UNIVERSIDADE FEDERAL DE SANTA CATARINA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

NAVEGAÇÃO DE ROBÔS MÓVEIS AUTÔNOMOS: ESTUDO E IMPLEMENTAÇÃO DE ABORDAGENS

Dissertação submetida à
Universidade Federal de Santa Catarina
como parte dos requisitos para a
obtenção do grau de Mestre em Engenharia Elétrica.

JERUSA MARCHI

Florianópolis, Fevereiro de 2001.

NAVEGAÇÃO DE ROBÔS MÓVEIS AUTÔNOMOS: ESTUDO E IMPLEMENTAÇÃO DE ABORDAGENS

Jerusa Marchi

'Esta Dissertação foi julgada adequada para a obtenção do grau de Mestre em Engenharia Elétrica, Área de Concentração em *Controle, Automação e Informática Industrial*, e aprovada em sua forma final pelo Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Santa Catarina.'

Edson Roberto De Pieri, Dr.
Orientador

Edson Roberto De Pieri, Dr.
Coordenador do Programa de Pós-Graduação em Engenharia Elétrica

Banca Examinadora:

Edson Roberto De Pieri, Dr.
Presidente

Guilherme Bittencourt, Dr.

Marcelo Ricardo Stemmer, Dr.

Mauro Roisenberg, Dr.

"De tantas dúvidas e erros me via embaraçado que me parecia não haver tirado outro proveito do procurar instruir-me senão o de ter descoberto mais e mais a minha ignorância."

René Descartes

Dedico este trabalho à sociedade brasileira, cujo apoio indireto fomenta o progresso científico-tecnológico de nosso país, ainda que o conhecimento gerado nem sempre seja revertido em benefícios à esta mesma sociedade.

AGRADECIMENTOS

Agradeço imensamente a meus pais pelo carinho, cuidado e apoio.

Agradeço ao meu "eterno (des)orientador" João Alberto Fabro pelo incentivo e força quando decidi fazer o mestrado e pelos conselhos e conversas durante a realização deste (valeu João!!). Agradeço também aos meus ex-professores da UNIOESTE, especialmente a Juan Carlos Sotuyo pelos conselhos sempre tão bem vindos, Jorge Hanna Habib El Khouri pelos ensinamentos e a Ricardo Krauskopf, afinal de contas tudo "é mamão com açúcar".

Gostaria de agradecer a todos os colegas e amigos que me acolheram em Florianópolis: Sonia Palomino, pela calma e sobriedade; Marcos Vallim, pelas conversas bem humoradas e orgias culinárias; Antonio Eduardo Carrilho pela simplicidade e eterno bom humor; aos colegas de créditos: Renato Donizete, Terezinha Faria, Carlos Ogawa, Fred Paes pela companhia agradável nas madrugadas e finais de semana passados no laboratório (além das pizzas e festas!); a Cesar Montejunas e Vinícius de Oliveira pelo companheirismo e ajuda; aos meninos do laboratório: Javier Triveño Vargas, César Torrico, Frank Siqueira, Hallthmann Lima, Elmer Llanos, Fernando Passold e José Maria - e às meninas do laboratório: Karina Barbosa, Cristiane Pain (muito obrigada por me estenderem suas mãos no momento mais difícil deste mestrado), Karen Farfan e Michelle Wangham - pela amizade; a Ivair Lourinho pelas divagações poéticas; a Augusto Loureiro pela amizade; a Carlos Brandrão por me mostrar o imenso prazer que há nas coisas mais simples, à Isabel Tonin pelas conversas; a Frederico Freitas pela visão colorida do mundo; a Rafael Obelheiro pelas dicas e ajuda com o LaTeX.

Agradeço a Ana pela amizade e apoio à distância!! A João Neto pelo apoio e conselhos. A Vitor Litwinczik, Paulo Boni, Sérgio Barra, Sílvio Neves, Sérgio Almeida, Euclides Júnior e Cristiano Lehrer pela amizade.

Agradeço aos professores do DAS pelo incetivo e apoio recebidos (valeu Werner! valeu Cury!) e em especial ao meu orientador, prof. Edson, por aceitar minha proposta de dissertação.

Por fim, agradeço aos professores membros da banca pelos comentários e sugestões referentes ao trabalho.

Resumo da Dissertação apresentada à UFSC como parte dos requisitos necessários para obtenção do grau de Mestre em Engenharia Elétrica.

NAVEGAÇÃO DE ROBÔS MÓVEIS AUTÔNOMOS: ESTUDO E IMPLEMENTAÇÃO DE ABORDAGENS

Jerusa Marchi

Fevereiro/2001

Orientador: Edson Roberto De Pieri
Área de Concentração: Controle, Automação e Informática Industrial
Palavras-chave: Robôs Móveis Autônomos, Navegação, Planejamento de Trajetória, Inteligência Artificial, Redes Neurais Artificiais.
Número de Páginas: 119

Dentro das arquiteturas de controle de robôs móveis autônomos, a navegação é uma das áreas fundamentais, constituindo a camada intermediária entre o controle dos motores e sensores e o planejamento de tarefa. Tratam-se aqui as três abordagens existentes à navegação: a abordagem planejada ou deliberativa, como uma tentativa inicial de delimitar o problema utilizando um modelo estático do ambiente; a abordagem reativa, como uma alternativa à atuação dos robôs móveis autônomos em ambientes dinâmicos reais, possibilitando ao robô agir segundo os estímulos recebidos do ambiente; e por fim, a abordagem híbrida, que une as melhores características das anteriores, provendo ao robô um comportamento mais "inteligente", com características de aprendizado e adaptação. A grande questão nesta abordagem é, justamente, dosar planejamento e reatividade. São descritas aqui, várias implementações das três abordagens que possibilitam observar as características de cada abordagem ressaltando as vantagens do uso de abordagens híbridas. Ao final, observa-se que a escolha de uma ou outra abordagem está, fundamentalmente, vinculada ao tipo de tarefa que o robô irá realizar. Ressalta-se também a necessidade de dotar os sistemas autônomos de capacidades cognitivas para a criação de robôs verdadeiramente "inteligentes".

Abstract of Dissertation presented to UFSC as a partial fulfillment of the requirements for the degree of Master in Electrical Engineering.

NAVIGATION OF AUTONOMOUS MOBILE ROBOTS: ANALYSIS AND SYNTHESIS OF APPROACHES

Jerusa Marchi

February/2001

Advisor: Edson Roberto De Pieri
Area of Concentration: Control, Automation and Industrial Computing
Keywords: Autonomous Mobile Robots, Navigation, Path Planning, Artificial Intelligence, Neural Networks.
Number of Pages: 119

In the control architecture of autonomous mobile robots, navigation is a fundamental area, representing the layer between the motor and sensor control and the task planning. This work addresses the three approaches for the autonomous navigation: (1) the planning approach, as a initial attempt to solve the problem using a static model of the environment; (2) the sensor-based approach, as an alternative to the autonomous mobile robots acting in real dynamic environments with stimulus received from the environment; and (3) the last one, the hybrid approach that joins the best features of the planning and sensor-based approaches. The latter supplies an "intelligent" behavior for the robot, with learning and adaptive capabilities. We describe several implementations which make it possible to observe the features of these approaches. The implementations will emphasize the advantages of using the hybrid approach. We conclude that the choice for one of the approaches is defined by the kind of task that the robot needs to perform. We emphasize that the autonomous systems needs to have cogintive capabilities. These capabilities are necessary to construct truly "intelligent" robots.

Sumário

Lista de Figuras

Capítulo 1

Introdução

Desde a antigüidade, a idéia de criar sistemas autônomos que imitem o comportamento humano acompanha as civilizações. Os egípcios construíam homens mecânicos ou autômatos em forma de estátuas com articulações móveis. Da Grécia antiga têm-se a descrição de estátuas munidas de tubos falantes, e a partir do século IV a.C., surgem marionetes acionadas por sistemas de polias e pesos [23].

Atualmente, a meta em robótica é criar robôs autônomos que aceitem descrições das tarefas em alto nível e as realizem sem a intervenção do operador. Apesar do desenvolvimento de tais sistemas ser altamente complexo, esta meta vem sendo alcançada com o emprego de técnicas de inteligência artificial [45].

1.1 Formulação do Problema

Um robô móvel autônomo deve ser capaz de perceber o ambiente à sua volta, tomar decisões sobre a melhor ação a ser executada e realizá-la com o mínimo erro, com o objetivo de cumprir sua tarefa. Se a tarefa for navegar de um ponto a outro do ambiente, o robô deve ser capaz de detectar obstáculos próximos, desviar-se deles e alcançar o ponto determinado como sendo seu objetivo.

Se o ambiente onde o robô navega é conhecido, o problema pode ser resumido ao *planejamento de uma trajetória.* Se o ambiente não for conhecido, o robô deve agir *reativamente* àquilo que é detectado pelos sensores. Estas duas estratégias podem ser unidas permitindo ao robô ter um conhecimento básico do ambiente e desviar-se dos obstáculos desconhecidos, ou mesmo armazenar as informações lidas buscando melhorar seu desempenho.

1.2 Histórico

A robótica móvel agrega conceitos da mecânica e da robótica fixa, e desde o começo, percebeu-se a grande complexidade envolvida no desenvolvimento de sistemas móveis robustos e adaptáveis. Devido à esta complexidade, várias simplificações foram feitas, buscando tornar o problema tratável.

Em 1969, Nilsson [35] descreve um sistema robótico móvel que utiliza *quadtrees*[1] para representar o ambiente e grafos de visibilidade para o planejamento da trajetória.

Problemas relacionados com as questões de movimentos rotacionais e translacionais tornavam crítica a modelagem do ambiente. Em 1983, Lozano-Pérez [30] introduziu a idéia de uma *região de incerteza*, criada através do "crescimento" dos obstáculos. Assim, utilizando grafos de visibilidade para o planejamento de trajetória, o robô poderia ser tratado como um simples ponto no espaço de configuração. Este método foi o primeiro método exato aplicado ao problema do planejamento de trajetória.

Ainda em 83, Brooks [9] introduziu o método *freeway* como uma alternativa à modelagem do espaço livre e ao planejamento de trajetória, utilizando o conceito de cones generalizados.

Todos estes métodos consideram o ambiente conhecido e estático e utilizam as informações disponíveis para o planejamento da trajetória antes de iniciar a etapa de execução.

Em 1986, Brooks [10] introduziu uma arquitetura reativa, denominada *arquitetura de subsunção* (*Subsumption*), na qual o robô age baseando-se na leitura de seus sensores. Esta arquitetura baseia-se na decomposição da inteligência em comportamentos individuais, gerando módulos que coexistem e cooperam para a emergência de comportamentos mais complexos. Esta arquitetura é um marco nas pesquisas de métodos reativos.

Ainda em 86, Khatib [21] introduziu o método dos campos potenciais, no qual o robô transforma-se em uma partícula sob a influência de campos eletromagnéticos, constituídos pelos obstáculos e pelo ponto objetivo. Contudo, a grande questão neste método é a existência de mínimos locais. Na tentativa de solucionar este problema, foram introduzidas informações sobre o ambiente, porém, com o uso destas informações o método passa a ser planejado.

Entre 87 e 89, Arkin escreveu vários artigos descrevendo uma arquitetura reativa baseada

[1] Estrutura em forma de árvore que é gerada através da decomposição de um ambiente bidimensional pelo refinamento sucessivo das células.

em esquemas motores, que mais tarde amadureceu em uma arquitetura híbrida, a qual denominou *AuRA - Autonomous Robot Architecture*. Em 89, Arkin [1] compara as duas abordagens e salienta as vantagens da arquitetura híbrida.

Em 1990, Kumpel [26] apresenta um artigo descrevendo o projeto *MARIE - Mobile Autonomous Robot in an Industrial Environment*, este projeto integra mapas geométricos e topológicos e usa um método hierárquico para navegação onde, globalmente utiliza mapas para encontrar o caminho e sensores para desvio local dos obstáculos.

Ainda em 90, Brooks [11] propõe alguns melhoramentos à arquitetura de subsunção e apresenta uma série de robôs que utilizam esta arquitetura.

Dois anos mais tarde, Mataric [31] ressalta a necessidade de alguma representação do ambiente para dotar o robô de capacidades mais elaboradas do que somente a navegação aleatória e propõe um método reativo baseado na arquitetura de subsunção, mas que utiliza um mapa, construído através das marcas (*landmarks*) detectadas no ambiente. O mapa é atualizado sempre que o robô detecta mudanças no ambiente.

No mesmo ano, Zelinsky [58] propõe um método simples para mapeamento do ambiente em tempo de execução, utilizando sensores de contato. Por este método, o ambiente é mapeado em uma *quadtree*, onde a menor célula tem o tamanho do diâmetro do robô. O caminho é, inicialmente, dado por um segmento de reta e durante a execução desta trajetória, as leituras obtidas pelos sensores são utilizadas para atualizar a estrutura da *quadtree* sobre a qual o caminho é replanejado.

1.3 Estado da Arte

A busca por sistemas mais robustos e autônomos tem conduzido as pesquisas em robótica móvel para o uso de abordagens híbridas, unindo planejamento com reatividade, fazendo com que o robô apresente um comportamento instintivo no contexto global, mas reativo diante de situações inesperadas. Trabalhos na área de construção de mapas em tempo de execução têm sido amplamente estudados [24, 29, 27, 57, 36, 37] como alternativa ao armazenamento de informações e planejamento dinâmico de trajetória.

Além disso, o uso de técnicas de inteligência artificial, como redes neurais, redes neuro-fuzzy e aprendizado por reforço, vêm sendo aplicadas ao problema da navegação, possibilitando grandes avanços nas áreas de reconhecimento e aprendizado, principalmente em método reativos e híbridos [48, 34, 15, 25, 13, 46, 8, 17]. Estas técnicas também têm sido

aplicadas ao tratamento da imprecisão e fusão dos sensores [52, 51, 16].

De uma forma geral, a busca por sistemas inteligentes com interfaces amigáveis [44] e que realizem serviços sem a necessidade do acompanhamento de um operador [42], está tornado cada vez mais próximas as áreas de robótica e inteligência artificial.

Verschure [55] comenta a necessidade de dotar o agente autônomo de capacidade cognitiva que o permita tomar decisões baseadas em experiências passadas. Desta forma percebe-se, cada vez mais, a necessidade de se conhecer melhor o funcionamento do cérebro e de como os processos mentais ocorrem para dotar os sistemas computacionais de comportamentos mais "inteligentes" [2, 1, 34], uma vez que isoladamente os comportamentos reativos ou instintivos não apresentam resultados completos e robustos.

1.4 Aplicações

Atualmente, as aplicações em robótica móvel conduzem para o desenvolvimento de três principais tipos de robôs: robôs industriais, de serviço e de campo.

Os robôs móveis industriais, em geral, são plataformas móveis utilizadas para tarefas pesadas como o transporte de materiais e produtos finais em sistemas de manufatura. Esses robôs são denominados *AGVs - Automated Guided Vehicles* e são programados para atuar em ambientes altamente estruturados seguindo linhas desenhadas no chão.

No setor de serviços os robôs móveis são utilizados para limpeza em geral (pisos, dutos de ar, metrôs etc.), em sistema de vigilância e no transporte de materiais leves (como correspondências internas, material hospitalar, etc.). Estes robôs trabalham em ambientes estruturados dos quais possuem um conhecimento prévio.

Já os robôs de campo, trabalham em ambientes não estruturados, pouco conhecidos e geralmente perigosos. As principais atividades destes robôs são: exploração (espacial, de cavernas e vulcões), mineração, limpeza de acidentes nucleares, navegação em estradas e em tarefas agrícolas.

1.5 Objetivos

Este trabalho visa focalizar a linha de evolução da robótica móvel autônoma, no que diz respeito às abordagens ao problema da navegação, através do estudo teórico e da implementação de métodos destas abordagens.

Inicialmente o enfoque seria dado unicamente à abordagem híbrida, contudo, como trata-se de uma nova linha de pesquisa no Laboratório de Controle e Micro Informática - DAS, o trabalho foi expandido para englobar as três abordagens - planejada, reativa e híbrida, com o intuito de prover informações e diretrizes necessárias à novas pesquisas nesta área.

1.6 Organização

No capítulo 2 são abordados alguns conceitos referentes à nomenclatura utilizada em robótica móvel e à teoria das abordagens existentes, onde são descritos os principais métodos e arquiteturas. Nos capítulos 3 e 4 são apresentadas as implementações desenvolvidas em linguagem C e C++, sendo que no capítulo 3 descrevem-se os métodos das abordagens planejada e reativa, desenvolvidas isoladamente, e no capítulo 4, os métodos da abordagem híbrida, com suas duas correntes principais. A seguir são apresentados os resultados obtidos nas simulações realizadas com o simulador Khepera [33] para todas as implementações. Por fim, no capítulo 6, são feitas algumas considerações a respeito deste trabalho e descritas algumas diretivas para trabalhos futuros.

Capítulo 2

Abordagens ao Problema da Navegação - Fundamentação Teórica

2.1 Introdução

O processo de automação iniciou-se visando melhorar processos e produtos, aumentar a produção e reduzir custos. Contudo, o avanço tecnológico não limitou o uso das máquinas às linhas de produção, hoje, elas fazem parte do dia-a-dia das pessoas.

O uso de termos como *máquinas orientadas a humanos* [44] e *robôs de serviço* [42] demonstram a preocupação em criar máquinas cada vez mais autônomas, robustas e "inteligentes", onde a interação com usuários humanos se dê em alto nível (nível de tarefa) e com uma interface amigável.

O desenvolvimento de robôs autônomos é uma tarefa complexa e interdisciplinar, envolvendo elementos das engenharias elétrica e mecânica e de ciência da computação, tais como raciocínio autônomo, percepção e controle [28, 45]. Um robô pode ser definido como um dispositivo mecânico equipado com sensores e atuadores que age sob o controle de um sistema computacional cuja função é executar tarefas repetitivas, enfadonhas ou insalubres. Eles podem ser classificados em três tipos:

- Teleoperados - o operador realiza todos os movimentos que o robô deve fazer.
- Semi-autônomos - o operador indica o macro comando a ser executado e o robô o faz sozinho.
- Autônomos - o robô realiza a tarefa sozinho, tomando suas próprias decisões

baseando-se nas informações recebidas por seus sensores.

Dadas estas definições, robôs móveis autônomos devem ser capazes de receber a descrição das tarefas em alto nível (o robô recebe informações sobre *o que* deve ser feito e não sobre *como* fazê-lo) e selecionar as ações para o seu cumprimento, tomando decisões mediante situações imprevistas, adaptando-se às mudanças que ocorrem no meio onde operam, exprimindo um comportamento coerente sem a necessidade de interferência externa durante o processo [14, 17].

De uma forma simplificada, o problema abordado pela navegação de robôs móveis autônomos consiste em fazer com que o robô navegue pelo ambiente de um ponto inicial a um ponto objetivo sem colidir com os obstáculos. As principais dificuldades encontradas para solucionar este problema estão relacionadas com as propriedades do mundo real [41]. O mundo real é:

- Não totalmente acessível - os sensores fornecem medições imprecisas e em alguns casos somente é possível perceber os estímulos próximos ao robô.
- Indeterminado - interagindo com o mundo real as rodas podem escorregar, as baterias podem falhar e o robô pode danificar-se de alguma forma. Sendo assim, é impossível assegurar que uma determinada ação seja executada sem falhas.
- Não totalmente previsível - não é possível pré-determinar ações futuras e seus efeitos. O robô deve ter a capacidade de manipular problemas de decisão seqüencial e aprendizado.
- Dinâmico - diante das mudanças no mundo real o robô deve ser capaz de saber quando é melhor esperar e quando é melhor agir imediatamente.
- Contínuo - os estados e ações modificam-se de acordo com as configurações e os movimentos. Há infinitas combinações, isto torna impossível enumerar o conjunto de ações.

Desta forma, a complexidade envolvida no problema da navegação encontra-se na multiplicidade de situações com as quais o robô pode se deparar enquanto navega em um ambiente.

Diante das dificuldades em solucionar o problema como um todo, assumiram-se algumas simplificações que retiram as características do mundo real. Assim, considerando o

ambiente estático e conhecido tornou-se possível desenvolver métodos para tratar o problema da navegação. Estes métodos compõem a chamada *Abordagem Planejada* ou *Deliberativa*, onde um modelo do ambiente é fornecido *a priori*. Sobre este modelo estático e considerando o movimento livre de falhas, procura-se por um caminho livre que conduza o robô do ponto inicial ao ponto objetivo.

Contudo, para trabalhar em ambientes reais, o sistema deve ser robusto e adaptável. Desta forma, Brooks [10] propôs uma nova arquitetura de controle, na qual o robô móvel assume comportamentos (desviar obstáculos, seguir paredes, caminhar em frente, alcançar alvo) e toma suas decisões baseando-se na leitura de seus sensores. Esta nova abordagem denomina-se *Abordagem Reativa* ou baseada na leitura de sensores e permite ao robô navegar em ambientes dinâmicos e não estruturados.

Porém, qualquer solução mais elaborada do que a navegação aleatória, necessita de um modelo onde a localização do robô, dos pontos pré-estabelecidos como objetivos e o relacionamento entre eles sejam conhecidos [31]. O sistema deve ser capaz de identificar e tratar cada situação adequadamente ou possuir capacidade de generalização e ser capaz de armazenar um conhecimento mínimo do ambiente no qual interage. Desta forma, buscando extrair as melhores características das duas abordagens existentes, surge a *Abordagem Híbrida*, na qual o robô conhece o ambiente, planeja a trajetória e age reativamente diante de situações imprevistas.

Neste capítulo, estas três abordagens serão descritas, com seus métodos e características. Descrevem-se ainda, algumas definições importantes para a compreensão do restante do capítulo, uma síntese sobre os principais tipos de sensores utilizados em robótica móvel e a teoria referente à construção de mapas do ambiente, utilizada pela abordagem planejada.

2.2 Definições

Antes de dar início à caracterização de cada uma das abordagens ao problema da navegação , faz-se necessário o conhecimento de alguns termos e suas respectivas definições, encontrados em [28, 47]:

- Robô Móvel (A) - um robô móvel é um dispositivo mecânico, montado sobre uma base não fixa, dotado de dispositivos para movimentação e equipado com sensores e atuadores. Possui movimentos de rotação e translação e responde à um sistema com-

putacional. É representado como um objeto rígido, que possui um sistema Cartesiano de coordenadas F_A, cuja origem é dada por O_A.

- Espaço de Trabalho (W) - é o espaço físico no qual o robô se movimentará. Este espaço é modelado por um espaço euclidiano $N - dimensional$ (R^N) onde N é igual a 2 ou 3. A este espaço é fixado um sistema Cartesiano de coordenadas F_W, cuja origem é dada por O_W.

- Obstáculos Fixos (B_i) - são objetos rígidos, fixos no espaço de trabalho, que oferecem restrições ao movimento do robô.

- Obstáculos Dinâmicos ($B_i(t)$) - são objetos que podem movimentar-se pelo espaço de trabalho e alteram dinamicamente o espaço de configuração. Obstáculos móveis podem ser pessoas, outros robôs móveis ou ainda, obstáculos não conhecidos *a priori* pelo robô.

- Configuração (q) - a configuração de um objeto é a especificação da posição de todos os pontos deste objeto com relação a um sistema de coordenadas fixo. Assim, uma configuração q de A é a especificação da posição e orientação de F_A com relação a F_W.

- Espaço de Configuração (C) - é a região formada por todas as possíveis configurações de A no espaço de trabalho, considerando-se as restrições impostas pelos obstáculos. Neste espaço o robô é representado por um ponto (a origem do seu sistema de coordenadas), e os obstáculos sofrem a expansão dada pela soma de uma *região ou zona de incerteza*. O subconjunto de W ocupado por A em uma dada configuração q é denominado $A(q)$.

- C-Obstáculo (CB_i) - um C-Obstáculo é o obstáculo (B_i) do espaço de trabalho (W) mapeado para uma região no espaço de configuração (C):

$$CB_i = \{q \in C / A(q) \cap B_i \neq \emptyset\}$$

O C-Obstáculo é obtido pelo obstáculo acrescido de uma *região de incerteza*. Esta região de incerteza é construída pela origem do sistema cartesiano de coordenadas do robô (O_A) quando este é "passado" sobre as bordas do obstáculo (B).

Um exemplo simples é mostrado na figura 2.1. O robô (A) é um disco e tem a origem do seu sistema de coordenada (O_A) fixa em seu centro. O obstáculo (B) é um obstáculo

poligonal inserido em um espaço de trabalho (W) bidimensional (R^2). O C-Obstáculo (CB) é obtido pelo "crescimento" de B isotropicamente pelo raio de A. A borda de CB é a curva formada pela origem O_A quando "passada" por toda a borda de B.

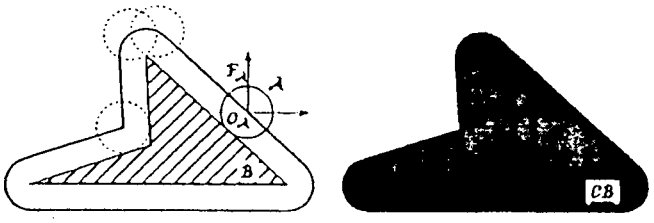


Figura 2.1: Obtenção de um C-Obstáculo

- Região C-Obstáculo - é a união de todos os C-Obstáculos CB_i no espaço de configuração C.

$$\bigcup_{i=1}^{q} CB_i$$

- Espaço Livre (C_{free}) - é a região no espaço de configuração livre para a atuação do robô. Também pode ser definido como a interseção vazia entre as configurações do robô com os obstáculos.

$$C_{free} = \{q \in C / A(q) \cap (\bigcup_{i=1}^{q} B_i) = \emptyset\}$$

- Caminho (τ) - é o mapa contínuo das configurações do robô que o conduzem do ponto inicial ao ponto objetivo.

$$\tau : [0,1] \rightarrow C_{free}$$

com $\tau(0)$ = ponto inicial e $\tau(1)$ = ponto objetivo.

2.3 Sensores

Os sensores têm como função prover ao robô as informações necessárias para que este perceba o ambiente à sua volta e aja no sentido de concluir sua tarefa. Em robótica móvel

utilizam-se principalmente dois grupos de sensores: sensores de colisão e sensores de posicionamento [6].

2.3.1 Sensores de Colisão

Os sensores de colisão são utilizados para detectar obstáculos e mapear o ambiente. Neste grupo encontram-se os sensores infravermelhos, sonares, lasers, câmeras de vídeo e sensores de contato.

Sensor infravermelho

É um dos mais utilizados por ser de baixo custo e funcionamento relativamente simples, contudo, tem seu raio de ação reduzido.

Este sensor emite um raio modulado que atinge o objeto. Parte da luz é refletida e captada por um receptor ótico. A distância é medida por triangulação dado que o ângulo de incidência da luz refletida se altera de acordo com a distância do obstáculo.

Sonar

O sonar é um sensor de baixo custo e demanda poucos recursos computacionais.

Seu funcionamento se dá pela transmissão de uma onda de som em alta freqüência - pulsos ultra-sônicos. Ao atingir o objeto, essa onda é refletida e captada pelo transdutor. A distância é calculada pelo tempo decorrido entre a emissão e o recebimento da onda.

Os principais problemas com o uso do sonar ocorrem quando o sinal enviado bate em uma superfície plana e não é devolvido para o transdutor ou quando há o cruzamento de informações, pois o raio de ação do sensor é muito amplo, causando imprecisão nas leituras e, conseqüentemente falhas na percepção do ambiente.

Laser

Há vários tipos de sensores laser. O mais simples deles utiliza um princípio semelhante ao sensor infravermelho, ou seja, um feixe de laser é emitido em uma seqüência rápida de pulsos curtos e um fotosensor capta a sua reflexão.

O sensor laser é menos sensível à luz que a câmera de vídeo, esta característica o torna mais apropriado para aplicações onde há mudanças de iluminação. Contudo, a princi-

pal desvantagem deste tipo de sensor é a necessidade de circuitos mais sofisticados para detecção devido à sua alta velocidade.

Câmera de vídeo

As câmeras de vídeo compõem os sensores de visão do robô. O processamento da imagem é feito utilizando métodos de reconhecimento de padrões.

Em geral utilizam-se duas câmeras, o que permite gerar padrões tridimensionais e calcular a distância entre o robô e os objetos.

As principais dificuldades na implementação de sensores de visão advém da sensibilidade das câmeras à luz e da complexidade envolvida com o processamento das imagens.

Sensor de contato

Estes sensores são compostos por botões que são acionados quando o robô colide em um obstáculo. Geralmente são utilizados como último recurso, em situações onde o robô não identifica os obstáculos que podem ser não reflexivos à luz infravermelha ou que absorvem os pulsos de alta freqüência do sonar.

2.3.2 Sensores de Posicionamento

Os sensores de posicionamento são utilizados para localização do robô no ambiente. São sensores de posicionamento: odômetros, bússolas, GPSs (*Global Positioning Systems*) e faróis (*beacon*).

Odômetro

O odômetro mede a distância percorrida pelo robô. Com ele é possível calcular a posição relativa do robô no ambiente.

Bússola

As bússolas têm como finalidade informar o ângulo do robô no ambiente, retornando um valor absoluto que independe do estado anterior do robô.

GPS

O GPS é utilizado para determinar a longitude, latitude e altitude em robôs de campo. Isso é feito com o auxílio de satélites em órbita da Terra.

Este tipo de sensor retorna valores absolutos, mas tem como desvantagem a ineficiência em ambientes urbanizados.

Farol

Os faróis são dispositivos instalados em pontos estratégicos do ambiente. Eles podem emitir sinais luminosos ou de rádio que tornam possível o cálculo da posição atual do robô.

Esse tipo de sensor é utilizado para eliminar erros de posicionamento causados por sensores de posicionamento que retornam valores relativos, como o odômetro.

2.3.3 Fusão Sensorial

Os sensores obtêm leituras imprecisas das distâncias entre o robô e os obstáculos ou alvos, sendo assim, torna-se necessário integrar as diversas leituras dos vários sensores instalados no robô para obter informações mais precisas. Este é o papel da fusão de sensores [6].

Há várias formas de fazer essa integração, por exemplo, o uso de técnicas como Redes Neurais Artificiais [1] [52] e Lógica Fuzzy, ou ainda métodos matemáticos como os filtros de Kalman e a fusão Bayesiana [18].

As Redes Neurais Artificiais têm como vantagem a flexibilidade e facilidade de utilização, não requerendo a elaboração de modelos matemáticos complexos. Para fusão dos sensores com uma rede neural, basta ter os dados dos sensores no formato adequado e submetê-los à rede durante a fase de treinamento [16].

2.4 Mapas

Um mapa é uma representação do ambiente. Há quatro tipos de modelos para criar mapas: modelo de decomposição em células, modelos geométricos, modelos topológicos e modelo de marcas [28, 47], os quais são descritos a seguir.

[1] No apêndiceB apresenta-se a teoria introdutória das Redes Neurais Artificiais.

2.4.1 Modelo de Decomposição em Células

Neste modelo, o ambiente é representado por meio de uma matriz de células. Os modelos de decomposição em células são métodos aproximados e podem ser aplicados em ambientes com duas ou três dimensões.

Há dois esquemas básicos e principais de decomposição do ambiente em células: enumeração da ocupação espacial e *quadtrees* ou *octrees*.

Modelagem por enumeração

A modelagem por enumeração é um método aproximado, onde a resolução está diretamente relacionada com o tamanho da célula. Isto torna o método bastante simples, principalmente para acessar um determinado ponto no ambiente. A figura 2.2 apresenta um exemplo de decomposição do ambiente em células utilizando a modelagem por enumeração.

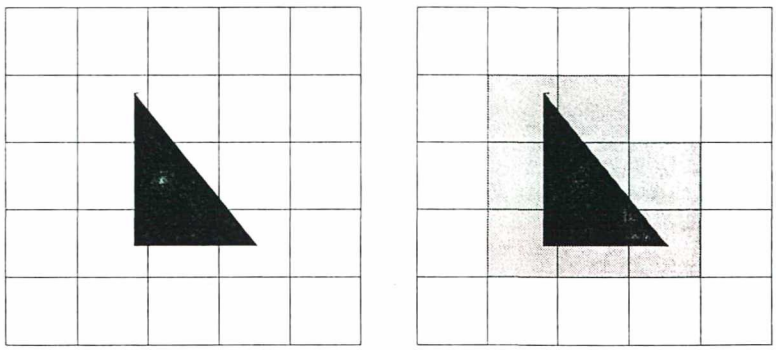


Figura 2.2: Modelagem por enumeração

Modelagem por *Quadtrees* e *Octrees*

Estes métodos são derivados do anterior, porém são mais eficientes, pois a resolução é facilmente ajustável e em uma mesma representação podem coexistir diferentes graus de resolução. As *quadtrees* e *octrees* são estruturas de dados em forma de árvore que modelam objetos em duas ou três dimensões respectivamente. A figura 2.3 apresenta a modelagem de um ambiente utilizando uma *quadtree*.

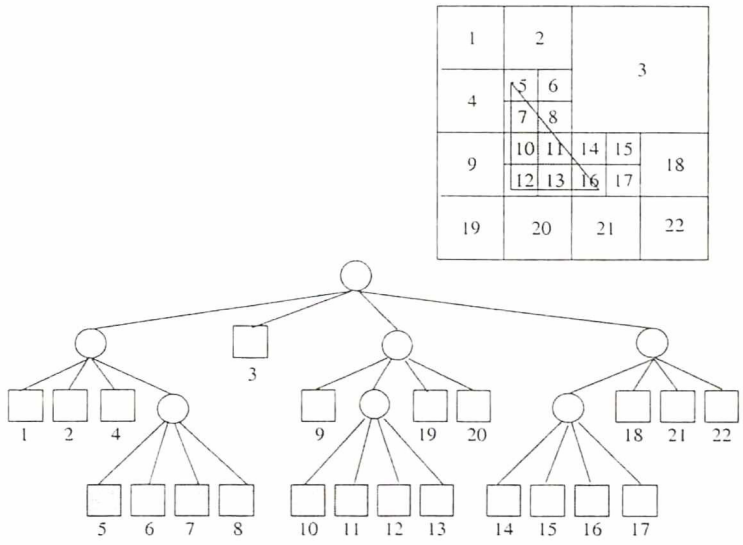


Figura 2.3: Decomposição do ambiente utilizando *quadtrees*

2.4.2 Modelo Geométrico

A geometria do ambiente é formulada a partir de um vetor (*array*) de componentes escalares simples. Tais modelos permitem a obtenção de uma boa quantidade de informações estatísticas que auxiliam na modelagem da incerteza do ambiente.

Neste tipo de modelo, a representação geométrica escolhida deve ser:

- Mínima - o número de parâmetros para descrever a entidade geométrica deve ser mínimo;
- Não ambígua - cada primitiva geométrica deve ter no máximo uma representação;
- Completa - cada primitiva geométrica deve ter no mínimo uma representação;
- Diferenciável - para permitir a linearização das equações de medida.

Uma forma de modelar os objetos geométricos com incerteza é descrevê-los por meio de uma função vetorial $g(x, p) = 0$ onde p é um vetor de parâmetros que especifica uma instância em particular do objeto geométrico representado por g.

Em robótica móvel os modelos geométricos são utilizados de duas formas diferentes, dando lugar a dois enfoques distintos, com suas aplicações e problemas. São eles: modelo geométrico determinístico e modelo geométrico estocástico.

Modelo Geométrico Determinístico

Estes modelos são definidos *a priori* e apenas as características dos objetos são modeladas (não se considera a incerteza). Eles são aplicados em planejamento de trajetória e localização do robô.

Modelo Geométrico Estocástico

O modelo estocástico não é definido *a priori*, e a incerteza é adicionada ao modelo. Estes modelos são construídos durante a operação do robô, sendo muito utilizados para a construção de mapas, além de planejamento de trajetória e localização.

Para construir o mapa, em cada posição, o robô deve ser capaz de construir uma representação geométrica local de seu ambiente combinando primitivas geométricas simples para formar primitivas geométricas globais. Por exemplo: a partir de pontos construir retas, ou combinar retas em planos.

O deslocamento do robô pelo ambiente tem uma incerteza associada e vários mapas locais são construídos de diferentes posições, estes mapas têm múltiplas ocorrências de primitivas geométricas. A fusão sensorial tem como objetivo utilizar todas estas informações para reduzir a incerteza, tanto no deslocamento como nas primitivas geométricas, tornando possível construir, a partir dos mapas locais, um mapa global do ambiente. Este tipo de fusão é feita geralmente utilizando filtros de Kalman [18, 47].

2.4.3 Modelo Topológico

A idéia básica deste modelo é representar relações entre entidades. Essa representação pode ser feita por meio de grafos onde os vértices representam as entidades e as arestas, as relações. A relação mais utilizada é a relação de adjacência.

Uma característica importante de tais modelos é a simplicidade da modelagem hierárquica do ambiente, essa característica facilita a manipulação do modelo e permite trabalhar com diferentes resoluções. Na figura 2.4 tem-se um exemplo de ambiente e o seu modelo topológico.

O ambiente é formado pelo meio externo (Ext.), por duas salas (H1 e H2) e um corredor (C1). A transição do meio externo (Ext.) para a sala H1 é feita através da porta G1. Estando em H1, é possível chegar ao corredor C1 utilizando a porta G2. De forma semelhante, utilizando G3 é possível chegar à sala H2.

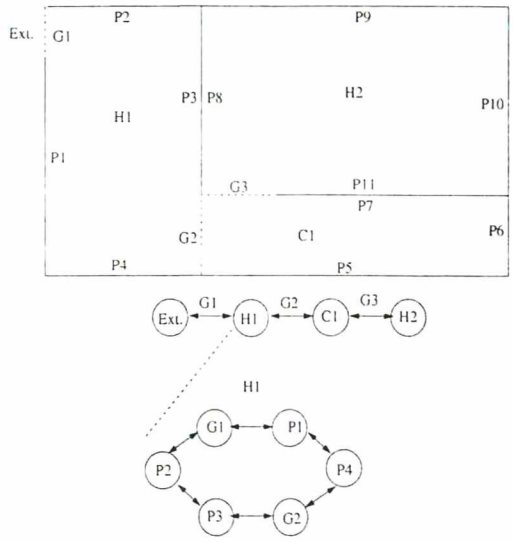


Figura 2.4: Modelo topológico

Ao refinar o modelo, as entidades são expandidas, gerando novos grafos. Assim, pode-se representar, por exemplo, a sala H1 como sendo o conjunto formado pelas paredes P1, P2, P3 e P4 e pelas portas G1 e G2. As arestas do grafo representam a relação de adjacência entre estas entidades.

2.4.4 Modelo de Marcas

Uma marca (*landmark*) pode ser qualquer objeto ou conjunto de objetos que integre o ambiente. As marcas são classificadas em dois grupos: naturais e artificiais. As marcas naturais não podem ser modificadas ou movidas (uma parede, uma porta, um edifício, uma montanha, etc.). Já as marcas artificiais podem ser movidas ou modificadas.

São 3 os tipos de objetos que podem compor uma marca:

- Objetos geométricos - estas marcas são formadas por objetos geométricos simples. Ex.: polígonos, prismas, etc.
- Objetos estruturados - são marcas formadas por objetos dotados de uma estrutura bem definida e mais complexa. Ex.: sinais de tráfego, marcas específicas para localização, portas, um cruzamento, um edifício de forma e cor determinados.
- Objetos não-estruturados - estas marcas são formadas por objetos sem estrutura bem definida. Ex.: uma árvore, uma rocha, etc.

Os modelos de marcas são utilizados para localização do robô, planejamento de ações (um evento sensorial ativa um determinado comportamento) e controle do robô (marcas que afetam a velocidade do robô, como por exemplo, um sinal de trânsito). Para o reconhecimento de uma marca no ambiente são necessários os seguintes passos: detecção, identificação e extração da informação.

2.5 Abordagem Planejada ou Deliberativa

Na *abordagem planejada*, são feitas duas considerações:

- o ambiente é estático - nenhum objeto, além do robô, move-se pelo ambiente.
- o ambiente é completamente conhecido - há o conhecimento global do ambiente, ou seja, todos os obstáculos têm sua forma e posicionamento conhecidos. Este conhecimento global está representado na forma de um mapa, como os descritos acima.

Com estas considerações, o problema da navegação resume-se a um problema geométrico de *planejamento de trajetória* [28]. Para planejar a trajetória são utilizadas as informações armazenadas no mapa.

As arquiteturas de controle desta abordagem propõem a decomposição funcional das atividades em uma seqüência de passos que leva ao cumprimento da tarefa, como apresentado na figura 2.5. Pode-se dizer que esta abordagem é *top-down* no sentido em que tenta simular o raciocínio que leva ao cumprimento da tarefa [29].

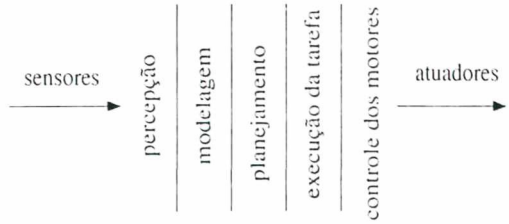


Figura 2.5: Decomposição em módulos funcionais

Primeiramente, se realiza um processamento sensorial para extrair as informações importantes do ambiente. Esta informação é integrada em um modelo. A partir deste modelo, todas as ações são planejadas e posteriormente executadas. Este tipo de decomposição é também chamada de *SMPA - Sense, Model, Plan and Act* [12].

Para planejar a trajetória de um ponto inicial a um ponto final existem diversos métodos, a maioria destes requer que o ambiente seja bidimensional e que os obstáculos sejam poligonais. Estes métodos dividem-se em três grandes classes de algoritmos [28, 47]: Métodos de Mapeamento de Caminhos (*Roadmap*), Métodos de Decomposição em Células e Método dos Campos Potenciais, descritos a seguir.

2.5.1 Métodos de Mapeamento de Caminhos (*Roadmap*)

A idéia destes métodos é encontrar a conectividade do espaço livre no qual o robô se encontra, formando uma rede de curvas unidimensional. Uma vez montada esta rede, está criado um conjunto de caminhos. Sendo assim, dados os pontos inicial e final, basta conectá-los utilizando as malhas da rede. O caminho final será composto por três sub-caminhos: o primeiro conecta o ponto inicial a um ponto da rede, o segundo é dado pelas conexões da própria rede e o terceiro, pela conexão entre a rede e o ponto final.

São métodos de Mapeamento de Caminho: grafos de visibilidade [28, 47], diagrama de Voronoi [28, 47] e método do caminho livre (*freeway*) [9].

Grafos de Visibilidade

Os grafos de visibilidade são aplicados aos espaços de configuração bidimensionais com a região C-Obstáculo poligonal. Neste método, considera-se que o robô translada com orientação fixa.

O grafo de visibilidade é um grafo (G) não dirigido onde os vértices são os pontos de configuração inicial e final e todos os vértices dos obstáculos. Os vértices são conectados entre si por arestas que são arestas dos obstáculos do espaço de configuração ou linhas que unem pares de vértices dos obstáculos que não sobrepõem o interior da região C-obstáculo. A figura 2.6 apresenta um exemplo de grafo de visibilidade.

Diagramas de Voronoi

O diagrama de Voronoi também aplica-se a espaços de configuração bidimensionais, com obstáculos poligonais em que o robô translada com orientação fixa. O espaço livre deve ser cercado por uma região poligonal.

O diagrama de Voronoi é construído da seguinte forma: para cada ponto no espaço livre, calcula-se a distância até o obstáculo mais próximo. A distância é zero nos limites das bordas

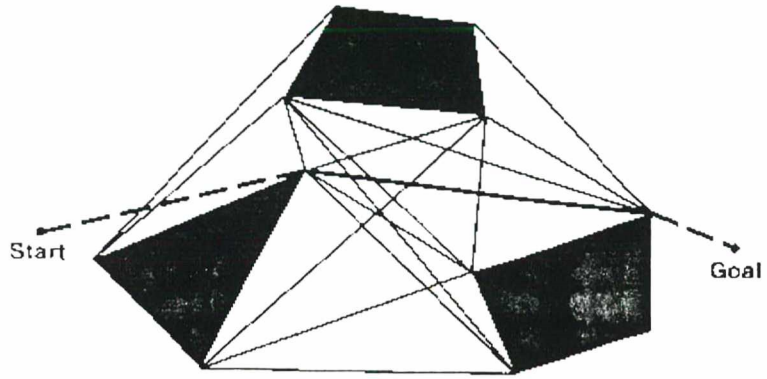


Figura 2.6: Grafo de visibilidade

dos obstáculos e aumenta gradualmente com o afastamento. O retângulo formado pelo limite do espaço de configuração também é considerado um obstáculo. Este procedimento é chamado de retração.

Ao final, a superfície é formada então por uma malha com cumes afinados que são os pontos eqüidistantes de dois obstáculos. O diagrama de Voronoi consiste nesta linha de pontos. A figura 2.7 mostra um diagrama de Voronoi para um conjunto de obstáculos poligonais.

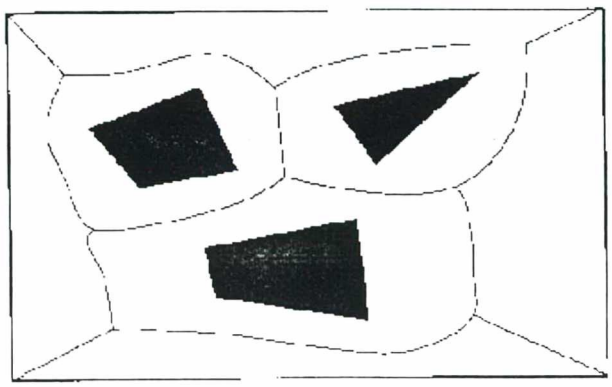


Figura 2.7: Diagrama de Voronoi

Método do Caminho Livre (*Freeway*)

O método do caminho livre [9], encontra um caminho livre de colisão para robôs poligonais, que têm movimento de rotação e translação, em ambientes bidimensionais com obstáculos poligonais.

A idéia do método é representar o espaço livre através de cones sobrepostos chamados de *cones generalizados*. Um cone generalizado é formado pela varredura de uma seção cruzada

bidimensional ao longo de um eixo, chamado *espinha*.

Para construir a representação do espaço livre, examinam-se todos os pares de arestas dos obstáculos poligonais. Se as arestas definem um *caminho livre* (*freeway*) natural através do espaço então elas são usadas para construir o cone generalizado. Representado o espaço livre, os cones são examinados em pares buscando a interseção entre as espinhas. Um grafo, chamado de *rede de caminhos livres* (*freeway net*) é construído e pesquisado para encontrar o caminho que leve o robô da configuração inicial à final. A figura 2.8 apresenta um exemplo de ambiente com o espaço livre representado por cones generalizados.

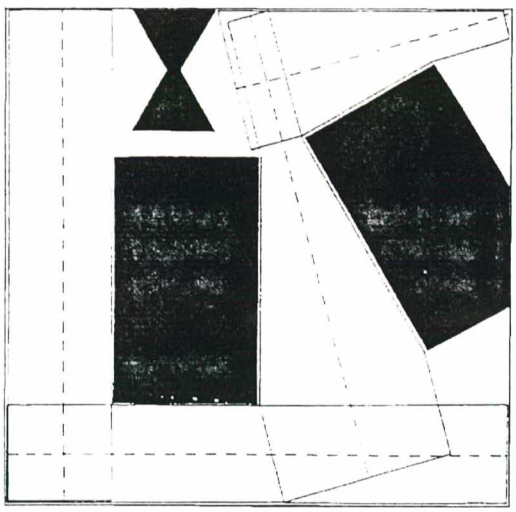


Figura 2.8: Representação do espaço livre por cones generalizados

A grande contribuição do método do caminho livre é prover uma descrição compreensivel da conectividade do espaço livre sem precisar representar exaustivamente a região C-obstáculo.

O método é rápido em ambientes com poucos obstáculos, mas não é aplicável em ambientes lotados. Além disso, o método é incompleto: há casos em que não retorna o caminho, ainda que ele exista. Além disso, incorpora muitas escolhas empíricas, por exemplo: a espinha em cada cone é o bisector das linhas compostas pelas arestas dos obstáculos e o cone é estendido até os pontos finais das arestas por linhas paralelas à espinha [28].

2.5.2 Método de Decomposição em Células

O método de decomposição em células [28, 47] consiste em:

- Decompor o espaço de configuração em regiões, denominadas *células*;

- Construir um grafo não-dirigido, chamado de *grafo de conectividade*, onde os vértices são as células e as arestas representam a relação de adjacência entre elas;
- Determinar qual é a célula inicial e final da configuração e buscar um caminho utilizando algum método de busca, no grafo de conectividade, entre estas células. Este caminho é denominado *canal*;
- Com a seqüência de células encontradas, calcular um caminho dentro de cada célula, passando pelos pontos médios das bordas formadas pelas células anteriores e posteriores.

Há dois tipos de métodos de decomposição em células: o método de decomposição em células exatas e o de decomposição em células aproximadas.

Decomposição em Células Exatas

Este método consiste em decompor o *espaço livre* do robô em uma coleção de células não sobrepostas. A união desta coleção de células representa *exatamente* o espaço livre.

As células geradas pela decomposição devem ter um formato simples, para facilitar a busca por um caminho entre duas células e o teste das adjacências. Em ambientes bidimensionais com obstáculos poligonais as células podem ser geradas utilizando a *decomposição poligonal convexa* ou a *decomposição trapezoidal*.

Na decomposição poligonal convexa, a região que delimita o espaço de configuração também é considerada como obstáculo. Os vértices dos obstáculos são interligados em formas triangulares. A figura 2.9 apresenta um exemplo de ambiente decomposto em células poligonais convexas.

A decomposição trapezoidal, ou também chamada decomposição vertical, é feita traçando-se a partir dos vértices no máximo dois segmentos de retas verticais. A região que delimita o espaço de configuração é utilizada como limite para estas retas, como mostra a figura 2.10(a). Após a decomposição o grafo de conectividade é montado representando as células adjacentes, como apresentado em (b).

Para fazer a busca no grafo pode-se utilizar vários métodos de busca como o algoritmo A*[2], ou ainda o algoritmo de Dijkstra [39].

[2]Descrito no apêndice C, seção C.3.

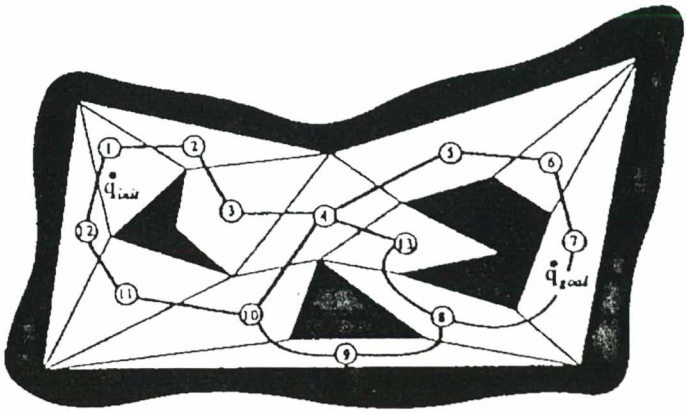


Figura 2.9: Decomposição em células poligonais convexas

Terminado o processo de busca é retornado ou o canal que leva da configuração inicial à configuração final, se este existir, ou insucesso, caso este não exista. Com a descrição do canal, o caminho que leva o robô da configuração inicial à final é dado pelos pontos inicial e final e pelos pontos médios calculados sobre as bordas das células do canal.

Decomposição em Células Aproximadas

Neste método o *espaço de configuração* é particionado em células que, diferentemente do método exato, têm formato pré-estabelecido. Sendo assim, o espaço livre não é exatamente representado.

O ajuste no tamanho das células possibilita melhorar a adaptação à geometria dos obstáculos. Entretanto, a escolha prévia de células grandes, apesar de reduzir a quantidade de memória necessária para o armazenamento e o tempo de execução, pode causar perda de precisão, fazendo com que possíveis caminhos não sejam encontrados. Já células muito pequenas permitem um melhor dimensionamento do espaço livre, porém o custo computacional torna-se elevado. Sendo assim, deve-se manter um compromisso entre o tempo de processamento e a busca por possíveis caminhos.

Uma forma de manter esse compromisso é utilizar a decomposição hierárquica do espaço de configuração. Iniciando-se com células grandes que são refinadas localmente até que uma boa representação do espaço livre seja obtida. Esta decomposição hierárquica, em ambientes bidimensionais, pode ser feita utilizando *quadtress*, como mostra a figura 2.11.

As células geradas na decomposição são classificadas em:

- Vazias - se e somente se o seu interior não intercepta a região C-obstáculo;

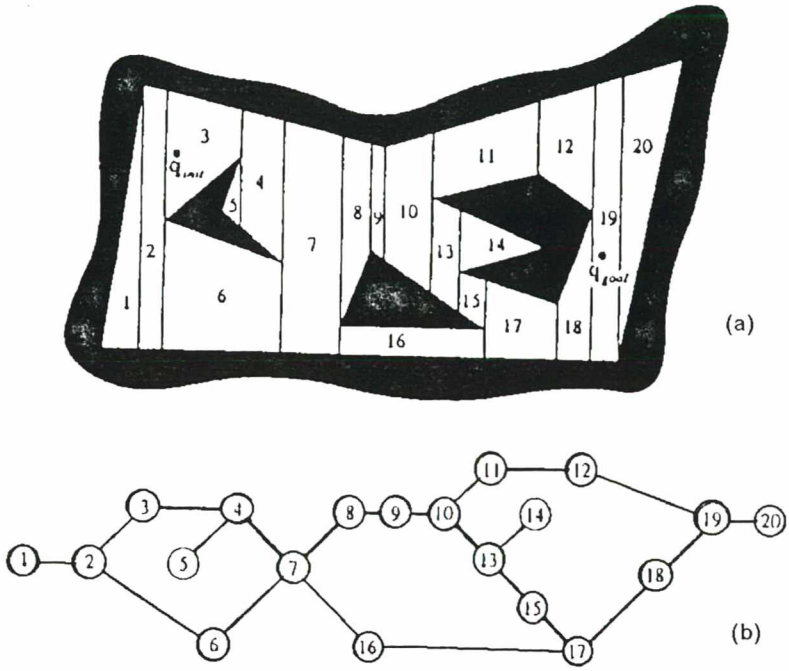


Figura 2.10: Decomposição em células trapezoidais

- Cheia - se e somente se o seu interior estiver contido na região C-obstáculo;
- Mista - caso contrário.

O grafo de conectividade será construído com células vazias ou mistas. Se o canal retornado, após a busca no grafo, contiver somente células vazias, o canal é chamado de *Canal Vazio* (*E-Channel*), e o caminho que leva da configuração inicial à final é um caminho livre.

Se o canal possuir células mistas, é denominado *Canal Misto* (*M-channel*), então as células mistas devem ser decompostas para encontrar um canal vazio, que assegure ao robô a navegação livre de colisão.

Além da decomposição hierárquica utilizando *quadtrees*, há uma série de outro tipos de decomposição. Pode-se utilizar uma matriz de células de tamanho fixo, no qual as células são marcadas como vazias ou cheias. Assim, cada célula está conectada a 4 ou 8 células vizinhas, dependendo da inclusão ou não das células diagonais. O caminho pode ser procurado, então, por algum método de busca[3].

Outra forma é utilizar a transformada da distância [20, 58] onde cada célula com tamanho fixo recebe o valor da distância até o ponto objetivo. Células que sobrepõem obstáculos são

[3]Veja apêndice C.

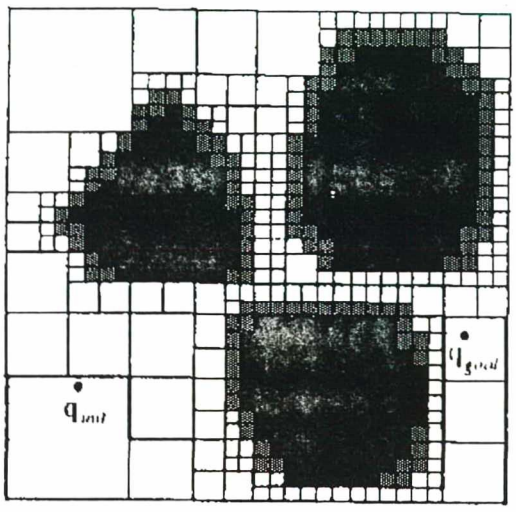


Figura 2.11: Decomposição em células aproximadas utilizando *quadtrees*

assinaladas com valor infinito (∞). O caminho é calculado como uma seqüência de células com valor decrescente até o ponto objetivo.

Todos os métodos têm suas vantagens e desvantagens, portanto, deve-se observar o compromisso entre o tempo de processamento e a representação do espaço livre.

Em geral, o método de decomposição em células aproximadas é menos complexo e os algoritmos para decomposição são mais simples de implementar. Por estas razões é mais utilizado do que o método de decomposição em células exatas.

2.5.3 Método dos Campos Potenciais

Neste método, o robô, que é representado como um ponto no espaço de configuração, torna-se uma partícula sob a influência de duas forças: uma força repulsiva, que o empurra para longe dos obstáculos, e uma força atrativa, que o puxa para o ponto objetivo.

Estas duas forças são combinadas por uma função potencial (geralmente utiliza-se a soma dos potenciais atrativo e repulsivo) que descreve o *campo potencial*. A figura 2.12 mostra um exemplo de campo potencial gerado para um ambiente com obstáculos poligonais. O campo com potencial atrativo é mostrado em (b) e em (c), o potencial repulsivo. A soma dos dois potenciais é apresentada em (d). Em (e), pode-se perceber os contornos do campo potencial formado e o caminho gerado seguindo o gradiente descendente. Em (f), apresenta-se a matriz do gradiente descendente, com os vetores sobre o espaço livre.

Este método foi proposto inicialmente como um método reativo que não necessita da

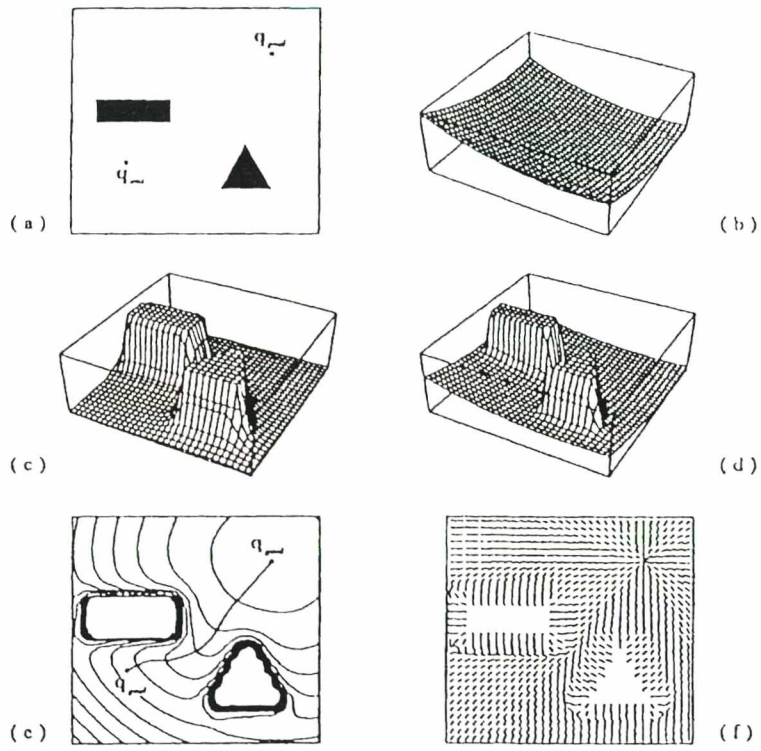


Figura 2.12: Campos potenciais

construção de um modelo do mundo. A função potencial fornece, a cada iteração, a direção mais promissora para o robô. Por sua característica de atuar como um procedimento de otimização descendente, também é muito rápido. Contudo, pode emperrar em uma situação de mínimo local[4] no campo potencial.

Para solucionar o problema dos mínimos locais, pode-se combinar o método de campos potenciais com técnicas de busca[5] [39]. Entretanto, mesmo com o uso dessas técnicas, os mínimos locais permanecem como uma importante causa da ineficiência do método.

A solução dos mínimos locais com estratégias de busca faz o método perder sua característica reativa. Se um modelo do mundo for criado *a priori*, o método passa a ser planejado, se o mapa for sendo construído durante a execução, o método passa a ser híbrido.

Outra forma de lidar com os mínimos locais, mantendo o método reativo, é definir uma função potencial com poucos ou nenhum mínimo local. A escolha desse tipo de função, em geral, não é genérica para qualquer configuração, ficando atrelada à distribuição e ao formato dos obstáculos.

[4]Situação de estabilização que não representa a solução.
[5]Veja apêndice C.

É importante ressaltar que os métodos apresentados aqui não são únicos, existem diversas variações. Contudo, a maioria deles é aplicada a espaços de configuração bidimensionais com obstáculos poligonais, nos quais o robô apenas translada. Tais espaços de configuração diferem em muito de ambientes reais tridimensionais onde, em geral, os obstáculos não são poligonais.

As grandes dificuldades encontradas na abordagem planejada são:

- O modelo do ambiente não é completamente preciso - as medidas dos sensores apresentam ruído (duas leituras obtidas em seqüência, sem alteração do ambiente, não fornecem o mesmo valor) e a operação dos sensores pode obter medidas falsas (causadas por reflexão especular, por exemplo) [6, 29].
- A abordagem é separada em duas etapas, inicialmente há uma etapa de planejamento, seguida da etapa de execução. Caso ocorram mudanças no ambiente após a fase de planejamento, o mapa tornar-se-á impreciso e o robô possivelmente colidirá.
- Quando aplicadas a ambientes com características dinâmicas, o sistema deve modelar o ambiente e pré-planejar todos os movimentos antes de executá-los, ocasionando um problema denominado de *Problema da Qualificação* (*Qualification Problem*) [1], onde o sistema permanecerá eternamente planejando. Uma forma de contornar isso é fazendo suposições grosseiras sobre a dinâmica do ambiente, que além de não refletirem a dinâmica natural, tornam o sistema extremamente frágil.

Desta forma, esta abordagem é mais adequada para ser utilizada em ambiente estáticos, onde a localização dos objetos ou obstáculos é conhecida e não se altera.

2.6 Abordagem Reativa ou Baseada na Leitura de Sensores

A *abordagem reativa* [10, 47] é oposta a abordagem planejada no sentido em que uma ação é produzida como resultado de um estímulo recebido, ou seja, o robô utiliza as informações vindas dos sensores para executar uma ação. Nenhum modelo do ambiente é gerado, visto que não há planejamento prévio das ações. Isto permite ao robô navegar em ambientes complexos e desconhecidos e que apresentem características dinâmicas.

Cada ação é responsável por um aspecto particular do controle do robô, para realizar uma determinada tarefa, isto é, cada ação corresponde a um *comportamento* que encapsula

uma tarefa. As tarefas podem ser: desviar obstáculos, seguir paredes, caminhar em frente, alcançar alvo.

Os comportamentos operam assincronamente e em paralelo, baseando-se unicamente nos dados sensoriais relevantes para a tomada de decisão, assim, o processamento da percepção do robô fica distribuído entre os diversos comportamentos.

A realização de tarefas mais complexas é obtida através de comportamentos mais elaborados, que devem emergir da ação conjunta de comportamentos primitivos [29]. Isso caracteriza esta abordagem como uma abordagem *bottom-up*.

2.6.1 Arquitetura de Subsunção (*Subsumption*)

Esta arquitetura é a primeira arquitetura reativa e a mais conhecida [10]. Ela baseia-se na *decomposição vertical* do planejamento em uma coleção de camadas concorrentes, chamadas *competências*. Cada competência é uma especificação informal de uma classe de *comportamentos* desejados, como mostra a figura 2.13.

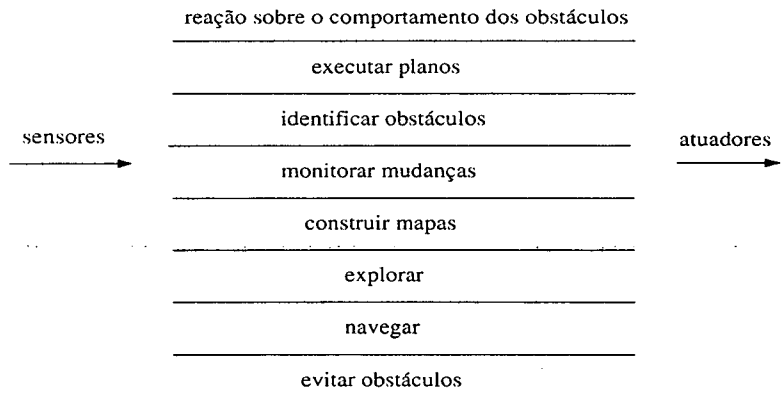


Figura 2.13: Arquitetura de subsunção

São níveis de competência:

1. Evitar contato com objetos (em repouso ou em movimento);
2. Navegar sem destino pelo ambiente sem colidir com obstáculos;
3. Explorar o ambiente identificando lugares cuja a distância pareça alcançável e ir em direção a eles;
4. Construir mapas do ambiente e planejar trajetórias de um lugar para outro;
5. Detectar mudanças no ambiente estático;

6. Raciocinar sobre o ambiente em termos de obstáculos identificáveis e realizar tarefas relacionadas com eles;

7. Formular e executar planos que impliquem modificar o estado do ambiente para a forma desejada;

8. Raciocinar sobre o comportamento dos objetos no ambiente e modificar os planos de acordo com as mudanças.

Cada nível de competência inclui os níveis anteriores e quanto maior o nível de competência, mais específica é a classe de comportamentos desejados, dado o aumento das restrições.

Um nível de competência corresponde a uma camada do sistema de controle. Assim, o nível de competência 0 constitui o nível de controle do robô. Novos níveis de controle podem ser adicionados sobre o nível 0. O próximo nível, o nível de competência 1, pode examinar os dados do nível 0 e pode injetar dados nas interfaces internas deste nível, suprimindo o fluxo normal de dados. Os níveis inferiores não tem conhecimento dos níveis superiores e podem trabalhar em objetivos distintos de forma concorrente, como mostra a figura 2.14.

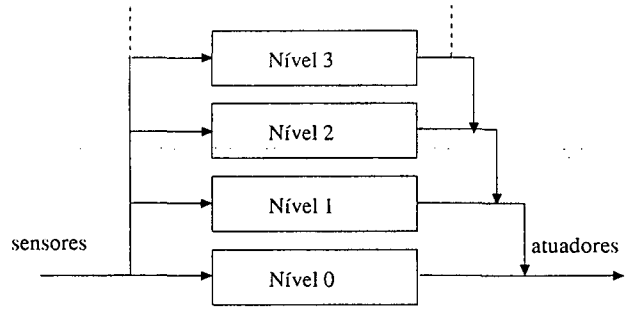


Figura 2.14: Níveis de controle da arquitetura de subsunção

A capacidade das camadas de mais alto nível de suprimir as saídas das camadas dos níveis inferiores é que dá nome a arquitetura.

Observa-se nesta arquitetura que as camadas operam assincronamente. Cada camada pode trabalhar para a realização de uma meta, concorrentemente com as demais camadas, enquanto o mecanismo de supressão intermedia as ações que serão executadas. Além disso, como cada camada utiliza sensores específicos para a tomada de decisão, a fusão dos sensores é, em parte, ignorada.

Contudo, para a implementação de cada nível de controle, faz-se necessária, porém não obrigatória, a decomposição funcional, como na abordagem planejada. Em [10] é proposto

o uso de máquinas de estados finitos, uma para cada nível, onde a comunicação é feita através de mensagens e não há nenhum tipo de memória compartilhada.

A grande vantagem da abordagem reativa está na rapidez com que responde a um estímulo, uma vez que as capacidades de percepção e ação são enfocadas. Esta característica assegura, aos métodos dessa abordagem, a autonomia no que diz respeito às ações do robô.

Sistemas reativos têm habilidade em desviar obstáculos, seguir paredes, passar através de portas. Porém, integrar estas habilidades em um plano maior torna-se uma tarefa complicada, pois nenhum tipo de análise mais complexa dos dados sensoriais é feita, ou seja, não são realizadas operações cognitivas de alto nível e não são incorporadas habilidades de conhecimento do ambiente. Isto torna sua aplicação difícil em tarefas mais elaboradas. Também não há como assegurar que o melhor caminho seja encontrado e seguido, o que diminui a eficiência da abordagem.

Como a capacidade cognitiva é ignorada, o comportamento do robô limita-se a imitar formas de vida inferiores e como informações globais não são armazenadas e utilizadas, a imprevisibilidade das situações com as quais o robô se defronta traz dificuldades à implementação de tais sistemas.

Somente com a inclusão de memória e de alguma forma de representação dinâmica do conhecimento seria possível exaltar e estender o comportamento para domínios de problemas mais significativos.

Vários trabalhos têm empregado técnicas de inteligência artificial no desenvolvimento de sistemas reativos. Técnicas como redes neurais artificiais [25, 38, 46], lógica fuzzy [13], aprendizado por reforço [8, 34] e redes neuro-fuzzy [17] têm apresentado bons resultados quando aplicadas a tais sistemas.

2.7 Abordagens Híbridas

Nenhuma das abordagens acima é completamente satisfatória quando tomadas isoladamente, mas implementadas juntas podem produzir sistemas robustos, flexíveis e genéricos, enfim, sistemas mais "inteligentes" [1], dado que cada uma das abordagens enfoca partes distintas e fundamentais do problema da navegação autônoma.

A abordagem planejada apresenta formas para a integração do conhecimento do ambiente e utiliza este conhecimento para obter um plano antes da execução. Porém, replanejar

com estes métodos em níveis onde os dados dos sensores são unidos em modelos do ambiente é insuficiente.

A abordagem reativa, por outro lado, trabalha muito bem a questão do processamento sensorial imediato, mas não o integra para formular um conhecimento global acerca do ambiente.

Assim sendo, pode-se procurar formas de unir as melhores características de cada abordagem em *abordagens híbridas*.

A união das abordagens planejada e reativa pode ser feita de duas formas. Através do *pré-mapeamento* do ambiente e do uso da leitura dos sensores para o desvio de obstáculos dinâmicos ou inexistentes no mapa e através do uso das leituras dos sensores para *construir mapas em tempo de execução*.

2.7.1 Pré-Mapeamento

Neste enfoque, um mapa do ambiente é construído e serve como base para um primeiro planejamento. Ao executar o plano, o robô obtém os dados dos sensores e ativa comportamentos reativos buscando desviar dos obstáculos dinâmicos ou não previamente conhecidos. Ao sair das situações perigosas, um novo plano é feito utilizando o mapa. Contudo este mapa não é atualizado.

Uma das primeiras arquiteturas híbridas, proposta em [1] é denominada *AuRA - Autonomous Robot Architecture* e combina reação e planejamento e tem capacidade de aprendizado.

Arquitetura AuRA

Nesta arquitetura os comportamentos são encapsulados em esquemas, que não possuem nenhum mecanismo de arbitragem e são divididos em esquemas motores e esquemas perceptivos.

Os esquemas motores são usados para descrever o conjunto de ações que o robô pode executar no ambiente, tais como: mover-se em frente, mover-se para o objetivo, evitar obstáculo, permanecer em um caminho. A saída destes esquemas é um vetor velocidade que representa a direção e a velocidade com as quais o robô deve mover-se dadas as condições atuais do ambiente.

Os esquemas perceptivos estão embutidos nos esquemas motores e fornecem a estes as

informações necessárias para calcular a reação diante do ambiente. Estes esquemas extraem do ambiente somente as informações necessárias para realizar a ação motora, nenhum modelo do ambiente é criado, contudo podem utilizar informações obtidas *a priori*.

Quanto ao conhecimento do ambiente, este pode ser persistente ou temporário. O conhecimento persistente consiste nas informações *a priori* acerca do ambiente, tais como modelo dos objetos, do espaço livre ou mesmo do próprio robô. Esta informação é armazenada em uma memória de longo prazo. Já o conhecimento temporário consiste na informação adquirida dinamicamente pelo robô enquanto este navega pelo ambiente. Com esta informação é construído um modelo dinâmico local do ambiente que somente é utilizado caso o controle reativo falhe. Estas informações são armazenadas em uma memória de curto prazo e são perdidas à medida que o robô se desloca no ambiente.

Para a construção dos mapas dinâmicos, a informação sensorial obtida é enviada para o subsistema cartográfico, e o planejamento global da trajetória é feito utilizando o conhecimento armazenado na memória de longo prazo pelo subsistema de planejamento. O controle reativo do robô utiliza o método dos campos potenciais.

Este tipo de abordagem é simples pois não necessita da fusão de mapas locais para a criação de um mapa global, atividade que geralmente é custosa e complexa.

Sempre que o ambiente sofrer grande alteração, um novo pré-mapeamento deve ser executado. É possível criar uma base de dados com vários modelos e utilizar modelo de marcas para identificação da localização do robô, estendendo sua ação para vários ambientes.

2.7.2 Construção e Manutenção do Mapa em Tempo de Execução

Outro enfoque possível é construir e atualizar o mapa em tempo de execução, utilizando os dados dos sensores para a construção de mapas locais que posteriormente são unidos em um mapa global do ambiente.

A maior motivação deste enfoque é dada à manutenção, uma vez que é possível construir modelos *a priori* do ambiente, mas estes raramente são estáticos.

Contudo, alguns fatores impõem limitações práticas às habilidades do robô de aprender e usar modelos precisos, tornando a tarefa de aquisição do modelo do ambiente difícil de ser resolvida [49]:

- Sensores - as condições de operação dos sensores podem resultar em perda, falha ou

informações irrelevantes, não permitindo obter a exata localização dos obstáculos.

- Limitação da percepção - a faixa de percepção de muitos sensores é limitada a uma zona próxima ao robô. Para adquirir informações globais, o robô precisa explorar o ambiente.

- Ruído na leitura dos sensores - as medições dos sensores estão corrompidas por ruídos. Apesar da distribuição destes raramente ser Gaussiana, esta questão pode ser tratada com técnicas específicas de filtragem de sinais, como por exemplo os filtros de Kalman [18], onde o ruído é representado por uma matriz de covariância.

- Imprecisão dos movimentos - os movimentos do robô são imprecisos, ocasionando erros de odometria freqüentes.

- Complexidade e dinâmica - os ambientes são, em geral, complexos e dinâmicos, o que torna impossível manter modelos exatos dos mesmos.

- Restrições temporais - as questões de tempo de execução requerem que o modelo do ambiente seja simples, acessível e fácil de manipular.

A construção do mapa pode ser feita utilizando mapas de situação, mapas geométricos, mapas topológicos ou mapas ocupacionais.

Mapas de situação

Nos mapas de situação, descritos em [27], o espaço livre é obtido da seguinte forma: a medida que o robô se desloca, as leituras são obtidas. As leituras similares são agrupadas em classes, utilizando um algoritmo de classificação adaptativo. Essas classes formam áreas locais, chamadas *áreas de situação*. Após identificadas as áreas de situação, o mapa de situação é construído na forma de um grafo, onde os vértices são as áreas e as arestas são as transições de uma área para outra. Contudo, a criação de tais mapas restringe-se à ambientes estáticos, uma vez que a presença de obstáculos dinâmicos pode alterar as leituras e prejudicar o agrupamento destas em classes.

Mapas geométricos

Os mapas geométricos consistem em transformar os dados dos sensores em formas idealizadas (como superfícies e retas). A construção deste tipo de mapa é bastante imprecisa,

devido à falta de precisão nas leituras dos sensores e à necessidade da fusão de mapas locais em mapas globais, levando a resultados imprecisos. São utilizados filtros de Kalman [18] para propagar a incerteza da posição do robô e das características geométricas. Após construído o mapa, métodos, como os apresentados no início do capítulo, são utilizados para o planejamento de trajetória. Contudo este tipo de mapa, em geral, não é muito utilizado.

Mapas topológicos

Nos mapas topológicos [29], o ambiente é representado na forma de um grafo onde os nós correspondem às situações, lugares ou marcas no ambiente. Os nós são conectados por arestas que indicam a existência de um caminho direto entre eles. Desta forma, tais mapas não apresentam uma representação explícita do espaço livre, contudo, são compactos, pois o que determina sua resolução é a complexidade do ambiente.

Mapas ocupacionais

Esta técnica consiste em decompor o ambiente em células e associar a cada uma delas a probabilidade de ocupação. A incerteza em torno da posição do obstáculo é representada por uma distribuição espacial destas probabilidades. Para o cálculo das probabilidades pode-se utilizar o método Bayesiano ou a regra de inferência de Dempster-Shafer como em [37].

Mapas ocupacionais com distribuição de probabilidades são interessantes quando aplicados ao desvio de obstáculos, pois geram uma representação explícita do espaço livre. Além disso, apresentam-se como uma poderosa ferramenta para lidar com a imprecisão dos dados dos sensores. Contudo, sua aplicação em ambientes dinâmicos ou superlotados compromete possíveis espaços livres, devido à questão do tamanho da célula e à necessidade de atualização constante.

Outra forma de gerar mapas ocupacionais é através de *mapas de acumulação*, onde é armazenado o número de vezes que o sensor indicou a existência de um obstáculo. Cada vez que o sensor lê o obstáculo o valor da célula é acrescido em 1, contudo não garante uma representação explícita do espaço livre, pois uma célula que contém 0 não necessariamente é uma célula vazia, uma vez que todas as células da matriz inicialmente têm valor 0. Para saber se uma célula está ocupada é necessário verificar sua vizinhança.

Vários trabalhos têm sido apresentados com a união de dois tipos de mapas. Em [49], mapas ocupacionais e topológicos são unidos em uma abordagem que utiliza redes neurais artificiais para interpretar os valores obtidos dos sensores, mapeando-os em probabilidades para a construção do mapa ocupacional. O mapa ocupacional é então particionado em regiões, separadas por linha críticas. Essas regiões, chamadas de regiões topológicas, são a base para a geração do mapa topológico.

Outra implementação mesclando dois tipos de mapas, pode ser vista em [57]. Nesta abordagem, um modelo global, representado em um mapa geométrico, é construído utilizando uma base de dados de referência, criada a partir de objetos identificáveis (paredes, mesas, cadeiras e pessoas). Mapas locais são construídos com mapas ocupacionais. A informação contida nestes mapas locais é identificada e transformada em descrições geométricas que atualizam o modelo global. O robô possui sonares para a construção dos mapas locais e câmaras de vídeo para a identificação dos objetos. Como há a identificação dos objetos, esta técnica pode ser aplicada a ambientes dinâmicos.

A principal questão no desenvolvimento de sistemas híbridos é equilibrar planejamento e reatividade. É preciso saber quando seguir o plano e quando agir reativamente.

As pesquisas em abordagens híbridas caminham em direção à integração de novos campos, tais como neurociência e psicologia, buscando conhecer melhor as capacidades reativa, instintiva e cognitiva dos animais para aplicá-las em sistemas computacionais. Capacidades de identificação de objetos, localização espacial e planejamento sugerem a necessidade de formas de representação do conhecimento e de capacidades cognitivas as quais permitem a predominância do comportamento "inteligente" do robô.

2.8 Conclusão

Neste capítulo introduziu-se a teoria relativa aos robôs móveis autônomos, enfocando principalmente as noções necessárias à compreensão do problema e as questões relacionadas à navegação autônoma.

A partir da revisão bibliográfica foi possível mostrar que as principais dificuldades encontradas no desenvolvimento de robôs móveis autônomos estão relacionadas com as características dos ambientes reais, onde uma série de fatores dificultam a criação de modelos precisos, tornando necessário trabalhar a questão da incerteza.

Inicialmente, diversas simplificações foram consideradas e vários métodos para ambientes estáticos foram desenvolvidos e empregados na solução do problema da navegação. Esta tentativa inicial constitui a abordagem planejada.

Contudo, ambientes reais raramente são estáticos. Buscando mudar o enfoque até então utilizado, surgiu a abordagem reativa, onde o uso de resposta ao estímulo possibilitou o tratamento simplificado do problema. Apesar de inicialmente serem vistas como a solução final ao problema da navegação, tais arquiteturas mostraram-se imperfeitas, pois para obter-se algo melhor do que somente a navegação aleatória, faz-se necessário ter ou armazenar informações acerca do ambiente.

Desta forma, deu-se o surgimento de abordagens híbridas, que permitem ao robô tratar com ambientes dinâmicos reais, armazenando informações e reagindo às situações imprevistas.

São duas as principais linhas de pesquisa em abordagens híbridas. Uma delas é unir as abordagens reativa e planejada com enfoque dado ao pré-mapeamento do ambiente, onde há alguma informação do ambiente armazenada, permitindo a localização do robô em um contexto global, enquanto localmente utiliza-se a reatividade que lhe permite desviar de obstáculos não mapeados.

A outra linha trata da construção e manutenção do mapa do ambiente em tempo de execução. Este enfoque permite armazenar as informações obtidas pelos sensores em um mapa. Este enfoque ainda está distante da perfeição, contudo, bons resultados têm sido obtidos principalmente com o uso de mapas ocupacionais.

É necessário ressaltar que o emprego de técnicas de inteligência artificial em problemas de robótica móvel e os avanços em áreas correlatas, tais como: visão computacional, processamento de sinais e engenharia e ciência dos materiais têm auxiliado no desenvolvimento de novos robôs e permitido a criação de sistemas mais robustos e flexíveis.

Nos próximos capítulos são apresentadas as implementações e análises de desempenho das três abordagens aqui apresentadas.

Capítulo 3

Implementação de Métodos das Abordagens Isoladas

3.1 Introdução

Neste capítulo apresentam-se os métodos das abordagens planejada e reativa implementados isoladamente. Estes métodos têm por objetivo preparar o escopo necessário à implementação dos métodos híbridos que serão descritos no capítulo 4.

Na implementação da abordagem planejada, optou-se pelo uso do método de decomposição em células aproximadas visando a utilização de mapas ocupacionais na implementação da abordagem híbrida. Para a abordagem reativa, implementou-se a arquitetura de subsunção utilizando a técnica de Redes Neurais Artificiais[1] para simular os comportamentos.

As implementações foram realizadas em linguagem C e C++ [5, 43] e o simulador Khepera[2] [33] foi utlitizado para os testes.

3.2 Abordagem Planejada - Método de Decomposição em Células Aproximadas

Como visto no capítulo 2, a abordagem planejada é utilizada em ambientes estáticos. Um ambiente estático pode ser modelado por um mapa. Uma forma de criar um mapa é

[1] Descrita em detalhes no apêndice B.
[2] Descrito no apendice A.

decompor o ambiente em regiões denominadas células. Se o ambiente for bidimensional, o modelo será dado por uma matriz bidimensional de células, como em [36].

A seguir, apresenta-se o processo de mapeamento utilizado na implementação dos métodos da abordagem planejada.

3.2.1 Mapeamento do Ambiente

O mapeamento do ambiente é feito particionando-o em células, onde cada célula $P(x, y)$ (sendo x a coluna e y a linha) representa um conjunto de pontos do ambiente real. A figura 3.1 apresenta um ambiente com 18 divisões horizontais e 18 divisões verticais, perfazendo um total de 324 células, iniciando no ponto $P(0, 0)$ e terminado em $P(17, 17)$.

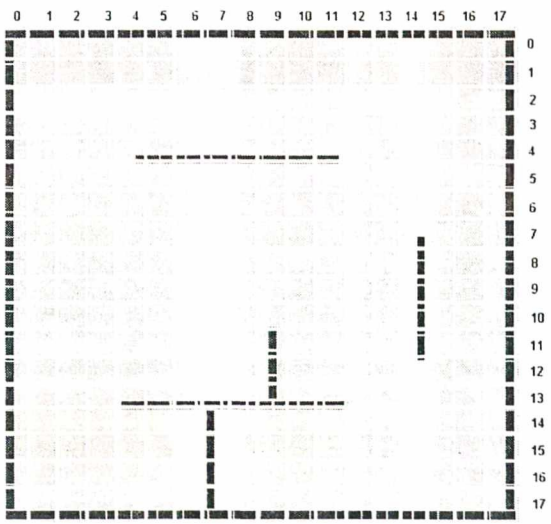


Figura 3.1: Decomposição do ambiente em células

A matriz de células é então preenchida com $0's$ e $1's$, sendo que uma célula é marcada com 1 quando sua intersecção com a região de obstáculos é não nula, ou seja, há obstáculos nos pontos que esta célula representa. Caso a intersecção seja nula - não há obstáculos nos pontos do ambiente delimitado pela célula - esta é então assinalada com 0. A figura 3.2 mostra o processo de mapeamento do ambiente da figura 3.1 para uma matriz de células.

Quando um mesmo obstáculo ocupa duas ou mais células, todas estas serão assinaladas com 1. Como visto na figura 3.1, os obstáculos são paredes formadas por pequenos tijolos, quando a linha que delimita as células sobrepõe um tijolo, fazendo com que ele pertença à duas células distintas, como visto nas células $P(4, 4)$ e $P(4, 5)$, ou nas células $P(14, 7)$ e $P(15, 7)$ as duas serão assinaladas com 1. Caso o tijolo não esteja localizado sobre a linha

```
0 1 2 3 4 5 6 7 8 9101112131415161 7

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1   0
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1   1
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1   2
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1   3
1 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 1   4
1 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 1   5
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1   6
1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1   7
1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1   8
1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1   9
1 0 0 0 0 0 0 0 0 1 0 0 0 0 1 1 0 1 10
1 0 0 0 0 0 0 0 0 1 0 0 0 0 1 1 0 1 11
1 0 0 0 0 0 0 0 0 1 0 0 0 0 1 1 0 1 12
1 0 0 0 1 1 1 1 1 1 1 1 1 0 0 0 0 1 13
1 0 0 0 1 1 1 1 1 1 1 1 1 0 0 0 0 1 14
1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 15
1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 16
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 17
```

Figura 3.2: Matriz de células

que delimita a célula, ou seja, ocupa exatamente uma célula, somente esta será assinalada com 1, como visto na célula $P(9, 10)$ da figura 3.2.

O robô utilizará uma seqüência de células livres para descrever a trajetória, dada esta característica, o tamanho da célula deve, necessariamente, ser maior ou igual ao diâmetro do robô. Para as implementações descritas aqui, as células tem tamanho igual a $5,5\ cm \times 5,5cm$ (diâmetro do robô Khepera [3] [33]). Como o mundo tem $1\ m^2$, o mapa de células terá dimensão de 18×18 células.

O mapeamento do ambiente pode também considerar a expansão dos obstáculos [28], assim sendo, as células onde a intersecção com a região de obstáculos é não nula são assinaladas com 1, as células onde a intersecção com a região de obstáculos é nula, mas ficam na região coberta pela expansão são assinaladas com 2 e as células livres com 0, como mostrado na figura 3.3.

Como medida para a expansão considerou-se o raio do robô. Dada a forma como é feito o mapeamento, com a expansão igual ao raio garante-se a não colisão. Como o robô é cilíndrico, somente a translação é considerada [28]. Para o mapeamento com expansão, a trajetória é dada por uma seqüência de células vazias e mistas.

Como visto no capitulo 2, uma célula é marcada como cheia quando em toda sua dimensão há obstáculos e como mista quando apenas parte de sua dimensão contém obstáculo. Porém, considerou-se que toda a célula cuja intersecção seja não nula é assi-

[3] Veja apêndice A.

```
0 1 2 3 4 5 6 7 8 910111213141516 17

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1   0
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1   1
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1   2
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1   3
1 0 0 0 1 1 1 1 1 1 1 1 2 0 0 0 0 1 4
1 0 0 0 1 1 1 1 1 1 1 1 2 0 0 0 0 1 5
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 6
1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 7
1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 8
1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 9
1 0 0 0 0 0 0 0 2 1 2 0 0 0 1 1 0 1 10
1 0 0 0 0 0 0 0 2 1 2 0 0 0 1 1 0 1 11
1 0 0 0 0 0 0 0 2 1 2 0 0 0 1 1 0 1 12
1 0 0 2 1 1 1 1 1 1 1 1 1 0 0 0 0 1 13
1 0 0 2 1 1 1 1 1 1 1 1 1 0 0 0 0 1 14
1 0 0 0 0 0 2 1 2 0 0 0 0 0 0 0 0 1 15
1 0 0 0 0 0 2 1 2 0 0 0 0 0 0 0 0 1 16
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 17
```

Figura 3.3: Matriz de células com expansão dos obstáculos

nalada como cheia e as células mistas são células vazias que têm como finalidade identificar uma região onde pode ocorrer uma colisão. A identificação e o tratamento destas regiões é melhor detalhado na descrição da 3^a implementação.

3.2.2 1^a Implementação - Algoritmo da Reta

O algoritmo de busca de caminho implementado foi proposto inicialmente em [53] e baseia-se na idéia do algoritmo reativo descrito em [41]. Este algoritmo consiste em traçar uma reta l de um ponto inicial S até um ponto final G, como apresentado na figura 3.4. Esta reta é seguida pelo robô até que um obstáculo seja encontrado, a partir desse ponto de colisão (Q), o robô segue a parede, independentemente do sentido, buscando contornar o obstáculo, sempre calculando sua posição para achar novamente a linha l. Quando o robô encontrar o ponto de intersecção entre a reta e o obstáculo P_0, ele volta a percorrer os pontos da reta em direção ao ponto meta.

Para a implementação do algoritmo da reta, utilizou-se, no mapeamento, a expansão dos obstáculos para demarcar zonas de incerteza. As células marcadas com 2 não foram utilizadas pelo robô, objetivando minimizar as possibilidades de colisão.

Igualmente ao algoritmo apresentado em [41], este algoritmo traça uma reta do ponto inicial ao ponto objetivo, utilizando para isto o mapa do ambiente, contudo se houver uma parede entre estes dois pontos (célula marcada com 1 ou 2) esta é então percorrida até que um ponto de borda (célula marcada com 0) seja encontrado. Identificado o tipo de parede

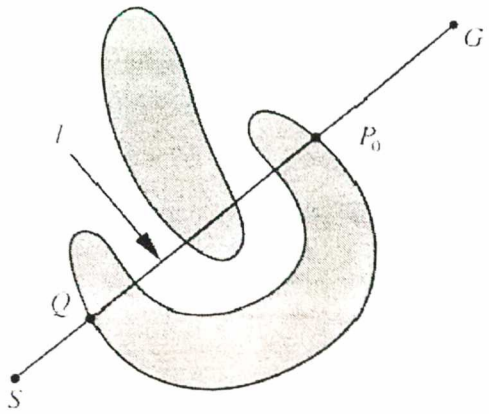


Figura 3.4: Estratégia on-line

e, dada a localização do ponto inicial, mais dois pontos de borda são assinalados para que a parede seja contornada. Então são feitas duas chamadas recursivas, uma enviando o ponto inicial e o ponto de borda mais próximo a ele, e outra enviando o ponto final da borda, agora novo ponto inicial, e o ponto objetivo, quando não houver obstáculos entre os pontos enviados, ou seja, o algoritmo da reta retornar o ponto objetivo, os pontos de borda são incluídos na lista de pontos que descreverão a trajetória do robô.

Há quatro situações possíveis durante a execução do algoritmo que traça a reta: o ponto objetivo ser encontrado, encontrar uma parede, encontrar um canto formado pelo encontro de duas paredes ou encontrar um ponto de borda.

Quando o ponto retornado pela reta (P_c) for pertencente a uma parede, basta identificar se esta é horizontal ou vertical (parede horizontal é paralela ao eixo das abscissas e vertical ao eixo das ordenadas), e percorrê-la até encontrar o seu término. Para percorrer a parede buscando o menor caminho até o ponto objetivo, uma verificação simples da localização deste ponto é feita. Após encontrar o termino da parede, e conhecendo a localização do ponto inicial, são marcados três pontos de borda P_{B_0}, P_{B_1} e P_{B_2}. A figura 3.5 ilustra este processo.

Se a parede for composta (formar um canto), basta saber se o obstáculo até então era vertical ou horizontal e o tipo do canto (superior direito, superior esquerdo, inferior direito ou inferior esquerdo), então altera-se a direção que até então vinha sendo seguida. A figura 3.6 mostra esse processo.

Quando o ponto retornado pela reta for um canto, não há como saber para onde deve-se seguir, então adotou-se a seguinte postura: sempre que o ponto retornado for um canto, a

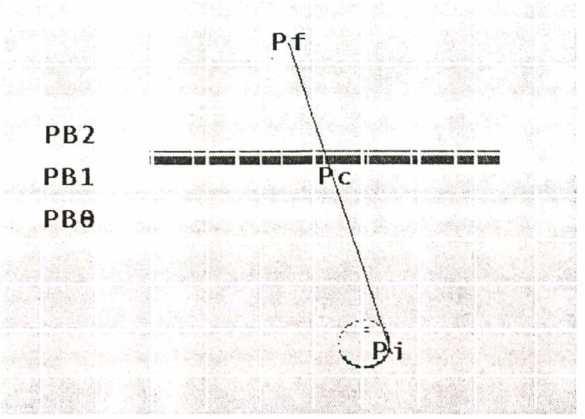


Figura 3.5: Marcação dos pontos de borda

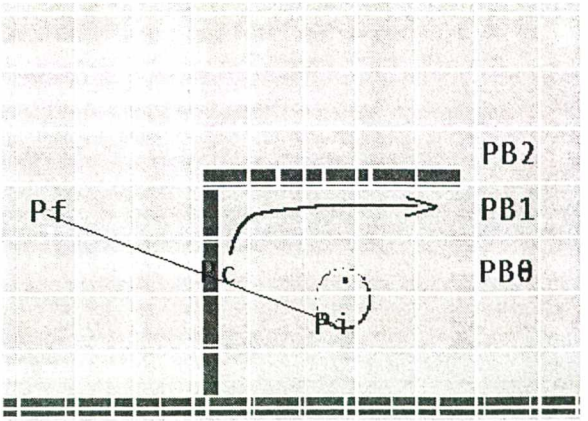


Figura 3.6: Ao encontrar um canto, a direção é alterada

parede vertical será seguida, como representado na figura 3.7.

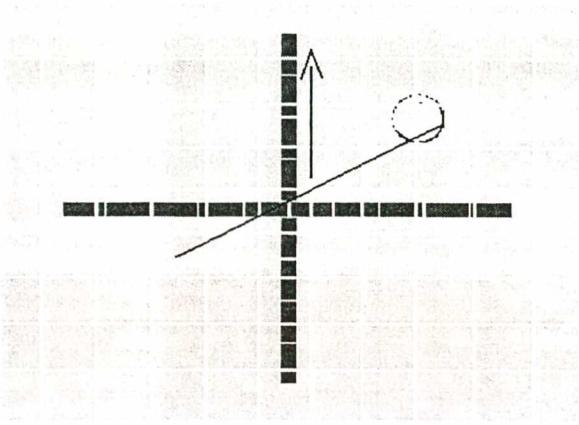


Figura 3.7: Diante de um canto, segue-se a parede vertical

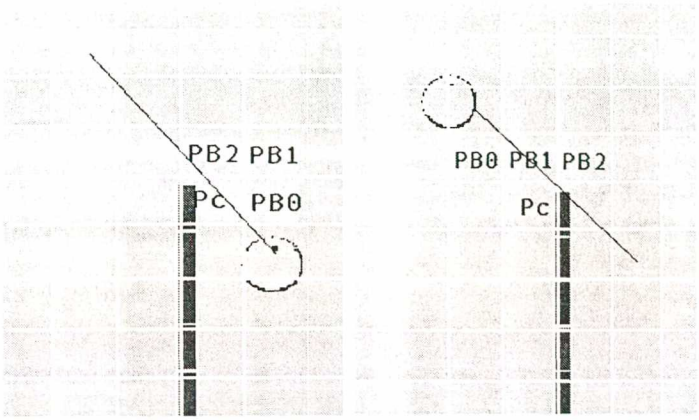
Quando o ponto retornado for uma borda, identifica-se se o tipo da borda (borda superior direita, superior esquerda, inferior direita ou inferior esquerda), e retorna-se o ponto diagonalmente adjacente. Para marcar os outros dois pontos verifica-se a posição do ponto objetivo com relação ao ponto de borda P_{B_1}. Há apenas um caso onde há necessidade de identificação do tipo de obstáculo cujo ponto de borda foi obtido, ele ocorre quando o ponto objetivo encontra-se atrás do obstáculo, assim, identificando-se este caso, é possível contornar o obstáculo assinalando corretamente os pontos de borda. A figura 3.8 apresenta duas das situações. Contudo, elas foram previstas e tratadas para cada um dos quatro tipos de borda.

O resultado da união destes tratamentos pode ser visto em detalhes no próximo exemplo, que descreve detalhadamente o funcionamento do algoritmo, para o ambiente apresentado na figura 3.9.

Exemplo de funcionamento do algoritmo da reta

Inicialmente o algoritmo guarda em uma lista encadeada [43] o ponto inicial P_i $(9, 10)$ e o ponto final P_f $(10, 2)$ e tenta traçar uma reta entre esses pontos, como mostra a figura 3.9, retornando o ponto de colisão P_{C_a} $(9, 8)$ do obstáculo, então a parede é percorrida até localizar o ponto de borda P_{B1_a} $(3, 8)$, sendo que a parede percorrida é uma parede horizontal e o ponto inicial encontra-se em uma coordenada y maior do que a coordenada y do ponto de borda, outros dois pontos serão marcados: P_{B0_a} $(3, 9)$ e P_{B2_a} $(3, 7)$.

Agora a chamada recursiva é feita enviando-se os pontos P_i e P_{B0_a}, a reta encontra um



(a) Assinalando três pontos que contornam a borda é possível alcançar o obstáculo

(b) Há a necessidade de saber qual tipo de obstáculo que se quer desviar, para poder assinalar corretamente os pontos de borda

Figura 3.8: Situações de contorno de borda

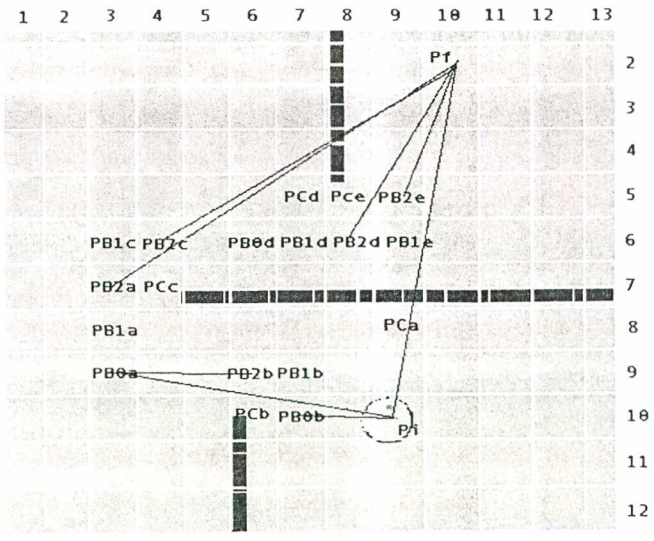


Figura 3.9: Exemplo de funcionamento do algoritmo

novo ponto de colisão P_{C_b} $(6, 10)$, por se tratar de uma colisão em uma célula de borda, o algoritmo que percorre a parede retorna o novo ponto P_{B1_b} $(7, 9)$ (diagonalmente adjacente) considerando-se que o ponto final desta iteração (P_{B0_a}) encontra-se em um x menor, o canto borda é formado pelos pontos P_{B0_b} $(7, 10)$ e P_{B2_b} $(6, 9)$, visando o desvio da parede.

Nova chamada recursiva é feita para os pontos P_i e P_{B0_b} $(7, 10)$. Desta vez, não há obstáculos, então os pontos P_{B0_b}, P_{B1_b} e P_{B2_b} são incluídos na lista de pontos que o robô seguirá.

O algoritmo agora entra na chamada recursiva para os outros dois pontos, sendo que o ponto P_{B2_b} $(6, 9)$ passa a ser o novo ponto inicial e P_{B0_a} $(3, 9)$ o novo ponto final. Como não há obstáculos entre eles, os pontos P_{B0_a} $(3, 9)$, P_{B1_a} $(3, 8)$ e P_{B2_a} $(3, 7)$ são incluídos na lista.

O ponto P_{B2_a} $(3, 7)$ passa agora a ser o novo ponto inicial e uma nova chamada se dá entre ele e o ponto final P_f $(10, 2)$, a reta colidirá agora no ponto P_{C_c} $(4, 7)$ e o ponto de borda retornado pelo algoritmo *percorre parede* é o ponto P_{B1_c} $(3, 6)$. Como trata-se de uma borda superior direita onde o ponto objetivo encontra-se em um x maior, o canto da borda é formado pelos pontos P_{B0_c} $(3, 7)$ e P_{B2_c} $(4, 6)$. Agora os dois pontos enviados para a reta são iguais (P_{B2_a} $(3, 7)$ e P_{B0_c} $(3, 7)$), então os pontos P_{B0_c} $(3, 7)$, P_{B1_c} $(3, 6)$ e P_{B2_c} $(4, 6)$ são incluídos na lista.

Marcado como novo ponto inicial P_{B2_c} $(4, 6)$ é enviado juntamente com o ponto final P_f $(10, 2)$, retornando o ponto de colisão P_{C_d} $(7, 5)$, novamente trata-se de uma borda P_{B1_d} $(6, 6)$, porém o ponto objetivo encontra-se em um x maior e em um y menor, o que significa que é necessário saber qual o tipo da parede para que os pontos de borda sejam marcados corretamente. Visto que se trata de uma parede vertical, P_{B0_d} recebe o ponto P_{B1_d}, o ponto P_{B1_d} fica deslocado em 1 com relação a x e o ponto P_{B2_d} deslocado em 2. A reta é traçada agora entre os pontos P_{B2_c} $(4, 6)$ e P_{B0_d} $(6, 6)$ e os pontos P_{B0_d} $(6, 6)$, P_{B1_d} $(7, 6)$ eP_{B2_d} $(8, 6)$ são incluídos na lista.

Agora a reta é traçada entre os pontos P_{B2_d} $(8, 6)$ e P_f $(10, 2)$, colidindo no ponto P_{C_e} $(8, 5)$, retornando o ponto de borda P_{B1_e} $(9, 6)$. Como se trata de um ponto de borda inferior esquerdo, os pontos assinalados serão P_{B0_e} $(8, 6)$ e P_{B2_e} $(9, 5)$. Ao enviar para o algoritmo da reta o mesmo ponto (P_{B2_d} $(8, 6)$ e P_{B0_e} $(8, 6)$), os pontos P_{B0_e}, P_{B1_e} e P_{B2_e} são incluídos na lista. Por fim, a reta é traçada entre os pontos P_{B2_e} $(9, 5)$ e P_f $(10, 2)$. Como o ponto final foi alcançado a lista de pontos $(9, 10); (7, 10); (7, 9); (6, 9); (3, 9); (3, 8); (3, 7); (3, 6); (4, 6); (6, 6); (7, 6); (8, 6); (9, 6); (9, 5); (10, 2)$

é retornada.

Após este processamento, os pontos da lista são convertidos para coordenadas do ambiente, e ela é completada com todos os pontos que descrevem o caminho do robô, que é apresentado na figura 3.10.

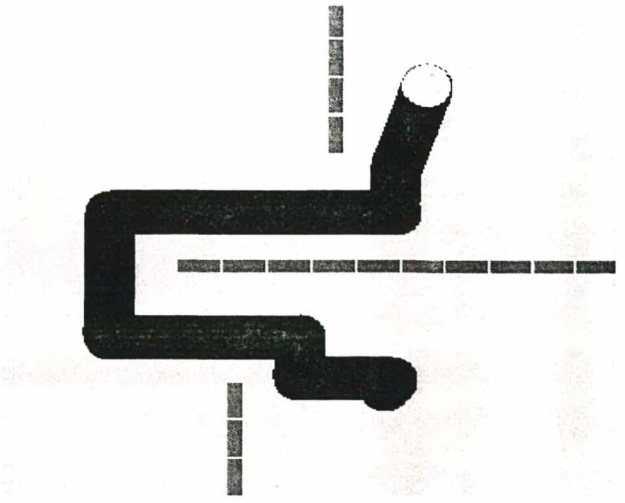


Figura 3.10: Execução do caminho pelo robô após o planejamento

O método foi expandido para tratar múltiplos pontos objetivo no ambiente. Isso possibilita que o robô simule a realização de tarefas ou execute o menor caminho passando por n luzes. Contudo, a realização do menor caminho não leva em conta a presença dos obstáculos.

Observações sobre o método

A implementação deste método tenta identificar as situações e tratá-las adequadamente, contudo, isto é trabalhoso, ainda que o ambiente seja considerado estático, bem definido e mapeado com alta precisão. Desta forma, o método é pouco robusto - dada a dificuldade de prever e tratar todas as situações e devido ao fato de que qualquer falha no mapa do mundo pode resultar na interpretação errada do ambiente - e limitado - pois somente trata com paredes horizontais e verticais e não trabalha com a possibilidade de não haver caminho ou de não encontrá-lo. Apesar da tentativa de generalizar ao máximo as situações de contorno dos obstáculos, há casos em que o algoritmo não consegue encontrar um caminho até o ponto objetivo, como ocorre quanto o ponto encontrado pela reta é um canto. A opção de sempre seguir as paredes verticais muitas vezes não é satisfatória.

Como o método não utiliza as células assinaladas com 2, devido à possibilidade de

colisão, o caminho do robô só será encontrado em ambientes onde os obstáculos sejam espaçados, pois é necessário que as células que formam o caminho sejam todas assinaladas com 0. Além disso, o robô não executará caminhos em diagonal entre dois obstáculos, mesmo que este caminho exista, pois o método ocupa as células de borda para contornar o obstáculo.

Uma forma de reduzir a complexidade é tratar o planejamento mais genericamente, optando, por exemplo, pela busca em um espaço de estados.

3.2.3 2^a Implementação - Árvore de Busca 4NF

Para generalizar a busca por um caminho, evitando o tratamento de cada situação específica, implementou-se um método bastante simples, que encontra a solução percorrendo todo o espaço de estados através da geração de uma árvore de busca com expansão em largura ou árvore de busca em amplitude [40, 41, 43].

A idéia é gerar uma árvore de busca exaustiva de caminho mínimo construída sobre o mapa do ambiente. Como a matriz de células do ambiente é preenchida apenas com $0's$ e $1's$, não considerando as regiões de incerteza, o método limita-se a realizar trajetórias na horizontal e vertical, devido a possibilidade de colisão. Ou seja, na árvore gerada cada nó terá no máximo quatro nós filhos (4NF), sendo consideradas somente as células adjacentes horizontais e verticais, como mostra a figura 3.11.

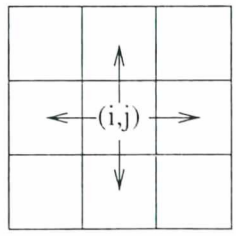


Figura 3.11: Expansão para os nós adjacentes verticais e horizontais

Como a árvore de busca é expandida em amplitude, garante-se encontrar o menor caminho entre o ponto inicial e o ponto objetivo, justamente por sua característica de construir a árvore sobre o espaço de estados, expandindo todos os nós de todos os ramos de um nível i da árvore, antes de expandir os nós do nível $i + 1$.

Para gerar a árvore, toma-se a célula que representa a posição inicial do robô como sendo o nó raíz. A partir desde nó raíz são gerados outros 4 nós, que representam as células adjacentes horizontal e verticalmente, se estas forem células vazias. E a partir de cada um dos

nós filhos do nó raiz, é possível expandir mais 3 nós (o nó pai não é gerado, para eliminar ramos cíclicos), e assim sucessivamente até que o nó que representa a célula do ponto objetivo seja gerado ou não hajam mais nós para expandir.

A partir do ambiente apresentado na figura 3.12, um mapa idêntico ao da figura 3.2 é criado. Sobre este mapa, que representa o espaço de estados do problema, é gerada a árvore de busca, apresentada na figura 3.13. Após alcançar o nó que representa a célula do ponto objetivo, uma lista de pontos contendo os nós que formam o caminho deste até a raiz é montada e retornada. Esta lista é extraída seguindo-se os apontadores dos nós filhos que apontam para os seus pais. Esta lista é então executada pelo robô, como apresentado na figura 3.14.

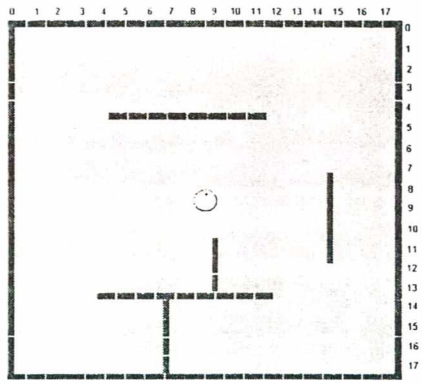


Figura 3.12: Ambiente proposto

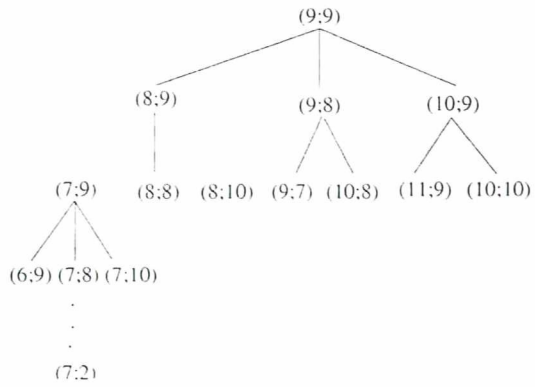


Figura 3.13: Árvore de busca de caminho mínimo

Observações sobre o método

Este método é simples e garante que o robô não colida, pois o caminho é dado por uma seqüência de células vazias. Contudo, há o problema da explosão combinatória [43] na

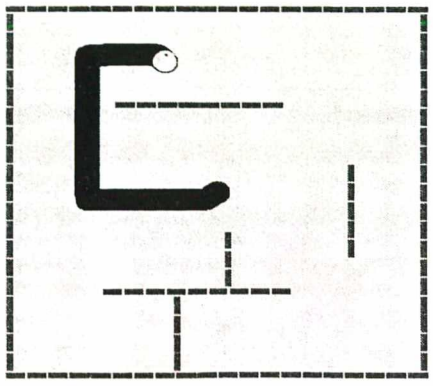


Figura 3.14: Caminho executado pelo robô

geração da árvore de busca, pois somente a geração de nós iguais em um mesmo ramo da árvore é evitada.

Além disso, como a trajetória do robô é dada por uma seqüência de segmentos de reta, apesar de sempre executar a menor trajetória (em número de células percorridas), esta não é a mais elegante pois os caminhos em diagonal não são aproveitados. O caminho diagonal não pode ser gerado devido à ausência da expansão dos obstáculos. Caso existam duas células livres em diagonal, mas as células próximas apresentam-se com obstáculos, o robô colidirá ao tentar passar entre elas.

Para solucionar os problemas citados, é necessário considerar a expansão dos obstáculos, gerar a árvore de busca considerando as células diagonais e implementar uma função de corte para evitar a explosão combinatória.

3.2.4 3ª Implementação - Árvore de Busca 8NF

Esta terceira implementação visa melhorar o método apresentado acima, tornando-o mais robusto e eficiente.

Considerou-se a expansão dos obstáculos, o que permite que a árvore de busca gerada tenha seus nós expandidos para os oito nós filhos (8NF) adjacentes, como apresenta-se na figura 3.15.

As células que pertencem a zona de incerteza são tratadas como células vazias, exceto quando se trata de uma expansão para nós diagonais, onde então é verificado o espaço disponível para o robô, assegurando desta forma a não colisão. Essa verificação do espaço livre é feita baseando-se no conteúdo da célula.

A expansão da árvore será segura quando se der para nós que representem células assi-

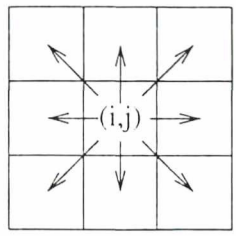
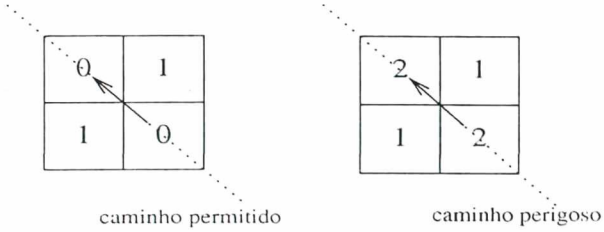


Figura 3.15: Expansão para os nós horizontais, verticais e diagonais

naladas com 0, e os nós que formam a outra diagonal representem células assinaladas com 1 ou 2, uma vez que 2 representa uma célula vazia e 1 implica que apenas parte da célula esteja ocupada, pois a expansão do obstáculo não ultrapassou os limites da célula; ou quando a expansão se der para nós que representem células assinaladas com 2 e a diagonal oposta for formada por células que contenham 2, pois tratam-se de 4 células vazias.

O caminho somente não será seguro quando a diagonal oposta ao sentido da expansão for formada por células assinaladas com 1 e a expansão se der de um nó que contém 2 para um nó que contém 2 ou 0 ou de um nó que contém 0 para um nó que contém 2.

Na figura 3.16 são apresentados dois casos, um dos quais não é possível realizar a expansão do nó devido a possibilidade de colisão do robô.



(a) A trajetória na diagonal é assegurada quando há espaço suficiente para o robô

(b) Quando não há espaço suficiente para o robô a expansão para a célula diagonal não é realizada

Figura 3.16: Dois casos onde o caminho diagonal precisa ser verificado

Para evitar a explosão combinatória utilizou-se a estratégia de corte de nós repetidos descrita no algoritmo A* [39, 40]. Os nós criados são armazenados em uma lista encadeada [43], a cada novo passo de expansão esta lista é verificada: se o nó que está sendo analisado, já tiver sido gerado anteriormente, sua expansão é interrompida, ou seja, aquele ramo da árvore deixa de ser gerado.

Na geração da árvore, a ordem com que os nós são expandidos e, conseqüentemente cortados afeta a trajetória. A figura 3.17 apresenta dois exemplos de geração e expansão. Em (a), inicialmente os nós diagonais são expandidos no sentido horário e posteriormente os horizontais e verticais, também no sentido horário. Saindo da célula $P\,(7,7)$ para chegar à célula $P\,(4,7)$, o caminho gerado pela expansão é dado pelas células $(7,7)$, $(6,6)$, $(5,7)$ e $(4,7)$. Em (b), gerando inicialmente os nós horizontais e verticais, no sentido horário e posteriormente os nós diagonais, para os mesmos pontos inicial e final, o caminho gerado é dados pelas células $(7,7)$, $(6,7)$, $(5,7)$ e $(4,7)$. Ambos os caminhos têm o mesmo comprimento, ou seja, 4 células, contudo, quando mapeados para coordenadas do ambiente real, o caminho gerado em (b) é menor.

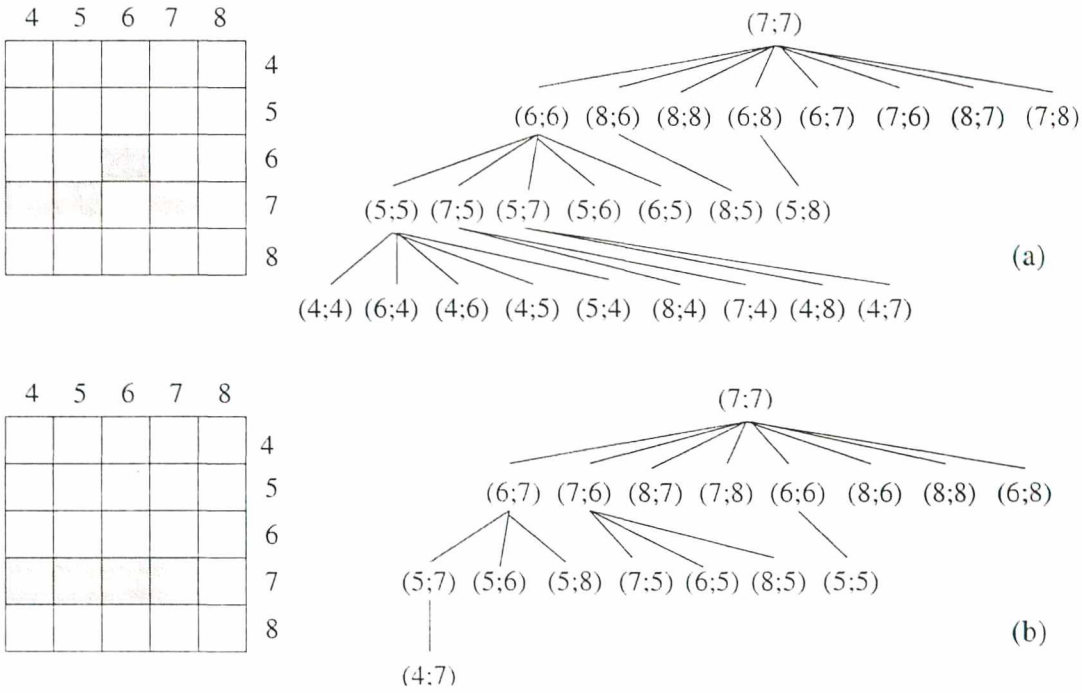


Figura 3.17: A trajetória depende da forma como a expansão é realizada

Semelhante ao método anterior, o algoritmo inicia mapeando o ambiente apresentado na figura 3.12, criando um mapa idêntico ao da figura 3.3. A árvore de busca de caminho mínimo como a apresentada na figura 3.18 é criada sobre este mapa. Ao executar o caminho descrito pelos nós da árvore o robô executa a trajetória, como apresenta-se na figura 3.19.

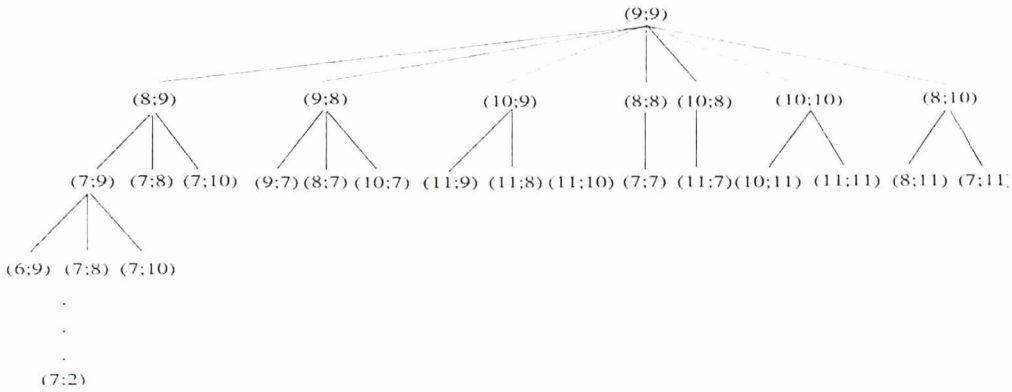


Figura 3.18: Árvore de busca de caminho mínimo com 8 nós filhos

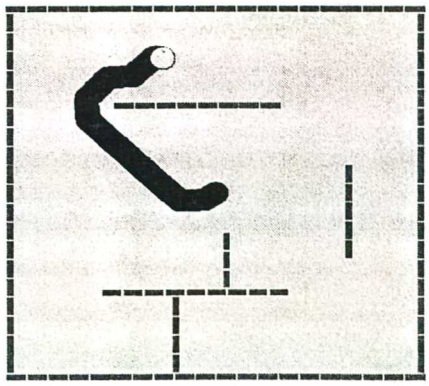


Figura 3.19: Execução do caminho pelo robô

Observações sobre o método

Este método soluciona os problemas do método anterior, ou seja, gera caminhos na diagonal e evita a explosão combinatória. Descrevendo trajetórias em diagonal, o caminho do robô é significativamente menor e mais elegante. O corte executado na árvore de busca torna o método computacionalmente mais rápido.

Para garantir a não colisão do robô na travessia entre células em diagonal, fez-se necessário identificar cada uma das possíveis situações. Ainda que mais oneroso computacionalmente, este método é muito mais genérico se comparado ao método do algoritmo da reta descrito no início do capítulo.

Entretanto, qualquer mudança no ambiente ocorrida após o mapeamento poderá implicar construção de uma árvore que não reflete o ambiente atual e possivelmente a colisão do robô com um obstáculo. Ou seja, o método não trata ambientes com características dinâmicas. Para contornar essas limitações faz-se necessária a leitura dos sensores em tempo de execução para que o robô possa identificar as mudanças enquanto executa a trajetória, desviando dos obstáculos dinâmicos que não estejam no mapa do ambiente.

3.3 Abordagem Reativa - Arquitetura de Subsunção

A abordagem reativa permite ao robô atuar em ambientes dinâmicos, uma vez que a decisão da ação de controle é baseada na leitura dos sensores obtida em tempo de execução. Como visto no capítulo 2, o uso de redes neurais artificiais (RNAs) para a implementação desta abordagem tem apresentado bons resultados, dadas as propriedades de adaptação e aprendizado da técnica, permitindo generalizar o comportamento do robô.

3.3.1 1ª Implementação - Navegação Direcionada Utilizando RNAs

Como visto em [53], o uso de redes neurais sem o conhecimento prévio da localização do ponto objetivo incorre na navegação aleatória do robô pelo ambiente. Ainda em [53] apresentou-se uma forma de contornar esse problema, criando uma estimativa do ponto objetivo, quando este fosse detectado pelos sensores. Contudo, o robô navega aleatoriamente até que os sensores de luminosidade sejam ativados e permitam o cálculo desta estimativa.

Na implementação aqui descrita, o robô conhece *a priori* sua localização e a localização do ponto objetivo, tornando a navegação *globalmente direcionada*, enquanto localmente responde

aos estímulos captados pelos sensores infravermelhos.

Seguindo o modelo proposto em [10], o robô apresenta dois comportamentos: *desviar obstáculos* e *procurar objetivo*. Os comportamentos são implementados por redes neurais dispostas hierarquicamente. Uma das redes é responsável pelo compotamento de desvio de obstáculos e outra pelo comportamento de procura do ponto objetivo, como mostra o diagrama da figura 3.20.

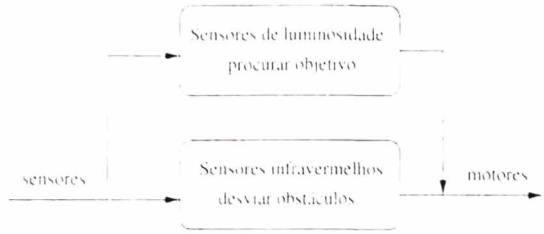


Figura 3.20: Diagrama hierárquico de comportamentos

A rede neural responsável pelo comportamento *desviar de obstáculos* tem prioridade sobre a rede neural responsável pelo comportamento *procurar objetivo*. Ou seja, a rede neural responsável por encontrar o ponto objetivo será acionada somente quando os sensores infravermelhos não detectarem obstáculos.

Quanto à arquitetura das redes neurais, estas são do tipo feedforward com 3 camadas, sendo 9 neurônios na camada de entrada (8 sensores + bias term), 27 neurônios na camada oculta e 2 neurônios na camada de saída (valores dos motores esquerdo e direito). O algoritmo de treinamento utilizado é o Backpropagation[4]. A figura 3.21 apresenta um modelo simplificado das redes neurais utilizadas.

Para tornar a navegação globalmente direcionada, utilizou-se um cálculo simples que considera o ângulo entre o ponto objetivo e o robô. Dado o ângulo do robô no ambiente, é possível prever qual sensor deve ser ativado, assim, a rede neural recebe uma *falsa* entrada (ao invés da leitura real dos sensores de luminosidade, dá-se a entrada calculada), o que faz com que o robô procure atingir o ponto objetivo, mesmo quando seus sensores não detectem o foco de luz.

A figura 3.22 apresenta um exemplo simples. Inicialmente o robô não detecta obstáculos e começa a navegar guiado pelo comportamento *procurar objetivo*. Ao encontrar a parede, o comportamento *desviar obstáculo* é acionado. Quando os sensores infravermelhos não detectarem mais o obstáculo, novamente o comportamento *procurar objetivo* assume o controle

[4]Descrito no apêndice B.

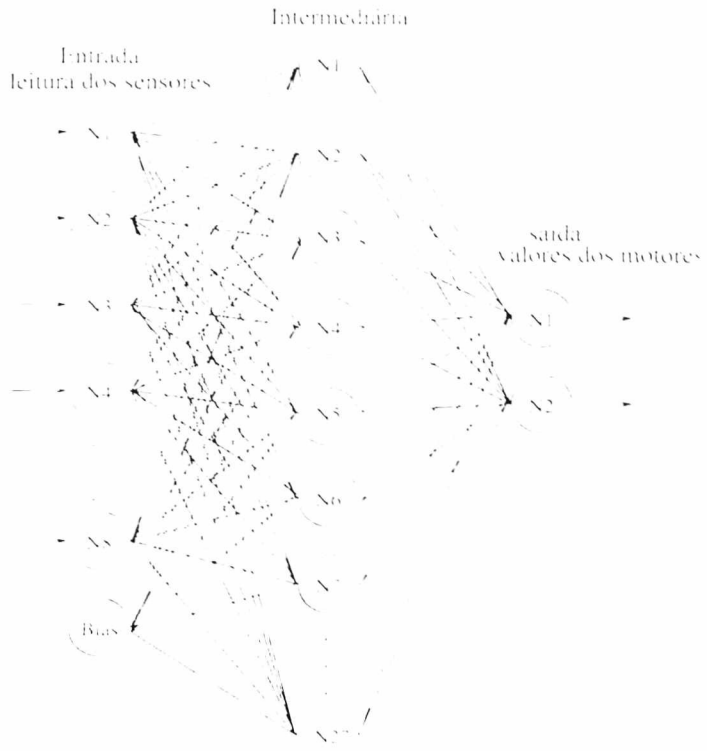


Figura 3.21: Modelo simplificado das redes neurais artificiais

sobre o robô, conduzindo-o até o ponto objetivo.

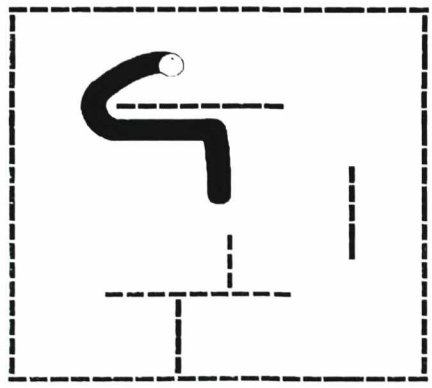


Figura 3.22: Comportamento reativo obtido com as redes neurais artificiais

Observações sobre o método

A grande vantagem de usar redes neurais com aprendizado supervisionado para a implementação da abordagem reativa está na capacidade das redes de "aprender" e "generalizar" através de um conjunto de exemplos de aprendizado.

Em sistemas reativos convencionais são utilizados modelos matemáticos explícitos para

mapear os dados sensoriais de entrada em saída dos motores, contudo esses modelos muitas vezes são difíceis ou impossíveis de alcançar. Com as redes neurais, é possível assimilar as características do sistema, sem a necessidade de modelos matemáticos do mesmo. Além disso, as redes neurais com treinamento supervisionado são fáceis de implementar e treinar.

Entretanto, o aprendizado supervisionado apresenta um sério problema: a ambigüidade nos exemplos de treinamento - quando há no conjunto de treinamento um exemplo onde o padrão de entrada produz um padrão de saída que conduz o robô a desviar o obstáculo pela direita e outro padrão de entrada idêntico que produz um padrão de saída que conduz o robô a desviar o obstáculo pela esquerda, por exemplo. Sendo assim, é necessário selecionar um conjunto de exemplos de treinamento que não seja ambíguo, garantindo a consistência dos valores dos pesos da rede. Outra possibilidade é implementar uma rede com treinamento não supervisionado.

3.3.2 2ª Implementação - Aprendizado Hebbiano

A segunda implementação da abordagem reativa, utilizando a técnica de redes neurais para simular os comportamentos é feita com a regra de aprendizado de Hebb[5]. A idéia é baseada na arquitetura proposta em [56]. Contudo, restringindo-a ao desvio dos obstáculos, em virtude da grande quantidade de interações necessárias ao aprendizado da rede e da baixa amplitude dos sensores de luminosidade.

O diagrama da figura 3.23 apresenta um esquema representativo da arquitetura implementada.

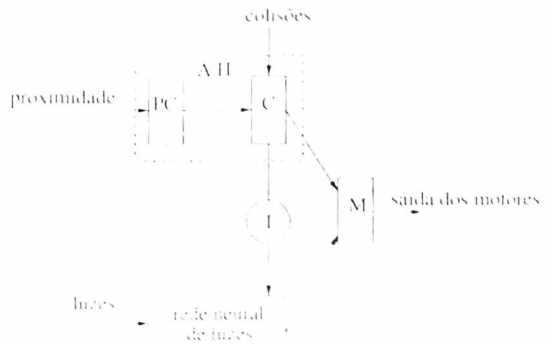


Figura 3.23: Arquitetura de controle utilizando aprendizado Hebbiano

A parte de desvio de obstáculos feita com o aprendizado Hebbiano (A.H.) é composta

[5]Descrito no apêndice B.

por duas camadas de neurônios. A primeira camada recebe leituras de proximidade de colisão (PC), e a segunda, leituras de colisão (C). O aprendizado Hebbiano ocorre entre estas camadas.

Inicialmente as conexões entre os neurônios recebem pesos aleatórios e são reforçadas sempre que ambos forem excitados simultaneamente. Desta forma, inicialmente o robô tocará o obstáculo e "aprenderá" a desviá-lo. Em uma segunda interação, o robô o desviará antes de tocá-lo. A figura 3.24 (a) apresenta uma primeira interação do robô com o ambiente e o contato deste com o obstáculo. Em (b) o robô "aprendeu" a desviar do obstáculo, e em (c), uma segunda interação onde o robô desvia antes de tocar o obstáculo.

(a) O robô toca o obstáculo

(b) Os pesos da rede são atualizados segundo a regra de aprendizado de Hebb

(c) O robô aprende a desviar do obstáculo antes de tocá-lo

Figura 3.24: O robô aprende a desviar do obstáculo

Na tentativa de otimizar o aprendizado, inicialmente, aplicou-se à rede um fator de "esquecimento" para reduzir os pesos das conexões não simultaneamente acionadas, conforme proposto por [18]. Contudo, ao interagir com um ambiente que exija mais de alguns sensores, o robô acaba "esquecendo" algumas reações, prejudicando o desempenho.

Na arquitetura proposta em [38], a parte responsável pela busca do ponto objetivo é semelhante a parte de desvio de obstáculos e a saída das redes gera dois tipos de reflexos: *bateu-retorna-vira* e *se-objetivo-vá-em-direção*. Ao implementar esta arquitetura, não foram obtidos bons resultados. Sem o conhecimento global da localização do ponto objetivo o robô navega aleatoriamente pelo ambiente e eventualmente atinge o ponto objetivo, pois os sensores de luminosidade têm baixa amplitude. Para melhorar o desempenho do robô, utilizou-se a rede neural com conhecimento global do ponto objetivo, descrita na implementação anterior.

Para implementar a parte do reflexo para desvio de obstáculo, a saída da rede é observada e uma adaptação do ângulo do robô é feita. Caso a colisão seja frontal, o robô terá seu

ângulo ajustado em -90° (o robô sempre vira para a esquerda), no caso de colisões laterais o ângulo é ajustado em $+30^{\circ}$ ou -30°. Se nenhuma colisão é verificada, a rede neural que implementa o comportamento *buscar objetivo* é ativada.

Após algum tempo navegando no ambiente e deparando-se com várias situações o robô melhora seu desempenho, uma vez que os pesos da rede vão sendo reforçados. A figura 3.25 apresenta o exemplo de uma execução final após algum tempo de navegação pelo ambiente.

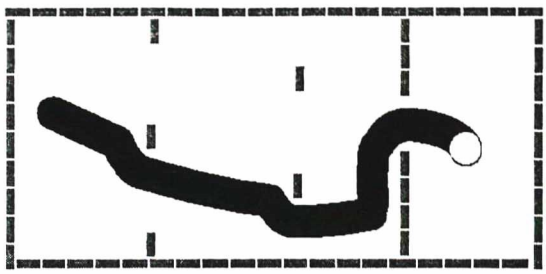


Figura 3.25: Comportamento reativo obtido com o aprendizado Hebbiano

Observações sobre o método

A arquitetura reativa implementada com aprendizado Hebbiano apresenta-se como uma alternativa simples à necessidade de um conjunto de treinamento.

A capacidade do robô reagir diante dos obstáculos vai sendo melhorada a medida que este interage com o ambiente, ainda que a ação reflexiva seja implementada de forma simples através de um ajuste no ângulo do robô.

O fator limitante do aprendizado do robô está relacionado com o raio de ação dos sensores, que impele a saturação dos pesos da rede com valores de leitura sensoriais muito altos, ou seja, a ação do robô é disparada quando este estiver muito próximo ao obstáculo.

3.4 Conclusão

Neste capítulo foram descritos os método implementados referentes às abordagens isoladas ao planejamento de trajetória de robôs móveis autônomos. Foram realizadas três implementações da abordagem planejada e duas da abordagem reativa.

Para a abordagem planejada utilizou-se o método de decomposição em células aproximadas e para a abordagem reativa, a arquitetura de subsunção, implementando os comportamentos com redes neurais artificiais.

Ao final, pode-se observar as características de cada abordagem e os problemas advindos da não completude das mesmas. Isoladamente, o uso da abordagem planejada se restringe a ambientes estáticos, assim como a abordagem reativa por vezes recai na ausência de um conhecimento global acerca do ambiente.

Além disso, cada método apresenta vantagens e desvantagens quando comparados com outros vistos no capítulo 2. O método de decomposição em células aproximadas é simples de ser implementado e permite o uso de outras técnicas para construção do caminho do robô. Entretanto, restringe o espaço livre do robô a células completamente vazias, o que causa uma sub-utilização do espaço.

Já a simulação de comportamentos utilizando a técnica de redes neurais artificiais, permitem captar as características do sistema sem a necessidade de um modelo matemático explícito. Contudo, apesar da variação dos algoritmos de aprendizado e estrutura das redes, cada uma apresenta limitações que vão desde a escolha dos pares de treinamento, como no caso do aprendizado supervisionado, ao tempo de aprendizado, como no caso do aprendizado Hebbiano.

Em resumo, pode-se dizer que nenhuma das abordagens é completa quando implementada isoladamente. Neste sentido, o desenvolvimento de abordagens híbridas visa unir as melhores características das abordagens planejada e reativa, buscando dar maior autonomia ao robô.

Capítulo 4

Abordagem Híbrida

4.1 Introdução

Como visto no capítulo anterior, quando implementados isoladamente, tanto os métodos da abordagem planejada quanto os da abordagem reativa apresentam falhas.

Na abordagem planejada, o robô conhece o ambiente *a priori* e constrói a trajetória baseando-se em um mapa. Essa abordagem falha quando o ambiente apresenta características dinâmicas.

Há duas formas de tratar ambientes com características dinâmicas utilizando somente métodos planejados. A primeira é reconstruir o mapa e replanejar a trajetória toda vez que mudanças forem detectadas pelo robô. Contudo isso pode ser uma tarefa demorada e o robô precisa responder a essas mudanças instantaneamente. Outra forma é armazenar vários modelos do ambiente com as trajetórias associadas e uma vez conhecidas as características do ambiente real selecionar dentre os modelos existentes o mais próximo. Porém, novamente não se trata de um processamento simples e há a necessidade de grande capacidade de armazenamento e processamento associados.

Sendo assim, uma forma factível de permitir ao robô navegar em um ambiente com obstáculos dinâmicos, mantendo um plano global que o permita otimizar a trajetória, é dotá-lo da capacidade de "perceber" o ambiente em tempo de execução e agir baseando-se em uma situação local. Ou seja, *pré-mapear* o ambiente, planejar a trajetória e agir reativamente somente quando obstáculos que não estão no mapa do ambiente forem percebidos.

Na abordagem reativa, o modelo do mundo não é conhecido e as ações do robô são realizadas em tempo de execução, baseando-se nas informações locais. Para melhorar as

execuções pode-se utilizar o conhecimento da posição do robô e do ponto objetivo, mas a falta de um plano global ainda é um fator determinante.

Uma forma de fazer com que o robô conheça o ambiente e melhore suas execuções é fazê-lo armazenar as informações lidas pelos sensores criando um mapa que é consultado para o planejamento da trajetória. Ou seja, inicialmente o robô navega reagindo aos estímulos detectados pelos sensores e *armazena* estas informações em um mapa. Em uma situação posterior, o robô consulta o mapa e busca planejar uma trajetória.

Neste capítulo são descritos estes dois métodos híbridos. O primeiro baseado em *pré-mapeamento do ambiente* e o segundo na *construção e manutenção do mapa em tempo de execução*. Tais métodos resultam da união de dois métodos apresentados no capítulo 3: o método de decomposição em células aproximadas com geração de uma árvore de busca de caminho mínimo com 8 nós filhos da abordagem planejada e a arquitetura de subsunção utilizando a técnica de redes neurais artificiais para simulação dos comportamentos da abordagem reativa.

O método de decomposição em células aproximadas permite a construção rápida de um modelo do ambiente, facilita a manipulação do mesmo e provê uma representação explícita do espaço livre. Já as redes neurais artificiais, com a capacidade de "aprender" e "generalizar" possibilitam a tomada de decisão em tempo de execução para desvio dos obstáculos dinâmicos.

4.2 1ª Implementação - Pré-Mapeamento

Em [54] um método baseado em pré-mapeamento foi desenvolvido utilizando o algoritmo da reta e redes neurais com estimativa da luz. Este método, denominado SNNAP - Sistema Neural para Navegação em Ambientes Pré-Mapeados, incorre em uma falha: a trajetória é planejada uma única vez com a estipulação de pontos meta. Isso significa que o robô precisa passar sobre os pontos meta para atingir o ponto objetivo.

Caso o ponto meta seja sobreposto por um obstáculo dinâmico, o robô ficará circulando o obstáculo na tentativa de atingir o ponto meta. Outro problema que decorre da não manutenção do plano acontece quando o robô se afasta muito do próximo ponto meta que deveria atingir e se aproxima do ponto objetivo, tornando desnecessária sua passagem pelo ponto meta. Ao sair da situação de colisão o robô buscará o ponto meta e posteriormente o ponto objetivo, fazendo um caminho desnecessário.

Se o plano fosse refeito ou a lista de pontos meta manipulada de acordo com a posição do robô no ambiente, essa falha seria contornada.

O método proposto a seguir, une o método de geração da árvore de busca com 8 nós filhos (seção 3.2.4) da abordagem planejada e o método de navegação direcionada (seção 3.3.1) da abordagem reativa, descritos no capítulo anterior. Desta forma o robô possui um plano inicial e dirige-se globalmente por este plano, mas reage localmente aos estímulos desviando-se de obstáculos que não foram previamente mapeados, refazendo o plano após sair da situação de colisão.

Fez-se a implementação da seguinte forma: inicialmente o ambiente é mapeado em uma matriz de células e a trajetória é planejada através da geração da árvore de busca sobre o espaço de estados. O robô inicia sua execução e a cada passo a leitura dos sensores é avaliada. Se algum dos sensores do robô indicar a presença de um obstáculo dinâmico, o plano é abandonado e o robô passa a agir reativamente utilizando as redes neurais com os comportamentos de *desviar obstáculos* e *procurar objetivo*, buscando desviar do obstáculo e ir em direção ao ponto objetivo. Ao sair da situação de colisão, um novo planejamento é feito, considerando-se a posição atual do robô.

Estipulou-se que o robô somente está fora de uma situação de colisão se seus sensores não detectarem obstáculos durante 15 passos. Esta escolha foi necessária devido à seguinte situação: se o número de passos não for considerado, a parte planejada do método (mais custosa computacionalmente) é acionada a cada 2 passos, pois sempre que há uma leitura indicando colisão, a rede neural responsável pelo comportamento de *desviar obstáculos* leva o robô em um próximo passo ao afastamento do obstáculo, se o planejamento for refeito, as células ocupadas pelo obstáculo dinâmico, que estão vazias no mapa, serão utilizadas e isto fará com que o robô volte à situação de colisão no passo seguinte. O mesmo acontece se o número de passos for pequeno.

A figura 4.1 detalha um exemplo de execução. O robô segue a trajetória planejada até o ponto A onde os sensores acusam um obstáculo que não foi mapeado. O robô abandona o plano e age reativamente até atingir um ponto fora da situação de colisão, ponto B, onde uma nova trajetória é planejada.

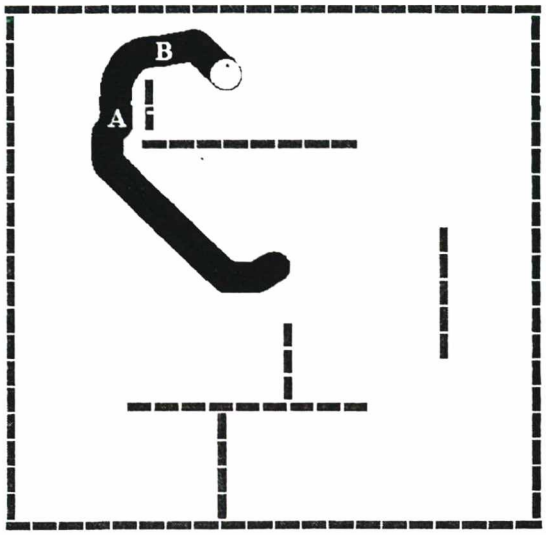


Figura 4.1: Execução com pré-mapeamento

4.2.1 Considerações sobre o Método

Com o método híbrido baseado em pré-mapeamento, o robô consegue desviar de obstáculos dinâmicos, sem contudo navegar aleatoriamente, levado apenas pela localização do ponto objetivo como acontecia no método de navegação direcionada descrito anteriormente.

Além disso, o plano é refeito sempre que o robô sai da situação de colisão. Isso garante que ele sempre execute a menor trajetória ao sair de uma colisão.

É importante salientar que o mapa não sofre nenhum tipo de manutenção. Isso significa que em uma próxima execução, caso o obstáculo não mapeado permaneça no ambiente, novamente o robô terá que desviá-lo utilizando a parte reativa do método. Da mesma forma, se houver outros obstáculos que inicialmente não foram mapeados, o novo planejamento não os considerará. Caso o ambiente sofra grandes mudanças, um novo mapeamento deve ser feito para garantir o bom desempenho do robô.

4.3 2^{a} Implementação - Construção e Manutenção do Mapa em Tempo de Execução

Para solucionar o problema da não manutenção do modelo do ambiente, pode-se implementar sua construção e manutenção em tempo de execução.

Nesta implementação, são conhecidos inicialmente apenas o ponto inicial e o ponto ob-

jetivo - nenhuma outra informação acerca do ambiente é fornecida. A trajetória inicial é gerada pela construção da árvore de busca com 8 nós filhos sobre todo o espaço de estados.

Ao percorrer o caminho gerado, o robô recebe informações do ambiente através da leitura dos sensores. Quando estes indicarem a presença de um obstáculo (ponto A na figura 4.2 (a)), o plano é abandonado e as redes neurais que simulam os comportamentos *desviar obstáculo* e *procurar objetivo* são ativadas, armazenando no mapa de células as informações lidas. Para marcar a célula corretamente, verifica-se o ângulo do robô no ambiente e qual o ângulo do sensor mais ativo. Ao sair da situação de colisão (ponto B na figura 4.2 (a)) um novo planejamento é feito, considerando as novas informações contidas no mapa.

Como apenas uma parte do obstáculo tornou-se conhecida, ao executar pela segunda vez o planejamento, o robô será conduzido à parte desconhecida do obstáculo, como mostra a figura 4.2 (b).

(a) Ao seguir a primeira trajetória, o robô encontra os obstáculos e as informações lidas são armazenadas no mapa

(b) Na segunda execução o robô irá ao encontro da parte desconhecida do obstáculo

Figura 4.2: Construção do mapa em tempo de execução

Com o obstáculo totalmente conhecido e as respectivas células no mapa atualizadas, a próxima execução se dará seguindo o planejamento de trajetória, como mostra a figura 4.3 (a). Ao executar o plano, novas leituras vão sendo obtidas com o objetivo de dar manutenção ao mapa. Desta forma, ao receber leituras que indiquem proximidade ou ausência de obstáculos, o mapa é alterado como segue:

- Se a leitura dos sensores indicar meia colisão (leituras entre 200 e 500), o valor da célula só será alterado para 2 se esta contiver 0;
- Se a leitura dos sensores indicar não colisão (leituras de valores inferiores a 10) e a célula respectiva tiver valor 1, seu valor é alterado para 2. Se a célula contiver 2, seu

valor passa a ser 0.

Esta forma de manutenção permitirá o "esquecimento" do obstáculo.

Como o método implementado não trabalha com probabilidade (veja capítulo 2), sempre que o robô passar por uma célula vizinha a uma com obstáculo, quando o obstáculo ocupa apenas parte da célula, os sensores não o acusarão e o valor desta célula será alterado. Ou seja, ao executar a trajetória pela segunda vez com o mapa completo, essa célula que contém parte do obstáculo será alterada para o valor 2. Em uma próxima execução, o planejamento incluirá esta célula e ao aproximar-se dela o robô acionará novamente seu comportamento reativo, como mostra a figura 4.3 (b).

(a) Com o mapa completo, o robô executa a trajetória planejada

(b) Com a manutenção do mapa, as células com obstáculos próximas a trajetória são alteradas

Figura 4.3: Trajetórias após a construção e manutenção do mapa

Contudo, essa alteração nas células é necessária, pois quando o obstáculo é retirado, a cada nova execução, o robô atualizará o mapa e otimizará a trajetória, como mostra a figura 4.4.

4.3.1 Considerações sobre o Método

Com a construção e manutenção do mapa, o problema de atualização do mapa citado no método baseado em pré-mapeamento é solucionado.

O método implementado é bastante simples e não trabalha com probabilidade de ocupação das células para a manutenção do mapa. Desta forma o desempenho do robô oscilará com relação aos obstáculos fixos do ambiente, ou seja o robô "esquece" o canto dos obstáculos. Contudo esta característica garante a questão da manutenção do mapa.

(a) O robô usa as células alteradas pela execução anterior

(b) Novas células são alteradas e o robô aproxima-se do obstáculo real

(c) A trajetória é otimizada

Figura 4.4: Esquecimento do obstáculo

4.4 Conclusão

Neste capítulo foram apresentados dois métodos híbridos. O primeiro baseado em pré-mapeamento e o segundo na construção e manutenção do mapa em tempo de execução.

O método baseado em pré-mapeamento usa um modelo do ambiente para o planejamento da trajetória. Contudo este mapa não é atualizado em tempo de execução. Após o ambiente sofrer grandes alterações faz-se necessário um novo mapeamento.

Já o método de construção e manutenção do mapa em tempo de execução altera o mapa a cada interação do robô com o ambiente. Isso proporciona uma maior autonomia, pois mesmo grandes mudanças no ambiente poderão ser detectadas e a atualização constante do mapa permite ao robô desempenhar sua tarefa.

Uma questão importante a salientar, quanto à implementação de métodos híbridos é o compromisso entre planejamento e reação. Para o método baseado em pré-mapeamento é importante que o método seja reativo diante da presença de obstáculos dinâmicos, contudo é necessário manter informações a respeito da posição do robô e dos obstáculos mapeados, buscando com isso evitar que o robô seja conduzido para uma situação de colisão na qual simplesmente a parte reativa não consiga contornar. Um exemplo deste tipo de situação é onde um obstáculo dinâmico forma um funil com um obstáculo pré-mapeado.

Para que o comportamento de "previsão" seja implementado é necessário acrescentar um nível cognitivo, que permita ao robô "raciocinar" sobre sua posição e "perceber" a necessidade de alterar a estratégia, por exemplo buscando contornar o obstáculo pelo outro lado. O mesmo ocorre no método de construção e manutenção do mapa em tempo de execução, através da leitura dos sensores e de quais sensores estão mais ativos o robô deve ter a ca-

pacidade de "prever" uma situação de colisão futura e "encontrar" uma forma de tratá-la adequadamente.

Capítulo 5

Simulações e Análise de Resultados

5.1 Introdução

No presente capítulo são descritas simulações, em vários ambientes, de todos os métodos implementados, a fim de ressaltar as características de cada um. Ao longo do capítulo é possível constatar os melhoramentos sucessivos oferecidos por cada uma das abordagens ao problema da navegação autônoma. Ao final apresentam-se exemplos de execução dos métodos híbridos, que se caracterizam pela robustez e "inteligência" com que tratam o problema do planejamento.

Todas as simulações foram feitas utilizando-se o simulador Khepera [1].

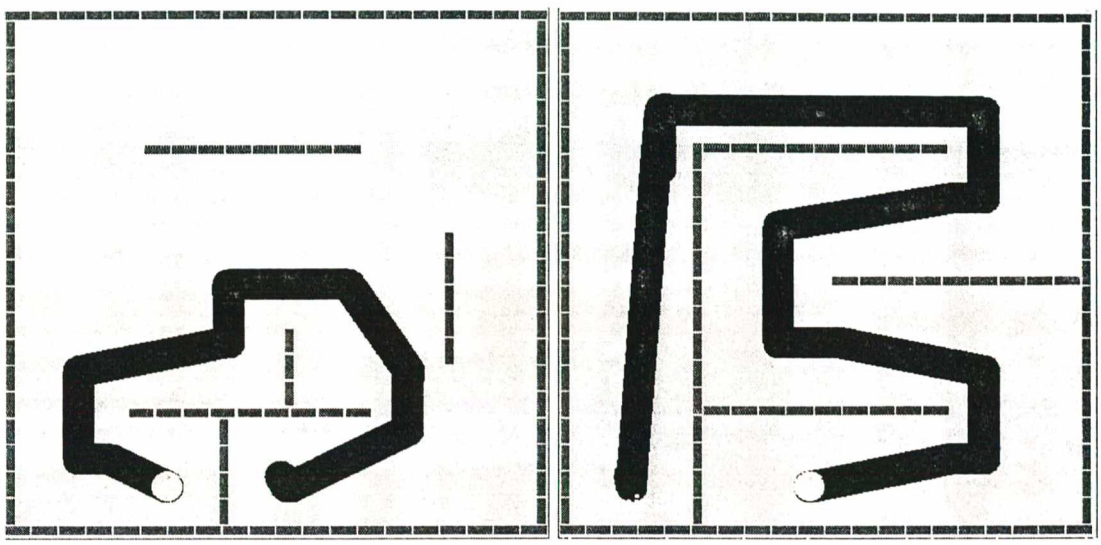
5.2 Abordagem Planejada

5.2.1 1^a Implementação - Algoritmo da Reta

O método do algoritmo da reta apresenta bom desempenho, isto é, encontra um caminho que conduza o robô do ponto inicial ao ponto final, em ambientes que possuam poucos obstáculos, uma vez que é necessário que haja um caminho de células assinaladas com 0 entre estes pontos. A figura 5.1 apresenta dois exemplos de ambientes com poucos obstáculos e onde o método apresenta um bom desempenho.

O método também apresenta bons resultados em ambientes onde os obstáculos formam um "U". O robô consegue sair e entrar do "U" pela característica do algoritmo de assinalar três pontos de borda. A figura 5.2 apresenta dois exemplos em ambientes deste tipo.

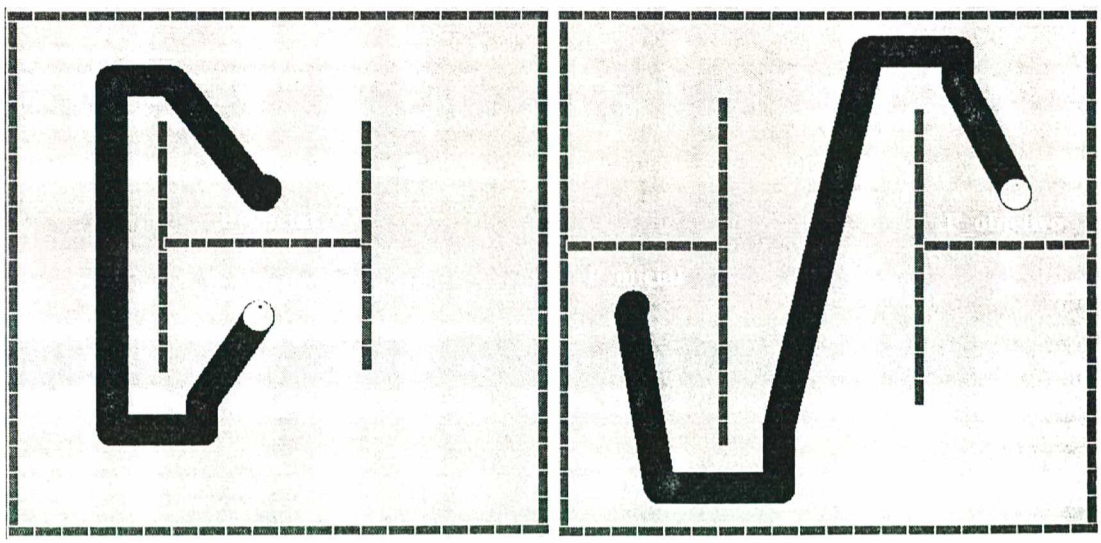
[1] Descrito no apêndice A.



(a) Apesar de se tratar de um obstáculo composto, o algoritmo obtém um bom desempenho.

(b) O robô desvia de obstáculos simples, atingindo o ponto objetivo.

Figura 5.1: Exemplos de execução com o algoritmo da reta

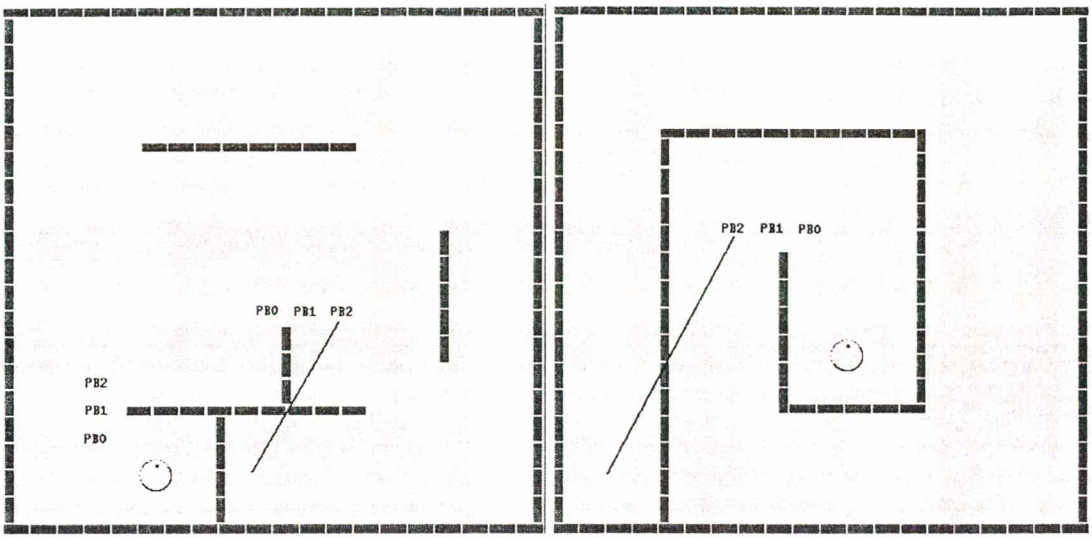


(a) O método permite ao robô entrar e sair facilmente de ambientes do tipo "U" devido a sua característica de assinalar 3 células de borda.

(b) Para um ambiente simples em forma de "U" o robô obtém um bom resultado.

Figura 5.2: Trajetórias em ambientes do tipo "U"

Contudo, há casos em que o algoritmo não conseguirá retornar um caminho. Um dos casos deve-se ao problema citado no capítulo 3 onde a reta traçada encontra um canto exato, como apresentado na figura 5.3(a). Outro caso ocorre devido a posição do ponto objetivo, como na figura 5.3(b), onde é necessário sair do caracol para alcançá-lo.



(a) O método não conseguirá retornar a trajetória, pois a iteração sempre colidirá com o ponto de borda.

(b) O método não conseguirá retornar uma trajeória para sair do caracol, pois sempre encontrará o mesmo ponto de borda.

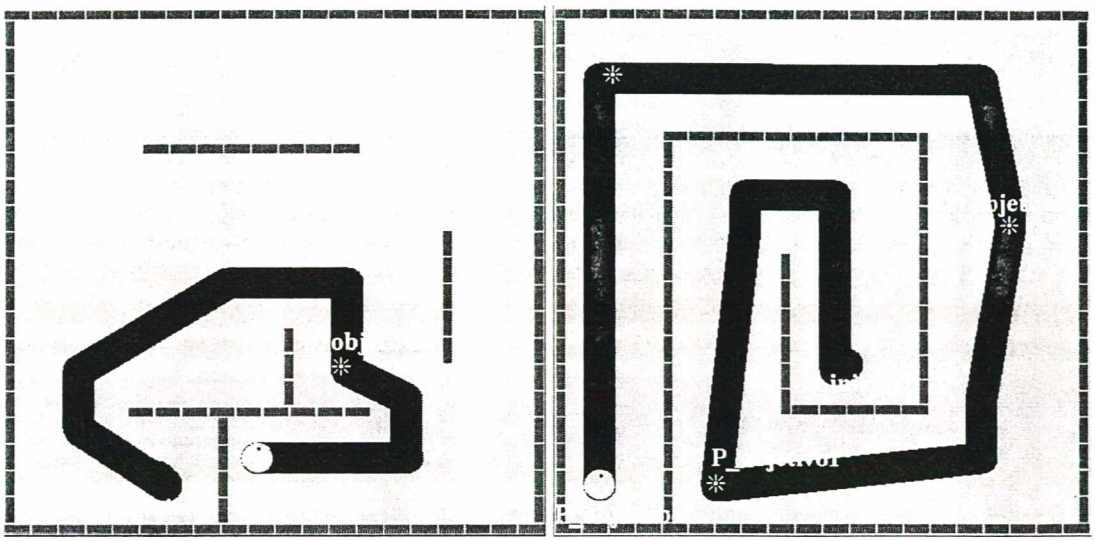
Figura 5.3: Exemplos onde o método não atingirá o ponto objetivo

Este método foi implementado para aceitar múltiplos pontos objetivos, isso permite que algumas destas falhas sejam contornadas, pois tendo pontos intermediários na trajetória, o método consegue sair das situações críticas, como apresentado na figura 5.4

Com a implementação assinalando vários pontos, torna-se possível melhorar as execuções do robô pois as limitações do método são contornadas, além de ser possível a realização de tarefas pelo robô. A figura 5.5 apresenta mais dois casos executados com múltiplas luzes no ambiente.

Avaliação dos Resultados

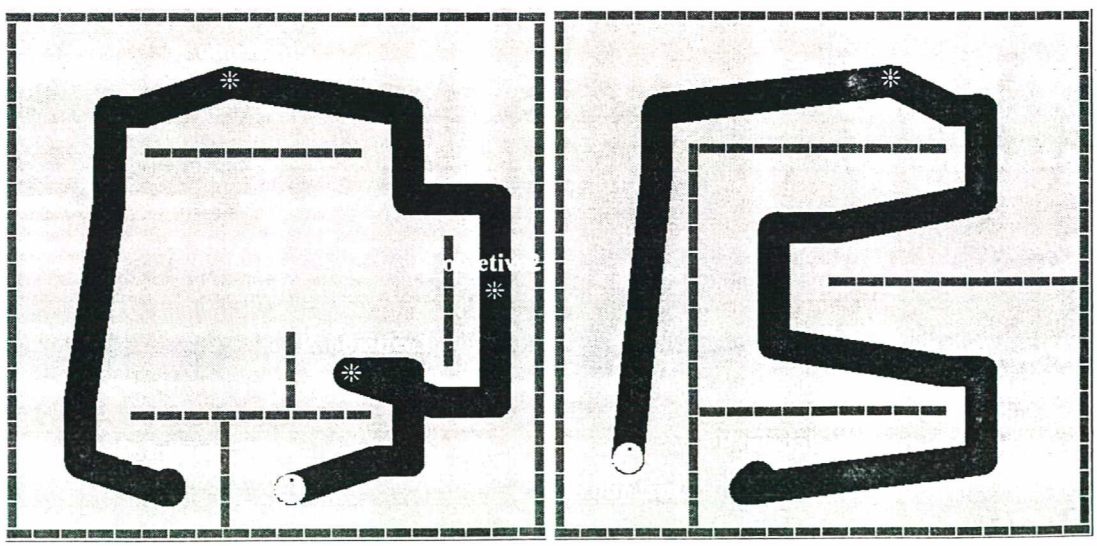
O método é pouco robusto, pois para muitos casos, onde há somente um ponto objetivo (dependendo da localização deste), o método não conseguirá retornar um caminho, pois não há generalização de seu comportamento. Além disso, o método é restrito pois trabalha somente em ambientes com poucos obstáculos, com paredes horizontais e verticais. A



(a) Acrescentando-se um ponto, é possível fazer o contorno do obstáculo adequadamente.

(b) Marcando pontos auxiliares é possível sair do caracol.

Figura 5.4: Uso de pontos auxiliares



(a) O robô simula a realização de tarefas, passando por vários pontos.

(b) Mesmo para ambientes simples o método necessita de um ponto auxiliar para desempenhar a trajetória.

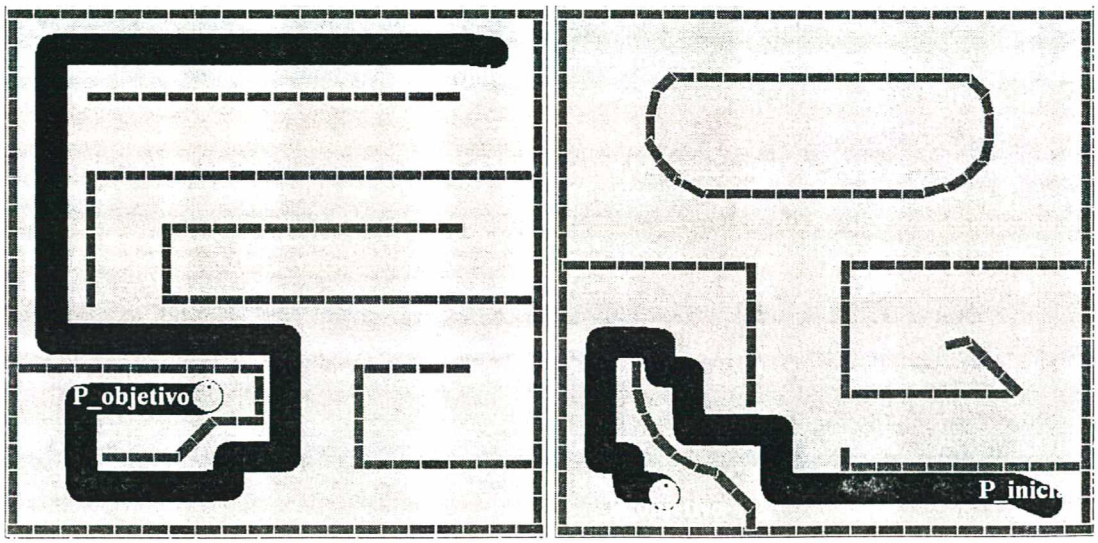
Figura 5.5: Trajetórias com vários pontos objetivo

complexidade do algoritmo de planejamento cresce de acordo com a complexidade do ambiente que se quer utilizar. O método apresenta ainda limitações, uma vez que não utiliza trajetórias em diagonal por necessitar de três pontos livres na extremidade dos obstáculos.

O desenvolvimento de métodos que buscam reconhecer e tratar cada situação específica é uma tarefa difícil. Mesmo para ambientes estáticos, pode ser infactível generalizar o algoritmo de planejamento, dadas situações muito semelhantes, nas quais o tratamento deve ser distinto.

5.2.2 2^a Implementação - Árvore de Busca 4NF

O método de busca de caminho gerando uma árvore sobre o espaço de estados sempre retornará o menor caminho. Como somente foram utilizados os nós horizontal e verticalmente adjacentes, o método retornará caminhos dados por segmentos de reta. Além disso, como a expansão dos obstáculos não é considerada, o robô ao executar a trajetória passa próximo ao obstáculo, sem contudo, colidir. A figura 5.6 apresenta dois casos onde observa-se a trajetória descrita por linhas horizontais e verticais e a proximidade com os obstáculos.

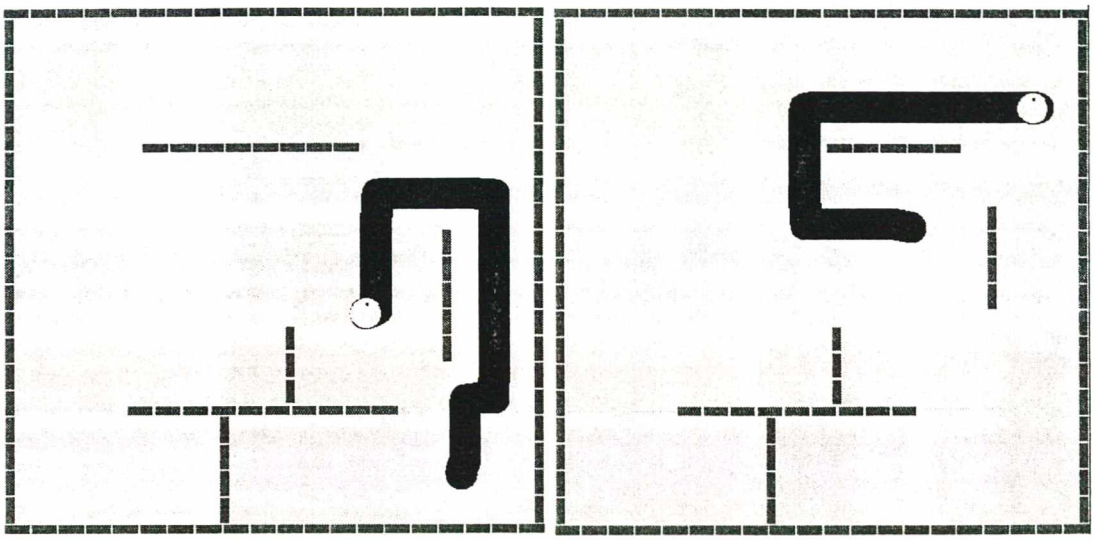


(a) O método sempre retorna o menor caminho.

(b) Por não utilizar a expansão dos obstáculos, o robô passa rente ao obstáculo.

Figura 5.6: Trajetórias dadas por segmentos de reta

Contudo, como o caminho em diagonal não é utilizado, o robô contorna os obstáculos para alcançar o ponto objetivo quando o caminho mais curto seria utilizar as células diagonalmente adjacentes. Dois exemplos são mostrados na figura 5.7



(a) O robô contorna a parede para atingir o ponto objetivo.

(b) O menor caminha seria utilizar as células em diagonal.

Figura 5.7: Casos onde o caminho diagonal possibilitaria a menor trajetória

Além disso, como somente o corte nos nós repetidos em um mesmo ramo é feito, há casos em que a geração da árvore deflagra o estouro da pilha. A explosão combinatória depende da quantidade de níveis gerados até o ponto objetivo e de quantos nós houverem por nível. A figura 5.8 apresenta um caso onde houve a necessidade de restrição do ambiente.

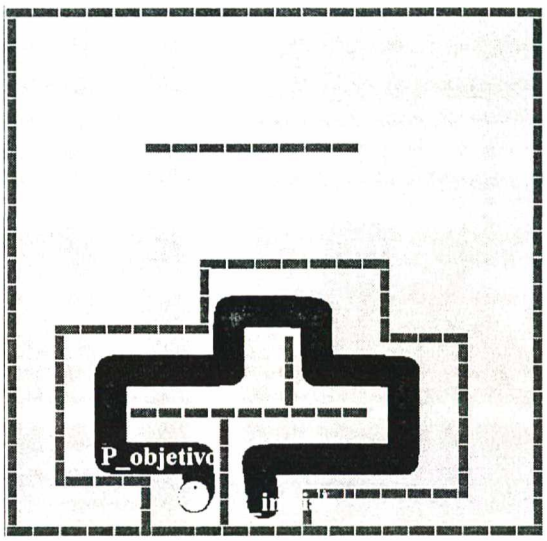
Avaliação dos Resultados

A implementação do método de geração de trajetória através da criação de uma árvore de busca sobre o espaço de estados é muito mais simples e mais eficiente do que o método do algoritmo da reta. Sua principal vantagem está na expansão da árvore de busca, onde não é necessário avaliar cada situação específica para contorno dos obstáculos, isso garante o retorno de um caminho mínimo e livre de colisão, o que torna o método mais robusto.

Além disso, contorna as restrições impostas pelo método do algoritmo da reta, pois qualquer tipo de ambiente pode ser tratado (com paredes horizontais, verticais ou com inclinação).

Contudo, ainda é limitado no que se refere ao tipo de trajetória, pois somente executa trajetórias horizontais e verticais.

Devido à ausência de uma estratégia de corte, o algoritmo apresenta um elevado tempo de execução e há o problema da explosão combinatória em ambientes que possuam um



(a) Como há poucos obstáculos fez-se necessário restringir o ambiente para evitar o estouro da pilha.

Figura 5.8: Explosão combinatória

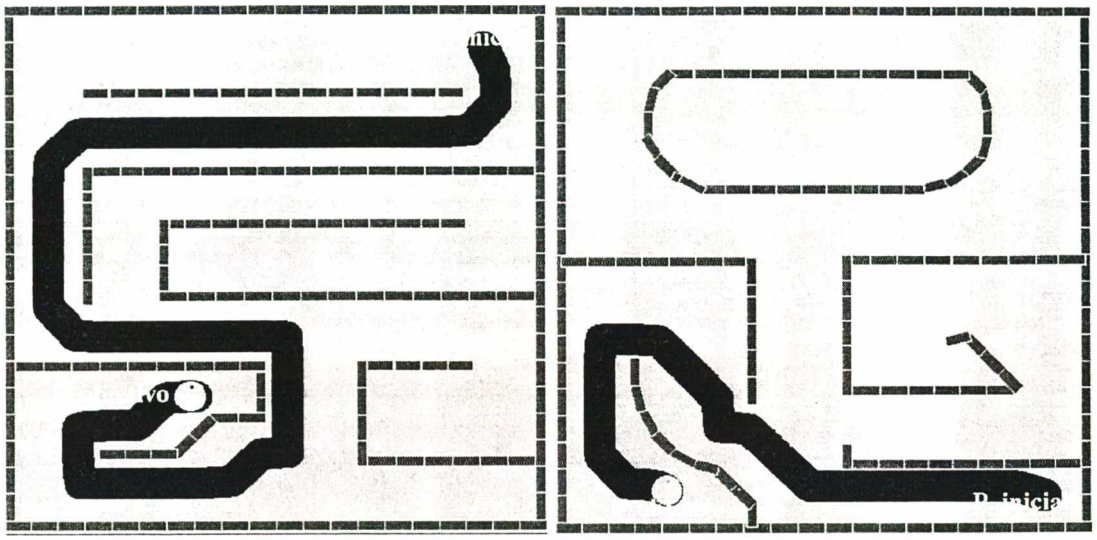
grande número de células vazias ou para o qual a árvore gerada deva atingir grande profundidade.

5.2.3 3^a Implementação - Árvore de Busca 8NF

Esta implementação parte da anterior, buscando solucionar suas falhas (aproveitando as células livres em diagonal e restringindo a geração de nós da árvore).

Para viabilizar o uso das células diagonais, utilizou-se o mapeamento com obstáculos expandidos. Como citado na descrição do método, somente quando a expansão da árvore se der para um nó diagonal, a expansão do obstáculo será considerada. Desta forma, os caminhos em diagonal são aproveitados e mantêm-se a proximidade com obstáculos, o que permite que o robô navegue em ambientes com muitos obstáculos, próximos uns dos outros, como apresenta a figura 5.9. Na simulação apresentada, foram mantidos os mesmos casos de teste realizados anteriormente (para o método 4NF), com o objetivo de comparar as execuções.

Vários testes foram feitos alterando-se a ordem com a qual os nós da árvore são expandidos. Pode-se perceber que há um compromisso entre como a expansão é feita e a estratégia de corte da árvore. Nesta implementação expandem-se inicialmente os nós horizontais e ver-



(a) A trajetória gerada é mais suave.

(b) Mantém-se a proximidade com os obstáculos.

Figura 5.9: Trajetórias em diagonal

ticais seguindo o sentido horário e, em seguida os diagonais, novamente no sentido horário. Isso garante a realização de caminhos em linha reta, como mostrado no capítulo 3, sem prejudicar os caminhos em diagonal.

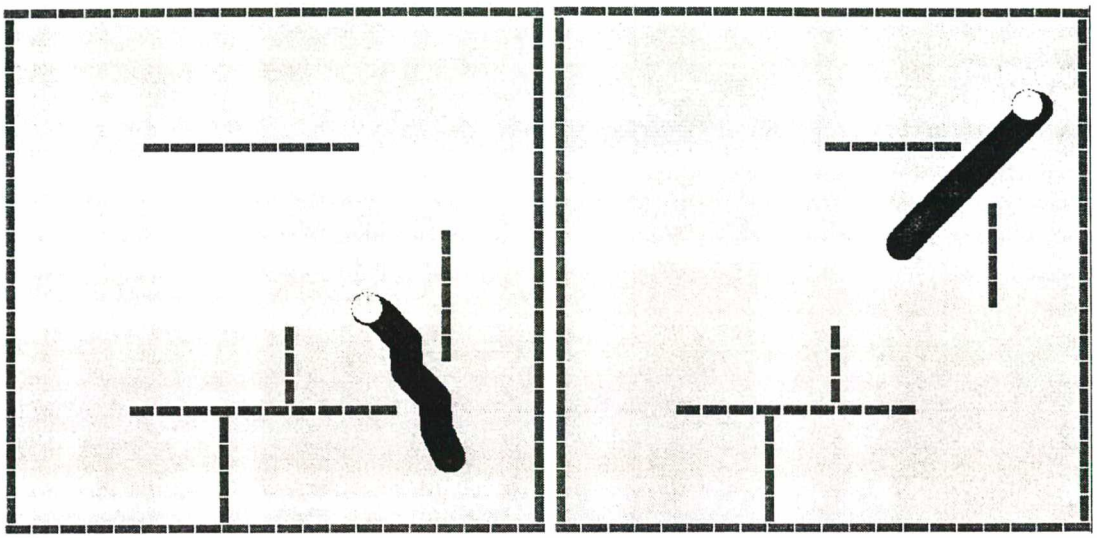
Com o aproveitamento dos caminhos em diagonal, figura 5.10, o robô executa trajetórias mais curtas e elegantes para os casos onde antes contornava os obstáculos.

Como o método implementa agora a estratégia de corte da árvore de busca, mesmo em ambientes onde a profundidade da árvore de busca gerada é elevada ou há muitas células vazias, a explosão combinatória é evitada. A figura 5.11 mostra casos onde a estratégia de corte da árvore permite a execução da trajetória em ambientes onde anteriormente ocorriam estouros de pilha.

Avaliação dos Resultados

O último método da abordagem planejada mostrou-se mais robusto e genérico do que os anteriores. Robusto, pois o método sempre retornará o menor caminho até o ponto objetivo, independente dos tipos de obstáculos presentes no ambiente, e genérico, uma vez que não há necessidade de se tratar especificamente cada situação para contorno dos obstáculos, como acontecia no método baseado no algoritmo da reta.

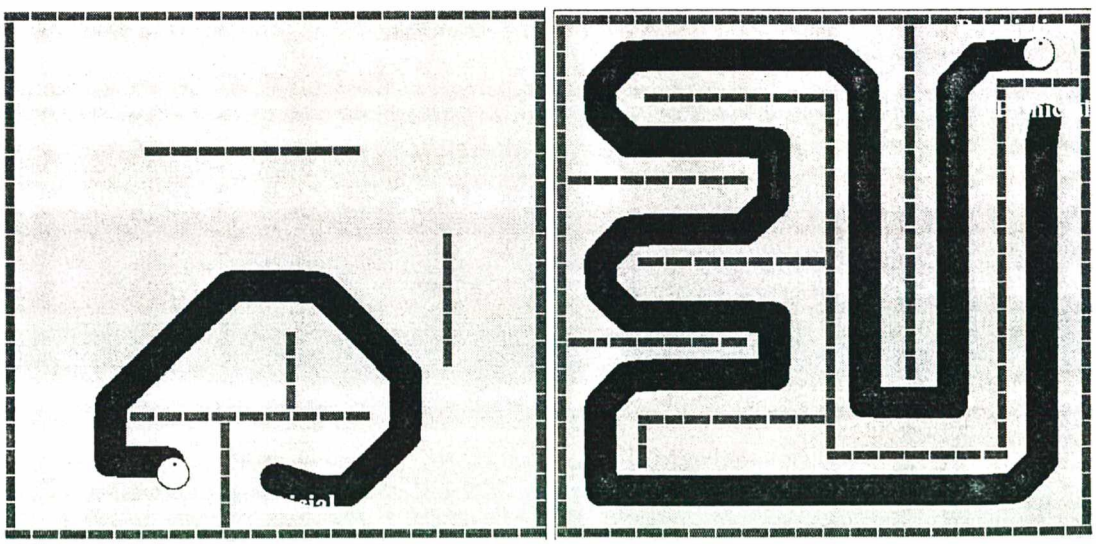
O tempo computacional envolvido na construção da árvore foi reduzido e o problema



(a) O robô aproveita o caminho diagonal.

(b) As trajetórias executadas são agora mais curtas.

Figura 5.10: Caminho utilizando células diagonais



(a) A estratégia de corte impede a explosão combinatória, e agora não é mais necessário restringir o ambiente.

(b) O método consegue agora, gerar árvores com elevada profundidade.

Figura 5.11: Evitando a explosão combinatória

da explosão combinatória contornado com a implementação da estratégia de corte.

5.2.4 Conclusão sobre os métodos da Abordagem Planejada

A implementação do método baseado no algoritmo da reta teve como objetivo apresentar as dificuldades encontradas no desenvolvimento de sistemas sem capacidade de generalização. Tratar cada situação específica requer o conhecimento global a respeito do problema, de forma que as ações de controle consigam conduzir o robô da posição inicial à posição final. Ações de controle geradas a partir de um conhecimento local, como no método implementado, recaem muitas vezes, em uma tomada de decisão errada, comprometendo o desempenho ou mesmo impedindo a construção do caminho e geração da trajetória.

Apesar de serem soluções pontuais, sistemas específicos para um tipo de tarefa ou determinado ambiente têm sido bem aceitos pela comunidade científica, dada a complexidade de se construir sistemas robustos e adaptáveis.

Os outros dois métodos implementados são mais genéricos, pois fazem uso de técnicas de busca [40, 41] percorrendo todo o espaço de estados para encontrar a solução. Essas técnicas oferecem a possibilidade de representar o espaço de estados como um grafo [39] e transformar o problema da trajetória em um problema de busca de caminho mínimo usando algum tipo de algoritmo específico como Dijkstra ou A*.

Os métodos que unem a abordagem planejada às técnicas da inteligência artificial clássica são mais robustos e genéricos do que os métodos que tentam tratar todas as possíveis situações com que o robô possa se defrontar no ambiente. Contudo, essa união apresenta limitações que advém da necessidade de manutenção do modelo do ambiente.

O principal problema encontrado no emprego de métodos da abordagem planejada refere-se justamente à dificuldade de obter modelos precisos de ambiente reais, pois em geral, ambientes reais possuem características dinâmicas.

Os recursos computacionais necessários para manutenção dos modelos do ambiente crescem exponencialmente com a complexidade do ambiente e com as modificações que nele possam ocorrer, tornando impossível manter um modelo completo e detalhado de um ambiente real altamente estruturado.

Sendo assim, o emprego de métodos da abordagem planejada restringe-se a ambientes estáticos e controlados. Além disso, esses métodos somente fazem uso do mapa do ambiente construído, não considerando a leitura dos sensores em tempo de execução, de forma que

qualquer mudança no ambiente posterior à construção do mapa pode implicar a colisão do robô.

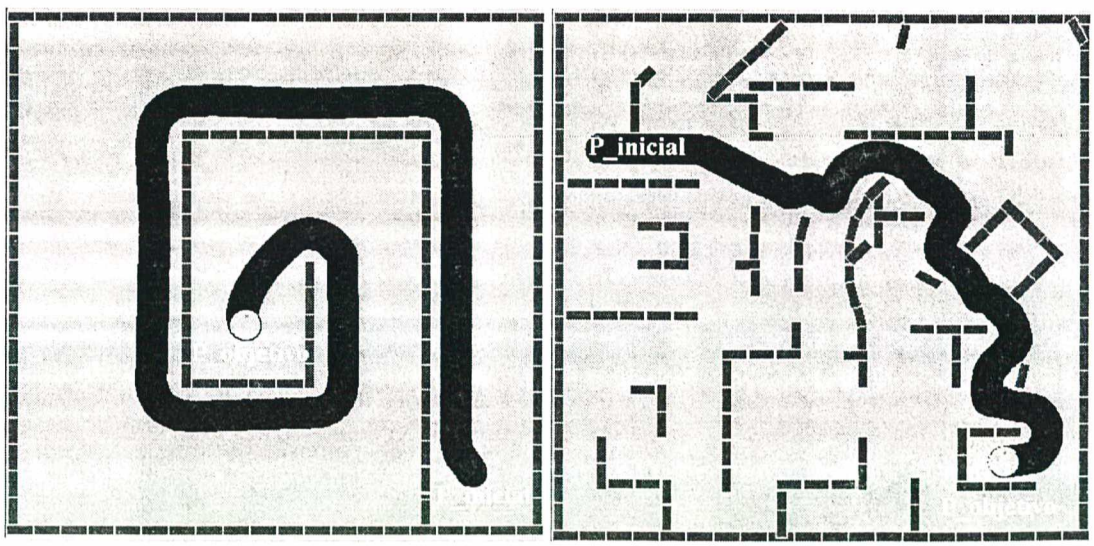
5.3 Abordagem Reativa

5.3.1 1ª Implementação - Navegação Direcionada Utilizando RNAs

Por conhecer sua localização e a localização do ponto objetivo, o robô consegue um desempenho melhor do que o visto em [53]. Com a navegação direcionada o robô não navega aleatoriamente até que os sensores de luminosidade sejam ativados, mas encaminha-se ao ponto objetivo desde o início da execução.

Contudo, como o robô reage aos estímulos recebidos pelos sensores e somente dirige-se ao ponto objetivo quando não há colisão, seu desempenho está diretamente relacionado com a disposição dos obstáculos e do ponto objetivo no ambiente e com as ações de controle emitidas pelas redes neurais que simulam os comportamentos *desviar obstáculos* e *procurar objetivo*.

Sendo assim, há ambientes onde a localização e/ou o formato dos obstáculos auxilia o robô a chegar até o ponto objetivo, como mostra a figura 5.12.



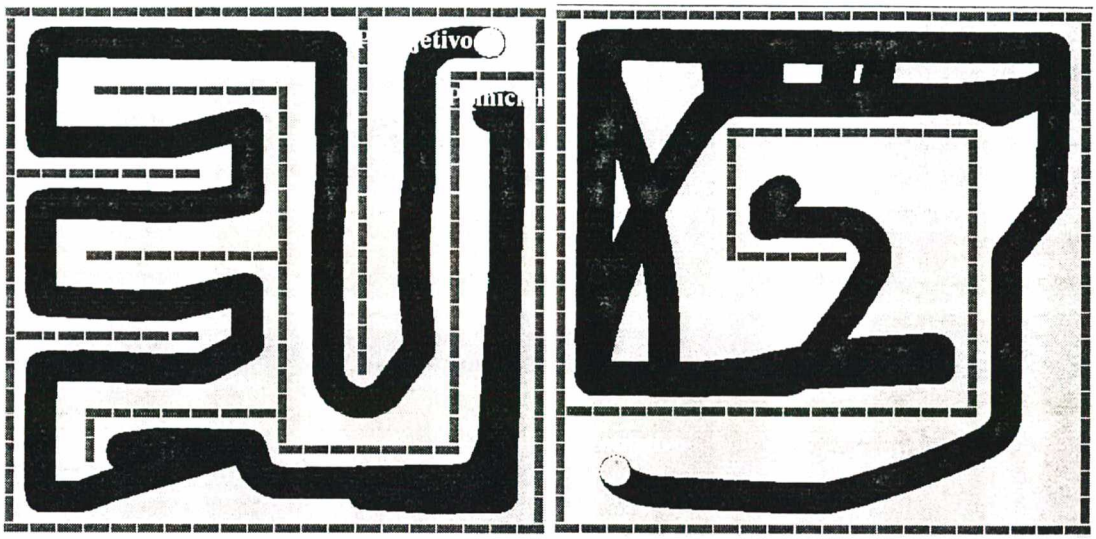
(a) Ao contornar as paredes, o robô aproxima-se do ponto objetivo.

(b) A disposição dos obstáculos auxilia o robô a alcançar o ponto objetivo.

Figura 5.12: Influência do ambiente na execução da trajetória

Nas execuções mostradas na figura 5.13, as ações de controle emitidas pela rede neural

desviar obstáculo (comportamento similar ao de seguir parede) fazem com que o robô consiga alcançar o ponto objetivo.



(a) O robô navega pelo labirinto seguindo as paredes .

(b) Após várias tentativas, o robô consegue sair do obstáculo em forma de caracol.

Figura 5.13: Comportamento *seguir parede* auxilia no contorno dos obstáculos

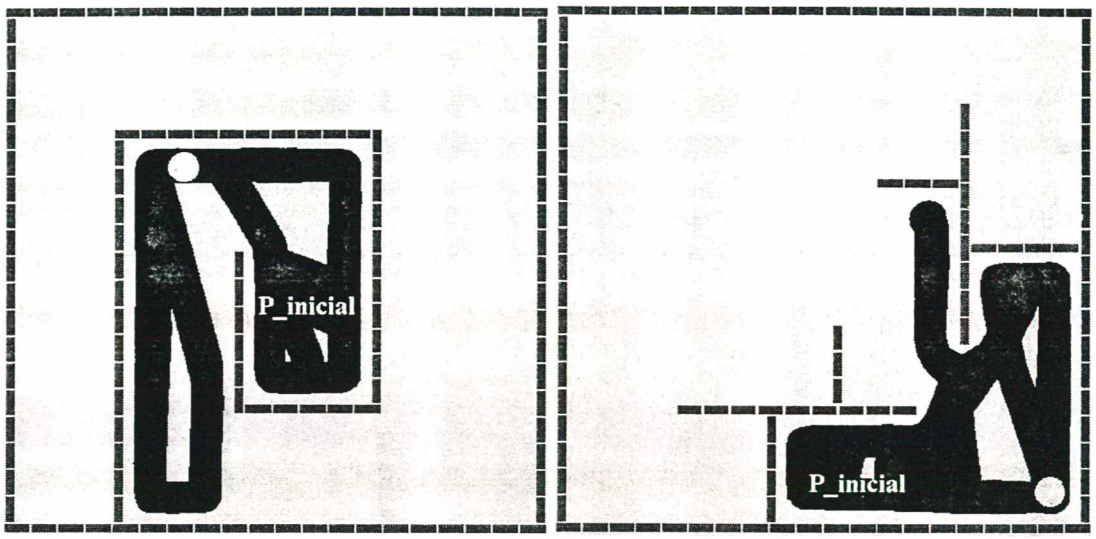
É importante ressaltar que no treinamento da rede neural *procurar objetivo*, não utilizou-se padrões onde os sensores traseiros do robô estivessem ativos.

Contudo, somente os comportamentos de procurar luzes no ambiente e desviar obstáculos, muitas vezes não são suficientes para que o robô contorne grandes obstáculos e atinja o ponto objetivo, como mostra a figura 5.14. Se faz necessário que o robô tenha o conhecimento global acerca do ambiente para poder alcançar o ponto objetivo.

Além disso, como nenhum tipo de planejamento de trajetória é realizado, o robô pode colidir ou ficar preso entre dois obstáculos, como mostrado na figura 5.15. Em (a), ao desviar do obstáculo 1, o robô colide com o obstáculo 2. Em (b), como as paredes formam um funil e somente os sensores laterais são ativados (os sensores dianteiros não acusam obstáculo), o robô ficará preso ao tentar passar.

Avaliação dos Resultados

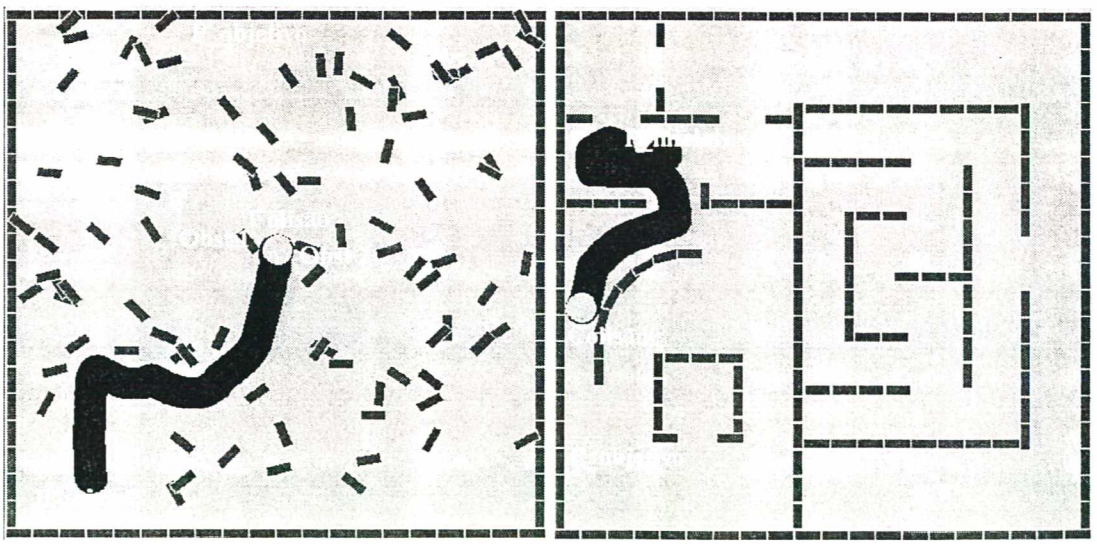
O método implementado apresenta bom desempenho em ambientes onde a disposição dos obstáculos e do ponto objetivo auxiliam na navegação do robô ou em ambientes onde o comportamento da rede neural permite ao robô sair do obstáculo.



(a) O robô não conseguirá sair do caracol, pois ao sair de uma situação de colisão, a falta de conhecimento global fará com que ele volte a esta situação.

(b) O robô consegue sair da sala onde se encontra, mas não tem o conhecimento global para atingir o objetivo, voltando a navegar pela sala.

Figura 5.14: A falta de conhecimento global impede o robô de atingir o objetivo



(a) O robô colidirá ao tentar desviar do obstáculo 1, pois não há como saber se o espaço entre os obstáculos é suficiente.

(b) O robô ficará preso ao tentar atravessar o obstáculo em forma de funil, pois os sensores dianteiros não são ativados.

Figura 5.15: A falta de um planejamento da trajetória prejudica a execução do robô

Sendo assim, percebe-se a necessidade do conhecimento global do ambiente, para evitar os casos onde o robô executa exaustivas tentativas de atingir o ponto objetivo voltando à uma situação de colisão.

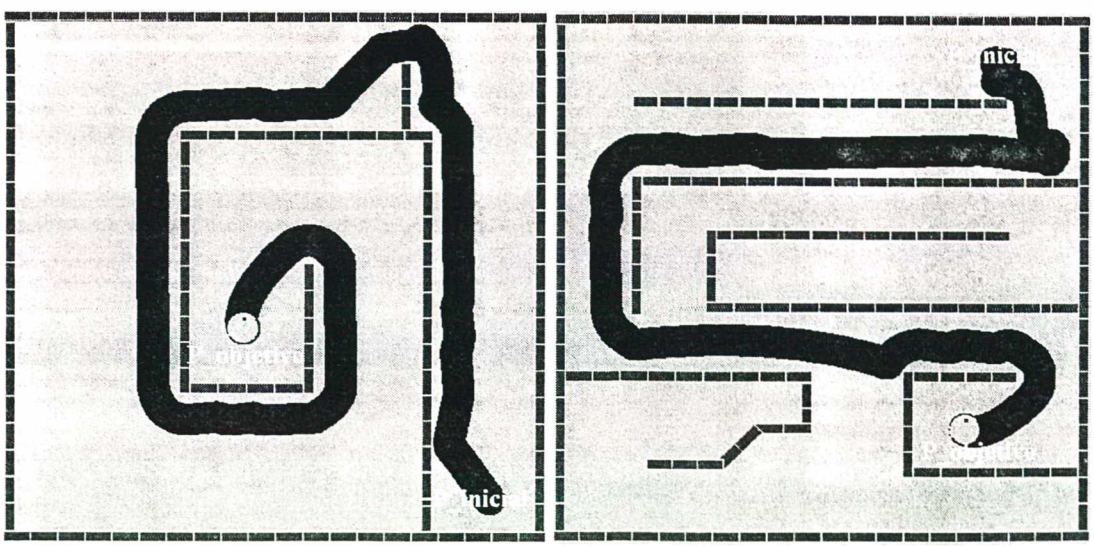
O fator predominante para o bom desempenho está na escolha do conjunto de treinamento. Contudo, deve-se manter o compromisso com a generalidade. Acrescentar ao conjunto de treinamento padrões muito específicos faz com que a rede se torne específica. Isso melhorará o desempenho em alguns casos, e piorará em outros. Além disto, questões com a ambigüidade dos padrões também deve ser observada.

Umas das formas de solucionar o problema da escolha do conjunto de treinamento é permitir ao robô "aprender" como desviar dos obstáculos.

5.3.2 2^a Implementação - Aprendizado Hebbiano

O método de aprendizado Hebbiano apresenta um desempenho similar ao método de navegação direcionada, ou seja, o robô conseguirá um bom desempenho em ambientes onde a localização e/ou o formato dos obstáculos ajudem-no a atingir o ponto objetivo.

A figura 5.16 apresenta dois exemplos de execuções onde as paredes "guiam" o robô ao ponto objetivo.



(a) O robô atinge o objetivo ao contornar as paredes, entrando no caracol.

(b) As paredes e a localização do ponto objetivo auxiliam o robô.

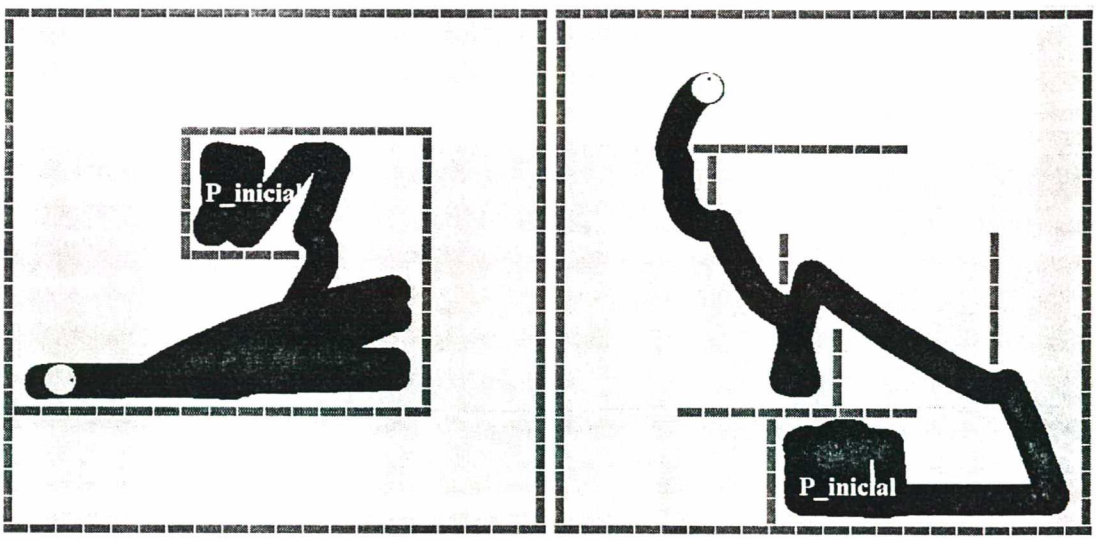
Figura 5.16: A influência do ambiente permanece no aprendizado Hebbiano

Em ambientes onde é necessário ter um conhecimento global (sair de um "U" ou de um

caracol), o robô não conseguirá atingir o objetivo. Além da falta de conhecimento global, seu comportamento reativo não lhe permite seguir as paredes virando à direita, pois ele sempre virará à esquerda diante de uma situação de colisão frontal.

Na figura 5.17 (a) o robô não consegue sair do caracol, devido a reação de virar à esquerda diante de uma colisão frontal. Isso inibe a possibilidade de seguir a parede vertical e sair do obstáculo como acontece com a navegação direcionada (veja figura 5.13 (b)).

Contudo, há casos em que o robô consegue atingir o ponto objetivo justamente por virar à esquerda, como acontece na figura 5.17 (b).



(a) Por sempre virar a esquerda, o robô não seguirá a parede vertical à esquerda do ambiente.

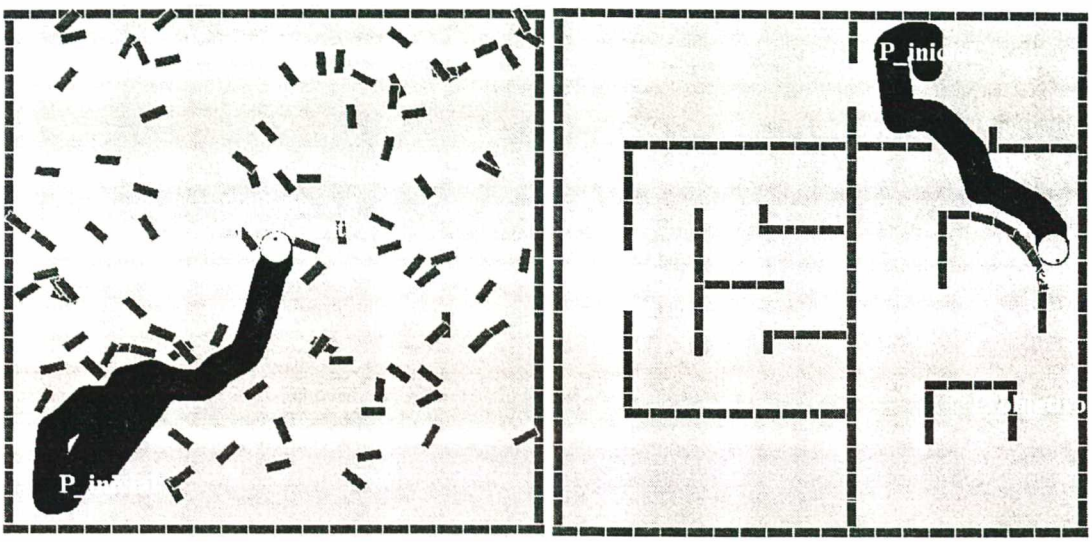
(b) O robô conseguirá sair do obstáculo em forma de "U" por seguir a parede e virar à esquerda.

Figura 5.17: Limitação do comportamento devido às características do método

Assim como no método de navegação direcionada, por não possuir conhecimento global acerca do ambiente e não fazer nenhum tipo de planejamento, há casos em que o robô ficará preso em uma situação de colisão, onde somente a ação reativa não conseguirá contornar. A figura 5.18 apresenta os mesmos casos citados no método acima. Em (a) o robô fica preso entre dois obstáculos. Em (b) o afunilamento do ambiente ocasiona a colisão do robô.

Avaliação dos Resultados

O método é menos genérico que o método da navegação direcionada, devido à sua simplicidade e à restrição dos reflexos implementados (sempre virar à esquerda). Contudo,



(a) A falta de conhecimento global impede o robô de perceber se há espaço sufiente entre dois obstáculos.

(b) Como os sensores frontais não indicam colisão, o robô fica preso quando as paredes afunilam.

Figura 5.18: Necessidade do conhecimento do ambiente e de planejamento

apresentará um desempenho similar em situações onde o conhecimento global do ambiente se faz necessário. Isso acontece pois ambos os métodos tem suas ações de controle baseadas nas informações locais vindas dos sensores.

5.3.3 Conclusão sobre os métodos da Abordagem Reativa

Os métodos da abordagem reativa obtêm bons resultados em ambientes desconhecidos e com obstáculos dinâmicos, quando se trata simplesmente da navegação livre de colisão.

O principal problema relacionado com o desvio de obstáculos está relacionado com a imprevisibilidade do ambiente e com a multiplicidade de situações com as quais o robô se depara. Essas situações podem conduzir o robô a uma colisão, a qual somente a ação reativa não consigue contornar.

Quando utilizados para alcançar um ponto objetivo e implementados sem o conhecimento global da localização deste, tais métodos implicam a navegação aleatória do robô, até que algum dos sensores receba alguma informação referente ao ponto objetivo.

Os dois métodos aqui analisados possuem o conhecimento do ponto objetivo *a priori*, eliminando a questão da navegação aleatória. Contudo a falta de um conhecimento global sobre a localização dos obstáculos ainda implica tentativas frustradas de atingir o ponto

objetivo e em muitos casos, o insucesso da busca.

O método da navegação direcionada surge como uma alternativa de melhorar a questão da navegação aleatória descrita em [53].

Já o método de aprendizado Hebbiano apresenta-se como uma alternativa à necessidade de um conjunto de treinamento imposta pela arquitetura *feedforward* do método de navegação direcionada. Apesar de simples e menos genérico, este método apresentou resultados semelhantes ao método da navegação direcionada.

A questão chave para o desenvolvimento de métodos reativos está relacionada com os sensores. Leituras imprecisas e ruídos impedem que a ação de controle seja livre de erro e prejudicam a navegação. Além disso, a escolha dos sensores deve ser feita baseando-se na necessidade de informações para a realização da tarefa.

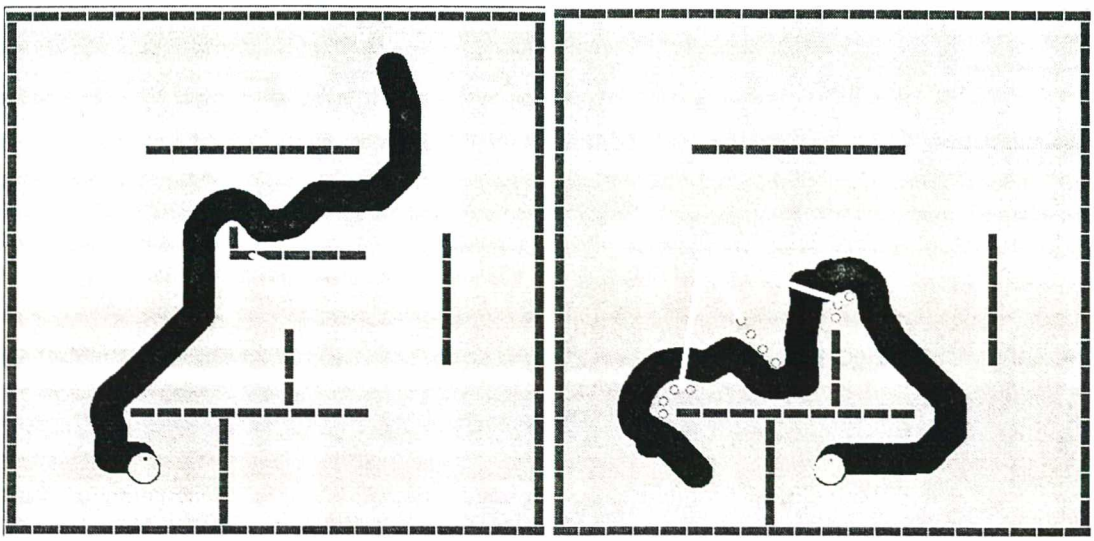
5.4 Abordagem Híbrida

5.4.1 1ª Implementação - Pré-Mapeamento

Com o pré-mapeamento, o robô consegue contornar situações onde somente a abordagem reativa não era suficiente. O robô navega seguindo um plano feito com o conhecimento global do ambiente e desvia dinamicamente dos obstáculos não mapeados respondendo aos sensores em tempo de execução. A figura 5.19 apresenta dois exemplos de execução do robô utilizando o pré-mapeamento.

A associação das partes reativa e planejada possibilita contornar situações onde o método reativo isolado não obtém bons resultados. A figura 5.20 apresenta dois casos onde o robô controlado pelo método da navegação direcionada não conseguiu atingir o ponto objetivo.

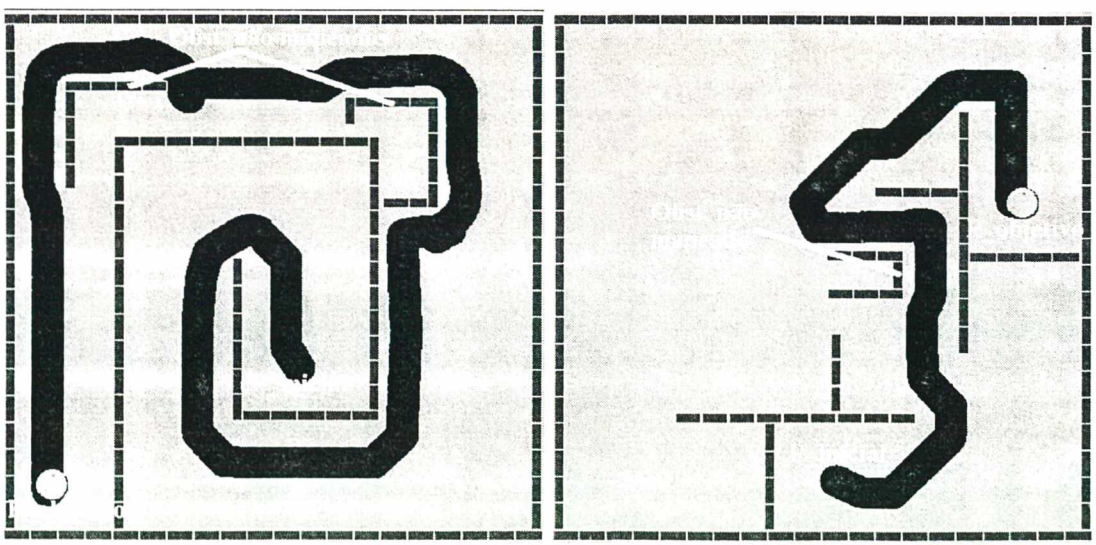
O método baseado em pré-mapeamento permite ao robô interagir em ambientes com obstáculos móveis (tais como pessoas e outros robôs móveis). Nestes ambientes, o robô responderá reativamente a estes obstáculos. A figura 5.21 apresenta um exemplo de execução com dois robôs. O robô 1 é controlado pelo método híbrido e o robô 2 apenas por uma rede neural de colisões. Em (a) o robô 1 inicia a navegação seguindo o plano e o robô 2 navega aleatoriamente. Ao se aproximarem (b), os dois respondem reativamente, efetuando o desvio. Em (c) o plano é refeito e em (d) o ponto objetivo é alcançado.



(a) O robô segue o plano até encontrar o obstáculo. Após desviá-lo, o robô refaz o plano.

(b) O robô age reativamente ao encontrar cada um dos três obstáculos não mapeados.

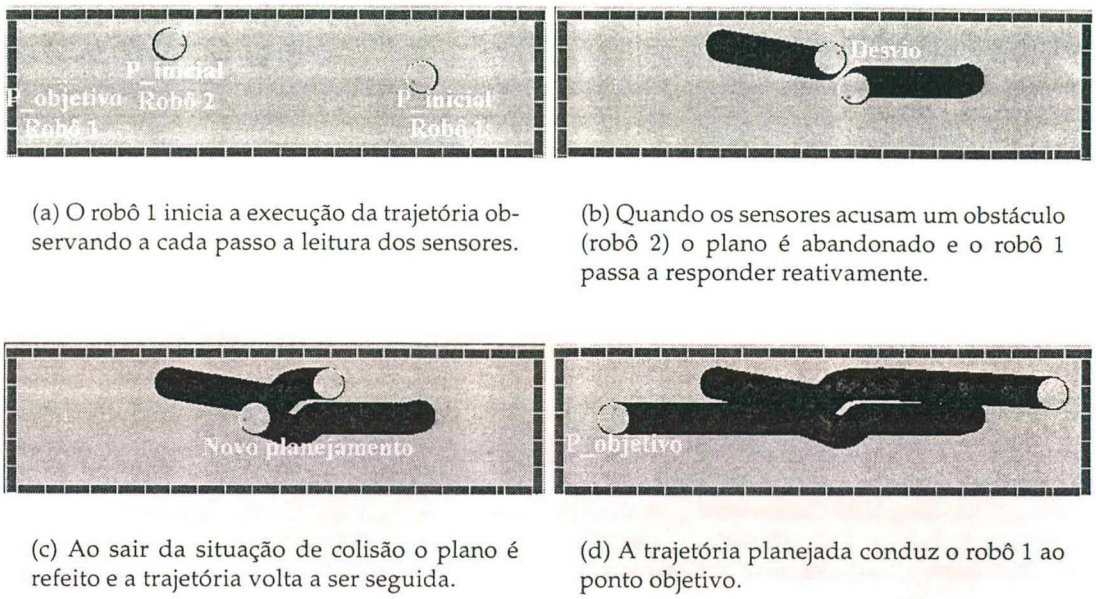
Figura 5.19: O pré-mapeamento associa planejamento e reatividade



(a) Com o conhecimento global do ambiente e a trajetória planejada o robô consegue sair do caracol.

(b) O robô atinge o ponto objetivo em uma situação onde apenas o método reativo não apresentou bom resultado.

Figura 5.20: Unindo conhecimento global e reatividade



(a) O robô 1 inicia a execução da trajetória observando a cada passo a leitura dos sensores.

(b) Quando os sensores acusam um obstáculo (robô 2) o plano é abandonado e o robô 1 passa a responder reativamente.

(c) Ao sair da situação de colisão o plano é refeito e a trajetória volta a ser seguida.

(d) A trajetória planejada conduz o robô 1 ao ponto objetivo.

Figura 5.21: Exemplo de execução em um ambiente com obstáculos dinâmicos móveis

Avaliação dos Resultados

Unindo as abordagens reativa e planejada no método baseado em pré-mapeamento é possível melhorar o desempenho do robô em ambientes conhecidos porém com obstáculos dinâmicos. O robô terá no mapa informações tais como a localização das paredes e mesas e poderá navegar desviando-se reativamente dos obstáculos dinâmicos não mapeados ou móveis.

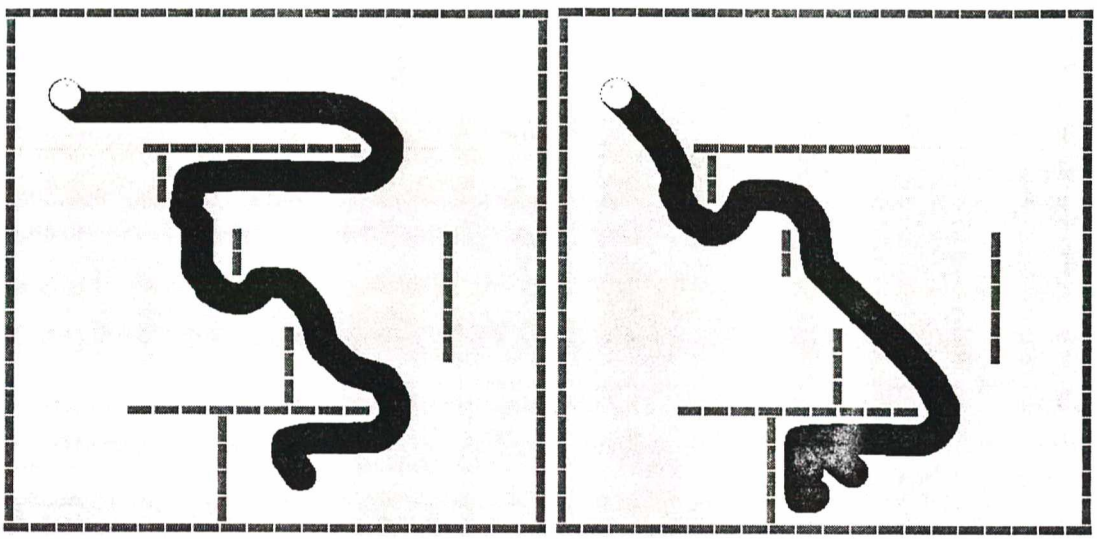
As execuções obtidas aqui são significativamente melhores do que as obtidas em [54], pois não é necessário manipular nenhum tipo de lista enquanto a parte reativa atua. A grande vantagem do método é refazer o plano sempre que o robô sai da situação de colisão.

Contudo, como o mapa não é alterado, as mudanças no ambiente não serão atualizadas. O mapa deverá ser refeito sempre que o ambiente sofrer grandes alterações. Para contornar este problema, o ambiente pode ser "lido" pelos sensores enquanto o robô navega, e o mapa atualizado em tempo de execução.

5.4.2 2^a Implementação - Construção e Manutenção do Mapa em Tempo de Execução

Como inicialmente há somente a informação da posição inicial e final do robô, a navegação para reconhecimento do ambiente e construção do mapa apresenta baixo de-

sempenho. Muitas vezes, nesta primeira execução somente parte dos obstáculos tornam-se conhecidos, fazendo com que na segunda execução o caminho gerado pela árvore de busca ainda utilize células ocupadas. Na figura 5.22 apresentam-se as duas execuções necessárias para que o robô conheça os obstáculos do ambiente.



(a) O robô conhece somente o ponto inicial e o ponto objetivo.

(b) A árvore de busca usa células que contém obstáculos.

Figura 5.22: Aprendizagem do ambiente

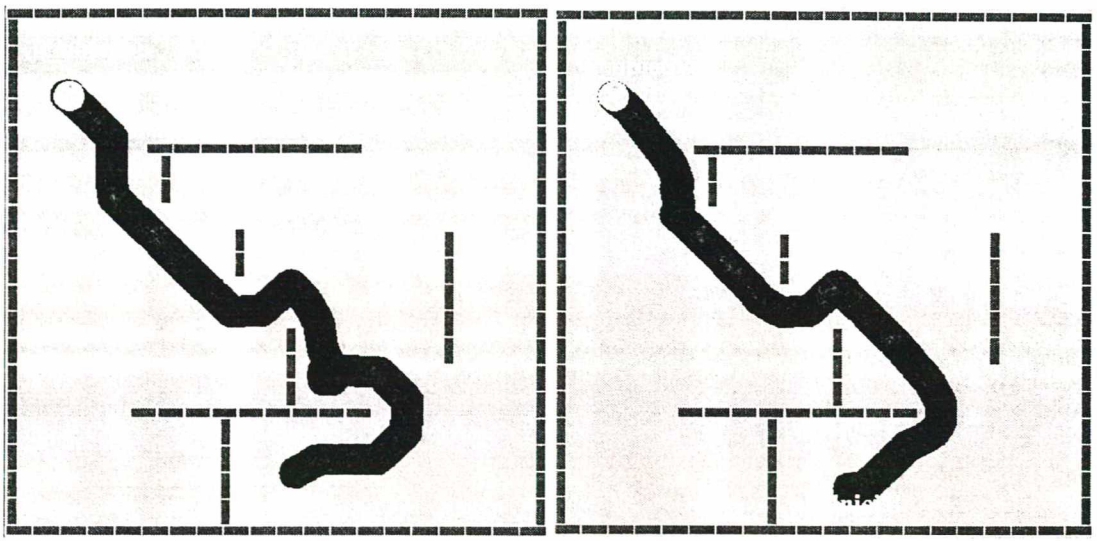
Após "aprendido" o ambiente, é possível perceber uma sensível melhora no desempenho. Na figura 5.23 (a), o robô armazena todas as informações necessárias para descrever a trajetória ótima, mostrada na figura 5.23 (b).

Ao retirar os obstáculos do ambiente, o robô ainda os tem em seu mapa, como é possível observar na execução da trajetória mostrada na figura 5.24 (a). Como os sensores não acusam mais os obstáculos, o mapa é atualizado e a trajetória melhorada, como apresenta a figura 5.24 (b).

Nas figuras 5.25, 5.26, 5.27 apresenta-se outra execução em 6 passos onde é possível perceber as etapas de construção do mapa, planejamento da trajetória e manutenção do mapa.

Na figura 5.25 (a), O robô reconhece todo o ambiente e usa o planejamento para sair da sala. Como o robô passa afastado da parede, o mapa sobre manutenção e na segunda execução (figura 5.25 (b)) a parede é novamente seguida pelo robô.

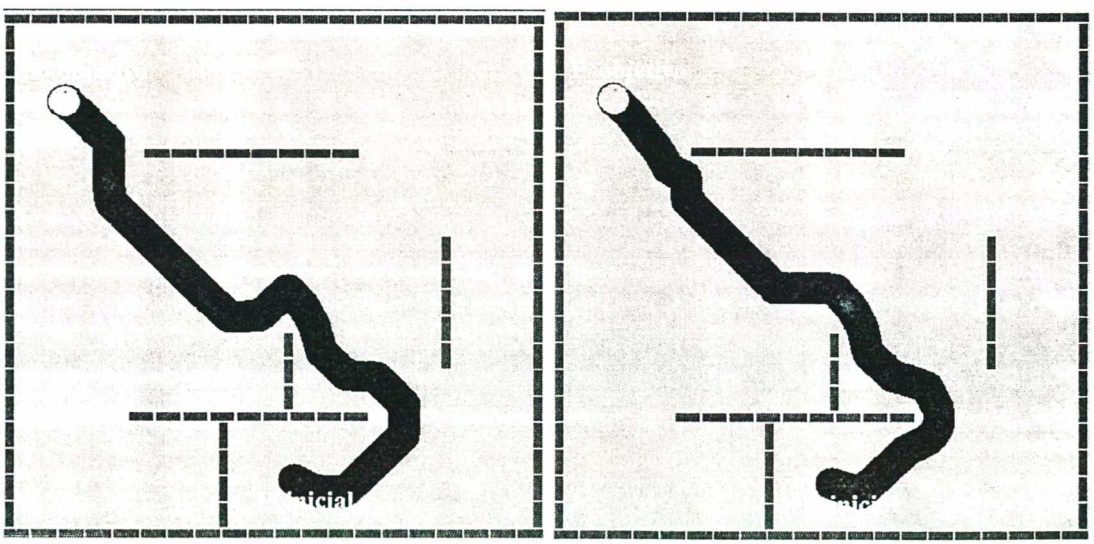
Após ter novamente o mapa completo, o robô desenvolve a trajetória ótima, como mostra



(a) O robô armazenou no mapa todas as informações necessárias.

(b) Na quarta execução, tem-se a trajetória ótima.

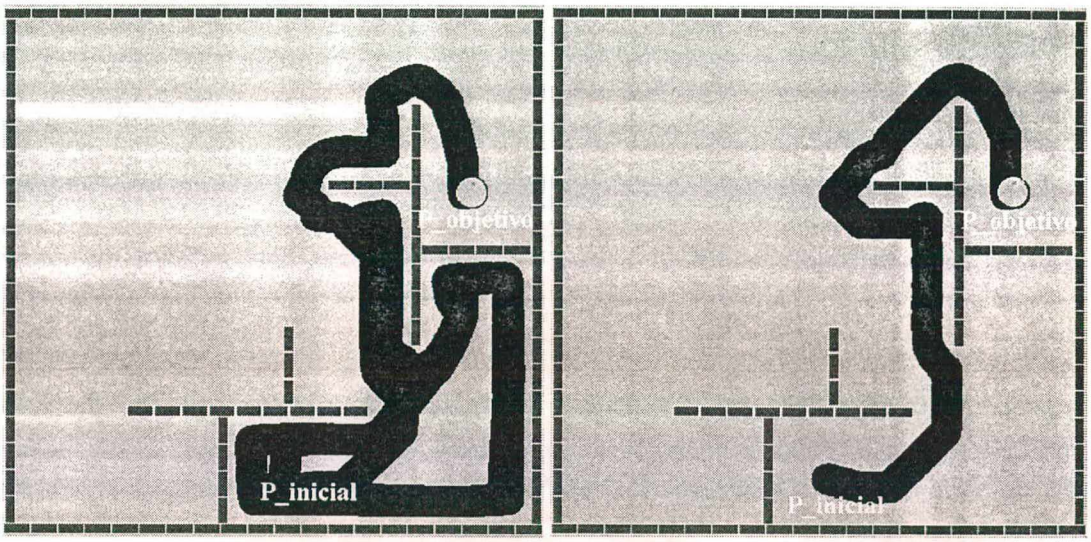
Figura 5.23: Execuções com o ambiente conhecido



(a) O robô executa a trajetória como se os obstáculos ainda estivessem no ambiente. Como os sensores não detectam os obstáculos o mapa é atualizado.

(b) O robô esquece os obstáculos e melhora a trajetória.

Figura 5.24: Esquecimento dos obstáculos



(a) O robô reconhece todo o ambiente necessário e usa estas informações para atingir o objetivo.

(b) Como a manutenção do mapa foi feita, o robô novamente segue a parede.

Figura 5.25: Aprendizagem e manutenção do mapa do ambiente

a figura 5.26 (a). Em (b), o ambiente é alterado e o mapa sofre manutenção.

O robô conclui a manutenção do mapa na execução apresentada a figura 5.27 (a). Na figura 5.27 (b), percebe-se a adaptação do robô às novas condições do ambiente.

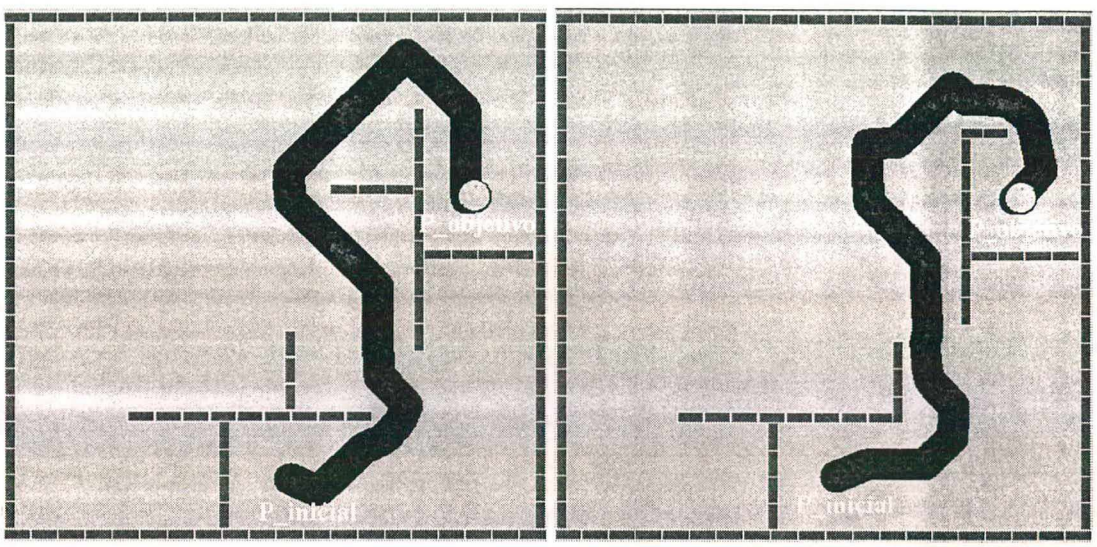
Avaliação dos Resultados

O método híbrido baseado na construção e manutenção do mapa do ambiente em tempo de execução sana o problema da atualização do mapa do método baseado em pré-mapeamento.

O método apresentado aqui não trabalha com probabilidades, isso impede uma manutenção perfeita do mapa. Também é possível perceber que a leitura dos sensores é fator determinante para uma boa execução.

5.4.3 Conclusão sobre os métodos da Abordagem Híbrida

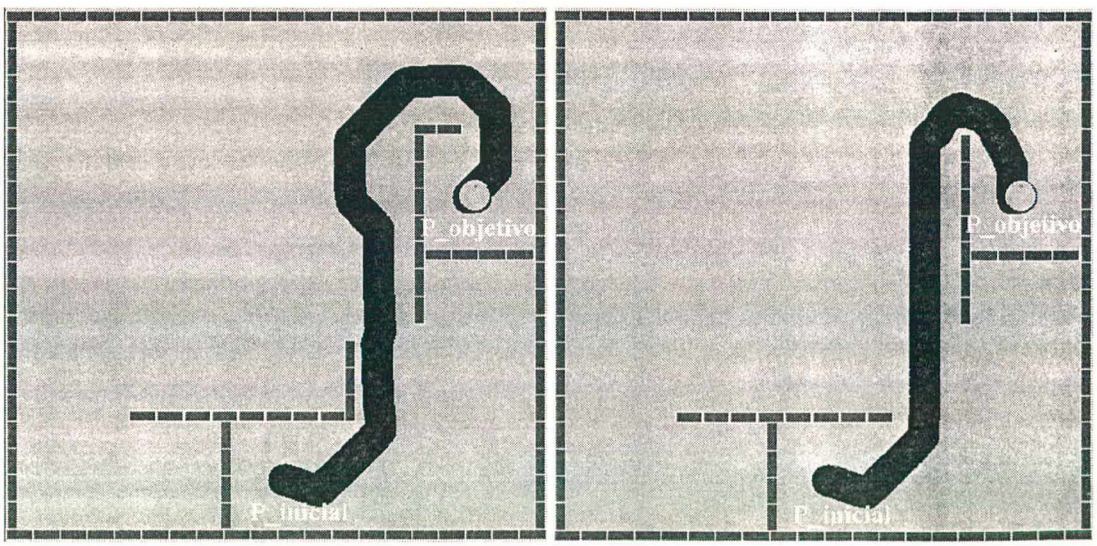
O método baseado em pré-mapeamento e o método de construção e manutenção do mapa em tempo de execução apresentam-se como alternativas à implementação de métodos das abordagens isoladas, uma vez que nenhuma das duas abordagens é completa, mas juntas podem produzir sistemas mais "inteligentes" e robustos.



(a) O robô executa a trajetória ótima.

(b) O ambiente é alterado e o mapa sofre manutenção.

Figura 5.26: Trajetória ótima e alteração do ambiente



(a) O robô ainda não armazenou todas as mudanças.

(b) Os obstáculos foram retirados e o robô está adaptado ao novo ambiente.

Figura 5.27: Incorporação das mudanças do ambiente

A união do planejamento com a reatividade permite trabalhar em ambientes desconhecidos ou parcialmente conhecidos, generalizando as ações de controle e permitindo ao robô desempenhar eficientemente sua tarefa.

A principal questão referente aos métodos híbridos está justamente no compromisso entre a parte reativa e planejada. O método deve ser reativo o suficiente para evitar colisões com obstáculos dinâmicos, mas planejado o bastante para assegurar que o robô não entre em uma situação de colisão da qual unicamente a parte reativa não consiga tirá-lo.

5.5 Conclusão

Neste capítulo foram apresentadas execuções de todos os métodos implementados, analisando os pontos falhos de cada método e como solucioná-los.

Ao final do capítulo, os métodos híbridos são apresentados como alternativa ao uso das abordagens isoladas, uma vez que estas não são completas. Pode-se perceber a grande vantagem de tais métodos quando empregados tanto em ambientes parcialmente conhecidos como em ambientes completamente desconhecidos.

Capítulo 6

Conclusão e Trabalhos Futuros

A navegação de robôs móveis autônomos engloba vários problemas. Um robô móvel autônomo deve ser capaz de navegar pelo ambiente mantendo sua *posição conhecida*. Neste ambiente, ele deve *identificar objetos, desviar de obstáculos, armazenar informações sobre o ambiente e os obstáculos*, com o objetivo de cumprir sua tarefa.

Neste trabalho, enfocaram-se as três abordagens ao problema da navegação: abordagem planejada, reativa e híbrida.

Na abordagem planejada, um mapa do ambiente é fornecido *a priori* e utilizam-se métodos que buscam extrair o espaço livre e sobre este construir a trajetória. Tais métodos, em geral, tratam ambientes bidimensionais com obstáculos poligonais e expandí-los para ambientes dinâmicos conduz ao problema denominado *qualification problem*. Além disso, em geral, ambientes reais são tridimensionais e os obstáculos não são poligonais.

A abordagem reativa surgiu como uma nova forma de abordar o problema. Esta abordagem baseia-se em comportamentos que são disparados diante do recebimento de um estímulo. A principal arquitetura desta abordagem, denominada *arquitetura de subsunção*, trouxe novos horizontes ao campo da robótica móvel, apesar de em pouco tempo apresentar suas limitações e dificuldades de implementação para tarefas mais complexas do que simplesmente a navegação aleatória.

Diante de duas abordagens que tratam o mesmo problema sobre distintos enfoques, novas implementações foram apresentadas, unindo as melhores características destas duas abordagens. As chamadas *abordagens híbridas* apresentam-se como uma forma inovadora de tratar o problema da navegação. Dentro destas abordagens, são duas as principais linhas de pesquisa, uma baseada em pré-mapeamento e outra em construção e manutenção do

mapa em tempo de execução.

Ainda existem várias questões a serem resolvidas e cada vez mais pesquisadores têm se interessado por solucioná-las. Pesquisadores na área de inteligência artificial têm dado grandes contribuições principalmente em aquisição e estruturação de conhecimento. Além disso, técnicas como redes neurais artificiais e lógica fuzzy têm sido cada vez mais utilizadas com sucesso em problemas de fusão de mapas, tratamento de dados sensoriais e reconhecimento de imagens.

Com a definição de cada uma das três abordagens à navegação de robôs móveis foram realizadas sete implementações, sendo três referentes à abordagem planejada, duas referentes à abordagem reativa e duas à abordagem híbrida. As implementações foram feitas nas linguagens C e C++, que são as linguagens aceitas pelo simulador Khepera utilizado para os testes. Optou-se somente por descrever os detalhes mais importantes das implementações, omitindo informações detalhadas referentes aos tipos de dados criados, manipulação de listas e apontadores ou fórmulas para cálculo dos ângulos dos sensores e estimativas das posições dos objetos. Também não foram considerados questões referentes a odometria, pressupondo que a posição e orientação do robô sejam livres de erro.

As implementações da abordagem planejada foram baseadas no método de decomposição em células aproximadas. Para o mapeamento utilizou-se uma matriz de células, sendo que cada célula tem tamanho fixo e pouco maior que o diâmetro do robô. Isso assegura que o robô tenha sempre espaço suficiente para passar entre os obstáculos. Não se utilizou uma estrutura quadtree, visando o desenvolvimento do método de construção do mapa, onde este inicialmente é vazio, pois tal estrutura necessitaria ser constantemente alterada ou, ao escolher uma estrutura fixa, os nós folhas da quadtree representariam as células da matriz implementada, dificultando o acesso direto e aumentando o consumo de memória computacional.

A primeira implementação, denominada *algoritmo da reta*, foi realizada com o objetivo de tratar diferentes ambientes (com paredes horizontais, verticais ou com inclinação). Contudo o tratamento pontual das células mostrou-se ineficiente do ponto de vista prático, impedindo generalizações. Esta implementação foi expandida para múltiplos objetivos, possibilitanto o contorno de algumas situações e a simulação da realização de tarefas.

Na segunda implementação, denominada *árvore de busca 4NF*, a matriz de células tornou-se o espaço de estados aplicado à árvore de busca. A árvore foi gerada expandindo-se os

nós vertical e horizontalmente adjacentes, evitando-se ciclos em um mesmo ramo. Sem utilizar as células diagonais para a geração da trajetória, a expansão dos obstáculos pôde ser ignorada. Como os nós podiam ser gerados repetidas vezes, o método apresentou-se lento e, em muitas situações, a quantidade de memória computacional, insuficiente. Esta implementação mostrou que somente o uso de técnicas de inteligência artificial, sem que o método seja dotado de mecanismos de otimização, apesar de resolver o problema, proporciona baixo desempenho e alto consumo de memória.

A terceira e última implementação da abordagem planejada, *árvore de busca 8NF*, é a complementação da segunda, adicionando "inteligência" à geração da árvore, utilizando os nós diagonais e implementando uma estratégia de corte. O algoritmo para geração da árvore e manipulação das listas de nós é baseado no algoritmo A*. Não utilizou-se nenhuma função heurística, pois o objetivo é encontrar o menor caminho.

Pôde-se constatar que a ordem com que os nós eram gerados afetava o desempenho, devido, justamente, ao corte executado nos nós já expandidos. Esta implementação é simples e sempre garante o menor caminho, otimizando o número de células da trajetória.

Na segunda abordagem foi tratado o problema de navegação em ambientes dinâmicos. Implementou-se uma arquitetura de subsunção utilizando a técnica de redes neurais artificiais em duas implementações.

Na primeira, denominada *navegação direcionada utilizando RNAs*, os comportamentos de desviar obstáculos e procurar objetivo foram implementados por duas redes neurais *feed-forward* com algoritmo de treinamento *back propagation*[1]. Para melhorar o desempenho em ambientes mais complexos utilizou-se o conhecimento *a priori* do ponto objetivo, assim, para fazer o robô detectar a luz emitida pelo ponto objetivo em qualquer ponto do ambiente, utilizou-se a posição do ponto objetivo e a posição e orientação do robô, determinando qual sensor deveria ser ativado e em conseqüência, gerando um sinal como entrada para a rede neural de procura objetivo.

Nesta implementação constatou-se que a escolha dos pares de treinamento influencia o desempenho da rede, podendo torná-la muito específica ou extremamente genérica. No primeiro caso o problema resolvido é muito restrito e no segundo a solução pode não ser encontrada.

Uma forma simples de fazer com que o robô não precise de pares de treinamento é uti-

[1] A rede neural utilizanda foi implementada em cadeira específica durante o período de créditos

lizar o *aprendizado Hebbiano*, como apresentado na segunda implementação. O método é simples e levou o robô a não colidir com os obstáculos. Neste método foram necessárias algumas interações do robô com o ambiente para que o mesmo "aprendesse" a melhor ação mediante a leitura dos seus sensores, desviando dos obstáculos. Para a busca do ponto objetivo manteve-se a rede neural de procura objetivo, em virtude do baixo raio de ação dos sensores de luminosidade.

A união da decomposição em células com árvore de busca 8NF e da técnica de redes neurais com a navegação direcionada possibilitou a implementação de dois métodos híbridos.

O primeiro, baseado em *pré-mapeamento*, permitiu ao robô navegar com o conhecimento global do ambiente, que estava armazenado em um mapa, e reagir aos estímulos locais baseado na leitura dos sensores. O planejamento da trajetória foi feito com a geração da árvore de busca. Ao executar o caminho, as leituras dos sensores iam sendo obtidas e testadas. Caso alguma leitura indicasse um obstáculo próximo, o plano era abandonado e as redes neurais de colisão e luzes assumiam o controle do robô. Ao sair do obstáculo, um novo plano era feito, sem contudo alterar as informações do mapa.

O segundo método híbrido, baseado na *construção e manutenção do mapa do ambiente em tempo de execução*, permitiu que o robô navegasse inicialmente com o conhecimento de sua posição e da posição do ponto objetivo, armazenando as informações lidas pelos sensores em um mapa utilizado para o planejamento da trajetória toda a vez que o robô saísse da situação de colisão. Se um obstáculo não fosse mais detectado, o mapa era alterado para refletir a nova realidade do ambiente.

Não utilizou-se nenhum tipo de método probabilístico para marcar as células. Para cada obstáculo detectado, a célula correspondente era marcada com 1. O obstáculo não sendo mais percebido, a célula era marcada com 2 e posteriormente com 0, indicando novamente a ausência de obstáculo no mapa.

Um problema conhecido na teoria e que pôde ser verificado na implementação é que após conhecidos os obstáculos, o robô executa uma trajetória passando pelas células livres próximas às células assinaladas como obstáculos no mapa, negligenciando o canto dos obstáculos ou mesmo uma parede inteira, caso estes ocupem apenas parte destas células. Isso se deve a baixa abrangência do raio de percepção dos sensores.

Estes dois métodos apesar de bastante simples, evidenciam a grande vantagem no uso da abordagem híbrida. Uma outra questão importante do ponto de vista prático é a por-

centagem de planejamento e reatividade. Nos métodos implementados, foi dada a mesma importância para as partes planejada e reativa.

Fundamentalmente, a escolha do tipo de abordagem e métodos para navegação depende de qual tipo de tarefa se pretende desempenhar. Construir um robô genérico, ainda esbarra em problemas como a falta de precisão das leituras dos sensores, tempo de processamento dos dados dos sensores e na necessidade de formas de representação e estruturação do conhecimento.

Estes problemas levam a questionamentos referentes à capacidade de armazenamento de conhecimento (memória) e cognição. Desenvolver formas para que o robô use conhecimento abstrato sobre o problema vem sendo a principal barreira ao desenvolvimento de robôs "inteligentes".

Como trabalhos futuros, propõem-se o estudo detalhado da abordagem híbrida, principalmente na questão de construção e manutenção de modelos, implementando modelos probabilísticos de ocupação das células e utilizando outros modelos de mapas para modelagem hierárquica do ambiente.

Também propõem-se estudar as capacidades cognitiva, reativa e instintiva dos seres vivos a aplicá-las na criação de sistemas "inteligentes" para robôs móveis.

Além disso, para a implantação da área de pesquisa em robótica móvel no DAS, propõem-se a criação de um grupo interdisciplinar, com sub-grupos responsáveis pelo desenvolvimento de módulos específicos, como reconhecimento e processamento de imagens, processamento sensorial, controle, odometria, planejamento de trajetórias, modelagem do ambiente usando dados sensoriais e controle reativo inteligente, visando a aplicação prática em robôs móveis.

Apêndice A

Ambiente de Simulação: Simulador Khepera

Para a implementação dos métodos de navegação citados nos capítulos 3 e 4, utilizou-se o simulador Khepera[33]. Este simulador é um programa de domínio público, que roda em estações Unix e usa a biblioteca X11 para gerar a interface gráfica.

Os algoritmos de controle desenvolvidos para o simulador podem ser escritos em linguagem C ou C++.

O simulador divide-se em duas partes: o *mundo* e o *robô*, como apresentado na figura A.1. O mundo simula um ambiente real de 1 m^2, ocupando a parte esquerda da tela, e o robô, localizado na parte direita da tela, simula o robô Khepera real que possui $5,5$ cm de diâmetro.

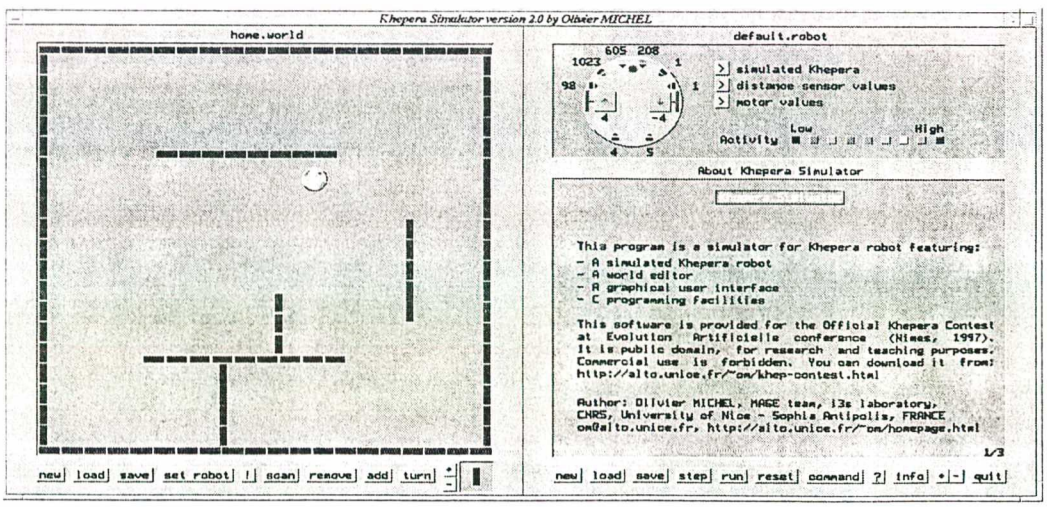


Figura A.1: Simulador Khepera

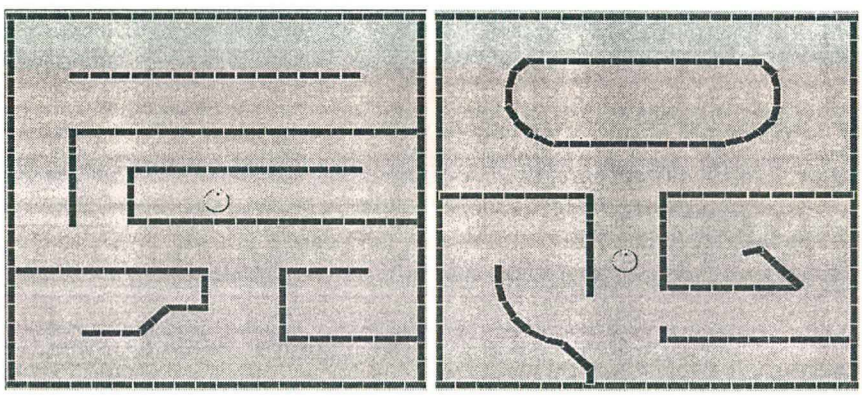
Na parte do mundo pode-se observar o comportamento do robô enquanto este navega pelo ambiente, e na parte referente ao robô tem-se informações sobre os valores lidos pelos sensores e acelerações dos motores.

A.1 Descrição do mundo

O simulador Khepera disponibiliza diversos modelos de mundo localizados no diretório *WORLD*, os quais podem ser carregados através do botão *load*. Além dos mundos disponíveis o Khepera permite que novos mundos sejam criados, através dos botões *new* e *save*.

O mundo é composto por objetos, como tijolos, lâmpadas e rolhas, que podem ser selecionados pelos botões + e - e adicionados ou removidos do mundo pelos botões *add* e *remove*. Os tijolos também podem ser rotacionados sobre seu próprio eixo pressionando-se o botão *turn*. O robô identificará os objetos contidos no ambiente após a execução da função *scan* e sua percepção atual é vista pressionando-se o botão *!*.

Na figura A.2 tem-se dois modelos de mundo criados no simulador Khepera.



(a) Lab1 (b) Lab2

Figura A.2: Mundos criados no simulador

A.2 Descrição do robô

O robô Khepera é composto por 8 sensores infravermelhos (representados por retângulos), que por reflexão, detectam a proximidade dos objetos e 8 sensores de luz (rep-

resentados por triângulos) que medem o nível da luminosidade do ambiente, como mostra a figura A.3.

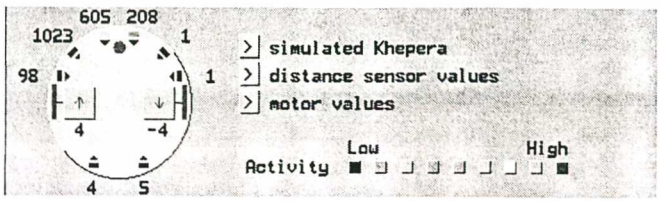


Figura A.3: Controles do robô

Os sensores de reflexão retornam valores entre 0 e 1023 que são representados pelos níveis de cores. Quando nenhum objeto é percebido o valor do sensor é 0, enquanto que 1023 representa que o objeto está muito próximo ao sensor. Os valores intermediários permitem uma estimativa aproximada da distância entre o sensor e o objeto.

Já os sensores de luminosidade retornam valores entre 0 e 500, também representados pelos níveis de cores, sendo 500 completamente escuro e 50 quando o robô está em frente à luz.

Além dos sensores, o robô Khepera possui 2 motores, que podem assumir valores de aceleração entre -10 e +10.

A.2.1 Operando o robô

A posição do robô no mundo pode ser estipulada através do botão *set robot*, que permite colocar o robô em qualquer parte do ambiente, também é possível orientar a direção do robô pressionando-se o botão *command* e digitando na linha de comando *set angle* e o ângulo desejado, por exemplo: *set angle 45*. O robô é acionado através do botão *run*. Para executar passo a passo, utiliza-se o botão *step*.

O simulador Khepera possui, em sua estrutura de diretórios, o diretório *SRC* que contém os fontes do simulador. Para as implementações do usuário, o diretório *USER* é disponibilizado, podendo-se gerar novas funções que são chamadas através das rotinas do arquivo *user.c*. A compilação dos mesmos se dá pela alteração das diretivas do arquivo *Makefile*.

O simulador traz ainda um manual e alguns exemplos de implementação de algoritmos de controle os quais auxiliam na compreensão de seu funcionamento e na geração de novos códigos.

Apêndice B

Redes Neurais Artificiais

Construir máquinas que simulem o comportamento humano é uma idéia que acompanha as civilizações desde a antigüidade.

Muito antes de ter-se o conhecimento detalhado do sistema nervoso, teorias mecanicistas surgiram, utilizando uma abordagem *top-down*, sendo embasadas unicamente na capacidade manifestada do sistema nervoso. Dentre elas, o esquema da máquina analítica de Charles Babbage, do século XVIII e a máquina de Turing, da década de 30 [23].

O modelo mecanicista *bottom-up* surgiu a partir das observações experimentais do potencial de ação e de algumas topologias em que neurônios se interligavam formando redes, onde Warren McCulloch apresenta-se como sendo um dos primeiros a gerar um destes modelos [32].

Estas duas abordagens são complementares, a linha de modelos mecanicistas *top-down* constitui, atualmente a área das técnicas essencialmente algoritmicas de Inteligência Artificial e a linha *bottom-up*, bem mais recente, refere-se a chamada Teoria Conexionista, também denominada Processamento Paralelo Distribuído, ou ainda, a Teoria de Redes Neurais Artificiais [23].

As redes neurais artificiais possuem como vantagens o controle altamente paralelo e distribuído [40]; elevada imunidade à ruídos, não se afetando com a ausência ou a falsidade de informações; capacidade de "aprender" através de exemplos e generalizar o aprendizado, demonstrando bom desempenho em tarefas mal definidas; além de não necessitarem de modelos matemáticos dos domínios de aplicação, apresentando-se desta forma, com a capacidade de simular o raciocínio [4].

B.1 História

Em 1943, Warren McCulloch e Walter Pitts, julgando ser booleana a natureza essencial da inteligência, construíram um modelo extremamente simples, onde as entradas possuíam ganho arbitrário, podendo ser excitatórias (positivas) ou inibitórias (negativas) e a saída do neurônio era determinada pela soma ponderada das entradas com os respectivos ganhos e poderia resultar em um *pulso*, caso ultrapassasse um certo limiar, ou um *não pulso*, caso contrário [22].

O neurônio de McCulloch é essencialmente um discriminador linear com entradas binárias, ou seja, ele atuava como um classificador de padrões para funções booleanas linearmente separáveis. Porém, para funções não linearmente separáveis, como é o caso do *Ou-Exclusivo (XOR)* e de seu complemento, o modelo proposto por McCulloch é ineficaz. Este modelo constituiu a primeira referência na área de pesquisa sobre redes neurais artificiais, influenciando o trabalho de outros pesquisadores como Marvin Minsky na área de Inteligência Artificial e John von Neumann na área de Ciência da Computação [18].

No final da década de 50, Rosenblatt, dando prosseguimento as idéias de McCulloch, criou uma rede com múltiplos neurônios do tipo discriminadores lineares, dispostos em camadas, a qual chamou de Perceptron [22]. O perceptron calcula a soma ponderada das entradas e envia o resultado 1 se esta soma for maior que o limiar, caso contrário o valor enviado é 0.

Rosenblatt pretendia criar um método de aprendizado para encontrar o valor dos ganhos sinápticos e do limiar para a implementação de uma função discriminatória não trivial com um número considerável de variáveis envolvidas, assim, a rede poderia responder a um estímulo x_l, retornando um y_l, ou seja a rede implementaria a função $y_l = f(x_l)$ para todos os l exemplos apresentados à rede.

Em 1949, Hebb propôs um princípio pelo qual o aprendizado em sistemas nervosos complexos poderia ser reduzido a um processo local, em que a intensidade das conexões sinápticas é alterada apenas em função dos erros detectados localmente.

Com o princípio Hebbiano de treinamento, Rosenblatt pôde alterar os parâmetros do discriminador linear. Se a saída y_l for igual a saída desejada y_l^d, os parâmetros não sofrem alteração, caso contrário, os pesos são ajustados repassando a eles os valores atualizados pelo erro gerado na saída. Desta forma, variáveis de entrada do Perceptron não ficariam limitadas à valores booleanos, podendo assumir qualquer valor real.

Contudo, Rosenblatt não conseguiu estender a lei de treinamento para múltiplas camadas e devido as limitações básicas de perceptrons isolados, como a impossibilidade de se implementar o *Ou-Exclusivo*, as pesquisas na área de redes neurais foram paralisadas na década de 70.

Nesta mesma época, na Universidade de Stanford, Widrow desenvolveu um modelo neural linear, chamado de ADALINE - ADAptive LINear Element, cuja generalização multidimencional foi denominada de MADALINE - Múltipla ADALINE. Este modelo era extremamente simples, um aproximador linear de funções, cuja saída é dada pela combinação linear das componentes do vetor de entrada, sendo relevante somente ao contexto acadêmico. Entretando, a principal importância do trabalho de Widrow foi a invenção de um princípio de treinamento conhecido como *Regra Delta*.

Por esta regra, é possível obter o ponto de mínimo através de um processo de interação local, utilizando um exemplo do conjunto de treinamento por vez.

A generalização do método da Regra Delta, desenvolvido por Rummelhart, Hinton e Williams, constitui-se em um dos métodos mais poderosos para o treinamento de redes neurais, chamado de *retropropagação do erro* ou *error backpropagation* [22].

As pesquisas na área de Redes Neurais Artificiais foram reiniciadas na década de 80 com o trabalho de Hopfield, que desenvolveu uma rede recorrente, altamente distribuída, onde cada elemento toma decisões baseadas em sua situação local e o conjunto destas ações forma a ação global. O conhecimento da rede é obtido através de um método de busca denominado Relaxamento Paralelo [40].

B.2 Estrutura

O neurônio é o elemento básico de processamento de uma rede neural. Ele possui várias entradas x_i, com o respectivo peso da conexão w_i e uma única saída y, a qual pode estar conectada a muitos outros neurônios, através de suas sinapses. A figura B.1 apresenta o modelo do neurônio artificial.

A saída do neurônio é dada pelo somatório (ω) das entradas (x_i) multiplicadas pelos respectivos pesos sinapticos (w_i) aplicado a função de transição (Θ). Ou seja:

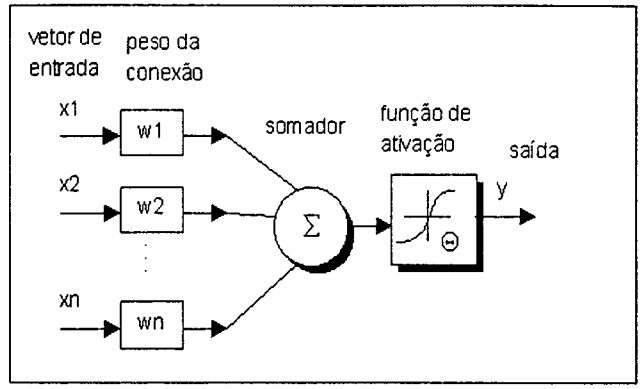


Figura B.1: Neurônio Artificial

$$y = \sum_{i=1}^{n} \omega_i x_i \tag{B.1}$$

A função de ativação de um neurônio pode ser uma função não linear, onde a saída é binária, por exemplo $y \in \{-1, 1\}$ como no caso da função degrau, mostrada na figura B.2 (a) ou uma função linear, onde a saída do neurônio é dada por um valor contínuo em um intervalo, por exemplo $y \in [0, 1]$ como no caso da função sigmóide, mostrada na figura B.2 (b).

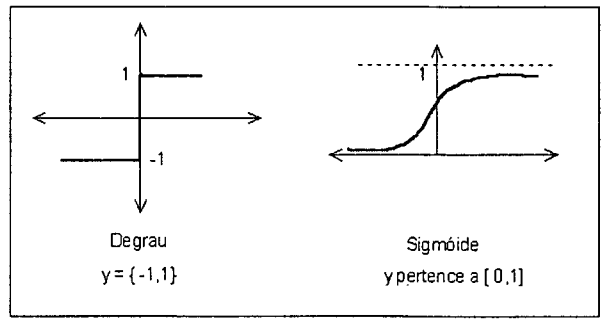


Figura B.2: Funções de ativação

As conexões entre os neurônios formam redes onde cada subgrupo de neurônios constitui uma camada. As redes geralmente possuem uma camada de entrada, uma camada de saída e camadas intermediárias, também chamadas de camadas ocultas. A figura B.3 mostra uma rede neural com 4 camadas.

O conhecimento de uma rede neural está armazenado em suas sinapses, que são

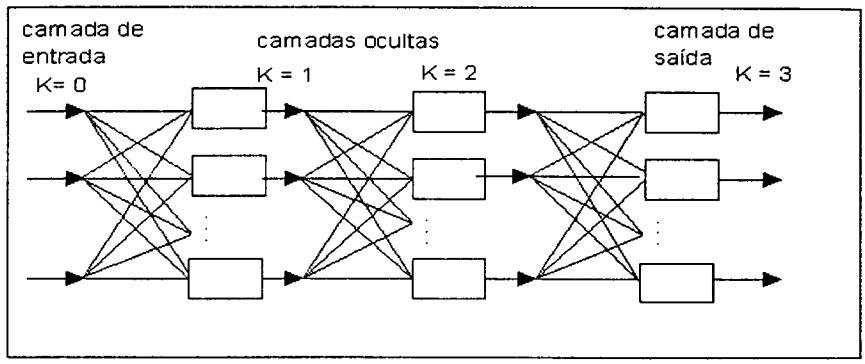


Figura B.3: Rede Neural Multicamadas

as conexões com outros neurônios, desta forma, ao modificar os pesos destas conexões, atualiza-se o conhecimento da rede. A este processo dá-se o nome de treinamento [5]. Então uma rede é dita treinada quando ao se aplicar um determinado vetor x à sua entrada, esta retorne o vetor y desejado, este treinamento é feito através de uma lei de treinamento.

Uma rede pode ser treinada com três objetivos distintos:

- Auto-associação: após o treinamento com o conjunto de vetores, quando submetida a um vetor similar a um dos exemplos, a rede reconstitui o vetor original;
- Hetero-associação : após o treinamento com um conjunto de pares de vetores, quando submetida a um vetor similar ao primeiro elemento de um par, a rede reconstitui o segundo elemento do par;
- Detecção de Regularidade: descobrir as regularidades inerentes aos vetores de treinamento e criar padrões para classificá-los de acordo com estas regularidades.

B.3 Classificação

As redes neurais podem ser classificadas de duas formas: quanto a topologia das conexões entre os neurônios e quanto ao tipo de treinamento utilizado.

Quanto a topologia das conexões, as redes podem ser classificadas em recorrentes ou não recorrentes. Nas redes neurais recorrentes os neurônios são realimentados, ou seja, a saída do ciclo atual é reaplicada no ciclo seguinte como sendo a nova entrada da rede. Este processo é repetido até a estabilização da rede. Já as redes neurais não recorrentes não possuem conexões de realimentação, suas conexões sempre partem da camada de entrada

em direção a camada de saída, este tipo de rede é também chamada de *feed forward*.

Quanto ao tipo de treinamento, elas podem ser classificadas em redes com treinamento supervisionado e redes com treinamento não supervisionado.

O treinamento supervisionado é o mais utilizado e exige a disponibilidade de um conjunto de treinamento formado por pares de vetores de entrada e saída e o algoritmo mais utilizado para isto é denominado *Error Back Propagation*. Este algoritmo consiste basicamente em retropropagar o erro gerado na saída da rede, de forma a estabilizar o peso das conexões.

No treinamento não supervisionado, o conjunto de treinamento consiste somente de vetores de entrada. Algoritmos utilizando o aprendizado de Kohonen ou o aprendizado por reforço são os mais utilizados para o treinamento não supervisionado.

B.4 Algoritmos de Treinamento

B.4.1 Aprendizado Hebbiano

O postulado de aprendizado de Hebb [18, 19] proposto em 1949, é o mais antigo e famoso de todas as regras de aprendizado. Ele diz o seguinte:

> "Quando um axônio de uma célula A está próximo o bastante para excitar uma célula B e repetidamente faz isso, algum processo crescente ou mudança metabólica toma lugar em uma ou outra célula tal que a eficiência de A em B é aumentada."

Particionando este postulado em duas partes, a chamada sinapse Hebbiana, tem-se que:

- Sempre que A e B são estimulados simultaneamente, a conexão é reforçada;
- Se os neurônios A e B são ativados assincronamente a conexão é enfraquecida ou desfeita.

A sinapse Hebbiana pode ser definida, mais especificamente, como uma sinapse que usa um mecanismo de dependência temporal - a modificação depende do tempo exato da ocorrência dos sinais pré e pós-sinápticos; altamente local - a modificação se dá pela avaliação local da atividade em andamento nas unidades pré e pós-sinápticas; e fortemente interativo - a modificação na sinapse depende do sinal de ambos os lados - para aumentar a eficiência sináptica como uma função de correlação (a condição para a modificação é dada pela conjunção dos sinais pré e pós-sinápticos) entre as atividades pré e pós-sinápticas.

Dadas estas características, os parâmetros w_{kj} devem ser atualizados segundo a regra:

$$w_{kj}^{novo} = w_{kj}^{velho} + \Delta w_{kj}$$

com

$$\Delta w_{kj} = \eta y_k x_j$$

e

$$y_k = sgn(\sum_{j=1}^{n+1} w_{kj} x_j)$$

onde η é definida como a taxa de aprendizagem.

Limitações

A grande limitação do aprendizado Hebbiano é o fato de ser um classificador de padrões para funções linearmente separáveis, não podendo ser aplicado para funções não linearmente separáveis, como é o caso da função *Ou-Exclusivo*. Esta limitação restringiu o uso do aprendizado Hebbiano entre os pesquisadores em redes neurais.

B.4.2 Back Propagation

O algoritmo de treinamento *Backpropagation*, também chamado de regra delta generalizada, é baseado no critério dos mínimos quadrados, isto é, o objetivo é minimizar o somatório do quadrado da distância em relação a curva objetivo de cada ponto amostrado.

Este algoritmo utiliza o método do gradiente descendente, o qual altera os elementos de ponderação a cada iteração, visando dar um passo contrário ao gradiente da função para minimizá-la em um número K de passos [7].

É subdividido em duas fases, sendo uma a fase *forward*, na qual são apresentados os vetores de entrada e calculado o vetor de saída. Na outra fase, *backward*, retropropaga-se o erro e o valor desejado para a saída, com a finalidade de atualizar os pesos. O objetivo é modificar os pesos da rede, minimizando uma função de erro global [3].

A seguir apresenta-se a descrição do algoritmo.

1. Inicialize todos os pesos ω_{ij}^{l} com pequenos valores aleatórios (tipicamente valores entre -0.1 e $+0.1$).

2. Inicialize o *thresholding* das unidades. Estes valores nunca mudam.

$$x_0 = 1, h_0^l = 1 \ para \ 1 \leq l < L.$$

3. Escolha o padrão P_k onde $P_k = (p_1, p_2, ..., p_n)$ e aplique na camada de entrada de forma que:

$$x_i = p_i, \ para \ 1 \leq l \leq L.$$

4. Propague o sinal pela rede, usando para a primeira camada:

$$h_j^1 = \frac{1}{1 + e^{-NET_j^1}} \ para \ 1 \leq j \leq H_1$$

e onde

$$NET_j^1 = \sum_{i=0}^{n} \omega_{ij}^1 x_i$$

Para as camadas $1 < l < L$:

$$h_j^1 = \frac{1}{1 + e^{-NET_j^l}} \ para \ 1 \leq j \leq H_l$$

e onde

$$NET_j^l = \sum_{i=0}^{H_{l-1}} \omega_{ij}^l h_i^{l-1}$$

Para a camada de saída:

$$y = \frac{1}{1 + e^{-NET_1^L}}$$

onde:

$$NET_1^L = \sum_{i=0}^{H_{L-1}} \omega_{ij}^L h_i^{L-1}$$

Onde H_l é o número de unidades da camada oculta l.

5. Compute o erro na camada de saída:

$$\delta_1^L = (y' - y)y(1 - y)$$

Comparando a saída atual y com a saída desejada y' para o padrão P_k inicialmente considerado

6. Compute os erros para as outras camadas, retro propagando o erro. Para a camada oculta $L - 1$:

$$\delta_i^{(L-1)} = h_i^{(L-1)}(1 - h_i^{(L-1)})\delta_1^L \omega_{i1}^L$$

Para $1 \leq i \leq H_{L-1}$

7. Atualize os pesos com a seguinte fórmula:

$$\omega_{ij}^l = \omega_{ij}^l + \Delta\omega_{ij}^l$$

Onde $\Delta\omega_{ij}^l$ para a primeira camada oculta é:

$$\Delta\omega_{ij}^1 = \eta\delta_j^1 x_i \ \ para \ \ 0 \leq j \leq H_1$$

Para os pesos entre as duas camadas ocultas:

$$\Delta\omega_{ij}^l = \eta\delta_j^l h_i^{l-1} \ \ para \ \ 0 \leq i \leq H_{l-1}, 1 < l < L$$

Para os pesos entre a última camada e a camada de saída:

$$\Delta\omega_{ij}^L = \eta\delta_1^L h_i^{L-1} \ \ para \ \ 0 \leq i \leq H_{L-1}.$$

8. Volte ao passo 3 e repita para o próximo padrão.

Limitações

Algumas limitações do algoritmo Back Propagation em sua forma original, motivam a busca de alternativas à sua utilização, embora este propicie facilidades para implementação e implique em um baixo custo computacional. Podem ser citadas como suas principais limitações:

- Tempo de Treinamento: Este, resulta principalmente da utilização do método do gradiente descendente, pois o gradiente é um ponto extremamente local em relação à mudança ótima da função, que mesmo para pequenas distâncias pode apontar para direções completamente diferentes. Este comportamento exige um tempo muito maior de treinamento do que uma rota direta de convergência sob uma direção ótima. Tentando minimizar esse fator, opera-se sobre a taxa de aprendizagem e o momentum, contudo estas mudanças não aceleram suficientemente o tempo de treinamento [50].

- Ajuste crítico dos parâmetros α e η: a escolha de uma taxa de aprendizado (α) não é trivial. Quando um valor muito alto é escolhido, ocorre uma grande oscilação em relação à variação dos pesos a cada iteração, contudo, a escolha de um valor subdimensionado implica em uma velocidade menor de convergência do treinamento, e em muitos casos, o estacionamento do aprendizado em um mínimo local. O uso do momentum (η) tenta reduzir a oscilação, porém sua escolha é igualmente não trivial. Outra forma de contornar o problema é utilizar uma taxa decrescente de aprendizagem, de forma a permitir uma grande ocilação no princípio do treinamento e sua gradativa redução ao passo em que a rede aprende os padrões [7].

- Mínimos locais: os algoritmos do tipo gradiente descendente são sensíveis ao problema dos mínimos locais, que pode ou não ser crítico para o treinamento de uma RNA. Em geral, o objetivo do treinamento não é alcançar o mínimo global e sim satisfazer um erro mínimo pré-definido como suficiente para a tarefa desejada [3].

Estes problemas têm impulsionado pesquisas no sentido de criar alternativas, tais como outros algoritmos de treinamento, que busquem outros princípios de minimização do erro em uma rede neural *feed forward* multicamadas que não o método do gradiente descendente. Vários algoritmos propõem mudar ou acrescentar algum aspecto novo ao Back Propagation, visando melhorar sua velocidade de treinamento. Tais algoritmos constituem a família Back Propagation, uma vez que todos são baseados no algoritmo original.

Apêndice C

Estratégias de Busca

C.1 Introdução

As estratégias de busca são utilizadas para encontrar a solução de um problema sobre o seu espaço de estados. O espaço de estados pode ser definido como o conjunto de todos os estados legais gerados, a partir de um estado inicial, através da aplicação de regras válidas. Dentro do espaço de estados é possível encontrar um estado final que representa a solução do problema [40].

Sendo assim, as estratégias de busca têm por finalidade percorrer o espaço de estados retornando o caminho que leva ao estado final ou o próprio estado.

As estratégias de busca dividem-se em 2 categorias [41]:

- Busca Cega ou Exaustiva
- Busca Heurística

C.2 Busca Cega ou Exaustiva

Nesta categoria, uma árvore de busca é construída sobre o espaço de estados, sendo o estado inicial sua raiz. Nenhum conhecimento global sobre o problema é considerado.

C.2.1 Busca em Largura ou Amplitude

A partir do nó raiz da árvore são gerados todos os possíveis estados sucessores, através a aplicação das regras. Em seguida, para cada nó sucessor do nó raiz, são gerados novos

sucessores. Este processo é repetido até que o estado final seja gerado em um dos nós-folha da árvore. A figura C.1 mostra o exemplo da geração de uma árvore com busca em largura.

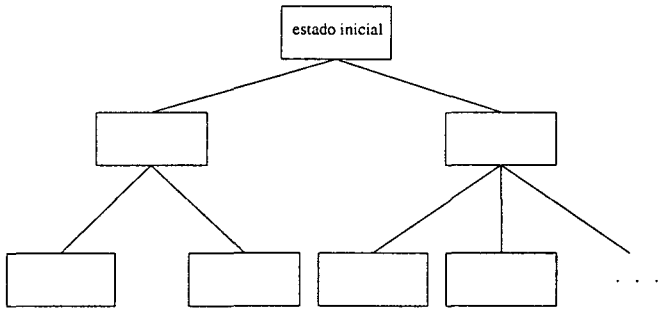


Figura C.1: Árvore de busca em largura

Vantagens da busca em largura:

- A busca em largura nunca explora um beco sem saída ou percorre um caminho infrutífero por longo tempo;
- Se houver uma solução, a busca em largura a encontrará;
- Caso existam várias soluções, sempre a solução mínima será a encontrada.

Desvantagens da busca em largura:

- Se o espaço de estados for amplo, pode ocorrer o fenômeno da explosão combinatória;
- A busca em largura requer grande quantidade de memória, pois todos os nós devem ser armazenados até que a solução seja encontrada;
- É necessário explorar todos os nós do nível n antes de explorar os de nível $n + 1$.

C.2.2 Busca em Profundidade

A busca em profundidade consiste em, a partir do nó inicial, perseguir uma única ramificação da árvore até encontrar a solução ou até ser tomada a decisão de abandonar este caminho. Esta decisão pode ser tomada quando o ramo produz um estado igual a um de seus antecessores ou quando o caminho apresente-se maior do que um limite determinado.

No caso do caminho ser abortado, ocorre o retrocesso e o nó mais recentemente criado é revisitado e a partir dele um novo estado é gerado.

A figura C.2 apresenta um exemplo de uma árvore de com busca em profundidade.

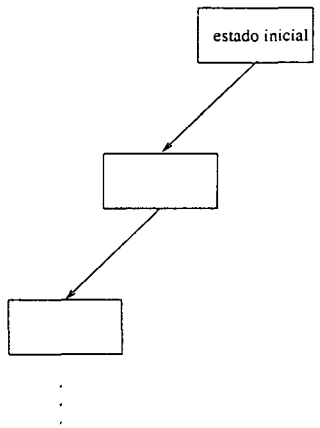


Figura C.2: Árvore de busca em profundidade

Vantagens da busca em profundidade:

- Baixo consumo de memória, uma vez que só os nós do caminho corrente precisam ser armazenados;
- É possível encontrar a solução sem examinar grande parte do espaço de estados, principalmente em problemas que apresentem várias soluções aceitáveis.

Desvantagens da busca em profundidade:

- Pode-se despender muito tempo em um caminho infrutífero, ou em caminhos cíclicos;
- Nem sempre a menor solução será a solução encontrada.

C.2.3 Técnica *Ramificar-e-Podar*

É possível unir as melhores características das estratégias de busca em largura e em profundidade utilizando a técnica de *ramificar-e-podar* [40].

Esta técnica procura a melhor solução, podando caminhos infrutíferos, cíclicos ou que sejam maiores do que o menor encontrado até o momento.

Contudo, para problemas grandes que apresentem um vasto espaço de estados, essa solução pode despender grande tempo.

C.3 Busca Heurística

Para resolver problemas grandes, em um tempo satisfatório, é possível utilizar a estratégia de busca heurística.

A heurística melhora a eficiência do processo de busca, possivelmente sacrificando a completude. O uso da heurística pode negligenciar possíveis soluções, e entre as quais as soluções ótimas, ou seja, a busca heurística garante encontrar boas soluções, mas não ótimas.

Quando introduzida no processo de busca, a heurística aponta para na direção de possíveis soluções, indicando entre os vários caminhos, o mais promissor. Contudo, a escolha de uma boa função heurística nem sempre é trivial.

C.3.1 O algoritmo A*

Neste algoritmo a função heurística é dada por $f' = g + h'$ que calcula o mérito de cada nó gerado, e de um critério de expansão, ou seja, de como gerar sucessores e de um critério de desempate [39].

A função g é a estimativa do custo da distância do nó atual ao inicial. A função h' é a estimativa do custo adicional de ir do nó atual ao nó que representa a solução.

A idéia do algoritmo é expandir um nó a cada passo, até que o nó gerado corresponda ao estado final. A cada passo, escolhe-se o nó mais promissor daqueles gerados mas não expandidos. Então gera-se seus sucessores e aplica-se a cada um deles a função heurística, acrescentando-os à lista de nós abertos após conferir se algum deles já havia sido gerado.

Sendo assim, para o desenvolvimento do algoritmo, torna-se necessária a manipulação de duas listas encadeadas de nós: uma contendo os nós gerados e ainda não expandidos e outra contendo os nós já examinados. O algoritmo é descrito a seguir:

1. Inicie com a lista de nós ABERTOS contendo apenas o nó que representa o estado inicial. Fixe o valor de g desse vértice com 0 e o valor de h' para o valor arbitrado. Inicie a lista de nós FECHADOS como uma lista vazia e um apontador P (que aponta para o seu melhor antecessor) com nil.

2. Até que o nó que representa a solução não seja encontrado, repita o seguinte procedimento: se não houver nó na lista de ABERTOS, retorne fracasso. Caso contrário, pegue um nó na lista de ABERTOS com menor valor de f', chame-o de MELHOR-NÓ e mova-o da lista de ABERTOS para a lista de FECHADOS. Se este nó for o estado final

então pare, pois a solução foi encontrada. Retorne o caminho do MELHOR-NÓ até o nó raiz (este caminho é dado seguindo-se os apontadores P). Caso contrário, gere os SUCESSORES de MELHOR-NÓ. Para cada SUCESSOR faça o seguinte:

(a) Compute g(SUCESSOR) = g(MELHOR-NÓ) + o custo de sair de MELHOR-NÓ e chegar a SUCESSOR.

(b) Verifique se SUCESSOR já está na lista de ABERTOS. Em caso positivo, coloque-o na lista de SUCESSORES do MELHOR-NÓ e chame este nó de ANTIGO. Verifique qual o custo de chegar a ANTIGO via seu pai e o custo de chegar ao SUCESSOR via MELHOR-NÓ (compare os valores de g). Se SUCESSOR for mais barato, então reajuste a ligação com o pai de ANTIGO para apontar para MELHOR-NÓ (P(SUCESSOR) aponta para MELHOR-NÓ). Coloque SUCESSOR na lista de FECHADOS. Guarde o novo valor de f'(SUCESSOR) = g(SUCESSOR) + h'(SUCESSOR).

Referências Bibliográficas

[1] ARKIN, R. Towards the unification of navigational planning and reative control. In *AAAI Spring Symposium on Robot Navigation* (1989).

[2] BARTO, A., SUTTON, R., AND ANDERSON, C. Learning to act using real-time dynamic programming. *Artificial Intelligence 73*, 1 (1995), 81–138.

[3] BATTISTELLA, S. Controle de força e posição de robôs manipuladores utilizando redes neurais artificiais. Master's thesis, UFSC - Universidade Federal de Santa Catarina, 1999.

[4] BITTENCOURT, G. *Inteligência Artificial - Ferramentas e Teorias.* Editora da UFSC (ISBN 85-328-0138-2), 362 p., 2ª edição, Florianópolis, SC, 2001.

[5] BLUM, A. *Neural Networks in C++; an object - oriented framework for building connectionist systems.* John Wiley and Sons, 1992.

[6] BORENSTEIN, J., EVERETT, H., AND FENG, L. Where am i? sensors and methods for mobile robots positioning. University of Michigan, Apr. 1996.

[7] BORGES, F. Um estudo de controladores robóticos utilizando redes neurais. Master's thesis, UFSC - Universidade Federal de Santa Catarina, 1999.

[8] BRAGA, A., AND ARAÚJO, A. Navegação em ambiente fechado e inicialmente desconhecido utilizando aprendizagem por reforço. In *XII Brasilian Automatic Control Conference - XII CBA* (Sept. 1998), pp. 587–592.

[9] BROOKS, R. Solving the find-path problem by good representation of free-space. *IEEE Transactions on Systems, Man., and Cybernetics 13*, 3 (Mar/Abr 1983), 190–197.

[10] BROOKS, R. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation 2*, 1 (Mar. 1986), 14–23.

[11] BROOKS, R. Elephants don't play chess. in Robotics and Autonomous Systems Vol. 6, págs. 3–15, 1990.

[12] BROOKS, R. New approaches to robotics. in Science, Vol. 253, September, págs 1227–1232, 1991.

[13] CARVALHO, J., AND SOUZA, J. Sistema inteligente de navegação para robôs móveis autônomos. In *XII Brazilian Automatic Control Conference - CBA* (Sept. 1998), pp. 575–580.

[14] FABRO, J. Grupos neurais e sistemas fuzzy - aplicação à navegação autônoma. Master's thesis, UNICAMP - Universidade Estadual de Campinas, Feb. 1996.

[15] FABRO, J., AND GOMIDE, F. Controle neural/nebuloso auto-organizável de veículos autônomos. In *II Congresso Brasileiro de Redes Neurais* (Nov. 1995).

[16] FACELI, K., CARVALHO, A., AND E F. MELFI, S. R. Sistemas inteligentes para fusão de sensores. In *Anais do XIX Cong. Nacional da SBC* (Nov. 1999), pp. 163–175.

[17] FIGUEIREDO, M. *Redes Neurais Nebulosas Aplicadas em Problemas de Modelagem e Controle Autônomo.* PhD thesis, UNICAMP - Universidade Estadual de Campinas, Aug. 1997.

[18] HAYKIN, S. *Neural Networks - a compreensive foundation.* Prentice Hall, New Jersey, 1999.

[19] HEBB, D. *The Organization of Behavior.* Wiley, New York, 1949.

[20] JARVIN, R., AND BYRNE, J. Robot navigation: Touching, seeing and knowing. In *Proceedings of 1st. Australian Conf. Artificial Intelligence* (1986).

[21] KHATIB, O. Real time obstacke avoidance for manipulators and mobile robots. *International Journal of Robotics Research*, 5(1) (1986), 90–98.

[22] KOVÁCS, Z. *Redes Neurais Artificiais: Fundamentos e Aplicações: um texto básico.* Edição Acadêmica, São Paulo, 1996.

[23] KOVÁCS, Z. *O Cérebro e Sua Mente: uma introdução à neurociência computacional.* Edição Acadêmica, São Paulo, 1997.

[24] KRÖSE, B., AND EECEN, M. A self-organizing representation of sensor space for mobili robot navigation. In *Proceedings of the IEEE/RSJ/GI International Conference on Intelligent Robots and Systems IROS'94* (1994), pp. 9–14.

[25] KRÖSE, B., AND VAN DAM, J. Neural vehicles. in Neural Systems for Robotic, (Omid Omidvar and P.P. van der Smagt, ed.), Academic Press, págs. 271-296, 1997.

[26] KUMPEL, D., AND SARRADILLA, F. Robot navigation strategies in a partially known environment using a space-time learning graph. In *Proceedings of the Sixth CIM - Europe Annual Conference* (1990), pp. 65–75.

[27] KURZ, A. Constructing maps for mobile robot navigation based on ultrasonic range data. *IEEE Transactions on Systems, Man., and Cybernetics 26*, 2 (Apr. 1996), 233–242.

[28] LATOMBE, J. *Robot Motion Planning*. Kluwer Academic Publishers, Boston, MA, 1991.

[29] LEONARD, J., DURRANT-WHYTE, H., AND COX, I. Dynamic map building for an autonomous mobile robot. *The International Journal of Robotics Research 11*, 4 (Aug. 1990), 286–298.

[30] LOZANO-PÉREZ, T. Spatial planning: A configuration space aproach. *IEEE Transactions on Computers C-32*, 2 (Feb. 1983), 108–120.

[31] MATARIC, M. Integration of representation into goal-driven behavior-based robots. *IEEE Transactions on Robotics and Automation 8*, 3 (June 1992), 304–312.

[32] MCCULLOCH, W., AND PITTS, W. A logical calculus of the ideias imminent in nervous activity. *Bullectin of Mathematical Biophysics* (1943).

[33] MICHEL, O. Khepera simulator package version 2.0: Freeware mobile robot simulator. User Manual, Mar. 1996.

[34] MILLÁN, J. Reinforcement learning of goal-directed obstacles-avoiding reaction strategies in an autonomous mobile robot. *Robotics and Autonomous Systems*, 15 (1995), 275–299.

[35] NILSSON, N. A mobile automaton: An application of artificial intelligence techniques. In *Proceedings of the 1st. International Joint Conference on Artificial Intelligence* (1969), pp. 509–520.

[36] OTTONI, G., AND LAGES, W. Planejamento de trajetória para robôs móveis em ambientes desconhecidos. In *CBA 2000* (2000), pp. 2239–2244.

[37] PAGAC, D., NEBOT, E., AND DURRANT-WHYTE, H. An evidential approach to probabilistic map-building. In *IEEE International Conference on Robotics and Automation* (Apr. 1996), pp. 745–750.

[38] PFEIFER, R. Cheap designs: Exploiting the dynamics of the system-environment interaction - three case studies on navigation. Tech. Rep. IFI - AI - 94.01, Artificial Intelligence Laboratory, University of Zurich, 1994.

[39] RABUSKE, M. *Introdução à Teoria dos Grafos*. Ed. da UFSC, Florianópolis, SC, 1992.

[40] RICH, E., AND KNIGHT, K. *Inteligência Artificial*. Mackron Books, São Paulo, SP, 1993.

[41] RUSSEL, S., AND NORVIG, P. *Artificial Intelligence: a modern approach*. Prentice-Hall, New Jersey, 1995.

[42] SCHAFT, R. Mechatronics and robotics for service applications. *IEEE Robotics & Automation Magazine* (Dec. 1994), 31–35.

[43] SCHILDT, H. *C, Completo e Total*. Mackron, McGraw-Hill, São Paulo, SP, 1990.

[44] SCHWEITZER, G. Motion control for human-oriented machines. In *Proceedings of the International Federation of Automatic Control (IFAC)* (Oct. 1995), pp. 51–58.

[45] SCHWEITZER, G. Mechatronics - basics, objetives, examples. *Journal of Systems and Control Engineering 210* (1996).

[46] SILVA, F., ROISENBERG, M., AND BARRETO, J. Redes neurais hierarquicas para a implementação de comportamentos em agentes autônomos. In *CBA 2000* (2000), pp. 1473–1478.

[47] SOAREZ, R. Curso de robots móviles. Universidad Carlos III de Madrid, Nov. 1999.

[48] THRUN, S. An approach to learning mobile robot navigation. *Robotics and Autonomous Systems*, 15 (1995), 301–319.

[49] THRUN, S., AND BÜCKEN, A. Integrating grid-based and topological maps for mobile robot navigation. In *Procedings of the Thirteenth National Conference on Artificial Intelligence* (Aug. 1996).

[50] TREJO, L., AND SANDOVAL, C. Improving back propagation. *XXI Conferência Latino-Americana de Informática* (1995).

[51] VAN DAM, J., KRÖSE, B., AND GROEN, D. Adaptive sensor models. in IEEE/SICE/RSJ Intr. Conf. on Multisensor Fusion and Integration for Intelligent Systems, Washington D.C, págs. 705-712, 1996.

[52] VAN DAM, J., KRÖSE, B., AND GROEN, D. Neural network applications in sensor fusion for an autonomous mobile robot. in Reasoning with Uncertainty in Robotics, (Dorst, L. and Lambalgen, M. van and Voorbraak, F., ed.), Springer, págs. 263-277, 1996.

[53] VAZ, J. *SNNAP - Sistema Neural de Navegação em Ambientes Pré-Mapeados.* Monografia, UNIOESTE - Universidade Estadual do Oeste do Paraná, Dec. 1998.

[54] VAZ, J., AND FABRO, J. Snnap - sistema neural de navegação em ambientes pré-mapeados. In *Anais do IV Congresso Brasileiro de Redes Neurais* (July 1999), pp. 118–123.

[55] VERSCHURE, P. Minds, brains and robots: Explorations in distributed adaptive control. In *Second Brazilian-International Conference on Cognitive Science* (1996), pp. 14–17.

[56] VERSCHURE, P., KRÖSE, B., AND PFEIFER, R. Distributed adaptative control: The self-organization of strutured behavior. *ras*, 9 (1992), 181–196.

[57] WALLNER, F., AND DILLMANN, R. Real-time map refinement by use of sonar and active stereo-vision. *Robotics and Autonomous Systems*, 16 (1995), 47–56.

[58] ZELINSKY, A. A mobile robot exploration algorithm. *IEEE Transactions on Robotics and Automation 8*, 6 (Dec. 1992), 707–717.