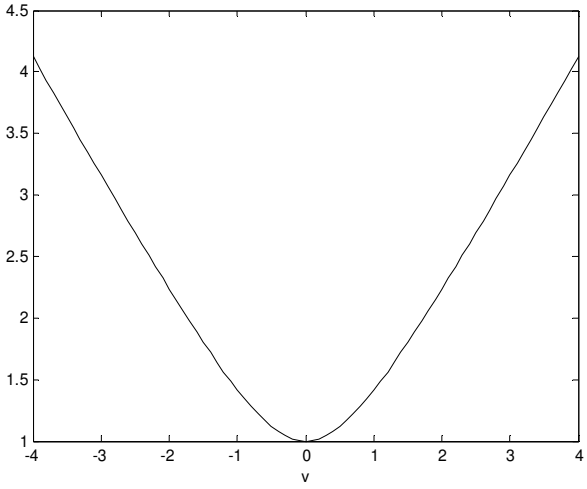

Fig. 2.9 – Função multiquádrica

2.3.4 - Arquitetura de Redes

A maneira pela qual os neurônios de uma rede neural são estruturados está fortemente relacionada ao algoritmo de aprendizagem usado para treiná-la.

Em geral, pode-se identificar quatro diferentes classes de arquitetura de redes:

2.3.4.1 - Rede *Feedforward* de Camada Simples

Na forma mais simples de uma rede neural, organizada em camadas, tem-se somente uma camada de entrada dos nós de origem que projetam os sinais para a camada de saída, mas não vice-versa, isto é, esta rede é estritamente do tipo *feedforward*, ilustrada na figura 2.10, para o caso de quatro nós na camada de entrada e na camada de saída, esta rede é chamada rede de camada simples, com a designação "camada simples" referindo-se a camada de saída dos nós de computação (neurônios).

Uma memória associativa linear é o exemplo de uma rede neural de camada simples. Em tal exemplo, a rede associa um padrão de entrada (vetor) com um padrão de saída (vetor), e a informação é armazenada na rede em função das modificações feitas nos seus próprios pesos sinápticos.

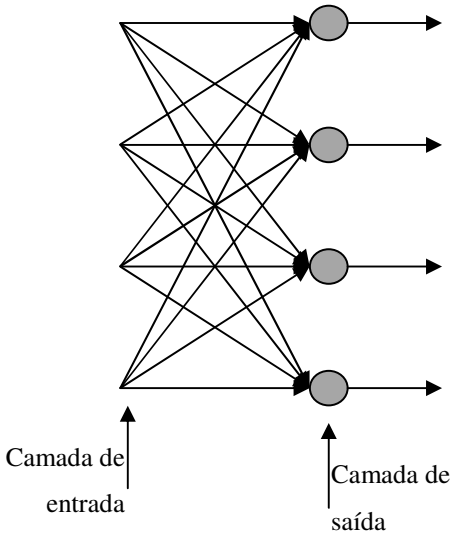

Fig. 2.10 - Rede alimentada com uma camada de neurônios.

2.3.4.2 - Rede *Feedforward* Multicamadas

A segunda classe de redes neurais *feedforward* distingue-se da primeira pela presença de uma ou mais camadas escondidas, cujos nós de computação são chamados de neurônios escondidos ou unidades escondidas. A função dos neurônios escondidos é aumentar a capacidade da rede de aprender novos padrões, ou seja, aumentar o número de estados que a rede pode assumir. Pela adição de uma ou mais camadas escondidas, a rede tem aumentada sua capacidade de extrair estatísticas de alta ordem para adquirir uma perspectiva global, apesar das suas conectividades locais, em virtude do conjunto extra de conexões sinápticas, e da dimensão extra de interações neurais. A habilidade dos neurônios escondidos extraírem estatísticas de alta ordem é particularmente valiosa quando o tamanho da camada de entrada for grande. Esta arquitetura é mostrada na figura 2.11, para o caso de uma camada escondida.

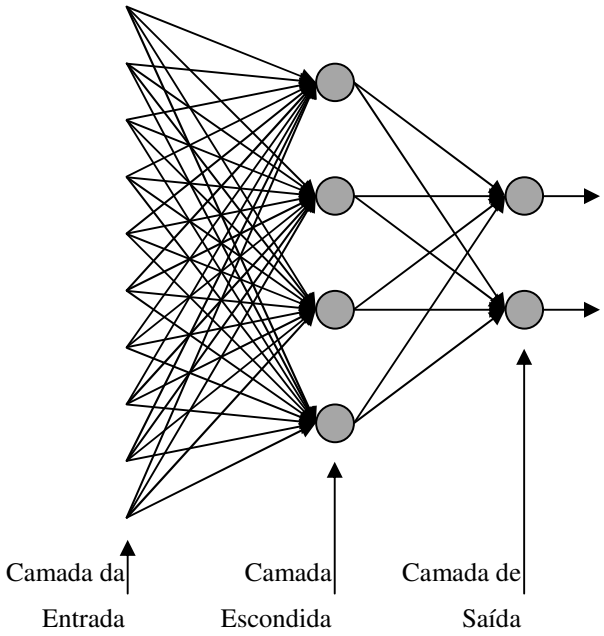

Fig. 2.11 - Rede completamente conectada com uma camada escondida e uma camada de saída.

A rede neural mostrada na figura 2.11 é dita ser completamente conectada, no sentido de que todo nó de cada camada é conectado a todos outros nós da camada diretamente adjacente. Se, entretanto, alguma das conexões sinápticas for perdida, diz-se que a rede estará parcialmente conectada. Uma forma parcialmente conectada de rede neural alimentada por multicamadas a partir de algum interesse particular é uma rede localmente conectada, conforme mostrado na figura 2.12.

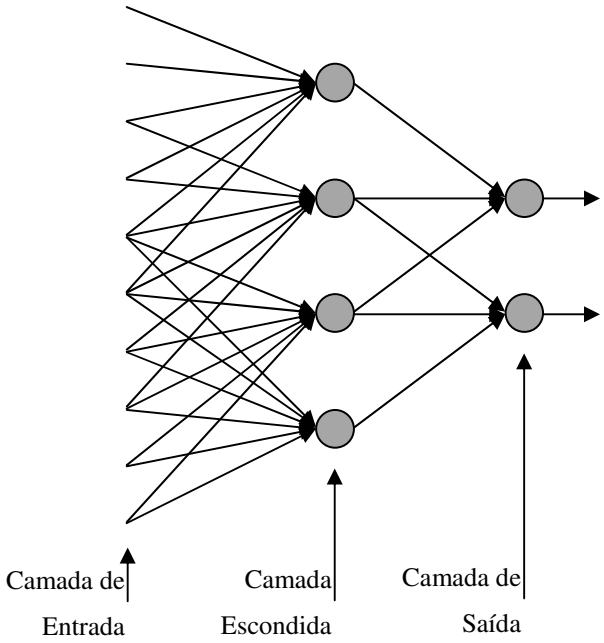

Fig. 2.12 - Rede parcialmente conectada.

2.3.4.3 - Redes Recorrentes

Uma rede neural recorrente distingue-se de uma rede neural alimentada pela presença de pelo menos um laço de realimentação. Por exemplo, uma rede recorrente pode consistir de uma camada simples de neurônios, com cada neurônio realimentando sua saída de volta para as entradas de todos os outros neurônios, como ilustrado na figura 2.13.

Na figura 2.14, mostra-se outro tipo de rede recorrente com neurônios escondidos. As conexões de realimentação mostradas nesta figura originam-se não somente nos neurônios escondidos como também nos neurônios de saída.

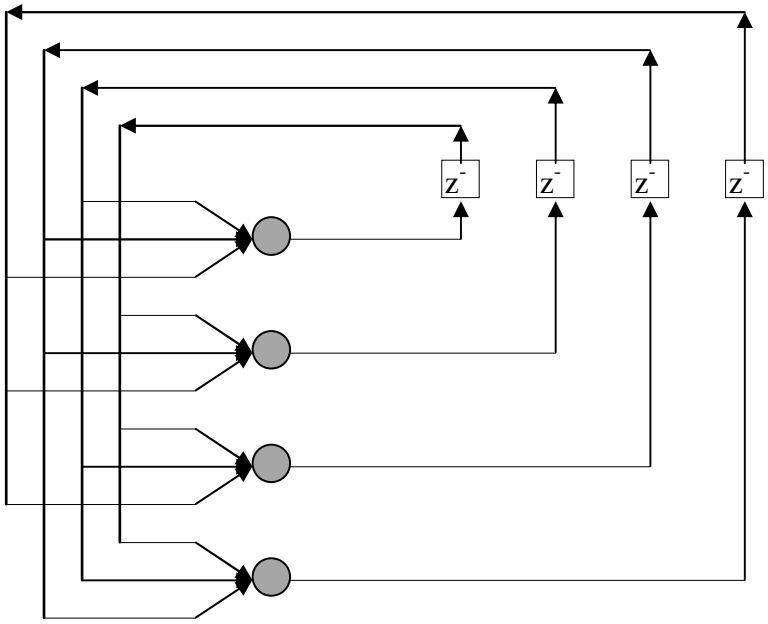

Fig. 2.13 - Rede recorrente com laços não auto-realimentados e sem neurônios escondidos.

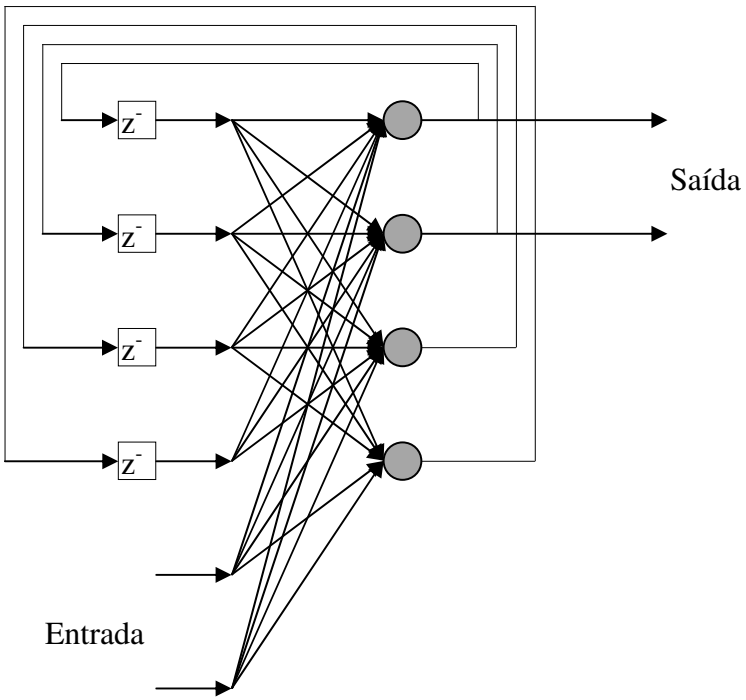

Fig. 2.14 - Rede recorrente com neurônios escondidos.

2.3.4.4 - Estrutura Treliça

Uma treliça consiste de um vetor de neurônios unidimensional, bidimensional, ou com dimensão maior, com um correspondente conjunto de nós de origem que fornecem os sinais de entrada para o vetor. A dimensão da treliça refere-se à dimensão do espaço no qual a estrutura encontra-se.

A figura 2.15 descreve uma treliça unidimensional de três neurônios alimentados de uma camada de três nós de origem, enquanto que a figura 2.16 descreve uma treliça bidimensional de 3×3 neurônios alimentados por uma camada de três nós de origem. Uma rede treliça é uma rede alimentada com os neurônios de saída arranjados em linhas e colunas.

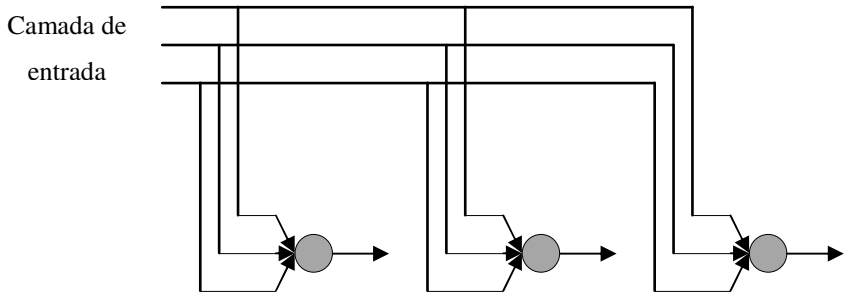

Fig. 2.15 - Treliça unidimensional com 3 neurônios.

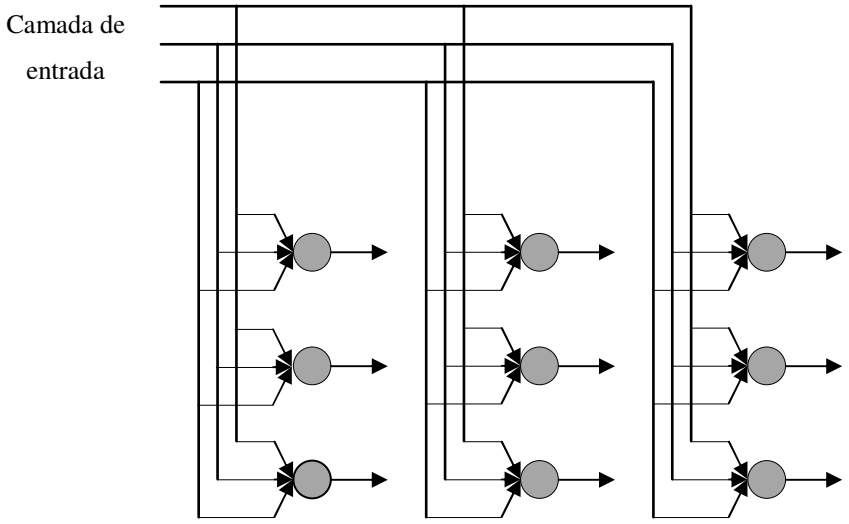

Fig. 2.16 - Treliça bidimensional de 3×3 neurônios.

2.3.5 - Derivação do Algoritmo da Retropropagação do Erro

Para que uma rede neural possa representar um determinado conhecimento, é necessário que algo possa "ensinar" para ela a melhor forma de organizar este conhecimento. O algoritmo de aprendizagem faz este papel de "professor" da rede neural, e monitora o quanto a rede aprendeu, e, se necessário, faz as mudanças adequadas nos seus pesos sinápticos. Este tipo de aprendizado é conhecido como aprendizado supervisionado.

O aprendizado supervisionado mais conhecido e utilizado é o algoritmo da retropropagação do erro ou algoritmo *backpropagation*, (Rumelhart e Willians, 1986).

O algoritmo *backpropagation* propaga o erro da saída para a entrada fazendo com que os pesos sinápticos sejam corrigidos à medida que o aprendizado ocorre. Normalmente, os pesos sinápticos são corrigidos a cada iteração, ou após um determinado número de iterações. Neste trabalho usa-se o primeiro caso.

A seguir é descrito e desenvolvido o algoritmo *backpropagation*.

2.3.5.1 - Algoritmo *Backpropagation*

O sinal da saída do neurônio j na iteração n é definido por

$$e_j(n) = d_j(n) - y_j(n) \tag{2.13}$$

Define-se o valor instantâneo do quadrado do erro para o neurônio j como $\frac{1}{2}e_j^2(n)$. Correspondentemente, o valor instantâneo $\xi(n)$ da soma dos quadrados dos erros é obtido pela soma dos $\frac{1}{2}e_j^2(n)$ sobre todos os neurônios da camada de saída. Estes são somente os neurônios visíveis para os quais o sinal do erro é calculado. A soma instantânea do quadrado dos erros da rede é escrita como:

$$\xi(n) = \frac{1}{2}\sum_{j \in C} e_j^2(n) \tag{2.14}$$

onde o conjunto C inclui todos os neurônios na camada de saída da rede. Denota-se como N o número total de padrões (exemplos) contidos no conjunto de treinamento. O erro quadrático médio é obtido pela soma de $\xi(n)$ sobre todo n, e então normalizado em função de N, como descrito por:

$$\xi_{AV} = \frac{1}{N}\sum_{n=1}^{N}\xi(n) \tag{2.15}$$

A soma instantânea dos erros quadráticos $\xi(n)$ e, portanto, a média quadrática do erro ξ_{AV}, é uma função de todos os parâmetros livres da rede. Para um dado conjunto de treinamento, ξ_{AV} representa a função custo como medida da performance de aprendizado do conjunto de treinamento. O objetivo do processo de aprendizagem é ajustar os parâmetros livres da rede para minimizar ξ_{AV}. Especificamente, considera-se um método simples de treinamento no qual os pesos são atualizados de padrão em padrão. O ajuste dos pesos é feito de acordo com o respectivo erro computado para cada padrão apresentado à rede. Considera-se a figura 2.17, na qual é mostrado um neurônio j sendo alimentado por um conjunto de sinais produzidos pelos neurônios à sua esquerda.

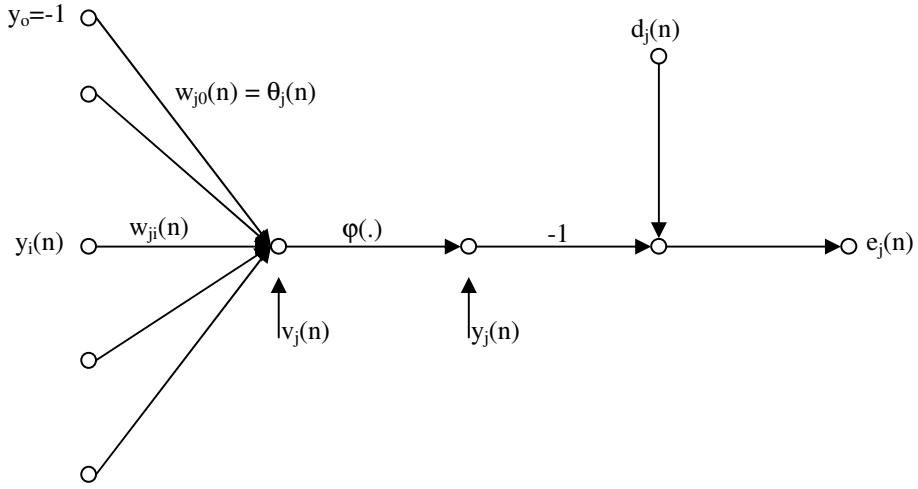

Fig. 2.17 - Gráfico de fluxo de sinal mostrando detalhes do neurônio de saída j.

O nível de atividade interna da rede $v_j(n)$ produzido na entrada da não linearidade associada com o neurônio j é dado por:

$$v_j(n) = \sum_{i=0}^{p} w_{ji}(n)\, y_i(n) \tag{2.16}$$

onde p é o número total de entradas aplicadas ao neurônio j. O peso sináptico w_{j0} é igual ao limiar θ_j aplicado. Por isso a função sinal $y_j(n)$ que aparece na saída do neurônio j na iteração n é:

$$y_j(n) = \varphi_j(v_j(n)) \tag{2.17}$$

De modo similar ao algoritmo dos mínimos quadrados recursivos, o algoritmo da retropropagação do erro aplica uma correção $\Delta w_{ji}(n)$ ao peso sináptico $w_{ji}(n)$, que é proporcional ao gradiente instantâneo $\frac{\partial \xi(n)}{\partial w_{ji}(n)}$. De acordo com a regra da cadeia da derivação, pode-se expressar este gradiente como:

$$\frac{\partial \xi(n)}{\partial w_{ji}(n)} = \frac{\partial \xi(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial w_{ji}(n)} \tag{2.18}$$

O gradiente $\frac{\partial \xi(n)}{\partial w_{ji}(n)}$ representa um fator de sensibilidade, determinando a direção de procura no espaço dos pesos para o peso sináptico w_{ji}.

Diferenciando ambos os lados da equação (2.14) em função de $e_j(n)$, tem-se:

$$\frac{\partial \xi(n)}{\partial e_j(n)} = e_j(n) \tag{2.19}$$

Diferenciando-se ambos os lados da equação (2.13) em função de $y_j(n)$, tem-se:

$$\frac{\partial e_j(n)}{\partial y_j(n)} = -1 \tag{2.20}$$

Diferenciando-se a equação (2.17) em função de $v_j(n)$, tem-se:

$$\frac{\partial y_j(n)}{\partial v_j(n)} = \varphi_j'\left(v_j(n)\right) \tag{2.21}$$

onde o uso do apóstrofo indica a diferenciação.

Finalmente, diferenciando-se a equação (2.16) em função de $w_{ji}(n)$, tem-se:

$$\frac{\partial v_j(n)}{\partial w_{ji}(n)} = y_i(n) \tag{2.22}$$

Substituindo-se (2.19) a (2.22) em (2.18), obtém-se:

$$\frac{\partial \xi(n)}{\partial w_{ji}(n)} = -e_j(n)\varphi_j'\left(v_j(n)\right) y_i(n) \tag{2.23}$$

A correção $\Delta w_{ji}(n)$ aplicada a $w_{ji}(n)$ é definida pela regra delta como:

$$\Delta w_{ji}(n) = -\eta \frac{\partial \xi(n)}{\partial w_{ji}(n)} \tag{2.24}$$

onde η é uma constante que determina a taxa de aprendizagem, chamada de parâmetro de taxa de aprendizagem do algoritmo da retropropagação do erro. O uso do sinal negativo na equação (2.24) é devido à característica de gradiente descendente no espaço dos pesos. O uso da equação (2.23) em (2.24) fornece:

$$\Delta w_{ji}(n) = \eta \delta_j(n) y_i(n) \tag{2.25}$$

onde o gradiente local $\delta_j(n)$ é definido por:

$$\delta_j(n) = -\frac{\partial \xi(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} = e_j(n) \varphi_j'(v_j(n)) \tag{2.26}$$

O gradiente local aponta para as mudanças requeridas nos pesos sinápticos. De acordo com a equação (2.26), o gradiente local $\delta_j(n)$ para a saída do neurônio j é igual ao produto do correspondente sinal de erro $e_j(n)$ e da derivada $\varphi_j'(v_j(n))$ da função de ativação associada.

Na equação (2.25) e (2.26) nota-se que o fator chave envolvido no cálculo do ajuste do peso $\Delta w_{ji}(n)$ é o sinal do erro $e_j(n)$ na saída do neurônio j. Neste contexto, pode-se identificar dois casos distintos, dependendo da localização do neurônio j na rede.

2.3.5.2 - Caso I: O neurônio j é um nó de saída

Quando o neurônio j está localizado na camada de saída da rede, pode ser fornecida a resposta desejada para o próprio neurônio. Para isso pode-se computar o sinal do erro $e_j(n)$ associado a este neurônio, sendo que uma vez determinado $e_j(n)$, é fácil computar o gradiente local $\delta_j(n)$ usando-se a equação (2.26).

2.3.5.3 - Caso II: O neurônio j é um nó da camada escondida

Quando o neurônio j está localizado em uma camada escondida da rede, não há uma resposta especifica desejada para aquele neurônio. O sinal do erro para um neurônio deste tipo pode ser determinado recursivamente em termos do sinal de erro de todos os neurônios que estão diretamente conectados a ele, ou seja, à sua direita. Neste caso, o desenvolvimento do algoritmo da retropropagação do erro se torna mais complexo.

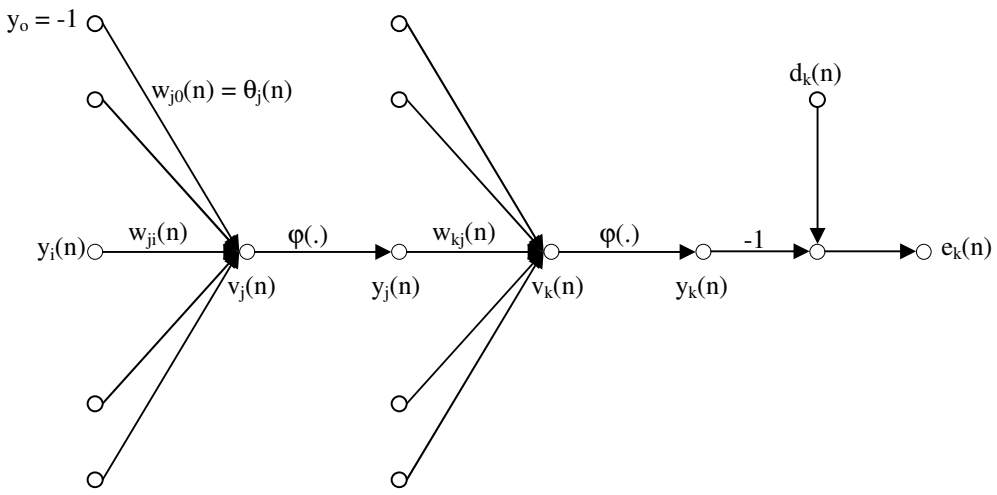

Fig. 2.18 - Gráfico de fluxo de sinal mostrando o neurônio de saída k conectado ao neurônio j da camada escondida.

Considerando-se a situação ilustrada na figura 2.18, pode-se redefinir o gradiente local $\delta_j(n)$ para o neurônio escondido j como:

$$\delta_j(n) = -\frac{\partial \xi(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \tag{2.27}$$

onde fez-se uso da equação (2.21). Para calcular a derivada parcial $\frac{\partial \xi(n)}{\partial y_j(n)}$, pode-se proceder como segue. Da figura 2.18, vê-se que:

$$\xi(n) = \frac{1}{2} \sum_{k \in C} e_k^2(n) \tag{2.28}$$

sendo (2.28) uma reescrita da equação (2.14), exceto pelo uso do índice k no lugar do índice j. Observe que o neurônio k está sendo considerado um nó de saída.

Diferenciando-se a equação (2.28) em relação a $y_j(n)$, obtém-se:

$$\frac{\partial \xi(n)}{\partial y_j(n)} = \sum_k e_k \frac{\partial e_k(n)}{\partial y_j(n)} \tag{2.29}$$

Usando-se a regra da cadeia para a derivada parcial $\frac{\partial e_k(n)}{\partial y_j(n)}$, e escrevendo a equação (2.29) na forma equivalente, tem-se:

$$\frac{\partial \xi(n)}{\partial y_j(n)} = \sum_k e_k \frac{\partial e_k(n)}{\partial v_k(n)} \frac{\partial v_k(n)}{\partial y_j(n)} \tag{2.30}$$

Contudo, pela figura 2.18 nota-se que:

$$e_k(n) = d_k(n) - y_k(n) = d_k(n) - \varphi_k\left(v_k(n)\right) \tag{2.31}$$

Por isso,

$$\frac{\partial e_k(n)}{\partial v_k(n)} = -\varphi_k'\left(v_k(n)\right) \tag{2.32}$$

Nota-se também da figura 2.18 que para o neurônio k, o nível de ativação interno da rede é dado por:

$$v_k(n) = \sum_{j=0}^{q} w_{kj}(n) y_j(n) \tag{2.33}$$

onde q é o número total de entradas aplicadas ao neurônio k. Diferenciando-se a equação (2.33) em função de $y_j(n)$, obtém-se:

$$\frac{\partial v_k(n)}{\partial y_j(n)} = w_{kj}(n) \tag{2.34}$$

Portanto, usando as equações (2.32) e (2.34) em (2.30), encontrando-se a derivada parcial desejada:

$$\frac{\partial \xi(n)}{\partial y_j(n)} = -\sum_k e_k(n)\varphi_k'\left(v_k(n)\right) w_{kj}(n) = -\sum_k \delta_k(n) w_{kj}(n) \tag{2.35}$$

onde usou-se a definição do gradiente local $\delta_k(n)$ dado na equação (2.26).

Finalmente, substituindo-se a equação (2.35) em (2.27), obtém-se o gradiente local $\delta_j(n)$ para o neurônio escondido j e, rearranjando os termos, encontra-se que:

$$\delta_j(n) = \varphi_j'\left(v_j(n)\right)\sum_k \delta_k(n) w_{kj}(n) \tag{2.36}$$

O fator $\varphi_j'\left(v_j(n)\right)$ envolvido na computação do gradiente local $\delta_j(n)$, definido na equação (2.36), depende somente da função de ativação associada com o neurônio j. O fator remanescente envolvido nesta computação, isto é, a somatória sobre k, depende de dois conjuntos de termos. O primeiro conjunto de termos, $\delta_k(n)$, requer o conhecimento do sinal erro, $e_k(n)$, para todos aqueles neurônios que estão localizados na camada imediatamente à direita do neurônio escondido j, e que estão diretamente conectados a ele. O segundo conjunto de termos, $w_{kj}(n)$, consiste dos pesos sinápticos associados a estas conexões.

2.3.5.4 - Taxa de aprendizagem

Quanto menor o parâmetro de taxa de aprendizagem η, menores serão as mudanças nos pesos sinápticos da rede, e mais suave será a trajetória no espaço de pesos. Esta melhoria, contudo, é conseguida ao custo de uma baixa taxa de aprendizado. Se, por outro lado, deseja-se η muito grande para aumentar a velocidade de aprendizado, isso geralmente resultará em grandes mudanças nos pesos sinápticos, inclusive podendo levar a rede à instabilidade. Um método simples para aumentar a taxa de aprendizado, e ainda diminuir o "perigo" da rede se tornar instável, é modificar a regra delta da equação (2.25) incluindo o termo momento, ou seja:

$$\Delta w_{ji}(n) = \alpha \Delta w_{ji}(n-1) + \eta \delta_j(n) y_i(n) \tag{2.37}$$

onde α usualmente é um número positivo chamado de constante de momento. Ela determina o nível do sinal da realimentação que age sobre $\Delta w_{ji}(n)$, como mostrado na figura 2.19 onde z^{-1} é o operador atraso-unitário. A equação (2.37) é chamada de regra delta generalizada; ela inclui a regra delta como caso especial, ou seja, quando $\alpha = 0$.

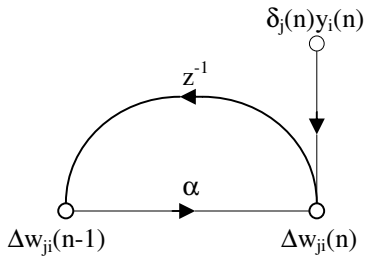

Fig. 2.19 - Gráfico de fluxo de sinal mostrando o efeito da constante momento α.

2.3.6 - Conclusão sobre as redes neurais artificiais

Conforme visto, as redes neurais possuem diversos parâmetros a serem ajustados para poderem operar, ou para aprenderem uma determinada tarefa. Isto normalmente impõe grandes obstáculos para a realização de determinadas tarefas, pois não se têm como construir, de modo genérico, uma rede neural artificial ótima para a resolução de um determinado problema.

O algoritmo *backpropagation* emprega o método do gradiente decrescente, que flui na superfície de erro, ajustando os pesos na direção de um mínimo. Assim, durante o treinamento a

rede pode ficar presa em um desses mínimos locais, não conseguindo chegar ao mínimo global.

A forma de minimizar esses problemas é encontrar o valor adequado para taxa de aprendizado. Se for muito grande demais o aprendizado é rápido, porém corre-se o risco da rede paralisar ou entrar em oscilação sem alcançar o mínimo desejado. Se for colocado muito baixo, aumentam as chances da rede ficar presa em um mínimo local ou, na melhor das hipóteses, o treinamento será muito lento.

Neste trabalho, alguns dos itens apresentados foram pesquisados com maior profundidade, e procurou-se encontrar a melhor solução possível empregando-se os algoritmos genéticos como auxiliares nas buscas, principalmente por melhores arquiteturas e taxas de aprendizagem satisfatórias, para as redes neurais quando aplicadas para o posicionamento de um robô.

Tais estudos são apresentados com mais detalhes no capítulo 4.

CAPÍTULO 3

PLANEJAMENTO DE TRAJETÓRIAS

Neste capítulo os algoritmos genéticos têm o propósito de planejar os diversos estágios de uma trajetória de robô, sendo usado o robô Jeca II como paradigma.

Os algoritmos genéticos são usados para planejar a trajetória no plano cartesiano com um obstáculo presente no curso da menor trajetória, neste caso uma reta.

O planejamento da trajetória é dividido em dois estágios: posicionamento inicial e posicionamento incremental. O posicionamento inicial possui a finalidade de posicionar a ferramenta terminal do robô no ponto inicial da trajetória, enquanto que o posicionamento incremental determina qual o próximo ponto da trajetória que deverá ser seguido, (Monteiro e Madrid, 1999).

3.1 - Modelo cinemático direto do robô Jeca II

O modelo apresentado abaixo representa o robô Jeca II, que possui 5 graus de liberdade e foi construído no Laboratório de Sistemas Modulares Robóticos do Departamento de Sistemas e Controle de Energia da Faculdade de Engenharia Elétrica e Computação da UNICAMP. Este robô pode ser visualizado na figura 3.1.

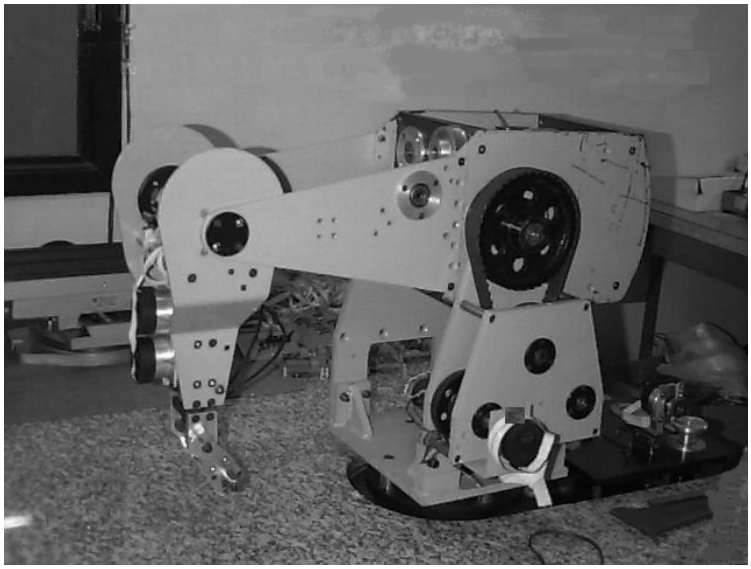

Fig. 3.1 – Robô Jeca II.

Devido à geometria básica do robô Jeca II, os eixos de cada elo podem ser representados através dos parâmetros adotados pela convenção de Denavit-Hatenberg (D-H), (Denavit e Hartenberg, 1955), conforme mostrado na tabela 3.1. A figura 3.2 mostra as posições em que os parâmetros de D-H foram obtidos.

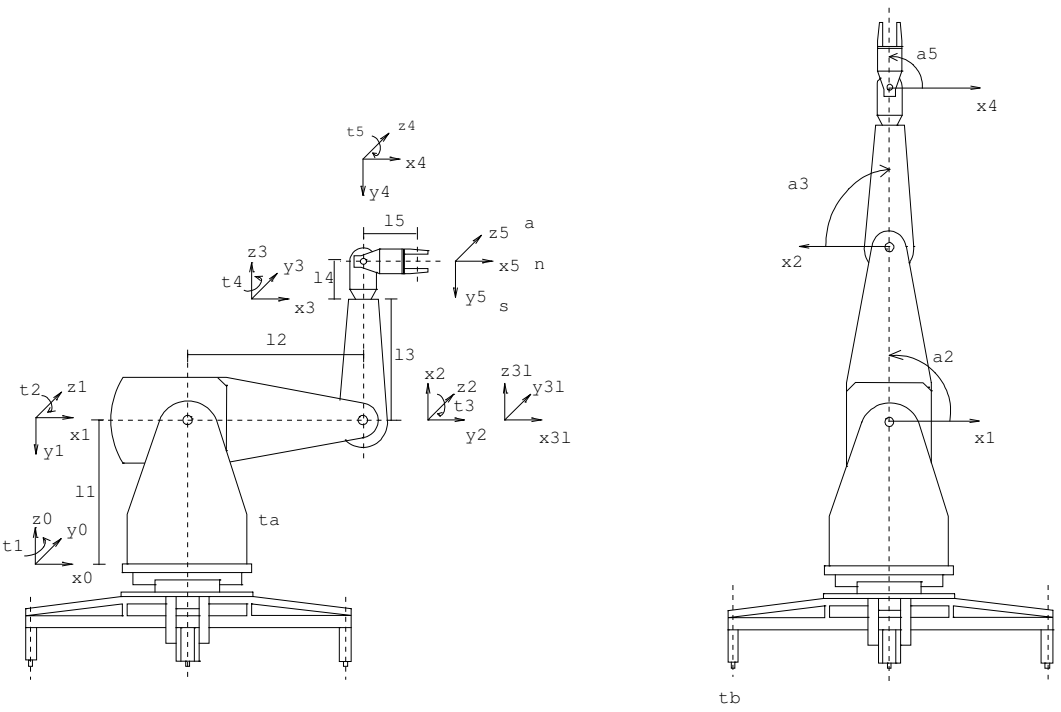

Fig. 3.2 – Fixação dos *frames* para a obtenção dos parâmetros de D-H do robô Jeca II.

Link i	θ_i	α_i	a_i	d_i
1	θ_1	-90°	0	L_1
2	θ_2-90°	0°	l_2	0
3	θ_3+90°	90°	0	0
3’	0°	0°	0	L_3
4	θ_4	-90°	0	L_4
5	θ_5-90°	0°	l_5	0

Tabela 3.1 – Parâmetros D-H do robô Jeca II

Note que algumas considerações foram feitas em função dos referenciais de medidas, levando em consideração o posicionamento dos sensores na estrutura. Considera-se que as medidas de θ_2 são feitas em relação à vertical (direção e sentido do eixo Z_0 do sistema de referência inercial, ou seja, linha que passa pelo centro do primeiro elo), assim θ_2 é considerado como θ_2 - 90º. As medidas de θ_3 são feitas em relação a linha que passa pelo centro do segundo elo, assim o ângulo θ_3 é considerado como θ_3 + 90º. Finalmente, as medidas de θ_5 são feitas em relação a linha que passa pelo centro do quarto elo, assim θ_5 é considerado como θ_5 – 90º.

Considerando os parâmetros D-H do Jeca II mostrados na tabela 1 e sabendo que a matriz de transformação homogênea entre elos sucessivos de um robô com cadeia cinemática serial é dada por:

$$ {}^{i-1}A_i = \begin{bmatrix} C(\theta_i) & -C(\alpha_i).S(\theta_i) & S(\alpha_i).S(\theta_i) & a_i C(\theta_i) \\ S(\theta_i) & C(\alpha_i).C(\theta_i) & -S(\alpha_i).C(\theta_i) & a_i S(\theta_i) \\ 0 & S(\alpha_i) & C(\alpha_i) & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.1} $$

Para validar-se o emprego dos algoritmos genéticos no planejamento de trajetória, somente os três primeiros graus de liberdade são usados para o rastreamento em posição da trajetória. Portanto, a matriz de transformação homogênea é determinada através da equação 3.2.

$$ A = {}^0A_{3'} = {}^0A_1.{}^1A_2.{}^2A_3.{}^3A_{3'} \tag{3.2} $$

onde,

$$ {}^0A_1 = \begin{bmatrix} C(\theta_1) & 0 & -S(\theta_1) & 0 \\ S(\theta_1) & 0 & C(\theta_1) & 0 \\ 0 & -1 & 0 & l_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad {}^1A_2 = \begin{bmatrix} S(\theta_2) & C(\theta_2) & 0 & l_2.S(\theta_2) \\ -C(\theta_2) & S(\theta_2) & 0 & -l_2.C(\theta_2) \\ 0 & 0 & 1 & l_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

$$ {}^2A_3 = \begin{bmatrix} -S(\theta_3) & 0 & C(\theta_3) & 0 \\ C(\theta_3) & 0 & S(\theta_3) & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad {}^3A_{3'} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & l_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.3} $$

Aplicando-se o sistema de equações (3.3) em (3.2), tem-se

$$A = {}^{0}A_{3'} = \begin{bmatrix} C(\theta_1).C(\theta_2+\theta_3) & -S(\theta_1) & C(\theta_1).S(\theta_2+\theta_3) & (l_2.S(\theta_2)+l_3.S(\theta_2+\theta_3)).C(\theta_1) \\ S(\theta_1).C(\theta_2+\theta_3) & C(\theta_1) & S(\theta_1).S(\theta_2+\theta_3) & (l_2.S(\theta_2)+l_3.S(\theta_2+\theta_3)).S(\theta_1) \\ -S(\theta_2+\theta_3) & 0 & C(\theta_2+\theta_3) & l_1+l_2.C(\theta_2)+l_3.C(\theta_2+\theta_3) \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.4)$$

onde

$C(\theta_i) = \cos(\theta_i)$

$S(\theta_i) = \text{sen}(\theta_i)$

A matriz da equação (3.4), denominada de matriz de transformação homogênea, determina a posição/orientação da ferramenta terminal do robô Jeca II. Em uma trajetória, várias matrizes de transformação homogênea são utilizadas para compor a trajetória, e são denominadas de T_k, onde cada T_k representa uma matriz A (Spong e Vidyasagar, 1989), que rastreia a trajetória desejada.

Como a matriz T_k é determinada em função dos ângulos atuais de cada junta do robô, tem-se como objetivo determinar quais são os ângulos das juntas que geram a matriz mais próxima ou igual a matriz T_k, ou seja, determinar o modelo cinemático inverso. Existem diversas técnicas para calcular este modelo, como pode ser visto em (Fu, Gonzalez e Lee, 1987).

Neste capítulo mostra-se o emprego dos algoritmos genéticos para resolver problemas de planejamento de trajetórias. O objetivo é planejar a trajetória de uma ferramenta terminal de um robô industrial com 5 graus de liberdade, tendo três estágios principais:

a) Planejamento de trajetória com desvio de obstáculo;
b) Posicionamento da ferramenta terminal no início da trajetória;
c) Geração dos ângulos que permitem que o robô execute a trajetória.

3.2 - Planejamento de trajetórias no plano cartesiano com desvio de obstáculo

A primeira tarefa a ser executada no planejamento de trajetórias é gerar um caminho a ser percorrido pela ferramenta terminal do robô dentro do espaço de trabalho. Para gerar esta trajetória, é usada como medida a distância entre o ponto inicial e o ponto final. A menor

distância entre estes dois pontos é um segmento de reta. Porém, caso exista algum tipo de obstáculo entre estes dois pontos, não existe a possibilidade de adotar-se este tipo de trajetória e, portanto, deve-se fazer uso de algum procedimento que gere uma trajetória que o robô possa executar, sem colisão com obstáculos existentes. Neste momento, os algoritmos genéticos são utilizados como uma ferramenta auxiliar na geração de trajetórias.

O espaço de trabalho composto por um ponto inicial, um ponto final e um obstáculo é utilizado para gerar trajetórias. A representação deste espaço de trabalho pode ser vista na figura 3.3, onde as medidas apresentadas nos eixos X e Y estão em milímetros.

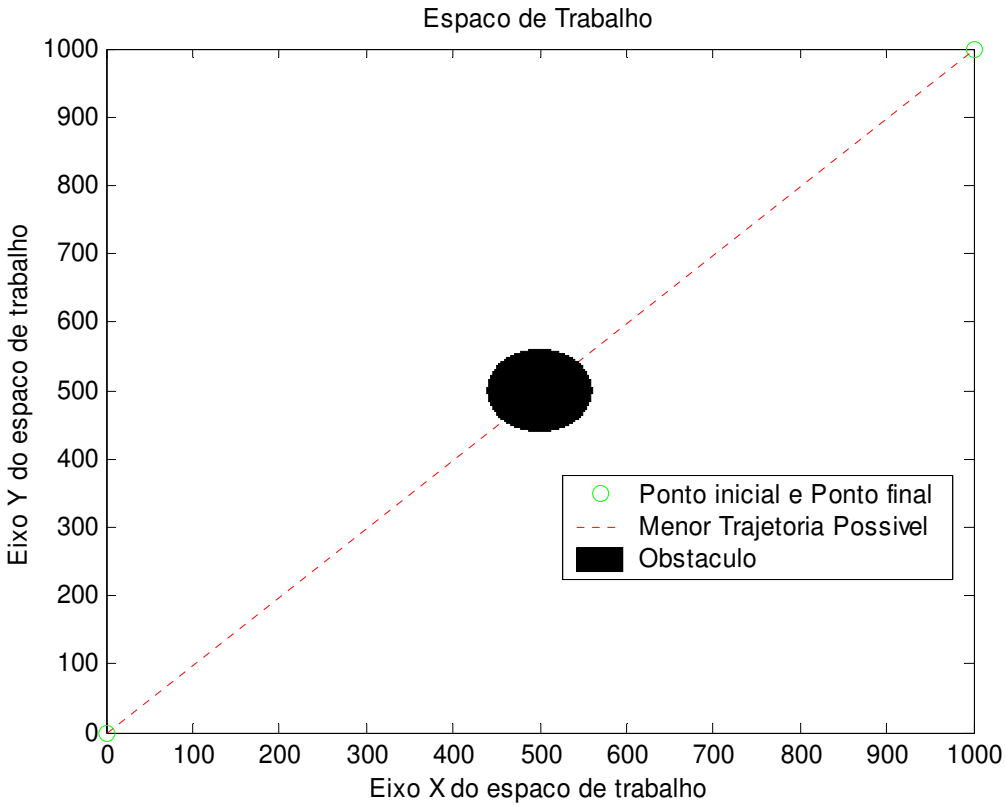

Fig. 3.3 – Exemplo de espaço de trabalho.

Como comentado anteriormente, o problema a ser resolvido é determinar uma trajetória entre dois pontos evitando contato da ferramenta terminal do robô com o obstáculo durante o rastreamento da trajetória. Para resolver este problema, a técnica baseada nos algoritmos genéticos é utilizada, conforme mostrado no fluxograma da figura 3.4. Nesta fase de planejamento, somente a equação 3.4 é usada.

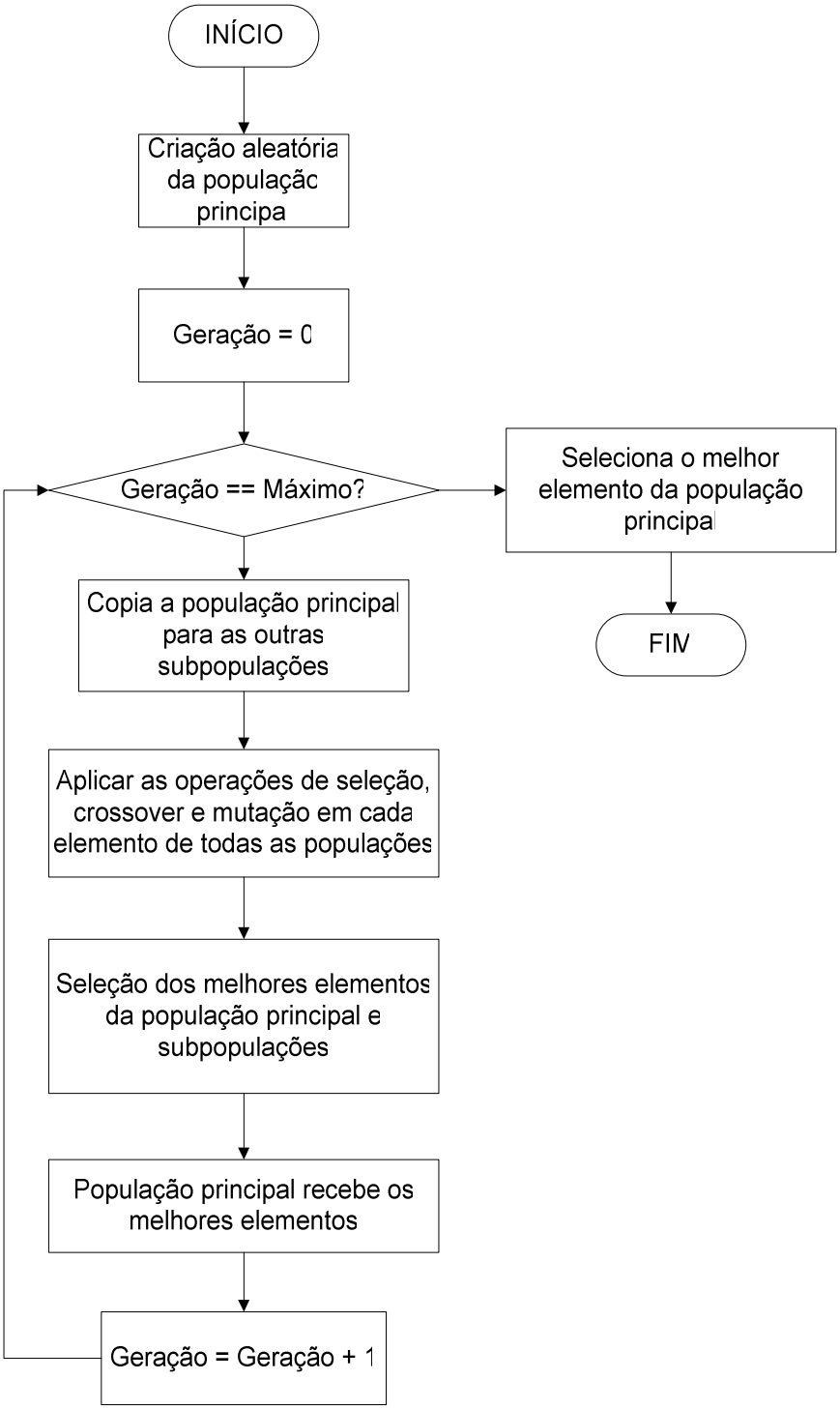

Fig. 3.4 – Fluxograma do algoritmo genético usado para o planejamento de trajetória.

O elemento descrito no fluxograma acima é composto por uma trajetória subdividida em vários segmentos de reta, e por um medidor de qualidade da trajetória chamado de *fitness*, que

mede a distância entre o ponto inicial e final, sem colisão.

As principais operações e características do algoritmo genético mostrado na figura 3.4 são mostradas a seguir.

3.2.1 - Elemento

Uma trajetória composta por um ponto inicial, um ponto final e um ou mais pontos intermediários, representa um elemento neste algoritmo genético. O número de pontos intermediários irá determinar a complexidade do algoritmo genético. Quanto maior o número de pontos a ser tratado maior a complexidade do algoritmo genético. O número de pontos intermediários pode ser determinado em função de cada tarefa, além da flexibilidade que se deseja fornecer ao algoritmo genético.

A população inicial do algoritmo genético deverá ter somente trajetórias válidas. O procedimento usado para se determinar se uma trajetória é válida ou não é verificar se existe sobreposição de pontos comuns entre a trajetória e o círculo que representa o obstáculo. Isto se consegue através da resolução de um sistema de equações lineares entre todos os segmentos de reta e o círculo.

3.2.2 - Função de *fitness*

Para avaliar uma trajetória é usado o valor da distância entre o ponto final e o ponto inicial, sem a colisão de obstáculos, tendo como objetivo minimizar esta distância. Porém, a natureza dos algoritmos genéticos é maximizar a performance de uma determinada função de *fitness* e não minimizá-la. Para contornar tal problema, a equação 3.5 é usada.

$$\text{Fitness}(\text{elemento}) = 1.2 \cdot (\text{NPontos} - 1) \cdot \sqrt{(\text{max\_x} - \text{min\_x})^2 + (\text{max\_y} - \text{min\_y})^2} - \text{distância} \quad (3.5)$$

onde,

Fitness(elemento): função de *fitness* do elemento.

NPontos: número de pontos que contém a trajetória.

max_x: máximo valor do eixo x.

min_x: mínimo valor do eixo x.

max_y: máximo valor do eixo y.

min_y: mínimo valor do eixo y.

distância: valor do comprimento da trajetória.

3.2.3 - Seleção

A operação de seleção é executada baseada no método *roullete-wheel*, ou seja, o elemento que tiver melhor performance possui maior probabilidade de ser selecionado para a operação de *crossover*.

A constante de valor 1.2 na equação 3.5 é usado para que o pior elemento da população tenha chances de ser escolhido, não determinando para o mesmo uma probabilidade de seleção igual a zero.

3.2.4 - *Crossover*

O *crossover* é realizado através da troca dos pontos intermediários, determinados aleatoriamente, entre duas trajetórias, selecionados pela operação de seleção comentada acima. Está operação é executada somente se a probabilidade de *crossover* for satisfeita. Esta operação é mostrada na figura 3.5. Caso a trajetória derivada desta operação não for uma trajetória válida, ou seja, passar por cima do obstáculo em questão, esta não será adicionada às subpopulações geradas a partir da geração principal. Novas operações com novas trajetórias serão requisitadas ao algoritmo genético.

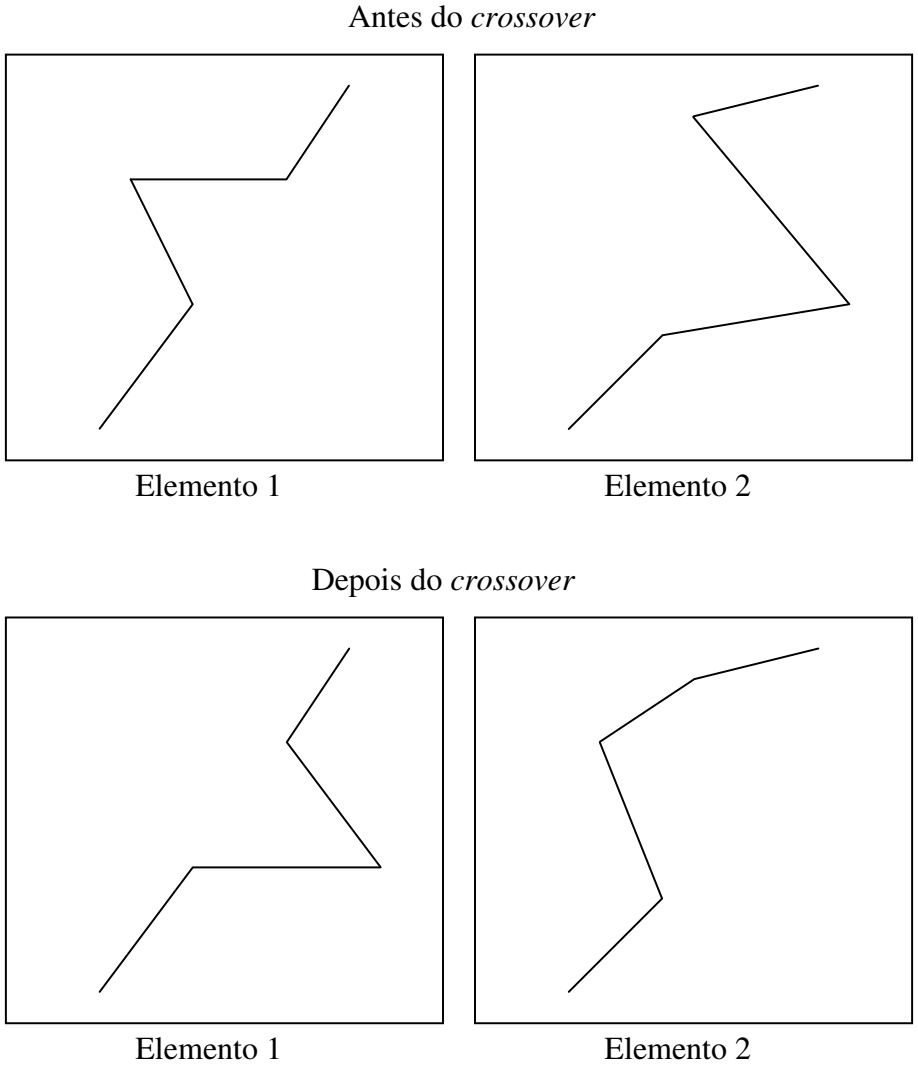

Fig. 3.5. – *Crossover* de trajetória entre dois elementos.

3.2.5 - Mutação

A operação de mutação é executada através do incremento ou decremento de uma coordenada de um ponto intermediário da trajetória, se a probabilidade de mutação for satisfeita. A coordenada do eixo x ou do eixo y, e a respectiva operação de incremento ou decremento, é selecionada aleatoriamente.

3.3 - Planejamento de trajetória no espaço de junta

Nesta etapa de planejamento, o algoritmo genético é implementado para determinar os ângulos das juntas que geram a matriz A que mais se aproxima da matriz T_k. Este processo é dividido em duas etapas:

a) Posicionamento da ferramenta terminal no ponto inicial da trajetória
b) Rastreamento da trajetória

A primeira etapa é executada quando a ferramenta terminal se encontra em um ponto diferente do ponto inicial da trajetória. Este movimento não é considerado parte da trajetória a ser rastreada, portanto, determina-se os ângulos das juntas que correspondem a este ponto inicial, e um controlador de posição é necessário para posicionar a ferramenta terminal do robô no ponto inicial.

A segunda etapa é composta pelo rastreamento contínuo da trajetória. Com isto, o próximo ponto a ser atingido é considerado estar bem próximo do ponto atual. Devido a essa proximidade, somente pequenos incrementos nos ângulos das juntas são necessários para se determinar o próximo ponto.

3.3.1 - Primeiro estágio

O algoritmo genético usado neste estágio é essencialmente o mesmo introduzido anteriormente neste capítulo, mas agora a trajetória a ser rastreada é considerada no espaço de juntas de um robô industrial e não no espaço cartesiano.

Cada ângulo de junta é codificado através de uma *string* binária, e estes ângulos podem assumir valores entre $[-\pi, +\pi]$ radianos. Para obter-se uma precisão de pelo menos 10^{-4} radianos, é usada uma *string* de 16 *bits* para codificar cada ângulo. Tal precisão foi determinada por ser suficiente para rastrear todos os pontos em função dos sensores usados no robô. A decodificação de cada ângulo é feita através da equação 3.6.

$$\hat{a}ngulo = \frac{2\pi \cdot \left[\sum_{i=0}^{Nb-1} string(i) \cdot 2^{Nb-1-i} \right]}{2^{Nb-1} - 1} - \pi \tag{3.6}$$

onde

string(i): i-ésimo *bit* da *string* que codifica o ângulo.

N_b: número de *bits* da *string*.

A equação 3.6 faz a conversão de uma *string* binária em um número decimal, conforme pode ser visto em (Tocci e Widmer, 2003). Após efetuar tal conversão, é efetuada a conversão para a escala em ângulo com a precisão determinada pela quantidade de *bits* da *string*.

A função de *fitness* usada para medir a performance de um elemento do algoritmo genético é baseada no valor máximo da matriz que resulta do módulo da diferença entre a matriz gerada por um elemento do algoritmo genético e a matriz desejada, conforme pode ser visto na equação 3.7.

$$Fitness(elemento) = \max\left(\left| a_{ij} - t_{ij} \right| \right), \quad \text{i, j = 1, ... , 4} \tag{3.7}$$

onde

a_{ij}: elemento ij da matriz atual A.

t_{ij}: elemento ij da matriz desejada T_k.

Como se quer minimizar o erro, a equação 3.7 foi adaptada para funcionar dentro de um algoritmo genético conforme a equação 3.8.

$$Fitness(elemento) = 1.2 \cdot \text{M} - \max\left(\left| a_{ij} - t_{ij} \right| \right), \text{ i,j=1,...,4} \tag{3.8}$$

onde M é o máximo valor de *fitness* encontrado na população principal e subpopulações, utilizando a equação 3.7.

A reprodução é feita através da aplicação de seleção baseada no método do *roullete-wheel*, *crossover* uniforme e mutação simples, com probabilidade de *crossover* de 0,3 e probabilidade de

mutação de 0,1.

3.3.2 - Segundo estágio

Com a ferramenta terminal do robô localizada no ponto inicial da trajetória, o mesmo algoritmo genético pode ser usado para rastrear a trajetória desejada, somente com algumas pequenas mudanças.

Como a trajetória a ser rastreada é considerada contínua, o próximo ponto a ser atingido pela ferramenta terminal do robô pode ser atingido através de pequenos incrementos ou decrementos em relação ao ângulo atual, ou seja:

$$\theta_i = \theta_{i-1} + \Delta\theta \tag{3.9}$$

onde

θ_i: ângulo a ser determinado (radianos).

θ_{i-1}: ângulo atual (radianos).

$\Delta\theta$: pequeno incremento ou decremento (radianos).

Devido ao incremento ou decremento ser pequeno, poucos *bits* na *string* de codificação do ângulo são necessários para representá-los.

O algoritmo genético usado neste estágio é idêntico ao apresentado para o posicionamento inicial, mas a *string* que codifica a variação do ângulo é codificada em 11 *bits*. Esta *string* mapeia os ângulos no intervalo [-0.1, +0.1] radianos, com uma precisão de 10^{-4} radianos.

As operações do algoritmo genético foram consideradas exatamente as mesmas.

3.4 - Aspectos computacionais e resultados

As simulações mostradas nesta seção foram efetuadas utilizando-se um computador do tipo IBM-PC PENTIUM II, 300 MHz e 64 MBytes de RAM. Os programas foram desenvolvidos em linguagem C e Java, e os compiladores usados foram o Borland C++ 5.5 e JDK 1.3.1.

3.4.1 - Planejamento da trajetória no plano

Para ilustrar uma aplicação do algoritmo genético mostrado anteriormente, é utilizado o espaço mostrado na figura 3.6.

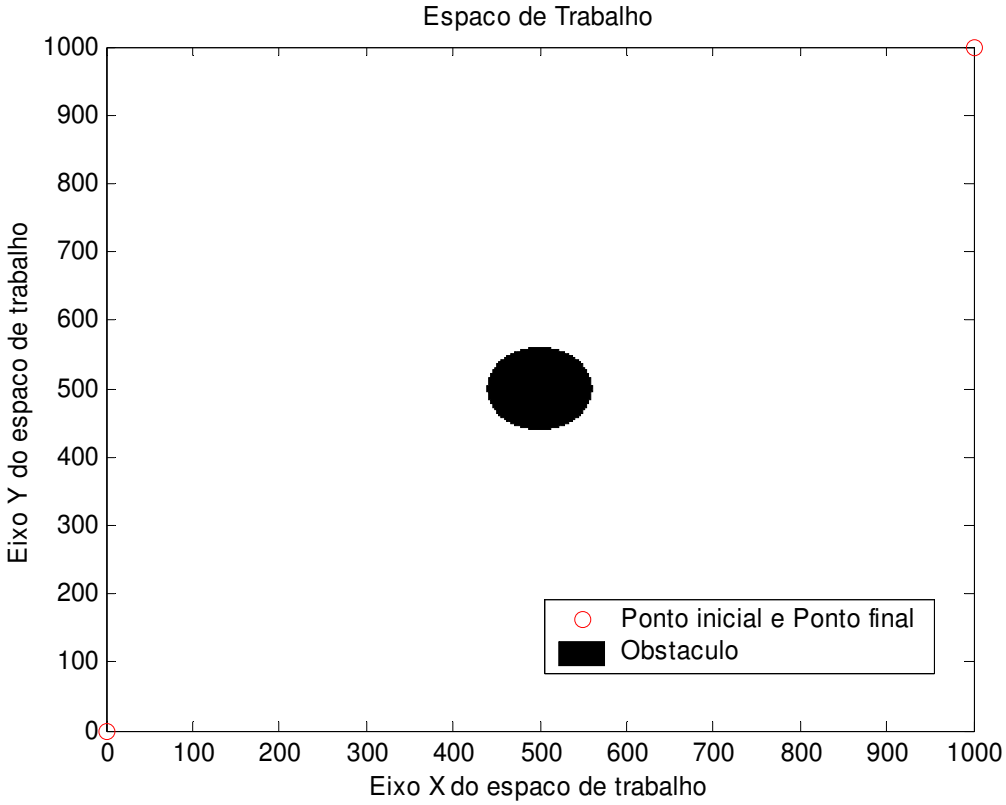

Fig. 3.6 – Trajetória entre dois pontos com obstáculo.

Pode-se observar que a menor distância entre o ponto inicial e o ponto final é uma linha reta. Porém, a existência de um obstáculo não permite que esta trajetória possa ser escolhida como solução.

Durante a execução do algoritmo genético, são criados novos elementos a partir da população principal, que é criada aleatoriamente no início do algoritmo genético. Esses novos elementos são criados em subpopulações através da aplicação das operações de seleção, *crossover* e mutação, como se fosse um processo de reprodução de elementos a partir da população principal inicial.

No exemplo abaixo, uma população de 20 elementos (população principal), criada aleatoriamente, é usada para gerar 3 subpopulações (populações filhas), com probabilidade de

crossover de 0.5 e probabilidade de mutação de 0.3. Cada elemento é composto por uma trajetória dividida em 5 segmentos de retas, figura 3.7.

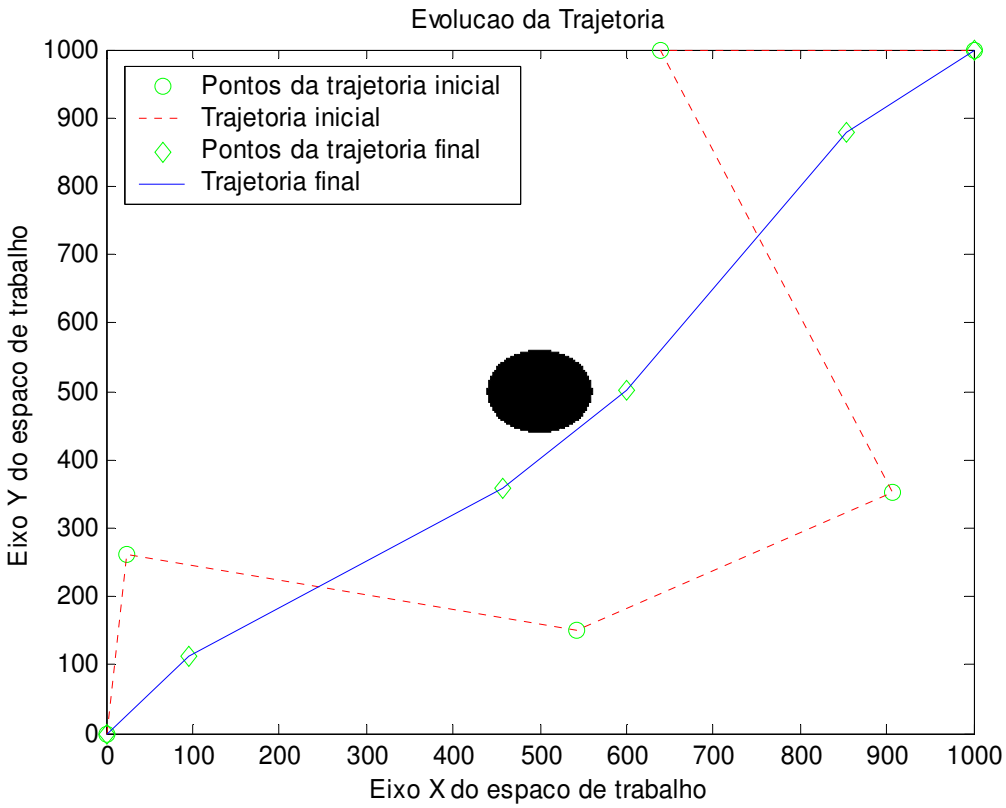

Fig. 3.7 – Trajetória composta por cinco segmentos de retas.

Em vermelho tracejado observa-se a melhor trajetória da primeira população, bem como os pontos que a compõe, representados pelos círculos verdes. A trajetória final é mostrada em azul e os seus pontos em um losângulo verde.

Para que se possa calcular a função de *fitness* é usada a somatória das distâncias dos 5 segmentos de retas, conforme visto na equação (3.10), onde NPontos é igual a 6.

$$Distância(elemento) = \sum_{i=1}^{NPontos-1} \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2} \tag{3.10}$$

A distância que o melhor elemento da população principal apresenta após a reprodução é mostrada na figura 3.8. O ponto inicial da trajetória considerado foi (0.0, 0.0) e o ponto final foi (1000.0, 1000.0).

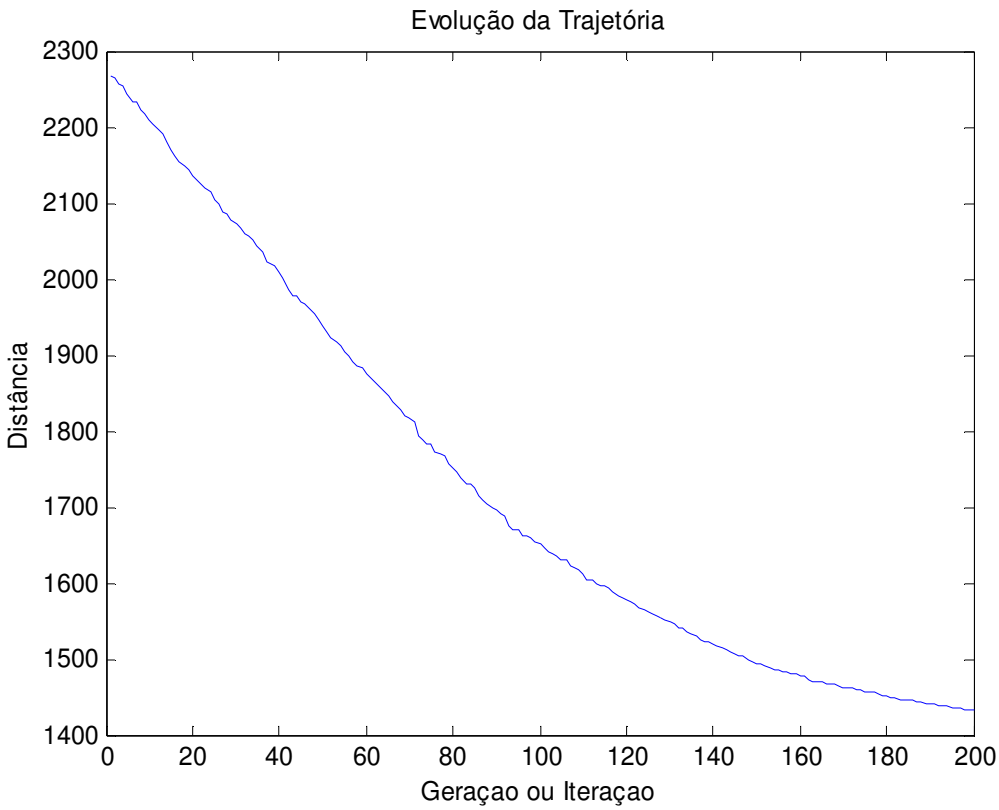

Fig. 3.8 – Evolução da trajetória no plano cartesiano.

O processo apresentado na figura 3.8 foi finalizado após 78,6 x 10^6 ciclos de máquina para a execução de 200 gerações, mas como o algoritmo genético é uma ferramenta que opera com partes aleatórias, este tempo varia para cada execução do algoritmo genético. Os 5 segmentos de reta ficaram definidos através dos pontos (0.0, 0.0), (95.0, 113.0), (457.0, 357.0) , (601.0, 502.0), (854.0, 878.0) e (1000.0, 1000.0).

Percebe-se que não existe retrocesso no processo de aprendizagem, pois não existe acréscimo na distância entre uma geração e a próxima. Isto se deve ao fato do algoritmo genético sempre armazenar na população principal o melhor elemento encontrado que venha a representar a menor distância a ser percorrida.

3.4.2 - Planejamento da trajetória no espaço de juntas

Como descrito anteriormente, o planejamento de trajetórias no espaço de juntas é composto por dois estágios: Posicionamento inicial e posicionamento incremental. Estes estágios são

descritos a seguir.

3.4.2.1 - Posicionamento inicial

Para os testes do algoritmo, a posição final desejada é o ponto cujos ângulos são $\theta_1 = 1.0$ radiano, $\theta_2 = 1.0$ radiano e $\theta_3 = 1.0$ radiano, que possui a matriz de transformação homogênea (MTH) igual a:

$$A = \begin{bmatrix} -0.2248 & -0.8415 & 0.4913 & 304.6834 \\ -0.3502 & 0.5403 & 0.7651 & 474.5162 \\ -0.9093 & 0.0 & -0.4161 & 452.0842 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix} \tag{3.11}$$

A matriz da equação 3.11 é aquela que deverá ser obtida pelo algoritmo genético mostrado anteriormente. As probabilidades de *crossover* e mutação tiveram seus valores definidos empiricamente como 0.3 e 0.05, respectivamente. Para a finalização do algoritmo genético foram utilizadas duas condições, sendo que a primeira é quando se atinge o número máximo de iterações, e a segunda quando se atinge o erro máximo aceitável. Com isto, o número de iterações necessário para se atingir uma solução aceitável pode ser reduzido consideravelmente.

A seguir apresenta-se dois experimentos relacionados as condições descritas acima.

Experimento 1

Neste primeiro experimento, a matriz obtida no final do algoritmo genético foi:

$$A = \begin{bmatrix} -0.224862 & -0.841486 & 0.491263 & 304.693074 \\ -0.350223 & 0.540280 & 0.765142 & 474.559468 \\ -0.909275 & 0.0 & -0.416196 & 451.998385 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix} \tag{3.12}$$

O erro máximo obtido é de 0.085829, onde os ângulos obtidos são iguais a $\theta_1 = 1.000027$ radiano, $\theta_2 = 1.000219$ radiano e $\theta_3 = 0.999835$ radiano. Para se atingir tais valores, foram necessários 4.957 iterações do algoritmo genético.

A evolução do algoritmo genético é mostrado na figura 3.9.

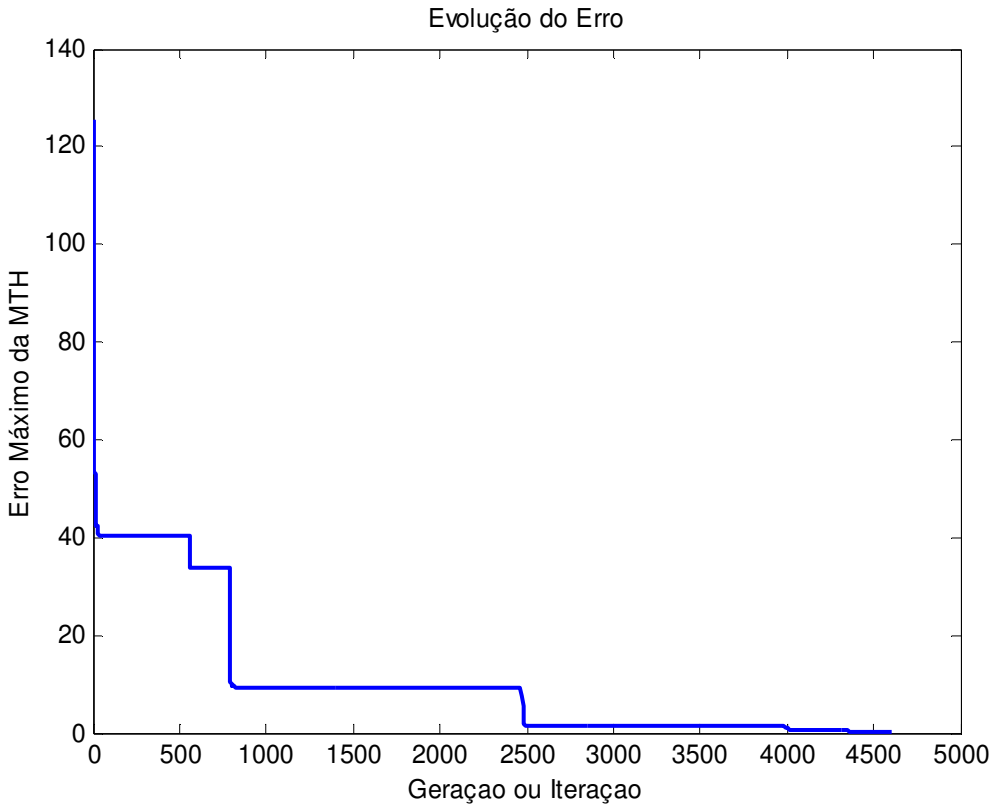

Fig. 3.9 – Evolução do erro para o posicionamento inicial.

Experimento 2

Neste experimento, a matriz de transformação homogênea obtida no final do algoritmo genético foi:

$$A = \begin{bmatrix} -0.224970 & -0.841537 & 0.491125 & 304.586652 \\ -0.350464 & 0.540199 & 0.765088 & 474.493739 \\ -0.909155 & 0.0 & -0.416457 & 452.062063 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix} \quad (3.13)$$

O erro máximo obtido é de 0.096708, onde os ângulos obtidos são iguais a $\theta_1 = 1.000123$ radiano, $\theta_2 = 0.999835$ radiano e $\theta_3 = 1.000506$ radiano. Para este experimento, foram executadas 349 iterações do algoritmo genético.

A evolução do algoritmo genético é mostrado na figura 3.10.

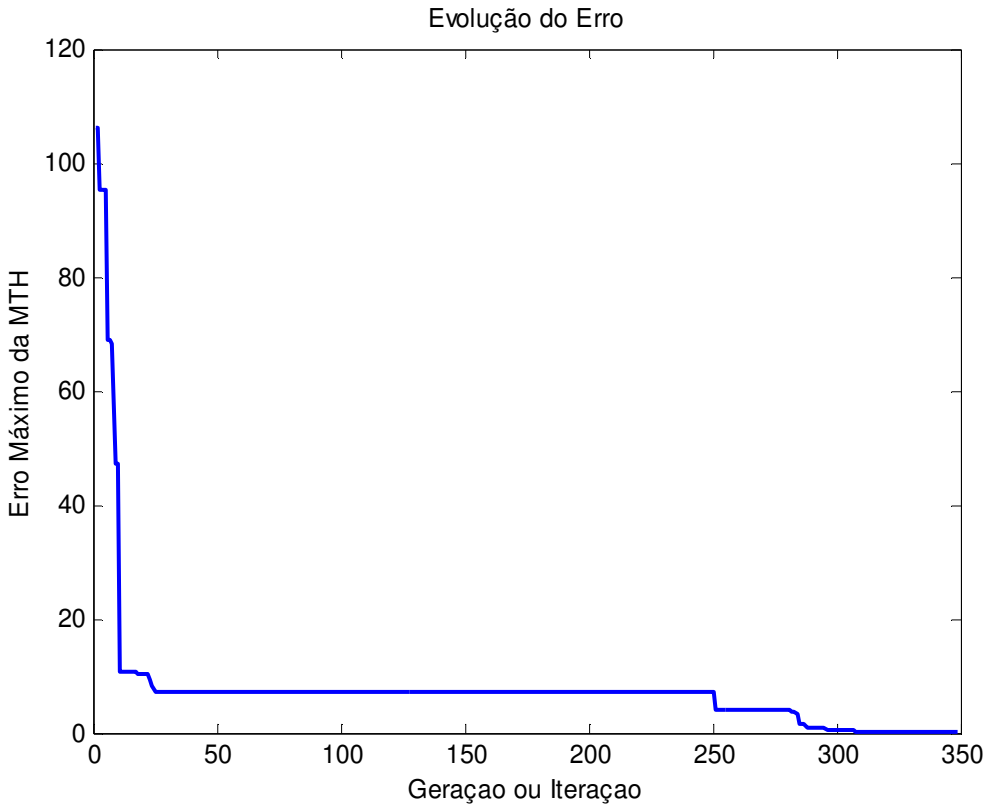

Fig. 3.10 – Evolução do erro para o posicionamento inicial.

Como o algoritmo genético usado para o rastreamento da trajetória é essencialmente o mesmo do planejamento da trajetória, todas as vantagens e desvantagens são praticamente as mesmas, inclusive referente à manutenção do melhor elemento para que não haja retrocesso na otimização do processo. Isto pode ser observado através do decréscimo que ocorre nos gráficos das figuras 3.9 e 3.10. Nota-se também nestas figuras que o número de iterações para se atingir um valor de erro aceitável varia bastante. Isto é uma característica do algoritmo genético, uma vez que utiliza-se propriedades probabilísticas na geração de populações.

O erro dos ângulos permaneceu dentro do esperado, mantendo-se dentro dos 4 dígitos decimais previsto para o posicionamento inicial no item 3.4.1.

3.4.2.2 - Posicionamento incremental

O posicionamento incremental requer que o ponto inicial esteja próximo do ponto desejado. Para a simulação, o ponto inicial escolhido foi o definido pelos ângulos iguais a $\theta_1 = 1.0$ radiano,

θ_2 = 1.0 radiano e θ_3 = 1.0 radiano, enquanto que o ponto desejado é definido pelos ângulos iguais a θ_1 = 1.01 radianos, θ_2 = 1.01 radianos e θ_3 = 1.01 radianos. O ponto inicial possui a matriz de transformação homogênea igual à mostrada na equação 3.14, enquanto que o ponto final possui a matriz de transformação homogênea mostrada na equação 3.15.

$$A_{Atual} = \begin{bmatrix} -0.2248 & -0.8415 & 0.4913 & 304.6834 \\ -0.3502 & 0.5403 & 0.7651 & 474.5162 \\ -0.9093 & 0.0 & -0.4161 & 452.0842 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix} \tag{3.14}$$

$$A_{Desejada} = \begin{bmatrix} -0.2310 & -0.8468 & 0.4791 & 299.9328 \\ -0.3677 & 0.5319 & 0.7628 & 477.5548 \\ -0.9008 & 0.0 & -0.4342 & 444.1822 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix} \tag{3.15}$$

Para a execução do algoritmo genético, as probabilidades de *crossover* e mutação foram consideradas iguais a 0.2 e 0.05, respectivamente. As condições de parada do algoritmo genético para o posicionamento incremental são exatamente as mesmas do algoritmo genético para posicionamento global. Dois experimentos são mostrados a seguir.

Experimento 1

A matriz obtida neste experimento é descrita na equação 3.16, após 159 iterações. Os ângulos obtidos pela execução do algoritmo genético foram: θ_1 = 1.010015 radiano, θ_2 = 1.010210 radiano e θ_3 = 1.009331 radiano, correspondendo a um erro máximo de 0.076057 na matriz de transformação homogênea da equação 3.15.

$$A_{Obtida} = \begin{bmatrix} -0.230734 & -0.846840 & 0.479191 & 299.976033 \\ -0.367388 & 0.531848 & 0.762996 & 477.639186 \\ -0.900993 & 0.0 & -0.433835 & 444.214476 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix} \tag{3.16}$$

O gráfico da figura 3.11 mostra a evolução do algoritmo genético ao longo de sua execução.

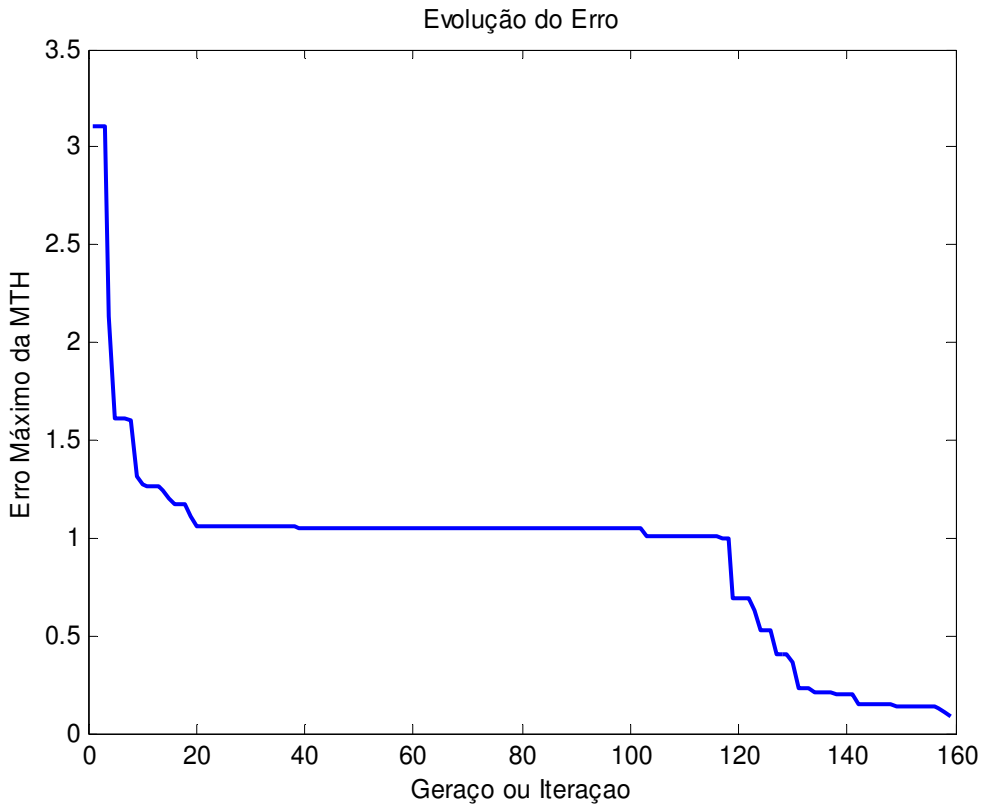

Fig. 3.11 – Evolução do erro no posicionamento incremental.

Experimento 2

Após 39 iterações, a matriz de transformação homogênea da equação 3.17 foi obtida. Os ângulos obtidos pela execução do algoritmo genético foram: θ_1 = 1.010112 radiano, θ_2 = 1.009917 radiano e θ_3 = 1.010308 radiano, correspondendo a um erro máximo de 0.076057 na matriz de transformação homogênea da equação 3.17. O algoritmo genético seguiu a evolução mostrada na figura 3.12.

$$A_{Obtida} = \begin{bmatrix} -0.231026 & -0.846892 & 0.478959 & 299.856710 \\ -0.367933 & 0.531766 & 0.762792 & 477.552834 \\ -0.900696 & 0.0 & -0.434451 & 444.159752 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix} \quad (3.17)$$

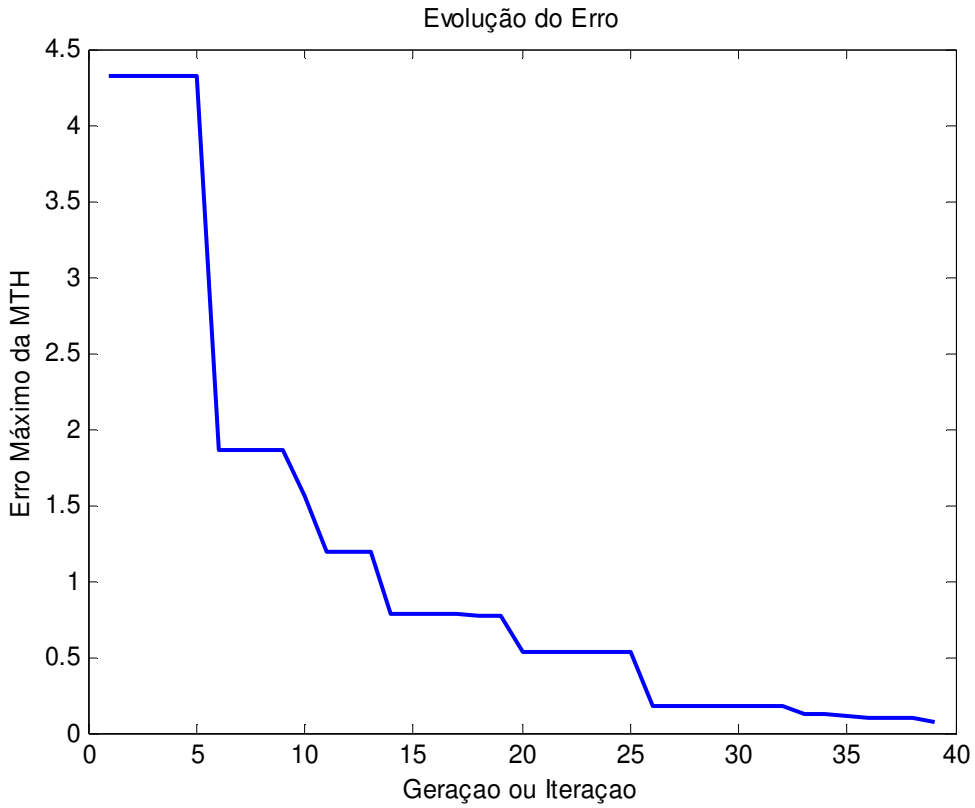

Fig. 3.12 – Evolução do erro no posicionamento incremental.

O processamento do posicionamento incremental é mais rápido do que o do posicionamento inicial devido à *string* que codifica os ângulos ser menor, 11 *bits* no posicionamento incremental contra 16 *bits* do posicionamento inicial.

3.5 – Conclusão

Inicialmente todos os programas computacionais foram desenvolvidos visando melhorar a performance, uma vez que os algoritmos genéticos exigem tempo de processamento bem maior quando comparado com outras técnicas, como a lógica *fuzzy* (Kosko, 1992), por exemplo. Após vários testes, verificou-se a necessidade de desenvolver os programas através da técnica de programação de orientação a objetos, uma vez que foi constatado que várias funções dos algoritmos genéticos poderiam ser reaproveitadas. Deste ponto em diante, utilizou-se a linguagem Java (Deitel e Deitel, 2002). Após o desenvolvimento do algoritmo genético em linguagem de

programação Java para o planejamento de trajetória, parte do código foi reaproveitado para o desenvolvimento do posicionamento inicial e incremental.

O estudo efetuado para o planejamento foi utilizando-se somente cinco segmentos de reta, mas o algoritmo funciona se existirem dois ou mais segmentos de reta, porém a complexidade, e conseqüentemente o tempo de execução, do algoritmo genético fica comprometido, uma vez que para cada aumento no número de segmentos de reta mais dados deverão ser manipulados pelo programa.

Somente um único obstáculo foi utilizado para mostrar o funcionamento desta aplicação, porém nada impede que mais de um possa ser utilizado, desde que seja fornecido a sua dimensão e o ponto onde está localizado. Cada trajetória testada pelo algoritmo genético deve ser antes de tudo uma trajetória válida, portanto todas as trajetórias tratadas pelo algoritmo genético não podem sob hipótese alguma passar sobre qualquer obstáculo. A dificuldade de se desenvolver um algoritmo que trate mais de um obstáculo se resume a implementar uma função que gere somente trajetórias válidas. Ainda neste caso, pode-se dividir o espaço de trabalho para que somente um obstáculo possa ser tratado de cada vez, facilitando a aplicação direta do algoritmo genético mostrado neste capítulo.

CAPÍTULO 4

SELEÇÃO DE CONTROLADORES NEURAIS UTILIZANDO ALGORITMOS GENÉTICOS

Através de redes neurais artificiais é possível, com pouco esforço humano, produzir controladores que apresentam características bastante satisfatórias quando aplicados na prática (Monteiro, 1996), (Monteiro e Takita, 1996) e (Monteiro e Takita, 1996a). Porém, tais controladores possuem muitas características que são obtidas empiricamente. Para reduzir ainda mais o esforço humano necessário para o desenvolvimento de controladores, este trabalho apresenta dois tipos básicos de controladores neurais. Neste capítulo é apresentado um algoritmo genético que visa a obtenção das características estruturais de um controlador neural, obtendo-se o melhor controlador encontrado no espaço pesquisado pelo algoritmo genético, (Monteiro, Madrid e Takita, 2001).

4.1 - Introdução

Verifica-se que a aplicação prática de controladores possui o empecilho de requerer que o desenvolvedor do controlador possua amplo conhecimento do processo que será controlado. Isto possui algumas implicações negativas que são: tempo para conhecer o processo a ser controlado; dependência de recursos humanos especializados; quando há variações nos parâmetros do processo, refazer quase que totalmente o controlador, etc.

Com o aparecimento de técnicas de inteligência artificial, tornou-se próspero o uso de controladores baseados nestas técnicas, visto que estes apresentam algumas vantagens interessantes do ponto de vista de implementação, pois necessitam de pouco, ou nenhum, cálculo matemático analítico. Por outro lado, algumas desvantagens são observadas nestes controladores, sendo que a principal é o conhecimento exigido pelo desenvolvedor do controlador sobre o processo a ser controlado, o que pode demandar um tempo razoavelmente grande para o projeto de um bom controlador. O conhecimento requerido para tais controladores se mostram bastantes presentes nas técnicas de sistemas especialistas e lógica fuzzy.

Neste trabalho apresentamos dois tipos de controladores que necessitam de pouquíssimo conhecimento a respeito do processo a ser controlado. Tais controladores são fundamentados na técnica de redes neurais artificiais. Como para determinar a estrutura de tal controlador (número de neurônios em cada camada) não há um método exato, utilizamos a técnica de algoritmos genéticos para determinar a estrutura mais apropriada em função do processo a ser controlado. O algoritmo genético testa os controladores através da simulação de uma junta de um robô, sendo que posteriormente, o controlador determinado por ele poderá ser avaliado na planta real.

4.2 - Estrutura de Controle

A figura 4.1 mostra a estrutura de controle que é usada para o controle de uma junta de um braço mecânico utilizando redes neurais.

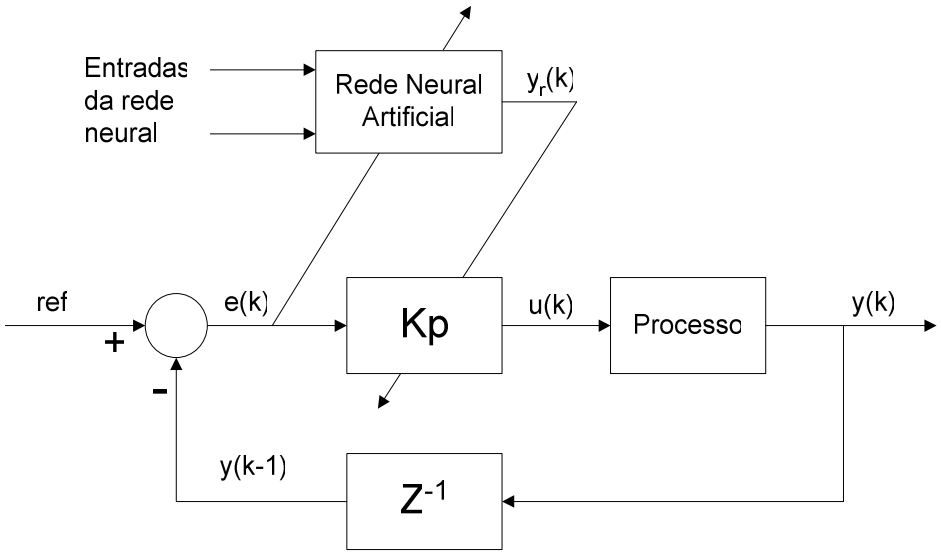

Fig. 4.1 - Estrutura de Controle

onde,

ref - posição de referência ou posição desejada.

e(k) - erro de posição atual.

u(k) - sinal de controle.

y(k) - saída atual do processo, ou seja, posição atual.

y(k-1) - saída anterior do processo, ou seja, posição anterior.

$y_r(k)$ - saída atual da RNA.

Observa-se que o sinal do erro de posição (referência - saída) é utilizado para corrigir o ganho do controlador proporcional através do algoritmo da retropropagação do erro, usado para o treinamento da rede neural. Este erro de posição é usado para corrigir os pesos sinápticos da rede neural, o que provoca alterações no ganho do controlador proporcional.

Como o erro de posição disponível está na unidade de graus, faz-se necessário um operador de normalização para que ele fique restrito ao intervalo [-1,+1]. O operador utilizado para esta normalização foi:

$$erro_N = \tanh(K_1 \cdot \text{erro}) \tag{4.6}$$

onde

$erro_N$ - erro de posição normalizado,

K_1 - constante de inclinação da função tanh(.),

erro - erro de posição em pulsos provenientes do *encoder*.

4.3 - Topologia da Rede

Foram utilizadas duas topologias de rede: uma estática e outra dinâmica. Chama-se rede neural estática a rede neural que possui entradas fixas, enquanto a rede neural dinâmica se caracteriza por ter cada saída da rede neural realimentada para a entrada, sendo que estas são deslocadas a cada iteração de aprendizagem. Cada RNA possui uma camada de entrada, uma camada escondida, e a camada de saída, que possui um único neurônio, sendo que este neurônio fornece o valor do ganho K_p que irá multiplicar o erro para o controle do processo.

Na primeira topologia, figura 4.2, observa-se que as entradas são fixas e com valores alternados de +1 e -1, e que se trata de uma rede neural estática. Esta topologia foi utilizada devido à sua fácil implementação e, também, aos estudos já bastante avançados sobre as RNA's destas topologias. Os valores das entradas da rede podem ser alterados sem qualquer complicação no modo de como a RNA irá executar a atualização do ganho K_p.

A segunda topologia, figura 4.3, utiliza as saídas passadas da RNA realimentadas para os neurônios de entrada, o que torna a RNA dinâmica.

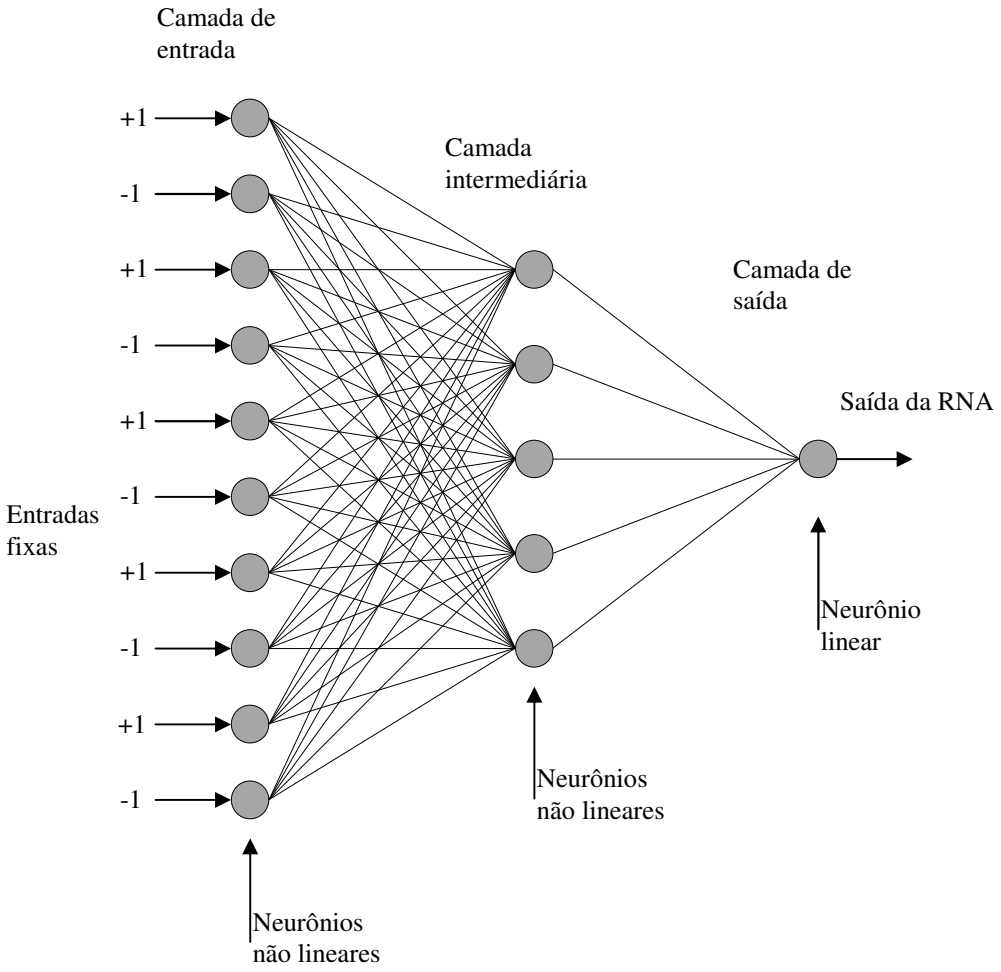

Fig. 4.2 - RNA estática com entradas fixas

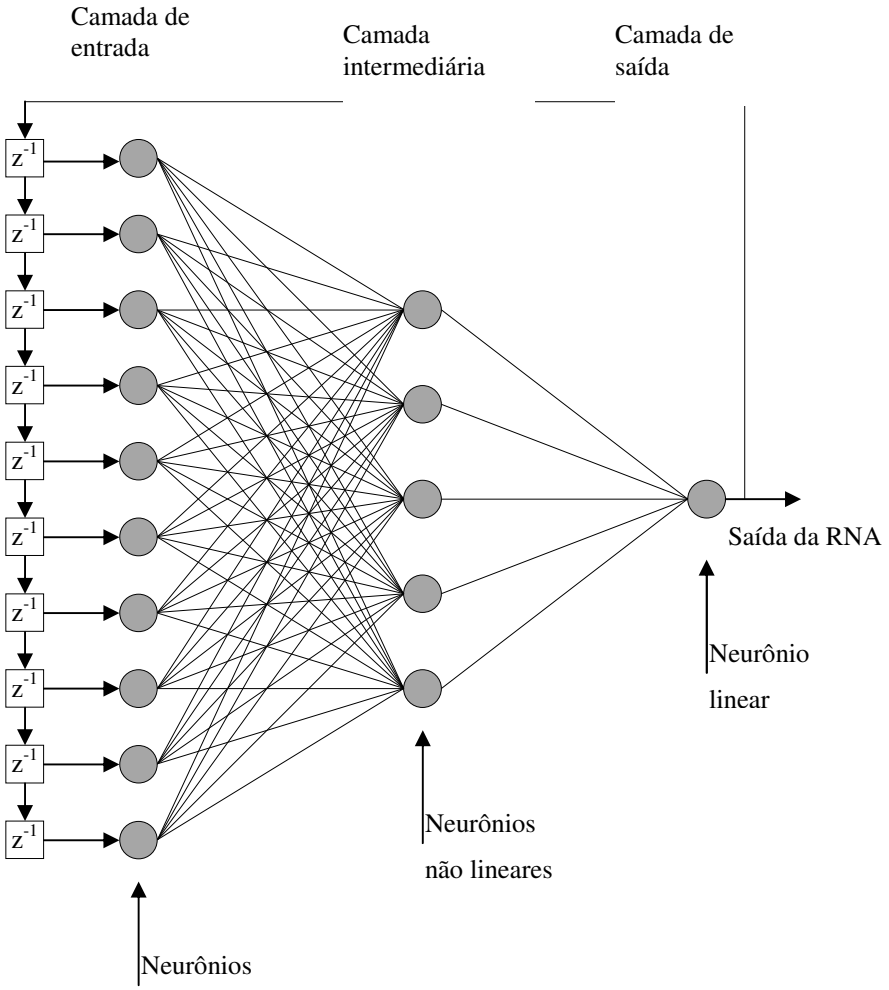

Fig. 4.3 - RNA dinâmica com saída da rede realimentada

Para o treinamento destas redes foi utilizado o algoritmo da retropropagação do erro (*back-propagation*), apresentado no capítulo 2. As funções de ativação da camada de entrada e da camada escondida são tangentes hiperbólicas, e a da camada de saída é uma função linear. As funções de ativação da camada de entrada e da camada escondida foram escolhidas como não lineares para poder-se considerar as não linearidades que o processo possui, ou seja, a inércia de repouso, a inércia de movimento, as zonas mortas, etc. Já a função de ativação da camada de saída, que é composta por um único neurônio, fornece o valor do ganho do controlador proporcional, e necessariamente deve ser do tipo linear, ou, quando se tem algum conhecimento do processo, esta pode ser do tipo parcialmente linear, justamente para apresentar um limite na saída da rede neural, pois se este ganho for alto demais, o processo poderá tornar-se instável.

A função de ativação parcialmente linear não foi utilizada neste trabalho devido o não conhecimento dos parâmetros que regem o processo, pois o que mais se quis mostrar é que os

controladores neurais servem para controlar processos que são totalmente desconhecidos, ou bem pouco conhecidos.

4.4 - Taxa de Aprendizagem

Quando se utiliza uma rede neural para o controle de um processo, deve-se ter em mente que as não linearidades e as perturbações que o processo sofre, ou poderá vir a sofrer, devem ser compensadas. Após varias observações efetuadas empiricamente no controle de processos envolvendo controladores neurais, notou-se que para atingir estes objetivos o principal fator do controlador neural usado era a taxa de aprendizagem. Para o caso do controle, a melhor solução conseguida para fazer com que a taxa de aprendizagem superasse os problemas que o processo apresenta (oscilações e zonas mortas) foi fazer com que a mesma assumisse a seguinte função:

$$\eta(erro) = \begin{cases} \eta_{MAX}, & \text{se } \Delta\text{erro} = 0 \\ K_1|erro|, & \text{se erro} > \text{erro}_{\min} \\ \dfrac{K_2}{|erro|}, & \text{se erro} \leq \text{erro}_{\min} \end{cases} \tag{4.7}$$

onde,

$\eta_{MAX} \rightarrow$ taxa de aprendizagem máxima.

$\text{erro}_{\min} \rightarrow$ limite entre as diferentes funções.

K_1 e $K_2 \rightarrow$ constantes a serem definidas.

O primeiro elemento da equação acima é utilizado para que a zona morta[4] inicial do processo seja vencida. Ele somente é necessário quando o processo entra em estado de repouso, ou seja, quando a diferença entre o erro atual e o erro anterior é nula. Ele deve ser suficientemente grande para que nos primeiros instantes o ganho K_p cresça o bastante para que ocorra o início do movimento, mas não deve ultrapassar um certo valor, pois o processo poderá entrar em estado de instabilidade, já que isso pode implicar em um valor bastante alto para K_p, colocando o processo a ser controlado em instabilidade.

---

[4] Entende-se aqui por zona morta a inércia que deverá ser vencida para que o motor possa girar. Neste caso, o sinal de controle deverá ser suficiente para romper está inércia.

O segundo termo informa ao sistema que a taxa de aprendizagem é proporcional ao erro detectado pela perturbação, e quanto menor for este erro, menor será a taxa de aprendizagem. Este termo diminui a aceleração provocada pelo primeiro termo, fazendo com que o processo chegue à posição com uma velocidade proporcional ao erro, ou seja, quanto menor for o erro menor será a taxa de aprendizagem e menores serão as alterações impostas pelo ganho K_p, o que imprime uma velocidade menor no deslocamento da junta do robô, resultando em maior lentidão no aprendizado.

O terceiro termo é comparado a um controlador integral, pois como é inversamente proporcional ao erro, ele faz com que a taxa de aprendizagem cresça à medida que o erro diminui. Este termo foi colocado para que o erro de regime fosse corrigido, e também é o responsável pela velocidade de convergência final, pois quanto menor for este valor maior será a velocidade de convergência final.

Esta taxa de aprendizagem adaptativa é obtida de forma empírica, devido à falta de conhecimento das característica intrínsecas do processo, o que é uma das dificuldades de se utilizar o controle para este processo.

Notou-se que a taxa de aprendizagem é sensível a alteração do período de amostragem e aos pesos iniciais e que, caso isso ocorra, é necessário fazer novos ajustes nas constantes K_1 e K_2, para que o controlador neural funcione de forma satisfatória, atingindo a referência desejada.

Nos próximos itens deste capítulo apresentaremos duas aplicações práticas destes controladores neurais.

4.5 – Aplicação: Mesa XY

Para validar os controladores neurais estáticos e dinâmicos, estes foram testados para controlar a posição de uma mesa XY, (Monteiro e Takita, 1996) e (Monteiro e Takita, 1996a), cujos *croquis* são apresentados na figura 4.4.

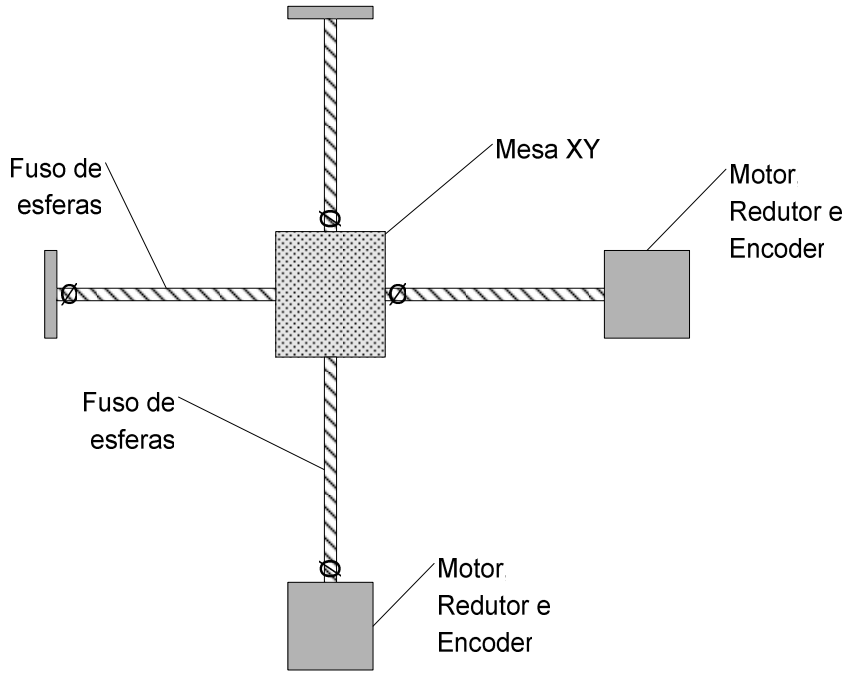

Fig. 4.4 – *Croquis* da mesa XY.

A planta acima é composta por dois fusos de esferas onde correm duas mesas em cada. Uma mesa é colocada sobre a outra de tal forma que os fusos formam um ângulo de 90º, um em relação ao outro, proporcionando movimentos no eixo x e no eixo y.

Em uma das extremidades de cada fuso são acoplados mecanicamente um sistema de acionamento composto por um motor CC, uma caixa de redução e um *encoder* relativo acoplado diretamente ao eixo do motor, sendo que cada *encoder* possui uma resolução de 200 pulsos por rotação. Nesta planta um pulso equivale a um deslocamento linear de 0,004166 mm. Na figura 4.5 pode-se ver como os principais componentes que servem para controlar a posição da mesa XY estão interligados.

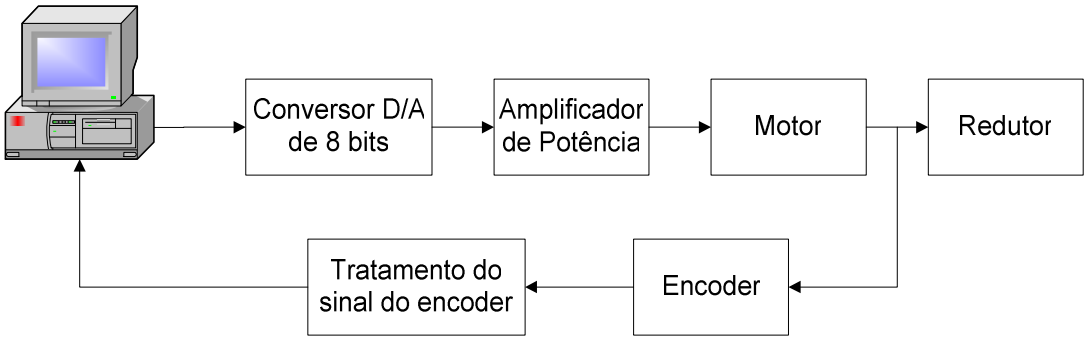

Fig. 4.5 – Controle de 1 eixo da mesa XY

Através de um circuito digital que possui conversores D/A de 8 bits e de um circuito eletrônico amplificador de potência, pode-se acionar os motores para que a mesa XY entre em funcionamento. Para se realizar o controle de posição, o sinal proveniente das duas fases de cada *encoder* é tratado para que gere um sinal digital que possa ser manipulado por um computador. Tanto o circuito digital com conversores D/A quanto o circuito de tratamento dos sinais do *encoder* estão localizados em uma única placa que é colocada em um barramento ISA dentro de um computador. Assim, o acionamento dos motores e os sinais do *encoder* estão à disposição de um programador para o desenvolvimento de um programa de controle.

O computador utilizado para o controle da mesa XY foi um microcomputador com processador Intel 486, 66 MHz e 16 MByte de memória RAM. Todos os controladores da mesa XY foram executados neste computador.

4.5.1 – Controlador neural para a mesa XY

Como o mecanismo de acionamento de cada eixo da mesa XY é exatamente o mesmo, alterando-se somente que o eixo inferior irá deslocar uma massa maior, referente ao eixo superior, o controlador neural foi testado somente para o controle de posição do eixo superior da mesa XY.

Os valores das constantes K_1 e K_2 dos controladores neurais, assim como o número de neurônios da camada de entrada e da camada escondida, foram obtidos empiricamente. Os valores foram:

$K_1 = 0.2$

$K_2 = 0.001$

Número de neurônios da camada de entrada = 10

Número de neurônios da camada escondida = 5

Os erros de posição dos controladores neurais estáticos e dinâmicos são mostrados nas figuras 4.6 e 4.8, e as curvas do esforço de controle são mostradas nas figuras 4.7 e 4.9, respectivamente.

Os pesos iniciais foram selecionados aleatoriamente dentro do intervalo [-0.1, +0.1].

Como a unidade básica de medida de posição é um pulso do *encoder*, a referência a ser atingida pelo controlador é igual a 10.000, que equivale a 41,66 mm.

O tempo, no eixo x, é mostrado em segundos.

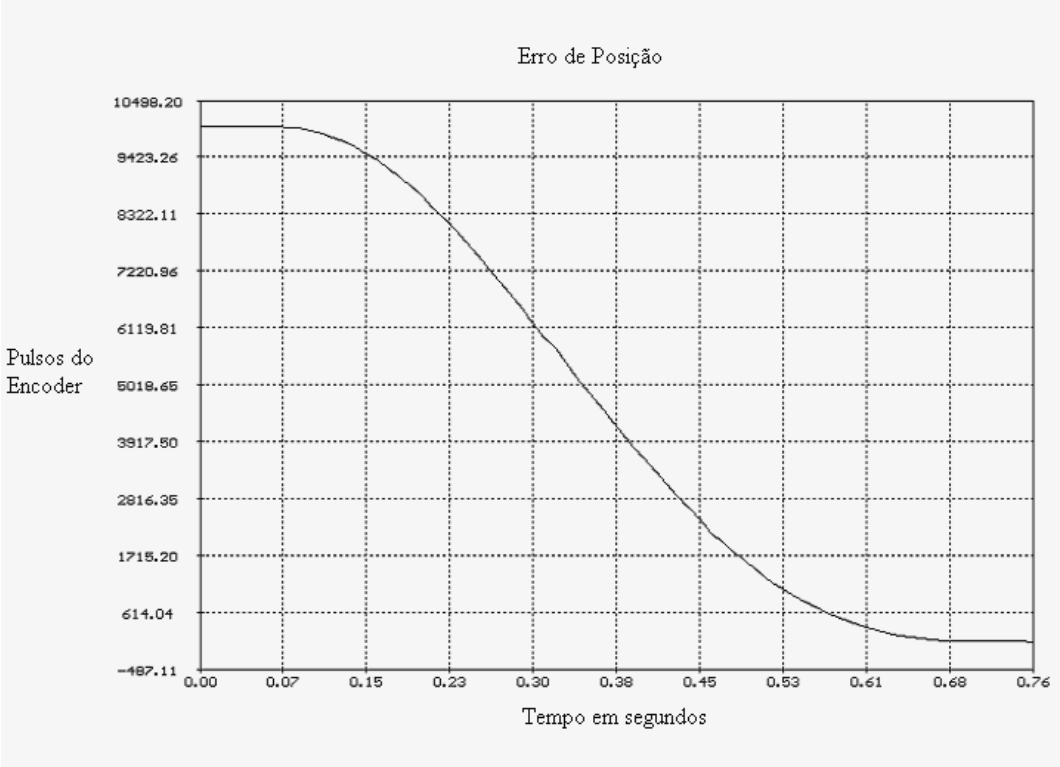

Fig. 4.6 – Erro de posição do controlador neural estático.

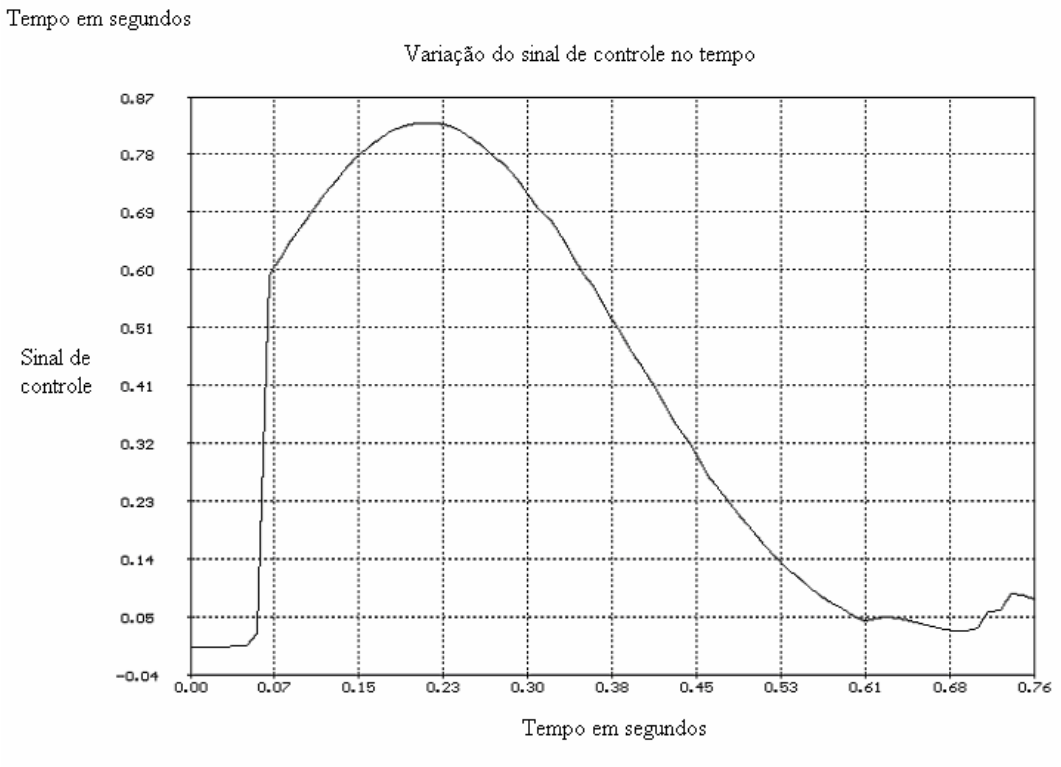

Fig. 4.7 – Sinal de controle do controlador neural estático.

O sinal do erro de posição do controlador neural estático da figura 4.6 está bastante próximo do ideal do ponto de vista do controle, pois no início o movimento se mantém sem alteração até a inércia ser vencida. Após isto, o movimento é feito com uma velocidade aproximadamente constante, e quando o erro se aproxima do zero a velocidade baixa, permanecendo baixa até que o erro esteja dentro da tolerância desejável.

O sinal de controle da figura 4.7 começa com zero e permanece assim enquanto a rede neural está "aprendendo", fazendo com que a taxa de aprendizagem tenha um valor máximo, o que permite que a rede atualize os seus pesos sinápticos com maior rapidez. Após o movimento começar, o sinal de controle aumenta, e de acordo com a aproximação da referência, diminui.

Este tipo de comportamento, principalmente do erro de posição, poupa bastante os esforços na mecânica do processo devido começar e terminar do movimento com suavidade, sem movimentos bruscos.

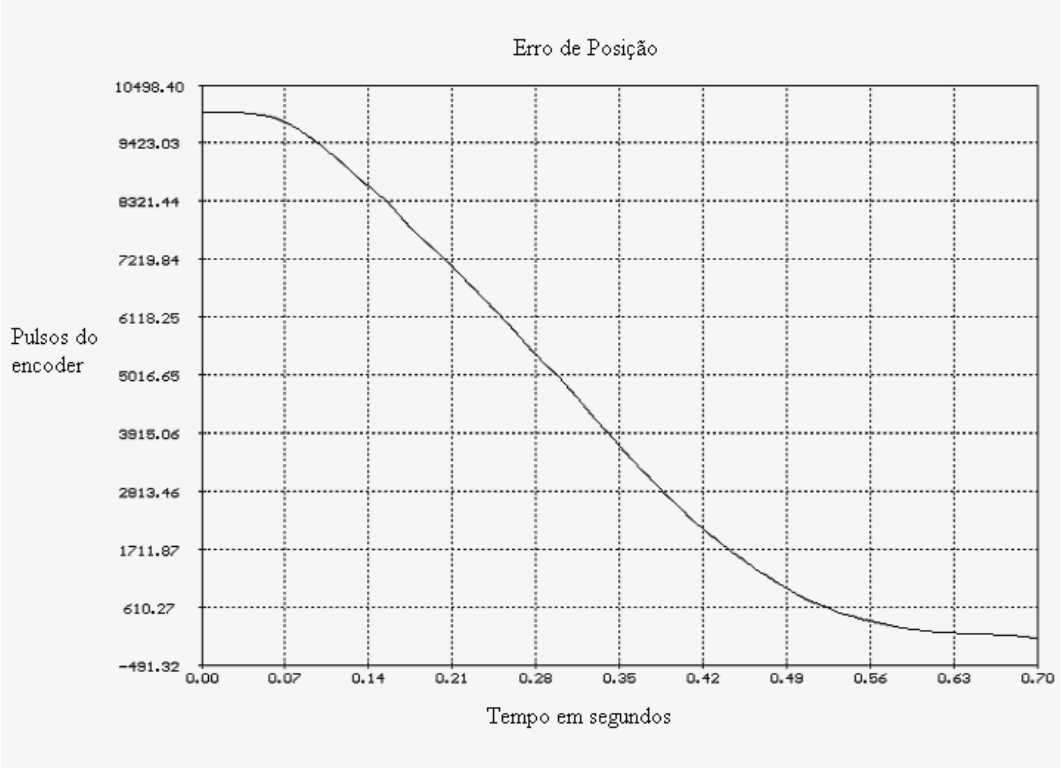

Fig. 4.8 – Erro de posição do controlador neural dinâmico.

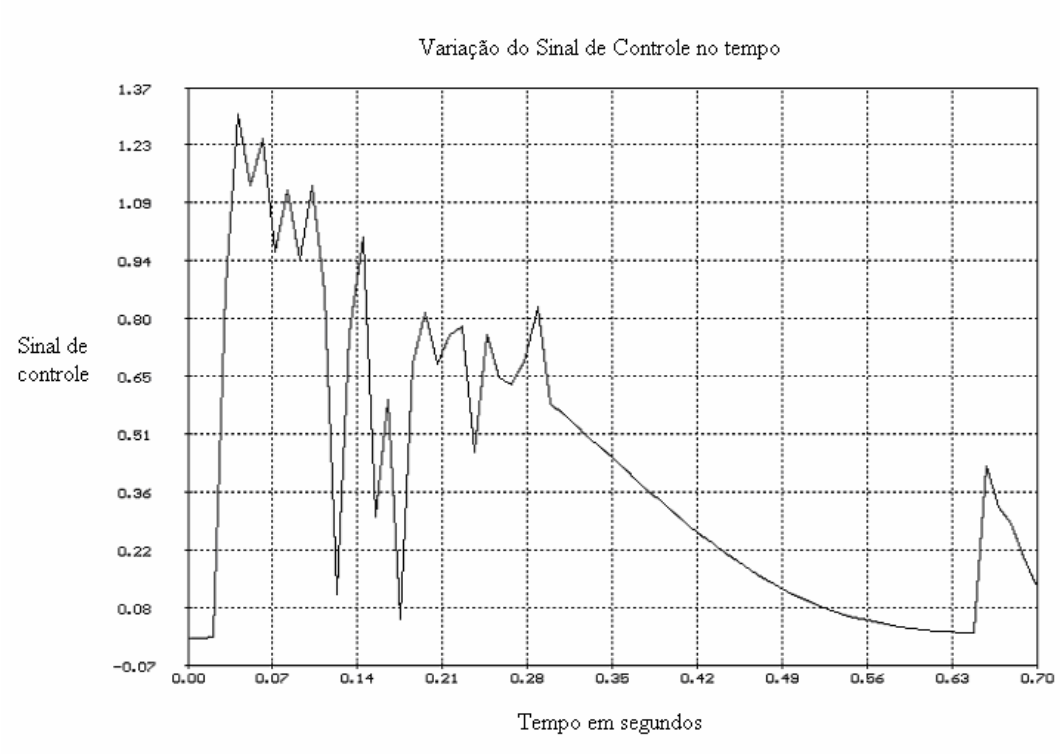

Fig. 4.9 – Sinal de controle do controlador neural dinâmico.

Nota-se que o gráfico de erro de posição do controlador neural dinâmico, figura 4.8, é bastante parecido com o do controlador neural estático, porém o sinal de controle é completamente diferente, figura 4.9. A principal diferença entre os dois tipos de controladores é o tempo que se leva para atingir a referência.

Para o controlador neural estático, o tempo para atingir a referência foi de 0,76 segundos, enquanto que para o controlador neural dinâmico este mesmo tempo para atingir a referência foi de 0,7 segundos. Neste caso, o controlador neural dinâmico apresenta melhor performance quando se vislumbra um menor tempo para que o processo atinja a posição de referência desejada.

4.6 – Aplicação: Pêndulo Invertido Acionado

O experimento com o pêndulo invertido acionado foi usado por este apresentar maior similaridade com um braço mecânico, uma vez que está sujeito à força da gravidade, e conseqüentemente a não linearidades que um braço de robô apresenta.

Este experimento foi realizado em um computador compatível com o IBM PC, cujo processador é o Intel 486, com *clock* de 100 MHz, e 8 MB de memória RAM. Além disso, o software de rede neural foi desenvolvido utilizando-se o compilador Turbo C++, e o programa final foi executado sob o sistema operacional Windows 95, em ambiente MS-DOS, ou seja, as interrupções que ocorrem freqüentemente não interferem no processo de controle. Como foi utilizado o compilador Turbo C++ para DOS, os valores do erro de posição e do sinal de controle para cada iteração não puderam ser armazenados em um vetor, devido tal compilador só permitir o armazenamento de 1 MByte de dados, sendo que são necessários pelo menos 100 MBytes por execução do controlador neural. Portanto, os dados foram gravados em disco para depois serem visualizados através de gráficos gerados com um programa gerador de gráficos.

A figura 4.10 mostra o processo do pêndulo invertido em detalhes.

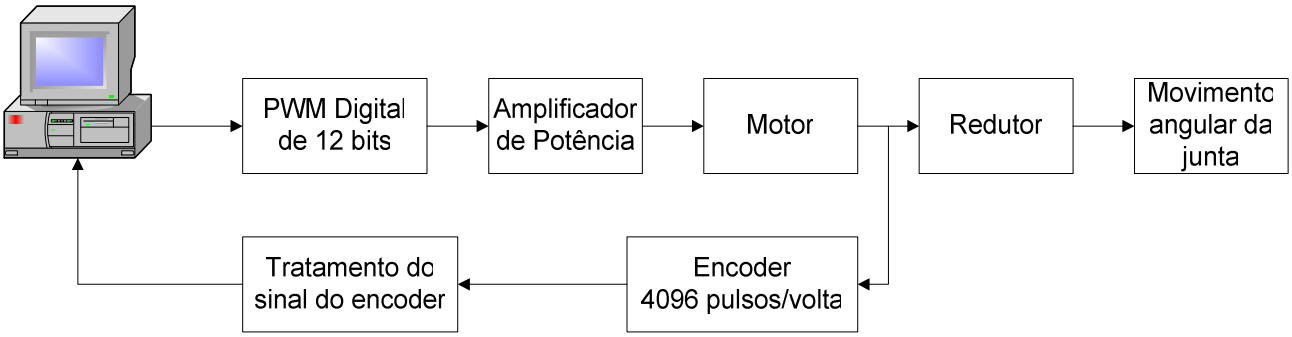

Fig. 4.10 - Estrutura do processo do pêndulo invertido.

Nas figuras 4.11 e 4.12 podem ser vistas fotos da parte mecânica do pêndulo invertido. Na figura 4.11 observa-se que está acoplada ao eixo do motor uma barra que nada mais é do que uma junta de um braço de robô. Na extremidade inferior desta barra é colocado um disco, cujo peso pode ser alterado, simulando a carga que a junta pode deslocar.

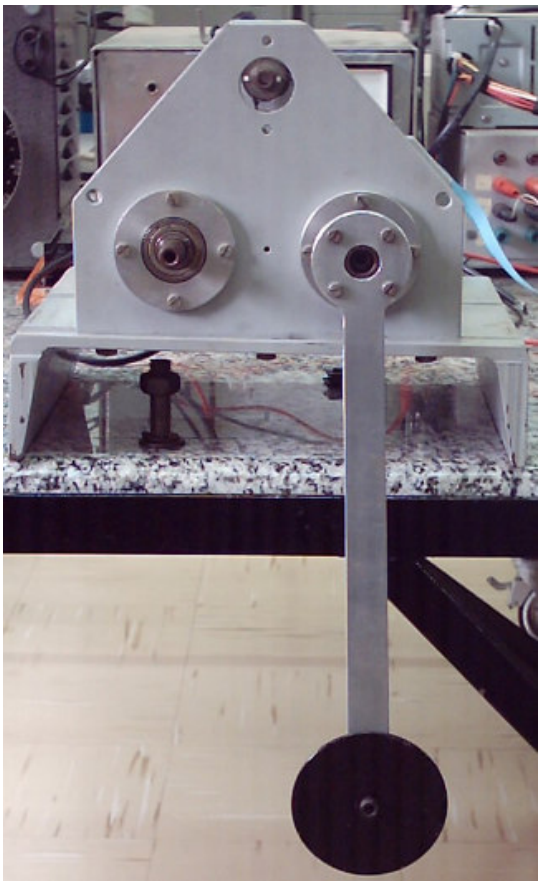

Fig. 4.11 – Pêndulo invertido visto de frente

Na figura 4.12, é detalhado o mecanismo de acoplamento do conjunto do motor, redutor e *encoder*. Caso uma maior redução seja necessária, basta retirar a barra de um eixo e colocar em outro. Neste eixo a relação de redução é de 1:16.

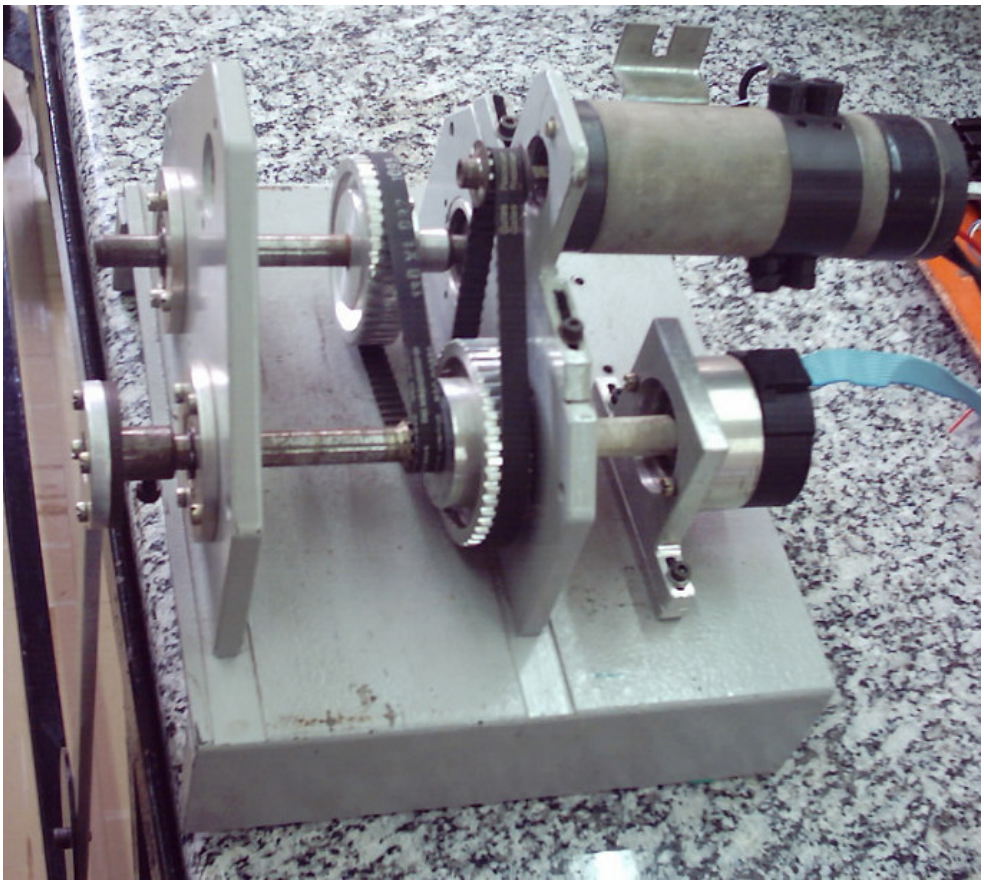

Fig. 4.12 – Motor, redutor e *encoder* do pêndulo invertido.

Todos os experimentos apresentados partiram da posição inicial de repouso do pêndulo invertido, considerando-se tal posição como a posição de 0˚, como mostrado na figura 4.11. A camada de entrada possui 10 neurônios, a camada intermediária 5 neurônios e os pesos iniciais foram selecionados aleatoriamente dentro do intervalo [-0.1, +0.1]. A topologia da rede neural escolhida para verificar o desempenho do controlador foi a estática.

O controlador neural foi executado para que o pêndulo atingisse as referências de 10˚, 45˚, 90˚ e 120˚, sendo que os erros de posição conseguidos nestes experimentos estão mostrados nas figuras 4.13, 4.14, 4.15 e 4.16, respectivamente. As primeiras posições foram escolhidas para verificar se o controlador atingiria o seu objetivo para posições onde a força de gravidade não estaria tendo uma forte influência, ao contrário do que ocorre com as duas últimas posições, onde é necessário vencer a maior intensidade da força da gravidade para que o pêndulo atinja a posição

de referência.

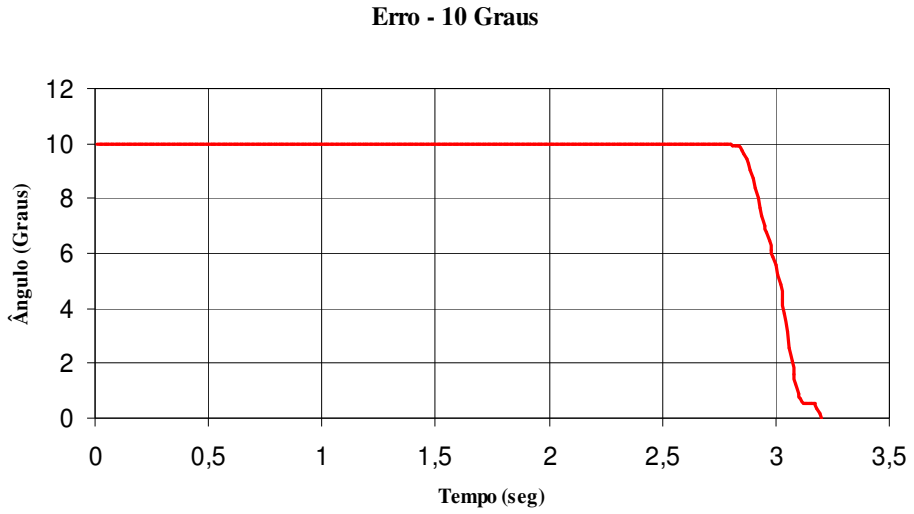

Fig. 4.13 – Sinal do erro de posição do pêndulo invertido para uma posição de 10 graus.

No gráfico da figura 4.13 observa-se que houve uma grande demora para que o movimento se inicie. Isto se deve ao fato da rede neural ainda não ter informações suficientes para saber iniciar o movimento. Após o início do movimento a referência foi atingida sem maiores problemas, só acontecendo uma pequena parada quando o erro estava bastante próximo do zero, o que equivale à terceira parte da equação (4.7).

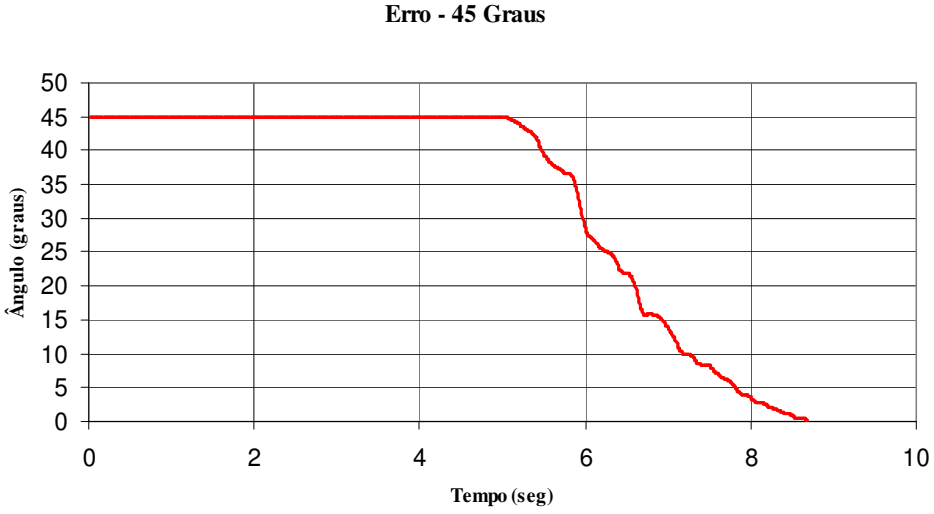

Fig. 4.14 – Sinal do erro de posição do pêndulo invertido para uma posição de 45 graus.

O sinal do erro de posição da figura 4.14 ainda apresenta um atraso significativo no início do movimento, o que é uma característica deste tipo de controlador. Como a posição a ser alcançada é de 45˚, já se observa um início de perturbação provocada pela força de gravidade no movimento, o que não é observado em nenhum dos gráficos obtidos pela emprego deste controlador neural na mesa XY. Este efeito da força da gravidade começa a ser observado através de alguns pontos no gráfico onde existe pequenas paradas no movimento durante o processo de aprendizagem.

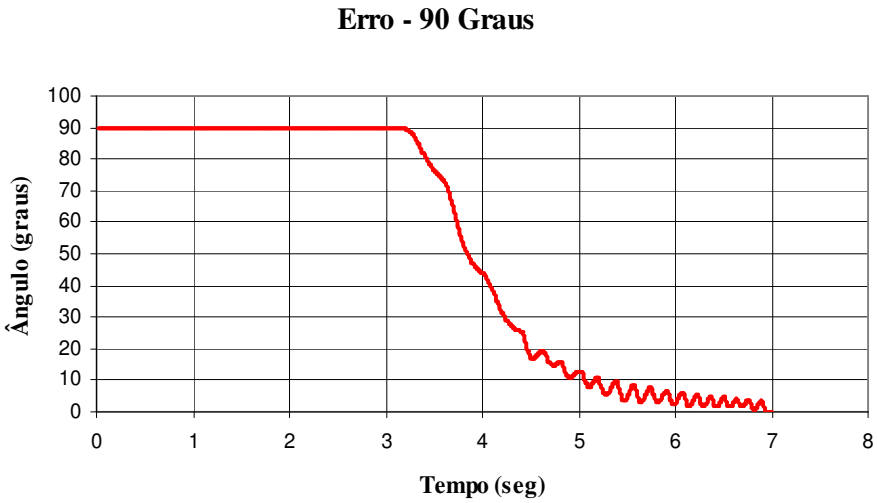

Fig. 4.15 – Sinal do erro de posição do pêndulo invertido para uma posição de 90 graus.

Para a referência de 90˚, a perturbação da força de gravidade é bastante evidente, principalmente quando se aproxima da referência, onde observa-se várias ondulações após o pêndulo passar de 80˚. Deste ponto em diante, a força de gravidade começa a exercer uma forte influência no movimento, tentando fazer com que o pêndulo retorne para as posições anteriores, porém o controlador torna a agir, imprimindo maior reação para que o pêndulo atinja a referência desejada.

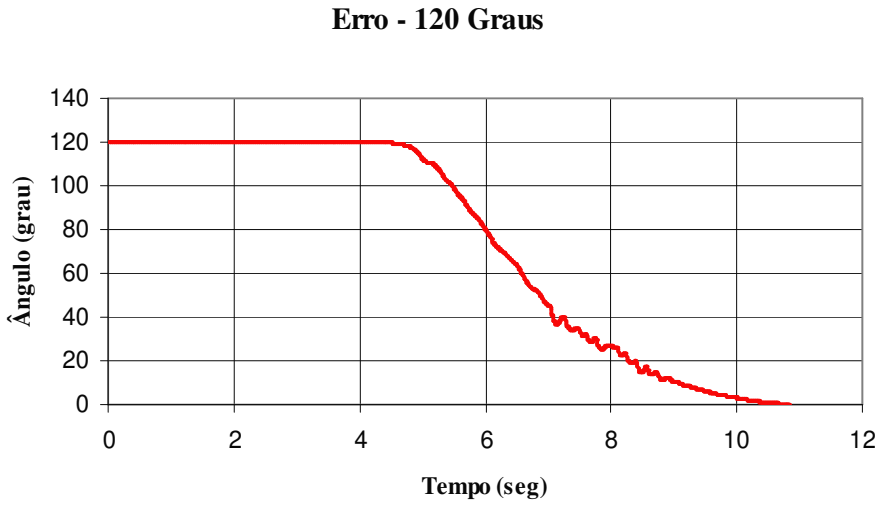

Fig. 4.16 – Sinal do erro de posição do pêndulo invertido para uma posição de 120 graus.

Na figura acima, 4.16, observa-se que as mesmas características obtidas nos gráficos anteriores se mantiveram inalteradas, inclusive apresentando demora para iniciar o movimento, e há leve oscilação quando a referência cruza o ângulo de 90˚. Como a referência é de 120˚, nota-se que após a posição ultrapassar o ângulo de 90˚, as oscilações diminuem até que o movimento se torna mais suave, atingindo a referência desejada.

O pêndulo invertido não teve maiores problemas para atingir as referências de 10˚ e 45˚, porém nota-se que no início do movimento houve uma demora de mais de 50% do tempo total de movimento. Após diversas repetições do teste com outros valores que influenciam o desempenho do controlador neural, tais como as constantes K_1 e K_2, os números de neurônios nas camadas de entrada e intermediária, verificou-se que os valores dos pesos iniciais influem mais intensamente quando o movimento inicia. Caso o valor dos pesos iniciais seja baixo, pertencentes ao intervalo [-0.1, +0.1], a inércia de repouso demora mais a ser vencida, mas caso os pesos iniciais possuam um valor relativamente alto, dentro do intervalo [-1, +1], tal inércia é mais facilmente vencida. Este aumento dos valores dos pesos iniciais traz consigo o problema da instabilidade, já que o movimento que começa mais rapidamente possui uma energia maior e, principalmente para pequenas distâncias, essa grande energia é mais dificilmente controlada, levando o processo a oscilar em torno da referência desejada ou até mesmo à instabilidade.

Para as referências de 90˚ e 120˚, nota-se que o movimento sofre oscilações ao aproximar-

se da posição referente a 90˚. Note que no ponto de 90˚ a força de gravidade atua de forma mais intensa. Estas oscilações se mostraram características em todos os experimentos realizados com este controlador.

4.7 - Algoritmo Genético

Conforme visto nas topologias apresentadas, não existe uma forma exata de determinar quantos neurônios deverão constituir as camadas de entrada e intermediária.

Para resolver tal problema, utiliza-se um algoritmo genético para determinar qual a melhor estrutura de controlador dentro do espaço dos controladores (Sousa, 2000).

Foi utilizado o esforço de controle como medida de desempenho de cada controlador, e baseado nela, o algoritmo genético determina qual o melhor controlador para o espaço dos controladores pesquisado.

Na Figura 4.17, está apresentado um fluxograma do algoritmo genético utilizado nesta experimentação.

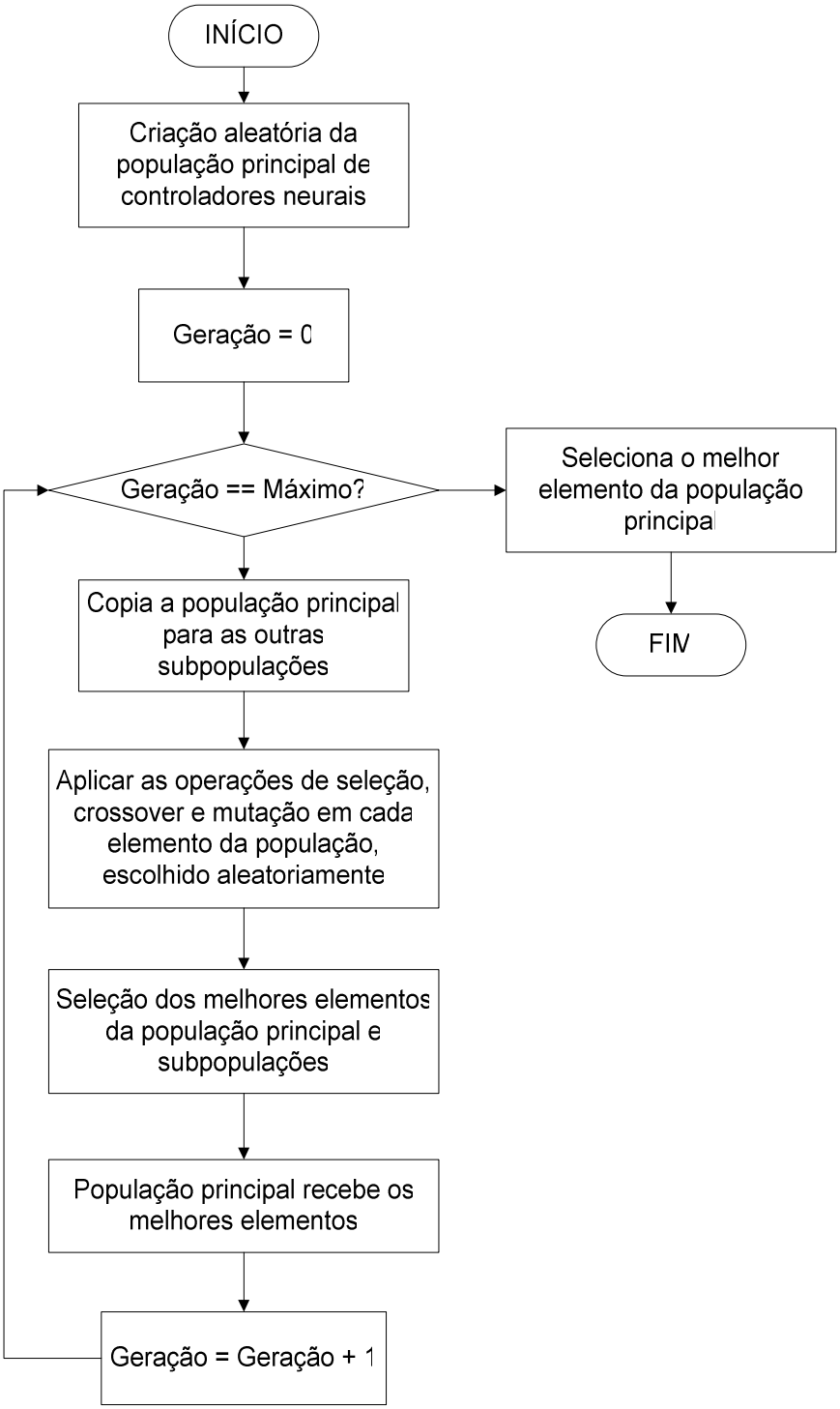

Fig. 4.17 – Fluxograma do algoritmo genético usado para a seleção dos controladores neurais.

As principais características e operações efetuadas pelo GA exposto acima são:

4.7.1 - Indivíduo

O indivíduo no GA utilizado é composto por uma *string* de *bits* que codifica o número de neurônios na primeira camada, o número de neurônios na camada escondida, valores das constantes da taxa de aprendizagem e o tipo de topologia.

Para determinarmos o número de neurônios de cada camada utiliza-se uma *string* de 5 *bits*, sendo que o número máximo de neurônios nas camadas de entrada e escondida é 32, e o mínimo é 1. Caso haja necessidade de aumentar o número de neurônios por camada, basta aumentar o número de *bits* na *string* que representa o número de neurônios em cada camada.

As constantes da taxa de aprendizagem são codificadas em uma *string* de 17 *bits* cada. Estas podem variar dentro do intervalo [0, +1]. Com os 17 *bits* pode-se ter uma precisão de 0.00000762939453125, que é suficiente para representar tais constantes.

A topologia é codificada com 1 bit. Caso este bit seja 0, uma RNA estática é usada, caso contrário uma RNA dinâmica é escolhida.

4.7.2 - Função de *fitness*

Para avaliar a trajetória utilizou-se o valor do esforço de controle para atingir a referência. Caso a referência não seja atingida em um determinado tempo, o esforço de controle recebe um valor máximo, que varia no intervalo [-24, +24] volts, indicando que tal controlador não atingiu o objetivo.

Como os GAs foram elaborados para maximizar uma determinada função, usou-se a equação empírica abaixo para determinar o valor de desempenho de cada controlador.

$$\text{Fitness(indivíduo)} = 1.2 * \text{MECG} - \text{ECI} \tag{4.8}$$

onde,

Fitness(indivíduo) – função de *fitness* do indivíduo.

MECG: Máximo Esforço de Controle da Geração.

ECI: Esforço de Controle do Indivíduo.

A utilização do fator 1.2 na equação (4.8) é para que haja, também, a probabilidade de se

selecionar o pior indivíduo da geração, fazendo com que o mesmo tenha alguma chance de participar do processo evolutivo do algoritmo genético.

O esforço de controle é calculado em função dos dados obtidos do sinal de controle, através da seguinte equação:

$$ECI = \sum_{i=1}^{N-1} x(i) \cdot \left[t(i+1) - t(i) \right] \tag{4.9}$$

onde,

x(i): amplitude do esforço de controle.

t(i): tempo onde o esforço de controle começa a ser aplicado.

t(i+1): tempo onde o esforço de controle x(i) foi terminado.

N: número de amostras do esforço de controle aplicadas ao processo.

4.7.3 - Seleção

A operação de seleção é realizada utilizando o método *roullete-wheel*, ou seja, o indivíduo que apresenta melhor desempenho tem maior probabilidade de ser selecionado para a operação de *crossover*.

4.7.4 - *Crossover*

O *crossover* utilizado no primeiro experimento é o *crossover* simples, enquanto o segundo experimento utiliza o *crossover* uniforme. Ambos os *crossovers* manipulam duas *strings* de *bits* para gerar outras duas novas *strings*. Tais *strings* são selecionadas através da operação de seleção acima e só ocorre caso a probabilidade de *crossover* for satisfeita.

4.7.5 - Mutação

A operação de mutação é realizada através da alteração de um determinado bit de um indivíduo da geração quando uma determinada probabilidade for atingida para tal indivíduo. Esta alteração consiste da troca do valor de *bit* de uma determinada posição na *string* que representa o indivíduo.

4.8 - RESULTADOS

Para avaliar a aplicação de controladores neurais em braços de robô, foi utilizado o modelo dinâmico da junta do braço do robô Jeca II, que foi construído no Laboratório de Sistemas Modulares Robóticos da UNICAMP. O modelo do braço possui as seguintes características:

Massa do braço (m) = 1.50 Kg.
Comprimento do braço (l) = 0.5 m
Resistência da armadura do motor (R) = 1.6 Ω.
Indutância da armadura do motor (La) = 0.0048 H.
Constante mecânica do motor (Kt) = 0.35 N-m/A
Constante elétrica do motor (Ke) = 0.35 V-s/rad
Momento de inércia do motor (Jm) = 0.0377 Kg-m^2
Constante de atrito do motor (Dm) = 0.25 N-m s/rad
Atrito viscoso da junta (Dl) = 0.05 N-m s/rad
Relação de redução (n) = 4.0
Momento de inércia do braço do robô (Jl) = 0.125 Kg-m^2

Para o modelo acima, a aceleração da gravidade (g) foi considerada igual a 9.8 m/s^2.

4.8.1 - Algoritmo genético

Para testarmos o algoritmo genético acima foram efetuados dois experimentos. O primeiro reduz o espaço dos controladores pesquisados através da fixação das constantes K_1 e K_2, além da utilização da operação de *crossover* simples para a geração de novos elementos em cada geração do algoritmo genético. No segundo experimento, todas as variáveis do controlador são otimizadas, e a operação de *crossover* usada é o *crossover* uniforme.

4.8.1.1 – Experimento 1

Os testes executados com o algoritmo genético mostram que existem diversas regiões onde os controladores selecionados não atingem a referência, ou seja, existem diversos pontos de singularidade na região explorada, onde vários controladores conduzem o processo à

instabilidade, não se tornando uma das possíveis soluções ao problema proposto. Para evitar tal problema, o número de variáveis a serem otimizadas foi reduzido, e uma busca local foi realizada. No exemplo mostrado a seguir, as variáveis escolhidas para serem otimizadas foram a topologia (estática ou dinâmica), o número de neurônios da camada de entrada, e o número de neurônios da camada escondida.

Para este caso, o algoritmo genético foi executado durante 20 gerações. Cada geração possuindo 6 populações com 6 indivíduos cada, resultando em 36 indivíduos por geração. Os valores para as constantes do controlador neural foram:

$K_1 = 0,02$

$K_2 = 0,05$

A referência a ser atingida pelos controladores neurais foi de 15 graus.

A evolução do algoritmo genético está mostrada na figura 4.18. O erro de posição do melhor indivíduo para as gerações 1, 3, 8 e 20 estão mostrados nas figuras 4.19, 4.20, 4.21 e 4.22, respectivamente. O sinal de controle para estas mesmas gerações está mostrado nas figuras 4.23, 4.24, 4.25 e 4.26, respectivamente. Os gráficos apresentados foram obtidos através de simulações feitas em um ambiente DOS, que não suportava a quantidade de dados gerados por cada controlador, portanto utilizou-se um programa gerador de gráficos para a criação dos gráficos mostrados abaixo.

A estrutura da rede neural artificial selecionada pelo algoritmo genético para controlar o processo em questão possuiu os seguintes parâmetros: 5 neurônios na camada de entrada, 1 neurônio na camada escondida e a topologia estática.

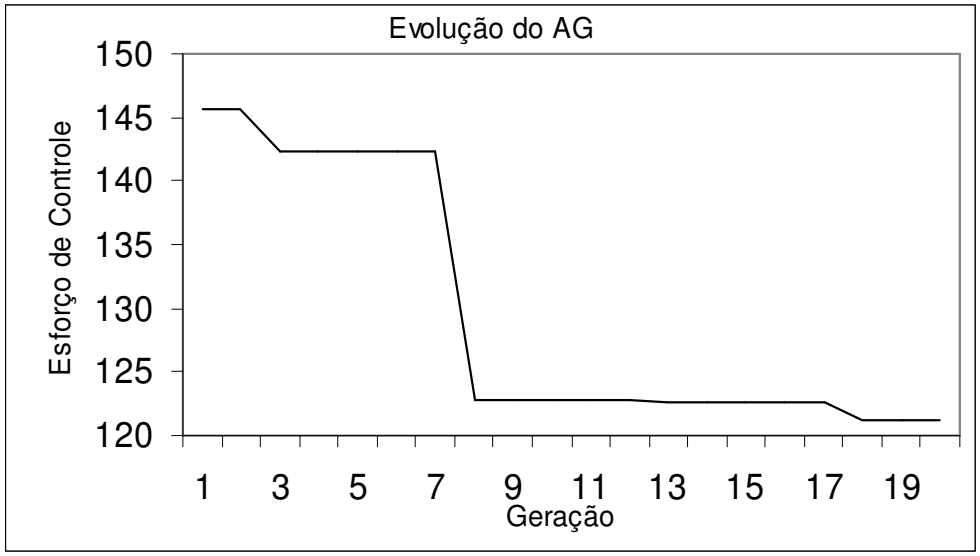

Fig. 4.18- Evolução do AG.

No gráfico da figura 4.18, observa-se que o esforço de controle tende sempre a decrescer. Com um menor esforço de controle, uma quantidade menor de energia é transferida para o processo, tornando-o mais eficiente.

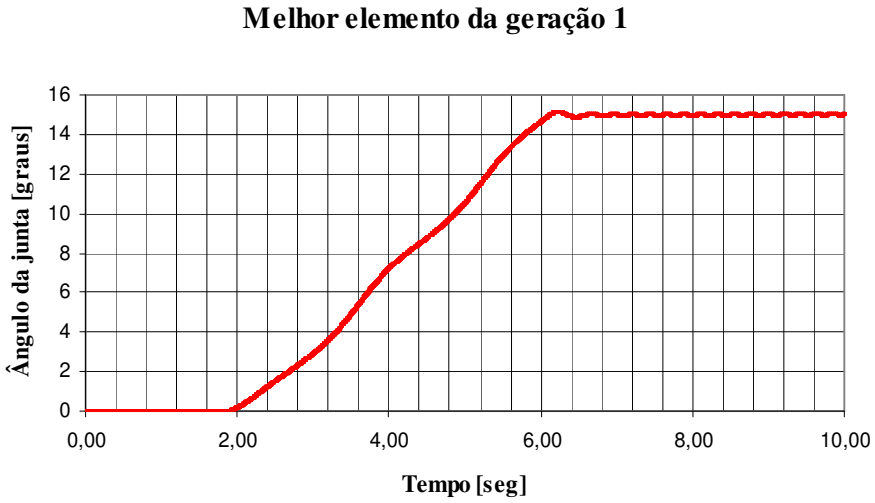

Fig. 4.19 – Saída do melhor controlador da geração 1.

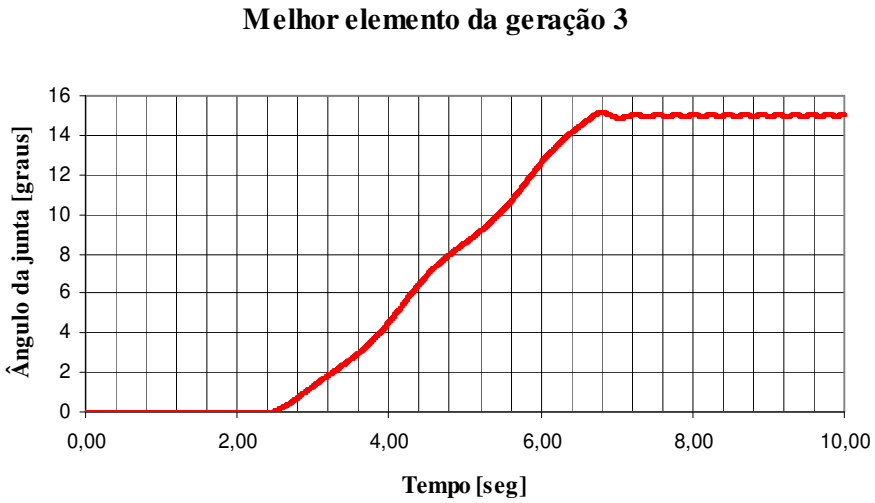

Fig. 4.20 – Saída do melhor controlador da geração 3.

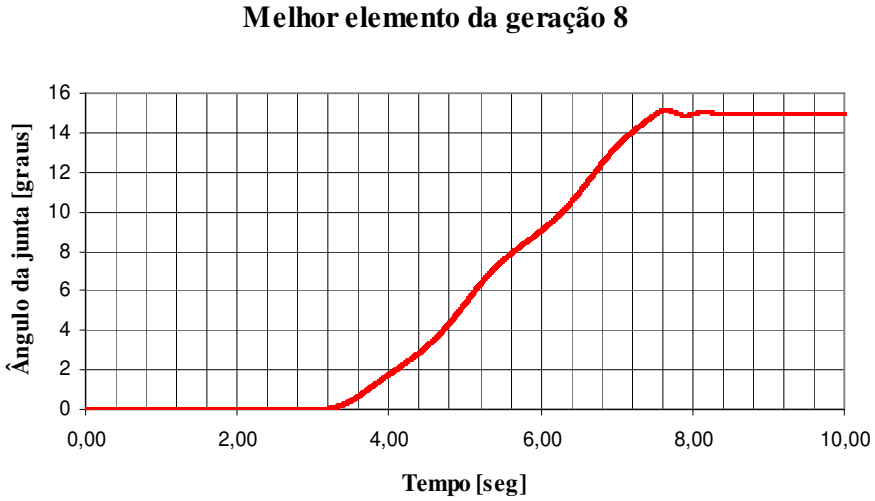

Fig. 4.21 – Saída do melhor controlador da geração 3.

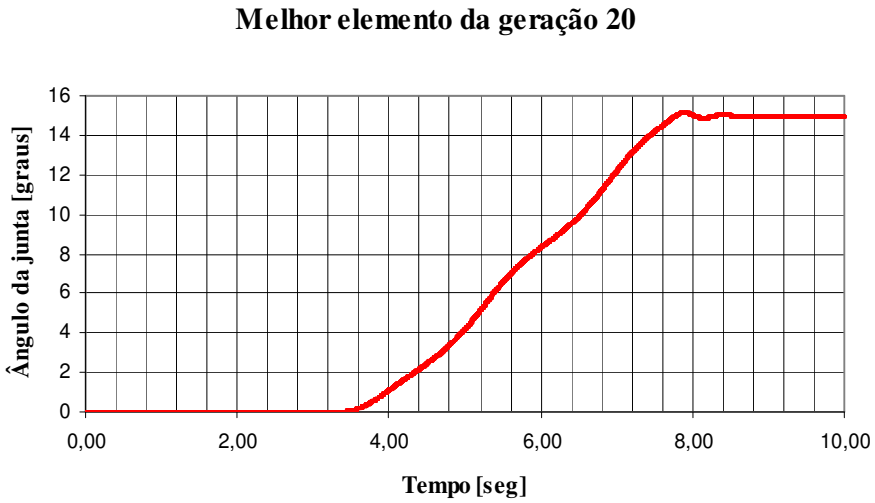

Fig. 4.22 - Saída do melhor controlador da geração 20.

Nota-se nos gráficos das figuras 4.19, 4.20, 4.21 e 4.22, que a principal diferença está entre os gráficos da terceira e oitava geração. Nas gerações 8 e 20, as diferenças não demonstram uma grande alteração no tempo para iniciar o movimento e no tempo de execução do movimento, principalmente por não serem tratados na função de *fitness* do algoritmo genético. A maior alteração pode ser vista nos gráficos das figuras abaixo.

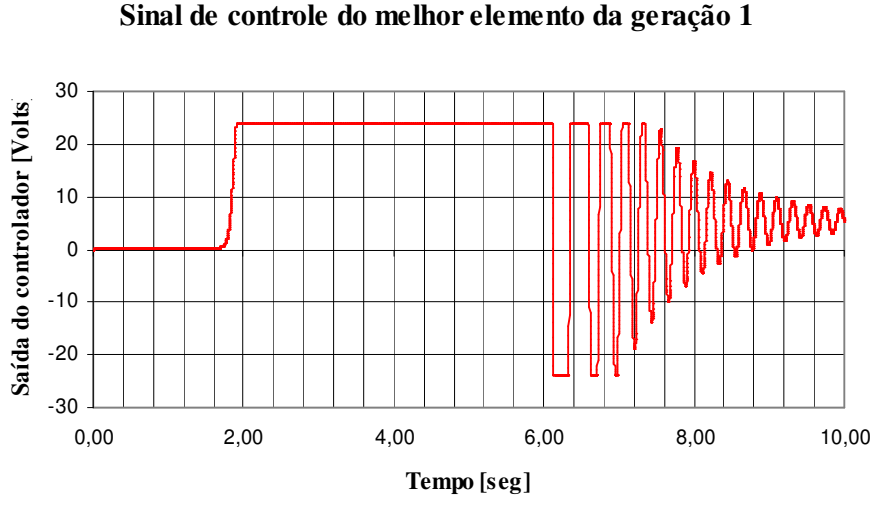

Fig. 4.23 – Sinal de controle do melhor controlador da geração 1.

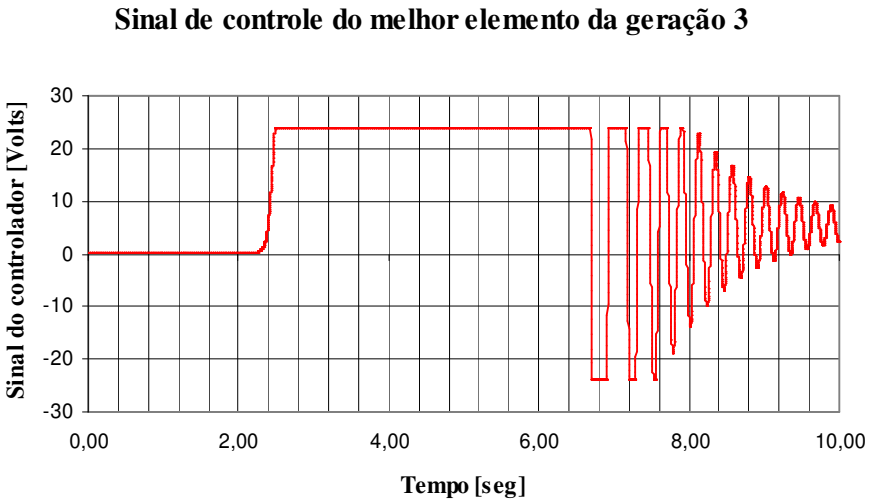

Fig. 4.24 – Sinal de controle do melhor controlador da geração 3.

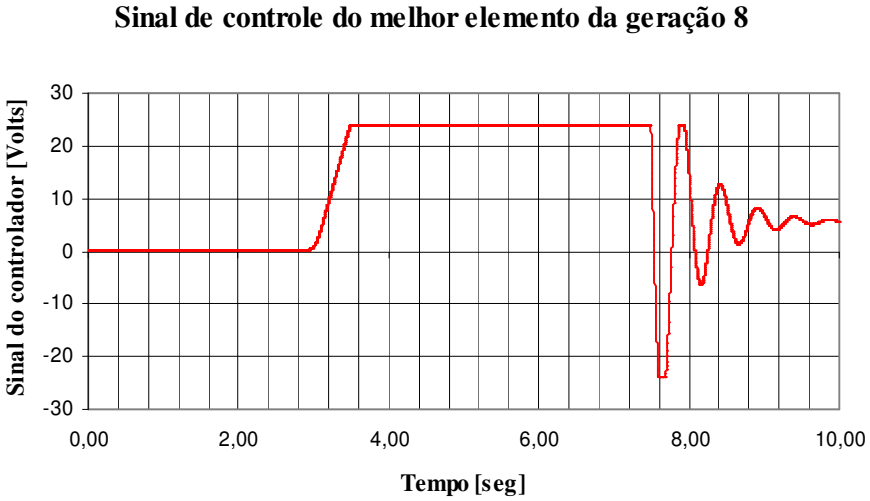

Fig. 4.25 – Sinal de controle do melhor controlador da geração 8.

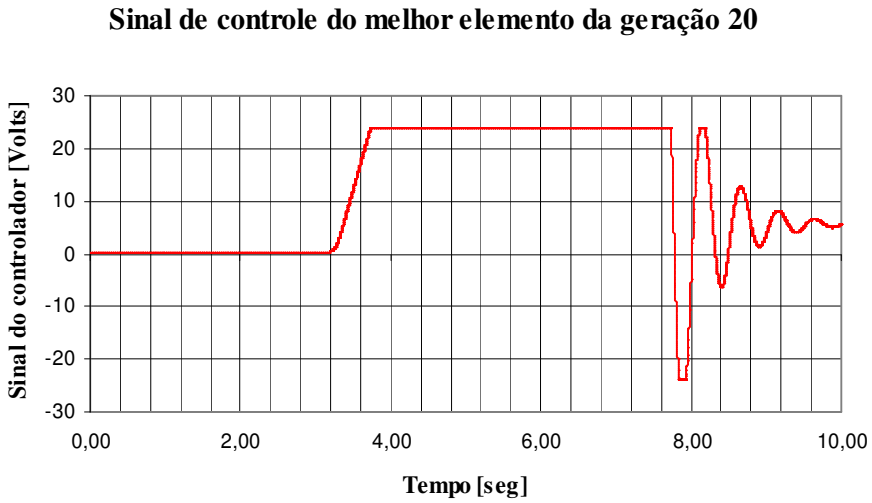

Fig. 4.26 – Sinal de controle do melhor controlador da geração 20.

Como mencionado anteriormente, o algoritmo genético foi utilizado para otimizar o esforço de controle necessário para atingir a referência. Observando-se as figuras 4.23, 4.24, 4.25 e 4.26, o esforço de controle apresentado para cada geração é menor, conforme a seqüência evolutiva do algoritmo genético.

Nota-se nestas figuras que o esforço de controle atinge rapidamente o seu limiar em uma determinada direção, que é de +24 Volts. Na figura 4.13, grande parte do controle é bastante parecido com o controle *bang-bang* (Ogata, 2000), fato observado pelo chaveamento entre +24 Volts e -24 Volts, sendo que este sinal de controle não é adequado para movimentar este tipo de mecanismo, principalmente por desgastar mais rapidamente a mecânica do processo. Durante a evolução, percebe-se pelos gráficos que esta característica vai diminuindo, principalmente por fornecer uma grande quantidade de energia ao processo, tornando o esforço de controle maior. Nas figuras 4.25 e 4.26, o gráfico já não apresenta, praticamente, a característica de um controlador do tipo *bang-bang*.

Apesar de durante a evolução o início do movimento ser mais tardio, o esforço de controle é realmente minimizado no decorrer da evolução.

4.8.1.2 – Experimento 2

Ao contrário do experimento anterior, neste todas as variáveis do controlador neural foram otimizadas, ou seja, número de neurônios da camada de entrada, número de neurônios da camada

escondida, topologia da rede e constantes K_1 e K_2. Para este caso, o *crossover* utilizado foi o *crossover* uniforme.

O algoritmo genético executado neste experimento teve 20 gerações. Cada geração possuindo 6 populações com 6 indivíduos cada, resultando em 36 indivíduos por geração. A referência a ser atingida pelos controladores neurais foi de 15 graus.

A evolução do algoritmo genético está mostrada na figura 4.27. O erro de posição do melhor indivíduo para as gerações 1, 6, 8 e 20 estão mostrados nas figuras 4.28, 4.29, 4.30 e 4.32, respectivamente. O sinal de controle para estas mesmas gerações está mostrado nas figuras 4.32, 4.33, 4.34 e 4.35, respectivamente. Os gráficos apresentados foram obtidos no mesmo ambiente do experimento anterior.

A estrutura da rede neural artificial selecionada pelo algoritmo genético para controlar o processo em questão possuiu os seguintes parâmetros: 20 neurônios na camada de entrada, 14 neurônio na camada escondida, topologia estática, $K_1 = 0.060808$ e $K_2 = 0.012528$.

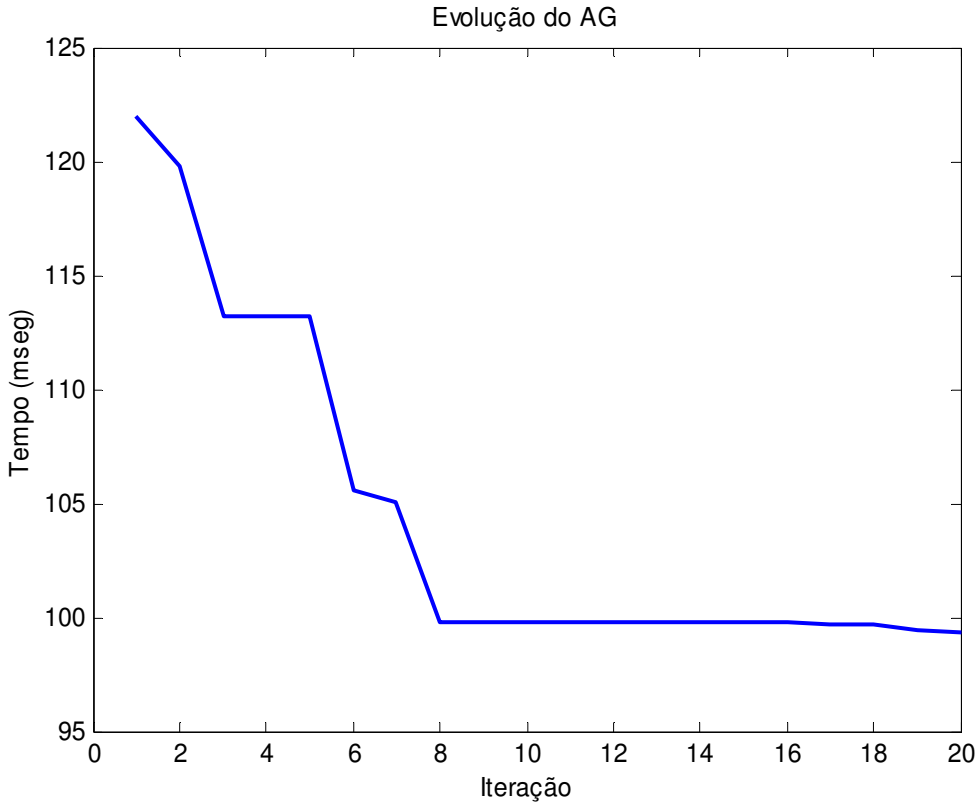

Fig. 4.27 - Evolução do AG.

Assim como no gráfico da figura 4.18, o esforço de controle observado neste experimento sempre tende a decrescer, tornando o processo mais eficiente quanto ao consumo de energia.

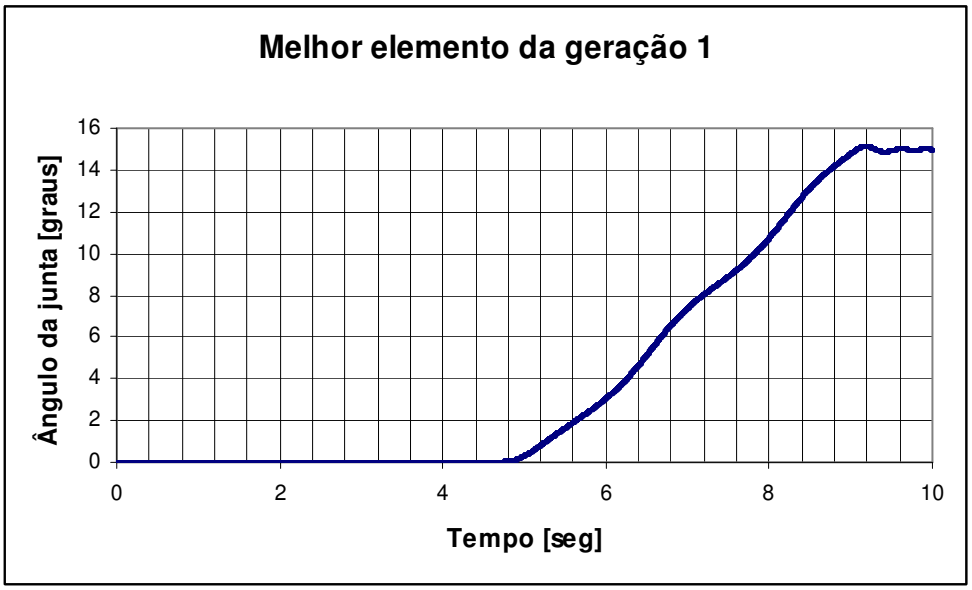

Fig. 4.28 – Saída do melhor controlador da geração 1.

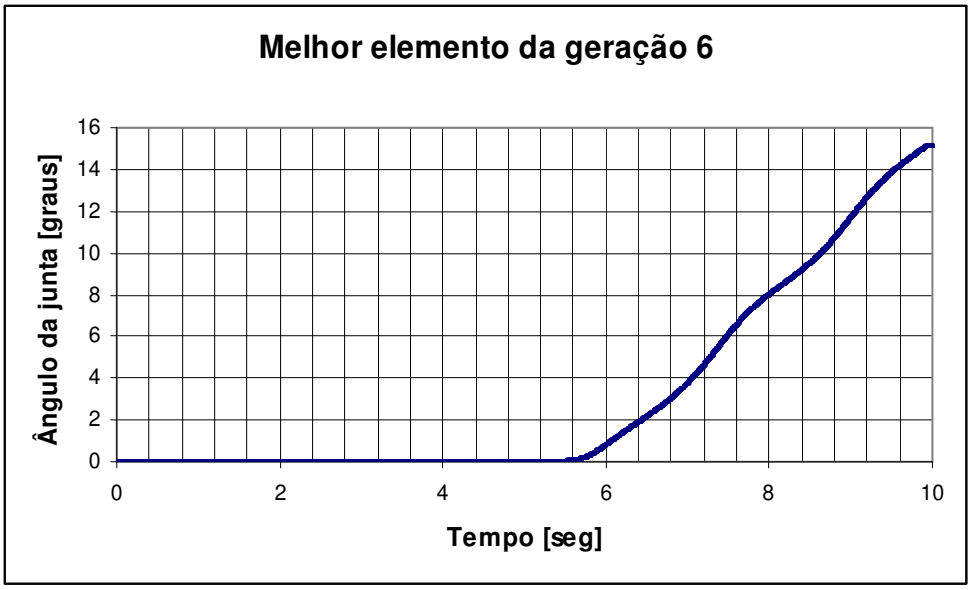

Fig. 4.29 – Saída do melhor controlador da geração 6.

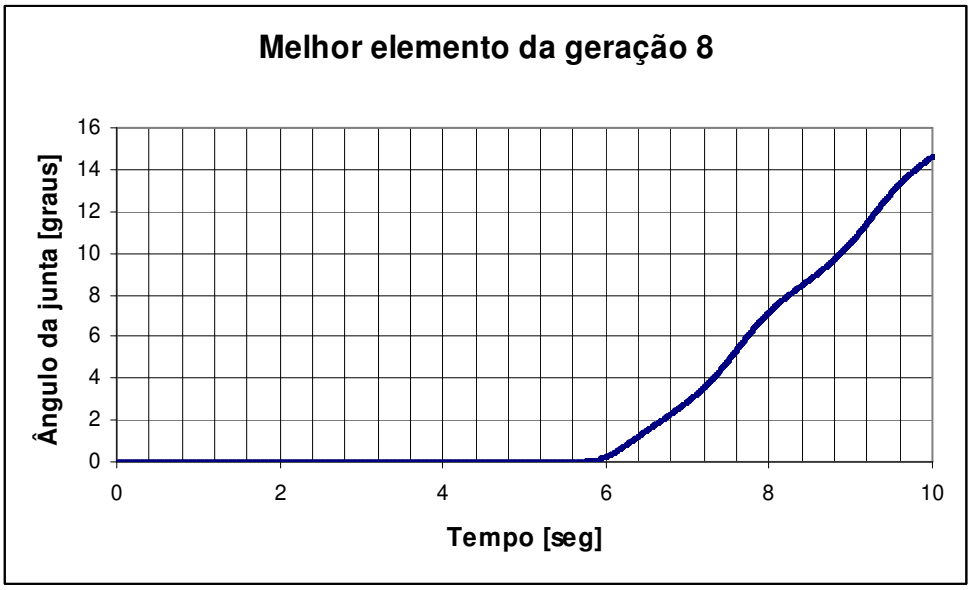

Fig. 4.30 – Saída do melhor controlador da geração 8.

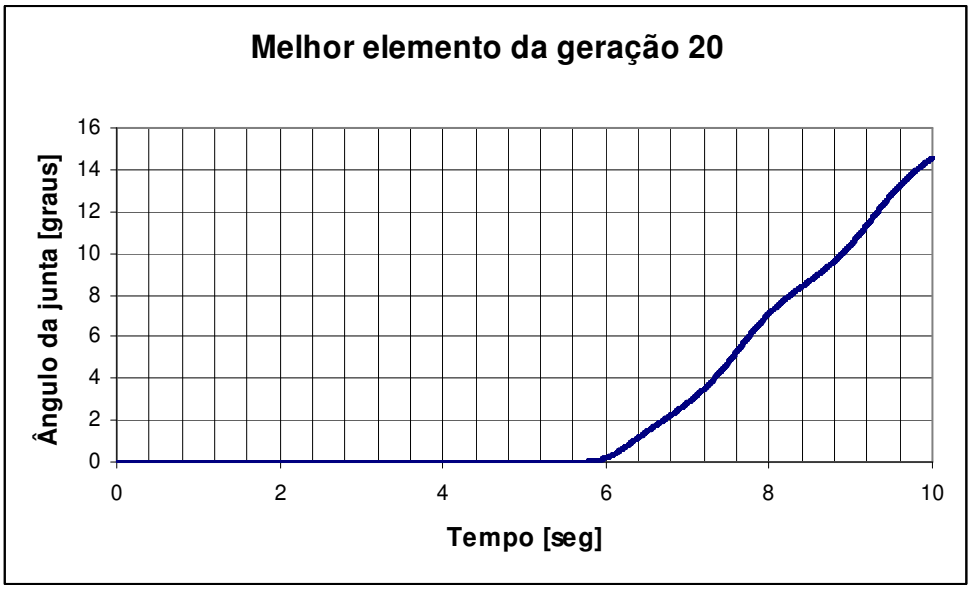

Fig. 4.31 - Saída do melhor controlador da geração 20.

Nas figuras 4.28, 4.29 e 4.31, observa-se que o tempo para iniciar o movimento começa sempre mais tarde conforme o algoritmo genético evolui. Neste trabalho só foi feito a otimização do sinal de controle e não do tempo de resposta. Para que possamos verificar a real aplicação do algoritmo genético, deve-se observar os gráficos dos sinais de controle.

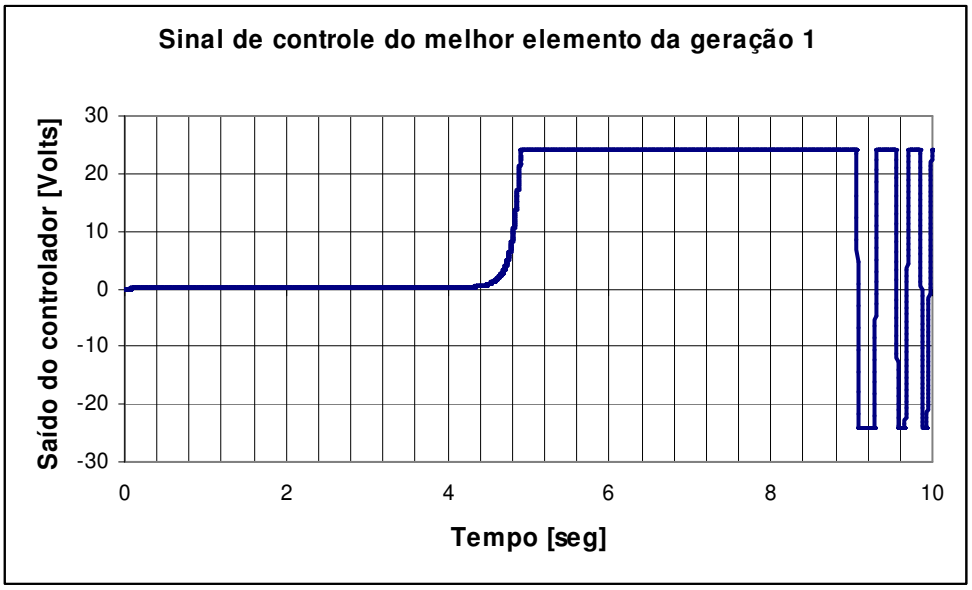

Fig. 4.32 – Sinal de controle do melhor controlador da geração 1.

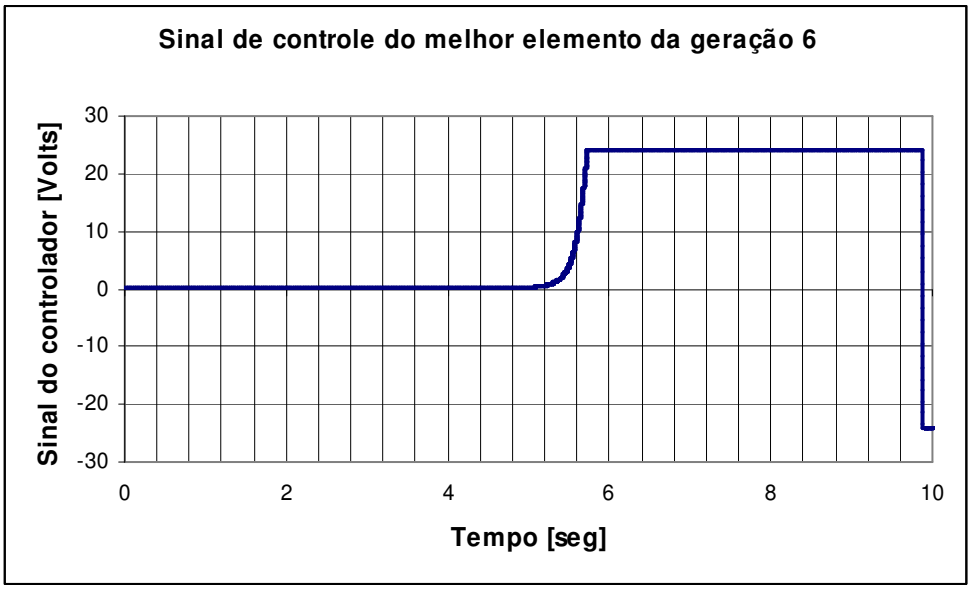

Fig. 4.33 – Sinal de controle do melhor controlador da geração 6.

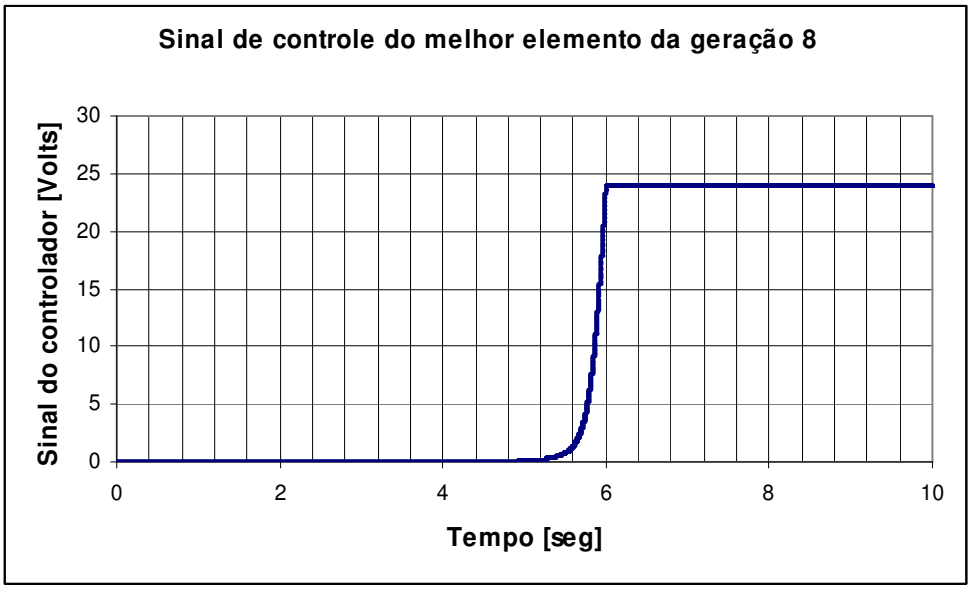

Fig. 4.34 – Sinal de controle do melhor controlador da geração 8.

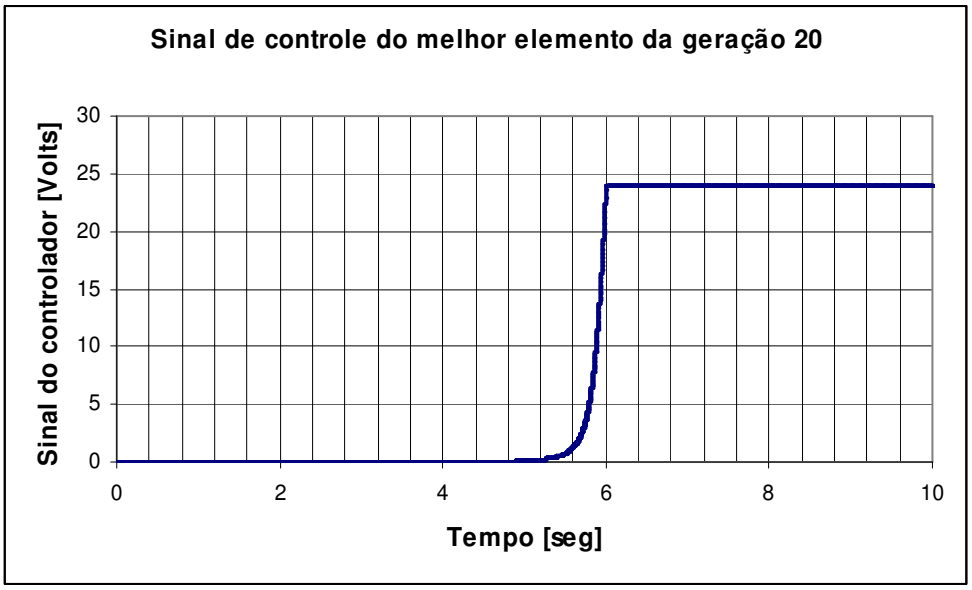

Fig. 4.35 – Sinal de controle do melhor controlador da geração 20.

Nota-se que existem diferenças visuais nas figuras 4.32, 4.33 e 4.34. No gráfico do esforço de controle da geração 1, figura 4.32, nota-se que o final do movimento é caracterizado pelo chaveamento entre o limiar positivo e negativo do controlador. Isto pode reduzir dramaticamente a vida útil do mecanismo do pêndulo invertido acionado. Nas figuras 4.33 o chaveamento só acontece uma única vez, enquanto que nas figuras 4.34 e 4.35 o chaveamento não ocorre mais.

Percebe-se nos dois últimos experimentos que as topologias das redes neurais artificias

apresentam uma grande diferença em relação ao número de neurônios nas camadas de entrada e intermediárias. Isto informa que não existe uma topologia definida, sendo que os pesos iniciais podem ser o fator principal desta alteração drástica entre as redes neurais artificiais determinadas pela execução do algoritmo genético. Porém, pode ainda ter ocorrido um outro caminho nas buscas pelas soluções, ocasionando soluções diferentes, mas próximas.

CAPÍTULO 5

CONCLUSÃO

Conforme visto através dos estudos realizados e mostrados nesta tese, o algoritmo genético é uma ferramenta que pode ser utilizada tanto para o planejamento, quanto para o controle de trajetórias de robôs em diferentes estágios, bastando para isso que os problemas sejam colocados na forma em que o algoritmo genético funciona, ou seja, codificando as trajetórias a serem pesquisadas para que as operações genéticas possam ser aplicadas. Como tais experimentos foram computados diversas vezes, notou-se que o tempo que o algoritmo genético leva para encontrar uma solução pode variar bastante. Isto se deve ao fato dele basear-se em probabilidades para a execução das operações genéticas, implicando que uma execução do algoritmo terá grande probabilidade de ser diferente de outra execução. Com isto, um experimento não poderá ser repetido facilmente, quando se trata de seguir o mesmo caminho de procura pela solução, a não ser que diversas variáveis sejam preservadas no ambiente computacional, porém todas as execuções do algoritmo genético levam a soluções satisfatórias, sendo todas próximas umas das outras.

Apesar do planejamento de trajetória no plano cartesiano ter somente 4 pontos intermediários, experimentos podem ser desenvolvidos com um número maior de pontos, sem alterações significativas no programa de geração de trajetória.

Durante a fase de programação dos algoritmos genéticos, optou-se, inicialmente, pela linguagem C do compilador Borland C++ 5.5, porém notou-se que a evolução do algoritmo era lenta, devido a pouca contribuição do gerador de números aleatórios deste compilador. Para que houvesse ganho na velocidade de convergência para uma solução satisfatória através do algoritmo genético, todos os programas criados em linguagem C foram convertidos para a linguagem Java, sendo está escolhida por apresentar funções de geração de números aleatórios mais elaboradas.

Outro problema que se encontra é a limitação dos algoritmos genéticos para resolver problemas que não tenham restrição de tempo, já que não se pode garantir uma solução viável em um determinado tempo. Este problema já está sendo estudado pela comunidade acadêmica, principalmente por pesquisadores ligados à microeletrônica, onde se espera que os algoritmos

genéticos sejam realizados em circuitos integrados, tal como aconteceu com outras técnicas importantes de inteligência artificial. Com isto, espera-se que o tempo de execução de um algoritmo genético seja minimizado o suficiente para ser usado em busca de soluções de problemas com restrições fortes.

Porém, do ponto de vista de planejamento de tarefas, um algoritmo genético possui tempo de planejamento minimizado, uma vez que não é necessária a confecção de modelos matemáticos e físicos tão completos quanto os requisitados por outras técnicas mais tradicionais. Isto diminui sensivelmente o tempo de preparação necessário para iniciar a resolução de vários problemas.

As redes neurais artificiais foram testadas para o controle de posição de uma mesa XY, um sistema que possui poucas não linearidades, e para o controle de posição de uma junta de um braço de robô, onde existe a presença de um grande número de não linearidades. Uma grande vantagem de se utilizar rede neural artificial para este tipo de controle é o tempo de projeto para a construção de um controlador, uma vez que, assim como acontece com os algoritmos genéticos, os controladores apresentados são simples, de fácil programação, requerem pouco rigor matemático, e o conhecimento prévio do modelo do processo não é necessário. Como nenhum tipo de treinamento é a rigor necessário, a rede neural artificial pode "aprender" no decorrer das ações de controle, enquanto tenta minimizar os erros existentes nas malhas de controle. Um dos problemas deste controlador é o tempo para se iniciar o movimento. Percebe-se que em todos os experimentos o tempo levado para iniciar o movimento é bastante longo, chegando em alguns casos a ser superior a 50% do tempo total do experimento.

Este tipo de solução possui o inconveniente de ter pelo menos 5 (cinco) variáveis a serem definidas. Tais variáveis são: topologia da rede, número de neurônios na camada intermediária e na camada de entrada, e as constantes K_1 e K_2. Inicialmente, os valores para estas variáveis foram fixados e testados empiricamente, para isso várias execuções foram necessárias na determinação de valores aceitáveis. Porém, percebeu-se que os algoritmos genéticos podiam ser utilizados para encontrar os valores automaticamente, sem a intervenção humana. Com isto, construiu-se um algoritmo genético com codificação em seus elementos de todas as cinco variáveis, automatizando a busca por valores satisfatórios.

O principal problema de se utilizar o algoritmo genético para a busca de um controlador é a necessidade da construção de um simulador para o processo a ser controlado. Com isto, perde-se um pouco o enfoque eminentemente prático deste trabalho. Isto visa, principalmente, não

danificar os sistemas eletromecânicos construídos. Porém, através dos resultados apresentados no capítulo 4, observa-se que o principal objetivo, que é diminuir o esforço de controle do controlador neural, foi bem sucedido.

Como trabalhos futuros, pode-se aumentar o número de obstáculos na geração de trajetórias, tendo como sugestão a separação do ambiente de trabalho em várias partes onde se possa definir somente um obstáculo. Caso não seja possível essa separação, ou seja, se os obstáculos não puderem gerar um plano côncavo, mais graus de liberdade na trajetória podem ser adicionados através do aumento do número de pontos intermediários das trajetórias em estudo.

Para os controladores neurais é necessário que o tempo de inércia inicial do movimento seja diminuído. Isto poderá ser feito por um algoritmo genético, através da função de *fitness*, onde poderá ser verificado o efeito da estrutura da rede neural artificial na inércia inicial. Os algoritmos genéticos poderão testar novas estruturas, principalmente com um maior número de camadas escondidas na rede neural artificial de controle.

Com as pesquisas em andamento no LSMR da FEEC, UNICAMP, percebe-se que tanto os algoritmos genéticos quanto as redes neurais artificiais podem ser integradas em um circuitos eletrônicos digitais e analógicos, do tipo FPGA, alcançando tempos de processamento bastante inferiores aos apresentado neste trabalho, além de serem reprogramados facilmente para processarem outras topologias de algoritmos genéticos e redes neurais artificiais. Isto será possível graças as estruturas simples dos algoritmos genéticos e rede neurais artificiais. Com isto, os tempos de processamentos envolvidos cairiam para a ordem de micro segundos, tornando totalmente viável a aplicação dos algoritmos genéticos para a otimização de parâmetros do controlador e planejamento em tempo real.

Durante todo o desenvolvimento do trabalho, prezou-se pela determinação de soluções para problemas envolvidos com robótica com o mínimo de intervenção humana nos procedimentos, e partindo-se dos princípios que não há modelos dos processos conhecidos previamente. O apelo, portanto, é a questão prática envolvida. Por outro lado, se houverem modelos conhecidos, ou estimados, isso pode ser também considerado, e muito provavelmente contribuem para a obtenção de soluções mais rápidas e estatisticamente mais comportadas e semelhantes.

Para finalizar, notou-se que o algoritmo genético pode ser uma ferramenta bastante útil para o planejamento e otimização de tarefas, atuando em um nível mais alto no processo de controle de plantas, como, por exemplo, robôs industriais, facilitando o desenvolvimento de ferramentas

de planejamento. Já as redes neurais artificiais foram utilizadas em uma estrutura simples, mostrando que nem sempre é necessária a utilização de estruturas complexas para obter-se bons resultados em controle, inclusive práticos.

REFERÊNCIAS BIBLIOGRÁFICAS

Aleksander, I. e Morton, H. (1990), 'An introduction to neural computing', Chapman & Hall. Londres.

Astrom, K. J. (1995), 'Adaptive Control', Addison-Wesley.

Chen, L-H. e Chiang, C-H. (2003), 'New Approach to Intelligent Control Systems With Self-Exploring Process', IEEE Transactions on Systems, Man and Cybernetics – Part B: Cybernetics, Vol. 33, No. 1, pp. 56-66.

Chen, L. e Narendra, K. S. (2000), 'Intelligent control using neural networks and multiple models', Proceedings of the 41st IEEE Conference on Decision and Control, Volume: 2 , 10-13 Dec. 2002 Page(s): 1357 -1362

Chen, S. Y. e Li, Y. F. (2004), 'Automatic Sensor Placement for Model-Based Robot Vision', IEEE Transaction on System, Man and Cybernetics – Part B: Cybernetics, Vol. 34, No. 1, pp. 393-408.

Corke, P. I. e Armstrong-Hélouvry, B. (1994), 'A Search for Consensus Among Model Parameters for Puma 560 Robot', Proceedings of IEEE International Conference on Robotics and Automation, Vol. 2, pp. 1608-1613.

Deitel, H. M. e Deitel, P. J. (2002), 'Java Como Programar', Bookman Companhia, 4ª Edição. Brasil.

Denavit, J. e Hartenberg, R. S. (1955), 'A kinematics notation for lower pair mechanisms based on matrices', ASME J. App. Mech., Vol. 77, pp. 215-221.

Feng, G. (1997), 'A New Stable Tracking Control Scheme for Robotic Manipulators', IEEE Transactions on Systems, Man and Cybernetics – Part B: Cybernetics, Vol. 27, No. 3, pp. 510-516.

Fu, K. S., Gonzalez, R. C. e Lee, C. S. G. (1987), 'Robotics, Control, Sensing, Intelligence', McGraw-Hill Book Company.

Gao, Y. e Er, M. J. (2003), 'Online Adaptive Fuzzy Neural Identification and Control of a Class of MIMO Nonlinear Systems', IEEE Transactions on Fuzzy Systems, Vol. 11, No. 4, pp. 462-477.

Goldberg, D. E. (1989), 'Genetic Algorithms in Search, Optimization and Machine Learning', Addison-Wesley.

Gupta, P. e Sinha, N. K. (2000), Intelligent Control of Robot Manipulator: experimental study using neural networks', Mechatronics, Vol. 10, pp. 289-305.

Fierro, R. e Lewis, F. L. (1998), 'Control of a Nonholonomic Mobile Robot using Neural Networks', IEEE Transactions on Neural Networks, Vol. 9, No. 4, pp. 589-600.

Fogel, D. B. (2002), 'Blondie 24 – Playing at The Edge of AI', Morgan Kaufmann Publishers, Academic Press, USA.

Ge, S. S., Hang, C. C. e Woon, L. C. (1997), 'Adaptive Neural Network Control of Robot Manipulators in Task Space', IEEE Transactions on Industrial Electronics, Vol. 44, No. 6, pp. 746-752.

Gill, M. A. C. e Zomaya, A. Y. (1998), 'A Parallel Collision-Avoidance Algorithm for Robot Manipulators', IEEE Concurrency, pp. 68-78.

Gutiérrez, L. B., Lewis, F. L. e Lowe, J. A. (1998), 'Implementation of a Neural Network Tracking Controller for a Single Flexible Link: Comparison with PD and PID Controllers', IEEE Transactions on Industrial Electronics, Vol. 45, No. 2, pp. 307-318.

Haykin, S. (1996), 'Neural Networks: A Comprehensive Foundation', Macmillan College Publishing Company.

Hitam, M. S. e Gill, K. F. (2000), 'Identification and Control of a Robot Using a Neural Network', Proceedings of TENCOM 2000, Volume 3, pp. 479-483.

Holland, J. H. (1992), 'Adaptation in Natural and Artificial Systems', University of Michigan Press.

Hong, Z., Kaifang, D. e Tingqi, L. (2002), 'A Online-Trained Neural Network Controller for Electro-hydraulic Servo System', Proceedingd of the 4th World Congress on Intelligent Control and Automation. pp. 2983-2986.

Joo, M. e Liew, K. C. (1997), 'Control of Adept One SCARA Robot Using Neural Networks', IEEE Transactions on Industrial Electronics, Vol. 44, No. 6, pp. 762-768.

Jorde, L. B., Carey, J. e White, R. L. (1995), 'Medical Genetics', Mosby-Year Book, Inc, USA.

Juidette, H. e Youlal, H. (2000), 'Fuzzy dynamic path planning using genetic algorithms', IEEE Electronic Letters, Vol. 36, No. 4, pp. 374-376.

Jung, S. e Hsia, T. C. (1998), 'Neural Network Impedance Force Control of Robot Manipulator', IEEE Transaction on Industrial Electronics, Vol. 45, N. 3, pp. 451-461.

Jung, S. e Hsia, T. C. (2000), 'Robust Neural Force Control Scheme Under Uncertainties in Robot Dynamics and Unknown Environment', IEEE Transactions on Industrial Electronics, Vol. 47, No. 2, pp. 403-412.

Jungbeck, M. (2002), 'Implementação de Controladores Neurais de Kim-Lewis-Dawson com Parâmetros Otimizados por Algoritmos Genéticos'. Orientador: MARCONI KOLM MADRID, Tese de mestrado. LSMR – DSCE – FEEC – UNICAMP, Campinas – SP – Brasil.

Kouvelis, P. e Yu, G. (1997), 'Robust discrete optimization and its applications', Kluwer.

Koza, J. R. (1992), 'Genetic Programming: On the Programming of Computers by Means of Natural Selection', The MIT Press.

Kosko, B. (1992), 'Neural Networks and Fuzzy Systems - A Dynamical Systems Approach to Machine Intelligence', Prentice-Hall, London.

Kwan, C., Lewis, F. L. e Dawson, D. M. (1998), 'Robust Neural-Network Control of Rigid-Link Electrically Driven Robots', IEEE Transactions on Neural Network, Vol. 9, No. 4, pp. 581-588.

Lee, C., Eom, T. e Lee, J. (2002), 'Neuro-adaptive control of mobile manipulators based on compensation of approximation error', IEEE Electronic Letter, Vol. 38, No. 16, pp. 935-936.

Lee, S. e Cho, S. (2001), 'Emergent Behaviors of a Fuzzy Sensory-Motor Controller Evolved by Genetic Algorithm', IEEE Transactions on System, Man and Cybernetics – Part B: Cybernetics. Vol. 31, No. 6, pp. 919-929.

Leung, F. H. F., Lam, H. K., Ling, S. H. e Tam, P. K. S. (2004), 'Optimal and Stable Fuzzy Controllers for nonlinear Systems Based on an Improved Genetic Algorithm', IEEE Transactions on Industrial Electronics, Vol. 51, No. 1, pp. 172-182.

Liu, J. e Brooke, M. (1999), 'A Fully Parallel Learning Neural Network Chip for Real-time Control', pp. 2323-2328.

Lunze, J. (1989), 'Robust multivariable feedback control', Prentice-Hall, New York.

McCulloch, W. S. e Pitts, W. (1943), 'A logical calculus of the ideas immanet innervous activity', Bulletin of Mathematical Biophysics, Vol 5, pp. 115-133.

Madrid, M. K. (1994), 'Controle de trajetórias contínuas por seccionamento em sub-trajetórias usando inteligência artificial num robô multi-tarefas', Doctoral Thesis, Faculdade de Engenharia Elétrica e de Computação - UNICAMP. Cidade Universitária Zeferino Vaz, Campinas – SP – Brazil.

Madrid. M. K. e Badan, A. G. P. (1997), 'Heuristic search method for continuous-path tracking optimization on high-performance industrial robots', Control Eng, Practice, Vol. 5, Nº 9, pp. 1261-1271.

Monteiro, D. C. (1996), 'Implementação de Controladores Fuzzy e Neurais', Master Thesis, Mestrado em Engenharia Elétrica – UFPA – Belém – PA – Brazil.

Monteiro, D. C. e Takita, K. (1996), 'Position Control of XY Table Utilizing Artificial Neural Networks', The Third International Conference on Motion and Vibration Control.

Monteiro, D. C. e Takita, K. (1996a), 'Controlador Proporcional Adaptativo Utilizando Redes Neurais Artificiais, XI Congresso Brasileiro de Automática. 1996.

Monteiro, D.C. e Madrid, M. K. (1999), 'Planning of Robot Trajectories with Genetic Algorithms', RoMoCo'99 – IEEE First Workshop on Robot Motion and Control, pp. 223-228, Polonia.

Monteiro, D. C.; Madrid, M. K.; Takita, K. (2001); 'Selection of Neural Controllers Using Genetic Algorithms', 5th World Multiconference on Systemics, Cybernetics and Informatics - SCI 2001 and the 7th International Conference on Information Systems Analysis ans Synthesis - ISAS 2001, Vol. 1, pp.453-458, Orlando, ESTADOS UNIDOS, 2001

Moon, S. e Kong, S. (2001), 'Block-Based Neural Networks', IEEE Transactions on Neural Network, Vol. 12, No. 2, pp. 307-317.

Narendra, K. S. e Parthasarathy, K. (1990), 'Identification and Control of Dynamical Systems Using Neural Networks', IEEE Trans. Neural Networks, Vol. 1, Nº 1, pp. 4-27.

Ogata, K. (2000), 'Engenharia de Controle Moderno', Editora LTC, 3ª Edição.

Otto, P. A., Otto, P. G. e Frota-Pessoa, O. (1998), 'Genética Humana e Clínica', Editora Roca.

Patino, H. D., Carelli, R. e Kuchen, B. R. (2002), 'Neural Networks for Advanced Control of Robot Manipulators', IEEE Transactions on Neural Networks, Vol. 13, No. 2, pp. 343-354.

Romero, S. M. B. (2000), 'Fundamentos da Neurofisiologia Comparada: Da recepção à Integração', Editora Holos, 2000.

Rumelhart, D. E., Hinton, G. E. e Willians, R. J. (1986), 'Learning representations by back-propagation erros', Nature (London), 323, 533-536.

Shepherd, G. M. e Koch, C. (1990), 'The Synaptic Organization of the Brain', Oxford University Press. New York.

Spong, M. W. e Vidyasagar, M. (1989), 'Robot Dynamics and Control', John Wiley & Sons.

Sousa, M. A. T. (2000), 'Otimização de Controladores Nebulosos de Takagi-Sugeno Utilizando Algoritmos Genéticos', Orientador: MARCONI KOLM MADRID, Tese de mestrado, LSMR – DSCE – FEEC – UNICAMP, Campinas – SP – Brasil.

Sun, F. C., Sun, Z. Q. e Chen, Y. B. (2000), 'Neural adaptive tracking controller for robot manipulators with unknown dynamics', IEE Proceedings – Control Theory Application, Vol. 147, no. 03, May 2000, pp. 366-370.

Swift, D. C., Kaufman, H. e Cummings, S. T. (1993), 'Direct Model Adaptive Reference Control of Puma Manipulator', IEEE International Conference on Robotics and Automation. Volume 2, pp. 346-351.

Tocci, R. J. e Widmer, N. S. (2003), 'Sistemas Digitais', Editora Pearson do Brasil, 8ª Edição.

Visioli, A. e Legnani, G. (2002), 'On the Trajectory Tracking Control of Industrial SCARA Robot Manipulators', IEEE Transactions on Industrial Electronics', Vol. 49, No. 1, pp. 224-232.

Wai, R. e Lee, M. (2004), 'Intelligent Optimal Control of Single-link Flexible Robot Arm', IEEE Transactions on Industrial Electronics, Vol. 51, No. 1, pp. 201-220.

Watanabe, K. (1992), 'Adaptive estimation and control: Partitioning approach', Prentice-Hall, New York.

Xia, Y. e Wang, J. (2001), 'A Dual Neural Network for Kinematic Control of Redundant Robot Manipulators', IEEE Transactions on Systems, Man and Cybernetics – Part B: Cybernetics, Vol. 31, No. 1, pp. 147-154.

Xiao, J., Michalewicz, Z., Zhang, L. e Trojanowski, K. (1997), 'Adaptive Evolutionary Planner/Navigator for Mobile Robots', IEEE Transactions on Evolutionary Computation, Vol. 1, No. 1, pp. 18-28.

Yildirim, S. (2001), 'Neural network controller for cooperating robots', Electronic Letters, Vol. 37, No. 22, pp. 1351-1352.

Yildirim, S. (2002), 'Robot trajectory control using neural networks', Electronics Letters, Vol. 38, No. 19, pp. 1111-1113.

Zeman, V., Patel, R. V. e Khorasani, K. (1997), 'Control of a Flexible-Joint Robot Using Neural Networks', IEEE Transactions on Control, Systems Technology, Vol. 5, No. 4, pp. 453-462.

Zhang, Y. e Wang, J. (2004), 'Obstacle Avoidance for Kinematically Redundant Manipulators Using a Dual Neural Network', IEEE Transactions on Systems, Man and Cybernetics – Part B: Cybernetics, Vol. 34, No. 1, pp. 752-759.