UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA MECÂNICA

Geração de Trajetórias para Robôs Móveis Autônomos Via Redes Neurais Artificiais

Autor : **José Fábio Abreu de Andrade**
Orientador: **Prof. Dr. Douglas Eduardo Zampieri**
Co-Orientador: **Prof. Dr. André Mendeleck**

36/01

UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA MECÂNICA
DEPARTAMENTO DE MECÂNICA COMPUTACIONAL

Geração de Trajetórias para Robôs Móveis Autônomos Via Redes Neurais Artificiais

Autor : **José Fábio Abreu de Andrade**
Orientador: **Prof. Dr. Douglas Eduardo Zampieri**
Co-Orientador: **Prof. Dr. André Mendeleck**

Curso: Engenharia Mecânica.
Área de concentração: Mecânica dos Sólidos e Projeto Mecânico

Dissertação de mestrado apresentada à comissão de Pós Graduação da Faculdade de Engenharia Mecânica, como requisito para obtenção do título de Mestre em Engenharia Mecânica.

Campinas, 2001
S.P. - Brasil

"O rio atinge os objetivos porque aprendeu a contornar os obstáculos."

André Luiz

Resumo

ANDRADE, José Fábio Abreu de, *Geração de Trajetórias para Robôs Móveis Autônomos Via Redes Neurais Artificiais*, Faculdade de Engenharia Mecânica, Universidade Estadual de Campinas, 2001. 150 p.. Dissertação (Mestrado).

A navegação autônoma de veículos há muito desperta o interesse de pesquisadores, principalmente na área de inteligência artificial. Um dos problemas de grande importância é a determinação de uma trajetória, pois dela depende o veículo para que possa navegar pelo ambiente, evitando colidir com obstáculos, até alcançar uma ou mais posições pré-estabelecidas. Neste trabalho, é proposto um sistema para a geração de trajetórias, utilizando redes neurais artificiais, aplicado na navegação de robôs móveis em ambientes desestruturados. Numa primeira etapa foi abordado o uso de uma única rede neural do tipo Perceptron de Múltiplas Camadas (MLP), com uma arquitetura 28-20-2, para a resolução do problema. Entretanto, o sistema proposto apresentou um alto custo computacional para o treinamento da rede, quando da generalização das condições de trabalho, uma vez que o número de padrões utilizados era muito elevado. Numa etapa posterior, esta rede neural MLP foi substituída por um sistema híbrido formado por funções lógicas e por 36 redes neurais MLP. Os resultados da simulação computacional, apresentados na forma de gráficos, comprovam a eficiência do método para ambientes desestruturados.

Palavras Chave

Redes Neurais Artificiais, Robótica, Navegação, Simulação Computacional.

Abstract

ANDRADE, José Fábio Abreu de, *Trajectory Generation for Autonomous Mobile Robots Using Artificial Neural Networks*, Faculdade de Engenharia Mecânica, Universidade Estadual de Campinas, 2001. 150 p.. Dissertação (Mestrado).

The autonomous navigation of vehicles has been an area of great interest for researchers for a long time, mainly in the area of artificial intelligence. One of the most interesting problems is that related to the trajectory generated for a vehicle, which shall be guided in an environment, avoiding colliding with obstacles in order to reach predefined positions. This thesis presents a methodology to generate trajectories using neural networks applied to mobile robot navigation. In a first approach, it has been used a single MLP neural network, with a 28-20-2 architecture, to solve this problem. However, this system presented a high computational effort for training the neural network, since the number of patterns increased due to the generalization of the environments. In a second approach this neural network has been improved to a hybrid system composed of logic operations and 36 MLP neural networks. The results of computer simulation presented graphically showed that the proposed methodology were efficient to guide the robot in unknown environments.

Key Words

Artificial Neural Networks, Robotic, Navigation, Computational Simulation.

Indice

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

An24g

Andrade, José Fábio Abreu de
Geração de trajetórias para robôs móveis autônomos via redes neurais artificiais / José Fábio Abreu de Andrade. -- Campinas, SP: [s.n.], 2001.

Orientadores: Douglas Eduardo Zampieri, André Mendeleck.
Dissertação (mestrado) - Universidade Estadual de Campinas, Faculdade de Engenharia Mecânica.

1. Redes neurais (Computação). 2. Robôs móveis. 3. Inteligência artificial. 4. Detectores. 5. Sistemas de veículos auto-guiados. I. Zampieri, Douglas Eduardo. II. Mendeleck, André. III. Universidade Estadual de Campinas. Faculdade de Engenharia Mecânica. IV. Título.

UNIVERSIDADE ESTADUAL DE CAMPINAS

FACULDADE DE ENGENHARIA MECÂNICA

DEPARTAMENTO DE MECÂNICA COMPUTACIONAL

DISSERTAÇÃO DE MESTRADO

Geração de Trajetórias para Robôs Móveis Autônomos Via Redes Neurais Artificiais

Autor : **José Fábio Abreu de Andrade**

Orientador: **Prof. Dr. Douglas Eduardo Zampieri**

Co-Orientador: **Prof. Dr. André Mendeleck**

Prof. Dr. Douglas Eduardo Zampieri, Presidente
FEM/UNICAMP

Prof. Dr. Robson Pederiva
FEM/UNICAMP

Prof. Dr. Fernando José Von Zuben
FEEC/UNICAMP

Campinas, 28 de Maio de 2001

Dedicatória:

Dedico este trabalho aos meus pais, que sempre me ensinaram que a maior herança que os pais podem deixar para os filhos é a educação.

Agradecimentos

Aos meus pais, minha irmã e Viviany pelo apoio, incentivo e compreensão que foram importantes para a realização deste trabalho. Ao meu irmão, companheiro em todos os momentos da minha vida, pela ajuda em todas as fases do mesmo.

Ao Prof. Douglas primeiro pela oportunidade e também pela forma justa e coerente de orientação. Muitas pessoas acham que o orientador deve estar sempre disponível no momento em que o orientado precise, mas na minha opinião em relação ao tempo, o que vale é a qualidade e não a quantidade. Assim, posso afirmar que todo tempo dispensado pelo Prof. Douglas para a minha orientação foi suficiente para o resultado final do trabalho. Obrigado pelos conselhos e pelas boas horas de conversa que nós tivemos.

Ao Prof. André Mendeleck pelas boas idéias e também pela grande contribuição na realização deste trabalho.

Ao Prof. Fernando Von Zuben pelos ensinamentos e pela atenção dispensada.

Aos meus amigos da Bahia, Adilson, Adriana e Mildo pelo grande incentivo.

Aos meus amigos de Campinas, Sílvia, Egnilson, Kelly, Zilda, Sérgio, John, Justo, Arcos e Tissiana pelo incentivo e pelos bons momentos, que ajudaram a suportar as saudades da Bahia.

Também aos meus colegas baianos de moradia, Alex, Daniel Motta e Ubatan por me suportar durante muito tempo.

Ao amigo Marco Antonio, precursor do êxodo baiano para a FEM, pelos ensinamentos de programação e de redes neurais, e também pelos incentivos.

Aos meus amigos e colegas de trabalho, Prof. André Bezerra e Danilo Benzatti pela grande ajuda e contribuição para a realização deste trabalho.

Ao Dr. Belisário N. Huallpa pela ajuda com as redes neurais.

A todos os meus colegas do DMC que de alguma forma contribuíram para o meu trabalho.

E finalmente para a Fundação Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) pela ajuda financeira para a realização deste trabalho.

Lista de Figuras

Lista de Tabelas

Nomenclatura

RN → Rede Neural.

Se → Informação dos Sensores.

DP → Direção Preferencial.

S → Direção corrente de movimento.

$\mathbf{W_{j,d}}$ → Peso da j-ésimo conexão sináptica, na direção d (Mendeleck, 1995).

$\mathbf{x_i}$ → i-ésima entrada de uma RN.

$\mathbf{w_i}$ → i-ésimo peso sináptico de uma RN.

$\mathbf{x_0}$ → Bias ou entrada de polarização.

n → Número de neurônios da camada de entrada de uma RN.

m → Número de neurônios da camada intermediária de uma RN.

m1 → Número de neurônios da primeira camada intermediária de uma RN.

m2 → Número de neurônios da segunda camada intermediária de uma RN.

p → Número de neurônios da camada de saída de uma RN.

q → q-ésimo neurônio.

z → z-ésimo instante.

$\mathbf{s_q(z)}$ → Resposta desejada para um neurônio q no instante z.

X → Vetor de entrada de uma RN.

$\mathbf{e_q(z)}$ → Sinal de erro para um neurônio q no instante z.

$\mathbf{P}$ → Número de pesos de uma RN.

$\mathbf{J}$ → Função de custo, soma dos erros quadráticos.

$\mathbf{c_s}$ → Contém o índice de todos os neurônios da camada de saída.

$\mathbf{Np}$ → Número de padrões de treinamento de uma RN.

$\vec{\mathbf{W}}$ → Vetor de Pesos de uma RN.

α → Taxa de aprendizado do treinamento de uma RN.

$\nabla \mathbf{J}$ → Gradiente da função de custo (soma dos erros quadráticos).

$\mathbf{s_i}$ → i-ésima saída desejada de uma RN.

$\mathbf{y_i^s}$ → i-ésima saída do neurônio da camada de saída de uma RN.

$\mathbf{f^e}$ → Função de ativação dos neurônios da camada de saída de uma RN.

$\mathbf{u_i^s}$ → i-ésimo potencial de ativação do neurônio da camada de saída de uma RN.

$\mathbf{w_{ij}^s}$ → Peso de um neurônio da camada de saída de uma RN.

$\mathbf{w_{jk}^e}$ → Peso de um neurônio da camada intermediária de uma RN.

$\mathbf{x_{ij}^s}$ → Entrada de um neurônio da camada de saída de uma RN.

$\mathbf{x_{jk}^e}$ → Entrada de um neurônio da camada intermediária de uma RN.

$\mathbf{f^e}$ → Função de ativação dos neurônios da camada intermediária de uma RN.

$\mathbf{u_j^e}$ → j-ésimo potencial de ativação do neurônio da camada intermediária de uma RN.

Li → Medida do i-ésimo sensor na direção Leste.

θ → Ângulo de inclinação dos sensores.

h → Hipotenusa que representa o raio de ação dos sensores

NSLO → Vetor formado pela menor distância detectada pelos sensores para as 4 direções.

Pi → Ponto inicial da trajetória

Pf → Ponto objetivo (final) da trajetória.

GT → Gerador de Trajetórias.

P1 → Nova posição do cursor no GT.

difx → Diferença entre as abcissas dos pontos Pi e Pf.

difx → Diferença entre as abcissas dos pontos Pi e Pf.

dify → Diferença entre as ordenadas dos pontos Pi e Pf.

k → k-ésima iteração do programa GT.

S1(k) → Saída 1 da RN do GT na iteração k.

S2(k) → Saída 2 da RN do GT na iteração k.

$Se_d(k)$ → Se na direção d e na iteração k.

$DP_d(k)$ → DP na direção d e na iteração k.

DM → Direção de Movimento

MCP → Memória de Curto Prazo

MLP → Memória de Longo Prazo

TF → Trajetória Final

Capítulo 1

Introdução

A navegação autônoma de veículos há muito desperta o interesse de pesquisadores, principalmente na área de inteligência artificial. Um dos problemas concentra-se na determinação de uma trajetória, de modo que o veículo possa navegar pelo ambiente evitando colidir com obstáculos até alcançar uma ou mais posições pré-estabelecidas no ambiente (Vaz, 1999).

O planejamento de trajetória envolve um grande número de etapas. Planejar significa organizar as informações disponíveis e previstas, de tal forma a conduzir um objeto de um estado a outro, de preferência com segurança e estabilidade. Como o ambiente de navegação geralmente não é estático, o planejamento das ações torna-se uma tarefa dinâmica e o sistema deve ter a capacidade de realizar as adaptações (ajustes) e a tomada de decisões diferentes da planejada, ou seja, um replanejamento. E mais, no planejamento deve-se prever o aprendizado de fatos e ações não previstas, para futura utilização (Mendeleck, 1995).

Neste trabalho, a simulação da interação do robô móvel com o meio é feita através de mecanismos sensoriais, permitindo que o mesmo seja capaz de operar autonomamente em um ambiente desconhecido e não necessariamente estruturado.

As abordagens baseadas em sensores utilizam medições realizadas diretamente no ambiente, para determinar o próximo movimento do veículo, com o objetivo de evitar obstáculos, ao mesmo tempo em que dirigem o veículo em direção ao alvo. A principal vantagem da navegação baseada em sensores é que o veículo pode navegar em ambientes dinâmicos e desconhecidos, pois as únicas informações disponíveis para a movimentação são obtidas dos próprios sensores (Fabro, 1996).

Enquanto se move, o robô deve possuir a capacidade de colecionar informações do ambiente (mapeamento e modelagem do ambiente) e também deve ter alguma noção da sua localização (Salichs, 2000).

Para realizar esta tarefa, foram escolhidas as redes neurais artificiais, pois as mesmas se mostraram eficientes para solucionar o problema básico da navegação, sendo utilizada uma rede do tipo MLP (*Multilayer Perceptron*) com uma camada intermediária e recorrência externa. No seu treinamento foram usados padrões de comportamentos esperados durante a navegação, padrões estes obtidos a partir de uma série de regras pré-definidas por um instrutor (Aprendizado Supervisionado).

A dissertação está organizada de acordo com os seguintes capítulos:

- Após esta introdução, o Capítulo 2 aborda a navegação autônoma de robôs móveis, através de uma revisão de alguns trabalhos nesta área, publicados nas décadas de 60, 70, 80 e 90. Também é descrito o problema de navegação autônoma e uma análise dos seus principais componentes: ambiente, agente e sensores. Além disso, é destacado neste capítulo uma breve revisão do trabalho descrito por A. Mendeleck em 1995 (Mendeleck, 1995), pois um de seus níveis da estrutura neural para o aprendizado, responsável pelo mapeamento da área de trabalho, foi utilizado como inspiração para o sistema proposto nesta dissertação;

- No capítulo 3, é feita uma breve descrição sobre redes neurais artificiais, destacando a arquitetura do tipo Perceptron Multicamadas ou Perceptron de Múltiplas Camadas (MLP) e no campo das aplicações de redes neurais, será enfatizada a tarefa de *classificação de padrões* (Mapeamento de Entrada-Saída).

- Um sistema gerador de trajetórias, utilizando redes neurais artificiais para a navegação de robôs móveis em ambientes desestruturados, é descrito no Capítulo 4. Numa primeira etapa do desenvolvimento deste sistema, foi abordado o uso de uma única rede neural do tipo Perceptron de Múltiplas Camadas (MLP), com uma arquitetura 28-20-2, para a resolução do problema. Esta rede foi implementada em um programa chamado de Gerador de Trajetórias I. Devido à necessidade de testar o sistema proposto para novas condições de trabalho, esta rede neural MLP foi substituída por um sistema híbrido formado por funções lógicas e por 36 redes neurais MLP. Este sistema foi implementado no programa Gerador de Trajetórias II;

- No Capítulo 5, são apresentados e discutidos os resultados gráficos de simulação do sistema proposto em 13 diferentes configurações do ambiente de trabalho.

- Finalmente, o Capítulo 6 apresenta as conclusões, incluindo uma avaliação do método de Geração de Trajetórias desenvolvido, e sugestões para trabalhos futuros.

Capítulo 2

Navegação Autônoma para Robôs Móveis

2. 1. Introdução

A navegação autônoma de veículos há muito desperta o interesse de pesquisadores, principalmente na área de inteligência artificial. Um dos problemas concentra-se na determinação de uma trajetória, de modo que o veículo possa navegar pelo ambiente evitando colidir com obstáculos até alcançar uma ou mais posições pré-estabelecidas no ambiente (Vaz, 1999).

Neste capítulo, será abordada a navegação autônoma de robôs móveis através de uma revisão de alguns trabalhos na área de planejamento de movimentos para robôs móveis. Também será descrito o problema de navegação autônoma e uma análise dos seus principais componentes: ambiente, agente e sensores, além da descrição de algumas abordagens, direcionadas para a tarefa da navegação de robôs móveis.

Mais ainda, é destacada neste capítulo uma breve revisão do trabalho de A. Mendeleck (Mendeleck, 1995), uma vez que um dos níveis da estrutura neural proposta por Mendeleck para o aprendizado, responsável pelo mapeamento da área de trabalho, foi utilizado como inspiração para o sistema proposto nesta dissertação, que será descrito no capítulo 4.

2. 2. Planejamento de Movimentos para Robôs Móveis

Uma das mais avançadas áreas de pesquisa na robótica é o desenvolvimento de robôs autônomos. Tais robôs aceitarão ordens para a execução de tarefas com um elevado grau de dificuldade e as cumprirão sem intervenção humana adicional (Latombe, 1996). Na entrada das ordens, será especificado o que o usuário quer que seja realizado e não uma explicação do que deve ser feito para realizá-las.

Desenvolver as tecnologias necessárias para robôs autônomos é um formidável empreendimento com profundas ramificações entrelaçadas no raciocínio automatizado, percepção e controle, criando assim muitos desfios importantes. Um deles é o *planejamento de movimentos* (Latombe, 1996).

Na maioria dos sistemas robóticos móveis já desenvolvidos, uma característica comum a todos foi o reconhecimento da necessidade de uma estrutura hierárquica de armazenagem e manipulação de conhecimento, da decomposição hierárquica de tarefas (como a locomoção) e de um modelo do ambiente sensorialmente percebido (Kumar, 1989).

Os níveis mais altos definem tarefas ou alvos secundários para os níveis mais baixos, monitorando também os seus estados. Ao mesmo tempo em que se eleva a hierarquia, na escala temporal, houve uma diminuição na freqüência de atualização das informações sensoriais, e na escala dimensional, o ambiente considerado é bem maior, porém menos detalhado. A cada sucessivo nível mais baixo da hierarquia, há uma diminuição na generalidade, na extensão da procura, e no aumento da resolução. Isik e Meystel (Isik, 1988) descreveram que podem ser identificados quatro níveis na hierarquia (Kumar, 1989): *o planejador* (módulo de planejamento de rotas), *o navegador* (módulo de seleção de rotas), *o piloto* (módulo de orientação) *e o controlador*.

Meystel nos seus trabalhos da década de 90 (Meystel, 1995) (Meystel, 1996) descreveu a chamada "Semiótica Multiresolucional", com o objetivo de formalizar os sistemas inteligentes baseados em conceitos semióticos. A semiótica teria por objetivo estudar os diferentes tipos de

signos, bem como o processo por meio do qual estes possuem a qualidade de significar e portanto representar o conhecimento (Suárez, 2000).

2.2.1. O Planejador

O planejamento da rota é feito no nível mais alto. As características gerais da habilidade do robô para se adaptar a terrenos diferentes e sobrepujar vários obstáculos são consideradas no planejamento de uma rota. Neste nível, o elemento básico para a locomoção (perna ou roda) não é considerado. Em um sistema completamente autônomo, este pode ser o único nível que interage com um operador humano. Esta interação pode ser limitada à especificação *off-line* da tarefa. O planejador prescreve um conjunto de alvos secundários para o próximo nível mais baixo. Este trabalha com um modelo do mundo, que tipicamente mede cem vezes o comprimento de seu corpo (Kumar, 1989).

2.2.2. O Navegador

O navegador está principalmente relacionado com a seleção das rotas. Ele usa os dados prévios do terreno e as informações de outros sensores para mapear um "melhor" curso para vários "comprimentos de corpo do veículo", partindo de um comando (alvo secundário) do nível superior. Evitar obstáculo é parte integrante de tal processo. O navegador prescreve alvos secundários para que o nível do piloto satisfaça as exigências do caminho selecionado. Ele mantém um mapa do terreno e um modelo do ambiente limitado por alguns (tipicamente dez) comprimentos de corpo, possuindo um nível mais alto de resolução do que o modelo usado pelo planejador (Kumar, 1989).

A maioria dos trabalhos de navegação autônoma estudam o problema da seleção de rotas. Estes são específicos para veículos que têm rodas ou que andam (com pernas) e estendidos apenas para as características do veículo que devem ser conhecidas (dimensões do corpo, tamanho do pneu ou do pé, comprimento máximo de passo, etc). Porém a maioria dos trabalhos existentes nesta área se referem a robôs que têm rodas (Kumar, 1989).

2.2.3. O Piloto

O piloto planeja uma sequência de ações elementares para o movimento no espaço e gera no tempo uma rota entre as metas prescritas pelo nível de seleção de rota. Podem ser toleradas divergências secundárias do caminho prescrito para evitar pequenos obstáculos. O piloto tem uma memória de curto prazo e sua percepção do mundo é limitada a um ou dois comprimentos de corpo. Este nível pode estar virtualmente ausente na locomoção por rodas, onde o movimento é quase completamente controlado ou especificado pelo nível de seleção de rota (Kumar, 1989).

Porém, em um sistema de locomoção provido de pernas, o piloto é livre para selecionar posições seguras no espaço e no tempo. Assim, a otimização dos parâmetros do andar, seleção ótima de posições seguras, balanceamento dinâmico e o controle de atitude do corpo do veículo, são tarefas que devem ser executadas por este nível. A escolha das pernas como elementos de locomoção introduz novos graus de flexibilidade e um novo nível de complexidade no planejamento de movimentos do corpo. Este nível também foi descrito como o módulo de orientação por alguns pesquisadores (Kumar, 1989).

2.2.4. O Controlador

No nível mais baixo, o controlador, é o único nível que interage diretamente com os atuadores. Representa o nível "espinhal" associado com o controle das juntas individuais em sistemas naturais e envolve o servo controle em tempo real dos *loops* e das informações sensoriais realimentadas no nível dos atuadores. Esta é a única área nesta hierarquia que é razoavelmente bem pesquisada e documentada pois é a base para toda a tecnologia da robótica. Em sistemas de locomoção providos de pernas, isto requer também o planejamento das trajetórias das pernas usando sensores de proximidade e atuadores. O nível correspondente no controle de sistemas que têm rodas é a atuação das rodas. Este nível envolve muito pouco refinamento e também é possível identificar o controlador como um módulo de "execução de plano", e assim, o excluindo da estrutura de tomada de decisão (Kumar, 1989). A figura 2.01 indica todos os níveis hierárquicos e suas respectivas características.

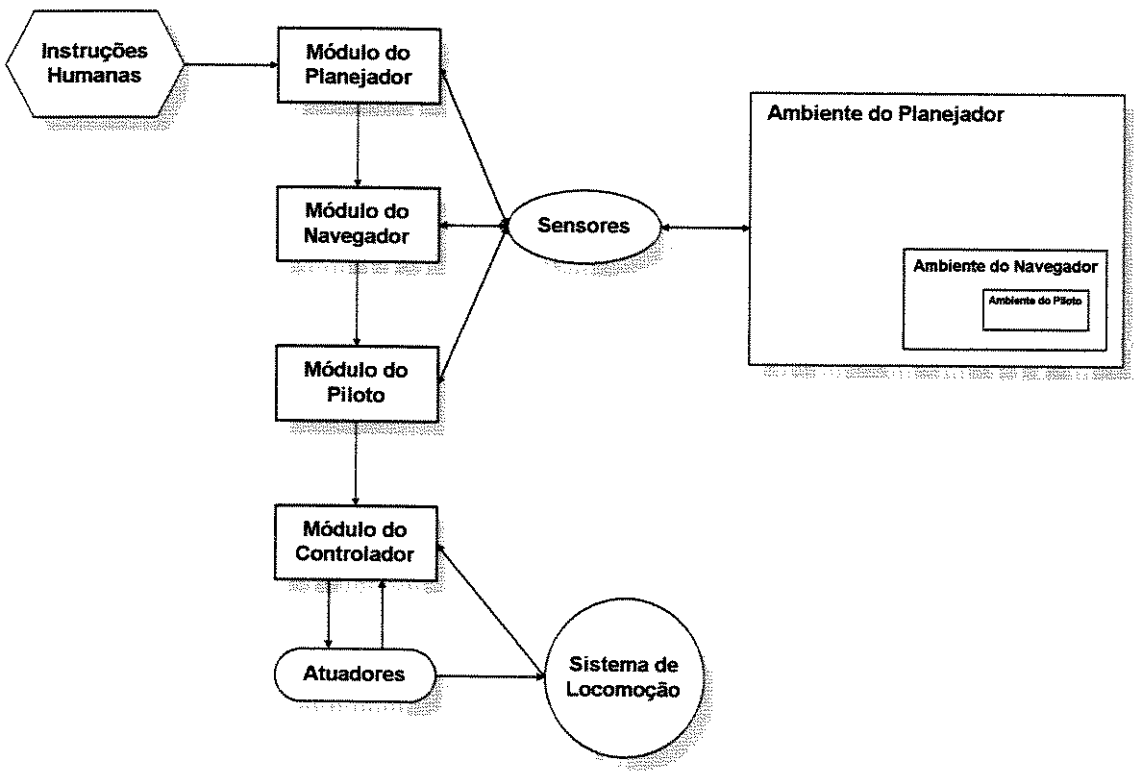


Figura 2.01 - Os níveis hierárquicos descritos por Isik e Meystel.

2.2.5. Uma Revisão dos Trabalhos Sobre o Planejamento de Movimentos

A pesquisa do planejamento de movimentos para robôs pode ser localizada a partir do final da década de 60, durante as fases primordiais do desenvolvimento de robôs controlados por computadores. Entretanto, a maioria dos esforços são mais recentes e têm sido aplicados durante a década de 80. Durante os últimos anos, o entendimento teórico e prático de alguns dos assuntos relacionados com o planejamento de movimentos para robôs aumentou rapidamente, graças ao trabalho combinado de pesquisadores nas seguintes áreas: Inteligência Artificial, Ciência da Computação, Matemática e Engenharia Mecânica (Latombe, 1996).

No seu livro Latombe (Latombe, 1996) faz uma breve discussão sobre algumas publicações chaves, que podem ser consideradas como marcos no desenvolvimento do planejamento de movimentos de robôs. Foram acrescentados, também, os trabalhos citados por Kumar e Waldron

(Kumar, 1989). As citações descritas a seguir são colocadas na ordem cronológica, sendo que esta seção também pode ser considerada como uma pequena revisão histórica do desenvolvimento do planejamento de movimento nas décadas de 70 e 80:

Nilsson (Nilsson, 1969) descreveu um sistema móvel integrado com rodas e com capacidade de planejamento, construído pelo Stanford Research Institute, chamado de SHAKEY. Embora boa parte do planejamento feito por SHAKEY usava um modelo do ambiente de trabalho e as ações do robô eram expressas na linguagem de cálculo de predicados (*Strips planner*), Nilsson introduziu o método do grafo V (*V-graph*) ou grafo de visibilidade para planejar os movimentos de SHAKEY.

Ambler e Popplestone (Ambler, 1975) descreveram um método para deduzir a localização quantitativa de objetos em termos das matrizes homogêneas de coordenadas, introduzindo relações simbólicas do espaço entre as características dos objetos. Este método foi estendido depois para um conjunto maior de relações e implementado como parte de uma programação robótica através de uma linguagem de alto nível (*Rapt*) por Popplestone, Ambler e Bellos (Popplestone, 1980).

Taylor (Taylor, 1976) investigou o problema do planejamento de movimentos na presença de incertezas. Ele propôs um método, conhecido como "*skeleton refinement*" (refinamento esquelético), baseado em técnicas numéricas para processamento de incertezas. Uma aproximação semelhante também foi proposta por Lozano-Pérez (Lozano-Pérez, 1976). Uma melhoria do método para propagação, usando técnicas simbólicas em lugar de numéricas, foi descrita depois por Brooks (Brooks, 1982).

O conceito, descrito por Udupa (Udupa, 1977), de encolher um único ponto de referência para o robô, enquanto as regiões de obstáculos são ampliadas, tem provado ser bem útil. Não era ainda usado o termo "*configuration space*" (espaço configurado). Lozano-Pérez e Wesley (Lozano-Pérez, 1979) exploraram esta idéia de uma forma mais sistemática, e propuseram o primeiro algoritmo para planejamento de rotas para robôs e obstáculos poligonais e poliédricos

sem giro. O trabalho deles normalmente é considerado como a primeira contribuição para a "exatidão" no planejamento de movimentos.

Thompson (Thompson, 1977) descreve um módulo de planejamento de rotas para um *rover* (Remote Operate Vehicle Robotic) JPL. Nesta abordagem, foi evitada a construção de um grafo V para o espaço inteiro, mas ela constrói (expande) o grafo como e quando precisar.

Reif (Reif, 1979) apresentou a primeira investigação teórica da complexidade computacional inerente ao problema de planejamento de rotas, mostrando que o planejamento de uma rota livre, em um espaço de configuração de dimensões arbitrariamente grandes, entre obstáculos fixos é muito difícil (*PSPACE-hard)*. Este trabalho contribuiu atraindo o interesse de cientistas teóricos da computação para o planejamento de movimentos. Foi seguido por uma sucessão contínua de artigos nos limites mais baixos da complexidade para uma variedade de problemas de planejamento de movimentos. Reif denominou esta área como Robótica Computacional em 1987 (Reif, 1987).

Lozano-Pérez (Lozano-Pérez, 1981) obteve emprestado a noção de "espaço configurado" da Mecânica, e a popularizou no planejamento de movimento. Ele também estendeu as suas idéias, descritas juntamente com Wesley (Lozano-Pérez, 1979), e introduziu o princípio da decomposição em células aproximadas (Lozano-Pérez, 1981) (Lozano-Pérez, 1983). Isto envolve uma abordagem dos obstáculos através de poliedros. No espaço bidimensional, a rota mais curta livre de colisões é composta por linhas que unem diretamente a origem ao destino, através do conjunto de vértices do obstáculo poligonal. Um grafo V (*V-graph*) ou grafo de visibilidade no qual cada ligação é representada por uma linha direta entre dois pontos, os quais podem "ver" um ao outro, pode ser construído e uma rotina de busca é usada para obter o caminho ótimo. Esta afirmação é decorrente dos trabalhos de Davis e Camacho (Davis, 1984), e de Nilsson (Nilsson, 1980).

Métodos de representação do ambiente do robô receberam uma atenção considerável, como a procura de uma rota satisfatória que pode ser facilmente executada através de algoritmos padrões, como o algoritmo A*, foi descrito por Nilsson (Nilsson, 1980). Brooks descreveu nos

seus trabalhos (Brooks, 1983a) (Brooks, 1983b), a modelagem do espaço livre (espaço sem obstáculos) como uma união de cones generalizados. Isto conduz eventualmente para uma utilização mais eficiente do espaço. Ele usou cones generalizados para construir um grafo *connectivity,* que era então a entrada da rotina de procura de caminhos. HILARE foi um robô com três rodas construído na França, descrito por Giralt et al. (Giralt, 1979), no qual o modelo do mundo também foi representado por um grafo *connectivity*. Tipicamente, os nós de um grafo *connectivity* seriam lugares ou espaços e ligações. Os espaços poderiam ser decompostos mais adiante em células poligonais que representariam o espaço livre.

Chatila (Chatila, 1982) investigou o planejamento de movimentos com conhecimento incompleto de um robô móvel representado como um ponto em um ambiente de trabalho bidimensional. Ele foi o primeiro a basear o seu planejador em uma decomposição exata do subconjunto vazio do ambiente de trabalho em células convexas (uma idéia que foi estendida depois para espaço livre na conformação do espaço). O planejador operava em tempo real, e a decomposição foi atualizada periodicamente com o objetivo de levar em conta novas informações providas pelos sensores.

Ó'Dúnlaing e Yap (Ó'Dúnlaing, 1982) introduziram uma retração como uma nova abordagem teórica para o planejamento de rotas. Independentemente, e quase simultaneamente, Brooks (Brooks, 1983a) apresentou uma abordagem mais empírica: o método de planejamento "*retraction-like*", conhecido como *freeway method.*

Nos anos de 1983 e 1984, Schwartz, Sharir, e Ariel-Sheffi publicaram uma série de cinco trabalhos chamada a série de *"The Piano Movers' Problem"* (Schwartz, 1983a) (Schwartz, 1983b) (Schwartz, 1983c) (Sharir, 1983) (Schwartz, 1983d), sendo os dois primeiros os mais importantes (Canny, 1989). O primeiro artigo da série descreve o primeiro método exato para planejamento de rotas livres de um objeto poligonal, permitido para ambos translação e giro em um ambiente poligonal e bidimensional. O segundo artigo estabeleceu o primeiro salto no tempo para os limites superiores da complexidade do planejamento de rotas em um espaço livre semi-algébrico com uma dimensão fixa qualquer.

Lozano-Pérez, Mason e Taylor (Lozano-Pérez, 1984) descreveram a abordagem de reencadeamento de pré-imagens para o planejamento de movimentos com incertezas no controle e sentido. Pela primeira vez, esta abordagem considerou movimentos complacentes no planejamento. No artigo original, a abordagem era essencialmente uma estrutura teórica, e não foi implementada. A partir deste fato, a teoria e a implementação de reencadeamento de pré-imagens foram investigadas posteriormente.

Dufay e Latombe (Dufay, 1984) descreveram uma abordagem implementada para planejamento de movimentos com incertezas, baseada no aprendizado indutivo. A abordagem consiste em executar a mesma tarefa várias vezes e combinar os rastros de execução em uma estratégia mais geral. Em seguida, outros pesquisadores (Latombe, 1996) investigaram diferentes técnicas de aprendizado para o planejamento de movimentos e aquisição de habilidades do movimento.

Koch et al. (Koch, 1985), semelhantemente ao trabalho de Thompson (Thompson, 1977), propuseram o conceito de setores (setores de um círculo que contém as posições do veículo atual e alvo, e com o mesmo ao centro) os quais produzem um subconjunto de grafos V. Assim, eles superam o problema da explosão combinatorial.

Khatib (Khatib, 1986) abriu caminho para a abordagem de potencial de campo (*potential field*). Embora ele o tenha implementado como um módulo para evitar colisões em tempo real no controlador de um robô, a abordagem é estendida para o planejamento de movimentos, o que foi desde então realizado. Koditschek (Koditschek, 1987) introduziu a noção de uma "função de navegação", uma função potencial com um mínimo local livre. Barraquand e Latombe (Barraquand, 1989) combinaram um método de potencial de campo com técnicas aleatórias para escapar de mínimos locais e implementaram um planejador para gerar rotas livres para robôs com muitos graus de liberdade.

Laumond (Laumond, 1986) (Laumond, 1987) considerou o problema do planejamento de rotas livres para robôs móveis não holonômicos com rodas. O planejamento de movimentos na presença de constantes cinemáticas (sempre evitando obstáculos), foi denominado como

planejamento de movimentos não holonômicos, quando os graus de liberdade de um sistema robótico não são independentes (Laumond, 1998). Neste caso, qualquer rota no espaço configurado livre não necessariamente pode ser uma rota possível. Ele produziu um resultado interessante, que uma rota livre para um robô *free-flaying* em um ambiente de trabalho bidimensional sempre pode ser transformado em uma rota livre para um robô não holonômico com rodas, tendo a mesma geometria e se movendo no mesmo ambiente de trabalho, pela introdução de simples manobras de auxílio. Li e Canny (Li, 1989) foram os primeiros que aplicaram ferramentas desenvolvidas da teoria de controle para sistemas não lineares aplicadas para robôs não holonômicos. Estas ferramentas tornaram possível generalizar os resultados de Laumond.

Continuando esta revisão, os trabalhos da década de 90 estão citados no item 2.4. (Abordagens para Navegação Autônoma), onde foram listadas algumas abordagens novas, principalmente no campo da inteligência artificial, aplicadas a solução do problema.

2. 3. O Problema de Navegação Autônoma

A navegação é considerada como uma capacidade básica da robótica autônoma (Salichs, 2000). Na expressão mais ambiciosa desta capacidade, um robô móvel deve poder passar no seguinte teste:

"O robô é colocado em um ambiente que é desconhecido, de grandes dimensões, complexo e dinâmico. Depois de um certo tempo explorando o ambiente, o robô deve poder ir para qualquer lugar selecionado e deve também tentar minimizar uma função de custo qualquer, como por exemplo: tempo, energia, etc".

Este teste, chamado de *Maximum Navigation Test* (MNT) (Salichs, 2000), para ser realizado com sucesso pelo robô vai necessitar da resolução de vários problemas. Primeiro, o robô deve poder mover-se rapidamente, seguramente controlado, evitando obstáculos estáticos e móveis (problema de controle do movimento). Enquanto se move, o robô deve poder guardar

informações do ambiente (problema de modelagem do ambiente) e estar sempre atento em relação à sua localização (problema de localização). Além destes problemas, para que o robô se movimente otimamente em uma grande área, necessita-se algum planejamento (problema de planejamento). Estes problemas não são independentes, mas podem ser considerados como subproblemas do problema geral (MNT).

O problema de navegação autônoma pode ser simplificado e definido resumidamente da seguinte forma (Oliveira, 1995):

"Sejam um ambiente de operação A contendo um conjunto de objetivos T, um conjunto não vazio de obstáculos O e um veículo V com um conjunto de sensores S. Considerar-se-á este veículo V como sendo um veículo autoguiado ou seja, capaz de navegação autônoma, se, utilizando apenas as informações sobre o ambiente A fornecidas pelos sensores S, o veículo V for capaz de, através de um algoritmo de controle conveniente C, que não contenha componente humano H algum, atingir os objetivos T sem se chocar com um ou mais obstáculos O".

Um ser humano facilmente resolveria esse problema, porém deve-se considerar os enormes recursos dos quais um humano dispõe, como por exemplo: memória, capacidade de abstração, aprendizagem não-supervisionada e a riqueza de informações colhidas através de seus sensores (Oliveira, 1995).

A resolução de qualquer problema passa obrigatoriamente por três fases: a caracterização do mesmo, a determinação dos requisitos mínimos para a solução mais simples e a determinação da solução em si.

O problema de navegação autônoma engloba três componentes: o ambiente, o agente (veículo) e os sensores utilizados. O sucesso da sua solução, ou seja, o algoritmo desenvolvido, dependerá da correta caracterização destes componentes.

2. 3.1. O Ambiente

Existem dois tipos de ambientes de operação: o mundo real e simulações do mundo real. A operação no mundo real exige a implementação física do agente e adiciona critérios como tempo de resposta e tolerância a ruído. Um erro de hardware é muito mais difícil de localizar e corrigir que um erro de software. Por outro lado, o uso do mundo real evita o trabalho da construção de uma simulação que possa emular o mundo real de maneira satisfatória e possíveis falhas de implementação originadas da transição da simulação para a aplicação prática.

Já o uso de uma simulação permite a rápida modificação e re-adequação do algoritmo de navegação face a imprevistos ou erros de caracterização. Suas desvantagens residem na necessidade de produzir níveis de detalhamento de uma simulação, cujas características se aproximam tal ao mundo real que permitam o uso do algoritmo de controle sem modificações significativas quando utilizados na práticas. A escolha do tipo de ambiente é restrita pelos recursos disponíveis, sejam computacionais, financeiros, ou de disponibilidade de hardware ou de tempo (Oliveira, 1995).

2. 3.2. O Agente

Embora não haja ainda um consenso sobre uma definição formal do que seja o agente tal que englobe todo o espectro possível, algumas características esperadas foram estabelecidas. A analogia feita com agentes no mundo real nos leva a conceituar um agente como uma entidade ativa e que possui conhecimento específico sobre um determinado domínio. De posse de bases de conhecimento e de mecanismos de raciocínio, os agentes devem ser capazes de reconhecer situações em que devam se ativar, sem que o usuário perceba, ou seja, de forma transparente ao usuário (GSI, 2001).

O agente (veículo) citado na descrição do problema da navegação autônoma é um robô móvel. Definido por Muir em 1988 (Victorino, 1998) como: "um robô capaz de se locomover sobre uma superfície somente através da atuação de rodas montadas no robô e em contato com a superfície". Mais generalizador, Lum (Lum, 1999) os definem como : "robôs que possuem como métodos de locomoção: rodas, trilhos ou pernas". O acionamento de tais componentes é

decorrente da intervenção de atuadores, mecanismos alimentados por uma fonte de energia e controlados a partir de alguma estratégia de navegação. São os atuadores que definem os parâmetros que representam o movimento: direção e velocidade. Há muitas aplicações para os robôs móveis, incluindo-se o uso em áreas perigosas para os seres humanos (Lum, 1999).

Entre os robôs com pernas, o robô bípede possui o potencial de mobilidade semelhante ao do ser humano, que pode mover-se em áreas contendo obstáculos tais como escadas, degraus e superfícies irregulares. Os problemas dos robôs bípedes são a instabilidade inerente à locomoção bípede e o respectivo controle. Para realizar uma locomoção bípede estável, é essencial obter um modelo dinâmico adequado e então simular e analisar o desempenho do sistema (Lum, 1999).

Para o caso dos robôs móveis com rodas, geralmente eles são estruturados das seguintes formas:

- *Com duas rodas motrizes e uma ou duas rodas direcionais*: para esta configuração basta mudar o ângulo da(s) roda(s) direcional(is) para mudar o sentido de movimento do agente. O movimento de ré pode ser obtido invertendo-se o sentido de movimento das rodas motrizes. A velocidade do veículo será proporcional à velocidade angular das rodas (Dozier, 1998) (Hague, 1996) (Ohya, 1998).

- *Com duas rodas motrizes independentes*: a direção de movimento poderá ser modificada fazendo com que uma roda gire mais rapidamente que a outra. Assim, para virar para a esquerda, a roda direita girará mais rapidamente que a esquerda (Feder, 1999) (Fierro, 1998) (Jetto, 1999) (Prassler, 1999) (Yang J., 1999).

Enquanto se move, o robô deve possuir a capacidade de colecionar informações do ambiente (mapeamento e modelagem do ambiente) e também deve ter alguma noção da sua localização (Salichs, 2000).

Uma vez escolhido o ambiente, passa-se a modelar a interação do ambiente e do veículo. Este processo de modelagem incorre na escolha dos sensores adequados e nas ações que o veículo possa tomar (Oliveira, 1995).

2. 3.3. Os Sensores

Uma das principais características dos robôs móveis considerados inteligentes é o uso de sensores externos que os tornam autônomos e capazes de operar em ambientes desestruturados. Tais sensores permitem ao robô autônomo extrair informações do ambiente levando-o a reagir às mudanças do mesmo de forma inteligente (Rezende, 1992).

A entrada de informações em um sistema inteligente ocorre sempre por meio de sensores. A informação dos sensores é sempre enviada ao módulo de processamento sensorial do sistema inteligente, que transformará os sinais oriundos dos sensores em informação útil ao sistema (Suárez, 2000).

Sistemas com pouca ou nenhuma capacidade sensorial externa, geralmente são limitados a operações de sequência fixa em ambientes altamente estruturados e não podem fornecer nenhum grau de autonomia substancial ou adaptabilidade (Rezende, 1992). Assim sendo, pode-se perceber que um robô sem sensores não é capaz de lidar devidamente com mudanças em seu ambiente.

Existem em robótica diversos modos de classificar os sensores (Ferreira, 1991), como por exemplo em sensores de percepção interna e de percepção externa ao robô. Entretanto, isto implicaria em colocar numa mesma classificação simples detectores de proximidade e complexos sistemas de visão.

Os sensores de percepção externa encontrados na robótica podem ser classificados da seguinte forma (GSI, 2001): sensores de toque, sensores de distâncias, sensores de proximidade e sensores de visão.

2. 3.3.1 Sensores de Toque

Sensores de toque (tato) são dispositivos que indicam o contato entre eles próprios e algum outro objeto, normalmente usados em robôs industriais ou manipuladores. A informação de toque

pode ser usada, por exemplo, para a localização e reconhecimento de objetos, indicando ainda a magnitude da força de contato entre os dois. Um dos objetivos de se usar sensores deste tipo é identificar e controlar diretamente a interação entre o robô e o ambiente onde ele se encontra.

2. 3.3.2 Sensores de Distância

São dispositivos que medem a distância entre um ponto de referência (normalmente outro sensor) e os objetos no campo de atuação do mesmo. Tais sensores são usados na navegação de robôs e no desvio de obstáculos, onde a sua utilização consiste em estimar as distâncias aos objetos mais próximos, em aplicações onde a localização desses objetos é necessária. Existem várias técnicas para efetuar os cálculos para determinar essa distância: o método da triangulação, a abordagem da luz estruturada e a técnica dos ultra-sons. Os sensores de distância mais utilizados nos robôs móveis autônomos inteligentes (Rezende, 1992) são os sensores de ultra-som.

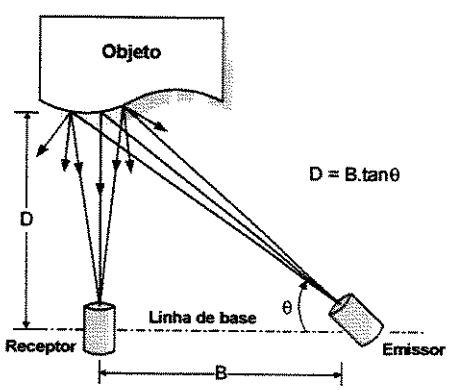


Figura 2.02 - Um exemplo de um sensor de distância.

2. 3.3.3 Sensores de Proximidade

Os sensores de distância discutidos anteriormente estimam a distância entre o sensor e um determinado objeto. Os sensores de proximidade, por outro lado, têm geralmente uma saída binária que indica a presença de um objeto a uma distância pré-definida. Os sensores de proximidades mais utilizados nos robôs móveis autônomos inteligentes (Rezende, 1992) são sensores de infravermelho. Em algumas aplicações é suficiente determinar a presença (ou

ausência) de um objeto a uma certa posição em lugar de medir a distância ao objeto (Bradshaw, 1990). Embora este tipo de sensor não devolva a distância entre ele e um objeto qualquer, eles podem identificar se algo está presente ou não, dentro do seu cone de detecção (Jones, 1993).

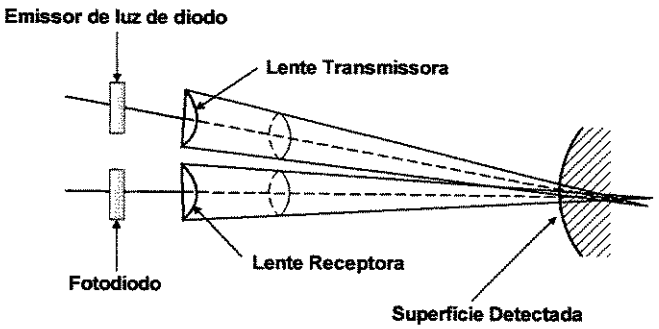


Figura 2.03 - Um exemplo de um sensor de proximidade.

2. 3.3.4 Sensores de Visão

O uso destes sensores tem despertado o interesse de muitos pesquisadores e é atualmente uma das áreas de pesquisa mais ativas (Han, 1994) (Hague, 1996) (Dozier, 1998) (Ohya, 1998). O avanço das pesquisas em sistemas de visão tornou possível o surgimento de robôs que interagem com o ambiente, recebendo informações de diversas formas para processá-las e, em função delas, executar determinadas tarefas. A recepção e obtenção de informações do meio é feita pelos "olhos" do robô, ou seja, por uma câmara de televisão e/ou algum outro sensor juntamente com o equipamento de digitalização e hardware/software necessários para o interfaceamento entre os mesmos.

Pode-se concluir que uma exigência fundamental para a solução do problema de navegação (Bradshaw, 1990) é a existência de um sistema sensorial, permitindo a aquisição dos dados necessários para uma descrição do ambiente do robô móvel em detalhes suficientes para o controle do seu movimento.

2. 4. Abordagens para Navegação Autônoma

Os métodos de navegação autônoma podem ser organizados por dois critérios (Oliveira, 1995): filosofia empregada e características operacionais.

A utilização de características operacionais para a classificação dos navegadores autônomos permite uma vasta quantidade de categorias. Foram estabelecidos como parâmetros relevantes (Oliveira, 1995): uso de memória; capacidade de aprendizagem e adaptabilidade; e o uso de informações locais ou globais.

2. 4.1. Memória

A memória está diretamente atrelada à aprendizagem, sendo uma condição necessária mas não suficiente. A memória consiste de um mecanismo de recuperação de eventos passados. Existem diversas formas de memória: genética, ontológica, e a cultural/social (Oliveira, 1995).

- *Memória genética*: possui significado restrito, existindo apenas quando existe um processo de aprendizagem ou otimização evolucionária. Ela consiste do conjunto de pares (genoma, desempenho) da população. Uma vez interrompido o processo evolucionário e selecionado o indivíduo de melhor desempenho, pode-se dizer que todo o conhecimento gerado pela evolução foi concentrado no genoma deste indivíduo.

- *Memória Ontológica:* é a memória dita "convencional". Aparece sob diversas formas: como locais pré-determinados para a armazenagem de dados, como parâmetros modificáveis pelo próprio agente, ou quando o sistema é dinâmico e causal. Neste último caso, a memória consiste de efeitos produzidos pelo comportamento passado do sistema. Como exemplos, existem os pesos das sinapses das redes neurais ou o banco de conhecimento de um sistema especialista.

- *Memória cultural/social:* pode ocorrer apenas em ambientes com múltiplos agentes com capacidade de comunicação entre si. O conhecimento está armazenado de modo distribuído, de modo parcial ou total na memória ontológica de cada agente.

Em todos os casos, a memória liberta o agente da localidade temporal, isto é, o agente não está restrito aos estímulos existentes naquele instante.

Como apenas alguns métodos de navegação autônoma não fazem uso da memória em suas variadas formas, os métodos atuais de navegação serão apresentados e classificados em função de capacidade de aprendizagem e do escopo dos seus sensores (Oliveira, 1995).

2. 4.2. Aprendizagem

Uma abordagem alternativa, a "abordagem de aprendizado", é especificada por Pattie Maes em 1994 (GSI, 2001). A hipótese por ela testada é que, sob certas condições, um agente pode "autoprogramar-se", isto é, o próprio agente pode adquirir o conhecimento de que necessita. Ao agente é fornecido um conhecimento mínimo, e ele aprende o comportamento que leva ao atendimento de objetivos predefinidos.

A aprendizagem não é condição necessária para a operação autônoma. Os agentes construídos por Brooks (Brooks, 1991), Gat (Gat, 1994) e Li (Li, 1994) não possuem mecanismos de aprendizagem e navegam em ambientes diversos (Oliveira, 1995). Entretanto, pode-se considerar que estes modelos são menos autônomos do que os dotados de aprendizagem, pois necessitam de mais informações colhidas a priori pelos projetistas.

Um método sem aprendizagem poderá ser estendido, através do acoplamento de um algoritmo de aprendizagem conveniente. De fato, esta agregação geralmente ocorre como uma segunda etapa no desenvolvimento dos algoritmos de controle autônomo (Oliveira, 1995).

2. 4.3. O Uso de Informações Locais

Um agente autônomo utiliza informações locais quando necessita apenas de dados (colhidos pelos seus sensores) sobre o ambiente ao seu redor (Oliveira, 1995).

As abordagens baseadas em sensores utilizam medições, realizadas diretamente no ambiente, para determinar o próximo movimento do veículo com o objetivo de evitar obstáculos, ao mesmo tempo que dirigem o veículo em direção ao alvo. A principal vantagem da navegação baseada em sensores é que o veículo pode navegar em ambientes dinâmicos e desconhecidos, pois as informações mais importantes para a movimentação são obtidas dos próprios sensores (Fabro, 1996).

Os agentes autônomos que utilizam apenas informações locais para desempenhar tarefas são mais flexíveis. Primeiro porque não requerem a caracterização de todo o ambiente de operação, reduzindo a quantidade de informação a ser processada, e segundo porque podem ser utilizados em um maior número de ambientes, pois ambientes macroscopicamente distintos podem ser localmente semelhantes (Oliveira, 1995).

No âmbito da filosofia de projeto, identificam-se três tipos de sistemas de controle autônomo (Oliveira, 1995): sistemas planejadores, sistemas reativos e sistemas comportamentais.

2. 4.4. Navegação por Planejamento

A navegação por planejamento baseia suas ações na correlação entre um modelo interno do ambiente de operação e o ambiente real. Todas as características consideradas relevantes são predeterminadas durante o projeto e incluídas na representação interna. Os sensores são responsáveis pela contínua atualização deste modelo interno de mundo. O agente utiliza este modelo interno para planejar suas ações (Oliveira, 1995).

Destacado como parte considerável do processamento em um sistema baseado em planejamento (Oliveira, 1995), a constante atualização do modelo interno poderá inviabilizar a reação em tempo hábil a eventos externos em um ambiente considerado complexo. Existe ainda a

questão da exatidão do modelo interno, pois se a representação interna não refletir os aspectos realmente relevantes do ambiente, as decisões nela baseadas estarão comprometidas.

O exemplo típico de sistemas baseados em planejamento são os sistemas especialistas (Oliveira, 1995). Sistemas Especialistas podem ser definidos como aqueles projetados e desenvolvidos para atender a uma aplicação determinada e limitada do conhecimento humano. É capaz de emitir uma decisão apoiado em conhecimento justificado, a partir de uma base de informações, da mesma forma que um especialista de determinada área do conhecimento humano (GSI, 2001). Esta base de dados está geralmente formulada em termos de regras lógicas e baseadas em um conjunto de símbolos. A correta operação do sistema não será garantida caso o conjunto de símbolos seja insuficiente. Isso pode decorrer de uma imperfeita caracterização do ambiente de operação do sistema.

A agregação de um mecanismo de aprendizagem será de alguma valia somente se for capaz de gerar novos símbolos. Os sistemas baseados em planejamento também dependem criticamente da correta tradução dos estímulos externos captados pelos sensores nos símbolos predeterminados. Isso somente ocorrerá se todos os estímulos possíveis forem antecipados na formulação do modelo interno. Tal fato é restritivo quando se trata de ambientes desestruturados. Este é o caso do problema de navegação (Oliveira, 1995).

2. 4.5. Navegação Reativa

A navegação reativa associa um conjunto de ações a cada conjunto possível de estímulos. Estas relações são compactamente representadas por pares do tipo: estímulos - ações. Esta forma de processamento é localizada, pois considera apenas os estímulos ambientais atuais resultantes do ambiente circundante, colhidos por sensores. A agregação de estados internos (memória) à arquitetura reativa, sob a forma de estímulos internos, permite uma globalização temporal da estratégia de navegação. Os estímulos internos forneceriam um modo de comparação do estado atual com experiências passadas. Exemplos típicos desta arquitetura são os sistemas nebulosos e as redes neurais. Estas duas arquiteturas serão descritas em itens posteriores.

2. 4.6. Navegação Comportamental

Os sistemas baseados em comportamentos podem ser considerados como uma evolução dos sistemas reativos. Comportamentos são modos distintos de operação caracterizados por conjuntos distintos de relações entre estímulos e ações. Um módulo pode compartilhar sensores com outros módulos. Os módulos interagem, modificando os sinais de entrada fornecidos a outros módulos, como está representado na figura 2.04. A composição das ações de cada módulo será a ação tomada pelo sistema.

O grau de interação dos módulos determinará se o sistema é realmente baseado em comportamentos. Se os módulos comportamentais influenciam a ação do sistema apenas através de uma agregação de suas saídas, o sistema é dito reativo, pois a distinção entre módulos é apenas formal.

A arquitetura *subsumption* descrita por Brooks (Brooks, 1986) apresenta módulos comportamentais que interagem (Oliveira, 1995). Sobre este trabalho, Chatila (Chatila, 1989) descreve a idéia básica proposta por Brooks como: "decompor o sistema robótico não em relação ao seu operacional interno (baseado na decomposição funcional, por exemplo), mas de preferência em relação ao seu comportamento externo. A base desta arquitetura é que um dado módulo pode suprimir ou inibir as entradas (saídas) de outro módulo por um dado período de tempo, realmente trocando suas entradas (saídas) usuais. Com isso, a interação ocorre apenas como uma inibição (*subsumption*) das entradas deste por aquele.

A figura 2.04 mostra o comportamento na navegação autônoma decomposto, por exemplo, nos seguintes comportamentos básicos ou módulos (Oliveira, 1995): seguir paredes; evitar obstáculos; seguir alvo (seja este estático ou dinâmico).

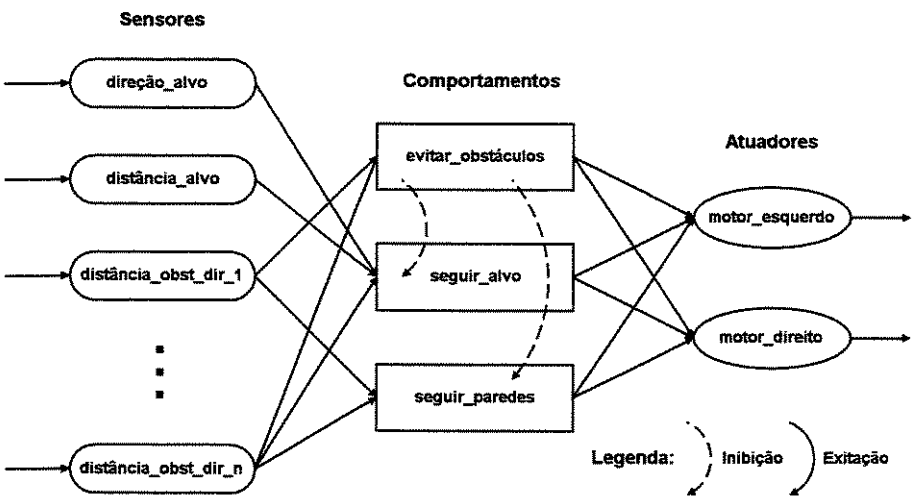


Figura 2.04 - Exemplo de um sistema de navegação comportamental.

A seguir, serão apresentadas outras abordagens para a resolução do problema da navegação autônoma. Elas são fundamentadas principalmente na inteligência artificial, no aprendizado por reforço, na teoria do potencial de campos, na teoria dos filtros de Kalman, e na pesquisa em grafos/árvores.

2. 4.7. Navegadores Utilizando Redes Neurais

Os navegadores utilizando redes neurais atuam como uma função não-linear multidimensional, a qual mapeia estímulos a saídas (Oliveira, 1995). Este mapeamento é implementado por uma rede neural multicamadas.

Estes navegadores são providos de memória e de aprendizagem. A memória está distribuída nos pesos sinápticos e a memorização ocorre no processo de aprendizagem (treinamento). Uma rede neural aprende a associar estímulos com saídas através da modificação dos seus pesos sinápticos. Em relação aos paradigmas de aprendizado, temos duas categorias de navegadores utilizando redes neurais:

- *Aprendizado supervisionado*: os seus esquemas de aprendizagem dependem do fornecimento de dados sobre o comportamento desejado do navegador, em termos dos possíveis estímulos experimentados por este. A complexidade do problema de navegação autônoma, representado pela vasta gama de situações possíveis, torna uma tarefa muito difícil a aplicação de mecanismos de aprendizado supervisionado, por ser difícil determinar previamente todas as situações que podem ocorrer.

- *Aprendizado não-supervisionado*: utiliza critérios mais genéricos para a otimização dos pesos sinápticos de uma rede neural. Como exemplo, as redes de Kohonen utilizam como parâmetros de aprendizagem a proximidade espacial de neurônios e seu grau de ativação face a estímulos. São fortalecidas as conexões entre sensores e neurônios com intensa ativação e enfraquecidas as conexões dos demais neurônios.

Tzafestas (Tzafestas, 1995) descreve no seu trabalho um breve estado da arte do uso de redes neurais no planejamento de caminhos para robôs móveis, destacando os trabalhos de Eckmiller em 1987, que desenvolveu uma rede neural chamada de "*neural triangular lattice*" (treliça triangular neural) para guardar e recuperar trajetórias para robôs, e de Seshadri em 1988, com uma rede neural tentado minimizar a dimensão do caminho para uma determinada posição alvo.

Oliveira em 1995 (Oliveira, 1995) destaca a proposta do navegador neural desenvolvido por Verschure, Kröse e Pfeifer (Verschure, 1992) por causa da sua simplicidade.

Verschure et al. utilizaram uma rede neural de estrutura simples e aprendizagem por condicionamento. A sua estrutura neural engloba três tipos de sensores: um medidor de distâncias em 37 direções distintas, um indicador de colisões e outro da direção na qual está o alvo. Sendo cada sensor associado a um conjunto de neurônios. Estes neurônios estimulam os neurônios dos atuadores, que interagem e são responsáveis por 5 tipos de movimentos predeterminados: desvio grande à esquerda, desvio pequeno à esquerda, em frente, desvio pequeno à direita e desvio grande à direita. Os neurônios de colisão inibem os neurônios associados à direção do alvo e os neurônios do medidor de distâncias estimulam tanto os neurônios de colisão quanto os neurônios de direção do alvo. Duas ações de reflexo são embutidas: virar para a direita quando houver

colisão à esquerda e virar para a esquerda quando houver colisão à direita do veículo. O aprendizado desta rede é baseado na associação entre padrões sensoriais e colisões utilizando uma forma da regra de Hebb.

Verschure et al. Demostraram, através de várias simulações, a capacidade do veículo de navegar em ambientes com muitos obstáculos, pela associação da proximidade de obstáculos com ações de desvio destes obstáculos antes da colisão.

A seguir, serão descritos outros trabalhos de navegadores utizando redes neurais apresentados nos últimos anos, mantendo-se uma ordem cronológica dos mesmos.

Thrun (Thrun, 1995) descreve uma abordagem para o aprendizado de um navegador robótico. Através do mesmo, um robô móvel equipado com sensores de visão, de ultra-som e laser, é capaz de, em menos de 10 minutos de operação, navegar até chegar em um objetivo fixado. Isto foi possível graças a um algoritmo de aprendizado chamado de EBNN (*Explanation-based Neural Network*),definido como um mecanismo de aprendizado híbrido que integra aprendizado indutivo e analítico. O indutivo é representado pelo aprendizado da rede neural (via Backpropagation) e o analítico pelo aprendizado baseado em explicações (EBL).

No Brasil, Uebel et al. (Uebel, 1995) apresentaram uma comparação entre o uso de redes neurais artificiais booleanas e de sistemas nebulosos (*fuzzy*), para controle da navegação de um robô autônomo em um labirinto, sendo os seus possíveis comportamentos: "escapar de obstáculos" e "seguir fonte de luz". As redes neurais, com um número menor de regras, apresentaram um comportamento semelhante aos sistemas nebulosos, devido à sua capacidade de generalização. Tal fato reforça o uso de redes neurais em situações mais complexas, em que é requerido um grande número de regras para os sistemas nebulosos operarem.

Ortega e Camacho (Ortega, 1996) apresentaram uma proposta para a implementação de um controlador baseado em um modelo preditivo (MBPC). Ele foi usado para a navegação de um robô móvel no seu ambiente de trabalho, e na presença de obstáculos estáticos não conhecidos. O método utiliza uma modelagem não linear da dinâmica do robô móvel, permitindo então uma predição acurada das suas futuras trajetórias. Um sistema sensorial ultra-sônico foi usado para a

detecção de obstáculos. Uma rede neural artificial do tipo perceptron multicamadas (MLP) foi usada no desenvolvimento do MBPC, permitindo uma implementação em tempo real e também eliminando a necessidade de um nível elevado de processamento dos dados dos sensores. A rede MLP foi treinada de maneira supervisionada para reproduzir os comportamentos do MBPC. Resultados experimentais satisfatórios foram obtidos quando foi aplicado o controle via rede neural a um robô móvel *TRC Labmate.*

Chang e Song (Chang, 1997) descreveram que o problema da navegação de um robô móvel entre obstáculos móveis é usualmente resolvido pela condição do conhecimento da velocidade dos obstáculos. Entretanto, suprir tal informação para um robô em tempo real é uma tarefa muito difícil. O trabalho apresentou um preditor do ambiente, que supre uma estimativa da sua configuração futura em tempo real pela fusão de dados multi-sensoriais. O preditor foi implementado através de uma rede neural artificial (ANN), que foi treinada via um algoritmo chamado "*relative-error-backpropagation*" (REBP). O algoritmo REBP possibilita à ANN estimar dados de saída com um erro relativo mínimo. Segundo os autores, ele é melhor do que o algoritmo do *backpropagation* convencional (BP) para o caso desta aplicação preditiva. Assim sendo, o robô móvel pode reagir antecipando mudanças no ambiente. A sua execução foi verificada por predições via simulações e navegação experimental.

Tani e Fukumura (Tani, 1997) apresentaram um novo esquema para navegação baseada em sensores de um robô móvel. Nos seus trabalhos anteriores, foi formulado o problema da navegação direcionada a alvos como um problema inserido na dinâmica dos sistemas: trajetórias desejadas em um espaço podem estar inseridas em um adequado espaço de estados interno baseado nas informações dos sensores, tanto que um único mapa do espaço de estados interno para os comandos dos motores possa ser estabelecido. Nesta formulação, uma rede neural recorrente é empregada, mostrando que um adequado espaço de estados interno pode ser auto-organizado através do treinamento supervisionado de sequências relacionadas com os sensores e motores. Esta auto-organização é decorrente de uma outra rede neural, uma rede de Kohonen, sendo que a sua saída é a informação de entrada para a rede neural recorrente, relacionada com as informações dos sensores. O experimento foi conduzido através de um robô móvel real equipado com um sensor de alcance (laser), demonstrando a validação do esquema apresentado no trabalho em um ambiente ruidoso do mundo real.

Janét et al. (Janét, 1997) apresentaram e compararam duas abordagens baseadas em redes neurais para auto-localização global (GSL) para robôs móveis autônomos usando: uma rede neural de Kohonen e uma rede neural chamada "*region-feature neural network*" (RFNN). Ambas as abordagens classificam regiões discretas no espaço (nós topográficos), de uma maneira semelhante ao reconhecimento de caracteres óticos (OCR). Isto é, os dados do mapeamento sonar assumem a forma de um caractere único para esta região. Portanto, é possível que um veículo autônomo possa determinar em que local ele está, através de dados sensoriais colhidos da exploração. Com uma rotina de exploração robusta, o GSL pode solucionar o problema em relação ao tempo, a translação e a invariância no giro. O GSL pode solucioná-lo também, tornando-se independente do robô móvel usado e do seu coletor de dados sensorias. Isto sugere que um simples robô pode transferir seu conhecimento das várias regiões aprendidas para outros robôs móveis. A taxa de classificação de ambas as abordagens são comparadas, validando a idéia apresentada.

Dracopoulos (Dracopoulos, 1998) apresenta a aplicação de uma rede Perceptron com Múltiplas Camadas (MLP) para o problema do planejamento de rotas para um robô móvel, e particularmente para a tarefa da navegação em labirintos. Em publicações anteriores, foram mostrados resultados implícitos no qual o treinamento da rede neural MLP falhou, por causa da utilização de dados não confiáveis. A rede neural MLP utilizada é treinada para achar a direção na qual o robô pode se movimentar até o próximo passo. A arquitetura da rede é do tipo 4-10-10-1, ou seja, com quatro entradas, que são as coordenadas das posições corrente e alvo do robô, duas camadas intermediárias, cada uma com 10 neurônios, e uma saída que indica a direção que deve ser adotada. Para o treinamento, foram usados 174 padrões distintos, conjunto este generalizado de uma série formada por 50 trajetórias consideradas ótimas.

Tse et al. (Tse, 1998) apresentaram um robô móvel para o uso doméstico, empregado para a realização de funções diárias tais como: limpeza e polimento do chão, entrega de pequenos objetos e patrulha de segurança. Para realização destas tarefas, este robô deve navegar por um ambiente fechado, que apresenta contínuas mudanças, e evitando obstáculos. Um algoritmo para auto-navegação foi desenvolvido para tais propósitos, usando redes neurais. Com a ajuda da auto-

navegação, este robô pode ser usado para cobrir inteiramente a área do chão com o propósito de limpá-lo ou para mover-se de um lugar para outro para realizar entregas.

No caso do sistema de navegação usado para a limpeza do chão, foi utilizada uma rede neural do tipo MLP (Perceptron Multicamadas) com treinamento via backpropagation e com arquitetura 4-4-4, ou seja, com quatro entradas, que descrevem os espaços em volta do robô (frente, direita, esquerda e atrás) e recebem valores iguais a 0, 1 e 2, representando respectivamente as seguintes configurações: sem obstáculos ou espaço livre, espaço que deve ser limpo e espaço ocupado por obstáculo. Quatro saídas, que correspondem às quatro direções de movimento do robô no próximo passo, nomeadas de: mover para frente, virar à esquerda, virar à direita e reverter. Quando uma das saídas recebe o valor 1, significa que a sua respectiva ação deverá ser adotada.

Para o outro caso, ou seja, a navegação para a tarefa de entrega, foi adotada uma outra rede neural MLP, com arquitetura 16-20-8, sendo a seguinte configuração para as entradas: 8 representam as direções do alvo e 8 representam as direções ocupadas por obstáculos, detectados por sensores. Vinte neurônios na camada intermediária e as 8 saídas que representam as 8 possíveis direções de movimento (norte, sul, leste, oeste, noroeste, sudeste, nordeste e sudoeste). Nestas saídas, a rede utiliza valores binários (0 e 1).

Billard (Billard, 1999) descreve a proposta de uma arquitetura conexionista, chamada de *Dynamic Recurrent Associative Memory Architecture* (DRAMA), para o controle dinâmico e aprendizado de robôs autônomos. Esta arquitetura, cuja primeira inspiração foi o modelo de memória associativa proposta por Willshaw (um modelo abstrato do hipocampo), é uma rede neural recorrente (*time-delay*), usando para sua atualização regras Hebbianas. Esta abordagem permite o aprendizado de regularidades no espaço temporal e de séries temporais discretas nas suas sequências de entrada, na presença de uma importante quantidade de ruído.

No Brasil, Vaz e Fabro (Vaz, 1999) apresentaram uma proposta de integração entre a abordagem de planejamento de trajetória baseada em um pré-mapeamento do ambiente , e a abordagem baseada em sensores que busca no desvio de objetos dinâmicos encontrados durante a execução da trajetória. O SNNAP (Sistema Neural de Navegação em Ambientes Pré-Mapeados)

é dividido em dois níveis: o primeiro destina-se ao mapeamento do ambiente e ao controle da trajetória do veículo; o segundo tem por objetivo desviar o veículo de possíveis obstáculos que foram previamente mapeados.

O desenvolvimento do SNNAP baseou-se no "Navegador em Redes Neurais" (Vaz, 1999), formado por duas redes neurais do tipo MLP, com topologia não recorrente e treinamento supervisionado. Sua arquitetura é do tipo 8-27-2, sendo utilizado o *bias*, as 8 entradas representam as medidas dos 8 sensores e as duas saídas representam o controle dos motores. Uma das redes é responsável pelo desvio de obstáculos e a outra por buscar alvos pré-estabelecidos no ambiente, neste caso fontes luminosas. O sistema prioriza a rede de colisões. Desta forma, quando há possibilidade de colisão, a rede de busca de alvos (luzes) não é utilizada, e somente ao se afastar do objeto com o qual havia possibilidade de colisão a rede de busca de alvos é reativada. Este controle faz com que o robô navegue pelo ambiente sem colidir com obstáculos, até que seus sensores de luminosidade sejam acionados, fazendo com que o robô vá na direção da luz.

Yang e Meng (Yang, 2000) apresentaram uma abordagem via rede neurais, com inspiração biológica, para o planejamento de movimentos, livres de colisões e em tempo real, para robôs móveis ou manipuladores robóticos em ambientes não estacionários. Cada neurônio na organização topográfica da rede neural tem apenas conexões locais, cuja dinâmica neural é caracterizada por equações de manobras. Assim, a complexidade computacional é linearmente depende da dimensão da rede neural. O movimento do robô em tempo real é planejado através da arquitetura da atividade dinâmica de uma rede neural sem nenhum conhecimento a priori da dinâmica do ambiente, sem explicitar a procura de locais livre ou rotas de colisões, e sem nenhum procedimento de aprendizado. Portanto, ele é computacionalmente eficiente. A estabilidade global da rede neural é garantida pela análise qualitativa e pela teoria de estabilidade de Lyapunov. A eficácia e a eficiência da abordagem proposta são avaliadas através de simulações.

2. 4.8. Navegadores Utilizando Lógica Nebulosa

Os sistemas nebulosos para navegação autônoma são outra instância de um sistema reativo (Oliveira, 1995). Estes sistemas são baseados na codificação de conhecimento na forma de relações lógicas entre variáveis nebulosas.

A forma clara das regras nebulosas facilita a síntese de conhecimento a partir da experiência de um especialista e permite a rápida implementação deste conhecimento em um sistema autônomo (Oliveira, 1995).

O projeto do sistema nebuloso requer, anteriormente à especificação do banco de regras, a definição das variáveis relevantes ao problema, do intervalo de variação destas variáveis nebulosas (universo de discurso), do particionamento deste intervalo e da associação destas partições a rótulos linguísticos e a uma função, conferindo um grau de pertinência de um estímulo daquele subintervalo àquela categoria, como está representado na figura 2.05.

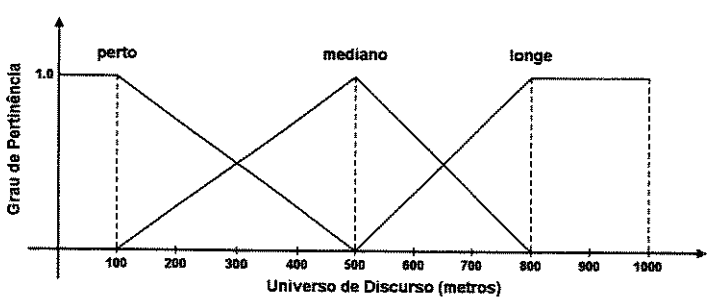


Figura 2.05 - Um exemplo de partição de um universo de discurso.

As regras são ativadas em paralelo e a resposta do sistema é formada por uma composição dos consequentes das regras ativadas. Como as informações provenientes dos sensores e os sinais esperados pelos atuadores não são nebulosos, é necessária a tradução dos valores nebulosos produzidos pelo mecanismo de interferência em valores *crisp* (defuzzificação) e vice-versa (fuzzificação).

O advento da lógica *fuzzy* forneceu uma ferramenta poderosa para o controle na presença de complexidades matemáticas, incertezas ou sistemas ruidosos. Controladores via lógica *fuzzy* ofereceram uma alternativa para os controladores convencionais existentes, de matemática pesada e recomendados para sistemas complexos. Controladores *fuzzy* são robustos na presença de perturbações, fáceis para projeto e implementação, e eficientes para sistemas que tratam com variáveis contínuas (Akbarzadeh-T, 2000).

As deficiências dos sistemas nebulosos residem na origem do conhecimento neles embutidos (Oliveira, 1995). Por não dispor de um mecanismo de aprendizagem integrado, todo o conhecimento é derivado da experiência humana. Esta dependência restringe a utilização do navegador a ambientes e situações já caracterizadas pelo operador. Dada a complexidade dos ambientes associados a problemas de controle autônomo, mesmo com a capacidade de manipular dados imprecisos, tais sistemas seriam de pouca eficácia.

Atualmente, já existem técnicas computacionais para se automatizar o processo de geração de um sistema fuzzy. Ou seja, o usuário não precisa mais se preocupar com o processo de partição do universo de discurso, definição do número de regras, posição e forma das funções de pertinência, etc. Não se trata de um problema superado, mas muito já se caminhou neste sentido. (Delgado, 2001).

A seguir serão descritos alguns trabalhos de navegadores utilizando lógica nebulosa apresentados nos últimos anos, mantendo-se uma ordem cronológica dos mesmos.

Graves et al. (Graves, 1993) descreveram o projeto e a implementação de uma plataforma autônoma móvel que usa lógica *fuzzy* para sua navegação e para evitar colisões. Tópicos específicos que foram discutidos incluem a motivação, análise racional, desenvolvimento e a avaliação do desempenho da plataforma móvel chamada de *FuzzBub*. A meta deste trabalho foi a construção de um robô móvel que pode ser controlado por um pequeno processador *on-board*.

O objetivo primário foi o uso de lógica *fuzzy* para a fusão de dados ruidosos dos sensores no caso de navegação em ambientes desconhecidos. O robô precisa rastrear um alvo (uma fonte de luz) e ao mesmo tempo evitar colisões com obstáculos. O sistema precisa também funcionar

adequadamente nas mudanças de ambiente. O aspecto recente sugere o uso do controle reativo, onde o robô móvel deve reagir às mudanças no ambiente. A principal motivação para o uso de lógica *fuzzy* neste projeto é que ela pode combinar dados de várias fontes. No projeto do robô foram primeiro usados três tipos de sensores, todos apresentando ruído, podendo exibir conflito de informações. Outra motivação para o uso de lógica *fuzzy* no projeto é que as características do ambiente utilizado são constantemente mudadas. De fato, como o robô se move por todo o ambiente, diferentes níveis de luminosidade no ambiente são frequentemente encontrados.

O método proposto tem algumas limitações. Primeiro, a complexidade da sua implementação e o seu custo computacional. Também, o método não faz a junção da operação de rastreamento do alvo com a operação de evitar obstáculos.

Oliveira (Oliveira, 1995) destaca a proposta do navegador nebuloso desenvolvido por Li (Li, 1994), pelo desempenho em situações complexas tipo "U". O navegador de Li demonstra um ótimo desempenho tanto em ambientes simples como corredores quanto em ambientes mais complexos como obstáculos em "U". Li utiliza quatro sensores (distâncias a obstáculos situados à esquerda, à direita e à frente do veículo, direção atual de movimento) e dois atuadores: as velocidades imprimidas às rodas esquerda e direita. As regras foram determinadas experimentalmente. Quatro comportamentos essenciais foram identificados e implementados por Li: busca de alvos, seguir paredes, evitar colisões e diminuir a velocidade em curvas e caminhos estreitos.

Choi et al. (Choi, 1998) descreveram que quando um robô se confronta com obstáculos grandes, não convexos ou espalhados, ele está apto a ser apanhado em uma armadilha nos pontos de mínimos locais. Este trabalho sugere um novo algoritmo via lógica *fuzzy* para resolver este problema usando alguns dados sensoriais simples. O algoritmo proposto tem duas camadas hierarquicamente estruturadas: uma camada mais baixa para evitar obstáculos e para se aproximar de alvos, e uma camada mais alta para a combinação destas lógicas. Alguns resultados de simulações computacionais para um robô móvel equipado com três sensores de proximidade mostram que o algoritmo de navegação sugerido é muito eficaz nos ponto de mínimos locais em ambientes desconhecidos. A estratégia de navegação é baseada na combinação de regras *fuzzy* para a aproximação de alvos e para evitar obstáculos.

Gasós e Rosetti (Gasós, 1999) apresentaram uma abordagem baseada nos conjuntos *fuzzy* para o problema da navegação de robôs móveis em ambientes desconhecidos. Conjuntos *fuzzy* são usados para representar as incertezas que são inerentes na percepção do ambiente através dos sensores do robô. Estas incertezas estão propagadas no processo de construção de mapas do ambiente. O mapa inicial construído pelo robô é tão usado para a auto-localização como para a navegação contínua no mesmo ambiente. A nova informação coletada pelos sensores é empregada no mapa inicial e as transformações trazidas juntamente por ela é usada para corrigir o cálculo dos erros. Representação de incertezas é o aspecto chave desse processo, desde que elas permitam a detecção imediata de pares de limites coincidentes, permitindo então em tempo real a auto-localização. A abordagem é ilustrada por experimentos em um ambiente de trabalho (um escritório).

Maaref e Barret (Maaref, 2000) apresentaram o problema de navegação de um robô móvel em um ambiente fechado e desconhecido ou parcialmente desconhecido. Um método de navegação baseado na combinação de comportamentos elementares foi desenvolvido para um ambiente desconhecido. Muito destes comportamentos são arquivados por meio da aplicação de sistemas *fuzzy*. O navegador proposto combina dois tipos de comportamentos para evitar obstáculos, um para obstáculos convexos e outro para obstáculos côncavos.

No caso do ambiente parcialmente conhecido, um método híbrido é usado para explorar as vantagens das estratégias de navegação global e local. A coordenação destas estratégias é baseada na aplicação dos sistemas *fuzzy*, que envolve uma comparação *on-line* entre a cena real e outra memorizada. Ambos os métodos foram implementados em uma miniatura de robô Khepera, que é equipado com sensores pouco precisos. O bom resultado obtido ilustra a robustez de um abordagem com lógica *fuzzy* considerando as imperfeições dos sensores.

2. 4.9. Navegadores Evolutivos (Genéticos)

Os chamados navegadores evolutivos ou genéticos são baseados na teoria dos algoritmos genéticos, que são mecanismos com rotinas robustas, usados para problemas de otimização e foram modelados a partir da Teoria Darwiniana da Evolução Natural (Akbarzadeh-T, 2000).

A seguir, serão descritos alguns trabalhos de navegadores evolutivos apresentados nos últimos anos, mantendo-se uma ordem cronológica dos mesmos.

Almássy e Vinkhuysen (Almássy, 1994) expandiram o navegador neural de Verschure, acrescentando um sensor de movimento e mais um mecanismo de aprendizagem, um algoritmo genético. Os experimentos visavam o estudo do surgimento do comportamento grupal de robôs, daí o detetor de movimentos, na verdade um detetor de outros agentes. O algoritmo genético foi empregado para determinar os valores ótimos dos parâmetros do mecanismo Hebbiano de aprendizagem.

As características dos robôs que foram evoluídas incluem parâmetros e mecanismos do seu controle interno. Almássy e Vinkhuysen argumentaram a necessidade do uso de técnicas evolutivas para a criação de comportamentos adaptativos em ambientes dinâmicos. Foi usado neste trabalho um simulador chamado de *BUGWORLD* desenvolvido por Almássy (Almássy, 1993), onde através de simulações ele busca a melhor interação possível do robô com o seu ambiente de trabalho, garantido a validação do método através da sua combinação com estudos em robôs reais. As suas experiências demonstraram ser possível o desenvolvimento de dispositivos capazes de navegação autônoma através de mecanismos de auto-organização e com o mínimo de interferência humana.

Hocaoglu e Sanderson (Hocaoglu, 1998) descreveram um flexível e eficiente algoritmo para planejamento de rotas baseado nos conceitos da computação evolutiva. Uma representação, da rota recente, interativa e com várias resoluções, é usada via algoritmos genéticos. O uso da representação da rota com várias resoluções pode reduzir a duração esperada da procura no problema do planejamento de rotas. Se uma rota de sucesso é encontrada no início da procura

hierárquica (a partir de um nível mais baixo de resolução), então o avanço da expansão da parte procurada da rota não é necessário. Esta vantagem é mapeada dentro do espaço de procura codificado, e ajusta da mesma forma, o tamanho da cadeia.

No planejador de rotas evolucionário, os candidatos individuais evoluem de acordo com o espaço de trabalho, tal que o cálculo do espaço configurado não é requerido. Este algoritmo pode ser aplicado para o planejamento de rotas de um robô móvel, para montagem, para o problema dos *piano-movers* (Canny, 1989) e para a articulação de manipuladores. A eficácia do algoritmo é demonstrada em alguns problemas de planejamento de rotas multi-dimensionais.

2. 4.10. Navegadores Híbridos

A aplicação da combinação de técnicas de computação flexível *(soft computing)*, como redes neurais, lógica fuzzy, algoritmos genéticos e outras, é cada vez mais popular entre vários pesquisadores, devido sua habilidade para manusear imprecisões e incertezas que estão com frequência presentes em muitos problemas do mundo real (Pratihar, 1999).

Os chamados navegadores híbridos são aqueles que utilizam sistemas híbridos de inteligência artificial como: os Neuro-Genéticos, os Neuro-Fuzzys, e os Genético-Fuzzys. Estes sistemas híbridos oferecem as vantagens dos seus componentes e compensam mutuamente as deficiências destes componentes (Oliveira, 1995). Com isso, o objetivo da combinação destas abordagens é para que os pontos fortes de cada uma delas possa compensar as dificuldades existentes na outra abordagem combinada. As características mais interessantes incluem auto-organização, processamento de informações imprecisas e capacidade de aprendizagem.

No Brasil, Oliveira (Oliveira, 1995) propôs um método de controle autônomo que combina uma rede neural, teoria de sistemas nebulosos e um algoritmo genético. A sinergia destes três paradigmas de computação oferece uma estrutura computacional paralela e robusta com fácil inserção/extração de conhecimento e capaz de aprendizagem não-supervisionada. O método proposto foi aplicado ao problema de navegação autônoma de veículos em um ambiente simulado. Os resultados demonstram que as limitações observadas podem ser atribuídas à quantidade e tipo de conhecimento utilizado na escolha dos parâmetros (Sensores e representação

de conhecimento). O método de navegação em si provou ser robusto e confiável, desenvolvendo comportamentos com desempenho comparável aos de um agente projetado heuristicamente.

Akbarzadeh-T et al. (Akbarzadeh-T, 2000) descreveram que redes neurais (NN), algoritmos genéticos (GA), e programação genética (GP) aumentaram, juntamente com esquemas baseados em lógica *fuzzy*, a aplicação da inteligência artificial em sistemas automatizados. A combinação híbrida de tais abordagens acrescentaram: raciocínio, adaptação, e habilidades de aprendizado. Neste trabalho, três abordagens híbridas dominantes para o controle inteligente são aplicadas experimentalmente para o controle de vários robôs, os quais são atualmente investigados no Centro de Engenharia de Controle de Autônomos da NASA.

Os controladores híbridos consistiam de: um controlador hierárquico NN-*fuzzy*, aplicado no acionamento de motores (direção), um controlador hierárquico GA-*fuzzy*, aplicado ao controle de posição das juntas flexíveis do robô, e um controlador baseado em comportamentos GP-fuzzy, aplicado à tarefa da navegação em um robô móvel. Várias características fortes de cada uma destas combinações híbridas são discutidas e utilizadas nestas arquiteturas de controle.

A arquitetura NN-*fuzzy* leva as vantagens das redes neurais para o manuseio de padrões de dados complexos, a arquitetura GA-*fuzzy* utiliza a habilidade dos algoritmos genéticos para otimizar parâmetros das funções de pertinência para a improvisação da resposta do sistema, e a arquitetura GP-*fuzzy* utiliza a capacidade da manipulação simbólica da programação genética para envolver os conjuntos de regras *fuzzy*.

O controlador *Neuro-fuzzy* usa a capacidade de um controlador via lógica *fuzzy* para controle de sistemas, sem conhecimento a priori das suas características dinâmicas, que são naturalmente não lineares e cuja modelagem matemática é dificultada. Então existe a necessidade de um controlador não linear *fuzzy* para adaptá-lo às mudanças nos parâmetros do modelo, condições de operação, etc. Isto requer um mecanismo de informação sobre estas mudanças, que podem ser acumuladas e usadas para a adaptação de um controlador *fuzzy*. A extração do conhecimento abstrato do sistema é realizada por redes neurais e sua capacidade de aprendizado é usada na designação de controladores *fuzzy*. Isto torna o controlador robusto e inteligente, consciente das mudanças no ambiente e nas condições físicas (Akbarzadeh-T, 2000).

Controladores *Genetic-fuzzy* não requerem a avaliação do gradiente, portanto eles são aplicáveis para resolver um grande número de problemas de otimização, incluindo a determinação de parâmetros ótimos de um conjunto de regras *fuzzys*. Algoritmos genéticos têm demonstrado a habilidade de codificação e representação de parâmetros nos domínios do conhecimento *fuzzy*, tais como os conjuntos de regras *fuzzy* e as funções de pertinência (Akbarzadeh-T, 2000).

A seguir serão descritos outros trabalhos de navegadores híbridos apresentados nos últimos anos.

Floreano (Floreano, 1996) descreveu uma abordagem *neuro-genetic* através da evolução de uma rede neural recorrente (tempo discreto) para o controle real de um robô móvel. Em todos os seus experimentos, o procedimento evolutivo é carregado inteiramente fora do robô físico, sem a intervenção humana. Foi mostrado o desenvolvimento autônomo de um conjunto de comportamentos para localização de um carregador de baterias e para retornos periódicos, para que ele possa ser arquivado via o levantamento de conflitos existentes no projeto das interações robô/ambiente que foram empregadas em experimentos preliminares. O surgimento do comportamento de retorno é baseado no desenvolvimento autônomo de um mapa topográfico neural interno (não desenhado previamente) que permite a mudança da trajetória corrente do robô em função da sua localização e da localização do suprimento de energia.

Li et al. (Li, 1997) apresentaram uma arquitetura de um sistema *neuro-fuzzy* para o controle baseado em comportamentos de um robô móvel em um ambiente desconhecido. Uma rede neural é usada para ambientes não conhecidos. Suas entradas são os ângulos entre o robô e um alvo específico, e a informação de aproximação é dada por uma série de sensores de ultra-som. A saída da rede neural é referente a uma direção de movimento treinada para a navegação do robô.

Neste sistema é abordado a metodologia do controle baseado em comportamentos: para análise e decomposição de tarefas complexas baseados nos padrões estímulos-respostas dos comportamentos e para formular quantitativamente cada tipo de característica simples do comportamento com conjuntos *fuzzy* e regras *fuzzy*, assim como, para o conflito de coordenadas e

a competição entre os múltiplos tipos de comportamentos pelo raciocínio *fuzzy*. Baseadas na direção de movimento referente e nas distâncias entre o robô e os obstáculos, diferentes tipos de comportamento são fundidos por lógica *fuzzy* para controlar as velocidades das duas rodas traseiras do robô. Simulações experimentais mostram que o sistema *neuro-fuzzy* proposto pode aperfeiçoar o desempenho da navegação em ambientes complexos e desconhecidos.

2. 4.11. Navegadores via Aprendizado por Reforço

A tarefa básica relativa ao comportamento de um sistema baseado no aprendizado por reforço consiste em como encontrar uma política ótima (isto é, de custo mínimo) após experimentar várias sequências possíveis de ações e observar os custos incorridos e as transições de estado que ocorrem (Janusz, 1995).

Janusz e Riedmiller (Janusz, 1995) descreveram que o aprendizado por reforço é um paradigma promissor para o treinamento de controladores inteligentes. A capacidade de aprendizado de uma arquitetura de controle baseada em redes neurais é mostrado pela aplicação no controle de um robô móvel em um ambiente desconhecido.

Fundamentada em informações multi-sensoriais providas por quatro sensores de infravermelho, o controlador tem que aprender a evitar colisões, recebendo apenas um sinal final do treinamento de sucesso ou falha.

O controle do aprendizado pode ser classificado pela quantidade de conhecimento a priori requerida para cumprir a tarefa aprendida. O problema básico do aprendizado pode ser declarado da seguinte forma: qual ação na sequência das ações deve ser mudada de ordem para conseguir uma recompensa satisfatória no final. Depois, foi descrita a arquitetura do controlador de aprendizado neural baseado na técnica de *Q-learning*. A aprendizagem Q (*Q-learning*) é um procedimento de programação dinâmica incremental que determina a política ótima de uma maneira passo a passo (Haykin, 1999). A tarefa básica do sistema é controlar dois motores de uma miniatura de robô móvel, Khepera, baseada na informação dos quatro sensores. O controlador incrementalmente aprende a evitar obstáculos, baseado no sinal final do treinamento de sucesso ou falha.

Stafylopatis e Blekas (Stafylopatis, 1998) descreveram que arquiteturas via aprendizado por reforço representam uma forma de procura direta e em tempo real no espaço de controle. Isto faz com que elas sejam apropriadas para modificar as regras do controle de modo a obter o aperfeiçoamento do desempenho do sistema. A eficácia da estratégia de aprendizado por reforço é estudada através do treinamento de um sistema classificador com aprendizado (*Learning Classifier System* - LCS), que controla o movimento de um veículo autônomo na simulação por rotas incluindo giros para esquerda e para direita.

O LCS compreende um conjunto de regras de condições/ações (classificadores) que competem para o controle do sistema e evoluem por meio de algoritmos genéticos (GA). A evolução e operação dos classificadores dependem de um mecanismo apropriado para a atribuição de créditos baseado no aprendizado por reforço. O desempenho do movimento do veículo usando a abordagem evolucionária proposta é superior quando comparada com o outra abordagem (neural) com aprendizado por reforço que foi aplicada anteriormente para a mesma configuração do problema.

2.4.12. Navegadores via Potencial de Campo

O método de potencial de campo consiste em equacionar funções de potencial que atraia o robô móvel para posições (ou configurações) específicas, ou então, repelindo quando estiver próximo de obstáculos (ou configurações indesejadas) (Mendeleck, 1995).

Dozier et al. (Dozier, 1998) apresentaram um relacionamento entre o potencial de campos artificiais (APF) e a otimização com restrições (*Constrained Otimization*), aplicados no planejamento de movimentos (navegação). Também, foi apresentado um algoritmo genético *Hill-Climbing* simplificado (SGHC), usado para navegação de um robô em um ambiente via a abordagem APF.

Na abordagem APF, obstáculos dentro de um ambiente emitem uma força repulsiva em relação ao robô, enquanto que o alvo emite uma força atrativa. A navegação baseada no APF

pode facilmente ser designado como um problema de otimização com restrições (COP) e ser resolvido através da otimização com restrição com base evolutiva (ECOs).

2.4.13. Navegadores via Pesquisa em Grafos/Árvores

A pesquisa em grafos/árvores consiste em um processo de busca por configurações (*C-space - Configuration Space*) representadas em grafos ou diagramas (Mendeleck, 1995).

No Brasil, Heuberger (Heuberger, 1990) propôs um algoritmo para a determinação automática de caminhos para um manipulador com duas juntas rotacionais, na presença de obstáculos conhecidos e fixos. A detecção de colisão é simplificada transformando-se os obstáculos para o espaço de configuração do manipulador. A determinação do caminho consiste então, em achar a linha de menor comprimento no espaço de configuração, ligando os pontos referentes às posições inicial e final, evitando as regiões de colisão. Esta linha será determinada pelo algoritmo de procura em grafos A*, que sempre encontra o caminho ótimo segundo um critério adotado, neste caso, o de menor movimento das juntas.

O algoritmo proposto também pode ser usado para o problema de navegação de um robô móvel, isto é, para a determinação automática do caminho de um veículo na presença de diversos obstáculos, por exemplo em uma fábrica. Para este caso o mapa de configuração é uma cópia do espaço físico real, já que as coordenadas destes coincidem, causando com isso um aumento dos obstáculos no mapa, dependendo das dimensões do objeto que se move (Heuberger, 1990).

Dudek et al. (Dudek, 1996) consideram o problema da construção de mapas de ambientes desconhecidos por agentes autônomos, tais como, robôs móveis. Por motivo da informação posicional exata ser sempre difícil de se achar, foi considerado o problema da exploração na ausência destas informações. Ambientes são representados por grafos (não necessariamente planos) consistindo de um número fixo de locais discretos ligados por rotas bidirecionais.

Um robô móvel é designado para a tarefa de criar um mapa topográfico, isto é, uma representação por grafos dos locais no ambiente e suas conectividades, através da sua movimentação de lugar para lugar ao longo das rotas encontradas. Foi assumido que o robô pode

constantemente enumerar as bordas dos grafos, partindo dos vértices (que podem ser designados por um ordenação cíclica). Ele pode detectar as bordas e contá-las, mas não pode diretamente sentir as marcações (etiquetas) associadas com um local ou com uma borda. Em princípio, este tipo de representação pode ser usada para ambientes não espaciais tal como as redes computacionais.

2.4.14. Navegadores via Filtros de Kalman

Outra abordagem muito utilizada nos problemas de navegação autônoma é o emprego da teoria dos chamados filtros de Kalman. As características inovadoras dos filtros de Kalman incluem (Haykin, 1999):

- A teoria é formulada em termos de conceitos de espaço de estados, fornecendo uma utilização eficiente da informação disponível.
- A estimativa do estado é calculada recursivamente; isto é, cada estimativa atualizada do estado é calculada a partir da estimativa anterior e dos dados atualmente disponíveis, e com isso apenas a estimativa prévia necessita de armazenamento.

Haugue e Tillett (Hague, 1996) apresentaram esquemas possíveis de navegação para um veículo robótico horticultor para ambientes abertos. Problemas enfrentados na aplicação de técnicas usadas com sucesso pela indústria de veículos guiados automatizados são ilustrados, com referência em particular para o problema de achar a posição de um veículo em coordenadas absolutas no plano cartesiano. Um esquema de navegação alternativo é proposto, o qual depende apenas das informações sensoriais locais.

Rotas demandadas são especificadas sem referência da posição absoluta do robô, no seu lugar de fileiras de colheitas são empregadas, como uma ajuda para a navegação. A posição do veículo é medida em relação à rota demandada. Foi descrito no trabalho, um algoritmo baseado na teoria dos filtros de Kalman, o qual mantém uma estimativa da posição do veículo nestas coordenadas do plano relativas a rota. A entrada é obtida pelo sensor de visão do sistema, que é capaz de localizar fileiras de colheitas através de uma imagem de vídeo. Este sistema permite

guiar um veículo experimental entre fileiras de colheitas artificiais, com uma velocidade acima de 1,5 m/s e com uma tolerância de ±20 mm.

Feder et al. (Feder, 1999) descreveram que a tarefa de construir um mapa de um ambiente desconhecido e ao mesmo tempo usar este mapa para a navegação é um problema central nas pesquisas da robótica móvel. Foi apresentado, o problema de como realizar ao mesmo tempo e de forma adaptativa o mapeamento e a localização (Concurrent Mapping and Localization - CML) usando sonares.

Mapeamento estocástico é uma abordagem baseada nas características do CML, generalizadas pelo uso da teoria do filtro de Kalman estendido para que o veículo incorpore a localização e o mapeamento ambiental. Foi descrita uma implementação do mapeamento estocástico que usa uma estratégia de associação de dados vizinhos, de proximidade atrasada, para inicializar uma nova característica dentro do mapa, jogar medidas para um mapa das características, e apagar as características *out-of-date*. Esta técnica é demostrada via simulações, experimentos sonares em terra e experimentos sonares submersos, mostrando que o veículo tende a selecionar diferentes objetos no ambiente.

Jetto et al. (Jetto, 1999) descreveram que um requisito básico para um robô móvel autônomo é a capacidade de elaborar medidas sensoriais para a sua localização em relação a um sistemas de coordenadas. Para este propósito, os dados fornecidos por sensores odométricos e sonares são fundidos por um filtro de Kalman estendido. O desempenho do filtro é melhorado por um ajuste em tempo real da entrada e pela medida da covariância do ruído obtida por uma definição adequada de um algoritmo de estimação.

Uma determinação exata da localização é um requisito fundamental quando tratamos com os problemas de controle de robôs móveis. Dois diferentes tipos de localização existem: relativa e absoluta. A primeira é realizada através de medidas providas por sensores que medem a dinâmica das variáveis internas do veículo. O inconveniente deste método é que o erro é acumulado a cada medida. Este acúmulo de erro degrada a posição e a orientação estimadas do veículo, especialmente para trajetórias longas e sinuosas. A localização absoluta é realizada processando os dados provenientes de um conjunto próprio de sensores que medem alguns parâmetros do

ambiente no qual o veículo está operando. Um conjunto de sonares é geralmente usado para o sensoriamento externo, medindo a sua distância em relação às partes conhecidas do ambiente. Estes sensores são também usados para guiar o veículo autônomo evitando obstáculos em ambientes desconhecidos. O inconveniente das medidas absolutas é sua dependência em relação às características do ambiente. Possíveis mudanças nos parâmetros do ambiente pode aumentar a probabilidade de interpretação errônea de medidas providas pelo algoritmo de localização.

A direção atual é para o uso de sensores de diferentes naturezas e para pesar propriamente dados relativos de acordo com as suas credibilidades. Para este propósito, técnicas de filtragem de Kalman representam uma ferramenta poderosa.

2. 5. O Modelo Conexionista Proposto por A. Mendeleck em 1995

No Brasil, Mendeleck (Mendelck, 1995) apresentou na sua tese de doutorado um modelo conexionista, baseado em estruturas neurais artificiais, para o mapeamento de uma área de trabalho, formação de rotas e a geração de pontos de referências que podem auxiliar na implementação de caminhos por ambientes desestruturados. Nesta tese, foi proposta uma estrutura neuronal artificial com auto-aprendizado para o auxílio à geração de trajetórias em um ambiente dinâmico e desconhecido.

Buscando uma similaridade biológica, a inspiração para este modelo foi baseada em três frentes de estudo:

- Sistema Neural Biológico (principalmente o Cerebelo e o Hipocampo);
- Teorias de Aprendizado (principalmente a proposta por R. M. Gagné);
- Redes Neurais Artificiais (principalmente rede tipo perceptron com treinamento via *backpropagation*).

O objetivo do sistema proposto por Mendeleck é fazer com que ele seja capaz de "sugerir" uma trajetória a ser seguida por um mecanismo em movimento em um ambiente desestruturado. O sistema proposto apresentava a seguinte configuração:

- *O mecanismo sensor.* É o elemento que contém sensores de posição e de aproximação. Suas funções são fornecer a posição absoluta do mecanismo e a distância aos obstáculos;
- *A estrutura de aprendizado.* Consiste em um conjunto de estratégias neuronais que permitem o mapeamento e a navegação do mecanismo pela área de trabalho;
- *A estrutura neuronal.* É composta por uma série de redes neurais artificiais, cujas características permitem a implementação das estratégias de aprendizado e auxílio à geração de trajetórias.

A seguir, será abordada com mais detalhes a estrutura neural para o aprendizado, devido a um dos níveis dessa estrutura, responsável pelo mapeamento da área de trabalho, ter servido como inspiração para o sistema proposto nesta dissertação, que será descrito no capítulo 4.

2. 5. 1. A Estrutura Neural para o Aprendizado

O aprendizado tem por objetivo permitir a navegação do mecanismo físico, auxiliada pelo cursor, simulação do elemento físico referente ao dispositivo com sensores, em um ambiente que em princípio é desconhecido, de forma segura e determinística. O aprendizado cuja estrutura está representada na figura 2.06, tem o seu procedimento dividido em 4 fases:

I. Identificar o meio;
II. Gerar pequenas rotas;
III. Sequenciar essas rotas formando macro-rotas;
IV. Verificar "vias expressas".

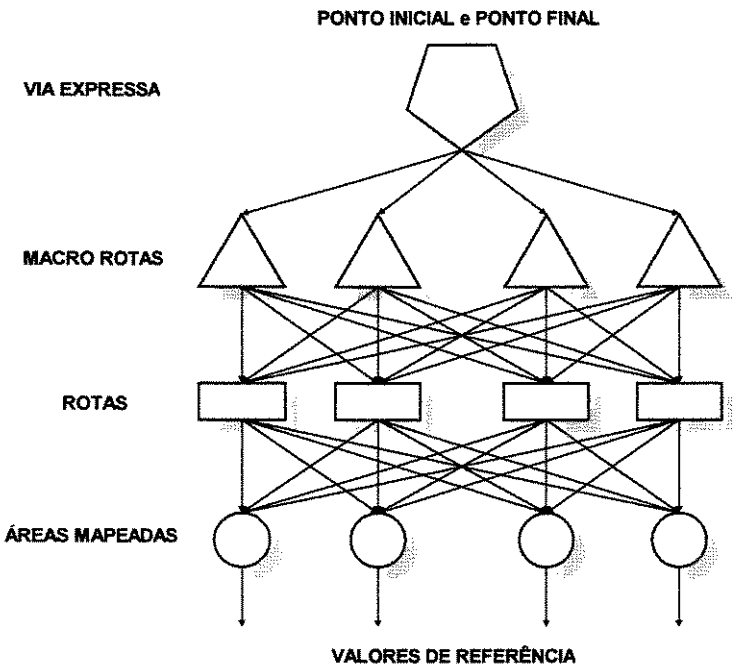


Figura 2.06 - Estrutura de Aprendizado (Mendelck, 1995).

A primeira fase consiste na identificação dos obstáculos imediatos na vizinhança do cursor. O sistema utiliza as informações provenientes dos sensores para "mapear (internamente) uma região circular". O diâmetro dessa região dependerá do alcance dos sensores, sendo considerado o menor diâmetro.

Durante o processo de aprendizado, o cursor executa movimentos pela área de trabalho mantendo a equidistância (equilíbrio de valores) na direção ortogonal à do sentido do movimento (centralização do cursor). O objetivo é aumentar a área mapeada pelos sensores.

A mudança de direção durante o aprendizado ocorre quando:

- A região a ser mapeada já foi identificada por alguma rede;
- Um obstáculo foi encontrado na direção do movimento ou;

- Os sensores, no sentido ortogonal ao do movimento, acusaram troca de valores (necessitando re-centralizar o cursor).

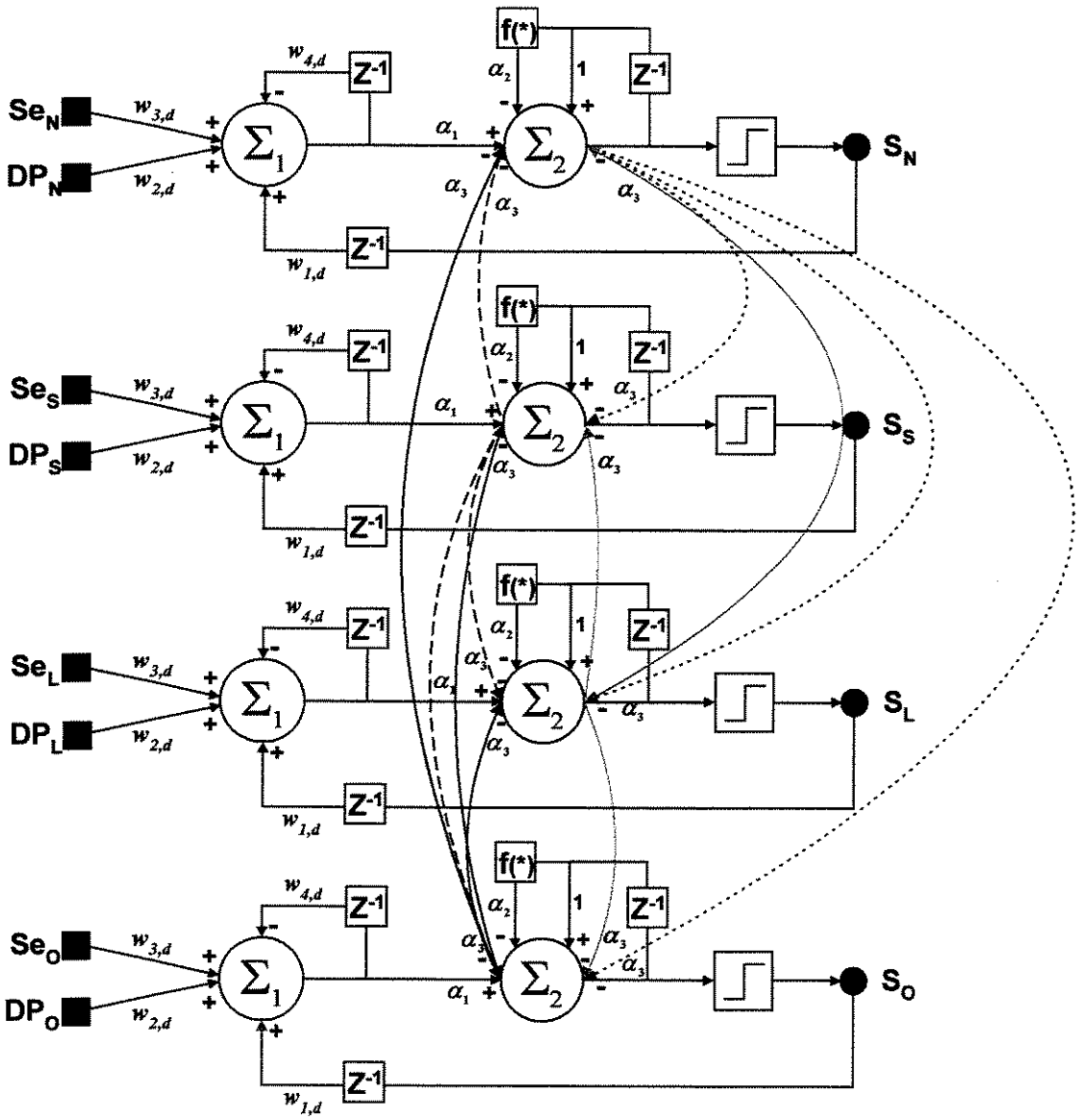
Para qualquer opção, o cursor deve ajustar as distâncias: se o sentido do movimento não for alterado, basta equalizar os valores, caso contrário, antes de executar a centralização, o cursor deve deslocar-se até encontrar uma nova referência, e então, reinicializar os procedimentos de ajuste.

Durante o processo de aprendizado, a direção do movimento é selecionada por dois grupos de neurônios acoplados sequencialmente. Os neurônios do 1° grupo recebem sinais de outros sub-sistemas responsáveis pela seleção da direção principal de movimento, sensores e sinais de retro-alimentação das portas de saída. Os valores de entrada são ponderados com valores pré-definidos das conexões sinápticas, e então totalizados na soma. A cada ciclo de cálculo, a soma sofre um decremento inversamente proporcional ao seu potencial de ativação.

A estrutura neural responsável pela seleção de uma nova direção de movimento durante o aprendizado está representada na figura 2.07.

O valor do potencial de disparo é proporcional ao valor da soma. Os pesos sinápticos ($w_{j,d}$) são previamente "calibrados" permanecendo constante durante o processo de aprendizagem. Não foi utilizado nenhum critério específico para atribuir valores aos pesos sinápticos. Simplesmente foi observado o comportamento do cursor em várias configurações da área de trabalho, e a partir daí foi concluído que o fator preponderante para orientar os movimentos era o valor dos sensores **Se** (possuem pesos sinápticos maiores). No entanto, sem critérios mais consistentes de seleção, em certas situações (principalmente área côncavas) o cursor deslocava-se aleatoriamente. Assim, foram agregados outros dois critérios: a *direção preferencial* **DP** e a *direção corrente de movimento* **S**, esta última com menor peso sináptico.

Para evitar a saturação da soma, foi introduzido um fator de esquecimento proporcional ao valor da soma. Como resultado, o movimento do cursor não sofre mudanças bruscas de direção e quando se faz necessário, o seu comportamento é o de manter a direção corrente de movimento, como uma "inércia" dificultando a alteração.



$Z^{-1} \longrightarrow$ atraso

$$\sqcap = \begin{cases} 0 \rightarrow \Sigma_2(t)_d \leq \vartheta \\ 1 \rightarrow \Sigma_2(t)_d > \vartheta \end{cases}$$

$$f(*) = e^{-(|1+|\Sigma_2(t)_d||)}$$

$$\vartheta = \begin{cases} \textbf{Valor mínimo se disparado} \\ \vartheta - \dfrac{1}{1+|\Sigma_2(t)_d|} \end{cases}$$

Figura 2.07 - Estrutura neural responsável pela mudança de direção no aprendizado.

Os neurônios do segundo grupo formam uma rede de uma camada. Sua função é selecionar o neurônio com maior soma (ou o que dispara primeiro). Os neurônios, cada qual representando uma direção, recebem sinais do primeiro grupo e sinais dos neurônios vizinhos. As interconexões têm caráter inibitório, resetando o valor da soma dos neurônios que não dispararam.

O fato de Mendeleck ter utilizado valores dos pesos sinápticos ($w_{j,d}$) previamente calibrados por métodos empíricos, baseados na observação do comportamento do cursor em várias configurações do ambiente, dificulta a reprodução ou uma possível alteração dos mesmos. Não foi usado, por exemplo, o ajuste destes valores via alguma técnica convencional de aprendizado, baseada em padrões que representariam o comportamento do cursor para as situações mais expressivas.

Esse fato motivou o uso de redes neurais artificiais com aprendizado supervisionado, para reproduzir o comportamento da estrutura responsável pela movimentação do cursor na fase do aprendizado, que está representada na figura 2.07.

2. 6. Conclusões

O estudo e a implementação de robôs móveis desperta há muito tempo o interesse de vários pesquisadores, principalmente na resolução do problema de navegação autônoma, descrito neste capítulo. A partir da década de 90, com o uso da inteligência artificial, houve um grande avanço nos estudos para a resolução deste problema, quando foram aplicadas as três características chaves de sistemas inteligentes, que são: a capacidade de representação, a capacidade de processamento de informação e a capacidade de aprendizado. Na área da inteligência artificial, destaca-se a utilização de redes neurais artificiais, que individualmente ou em composição com outro tipo de abordagem da mesma área, se mostraram eficientes na resolução do problema em questão.

Capítulo 3

Redes Neurais Artificiais

3. 1. Introdução

O estímulo inicial que conduziu ao desenvolvimento de modelos matemáticos de redes neurais, denominadas redes neurais artificiais, foi um esforço para entender mais detalhadamente o funcionamento do cérebro humano. O objetivo era construir mecanismos que operassem de modo similar, ou seja, que tomassem decisões, processassem informação, aprendessem, lembrassem e otimizassem da mesma forma e, se possível, até de forma mais eficiente que o cérebro humano (Von Zuben, 1999).

Tomando por base os protótipos até aqui desenvolvidos, é de consenso geral que este objetivo ainda está longe de ser atingido (Von Zuben, 1999). No entanto, continua elevado e em forte expansão o interesse na formalização e aplicação de modelos de redes neurais artificiais.

A razão para o estabelecimento desta aparente contradição é o fato do referido estímulo inicial ter dado lugar a evidências concretas e convincentes acerca do enorme potencial destas estruturas quando aplicadas na análise e síntese de sistemas não-lineares e na generalização de resultados expressivos já obtidos em outras área de atuação científica (Von Zuben, 1999). Dentre estas áreas, destacam-se estatística, teoria de informação, teoria de aproximação de funções, teoria de processamento de sinais, teoria de controle de processos.

Para alcançarem bom desempenho, as redes neurais empregam uma interligação maciça de células computacionais simples denominadas "neurônios" ou "unidades de processamento". No seu livro, Haykin (Haykin, 1999) oferece a seguinte definição de uma rede neural vista como uma máquina adaptativa:

Uma rede neural é um processador paralelo e distribuído constituído de unidades de processamento simples, que têm a propensão natural para armazenar conhecimento experimental e torná-lo disponível para o uso. Ela se assemelha ao cérebro humano em dois aspectos:

- *O conhecimento é adquirido pela rede a partir de sua interação com o ambiente, através de um processo de aprendizagem.*
- *Forças de conexão entre neurônios, conhecidas como pesos sinápticos, são utilizadas para armazenar o conhecimento adquirido.*

Uma rede neural extrai seu poder computacional através, primeiro, de sua estrutura distribuída e segundo de sua habilidade de aprender e portanto de generalizar. A "generalização" se refere ao fato de uma rede neural produzir saídas adequadas para entradas que não estavam presentes durante o seu treinamento (aprendizado). Estas duas capacidades de processamento de informação tornam possível para as redes neurais resolver problemas complexos (de grande escala. Na prática, contudo, as redes neurais não podem fornecer uma solução trabalhando individualmente. Em vez disso, elas precisam ser integradas em uma abordagem consistente de engenharia de sistemas. Especificamente, um problema complexo de interesse é decomposto em um número de tarefas relativamente simples, e atribui-se a redes neurais um subconjunto de tarefas que coincidem com as suas capacidades inerentes (Haykin, 1999).

O uso de redes neurais oferece as seguintes propriedades úteis e capacidades (Haykin, 1999):

- Não-linearidade;
- Mapeamento de Entrada-saída;
- Adaptabilidade;

- Resposta a Evidências;
- Informação Contextual;
- Tolerância a Falhas;
- Implementação em VLSI;
- Uniformidade de Análise e Projeto e
- Analogia Neurobiológica.

No campo das redes neurais artificias, buscando a solução do problema da navegação autônoma, foram estudadas neste trabalho as redes neurais do tipo Perceptron Multicamadas ou Perceptron de Múltiplas Camadas (MLP) por serem estas redes as mais utilizadas na literatura para resolução do mesmo (Oliveira, 1995) (Mendeleck, 1995) (Thrun, 1995) (Ortega, 1996) (Fabro, 1996) (Chang, 1997) (Dracopoulos, 1998) (Tse, 1998) (Vaz, 1999).

Com referência às aplicações de redes neurais, será enfatizada a tarefa de *classificação de padrões* (Mapeamento de Entrada-Saída), por ser este o tipo de problema abordado no sistema proposto para a geração de trajetórias, que será apresentado no capítulo 4.

3. 2. Redes Neurais Multicamadas

Redes neurais artificiais têm sido aplicadas com sucesso nos mais diversos problemas. Dentre as principais áreas de aplicação de redes neurais artificiais (Iyoda, 2000) destacam-se: sistemas de controle, reconhecimento de padrões e aproximação de funções. Embora existam inúmeras arquiteturas de redes neurais, a arquitetura multicamadas é, sem dúvida, a mais freqüentemente encontrada na literatura. Entre as razões para a sua popularidade podemos citar sua capacidade de *aproximação universal* e sua flexibilidade para formar soluções de qualidade para uma ampla classe de problemas, a partir de um mesmo algoritmo de aprendizado.

As redes neurais multicamadas apresentam pelo menos duas camadas: uma camada de entrada e uma camada de saída. Os neurônios da camada de entrada são neurônios especiais, cujo papel é exclusivamente distribuir cada uma das entradas da rede (sem modificá-las) a todos os

neurônios da camada seguinte. Para as camadas seguintes, cada neurônio corresponde a uma unidade de processamento simples, sendo conhecida como *perceptron* (Iyoda, 2000). Um exemplo de um perceptron, com n entradas está representado na figura 3.01. Nesta figura também estão representados os principais componentes de um neurônio artificial:

- Um conjunto de *sinapses* ou *conexões* de entradas, sendo cada entrada (x_i) ponderada por um peso sináptico (w_i). Neste caso, também está presente na figura 3.01 o *bias* ou entrada de polarização (x_0=1);

- Uma junção de soma, responsável pela combinação aditiva dos sinais de entrada, ponderados pelos respectivos pesos das sinapses do neurônio, sendo que u é chamado de nível de ativação interna ou potencial de ativação do neurônio;

- Uma função de ativação geralmente não-linear e de formato sigmoidal, representando um efeito de saturação na ativação de saída y do neurônio. Tipicamente a excursão da ativação de neurônio é confinada ao intervalo (0,1) ou (-1,1). Na parte inferior da figura 2.01 estão representados os quatro tipos de funções de ativação mais utilizados, sendo respectivamente: uma função sinal, uma função sigmoidal do tipo logística, uma função sigmoidal do tipo tangente hiperbólica e por último uma função linear (representando a ausência de saturação);

O perceptron foi objeto de intensa pesquisa durante os anos 50 e 60, mas em 1969 M. Minsky e S. Papert provaram matematicamente que este tipo de estrutura de processamento apresenta limitações importantes e podem ser aplicadas com sucesso a uma classe muito restrita de problemas (Iyoda, 2000). Mais especificamente foi provado que o perceptron é capaz de resolver apenas problemas *linearmente separáveis*.

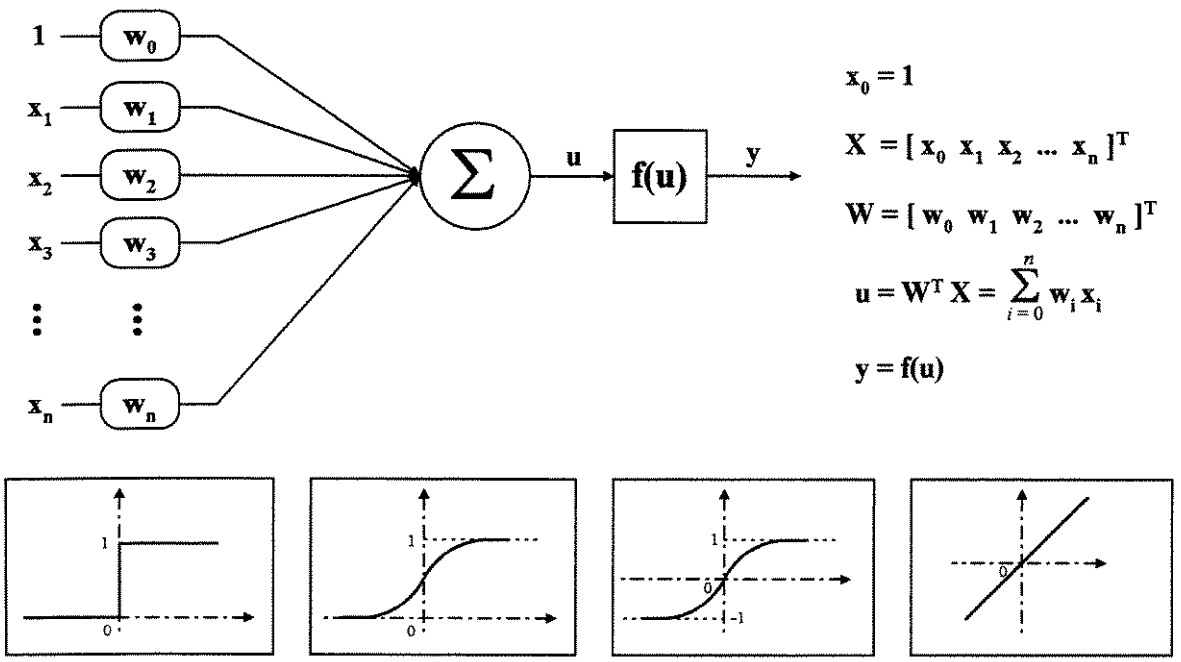


Figura 3.01 - O Perceptron e seus componentes.

No entanto, com a utilização de *redes de múltiplas camadas* com pelo menos uma camada escondida (camada que não é nem entrada, nem saída), ou *perceptron multicamadas* (MLP – *multilayer perceptron*), muitas das limitações apresentadas pelo perceptron deixam de existir. Na Figura 3.02 é representada uma rede neural com uma camada escondida. Por simplicidade, esta arquitetura é referida como uma rede n-m-p, isto é, n neurônios na camada de entrada, m neurônios na camada escondida e p neurônios na camada de saída. Como outro exemplo, uma rede neural com n entradas, m_1 neurônios na primeira camada escondida, m_2 na segunda camada escondida e p neurônios na camada de saída é dita ser uma rede n-m_1-m_2-p.

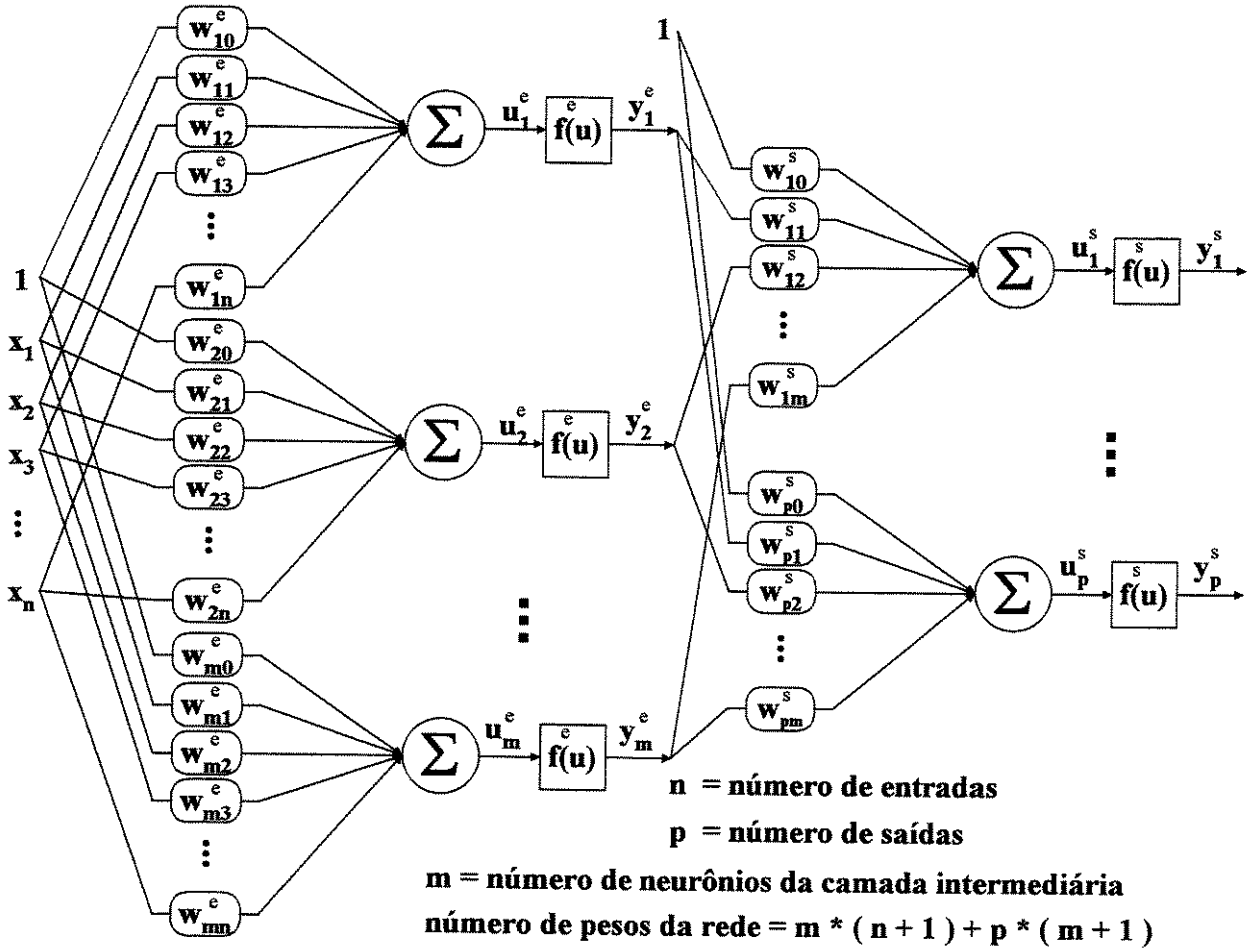


Figura 3.02 - A arquitetura de uma rede neural do tipo MLP.

3. 2.1. Aprendizado Supervisionado

A mais importante propriedade de uma rede neural artificial é sua capacidade de aprendizado. Uma rede neural aprende através de um processo iterativo de ajustes aplicados aos seus pesos sinápticos e limiares, o qual pode ser expresso na forma de um algoritmo computacional (Iyoda, 2000).

Não existe uma definição precisa universalmente aceita de "aprendizado" (Iyoda, 2000). No contexto de redes neurais artificiais, Haykin (Haykin, 1999) define aprendizado da seguinte forma: "*Aprendizado é um processo pelo qual os parâmetros livres de uma rede neural são adaptados através de um processo de estímulo pelo ambiente no qual a rede está inserida*". Assim, um processo de aprendizagem de uma rede neural implica na seguinte seqüência de eventos:

I. A rede neural é estimulada pelo ambiente de informação;

II. A estrutura interna da rede é alterada como resultado do estímulo;

III. Devido às alterações que ocorreram em sua estrutura interna, a rede tem modificada sua resposta aos estímulos do ambiente.

Figura 3.03 - Diagrama de blocos de um sistema com aprendizado supervisionado.

As redes neurais MLP têm sido aplicadas com sucesso para resolver diversos problemas difíceis graças ao seu tipo de treinamento, que utiliza o *aprendizado supervisionado*. Este tipo de aprendizado é caracterizado pela presença de um *"professor"* externo. A função do "professor" durante o processo de aprendizado é suprir a rede neural com uma *resposta desejada* a um determinado estímulo apresentado pelo ambiente. Definimos um *sinal de erro* como a diferença

entre resposta desejada e a resposta observada na saída da rede neural. Os parâmetros da rede são então ajustados de acordo com o sinal de erro. Na figura 3.03 está representado um diagrama de blocos de um sistema com aprendizado supervisionado.

Uma forma de implementar uma estratégia de aprendizado supervisionado em redes neurais é através de procedimentos iterativos de correção de erro. Seja $s_q(z)$ a resposta desejada para um neurônio q no instante z e seja $y_q(z)$ a resposta observada para este neurônio. A resposta é produzida por um estímulo (vetor) x(z) aplicado à entrada da rede da qual o neurônio q faz parte. O sinal de erro $e_q(z)$ para o neurônio q é definido como a diferença entre a resposta desejada e a resposta observada (Iyoda, 2000).

O objetivo do procedimento de aprendizado por correção de erro é minimizar alguma *função de custo* baseada no sinal de erro $e_q(z)$, de modo que a resposta observada de cada neurônio da rede se aproxime da resposta desejada para aquele neurônio, em algum sentido estatístico. De fato, uma vez definida uma função de custo, o problema de aprendizado torna-se um problema de otimização. Uma função de custo comumente empregada é a soma dos erros quadráticos:

$$\mathbf{J} = \frac{\mathbf{1}}{\mathbf{2}} \sum_{\mathbf{q} \in \mathbf{c_s}} \mathbf{e_q^2(z)} \tag{3.01}$$

Onde c_s contém o índice de todos os neurônios da camada de saída, de modo que o somatório é realizado sobre todos os neurônios da camada de saída da rede neural. A rede neural aprende através da minimização de J em relação aos pesos sinápticos da rede.

Nota-se que J define uma *superfície de erro* sobre o espaço dos pesos. Se P é o número de pesos ajustáveis da rede neural, então $J : \Re^P \rightarrow \Re$. A superfície de erro é caracterizada pela presença de mínimos locais e um ou mais mínimos globais. Os métodos de otimização utilizados na minimização de J usualmente recorrem à informação de gradiente do erro para ajustar os parâmetros da rede. Teoricamente, estes métodos sempre atingem um ponto de mínimo da

superfície de erro, mas nada se pode afirmar sobre a natureza (local ou global) do ponto de mínimo obtido a partir de uma condição inicial arbitrária (Iyoda, 2000).

3. 2.2. O Algoritmo de Retropropagação (Backpropagation)

O algoritmo mais utilizado no treinamento de redes neurais multicamadas com uma ou mais camadas intermediárias é chamado retropropagação do erro ou simplesmente retropropagação (*backpropagation*). Este algoritmo é baseado numa regra de aprendizagem que "corrige" o erro realizando um ajuste iterativo durante o treinamento, na direção oposta à do vetor gradiente da função custo.

Basicamente, o algoritmo de retropropagação fornece o vetor gradiente em dois passos de computação: o processamento direto (*feedforward*) e o processamento reverso (*backward*). No processamento direto, uma entrada é aplicada à rede neural e seu efeito é propagado pela rede, camada a camada. Durante o processamento direto, os pesos da rede permanecem fixos. No processamento reverso, um sinal de erro calculado na saída da rede é propagado no sentido reverso, camada a camada, e ao final deste processo calcula-se o vetor gradiente da função custo e os pesos são ajustados de acordo com a regra do gradiente descendente (*steepest descent*). Sempre que uma determinada entrada é apresentada à rede também é disponibilizada a resposta desejada para aquela entrada, caracterizando o aprendizado supervisionado (Iyoda, 2000).

3. 2.3. O Treinamento de uma Rede Neural com uma Camada Intermediária

Para a rede neural apresentada anteriormente na figura 3.02 considere as seguintes características de processamento:

- Cada camada contém um determinado número de neurônios do tipo perceptron (figura 3.01), onde todos possuem a entrada adicional de polarização $x_0=1$;
- O vetor de entrada de cada camada da rede neural é distribuído a todos os neurônios da mesma camada;

- As saídas de todos os neurônios de uma camada formam o vetor de entrada para a camada seguinte.

Antes de descrever quais os procedimentos necessários para o treinamento desta rede neural, serão descritas algumas características básicas sobre o treinamento de redes neurais do tipo MLP, que utilizam o algoritmo de retropropagação. São elas:

- *Tipo de treinamento* (Castro, 1998a). Como o treinamento de redes neurais do tipo MLP é um problema de otimização não-linear de uma função de custo, que mede o erro quadrático médio produzido pela saída da rede neural frente a uma saída desejada, pode-se classificá-lo de acordo com o método de otimização não-linear aplicado. Assim sendo, tem-se os chamados métodos de primeira ordem, como por exemplo o método do gradiente, onde é utilizada apenas a informação do gradiente da função de erro para ajustar os pesos da rede, e os métodos de segunda ordem, destacando-se o método do gradiente conjugado e escalonado proposto por Moller (Moller, 1993), onde além do vetor gradiente da função objetivo, é feito uso também da matriz hessiana da função erro.

- *Modo de treinamento* (Iyoda, 2000). Uma apresentação completa do conjunto de treinamento durante o processo de aprendizagem é denominada *época*. Dado um conjunto de treinamento, o algoritmo de retropropagação pode ser executado em dois modos distintos. O *padrão-a-padrão ou instantâneo (pattern mode)*: neste caso, os pesos são atualizados após a apresentação de cada padrão à rede neural, sendo neste modo importante apresentar os padrões em ordem aleatória a cada época, já que a ordem de apresentação dos padrões de treinamento não representa nenhuma informação que deva condicionar o processo de ajuste dos pesos. O outro modo é o por *lote ou batelada (batch mode):* neste, os pesos são ajustados somente após a apresentação de todos os exemplos que constituem uma época.

- *Inicialização dos pesos* (Iyoda, 2000). Este é o primeiro passo do algoritmo de retropropagação e corresponde à definição de um ponto inicial da superfície de erro. Uma boa escolha inicial dos pesos é fundamental para um bom desempenho do algoritmo de

retropropagação. Uma inicialização inadequada (ponto inicial mal localizado) pode fazer com que o algoritmo de treinamento fique preso em um mínimo local ou apresente problemas numéricos que de outra forma poderiam ser evitados. Como geralmente não temos nehuma informação que possa ser diretamente empregada na inicialização dos pesos da rede, um método comumente empregado é inicializar os pesos aleatoriamente, com distribuição uniforme sobre um pequeno intervalo em torno de zero. Outro método de inicialização (Castro, 1998b) (Castro, 1998c) utiliza os dados de treinamento para inicializar os pesos da rede de forma que os neurônios estejam operando inicialmente nas proximidades de sua região mais linear, sendo proposto um paradigma híbrido que explora as informações contida no conjunto de dados de treinamento ao mesmo tempo que tenta considerar aspectos do processamento de sinal por toda a rede neural.

- *Critério de Parada* (Haykin, 1999). O algoritmo de retroprapagação converge quando a taxa de variação absoluta no erro quadrático médio por época for suficientemente pequena. Outro critério também usado, é a norma euclidiana do gradiente do erro, sendo que, a superficie de erro é definida no espaço de pesos $\Re^P$.

Agora, considere, então, o treinamento da rede neural MLP, apresentada na figura 3.02 e que utiliza o algoritmo de retropropagação, com as seguintes características:

- Método do Gradiente (Primeira ordem);
- Modo Padrão-a-padrão;
- Inicialização dos pesos de forma aleatória, com média zero e distribuição uniforme no intervalo [-0.1, +0.1];
- Como critério de parada, interrompa as iterações assim quea taxa de variação absoluta no erro quadrático médio por época caia abaixo de um limiar pré-estabelecido;
- Funções de ativação para os neurônios da camada intermediária do tipo tangente hiperbólica e para os neurônios da camada de saída do tipo linear;
- Um determinado conjunto de dados de entrada-saída ($\mathbf{X}_l$, $\mathbf{S}_l$), com l = 1, 2, ... Np.

Para o treinamento, considere, então, o seguinte problema (Von Zuben, 1999):

- Ajustar os pesos da rede neural da figura 3.02 de forma a minimizar a seguinte função de custo:

$$J(\vec{W}) = \frac{1}{2}\sum_{i=1}^{p}(y_i^s(\vec{W}) - s_i)^2 = \frac{1}{2}\sum_{i=1}^{p}(f^s(u_i^s(\vec{W})) - s_i)^2 = \sum_{i=1}^{p} J_i(\vec{W}) \qquad (3.02)$$

Solução proposta: Neste caso, a solução pode ser obtida com o auxílio da equação 3.03, onde α é a taxa de aprendizado (assume um valor positivo), z é um instante qualquer e $\nabla J(\vec{W}(z))$ é o gradiente da função de custo.

$$\vec{W}(z+1) = \vec{W}(z) - \alpha.\nabla J(\vec{W}(z)) \qquad (3.03)$$

Para aplicar a equação 3.03 no processo de ajuste dos pesos da rede neural, basta calcular $\nabla J(\vec{W}(z))$, associado ao valor do vetor de pesos $\vec{W}$ no instante z.

Cálculo do vetor gradiente (Von Zuben, 1999):

- *Camada de saída*: w_{ij}^s (i=1, ... ,p; j=0, ... ,m).

$$\frac{\partial J}{\partial w_{ij}^s} = \frac{\partial J}{\partial u_i^s}\frac{\partial u_i^s}{\partial w_{ij}^s} = \frac{\partial J_i}{df^s}\frac{df^s}{du_i^s}\frac{\partial u_i^s}{\partial w_{ij}^s} = (f^s(u_i^s(\vec{W})) - s_i).\frac{df^s}{du_i^s}.x_{ij}^s \qquad (3.04)$$

- *Camada intermediária*: w_{jk}^e (j=1, ... ,m; k=0, ... ,n).

Primeiramente, é necessário calcular as derivadas parciais de $J_i(\vec{W})$ em relação a x_{ij}^s (i=1, ... ,p; j=0, ... ,m):

$$\frac{\partial J_i}{\partial x_{ij}^s} = \frac{\partial J_i}{\partial u_i^s}\frac{\partial u_i^s}{\partial x_{ij}^s} = \frac{\partial J_i}{df^s}\frac{df^s}{du_i^s}\frac{\partial u_i^s}{\partial x_{ij}^s} = (f^s(u_i^s(\vec{W})) - s_i).\frac{df^s}{du_i^s}.w_{ij}^s \qquad (3.05)$$

e então obtém-se:

$$\frac{\partial J}{\partial w_{jk}^e} = \frac{\partial J}{\partial u_j^e}\frac{\partial u_j^e}{\partial w_{jk}^e} = \frac{\partial J_i}{\partial u_j^e}\frac{df^e}{du_j^e}\frac{\partial u_j^e}{\partial w_{jk}^e} = \sum_{i=1}^{p}\frac{\partial J_i}{\partial x_{ij}^s}\frac{df^e}{du_j^e}x_{jk}^e \qquad (3.06)$$

finalmente, substituindo a equação 3.05 na equação 3.06, tem-se:

$$\frac{\partial J}{\partial w_{jk}^e} = [\sum_{i=1}^{p}(f^s(u_i^s(W)) - s_i).\frac{df^s}{du_i^s}.w_{ij}^s].\frac{df^e}{du_j^e}.x_{jk}^e \qquad (3.07)$$

Este método de obtenção das derivadas parciais é conhecido como algoritmo de retropropagação (*backpropagation*), porque, como pode ser verificado, a aplicação da regra da cadeia vai "propagar" cada derivada parcial da função J, definida na saída da rede neural, pelo caminho inverso, até a entrada da rede (Von Zuben, 1999). Como resultado, tem-se o vetor gradiente, necessário para a aplicação da equação 3.03.

3. 3. Classificação de Padrões

No aprendizado supervisionado, apresenta-se para a rede neural um padrão de entrada escolhido ao acaso do conjunto de treinamento, e os pesos sinápticos (parâmetros livres) da rede são modificados para minimizar a diferença entre a saída desejada e a resposta real da rede. O

treinamento da rede é repetido várias vezes para todo o conjunto de padrões de entrada-saída, até que a rede alcance um estado estável em que não haja mais modificações significativas nos pesos sinápticos. A cada repetição ou época de treinamento, a ordem de apresentação dos padrões deve ser diferente. Assim, a rede aprende os padrões ao construir um "mapeamento de entrada-saída" para o problema considerado (Haykin, 1999).

Considerando, por exemplo, uma tarefa de classificação de padrões, na qual o objetivo seja associar um sinal de entrada, representando um objeto ou evento a uma entre várias categorias (classes) preestabelecidas. O objetivo é "estimar" fronteiras de decisão para a tarefa de classificação de padrões, e fazê-lo sem invocar um modelo de distribuição probabilístico. Um ponto de vista similar está implícito no paradigma de aprendizado supervisionado, o que sugere uma analogia próxima entre o mapeamento de entrada-saída realizado por uma rede neural e a inferência estatística (Haykin, 1999).

A relação existente entre duas aplicações clássicas de redes neurais MLP, aproximação de funções contínuas e classificação de padrões, é que o problema relacionado com a primeira aplicação é considerado um mapeamento de um espaço contínuo para outro espaço contínuo, enquanto que o problema relacionado com a segunda aplicação pode ser visto como um mapeamento de um espaço contínuo para um espaço discreto. Com isso, todo problema de classificação de padrões pode ser representado na forma de uma composição de um problema de aproximação de uma função contínua com um mapeamento elementar de um espaço contínuo para um espaço discreto (Von Zuben, 1996).

Assuma que todo padrão definido no espaço $\Re^n$ pertença a uma dentre r classes. Neste caso, o problema de classificação de padrões pode ser definido na forma (Von Zuben, 1996):

- Dado o valor de y associado a um determinado padrão de entrada, defina um mapeamento elementar h que associe y à classe a qual pertence o padrão de entrada;
- Uma vez definido o mapeamento elementar h, encontre uma função g tal que a composição g o h associe o padrão à sua respectiva classe, conforme ilustrado na figura 3.04.

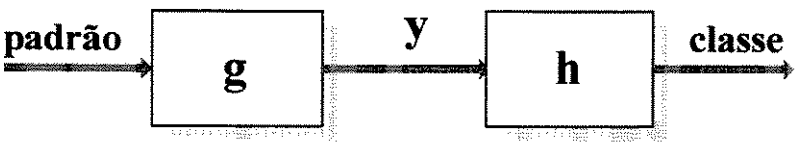


Figura 3.04 - O problema de classificação de padrões.

Por exemplo, considere o problema de classificação de padrões pertencentes ao $\Re^2$ em uma de duas classes, denominadas A e B, conforme apresentado na figura 3.05. A função f desconhecida representa a fronteira que delimita as classes, de forma que todo ponto (x_1,x_2) "abaixo" de f pertence à classe A e "acima" de f pertence à classe B. Dependendo do caso, pontos exatamente na fronteira podem ser atribuídos a nenhuma das classes, a ambas ou a apenas uma delas.

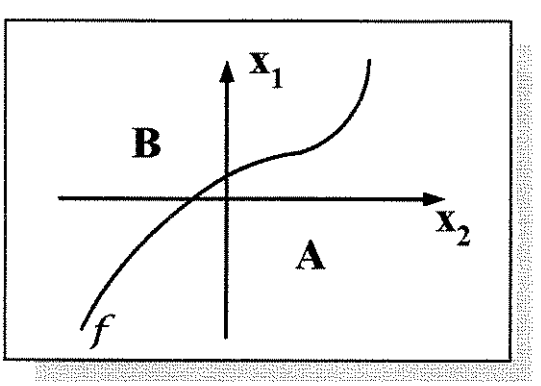


Figura 3.05 - Classificação de padrões (x_1,x_2) nas classes A e B.

Este problema de classificação de padrões pode ser resolvido conforme indicado na figura 3.06. Uma vez definido o mapeamento elementar h, resulta um problema de aproximação de uma função g tal que:

$$g(x_1,x_2)\begin{cases} <0 & \text{se} \quad x_2 < f(x_1) \\ =0 & \text{se} \quad x_2 = f(x_1) \\ >0 & \text{se} \quad x_2 > f(x_1) \end{cases} \tag{3.08}$$

Redes neurais artificiais podem, então, ser utilizadas na geração do modelo de aproximação $\hat{g}$ a partir de um conjunto de dados de aproximação.

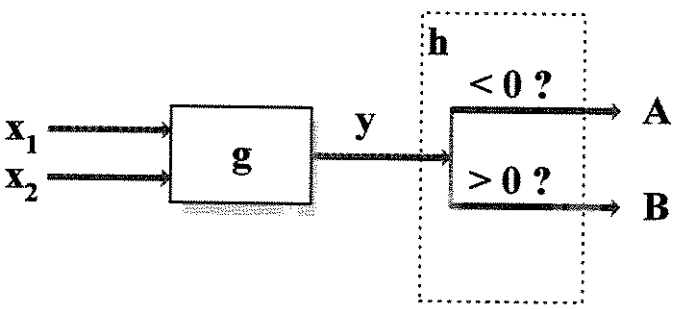


Figura 3.06 - Classificação de padrões (x_1,x_2) nas classes A e B.

Devido à estruturação da rede em camadas, o processamento dos sinais pela rede faz com que os sinais de saída sejam uma composição particular de funções de ativação, as quais devem atender a um conjunto mínimo de propriedades para conferir à rede neural o que se denomina de *capacidade de aproximação universal de mapeamentos não-lineares contínuos, definidos em regiões compactas do espaço de aproximação* (Cybenko, 1989) (Hornik, 1989).

3. 4. Conclusão

Redes neurais artificiais são ferramentas poderosas na resolução de vários tipos de problemas, em destaque, o problema da classificação de padrões. Com este propósito, as redes utilizam a sua capacidade de realizar mapeamentos não-lineares a partir dos padrões usados no seu treinamento (aprendizado supervisionado). Uma rede neural do tipo MLP, com uma única camada intermediária, é uma arquitetura suficiente e geralmente empregada para a resolução deste tipo de problema. As redes neurais MLP utilizadas no sistema proposto para a geração de trajetórias, conforme será descrito no capítulo 4, resolvem um o problema da classificação de padrões.

Capítulo 4

O Gerador de Trajetórias

4. 1. Introdução

Neste capítulo, é descrito um sistema gerador de trajetórias utilizando redes neurais artificiais, para a navegação de robôs móveis em ambientes desestruturados.

Numa primeira etapa do desenvolvimento deste sistema, foi abordado o uso de uma única rede neural do tipo Perceptron de Múltiplas Camadas (MLP), com uma arquitetura 28-20-2, para a resolução do problema. Assim sendo, foi implementado um programa chamado de Gerador de Trajetórias I, que utiliza a rede neural citada para decidir qual a direção de movimento que deverá ser adotada para que um cursor (simulação do robô móvel) possa chegar em uma dada posição alvo no ambiente de trabalho, considerado desestruturado.

Devido à necessidade de testar o sistema proposto para novas condições de trabalho, o que resultou em um aumento do custo computacional do treinamento da rede como consequência do número elevado de padrões usados para o mesmo, numa etapa posterior esta rede neural MLP foi substituída por um sistema híbrido formado por funções lógicas e por 36 redes neurais MLP, sendo esse sistema implementado no programa Gerador de Trajetórias II.

4. 2. O Gerador de Trajetórias I

O programa chamado de Gerador de Trajetórias I encontra um caminho que realiza a conexão entre uma configuração (posição e orientação) inicial e uma final, ou seja, gera os pontos intermediários entre as posições inicial e final (alvo) previamente definidas, de acordo com o ambiente onde estão inseridos estes pontos. Desta forma, a presença ou não de obstáculos, a localização e as dimensões dos mesmos são informações iniciais importantes para o programa. Isso permite que o cursor (simulação de um robô móvel real) se movimente por um ambiente desestruturado, evitando colidir com obstáculos, até que o mesmo encontre um determinado alvo.

No programa, foi utilizada uma rede neural artificial do tipo MLP (*Multilayer Perceptron*), responsável pela seleção de uma nova direção de movimento durante a geração dos pontos da trajetória (Andrade, 2001).

A linguagem utilizada para a implementação do Gerador de Trajetórias I foi a do MATLAB® versão 5.3.

O sistema proposto pode ser classificado de acordo com o tipo de abordagem adotada para a resolução do problema de navegação autônoma (veja capítulo 2). Trata-se de um navegador utilizando redes neurais. Assim sendo, pode-se identificar três componentes básicos no programa: a simulação do ambiente, a simulação do veículo dotado de sensores e a presença de uma rede neural que aprende a associar estímulos com ações através da modificação dos seus pesos sinápticos. Estes componentes serão apresentados a seguir.

4. 2. 1. A Simulação do Ambiente

A simulação do ambiente desestruturado, com um formato retangular, possui dimensões fixas iguais a 650x340 pixels, como está representado na figura 4.01. Entenda-se como desestruturada toda a área de trabalho cuja estrutura física e seus elementos agregados são

desconhecidos do sistema que gerenciará os movimentos do mecanismo físico (Mendeleck, 1995).

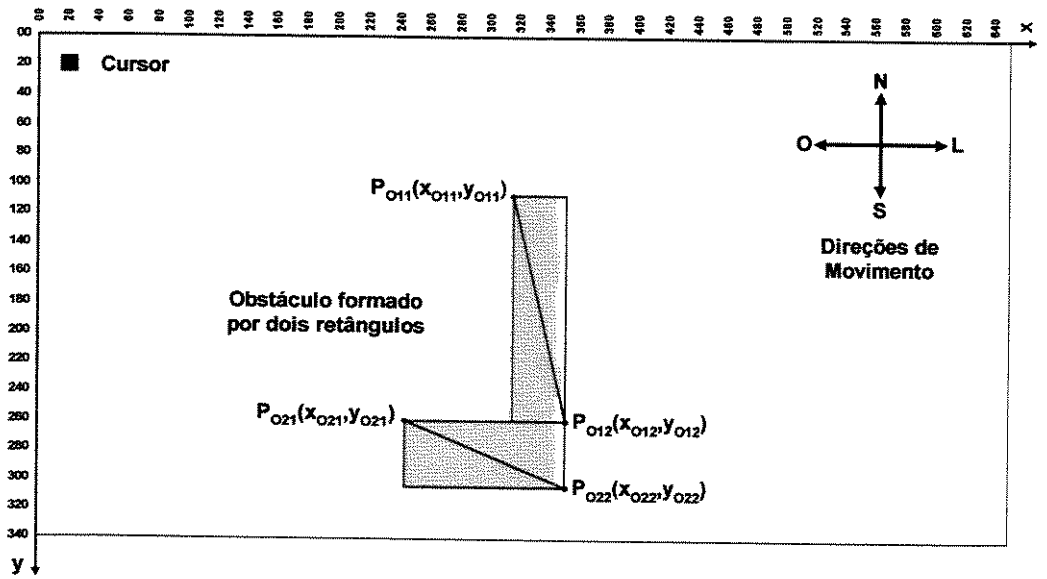


Figura 4.01 - A simulação do ambiente do Gerador de Trajetórias I.

Para implementar o ambiente no MATLAB®, foi usada uma matriz com dimensões de 341 linhas e 651 colunas, conforme as dimensões do ambiente. Esta matriz foi preenchida com os valores -2, 0, e 1, sendo o número -2 usado na representação da trajetória, o número 0 usado na representação dos obstáculos e o número 1 que foi usado na representação das áreas livres do ambiente. A escolha destes valores está relacionada com a definição das cores, que as funções do MATLAB® "*colormap (gray)*" e "*imagesc*" retornam na representação gráfica do ambiente, sendo estas cores respectivamente o preto, o cinza e o branco.

Os obstáculos, incluindo as paredes que limitam o ambiente, foram construídos através da composição de vários retângulos. Funcionando como blocos na construção dos obstáculos, esses retângulos também foram implementados como matrizes (preenchidas com zeros) que são internas à matriz que representa o ambiente. A dimensão e a localização desses retângulos foram determinadas a partir das coordenadas de apenas dois pontos, que são as extremidades da sua

diagonal principal. Na figura 4.01 estão representados um obstáculo, formado por dois retângulos, e também as coordenadas dos pontos necessários para a construção dos mesmos.

4. 2. 2. A Simulação do Veículo

A simulação do veículo foi feita a partir de um elemento fictício, que foi chamado de "cursor", representado nas figuras 4.02 e 4.03. O cursor, dotado de sensores, possui a capacidade de deslocar-se pela área de trabalho com um avanço fixo. Estando o mesmo habilitado a movimentar-se nas quatro direções identificadas pelos pontos cardeais: Norte, Sul, Leste e Oeste.

Ele tem um formato quadrangular e suas dimensões são fixas e reduzidas: iguais a 10x10 pixels. Essas dimensões podem ser consideradas como desprezíveis em relação às dimensões dos demais obstáculos da área de trabalho. Desse modo, foi suficiente a definição de 12 sensores de distância, agrupados três a três nas quatro direções de movimento, conforme a figura 4.02. A combinação do valor dos sensores fornece subsídios para a rede neural que seleciona a direção de movimento.

Para a localização do cursor no ambiente foi adotado um sistema de referências cartesiano ortogonal no plano (x-y), com origem conforme representado na figura 4.01.

Simulando a limitação dos sensores em medir distâncias muito longas, foi estabelecido um limite para a distância medida. Dessa forma o sensor fornece a distância até o próximo obstáculo dentro de um certo limite (*range*).

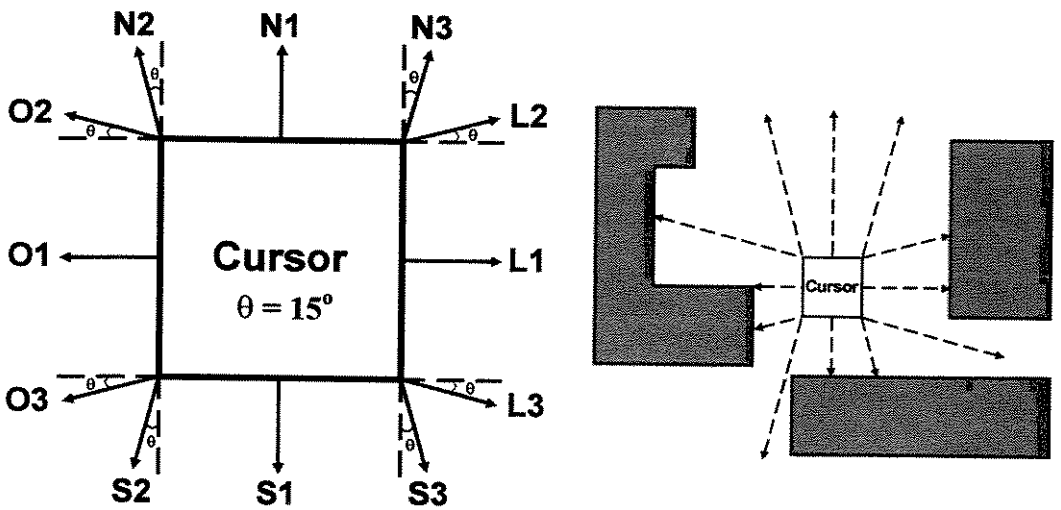


Figura 4.02 - O cursor e a leitura dos sensores.

Como já foi descrito, a simulação do ambiente é definida num plano x-y com origem e direções mostradas na figura 4.01. Um ponto (x, y) nesse plano pertence ao ambiente se $0 < x < 650$ e $0 < y < 340$, com x e y inteiros. A distância entre um ponto (x, y) e (x+1, y), bem como a distância entre (x, y) e (x, y+1) corresponde a 1 (uma) unidade de medida, que no caso é o *pixel*. Tal unidade de medida pode ser considerada com outra escala, segundo a necessidade do usuário.

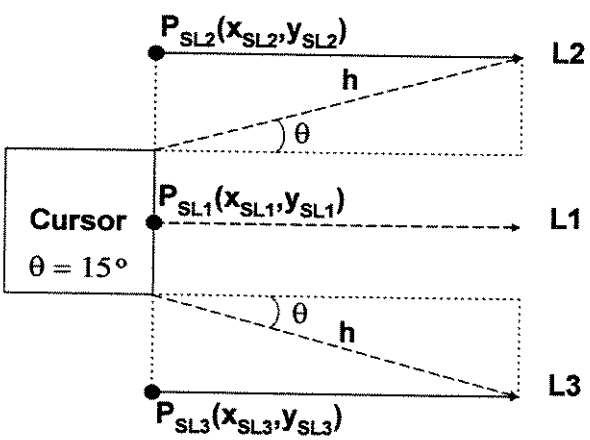


Figura 4.03 - Coordenadas dos sensores.

Para o cursor em uma posição qualquer no ambiente, cada um dos sensores tem uma coordenada (x, y) própria e uma linha de ação, ou seja, uma direção na qual o sensor irá medir a distância até o próximo obstáculo, como está detalhado na figura 4.03.

No caso do sensor L1 na direção Leste, representado no exemplo da figura 4.03, sua linha de ação é definida pelo segmento de reta mostrado na figura e cujos pontos são dados por (x, y_i) onde y_i é a coordenada y do sensor (constante) e $x_i < x < x_i + range$.

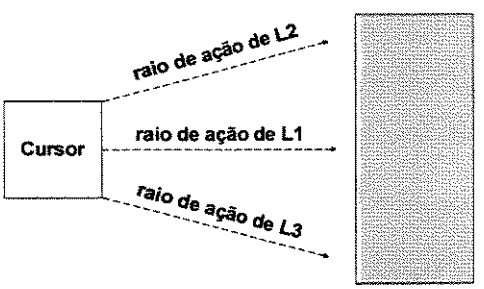


Figura 4.04 - O raio de ação dos sensores.

Conforme a figura 4.03, os sensores que formam com a horizontal um ângulo de 15°, neste caso os sensores L2 e L3, medem da mesma forma que o sensor L1 que está na horizontal, sendo que as suas linhas de ação serão transladadas na vertical (em y) de + h.senθ para o caso de L2 e de - h.senθ para L3. O valor de h (hipotenusa) vai ser escolhido de acordo com o *range* dos sensores, ou seja, como o *range* de L1 é igual a h.cosθ, define-se um valor para o *range* e calcula-se o valor de h. Os valores de h, + h.senθ e - h.senθ são aproximados para inteiros pois a unidade de medida de distância adotado no ambiente é o *pixel*.

Todos os demais sensores possuem igualmente uma linha de ação que caracteriza-se por ter uma das coordenadas constantes e a outra variando. Para os sensores das direções Leste e Oeste, as linhas de ação têm a coordenada y constante e igual à respectiva coordenada do sensor e nas direções Norte e Sul, as linhas de ação caracterizam-se pela coordenada x constante e igual à respectiva coordenada do sensor. Em todos os casos, tendo uma coordenada constante e variando-

se a outra coordenada dentro de um intervalo pré definido (*range*), obtém-se os pontos do segmento de reta que representa o raio de ação dos sensores, por exemplo, como está representado na figura 4.04.

Assim, a partir da posição inicial de um sensor e definida a sua direção de atuação (sabemos que o conjunto de pontos obtidos a partir do par ordenado formado pela coordenada constante e pela coordenada variável ou direção de atuação), pode-se conhecer todos os pontos do raio de ação de um sensor.

Finalmente, para medir a distância a partir de um sensor, é analisado se as coordenadas de cada ponto dentro do raio de ação do sensor pertencem a um obstáculo, podendo assim calcular a distância até o próximo obstáculo (dentro do raio de ação).

Por exemplo, na figura 4.04, é mostrado o sensor L1 e sua linha de ação (direção x). Assim, se fixarmos a coordenada y do sensor e verificarmos todos os pontos (x_i, y), onde $x_i = x + i$ e $0 < i < range$, podemos saber qual a distância entre o Cursor e o obstáculo mais próximo em relação ao sensor L1.

Os três valores das distâncias podem ser analisados individualmente ou segundo a menor distância retornada. Por exemplo, ao medir-se a distância na direção Leste, antes de executar um movimento nesta direção, é importante saber qual o valor mínimo entre as distâncias retornadas pelos sensores. A Figura 4.05 ilustra uma situação em que são necessárias as considerações feitas.

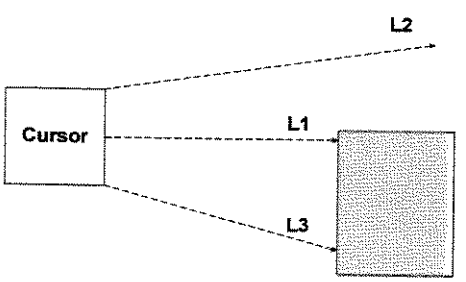


Figura 4.05 - Leitura diferente para os sensores na mesma direção.

No caso acima, o valor da distância medida por L2 pode ser seguro para o avanço do Cursor na direção Leste, porém as distâncias medidas por L1 e L3 não são seguras. Assim, para checar a possibilidade de movimento em uma determinada direção é necessário avaliar o valor mínimo entre as distâncias medidas pelos três sensores naquela direção. Com isso, foi criado um vetor **NSLO**, com dimensão 1x4, formado pela menor distância detectada pelos sensores para cada direção. Um exemplo para este vetor, quando o *range* dos sensores é 20, está representado na figura 4.06.

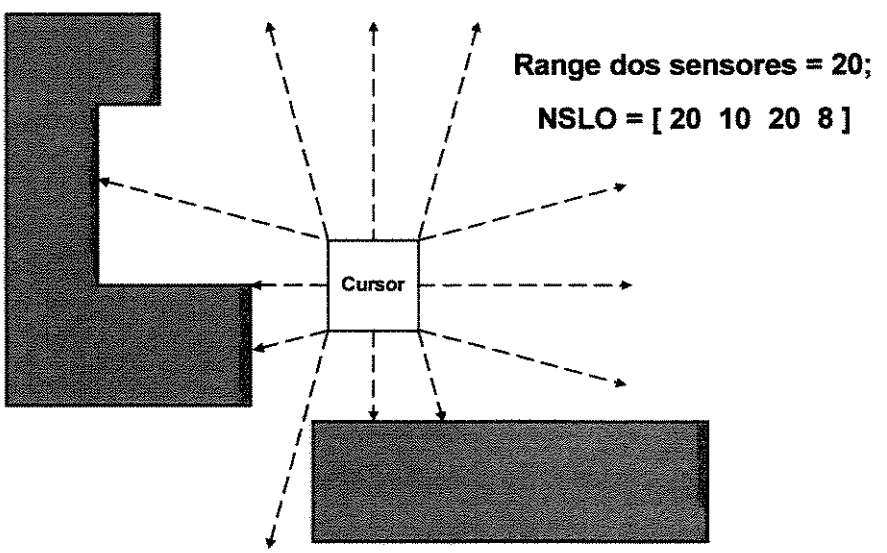


Figura 4.06 - O vetor representando o valor mínimo da leitura dos sensores em cada direção.

As informações do vetor **NSLO** são codificadas para valores binários formando um novo vetor chamado de **Se**, com a mesma dimensão do vetor **NSLO**. Para isso, foi adotado o seguinte critério: primeiro calcula-se o termo de valor máximo no vetor **NSLO**, no caso do exemplo da figura 4.06 é o primeiro termo, que é igual ao terceiro, ou seja, 20 (*range* dos sensores). Em seguida o vetor **Se** vai ter preenchido com o valor 0 todas as suas posições relativas ao valor máximo de **NSLO** e as outras posições vão receber o valor 1, assim sendo, o vetor **Se** seria [0 1 0 1], para o exemplo citado. Este vetor **Se** vai alimentar a rede neural que faz parte do sistema, informando que, numa dada iteração as direções Norte e Leste estão livres para a movimentação

do cursor e as direções Sul e Oeste estão ocupadas, ou seja, foi detectada a presença de obstáculos nas mesmas.

4. 2.3. A Rede Neural utilizada no Gerador de Trajetórias I

Antes de começar uma descrição sobre a rede neural que foi utilizada, é necessária uma explicação do funcionamento do programa, no qual a rede está inserida, a partir das suas entradas. O programa recebe do usuário, como está representado na figura 4.07, as seguintes entradas:

- As coordenadas do ponto que corresponde à posição inicial do cursor chamado de Pi (x_i , y_i) e;

- As coordenadas do ponto que corresponde à posição final (alvo) da trajetória do cursor chamado de Pf (x_f , y_f).

A partir das coordenadas destes dois pontos são calculadas as chamadas "direções preferenciais". A direção preferencial é uma direção que o cursor deve adotar para ir de um determinado ponto no ambiente para a posição final da trajetória, ou seja, o programa indica quais as direções que o cursor deve seguir para atingir o seu alvo.

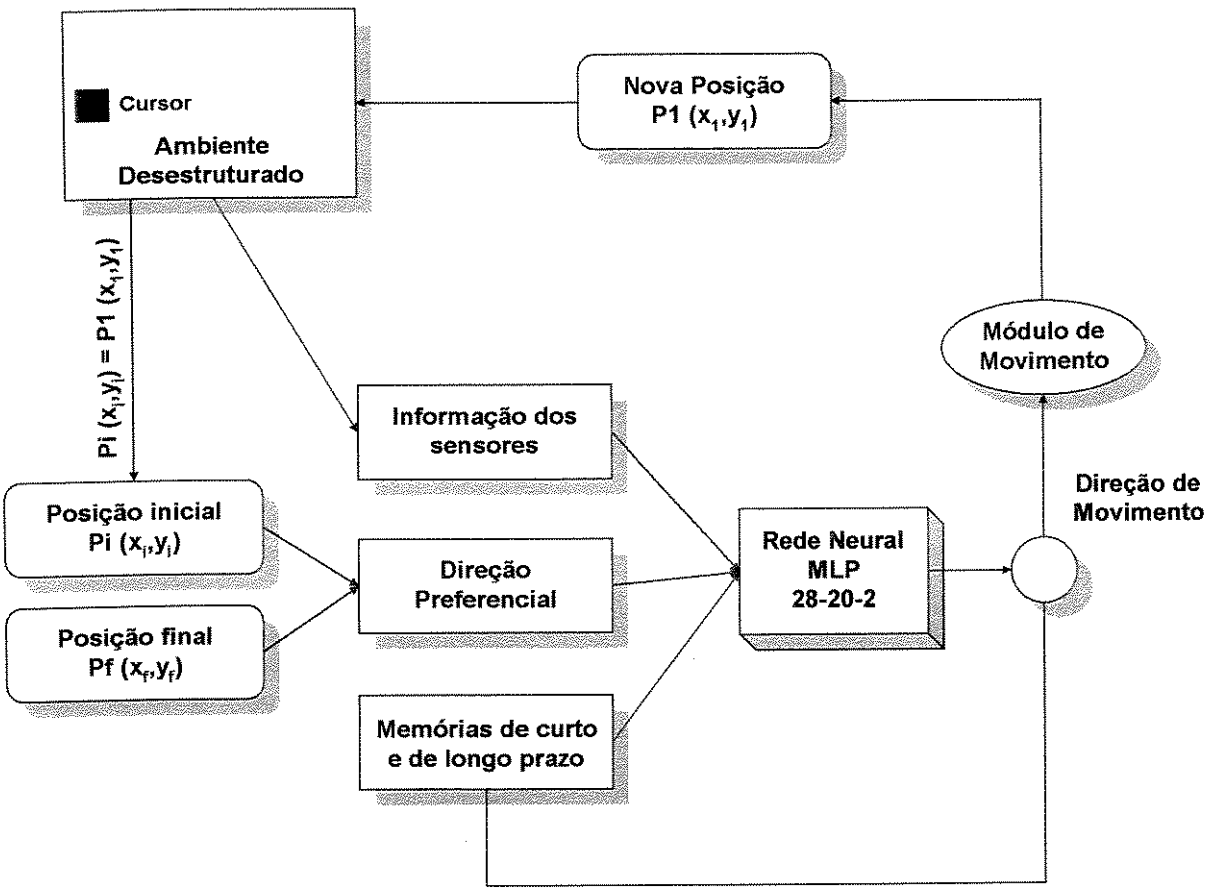


Figura 4.07 - O esquema do Gerador de Trajetórias I.

A direção preferencial é obtida após o cálculo da diferença entre as abcissas dos pontos inicial e final, denominada de dif_x, e da diferença entre as ordenadas destes pontos, chamada de dif_y. Com isso, pode-se determinar qual direção vai ser considerada como preferencial segundo os possíveis valores de dif_x e dif_y descritos na tabela 4.01. Nesta tabela foram também listados os vetores que indicam qual direção é considerada preferencial: esses vetores possui em dimensão 1x4, sendo que o seu primeiro termo corresponde à direção Norte, o segundo à direção Sul, o terceiro à direção Leste e o quarto à direção Oeste. Eles são formados por valores binários, onde o 0 indica que a direção relacionada com a sua posição no vetor é considerada preferencial e o 1 para o caso contrário.

Valores de dif_y e dif_x:	Direções Preferenciais:	Vetor DP (Direção Preferencial):
$dif_y < 0$ e $dif_x > 0$	Norte e Leste	[0 1 0 1]
$dif_y < 0$ e $dif_x < 0$	Norte e Oeste	[0 1 1 0]
$dif_y > 0$ e $dif_x > 0$	Sul e Leste	[1 0 0 1]
$dif_y > 0$ e $dif_x < 0$	Sul e Oeste	[1 0 1 0]
$dif_y < 0$ e $dif_x = 0$	Norte	[0 1 1 1]
$dif_y > 0$ e $dif_x = 0$	Sul	[1 0 1 1]
$dif_y = 0$ e $dif_x > 0$	Leste	[1 1 0 1]
$dif_y = 0$ e $dif_x < 0$	Oeste	[1 1 1 0]

Tabela 4.01 - As Possíveis Direções Preferenciais.

Também está representado na figura 4.07 o chamado módulo de movimento, que recebe da rede neural a direção de movimento a ser seguida e faz com que o cursor se desloque com um avanço fixo, no caso foi adotado o valor de 11 unidades (*pixels*) para a nova posição $P1(x_1,y_1)$. Em seguida, esse novo ponto passa a ser considerado como o ponto inicial $Pi(x_i,y_i)$ e assim sucessivamente, até que um desses pontos da trajetória se aproxime do ponto final $Pf(x_f,y_f)$. O critério de parada para a busca do ponto final será explicado no capítulo 5.

Foi adotada, no Gerador de Trajetórias I, uma Rede neural do tipo MLP responsável pela seleção de uma nova direção de movimento durante a geração dos pontos da trajetória. A arquitetura desta rede neural, representada na figura 4.08, é do tipo 28-20-2, ou seja, com 28 entradas, uma camada intermediária com 20 neurônios e 2 saídas.

Pode-se também notar na figura 4.08 a presença de recorrência externa na rede neural, sendo que o mapeamento realizado por esta rede neural é estático.

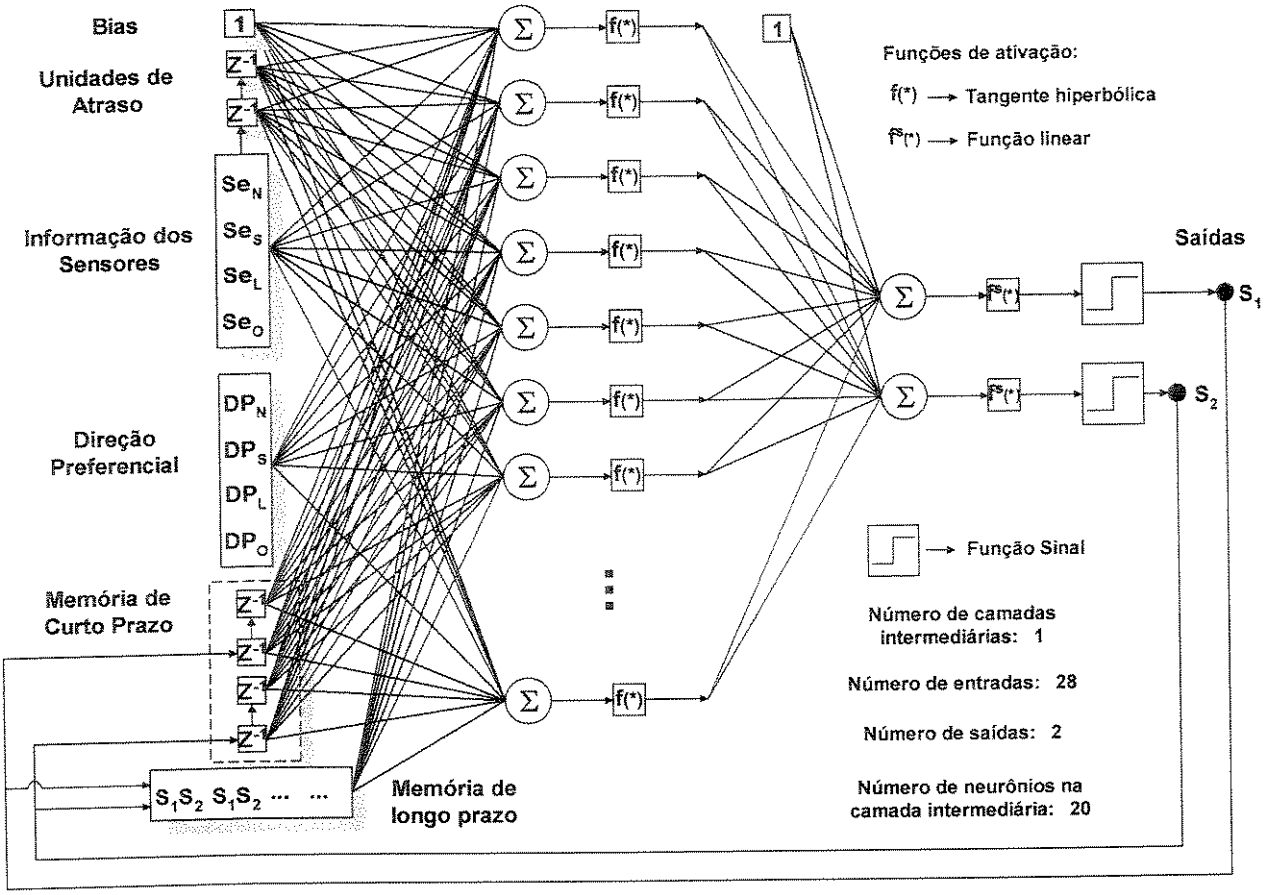


Figura 4.08 - A arquitetura da rede neural MLP utilizada.

As 28 entradas da rede neural, com valores iguais a 0, 1 e 2, formam um vetor com dimensão 1x28, seguindo a sequência que está representada na tabela 4.02. O número 2 foi utilizado para representar os espaços vazios nas memórias de curto e longo prazo.

Entrada:	Ordem da Entrada:	Variável da Entrada:
Informação dos Sensores	1	$Se_N(k)$
	2	$Se_S(k)$
	3	$Se_L(k)$
	4	$Se_O(k)$
	5	$Se_N(k-1)$
	6	$Se_S(k-1)$
	7	$Se_L(k-1)$
	8	$Se_O(k-1)$
	9	$Se_N(k-2)$
	10	$Se_S(k-2)$
	11	$Se_L(k-2)$
	12	$Se_O(k-2)$
Direção Preferencial	13	$DP_N(k)$
	14	$DP_S(k)$
	15	$DP_L(k)$
	16	$DP_O(k)$
Memória de Curto Prazo	17	S1(k-1)
	18	S2(k-1)
	19	S1(k-2)
	20	S2(k-2)
Memória de Longo Prazo	21	S1(k-o)
	22	S2(k-o)
	23	S1(k-q)
	24	S2(k-q)
	25	S1(k-r)
	26	S2(k-r)
	27	S1(k-s)
	28	S2(k-s)

Tabela 4.02 - As entradas da rede neural.

As duas saídas da rede neural S1 e S2 determinam qual a direção que o cursor deve adotar de acordo com os padrões de entrada recebidos pela mesma. Cada uma dessas informações de saída foi codificada para um valor binário, através de uma função do tipo sinal representada na figura 4.08, sendo adotado o valor 0 quando o valor de saída da rede neural era menor do que 0.5, e 1 caso contrário. Na tabela 4.03 estão representadas as saídas codificadas e as suas respectivas direções para movimentação do cursor.

Direção:	Saída:	
	S1	S2
Norte	1	1
Sul	0	0
Leste	0	1
Oeste	1	0

Tabela 4.03 - As saídas da rede neural e as suas direções correspondentes.

Uma descrição de cada uma das entradas da rede neural pode ser apresentada da seguinte forma:

Informação dos sensores. Foi simulado o uso de sensores de distância, cuja função é fornecer a distância do cursor até a próxima colisão (obstáculo). As variáveis $Se_N(k)$, $Se_S(k)$, $Se_L(k)$ e $Se_O(k)$ correspondem à informação dos sensores nas direções possíveis de movimento respectivamente Norte, Sul, Leste e Oeste. Estas variáveis são os termos do vetor **Se**, os quais. informam à rede quais direções estão livres para a movimentação do cursor, numa dada iteração "k". Por exemplo, se $Se_N(k)$ for igual a zero significa que na iteração k a direção norte está livre. Quando ocorre a detecção de um obstáculo numa dada direção o valor informado à rede é 1.

Devido à importância deste tipo de informação para a rede neural, ou seja, o uso de informações sensoriais, já destacado no capítulo 2, foram inseridos também como entrada, dois atrasos do vetor **Se**. Assim sendo, foi considerada também a informação dos sensores nas iterações "k-1" e "k-2". Com isso, as informações do sensores totalizam 12 entradas.

Direção preferencial. Outra informação importante para a rede, como já foi descrito anteriormente, é a direção preferencial que o cursor deve adotar para ir de um determinado ponto no ambiente para a posição alvo, numa dada iteração. Através do vetor **DP**, a rede neural é informada sobre quais as direções que são consideradas preferenciais, sendo as variáveis $DP_N(k)$, $DP_S(k)$, $DP_L(k)$ e $DP_O(k)$ os componentes deste vetor para uma dada iteração k. O número de entradas relacionadas com a direção preferencial são 4.

Memória de curto prazo. No caso deste tipo de entrada da rede, são considerados dois atrasos das saídas (binárias) da rede neural, informando para a mesma as duas últimas direções adotadas pelo cursor. Cada saída da rede vai realimentar, servindo como uma nova entrada, a rede na próxima iteração. A memória de curto prazo totaliza 4 entradas para a rede neural, sendo S1(k-1) e S2(k-1) as saídas da rede na iteração k-1, e S1(k-2) e S2(k-2) as saídas para a iteração k-2.

Memória de longo prazo. A chamada memória de longo prazo guarda e informa para a rede quais foram as quatro últimas mudanças de direções adotadas pelo cursor. Neste caso não ocorre a repetição de uma direção, e pode-se afirmar que a memória de longo prazo funciona como um *buffer* das direções, ativado no momento que ocorre a mudança para uma nova direção. A importância desta entrada da rede foi verificada principalmente para os casos de ambientes com obstáculos que exigem várias repetições de sequências de direções, como por exemplo, obstáculos formando um zigue-zague que serão mostrados no capítulo 5.

A memória de longo prazo totaliza 8 entradas para a rede neural, sendo que k>o>q>r>s: são índices que variam ao longo das iterações, pois dependem da frequência de mudanças de direção. S1(k-o) e S2(k-o) são as saídas da rede na iteração k-o, S1(k-q) e S2(k-q) as saídas da rede na iteração k-q, S1(k-r) e S2(k-r) as saídas da rede na iteração k-r e S1(k-s) e S2(k-s) as saídas da rede na iteração k-s. Toda vez que ocorre uma mudança de direção, esta nova direção passa a ocupar as duas primeiras posições do vetor memória de longo prazo na próxima iteração, deslocando a antiga primeira direção para a segunda posição no vetor, esta por sua vez desloca a

antiga segunda direção para a terceira posição no vetor, e assim sucessivamente, até que a antiga quarta direção seja desconsiderada.

Em síntese, a rede neural decide qual a direção a ser seguida considerando a disposição dos obstáculos à sua volta (*Informações dos Sensores*), o objetivo a ser alcançado (*Direção Preferencial*) e qual o caminho seguido até o momento (*Memórias de longo e curto prazo*) (Benzatti, 2001).

Para o treinamento da rede neural foi utilizado um conjunto de treinamento com 258 padrões compostos por estímulos, que são as entradas da rede neural, e ações, que são as saídas desejadas. Foram definidos padrões de comportamentos esperados durante a navegação nos seguintes ambientes: ambiente sem obstáculos, ambiente com obstáculos simples, ambiente com obstáculos formando um zigue-zague e ambiente com um obstáculo côncavo. Os padrões foram obtidos a partir de uma série de regras pré-definidas por um instrutor (aprendizado supervisionado).

Exemplos destas regras foram representados na figura 4.09. Nesta figura, pode-se notar que as direções preferenciais adotadas para a composição das regras, e consequentemente dos padrões usados no treinamento, foram o Sul e o Leste. Explicando o primeiro exemplo da figura 4.09, para a mudança da direção de movimento, tem-se que: quando a direção de movimento era o Sul e os sensores indicavam que as direções Norte, Leste e Oeste estão livres para a movimentação, a nova direção (indicada pelo instrutor) deverá ser o Leste, que é uma direção preferencial. Nessa fase da construção dos padrões, não foram consideradas as informações atrasadas dos sensores e as informações das memórias de curto e longo prazo. Estas informações só foram consideradas quando as regras foram aplicadas nos ambientes citados anteriormente.

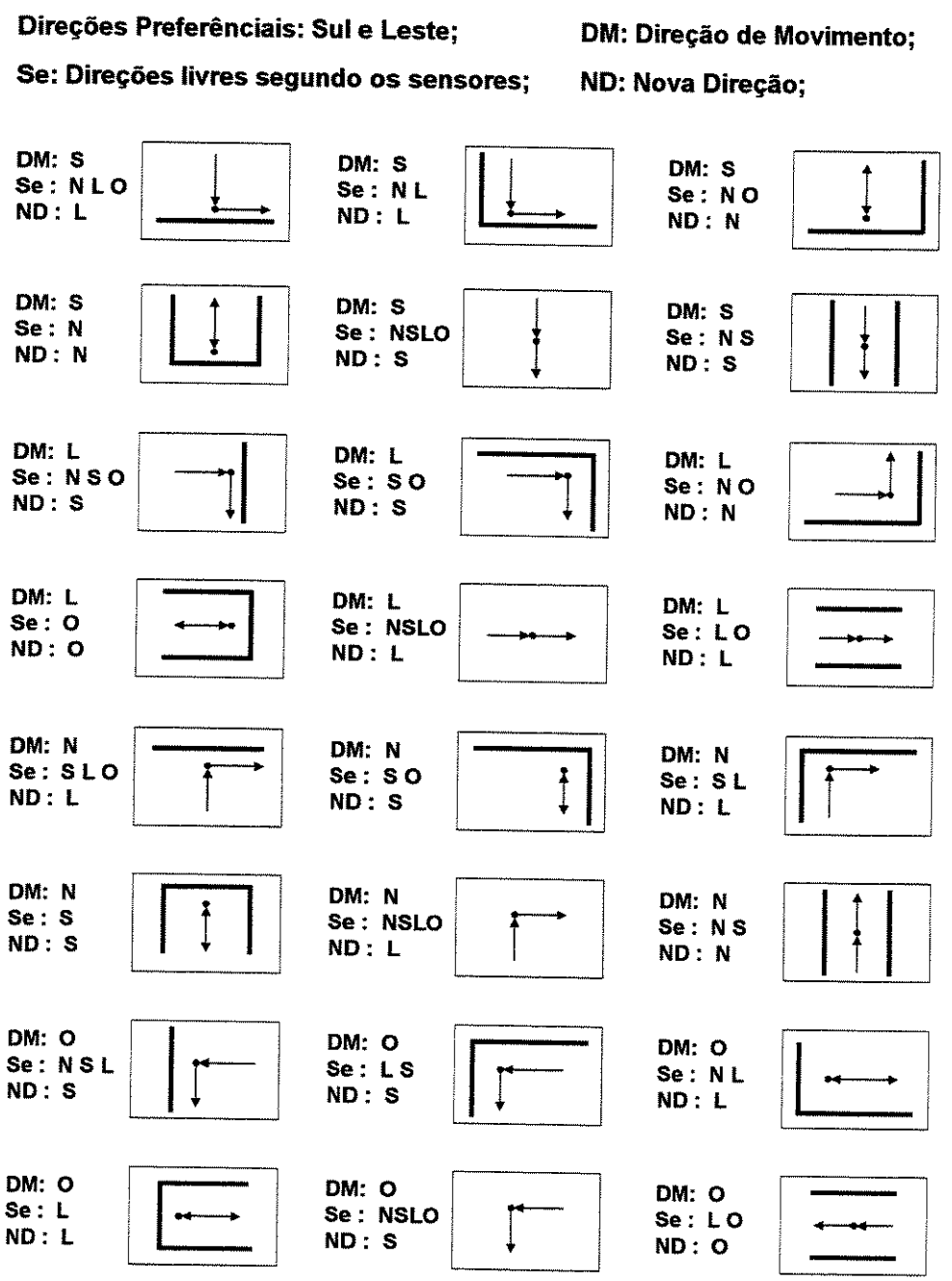


Figura 4.09 - Regras para a seleção da nova direção de movimento.

Foi utilizada uma planilha do Microsoft Excel® para a criação e arquivamento dos padrões de treinamento. Os padrões eram criados com variáveis simbólicas N, S, L e O, representando Norte, Sul, Leste e Oeste, respectivamente, as quais eram convertidas, através de funções lógicas, para os valores numéricos 0, 1 e 2. Como já foi citado, o número 2 foi utilizado para representar os espaços vazios nas memórias de curto e longo prazo, estes espaços ocorriam durante as primeiras mudanças de direções.

As tabelas 4.04 e 4.05 foram criadas a partir da planilha do Microsoft Excel®, nas quais estão listados 20 exemplos de padrões usados no treinamento, sendo que na tabela 4.04 eles foram representados pelas variáveis simbólicas e na tabela 4.05 pelos valores numéricos.

Se(k)				**Se(k-1)**				**Se(k-2)**				**DP**		**MCP(k-1)**				**MCP(k-2)**				**MLP**				**S1S2(k)**			
	S	L										S	L													S		
N	S	L			S	L						S	L		S							S				S		
N	S	L		N	S	L			S	L		S	L		S				S			S				S		
N	S	L	O	N	S	L		N	S	L		S	L		S				S			S				S		
N		L	O	N	S	L	O	N	S	L	O	S	L		S				S			S					L	
N		L	O	N		L	O	N	S	L	O	S	L			L			S			L	S				L	
N		L	O	N		L	O	N		L		S	L			L				L		L	S				L	
N		L	O	N		L	O	N		L	O	S	L			L				L		L	S				L	
N	S	L	O	N		L	O	N		L	O	S	L			L				L		L	S				L	
N	S		O	N	S	L	O	N	S	L	O	S	L			L				L		L	S				S	
N	S		O	N	S		O	N	S	L	O	S	L		S					L		S	L	S			S	
N	S		O	N	S		O	N	S		O	S	L		S				S			S	L	S			S	
N	S	L	O	N	S		O	N	S		O	S	L		S				S			S	L	S			S	
N	S		O	N	S	L	O	N	S		O	S	L		S				S			S	L	S			S	
N	S		O	N	S		O	N	S	L	O	S	L		S				S			S	L	S			S	
N	S		O	N			O	N	S		O	S	L	N					S			N	S	L	S	N		
N	S		O	N	S		O	N			O	S	L	N				N				N	S	L	S	N		
N	S		O	N	S		O	N	S		O	S	L	N				N				N	S	L	S	N		
N	S	L	O	N	S		O	N	S		O	S	L	N				N				N	S	L	S			L	
N	S	L	O	N	S	L	O	N	S		O	S	L			L		N				L	N	S	L			L	

Tabela 4.04 - 20 exemplos de padrões representados pelas variáveis simbólicas.

Se(k)				Se(k-1)				Se(k-2)				DP				MCP				MLP								S1S2	
1	0	0	1	1	1	1	1	1	1	1	1	1	0	0	1	2	2	2	2	2	2	2	2	2	2	2	2	0	0
0	0	0	1	1	0	0	1	1	1	1	1	1	0	0	1	0	0	2	2	0	0	2	2	2	2	2	2	0	0
0	0	0	1	0	0	0	1	1	0	0	1	1	0	0	1	0	0	0	0	0	0	2	2	2	2	2	2	0	0
0	0	0	0	0	0	0	1	0	0	0	1	1	0	0	1	0	0	0	0	0	0	2	2	2	2	2	2	0	0
0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	2	2	2	2	2	2	0	1
0	1	0	0	0	1	0	0	0	0	0	0	1	0	0	1	0	1	0	0	0	1	0	0	2	2	2	2	0	1
0	1	0	0	0	1	0	0	0	1	0	1	1	0	0	1	0	1	0	1	0	1	0	0	2	2	2	2	0	1
0	1	0	0	0	1	0	0	0	1	0	0	1	0	0	1	0	1	0	1	0	1	0	0	2	2	2	2	0	1
0	0	0	0	0	1	0	0	0	1	0	0	1	0	0	1	0	1	0	1	0	1	0	0	2	2	2	2	0	1
0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	0	1	0	1	0	0	2	2	2	2	0	0
0	0	1	0	0	0	1	0	0	0	0	0	1	0	0	1	0	0	0	1	0	0	0	1	0	0	2	2	0	0
0	0	1	0	0	0	1	0	0	0	1	0	1	0	0	1	0	0	0	0	0	0	0	1	0	0	2	2	0	0
0	0	0	0	0	0	1	0	0	0	1	0	1	0	0	1	0	0	0	0	0	0	0	1	0	0	2	2	0	0
0	0	1	0	0	0	0	0	0	0	1	0	1	0	0	1	0	0	0	0	0	0	0	1	0	0	2	2	0	0
0	0	1	0	0	0	1	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	1	0	0	2	2	0	0
0	0	1	0	0	1	1	0	0	0	1	0	1	0	0	1	1	1	0	0	1	1	0	0	0	1	0	0	1	1
0	0	1	0	0	0	1	0	0	1	1	0	1	0	0	1	1	1	1	1	1	1	0	0	0	1	0	0	1	1
0	0	1	0	0	0	1	0	0	0	1	0	1	0	0	1	1	1	1	1	1	1	0	0	0	1	0	0	1	1
0	0	0	0	0	0	1	0	0	0	1	0	1	0	0	1	1	1	1	1	1	1	0	0	0	1	0	0	0	1
0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	1	0	1	1	1	0	1	1	1	0	0	0	1	0	1

Tabela 4.05 - 20 exemplos de padrões representados pelos valores númericos.

O tipo de treinamento usado foi o método do gradiente (método de primeira ordem) usando o *backpropagation* para o cálculo do vetor gradiente, utilizado para o ajuste dos pesos da rede neural. Este ajuste foi executado de acordo com o método padrão-a-padrão ou instantâneo (*pattern mode*).

A linguagem usada para a implementação do algoritmo de treinamento da rede neural (Von Zuben, 1999) foi a do software MATLAB® versão 5.3. Os padrões de treinamento eram copiados da planilha do Microsoft Excel® e colados num arquivo do MATLAB®, que era utilizado pelo programa de treinamento. O gráfico do treinamento está representado na figura 4.10.

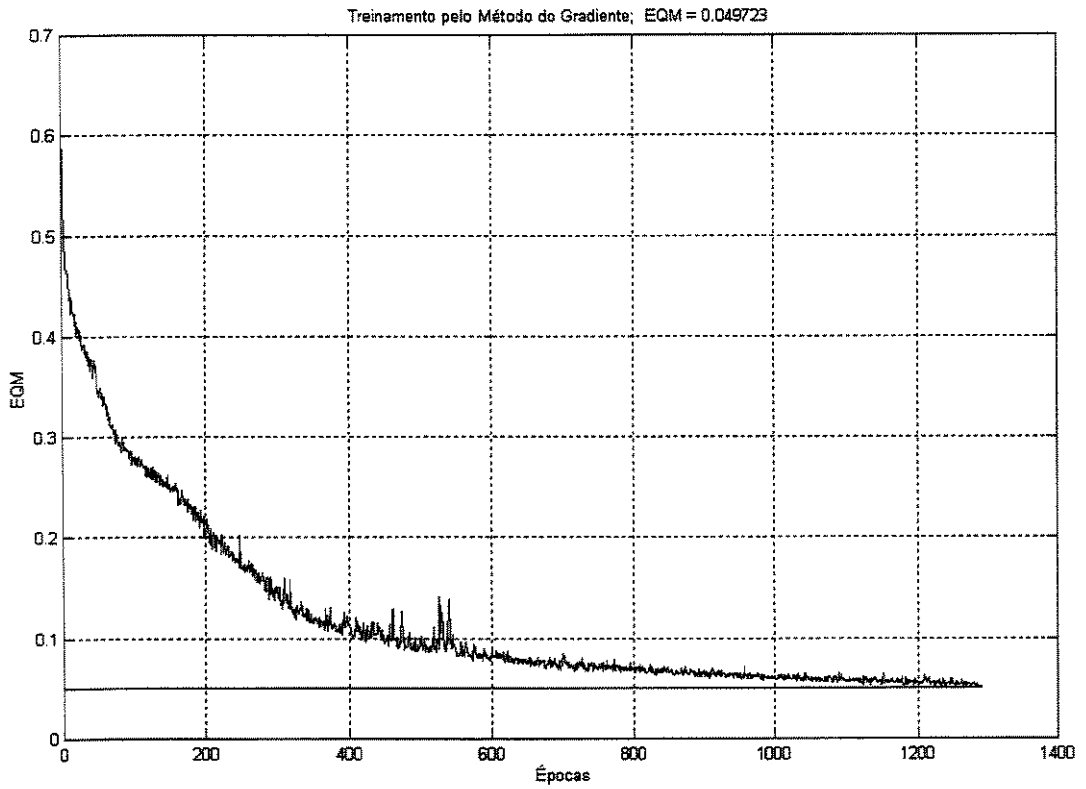


Figura 4.10 - O gráfico do treinamento dos 258 padrões da rede neural.

Foi utilizada uma taxa de aprendizado fixa com valor igual a 0.1, sendo adotada a taxa de variação absoluta no erro quadrático médio por época para o critério de convergência com valor igual a 0.05.

A inicialização dos pesos da rede foi feita de maneira aleatória, com média zero e distribuição uniforme no intervalo [-0.1, +0.1].

Como o número de padrões adotados no treinamento não era muito grande, o método do gradiente com taxa de aprendizado fixa foi suficiente para que ocorresse a convergência com erro de treinamento baixo, pois se trata de um algoritmo de custo computacional relativamente baixo

por época de treinamento. Em contra partida, o tempo de treinamento foi elevado, aproximadamente de 4 horas para 1293 épocas (Pentium III, 550 MHz).

Após o treinamento, o vetor resultante dos pesos foi salvo como um arquivo do MATLAB®, para que o mesmo possa ser usado pela rede neural do Gerador de Trajetórias I.

A seguir, será descrito o desenvolvimento e a implementação de um segundo programa chamado de Gerador de Trajetórias II.

4. 3. O Gerador de Trajetórias II

Uma segunda versão do sistema utilizado para gerar trajetórias, chamada de Gerador de Trajetórias II, foi desenvolvido e implementado devido à necessidade de aplicar o sistema gerador de trajetórias para outras direções preferências e também para outras configurações de ambientes.

A consequência desta necessidade foi o grande aumento do número de padrões de treinamento que seriam usados pela rede neural. É de conhecimento que, à medida que o número de padrões de treinamento cresce, eleva-se também o custo computacional do treinamento, geralmente causando uma diminuição na velocidade de aprendizado e também influindo na qualidade da solução obtida após a convergência.

Uma classificação em relação ao Tamanho do Conjunto de Dados (TCD), ou seja, a quantidade de exemplos disponíveis para o treinamento, foi descrita por Castro (Castro, 1998a) como:

- *Pequeno*: menos de 210 amostras;
- *Médio*: entre 210 e 3170 amostras e
- *Grande*: mais que 3170.

Para o sistema gerador de trajetórias, o grande problema encontrado, à medida que o número de padrões de treinamento passou a ser considerado grande, foi a elevação do tempo de treinamento. A justificativa para este aumento de tempo, deve-se ao fato de que a inclusão de um novo padrão ou grupo de padrões implicaria em um novo treinamento da rede, que posteriormente deveria ser testado, onde uma resposta satisfatória do sistema validaria a inclusão destes novos padrões. Assim sendo, era impraticável ter-se um tempo de treinamento elevado, frente à constante necessidade da inclusão de novos padrões, para que o sistema fosse considerado satisfatório para novas direções preferenciais ou novas configurações de ambientes.

Algumas técnicas podem ser usadas para tentar minimizar este problema, como por exemplo:

- O uso de outras formas de implementação dos algoritmos de treinamento. Formas otimizadas de implementação, nas quais, quando possível, estratégias de grande custo computacional possam ser eliminadas, como por exemplo, a redução no uso excessivo de *loops* nos programas. E também a utilização de outros tipos de linguagens, como o C++;
- O uso de técnicas mais eficientes para a inicialização dos pesos da rede neural (Castro, 1998c);
- Uma escolha inicial adequada da taxa de aprendizado (taxa fixa) ou o ajuste automático da mesma durante o treinamento (taxa variável) (Castro, 1998a);
- A adição de um termo de momento (Castro, 1998a);
- O uso de métodos de treinamento de segunda ordem (Castro, 1998a);

- O uso de métodos construtivos para determinar automaticamente uma arquitetura ótima da rede neural (Castro, 1998a);
- A partição do conjunto de padrões de treinamento (McLonne, 1997);

Como solução, foi empregada a técnica de dividir o conjunto de padrões de treinamento em vários subconjuntos. Desta forma, a única rede neural MLP usada no primeiro gerador de trajetórias foi substituída por 36 redes neurais MLP, sendo que o processamento de todas estas redes ocorre em paralelo. Pode-se citar algumas vantagens para este tipo de abordagem (McLoone, 1997):

- Os algoritmos paralelos são relativamente fáceis de serem criados, a partir dos subconjuntos formados na partição dos dados de treinamento, que podem ser considerados como novos conjuntos de treinamento;

- Cada processador paralelo trabalha com apenas uma parte do conjunto de treinamento, portanto um grande número de problemas podem ser atendidos, o que não ocorreria se o conjunto completo de treinamento fosse requerido para cada processador;

- Técnicas de decomposição de dados de treinamento, onde um grande problema de treinamento é subdividido em problemas mais simples, são facilmente implementadas.

Este método reproduz uma forma geralmente empregada para a resolução de grandes problemas, ou seja, a divisão destes em vários subproblemas menores. A resolução de cada um deste subproblemas, possui um nível de dificuldade menor quando comparado com o do problema original. Assim, resolvendo estes subproblemas chega-se a uma solução satisfatória do problema original.

O problema em questão foi o de classificação ou mapeamento dos padrões de treinamento, dividindo-os em vários subconjuntos de padrões, o que, de certa forma, não deixa de ser um mapeamento. Com isso, a resolução do problema foi facilitada.

Também foi utilizada uma nova linguagem computacional: na implementação do algoritmo de treinamento da rede neural, foi usada a linguagem C++®. O algoritmo foi o mesmo que já tinha sido implementado na linguagem do software MATLAB® versão 5.3.

A decisão por esta nova versão do algoritmo de treinamento foi baseada na sua comparação com outros dois diferentes programas para o treinamento de redes neurais, sendo que os mesmos já tinham sido empregados em fases anteriores no desenvolvimento do sistema gerador de trajetórias. A seguir, será apresentada uma breve explicação sobre estes programas, destacando o tipo de método usado para treinamento e qual a linguagem de programação que foi empregada. São eles:

- *MPO_Matlab.* Um programa que usa um método de primeira ordem, neste caso, o método do gradiente, e que foi implementado na linguagem do software MATLAB® versão 5.3. O mesmo foi usado para o treinamento da rede neural do Gerador de Trajetórias I;

- *MSO_Matlab.* Um programa que usa um método de segunda ordem, neste caso, o método do gradiente conjugado escalonado proposto por Moller (Moller, 1993) (Castro, 1998a). Na implementação deste programa foram usadas as funções desenvolvidas por L. N. de Castro, cujos códigos das mesmas, na linguagem do software MATLAB®, foram descritos no seu trabalho de 1998 (Castro, 1998d).

- *MPO_C++.* Um programa que usa um método de primeira ordem, neste caso, o método do gradiente, e que foi implementado na linguagem C++, já citado anteriormente.

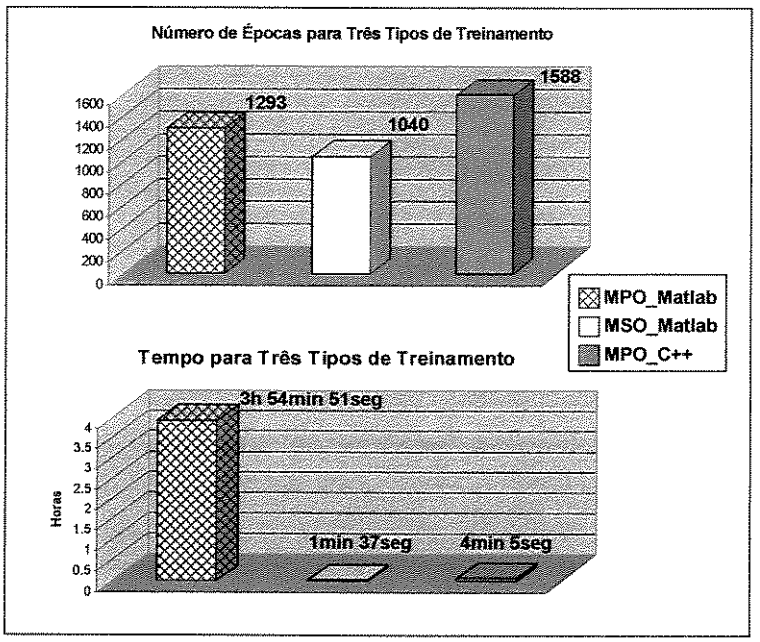


Figura 4.11 - Comparação entre os três programas de treinamento.

No gráfico da figura 4.11 estão representados, para os três programas de treinamento, o número de épocas e o tempo total (Pentium III, 550 MHz), resultantes do treinamento dos 258 padrões, que foram utilizados no primeiro sistema desenvolvido (Gerador de Trajetórias I).

Apesar do programa *MSO_Matlab* apresentar o menor tempo de treinamento dos três programas, foi escolhido o programa *MPO_C++* devido à diferença entre o seu tempo e o menor tempo de treinamento (*MSO_Matlab*) ter sido considerada pequena, quando comparada com a diferença entre o seu tempo e o maior tempo de treinamento (*MPO_Matlab*), e também pelo mesmo ser baseado numa metodologia muito mais simples do que o método de segunda ordem do *MSO_Matlab*.

Desta forma, o programa *MPO_C++* foi considerado suficiente para realizar o treinamento dos 36 subconjuntos de padrões de treinamento, resultantes da divisão do antigo conjunto de padrões considerado grande (mais que 3170 padrões).

Antes de se chegar ao número de 36 divisões, foi testada a divisão do conjunto de padrões de treinamento em 4 subconjuntos, sendo cada um deles relacionado com um determinado par de direções preferenciais, referentes aos pontos colaterais, ou seja, Nordeste, Noroeste, Sudeste e Sudoeste. Na tabela 4.06 estão listados uma média dos tempos de treinamentos, nas 6 vezes em que foram necessários novos treinamentos, destes 4 subconjuntos de padrões. Nesta fase, foi utilizado para estes treinamentos o programa *MSO_Matlab*, pois o programa *MPO_C++* ainda não tinha sido implementado.

Após essa divisão por 4, o número de entradas da rede neural, para cada um dos subconjuntos de padrões, ficou 24, pois foram retiradas as 4 entradas relacionadas com as direções preferenciais.

Ponto Colateral:	Direções Preferenciais:	Média dos Tempos:		
		h	min	seg
Nordeste	Norte e Leste	6	21	42
Noroeste	Norte e Oeste	6	20	59
Sudeste	Sul e Leste	6	43	25
Sudoeste	Sul e Oeste	7	33	12
	TOTAL:	26	59	18

Tabela 4.06 - As médias dos tempos de treinamento dos 4 subconjuntos de padrões.

O número de padrões para cada subconjunto de treinamento, na média, foi de aproximadamente 15000. Estes padrões foram gerados a partir dos 258 padrões de treinamento empregados no Gerador de Trajetórias I, através do aumento do número de combinações que descreveram cada um dos comportamentos representados nestes 258 padrões. O tempo total de

aproximadamente 27 horas para o treinamento dos 4 subconjuntos de padrões indica que essa divisão por 4 do conjunto de padrões ainda não foi satisfatória. Com isso, foi testada a divisão do conjunto total de padrões em 36 subconjuntos, sendo a mesma feita a partir da divisão de cada um dos 4 subconjuntos de padrões relacionados com os pontos colaterais em 9 novos subconjuntos, totalizando desta forma 36 subconjuntos.

O critério para essa divisão por 9, de cada subconjunto de ponto colateral, foi a configuração das memórias de curto e de longo prazo para cada um dos mesmos. A tabela 4.07 lista estas configurações possíveis das memórias e também como ficou o número de entradas para cada uma delas, representado pela variável n. Além disso, foi listada uma nomenclatura que será adotada na tabela 4.08 para cada configuração.

Ordem da Divisão:	Nome:	Configuração da Memória de Curto Prazo:	Configuração da Memória de Longo Prazo:	n
1	00	[2 2 2 2]	[2 2 2 2 2 2 2 2]	12
2	01_1	[S1 S2 2 2]	[S1 S2 2 2 2 2 2 2]	16
3	01_2	[S1 S2 S1 S2]	[S1 S2 2 2 2 2 2 2]	18
4	02	[S1 S2 S1 S2]	[S1 S2 S1 S2 2 2 2 2]	20
5	03	[S1 S2 S1 S2]	[S1 S2 S1 S2 S1 S2 2 2]	22
6	04N	[1 1 S1 S2]	[1 1 S1 S2 S1 S2 S1 S2]	24
7	04S	[0 0 S1 S2]	[0 0 S1 S2 S1 S2 S1 S2]	24
8	04L	[0 1 S1 S2]	[0 1 S1 S2 S1 S2 S1 S2]	24
9	04O	[1 0 S1 S2]	[1 0 S1 S2 S1 S2 S1 S2]	24

Tabela 4.07 - As 9 configurações das memórias adotadas para a divisão dos padrões.

Na tabela 4.08 estão listados os 36 arquivos de pesos gerados através do treinamento, da configuração final, dos seus respectivos subconjuntos de padrões, sendo para cada um desses listados também: o número de padrões, o número de épocas do treinamento, o tempo do treinamento (em horas, minutos e segundos), o número de neurônios usados na camada intermediária, representado pela variável m, e o número de entradas, representado pela variável n.

No final da tabela, encontra-se o total de padrões que formam o conjunto final de treinamento e o tempo total gasto para o treinamento dos 36 subconjuntos de padrões.

O número de neurônios usados na camada intermediária (m) foi definido de acordo com o número de padrões do subconjunto de treinamento da seguinte forma: 5 neurônios para uma quantidade pequena de padrões (menor que 210), 10 neurônios para uma quantidade média de padrões (entre 210 e 3170) e 20 neurônios (a mesma do Gerador de Trajetórias I) para uma quantidade de padrões considerada grande (maior que 3170). Desta forma, para o Gerador de Trajetórias II, todos os padrões que formam os 36 subconjuntos de treinamento das suas respectivas 36 redes neurais são formados por valores binários.

O número de entradas (n), presente nas tabelas 4.07 e 4.08 e para cada subconjunto de padrões, foi definido após a retirada das antigas entradas, relacionadas com as memórias, que tinham o valor 2, ou seja, eram consideradas vazias. Por exemplo, para o subconjunto de padrões que gerou o arquivo de pesos Pesos_th_GT_C_NL_00, que antes tinha 24 entradas, após a retirada das 12 entradas com valor 2, restaram 12 entradas, neste caso correspondendo às informações dos sensores.

O tempo total de aproximadamente 1 hora e 21 minutos, para os 62895, padrões foi considerado satisfatório e o resultado destes treinamentos foram validados através das simulações que serão apresentadas no capítulo 5. Nesta nova abordagem, toda vez que foi necessário acrescentar um novo padrão para o treinamento, somente o subconjunto que recebia este novo padrão era mais uma vez treinado e o novo arquivo de pesos gerado substituía o antigo, enquanto que os demais 35 arquivos de pesos permaneciam os mesmos.

Da mesma forma que no Gerador de Trajetória I os padrões de treinamento foram gerados e arquivados em 36 diferentes planilhas do Microsoft Excel®.

Nome do Arquivo:	Nº de Padrões:	Nº de Épocas:	Tempo:			m	n
			h	min	seg		
Pesos_th_GT_C_NL_00	13	991	0	0	31	5	12
Pesos_th_GT_C_NL_01_1	74	122	0	0	5	5	16
Pesos_th_GT_C_NL_01_2	352	657	0	1	4	10	18
Pesos_th_GT_C_NL_02	1186	445	0	1	58	10	20
Pesos_th_GT_C_NL_03	3606	335	0	7	52	20	22
Pesos_th_GT_C_NL_04N	6054	26	0	1	6	20	24
Pesos_th_GT_C_NL_04S	963	841	0	3	19	10	24
Pesos_th_GT_C_NL_04L	3330	82	0	3	58	20	24
Pesos_th_GT_C_NL_04O	146	1756	0	1	26	5	24
Pesos_th_GT_C_NO_00	13	1182	0	0	38	5	12
Pesos_th_GT_C_NO_01_1	74	131	0	0	5	5	16
Pesos_th_GT_C_NO_01_2	352	462	0	0	43	10	18
Pesos_th_GT_C_NO_02	1186	384	0	1	43	10	20
Pesos_th_GT_C_NO_03	3606	316	0	7	28	20	22
Pesos_th_GT_C_NO_04N	6054	25	0	1	3	20	24
Pesos_th_GT_C_NO_04S	963	1661	0	6	46	10	24
Pesos_th_GT_C_NO_04L	146	1290	0	1	4	5	24
Pesos_th_GT_C_NO_04O	3330	79	0	1	54	20	24
Pesos_th_GT_C_SL_00	13	450	0	0	15	5	12
Pesos_th_GT_C_SL_01_1	74	136	0	0	5	5	16
Pesos_th_GT_C_SL_01_2	352	353	0	0	34	10	18
Pesos_th_GT_C_SL_02	1186	347	0	1	33	10	20
Pesos_th_GT_C_SL_03	3605	212	0	5	20	20	22
Pesos_th_GT_C_SL_04N	963	2073	0	8	39	10	24
Pesos_th_GT_C_SL_04S	6054	13	0	0	35	20	24
Pesos_th_GT_C_SL_04L	3330	104	0	2	27	20	24
Pesos_th_GT_C_SL_04O	147	1459	0	1	12	5	24
Pesos_th_GT_C_SO_00	13	717	0	0	24	5	12
Pesos_th_GT_C_SO_01_1	74	195	0	0	7	5	16
Pesos_th_GT_C_SO_01_2	352	404	0	0	39	10	18
Pesos_th_GT_C_SO_02	1186	336	0	1	31	10	20
Pesos_th_GT_C_SO_03	3605	175	0	4	22	20	22
Pesos_th_GT_C_SO_04N	963	1063	0	4	15	10	24
Pesos_th_GT_C_SO_04S	6054	32	0	1	22	20	24
Pesos_th_GT_C_SO_04L	146	1360	0	1	6	5	24
Pesos_th_GT_C_SO_04O	3330	152	0	3	37	20	24
TOTAIS:	62895		1	20	46		

Tabela 4.08 - Os 36 vetores de pesos com as informações dos seus treinamentos.

Para se chegar a este número final de padrões (62895) foram usados dois diferentes tipos de procedimentos, sendo que cada um deles foi aplicado em fases distintas do desenvolvimento dos padrões.

Numa primeira etapa, logo após ao Gerador de Trajetórias I, para as direções preferenciais Sul e Leste (Sudeste em relação aos pontos colaterais) foram criados novos padrões com o objetivo de que novos ambientes fossem testados e que as regras adotadas na criação dos 258 padrões fossem mais fortalecidas, ou seja, elas deveriam ser cumpridas, independente da qualidade da generalização apresentada pela rede neural. Com isso, foram criados aproximadamente 15000 padrões, esta fase corresponde aos treinamentos que deram origem aos dados listados na tabela 4.06.

Para os demais pares de direções preferenciais, relacionados com os pontos colaterais Nordeste, Noroeste e Sudoeste, os padrões foram criados a partir de regras de projeção que foram aplicadas aos padrões existentes, relacionados com o ponto colateral Sudeste. Com isso, os padrões relacionados com o Sudeste foram modificados, formando estes 3 novos conjuntos de padrões. Estas regras estão listadas na tabela 4.09.

DP de Origem:	**DP Modificadas:**	**Regras em relação às variáveis simbólicas (N, S, L e O):**
Sul e Leste	**Sul e Oeste**	**Substituir o L pelo O e o O pelo L**
Sul e Leste	**Norte e Leste**	**Substituir o S pelo N e o N pelo S**
Sul e Leste	**Norte e Oeste**	**Substituir o S pelo N, o L pelo O, o N pelo S e o O pelo L**

Tabela 4.09 - As 3 regras usadas na projeção das direções.

As modificações eram feitas na planilha do Microsoft Excel®, sendo que as variáveis simbólicas usadas na representação dos padrões eram alteradas conforme as regras da tabela 4.09. A seguir, na tabela 4.10, têm-se 2 exemplos da aplicação de cada uma destas regras, exemplos estes retirados da tabela 4.04.

Par Sudeste de Direções Preferenciais (Origem):																															
Se(k)				**Se(k-1)**				**Se(k-1)**				**DP**				**MCP(k-1)**				**MCP(k-2)**				**MLP**				**S1S2(k)**			
	S	L											S	L														S		
N	S	L	O	N	S	L	O	N	S		O		S	L				L		N				L	N	S	L			L	
Par Sudoeste de Direções Preferenciais:																															
Se(k)				**Se(k-1)**				**Se(k-1)**				**DP**				**MCP(k-1)**				**MCP(k-2)**				**MLP**				**S1S2(k)**			
	S		O										S		O													S		
N	S	L	O	N	S	L	O	N	S	L			S		O				O	N				O	N	S	O				O
Par Nordeste de Direções Preferenciais:																															
Se(k)				**Se(k-1)**				**Se(k-1)**				**DP**				**MCP(k-1)**				**MCP(k-2)**				**MLP**				**S1S2(k)**			
N		L										N		L														N			
N	S	L	O	N	S	L	O	N	S		O	N		L				L			S			L	S	N	L			L	
Par Noroeste de Direções Preferenciais:																															
Se(k)				**Se(k-1)**				**Se(k-1)**				**DP**				**MCP(k-1)**				**MCP(k-2)**				**MLP**				**S1S2(k)**			
N			O									N			O													N			
N	S	L	O	N	S	L	O	N	S	L		N			O				O		S			O	S	N	O				O

Tabela 4.10 - Dois exemplos da aplicação das regras de projeção das direções.

Com isso, foram criados aproximadamente novos 15000 padrões para cada uma das configurações de direções preferenciais, totalizando 60000 padrões de treinamento.

O outro procedimento, aplicado na fase final de desenvolvimento dos padrões, consistia da simples inclusão de novos padrões, nos seus respectivos conjuntos, sem que fosse necessária a projeção dos mesmos para os outros conjuntos. Estas inclusões serviram para pequenos ajustes, fazendo com que a aplicação do sistema fosse satisfatória para os novos ambientes testados.

A seguir, será apresentada a configuração final do segundo sistema proposto, o qual foi chamado de Gerador de Trajetória II. A sua diferença em relação ao Gerador de Trajetórias I foi a substituição da sua rede neural por um sistema formado por 36 redes neurais em paralelo. Na figura 4.12, está representado o esquema do Gerador de Trajetórias II.

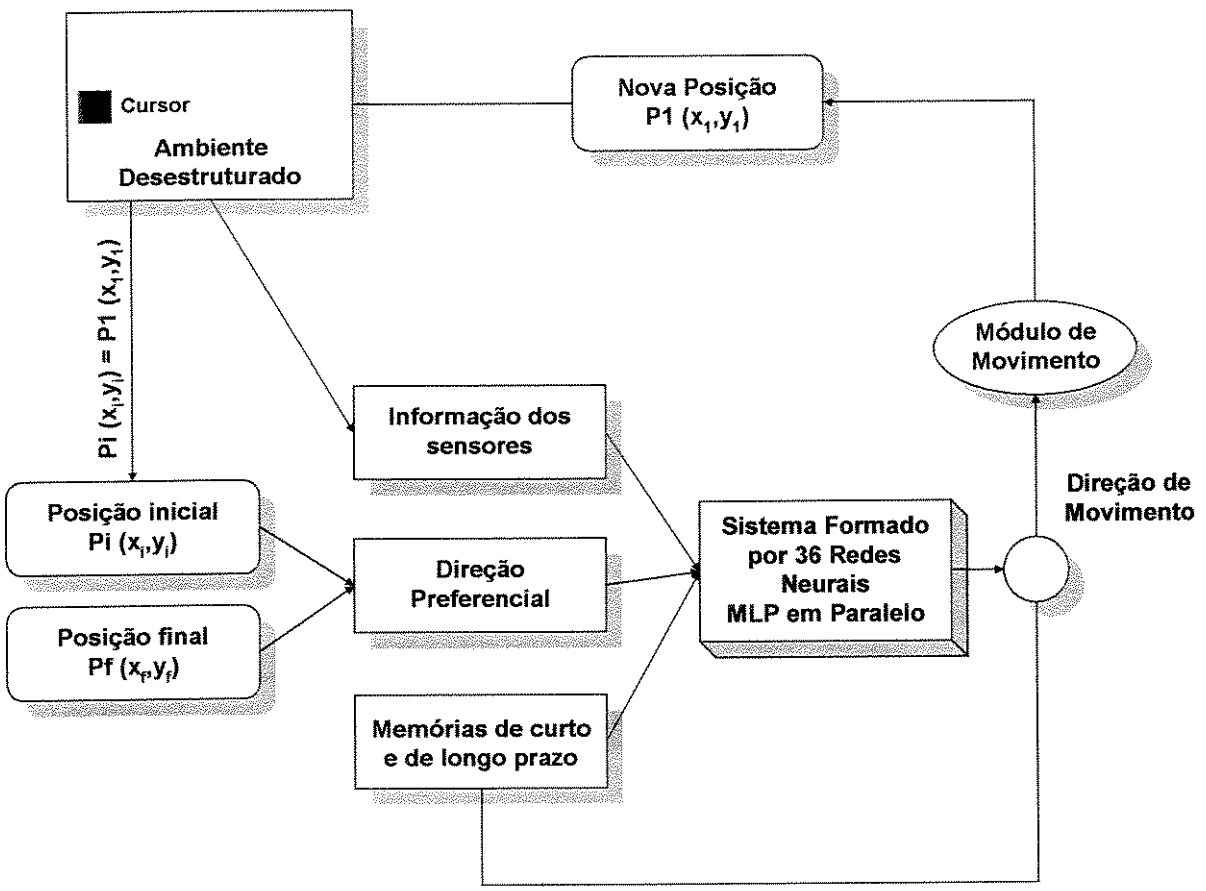


Figura 4.12 - O esquema do Gerador de Trajetórias II.

Antes, era aplicada uma abordagem via redes neurais para a decisão da direção de movimento do cursor. Já na segunda proposta, esta abordagem foi substituída por outra que pode ser considerada como híbrida, formada pela junção de redes neurais com funções lógicas. No programa, somente uma única rede é ativada a cada iteração, o que não compromete o seu custo computacional, como ocorria no caso do Gerador de Trajetórias I.

As funções lógicas são usadas na escolha da rede neural, entre as 36 existentes, que deve ser ativada em uma iteração qualquer, ou seja, as funções lógicas são usadas para o chaveamento da rede neural que deve ser aplicada. A cada iteração, essas funções decidem com base em dois níveis, sendo o primeiro relacionado com as informações das direções preferenciais e o segundo em relação à configuração das memórias de curto e longo prazo.

Na figura 4.13, tem-se o primeiro nível de decisão onde a partir das informações das direções preferenciais uma entre as 8 funções representadas é chaveada por iteração. Estas funções estão relacionadas com as possíveis configurações do vetor **DP**, que foram listadas na tabela 4.01. Têm-se dois grupos de funções: as chamadas funções de pontos colaterais e as chamadas funções de pontos cardeais. Na tabela 4.11 estão relacionadas estas funções com as possíveis configurações do vetor **DP**.

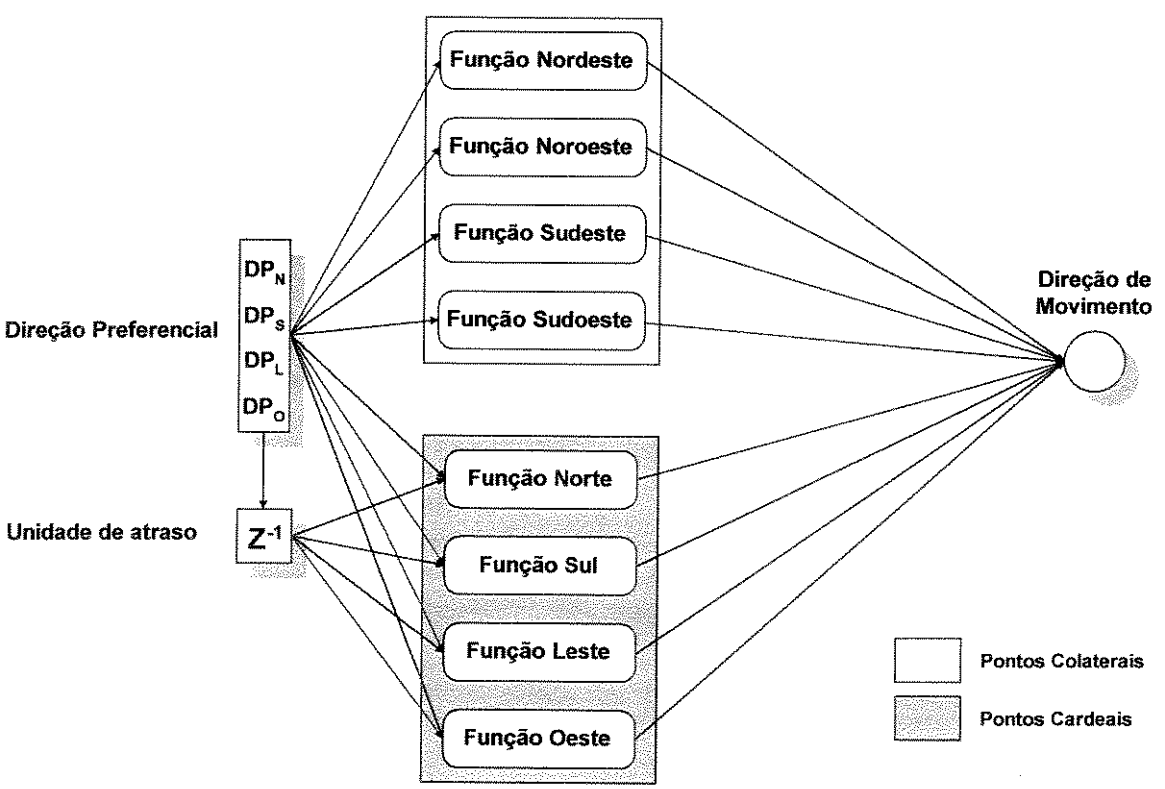


Figura 4.13 - O primeiro nível de decisão.

Vetor DP:	Primeiro Nível de Decisão:
[0 1 0 1]	Função Nordeste
[0 1 1 0]	Função Noroeste
[1 0 0 1]	Função Sudeste
[1 0 1 0]	Função Sudoeste
[0 1 1 1]	Função Norte
[1 0 1 1]	Função Sul
[1 1 0 1]	Função Leste
[1 1 1 0]	Função Oeste

Tabela 4.11 - O primeiro nível de decisão.

A figura 4.14 representa um exemplo de uma função de ponto cardeal, neste caso, a Função Norte.

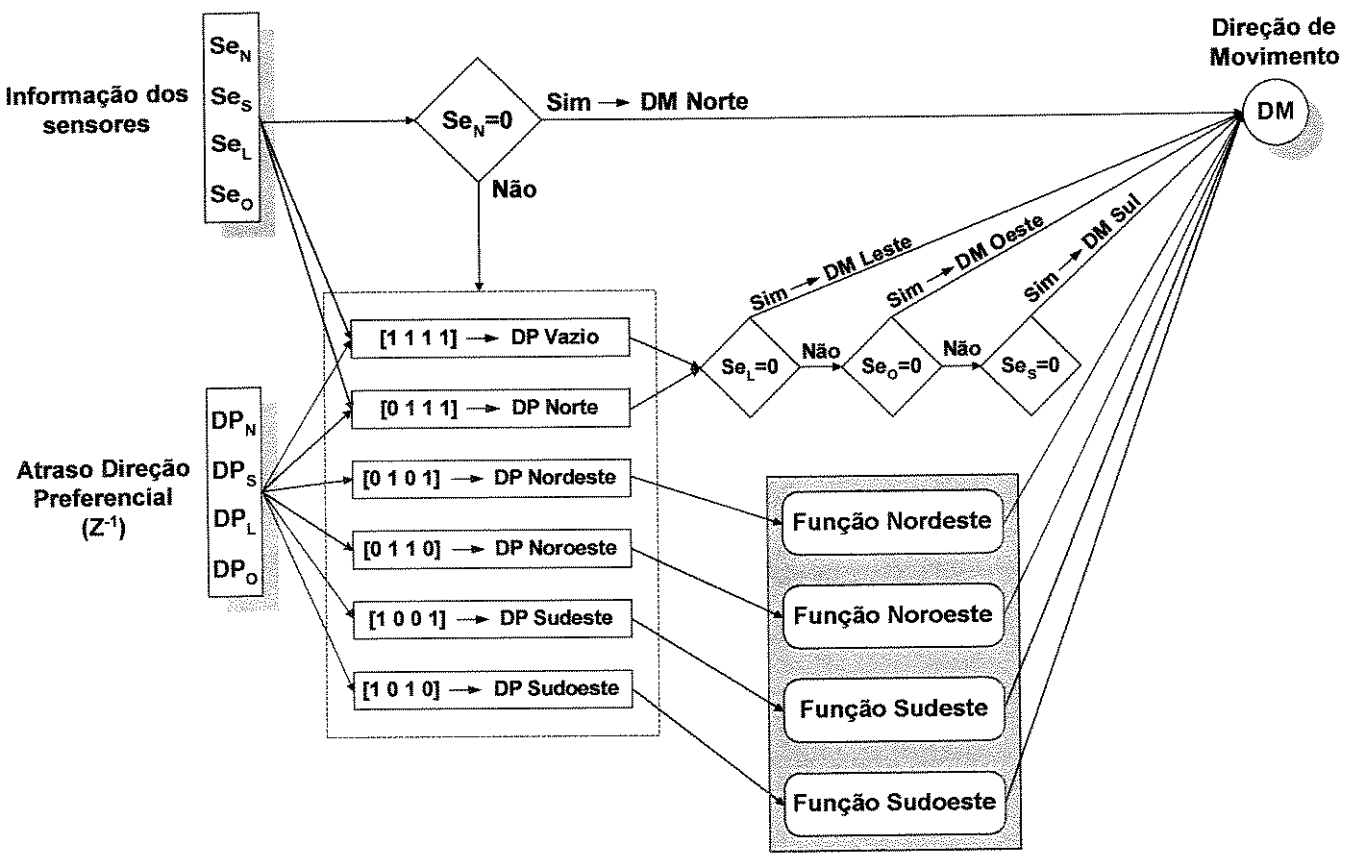


Figura 4.14 - O esquema da função de ponto cardeal Norte.

Caso seja chaveada uma das quatro funções de pontos cardeais, a escolha da direção de movimento do cursor ocorre por meio de funções lógicas baseada nas informações dos sensores e nas informações do vetor **DP**(k-1), ou seja, da iteração anterior.

A primeira escolha, representada na figura 4.14 para a função Norte, é bem simples: caso a variável de entrada $Se_N(k)$ seja 0 a direção de movimento é a Norte, ou seja, os sensores indicam que a única direção preferencial adotada está livre, logo ela deve ser também a direção de movimento. Caso contrário deve-se considerar além das informações dos sensores as informações do vetor **DP**(k-1).

Neste caso, a segunda escolha é quando **DP**(k-1) for igual a [1 1 1 1] ou [0 1 1 1], ou seja, é a primeira iteração ou a direção preferencial é novamente o Norte. Para estas possibilidades, verifica-se o valor da variáveis de entradas dos sensores. A primeira opção é se $Se_L(k)$ é 0, sendo escolhida a direção de movimento Leste, caso contrário, verifica-se $Se_O(k)$. Se ela for 0 adota-se a direção de movimento Oeste, caso contrário, a última escolha é se $Se_S(k)$ for 0. Assim sendo, adota-se o Sul como direção de movimento. A escolha dessa ordem das direções foi baseado no seguinte critério: antes de que seja adotada uma direção de sentido contrário da preferencial (Norte para Sul) deve-se adotar uma das direção perpendiculares à direção preferencial (Leste ou Oeste).

Nos outros casos de **DP**(k-1), a escolha da direção de movimento é através de uma das funções de ponto colateral que foi chaveada conforme **DP**(k-1), como está representado na figura 4.14.

A figura 4.15 representa um exemplo de uma função de ponto colateral, neste caso, a Função Nordeste. Nesta figura, tem-se também a representação do segundo nível de decisão, citado anteriormente. Aqui, a decisão ocorre em relação à configuração das memórias de curto e longo prazo. Estas configurações, listadas na tabela 4.07, vão fazer com que as funções lógicas

façam o chaveamento com as 9 arquiteturas de redes neurais possíveis, que estão representadas nas figuras 4.16, 4.17, 4.18, 4.19, 4.20, 4.21, 4.22, 4.23 e 4.24.

Estas arquiteturas vão variar em relação às suas informações de entradas, e consequentemente ao número de entradas e ao número de neurônios das suas camadas intermediárias.

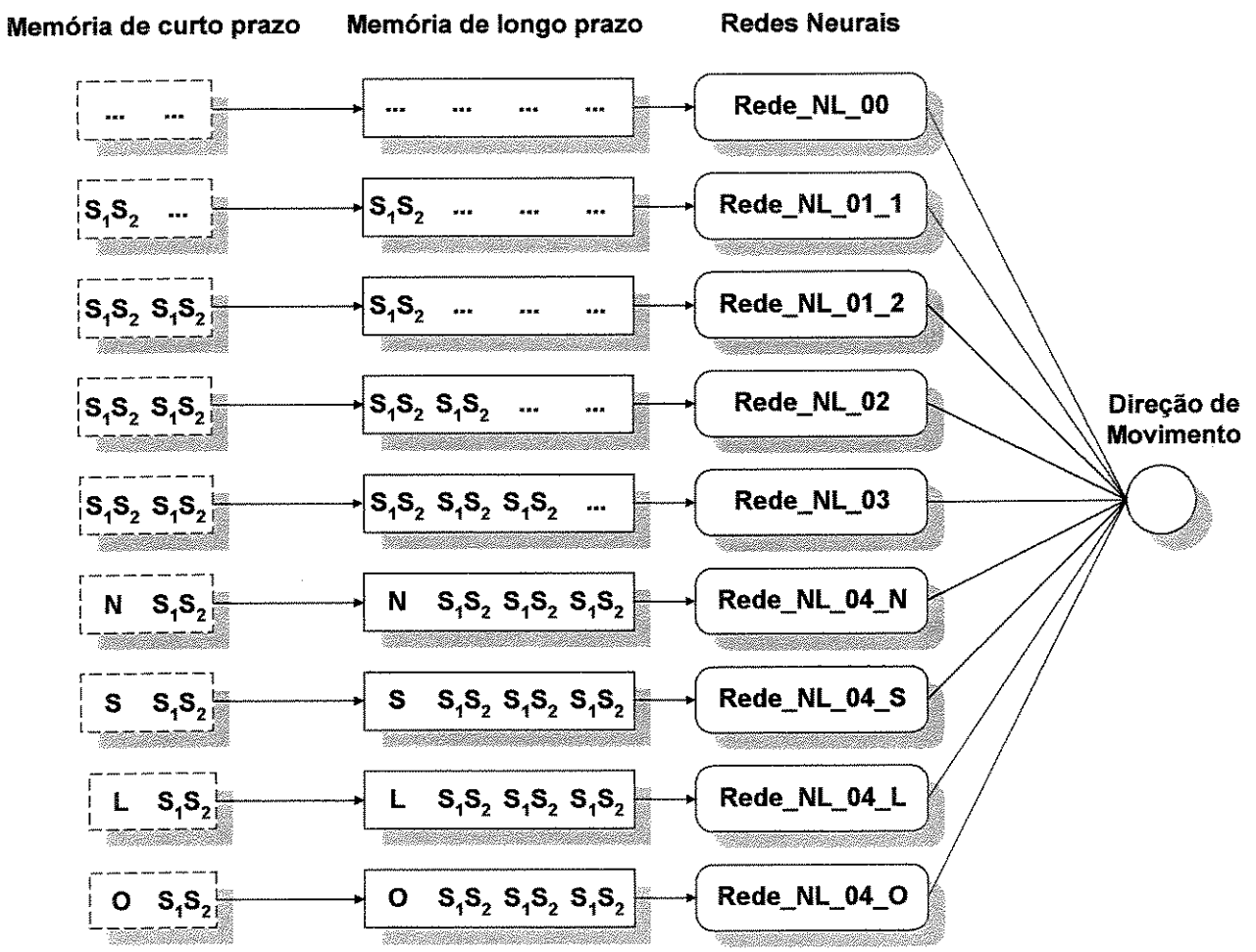


Figura 4.15 - O esquema da função de ponto colateral Nordeste.

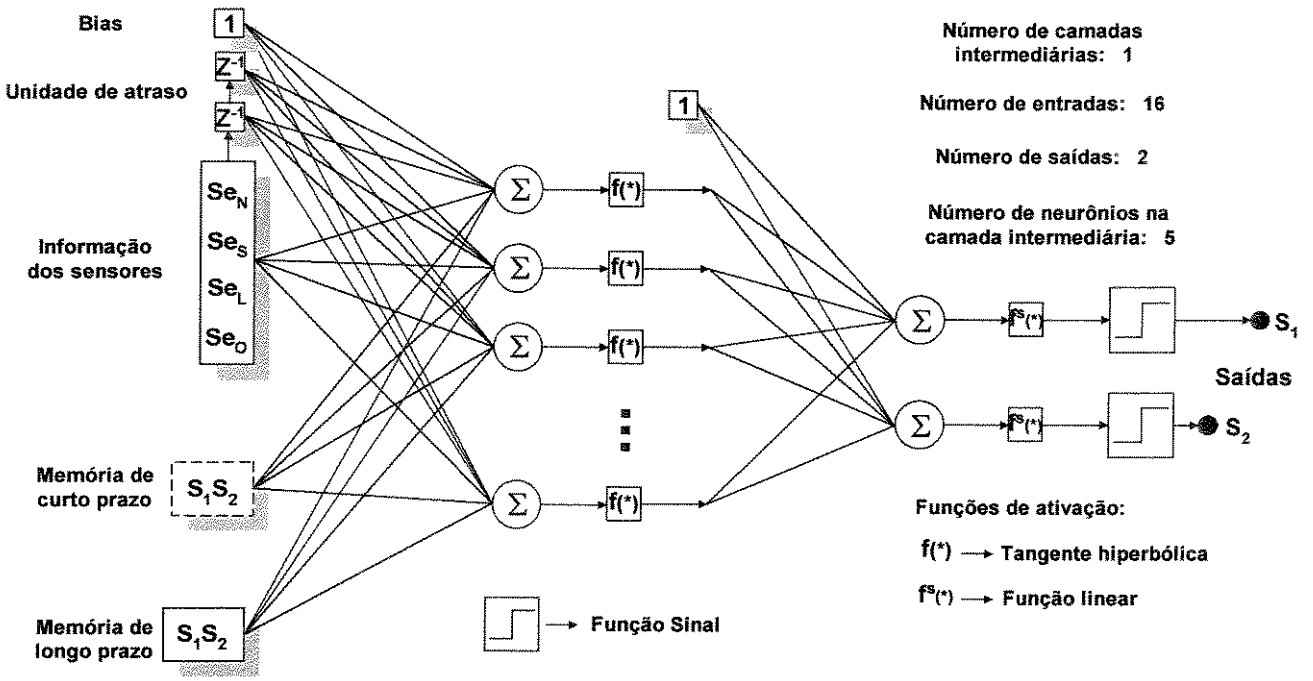


Figura 4.16 - A arquitetura da Rede_NL_00.

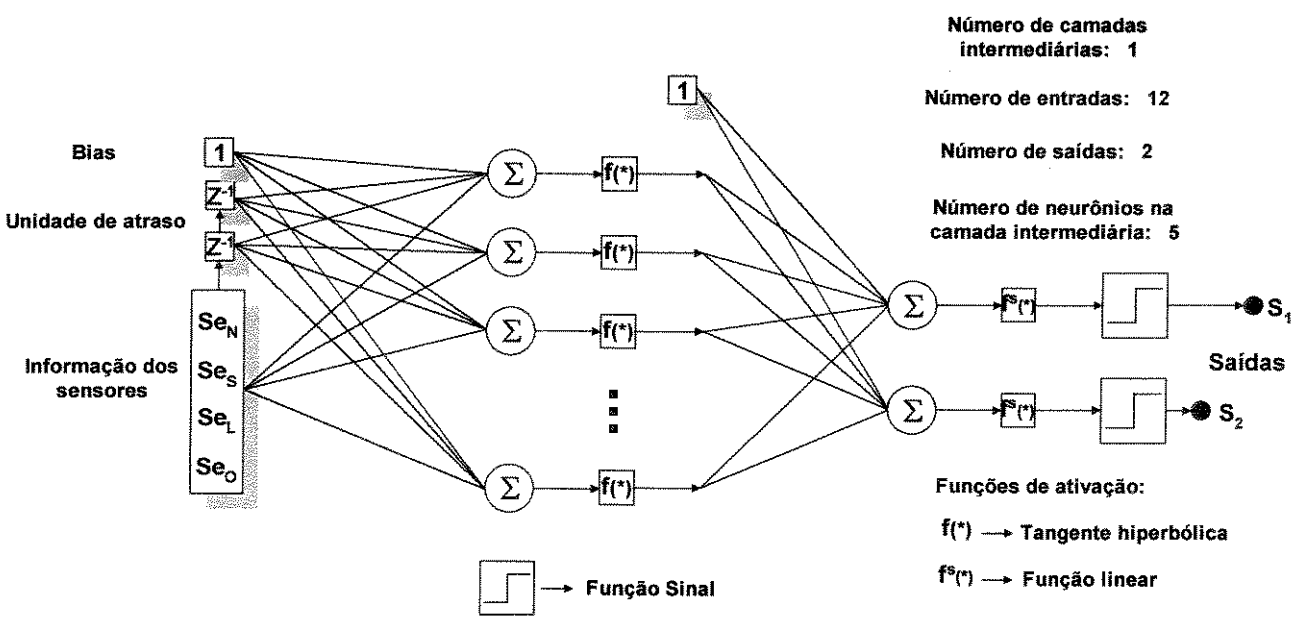


Figura 4.17 - A arquitetura da Rede_NL_01_1.

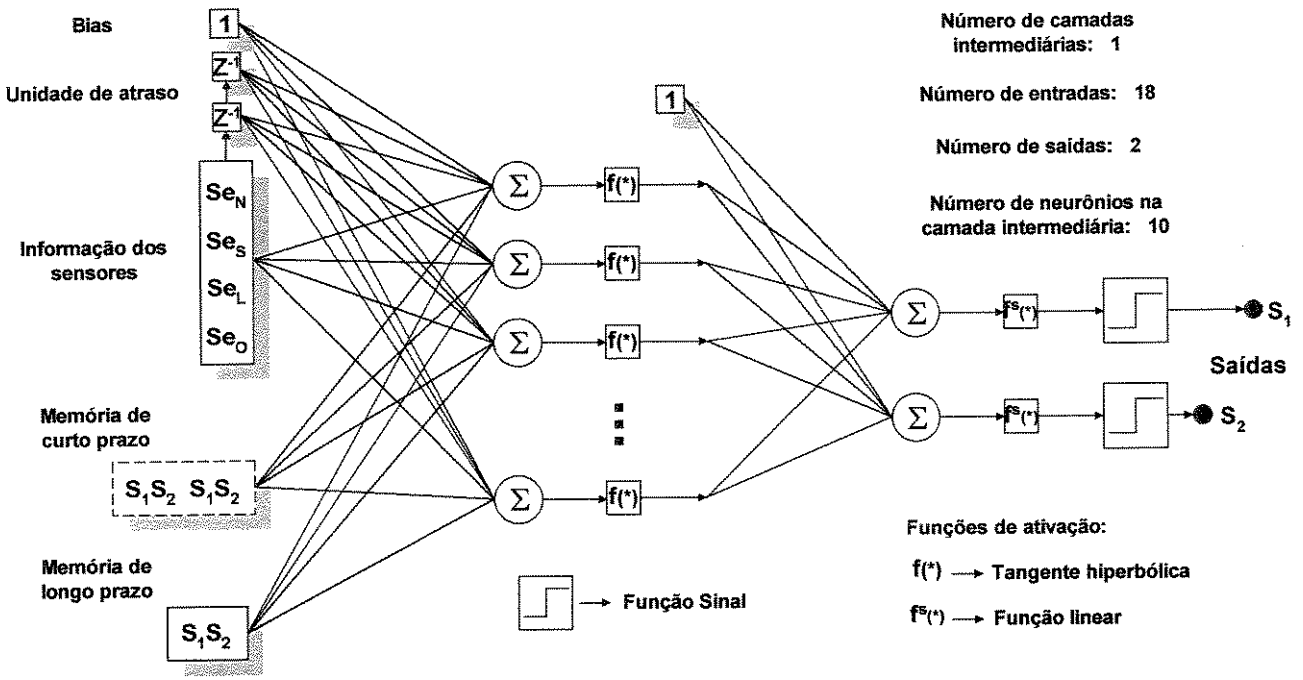


Figura 4.18 - A arquitetura da Rede_NL_01_2.

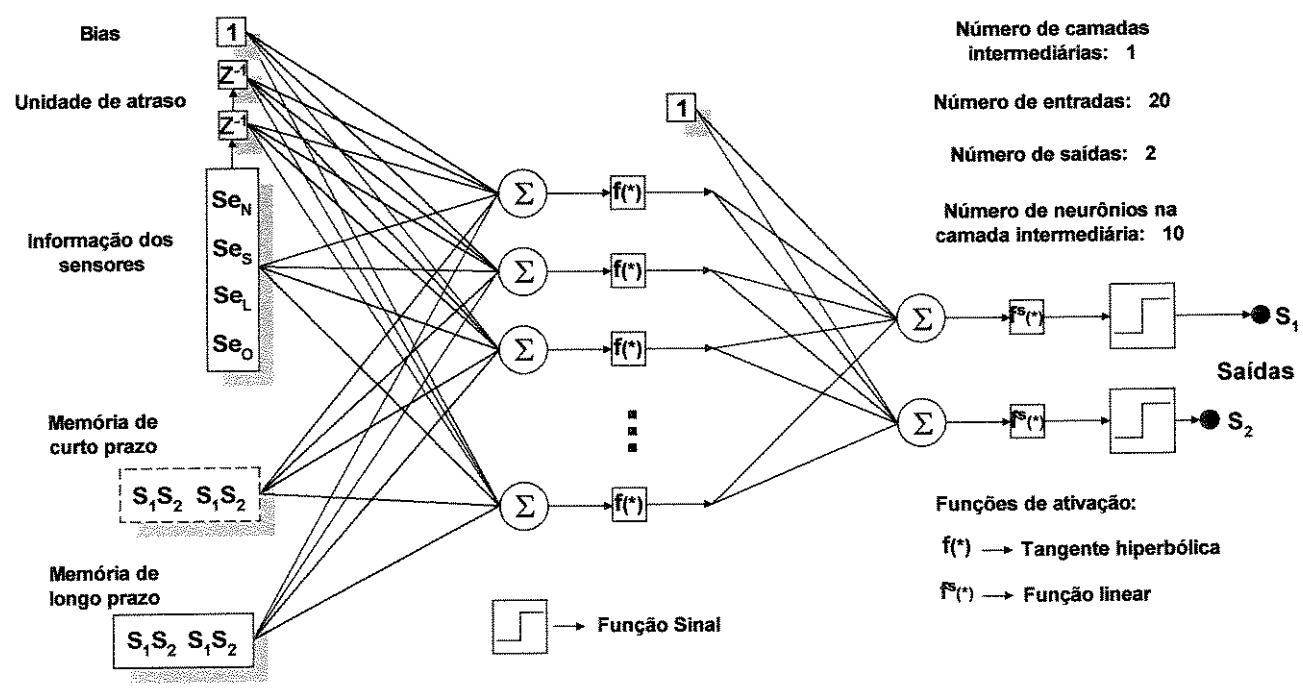


Figura 4.19 - A arquitetura da Rede_NL_02.

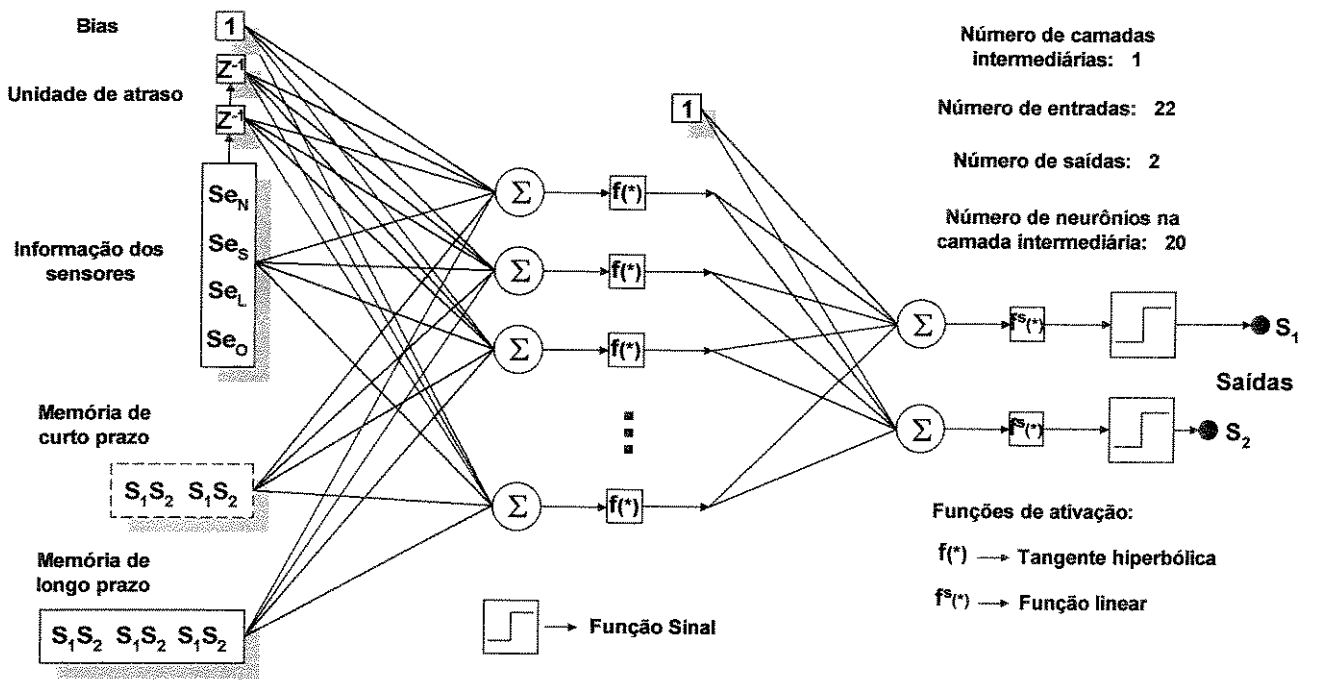


Figura 4.20 - A arquitetura da Rede_NL_03.

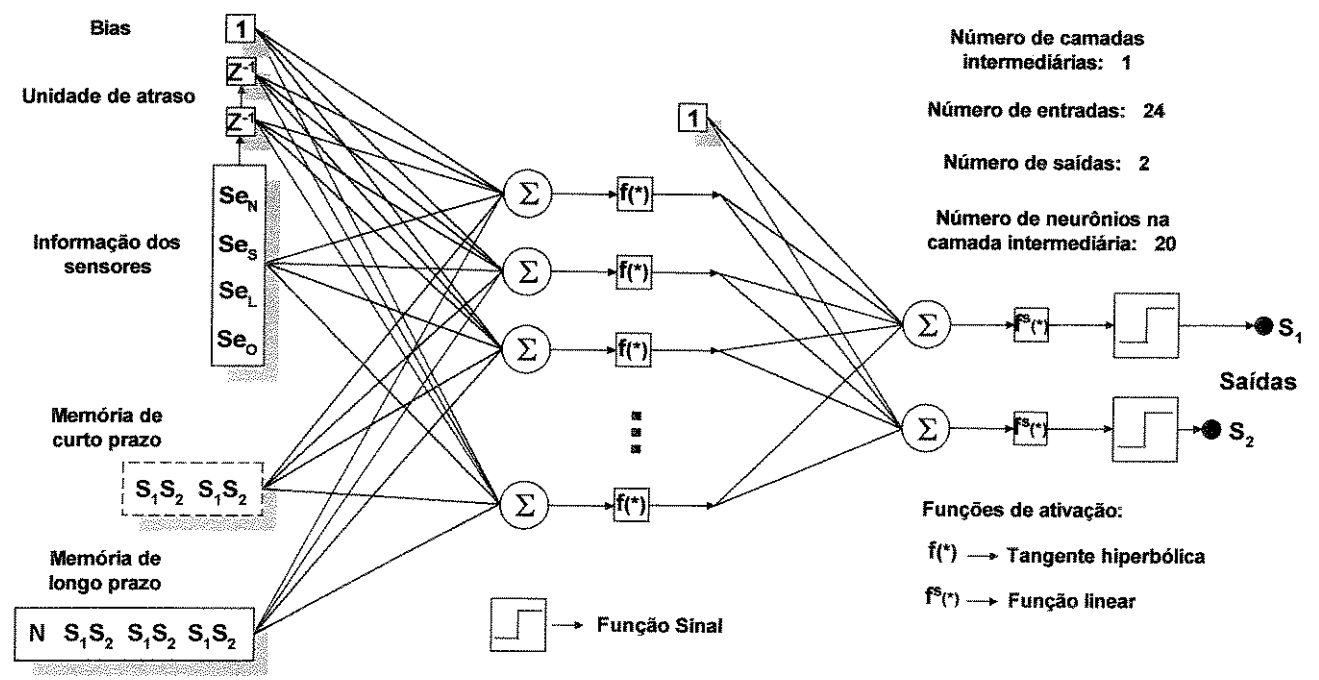


Figura 4.21 - A arquitetura da Rede_NL_04_N.

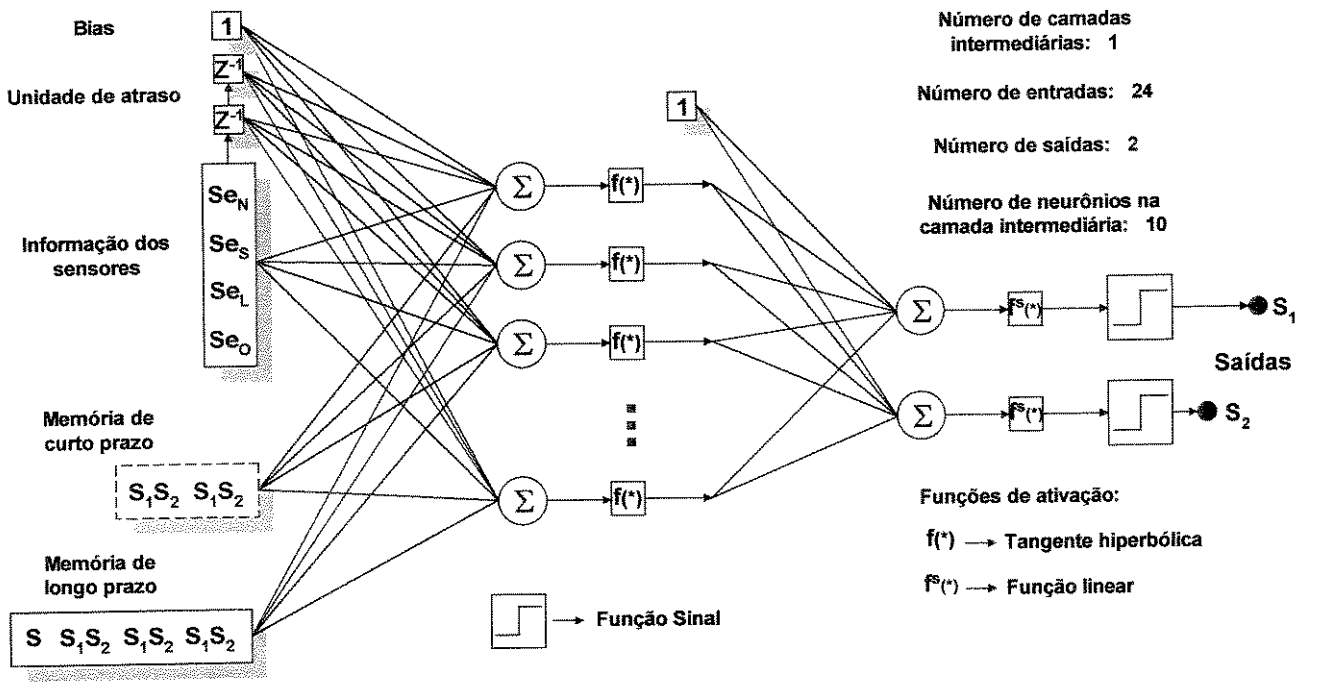


Figura 4.22 - A arquitetura da Rede_NL_04_S.

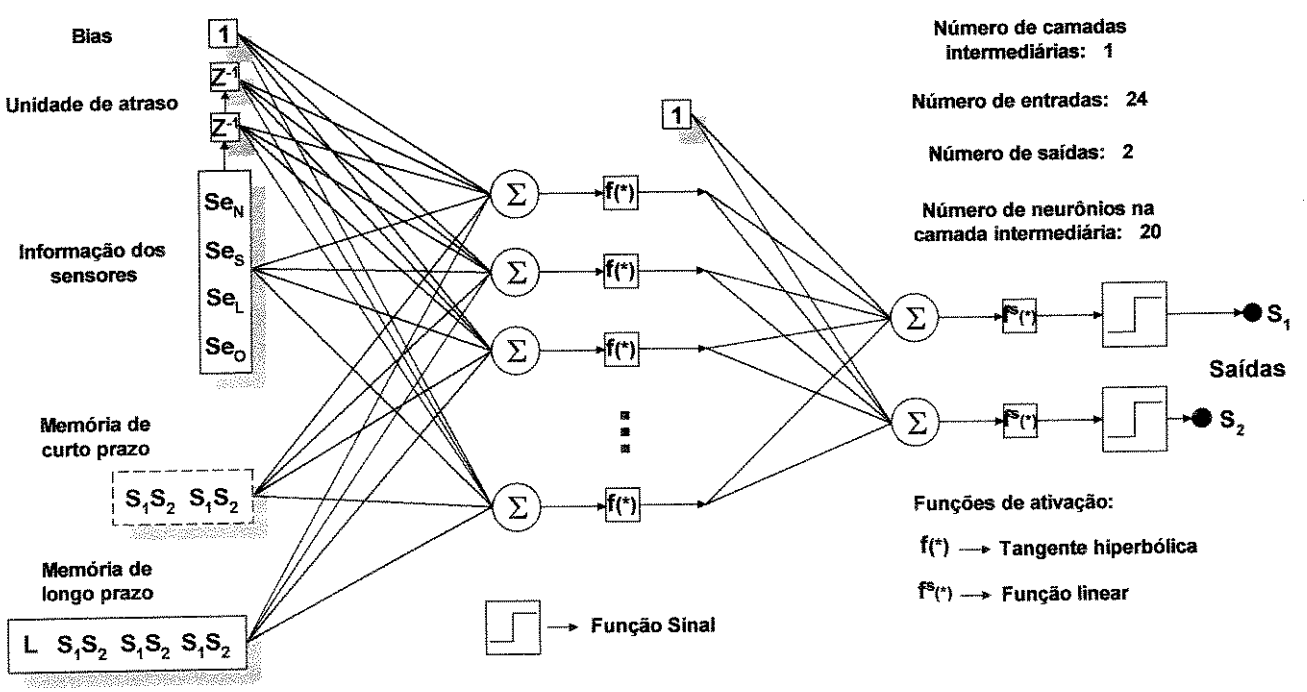


Figura 4.23 - A arquitetura da Rede_NL_04_L.

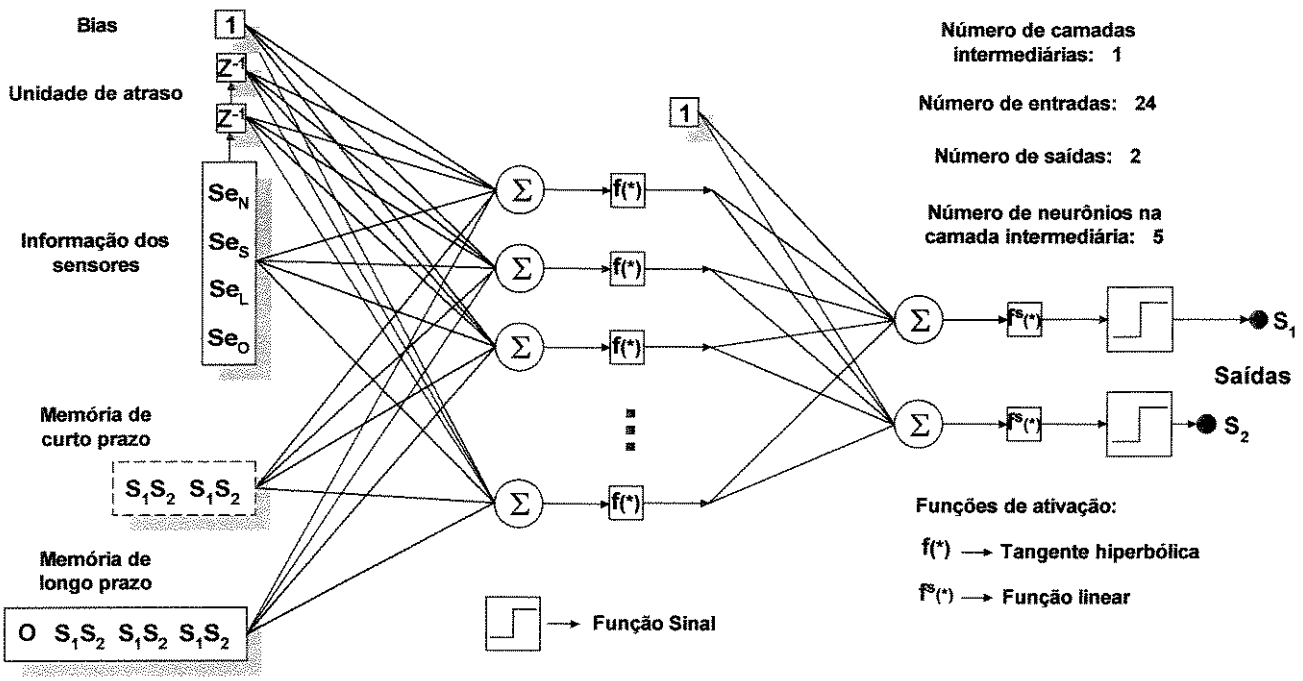


Figura 4.24 - A arquitetura da Rede_NL_04_O.

4. 4. Conclusões

As duas abordagens descritas neste capítulo foram desenvolvidas com o objetivo de resolver o problema de navegação autônoma. A principal barreira encontrada na utilização das redes neurais do tipo MLP, que levou ao desenvolvimento da segunda abordagem, foi a relação existente entre o número de padrões usados no treinamento e o tempo de convergência do mesmo. À medida que foi se elevando o número de padrões, o tempo de treinamento também aumentou, o que dificultou o teste dos padrões usados e a inclusão de novos padrões. O emprego de duas estratégias visando a diminuição do tempo do treinamento, a divisão do conjunto dos padrões e o uso de um programa de treinamento implementado em C++, se mostraram eficientes na realização da tarefa de navegação autônoma, cujos resultados serão mostrados no próximo capítulo.

Capítulo 5

Simulações

5. 1. Introdução

Como já foi descrito no capítulo 4, o cursor, dotado de sensores e com a capacidade de deslocar-se pela área de trabalho, realizará movimentos reconhecendo regiões "livres de obstáculos" e caminhos que poderão ser utilizados por um mecanismo físico (robô móvel) (Mendeleck, 1995).

Deslocando-se com um avanço fixo, o cursor adota a direção de movimento que foi determinada pela rede neural, chegando com isso, em um novo ponto $P1(x_1,y_1)$. Em seguida, esse novo ponto passa a ser considerado como o ponto inicial $Pi(x_i,y_i)$, ou seja, para a iteração k+1, Pi(k+1)=P1(k), e assim sucessivamente, até que um desses pontos da trajetória faça parte do conjunto de pontos que formam uma região de tolerância (circular), que tem como centro o ponto final $Pf(x_f,y_f)$ definido anteriormente. Assim sendo, foi definido como critério de parada do programa a condição de que o ponto P1 pertença a uma região circular de raio fixo, neste caso foi adotado o valor de 20 unidades (*pixels*), que tem como centro o ponto final Pf.

A existência dessa região circular, representada na figura 5.001, deve-se ao fato de que o objetivo do gerador de trajetórias é determinar uma série de pontos intermediários entre os pontos inicial e final, e não achar um ponto que tenha as mesmas coordenadas do ponto alvo. Essa região pode ser considerada como uma região de tolerância, pois o cursor se movimenta com um avanço

fixo e diferente de 1, sendo muito improvável que um ponto da trajetória coincida com o ponto alvo (Pf).

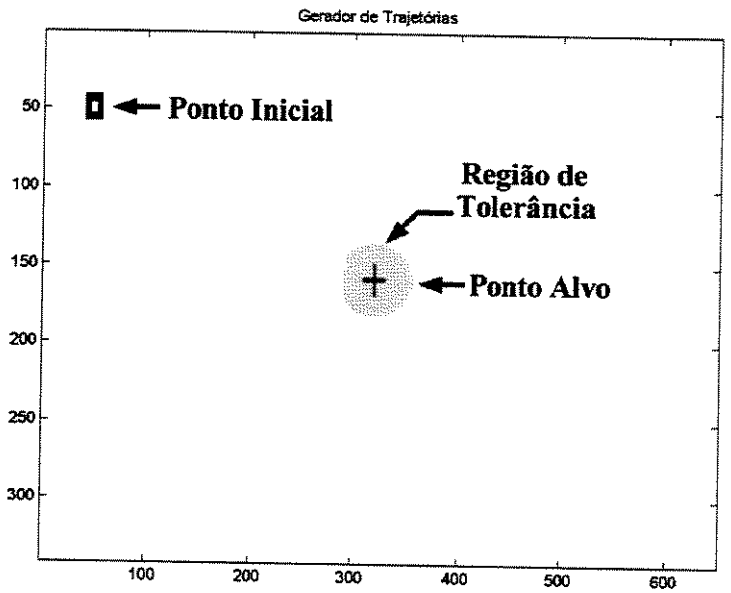


Figura 5.001 - A região de tolerância e os pontos inicial e alvo da trajetória.

A simulação da movimentação do robô móvel por um ambiente desestruturado usando o gerador de trajetórias, sendo conhecidas as suas posições inicial e alvo, foi feita no próprio programa. Um gráfico, representando o ambiente juntamente com o cursor era gerado para cada posição do mesmo, dando uma idéia de movimento do cursor.

Neste gráfico, também representado na figura 5.001, tem-se o ambiente de trabalho do cursor, com um formato retangular possuindo dimensões fixas iguais a 650x340 *pixels*, dimensões estas conforme as escalas dos eixos das abcissas e das ordenadas também presentes no gráfico.

A seguir, serão apresentados os resultados gráficos das simulações para 13 diferentes configurações do ambiente de trabalho do cursor, sendo adotada a legenda representada na figura 5.002 para a identificação dos principais componentes destes gráficos.

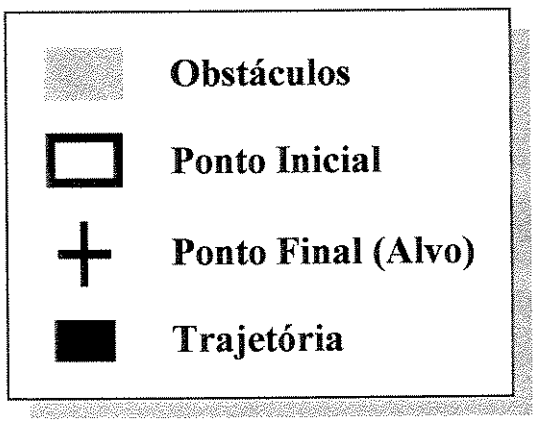


Figura 5.002 - A legenda dos gráficos.

5. 2. Ambientes

Foram utilizadas 13 diferentes configurações de ambientes para testar a eficiência do sistema proposto. A seguir, serão citados estes tipos de ambientes, sendo descrita também a referência da figura usada na representação de cada um destes ambientes.

- Ambiente sem obstáculos (vazio) → Figura 5.003. Neste tipo de ambiente, somente as paredes limítrofes do mesmo podem causar a condição de direção ocupada ou não livre por parte dos sensores;
- Ambiente com um obstáculo côncavo (Mendeleck, 1995) → Figura 5.004;
- Ziguezague na vertical I (Mendeleck, 1995) → Figura 5.005. Ambiente com obstáculos formando um ziguezague na vertical sendo sua entrada localizada na parte superior do mesmo;
- Dois Ziguezagues na horizontal → Figura 5.006;
- Ziguezague na vertical II → Figura 5.007. Neste caso a entrada do ziguezague está localizada na parte inferior do mesmo;
- Ambiente com um ziguezague na horizontal e um obstáculo côncavo → Figura 5.008;

- Labirinto (Oliveira, 1995) → Figura 5.009. Na verdade este tipo de ambiente pode ser considerado como um ziguezague, diferenciando dos demais por possuir mais espaço entre as suas paredes;
- Ambiente com obstáculos I → Figura 5.010. Neste tipo de ambiente foi testada a movimentação do cursor na presença de obstáculos mais simples e também a passagem entre dois obstáculos na forma de um corredor (restrição na entrada ou na saída);
- Ambiente com obstáculos II (Vaz, 1999) → Figura 5.011. Para este tipo de ambiente foi testado o sistema na presença de obstáculos em "U" (Oliveira, 1995);
- Ambiente tipo Estrela I (Oliveira, 1995) → Figura 5.012;
- Ambiente tipo Estrela II (Oliveira, 1995) → Figura 5.013;
- Planta Baixa → Figura 5.014. Este tipo de ambiente é uma tentativa de testar o sistema proposto em um ambiente real de trabalho, com paredes formando salas, portas e um corredor, sendo utilizada uma planta baixa para a representação do mesmo;
- Obstáculo Móvel → Figura 5.015. Neste tipo de ambiente foi testado o sistema na presença de um obstáculo móvel, com dimensões de 20x20 *pixels* e com um avanço 4 vezes menor do que o avanço do cursor;

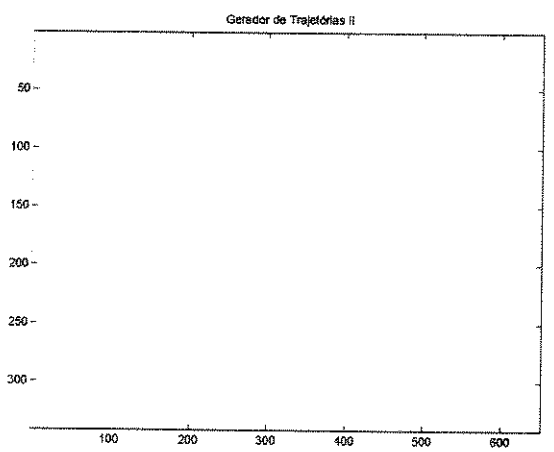


Figura 5.003 - Ambiente sem obstáculos.

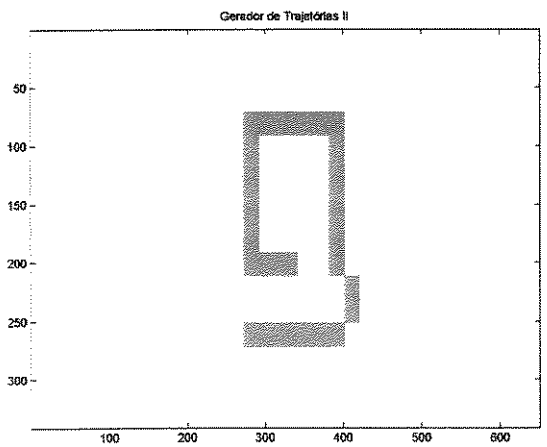


Figura 5.004 - Obstáculo côncavo.

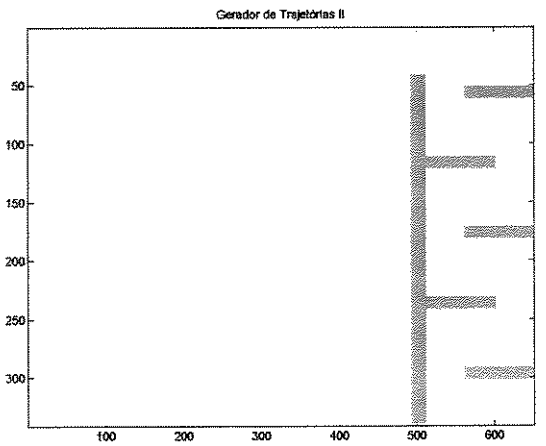


Figura 5.005 - Ziguezague na vertical I.

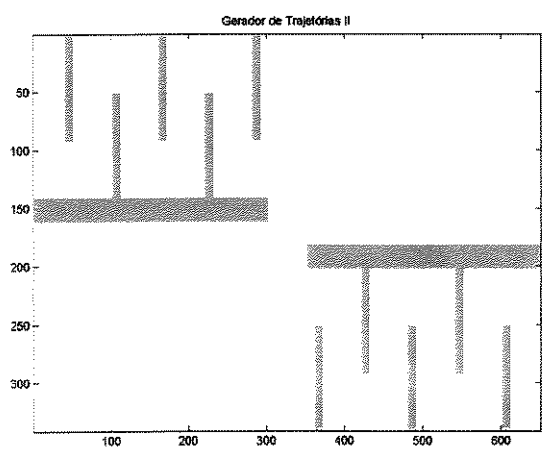


Figura 5.006 - 2 Ziguezagues na horizontal.

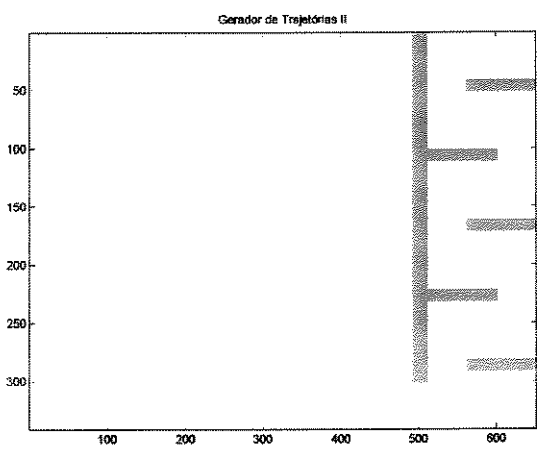


Figura 5.007 - Ziguezague na vertical II.

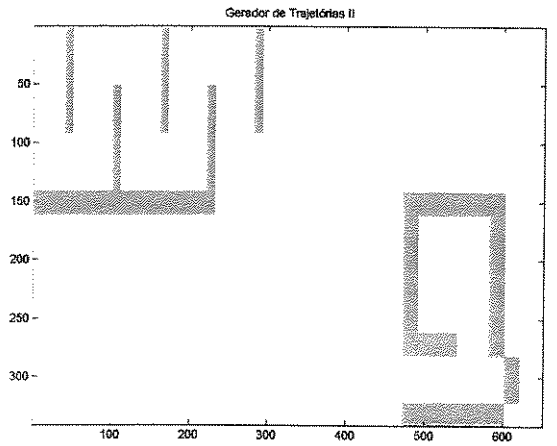


Figura 5.008 - Ziguezague e um côncavo.

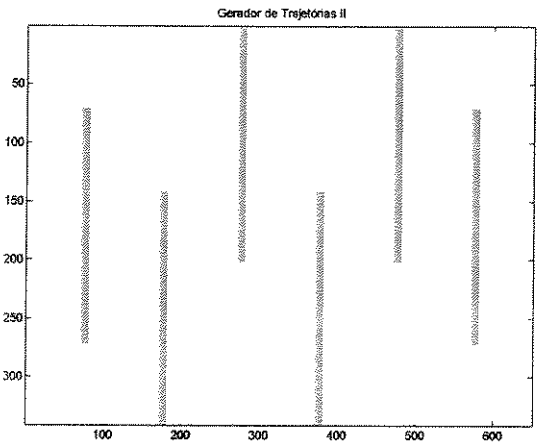


Figura 5.009 - Labirinto.

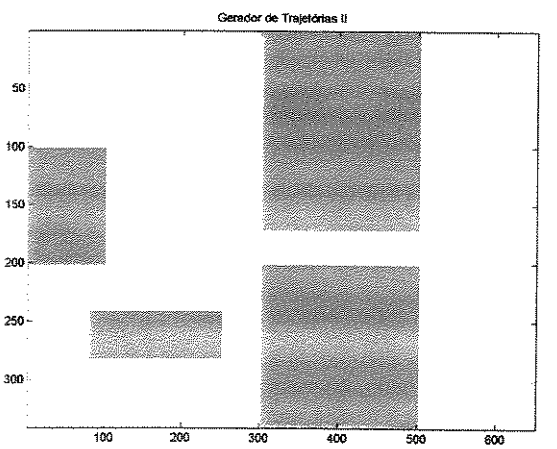


Figura 5.010 - Ambiente com obstáculos I.

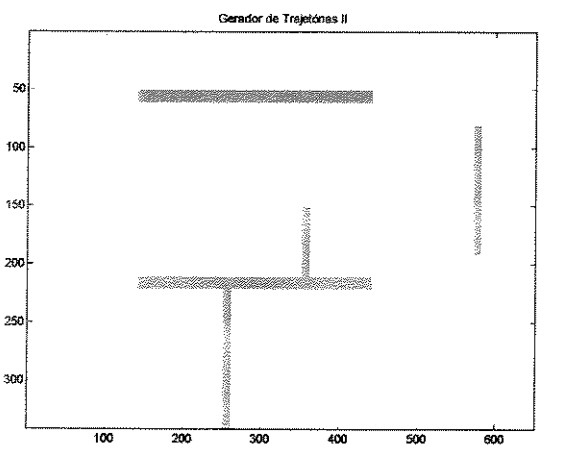


Figura 5.011 - Ambiente com obstáculos II.

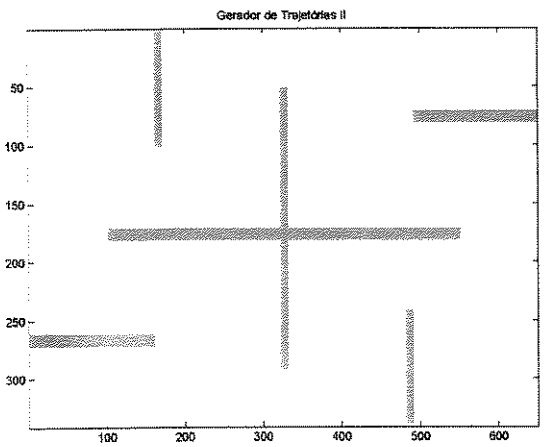


Figura 5.012 - Ambiente tipo Estrela I.

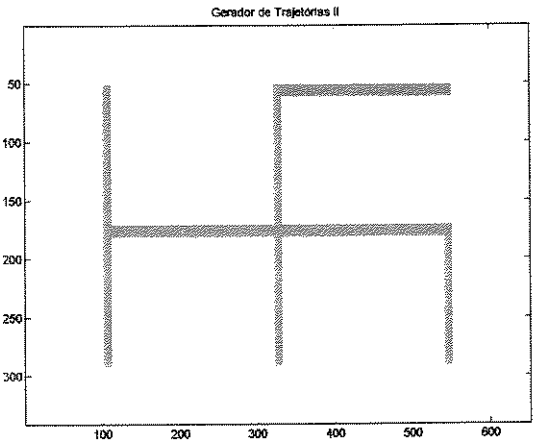


Figura 5.013 - Ambiente tipo Estrela II.

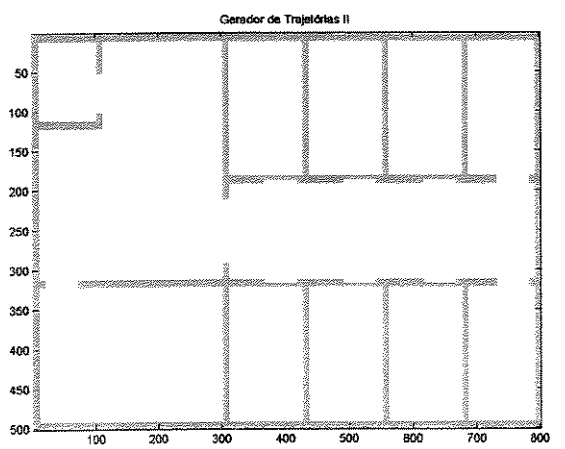


Figura 5.014 - Planta Baixa.

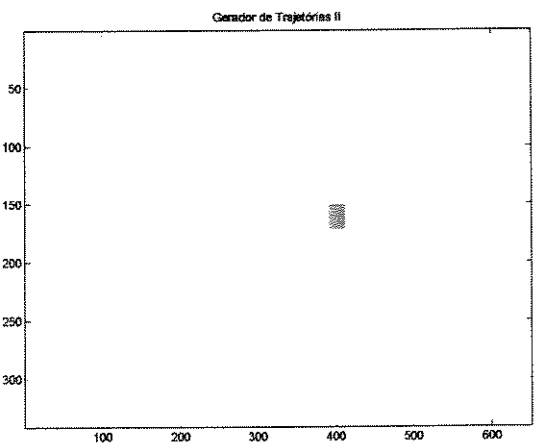


Figura 5.015 - Obstáculo Móvel.

5. 3. Resultados do Gerador de Trajetórias I

A seguir, serão mostrados os resultados finais das simulações do Gerador de Trajetórias I, destacando a representação dos ambientes, utilizados na geração dos padrões de treinamento da rede neural, nas figuras 5.016, 5.017, 5.018 e 5.019, devido ao fato de suas configurações terem sido consideradas as mais básicas e representativas dentre as configurações geralmente encontradas (Mendeleck, 1995).

Também as figuras 5.020 e 5.021, que representam outros ambientes, derivados dos que foram usados para treinar a rede neural, serviram para avaliar a generalização dos resultados da mesma. Estes ambientes serviram para testar os resultados do aprendizado e comprovar a eficiência do processo de navegação.

Explicando a generalização dos resultados da rede neural, tem-se a figura 5.020, comprovando a eficiência do sistema para um ambiente com dois ziguezagues na horizontal, quando foi utilizado no treinamento da rede, por exemplo, um ambiente com um ziguezague na vertical, representado na figura 5.019.

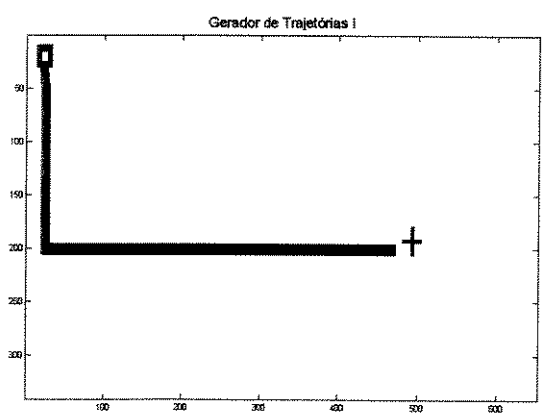


Figura 5.016 - Ambiente sem obstáculos.

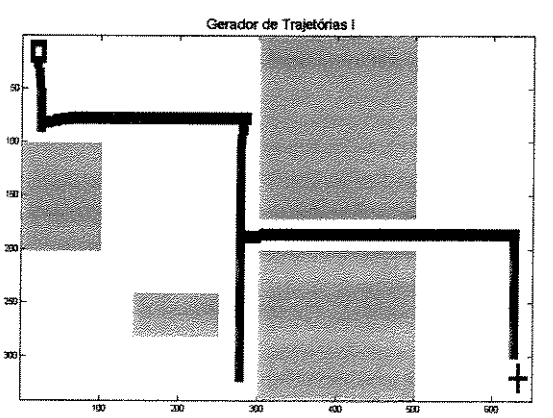


Figura 5.017 - Ambiente com obstáculos I.

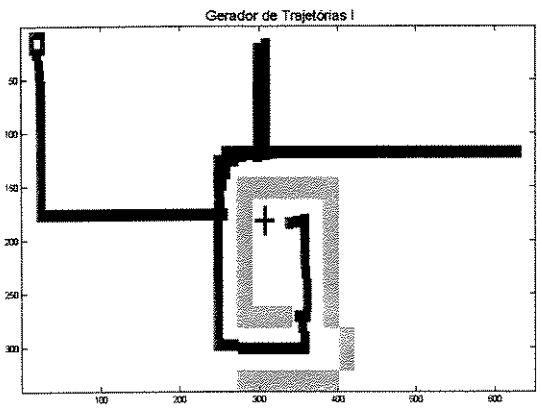


Figura 5.018 - Obstáculo Côncavo.

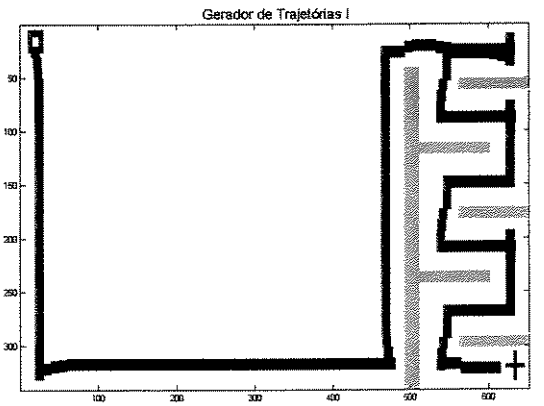


Figura 5.019 - Ziguezague na Vertical I.

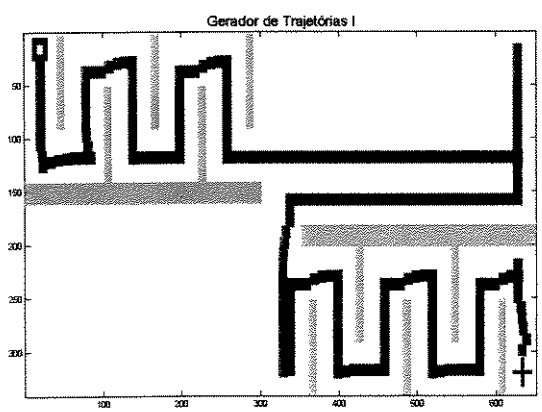


Figura 5.020 - 2 ziguezagues na horizontal.

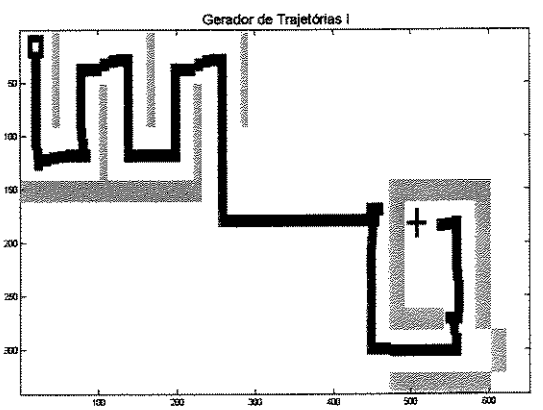


Figura 5.021 - Ziguezague e um côncavo.

5. 4. Resultados do Gerador de Trajetórias II

A seguir, serão mostrados os resultados finais das simulações do Gerador de Trajetórias II. No título das figuras foram descritas as direções preferenciais para cada um dos casos representados.

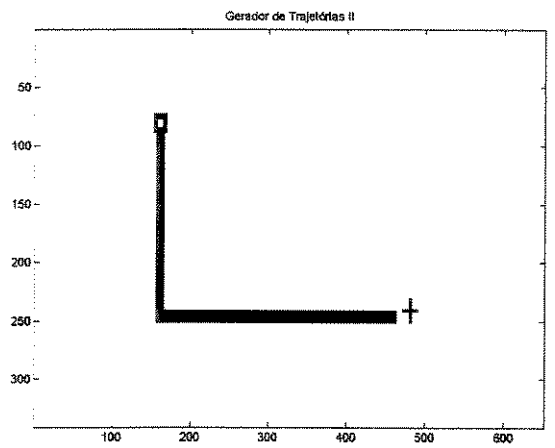


Figura 5.022 - DP Sul e Leste.

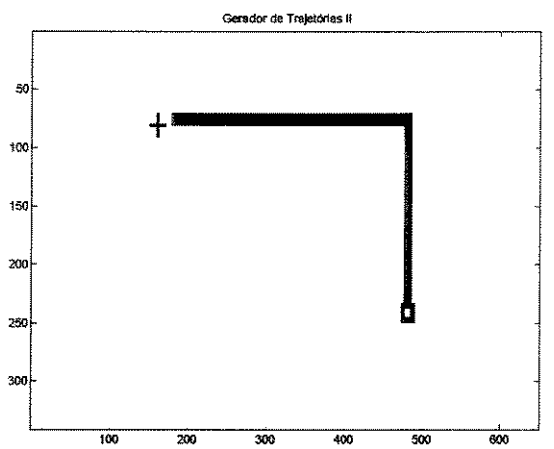


Figura 5.023 - DP Norte e Oeste.

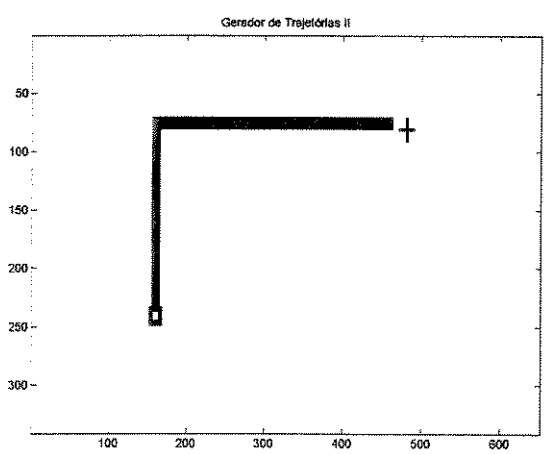


Figura 5.024 - DP Norte e Leste.

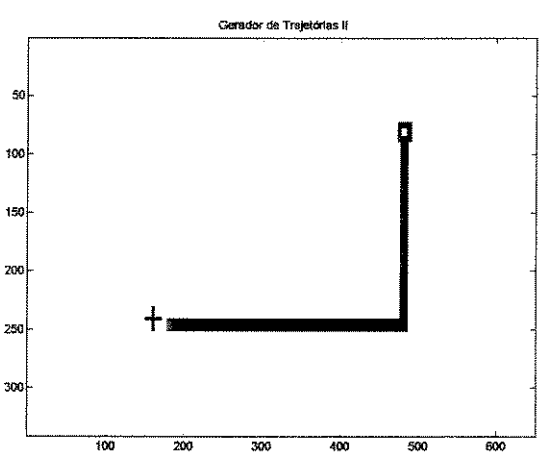


Figura 5.025 - DP Norte e Oeste.

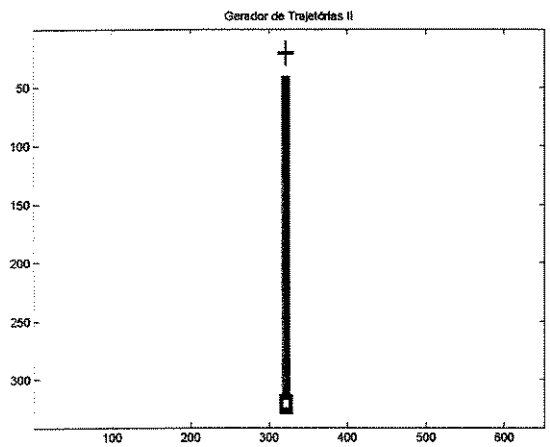


Figura 5.026 - DP Norte.

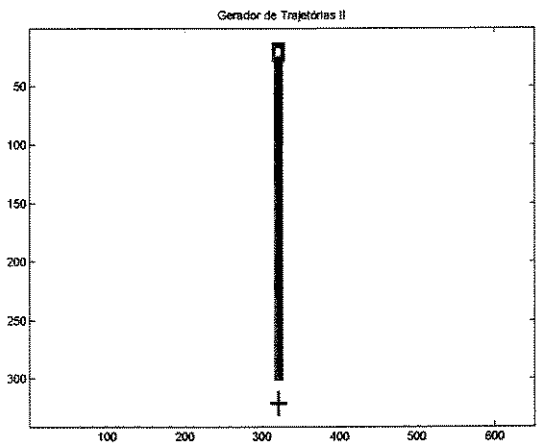


Figura 5.027 - DP Sul.

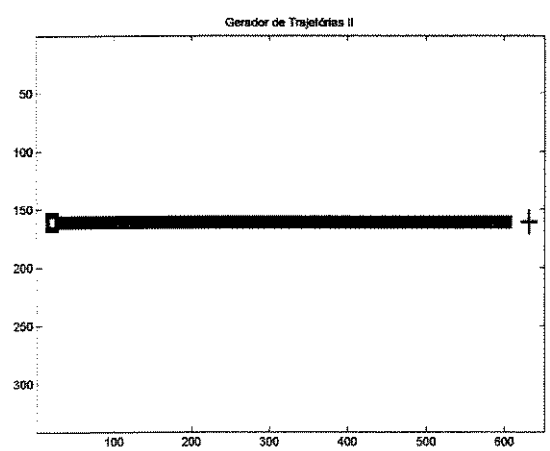


Figura 5.028 - DP Leste.

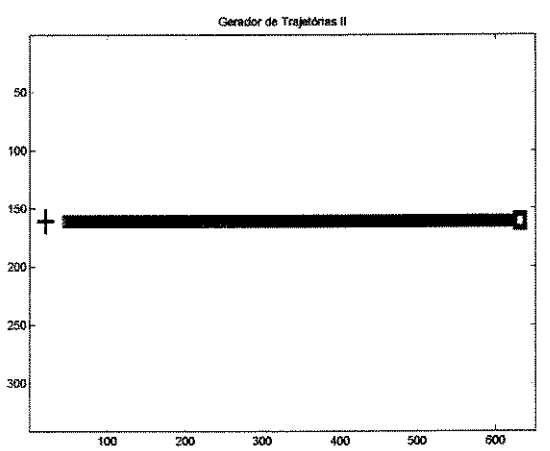


Figura 5.029 - DP Oeste.

No caso do ambiente com um obstáculo côncavo têm-se duas situações distintas, sendo que a primeira é a "entrada" neste tipo de obstáculo e a segunda é a "saída" do mesmo. Estas também estão descritas no título de cada figura.

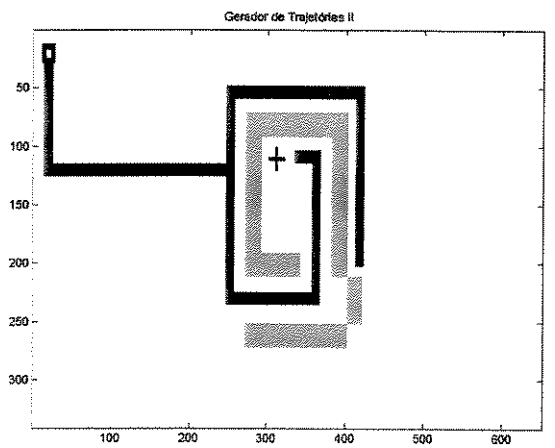


Figura 5.030 - DP Sul e Leste, entrada.

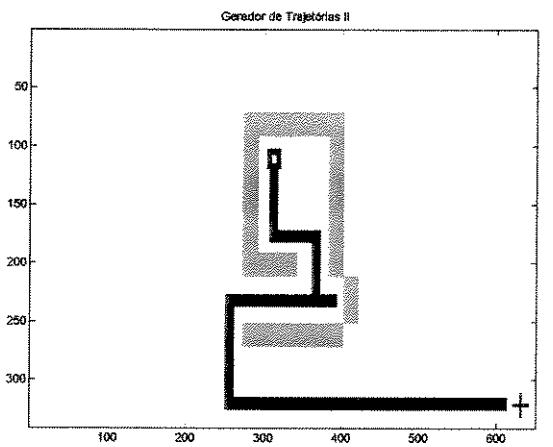


Figura 5.031 - DP Sul e Leste, saída.

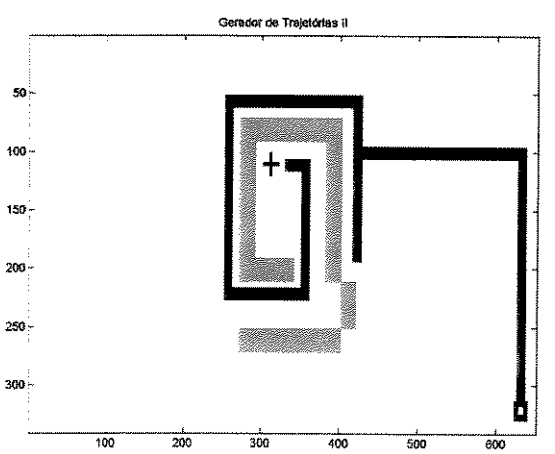


Figura 5.032 - DP Norte e Oeste, entrada.

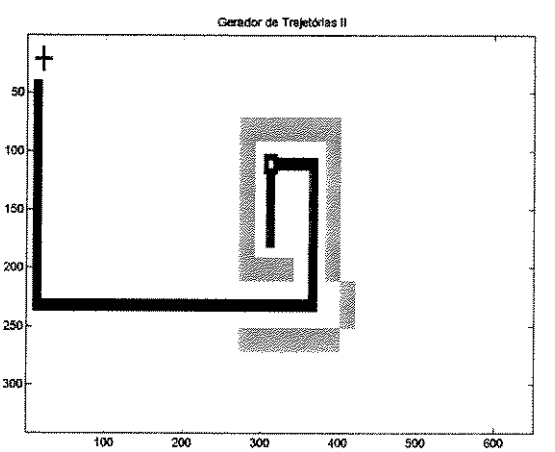


Figura 5.033 - DP Norte e Oeste, saída.

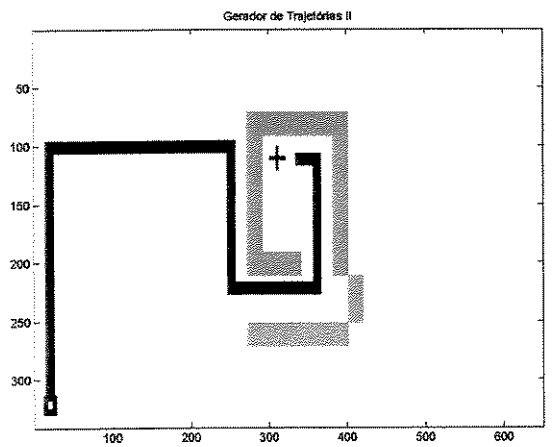


Figura 5.034 - DP Norte e Leste, entrada.

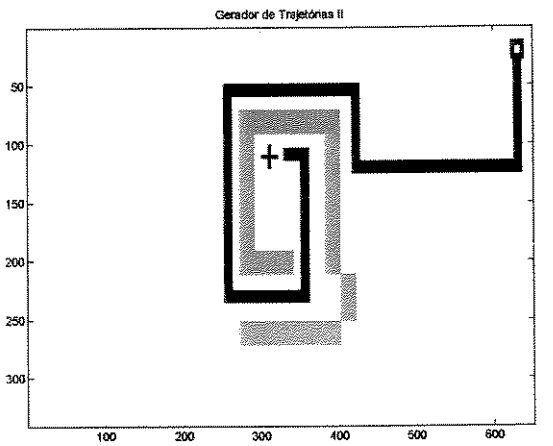


Figura 5.035 - DP Sul e Oeste, entrada.

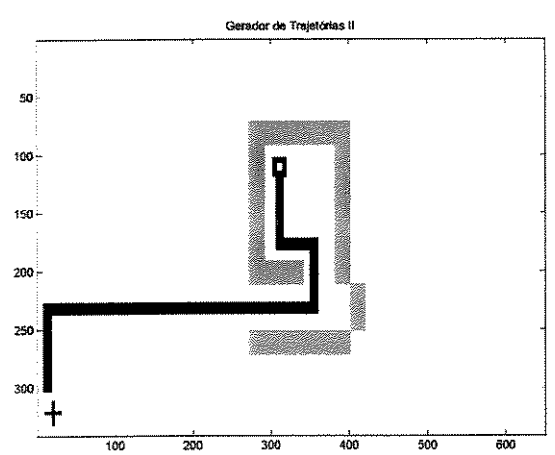


Figura 5.036 - DP Sul e Oeste, saída.

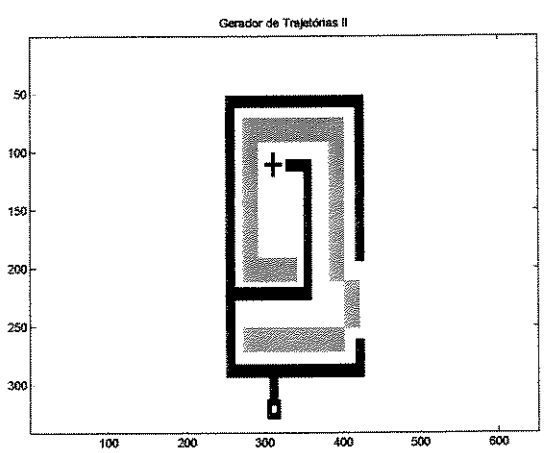


Figura 5.037 - DP Norte, entrada.

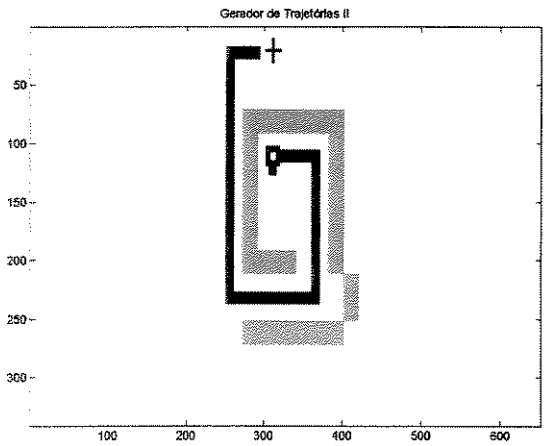


Figura 5.038 - DP Norte, saída.

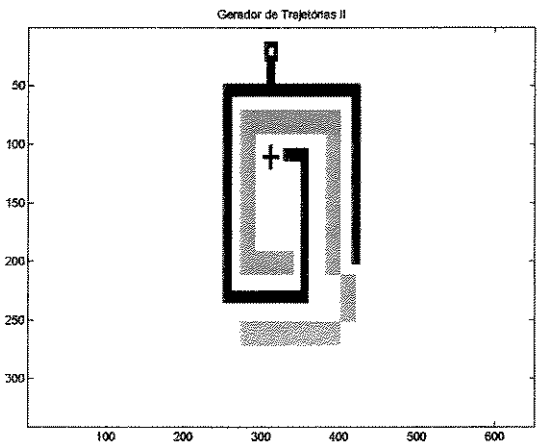


Figura 5.039 - DP Sul, entrada.

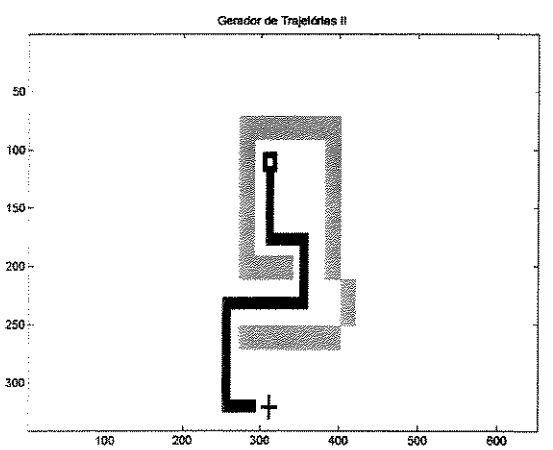


Figura 5.040 - DP Sul, saída.

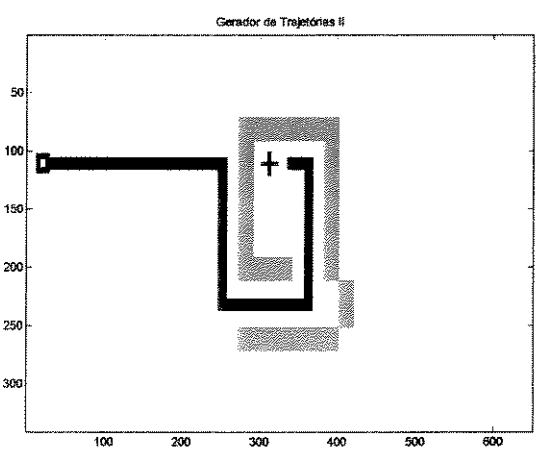


Figura 5.041 - DP Leste, entrada.

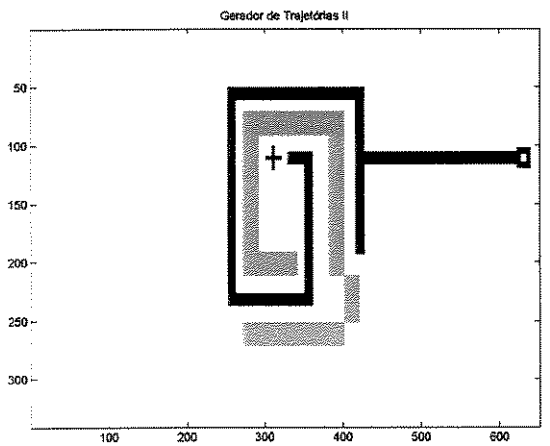


Figura 5.042 - DP Oeste, entrada.

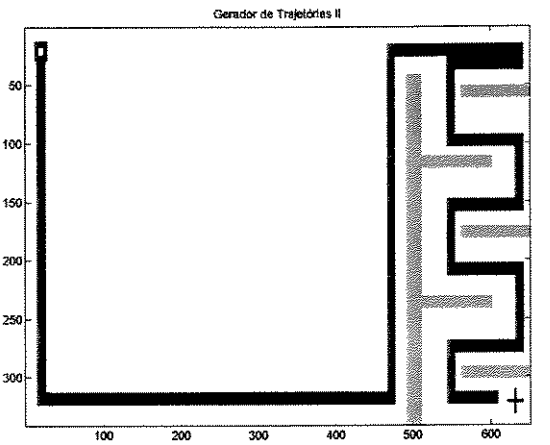


Figura 5.043 - DP Sul e Leste.

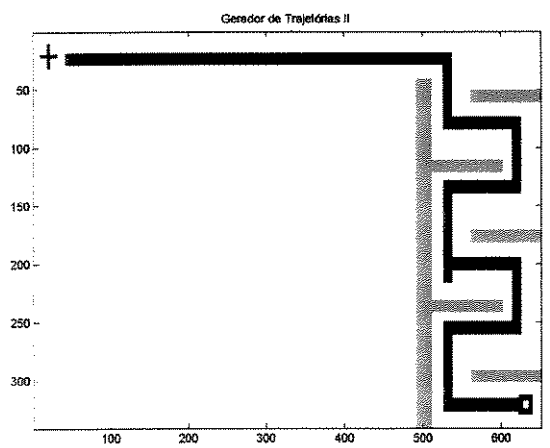


Figura 5.044 - DP Norte e Oeste 1.

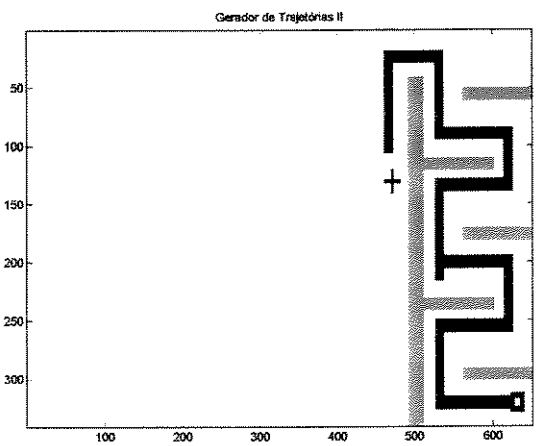


Figura 5.045 - DP Norte e Oeste 2.

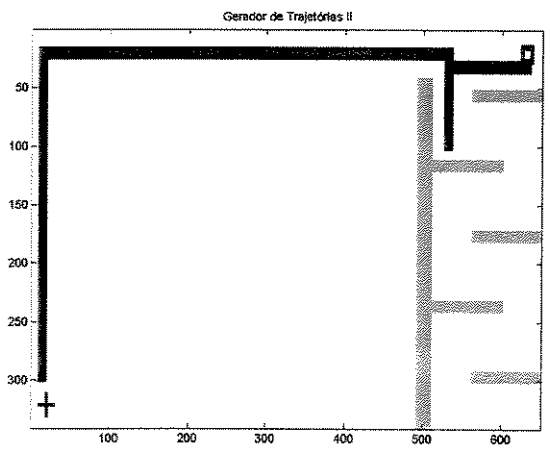


Figura 5.046 - DP Sul e Oeste.

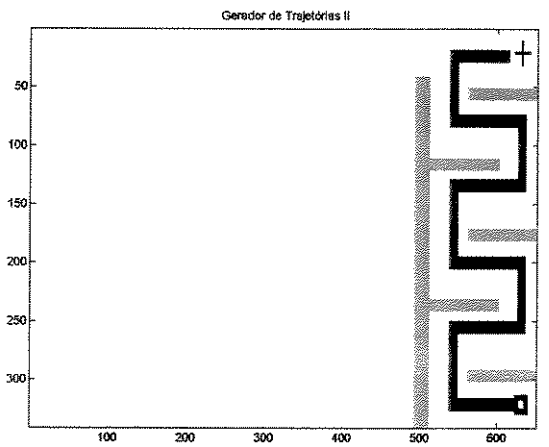


Figura 5.047 - DP Norte.

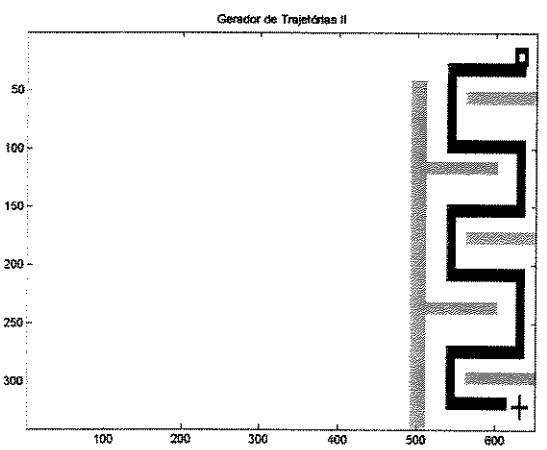


Figura 5.048 - DP Sul.

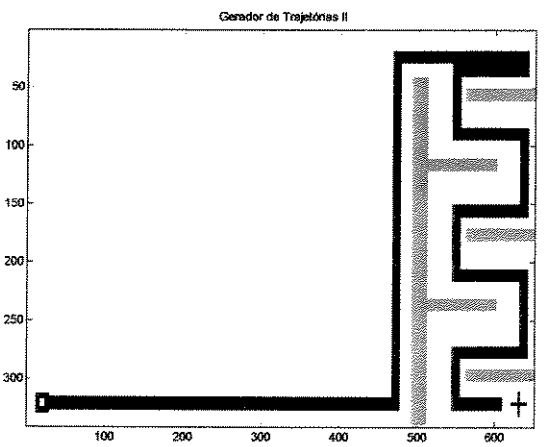


Figura 5.049 - DP Leste.

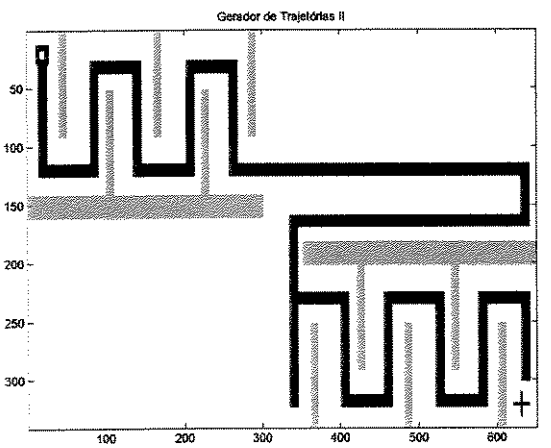


Figura 5.050 - DP Sul e Leste.

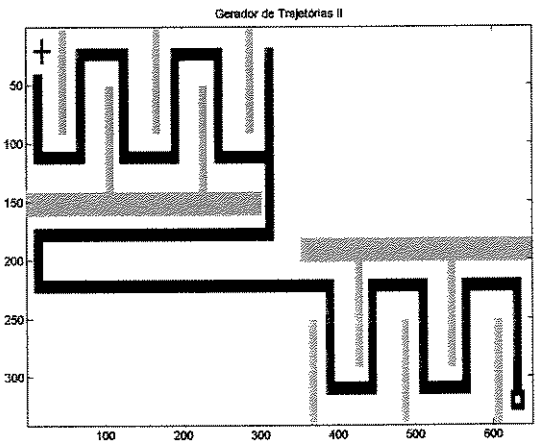


Figura 5.051 - DP Norte e Oeste.

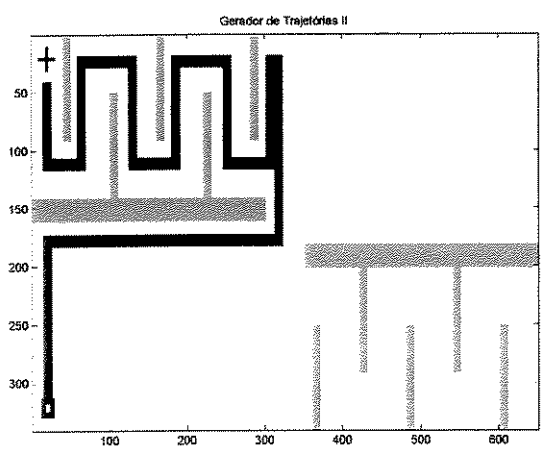


Figura 5.052 - DP Norte, entrada.

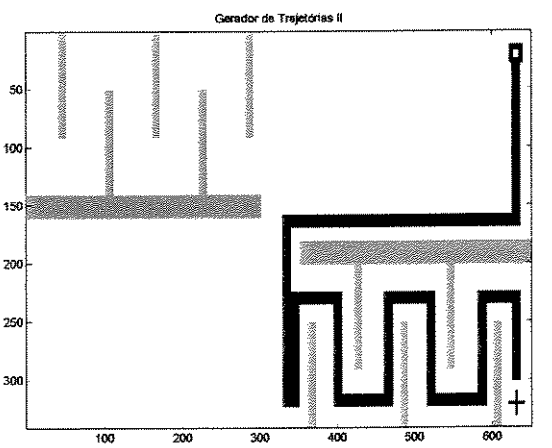


Figura 5.053 - DP Sul, entrada.

Da mesma forma que para o obstáculo côncavo, têm-se também para os dois ziguezagues na horizontal as situações de entrada e saída.

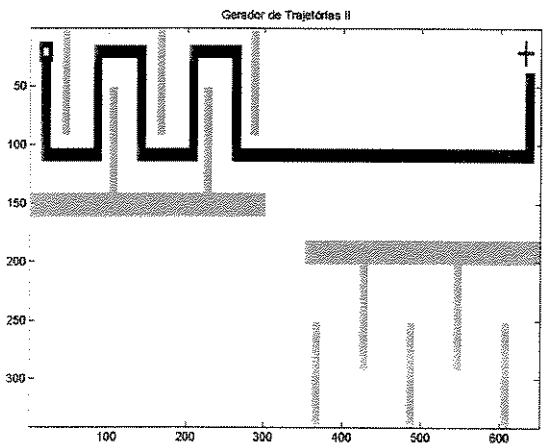


Figura 5.054 - DP Leste, saída.

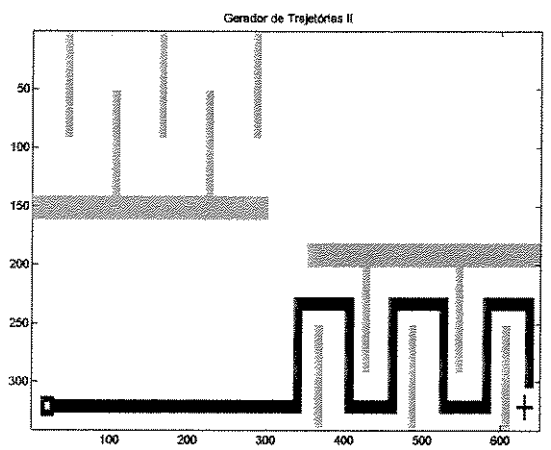


Figura 5.055 - DP Leste, entrada.

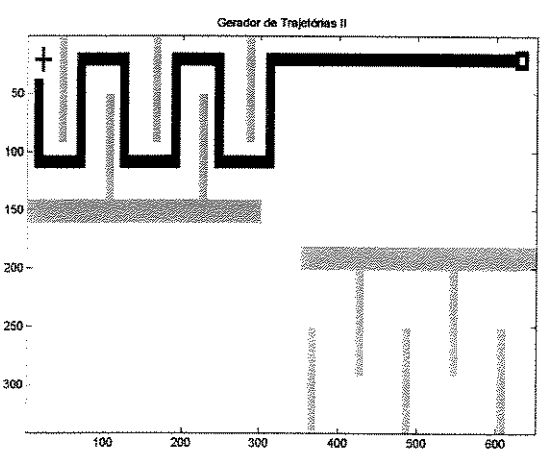


Figura 5.056 - DP Oeste, entrada.

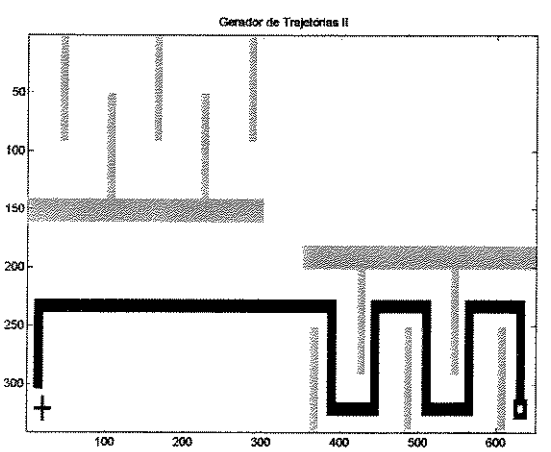


Figura 5.057 - DP Oeste, saída.

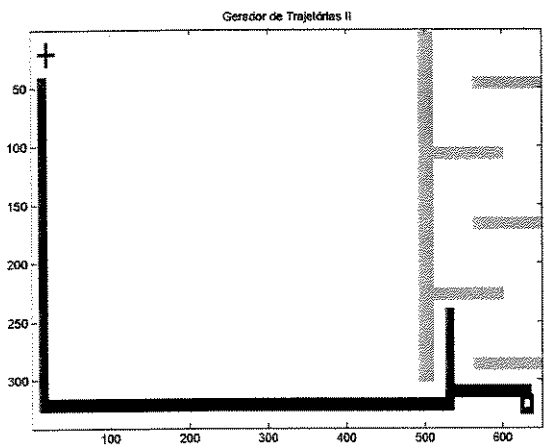


Figura 5.058 - DP Norte e Oeste.

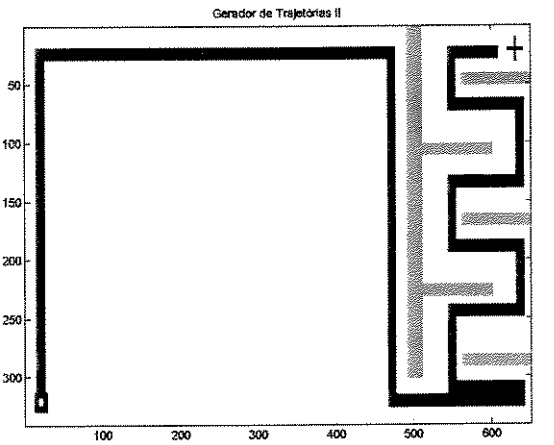


Figura 5.059 - DP Norte e Leste.

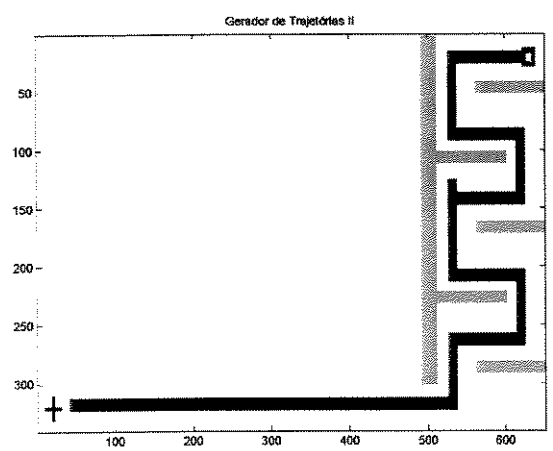


Figura 5.060 - DP Sul e Oeste.

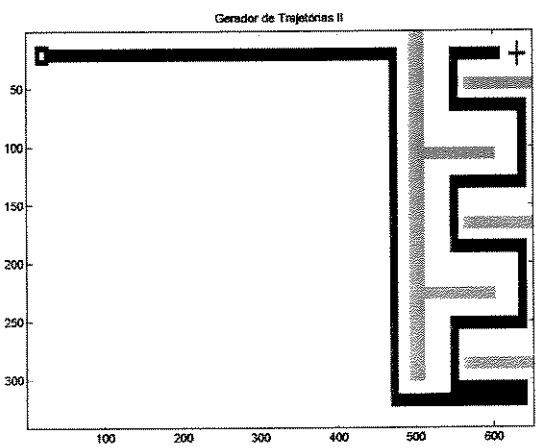


Figura 5.061 - DP Leste.

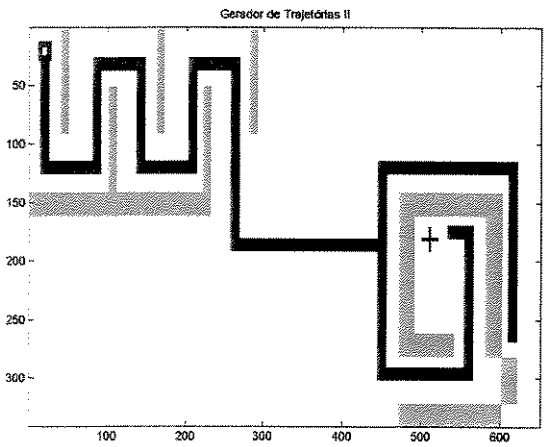


Figura 5.062 - DP Sul e Leste.

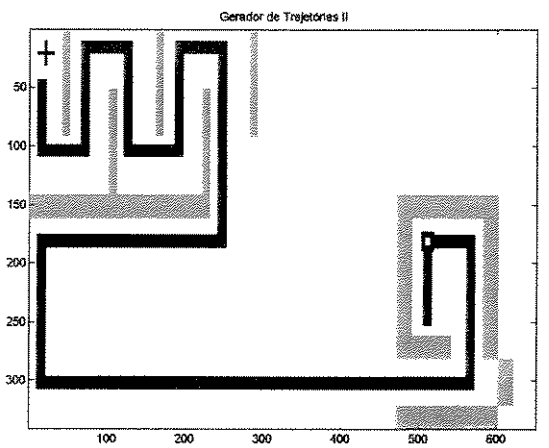


Figura 5.063 - DP Norte e Oeste.

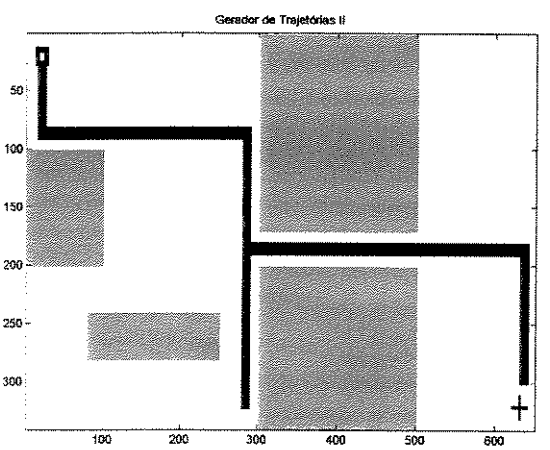


Figura 5.064 - DP Sul e Leste.

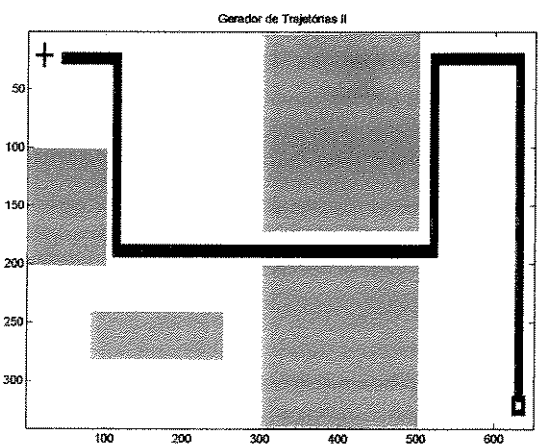


Figura 5.065 - DP Norte e Oeste.

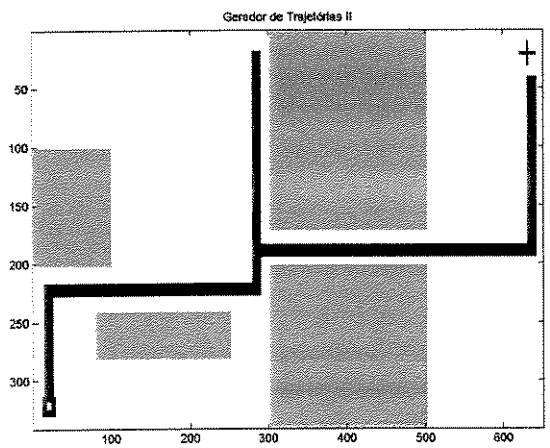


Figura 5.066 - DP Norte e Leste.

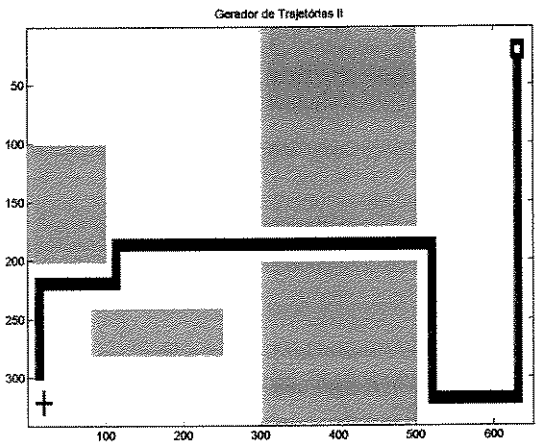


Figura 5.067 - DP Sul e Oeste.

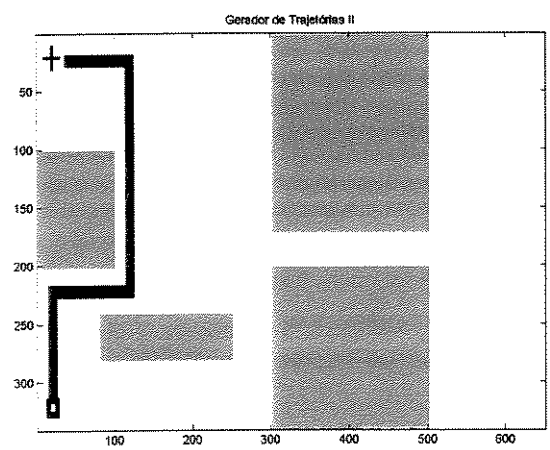


Figura 5.068 - DP Norte.

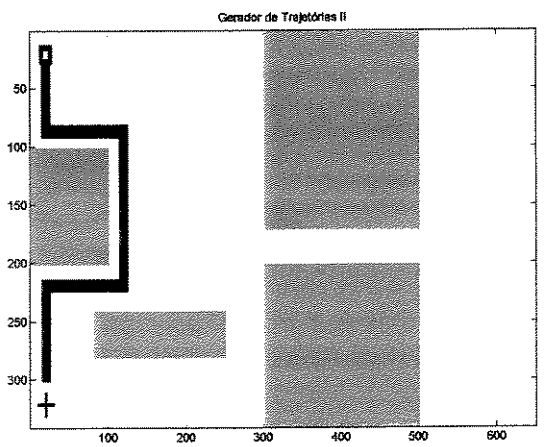


Figura 5.069 - DP Sul.

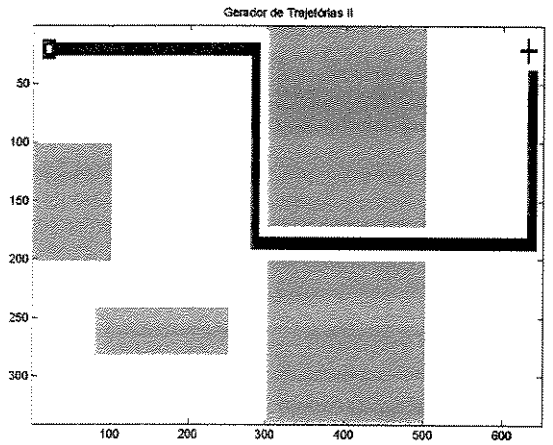


Figura 5.070 - DP Leste.

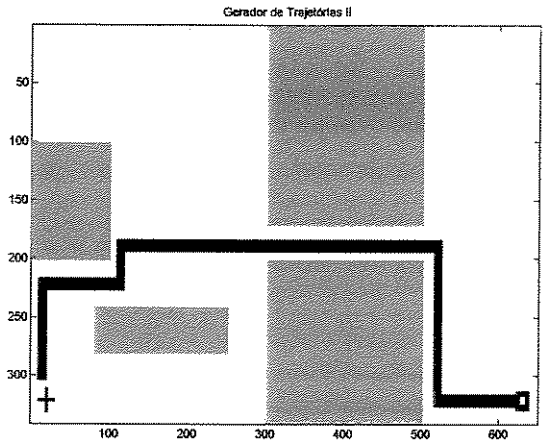


Figura 5.071 - DP Oeste.

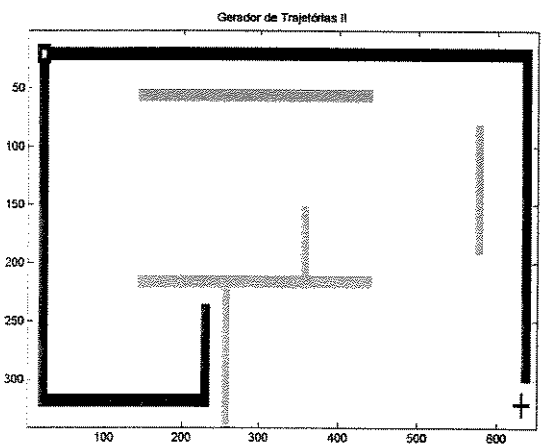


Figura 5.072 - DP Sul e Leste.

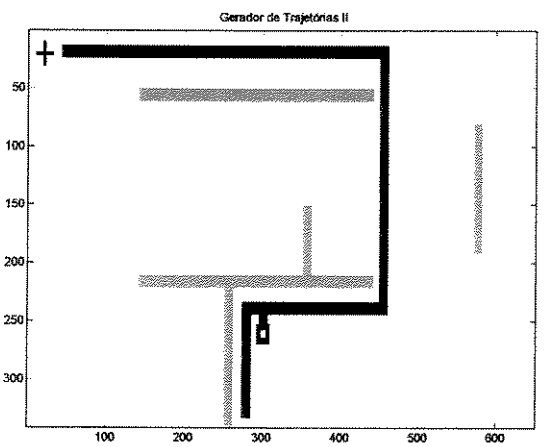


Figura 5.073 - DP Norte e Oeste.

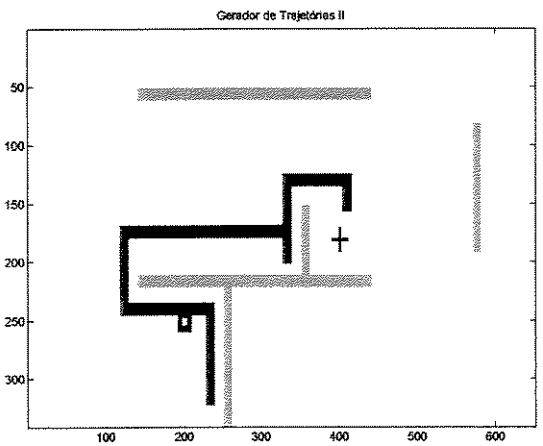


Figura 5.074 - DP Norte e Leste.

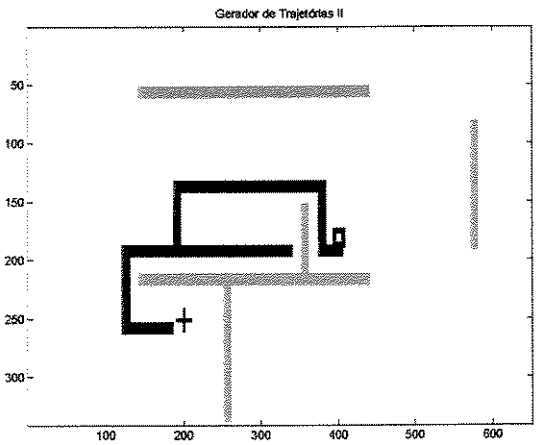


Figura 5.075 - DP Sul e Oeste.

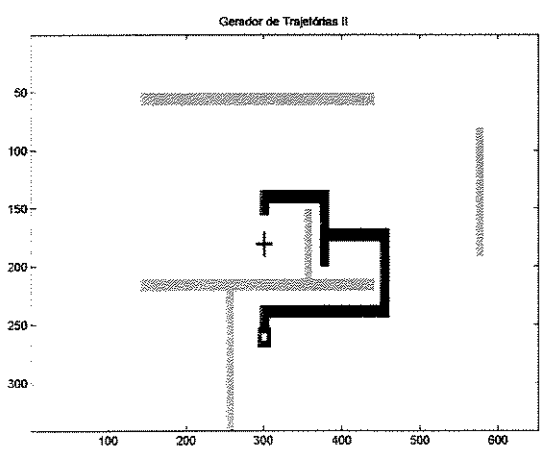


Figura 5.076 - DP Sul e Leste.

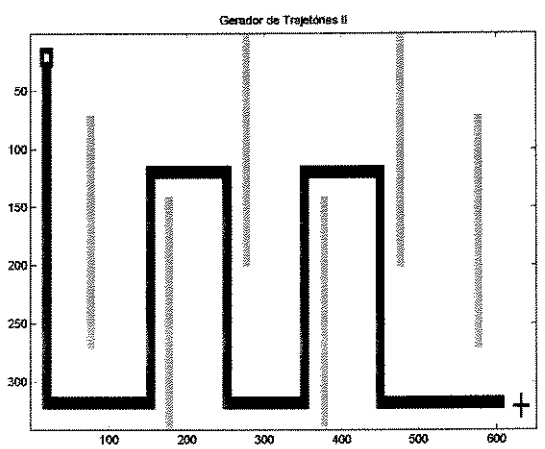


Figura 5.077 - DP Sul e Leste.

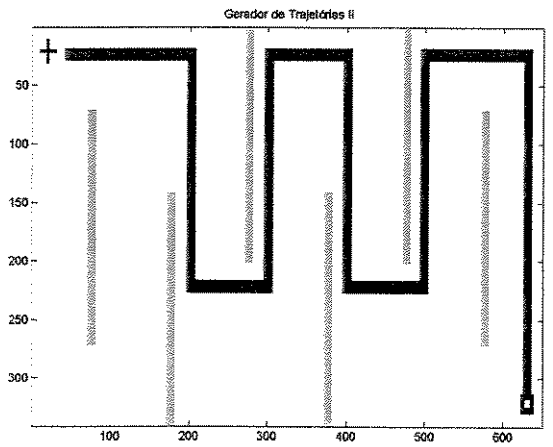


Figura 5.078 - DP Norte e Oeste.

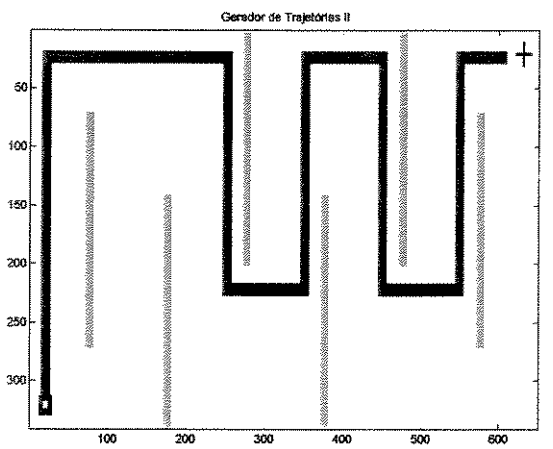


Figura 5.079 - DP Norte e Leste.

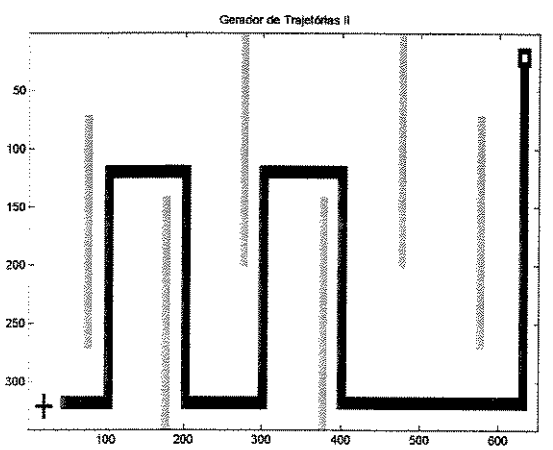


Figura 5.080 - DP Sul e Oeste.

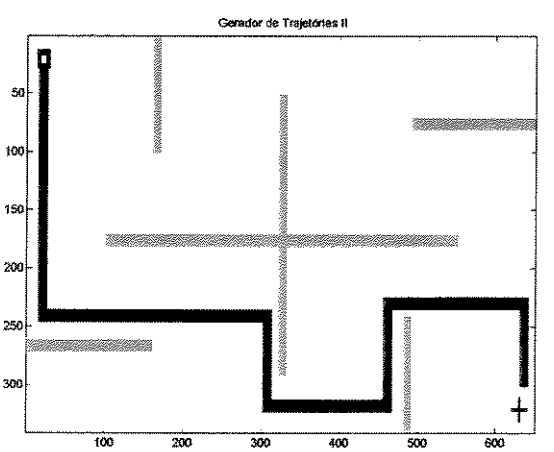


Figura 5.081 - DP Sul e Leste.

Para o ambiente do tipo Estrela I foram testada duas possibilidades de posicionamento dos pontos inicial e final em relação à estrela, sendo a primeira com os pontos colocados mais afastados do centro da estrela e a segunda com os mesmos mais próximos do centro da estrela.

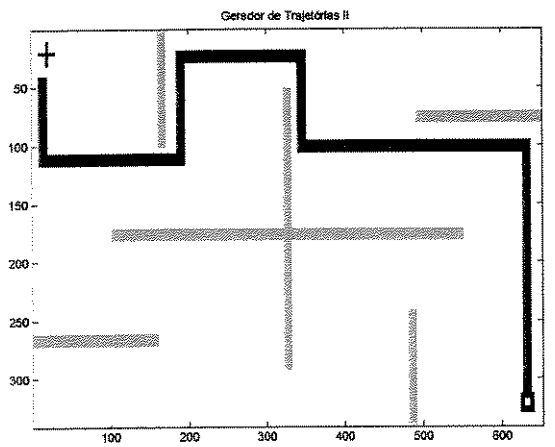


Figura 5.082 - DP Norte e Oeste.

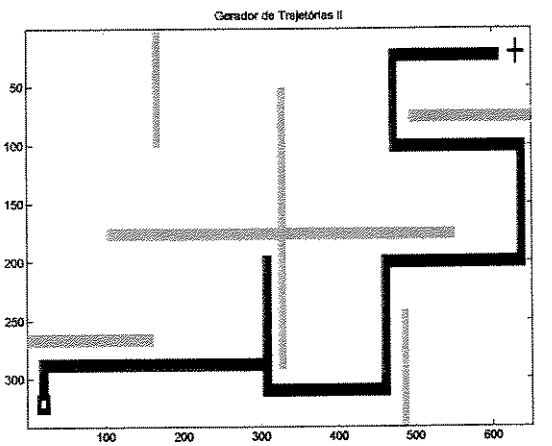


Figura 5.083 - DP Norte e Leste.

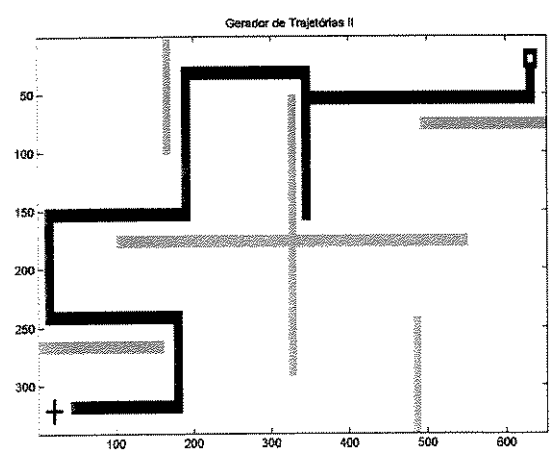


Figura 5.084 - DP Sul e Oeste.

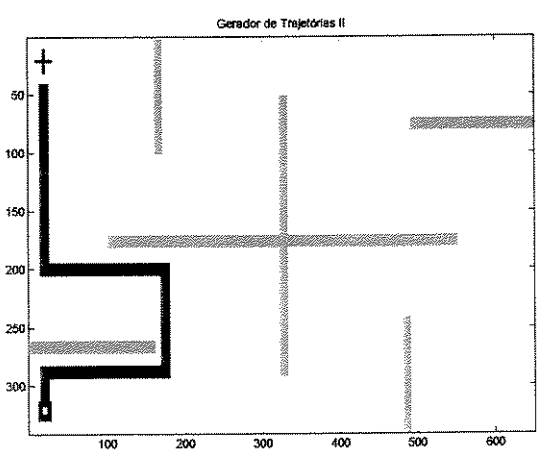


Figura 5.085 - DP Norte.

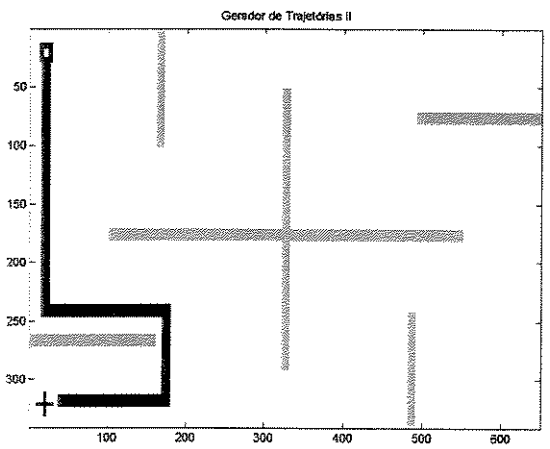


Figura 5.086 - DP Sul.

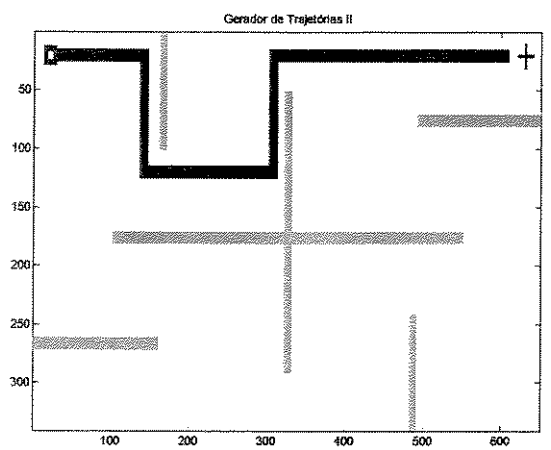


Figura 5.087 - DP Leste.

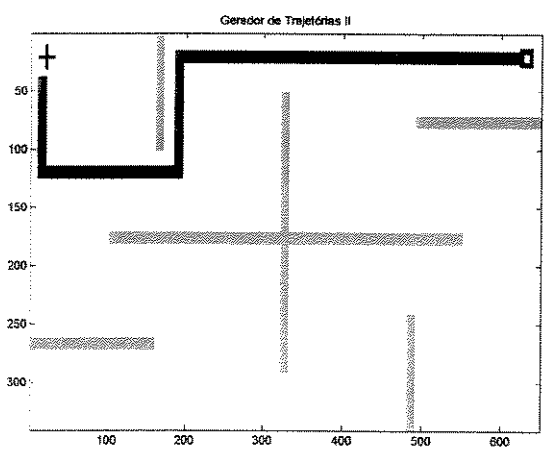


Figura 5.088 - DP Oeste.

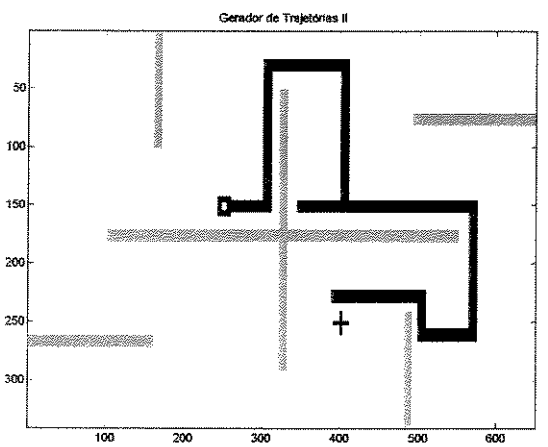


Figura 5.089 - DP Sul e Leste.

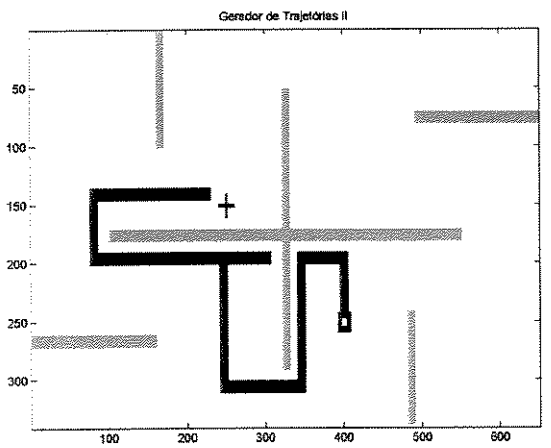


Figura 5.090 - DP Norte e Oeste.

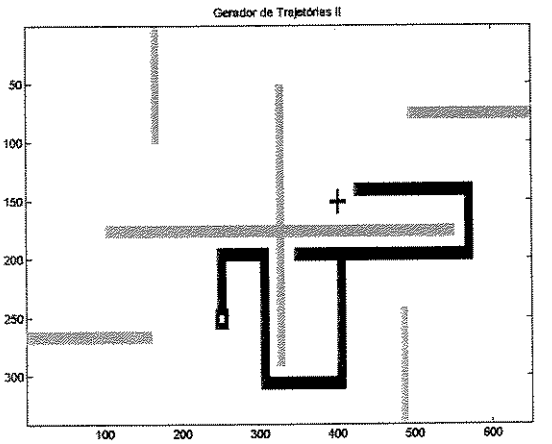


Figura 5.091 - DP Norte e Leste

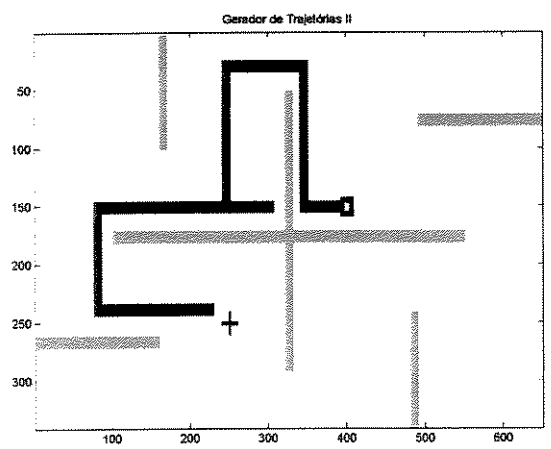


Figura 5.092 - DP Sul e Oeste.

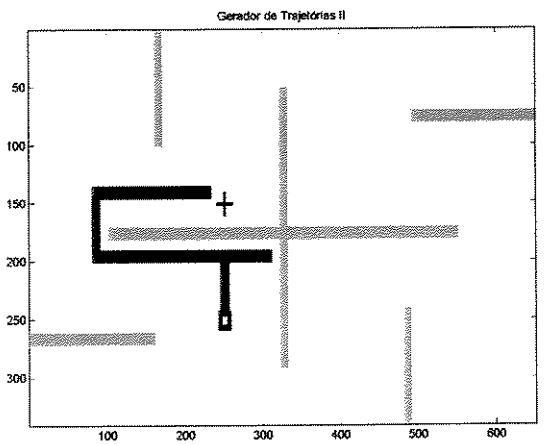


Figura 5.093 - DP Norte, região esquerda.

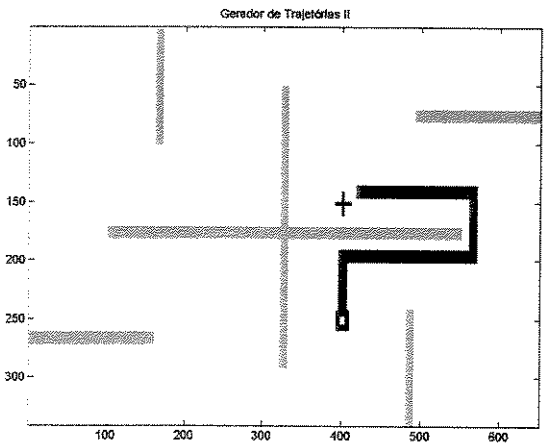


Figura 5.094 - DP Norte, região direita.

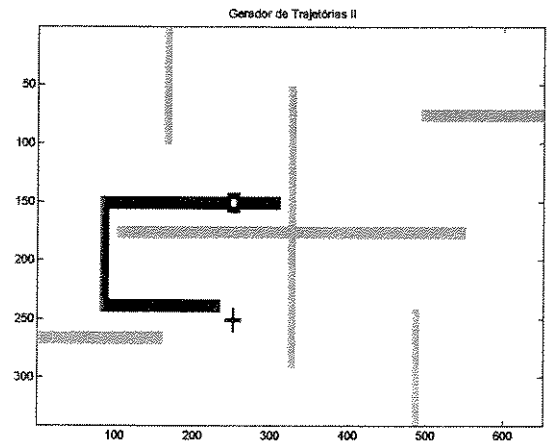


Figura 5.095 - DP Sul, região esquerda.

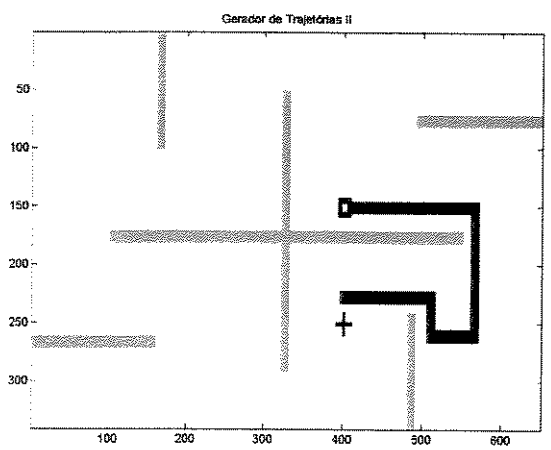


Figura 5.096 - DP Sul, região direita.

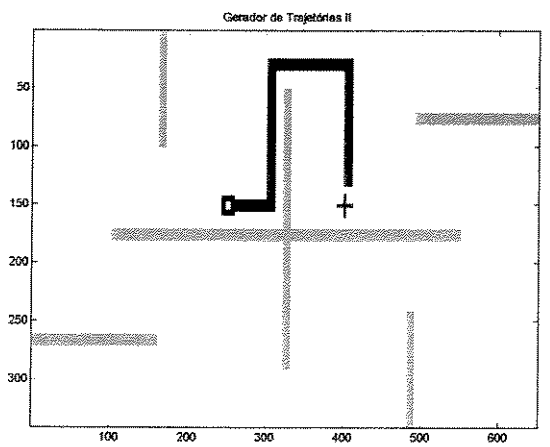


Figura 5.097 - DP Leste, região superior.

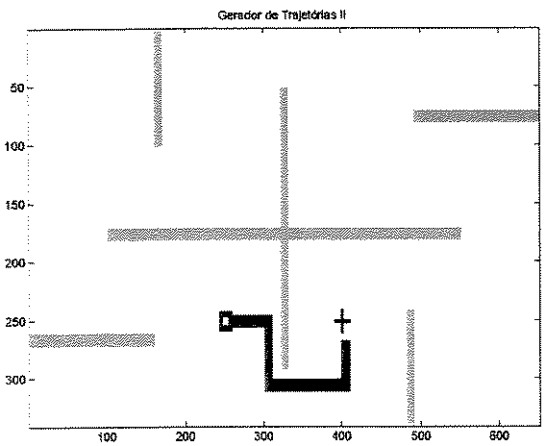


Figura 5.098 - DP Leste, região inferior.

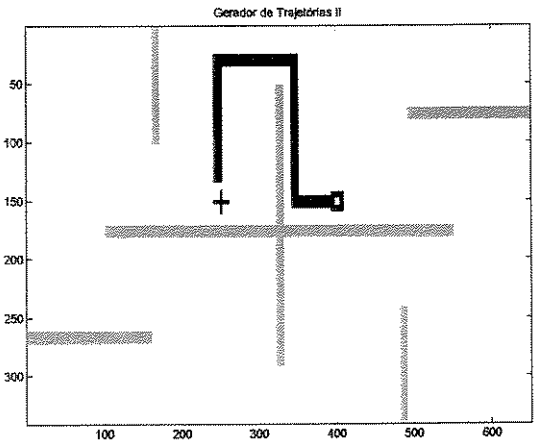


Figura 5.099 - DP Oeste, região superior.

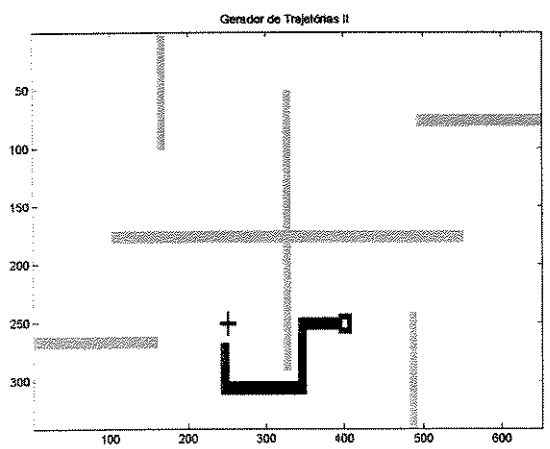


Figura 5.100 - DP Oeste, região inferior.

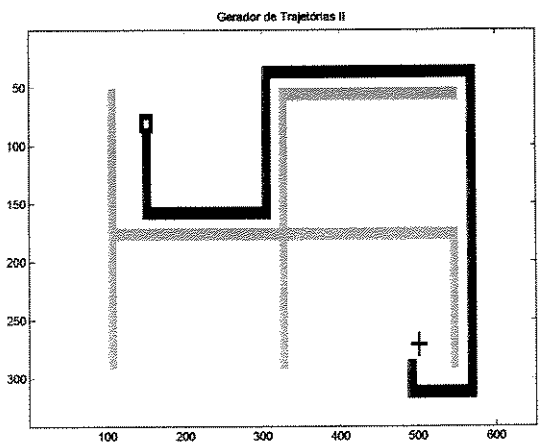


Figura 5.101 - DP Sul e Leste.

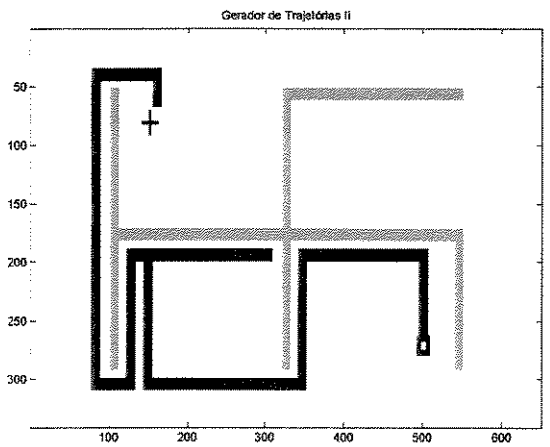


Figura 5.102 - DP Norte e Oeste.

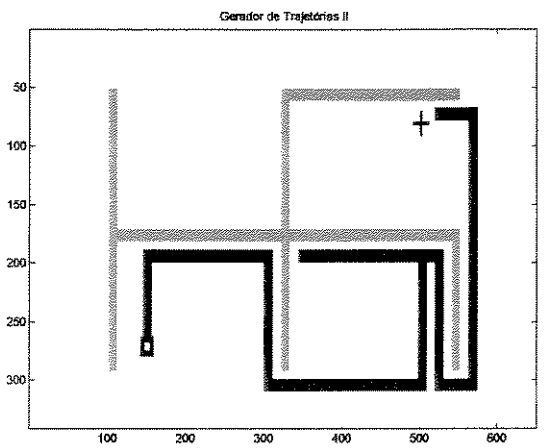


Figura 5.103 - DP Norte e Leste.

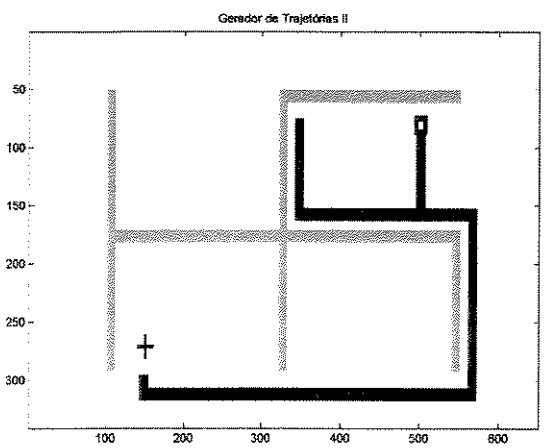


Figura 5.104 - DP Sul e Oeste.

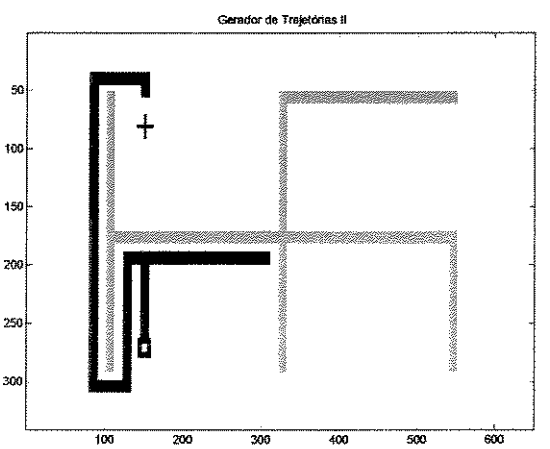


Figura 5.105 - DP Norte, região esquerda.

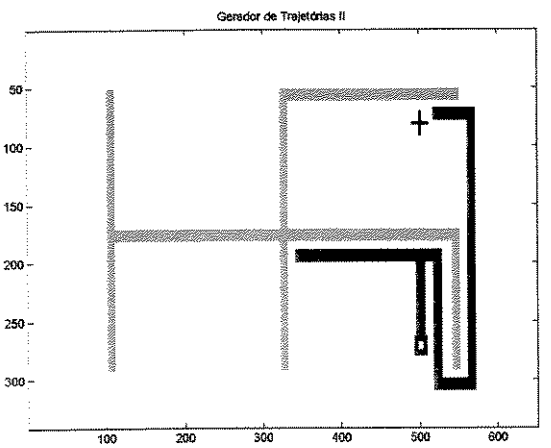


Figura 5.106 - DP Norte, região direita.

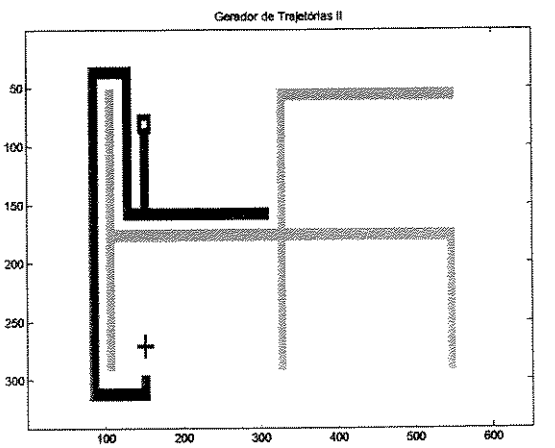


Figura 5.107 - DP Sul, região esquerda.

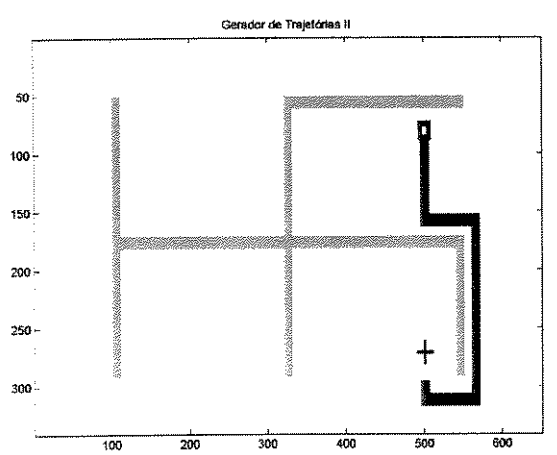


Figura 5.108 - DP Sul, região direita.

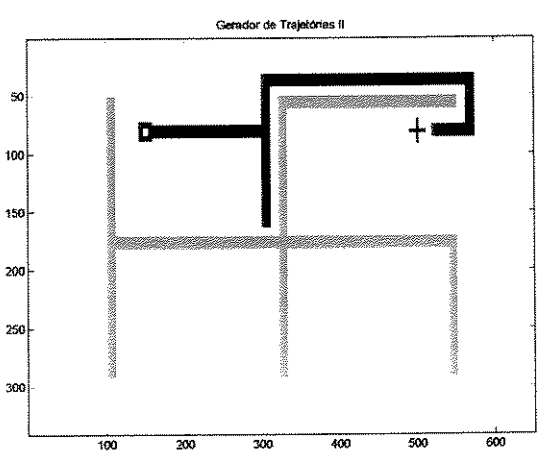


Figura 5.109 - DP Leste, região superior.

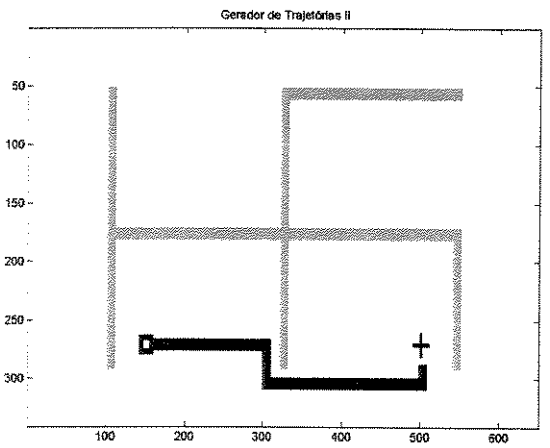


Figura 5.110 - DP Leste, região inferior.

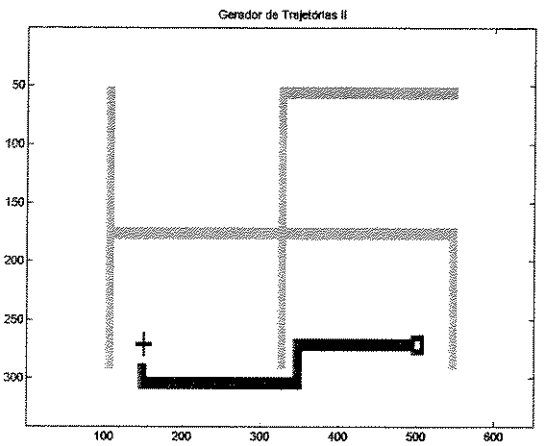


Figura 5.111 - DP Oeste, região inferior.

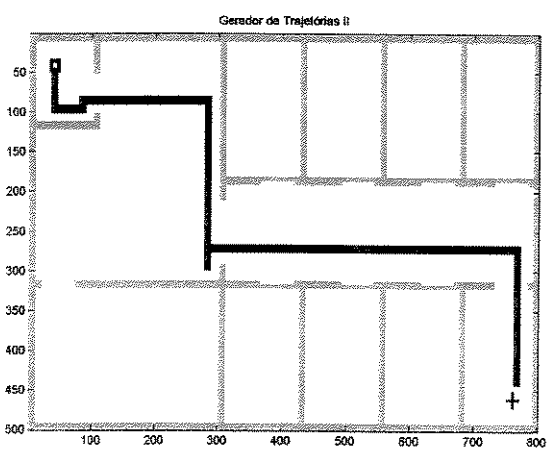


Figura 5.112 - DP Sul e Leste.

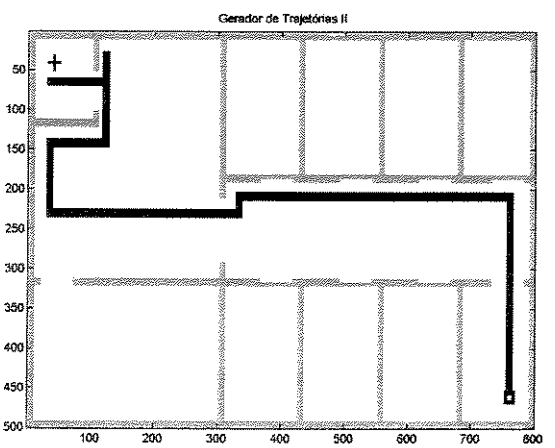


Figura 5.113 - DP Norte e Oeste.

Para a Planta Baixa foram alteradas as dimensões do ambiente para 500x800 *pixels*.

Neste ambiente, também foi necessário um controle das portas das salas, sendo deixadas abertas apenas as portas das duas salas que correspondiam à origem e ao objetivo do cursor. As demais portas deveriam ficar fechadas.

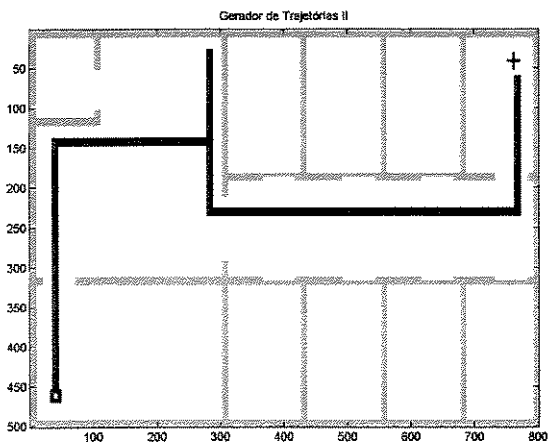


Figura 5.114 - DP Norte e Leste.

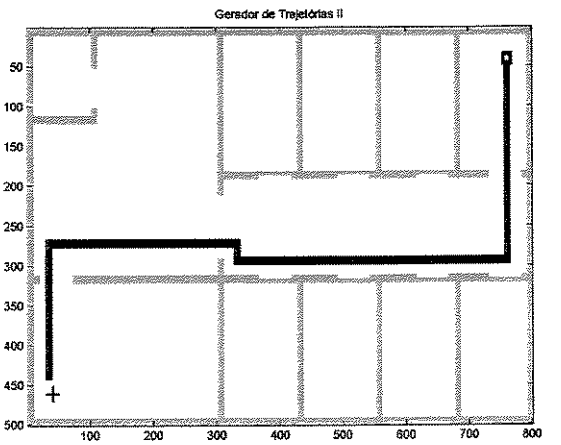


Figura 5.115 - DP Sul e Oeste.

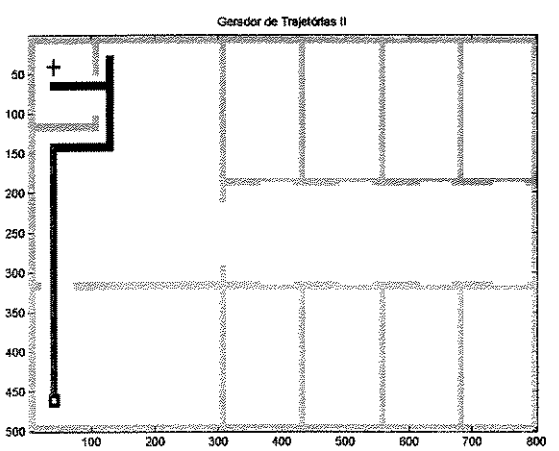


Figura 5.116 - DP Sul e Leste.

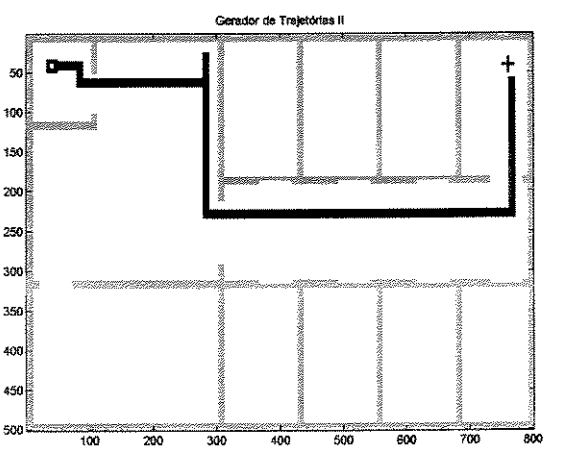


Figura 5.117 - DP Norte e Oeste.

No caso do ambiente com obstáculo móvel, tem-se no título de cada figura o número da iteração do programa correspondente ao gráfico em questão. Na figura 5.121 representando a iteração 30 do programa nota-se que o cursor já se evadiu com segurança do obstáculo móvel.

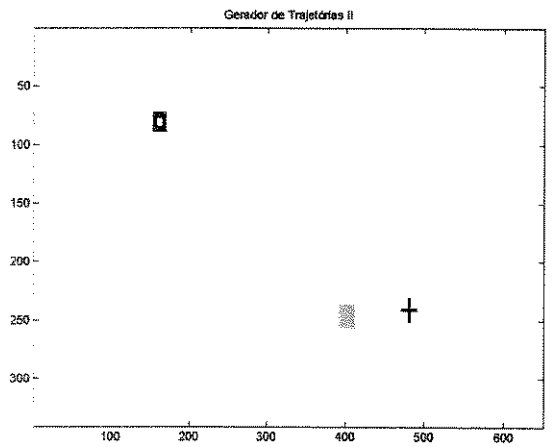


Figura 5.118 - Iteração 01.

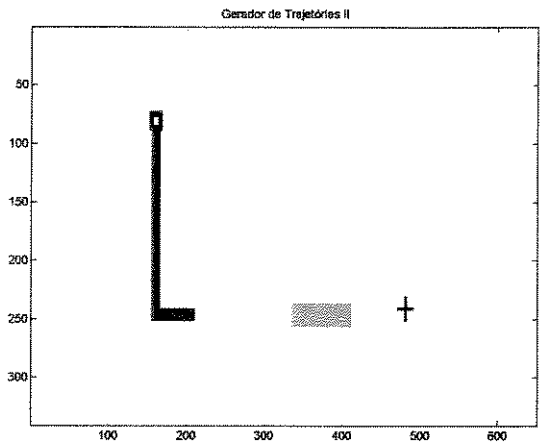


Figura 5.119 - Iteração 21.

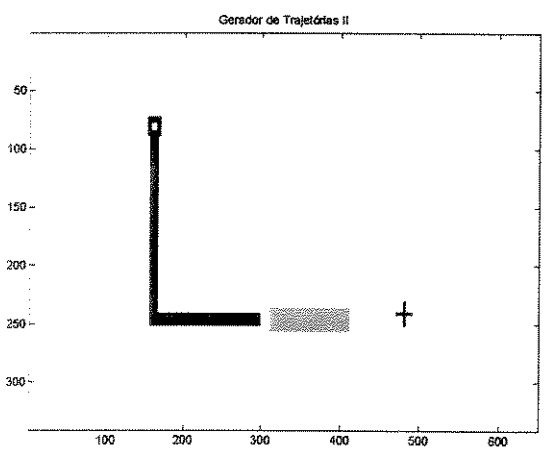


Figura 5.120 - Iteração 28.

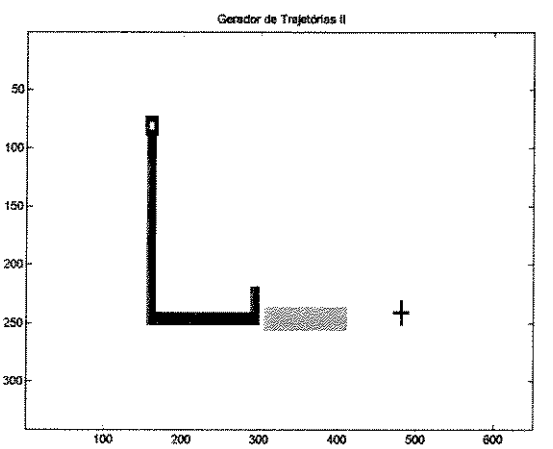


Figura 5.121 - Iteração 30.

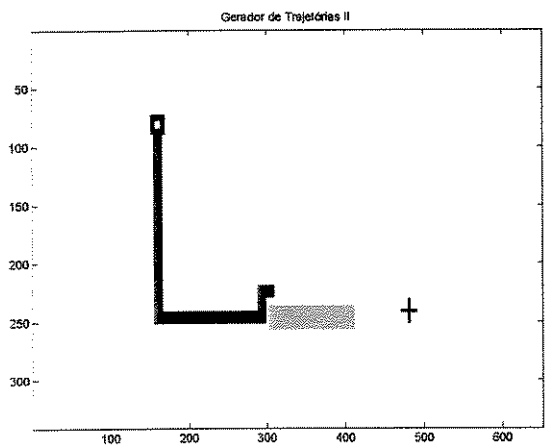


Figura 5.122 - Iteração 31.

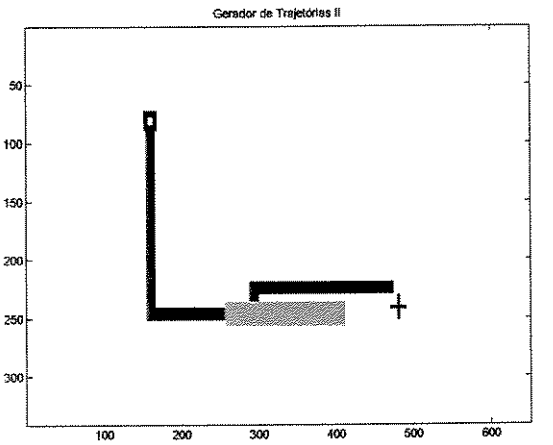


Figura 5.123 - Iteração 46.

5. 5. Resultados Insatisfatórios do Gerador de Trajetórias II

O sistema não foi satisfatório para alguns casos, a depender das posições dos pontos inicial e final adotados e da configuração do ambiente. Principalmente na presença de obstáculos em "U", e para as situações em que a posição alvo está localizada em uma região oposta e nas proximidades do obstáculo, como se o cursor fosse obrigado a transpor o mesmo.

A seguir, serão mostrados exemplos de resultados insatisfatórios das simulações do Gerador de Trajetórias II. No título das figuras foram descritas as direções preferenciais, relacionadas com os alvos que não foram atingidos, para cada um dos casos representados.

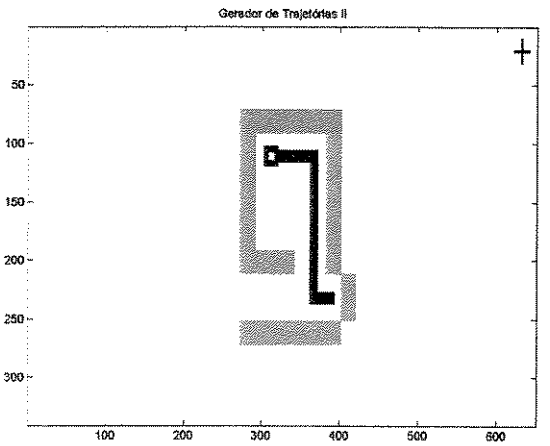


Figura 5.124 - DP Norte e Leste.

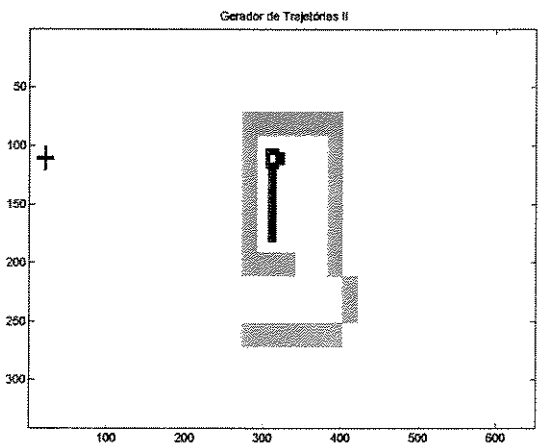


Figura 5.125 - DP Oeste.

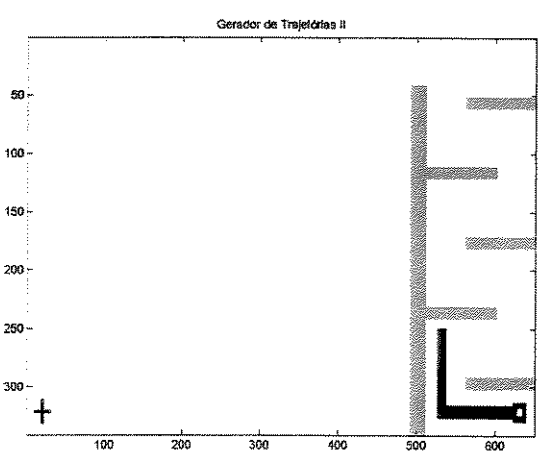


Figura 5.126 - DP Oeste.

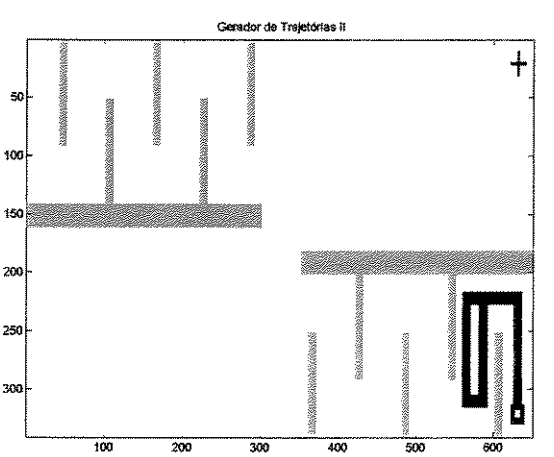


Figura 5.127 - DP Norte.

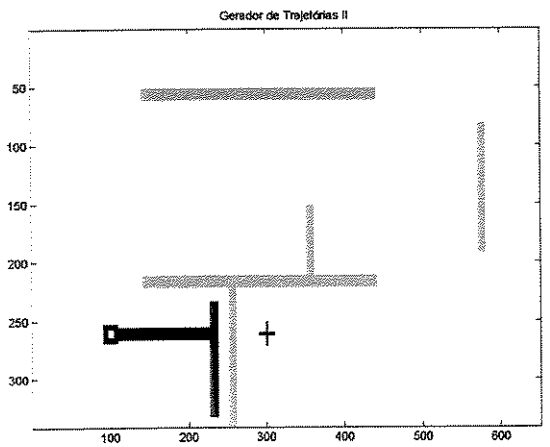


Figura 5.128 - DP Leste.

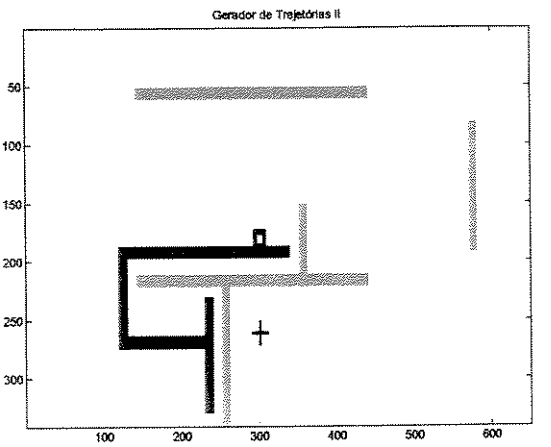


Figura 5.129 - DP Sul.

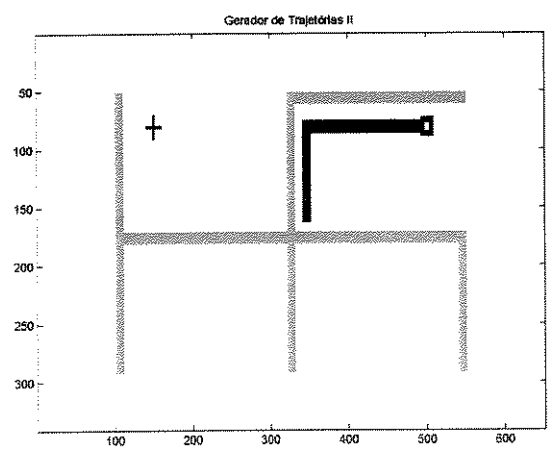


Figura 5.130 - DP Oeste.

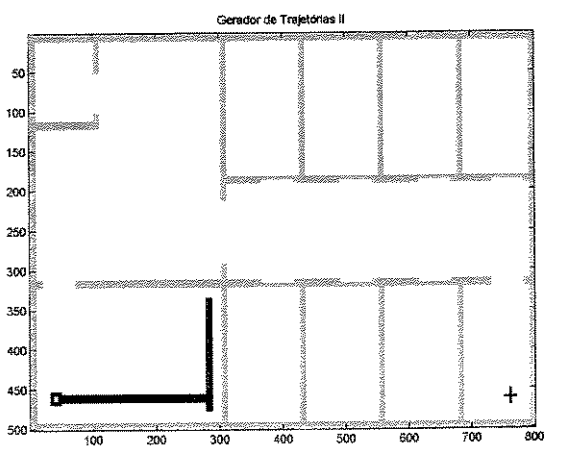


Figura 5.131 - DP Leste.

5. 6. Possíveis Modificações no Gerador de Trajetórias II

Foram adicionadas ao programa algumas modificações com o objetivo de testar o sistema proposto para outras condições não previstas. Algumas destas modificações têm por objetivo futuro a adaptação do sistema proposto para os demais níveis de aprendizado da estrutura neural de A. Mendeleck (Mendeleck, 1995), que foram descritos no capítulo 2. São elas:

- A centralização do cursor (Mendeleck, 1995);
- O avanço do cursor com passo variável (Mendelck, 1995);
- A modificação do tipo de sensor utilizado, sendo testado o uso de sensores de toque;
- O uso da chamada Trajetória Final.

5. 6.1. A centralização do cursor

Com o objetivo de aumentar a área mapeada por seu sistema, A. Mendeleck (Mendeleck, 1995) adotou o procedimento de centralizar o cursor a cada nova posição. Desta forma, durante o processo de aprendizado, o cursor executa movimentos pela área de trabalho mantendo a equidistância (equilíbrio de valores) na direção ortogonal à do sentido do movimento. A figura 5.132 representa o cursor nas situações de não centralizado e de centralizado, destacando a maior circunferência mapeada para o caso centralizado.

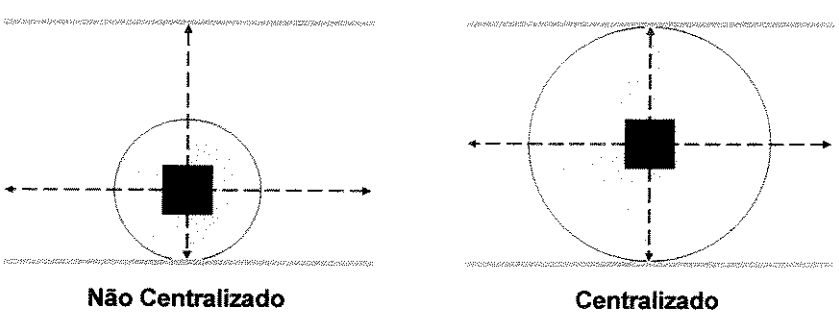


Figura 5.132 - A centralização do cursor.

Outra consequência da utilização do procedimento de centralização do cursor é o fato do mesmo se afastar mais ainda dos obstáculos, além da distância considerada segura que é proveniente da relação entre o range dos sensores e do avanço (fixo) do cursor.

Como exemplos desta aplicação, têm-se todos os resultados das simulações do Gerador de Trajetórias I (figuras 5.016 à 5.021) que apresentam o procedimento de centralização do cursor. No caso do Gerador de Trajetórias II têm-se os exemplos mostrados nas figuras 5.133 e 5.134.

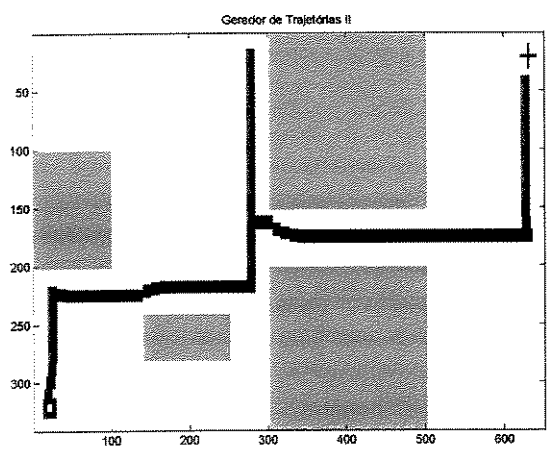


Figura 5.133 - DP Norte e Leste.

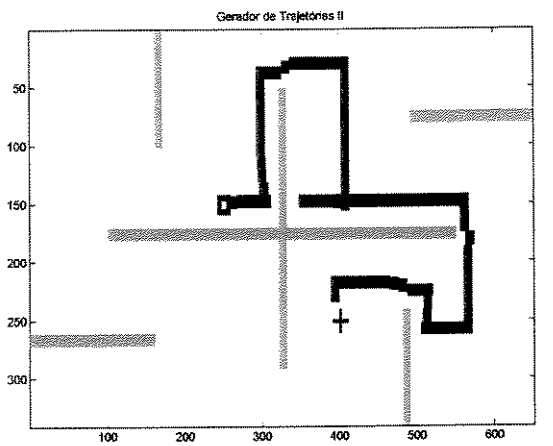


Figura 5.134 - DP Sul e Leste.

5. 6.2. O avanço do cursor com passo variável

Uma alternativa ao avanço do cursor é torná-lo variável. Neste caso, o avanço é determinado, a cada nova posição, pela menor distância medida pelos sensores. Isto faz com que o cursor se movimente mais rápido em regiões livres de obstáculos e com uma velocidade menor na presença dos mesmos, aumentando a segurança nas manobras do cursor, em relação as colisões.

As figuras 5.135 e 5.136 são exemplos do uso do avanço variável.

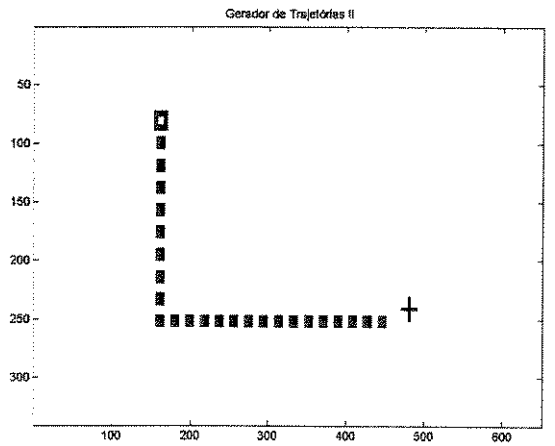


Figura 5.135 - DP Sul e Leste.

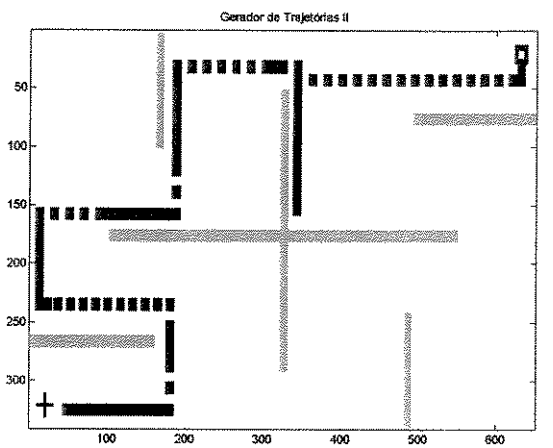


Figura 5.136 - DP Norte e Oeste.

5. 6.3. O uso de sensores de toque

Com o objetivo de testar o sistema proposto para outros tipos de sensores, foi substituída a abordagem com sensores de distância para o uso de sensores de toque, ou seja, sensores de contato. Para manter uma distância segura dos obstáculos, foi necessário o uso de hastes de segurança, que funcionam como um prolongamento destes sensores.

A figura 5.137 representa o cursor com os sensores de toque, sendo necessário para este caso a utilização de apenas 4 sensores, com as suas hastes posicionadas nas direções das diagonais do cursor, ou seja, 45° com a horizontal. Na figura 5.138, tem-se a representação do movimento do cursor com as hastes dos sensores de toque formando uma região de segurança em torno do cursor. Dois exemplos do resultado final para esta abordagem estão representados nas figuras 5.139 e 5.140.

A grande desvantagem desta abordagem é a necessidade de um avanço muito pequeno, no caso foi usado um avanço unitário, para garantir a segurança das hastes.

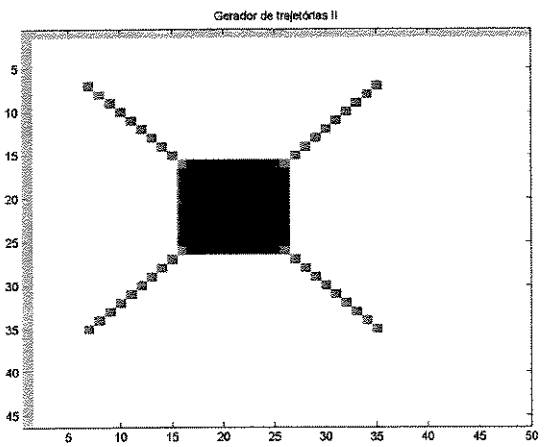


Figura 5.137 - O cursor com as hastes.

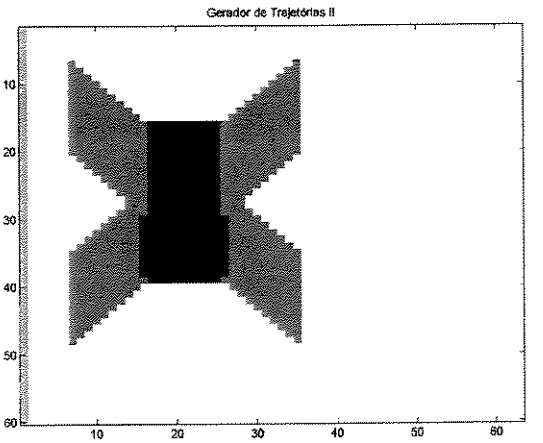


Figura 5.138 - O movimento do cursor.

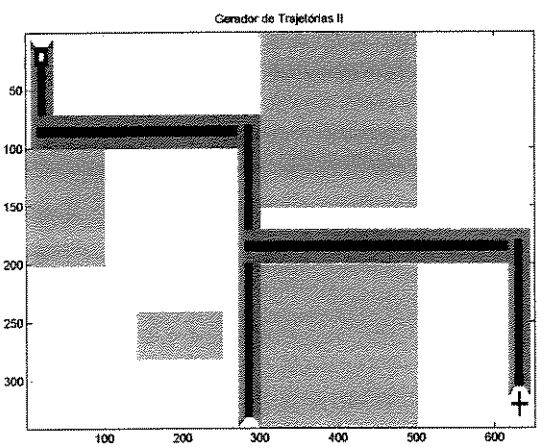


Figura 5.139 - DP Sul e Leste.

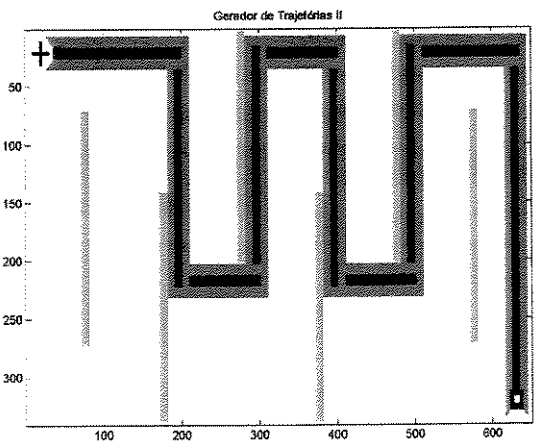


Figura 5.140 - DP Norte e Oeste.

5. 6.4. A Trajetória Final

Funcionando como uma otimização da trajetória gerada, a chamada Trajetória Final (TF), consiste na simples eliminação dos pontos por onde o cursor passou mais de uma vez. Com isso, são desconsiderados os pontos das situações em que o cursor vai para o Norte e em seguida retorna para o Sul e vice-versa, ou para os casos do cursor ir para o Leste e depois retornar para o Oeste e vice-versa. As figuras 5.141 e 5.142 representam a trajetória final para os exemplos que foram mostrados nas figuras 5.037 e 5.072, respectivamente.

Para que a trajetória final seja possível é necessário que o cursor tenha um avanço fixo, ou seja, que se movimente com velocidade constante.

Como saída dos programas Gerador de Trajetória I e II tem-se uma matriz formada pelas coordenadas de todos os pontos da trajetória gerada. Assim, com a utilização da trajetória final esta matriz é reduzida com a eliminação dos pontos repetidos, possibilitando uma otimização para a armazenagem e a aplicação futura destes dados.

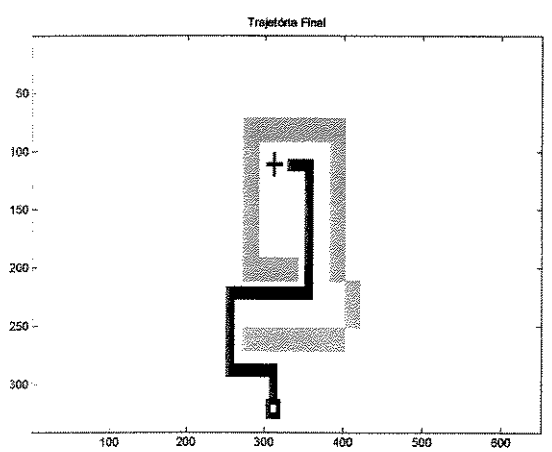


Figura 5.141 - TF da figura 5.037.

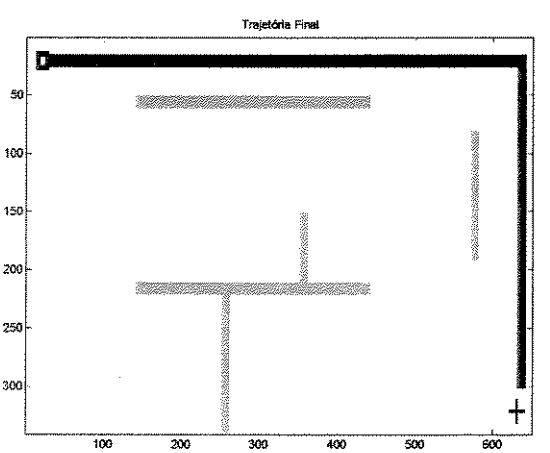


Figura 5.142 - TF da figura 5.072.

5. 7. Conclusões

A formatação final do sistema proposto, presente no programa Gerador de Trajetórias II, foi testada para 13 diferentes configurações do ambiente de trabalho e para várias posições inicial e final adotadas no mesmo. O sistema não foi satisfatório para algumas dessas posições, a depender da configuração do ambiente, como por exemplo, na presença de obstáculos em "U" e para os casos em que a posição alvo está localizada em uma região oposta e nas proximidades do obstáculo, como se o cursor fosse obrigado a transpor o mesmo. A complexidade e a dificuldade para a transposição deste tipo de obstáculo foi descrita nos trabalhos de Fabro (Fabro, 1996) e de Oliveira (Oliveira, 1995).

Em contrapartida, o grande número de resultados, apresentados através das simulações gráficas, comprovam a eficiência do sistema proposto.

Capítulo 6

Conclusões e Perspectivas Futuras

Neste trabalho, foi proposto um sistema para a geração de trajetórias, utilizando redes neurais artificiais, aplicado na navegação de robôs móveis, permitindo que os mesmos possam se deslocar de uma maneira mais autônoma entre dois determinados pontos no seu ambiente de trabalho, considerado desestruturado.

A navegação autônoma de robôs móveis desperta há muito tempo o interesse de vários pesquisadores, principalmente na resolução do problema de navegação autônoma, descrito no Capítulo 2. Na área da inteligência artificial, destaca-se a utilização de redes neurais artificiais, que individualmente, ou em composição com outro tipo de abordagem da mesma área, se mostraram eficientes na resolução do problema em questão

Redes neurais artificiais são ferramentas poderosas na resolução de vários tipos de problemas, em destaque, o problema da classificação de padrões. Com este propósito, as redes utilizam a sua capacidade de realizar mapeamentos não-lineares dos padrões usados no seu treinamento (aprendizado supervisionado). Uma rede neural do tipo MLP, com uma única camada intermediária, é uma arquitetura adequada e geralmente empregada para a resolução deste tipo de problema.

As duas abordagens descritas no Capítulo 4 foram desenvolvidas com o objetivo de resolver o problema de navegação autônoma. A principal barreira encontrada na utilização das

redes neurais do tipo MLP, e que levou ao desenvolvimento da segunda abordagem, foi a relação existente entre o número de padrões usados no treinamento e o tempo de convergência do mesmo. À medida que foi se elevando o número de padrões, o tempo de treinamento também aumentou, o que dificultou o teste dos padrões usados e a inclusão de novos padrões. O emprego de duas estratégias visando a diminuição do tempo do treinamento, a divisão do conjunto dos padrões e o uso de um programa de treinamento implementado em C++, se mostraram eficientes para a determinação do número final dos padrões de treinamento.

A formatação final do sistema proposto, presente no programa Gerador de Trajetórias II, foi testada para 13 diferentes configurações do ambiente de trabalho e para várias posições inicial e final adotadas no mesmo.

Por desvantagens da metodologia empregada, têm-se o fato da locomoção ser restrita apenas a quatro direções (Norte, Sul, Leste e Oeste) e o custo computacional do treinamento das redes neurais. Dependendo do número de neurônios da camada intermediária e da quantidade de padrões usados, o tempo de processamento cresce expressivamente.

Em contrapartida, o grande número de resultados, apresentados através das simulações gráficas, comprovam a eficiência do sistema proposto. O sistema apresentou bons resultados, permitindo que, numa etapa posterior, o mesmo possa ser testado em robôs móveis reais.

A principal vantagem da metodologia apresentada reside no fato de que o robô pode ter um nível de aprendizado proporcional à quantidade de padrões utilizados no treinamento da rede neural: com isso, a depender da complexidade da sua aplicação, o sistema necessitará de um número menor ou maior de padrões de treinamento.

Também no custo de implementação, pois para um número reduzido de padrões não vai ser necessária a modulação dos padrões e assim o uso de uma única rede neural pode ser suficiente. Neste caso, têm-se a simplicidade do sistema e o seu baixo custo, tanto de implementação quanto computacional, visto que, após o treinamento os pesos são transferidos para a rede neural inserida no Gerador de Trajetórias e a mesma, através do processamento direto das informações de

entrada, decide qual a nova direção que deverá ser adotada. Seu baixo esforço computacional decorre da utilização de um único passo computacional (*forward*) e também pelo fato das informações serem fornecidas à rede na forma de valores binários.

Sua aplicação futura pode estar no campo dos robôs móveis usados para serviços em escritórios, como o transporte de correspondência; em bibliotecas, arquivando livros nas suas respectivas estantes; em feiras de eventos, servindo como guia; etc.

Como proposta de trabalhos futuros têm-se:

- A transposição do sistema proposto para a estrutura neural proposta por Mendeleck (Mendeleck, 1995);
- A implementação do Gerador de Trajetórias em robôs móveis reais;
- O estudo de outras técnicas de inteligência artificial buscando uma melhoria do sistema;
- Continuar o desenvolvimento do sistema para que o mesmo possa ter um desempenho satisfatório frente às seguintes modificações: o aumento do número das direções possíveis de movimento e a sua implementação em C++®.

Referências Bibliográficas

Akbarzadeh-T, M. -R.; Kumbla, K.; et al. Soft Comuting. for Autonomous Robotic Systems. *Computers and Electrical Engineering*, v. 26, p. 5-32, 2000

Almássy, Nikolaus. BUGWORLD: A Distributed Environment for the Development of Control-Architectures in Multi-Agent Worlds. *Technical Report 93-32*, Department of Computer Science, University Zurich, 1993.

Almássy, Nikolaus; Vinkhuyzen, Erik. Evolution of Adaptive Behavior in Dynamic Environments. In: Jamashidi, M. et al. *Intelligent Automation and Soft Computing: Trends in Research, Development and Applications*, Albuquerque, NM, TSI Press, 1994. p. 419-424.

Ambler, A. P.; Popplestone, R. J. Inferring the Positions of Bodies from Specified Spatial Relationships. *Artificial Intelligence*, v. 6, n. 2, p. 157-174, 1975

Andrade, José F. A. de; Mendeleck, A.; Zampieri, Douglas E. Geração de Trajetórias para Robôs Móveis Autônomos Usando Redes Neurais Artificiais. In: V Congresso Brasileiro de Redes Neurais, Rio de Janeiro - RJ, Brasil. *Anais...* abril 2001. p. 571-576

Barraquand, J.; Latombe, J. C. Robot Motion Planning: A Distributed Representation Approach. *Report Nº. STAN-CS-89-1257*, Department of Computer Science, Stanford University (To appear in *International Journal of Robotics Research*). 1989

Benzatti, Danilo L.; Andrade, José F. A. de; Mendeleck, A.; Zampieri, Douglas E. Autonomous Locomotion of a Robot Based on Neural Networks Trajectory Generation. In: IX DINAME, 2001, Florianópolis - SC. *Proceedings...* march 2001. p. 175-180

Billard, Aude; Hayes Gillian. DRAMA, a Connectionist Architecture for Control and Learning in Autonomous Robots. *Adaptive Behavior Journal,* v. 7:1, p. 35-64, jan. 1999

Bradshaw, A. Sensors for Mobile Robots. *Measurement + Control,* v. 23, p. 48-52, march 1990

Brooks, R. A. Planning Collision-Free Motions for Pick-and-Place Operations. *International Journal of Robotics Research,* v. 2, n. 4, p. 19-44, 1983b

Brooks, R. A. Solving the Find-Path Problem by Good Representation of Free Space. *IEEE Transactions on Systems, man and Cybernetics,* SMC - v. 13, n. 3, p. 190-197, 1983a

Brooks, R. A. Symbolic Error Analysis and Robot Planning. *International Journal of Robotics Research,* v. 1, n. 4, p. 29-68, 1982

Brooks, R. A. A robust Layered Control System for a Mobile Robot. *IEEE Journal of Robotics and Automation,* v. 2, n. 1, p. 14-23, 1986

Brooks, R. A. Intelligence Without Reason. In: IJCAI-91, Morgan Kaufmann, San Mateo CA USA. *Proceedings...*, p. 569-595, 1991

Canny, John. On the "Piano Movers'" Serie by Schwartz, Sharir and Ariel-Sheffi. In: Khatib, Oussama; et al. *The Robotics Review 1.* Cambridge: Massachusetts, The MIT Press, 1989. p. 33-40

Castro, Leandro N. de. *Análise e Síntese de Estratégias de Aprendizado para Redes Neurais Artificiais*. Campinas - SP: Faculdade de Engenharia Elétrica e de Computação, UNICAMP, 1998. 248 p. Dissertação (Mestrado)

Castro, Leandro N. de; et al. Feedforward Neural Network Initialization: an Evolutionary Approach. In: Vth Brazilian Symposium on Neural Networks, Belo Horizonte - MG, Brazil. *Proceedings...* december 1998. p. 43-48

Castro, Leandro N. de; Von Zuben, Fernando J. A Hybrid Paradigm for Weight Initialization in Supervised Feedforward Neural Network Learning. In: 1998 International Computer Symposium – Workshop on Artificial Intelligence, Tainan, Taiwan. *Proceedings...* december 1998. p. 30-37

Castro, Leandro N. de; Von Zuben, Fernando J. Optimised Training Techniques for Feedforward Neural Networks. *Technical Report DCA-RT 03/98*. Campinas - SP: Department of Computer Engineering and Industrial Automation, School of Electrical and Computer Engineering, UNICAMP. july 1998. 41 p. Disponível na Internet: http://www.dca.fee.unicamp.br/~vonzuben/research/rt_dca.html

Chang, Charles C.; Song; Kai-Tai. Environment Prediction for a Mobile Robot in a Dynamic Environment. *IEEE Transactions on Robotics and Automation*, v. 13, n. 6, p. 862-872, december 1997

Chatila, R. Path Planning and Environment learning in a Mobile Robot System. In: European Conference on Artificial Intelligence, Orsay, France. *Proceedings...* 1982

Chatila, Raja G. A Robust Layered Control System for a Mobile Robot by Rodney A. Brooks. In: Khatib, Oussama; et al. *The Robotics Review 1*. Cambridge: Massachusetts, The MIT Press, 1989. p. 103-108

Choi, Jung W.; Kwon, Soon H.; et al. Navigation Strategy of na Intelligent Mobile Robot Using Fuzzy Logic. In: IEEE International Conference on Fuzzy Systems (FUZZ-IEEE), 1998, Anchorage, Alaska. *Proceedings...* 1998. p. 602-605

Cybenko, G. Approximation by Superpositions of a Sigmoidal Function. *Mathematics of Control, Signals and Systems*, v. 2, p. 303-314, 1989

Davis, R. H.; Camacho, M. The Application of Logic Programming to the Generation of Paths for Robots. *Robotic,*. v. 2, p. 93-103, 1984

Delgado, Myriam R.; Von Zuben; F. J.; Gomide, F. Hierarchical Genetic Fuzzy Systems. *Informatio Sciences - Special Issue on Genetic Fuzzy Systems*, a ser publicaado em 2001

Dozier, Gerry; Homaifar, Abdollah; et al. Artificial Potential Field Based Robot Navigation, Dynamic Constrained Optimization, and Simple Genetic Hill-Climbing. In: IEEE International Conference on Evolutionary Computation (ICEC), 1998, Anchorage, Alaska. *Proceedings...* 1998. p. 165-170

Dracopoulos, Dimitris C. Robot Path Planning for Maze Navigation. In: International Joint Conference on Neural Networks (IJCNN), 1998, Anchorage, Alaska. *Proceedings...* Anchorage: IEEE Neural Networks Council,1998. p. 2081-2085

Dudek, Gregory ; Freedman, Paul; Hadjres, Souad. Mapping in Unknown Graph-Like Worlds. *Journal of Robotic Systems*, v. 13, n. 8, p. 539-559, 1996

Dufay, B.; Latombe, J. C. An Approach to Automatic Robot Programming Based on Inductive Learning. *International Journal of Robotics Research*, v. 3, n. 4, p. 3-20, 1984

Fabro, João A. *Grupos Neurais e Sistemas Nebulosos: Aplicação à Navegação Autônoma.* Campinas - SP: Faculdade de Engenharia Elétrica, UNICAMP, 1996. 64 p. Dissertação (Mestrado)

Feder, Hans J. S.; Leonard, John J.; Smith, Christopher. Adaptive Mobile Robot Navigation and Mapping. *The International Journal of Robotics Research,* v. 18, n. 7, p. 650-668, july 1999

Ferreira, Edson de P. *Robótica Básica.* Rio de Janeiro: Versão Preliminar Publicada para a V Escola Brasileiro - Argentina de Informática, janeiro 1991.

Fierro, R; Lewis, F. L. Control of a Nonholonomic Mobile Robot Using Neural networks. *IEEE Transactions on Neural Networks*, v. 9, n. 4, july 1998.

Floreano, D.; Mondada, F. Evolution of Homing Navigation in Real Mobile Robot. *IEEE Transactions on Systems, Man, and Cybernetics - Part B,* v. 26, n. 3, p. 1-13, 1996

Gasós, Jorge; Rosetti, Ailson. Uncertainty representation for mobile robots: Perception, modeling and navigation in unknown environments. *Fuzzy Sets and Systems*, v. 107, p. 1-24, 1999

Gat, E.; Desai, R.; et al. Behaviour Control for Robotic Exploration of Planetary Surfaces. *IEEE Transactions on Robotics and Automation,* v. 10, n. 4, p. 490-503, 1994

Giralt, G.; Sobek, R.; Chatila, R. A Multi-Level Planning and Navigation System for a Mobile Robot. Sixth IJCAI. Tokyo, Japan. *Proceedings...*, v.1, 1979

Graves, S.; Skubic, M.; Mollenhauer, J. The FuzzBug Mobile Robot. *Technical Report 93-041,* Department of Computer Science, Texas A&M University, May 1993. 22 p. Disponível na Internet: http://www.cs.tamu.edu/research/robotics/Skubic/Papers/fuzzbug_report.ps.gz

GSI, DIN - UEM. *Grupo de Sistemas Inteligentes. Departamento de Informática - Universidade Estadual de Maringá.* Fevereiro de 2001. Material descrito na homepage. Disponível na Internet: http://www.din.uem.br/~ia/1024x768/index.html

Hague, T.; Tillett, N. D. Navigation and Control of a Autonomous Horticultural Robot. *Mechatronics,* v. 6, n. 2, p. 165-180, 1996

Han, Min-Hong; Rhee, Sang-Yong. Navigation Control for a Mobile Robot. *Journal of Robotic Systems,* v. 11, n. 3, p. 169-179, 1994

Haykin, S. *Neural Networks – A Comprehensive Foundation.* New Jersey: Prentice Hall, Second Edition, 1999, 842 p.

Heuberger, Carlos Florian. *Determinação Automática de Trajetórias Ótimas para um Manipulador na Presença de Obstáculos.* Campinas - SP: Faculdade de Engenharia Mecânica, UNICAMP, 1990. 84 p. Dissertação (Mestrado)

Hocaoglu, Cem; Sanderson, Arthur C. Multi-Dimensional Path Planning Using Evolutionary Computation. In: IEEE International Conference on Evolutionary Computation (ICEC), 1998, Anchorage, Alaska. *Proceedings...* 1998. p. 165-170

Iyoda, Eduardo M. *Inteligência Computacional no Projeto Automático de Redes Neurais Híbridas e Redes Neurofuzzy Heterogêneas.* Campinas - SP: Faculdade de Engenharia Elétrica e de Computação, UNICAMP, 2000. 166 p. Dissertação (Mestrado)

Isik, C.; Meystel, A. M. Pilot Level of a Hierarchical Controller for an Unmanned Robot. *IEEE J. Robotics and Automation,* v. 4, n. 3, p. 241-255, 1988

Janét, Jason A.; Gutierrez, Ricardo; et al. Autonomous Mobile Robot Global Self-Localization Using Kohonen and Region-Feature Neural Networks. *Journal of Robotic Systems,* v. 14, n. 4, p. 263-282, december 1997

Janusz, B.; Riedmiller, M. Self-learning Neural Control of a Mobile Robot. In: International Conference on Neural Networks (ICNN), 1995, Perth, Australia. *Proceedings...* Anchorage: IEEE Neural Networks Council,1995. p. 2358-2363

Jetto, Leopoldo; Longhi, Sauro; Venturini, Giuseppe. Development and Experimental Validation of na Adaptive Extended Kalman Filter for the Localization of Mobile Robot. *IEEE Transactions on Robotics and Automation,* v. 15, n. 2, p. 219-228, april 1999

Jones, Joseph L.; Flynn, Anita M. *Mobile Robots – Inspiration to Implementation.* Wellesley, Massachusetts: A. K. Peters Ltd., 1993, 345 p.

Hornik, K.; Stinchcombe, M.; White, H. Multilayer Feedforward Neural Networks are Universal Approximators. *Neural Networks*, v. 2, p. 359-366, 1989

Khatib, O. Real-Time Obstacle Avoidance for manipulators and Mobile Robots. *International Journal of Robotics Research*, v. 5, n. 1, p. 90-98, 1986

Koch, E., Yeh, C.; Hillel, G., Meystel, A.; Isik, C. Simulation of Path Planning for a System with Vision and Map Updating. IEEE Conf. On Robotics and Automation, St. Louis, Missouri. *Proceedings...*, p. 146-160, 1985

Koditschek, D. E. Exact Robot Navigation by Means of Potential Functions: Some Topological Considerations. In: IEEE International Conference on Robotics and Automation, Raleigh, NC. *Proceedings...* p. 1-6, 1987

Kumar, Vijay R.; Waldron, Kenneth J. A Review of Research on Walking Vehicles. In: Khatib, Oussama; et al. *The Robotics Review 1*. Cambridge: Massachusetts, The MIT Press, 1989. p. 244-266

Latombe, Jean-Claude. *Robot Motion Planning.* Massachusetts: Kluwer Academic Publishers, 1996, 651 p.

Laumond, J. P. Feasible Trajectories for Mobile Robots with Kinematic and Environment Constraints. In: International Conference on Intelligent Autonomous Systems. Elsevier Science Publishers B. V., Amsterdam, The Netherlands. *Preprints...* p. 346-354, 1986

Laumond, J. P. Finding Collision-Free Smooth Trajectories for a Non-Holonomic Mobile Robot. In: 10th International Joint Conference on Artificial Intelligence, Milan, Italy. *Proceedings...* p. 1120-1123, 1987

Laumond, J. P. *Robot Motion Planning and Control.* Lectures Notes in Control and Information Sciences 229, Springer, 1998, 343 p. Disponível na Internet: http://www.laas.fr/~jpl/book.html

Li, Wei; Ma, Chenyu; Wahl, F. M. A Neuro-Fuzzy System Architecture for Behavior-Based Control of a Mobile Robot in Unknown Environments. *Fuzzy Sets and Systems,* v. 87, p. 133-140, 1997

Li, W. Fuzzy Logic-Based 'Perception-Action' Behaviour Control of a Mobile Robot in Uncertain Environments. In: Third IEEE International Conference on Fuzzy Systems, Piscataway - NJ, USA. *Proceedings...,* v. 3, p. 1626-1631, 1994

Li, Z.; Canny, J. F. Robot Motion Planning with Non-Holonomic Constraints. *Memo UCB/ERL M89/13,* Electronics Research Laboratory, University of California, Berkeley. 1989

Lozano-Pérez, T. Automatic Planning of Manipulator Transfer Movements. *IEEE Trans. on Systems, Man, Cybernetics,* v. *SMC*-11, n. 10, p. 681-689, 1981

Lozano-Pérez, T. Spatial Planning: A Configuration Space Approach. *IEEE Trans. on Computers*, v. C-32, n. 2, 1983

Lozano-Pérez, T. The Design of a Mechanical Assembly System. *Technical Report AI-TR 397*, Artificial Intelligence Laboratory, MIT. 1976

Lozano-Pérez, T.; Mason, M. T.; Taylor, R. H. Automatic Synthesis of Fine-Motion Strategies for Robots. *International Journal of Robotics Research*, v. 3, n. 1, p. 3-24, 1984

Lozano-Pérez, T.; Wesley, M. A. An Algorithm for Planning Collision-Free Paths Among Polyhedral obstacles. *Communications of the ACM*, v. 22, n. 10, p. 560-570, 1979

Lum, H.K.; Zribi, M.; Soh, Y.C. Planning and control of a biped robot. *Int. Journal of Engineering Science,* v. 37, p. 1319-1349, 1999

Maaref, H.; Barret, C. Sensor-based Fuzzy Navigation of an Autonomous Mobile Robot in an Indoor Environment. *Control Engineering Practice,* v. 8, p. 757-768, 2000

MacLoone, S.; Irwin, G. W. Fast Parallel Off-Line Training of Multilayer Perceptrons. *IEEE Transactions on Neural Networks*, v. 8, n. 3, p. 646-653, 1997

Mendeleck, André. *Um Modelo Conexionista para a Geração de Movimentos Voluntários em Ambiente Desestruturado*. Campinas - SP: Faculdade de Engenharia Mecânica, UNICAMP, 1995. 182 p. Tese (Doutorado)

Meystel, A. M. *Semiotic Modeling and Situation Analysis: An Introduction*. AdRem, 158 p., 1995

Meystel, A. M. Intelligent Systems: A Semiotic Perspective. *International Journal of Intelligent Control and Systems*, v. 1, n. 1, p. 31-57, 1996

Moller, M. F. A Scaled Conjugate Gradient Algorithm for Fast Supervised Learning. *Neural Networks*, v. 6, p. 525-533, 1993

Nilsson, N. J. A Mobile Automaton: An Application of Artificial Intelligence Techniques. In: 1st International Joint Conference on Artificial Intelligence, Washington D.C. *Proceedings...* p.509-520, 1969

Nilsson, N. J. Principles of Artificial Intelligence. Tioga Publishing. California, 1980

Ó'Dúnlaing, C.; Yap, C. K. A Retraction Method for Planning the Motion of a Disc. *Journal of Algorithms*, v. 6, p. 104-111, 1982

Ohya, Akihisa; Kosaka, Akio; Kak, Avinash. Vision-Based Navigation by a Mobile Robot with Obstacle Avoidance Using Single-Camara Vision and Ultrasonic Sensing. *IEEE Transactions on Robotics and Automation*, v. 14, n. 6, p. 969-978, 1998

Oliveira, Marco A. A. de. *Aplicação de Métodos de Computação Flexível em Navegação Autônoma de Veículos*. Campinas - SP: Faculdade de Engenharia Elétrica, UNICAMP, 1995. 135 p. Dissertação (Mestrado)

Ortega, J. G.; Camacho; E. F. Mobile Robot Navigation in a Partially Structured Static Environment, Using Neural Predict Control. *Control Eng. Practice*, v. 4, n. 12, p. 1669-1679, 1996

Popplestone, R. J.; Ambler, A. P.; Bellos, I. M. An Interpreter for a Language for Describing Assemblies. *Artificial Intelligence*, v. 14, n. 1, p. 79-107, 1980

Prassler, E.; Scholz; J.; Fiorini, P. Navigation a Robotic Wheelchair in a Railway Station During Rush Hour. *The International Journal of Robotics Research,* v. 18, n. 7, p. 711-727, july 1999

Pratihar, Dilip K.; et al. A Genetic-Fuzzy approach for Mobile Robot Navigation Among Moving Obstacles. *International Journal of Approximate Reasoning,* v. 20, p. 145-172, 1999

Reif, J. H. Complexity of the Generalized Mover's Problem. In: [Schwartz, Sharir and Hopcroft, 1987], p. 267-281, 1987

Reif, J. H. Complexity of the Mover's Problem and Generalizations, In: 20th IEEE Symposium on Foundations of Computer Science. *Proceedings...* p. 421-427, 1979

Rezende, Marcos F. de. *Desenvolvimento de um Robô Móvel Autônomo Inteligente Utilizando a Arquitetura de Assunção.* Uberlândia - MG: Universidade Federal de Uberlândia, 1992. 102 p. Dissertação (Mestrado)

Salichs, M. A.; Moreno, L. Navigation of Mobile Robots: Open Questions. *Robotica,* v. 18, n. 3, p. 227-234, 2000

Schwartz, J. T.; Sharir, M. On the Piano Movers' Problem: I. The Case if a Two-Dimensional Rigid Polygonal Body Moving Amidst Polygonal Barriers. *Communications on Pure and Applied Mathematics*, v. 36, p. 345-398, 1983a

Schwartz, J. T.; Sharir, M. On the Piano Movers' Problem: II. General Techniques for Computing Topological Properties of Real Algebraic Manifolds. *Advances in Applied Mathematics*, Academic Press, v. 4, p. 298-351, 1983b

Schwartz, J. T.; Sharir, M. On the Piano Movers' Problem: III. Coordinating the Motion of Several Independent Bodies: The Special case of Circular Bodies Moving Amidst Polygonal Barriers. *International Journal of Robotics Research*, v. 2, n. 3, p. 46-75, 1983c

Schwartz, J. T.; Sharir, M. On the Piano Movers' Problem: V. The Case of a Rod Moving in Three-Dimensional Space Amidst Polyhedral Obstacles. *Communications on Pure and Applied Mathematics*, v. 37, p. 815-848, 1983d

Sharir, M; Ariel-Sheffi, E. On the Piano Movers' Problem: IV. Various Decomposable Two-Dimensional Planning Problems. *Communications on Pure and Applied Mathematics*, v. 37, p. 479-493, 1983

Stafylopatis, A.; Blekas, K. Autonomous Vehicle navigation Using Evolutionary Reinforcement Learning. *European Journal of Operational Research*, v. 108, p. 306-318, 1998

Suárez, Lizet L. *Conhecimento Sensorial - Uma Análise Segundo a Perspectiva da Semiótica Computacional*. Campinas - SP: Faculdade de Engenharia Elétrica e de Computação, UNICAMP, 2000. 144 p. Dissertação (Mestrado)

Tani, Jun; Fukumura, Naohiro. Self-organizing Internal Representation in Learning of Navigation: A Physical Experiment by the Mobile Robot YAMABICO. *Neural Networks*, v. 10, n. 1, p. 153-159, 1997

Taylor, R. H. *Synthesis of Manipulator Control Programs from task-Level Specifications.* Department of Computer Science, Stanford University, Ph.D. Dissertation, 1976

Thompson, A. M. The Navigation System of the JPL Robot. Fifth IJCAI. Cambridge. MA. *Proceedings...*, p. 749-757, 1977

Thrun, Sebastian. An Approach to Learning Mobile Robot Navigation. *Robotics and Autonomous Systems*, v. 15, n. 4, p. 301-319, 1995

Tse, P. W.; Lang, S.; et al. Design of a Navigation System for a Household Mobile Robot Using Neural Networks. In: International Joint Conference on Neural Networks (IJCNN), 1998, Anchorage, Alaska. *Proceedings...* Anchorage: IEEE Neural Networks Council,1998. p. 2151-2156

Tzafestas, Spyros G. Neural Networks in Robotics: State of the Art. In: IEEE Int. Sympo. on Ind Electronics (ISIE'95),1995, New Jersey, USA. *Proceedings...* July 1995, v. 1, p. 12-20

Udupa, S. *Collision Detection and Avoidance in Computer Controlled Manipulators.* Department of Electrical Engineering, California Institute of Technology, Ph.D. Dissertation, 1977

Uebel, Luís F.; et al. Controle Inteligente de Robôs Móveis Autônomos: RAM x Fuzzy. In: II Simpósio Brasileiro de Redes Neurais, 1995, São Carlos - SP. *Anais...* São Carlos: Laboratório de Inteligência Computacional, Instituto de Ciências Matemáticas de São Carlos, Universidade de São Paulo, 1995. p. 43-48

Vaz, Jerusa M.; Fabro, João A. SNNAP – Sistema Neural de Navegação em Ambientes Pré-Mapeados. In: IV Brazilian Conference on Neural Networks, 1999, São José dos Campos - SP. *Proceedings...* São José dos Campos: Conselho Nacional de Redes Neurais , 1999. p. 118-123

Victorino, A. C. *Controle de Trajetória e estabilização de Robôs Móveis Não-Holonômicos.* Campinas-SP: Faculdade de Engenharia Mecânica, UNICAMP, 1998. Dissertação (Mestrado)

Verschure, P. F. M. J.; Kröse, B. J. A.; Pfeifer, Rolf. Robotics and Autonomous Systems, volume 9, chapter Distributed Adaptive Control: The Self-Organization of Structured Behaviour, Elsevier Science Publishers B. V.., p. 181-196, 1992

Von Zuben, Fernando J. *Modelos Paramétricos e Não-Paramétricos de Redes Neurais Artificiais e Aplicações*. Campinas-SP: Faculdade de Engenharia Elétrica, UNICAMP, 1996. 244 p. Tese (Doutorado)

Von Zuben, Fernando J. *Redes Neurais I.* Campinas-SP: Faculdade de Engenharia Elétrica e de Computação, UNICAMP, Segundo semestre 1999, Notas de Aulas.

Yang, Jung-Min; Kim, Jong-Hwan. Slidind Mode Control for Trajectory Tracking of Nonholonomic Wheeled Mobile Robots. *IEEE Transactions on Robotics and Automation,* v. 15, p. 578-587, 1999

Yang, Simon X.; Meng, Max. An Efficient Neural Network Approach to Dynamic Robot Motion Planning. *Neural Networks,* v. 13, p. 143-148, 2000