Universidade do Vale do Rio dos Sinos - UNISINOS
Centro de Ciências Exatas e Tecnológicas
Mestrado em Computação Aplicada - PIPCA

Dissertação de Mestrado

Sistema de Controle Híbrido para Robôs Móveis Autônomos

Farlei José Heinen

Prof. Dr. Fernando Santos Osório
Orientador

São Leopoldo, Maio de 2002

Resumo

Neste trabalho foi desenvolvido um sistema de controle robusto para robôs móveis autônomos que é capaz de operar e de se adaptar a diferentes ambientes e condições. Para isso foi proposta uma arquitetura de controle híbrida (COHBRA), integrando as duas principais técnicas de controle robótico (controle deliberativo e controle reativo). Esta arquitetura de controle utiliza uma abordagem de três camadas para integrar uma camada vital (controle reativo), uma camada funcional (seqüenciador) e uma camada deliberativa (controle deliberativo). A comunicação entre as diversas camadas é realizada através de uma área de memória compartilhada, inspirada na abordagem Blackboard. A arquitetura de controle possui um esquema de múltiplas representações internas do ambiente: representação poligonal, representação matricial e representação topológica / semântica.

O sistema de controle desenvolvido tem a capacidade de navegar em um ambiente dinâmico, desviando tanto de obstáculos estáticos como de obstáculos móveis imprevistos. A camada deliberativa utiliza o algoritmo A* para calcular um plano até o objetivo, evitando os obstáculos conhecidos presentes no mapa do ambiente. A camada vital utiliza uma série de comportamentos reativos para guiar o robô até o seu objetivo, e ao mesmo tempo evitar colisões com obstáculos estáticos e móveis. A camada funcional tem como principal função integrar a camada vital e a camada deliberativa, fornecendo parâmetros para os comportamentos da camada vital com base no plano calculado pela camada deliberativa. A camada funcional também possui módulos que monitoram o ambiente e adaptam o mapa no caso de uma alteração do ambiente ser percebida.

Para garantir a robustez da arquitetura de controle, foi integrado um módulo localizador. O sistema de controle implementa a técnica de localização Monte Carlo. O módulo localizador é capaz de localizar o robô no ambiente utilizando as informações sensoriais e um mapa. O localizador é capaz de manter uma estimativa de posição correta quando a localização inicial é conhecida (localização local), é capaz de estimar uma posição global quando não se dispõe de informações sobre a posição inicial do robô móvel (localização global), e também possui a capacidade de detectar uma estimativa de posição incorreta e se relocalizar no ambiente (relocalização). Através da utilização de um filtro de distância, é possível se localizar inclusive em um ambiente dinâmico. Este módulo localizador possui um papel de destaque no sistema de controle, fornecendo uma base sólida para o controle e navegação do robô móvel autônomo.

Para a validação do sistema de controle proposto, foi implementado um simulador de robôs móveis (SimRob3D) que permite a utilização de modelos de ambiente tridimensionais, bem como diversos modelos sensoriais e cinemáticos. Para a visualização tridimensional do ambiente é utilizada a biblioteca OpenGL. O ambiente pode ser alterado em tempo real, e é possível a utilização de objetos móveis que seguem trajetórias predefinidas. É possível utilizar a cinemática Ackerman ou a cinemática diferencial, e sensores tais como: *encoder*, sonar, laser e infravermelho.

Os resultados de diversos experimentos realizados demonstraram a robustez da arquitetura proposta em tarefas de localização local, localização global, relocalização e navegação na presença de obstáculos estáticos e/ou móveis imprevistos.

Palavras-Chave: Robótica Móvel Autônoma, Robótica Inteligente, Inteligência Artificial, Arquitetura de Controle Robótico, Localização e Navegação Robótica.

Abstract

In this work we developed a robust control system for autonomous mobile robots capable of operating and adapting in various environments and conditions. In order to accomplish this objective an hybrid control architecture (COHBRA) was proposed, integrating the two main techniques of robotic control: deliberative control and reactive control. This control architecture uses a three layers approach to integrate a vital layer (reactive control), a functional layer (sequencer) and a deliberative layer (deliberative control). The communication between the three layers uses a shared memory approach, inspired in the Blackboard approach. The control architecture has a structure of multiple internal representations of the environment: polygonal representation, matricial representation and topological/semantic representation.

The control system has the ability to navigate in a dynamic environment, avoiding static obstacles and unexpected mobile obstacles. The deliberative layer uses the A* algorithm to calculate a plan to the goal avoiding the known obstacles in the environment. The vital layer uses a series of reactive behaviors to guide the robot until its goal and at the same time prevents collisions with static and mobile obstacles. The main function of the functional layer is to integrate the vital layer and the deliberative layer, supplying parameters to the behaviors of the vital layer on the basis of the plan calculated by the deliberative layer. The functional layer also possess modules that monitor the environment and adapt the map in the case of an alteration in the environment.

To guarantee the robustness of the control architecture, a localization module was integrated in the system. The control system implements the Monte Carlo localization technique. The localization module is capable of locating the robot in the environment using the sensorial information and a map. The localizer is capable of keeping an estimative of the correct position when the initial localization is known (local localization), it is capable of estimating a global position when it is not making use of information of the initial position of the mobile robot (global localization), and possess also the ability to detect an incorrect position and perform a relocalization in the environment (relocalization). Through the use of a distance filter, it is also possible to locate the robot in a dynamic environment. This localization module has a prominent role in the control system, supplying a solid base to the control and navigation of the autonomous mobile robot.

For the validation of the proposed control system, a mobile robot simulator was implemented (SimRob3D) that allows the use of three-dimensional environment models, as well as diverse sensorial and kinematic models. For the three-dimensional visualization of the environment the OpenGL library is used. The environment can be modified in real time, and it is possible the use of mobile objects that follow predefined trajectories. It is possible also to use the Ackerman kinematics or the diferential kinematics, and sensors such as encoder, sonar, laser and infra-red.

The results of diverse experiments carried through had demonstrated the robustness of the proposed architecture in tasks of local localization, global localization, relocalization and navigation in the presence of unexpected static and mobile obstacles.

Keywords: Autonomous Mobile Robots, Intelligent Robotics, Artificial Intelligence, Robotic Control Architecture, Robotic Localization and Navigation.

Agradecimentos

Agradeço a Unisinos e ao PIPCA (Programa Interdisciplinar de Pós-Graduação em Computação Aplicada) pelos recursos disponibilizados durante a elaboração desta dissertação. Agradeço ao meu orientador, Fernando S. Osório, pela sua dedicação na orientação desta dissertação. Agradeço a todo o corpo docente do PIPCA. E um muito obrigado a todo o corpo discente pela cooperação e coleguismo.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas

A*	– *AStar* (Algoritmo de Procura)
COHBRA	– Controle Híbrido de Robôs Autônomos
DLL	– *Dynamic Load Library*
DXF	– *Data Exchange Format*
FSR	– *Feedback Shift Register*
GNA	– Gerador de Números Aleatórios
GNPA	– Gerador de Números Pseudo Aleatórios
GPS	– *Global Positioning System*
LCG	– *Linear Congruent Generator*
MCL	– *Monte Carlo Localization*
MIDA	– Módulo Indicador de Direção do Alvo
MMAA	– Módulo Monitor de Alterações no Ambiente
MMPT	– Módulo Monitor de Posição Topológica
OpenGL	– *Open Graphics Library*
RMA	– Robô Móvel Autônomo
SMPA	– *Sense, Model, Plan, Act*

Capítulo 1 - Introdução

1.1 - Apresentação

A robótica móvel [Dud00] é uma área de pesquisa que lida com o controle de veículos autônomos ou semiautônomos. O que diferencia a robótica móvel de outras áreas de pesquisa em robótica tais como a robótica de manipuladores, é a sua ênfase nos problemas relacionados com a operação (locomoção) em ambientes complexos de larga escala, que se modificam dinamicamente, compostos tanto de obstáculos estáticos como de obstáculos móveis. Para operar neste tipo de ambiente o robô deve ser capaz de adquirir e utilizar conhecimento sobre o ambiente, estimar uma posição dentro deste ambiente, possuir a habilidade de reconhecer obstáculos, e responder em tempo real as situações que possam ocorrer neste ambiente. Além disso, todas estas funcionalidades devem operar em conjunto. As tarefas de perceber o ambiente, se localizar no ambiente, e se mover pelo ambiente são problemas fundamentais no estudo dos robôs móveis autônomos.

1.1.1 - História da robótica móvel

A robótica móvel vem gradativamente se desenvolvendo a muitos anos. Desde a década de 50 que pesquisadores já se interessavam no desenvolvimento de robôs móveis. Willian Walter construiu diversos robôs móveis em 1950 que eram capazes de executar tarefas tais como desviar de obstáculos e seguir fontes luminosas, utilizando capacitores para controlar o robô [Wal50].

Em Stanford, Nils Nilsson desenvolveu o robô móvel SHAKEY em 1969 [Nil69]. Este robô utilizava dois motores de passo em uma configuração diferencial (cinemática diferencial) para se locomover e era equipado com sensores de distância, câmeras de vídeo e sensores táteis. Ele era conectado a dois computadores por links de rádio e de vídeo. O robô SHAKEY utilizava programas para percepção, modelagem, e atuação no ambiente. As tarefas desempenhadas pelo robô incluíam desviar de obstáculos e a movimentação de blocos coloridos. O robô móvel SHAKEY tinha grandes dificuldades em processar e interpretar as informações sensoriais obtidas do ambiente, e nunca foi capaz de completar uma seqüência completa de ações em um ambiente real.

Novamente em Stanford, Hans Moravec desenvolveu o robô móvel CART no final da década de 70 [Mor90]. A tarefa do robô era desviar de obstáculos utilizando uma câmera de vídeo. O robô móvel CART conseguia desviar dos obstáculos com sucesso, mas era muito lento. No entanto, ele tinha problemas em se localizar no ambiente e necessitava de uma iluminação adequada para perceber os obstáculos adequadamente.

No final da década de 70 foi desenvolvido o robô móvel HILARE no LAAS em Toulouse [Bri79]. Este foi um dos primeiros projetos de robô móvel desenvolvido na Europa. HILARE utilizava câmeras de vídeo, sensores de distância a laser e ultra-som para navegar no ambiente. O planejamento de trajetória era executado utilizando-se uma

representação poligonal do ambiente. Os sensores ultra-sônicos eram utilizados para evitar os obstáculos próximos. O sistema de visão era utilizado para detectar obstáculos distantes e tinha a limitação de ser muito lento.

Todos estes exemplos reforçam a idéia de que controlar robôs móveis é uma tarefa complexa que desafia os pesquisadores da Inteligência Artificial até hoje.

1.2 - Motivação

O desenvolvimento de sistemas de controle para robôs móveis autônomos tem se mostrado um grande desafio para a Inteligência Artificial até os dias atuais. Diferentes abordagens para o projeto de sistema de controle para robôs móveis autônomos vem sendo utilizadas em diversas áreas de pesquisa. Por muitos anos os pesquisadores de Inteligência Artificial tem construído sistemas de controle que apresentam um comportamento inteligente, mas normalmente não no mundo real e somente em ambientes controlados. Alguns pesquisadores desenvolveram certos sistemas de controle para serem utilizados no mundo real, mas geralmente estes sistemas são limitados e não apresentam um comportamento autônomo ou inteligente.

Existem diversas aplicações possíveis para os robôs móveis. No transporte, vigilância, inspeção, limpeza de casas, exploração espacial, auxílio a deficientes físicos, entre outros. No entanto, os robôs móveis autônomos ainda não causaram muito impacto em aplicações domésticas ou industriais, principalmente devido a falta de um sistema de controle robusto, confiável e flexível que permitiria que estes robôs operassem em ambientes dinâmicos, pouco estruturados, e habitados por seres humanos.

O desenvolvimento de um sistema de controle robusto, que possibilite a operação de um robô móvel em um ambiente do mundo real é uma das principais motivações desta dissertação de mestrado.

Durante o ano de 1999 foi desenvolvido o trabalho de conclusão de curso intitulado "**Robótica Autônoma: Integração entre Planificação e Comportamento Reativo**" [Hei00]. Este trabalho formou a base de conhecimento que agora será utilizada para o desenvolvimento desta dissertação de mestrado.

Neste trabalho inicial foi desenvolvido um sistema de controle híbrido para robôs móveis autônomos. Este sistema era capaz de calcular uma trajetória da posição do robô até um determinado objetivo utilizando um mapa do ambiente fornecido "*a priori*". O robô tinha a capacidade de detectar obstáculos inesperados (não previstos no mapa do ambiente), atualizar o mapa adaptando seus conhecimentos sobre o ambiente e recalcular a trajetória até o objetivo.

Durante a elaboração deste trabalho anterior muitos problemas foram encontrados, mas devido ao tempo e recursos limitados alguns destes problemas não puderam ser tratados. A partir desta experiência inicial foi criado um novo sistema de controle híbrido que incorporou novos elementos e módulos que visam tratar as limitações do projeto anterior. Consideramos a localização como um dos principais problemas enfrentados anteriormente.

Neste sistema de controle desenvolvido anteriormente, a posição do robô relativa ao mapa era considerada sempre conhecida e sem erros, nenhum módulo localizador foi implementado. Isso acabou tornando o sistema muito limitado e sensível aos erros de posicionamento do robô, de difícil aplicação em um ambiente real.

Para poder navegar de forma eficiente, um robô precisa ser capaz de determinar sua posição de forma rápida e precisa. Estimativas de posição relativamente precisas podem ser obtidas integrando as informações cinéticas do robô com os seus sensores de velocidade e direção (*encoders*). Mas o acúmulo de erro pode levar a níveis inaceitáveis de discrepância, prejudicando de forma significativa a performance do sistema de controle e navegação.

O solução do problema de posicionamento é a base para que toda a navegação seja validada, sem uma posição com um nível mínimo de precisão as tarefas executadas pelo robô acabam ficando limitadas. Este problema é um dos pontos principais de um projeto de um sistema de controle robusto para robôs móveis autônomos, sendo um dos módulos mais importantes do sistema híbrido proposto nesta dissertação.

1.3 - Justificativa

Por muito tempo os sistemas de controle para robôs móveis autônomos se dividiam em duas abordagens principais: sistemas deliberativos e sistemas reativos.

Os sistemas deliberativos se baseiam fortemente em um modelo do ambiente. Eles conseguem tirar proveito do conhecimento "a priori" sobre o ambiente. Os sistemas deliberativos possuem características essenciais para a elaboração de planos. Os sistemas reativos possibilitam um navegação robusta, bem adaptada as características do mundo real. São modulares e permitem, com a adição de novos módulos, uma fácil melhoria de seus comportamentos. Os sistemas reativos possuem características essenciais para a execução de um plano com uma capacidade de reação mais imediata a eventos imprevistos. A união destas duas técnicas pode produzir um sistema híbrido que possua o melhor de cada um.

Em 1987 Arkin desenvolveu a arquitetura AuRA [Ark87] (Autonomous Robot Architecture). Ela foi uma das primeiras arquiteturas robóticas híbridas que possuía um planejador deliberativo e um controlador reativo em um mesmo sistema, apesar de serem componentes separados. O planejador deliberativo recebia informações do usuário sobre os objetivos do robô, e dados sobre o ambiente, e os utilizava para calcular um plano para atingir os objetivos. Este plano era passado para o controlador reativo. A partir deste ponto o controlador reativo assumia o controle e executava o plano, o planejador deliberativo somente era ativado novamente quando algo errado ocorria durante o plano.

Em 1992 Gat desenvolveu um sistema híbrido chamado Atlantis [Gat92]. A arquitetura possui três camadas, uma camada de controle reativo, uma camada com um seqüenciador, e uma camada deliberativa. Cada uma destas camadas roda independentemente e de forma assíncrona. Nenhuma das camadas controla as outras, e a atividade é dividida entre as três camadas da arquitetura.

Em um ambiente desconhecido em constante mudança, nem um sistema puramente reativo ou um sistema puramente deliberativo podem resolver todos os diferentes problemas que irão surgir. Pode-se notar que a combinação de elementos das arquiteturas reativa e deliberativa para formar uma arquitetura híbrida, pode ser uma maneira efetiva de produzir um sistema de controle para robôs móveis autônomos que possuam comportamentos inteligentes, planejando para atingir objetivos de longo prazo, e reagindo as mudanças de um ambiente dinâmico.

A localização é outro importante componente dos sistemas de controle para robôs móveis autônomos. A localização é um componente chave em vários sistemas de sucesso para o controle de robôs móveis autônomos (ver [Bor96b], [Kor98]). Encontramos referências a localização como sendo "*o problema mais fundamental para fornecer capacidades autônomas para um robô móvel*" [Cox91].

Para navegar de forma confiável em ambientes internos (indoor), um robô móvel necessita saber sua localização (posição e orientação) dentro deste ambiente. A partir das entradas sensoriais, o robô deve ser capaz de inferir sua posição e orientação relativa a um mapa global. A competência da localização não se restringe a ajudar na navegação, mas também para a exploração e para a adaptação do mapa do ambiente pelo robô, bem como para seguir planos. Estimar a localização de um robô baseado em dados sensoriais é um dos problemas fundamentais da robótica móvel. Integrando no sistema de controle robótico um módulo localizador, amplia-se a capacidade de navegação e a robustez do robô móvel autônomo.

1.4 - Objetivos

1.4.1 - Objetivo geral

O principal objetivo deste trabalho é desenvolver um sistema de controle robusto para robôs móveis autônomos que seja capaz de operar e de se adaptar a diferentes ambientes e condições, e para isso será proposta uma arquitetura de controle híbrida.

1.4.2 - Objetivos específicos

- Estudar o estado da arte das técnicas para o controle de robôs móveis autônomos, avaliando suas capacidades e limitações;

- Propor uma arquitetura de controle híbrida que integre as melhores características das técnicas que existem atualmente. Este sistema deverá permitir que o robô seja capaz de:

  - se localizar no ambiente utilizando um mapa e os dados sensoriais. O sistema deverá ser capaz de manter uma estimativa de posição correta a partir de uma posição conhecida (localização local), se localizar globalmente sem uma posição inicial conhecida (localização global), e se recuperar de possíveis erros de localização (relocalização);

  - reagir a situações inesperadas, evitando inclusive colisões com obstáculos móveis;

- se adaptar as mudanças do ambiente. O sistema deverá ser capaz de atualizar o mapa do ambiente adicionando obstáculos estáticos não previstos, e remover obstáculos modelados mas que não estão presentes no ambiente real. Para isso o sistema deverá ter a capacidade de detectar e de diferenciar um obstáculo móvel de um obstáculo estático;

- navegar e executar tarefas em um ambiente complexo tirando proveito de informações topológicas sobre este ambiente;

- Implementar um simulador que será utilizado para validar o sistema proposto, comparando-o com as principais técnicas existentes.

1.5 - Escopo do Trabalho

A robótica móvel autônoma abrange um grande conjunto de áreas de aplicação, em diferentes tipos de ambiente e utilizando diversos tipos e configurações de robôs móveis.

Esta dissertação de mestrado se concentra em algumas destas áreas, restringindo a sua atuação a certos ambientes e a determinado tipo de robô móvel. Os robôs móveis devem ser de pequeno porte e se locomoverem com a utilização de rodas. Os sensores utilizados são os *encoders* e os sensores de distância (laser, infravermelho, ultra-sônico).

O ambiente de atuação deve ser interno (indoor) e plano, tais como escritórios, fábricas, museus, casas, apartamentos, entre outros. Assume-se que existe um conhecimento inicial, pelo menos parcial, sobre a estrutura do ambiente (paredes, portas, e obstáculos estáticos em geral).

Também é necessário que o modelo cinemático dos atuadores e os modelos sensoriais dos sensores utilizados no robô sejam conhecidos, pois estes modelos são utilizados pelo módulo localizador para estimar a posição do robô no ambiente.

1.6 - Organização do trabalho

Esta dissertação encontra-se estruturada da seguinte forma:

Capítulo 2: Neste capítulo são apresentadas definições sobre sistemas de controle e arquiteturas de controle para robôs móveis. Também são apresentados os principais tipos de sistema de controle, bem como o estado da arte dos sistemas de controle desenvolvidos para robôs móveis autônomos.

Capítulo 3: Neste capítulo são apresentados os problemas relacionados com a navegação de robôs móveis. Também são vistas as principais técnicas de navegação de robôs móveis divididas em três categorias: Sensoriais/Reativas, *Roadmaps* e Matriciais.

Capítulo 4: Este capítulo apresenta o estado da arte das técnicas utilizadas para a localização de robôs móveis, abordando em destaque a localização Markov e a localização Monte Carlo.

Capítulo 5: Neste capítulo são apresentados os principais modelos de ambiente, modelos cinemáticos e modelos sensoriais utilizados pela robótica móvel, e os principais simuladores disponíveis para robôs móveis. O simulador SimRob3D, desenvolvido para validar o sistema de controle proposto nesta dissertação é detalhado ao final do capítulo.

Capítulo 6: Este capítulo apresenta a arquitetura de controle para robôs móveis autônomos proposta, bem como o sistema de controle implementado com base nesta arquitetura.

Capítulo 7: Apresenta os resultados obtidos através de experimentos realizados no simulador SimRob3d, visando demonstrar as principais propriedades do sistema: capacidade de localização (local, global e relocalização) e navegação robusta na presença de alterações no ambiente (obstáculos estáticos e móveis).

Capítulo 8: Concluímos a dissertação com uma discussão final dos resultados e apresentamos as perspectivas de trabalhos futuros.

Capítulo 2 - Sistemas de Controle de Robôs Móveis

2.1 - O que é um sistema de controle

Para definir o que é um sistema de controle, precisamos definir quais são os elementos que interagem com o sistema de controle. Ligado diretamente ao sistema de controle está o sistema controlado, que é o sistema que desejamos controlar. O ambiente não é, ou não pode ser, totalmente controlado diretamente pelo sistema de controle, mas interfere de maneira significativa em seu funcionamento Fig. 2.1.

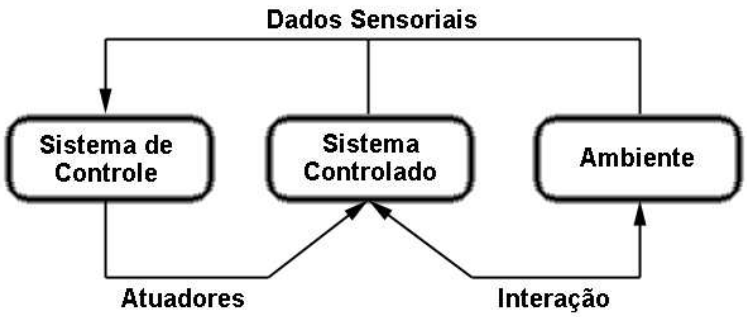

Fig. 2.1 Sistema de Controle de Robôs Móveis

A tarefa do sistema de controle é fazer com que todo o sistema alcance um determinado estado. Alcançar este estado pode envolver ou depender de mudanças que ocorrem no ambiente, no sistema controlado ou devido a interação entre os dois. Logo um sistema de controle é um processo que *pode* utilizar seus sensores para obter informações sobre o sistema controlado e sobre o ambiente. Ele pode utilizar este conhecimento para controlar seus atuadores[1] fazendo com que todo o sistema alcance um determinado estado.

Mas como podemos relacionar esta terminologia com o controle de robôs móveis autônomos? Considere o seguinte exemplo: um robô esta se locomovendo dentro de uma arena tentando evitar a colisão com os muros. Neste caso o sistema controlado é o robô, e os muros são parte do ambiente. A interação entre eles ocorre através da fricção das rodas do robô com o chão da arena, e se o robô acidentalmente colidir com um muro. O sistema de controle obtém informações sobre o próprio robô e sobre o ambiente através dos sensores. O sistema de controle então utiliza estas informações para controlar os atuadores e manter o robô distante dos muros, que seria o estado desejado do sistema.

[1] Por atuadores definimos todos os mecanismos capazes de modificar o estado do sistema controlado em relação ao ambiente. Ex: motores, pistões hidráulicos, etc.

2.2 - Estratégias de Controle

Um sistema de controle pode utilizar sensores inseridos no sistema controlado ou no ambiente, mas isso é opcional. É possível que o sistema de controle utilize sensores somente no sistema controlado, somente no ambiente, ou em lugar nenhum. Existem diferentes maneiras de se obter informações sobre o estado do ambiente e do sistema controlado. A seguir 3 técnicas são brevemente descritas, citadas por [Hal90] e [Smi90]:

* Sistemas de controle **"*Open Loop*"** não utilizam nenhum sensor. Por exemplo, considere um sistema de controle que deve fazer um robô se mover a uma velocidade de 6 quilômetros por hora. Cálculos baseados no modelo físico do sistema podem ser utilizados para construir um modelo que pode prever quanto energia deve ser fornecida aos motores do robô para que ele atinja a velocidade desejada.

* Sistemas de controle **"*Feedforward*"** utilizam sensores somente para perceber o ambiente. Neste tipo de sistema de controle, medições do ambiente são utilizadas para atualizar variáveis no modelo do sistema. Utilizando como base o exemplo anterior, podemos adicionar um sensor que informa a inclinação do terreno. Esta informação pode ser utilizada para recalcular a força necessária para que o robô atinja a velocidade desejada.

As técnicas anteriores podem ser utilizadas somente quando o ambiente é praticamente estático e previsível. Isso não ocorre na maioria dos casos de controle robótico. Em robótica o sistema de controle mais utilizado é o "*Feedback*".

* Um sistema de controle **"*Feedback*"** monitora continuamente a situação dos sensores e ajusta seus atuadores de acordo. Retornando ao exemplo anterior, um velocímetro pode ser utilizado ao invés de um sensor de inclinação. Utilizando o velocímetro a diferença entre a velocidade atual e a velocidade desejada é utilizada para ajustar continuamente a força enviada para os motores do robô.

É importante ressaltar que na prática uma estratégia de controle é quase sempre uma combinação das técnicas apresentadas acima. Neste trabalho todos os sistemas de controle discutidos são do tipo **"*Feedback*"**.

2.3 - Arquiteturas de Controle

Uma definição tradicional de arquitetura de software é descrita por Shaw & Garlan [Sha96]:

> "A arquitetura de um sistema de software define este sistema em termos de componentes computacionais e as interações entre estes componentes."

Mais de acordo com o propósito deste trabalho utilizaremos a definição apresentada por Russel & Norvig [Rus95]:

> "A arquitetura de um robô define como é organizada a tarefa de gerar ações através da percepção."

A arquitetura é uma abstração do sistema de controle, enquanto o sistema de controle é a realização da arquitetura. A seguir são apresentadas as principais arquiteturas utilizadas no projeto de sistemas de controle para robôs autônomos móveis.

2.3.1 - Arquitetura Horizontal

A arquitetura tradicional da Inteligência Artificial que é utilizada no controle de sistemas robóticos é a arquitetura horizontal [Bro91]. Neste tipo de arquitetura as tarefas do sistema de controle são divididas em várias sub-tarefas baseadas em suas funcionalidades. Uma abordagem comum é dividir as tarefas como mostra a Fig. 2.2.

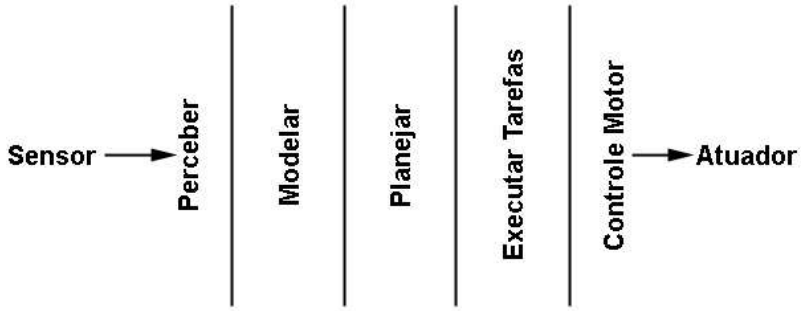

Fig. 2.2 Arquitetura Horizontal (baseada no modelo SMPA)

Sistemas de controle baseados nesta arquitetura resolvem suas tarefas em várias etapas. Primeiro, as entradas sensoriais são utilizadas para modificar a representação interna do ambiente. Segundo, baseado nesta representação um plano a longo prazo é elaborado. Isto resulta em uma série de ações que o robô deve executar para alcançar o seu objetivo. Terceiro, esta série de ações é utilizada para comandar os atuadores do robô. Isto completa o ciclo de controle e o sistema é reiniciado para atingir novos objetivos.

A idéia geral é extrair as informações relevantes do ambiente e construir um modelo o mais completo possível. Com este modelo estático do mundo real os algoritmos podem planejar eficientemente as ações necessárias para alcançar os objetivos. Esta abordagem possui diversos problemas. Manter um modelo na maioria dos casos é difícil devido as limitações e imperfeições dos sensores. O plano elaborado pelo planejador não se aplica de forma dinâmica no mundo real. Outro problema é o fato de que enquanto o sistema de controle esta planejando a próxima série de ações, ele não é capaz de perceber as mudanças no ambiente. Então, se algo diferente ocorrer no ambiente enquanto o sistema esta planejando, estas diferenças não serão consideradas no plano. Isto resulta, no mínimo, em um plano defasado em relação a realidade atual, mas possivelmente em um plano que será extremamente perigoso para o robô e para o ambiente.

2.3.2 - Arquitetura Vertical

A arquitetura "***subsumption***" foi introduzida por Brooks em 1986 [Bro86]. Esta foi a primeira vez que uma arquitetura vertical foi utilizada. Utilizaremos a arquitetura "*subsumption*" como exemplo de arquitetura vertical, e as descrições serão baseadas em uma arquitetura vertical mais genérica.

A nova idéia proposta por Brooks, é que ao invés de as tarefas serem divididas em função da funcionalidade, a divisão deveria ser feita baseando-se em comportamentos que executam tarefas, organizados em camadas (*layers*) como mostra a Fig. 2.3.

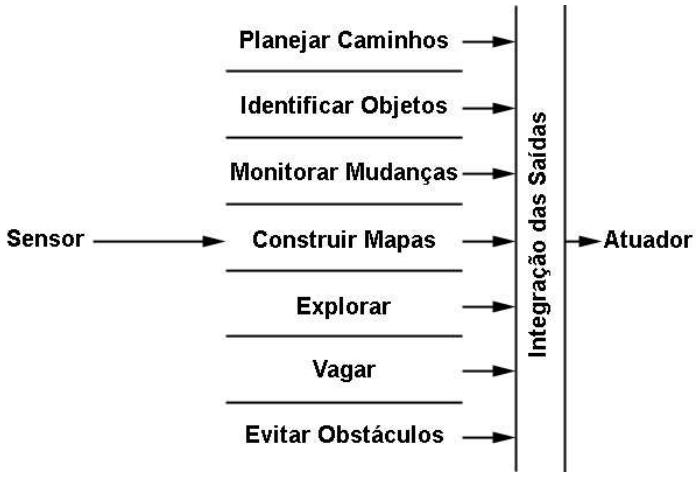

Fig. 2.3 Arquitetura Vertical

Um sistema de controle de robôs móveis baseado em comportamentos é constituído de diversos comportamentos executados em paralelo. Cada comportamento calcula as suas saídas diretamente para os atuadores utilizando as entradas sensoriais. As saídas sugeridas pelo comportamento com a mais alta prioridade são então utilizadas para controlar os atuadores do robô[2]. No texto original de Brooks a integração entre as saídas não é claramente explicitada. Isto porque na abordagem original de Brooks os comportamentos são muito dependentes uns dos outros pois comportamentos de alto nível utilizam saídas dos comportamentos de nível mais baixo. Em muitas aplicações de sucesso que utilizaram uma arquitetura vertical para construir o sistema de controle, a integração das saídas é feita utilizando-se um esquema de prioridades fixas.

A mudança mais significativa que esta nova arquitetura introduziu foi que uma entrada sensorial não precisa mais passar por uma série de camadas de processamento antes de se transformar em uma saída para os atuadores. Em uma abordagem baseada em comportamentos, cada comportamento é auto-contido. Somente as entradas sensoriais relevantes para a tarefa executada por aquele determinado comportamento são utilizadas, não para atualizar uma representação interna, mas para gerar diretamente as saídas para os atuadores. Isto torna a ligação entre os sensores e os atuadores mais forte. A estratégia "*open loop*" utilizada na arquitetura horizontal é abandonada e os

[2] Em geral também é possível combinar as saídas de diversos comportamentos utilizando somatórios, como sugerido por Arkin [Ark90].

sistemas baseados na arquitetura vertical são na maioria construídos utilizando-se uma estratégia de controle do tipo "*feedback*".

Comportamentos de baixo nível, como os que evitam obstáculos, tem um tempo de resposta rápido. Isto reduz o tempo que os sensores do robô ficam cegos, possibilitando uma performance em tempo real. Apesar desta arquitetura ter demonstrado com muito sucesso a execução de comportamentos autônomos, é difícil coordenar os comportamentos para executar tarefas muito complexas, como por exemplo, a navegação global (ver Capítulo 3).

2.3.3 - Arquitetura Híbrida

Arquiteturas híbridas integram um componente horizontal para o planejamento a longo prazo e resolução de problemas com um componente vertical para o controle em tempo real. Um arquitetura híbrida pode ser enquadrada entre a vertical e a horizontal por combinar aspectos de ambas. No entanto, as arquiteturas híbridas também podem ser vistas como o oposto das arquiteturas vertical e horizontal. De acordo com a abordagem híbrida nenhuma estratégia isolada seria adequada para a execução de todas as tarefas relevantes para um sistema de controle robótico.

2.4 - Principais Tipos de Sistemas de Controle

Com base nas arquiteturas descritas anteriormente podemos classificar [Med98] diferentes tipos de sistemas de controle que os pesquisadores da área de robótica autônoma móvel vem desenvolvendo ao longo dos anos.

2.4.1 - Sistemas de Controle Deliberativos

Wooldrige define um sistema deliberativo como aquele que contém, explicitamente representado, um modelo simbólico do mundo, e onde as decisões são tomadas via raciocínio lógico ou pseudológico baseado em reconhecimento de padrões ou manipulação simbólica [Woo94].

Embora essa técnica tenha demonstrado altos níveis de sofisticação, suas limitações logo se tornaram aparentes. Brooks se refere a esta técnica de controle como o modelo **SMPA** (*Sense - Model - Plan - Act*). Os robôs primeiro recebem os dados do ambiente através de seus sensores e depois usam estes dados para construir um modelo o mais completo possível do ambiente. Então, usando este modelo, o robô gera um plano para conseguir atingir seus objetivos e finalmente executa este plano [Bro86]. A maior parte do tempo de processamento é gasto em sentir o ambiente e construir o modelo. Muito pouco tempo é gasto para planejar e agir.

Os diversos exemplos de sistemas de controle deliberativo tem em comum uma mesma estrutura, e uma subdivisão de funcionalidade de fácil identificação e são baseados na arquitetura horizontal. A funcionalidade é atribuída a módulos distintos que se comunicam de uma forma previsível e predeterminada.

Estes sistemas são freqüentemente muito frágeis, incapazes de operar fora de um ambiente controlado. Devido ao fato de que esses sistemas necessitam de um modelo exato do ambiente para decidir exatamente o que fazer, ruídos ou imprecisões dos sensores e do modelo de mundo tornam esses sistemas frágeis.

2.4.2 - Sistemas de Controle Reativos

Em geral, um sistema reativo não possui nenhuma forma de representação interna do modelo do ambiente, e não utiliza um raciocínio simbólico complexo. Ele é baseado no princípio da reatividade, ou seja, na suposição de que comportamentos inteligentes podem ser gerados sem nenhuma representação simbólica explícita e de que a inteligência é uma propriedade que emerge de certos sistemas complexos.

Esta técnica utiliza como base a arquitetura vertical para dividir o controle em uma série de processos que concorrem entre si. Cada um destes processos é conectado com suas próprias entradas sensoriais, com a possibilidade de inibir as entradas ou saídas de outros comportamentos, dependendo das prioridades de cada um.

2.4.3 - Sistemas de Controle Híbridos

Em um controlador híbrido, o objetivo é combinar o melhor dos sistemas de controle deliberativo e reativo. Neste tipo de sistema de controle um módulo é responsável por executar o planejamento a longo prazo, enquanto outro tem a responsabilidade de lidar com situações de reação imediata, tais como evitar obstáculos e se manter em uma estrada. A principal dificuldade deste tipo de sistema de controle é integrar estes dois módulos e resolver os conflitos que surgem desta união. Isto geralmente requer um terceiro módulo que gerencia a comunicação entre o módulo planejador e o módulo reativo. Por este motivo estes sistemas são também chamados de sistemas de 3 camadas [Gat92].

2.4.4 - Sistemas de Controle Baseados em Comportamentos

Os sistemas de controle baseados em comportamentos tem sua inspiração obtida da biologia, e tentam modelar o cérebro dos animais para lidar com problemas complexos, tanto de planejamento quanto de execução de ações. Os sistemas de controle baseados em comportamentos, assim como os sistemas híbridos, também possuem diferentes camadas ou módulos, mas diferente dos sistemas híbridos, estas partes não são muito diferentes entre si. Todas são codificadas como comportamentos, processos que recebem entradas sensoriais e enviam saídas para outros processos. Portanto, se um robô necessita elaborar um plano para alcançar um determinado objetivo, isto é executado em uma rede de comportamentos que conversam entre si e enviam informações, ao invés de utilizar um único módulo planejador como é feito nos sistemas híbridos.

Os sistemas de controle baseados em comportamentos são uma alternativa para os sistemas híbridos, mas atualmente os dois sistemas são igualmente capazes e

populares entre os pesquisadores da área. No entanto, eles não são iguais e cada um tem sido utilizado em diferentes tipos de aplicações da robótica.

2.5 - Estado da Arte em Sistemas de Controle

Apresentaremos agora uma descrição simplificada de alguns sistemas de controle que foram implementados para robôs móveis autônomos. O objetivo é mostrar os sistemas de controle mais conhecidos e apresentar algumas abordagens diferentes. O critério utilizado para selecionar os sistemas de controle foi incluir os mais referenciados em artigos e também alguns dotados de características singulares para ampliar a quantidade de abordagens diferentes.

É importante notar que o escopo e os objetivos destes sistemas de controle diferem muito uns dos outros. Os objetivos de alguns sistemas de controle são mais gerais e outros mais restritos, portanto os sistemas não podem ser comparados como diferentes soluções para um mesmo problema.

2.5.1 - SOAR

O sistema SOAR (*State, Operator and Result*) descrito por Rosenbloom et al. [Ros93] foi implementado na universidade de Carnegie Mellon (CMU) como uma plataforma de testes para as teorias de Inteligência Artificial introduzidas por Newell [New90]. Ele define a inteligência como uma habilidade do sistema em utilizar conhecimentos para alcançar um objetivo. O uso do conhecimento foi a base para o projeto do sistema SOAR.

O sistema SOAR foi construído baseado em uma representação simbólica armazenada em um sistema de produção não modular (SP), e utiliza uma estrutura e um método de acesso/aprendizado simples para todos os tipos de conhecimento.

Como complemento ao sistema de produção, SOAR inclui uma memória de trabalho (MT) que armazena todos os estados e preferências. Percepção e ação influenciam a MT, e os procedimentos de decisão utilizam as informações contidas na MT. Novas produções são adicionadas ao SP por um mecanismo de divisão que opera todo o procedimento de aprendizado do sistema. As informações contidas na MT são manipuladas pelo gerenciador de MT.

SOAR foi originalmente concebido para se tornar um sistema de inteligência genérico, não especificamente projetado para robôs móveis autônomos. Laird et al [Lai89] descreve uma extensão do sistema SOAR, **Robo-SOAR**, para ser utilizado em robótica.

SOAR é um **sistema deliberativo** construído sobre uma representação global, homogênea e uniforme. O aprendizado é executado através de modelos simbólicos de mundo, e uma organização hierárquica temporal foi incorporada.

2.5.2 - Blackboard

Um sistema blackboard [Hay85] é baseado na idéia de se utilizar módulos distribuídos para o processamento de entradas sensoriais, acionamento dos atuadores e planejamento, todos se comunicando através de uma memória compartilhada. Esta abordagem deve tornar o sistema modular e facilitar a construção em paralelo dos módulos.

Um blackboard (BB) é um depósito central de dados do sistema que é utilizado por um determinado número de fontes de conhecimento independentes.

As fontes de conhecimento (FC) são sub-sistemas como por exemplo um sistema de visão, sonares, sistemas para desviar de obstáculos, sistemas planejadores. O princípio chave de um sistema blackboard é a de que toda a comunicação entre os sub-sistemas é administrada pelo blackboard. Todos os sistemas de conhecimento são independentes.

As entradas no blackboard podem ser dados, requisições de uma fonte de conhecimento ou algum resultado parcial. Diferentes tipos de representação de conhecimento são integradas no BB. Algumas implementações de blackboard dividem cada tipo de conhecimento em sub-blackboards [Pan90].

Para a ativação de cada fonte de conhecimento é utilizada uma política de escalonamento gerenciado por uma unidade de controle.

Um exemplo de sistema de controle blackboard foi apresentado por Hayes-Roth [Hay85]. Mais tarde o projeto foi desenvolvido e se transformou no sistema AIS (Adaptative Intelligent Systems) na universidade de Stanford.

Em um sistema blackboard o conhecimento é centralizado, homogêneo mas não uniforme. A estrutura é modular. Ele pode ser classificado como um **sistema deliberativo** apesar de não utilizar uma abordagem tradicional de linha de produção.

2.5.3 - THEO

O sistema THEO desenvolvido por Mitchell et al. [Mit89], é um solucionador geral de problemas.

THEO armazena todos os dados em uma grande base de conhecimento. Os dados são organizados simbolicamente utilizando entidades com uma representação uniforme que possibilita que todo o conhecimento seja acessado e manipulado.

Um controlador reativo foi adicionado ao sistema THEO para possibilitar o controle de robôs móveis autônomos [Mit90]. O controlador reativo opera o robô até que ele não possa mais definir uma seqüência de ações, então THEO assume o controle e cria um novo plano para o robô.

THEO é um sistema híbrido construído em uma estrutura centralizada com o foco principal no aprendizado e no planejamento, utilizando uma representação do conhecimento centralizada de forma homogênea e uniforme.

2.5.4 - NASREM/RCS

A arquitetura NASREM/RCS (NASA Standard Reference Model for Telerobot Control System Architectures / Real-time Control System) foi apresentada por Albus [Alb89] para descrever sistemas inteligentes, tanto naturais como artificiais. Ele define a inteligência como a habilidade de um sistema agir apropriadamente em um ambiente incerto.

O modelo proposto por Albus é hierárquico definido em níveis. Em cada nível são identificados 4 elementos de inteligência:

Geração de Comportamentos: seleciona os objetivos, planeja e executa tarefas. As tarefas são recursivamente decompostas em sub-tarefas.
Modelo de Mundo: contém uma estimativa do estado do ambiente. O modelo de mundo também contém capacidades de simulação e gera expectativas e previsões.
Processamento Sensorial: processa as entradas sensoriais e as previsões do modelo de mundo, atualizando o modelo de mundo de acordo.
Julgamento de Valor: calcula os custos e riscos e avalia tanto os estados observados como os estados previstos pelos planos hipotéticos.

NASREM/RCS foi utilizado em um grande número de veículos tele-operados ou semi-autônomos em missões espaciais e subaquáticas. A arquitetura NASREM/RCS utiliza uma representação do conhecimento homogênea mas não uniforme, e uma estrutura SMPA para cada nível hierárquico.

2.5.5 - AuRA

A arquitetura AuRA (*Autonomous Robot Architecture*) foi introduzida por Arkin em 1987 [Ark87]. Informações mais aprofundadas sobre a estrutura da arquitetura AuRA e das suas raízes na biologia podem ser encontradas em Arkin & Balch [Ark97].

Foi utilizada como abordagem a teoria de "*schemas*" como base para a arquitetura AuRA. Um *schema* produz determinados comportamentos e tanto armazena conhecimento quanto descreve o processo necessário para aplicar esse conhecimento. Ele pode ser definido recursivamente e é independente de implementação. O conceito de *schema* vem sendo utilizado tanto na psicologia cognitiva, neuro-psicologia, como na Inteligência Artificial.

Para o propósito do controle de robôs móveis autônomos, a seguinte definição de *schema* descrita por Arkin [Ark90] é mais adequada:

" São as primitivas que servem como blocos de construção para as atividades motoras e de percepção."

Uma revisão da teoria de "schemas" na área de robótica baseada em comportamentos pode ser encontrada em Beer et al. [Bee93]. Três diferentes tipos de schemas são definidos:

Schemas Motores. A unidade básica do comportamento motor para que um robô codifique respostas aos estímulos do ambiente.
Schemas de Percepção. Incorpora o paradigma de percepção/ação e fornece informação sensorial para os *schemas* motores.
Schemas de Sinais. Monitoram os estados internos do robô e modificam os comportamentos do robô de acordo.

A arquitetura AuRA consiste em 5 subsistemas principais:

Percepção. Obtém e filtra todos os dados sensoriais.
Cartográfico. Conhecimento sobre o ambiente.
Planejamento. Um planejador hierárquico e um controlador utilizando um *schema* motor.
Motor. Interface para com o robô que será controlado.
Homeostático. Monitora as condições internas para o replanejamento dinâmico.

O subsistema cartográfico consiste em um modelo *a priori* do ambiente armazenado em uma memória de longa duração, conhecimentos sobre o ambiente obtidos dinamicamente, e um modelo de incerteza espacial.

O subsistema de planejamento possui um componente deliberativo e um componente reativo. O componente deliberativo é um planejador hierárquico com 3 níveis de hierarquia (Planejador de Missões, Raciocínio Espacial, Sequênciador de Planos). O componente reativo é um controlador baseado em um *schema* motor.

AuRA define uma arquitetura híbrida que incorpora propriedades tanto de sistemas reativos, deliberativos como hierárquicos. Modularidade, robustez e o aprendizado são mencionados como vantagens desta arquitetura e dos sistemas baseados em *schemas* em geral.

2.5.6 - ATLANTIS

A arquitetura Atlantis (*A Three-Layer Architecture for Navigating Through Intricate Situations*) foi apresentada por Gat em 1992 [Gat92] em uma cooperação da NASA com a CalTech. Atlantis incorpora 3 camadas: um deliberador baseado em Lisp, um seqüenciador, e um controlador reativo.

A **camada deliberativa** responde as requisições da camada do seqüenciador para executar tarefas deliberativas. A **camada do seqüenciador** possui uma visão de alto nível dos objetivos do robô. Ela indica para a camada de controle quando iniciar ou parar suas ações e também resolve suas falhas. A **camada de controle** recebe diretamente os dados dos sensores e envia comandos reativos para os atuadores. Os mapeamentos de estímulo/resposta são dados pela camada do seqüenciador.

Atlantis é uma arquitetura híbrida e a sua representação do conhecimento é distribuída e heterogênea. A estrutura é uma abstração representativa com 3 camadas.

2.5.7 - DAMN

A arquitetura DAMN (Distributed Architecture for Mobile Navigation) foi desenvolvida por Rosenblatt & Thorpe em 1995 [Ros95].

A arquitetura consiste de um árbitro central e um certo número de módulos distribuídos. Os módulos são processos independentes que podem representar comportamentos reativos ou deliberativos. O árbitro recebe votos a favor ou contra determinados comandos de cada módulo e decide a seqüência de ações dependendo dos votos recebidos. O árbitro executa uma fusão de comandos para selecionar a ação mais apropriada.

A arquitetura DAMN foi aplicada com sucesso em um sistema que dirigia um carro de forma autônoma em uma estrada. DAMN é uma arquitetura híbrida com uma representação de conhecimento distribuída. A estrutura é distribuída com um árbitro central.

2.5.8 - LAURON

O sistema LAURON foi desenvolvido para o controle de um robô de 6 pernas desenvolvido pela universidade de Karlsruhe. Ele utiliza como componentes básicos redes neurais *feedforward* e redes recorrentes. O sistema foi proposto por Berns et al. [Ber95].

O sistema de controle é organizado de forma hierárquica e consiste de 3 camadas. A primeira camada é composta por um elemento reativo que avalia as entradas sensoriais e seleciona a melhor ação para os atuadores. A segunda camada coordena o movimento dos atuadores, seguindo determinados padrões pré-definidos. A terceira camada coordena cada atuador individualmente, recebendo comandos da segunda camada.

O sistema LAURON é baseado em redes neurais artificiais que incorporam aprendizado por reforço. A estrutura é hierárquica com três camadas de controle.

2.6 - Discussão Final

Neste capítulo foram vistas as principais arquiteturas e os principais sistemas de controle utilizados em robôs móveis autônomos. Cada uma destas arquiteturas tem suas vantagens e desvantagens, nesta dissertação busca-se a integração das melhores características de cada um destes sistemas em uma única arquitetura híbrida. Um das tendências dos sistemas de controle robóticos atuais é a utilização de um esquema de três camadas com visto na arquitetura Atlantis (item 2.5.6). A arquitetura Blackboard demonstra vantagens na forma que as informações são tratadas, e por isso também será utilizada no desenvolvimento da arquitetura de controle proposta neste trabalho.

No próximo capítulo são apresentadas as principais técnicas de navegação para robôs móveis, utilizadas para guiar o robô até um determinado objetivo no ambiente.

Capítulo 3 - Navegação

Quando nos referimos a navegação de robôs móveis autônomos, queremos descrever as técnicas que fornecem os meios para que um robô autônomo se mova de forma segura de um local a outro do ambiente. Encontrar um caminho de uma determinada posição até um destino é um dos problemas fundamentais da robótica móvel autônoma. Um algoritmo de planejamento de trajetória deve garantir um caminho até o destino, ou indicar se o destino é inacessível.

Os métodos tradicionais de planejamento de trajetória [Per79] assumem que o conhecimento sobre o ambiente é completo e perfeito. Utilizando modelos de mundo, um caminho livre de colisões é planejado e uma trajetória é traçada. O sistema de controle motor é então acionado para que o robô siga a trajetória da melhor maneira possível.

Em uma situação mais realística o ambiente pode ser alterado com o passar do tempo. Por exemplo, um robô móvel autônomo que atua em um ambiente onde pessoas circulam deve ser capaz de lidar com a troca de posição de certos objetos (cadeiras, caixas, etc.). O robô deve se basear em seus sensores para perceber o ambiente e planejar de acordo. Uma abordagem comum é utilizar os sensores para criar um mapa do ambiente (mapeamento do ambiente). Uma vez que o ambiente esteja mapeado, as técnicas de planejamento tradicionais podem ser aplicadas. Reconstruir um modelo de mundo baseando-se em informações sensoriais não é uma tarefa simples pois os dados geralmente tem um nível de incerteza muito alto [Leo90].

Os algoritmos de planejamento de trajetória baseados em sensores, utilizam as informações sensoriais locais para controlar diretamente a movimentação do robô. Estas informações possibilitam que o robô opere em ambientes desconhecidos e/ou imprevisíveis. A cada instante de tempo o robô utiliza seus sensores para localizar os obstáculos mais próximos, e então planeja a próxima parte de seu caminho. As estratégias reativas de navegação enfatizam as relações diretas e imediatas entre percepção (entradas) e ação (saídas), e não dependem de um modelo de ambiente. A simplicidade das relações entre as entradas e saídas possibilita que o planejamento seja executado em tempo real. O uso de navegação reativa possibilita ao robô uma operação em ambientes dinâmicos e em constante mudança.

Neste capítulo serão apresentados os principais problemas da navegação, bem como várias técnicas que representam o estado da arte em planejamento de trajetória. Serão apresentadas as principais vantagens, bem como as principais desvantagens de cada método. Ao final todas as técnicas serão examinadas em conjunto utilizando-se uma tabela comparativa.

3.1 - Problemas Relacionados

Diversos problemas vem desafiando os pesquisadores da área de robótica móvel a muitos anos (localização do robô, obstáculos móveis, imprecisão do mapa, imprecisão dos sensores). Diversas técnicas foram desenvolvidas para tentar resolver estes

problemas e tornar a navegação mais robusta e capaz de atuar de forma satisfatória em ambientes complexos.

Na robótica móvel autônoma, o principal problema que dificulta em muito a navegação é o **problema da localização**. Sem saber a posição do robô em relação a sua representação do ambiente (mapa), pouquíssimos sistemas são capazes de controlar o robô de forma adequada. Este problema é complexo, e forma a base de um sistema de controle robótico. Discutiremos mais profundamente este problema no próximo capítulo apresentando diversas abordagens.

Apresentaremos a seguir alguns dos principais problemas, e apresentaremos algumas das soluções encontradas na literatura.

3.1.1 - Obstáculos Móveis

Um dos grandes problemas da navegação de robôs móveis autônomos é a presença de obstáculos que se movem no ambiente. Embora o comportamento de alguns destes obstáculos (um carro andando em uma estrada) possa ser caracterizado e previsto, outros objetos (pessoas andando em um corredor) demonstram um certo nível de movimentação não previsível. O planejamento de trajetória gera uma rota de um local do ambiente para outro, esta rota deve seguir uma trajetória livre de colisões durante o período de tempo em que o robô se movimentar. O planejamento de trajetória é um problema considerado *NP-Hard* em um ambiente dinâmico [Rei85].

Apesar da dificuldade do problema de planejamento de trajetória, vários métodos tem sido propostos para atacar o problema em ambientes dinâmicos. Um destes métodos se baseia em uma formulação espaço/tempo [Fra93]. Dado os movimentos previstos de todos os obstáculos no ambiente, é criada uma **área de alcance** contendo todas as possíveis posições do robô no futuro. Então esta área é utilizada para guiar o robô para a posição desejada. O maior problema com essa abordagem é a quantidade de tempo necessário para construir a área de alcance. Para reduzir o tempo computacional, Suzuki e Arimoto propuseram uma estratégia de divisão e conquista para resolver o problema, decompondo-o em sub-objetivos que podem ser solucionados mais facilmente e então concatenados para se obter o plano final [Suz90].

A abordagem descrita acima é extremamente limitada pelo conhecimento preciso da movimentação e trajetória de todos os obstáculos do ambiente. Um segundo grupo de métodos, normalmente chamados de **detectores de colisão**, são capazes de lidar com obstáculos inesperados e trajetórias desconhecidas. Estes métodos são baseados na navegação reativa e não estão sujeitos as limitações impostas pelo problema de planejamento [Aok96].

Enquanto a maioria dos sistemas de navegação reativa são baseados em projetos *ad-hoc*, eles também podem ser gerados através de aprendizado a partir de observações do ambiente [Beo95]. Os métodos reativos não necessitam de nenhuma informação *a priori* da movimentação dos obstáculos, mas apesar disso eles possuem algumas desvantagens. O aprendizado pode ser uma tarefa de alto consumo de tempo computacional, e deve ser repetido toda a vez que as restrições impostas pelo ambiente são modificadas. Outra desvantagem é o fato destes métodos não utilizarem nenhuma

forma de planejamento, se considerarmos aqueles que são puramente reativos, o que limita sua utilização prática.

Uma possível solução para o problema de navegação com obstáculos móveis seria a integração, em um sistema de navegação híbrido, de técnicas que utilizam planejamento com técnicas reativas. Isso possibilitaria que o sistema utilizasse as informações *a priori* disponíveis para gerar um caminho até o destino, e ao mesmo tempo fosse capaz de evitar obstáculos móveis. O sistema de controle proposto neste trabalho utiliza um sistema de navegação híbrido, e maiores detalhes podem ser vistos no capítulo 6.

3.1.2 - Mapeamento do Ambiente

Outro grande problema da navegação de robôs móveis, é a necessidade do robô atualizar seus conhecimentos sobre o ambiente onde ele atua. Para executar tarefas complexas de forma eficiente em um ambiente fechado (*indoor*), robôs móveis autônomos devem ser capazes de adquirir e/ou manter um modelo do ambiente.

Em geral o robô inicia a navegação somente com um conhecimento parcial, ou até mesmo sem nenhum conhecimento sobre o ambiente. A medida que o robô se move no ambiente novos obstáculos são detectados, e obstáculos conhecidos podem ser removidos ou deslocados. Então o modelo de ambiente inicial se torna incompleto e necessita ser atualizado. Além disso o robô deve ter a capacidade de diferenciar obstáculos móveis de obstáculos estáticos, pois somente os obstáculos considerados fixos no ambiente poderão fazer parte do mapa.

Os seguintes fatores impõem limitações práticas na habilidade do robô em adquirir e utilizar de forma eficiente um modelo de ambiente:

- Sensores: Os sensores nem sempre são capazes de medir diretamente a informação de interesse. Por exemplo, câmeras de vídeo percebem a cor, o brilho e a saturação da luz, mas em um problema de navegação o interesse é descobrir fatos tais como, " existe uma porta em frente ao robô? ".

- Limitações Sensoriais: O intervalo sensorial de muito sensores (ex. sensores ultra-sônicos de distância) é limitado a uma certa região em torno do robô. Para adquirir informações globais, o robô deve explorar de forma ativa seus ambiente.

- Ruído Sensorial: As leituras sensoriais são tipicamente corrompidas por ruído (distorções, leituras incorretas, etc.). Muitas vezes, a freqüência e distribuição deste ruído sobre os sinais sensoriais não é conhecida.

- Desgarrar/Escorregar: O movimento do robô não é preciso. Infelizmente, erros odométricos se acumulam com o tempo. Por exemplo, mesmo o menor dos erros rotacionais pode afetar de forma significativa os erros translacionais quando a posição do robô precisar ser estimada.

- Complexidade e Dinâmica: Os ambientes são complexos e dinâmicos, tornando impossível manter um modelo exato e prever eventos de forma precisa.

- Necessidades de Tempo Real: As necessidade de tempo de processamento das informações geralmente requerem que o modelo interno seja simples e de fácil acesso. Por exemplo, modelos CAD extremamente detalhados de ambientes complexos geralmente são inapropriados se as ações precisam ser geradas em tempo real.

As técnicas de mapeamento podem ser divididas em duas categorias: métricas e topológicas. As técnicas métricas [Cox91] geralmente são implementados utilizando-se matrizes de ocupação [Elf89] que dividem a área a ser mapeada em unidades menores. Cada uma destas unidades possui um atributo que indica a probabilidade daquele espaço estar ou não ocupado. As técnicas topológicas representam o ambiente como uma coleção de pontos de referência interconectados [Kui91]. Tanto a posição do robô como as conexões entre os pontos de referência podem ser modelados como distribuições de probabilidade [Sim95].

3.1.3 - Fusão de Sensores

Um fator que contribui para o aumento da incerteza na navegação de robôs móveis são os conflitos e inconsistências entre múltiplos sensores no robô. Robôs autônomos dependem de numerosos sensores para obter uma visão coerente e consistente do estado atual do ambiente. Este problema introduz incertezas na medida que diferentes sensores reagem de forma diferente ao mesmo estímulo, ou fornecem dados incorretos ou inconsistentes. Estas discrepâncias entre os sensores devem ser tratadas de alguma forma para permitir que o robô tenha uma visão unificada do ambiente. A maioria dos avanços nas técnicas de fusão de sensores surgiram do redescobrimento e adaptação das técnicas da *teoria de estimativas* (estimation theory) [Cro95].

Os primeiros trabalhos caracterizavam o problema de fusão de sensores como uma combinação incremental de informações geométricas. Muitas destas técnicas utilizavam informações *ad-hoc* e não caracterizavam as incertezas do sistema. Um dos primeiros trabalhos que incorporou a incerteza de uma maneira explícita foi realizado por Smith e Cheeseman [Smi87]. Eles propuseram a utilização da teoria de estimativa Bayesiana e derivaram uma função de combinação que é uma forma equivalente de filtro de Kalman [Kal60]. Isto causou uma rápida troca de paradigma para as teorias de estimativa probabilistas, mais relacionados com a estimativa Bayesiana, *maximum likelihood estimation* e os métodos de mínimos quadrados [Cro95].

Apesar destes métodos terem tido sucesso em combinar dados sensoriais para determinar um subconjunto comum de estados utilizando sensores similares (ex: a localização do robô derivada de um GPS, odômetria e leituras de um giroscópio), ainda não parece existir uma extensão óbvia para a fusão de sensores multi-modal. A vantagem das abordagens multi-modais é que elas podem combinar dados de sensores não similares (ex: um sensor de ruído e uma câmera de vídeo) para imitar o raciocínio humano (ex: ouvir uma carro indica uma maior probabilidade de se estar vendo um carro).

3.2 - Principais Técnicas de Navegação

A seguir são apresentadas as 3 principais abordagens utilizadas pelas técnicas de navegação de robôs móveis autônomos (Sensoriais/Reativas, *Roadmaps* e Matriciais).

3.2.1 - Abordagem Sensorial / Reativa

A navegação baseada em sensores incorpora as informações sensoriais, refletindo o estado atual do ambiente, diretamente no processo de planejamento do robô. Isto é justamente o oposto das técnicas de navegação clássicas, onde o conhecimento total sobre a geometria do ambiente precisa ser estabelecido antes de se iniciar o processo de planejamento. As principais vantagens da navegação baseada em sensores em relação aos métodos clássicos são:

- em muitos casos o robô não possui conhecimento antecipado sobre o ambiente onde ele ira atuar;
- o robô dispõe de um conhecimento parcial sobre o ambiente devido a limitação de memória;
- os modelos de ambiente geralmente não são precisos;
- no ambiente podem ocorrer situações inesperadas ou mudanças rápidas.

Existem diversas técnicas baseadas em sensores que podemos citar:

3.2.1.1 - DistBug

O algoritmo Distbug, proposto por Kamon & Rivlin [Kam97], é um exemplo de como regras simples podem gerar um comportamento complexo utilizando as técnicas de navegação baseadas em sensores. O algoritmo Distbug tenta garantir que o destino seja atingido se possível, ou indicar se o destino for inacessível. O algoritmo é reativo no sentido de que ele depende dos dados sensoriais para tomar decisões locais, e não utiliza nenhuma forma de representação interna do ambiente.

O algoritmo consiste em dois modos de locomoção: se mover diretamente em direção ao alvo entre os obstáculos e seguir as bordas dos obstáculos. As bordas do obstáculos são contornadas até que uma certa condição indique que ele deva novamente ir em direção ao alvo. A direção em que o modo de seguir as bordas decide se locomover irá influenciar o tamanho do caminho que será percorrido. O algoritmo decide em que direção ele se movimentará baseado em informações locais. Os passos da navegação são os seguintes:

1 - Ir direto em direção ao alvo, até que uma destas condições ocorra:
- O alvo foi encontrado. Pare.
- Um obstáculo foi encontrado. Vá para o Passo 2.

2 - Determinar uma direção que será usada para seguir a borda do obstáculo. Seguir a borda do obstáculo até que uma destas condições ocorra:
- O alvo foi encontrado. Pare.

- A distância livre em direção ao alvo garante que o próximo ponto de colisão estará mais perto do alvo do que o ponto de colisão anterior. Vá para o passo1.

- O robô completou um volta completa em torno do obstáculo. Indicar que o alvo esta inacessível.

Esta técnica é simples, mas em um ambiente controlado é capaz de desviar de obstáculos e evitar situações de armadilha como os mínimos locais. Mas ela se limita a ambiente estáticos, que não possuam obstáculos móveis.

3.2.1.2 - Navegação Baseada em Comportamentos

A navegação baseada em comportamentos utiliza uma ligação forte entre percepção e ação para se locomover de forma robusta na presença de incerteza no ambiente. O robô executa *comportamentos* como "abrir a porta" ou "entrar na sala". Para organizar estes comportamentos, geralmente se utilizam autômatos de estados finitos nos quais os estados correspondem aos comportamentos e os arcos correspondem aos gatilhos que indicam um novo comportamento. O estado atual indica o comportamento do robô. Quando o robô recebe as informação sensoriais, e estas informações correspondem a um determinado gatilho que o estado atual possui, o comportamento que o gatilho indica se torna o novo estado atual.

Já que os autômatos de estados finitos são baseados em comportamentos e gatilhos, o robô não necessita de um modelo do ambiente ou informações completas sobre o estado atual do mundo.

Um exemplo de sistema que utiliza navegação baseada em comportamentos pode ser encontrado em Endo *et al* [End00]. O sistema proposto, Missionlab, possui um grande número de comportamentos e gatilhos pré-programados que o usuário pode utilizar para construir um autômato, que então pode ser executado em robôs ou em simulações.

Outro exemplo de sistema baseado em comportamentos seqüenciados por um autômato finito é o Sistema de Estacionamento de Veículos Autônomos SEVA [Hei01], utilizado para a automatização do processo de estacionamentos de carros.

3.2.1.3 - Mapas Neurais

Os mapas neurais foram propostos por Glasius et al [Gla95] como um método alternativo para o planejamento de trajetória de robôs móveis.

Um mapa neural é uma "representação neural localizada dos sinais do mundo exterior" [Ama89]. Sinais provenientes de um espaço **X** são mapeados através de uma função **f(X)** em um campo neural **F**. O campo neural **F** é uma rede neural recorrente, onde as dinâmicas definem as interações inibitórias e exitatórias entre as unidades. Amplificação e preservação de vizinhança (neighborhood preservation) são duas propriedades dos mapas neurais. Além disso, a propriedade de "self organization" é uma forma de aprendizado não supervisionado que adapta o mapa dinamicamente. A

combinação de mecanismos computacionais e de representação faz dos mapas neurais uma ferramenta poderosa para resolver problemas de interesse prático.

Glasius et al [Gla95] demonstrou que os mapas neurais podem ser utilizados de forma efetiva para o planejamento de trajetória. O mapeamento assemelha-se ao espaço de configuração n-dimensional **C** de um robô e o sinal **X** é a informação atual sobre o ambiente (áreas livres, obstáculos, destino). As unidades da rede são distribuídas sobre **C** implementando uma representação discreta ordenada topologicamente de **C**. Assim, cada unidade **i** corresponde a uma única configuração representativa (**ci**) do robô e cada configuração possível de **C** é representada pelo **ci** mais próximo. O peso entre as unidades **i** e **j** refletem o custo de mover o robô de uma configuração **ci** para **cj**. Quanto maior o custo, menor é o peso. A dinâmica difusa do mapa é devido as interações excitatórias locais, que resultam em uma superfície de ativação estável sobre **C**, gerando um mapa de navegação completo até o destino. Um procedimento simples de subida passo-à-passo (*steepest ascent*) no equilíbrio da superfície de ativação de qualquer configuração inicial irá retornar um caminho livre de obstáculos até o destino mais próximo.

Para um robô móvel, **C** é um plano bidimensional. Em geral, as unidades do mapa são distribuídas de forma homogênea sobre **C**. A topologia e o número de conexões pode variar. As opções comuns incluem topologias retangulares e hexagonais com conexões de (e para) uma unidade **i** se estendendo para os seus vizinhos. O custo utilizado para determinar o peso das conexões é a distância euclidiana entre as diferentes configurações.

3.2.2 - Abordagem Roadmap

A abordagem roadmap para a geração de caminhos consiste em reduzir as informações ambientais em um grafo representando os possíveis caminhos existentes. Uma vez que um roadmap foi construído, um caminho pode ser calculado conectando a posição atual e o objetivo ao grafo e encontrando uma trajetória dentro deste grafo.

Em geral os métodos roadmap são rápidos e simples de se implementar, mas eles não fornecem uma boa forma de representação das informações do ambiente.

A seguir serão apresentados alguns métodos que utilizam a abordagem roadmap, mas antes é necessário apresentar algumas definições:

Espaço de Configuração: Este conceito é utilizado quando o planejamento de trajetória é executado com objetos estacionários e conhecidos. O espaço de configuração é uma transformação do espaço físico onde o robô tem um tamanho bem definido, para um espaço onde o robô é tratado como um ponto. Em outras palavras, o espaço de configuração é obtido encolhendo o robô até o tamanho de um ponto, e ao mesmo tempo expandindo os obstáculos pelo tamanho do robô.

Espaço Livre: O espaço livre de um espaço de configuração simplesmente consiste das áreas que não são ocupadas por obstáculos.

Caminho Livre: O caminho livre entre uma posição inicial e um objetivo é o caminho que pertence inteiramente ao espaço livre e não entra em contato com nenhum obstáculo.

3.2.2.1 - Grafos de Visibilidade

Um grafo de visibilidade é obtido gerando-se segmentos de reta entre os pares de vértices dos obstáculos. Todo o segmento de reta que estiver inteiramente na região do espaço livre é adicionada ao grafo (Fig. 3.1). Para executar o planejamento de trajetória, a posição atual e o objetivo são tratados como vértices, isso gera um grafo de conectividade onde utiliza-se algoritmos de procura para se encontrar um caminho livre. Os métodos mais simples utilizados para construir um grafo de visibilidade tem complexidade $O(n^3)$.

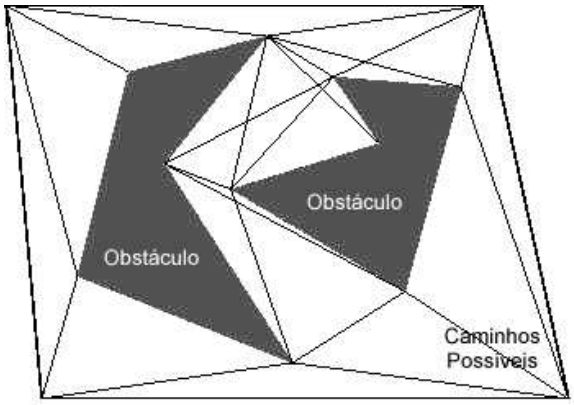

Fig. 3.1 Grafo de visibilidade

O caminho mais curto que for encontrado no grafo de visibilidade é o caminho ótimo para o problema especificado. Mas os caminhos encontrados no grafo "tocam" os obstáculos. Isso não é aceitável em um problema de navegação robótica, por esse motivo os obstáculos devem ser modificados, criando-se um espaço de configuração.

Para a utilizar um método de navegação com grafo de visibilidade é necessário quer o mapa seja completo e bem definido. Não é possível navegar em um ambiente que possua obstáculos móveis e a localização do robô móvel deve ser conhecida com precisão durante toda a navegação. Um trabalho que utiliza o grafo de visibilidade para o planejamento de trajetória de um robô móvel autônomo pode ser visto em [Hei00].

3.2.2.2 - Diagramas de Voronoi

Os diagramas de Voronoi já vinham sendo discutidos desde 1850 por Peter Lejeune-Dirichlet. Mas foi somente em 1908 que eles apareceram em um artigo escrito por Voronoi.

Um diagrama de Voronoi é um estrutura geométrica que representa informações de proximidade sobre uma série de pontos ou objetos. Dada uma série de *sites* ou objetos, o plano é particionado atribuindo para cada ponto o seu *site* mais próximo. Os pontos que não possuem um único *site* mais próximo formam o diagrama de Voronoi. Isto é, os pontos no diagrama de Voronoi são eqüidistantes de dois ou mais *sites*.

Os diagramas de Voronoi podem ser generalizados de diferentes maneiras, dependendo da aplicação prática desejada. Os diagramas de Voronoi podem ser construídos também para polígonos, não somente para *sites* pontuais. Este tipo de aplicação é utilizada no planejamento de trajetória [Row79].

No caso do planejamento de trajetória, a região bidimensional em que o robô se locomove geralmente contém obstáculos, cada um destes obstáculos pode ser representado por polígonos côncavos ou convexos. Para encontrar o diagrama generalizado de Voronoi para esta coleção de polígonos, podemos calcular o diagrama através de uma aproximação, convertendo os obstáculos em uma série de pontos. Primeiramente as faces dos polígonos são subdivididas em uma série de pontos. O próximo passo é calcular o diagrama de Voronoi para esta coleção de pontos. Após o diagrama de Voronoi ser calculado, os segmentos do diagrama que intersectam algum obstáculo são eliminados.

Um vez que o diagrama esteja calculado para todo o conjunto de obstáculos, é necessário adicionar a posição do robô e o destino ao conjunto de pontos. Desta forma podemos então utilizar uma algoritmo de procura para encontrar um caminho do ponto inicial até o destino, que é um subconjunto do diagrama de Voronoi.

Este método gera uma rota que na sua maior parte permanece eqüidistante dos obstáculos (Fig. 3.2), criando um caminho seguro para o robô se locomover, apesar de não ser o caminho mais curto.

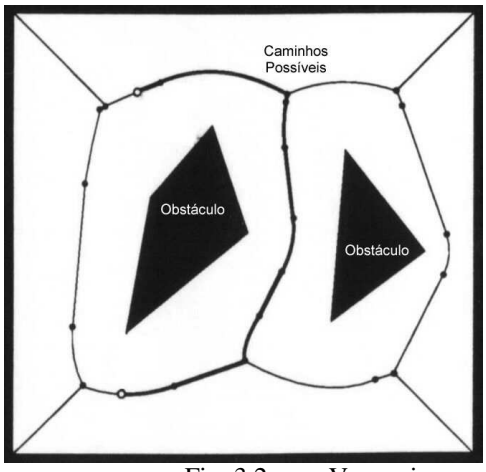

Fig. 3.2 Voronoi

Este método possui os mesmos inconvenientes dos grafos de visibilidade. O mapa deve ser preciso e o ambiente não pode conter obstáculos móveis. A posição do robô precisa ser conhecida com precisão para que os caminhos possam ser seguidos.

3.2.2.3 - Decomposição Celular

A idéia básica por traz do método de decomposição celular é que um caminho entre a posição inicial e o objetivo pode ser determinado subdividindo-se o espaço livre em regiões menores chamadas células. Após esta decomposição, um grafo de conectividade é construído de acordo com as adjacências entre as células, onde o nodos representam as células no espaço livre, e os arcos entre os nodos indicam as adjacências. Através do grafo de conectividade, um caminho contínuo, ou canal, pode ser

determinado simplesmente seguindo as células livres adjacentes. Estes passos são ilustrados abaixo utilizando-se os métodos de decomposição celular exata e aproximada.

*** Decomposição Celular Exata**

O primeiro passo neste tipo de decomposição celular é subdividir o espaço livre, que é limitado tanto externamente como internamente por polígonos, em células triangulares e trapezoidais traçando segmentos de reta paralelos a cada vértice interno até o polígono limitador externo. Então cada célula é numerada e representada como um nodo no grafo de conectividade. Nodos que são adjacentes no espaço de configuração são ligados no grafo (Fig. 3.3). Um caminho neste grafo corresponde a um canal no espaço livre. Este canal é então transformado em um caminho livre conectando a posição inicial e o objetivo através dos pontos médios das interseções das células adjacentes do canal.

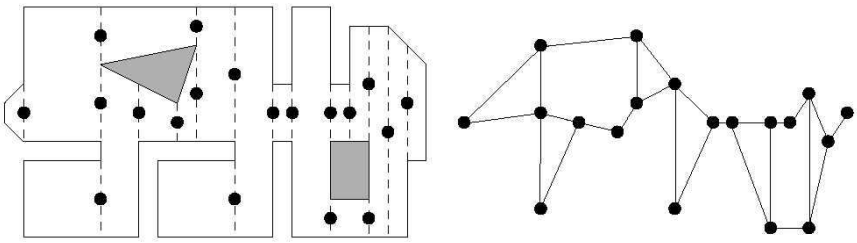

Fig. 3.3 Decomposição Celular Exata

* **Decomposição Celular Aproximada**

Esta técnica de decomposição celular é diferente pois utiliza um método recursivo para subdividir continuamente as células até que um determinado cenário ocorra:

- Cada célula esta inserida completamente no espaço livre;
- Uma resolução limite é atingida.

Um vez que uma célula satisfaz estes critérios, ela não é mais decomposta. Este método também é chamado de decomposição "quadtree" pois a célula é dividida em quatro células menores do mesmo formato em cada decomposição. Após a subdivisão, um caminho livre pode ser encontrado seguindo as células adjacentes.

Assim como as demais técnicas do tipo *roadmap*, a decomposição celular depende de um mapa preciso do ambiente sem obstáculos móveis, e necessita que a posição do robô em relação ao mapa seja conhecida.

3.2.3 - Abordagem utilizando Matrizes

A abordagem utilizando matrizes, tais como propostas por Elfes [Elf87], Borenstein [Bor91] e outros, representam o ambiente utilizando uma matriz de tamanho fixo. Cada célula da matriz pode conter diferentes atributos que auxiliam na navegação do robô. Cada célula, por exemplo, pode indicar a presença de um obstáculo na região correspondente do ambiente, ou indicar a probabilidade desta determinada região estar ou não ocupada por um obstáculo.

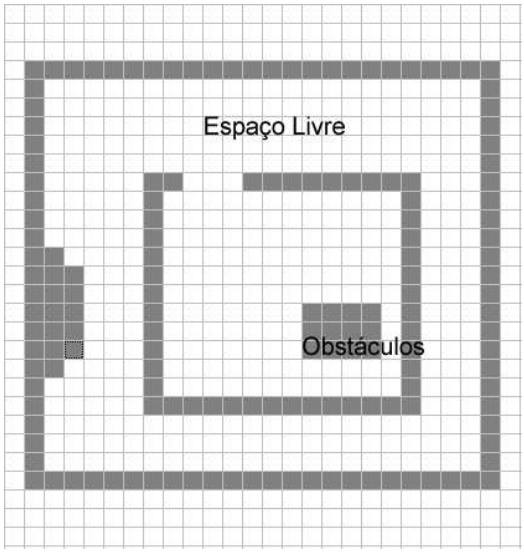

Fig. 3.4 Mapa representado por uma matriz

As técnicas que utilizam matrizes (Fig. 3.4), também chamadas de matrizes de ocupação, são simples de se construir, representar e manter em ambientes de larga escala. O uso de matrizes facilita a determinação de caminhos até o destino.

No entanto as matrizes consomem muita memória e requerem muito tempo para atualizações globais. Isto ocorre pois a resolução da matriz deve ser precisa o suficiente para capturar todos os detalhes importantes do ambiente.

3.2.3.1 - Transformada de Distância

Neste método, descrito em [Len90], a célula do destino é marcada com uma distância de valor 0. Todas as outras posições são marcadas com um valor muito alto, ou "infinito". O algoritmo inicia o cálculo no destino, e em cada passo ele visita todas as posições adjacentes às posições visitadas no passo anterior. O valor de distância para a posição "**i"** adjacente a posição "**j"** é atualizado da seguinte forma:

Se **map(i)** estiver ocupado então **d(i) = infinito**;
Senão **d(i)** recebe o menor entre **d(i)** ou **(d(j) + c(i,j)**);

Onde **c(i,j)** é o custo associado para se mover da posição **j** para a posição **i**.

A transformada de distância se expande em torno do destino como uma onda, se propagando em torno dos obstáculos. O caminho mais curto de qualquer posição até o destino pode ser encontrado seguindo os vizinhos com o menor **d(i)** até que **d(i)** seja igual a **0 (zero)**.

Cada vez que a posição do destino for alterada, a transformada de distância para todo o mapa deve ser recalculada. Outro problema associado com este método, é o fato de que os caminhos são gerados muito próximos dos obstáculos, não aceita obstáculos móveis e o mapa deve ser bem definido.

3.2.3.2 - Campos Potenciais

Os métodos de campos potenciais para a navegação de robôs móveis são muito populares entre os pesquisadores da área. A idéia de forças imaginárias agindo em um robô foi sugerida por Andews & Hogan [And83] e Khatib [Kha90]. Nestas abordagens os obstáculos exercem uma força repulsiva no robô, enquanto o objetivo aplica uma força atrativa no robô. A soma de todas as forças, a força resultante R, determina a direção e a velocidade do movimento. Uma das razões para a popularidade deste método é a sua simplicidade e elegância. Um método de campos potenciais simples pode ser implementado rapidamente e produz um resultado inicial aceitável sem a necessidade de muitos refinamentos. Thorpe utilizou um método de campo potencial para planejamento de trajetória *offline* e Krogh & Thorpe [Kro86] sugeriram um método generalizado de campo potencial que combina planejamento de trajetória local e global.

O método de campos potenciais foi implementado em robôs móveis com dados sensoriais reais por Brooks [Bro86], e por Arkin [Ark89]. No entanto, o robô de Arkin trabalhava muito lentamente, sua performance em um caminho com obstáculos era de aproximadamente 0.12 cm/sec.

Uma abordagem que produziu bons resultados foi proposta por Borenstein & Koren [Bor89]. O método *Virtual Force Field* (VFF), foi especialmente desenvolvido para evitar obstáculos em tempo real. O método VFF permite um controle de movimento rápido, contínuo e preciso através de um ambiente com obstáculos desconhecidos.

O método VFF utiliza uma matriz cartesiana bidimensional chamada *histogram grid* **C**, para representar os obstáculos. Cada célula **cij** no *histogram grid* armazena um determinado valor de certeza **(CV)cij** que representa a confiança do algoritmo na existência de um obstáculo naquela posição. Esta representação é derivada do conceito de *certainty grid* originalmente proposta por Moravec & Elfes [Mor85]. No *histogram grid*, **(CV)cij** é incrementado quando uma leitura sensorial indica a presença de um obstáculo naquela célula.

Simultaneamente, o conceito de campo potencial é aplicado ao *histogram grid*. A medida que o robô se move, uma janela de **(Ws x Ws)** células o acompanha, se sobrepondo a região **C**. Esta região é chamada de região ativa **(C*)**, e as células que momentaneamente pertencem a esta região são chamas de células ativas **(cij*)**.

Cada célula ativa exerce um força repulsiva virtual **Fij** contra o robô. A magnitude desta força é proporcional a **cij*** e inversamente proporcional a $\mathbf{d^n}$, onde **d** é a distância entre a célula e o centro do robô, e **n** é um número positivo.

Na implementação de Borenstein & Koren [Bor89] foi assumido um valor de **n=2**. A soma de todas as forças repulsivas gera a força repulsiva resultante **Fr**. Ao mesmo tempo, uma força atrativa virtual **Ft** de magnitude constante é aplicada no robô, puxando-o em direção ao objetivo. A soma de **Fr** e **Ft** gera o vetor de força resultante **R**. A direção de **R** é utilizada como referência para determinar a velocidade de rotação do robô.

Apesar dos bons resultados fornecidos pelos métodos de campos potenciais, eles possuem uma série de problemas inerentes, que independem de uma implementação em particular:

- Situações de armadilha devido a mínimos locais (comportamento cíclico);
- Não conseguem passar através de obstáculos muito próximos;
- Oscilação na presença de obstáculos ou em corredores estreitos;

O problema mais conhecido e mais citado dos campos potenciais são os mínimos locais ou situações de armadilha (Andrew & Hogan [And83], Tilove [Til89]). Um mínimo local pode ocorrer quando um robô entra em uma área sem saída (por exemplo, um obstáculo em forma de U). Estas armadilhas podem ser criadas por diversas configurações de obstáculos. No entanto, estas situações de armadilha podem ser solucionadas através de heurísticas ou através de um replanejador global.

As situações de armadilha solucionadas com o uso de heurísticas geralmente acabam gerando um caminho muito longo. O uso de um replanejador global aumenta a qualidade do caminho gerado, mas tem um custo computacional mais alto.

Outro problema é que a utilização de campos potenciais na navegação de robôs móveis necessita que o robô seja capaz de identificar a posição, ou direção do objetivo.

A grande vantagem deste método é que ele não necessita de uma representação interna do ambiente (mapa), e pode navegar por um ambiente que possua obstáculos móveis.

3.2.3.3 - AStar (A)*

Apesar dos algoritmos de procura AStar(A*) e DStar(D*) não necessitarem que o espaço de estados seja representado como uma matriz, na maioria das aplicações de navegação para robôs móveis em que se utiliza AStar ou DStar o ambiente é representado desta forma. Por este motivo esses algoritmos serão descritos aqui.

Praticamente qualquer problema pode ser solucionado (assumindo que existem recursos infinitos) utilizando-se um algoritmo de procura. Primeiro é necessário encontrar uma maneira de descrever o problema como uma coleção de estados, e regras que possam ser aplicadas para se alterar estes estados. Pode-se imaginar o espaço de estados como sendo uma árvore com o estado inicial do problema sendo a raiz, e se expandindo para baixo criando um galho em cada ponto, dependendo de quantas regras possam ser aplicadas nesses pontos.

Para solucionar um problema como esse é necessário encontrar um estado de destino nesta árvore e então traçar o seu caminho de volta ao estado inicial.

Um dos principais algoritmos utilizados para solucionar o problema de planejamento de trajetória através de procura em grafos é o algoritmo AStar (A*). Ele foi desenvolvido em 1968 para combinar métodos heurísticos (que usam informações sobre o problema a ser resolvido para tomar decisões) e métodos formais (que não utilizam informações sobre o problema, mas podem ser analisados formalmente).

Informações mais detalhadas sobre o método AStar podem ser encontradas em Nilsson [Nil80].

O problema é que até mesmo o mais simples dos problemas cria árvores de busca enormes. O algoritmo AStar necessita de alguma ajuda para executar a procura de forma eficiente. Esta ajuda é fornecida por heurísticas. As heurísticas fornecem informações sobre a qualidade de cada estado, de forma que a procura seja executada primeiro nas rotas mais promissoras (busca direcionada pela heurística).

No caso da navegação de robôs móveis autônomos, nosso problema é encontrar um caminho da posição inicial até um destino especificado evitando a colisão com os obstáculos do ambiente. Neste problema o espaço de estados é representado como uma matriz. Cada célula desta matriz é um estado. As regras que podem ser aplicadas são as 8 direções que o robô pode ser mover a partir da célula, dependendo dos obstáculos.

A partir da descrição do problema, a idéia e executar uma procura neste espaço de estados para encontrar o rota mais curta até o destino.

Durante a procura é necessário manter duas listas de estados (para a descrição deste algoritmo os estados também serão referidos como nodos). A primeira é a lista **OPEN**. Ela armazena os nodos que foram gerados pelas regras a partir de um determinado nodo, mas não se sabe ainda para que direção estes nodos apontam. A segunda lista é chamada de **CLOSED**, e armazena os nodos que foram gerados e já foram explorados.

Cada estado é armazenado com alguns atributos extras para auxiliar o algoritmo de procura. Um atributo importante é um ponteiro para o nodo pai. É necessário saber como se chegou até determinado nodo para se gerar um caminho de volta ao estado inicial.

Cada nodo possui três valores associados que são utilizados durante a procura. O custo do nodo '**c**' (custo do caminho já percorrido), o valor estimado pela heurística para este nodo '**h**'(custo estimado do caminho que falta percorrer), e o total dos dois chamado '**f**'.

Para cada movimento é atribuído um custo de valor constante. Como é necessário descobrir o custo do caminho do nodo atual de volta até o estado inicial, quando o custo é calculado para este nodo ele é somado com o custo do seu pai.

Para se calcular a heurística utilizamos a distância do nodo atual até o nodo destino. Este tipo de heurística funciona bem em matrizes, mas isso não é uma regra geral. De fato quanto mais informações a heurística for capaz de processar de forma eficiente, melhor será o caminho encontrado.

Finalmente o valor '**f**' representa a qualidade do nodo em termos de custo para se chegar até a posição, e se a posição esta próxima do destino. Esta informação permite ao algoritmo de procura dar prioridade aos caminhos mais promissores.

O algoritmo é apresentado a seguir em pseudo código C:

```
BOOL AStar()
{
  list OPEN;
  list CLOSED;

  NODE node;
  node.cost = 0;
  node.heuristic = GetNodeHeuristic( node );
  node.f = node.cost + node.heuristic;
  node.parent = NULL;

  push node on OPEN;

  while OPEN is not empty
  {
    // o melhor nodo é aquele como menor 'f'
    node = remove o melhor nodo de OPEN;

    if node é o destino
    {
      constrói caminho (seguindo os ponteiros para os pais)
      return success
    }
       // senão é necessário calcular o nodos
       // que podem ser gerados a partir deste nodo
       // aplicando a regras, calculando o custo e
       // o valor de heuristica para cada novo nodo
    NodePushSuccessors( OPEN, CLOSED, node );

       // uma vez que ele ja foi visitado
    push node onto CLOSED;
  }
  // se o algoritmo chegar neste ponto
  // nenhum caminho até o destino foi encontrado
  return FALSE;
}
```

Algoritmo AStar

O método AStar necessita de um mapa bem definido do ambiente e não é capaz de planejar um caminho preciso quando o ambiente possui obstáculos móveis. É necessário que a posição do robô seja conhecida.

Apesar disso, o método AStar é capaz de executar uma busca rápida do caminho ótimo (segundo a heurística definida).

3.2.3.4 - DStar (D)*

Apesar do algoritmo AStar [Nil80] se mostrar eficiente em ambientes estáticos onde ocorrem poucas mudanças na estrutura do mapa, ele se mostra bem menos eficiente quando o ambiente se modifica constantemente, ou quando não são fornecidas informações suficientes sobre a estrutura do ambiente.

Nestes casos é possível produzir caminhos válidos utilizando somente o algoritmo AStar. Se calcula um caminho até o destino utilizando as informação iniciais, então o robô segue este caminho até que ele encontre o destino ou até que os sensores percebam alguma discrepância entre o mapa e o ambiente, atualizando o mapa. Após o mapa ser atualizado um novo caminho é calculado da posição atual até o destino. Esta técnica é chamada de replanejamento, e é considerada uma técnica de força bruta pois é muito ineficiente, principalmente em ambientes complexos onde poucas informações são fornecidas antes do início da navegação.

O algoritmo DStar (D* ou *Dynamic* A*) é equivalente ao replanejador de força bruta, mas fornece resultados de maneira mais eficiente. Para grandes ambientes, que necessitam de matrizes com milhões de células para serem representados, resultados experimentais indicaram que ele é até 200 vezes mais rápido que o algoritmo AStar em replanejamento, possibilitando operações em tempo real. Ver Stentz [Ste94] para uma descrição mais detalhada e resultados experimentais.

Como o AStar, DStar utiliza uma matriz para representar o mapa. Cada célula (ou estado) possui um custo, uma estimativa do custo para se chegar no destino (heurística) e um ponteiro para o vizinho que originou o caminho.

DStar mantém um lista OPEN de estados que ainda necessitam ser visitados. Inicialmente o estado de destino é colocado na lista OPEN com um custo inicial de zero. O estado com o menor custo de caminho na lista OPEN é repetidamente expandido, propagando o custo do caminho para os seus vizinhos, até que um custo ótimo é calculado para todas as células no mapa. O robô então se movimenta, seguindo os ponteiros em direção ao destino. Enquanto o robô se movimenta os sensores podem detectar algum obstáculo que não era esperado, então todos os caminhos que contém a célula onde o obstáculo se localiza não são mais válidos. O custo desta célula é atualizado com um valor muito alto para identificá-la como um obstáculo, as células adjacentes são inseridas na lista OPEN, então os estados são repetidamente expandidos na lista OPEN para propagar o aumento no custo dos caminhos que foram invalidados. Os estados que transmitiram o aumento do custos são chamados RAISE. A medida que os estados RAISE se propagam, eles entram em contato com seus vizinhos que podem então reduzir seus custos. Estes são chamados de estados LOWER. A medida que os estados LOWER são propagados pela lista OPEN, os ponteiros são redirecionados e um novo caminho é calculado contornando os estados invalidados.

No caso oposto, se os sensores do robô detectam a inexistência de um obstáculo esperado, então um novo caminho pode existir até o destino. DStar atualiza o custo da célula que não possui mais um obstáculo com um valor menor, e adiciona as células vizinhas na lista OPEN como estados LOWER. A medida que os estados LOWER se propagam, se for possível, um novo caminho é calculado.

Em ambos os casos, DStar determina uma distância máxima que a propagação de custo deve ser expandida para se encontrar um novo caminho, caso contrário o caminho atual ainda é considerado ótimo.

Um dos problemas com este método é o fato de que ele só mostra uma grande diferença de eficiência em relação ao algoritmo AStar quando as diferenças entre o mapa e o ambiente são encontradas próximas ao ponto de origem, pois a propagação de caminho é menor. Quando o destino se encontra a grandes distâncias do ponto de origem, e uma diferença no mapa é percebida em um local próximo ao ponto de destino, a propagação se torna muito pesada computacionalmente, tornando a eficiência do algoritmo equivalente a de um replanejador de força bruta.

3.3 - Tabela Comparativa

Para que fosse possível comparar todos os métodos de uma forma mais estruturada, eles foram organizados em uma tabela. Nesta tabela são comparados os seguintes atributos:

- *representação*: indica o tipo de representação de ambiente que este método utiliza;
- *custo computacional*: indica o custo computacional do método;
- *mínimos locais*: indica se o método possui o problema de mínimos locais;
- *global*: indica se o método tem a capacidade de navegar de forma global no ambiente;
- *ambiente dinâmico*: indica se o método tem a capacidade de navegar em um ambiente dinâmico;
- *posição precisa*: indica se o método necessita da posição precisa do robô no ambiente;

Método	Representação	Custo Computacional	Mínimos Locais	Global	Ambiente Dinâmico	Posição Precisa
DistBug	Não Utiliza	Baixo	Não	Não	Não	Não
Nav. Baseada em Comportamentos	Não Utiliza	Baixo	Sim	Não	Sim	Não
Mapas Neurais	Campo Neural	Alto	Sim	Sim	Não	Sim
Grafo de Visibilidade	Poligonal	Alto	Não	Sim	Não	Sim
Voronoi	Poligonal	Alto	Não	Sim	Não	Sim
Decomposição Celular	Poligonal Topológica	Médio	Não	Sim	Não	Sim
Transformada de Distância	Grade	Médio	Não	Sim	Não	Sim
Campos Potenciais	Grade, Poligonal	Médio	Sim	Não	Sim	Não
Astar	Grade	Baixo	Não	Sim	Não	Sim
Dstar	Grade	Médio	Não	Sim	Sim	Sim

Tab. 3.1 Tabela Comparativa das Técnicas de Navegação

Apesar de alguns dos métodos apresentados não necessitarem de uma posição precisa para executar a navegação do robô móvel autônomo, a maioria deles necessita saber a localização do robô. Um dos principais problemas relacionados com o controle e navegação de robôs móveis autônomos é a localização. Este problema é apresentando em maiores detalhes no próximo capítulo.

Capítulo 4 - Localização

Um importante subproblema na navegação de robôs móveis autônomos é a tarefa de localização. A partir de um conhecimento acumulado sobre o ambiente (mapa) e utilizando as leituras atuais dos sensores, o robô deve ser capaz de determinar e manter atualizada a sua posição e orientação em relação a este ambiente, mesmo que os sensores apresentem erro e/ou ruído.

Considere, por exemplo, uma pessoa com um mapa tentando localizar um certo local em uma cidade desconhecida. Para conseguir utilizar o mapa, a pessoa primeiro precisa determinar a sua posição na cidade. Para isso, ela observa o local a sua volta e compara suas observações com as informações representadas no mapa. No exemplo, as observações podem ser um sinal de trânsito ou um ponto de referência. Em muitos casos uma única observação não é suficiente para determinar a posição. Isto ocorre pois diversos locais são parecidos. Assim, o problema de ambigüidade é uma parte inerente do problema de localização. Para acabar com a ambigüidade entre diferentes posições possíveis, é necessário manter estas posições como válidas e coletar informações adicionais em outras posições do ambiente. Em muitos casos basta se locomover em uma direção aleatória até que o número de observações seja suficiente para acabar com a ambigüidade.

Outro problema que esta pessoa pode enfrentar é o seguinte: devido a mudanças em alguns locais, a cidade parece diferente do que está representado no mapa. É o problema dos ambientes dinâmicos.

Uma vez que a pessoa conseguiu se localizar na cidade, ela precisa manter a sua posição conhecida a medida que ela se move no ambiente. Esta tarefa pode ser executada de forma eficiente comparando continuamente as observações com os locais esperados no mapa da cidade. No entanto, assim que a pessoa não estiver mais certa sobre a sua posição ela deve ser capaz de se relocalizar.

Este exemplo mostra que o conhecimento sobre a posição de uma pessoa em um determinado ambiente é uma precondição para esta pessoa conseguir utilizar um mapa deste ambiente. Robôs móveis autônomos enfrentam o mesmo problema quando precisam executar suas tarefas. Um posicionamento correto e uma navegação efetiva é uma precondição básica para uma aplicação robótica móvel de sucesso. Para poder utilizar um mapa do ambiente para executar os planos de navegação, o sistema de controle do robô deve possuir alguma forma de localização. Localização é um dos problemas fundamentais na robótica móvel autônoma.

A competência da localização não se restringe somente a ajudar na navegação, mas também servirá para a exploração e para a adaptação do mapa do ambiente pelo robô. Sem um método confiável e relativamente preciso de localização, são encontrados os seguintes problemas:

- fica mais difícil integrar múltiplas leituras sensoriais de um mesmo objeto;
- características, pontos de referência e obstáculos descobertos durante a navegação não podem ser inseridos em suas posições corretas no mapa;
- não se pode seguir de modo adequado um plano traçado pelo planejador.

4.1 - Definições

A seguir são apresentadas algumas definições iniciais sobre o problema da localização de robôs móveis autônomos.

Localização Local: Quando a posição inicial do robô é conhecida, a tarefa do algoritmo de localização é manter a posição conhecida com o menor erro possível, isto é chamado de localização local. Este tipo de localização necessita de recalibragem quando a estimativa do erro excede um limite pré-estabelecido.

Localização Global: É a habilidade de se estimar a posição do robô sem nenhum conhecimento de sua posição inicial e a habilidade do robô em se relocalizar quando este se perder no ambiente.

Localização Robusta: Muitas técnicas de localização se baseiam em um modelo estático do ambiente. Conseqüentemente, uma localização robusta em um ambiente dinâmico requer a habilidade de se estimar a posição do robô mesmo que um número significante de observações não possam ser encontradas no mapa. Por exemplo, a aparência de um ambiente pode mudar significativamente quando os móveis são trocados de posição, portas são abertas ou fechadas, e pessoas se locomovem no ambiente.

Localização Ativa: Para aumentar a eficiência da localização pode-se optar por eliminar as ambigüidades entre as possíveis posições de forma ativa. Controlando de forma ativa a movimentação do robô para que ele se locomova em direção a objetos ou lugares que possam facilitar a quebra da ambigüidade. As frases chaves na localização ativa são: "para onde se mover" e "para onde olhar" de forma a melhor localizar o robô.

4.2 - Representação do Ambiente (Mapa)

A escolha correta da representação interna do ambiente é um dos principais fatores que influencia o sucesso de uma técnica de localização. O mapa do ambiente deve ser capaz de armazenar informações suficientes para que o robô se localize, mas não pode ser muito pesado a ponto de comprometer a performance do sistema de controle. Atualmente existem duas principais formas de representação de ambiente utilizadas pelas técnicas de localização: os mapas geométricos e os mapas topológicos.

Os mapas geométricos representam os objetos de acordo com a sua posição geométrica absoluta. Este tipo de mapa pode ser poligonal (os obstáculos são representados por polígonos) ou baseado em grades (o valor de cada célula indica a presença de um obstáculo naquela posição).

Os mapas topológicos são baseados nas relações geométricas observadas entre características do ambiente, ao invés de sua posição absoluta. Esta representação toma a forma de grafos, onde os nodos são os pontos de referência (ou características) e os arcos indicam a relação entre estes pontos e como o robô pode se mover entre eles. Os mapas topológicos podem ser construídos sem a necessidade de estimar a posição exata do robô. Esta abordagem permite a integração de grandes mapas, minimizando o

problema de erros sensoriais, pois as conexões entre os nodos são relativas. O problema com este tipo de representação é o fato de que é necessário que o sistema seja capaz de reconhecer os pontos de referência e diferencia-los uns dos outros.

4.3 - Técnicas de Localização para Robôs Móveis

Existe um grande número de abordagens que atacam o problema de manter uma informação precisa sobre a posição de um robô móvel. As mais óbvias incluem técnicas perfeccionistas que utilizam coordenadas absolutas geradas por um GPS[3] ou outro tipo de sistema de rastreamento de alta precisão. Outras abordagens necessitam de muros paralelos ou perpendiculares que podem ser facilmente reconhecidos e orientados. Sobre situações controladas, estas técnicas funcionam satisfatoriamente, mas limitam a autonomia do robô, e não se adaptam as condições encontradas em ambientes reais.

Serão apresentadas a seguir algumas das principais técnicas de localização para robôs moveis, limitando o escopo para as técnicas que valorizam a autonomia e que necessitem de pouca ou nenhuma modificação na estrutura do ambiente onde irão atuar.

4.3.1 - Dead Reckoning

Existem diversas abordagens possíveis para estimar a localização de um robô móvel. A maioria dos seres humanos, se baseiam na visão para determinar exatamente a sua posição em um determinado local. Por exemplo, se uma pessoa tem seus olhos vendados e decide sair de uma sala, nos primeiros passos ela ainda estará com uma certeza alta sobre a sua própria posição e sobre a posição dos objetos na sala, mas quanto mais ela se deslocar, maior será a sua incerteza. No entanto, uma pessoa com problemas visuais tem alta capacidade de lidar com situações como essa. Isto se deve ao fato dela ter desenvolvido uma alta capacidade de julgar as distâncias percorridas, bem como uma habilidade de se lembrar com maiores detalhes da estrutura tridimensional do ambiente.

Em termos robóticos isto é chamado de "dead reckoning" [Bor96] ou localização baseada em odometria. Armazenando precisamente os movimentos das rodas utilizando encoders (ver item 5.2.3.1), um robô pode determinar o quanto ele se locomoveu e o quanto ele rotacionou.

Mas este tipo de técnica possuiu diversas fontes de erros que afetam a precisão da localização. A primeira são as derrapagens das rodas e a imprecisão dos motores. A medida que o robô se move no ambiente ou entra em contato com algum objeto, é inevitável que aconteça alguma derrapagem ou pequenas falhas no motores. Se o robô estiver andando em uma linha reta estes pequenos erros não influenciam muito diretamente a estimativa de posição, mas se o robô estiver fazendo uma curva, o erro angular vai causar um grande erro translacional se o robô percorrer um caminho mais longo. Outro problema existente de deve ao fato do robô calcular apenas uma aproximação discreta da sua posição. A quantidade de erro resultante desta aproximação depende da freqüência que o calculo é realizado e da precisão dos encoders.

[3] O GPS (*Global Positioning System*) é uma rede de satélites artificiais em órbita da terra utilizados para fornecer as coordenadas de algum ponto do planeta através de uma aparelho receptor dos sinais de rádio emitidos por estes satélites.

4.3.2 - Triangulação

Os métodos de triangulação para a localização de robôs móveis se baseiam nos métodos tradicionais da cartografia e navegação, que utilizam os ângulos observados entre pontos de referência do ambiente para calcular a posição relativa do observador.

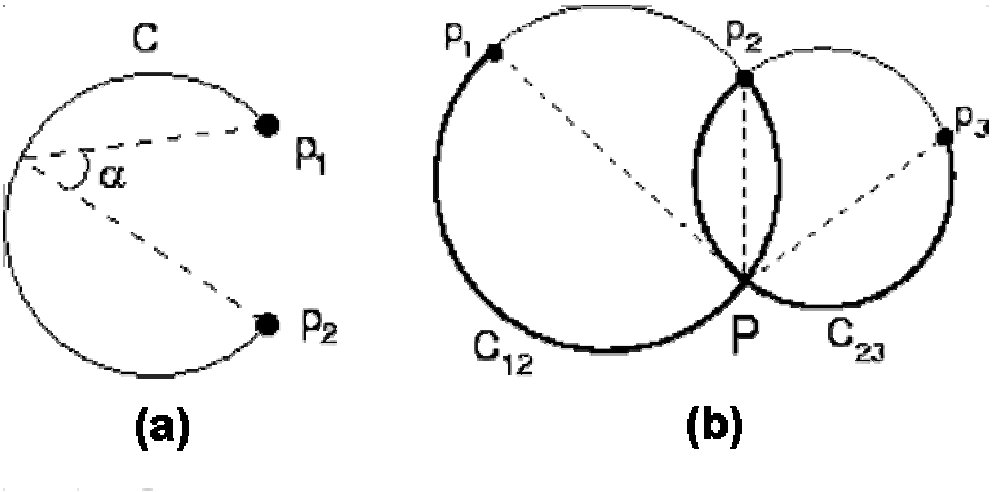

Fig. 4.1 Triangulação

Fica claro que com apenas o ângulo entre 2 pontos de referência, a posição do observador é delimitada pelo arco de um círculo (Fig. 4.1a). No caso com 3 pontos de referência, a localização do observador se encontra em um único ponto, que é a intersecção entre dois círculos (Fig. 4.1b), desde que os pontos de referência não sejam coincidentes.

Sugihara [Sug88] desenvolveu uma técnica que considera o problema de localização quando os pontos de referência observados são indistinguíveis uns dos outros. Neste trabalho foi proposto um método computacionalmente eficiente de encontrar uma correspondência consistente entre os pontos de referência detectados e pontos em um mapa do ambiente. Este método de correspondência foi aprimorado por Avis & Imai [Avi90]. Os dois trabalhos se baseiam na extração confiável de pontos de referência a partir de dados sensoriais e da precisão das medidas dos ângulos entre estes pontos de referência. Somente uma leve consideração é dada para o problema de se utilizar ângulos incertos.

Sutherland & Thompson [Shu93] propuseram uma abordagem que considera o problema de correspondência resolvido, onde os ângulos observados entre os pontos de referência não são precisos. Foi demonstrado que uma série de pontos de referência podem ser selecionados para minimizar a área de incerteza, isto é, a área em que o robô pode se localizar a partir de um certo nível de incerteza na observação dos ângulos. A área de incerteza pode variar significativamente dependendo da configuração dos pontos de referência. O objetivo do trabalho era selecionar os pontos de referência que minimizassem a área de incerteza.

Todos os métodos de triangulação apresentados fazem uma série de restrições sobre o ambiente e o robô. Em todos os casos o robô possui um mapa preciso de todos os pontos de referência do ambiente, e em alguns casos se assume que todos os pontos

de referência são distinguíveis uns dos outros. Fica claro que estes métodos não são aplicáveis em um ambiente real.

Moon et. al. [Moo01] propuseram um método para extrair automaticamente pontos de referência visuais estáveis a partir de imagens obtidas por câmeras instaladas em um robô móvel, tornando possível a aplicação dos métodos de triangulação em ambientes reais. Segmentos de linha verticais distintas em superfícies planares são selecionadas como atributos estáveis para observação visual pois elas podem ser observadas de forma confiável de diversos pontos de vista. A posição dos atributos selecionados incluem um nível de incerteza devido a erros na visão e no movimento do robô. Para utilizar estes atributos como pontos de referência, suas posições são modeladas por uma distribuição de probabilidade e então estimadas por um filtro de Kalman [Kal60].

4.3.3 - Localização Baseada em Atributos

A localização baseada em atributos não necessita de modificações no ambiente [Bor96]. Ao invés disso, atributos físicos que ocorrem naturalmente no ambiente (cantos, portas, muros, etc.) são extraídas dos dados obtidos pelos sensores do robô. É necessário então estabelecer a correspondência entre o atributo detectado e os atributos armazenados no mapa do ambiente. Muitas vezes atributos do mesmo tipo são detectados. Neste caso, a única maneira de eliminar esta ambigüidade é analisando a posição e a orientação de um atributo relativo a outro. A correspondência é então estabelecida encontrando uma transformação que traga as coordenadas do robô em alinhamento com as coordenadas do mapa global. Este processo é prejudicado quando se detectam atributos fantasma, falha em detectar certos atributos, incerteza e ruído na posição e orientação dos atributos, e erro na posição e orientação dos atributos armazenados no mapa global. O problema da correspondência entre os atributos encontrados e os armazenados no mapa, é o principal motivo que torna a localização baseada em atributos pesada computacionalmente.

4.3.4 - Inversão Sensorial

A maioria dos métodos de posicionamento possuem uma série de limitações. O robô geralmente só se move sobre uma superfície plana, em ambientes compostos de estruturas retilíneas. Eles dependem da extração robusta de características, que na maior parte dos casos se baseiam em suposições sobre a estrutura do ambiente. Muitos destes métodos necessitam de uma mapa completo e preciso do ambiente.

Alguns pesquisadores desenvolveram métodos para evitar o uso de características explícitas ou mapas. Estes métodos expressam os dados sensoriais como uma função da posição do robô, e tentam inverter esta função. A análise de componentes principais (PCA), também conhecida como análise *eigenspace*, é uma técnica de classificação de padrões que apresenta aplicações de sucesso na área de reconhecimento de faces e objetos e que também vem sendo aplicada no problema de localização robótica [Cro98]. A PCA trata dados sensoriais densos ou vetores de características extraídas como um vetor n-dimensional, e classifica os dados de entrada baseado em uma projeção deste vetor em um subespaço que maximiza a discriminação

de outras amostras.Estes métodos são similares ao filtro de Kalman a medida que se baseiam em uma aproximação linear do comportamento subjacente dos dados, mas se diferem pois não dependem de uma interpretação explícita de características e sim alinham a variação estatística dos dados para selecionar as características com discriminação máxima.

Dudek & Zhang [Dud96] utilizaram técnicas de inversão sensorial em seu método de localização baseado em imagens. Neste trabalho, uma rede neural foi utilizada para inverter as estatísticas de bordas de uma imagem em função da posição. Em um trabalho similar, Oore, Hilton & Dudek implementaram um estimador de posição utilizando redes neurais que processam dados sensoriais de sonares [Oor97]. Apesar das redes neurais apresentarem bons resultados com entradas não lineares ou complexas, elas podem ser difíceis de configurar, o que é particularmente problemático, pois quando o ambiente é modificado a rede necessita ser retreinada.

Um problema associado com a inversão sensorial é o fato da função a ser invertida não ser um-para-um, uma situação que pode ser facilmente detectada em um pré-processamento. Dudek & Zhang [Dud96] consideram esta dificuldade em seu trabalho implementando uma medida de consistência que incorpora múltiplas leituras sobre diferentes condições de visualização para conseguir uma consistência ótima no resultado da estimativa de posição.

4.3.5 - Scan Matching

Scan Matching [Cox91] é o processo de transladar e rotacionar uma varredura sensorial de distância (obtida de sensores ultra-sônicos, por exemplo) de forma que uma sobreposição entre a leitura dos sensores e um mapa a priori seja encontrada. A maioria dos métodos de scan matching assumem que a estimativa da posição inicial é muito próxima da posição real do robô para poder limitar o espaço de procura.

A posição do robô e a sua atualização através do método de scan matching são modelados utilizando-se uma única distribuição gaussiana. Isto tem a vantagem de que a posição do robô pode ser calculada com uma boa precisão, e um método eficiente para calcular o passo de atualização pode ser utilizado: o filtro de Kalman.

O filtro de Kalman é uma técnica de processamento de sinais introduzida por Kalman [Kal60]. Os métodos que utilizam o filtro de Kalman representam a crença na posição do robô através de uma distribuição gaussiana unimodal sobre o espaço de estados tridimensional do robô (**x,y,dir**). O módulo desta distribuição mantém a posição corrente do robô, e a sua variância representa a incerteza sobre a posição. A medida que o robô se move, a gaussiana é deslocada de acordo com a distância medida pelos sensores. Simultaneamente, a variância da gaussiana é incrementada de acordo com o modelo odométrico do robô. Novas entradas sensoriais são incorporadas na estimativa da posição combinando a percepção com o modelo do ambiente.

O método de filtro de Kalman tem a seguinte forma: A probabilidade da posição do robô é modelada por uma distribuição gaussiana $\mathbf{l(t) \sim N(\mu_l , K_l)}$, onde $\boldsymbol{\mu_l = (x,y,\alpha)^T}$ é a média, e $\mathbf{K_l}$ é uma matriz de covariância 3 X 3.

Na locomoção do robô $\mathbf{a \sim N((\delta,\theta)^T, K_a)}$ onde o robô se move para a frente uma certa distância δ e rotaciona θ, a posição é atualizada de acordo com:

$$\mu_l := E(F(l,a)) = \begin{pmatrix} x + \delta\cos(\alpha) \\ y + \delta sin(\alpha) \\ \alpha + \theta \end{pmatrix}$$

$$\mathrm{K}_l := \nabla F_l \mathrm{K}_l \nabla F_l^T + \nabla F_a \mathrm{K}_a \nabla F_a^T$$

Onde **E** denota o valor esperado da função **F** ,e $\nabla \mathbf{F_l}$ e $\nabla \mathbf{F_a}$ são as suas Jacobianas com respeito a '**l**' e '**a**'.

Através do método scan matching uma atualização de posição $\mathbf{s \sim N(\mu_s , K_l)}$ é obtida e a posição do robô é atualizada utilizando as equações do filtro de Kalman [May90]:

$$\mu_l := (\mathrm{K}_l^{-1} + \mathrm{K}_s^{-1})^{-1} \cdot (\mathrm{K}_l^{-1} + \mathrm{K}_s^{-1} \mu_s)$$

$$\mathrm{K}_l := (\mathrm{K}_l^{-1} + \mathrm{K}_s^{-1})^{-1}$$

O sucesso do filtro de Kalman depende da habilidade do método de scan matching para corrigir a posição do robô. Uma série de abordagens foram propostas:

Cox [Cox91] combina as leituras dos sensores com os segmentos de linha de um mapa em CAD do ambiente. Ele atribui pontos de varredura para os segmentos de linha baseados no vizinho mais próximo e então procura por uma translação e uma rotação que minimize a distância quadrada total entre os pontos de varredura e as linhas alvo.

Weiss et. al. [Wei95] utiliza histogramas para combinar pares de varreduras. Primeiro ele calcula o histograma de ângulos para determinar a rotação de duas varreduras e então utiliza os histogramas **x** e **y** para calcular a translação. Esta abordagem é pesada computacionalmente e a precisão depende do tamanho da discretização dos histogramas.

Lu e Milios [Lu97] combinam pares de varreduras atribuindo pontos de uma varredura para pontos da outra varredura. Para encontrar o ponto de varredura correspondente duas heurísticas chamadas "closest-point-rule" e "matching-range-rule" são aplicadas e uma combinação é utilizada para calcular a rotação e a translação das duas varreduras. este algoritmo chamado IDC (interactive dual correspondence) é bem adaptado para qualquer tipo de ambiente, inclusive ambiente não poligonais.

Gutmann e Schlegel [Gut96] utilizam uma combinação da abordagem de Cox e do método IDC, integrando de forma robusta e eficiente o método de combinação de linhas com as capacidade universais do algoritmo IDC. Eles chamaram o algoritmo de "combined scan matcher" CSM.

Infelizmente todas essas abordagens de scan matching possuem uma alta complexidade computacional, geralmente **O(n^2)** onde n é o número de pontos de

varredura, e a sua robustez é limitada devido a necessidade de limitar o espaço de procura a uma região muito pequena.

4.3.6 - Localização Topológica

Uma abordagem que evita as representações geométricas, é a localização topológica, que se baseia em descrições topológicas do ambiente. Kuipers [Kui88] descreve uma técnica de localização que utiliza esta abordagem. O trabalho de Kuipers é motivado pelas evidências da ciência cognitiva que indicam que os humanos dependem inicialmente de informações topológicas, ao invés de métricas, para navegação. O modelo topológico de Kuipers é organizado como um grafo, com os nodos representando pontos de referência (chamados "distinctive places" - DPs), e os arcos representando rotas conectadas. Em acréscimo, conhecimento procedural sobre como detectar os DPs e de como seguir as rotas descritas pelos arcos são incorporados no modelo. Informações métricas são associadas com cada nodo e arco para permitir uma precisão geométrica local.

O processo ocorre em duas fases: exploração e navegação. Na fase de exploração, o robô navega pelo ambiente, localizando DPs e rotas, e organizando o modelo topológico. Na fase de navegação, o robô utiliza os resultados da fase de exploração para se mover de forma eficiente de um DP para outro seguindo as rotas encontradas anteriormente.

O paradigma de Kuipers depende muito das noção de DPs. Um DP é definido como um ponto no ambiente que é distinguível localmente de acordo com alguns critérios geométricos. Isto é definido como "assinatura", um sub-conjunto de características que são maximizadas no DP, tais como: distância entre objetos, simetria do ambiente, entre outros.

O trabalho de Kuipers tem um apelo intuitivo devido ao uso de uma representação topológica do ambiente. Suas simulações pareceram demonstrar algum sucesso, no entanto, a definição de DPs parece ser muito vaga para uma implementação no mundo real e erros sistemáticos podem fazer com que os DPs não sejam detectados na fase de navegação.

4.3.7 - Localização Markov

A localização Markov [Fox99] trata do problema de estimativa de estado utilizando dados sensoriais. A localização Markov é um algoritmo probabilista: Ao invés de manter uma única hipótese sobre a posição do robô, a localização Markov mantém uma distribuição de probabilidade sobre um espaço de hipóteses. A representação probabilista permite que cada uma destas hipóteses seja avaliada de uma forma matemática.

Essa distribuição pode ter formas arbitrárias representando diferentes tipos de informação sobre a posição do robô. Por exemplo, o robô pode iniciar a navegação utilizando uma distribuição uniforme representando que ele está completamente incerto sobre sua posição real. Além disso é possível trabalhar com múltiplos modos no caso de

situações de ambigüidade. Quando o robô tem uma certeza elevada sobre a sua posição, ela é representada por uma distribuição unimodal centrada em volta da localização real do robô. Baseada em sua natureza probabilista, a localização Markov pode estimar a localização do robô globalmente, pode lidar com situações ambíguas, e pode relocalizar o robô no caso de falhas. A técnica de localização Markov pode utilizar diversas formas de representação interna do ambiente, mas a mais comum é o mapa baseado em "grid". A principal desvantagem da localização Markov é o seu elevado custo computacional.

Vamos ilustrar o conceito básico da localização Markov com um exemplo simples. Considere o ambiente apresentado na Fig. 4.2. Para manter a simplicidade, vamos assumir que o espaço de posições do robô é unidimensional, isto é, o robô só pode se mover horizontalmente. Vamos supor que o robô é colocado em algum lugar do ambiente, mas a sua posição não é informada. A localização Markov representa este estado de incerteza como um distribuição uniforme sobre todas as possíveis posições, como mostra a Fig. 4.2a. A seguir o robô verifica os seus sensores e descobre que esta ao lado de uma porta. A localização Markov modifica sua "crença" aumentando a probabilidade dos locais que são próximos a portas, e diminuindo a probabilidade para todos os outros locais. Isto pode ser observado na Fig. 4.2b. Pode-se notar que a crença resultante é multi-modal, refletindo o fato que a informação disponível é insuficiente para uma localização global. Nota-se também que os locais que não são próximos a portas não possuem probabilidade nula. Isto se deve ao fato de que os sensores apresentam muitos ruídos, e um único sinal de uma porta normalmente é insuficiente para excluir a possibilidade de não se estar perto de uma porta.

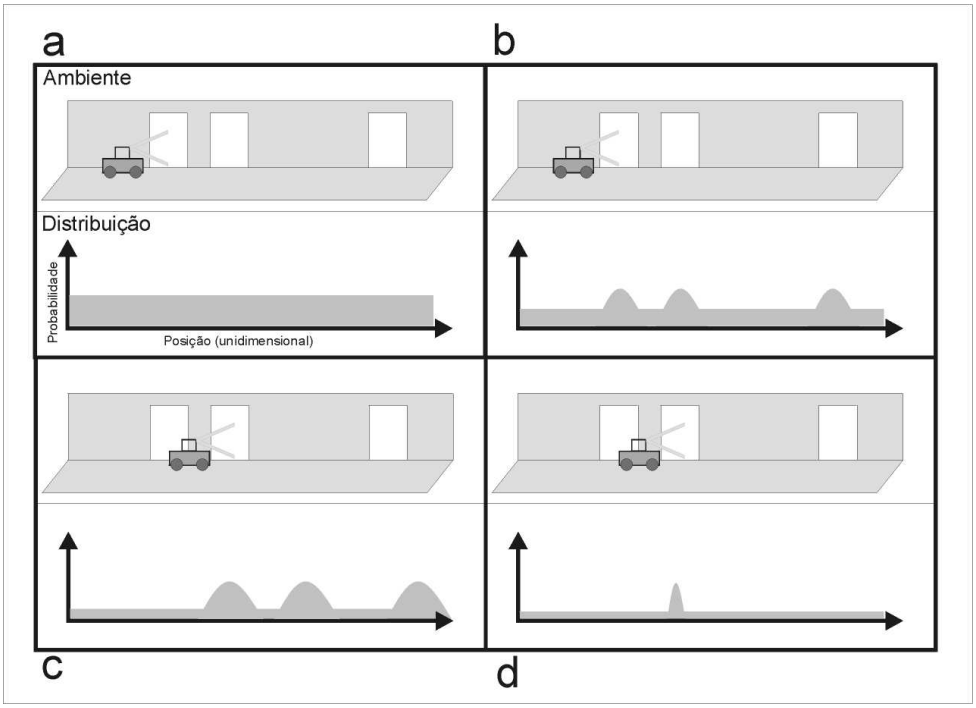

Fig. 4.2 Localização Markov

O próximo passo do robô é se mover um metro para frente. A localização Markov incorpora esta informação deslocando a distribuição de probabilidade proporcionalmente, como pode ser visto na Fig. 4.2c. Para levar em conta o erro que

sempre esta presente quando um robô se move, que inevitavelmente causa uma perda de informação, a nova crença é menor do que a anterior. Finalmente, o robô verifica seus sensores uma segunda vez, e novamente ele percebe que esta ao lado de uma porta. Esta observação é multiplicada pela distribuição de probabilidade atual, que gera a crença final que é apresentada na Fig. 4.2d. Neste instante de tempo, a maior parte da probabilidade esta centrada em volta de uma única posição. A certeza do robô sobre a sua posição é muito alta.

De uma maneira mais formal, vamos definir a posição (ou localização) de um robô móvel com uma variável tridimensional **l = (x,y,θ)**, onde **x,y** são as coordenadas cartesianas e **θ** é a direção do robô. Sendo **lt** a posição real do robô no tempo **t**, e **Lt** sendo a variável aleatória correspondente.

Normalmente, o robô não sabe a sua posição exata. Ao invés disso, ele possui uma crença sobre onde ele possa estar. Sendo **Bel(Lt)** a crença do robô sobre a sua posição no tempo **t**. **Bel(Lt)** é uma distribuição de probabilidade sobre o espaço de posições. Por exemplo, **Bel(Lt = l)** é a probabilidade do robô estar na posição **l** no tempo **t**. A crença é atualizada em resposta a dois tipos diferentes de eventos: a leitura de dados sensoriais sobre o ambiente (sonares, câmera, etc.), e a movimentação do robô detectada por sensores odométricos. Sendo as leituras sensoriais do ambiente representadas por **s**, e as leituras odométricas da movimentação do robô representadas por **a**, e as suas variáveis aleatórias correspondentes **S** e **A**, respectivamente.

O robô percebe uma seqüência de leituras, sensoriais '**s**' e odométricas '**a**'. Sendo **d = {d0,d1,...dT}** a seqüência de leituras, onde cada **dt** (com **0 <= t <= T**) é uma leitura sensorial ou uma leitura odométrica. A variável **t** indexa os dados, e **T** representa os dados mais recentes.

A seguir o algoritmo geral da localização Markov é apresentado. **P(l | a,l')** representa o modelo cinemático do robô, pois modela como o movimento afeta a posição do robô. A probabilidade condicional **P(s | l)** é chamada de modelo sensorial, pois modela os dados sensoriais adquiridos pelos sensores do robô.

```
// inicializa a crença para todas as posições do ambiente
PARA CADA POSIÇÃO 'l' FAÇA
        Bel(L0 = l)  <-  P(L0 = l)
FIM PARA

LAÇO PRINCIPAL
        SE (uma nova entrada sensorial 'sT' é recebida) FAÇA

                alfaT <- 0

                // aplica o modelo sensorial
                PARA CADA POSIÇÃO 'l' FAÇA
                        Bel(LT = l) <- P(sT | l)*Bel(LT-1 = l)
                        alfaT <- alfaT + Bel(LT = l)
                FIM PARA
```

```
        // normaliza a crença
        PARA CADA POSIÇÃO 'l' FAÇA
            Bel(LT=l) <- alfaT ^ (-1) * Bel(LT=l)
        FIM PARA
    FIM SE

    SE (uma leitura odométrica  aT é recebida) FAÇA

        // aplica o modelo cinemático
        PARA CADA POSIÇÃO 'l' FAÇA
            Bel(LT = l) <-   integral(    P(l | l', aT ) * Bel(LT-1 = l') dl'   )
    FIM PARA
    FIM SE

FIM LAÇO PRINCIPAL
```

No algoritmo de localização Markov **P(L0 = l)**, que inicializa a crença **Bel(L0)**, reflete o conhecimento anterior sobre a posição inicial do robô. Esta distribuição pode ser inicializada de forma arbitrária, mas na prática dois casos prevalecem: se a posição do robô é completamente desconhecida, **P(L0)** é uma distribuição uniforme. Se a posição inicial do robô é conhecida aproximadamente, então **P(L0)** é tipicamente uma distribuição gaussiana centrada na posição do robô.

4.3.8 - Localização Monte Carlo

O algoritmo de localização Monte Carlo (MCL) [Fox99b] é uma versão do algoritmo de amostragem/importância re-amostragem (sampling/importance re-sampling - SIR)[Rub88]. Ele também é conhecido como algoritmo de concentração (condensation algorithm) [Isa98], filtro de Monte Carlo[Kit96], entre outros. Todos estes métodos são conhecidos genericamente como filtros de partículas, e uma apresentação mais completa sobre as suas propriedades pode ser encontrada em Doucet [Dou98].

A idéia principal sobre o algoritmo MCL é o fato dele representar a crença nas possíveis posições **Bel(l)** por uma série de **N** amostras aleatórias ou partículas **S = { si | i = 1..N }**. Uma série de amostras constituem uma aproximação discreta da distribuição de probabilidade. As amostras no algoritmo MCL são do tipo:

<<x,y,θ>,p>

Onde **<x,y,θ>** denota a posição do robô, e **p >= 0** é um fator numérico de peso, análogo a uma probabilidade discreta. Por consistência, se assume que:

$$\sum_{n=1}^{N} p_n = 1$$

Em analogia com a localização Markov, vista anteriormente, o algoritmo MCL possui duas fases: Fase da movimentação do robô e fase de leituras sensoriais.

Na fase de movimentação do robô, o MCL gera **N** novas amostras que aproximam a posição do robô após que ele tenha se movimentado. Cada amostra é gerada aleatoriamente retirando-se uma amostra do conjunto de amostras anterior, com a chance sendo determinada pelo valor **'p'** da amostra antiga. Sendo **l'** a posição da amostra antiga. A posição **l** da nova amostra é gerada utilizando **P(l | l', a)**, utilizando a ação de movimento **'a'** observada. O valor **p** da nova amostra é $\mathbf{N^{(-1)}}$.

Na fase de leitura dos sensores, as informações são incorporadas redimensionando o peso da série de amostras, de uma forma semelhante ao algoritmo de localização Markov. Mais especificamente, sendo **<l,p>** uma amostra. Então:

p <- alfa * P(s | l)

Onde **s** é a leitura do sensor, e **alfa** é uma constante de normalização que garante que $\sum_{n=1}^{N} p_n = 1$. A incorporação das leituras dos sensores é realizada em duas etapas, uma em que **p** é multiplicado por **P(s | l)**, e outra em que os vários valores **p** são normalizados.

É necessária a adição de um pequeno número de amostras aleatórias, uniformemente distribuídas após cada passo, para auxiliar o algoritmo MCL a se relocalizar caso a posição estimada seja incorreta.

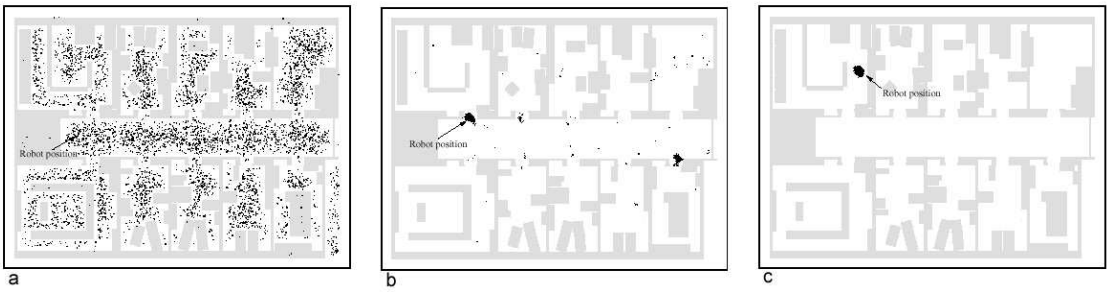

Fig. 4.3 Localização Monte Carlo

A Fig. 4.3 mostra a distribuição das partículas no ambiente durante o processo de localização (**a**: distribuídas uniformemente – se posição; **b**: concentradas em diferentes pontos – posição ambígua; **c**: concentradas em um único ponto – posição encontrada).

Uma das grandes vantagens do algoritmo MCL, é a maneira que ele distribui os recursos computacionais. Amostrando proporcionalmente à chance de uma determinada posição ser a correta, os recursos computacionais acabam se concentrando nas regiões com maior probabilidade de possuírem a posição real do robô.

O algoritmo MCL foi utilizado na implementação do sistema de controle proposto neste trabalho. Os resultados obtidos demonstraram sua capacidade de localizar o robô móvel autônomo em diversas condições do ambiente. Mais detalhes podem ser vistos no item 6.2.3 e no item 7.1.

4.4 - Localização em Ambientes Dinâmicos

A maioria das técnicas de posicionamento assumem que o ambiente é estático para conseguir localizar o robô com sucesso dentro de um mapa. Um ambiente que se modifica constantemente, e que possua obstáculos móveis (outros robôs, pessoas, veículos, etc.), dificulta muito a tarefa de posicionamento, e em alguns casos até impede que um posição correta seja encontrada. Para contornar este problema, diversas técnicas foram desenvolvidas. As que mais se destacam são as utilizadas na localização probabilista (localização Markov e Monte Carlo).

Para evitar as interferências causadas por obstáculos não modelados no ambiente, estas técnicas selecionam somente as leituras dos sensores que são relacionadas aos obstáculos estáticos, filtrando as leituras dos sensores que são relacionadas aos obstáculos desconhecidos e/ou móveis. Existem duas técnicas principais [Die98]:

4.4.1 - Filtro de Entropia

Entropia é a medida de incerteza sobre o resultado de uma variável randômica. Quanto maior for a entropia da estimativa de estado, maior será a incerteza do robô sobre onde ele se localiza. O filtro de entropia calcula a mudança relativa da entropia após uma nova leitura dos sensores. Caso haja uma mudança negativa na entropia após ser incorporada a leitura dos sensores, isso indica que o robô esta menos certo a respeito de sua localização e a leitura s é então eliminada.

4.4.2 - Filtro de Distância

Neste filtro, as leituras dos sensores são selecionadas baseado em suas distâncias relativas a distância até o obstáculo mais próximo do mapa. Mais especificamente, uma leitura de sensor s é considerada corrompida se ela for mais curta que a distância esperada do modelo do ambiente. O filtro remove as leituras **'s'** que com probabilidade **P** (geralmente **P** fica entre **0,9 - 0,99**) são mais curtas do que o esperado.

4.5 - Discussão Final

O sistema de controle proposto neste trabalho tem como base fundamental a capacidade de localização do robô móvel autônomo. A partir do momento em que o robô possua uma posição estimada em relação a um mapa, as tarefas de navegação se tornam mais simples e robustas, ampliando a sua autonomia no ambiente. A capacidade de localização possibilita que um robô móvel autônomo atue em um ambiente do mundo real. Na arquitetura proposta nesta dissertação se optou pela utilização da localização Monte Carlo em conjunto com o filtro de distância, pois estes métodos se mostram mais robustos em ambientes estáticos e em ambientes dinâmicos, capacitando o sistema de controle a realizar localização local, global e relocalização.

Nos capítulos anteriores foram vistos os principais sistemas de controle, e as principais técnicas de navegação para robôs móveis autônomos. Neste capítulo foram apresentados os principais métodos para a localização de robôs móveis, concluindo

assim o estudo das técnicas necessárias para o desenvolvimento de uma arquitetura de controle robusta. No próximo capítulo serão apresentados os modelos de ambiente, modelos cinemáticos e os modelos sensoriais estudados durante o desenvolvimento do trabalho. Também serão apresentados diversos simuladores de robôs móveis disponíveis, e o simulador SimRob3D desenvolvido para a validação da arquitetura de controle.

Capítulo 5 - Modelagem e Simulação

Sempre existiu um caloroso debate entre os pesquisadores que desenvolvem sistemas de controle para robô móveis autônomos, este debate pode ser resumido assim: "A simulação é uma ferramenta poderosa o suficiente para gerar conclusões, ou uma teoria deve sempre ser testada em um robô real ?"

Apesar de tanto as simulações quanto as implementações físicas possuírem seus méritos em diferentes campos de pesquisa, o assunto se torna mais importante quando se discute o caso dos robôs móveis autônomos. Neste caso em particular, cada uma destas metodologias tem suas respectivas vantagens e desvantagens (Tab. 5.1). Normalmente se afirma que simulações são rápidas. Computadores seqüenciais de alta performance ou séries de computadores em paralelo são ferramentas poderosas para a reprodução e análise da dinâmica de sistemas complexos. Em poucos dias de simulações um cientista pode reproduzir a vida e morte de uma população inteira de organismos. Mas somente até um certo nível de sofisticação. No caso específico do robôs móveis, ainda é muito mais rápido adquirir imagens com uma câmera real de um ambiente real, ao invés de simular o ambiente, a câmera e o processo de aquisição da imagem. Isto se deve ao fato de que as vezes enormes cálculos são necessários para simular fenômenos físicos relativamente triviais.

Prós	Contras
Custo usualmente inferior quando se trata da implementação de sistemas complexos que utilizam alta tecnologia;	Custo computacional para reproduzir fenômenos e comportamentos reais;
Repetibilidade dos experimentos facilita a análise e a comparação de algoritmos;	Não consegue modelar precisamente todos os elementos do mundo real;

Tab. 5.1 Prós e contras do uso de simulações

As simulações em computador são mais baratas do que uma implementação real. O pesquisador pode explorar uma hipótese ou um novo algoritmo em uma simulação antes de investir tempo e dinheiro em um robô real. Apesar disto ser verdade na maioria das vezes, em alguns casos isto não se aplica. Tudo depende do nível de confiança e precisão da simulação. Se o padrão deve ser alto, então a implementação do simulador deverá envolver diversos programadores especializados durante vários meses. Isto em certas circunstâncias pode custar mais caro do que a construção de um robô real.

É amplamente aceito o fato de que as simulações permitem um controle completo de todas as variáveis do sistema, sendo possível replicar os resultados, analisar os fenômenos, acelerar ou desacelerar os processos. Os simuladores são uma ferramenta importante para os pesquisadores. Os computadores permitem que utilizemos nossa imaginação para testar as hipóteses mais bizarras para criar robôs autônomos atuando nos mais diferentes ambientes e situações. No entanto, quando utilizamos uma ferramenta de simulação, precisamos estar cientes de que em diversos pontos estamos criando restrições. É preciso estar claro que não estamos lidando diretamente com o mundo real.

Levando em consideração as idéias acima, as restrições de tempo e de recursos materiais, o fato de não existir um simulador disponível capaz de simular o nosso sistema de controle de forma adequada, optou-se por implementar neste trabalho um simulador para validar o sistema de controle proposto.

A seguir serão apresentados os princípios que seguiremos para a implementação do simulador, os modelos que serão utilizados e uma breve revisão dos principais simuladores de robôs móveis existentes. Ao final será descrita a implementação do simulador e as suas capacidades e restrições.

5.1 - Como Simular

Torrance [Tor92] apresenta uma série de princípios que devem ser seguidos no desenvolvimento e utilização de um simulador de robôs móveis autônomos.

O primeiro e mais importante princípio é que um simulador deve ser implementado com um senso claro sobre as suposições feitas sobre o domínio que ele está modelando, e sobre as capacidades e limites das simulações. Afirmações sobre generalidade, escalabilidade e utilidade dos resultados demonstrados em um simulador sempre devem ser feitas sobre o próprio simulador e sobre as suas limitações em relação ao mundo real.

Uma importante característica de um bom simulador é a modularidade e a extensibilidade. Um simulador ideal permite que o usuário construa robôs customizados utilizando diversos sensores e atuadores. Os modelos de incerteza (erro, ruído) utilizados também devem ser customizáveis.

O simulador deve fornecer modelos dos principais tipos de sensores que são utilizados normalmente em robôs móveis, incluindo sonares, sensores infravermelhos, etc. Diversos tipos de atuadores e plataformas devem ser fornecidas também. O robô deve fornecer modelos precisos destes componentes e também modelos aproximados para que o usuário possa contar com uma opção de mais alta performance. As simulações contínuas devem ser utilizadas sempre que possível, pois efeitos indesejados das simulações discretas com robôs móveis já foram documentados [Bro91b].

As seqüências pseudo-aleatórias utilizadas para gerar os dados das simulações devem ser reproduzíveis de forma exata, pois isso possibilita que certas situações em modelos determinísticos sejam reproduzíveis, fornecendo uma grande vantagem que não seria possível no mundo real.

Um interface gráfica interativa é desejável pois facilita a operação, o desenvolvimento e a depuração dos sistemas de controle e dos modelos de robô. O simulador deve permitir que se trabalhe com obstáculos móveis que possam ser controlados pelo usuário ou por programas simples. Uma arquitetura cliente servidor ou outra abordagem de sistema distribuído seria muito útil para aumentar a modularidade do simulador. Isto permite que o sistema de controle seja implementado separadamente do módulos de simulação dos componentes do robô (sensores e atuadores) e da interface gráfica, permitindo assim que a carga computacional possa ser distribuída em diversos computadores.

5.2 - Modelos

Com base nos estudos realizados sobre os robôs móveis autônomos, sua cinemática, seus sensores e o ambiente onde atuam, realizou-se uma revisão dos diversos modelos que serão utilizados tanto no simulador como no sistema de controle.

5.2.1 - Modelos de Ambiente

Existem diversas formas de se representar o ambiente onde um robô móvel autônomo executará suas tarefas, mas dado o escopo deste trabalho, focalizaremos nossa atenção em três modelos de representação principais que serão discutidos a seguir.

5.2.1.1 - Modelo Geométrico

O modelo de ambiente geométrico (Fig. 5.1a) é o tipo de representação espacial mais tradicional na área de robótica móvel [Mil85][Leo91]. Neste tipo de modelo de ambiente, primitivas geométricas são extraídas dos dados sensoriais, ou fornecidas pelo usuário, para se construir uma abstração de alto nível do ambiente (Ex. planta baixa de um prédio). O mundo pode ser construídos por segmentos de reta ou polígonos, por exemplo. Esta abordagem permite uma representação consistente que pode ser facilmente manipulada e apresentada por computadores, além disso ela é de fácil compreensão para o usuário. No entanto, o paradigma geométrico possui desvantagens, os modelos se tornam relativamente frágeis e pouco representativos pois muitas vezes dependem de informações "a priori" sobre o ambiente. Os modelos geométricos são de fácil interpretação por parte dos seres humanos, mas são menos representativos do ponto de vista dos sensores do robô.

5.2.1.2 - Modelo baseado em Grades

As abordagens baseadas em grades (Fig. 5.1b), tal como a proposta por Moravec & Elfes [Mor88] entre outros, representa o ambiente por uma grade ou matriz. Cada célula da grade pode, por exemplo, indicar a presença de um obstáculo na região correspondente do ambiente. As grades de ocupação são consideradas fáceis de se construir e de se atualizar em ambientes de larga escala[Thr96]. Desde que a geometria intrínseca da grade corresponda diretamente a geometria do ambiente, a posição do robô em relação ao modelo pode ser determinada pela sua posição e orientação no mundo real. Como conseqüência, em diferentes posições onde o robô percebe um mesmo valor sensorial (locais parecidos ou ambíguos) são naturalmente diferenciáveis se utilizando uma abordagem baseada em grades. Um problema que ocorre com este tipo de modelo de ambiente é a grande quantidade de memória necessária para representar o ambiente, principalmente quando é exigida uma maior precisão.

5.2.1.3 - Modelo Topológico / Semântico

Alguns pesquisadores da robótica móvel desenvolveram um paradigma de representação topológica (Fig. 5.1c) [Kui88][Bas89]. Esta abordagem é muitas vezes considerada qualitativa pois as distâncias exatas entre os pontos de referência do ambiente não são utilizadas, ao invés disso se utiliza as informações sobre adjacência e

direção (normalmente em uma estrutura de dados na forma de grafo). Os pontos de referência e os arcos que indicam as adjacências, armazenam também informações semânticas sobre as característica do local, sobre as formas de se locomover de um ponto de referência para outro, ou outra informação que seja necessária e que facilite a navegação do robô para executar uma determinada tarefa. O paradigma topológico é apoiado pelas evidências da ciência cognitiva que indicam que os seres humanos utilizam esta abordagem para representar mentalmente o espaço. Assim, este paradigma é considerado intuitivo para os humanos. Infelizmente, ele não se adapta muito bem em implementações com sistemas sensoriais comuns, que utilizam tipicamente dados métricos.

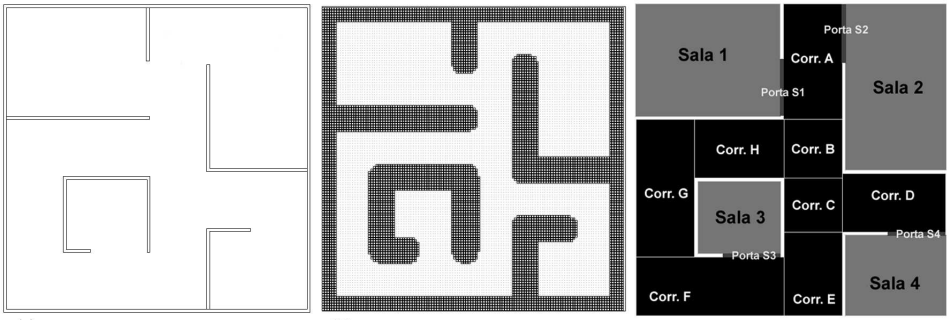

Fig. 5.1 Geométrico (a); Baseado em Grade (b); Topológico/semântico (c);

5.2.2 - Modelos Cinemáticos

Os modelos cinemáticos descrevem a maneira como o robô se locomove pelo ambiente. A seguir são apresentados dois importantes modelos utilizados na robótica móvel.

É importante notar que algumas simplificações foram feitas: a velocidade é considerada constante. Os efeitos da aceleração foram ignorados. Quando se trabalha com robô pequenos, onde a massa (e inércia) da plataforma é pequena, as mudanças na velocidade são quase instantâneas permitindo esta simplificação sem muito impacto nos resultados das simulações. Se fossemos trabalhar com robôs maiores, a massa da plataforma é um fator importante e a aceleração seria considerada.

5.2.2.1 - Cinemática Ackerman

O modelo cinemático de Ackerman é derivado da industria automotiva, que utiliza um sistema mecânico no sistema de direção do veículo que permite que a roda interna gire com um ângulo maior do que a externa. Geometricamente as rodas frontais de um carro, se estiverem seguindo um caminho circular, precisam girar com ângulos diferentes. A roda interna (roda do lado que se deseja virar) precisa girar mais do que a externa para evitar derrapagens.

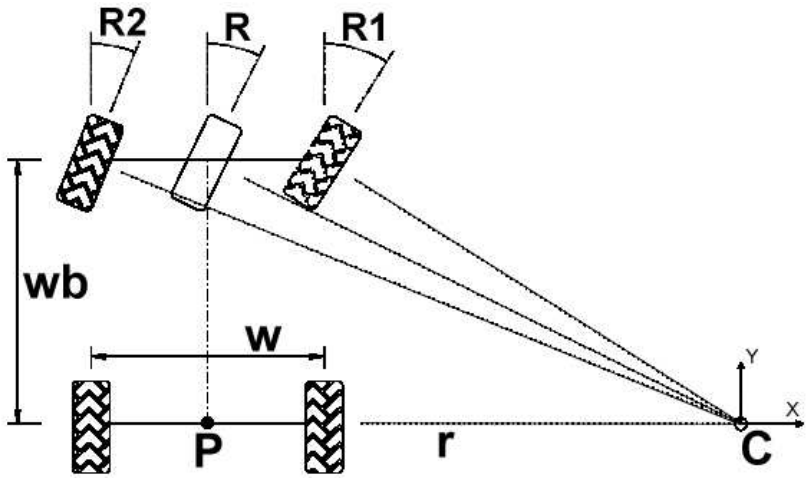

Fig. 5.2 Cinemática Ackerman

A Fig. 5.2 mostra que os eixos que se estendem das rodas dianteiras intersectam um ponto comum C sobre o eixo estendido das rodas traseiras. O ponto C é o centro de rotação do veículo. Esta geometria de direção satisfaz a equação de Ackerman [Byr92]:

$$\mathrm{R} = \frac{R1 + R2}{2} \qquad r = \frac{wb}{\tan(R)} \qquad DR = \frac{s}{r}$$

$$C(x) = P(x)(t-1) + r \cdot \cos(DV + \pi)$$
$$C(y) = P(y)(t-1) + r \cdot \sin(DV + \pi)$$

$$P(x)(t) = C(x) + r \cdot \cos(DV + DR - \pi)$$
$$P(y)(t) = C(y) + r \cdot \sin(DV + DR - \pi)$$
$$DV(t) = DV + DR$$

onde:

t = instante de tempo
s = velocidade do veículo
r = raio do eixo traseiro estendido
wb = comprimento do veículo
R1 = rotação da roda dianteira interna
R2 = rotação da roda dianteira externa
R = rotação média das rodas dianteiras
DR = delta de rotação do veículo
DV = direção do veículo
P = ponto pivô do veículo
C = centro de rotação do veículo

Este tipo de configuração é muito utilizado no desenvolvimento de robôs móveis autônomos, tanto para operação em ambiente internos, quanto para o uso em terrenos acidentados. A cinemática Ackerman permite o desenvolvimento de robôs móveis com uma maior capacidade de carga e facilita a sua construção e operação em diversos tipos de ambiente.

5.2.2.2 - Cinemática Diferencial

O sistema de direção diferencial é bastante conhecido, pois é o mesmo sistema utilizado nas cadeiras de roda. Duas rodas instaladas em um único eixo são acionadas e controladas independentemente uma da outra, possibilitando que o veículo se movimente e se direcione. Rodas passivas adicionais podem ser instaladas para suporte. Se as duas rodas são acionadas na mesma velocidade e sentido, o veículo se move em uma linha reta. Se uma roda se move mais rápido do que a outra o veículo segue um caminho em curva. Se ambas as rodas giram na mesma velocidade mas em sentidos opostos, o veículo gira em torno do próprio centro.

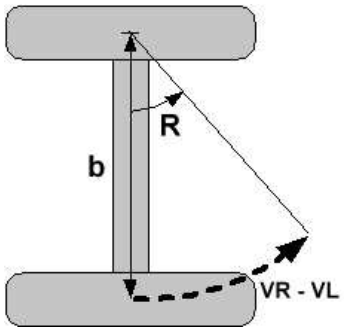

Fig. 5.3 Cinemática Diferencial

As equação cinemáticas de um sistema de direção diferencial, aplicadas em um robô móvel, podem ser facilmente derivadas [Sam90]. Com o auxílio da Fig. 5.3 podemos resumir estas equações da seguinte forma:

$$x(t) = x0 + \frac{b(VR+VL)}{2(VR-VL)}\left[\sin((VR-VL)t/b + R0) - \sin(R0)\right]$$

$$y(t) = y0 + \frac{b(VR+VL)}{2(VR-VL)}\left[\cos((VR-VL)t/b + R0) - \cos(R0)\right]$$

$$R(t) = (VR-VL)t/b + R0$$

t	= tempo
R0	= orientação inicial do robô
(x0,y0)	= posição inicial do robô
VR	= velocidade da roda direita
VL	= velocidade da roda esquerda
b	= tamanho do eixo
[x(t),y(t),R(t)]	= posição e orientação no tempo t

O sistema de direção diferencial é utilizado principalmente em robôs móveis que atuam em ambientes internos, com pisos planos e lisos. É um dos sistemas de direção mais populares entre os construtores de robôs por ser barato e por facilitar o desenvolvimento do sistema de controle. A capacidade do robô girar sobre o próprio eixo quando utiliza um sistema de direção diferencial, facilita o desenvolvimento dos módulos de navegação.

5.2.3 - Modelos Sensoriais

Existe uma enorme gama de sensores que podem ser utilizados em robôs móveis autônomos. A seguir são descritos os modelos dos sensores utilizados neste trabalho.

5.2.3.1 - Encoder

A maioria dos robôs móveis utilizam rodas para se locomover. Para medir o movimento do robô, os sensores mais populares são os encoders de roda, por serem simples e baratos. Normalmente estes sensores são óticos e são instalados nas rodas e no eixo para medir o número de rotações de um disco com fendas. O número de fendas determina a resolução do encoder. Conhecendo o raio das rodas e a relação de tamanho entre a roda e o eixo, é possível calcular o movimento do robô baseando-se nas informações obtidas do encoder. Este tipo de informação é conhecido como odometria.

Uma das grandes desvantagens das informações odométricas obtidas através de encoders de roda, é que elas são suscetíveis a erros. Os erros que ocorrem na odometria podem ser divididos em dois tipos: sistemáticos e não-sistemáticos.

As principais causas de erros sistemáticos são:

- diferentes diâmetros das rodas
- diâmetro da roda difere da especificação
- desalinhamento das rodas
- resolução do encoder limitada

As principais causas de erros não-sistemáticos são:

- derrapagem das rodas quando o robô se locomove em um carpete ou em um piso liso
- o robô atravessa o batente de uma porta
- intervenções externas (humanos)

Os erros sistemáticos se acumulam constantemente e causam um erro de deslocamento ilimitado com o passar do tempo. É importante manter esses erros em níveis baixos. Borenstein & Feng [Bor94] apresentam um teste de performance que permite que os erros sistemáticos sejam estimados fazendo com que o robô se mova repetidamente sobre uma trajetória pré-programada. Utilizando marcas no piso, os erros sistemáticos podem ser compensados, diminuindo até um certo nível a quantidade de erro.

Os erros não-sistemáticos são mais difíceis de tratar pois são de natureza aleatória. O grau de derrapagem é dependente do tipo de piso em que o robô esta se locomovendo, do tipo de roda, e da sua velocidade. Carpetes provocam mais derrapagens do que outros pisos. Passar pelo batente de uma porta pode provocar um salto na posição do robô que os encoders de roda não são capazes de detectar. Estes saltos devem ser compensados, senão o robô não será capaz de manter uma posição correta.

Do ponto de vista da localização, os erros odométricos são altamente indesejáveis, pois não permitem que um robô execute tarefas complexas. Este problema se torna mais claro quando pensamos em como um ser humano executa a tarefa de caminhar diretamente para a frente por 100 metros de olhos fechados. Nos primeiros metros a pessoa vai conseguir manter a direção, mas eventualmente ela vai começar a se deslocar para uma direção diferente. Após caminhar 100 metros a pessoa estará bem distante do objetivo. Este exemplo ilustra duas importantes características dos erros odométricos, erro de orientação e erro de translação. Um pessoa é capaz de caminhar 100 metros com apenas um pequeno desvio translacional, mas devido ao erro de orientação, o erro total se torna bem maior.

5.2.3.2 - Sonar

Os sonares possuem propriedades únicas que são muito úteis na robótica móvel. Este tipo de sensor é relativamente barato, confiável, fisicamente robusto, e pode perceber uma ampla variedade de objetos. O mais importante é que estimativas robustas de distância podem ser extraídas dos dados fornecidos pelo sonar. Estas informações podem ser utilizadas para evitar obstáculos, e geralmente são difíceis de se obter por outros métodos. Diversos robôs possuem um anel periférico de sonares que permite a detecção de obstáculos ao seu redor. Este tipo de anel sensorial é de fácil construção e tem se mostrado efetivo nas aplicações práticas.

Em um sistema sensorial que utiliza sonares para determinar distâncias, um pulso acústico é emitido de um transdutor. O transdutor então troca para um modo de recepção, esperando pelo retorno de um eco por um determinado período de tempo. Se um eco de retorno é detectado a distância R pode ser encontrada multiplicando a velocidade do som pela metade do tempo medido. O tempo é dividido pois é necessário levar em conta o tempo para atingir o objeto e o tempo para que o eco retorne até o transdutor:

$$\mathbf{R = c.t / 2}$$

onde c é a velocidade do som e t é o tempo em segundos. A velocidade do som, c, pode ser encontrada considerando o ar como um gás ideal e utilizando a equação:

$$\mathbf{c = 20 \bullet \sqrt{(T)}\ m/s}$$

que é valida com 1% de variação para a maioria das condições. A velocidade do som é proporcional a temperatura T (em Kelvins). Em temperatura ambiente (± 20 graus C) o valor é de 343,3 m/s.

Um exemplo de sonar é o sensor ultra-sônico da Polaroid utilizado em diversos robôs móveis. Estes sensores possuem transdutores desenvolvidos para operar no ar. A Fig. 5.4 mostra o padrão do cone de um sonar na freqüência de 50 Khz. A abertura do cone é cerca de 30 graus, e o intervalo confiável de distância varia entre 15 centímetros até 10 metros, com uma precisão de ± 1%.

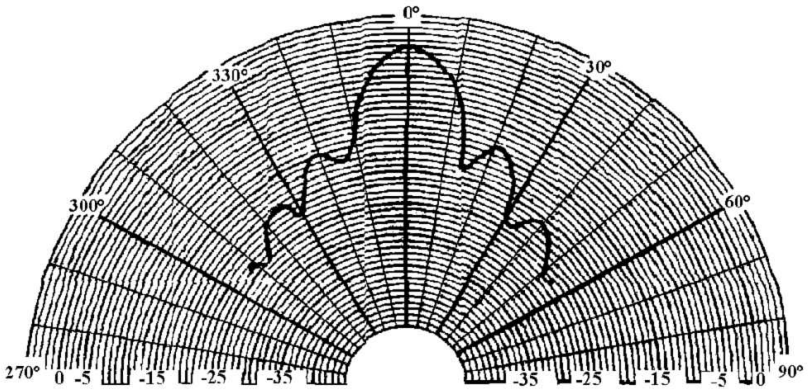

Fig. 5.4 Cone do Sonar (50Khz)

Um problema comum a todos os sonares é a reflexão do sonar. Por exemplo, fazendo uma analogia com a propagação das ondas de luz: o nosso olho pode ver os objetos pois a energia da luz incidente é dispersa pelos objetos, o que significa que alguma energia acaba por atingir nossos olhos, não importando o ângulo do objeto em relação a luz ou aos nossos próprios olhos. Esta dispersão ocorre porque a rugosidade da superfície de um objeto é mais larga se comparada ao comprimento de onda da luz (0,550 nm). Somente com superfícies muito lisas (como um espelho) a refletividade se torna altamente direcional para os raios de luz. Já o comprimento de onda de um sonar é muito maior (0,25 polegadas). Portanto, as ondas ultra-sônicas se refletem de forma direcional em praticamente todas as superfícies planas. A quantidade de energia retornada é dependente do ângulo de incidência do ultra-som. Executar uma tarefa de navegação utilizando sonares pode ser comparado com um ser humano tentando achar a saída de um labirinto de espelhos, no escuro, somente com uma lanterna.

5.2.3.3 - Laser

Outro tipo de sensor que vem sendo utilizado na robótica móvel é o laser. O laser é utilizado principalmente para medir distâncias até os obstáculos, e é utilizado em aplicações que necessitem de grande precisão. Sua principal desvantagem é o alto custo do equipamento.

O método mais utilizado para determinar distâncias utilizando um sensor laser é o de diferença de fase. A diferença de fase envolve a utilização de uma portadora (raio laser) que é modulada em diferentes comprimentos de onda. Medindo a diferença de fase entre o sinal transmitido e o sinal recebido após ser refletido por um objeto, a distância pode ser determinada.

Este tipo de sensor pode operar em diversos tipos de ambiente, tanto internos como externos. Em ambientes externos sua precisão é afetada principalmente pela luz solar.

A abertura do cone do laser é pequena, em média cerca de 0,5 graus. Os sensores de distância que utilizam laser são os mais precisos da robótica móvel, operando em um intervalo médio de 30cm até 30m com um erro de ± 3mm em um ambiente interno, e ± 5mm em um ambiente externo.

5.2.4 - Gerador de Números Pseudo-Aleatórios

Um gerador de números aleatórios é uma parte muito importante no processo de construção de um simulador, principalmente quando se utiliza técnicas probabilistas (localização Monte Carlo).

Um bom gerador de números aleatórios (GNA) deve ser uniformemente distribuído, estatisticamente independente e reproduzível [Rub81][Nie92]. Números realmente aleatórios não são reproduzíveis a não ser que eles sejam armazenados, então os números pseudo aleatórios são preferíveis em simulações. Repetir uma seqüência de números aleatórios é necessário para depurar as simulações e duas diferentes abordagens podem ser comparadas de forma mais eficiente se ambas utilizarem a mesma seqüência de números aleatórios [Bra87].

De acordo com Knuth [Knu81], uma técnica padrão para gerar números pseudo aleatórios é a utilização dos geradores lineares congruentes (LCG) que foram introduzidos por Lehmer em 1949 [Leh51]. Os LCGs computam o i-*nésimo* inteiro em uma seqüência pseudo aleatória pela recursão vista na equação:

$$x_i = (ax_{i-1} + c) \bmod m$$

onde a,c, e m determinam a qualidade estatística do gerador. Este gerador satisfaz muitos dos critérios estabelecidos para um bom GNA incluindo o fato de que ele requer muito pouca memória para ser implementado. O problema com os LCGs é que o seu período é limitado por m [Rub81]. Todos os geradores de números pseudo aleatórios geram seqüências que são periódicas, isto é, existe um inteiro t tal que $y_{n+t} = y_n$ para todo $n \geq 0$, onde y_n é o n-*ésimo* número em uma seqüência aleatória [Nie92]. Sem utilizar uma pesada aritmética de multi-precisão, o ciclo gerado pelos LCGs é limitado pelo tamanho da palavra da arquitetura do computador. Em uma máquina de 32 bits (31 com o bit de sinal), um máximo de $2^{31} - 1 = 2,147,483,647$ números podem ser gerados antes que a seqüência se repita. Dois bilhões de números podem parecer suficientes para uma única simulação mas em certos casos, tais como simulações Monte Carlo, onde milhares de partículas são calculadas em cada passo da simulação, o período se repetiria rapidamente.

Park e Miller [Par88] propuseram que o LCG com a = 16807, c = 0 e $m = 2^{31}-1$, o primo de Mersenne, seja adotado como padrão mínimo para um gerador de números aleatórios pois estes parâmetros foram testados exaustivamente e suas características são conhecidas.

Outro gerador revolucionário que utiliza um FSR (feedback shift register) foi introduzido em 1965 por Tausworthe [Tau65]. Ele pode gerar seqüências arbitrariamente longas de números aleatórios sem as desigualdades multidimensionais encontradas nos LCGs. O k-*nésimo* bit, a_k, de uma seqüência de bits randômicos é definida como:

$$a_k = c_1 a_{k-1} + c_2 a_{k-2} + \ldots + c_{p-1} a_{k-p+1} + a_{k-p}$$

Entretanto, Toothill *et al.* descobriram resultados negativos rodando um teste estatístico do método FSR em 1971 [Too71]. Dois anos mais tarde Lewis e Payne

refinaram o gerador de Tausworthe e criaram o *generalized FSR* (GFSR) [Lew73]. Este algoritmo utiliza base dois, com isso a operação de adição sem carry é o mesmo que um OU exclusivo (XOR ou ⊕). Então a equação anterior equivalente para inteiros (grupos de bits) é:

$$x_k = c_1 a_{k-1} \oplus c_{2x} x_{k-2} \oplus \ldots \oplus c_{p-1} x_{k-p+1} \oplus x_{k-p}$$

Utilizando primitivas trinomiais , $1 + x^q + x^p$, $p > q$, a equação anterior é reduzida para:

$$x_k = x_{k-q} \oplus x_{k-p}$$

Com isso é necessário somente uma operação de XOR e alguns cálculos de endereço para obter um inteiro gerado pelo GFSR, que se torna tão rápido quanto os LCGs. Kirkpatrick e Stoll apresentaram uma implementação específica do gerador apresentado na equação anterior (chamado R250) utilizando $q = 103$ e $p = 250$ [Kir81]. O algoritmo R250 foi escolhido para ser implementado no simulador desenvolvido neste trabalho, principalmente pela sua performance e pelo seu longo período (2^{249}).

5.3 - Estado da Arte em Simuladores de Robôs Móveis

Antes de se optar pela implementação de um novo simulador, alguns dos principais simuladores de robôs móveis existentes foram avaliados para se determinar se eles não seriam capazes de fornecer as ferramentas e condições necessárias para a validação do sistema de controle proposto neste trabalho.

Depois de avaliados, chegou-se a conclusão que nenhum dos simuladores poderia ser utilizado, ou por possuírem um alto custo, ou por não possuírem todos os recursos necessários (ver item 5.1) para as simulações pretendidas.

Os seguintes simuladores foram avaliados:

5.3.1 - Khepera Simulator

O simulador do robô Khepera [Mic97] foi um dos primeiros pacotes gratuitos de simulação para robôs móveis autônomos. Foi desenvolvido por Olivier Michel na EPFL em Lausanne, Suiça. Ele possibilita a desenvolvimento de controladores para o robô Khepera utilizando C/C++. Ele inclui um editor de ambientes e uma interface gráfica (Fig. 5.5). O código desenvolvido para o simulador pode ser utilizado também para controlar um robô real. Sua principal função é o aprendizado e a pesquisa na área de agentes autônomos.

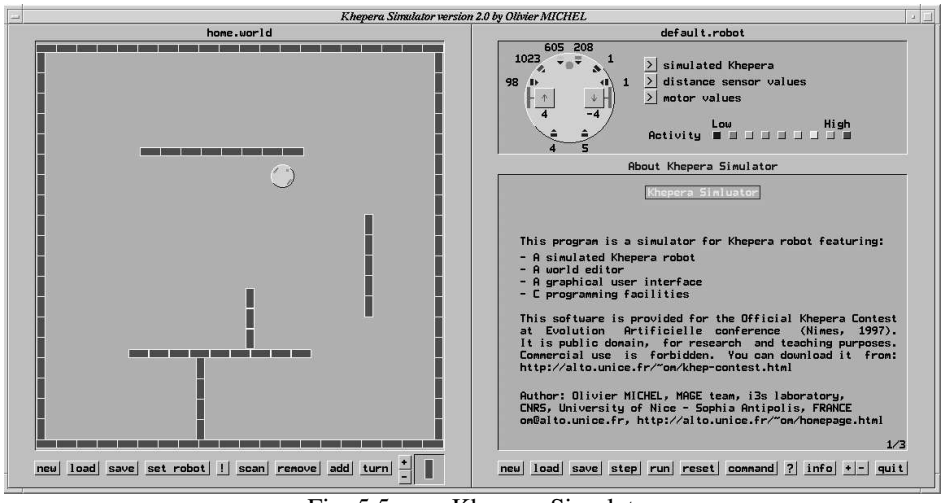

Fig. 5.5 Khepera Simulator

5.3.2 - Webots

Webots é um simulador de robôs móveis em 3D comercializado pela Cyberbotics [Cyb02], ele permite a simulação de robôs de duas rodas com cinemática diferencial. Seu público alvo são pesquisadores e professores da área de agentes autônomos, visão computacional e inteligência artificial. Ele inclui modelos prontos para os robôs Khepera, Alice, Koala, Pioneer2 e Moorebot. Modela sensores com a capacidade de detecção de obstáculos, visão, e manipuladores simples. O usuário pode programar cada robô utilizando C/C++ sendo compatível também com alguns dos robôs reais. O ambiente virtual é tridimensional (Fig. 5.6) e possui um editor para customizar os ambientes que são compostos de primitivas geométricas simples.

Apesar do simulador Webots ser um dos mais avançados existentes para a área de robôs móveis, seu custo elevado impossibilita sua utilização em muitas pesquisas. Além disso, o simulador só possui um tipo de cinemática e não permite a customização dos sensores.

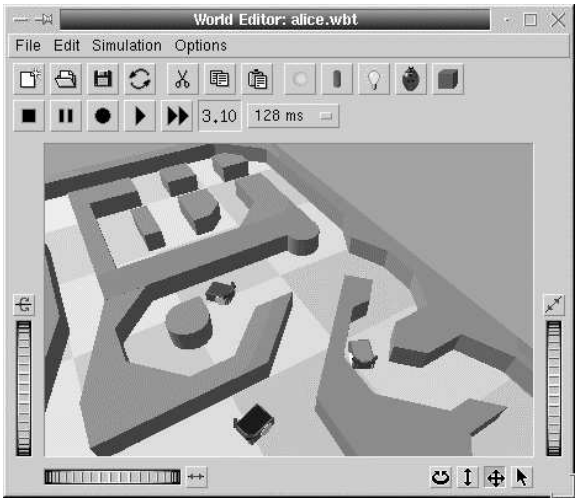

Fig. 5.6 Webots

5.3.3 - Mobotsim

Mobotsim é um simulador desenvolvido por Gonzalo Rodriguez Mir [Mir02]. Este simulador utiliza um modelo de ambiente bidimensional e a cinemática dos robôs é diferencial. Uma interface gráfica permite criar e editar robôs e objetos. A programação dos robôs é feita utilizando a linguagem BASIC (Fig. 5.7). Pode-se criar macros utilizando funções para obter as informações dos sensores e do robô.

O simulador Mobotsim foi desenvolvido para pesquisadores, estudantes e hobistas que queiram desenvolver, testar e simular robôs móveis.

Apesar de ser um simulador comercial com preço acessível, o Mobotsim é extremamente limitado, seu ambiente é bidimensional e só um tipo de sensor é modelado (ultrasônico).

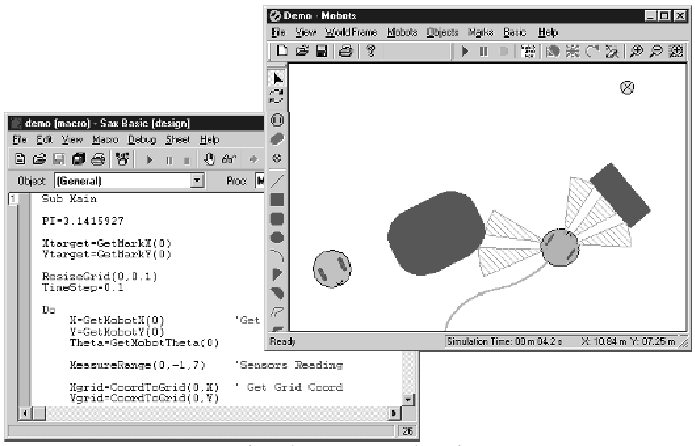

Fig. 5.7 Mobotsim

5.3.4 - Rossum's Playhouse

Rossum's Playhouse (RP1) é um simulador de robôs móveis desenvolvido pelo projeto Rossum [Luc99], totalmente programado em JAVA. Este simulador foi implementando inicialmente para testar projetos para o Trinity College Fire-Fighting Home Robot Contest [Tri02].

O RP1 é uma ferramenta para os pesquisadores que estejam desenvolvendo softwares para o controle de robôs móveis autônomos e pode ser utilizado como plataforma de teste para algoritmos e lógicas de controle. O simulador RP1 tem seu código aberto e é fornecido gratuitamente.

Este simulador possui um modelo de ambiente bidimensional (Fig. 5.8) e implementa somente um tipo de sensor (sensor de distância a laser). A principal vantagem é o fato dele ter o código aberto, possibilitando a implementação de novas características pelo próprio usuário.

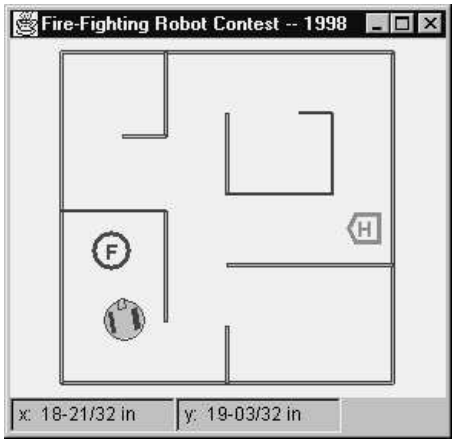

Fig. 5.8 Rossum's Playhouse

5.4 - O Simulador SimRob3D

Depois de estudar vários simuladores existentes, e verificar a impossibilidade de se utilizar algum deles, optou-se por implementar um novo simulador que possuísse todos os recursos necessários para a validação do sistema de controle para robôs móveis proposto nesta dissertação. Como base para definir quais os recursos seriam implementados foram utilizados os princípios apresentados por Torrance [Tor92] (ver item 5.1).

O simulador, chamado de SimRob3D (Simulador de Robôs Móveis em Ambiente Tridimensional), tem como principal característica o fato de se utilizar um ambiente tridimensional para a navegação dos robôs móveis simulados. Este ambiente pode ser modelado em diversos softwares de modelagem tridimensional existentes no mercado (AutoCad, 3D Studio, entre outros), pois o simulador utiliza o formato de arquivo ".3DS" que é um formato bastante conhecido na área de computação gráfica. Este formato de arquivo permite que sejam especificados os diversos elementos de um ambiente (objetos, luzes, texturas), o que resulta em um ambiente com um nível de realismo muito superior aos ambientes utilizados nos simuladores bidimensionais.

O simulador possui diversos modelos sensoriais e cinemáticos, permitindo a configuração de diversos tipos de robôs. Todos os sensores e atuadores interagem com o ambiente tridimensional, tornando a simulação mais realista.

Um característica importante do simulador é a sua modularidade. O controlador é programado separadamente como uma biblioteca dinâmica. O controlador é carregado em tempo de execução, e pode ser implementado na linguagem de preferência do pesquisador.

5.4.1 - Interface

Utilizou-se uma interface gráfica interativa (Fig. 5.9) composta dos seguintes elementos: menu de opções e barra de ferramentas, barra de histórico de operações, janela de informações sobre a simulações, janela do ambiente tridimensional, janela do controlador.

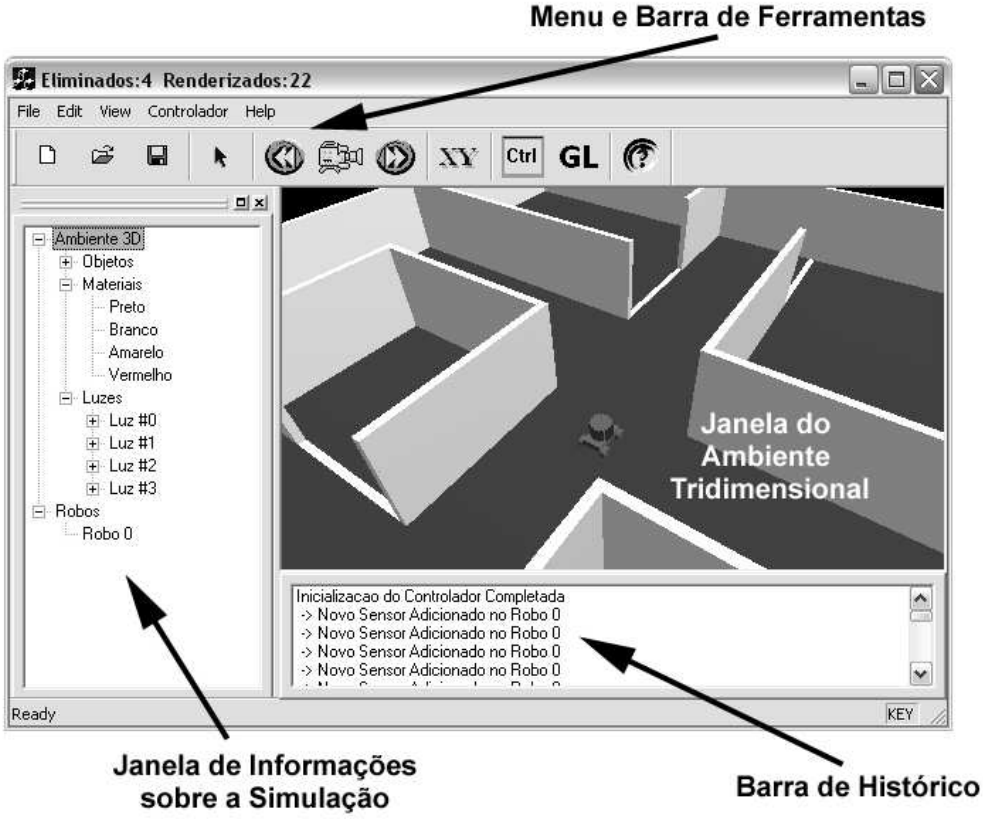

Fig. 5.9 Interface do Simulador

5.4.1.1 - Menu de Opções e Barra de Ferramentas

O menu de opções e a barra de ferramentas do simulador possibilitam o acesso a todas as ferramentas disponíveis para o controle da simulação. A partir do menu ou da barra de ferramentas é possível:

- Acessar a janela de visualização e operação do controlador;
- Obter informações sobre o *driver* de OpenGL;
- Acessar a ferramenta de seleção e movimentação de objetos;
- Alterar os eixos de movimentação do ambiente;
- Ocultar ou mostrar as diversas janelas do simulador;

5.4.1.2 - Barra de Histórico de operações

A barra de histórico permite que o usuário acompanhe de forma textual o histórico de operações executadas durante a simulação. Diversas mensagens a respeito do simulador e do controlador são apresentadas nesta janela.

5.4.1.3 - Janela de Informações sobre a Simulação

A janela de informações sobre a simulação apresenta de forma estruturada os componentes do ambiente e detalhes sobre os robôs inseridos na simulação. As informação são organizadas em forma de árvore, mostrando as informação sobre todos os objetos do ambiente, sobre os materiais, sobre a iluminação e sobre os robôs.

5.4.1.4 - Janela do Ambiente Tridimensional

A janela principal do simulador é a janela do ambiente tridimensional. Nesta janela é visualizada, em um ambiente tridimensional, toda a simulação. A partir desta janela, utilizando-se o mouse, é possível modificar o angulo de visualização do observador, a posição do observador, selecionar e movimentar os objetos.

A movimentação dos objetos é feita em tempo real, e os objetos podem ser movimentados durante as simulações. Isto possibilita que o usuário possa simular um ambiente dinâmico, alterando a posição dos objetos no ambiente.

5.4.1.5 - Janela do controlador

A janela do controlador (Fig. 5.10) é a área gráfica disponibilizada para que o controlador possa apresentar seus dados para o usuário. É nesta janela que o usuário envia comandos para o controlador, tanto pela utilização do mouse como por atalhos do teclado.

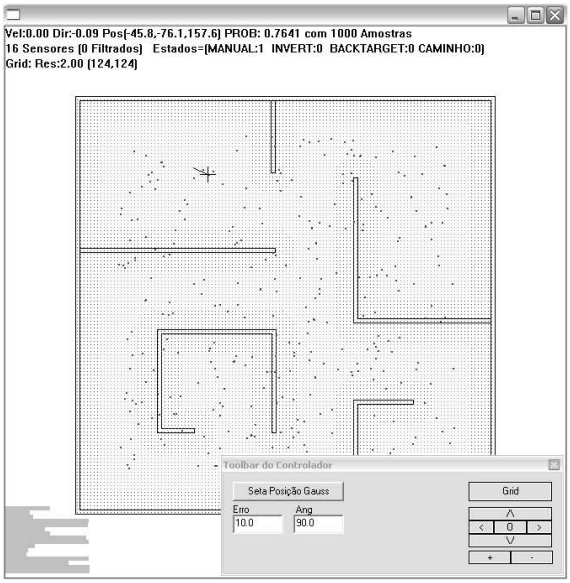

Fig. 5.10 Janela do Controlador

5.4.2 - Implementação do Simulador

O simulador foi implementado em Visual C++ utilizado a biblioteca de classes MFC (Microsoft Foundation Classes) e para o ambiente tridimensional foi utilizada a biblioteca OpenGL.

O Visual C++ possui um ótimo ambiente de desenvolvimento (Visual Studio) que permite de forma visual o implementação e depuração do software. Além disso a biblioteca de classes MFC fornece diversos recursos para o desenvolvimento rápido de aplicações para o sistema operacional Windows.

Para a implementação da visualização do ambiente tridimensional, optou-se pela utilização da biblioteca gráfica OpenGL [Woo98]. O OpenGL é atualmente o padrão internacional para o desenvolvimento de aplicações gráficas tridimensionais, e é implementado em hardware na maioria das placas gráficas aceleradoras. A biblioteca OpenGL permite que se utilize todos os recursos de hardware disponível nos computadores, aumentando sensivelmente a performance da simulação. Apesar de ser uma biblioteca bastante poderosa sua programação é bastante simples, o que acelerou o processo de implementação do simulador.

A seguir são apresentados detalhes sobre a implementação de alguns dos principais componentes internos do simulador.

5.4.2.1 - Controlador

A simulador não possui internamente nenhuma forma de controle da simulação. Todo o controle da simulação, inserção de robôs e sensores, é realizado por um controlador. Este controlador é externo ao simulador, e a interface entre os dois softwares é realizada utilizando-se uma biblioteca dinâmica (DLL).

O simulador disponibiliza uma API (*Aplication Programming Interface*) que fornece diversos recursos para que o controlador se integre ao ambiente da simulação (ver apêndice B).

Apesar do simulador ser implementado em Visual C++, a DLL do controlador pode ser programada em qualquer linguagem disponível capaz de cria bibliotecas dinâmicas para o sistema operacional Windows. Isto possibilita que o usuário do simulador utilize a sua linguagem de preferência para programar o controlador.

Esta abordagem utilizando um controlador externo, amplia a modularidade do simulador, e permite que um número maior de usuários possa utilizar seus recursos.

5.4.2.2 - Laço de Controle

O laço de controle principal do simulador foi separado em duas *threads* com funções distintas, *thread* de controle e *thread* de visualização.

A *thread* de controle é responsável por atualizar os estados internos do simulador e os estados internos do controlador. Esta *thread* é chamada a cada período de tempo definido pelo usuário. Para garantir uma maior precisão no intervalo de tempo, foi utilizado um recurso de multimídia disponível no sistema operacional Windows: os temporizadores multimídia. Este recurso permite que uma aplicação agende eventos periódicos, e possui uma ótima precisão, com uma resolução de cerca de 1ms.

A *thread* de visualização tem a função de apresentar os dados da simulação para o usuário. Esta *thread* é chamada nos períodos de inatividade (*idle*) do usuário e da *thread* de controle. Desta forma a tarefa de visualização gráfica da simulação nunca interfere com os mecanismos internos de simulação, ou com as operações executadas pelo usuário. Além disso os algoritmos de visualização são robustos o suficiente para que consigam executar suas tarefas mesmo utilizando dados incompletos ou inválidos.

Esta separação garante que o passo da simulação será executado em um período de tempo exato, independentemente dos processos de visualização gráfica, que geralmente são mais lentos.

5.4.2.3 - Modelo Tridimensional de Ambiente

O modelo de ambiente utilizado pelo simulador é tridimensional. Este ambiente é fornecido pelo usuário e pode ser construído em diversos softwares disponíveis para modelagem tridimensional (3D Studio MAX, Maia, entre outros). O simulador SimRob3D utiliza o formado de arquivo .3DS para carregar o ambiente tridimensional, que é um formato muito conhecido na área de computação gráfica. Originalmente criado pela Autodesk para ser utilizado no software 3D Studio, este formato de arquivo permite que sejam definidos objetos tridimensionais, materiais, texturas e iluminação. O simulador SimRob3D faz uso de todos esses recursos para visualizar o ambiente tridimensional.

No arquivo .3DS os objetos são modelados utilizando-se somente faces triangulares, o que simplifica uma série de cálculos necessários para a simulação dos sensores, e facilita também o armazenamento em memória e a visualização utilizando OpenGL.

Para otimizar a velocidade da visualização alguma técnicas da computação gráfica foram utilizadas. Uma das primeiras técnicas implementadas foi a de *Backface Culling* [Woo98], pois é uma técnica disponibilizada pela biblioteca OpenGL. Nesta técnica as faces triangulares somente são visualizadas se estiverem com a sua parte visível (parte frontal) direcionadas para o visualizador.

Outra técnica que foi implementada para acelerar a visualização do ambiente foi a *Frustum Culling* [Fol94]. Nesta técnica os objetos que não estiverem dentro do campo de visão do observador não são renderizados, diminuindo a quantidade de faces que necessitam ser apresentadas para o usuário pelo OpenGL.

A utilização de um ambiente tridimensional torna as simulações mais complexas matematicamente, mas ao mesmo tempo possibilita simulações mais realistas, principalmente do ponto de vista sensorial. Como trabalho futuro, estuda-se a possibilidade de se implementar sensores visuais (câmeras de vídeo) simulados, que tirem proveito da capacidade do simulador de trabalhar com ambientes foto realistas (iluminação e texturas).

5.4.2.4 - Modelos Sensoriais

Nesta primeira versão do simulador foram implementadas duas categorias de sensores: sensores de distância e encoders.

Foi utilizado um modelo simplificado de **sensor de distância** para aumentar a performance. Variando somente o erro médio, número de raios, ângulo de abertura, distância mínima e máxima, podemos utilizar um mesmo modelo matemático para representar 3 diferentes tipos de sensores: Sonar, Laser e Infravermelho.

O modelo utilizado para simular os sensores de distância é estocástico. Diversos raios são traçados da posição do sensor em direção a sua orientação. Uma técnica de *RayCast* [4] é utilizada para traçar os raios. O número de raios traçados em cada sensor é configurado pelo usuário, onde quanto maior o número de raios, maior será a precisão e maior será o custo computacional.

Os raios são distribuídos aleatoriamente dentro do cone do sensor, se algum deles colidir com alguma das faces tridimensionais do ambiente, a distância até o ponto de colisão é informado. O resultado final é a menor distância encontrada entre todos os raios, mais um erro médio informado pelo usuário que configurou o sensor.

Os **encoders** são os sensores que informam a velocidade ou a posição de determinado dispositivo. Dois sensores deste tipo foram implementados: Sensor odométrico e sensor de ângulo.

O sensor odométrico informa a velocidade dos motores do robô móvel. Nesta velocidade é adicionado um erro gaussiano pré-determinado pelo usuário, dependendo do tipo de motor utilizado. O sensor de ângulo informa a posição da rodas dianteiras do robô móvel, este tipo de sensor só é utilizado pelo robôs configurados com a cinemática Ackerman, e tem uma funcionalidade semelhante a do sensor odométrico, também adicionando um erro gaussiano ao valor calculado.

5.4.2.5 - Modelos Cinemáticos

O simulador disponibiliza dois modelos cinemáticos que podem ser utilizados na configuração de um robô móvel: o modelo diferencial e o modelo Ackerman. Para cada um destes modelos o usuário define uma série de atributos para compor um modelo completo.

No **modelo diferencial** os atributos cinemáticos principais são:

- raio do robô;
- largura do eixo;
- raio das rodas;
- precisão dos motores;

No **modelo Ackerman** os atributos são:

- largura dos eixos;
- comprimento entre os eixos;
- raio das rodas;
- rotação máxima das rodas dianteiras;
- precisão do motor de tração;
- precisão do motor de direção;

[4] RayCast é uma técnica da computação gráfica que é utilizada normalmente em algoritmos para remoção de superfícies ocultas. Este método tenta mimetizar os efeitos físicos associados com a propagação de raios de luz [Fol94].

É importante lembrar que, como visto no item 5.2.2, a velocidade é considerada constante. Os efeitos da aceleração foram ignorados. Quando se trabalha com robôs pequenos o impacto destas simplificações pode ser ignorado sem muitas conseqüências nas simulações.

Apesar do simulador SimRob3D possibilitar a configuração de um modelo de grande porte, o escopo deste trabalho se restringe a robôs móveis de pequeno porte, onde tais simplificações podem ser aplicadas.

5.4.2.6 - Gerador de Números Pseudo Aleatórios

O simulador SimRob3D implementa um gerador de números pseudo aleatórios (GNPA). Este GNPA é utilizado tanto pelos modelos internos do simulador, como também é disponibilizado pela API para que seja utilizado pelo controlador (ver apêndice B).

O algoritmo do GNPA utilizado foi proposto por Kirkpatrick & Stoll em 1981 e se chama R250 [Kir81]. Ele permite períodos muito longos de geração de números aleatórios sem que o ciclo de repetições se reinicie, o que é uma característica muito desejada em simulações que trabalham com um grande número de chamadas ao GNPA, tal como as simulações Monte Carlo. Além disso possui uma boa performance, permitindo que as simulações sejam realizadas em tempo real.

5.4.3 - Disponibilidade do Simulador

O simulador SimRob3D se encontra disponível para download no seguinte endereço:

http://ncg.unisinos.br/robotica/simulador

O simulador ainda se encontra em fase de desenvolvimento, mas já possui a maior parte de sua funcionalidade implementada. No pacote disponível para download são fornecidos modelos de dois ambiente utilizados na dissertação (Trinity e PIPCA), bem como um controlador de exemplo (com o código fonte).

Capítulo 6 - Arquitetura Proposta

Com base nos estudos realizados, foi proposta uma arquitetura de controle híbrida, integrando em uma única estrutura os principais métodos utilizados atualmente no controle de robôs móveis autônomos.

A idéia principal é integrar as melhores características de cada método apresentado nos capítulos 2,3 e 4 em uma arquitetura de controle robusta, que possa ser facilmente adaptada a diferentes problemas que um robô móvel autônomo possa enfrentar, tendo como pré-requisitos a aplicação deste sistema em um ambiente interno plano (indoor) e com um mapa fornecido a priori.

O controle do robô móvel foi separado em 3 camadas: camada vital, camada funcional e camada deliberativa, sendo cada uma delas responsável pelo controle reativo, controle de execução de planos, e tarefas de planejamento a longo prazo, respectivamente. Este tipo de sistema pode ser visto nos trabalho realizados por Gat [Gat92] com a arquitetura ATLANTIS (ver item 2.5.6), e por Bonasso et al. [Bon97] com a arquitetura 3T.

Para fornecer uma base sólida para a execução das tarefas desempenhadas pelas camadas de controle, foi integrado na arquitetura um módulo localizador. Este módulo localizador deve fornecer uma estimativa da posição do robô em relação a um mapa utilizando os dados sensoriais. *Um módulo localizador é uma parte essencial de uma arquitetura de controle para robôs móveis autônomos*, e na arquitetura proposta neste trabalho ele desempenha um papel de destaque.

A forma como o ambiente é representado internamente no sistema de controle, determina a sua precisão e performance. Cada uma das principais abordagens para o controle de robôs móveis autônomos utiliza, e melhor se adapta, a uma determinada representação de ambiente. Como a arquitetura proposta neste trabalho integra diversas abordagens de uma forma híbrida, a representação de ambiente utilizada é composta de diversas camadas: camada poligonal, camada matricial e camada topológica/semântica.

Para permitir a comunicação entre os vários componentes da arquitetura de controle é disponibilizada uma área de memória compartilhada. Através do uso deste depósito central de informações, os diversos módulos podem trocar informações vitais para o funcionamento do robô móvel autônomo. Este tipo de abordagem pode ser vista no trabalho de Hayes & Roth [Hay85], com a arquitetura Blackboard.

Seguindo a estrutura da arquitetura de controle proposta (item 6.1), foi implementado um sistema de controle. Este sistema foi aplicado na prática através do desenvolvimento de um simulador (SimRob3D) que teve como objetivo permitir que fosse validada a arquitetura proposta através de simulações.

6.1 - Arquitetura de Controle para Robôs Móveis Autônomos

A arquitetura de controle é organizada a partir de um sistema de 3 camadas, integrando também um localizador para fornecer a base funcional para as camadas de

controle, uma memória compartilhada para facilitar a comunicação entre os diferentes componentes, e uma representação interna do ambiente (Fig. 6.1).

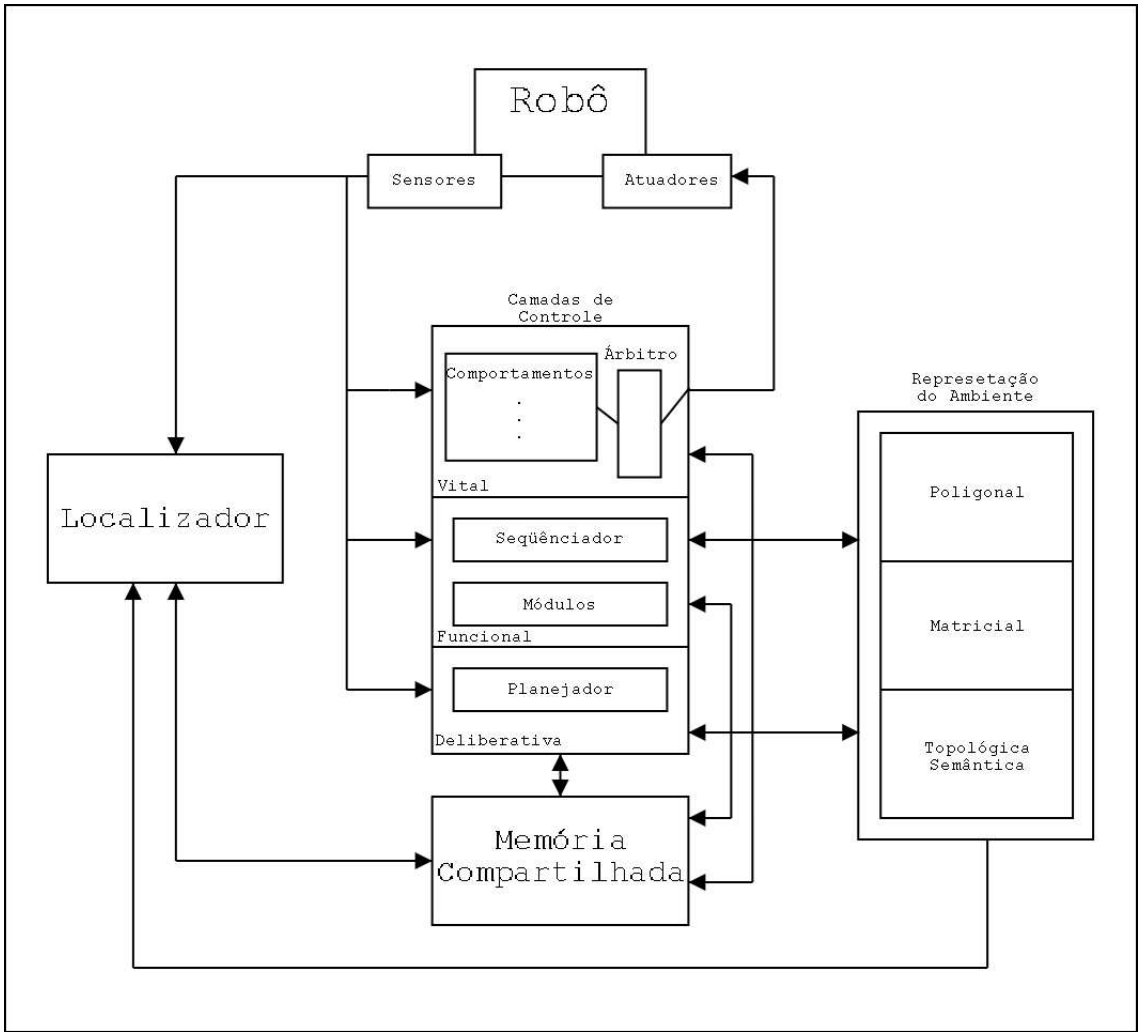

Fig. 6.1 Diagrama da Arquitetura de Controle Robótica

A seguir, cada um dos componentes principais são apresentados, bem como as relações entre os diversos módulos.

6.1.1 - Memória Compartilhada

A comunicação entre os diversos módulos de controle é realizada através de uma área de memória comum, chamada de memória compartilhada. A memória compartilhada é um depósito central de informações que é utilizado pelos módulos do sistema de controle de forma independente. Esta abordagem foi inspirada na arquitetura de controle Blackboard (Hayes&Roth) [Hay85].

Cada informação inserida na memória compartilhada só pode ser modificada pelo módulo que a criou, mas pode ser lida por todos os módulos. Diferentes tipos de dados podem ser disponibilizados: localização do robô, planos, estados, ou qualquer tipo de dado que for necessário na comunicação entre os módulos. Por exemplo: o módulo posicionador disponibiliza na área de memória compartilhada informações

sobre a localização atual do robô (posição, orientação, nível de certeza, dispersão). Diversos módulos que necessitam desta informação, acessam a área de memória compartilhada para obter a melhor estimativa sobre a localização do robô.

Quais informações serão disponibilizadas e quais módulos irão acessar cada informação, são definidos no momento em que os módulos são desenvolvidos.

6.1.2 - Módulo Localizador

A função do módulo localizador é, a partir das entradas sensoriais recebidas do robô e da representação interna (mapa) do ambiente, estimar uma posição no ambiente real. A estrutura da estimativa depende do tipo de ambiente onde o robô vai operar, no caso do sistema de controle proposto neste trabalho, que utiliza a localização Monte Carlo e atua em um ambiente interno plano, a estimativa de posição apresenta a seguinte forma: ((x,y,ang),probabilidade, dispersão).

Este é o principal módulo da arquitetura de controle, ele fornece a base para que todos os outros componentes sejam capazes de executar suas tarefas. O módulo localizador é executado em paralelo a todos os outros componentes, e constantemente monitora os dados sensoriais e a representação interna para estimar uma posição, e a sua operação não pode ser inibida.

A estimativa de posição é armazenada na memória compartilhada para que todos os outros componentes da arquitetura de controle possam ter acesso. Uma vez que esta estimativa de posição é constantemente atualizada, os componentes que a utilizarem devem monitorar suas alteração constantemente para que não sejam utilizadas estimativas desatualizadas.

6.1.3 - Camadas de Controle

6.1.3.1 - Camada Vital

A camada vital é composta de um ou mais processos implementando estratégias de controle do tipo feedback, relacionando fortemente as entradas sensoriais com as saídas para os atuadores. Estes processos são chamados "comportamentos" e executam tarefas simples, tais como:

- seguir muros;
- se mover até um destino;
- evitar colisões;
- passar por portas;
- vagar, entre outras;
- seguir em uma direção;
- seguir um alvo;

A composição e seqüênciamento de vários destes comportamentos primitivos pela camada funcional, produz comportamentos mais complexos capazes de executar tarefas mais complicadas.

Cada comportamento primitivo pode ser visto como uma "reação motora", reagindo diretamente aos estímulos do ambiente. Os comportamentos da camada vital também são responsáveis pela integridade física do robô. O processo que evita colisões com os obstáculos do ambiente é um exemplo claro de comportamento vital para manter a integridade física do robô, bem como vital para a execução das tarefas seqüenciadas pela camada funcional.

É importante que cada comportamento da camada vital seja executado em um tempo constante, e que este tempo seja curto o suficiente para permitir que todos os comportamentos possam detectar e reagir as mudanças do ambiente. A quantidade de mudanças no ambiente determina a velocidade que os comportamentos devem ser executados. Um ambiente dinâmico com alterações constantes na sua estrutura e com um grande número de obstáculos móveis, necessita de comportamentos projetados para execução simples e rápida. Já um ambiente mais estável e com poucas mudanças, permite que cada comportamento seja mais complexo e com uma velocidade de execução menor. O tipo de ambiente determina o conjunto de comportamentos que será utilizado na camada vital.

Cada comportamento da camada vital não deve interferir com a execução de outro comportamento que esteja sendo executado nesta mesma camada. Os comportamentos devem operar em paralelo, a responsabilidade de seqüênciar a execução dos comportamentos, inibindo ou não suas saídas, é da camada funcional.

Outro componente da camada vital é o árbitro. O árbitro tem como função principal unificar as saídas dos diversos comportamentos em um comando único para os atuadores. A partir do seqüênciamento determinado pela camada funcional, que indica quais comportamentos serão inibidos, é tarefa do árbitro executar as inibições. Além disso o árbitro também tem a função de unificar saídas que modifiquem o mesmo atuador.

Por exemplo: os comportamentos "*Desviar de Obstáculos*" e "*Seguir o Alvo*" modificam ao mesmo tempo o atuador de direção do robô. Na tarefa de seguir um plano, a camada funcional precisa manter os dois comportamentos ativos, e muitas vezes estes dois comportamentos indicam direções diferentes em suas saídas. A tarefa do árbitro nesta situação é integrar as duas saídas utilizando um política de pesos programada pelo usuário. A saída indicada pelo comportamento "*Desviar de Obstáculos*" tem um peso maior, quando o árbitro soma as duas saídas a direção do robô resultante tem uma tendência maior a desviar do obstáculo, mas ao mesmo tempo não se distância demais do seu objetivo. Em caso de bloqueio, a camada funcional implementa um módulo que monitora constante a posição do robô em relação aos obstáculos, e se for detectando um bloqueio do robô este módulo envia uma interrupção para a camada deliberativa requisitando um novo plano.

6.1.3.2 - Camada Funcional

A camada funcional é composta de diversos módulos que interagem entre si, executando diversas funções de integração entre os componentes do sistema de controle.

Uma das funções da camada funcional é selecionar quais comportamentos primitivos a camada vital executará em um determinado instante de tempo, e fornecer parâmetros para estes comportamentos. Fornecendo informações e alterando a seqüência de execução destes comportamentos, o robô consegue executar as tarefas de alto nível planejadas pela camada deliberativa.

O seqüênciamento é executado através da inibição das saídas dos comportamentos. Nenhum módulo da camada vital deve atuar diretamente no controle dos atuadores do robô. Ao invés disso, os módulos da camada funcional fornecem informações para os comportamentos da camada vital que estiverem ativos, e inibem as saídas dos comportamentos que precisam ser desativados.

A seleção de quais comportamentos devem ser ativados ou desativados em um determinado instante de tempo é geralmente executada através de um Autômato[5]. O estado inicial do autômato é determinado pelo plano fornecido pela camada deliberativa, e uma vez que o autômato é acionado cada estado pode consultar o plano para determinar o próximo passo de controle. Se a camada funcional determina que o objetivo não pode ser atingido, ela aciona uma interrupção e faz com que a camada deliberativa calcule um novo plano.

Outra tarefa da camada funcional é monitorar o estado do ambiente, coletando informações para auxiliar o localizador, atualizar as representações internas do ambiente, e fornecer parâmetros para outros componentes do sistema de controle.

6.1.3.3 - Camada Deliberativa

Na camada deliberativa são executadas as tarefas de planejamento de alto nível e de longo prazo. Dada a condição corrente (atual) do robô e uma condição final fornecida pelo usuário ou por uma interrupção da camada funcional, a camada deliberativa deve planejar a seqüência de ações que devem ser executadas para que o robô atinja seu objetivo principal.

A seqüência de ações é armazenada na memória compartilhada na forma de um plano, cada passo deste plano indica um sub-objetivo de alto nível. A camada funcional utiliza o plano para ativar os comportamentos da camada vital, decidindo quais comportamentos são mais adequados para atingir o próximo sub-objetivo.

Os módulos da camada deliberativa necessitam de mais recursos computacionais, e geralmente consomem bastante tempo para serem executados. Mas normalmente as tarefas da camada deliberativa somente são executadas no início de cada nova operação do robô, ou quando a camada funcional determina que não foi capaz

[5] Entende-se por autômatos, qualquer mecanismo auto-operante que processa informações de acordo com certas regras, sendo capaz de avaliar seu estado e reagir a ele, passando de um estado a outro de modo a realizar uma tarefa.

de executar o plano fornecido e envia uma interrupção para a que um novo plano seja calculado.

6.1.4 - Representação do Ambiente

O módulo de representação do ambiente armazena as diversas informações sobre o ambiente de operação do robô. Ele é organizado em diversas camadas, cada uma destas camadas de representação se apresenta mais adequada para uma determinada tarefa executada pelo sistema de controle.

6.1.4.1 - Camada Poligonal

A camada poligonal é uma representação vetorial do ambiente, composta de polígonos. Esta representação deve ser fornecida inicialmente pelo usuário, na forma de um mapa do ambiente onde o robô vai operar. Ela também é atualizada pela camada funcional, a medida que o robô explora o ambiente e percebe novos obstáculos estáticos, ou a ausência de um obstáculo previamente fornecido.

Esta camada de representação é utilizada principalmente pelo módulo localizador, e pelos módulos da camada funcional.

6.1.4.2 - Camada Matricial

A camada matricial é gerada a partir da camada poligonal, e é composta de células de um tamanho fixo que representam posições do ambiente. Estas células são dispostas em uma matriz.

Uma representação matricial facilita as operações de planejamento pela camada deliberativa. Cada célula armazena diversas informações sobre uma determinada posição do ambiente (presença de um obstáculo, distância até o objetivo, etc.), o que auxilia na determinação de trajetórias pelos módulos planejadores.

6.1.4.3 - Camada Topológica / Semântica

Esta camada de representação do ambiente é composta por diversas regiões definidas no ambiente, as relações topológicas entre estas regiões e informações semânticas sobre cada região.

As informações topológicas entre as regiões são representadas na forma de um grafo. Este grafo é utilizado pela camada deliberativa para auxiliar na criação dos planos. Estas informações são estáticas, e fornecidas pelo usuário em conjunto com as informações da camada poligonal.

6.2 - Desenvolvimento do Sistema de Controle

Para implementar o sistema de controle foi utilizado o Visual C++ 6.0. Como o sistema de controle vai ser utilizado em um robô simulado através do SimRob3D, é necessário que o controlador seja implementado no formato de biblioteca dinâmica (DLL). O simulador é então configurado para utilizar esta biblioteca dinâmica como o controlador de robôs no ambiente simulado.

6.2.1 - Objetivos do Sistema de Controle

O objetivo do sistema de controle implementado é possibilitar que um robô móvel autônomo seja capaz de navegar em um ambiente dinâmico, com obstáculos móveis. Este robô móvel autônomo é configurado com um modelo cinemático Ackerman, encoders de velocidade e direção, e sensores de distância. O ambiente onde o robô deve navegar é interno (casas, apartamentos, escritórios, fábricas, etc.), com o piso plano.

O robô deve ser capaz de se localizar neste ambiente, utilizando os dados sensoriais e um mapa do ambiente. Uma vez localizado, o mapa do ambiente não precisa ser necessariamente completo, pois o robô deve ser capaz de se manter localizado mesmo na presença de obstáculos inesperados, que não estejam modelados no mapa. Quando o robô móvel detectar algum obstáculo estático que não esteja modelado no mapa, ele deve ser capaz de atualizar o mapa inserindo este novo obstáculo, ou removendo do mapa um obstáculo que não existe no ambiente real.

O sistema de controle deve ser capaz de fazer com que o robô navegue de uma posição inicial até uma posição indicada pelo usuário dentro do mapa, sem colidir com os obstáculos do ambiente, mesmo que os obstáculos sejam móveis. O robô deve também ser capaz de detectar quando um objetivo é inatingível.

O sistema de controle deve ser capaz de tirar proveito de informações topológicas e semânticas fornecidas pelo usuário para otimizar a navegação dentro do ambiente.

Com base nestes objetivos foi implementado o sistema de controle **COHBRA (Controle Híbrido de Robôs Autônomos)** utilizando a arquitetura proposta neste trabalho. Foi dada uma atenção especial no desenvolvimento do módulo localizador, que consideramos o componente principal do sistema de controle e um dos focos principais deste trabalho. Uma vez que o robô móvel autônomo possua uma boa estimativa da sua posição real no ambiente, as tarefas de navegação se tornam mais simples e precisas.

A seguir os principais componentes do sistema de controle COHBRA são apresentados com maiores detalhes sobre sua implementação, funcionalidade e interação com os demais componentes.

6.2.2 - Representação do Ambiente

Para a representação interna do ambiente foram utilizadas todas as camadas definidas na arquitetura de controle: camada poligonal, camada matricial e a camada topológica/semântica.

A camada de representação poligonal é utilizada pelo módulo localizador; pelos módulos da camada de controle funcional para atualizações de possíveis alterações no ambiente real; e também para gerar a camada de representação matricial. O usuário fornece um arquivo .DXF (que é um formato padrão para troca de dados entre softwares de CAD, tais como AutoCad, DataCad, etc.), e o sistema de controle importa os dados deste arquivo para o seu formato interno de representação poligonal.

A camada de representação matricial é gerada a partir da camada poligonal. O usuário deve informar durante a configuração do sistema de controle, o tamanho desejado de cada célula e a distância limite que cada célula deve estar de um polígono para ser considerada uma célula ocupada. Se a distância de uma célula do mapa matricial até um determinado polígono for menor ou igual ao limite configurado, ela é marcada como ocupada.

A representação matricial é utilizada somente pela camada de controle deliberativo, para planejar a trajetória do robô móvel autônomo até o seu objetivo. A precisão do plano depende da tamanho da célula definido pelo usuário. Já a distância limite serve para estabelecer uma região de segurança para o robô, desta forma a camada de controle deliberativo não traça uma trajetória que se aproxime demais dos obstáculos.

A representação topológica/semântica também deve ser fornecida pelo usuário através de dois arquivos: o arquivo de regiões, e o arquivo de relações topológicas/semânticas. O arquivo de regiões delimita áreas que se sobrepõe sobre o mapa poligonal, e também é fornecido no formato .DXF. Cada área define uma determinada região funcional do ambiente, por exemplo:sala da secretaria, sala de convivência, porta, mobília leve, mobília pesada. Cada região recebe uma identificação numérica única.

O arquivo de relações topológicas/semânticas, que é fornecido no formato texto, possui dois tipos de informação: as relações topológicas entre as regiões, e as informações semânticas sobre cada região. As relações topológicas são fornecidas através de uma matriz de adjacência. As informações semânticas são definidas para cada região, relacionando o identificador numérico com um determinado tipo: sala, porta, mobília leve ou mobília pesada. Cada tipo pode conter uma série de estados, no sistema de controle atual a porta é o único tipo que faz uso de estados (aberta/fechada).

A forma como cada uma destas representações de ambiente são utilizadas pelos diferentes módulos do sistema de controle são apresentadas a seguir, em conjunto com as descrições dos demais módulos.

6.2.3 - Localizador

A arquitetura de controle determina que um dos componentes principais para o sistema de controle é o módulo localizador. O tipo de localizador utilizado na implementação deste módulo depende muito das necessidades específicas do sistema de controle.

Dada a configuração do robô e as características do ambiente em que ele atua, optou-se pela utilização da técnica de localização Monte Carlo (ver item 4.3.8) para implementar o módulo de localização do sistema de controle.

6.2.3.1 - Localização Monte Carlo

A técnica de localização Monte Carlo [Fox99b] possui uma série de vantagens: ela utiliza menos recursos computacionais do que a maioria das outras técnicas; concentra os recursos utilizados nas áreas de maior interesse para a localização; por ser uma técnica probabilista fornece mais informação além da localização do robô (a certeza sobre a localização, por exemplo).

Um dos principais motivos da escolha do método de localização Monte Carlo para a implementação do módulo localizador do sistema de controle COHBRA, é a sua capacidade de resolver os 3 grandes problemas da localização: localização local, localização global, e relocalização.

Na localização local (ver item 4.1), o localizador Monte Carlo é capaz de manter uma posição correta a partir de uma posição inicial com um nível aceitável de erro, sem a necessidade de uma função externa de recalibragem. O localizador Monte Carlo é capaz de se localizar globalmente, não necessitando de uma informação inicial sobre a sua posição. Esta técnica de localização também é capaz de se relocalizar, conseguindo detectar quando a posição atual, aparentemente correta, não reflete a posição real do robô.

A localização Monte Carlo utiliza um conjunto de N partículas, ou amostras, que são distribuídas pelo espaço de posições possíveis no mapa do ambiente. Cada partícula representa uma possível posição do robô no mapa em relação ao ambiente real que é percebido pelos sensores. Cada partícula é composta por 4 valores: Posição (X,Y), Direção (Ang), e Probabilidade(Prob). O espaço de estados de posição é tridimensional (X,Y,Ang).

A forma como estas partículas são distribuídas inicialmente, depende das informações disponíveis. Se a posição é conhecida aproximadamente, as partículas são distribuídas em torno desta posição, com uma certa margem de erro. Se a informação disponível indica que o robô está em uma determinada sala, as partículas são distribuídas uniformemente dentro desta sala no mapa. Se a posição do robô é totalmente desconhecida, as partículas são distribuídas uniformemente por todo o mapa. A quantidade de partículas necessárias vária de acordo com a complexidade e ambigüidade do ambiente, e da precisão desejada. Mais detalhes podem ser vistos nos experimentos práticos apresentados no Cap. 7.

Quando se utiliza a localização Monte Carlo é necessário possuir os modelos cinemáticos e sensoriais utilizados no robô que se deseja localizar. Estes modelos são utilizados nas duas fases da técnica de localização Monte Carlo: fase de movimentação e fase de leitura sensorial.

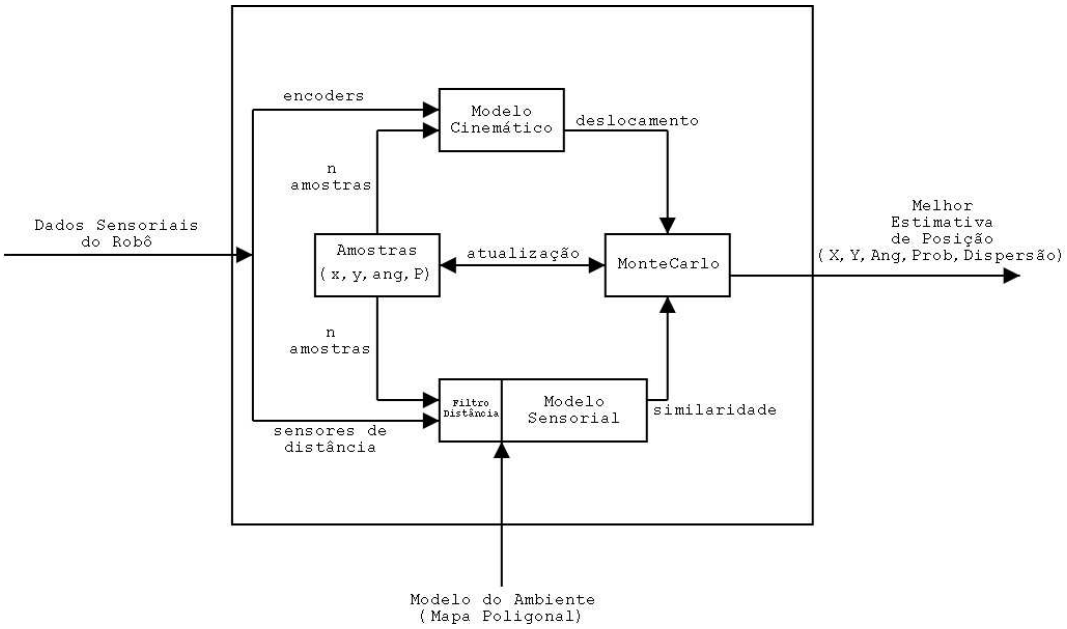

Fig. 6.2 Diagrama do Localizador Monte Carlo

Na fase de movimentação, o sistema de controle recebe dos encoders do robô móvel autônomo a informação de quanto o robô se moveu e em que direção. Utilizando esta informação o modelo cinemático (Fig. 6.2) é utilizado para movimentar as partículas de acordo com a movimentação do robô. Os possíveis erros na leitura dos encoders também são considerados durante esta fase. Para cada leitura dos encoders este processo é novamente realizado.

Na fase de leitura sensorial cada partícula é comparada com a leituras reais dos sensores do robô. Com base no modelo sensorial (Fig. 6.2) cada partícula simula as leituras dos sensores em sua posição, baseadas na representação poligonal do ambiente. As leituras obtidas para as possíveis posições estimadas são comparadas com as leituras reais obtidas do robô utilizando a seguinte função de similaridade:

$$W_p = \sqrt{\sum_{s=1}^{NS} (SR_s - SS_s)^2}$$

onde,

Wp = similaridade da partícula p
NS = número de sensores;
SR = sensor real;
SS = sensor simulado;

O valor de **Wp** indica o quanto a leitura sensorial estimada da partícula **p** é similar com as leituras reais dos sensores do robô. O próximo passo é normalizar o valor de **Wp** de todas as partículas para que elas fiquem no intervalo de [0,1].

Uma vez que todos os valores de **Wp** do conjunto de partículas estejam normalizados é necessário ordenar o conjunto em ordem decrescente de similaridade. Com o conjunto de partículas ordenado é necessário calcular a *similaridade acumulada* **WA** de cada partícula através da função:

$$WA_p = \sum_{i=p+1}^{NP} W_i$$

onde,

NP = número de partículas;
WAp = similaridade acumulada da partícula p;

A similaridade acumulada é o somatório de todas as similaridades das partículas menores. Também é necessário calcular o somatório de todos os **W** das partículas. Este valor é armazenado em **TW**. Os valores **WA** e **TW** são necessários no passo de reamostragem.

Na reamostragem das partículas, o conjunto antigo é descartado e um novo conjunto de partículas é criado, selecionando do conjunto antigo as partículas com mais probabilidade de serem a posição real do robô móvel autônomo.

A seleção é realizada gerando-se um valor aleatório R entre 0(zero) e TW, e encontrando no conjunto de partículas, a partícula com o maior WA que seja menor do que R. Esta partícula é então inserida no novo conjunto. E novamente o conjunto de partículas é ordenado em ordem decrescente de similaridade.

Neste ponto a primeira estimativa de posição foi gerada. A primeira partícula do conjunto, aquela com a maior similaridade, é considerada então a melhor estimativa atual com **Probabilidade = Similaridade**. Todo o processo é recursivo, a cada leitura sensorial este passo é novamente realizado, quanto maior o número de leituras sensoriais melhor será a estimativa de posição.

Até aqui o módulo localizador é capaz de realizar uma localização local ou global, ele ainda não é capaz de se relocalizar. Para isso foi necessário implementar um segundo passo de **reamostragem**. Thrun et al. [Fox99b] sugere que é necessária a adição de um pequeno número de amostras aleatórias, uniformemente distribuídas após cada passo, para auxiliar o robô a se relocalizar. Baseando-se nesta reamostragem aleatória sugerida por Thrun et al., foi criado um mecanismo de relocalização que leva em conta a probabilidade da melhor estimativa de posição.

Se a probabilidade da melhor estimativa de posição estiver abaixo de um limite preestabelecido, uma porcentagem das piores partículas do conjunto é remostrada aleatoriamente de forma uniforme no mapa. Isto garante que o robô consiga se relocalizar. O limite para a reamostragem e a quantidade reamostrada são definidas pelo usuário que configura o controlador.

Outro problema comum ao métodos de localização probabilista é a ambigüidade das estruturas que compõem o ambiente. Diversas posições em um mesmo ambiente

podem ser, do ponto de vista sensorial, idênticas. Isto faz com que o robô móvel autônomo, durante uma localização global, encontre mais de uma posição com alta probabilidade de ser a posição real. Quando isso ocorre as camadas de controle ainda não podem considerar o robô localizado, apesar de que a probabilidade da posição estimada é alta.

Foi necessário então encontrar uma forma de detectar quando a localização do robô é ambígua. A utilização de uma posição estimada apenas baseando-se na probabilidade desta posição não é suficiente. Para completar a informação de posição estimada, foi adicionada uma informação de dispersão.

A **dispersão** é a média da distância aproximada entre todas as partículas com similaridade maior do que um limiar preestabelecido. Quanto maior a dispersão, mais ambíguo será o conjunto de partículas. As camadas de controle devem levar em conta esta informação na hora de comandar o robô. O robô somente pode ser considerado localizado quando a dispersão das partículas for "baixa", um valor limite para definir o que é uma dispersão "baixa" deve ser configurado para cada tipo de ambiente onde o robô for operar.

Uma vez que todos os passos necessários para a localização sejam executados, o módulo localizador disponibiliza na área de memória compartilhada as informações sobre a melhor estimativa de posição do robô móvel autônomo. As informação disponibilizadas são: **Posição(X,Y,Ang), Probabilidade e Dispersão**.

6.2.3.2 - Filtro de Distância

Um dos problemas da localização Monte Carlo é que se assume que o ambiente é estático, que a representação do ambiente corresponde ao ambiente real e que não existem obstáculos móveis. Uma situação deste tipo dificilmente é encontrada em um ambiente real, por esse motivo foi necessária a utilização de certos recursos para contornar este problema.

Uma abordagem para tentar solucionar este problema é proposta por Fox [Fox98], que utiliza uma técnica de filtragem para ignorar certas leituras dos sensores quando estas não representarem uma leitura esperada. Fox propôs duas técnicas, filtro de entropia e filtro de distância (para mais detalhes ver item 4.4). A técnica utilizada na implementação do localizador Monte Carlo foi a de filtro de distância, pois ela se adapta melhor com sensores de distância utilizados no robô.

Com o filtro de distância, todas a partículas com probabilidade maior do que um certo limite (geralmente alto: prob > 0.95), são selecionadas para filtragem. Quando uma partícula é selecionada para filtragem, os valores dos sensores simulados são comparados com os valores dos sensores reais. Se a diferença entre algum destes sensores for menor do que um certo limite, o sensor é ignorado durante o cálculo de similaridade (ver Fig. 6.2). Isto significa que o sensor que esta detectando um obstáculo inesperado, não é utilizado no processo de localização do robô.

Além de permitir que o robô móvel autônomo se localize em um ambiente dinâmico, o filtro de distância auxilia outros módulos do sistema de controle. Ele

disponibiliza na área de memória compartilhada as informações sobre os sensores filtrados, assim permitindo que se detecte novos obstáculos pois um sensor filtrado indica um diferença entre a representação interna e o ambiente real. Estes dados são então utilizados pela camada de controle funcional para atualizar a representação interna do ambiente.

6.2.4 - Camada Vital

Na camada vital foram implementados 5 comportamentos primitivos, tendo como objetivo capacitar o robô a seguir as trajetórias calculadas pela camada deliberativa; auxiliar o módulo localizador; e manter a integridade física do robô. Ao seguir a trajetória o robô móvel autônomo deve ser capaz de desviar dos obstáculos inesperados, tanto estáticos como móveis.

6.2.4.1 - Comportamentos

A seguir são descritos os 5 comportamentos da camada vital: *Parar, Vagar, Desviar de Obstáculos, Ir em direção ao Alvo, Inverter Direção*. As relações entre os comportamentos são gerenciadas através do **árbitro**, descrito na seqüência.

O comportamento *Parar* é o mais simples, e tem basicamente como saída comandos para que os atuadores não sejam acionados, mantendo o robô parado. É o comportamento inicial seqüênciado pela camada funcional.

O comportamento *Vagar* comanda o robô para que ele se movimente de forma aleatória pelo ambiente. Este comportamento é utilizado para que o robô possa se localizar, antes que um plano seja calculado até o objetivo. Uma direção e um período de tempo são gerados aleatoriamente, o robô então é comandado para se mover nesta direção até que o tempo se esgote. Com o tempo esgotado uma nova direção e um novo período de tempo são gerados e o processo se repete. Este comportamento atua em conjunto com o comportamento de *Desviar de Obstáculos*, assim o robô é capaz de se mover aleatoriamente pelo ambiente sem colidir com os obstáculos.

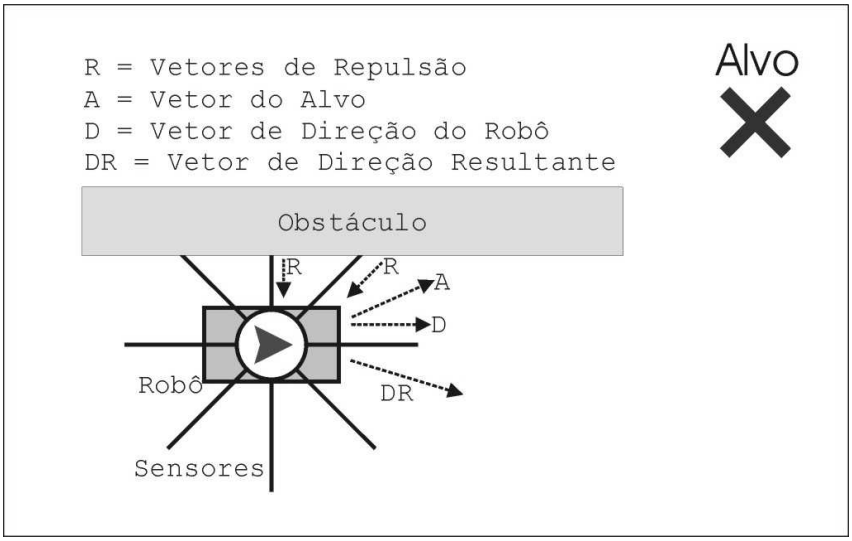

Fig. 6.3 Campos Potenciais

O comportamento de *Desviar de Obstáculos* é baseado no método de campos potenciais proposto por Borenstein & Koren [Bor89] (ver item 3.2.3.2), mas ao invés de se utilizar uma matriz para representar as forças repulsivas, foram utilizados vetores.

Cada sensor que detecta um obstáculo gera um vetor **R** que exerce uma força repulsiva contrária a direção do sensor, proporcionalmente a distância até o obstáculo detectado. A soma destes vetores com o vetor de direção **D** do robô produz o vetor de direção resultante **DR** (Fig. 6.3), que é normalizado e utilizado pelo árbitro para direcionar os atuadores de direção do robô móvel autônomo. Somente os sensores laterais e frontais são utilizados para a geração dos vetores de repulsão.

Utilizando esta abordagem, o robô é capaz de desviar tanto de obstáculos estáticos como de obstáculos móveis. O comportamento reage em tempo real utilizando as leituras dos sensores, e responde no mesmo instante que um obstáculo é detectado.

Durante a navegação, quando o robô esta seguindo o plano traçado pela camada deliberativa, o comportamento de *Desviar de Obstáculos* trabalha em conjunto com o comportamento de *Ir em Direção ao Alvo*.

O comportamento de *Ir em Direção ao Alvo* utiliza a mesma abordagem de campos potenciais utilizada no comportamento anterior. Utilizando a posição do alvo em relação a posição estimada do robô, o comportamento de *Ir em Direção ao Alvo* gera um vetor de direção do alvo **A** (Fig. 6.3). Este vetor é a saída do comportamento, e o árbitro fica encarregado de somar o vetor **A** com o vetor **DR** fornecido pelo comportamento de *Desviar de Obstáculos* para direcionar os atuadores de direção do robô**.** Este comportamento somente opera em conjunto com o comportamento de *Desviar de Obstáculos*. As informações sobre a posição do alvo e sobre a posição do robô são recebidas de um dos módulos auxiliares da camada funcional: o módulo indicador de posição do alvo.

O último comportamento, *Inverter Direção*, é utilizado para otimizar a navegação permitindo que o robô execute uma operação de marcha-a-ré para se posicionar em direção ao alvo. Quando este comportamento é acionado pelo árbitro, todos os demais comportamentos são inibidos. O robô inverte sua velocidade e inverte a sua direção executando uma marcha-a-ré em curva, até que alvo esteja posicionado a frete do robô. Este comportamento facilita a navegação, e é utilizado principalmente no inicio das trajetórias quando a direção do alvo estiver contrária a direção do robô.

6.2.4.2 - Árbitro

O árbitro tem a função de ativar ou inibir certos comportamentos, dependendo dos comandos recebidos pelo seqüênciador da camada funcional. O árbitro também pode ser programado com regras de fusão de saídas, para unificar saídas ambíguas dos comportamentos. Foram incluídas duas regras de fusão de saídas no árbitro do sistema de controle COHBRA.

A primeira regra é ativada quando os comportamentos *Vagar* e *Desviar de Obstáculos* são ativados simultaneamente. A direção fornecida pelo comportamento

Desviar de Obstáculos recebe um peso maior quando somada com a direção recebida do comportamento *Vagar*. Assim o robô tem um tendência maior a desviar dos obstáculos garantindo que não aconteçam colisões, e ao mesmo tempo permitindo que o robô navegue na direção fornecida pelo comportamento *Vagar*.

A segunda regra é ativada quando os comportamentos *Desviar de Obstáculos* e *Ir em Direção ao Alvo* estão ativos ao mesmo tempo. O árbitro recebe destes dois comportamentos os vetores **DR** e **A** respectivamente. Estes dois vetores são somados, o vetor resultante é normalizado e utilizado para direcionar os atuadores de direção do robô.

6.2.5 - Camada Funcional

6.2.5.1 - Autômato

O seqüênciador da camada funcional foi implementado na forma de um autômato finito. Cada estado deste autômato indica para o árbitro da camada vital quais comportamentos devem ser acionados ou inibidos. O diagrama de estados do autômato é apresentado na 0. O estado corrente do autômato é disponibilizado na área de memória compartilhada.

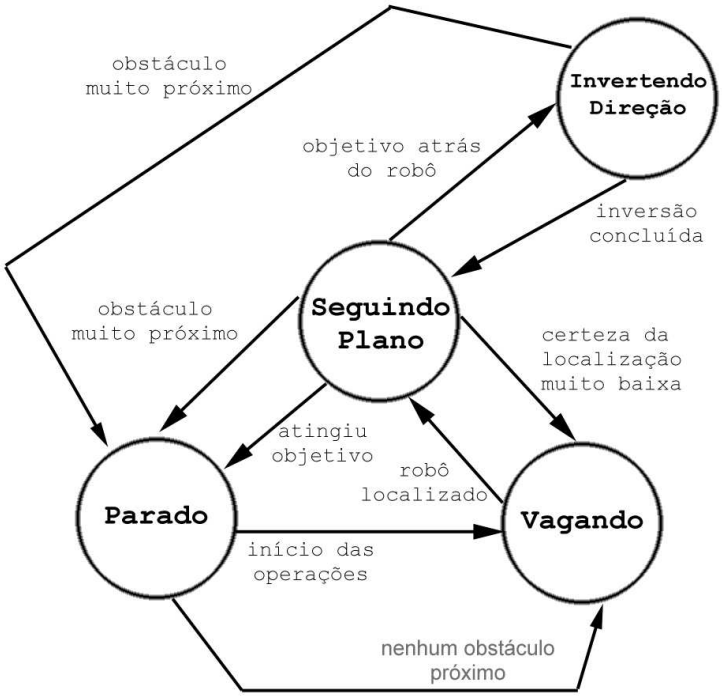

Fig. 6.4 Diagrama de Estados

A Tab. 6.1 apresenta um resumo das relações entre os estados e os comportamentos que são ativados durante a navegação do robô móvel autônomo.

<table>
<tr><th>Estados</th><th colspan="2">Comportamentos</th><th>Módulos Auxiliares</th></tr>
<tr><td>Parado</td><td colspan="2">Parar</td><td></td></tr>
<tr><td>Vagando</td><td colspan="2">Desviar de Obstáculos
Vagar</td><td></td></tr>
<tr><td rowspan="2">Seguindo Plano</td><td>Sem plano
(Dispara Interrupção!)</td><td>Com plano</td><td rowspan="2">Indicador de Direção do Alvo</td></tr>
<tr><td>Parar</td><td>Ir em Direção ao alvo
Desviar de Obstáculos</td></tr>
<tr><td>Invertendo Direção</td><td colspan="2">Inverter Direção</td><td></td></tr>
</table>

Tab. 6.1 Relação Estado x Comportamento

O estado inicial é *Parado.* Este estado mantém somente o comportamento *Parar* ativo na camada vital, quando o robô inicia as suas operações o autômato passa para o estado *Vagando*.

O estado *Vagando* comanda o árbitro da camada vital para acionar os comportamentos *Vagar* e *Desviar de Obstáculos* e inibir os demais comportamentos. Assim que o robô esta localizado o autômato troca para o estado *Seguindo Plano*. Para que o robô móvel autônomo seja considerado localizado, é necessário que a probabilidade da melhor posição estimada seja alta e a dispersão seja baixa (ver item 6.2.3 e item 7.1.1). Os valores para os limites que definem uma probabilidade alta e uma dispersão baixa são configuradas pelo usuário.

A partir do momento que o robô possui uma estimativa de posição válida, o estado muda para *Seguindo Plano*. A primeira ação executada neste estado é comandar o árbitro da camada vital para ativar o comportamento *Parar* e inibir os demais comportamentos, e então uma interrupção é enviada para a camada deliberativa requisitando que seja calculado um plano até o objetivo. Uma vez que um plano esteja disponível na memória compartilhada, o árbitro da camada vital é comandado para ativar os comportamentos de *Desviar de Obstáculos* e *Ir em Direção ao Alvo,* e inibir todos os outros comportamentos.

O plano é composto de uma série de alvos secundários que precisam ser seguidos para se chegar ao objetivo final. O módulo funcional "indicador de direção do alvo" é responsável de informar ao comportamento *Ir em Direção ao Alvo*, para qual alvo ele deve se dirigir.

Se for detectado que o plano não pode ser seguido, tanto por uma eminência de colisão, quanto pelos módulos funcionais (Ex. rota não pode ser seguida; caminho bloqueado), uma nova interrupção é enviada e um novo plano é requisitado. O módulo funcional responsável por detectar um plano inválido é o monitor de alterações no ambiente.

O localização estimada do robô é constantemente monitorada, caso a probabilidade e a dispersão não estejam com os valores desejados o estado volta a ser *Vagando*, e o processo de controle se reinicia.

O estado *Invertendo Direção* é selecionado quando a direção do alvo é oposta a direção do robô. Neste estado o árbitro da camada vital ativa o comportamento *Inverter Direção* até que o robô se posicione corretamente em relação ao alvo.

6.2.5.2 - Módulos Funcionais

Os módulos funcionais desempenham diversas tarefas auxiliares no processo de controle do robô móvel autônomo. No sistema de controle COHBRA foram utilizados uma série de módulos funcionais. Um destes módulos é responsável pelo monitoramento e atualização da representação interna do ambiente, e os demais fornecem parâmetros que auxiliam outros módulos nas diferentes camadas de controle.

A seguir cada um destes módulos é apresentado em maiores detalhes:

- **Módulo Monitor de Alterações no Ambiente (MMAA)**

Este módulo tem como função específica monitorar o ambiente real e atualizar a representação interna. Ele utiliza para isso as informações sobre os sensores filtrados disponibilizadas pelo módulo localizador na área de memória compartilhada. O módulo localizador disponibiliza quais sensores foram filtrados e seus respectivos valores no momento da filtragem. Somente ficam disponíveis os últimos sensores filtrados da partícula com a melhor estimativa de posição, ficando sob a responsabilidade do MMAA monitorar estes valores e garantir que eles somente sejam utilizados quando o robô estiver localizado.

Através do modelo sensorial do robô é possível estimar o ponto de localização de um obstáculo utilizando o valor de um determinado sensor. Desta forma o MMAA transforma as informações sobre os sensores filtrados em pontos (X,Y) que indicam a posição dos obstáculos desconhecidos no ambiente. Cada ponto tem um valor **T** associado que é utilizado para determinar se ele será utilizado para criar um novo obstáculo.

No momento que um destes pontos é detectado, ele é inserido em uma lista de pontos **LP** contendo os pontos que poderão se tornar obstáculos na representação interna do ambiente.

Os pontos da lista **LP** inicialmente recebem um valor de **T = 5**. A cada nova leitura sensorial o valor **T** de cada ponto é decrementado em 1 e novos pontos são processados. Caso algum destes novos pontos esteja muito próximo de um ponto antigo, o ponto antigo recebe **T = 5** novamente e o novo ponto é eliminado, assim evitando pontos duplicados. Quando um ponto estiver com **T = 0** ele é descartado da lista.

A cada intervalo de **N** leituras sensoriais, sendo o valor de **N** definido pelo usuário, os pontos da lista **LP** são movidos para uma lista definitiva **LD** que vai dar origem aos novos obstáculos. Com esta técnica os obstáculos móveis dificilmente são inseridos na representação interna do ambiente, pois a medida que o obstáculo

desconhecido se afasta do robô ele não é mais percebido pelos sensores e os valores **T** dos pontos que o representam não são mais renovados. O objetivo é inserir somente os obstáculos estáticos.

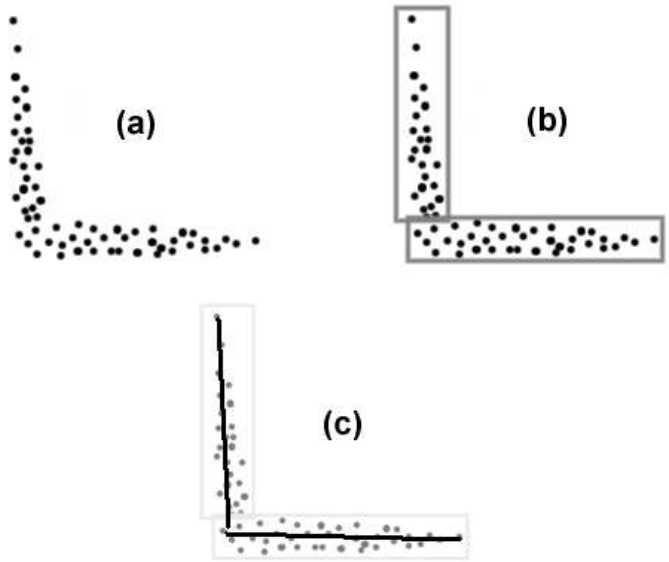

Fig. 6.5 Nuvem de Pontos, criação de obstáculo.

Um vez determinado que os pontos da lista **LD** devem ser convertidos para obstáculos definitivos, inicia-se o processo de conversão de pontos (Fig. 6.5a) para polígonos. Estes polígonos representarão os obstáculos encontrados e serão inseridos na camada de representação poligonal. Os pontos da lista **LD** são agrupados por proximidade (Fig. 6.5b) utilizando uma técnica de *Nearest Neighbor* [Oro93], e simplificados com a técnica de *Douglas-Peucker* [Hei00]. Este processo dá origem a uma descrição poligonal composta de segmentos de reta (Fig. 6.5c), descrevendo o contorno do obstáculo.

O módulo localizador e os comportamentos da camada vital são robustos o suficiente para possibilitar a navegação pelo ambiente mesmo na presença de obstáculos desconhecidos. Por isso o lista de pontos **LD** contendo a posição dos novos obstáculos estáticos somente é convertida para a representação interna em duas situações: quando o robô termina de seguir um plano, ou quando é detectado que o plano não pode ser mais seguido devido a alguma inconsistência. O próprio MMAA determina uma inconsistência quando algum dos pontos detectados está próximo demais da trajetória que o robô deve seguir. Quando isso ocorre é enviada uma interrupção para a camada deliberativa que calcula um novo plano.

- **Módulo Indicador de Direção do Alvo (MIDA)**

Este módulo utiliza o plano calculado pela camada deliberativa para informar ao comportamento da camada vital *Ir em Direção ao Alvo* qual a próxima direção a seguir. O plano é composto de alvos secundários, cada vez que um alvo é atingido o MIDA informa uma nova direção a seguir.

Através da posição estimada o MIDA determina se um alvo secundário foi atingido, então calcula a nova direção baseando na posição estimada atual e a posição do novo ponto de destino. Esta informação é disponibilizada na memória compartilhada.

- **Módulo Monitor de Posição Topológica (MMPT)**

O MMPT tem a função de calcular e disponibilizar na área de memória compartilhada a posição topológica do robô.

Cada área topológica é representada por um retângulo dentro do mapa, utilizando a posição estimada o MMPT determina em qual destas áreas o robô esta localizado. Esta informação é utilizada pela camada deliberativa para calcular o plano até um determinado objetivo.

6.2.6 - Camada Deliberativa

A camada deliberativa tem a função exclusiva de planejar a trajetória até um objetivo indicado pelo usuário, ou requisitado através de uma interrupção da camada funcional.

O planejamento é processado em duas fases. Na primeira fase, utilizando as informações topológicas, é executado um pré-planejamento que determina a seqüência de regiões topológicas que farão parte do caminho final, o algoritmo utilizado é Dijkstra [Cor90]. Esta informação otimiza a fase final de planejamento. Na fase final, o algoritmo A* [Nil80] é utilizado para calcular a trajetória definitiva. A representação matricial, otimizada com o pré-planejamento, é utilizada nesta fase.

Na fase de pré-planejamento as informações sobre a topologia do mapa são utilizadas para encontrar um caminho entre a região atual e a região de destino. Uma vez que um plano só pode ser construído quando o robô estiver localizado, estas regiões são disponibilizadas na memória compartilhada pelo módulo de monitoramento de regiões topológicas, que é um componente da camada funcional.

O módulo de pré-planejamento da camada deliberativa busca na memória compartilhada as informações sobre a região topológica onde o robô está localizado e sobre a região topológica onde o objetivo esta localizado. A partir destas informações o módulo de pré-planejamento utiliza o algoritmo de Dijkstra [Cor90] para encontrar o menor caminho entre estas duas regiões dentro do grafo que armazena a topologia das diversas regiões do mapa.

Uma vez que o módulo de pré-planejamento tenha calculado a seqüência de regiões que forma o caminho até o objetivo do robô, esta informação é utilizada para otimizar a representação de ambiente matricial, que será utilizada na fase final de planejamento. Esta otimização é feita da seguinte forma: todas as regiões topológicas que não fazem parte do caminho calculado são marcadas como ocupadas na representação de ambiente matricial. Assim, o espaço de procura do algoritmo A* utilizado na fase final de planejamento é reduzido, otimizado sensivelmente o cálculo

final de trajetória. Além disso o caminho gerado em geral se torna mais curto (ver os resultados obtidos no capítulo 7).

A fase final utiliza o algoritmo A* (ver item 3.2.3.3) para calcular a trajetória definitiva até o objetivo. O algoritmo A* utiliza a camada de representação matricial para calcular a trajetória, tirando proveito do pré-processamento executado na fase anterior.

Se o algoritmo A* não encontrar um caminho até o objetivo utilizando o pré-processamento topológico, é executado um novo planejamento utilizando todo o espaço de procura da representação matricial. Isto é necessário pois pode ocorrer de o caminho topológico ótimo estar obstruído, se fazendo necessário considerar rotas alternativas.

A heurística utilizada na implementação do algoritmo A* foi a minimização da distância euclidiana até o objetivo. A regra de movimentação é padrão permitindo que o algoritmo A* explore o mapa nas 8 direções possíveis (cada célula da representação matricial possui 8 células vizinhas). O custo de se movimentar uma célula em uma diagonal recebe um valor 40% maior do que se movimentar uma célula em linha reta para compensar a distância maior que é percorrida diagonalmente.

A planejamento de trajetória utilizando o algoritmo A* produz um caminho que é composto da seqüência de células que devem ser seguidas para se chegar até o objetivo. Esta seqüência de células é convertida para uma seqüência de pontos na representação poligonal utilizando como base o centro da célula.

O plano final é a trajetória a ser seguida pelo robô móvel autônomo, da sua posição atual até o seu objetivo, composta de uma seqüência de pontos na camada de representação poligonal. Este plano é disponibilizado na memória compartilhada para a ser utilizado pela camada funcional.

No próximo capítulo serão apresentados os resultados obtidos com o sistema de controle. O sistema de controle foi implementado em um robô simulado. O simulador apresentado no capítulo anterior, SimRob3D, é utilizado para realizar as simulações.

Capítulo 7 - Resultados

Este capítulo apresenta os resultados experimentais obtidos em ambientes simulados que demonstram a viabilidade da utilização do sistema de controle proposto. Cada experimento considera separadamente um determinado aspecto da implementação, incluindo os efeitos da variação de parâmetros e das condições iniciais (aleatórias).

Dois ambientes foram modelados e utilizados no SimRob3D para executar as simulações. O primeiro é o modelo virtual do ambiente utilizado pelo *Trinity College Fire Fighting Contest* [Tri02] Este ambiente (Fig. 7.1) tem dimensões de 2,48m por 2,48m, com área total de 6.15m². O segundo é o modelo virtual do andar onde está instalado o PIPCA na Unisinos, com dimensões de 50m por 17m, sendo a área total de 850m².

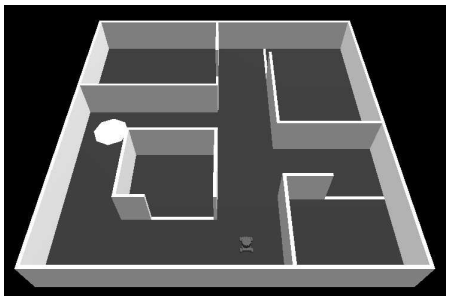

Fig. 7.1 Ambiente Trinity

Todos os modelos de robô utilizados foram configurados com a cinemática Ackerman (item 5.2.2.1) e sensores de distância. Os sensores foram configurados em um anel, com 16 sonares (item 5.2.3.2) posicionados a cada 22,5°. No ambiente Trinity o robô móvel tem as seguintes medidas: 6cm de largura, 8cm de comprimento, rodas com 2cm de diâmetro. No ambiente PIPCA o robô possui as seguintes medidas: 30cm de largura, 50cm de comprimento, rodas com 10cm de diâmetro. Em todas as simulações o robô se movimenta com uma velocidade constante.

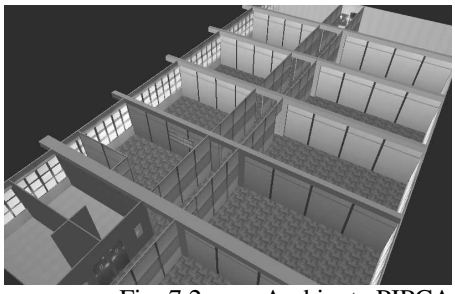

Fig. 7.2 Ambiente PIPCA

Os sonares (ver item 5.4.2.4) foram configurados com um ângulo de abertura de 30°, 5 raios e um erro médio de 5%. A distância máxima é de 100cm e mínima 1cm no ambiente Trinity, máxima de 500cm e mínima de 1cm no ambiente PIPCA. Os encoders, tanto de velocidade como de ângulo tem um erro médio associado de 10%. Os atuadores tem um erro de deslocamento médio de 1%.

O computador utilizado para a execução das simulações foi um AMD Athlon Thunderbird de 900Mhz com 256Mb de memória RAM.

7.1 - Localizador

O primeiro aspecto da implementação avaliado foi o módulo localizador. Este módulo é o principal componente do sistema de controle, e seu funcionamento afeta todos os outros módulos.

Cada simulação foi repetida 10 vezes com sementes aleatórias iniciais diferentes, e a cada novo conjunto de simulações as sementes eram utilizadas novamente. Todas as simulações que avaliaram o módulo localizador utilizaram os seguintes parâmetros, escolhidos de forma empírica, como padrão (itens 6.2.3.1 e 6.2.3.2):

Limite de Reamostragem: 77%
Quantidade Reamostrada: 30%
Limite de Filtragem: 95%

O ambiente Trinity foi utilizado como padrão para todas as simulações. O ambiente PIPCA foi utilizado no experimento de localização local para avaliar a capacidade do sistema de controle também em um ambiente mais complexo.

7.1.1 - Localização local e global em um ambiente estático.

O primeiro experimento avaliou a capacidade de localização local e global em um ambiente totalmente estático com a representação interna (mapa) correspondendo ao ambiente real, sem obstáculos móveis.

Para determinar a capacidade de localização global, foi observado o números de ciclos necessários para que o robô móvel autônomo se localizasse. Cada ciclo corresponde a um conjunto de leituras sensoriais recebidas do robô e utilizadas pelo módulo localizador para estimar a posição. Estes ciclos ocorrem em períodos de tempo regulares (+/- 50ms).

Para ser considerado localizado, o módulo localizador deveria fornecer uma posição estimada com **probabilidade maior que 90%**, uma **dispersão menor do que 2cm**, e **90% das partículas agrupadas** na área de dispersão. O robô somente passava ao status de localizado se estes valores se repetissem por **mais de 10 ciclos**.

Uma vez localizado, a localização local era avaliada para determinar a capacidade do módulo localizador em manter a posição estimada do robô com um erro aceitável. Para isso foram observados os seguintes fatores:

Erro Real: usando a posição do robô fornecida pelo simulador SimRob3D, foi possível determinar o erro real entre a posição estimada (melhor partícula) e a posição real do robô no ambiente. O cálculo da média considera a variação deste erro durante os 'N' ciclos da simulação (2700 ciclos).

Probabilidade: a probabilidade fornecida pelo módulo localizador que indica a certeza sobre a posição estimada.

Dispersão: parâmetro também fornecido pelo módulo localizador, que indica a distância média entre as partículas, e o percentual de partículas que estão agrupadas na posição estimada.

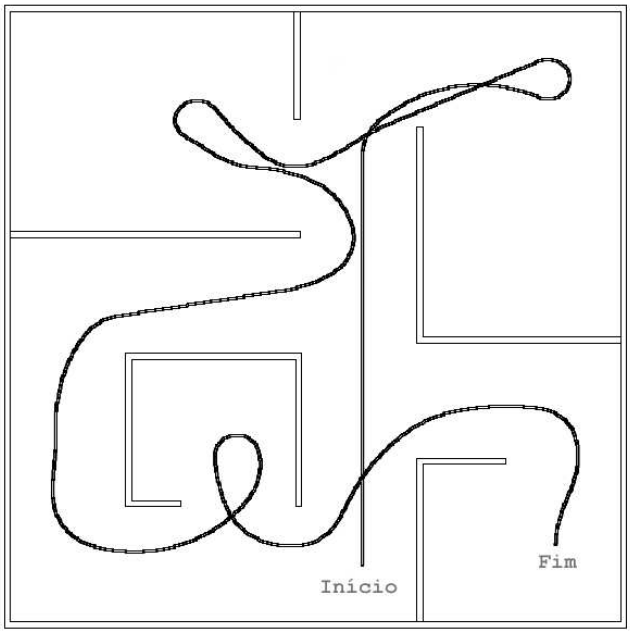

Fig. 7.3 Mapa e Trajetória do Robô no Ambiente Trinity

No primeiro conjunto de simulações foi utilizado o ambiente Trinity. O robô segue uma trajetória predefinida (Fig. 7.3) em todas as simulações, e sua posição inicial é totalmente desconhecida pelo sistema de controle. Cada simulação dura 2700 ciclos, quando o robô atinge a posição "Fim".

Foram realizados 3 conjuntos de simulações, cada um com uma quantidade diferente de partículas: 100, 500 e 1000 partículas respectivamente.

		Após estar Localizado (Localização Local)			
Simulação	Ciclos para se Localizar	Média e Desvio Padrão do Erro Real (cm)		Probabilidade Média Estimada (%)	Dispersão Média Estimada (cm ; %)
1	312	4,07	3,67	0,96	(0,2 ; 0,9)
2	554	3,03	1,75	0,98	(0,25 ; 0,95)
3	*Não se Localizou*	-	-	-	-
4	1444	5,03	2,51	0,95	(0,19 ; 0,87)
5	455	3,17	1,65	0,98	(0,23 ; 0,97)
6	1930	5,54	0,75	0,96	(0,17 ; 0,93)
7	*Não se Localizou*	-	-	-	-
8	2143	3,97	0,84	0,97	(0,2 ; 0,96)
9	799	2,52	1,1	0,98	(0,22 ; 0,97)
10	542	3,24	8,46	0,96	(0,23 ; 0,85)
Média	1022,37 (80% de sucesso)	3,82	2,59	0,96	(0,21 ; 0,92)

Tab. 7.1 1º Conjunto: com 100 partículas, ambiente Trinity

		Após estar Localizado (Localização Local)			
Simulação	Ciclos para se Localizar	Média e Desvio Padrão do Erro Real (cm)		Probabilidade Média Estimada (%)	Dispersão Média Estimada (cm ; %)
1	307	1,54	0,42	0,99	(0,44 ; 0,99)
2	990	2,26	1,17	0,98	(0,53 ; 0,97)
3	500	3,26	5,62	0,96	(0,70 ; 0,92)
4	941	1,91	0,54	0,99	(0,60 ; 0,98)
5	350	3,07	3,23	0,98	(0,41 ; 0,96)
6	1278	2,53	1,03	0,98	(0,45 ; 0,99)
7	703	2,04	0,61	0,99	(0,53 ; 0,98)
8	530	2,5	2,06	0,98	(0,46 ; 0,97)
9	1329	3,49	1,7	0,98	(0,41 ; 0,99)
10	631	3,72	6,41	0,97	(0,44 ; 0,93)
Média	756,90	2,63	2,28	0,98	(0,50 ; 0,97)

Tab. 7.2 2º Conjunto: com 500 partículas, ambiente Trinity

		Após estar Localizado (Localização Local)			
Simulação	Ciclos para se Localizar	Média e Desvio Padrão do Erro Real (cm)		Probabilidade Média Estimada (%)	Dispersão Média Estimada (cm ; %)
1	32	1,98	0,48	0,99	(0,67 ; 0,98)
2	898	2,74	1,71	0,98	(0,55 ; 0,96)
3	1416	2,43	1,74	0,98	(0,62 ; 0,98)
4	332	2,33	1,13	0,99	(0,68 ; 0,98)
5	565	1,87	0,54	0,99	(0,59 ; 0,99)
6	853	1,96	0,77	0,99	(0,70 ; 0,97)
7	75	1,84	0,51	0,99	(0,74 ; 0,97)
8	571	3,76	3,27	0,97	(0,52 ; 0,88)
9	709	2,05	0,77	0,98	(0,57 ; 0,98)
10	1293	2,69	1,46	0,97	(0,61 ; 0,95)
Média	674,40	2,37	1,24	0,98	(0,63 ; 0,96)

Tab. 7.3 3º Conjunto: com 1000 partículas, ambiente Trinity

Os resultados obtidos mostraram que módulo localizador é capaz de estimar uma posição correta em um ambiente estático, mesmo quando a localização inicial do robô é desconhecida.

No primeiro conjunto de simulações o módulo localizador não foi capaz de se localizar globalmente em duas simulações (simulações 3 e 7). Além disso a média de ciclos para se localizar ficou em 1022, mais de 37% do total de ciclos da trajetória. Para uma localização global no ambiente Trinity (com 6.15m²), a utilização de somente 100 partículas (16,26 partículas/m²) se mostrou insuficiente. Neste mesmo experimento, na localização local foi possível manter um erro médio na posição de 3,82cm com um

desvio padrão de 2,59cm. O módulo localizador se mostrou robusto durante a localização local conseguindo uma probabilidade média de 96% e uma dispersão de 0,21cm concentrando 92% das partículas na posição estimada.

Nos dois últimos conjuntos, com 500 e 1000 partículas, o módulo localizador foi capaz de se localizar globalmente em todas as simulações, conseguindo se localizar em média a partir de 756 ciclos (2º conjunto) e 674 ciclos (3º conjunto).

Na localização local o 2º conjunto foi capaz de manter um erro médio real de 2,63cm com um desvio padrão de 2,28cm, a probabilidade estimada da posição foi de 98% em média, com um dispersão de 0,57cm com 97% das partículas agrupadas. A localização local no 3º conjunto, com 1000 partículas, obteve um erro real de posição chegando a 2,37cm e desvio de 1,24cm em média. A probabilidade e a dispersão do 2º e 3º conjunto foram praticamente as mesmas.

Em nenhum dos conjuntos foi necessário executar uma relocalização, uma vez localizado o módulo localizador foi capaz de manter a posição correta até o fim da trajetória em todas as simulações. Em um ambiente estático, a estimativa de posição converge para a posição real, como pode ser visto no gráfico apresentado na Fig. 7.4.

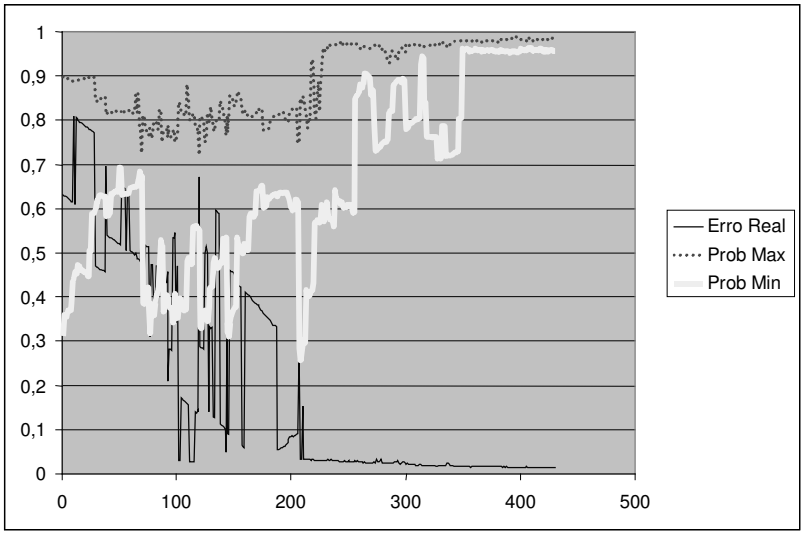

Fig. 7.4 Erro Real e Probabilidade da Localização

O gráfico da Fig. 7.4 foi obtido em uma simulação no ambiente Trinity, seguindo-se a mesma trajetória dos experimentos anteriores. Ele apresenta o **erro real** da posição do robô em relação ao mapa normalizado entre [0,1], e as **probabilidades máxima e mínima** das partículas estimadas pelo módulo localizador. Foram executados 500 ciclos, e utilizadas 500 partículas.

Na simulação apresentada no gráfico o robô móvel foi considerado localizado a partir do ciclo 346, mas a partir do ciclo 212 já estava convergindo para a posição real. Até o ciclo 212 o gráfico mostra o comportamento típico de um método estocástico, com as partículas sendo distribuídas pelo ambiente para tentar localizar a posição correta do robô. Entre o ciclo 212 e 346 as partículas, que antes estavam distribuídas em

diversos pontos do mapa, começam a se concentrar na localização correta do robô. A partir do ciclo 346 todas as partículas estão agrupadas próximas a uma mesma posição, com uma dispersão baixa. A dispersão baixa pode ser notada pela pouca diferença entre **Prob. Max.** e **Prob. Min.**

Foi realizado também um experimento no ambiente PIPCA para avaliar somente a localização local em um ambiente complexo. A Fig. 7.5 mostra a trajetória percorrida pelo robô no ambiente durante o experimento. A posição inicial foi informada para o módulo localizador, e foram utilizadas 100 partículas. Foram realizadas 10 simulações neste ambiente, cada uma com uma duração de 3120 ciclos.

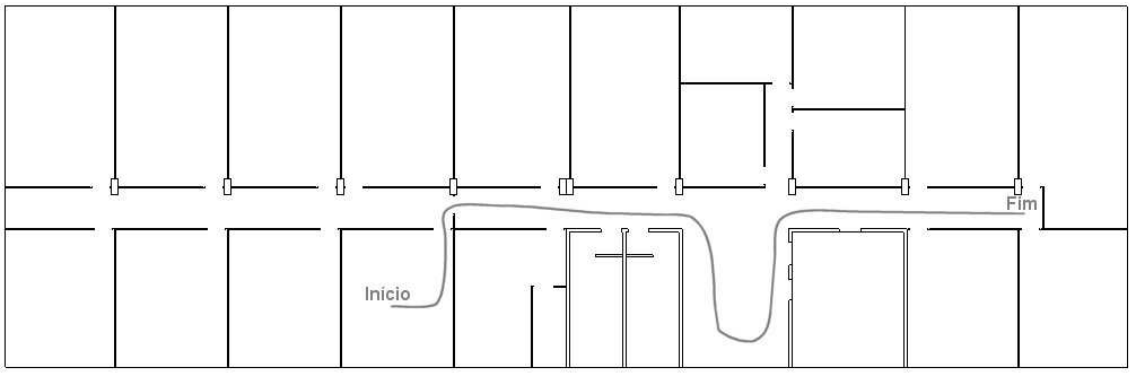

Fig. 7.5 Trajetória no Ambiente PIPCA

O módulo localizador manteve a estimativa de posição correta durante toda a simulação, não sendo necessária nenhuma relocalização. Isto demostra a exelente capacidade de localização local do módulo localizador em um ambiente estático. Mesmo utilizando somente 100 partículas em um ambiente complexo, foi possível manter uma estimativa de posição correta durante toda a trajetória. Os seguintes valores médios foram obtidos nas simulações:

Erro real: 2,13cm
Probabilidade: 94%
Dispersão: 2,83cm com 93% das partículas agrupadas
Relocalizações: 0
Ciclos Localizado: 3120 (durante toda a trajetória)

7.1.2 - Relocalização em um ambiente estático

O objetivo deste experimento é avaliar a capacidade do robô em perceber que sua posição está errada, e encontrar a posição correta. Foram realizadas 10 simulações utilizando posições incorretas diferentes, que podem ser vistas na Fig. 7.6. As 10 simulações foram repetidas para 100, 500 e 1000 partículas.

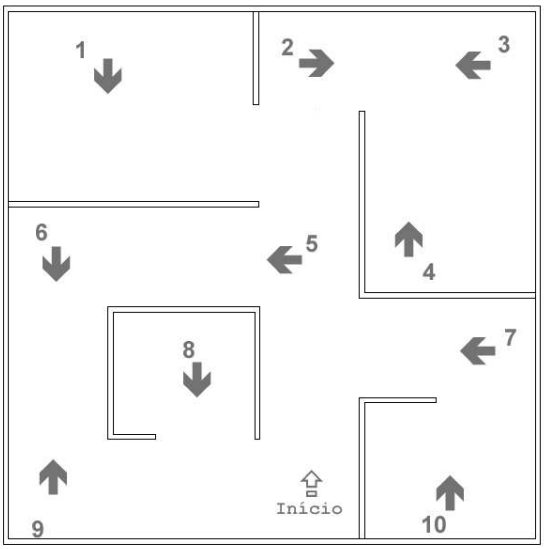

Fig. 7.6 Pontos Incorretos

A trajetória seguida neste experimento é a mesma da Fig. 7.3, mas em cada uma das simulações o sistema de controle recebe uma posição incorreta sobre a posição inicial do robô (Fig. 7.6). Esta posição incorreta é configurada com uma alta probabilidade de ser a posição real.

Simulação	Ciclos para se Relocalizar (100 partículas)	Ciclos para se Relocalizar (500 partículas)	Ciclos para se Relocalizar (1000 partículas)
1	1302	470	482
2	585	1176	600
3	990	1388	560
4	1506	313	282
5	2184	639	366
6	1457	583	582
7	991	419	982
8	2645	1307	494
9	1663	360	502
10	277	461	968
Média	1370	761,6	681,8

Tab. 7.4 Relocalização em ambiente estático

O módulo localizador foi capaz de relocalizar o robô em todas as simulações. A quantidade de ciclos necessários para se relocalizar foi bem próxima dos valores obtidos na localização global do experimento anterior. O número de ciclos necessários para se relocalizar é um pouco maior do que para se localizar globalmente a partir de uma posição desconhecida. Isso se deve ao fato de que o módulo localizador precisa de um certo número de ciclos para detectar que a posição indicada como correta não reflete a posição real.

7.1.3 - Localização em um ambiente dinâmico

O objetivo deste experimento foi o de avaliar a capacidade do módulo localizador em estimar uma posição mesmo em um ambiente dinâmico, onde o mapa não corresponde exatamente ao ambiente real.

Neste experimento o ambiente teve sua estrutura modificada. O mapa fornecido para o sistema de controle não sofre alterações. São utilizados 3 níveis de alterações (Fig. 7.7) :

Alterações Pequenas: 2 paredes são removidas do ambiente (2 pontos de alteração).

Alterações Médias: 2 paredes são removidas e são colocadas em locais diferentes do ambiente (4 pontos de alteração).

Alterações Grandes: 2 paredes são removidas e são colocadas em locais diferentes do ambiente, e 3 obstáculos estáticos são adicionados no ambiente (7 pontos de alteração).

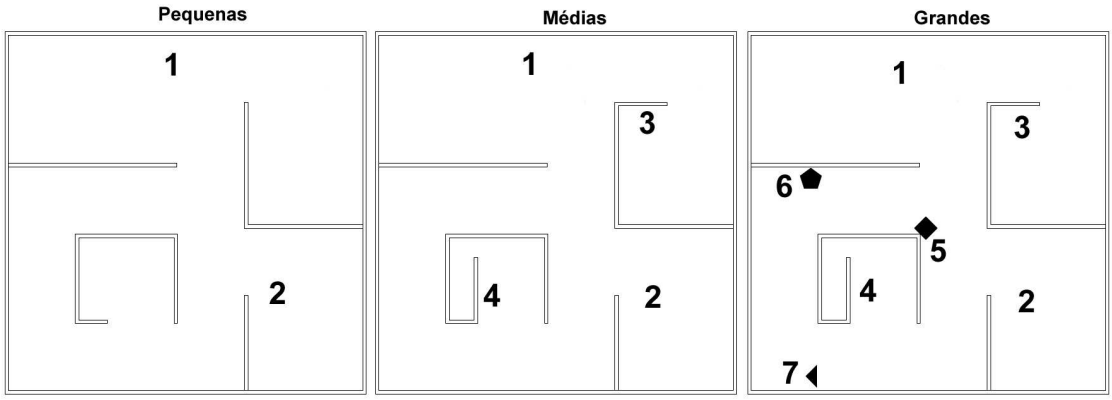

Fig. 7.7 Alterações no Ambiente

Foram realizadas 10 simulações com 500 partículas para cada configurações do ambiente seguindo a trajetória apresentada na Fig. 7.3, sendo avaliados os seguintes parâmetros em um total de 2700 ciclos:

Ciclos para se Localizar: o número de ciclos necessários para que o robô se localize inicialmente a partir de uma posição desconhecida.

Quantidade de Relocalizações: número de vezes que o módulo localizador, depois de se localizar inicialmente, perdeu sua estimativa de posição e foi necessário executar uma relocalização.

Quantidade Total de Ciclos Localizado: número total de ciclos em que o robô se manteve localizado durante a simulação.

Média de Sensores Filtrados: quantidade média de sensores que foram eliminados do cálculo de similaridade pelo filtro de distância (ver item 6.2.3.2), quando uma alteração do ambiente é percebida pelos sensores.

Simulação	Ciclos para se Localizar	Quantidade de Relocalizações	Quant. de Ciclos Localizado	Média Sensores Filtrados
1	56	2	1579	1,71
2	901	0	1799	1,69
3	1540	0	1160	2,54
4	1399	1	731	2,09
5	96	1	1922	1,89
6	557	1	1189	1,73
7	632	1	1655	1,87
8	390	2	1770	2,07
9	1368	0	1332	1,70
10	674	1	1511	2,16
Média	761,3	0,9	1464,8	1,94

Tab. 7.5 Nível 1: Alterações Pequenas

Simulação	Ciclos para se Localizar	Quantidade de Relocalizações	Quant. de Ciclos Localizado	Média Sensores Filtrados
1	1292	0	1408	2,02
2	1604	1	560	2,40
3	1061	0	1639	2,12
4	*Não se Localizou*	-	-	-
5	2408	0	292	1,88
6	1156	0	1544	1,90
7	*Não se Localizou*	-	-	-
8	1146	0	1554	2,01
9	287	1	1456	1,99
10	1263	0	1437	2,33
Média	1277,1 (80% de sucesso)	0,25	1236,2	2,08

Tab. 7.6 Nível 2: Alterações Médias

Simulação	Ciclos para se Localizar	Quantidade de Relocalizações	Quant. de Ciclos Localizado	Média Sensores Filtrados
1	*Não se Localizou*	-	-	-
2	1284	0	1416	2,37
3	*Não se Localizou*	-	-	-
4	1825	0	875	2,57
5	*Não se Localizou*	-	-	-
6	*Não se Localizou*	-	-	-
7	*Não se Localizou*	-	-	-
8	*Não se Localizou*	-	-	-
9	849	0	1851	2,16
10	*Não se Localizou*	-	-	-
Média	1319,3 (30% de sucesso)	0	1380,6	2,36

Tab. 7.7 Nível 3: Alterações Grandes

As simulações executadas nos ambientes alterados demonstrou que o módulo localizador é capaz de estimar a posição do robô mesmo em um ambiente dinâmico, desde que este ambiente possua pelo menos alguns locais inalterados, que sejam similares aos fornecidos para a representação interna (mapa) do sistema de controle.

Em todas as simulações, o filtro de distância eliminou do cálculo de similaridade cerca de 2 sensores em média. Em regiões com grandes alterações se obteve picos de até 8 sensores eliminados do cálculo de similaridade (de um total de 16).

Todas as simulações executadas no nível 1 (alterações pequenas) obtiveram sucesso em estimar uma posição, a partir de 761,3 ciclos em média. O ambiente do nível 1 foi configurado com duas alterações, uma no ínicio da trajetória em outra no final. Desta forma, apesar das modificações, uma parte do ambiente ainda corresponde ao mapa fornecido para o sistema de controle. Em certos casos o robô conseguiu se localizar, mas devido às alterações do ambiente a certeza desta posição era baixa, ocasionando a necessidade de algumas relocalizações (0,9 em média). Apesar das relocalizações o módulo localizador conseguiu manter a estimativa de posição durante 1464 ciclos em média, de um total de 2700.

As simulações executadas no nível 2 (alterações médias), demonstraram a robustez do módulo localizador em um ambiente dinâmico. Neste nível foram alterados 4 locais do ambiente, diminuindo ainda mais a correspondência do ambiente em relação ao mapa. Apesar disso, o módulo localizador foi capaz de estimar uma posição em 8 de um total de 10 simulações, conseguindo se localizar a partir de 1277,1 ciclos, e se manter localizado durante 1236,2 ciclos em média.

O ambiente nível 3 (alterações grandes) foi configurado para não possuir locais que correspondam exatamente com o mapa. As simulações executadas neste nível demonstraram as limitações do módulo localizador em executar uma localização global em um ambiente dinâmico com muitas alterações em relação ao mapa. Somente em 3 simulações foi possível estimar uma posição de forma global (sem informação inicial sobre a localização do robô). Em relação a localização local neste ambiente, a partir do momento que a posição era conhecida, o robô conseguiu mantê-la correta, o que é demonstrado pelo número de relocalizações médio (zero) e pela quantidade de ciclos localizado (1380,6 em média). O módulo localizador ainda se mostra robusto, mesmo em ambientes com grandes alterações.

Para comprovar esta última afirmação foram realizadas 10 novas simulações no ambiente nível 3, fornecendo para o módulo localizador a posição correta do robô desde o início das simulações. A posição correta foi fornecida distribuindo as partículas em torno da posição real do robô ao invés de distribuir as partículas uniformemente em todo o mapa. As partículas foram distribuídas com um erro de 4cm na posição e 4^o na orientação.

Os seguintes resultados foram obtidos seguindo-se a mesma trajetória dos experimentos anteriores:

Erro real: 1,96cm
Probabilidade: 98%
Dispersão: 1,58cm com 95% das partículas agrupadas
Relocalizações: 0
Ciclos Localizado: 2700 (durante toda a trajetória)

Para se localizar globalmente a partir de uma posição totalmente desconhecida, o módulo localizador precisa de um mínimo de correspondência entre o mapa e o ambiente para que obtenha sucesso em sua tarefa. Mas se uma posição inicial é informada, ele é capaz de manter a estimativa de posição correta, mesmo em um ambiente com grandes alterações, como comprova a simulação anterior.

7.2 - Navegação

O objetivo dos experimentos de navegação foi o de validar as camadas de controle deliberativa, funcional e vital. A capacidade do sistema de controle em guiar o robô móvel até o seu destino depende da integração dos diversos módulos, e os experimentos a seguir avaliam a performance e a robustez desta integração em seus diferentes aspectos.

Em todas as simulações relacionadas a navegação o módulo localizador já possuía uma posição estimada. Foi avaliado se o módulo localizador conseguia manter esta posição correta durante as tarefas de navegação, tanto em ambientes estáticos como em ambientes dinâmicos com obstáculos estáticos e com obstáculos móveis.

Em todas as simulações executadas, foi utilizada uma representação matricial com uma precisão de 2cm, resultando em uma grade de 124 por 124 células. As células que estiverem a até 10 centímetros de distância de um obstáculo da camada poligonal são marcadas como ocupadas. Os parâmetros do módulo localizador são os mesmos dos experimentos anteriores: 500 partículas, localizado se satisfazer as condições do item 7.1.1.

7.2.1 - Navegação em um ambiente estático

O primeiro experimento teve como objetivo validar a capacidade do sistema de controle em guiar o robô até um objetivo em um ambiente estático, sem alterações em relação ao mapa e sem obstáculos móveis.

Foram selecionadas no mapa 10 posições para onde o robô deveria se locomover. A Fig. 7.8 mostra cada um destes pontos marcados com um X e a posição inicial do robô marcada com um círculo.

Em cada uma das simulações a performance do planejamento e a capacidade do robô em se deslocar até o seu objetivos foram observadas. A performance é medida pelo **tempo** que a camada deliberativa precisa para calcular o caminho, e pelo **tamanho do caminho** gerado. É indicado também se o robô foi capaz de chegar até o seu destino, sem colidir com nenhum obstáculo.

Neste experimento a camada deliberativa não utilizou a fase de pré-planejamento para calcular a trajetória. No experimento final, utilizando as informações topológicas a fase de pré-planejamento será utilizada, e a sua performance será comparada com as simulações executadas somente com o algoritmo A* (ver item 6.2.6).

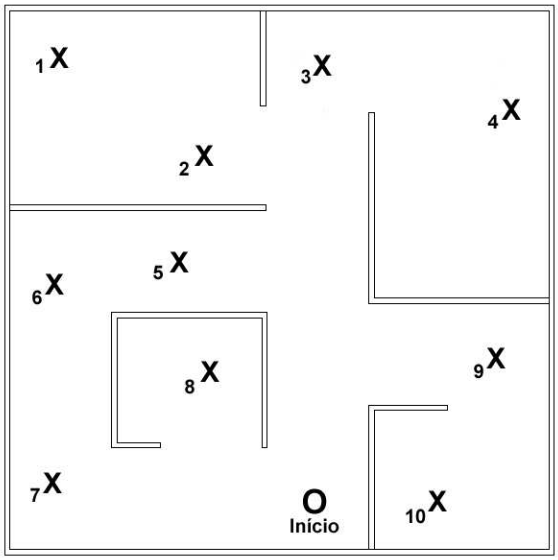

Fig. 7.8 Pontos de Destino da Navegação

Simulação	Tempo para Calcular Caminho (seg.)	Tamanho do Caminho (cm)	Capacidade de Navegar até o Destino
1	2,92	265,54	SIM
2	0,71	180,43	SIM
3	0,79	196,83	SIM
4	2,25	255,60	SIM
5	0,43	150,77	SIM
6	1,01	197,25	SIM
7	0,10	114,12	SIM
8	0,08	88,91	SIM
9	0,13	125,25	SIM
10	0,62	168,23	SIM

Tab. 7.8 Navegação em Ambiente Estático

Em todas as simulações o sistema de controle foi capaz de guiar o robô móvel até o seu objetivo. A estimativa de posição permaneceu correta durante toda a trajetória, não sendo necessária nenhuma relocalização.

O planejador, utilizando somente o algoritmo A*, se mostrou eficiente no ambiente Trinity. O maior tempo necessário para calcular uma trajetória foi cerca de 2,92s para uma alvo distante 265,54cm, distância que necessita uma maior exploração do mapa matricial para encontrar o caminho. O tempo para calcular a trajetória se torna ainda menor quando são utilizadas as informações topológicas, como mostra o experimento realizado no item 7.2.4.

7.2.2 - Navegação em um ambiente dinâmico (sem obstáculos móveis)

Neste experimento foi avaliada a capacidade do sistema de controle em guiar o robô de uma posição inicial conhecida até um destino em um ambiente dinâmico, onde o mapa não possui todos os elementos presentes no ambiente. Não foram utilizados obstáculos móveis.

Utilizando o plano fornecido pela camada deliberativa, o sistema de controle deve ser robusto o suficiente para se locomover de um ponto ao outro do ambiente, mesmo que alguns obstáculos não estejam modelados no mapa fornecido pelo usuário. A partir de uma posição conhecida, o sistema de controle deve ser capaz de atingir o objetivo, e manter a sua posição correta durante a navegação.

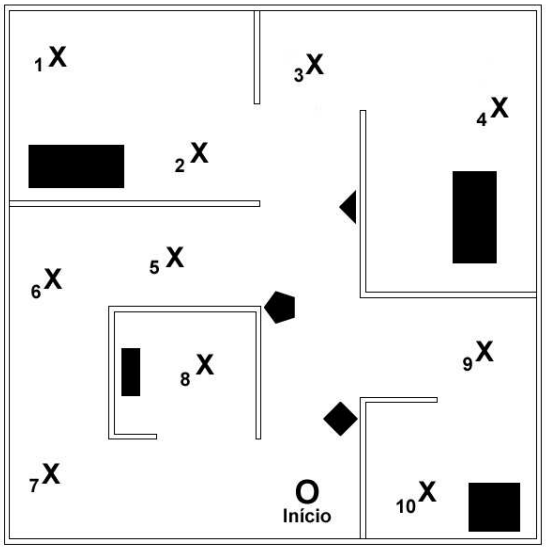

Fig. 7.9 Navegação em Ambiente Dinâmico

O ambiente foi configurado para simular uma situação onde foi informada no mapa somente a estrutura fixa do ambiente (paredes), mas não foi informada a posição da mobília que compõe o ambiente (mesas, armários, etc.). A Fig. 7.9 apresenta a configuração de ambiente utilizada. A partir do ponto de início, o sistema de controle tinha a tarefa de guiar o robô até os alvos indicados pelos X.

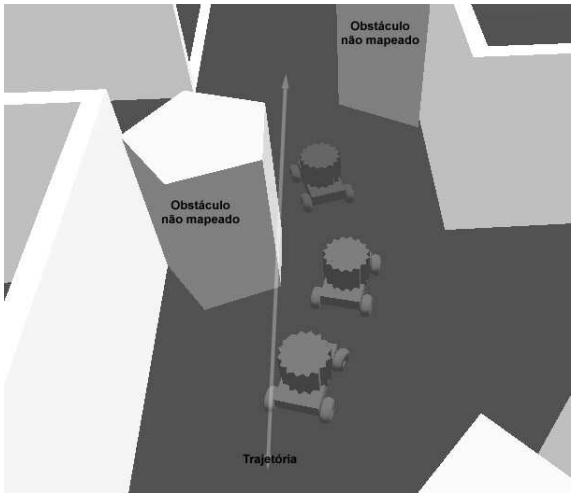

Fig. 7.10 Seqüência de Movimento do Robô

Os resultados obtidos mostraram que em todas as simulações o sistema de controle foi capaz de guiar o robô móvel até o seu destino desviando dos obstáculos não mapeados, e o módulo localizador foi capaz de manter a posição correta durante todo o percurso. O módulo localizador manteve a posição estimada com uma probabilidade média de 98% e uma dispersão de 0,73cm com 91% das partículas agrupadas. O erro real de posição ficou em 2,16cm em média.

A Fig. 7.10 mostra a seqüência do deslocamento do robô observadas durante a simulação número 3 (ver Fig. 7.9). A trajetória calculada pela camada deliberativa intersecta um obstáculo desconhecido (não mapeado), mas mesmo assim o sistema de controle é capaz de guiar o robô fazendo com que ele desvie do obstáculo inesperado, e ao mesmo tempo continue seguindo a trajetória especificada.

Estes resultados demonstram a boa integração entre a camada deliberativa e a camada vital (reativa). Também pode-se observar a capacidade que o módulo localizador possui em manter uma posição correta, mesmo em um ambiente com várias alterações em relação ao mapa.

7.2.3 - Navegação em um ambiente dinâmico (com obstáculos móveis)

O objetivo deste experimento era verificar a capacidade do sistema de controle em guiar o robô móvel até um determinado objetivo em um ambiente que possua obstáculos móveis.

O ambiente foi configurado com 7 obstáculos móveis, que se movimentam seguindo de modo cíclico trajetórias predefinidas. A Fig. 7.11 apresenta as trajetórias que os obstáculos móveis seguem no ambiente, e os objetivos de navegação do robô. As trajetórias foram configuradas tentando simular o movimento de uma pessoa se movimentando por um corredor ou por uma sala. Os obstáculos móveis em nenhum momento causavam um bloqueio total da passagem do robô.

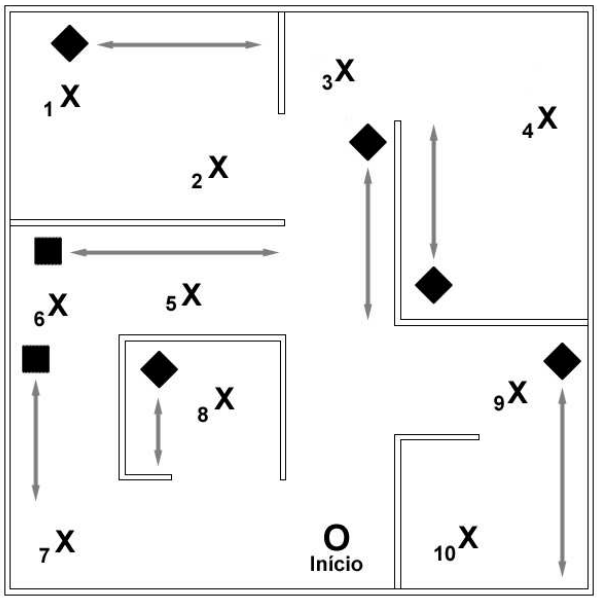

Fig. 7.11 Trajetória dos Obstáculos Móveis

Em todas as 10 simulações realizadas o sistema de controle foi capaz de guiar o robô até o seu objetivo, evitando colisões com os obstáculos que se movimentavam no ambiente. Também não foi necessário nenhuma relocalização, com o robô mantendo sua posição correta durante todo o percurso.

7.2.4 - Navegação com o uso das informações topológicas

O objetivo deste experimento é demonstrar o ganho de performance que é obtido quando se utilizam as informações topológicas durante o planejamento de trajetória.

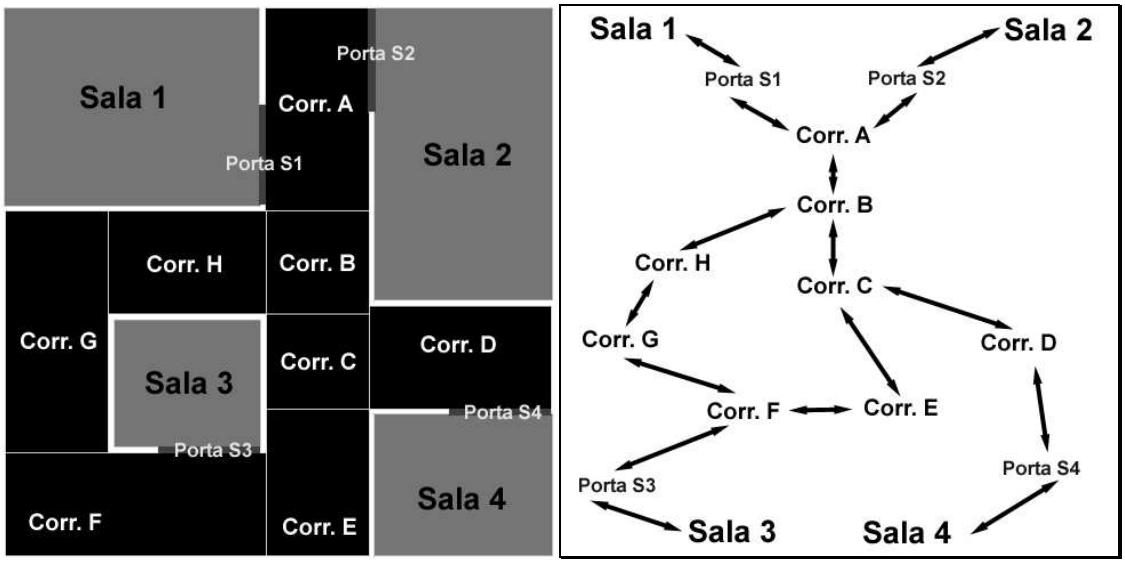

Fig. 7.12 Topologia do Ambiente Trinity

As configurações utilizadas são as mesmas do item 7.2.1, mas a camada deliberativa executa uma fase de pré-planejamento utilizando as informações topológicas fornecidas pelo usuário. A Fig. 7.12 mostra as informações topológicas disponibilizadas para o ambiente Trinity.

Simulação	Tempo para Calcular Caminho (somente A*) (seg.)	Tempo para Calcular Caminho (com topologia) (seg.)	Ganho de Performance (%)
1	2,92	0,63	78,4
2	0,71	0,11	84,5
3	0,79	0,12	84,8
4	2,25	0,29	87,1
5	0,43	0,08	81,3
6	1,01	0,16	84,1
7	0,10	0,06	40,0
8	0,08	0,07	12,5
9	0,13	0,06	53,8
10	0,62	0,13	79,0
		Média	68,5

Tab. 7.9 Planejamento de Trajetória utilizando as Informações Topológicas

Os resultados obtidos demonstraram que o pré-planejamento executado pela camada deliberativa utilizando as informações topológicas melhora significativamente a performance do planejamento de trajetória. Nas trajetórias mais longas a utilização das informações topológicas foi capaz de um ganho de performance de até 87,1%, e em média o ganho foi de 68,5%.

7.3 - Discussão final

Foram feitos experimentos de localização em ambientes estáticos, dinâmicos, e dinâmicos com obstáculos móveis para avaliar a capacidade de localização local, global e de relocalização. Foram realizados experimentos de navegação em ambientes estáticos, dinâmicos, e dinâmicos com obstáculos móveis, com o objetivo de avaliar a capacidade do sistema de controle em se locomover nestes ambientes, seguindo os planos fornecidos pela camada deliberativa. Foram realizados também experimentos para avaliar o ganho de performance obtido quando se utiliza as informações topológicas para auxiliar no planejamento de trajetória.

Os experimentos demonstraram que o sistema de controle foi capaz de manter um estimativa de posição correta em todas as configurações, possuindo uma capacidade de localização local excelente. Os experimentos com localização global e relocalização obtiveram bons resultados na maioria dos ambientes, somente tendo dificuldades nos ambientes com grandes alterações. Na tarefa de navegação o sistema de controle foi capaz de locomover o robô até os objetivos determinados em todos os tipos de ambiente utilizados nos experimentos. A utilização das informações topológicas para auxiliar o planejamento de trajetória obteve um ganho de até 87,1% na velocidade do cálculo da trajetória.

A Tab. 7.10 apresenta um resumo das capacidades e limitações do sistema de controle verificadas após a análise dos resultados obtidos com as simulações.

Tarefa	Sub-Tarefa	Capacidade
Localização Local	Ambiente Estático	✓✓
	Ambiente Dinâmico	✓✓
	Ambiente Dinâmico com Obstáculos Móveis	✓✓
Localização Global	Ambiente Estático	✓✓
	Ambiente Dinâmico (com ou sem obstáculos móveis) Alterações Médias no Ambiente	✓
	Ambiente Dinâmico (com ou sem obstáculos móveis) Alterações Grandes no Ambiente	⇩
Relocalização	Ambiente Estático	✓✓
	Ambiente Dinâmico (com ou sem obstáculos móveis) Alterações Médias no Ambiente	✓
	Ambiente Dinâmico (com ou sem obstáculos móveis) Alterações Grandes no Ambiente	⇩

Navegação	Ambiente Estático	✓✓
	Ambiente Dinâmico	✓✓
	Ambiente Dinâmico Com Obstáculos Móveis	✓✓
Inserção / Remoção de Obstáculos; Mapeamento do Ambiente		⌛
Uso de Informações Topológicas		✓✓
Uso de Informações Semânticas		⌛

(✓✓ - Muito Bom; ✓ - Bom; ⇩ - Ruim; ⌛ - A Ser Implementado; × - Incapaz)

Tab. 7.10 Resumo de Capacidade e Limitações do Sistema de Controle

Capítulo 8 - Conclusão

Apresentamos nesta dissertação os principais tipos de sistemas e arquiteturas de controle para robôs móveis autônomos, bem como o estado da arte dos sistemas de controle desenvolvidos pelos pesquisadores da robótica móvel autônoma. Foram estudadas as principais técnicas de navegação das abordagens sensorial/reativa, *roadmap* e matricial. Foi realizado também um estudo sobre o estado da arte das técnicas utilizadas para a localização de robôs móveis autônomos. Foram apresentados também os principais modelos utilizados pela robótica móvel: modelos de ambiente, modelos cinemáticos e os modelos sensoriais. Foi realizado também um estudos dos principais simuladores disponíveis para robôs móveis autônomos.

Foi apresentada uma proposta de uma arquitetura de controle híbrida (COHBRA) que busca integrar em uma única estrutura as melhores características dos métodos estudados, considerando diferentes arquiteturas e sistemas de controle, bem como métodos de navegação e localização de robôs móveis autônomos. A arquitetura COHBRA possui uma estrutura de controle dividida em três camadas: camada vital que é responsável pelo controle reativo, camada funcional responsável pelo controle de execução dos planos, e camada deliberativa que é encarregada das tarefas de planejamento a longo prazo. A arquitetura COHBRA possui um módulo localizador que tem a função de estimar a posição do robô móvel no ambiente em relação a um mapa utilizando as informações sensoriais. A representação interna do ambiente (mapa) foi dividida em múltiplas camadas, cada uma melhor adaptada a uma determinada tarefa do sistema de controle: camada poligonal, camada matricial, e camada topológica/semântica. A comunicação entre os diversos módulos da arquitetura COHBRA é realizada através de um depósito central de informações, chamado de memória compartilhada, onde os módulos disponibilizam informações relevantes para outros módulos da arquitetura.

Foi desenvolvido o simulador de robôs móveis chamado SimRob3D. Utilizando este simulador foi implementado o sistema de controle híbrido proposto a fim de validar a arquitetura de controle proposta. Os resultados obtidos com os experimentos executados com o SimRob3D demonstraram que o sistema de controle foi capaz de se localizar no ambiente: de forma local a partir de uma posição inicial conhecida; de forma global sem possuir nenhuma informação inicial sobre a sua posição; e se relocalizando quando detectava um erro na sua estimativa de posição. O sistema de controle conseguiu se localizar mesmo na presença de obstáculos móveis, mostrando que o filtro de distância utilizado foi capaz de diferenciar obstáculos conhecidos e estáticos de obstáculos móveis ou desconhecidos.

Os resultados mostraram que o sistema de controle foi capaz de controlar o robô móvel autônomo e executar uma tarefa de navegação tanto em um ambientes estáticos como em ambientes dinâmicos com obstáculos móveis. O robô móvel foi capaz de seguir as trajetórias globais calculadas sem colidir com nenhum obstáculo ou ficar preso em mínimos locais, mesmo em um ambiente dinâmico. Considerando o item 3.3 (métodos de navegação), a arquitetura proposta atingiu plenamente os objetivos de realizar uma navegação global com um baixo custo computacional, evitando os mínimos locais em um ambiente dinâmico. Isto demonstra que foi possível através da

combinação dos diferentes métodos obter um sistema de controle que integra as melhores características de cada um destes métodos.

Os experimentos revelaram que o robô consegue manter uma boa estimativa de posição durante as tarefas de navegação, mantendo uma posição bastante precisa na maior parte do tempo. O uso das informações topológicas conseguiu aumentar de forma satisfatória a performance do planejamento de trajetórias.

A principal contribuição desta dissertação é a proposta de uma nova arquitetura de controle para robôs móveis autônomos, que foi validada através de experimentos e se mostrou robusta e capaz de operar em ambientes dinâmicos. A arquitetura COHBRA se mostrou capaz de operar um robô móvel autônomo em ambientes em constante alteração e na presença de obstáculos móveis, foi capaz de localizar o robô nestes ambientes mesmo quando o ambiente não refletia por completo o modelo armazenado na representação interna (mapa).

Como perspectiva para trabalhos futuros pretendemos incrementar o sistema de controle implementando o módulo de inserção e remoção de obstáculos no mapa, e implementar também um módulo que utilize as informações semânticas sobre o ambiente para melhorar ainda mais a performance do sistema de controle. Pretendemos implementar o sistema de controle em um robô móvel real, validando assim a arquitetura de controle COHBRA em um ambiente do mundo real.

Apêndice A

Esqueleto básico de um programa controlador (em linguagem C):

```
int SendMsg(UINT msg,WPARAM wParam0, WPARAM wParam1) {
   (*pSendMsg)(msg,wParam0,wParam1);
   return 1; }

void AdicionaRobo(STRUCT_ROBO novorobo){
   (*pAdicionaRobo)(novorobo); }

void AdicionaSensor(long robo, STRUCT_SENSOR novosensor){
   (*pAdicionaSensor)(robo,novosensor); }

float LerSensor(long robo, long sensor){
   return (*pLerSensor)(robo,sensor); }

float LerSensorInterno(long robo, long sensor){
   return (*pLerSensorInterno)(robo,sensor); }

float dist_uniforme(){
   return (*pdist_uniforme)(); }

float dist_normal(float m, float s) {
   return (*pdist_normal)(m,s); }

BOOL APIENTRY DllMain( HANDLE hModule, DWORD  ul_reason_for_call, LPVOID lpReserved )
{
    HMODULE hMod;

   gInst = (HINSTANCE)hModule;

   switch (ul_reason_for_call)
   {
      case DLL_PROCESS_ATTACH:

         hMod=GetModuleHandle("simrob3d.exe");
         if(hMod)
         {
            pSendMsg=(int(*)(UINT ,WPARAM , WPARAM )) (GetProcAddress(hMod,"SendCmd"));
            pAdicionaRobo = (void(*)(STRUCT_ROBO))(GetProcAddress(hMod,"AdicionaRobo"));
            pAdicionaSensor = (void(*)(long,STRUCT_SENSOR))(GetProcAddress(hMod,"AdicionaSensor"));
            pLerSensor = (float(*)(long,long))(GetProcAddress(hMod,"LerSensor"));
            pLerSensorInterno = (float(*)(long,long))(GetProcAddress(hMod,"LerSensorInterno"));
            pdist_uniforme = (float(*)())(GetProcAddress(hMod,"dist_uniforme"));
            pdist_normal = (float(*)(float,float))(GetProcAddress(hMod,"dist_normal"));

            if(!pSendMsg)
            {
               MessageBox(NULL,"Erro em simrob3d.exe! Funcoes de Comando nao Implementadas","Error",MB_ICONERROR);
               return FALSE;
            }

            if(!pAdicionaRobo)
            {
               MessageBox(NULL,"Erro em simrob3d.exe! Funcao AdicionaRobo nao Implementada","Error",MB_ICONERROR);
               return FALSE;
            }

            if(!pAdicionaSensor)
            {
               MessageBox(NULL,"Erro em simrob3d.exe! Funcao AdicionaSensor nao Implementada","Error",MB_ICONERROR);
               return FALSE;
            }

            if(!pLerSensor)
            {
               MessageBox(NULL,"Erro em simrob3d.exe! Funcao LerSensor nao Implementada","Error",MB_ICONERROR);
               return FALSE;
            }
```

```
            if(!pLerSensorInterno)
            {
               MessageBox(NULL,"Erro em simrob3d.exe! LerSensorInterno nao Implementada", "Error", MB_ICONERROR);
               return FALSE;
            }

            if(!pdist_uniforme)
            {
               MessageBox(NULL,"Erro em simrob3d.exe! dist_uniforme nao Implementada","Error",MB_ICONERROR);
               return FALSE;
            }

            if(!pdist_normal)
            {
               MessageBox(NULL,"Erro em simrob3d.exe! dist_normal nao Implementada","Error",MB_ICONERROR);
               return FALSE;
            }
         }
         break;

      case DLL_THREAD_ATTACH:
         break;
      case DLL_THREAD_DETACH:
         break;
      case DLL_PROCESS_DETACH:
         break;
   }
   return TRUE;
}

// Funções chamadas pelo Simulador

DLL_API void Init(void)
{
        // Inicializa o Controlador
}

DLL_API void UpdateControl(void)
{
        // Função chamada a cada passo da Simulação
}

DLL_API WORD OnProc(WPARAM wParam0,WPARAM wParam1)
{
        // Comandos enviados pelo Simulador
}

LRESULT CALLBACK Dialogo(HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
        // Mensagens enviadas pelos controles da janela do controlador
}

DLL_API void OnMouse(WPARAM message,int x, int y, WPARAM botao, HWND janela)
{
        // Comandos de Mouse
}

DLL_API void OnChar(UINT nChar,BYTE down)
{
        // Comandos do Teclado
}

DLL_API void InfoDlg(void)
{
        // Mostra Informações sobre o Controlador
}

DLL_API void DrawInfo(HDC dc, RECT cRect)
{
        // Fornece um Contexto de Dispositivo para o controlador desenhar na tela
}
```

Apêndice B

API disponível para o controlador (funções disponibilizadas pelo Simulador):

float dist_uniforme()

Retorna um valor pseudo aleatório entre 0 e 1 com uma distribuição uniforme.

float dist_normal(float m, float s)

Retorna um valor pseudo aleatório entre 0 e 1 com uma distribuição normal.
Parâmetros: m – média; s – desvio padrão;

AdicionaRobo(STRUCT_ROBO robo)

Adiciona um novo robô móvel no ambiente.
Parâmetros:
STRUCT_ROBO
short tipo; // Tipo de Robô (Ackerman ou Diferencial)
float x; // Posição inicial do robô móvel no ambiente
float y;
float z;
float ang_dir; // Direção inicial do robô móvel no ambiente
float largura; // Largura do Robô
float comprimento; // Comprimento do Robô
float raio_roda; // Raio das rodas do Robô
float largura_roda; // Largura das rodas do Robô

AdicionaSensor(long robo, STRUCT_SENSOR novosensor)

Adiciona um sensor no robô especificado.
Parêmetros: robo – identificador do robô
STRUCT_SENSOR
short tipo; // tipo de sensor
float x; // posição do sensor em relação ao robô
float y;
float z;
float rot; // direção do sensor em relação ao robô
float abertura; // ângulo de abertura do cone do sensor
float num_raios; // número de raios
float erro; // erro médio
float minrange; // valor mínimo
float maxrange; // valor máximo

float LerSensor(long robo, long sensor)

Retorna o valor atual do sensor especificado (sensores de distância).
Parâmetros: robo – identificador do robô; sensor – identificador do sensor;

float LerSensorInterno(long robo, long sensor)

Retorna o valor atual do sensor especificado (sensores internos – encoders de velocidade e direção).
Parâmetros: robo – identificador do robô; sensor – identificador do sensor;

SendCmd(UINT msg, WPARAM wparam0, WPARAM wparam1)

Envia comandos para os atuadores do robô especificado.
Parâmetros:
msg – comando:
0 - parar robo
1 - velocidade
2 - Virar (relativa)
3 - Virar (absoluta)
wparam0, wparam1 – parâmetros do comando;

Bibliografia

[Alb89] J. S. Albus, H. G. McCain, R. Lumia. **NASA/NBS Standard Reference Model for Telerobot Control System Architecture.** Tech. Report, NASA, 1989.

[Ama89] Amari, S. **Dynamical Stability of Formation of Cortical Maps.** In Dynamic Interaction Neural Networks: Models and Data, Springer-Verlag, pp. 15-34. 1989.

[And83] Andrews, J. R. and Hogan, N. **Impedance Control as a Framework for Implementing Obstacle Avoidance in a Manipulator.** Control of Manufacturing Processes and Robotic Systems, Eds. Hardt, D. E. and Book, W., ASME, Boston, pp. 243-251. 1983.

[Aok96] Aoki T. et al. **Acquisition of Optimal Action Selection to Avoid Moving Obstacles in Autonomos Mobile Robot.** Proceedings of ICRA-96, Minneapolis, MN, 22-28 Apr, pp 2055-60. 1996.

[Ark87] R. C. Arkin. **Motor Schema Based Navigation for a Mobile Robot: an Aproach to Programming to Behavior.** IEEE Int. Conf. on Robotics and Autonomation, 1987.

[Ark89] Arkin, R. C., **Motor Schema-Based Mobile Robot Navigation.** The International Journal of Robotics Research, pp. 92-112. 1989.

[Ark90] R. C. Arkin **Integrating Behavioral, Perceptual, and World Knowledge in Reactive Navigation.** Robotics and Autonomous Systems 6, 1990.

[Ark97] R. C. Arkin, T. Balck. **AuRA: Principles and Pratice in Reviews.** Journal of Experimental and Theoretical AI, 2-3:175-189. 1997.

[Ark98] R. C. Arkin. **Behaviour-Based Robotics.** The MIT Press, 1998.

[Avi90] Avis D. and Imai H. **Locating a Robot With Angle Measurements.** Journal of Symbolic Computation, no. 10, pp. 311-326. 1990.

[Bas89] Basye, K.; Dean, T. and Vitter, J. S. **Coping with uncertainty in map learning.** Proc. of the Eleventh International Joint Conference on Artificial Intelligence, pp. 663-668. 1989.

[Bee93] R. D. Beer, R. E. Ritzmann, T. M. McKenna. **Biological Neural Networks in Invertebrate Neuroethology and Robotics.** Boston Academic Press, 1993.

[Beo95] Beom H. R. and Cho H. S. **A Sensor Based Navigation for a Mobile Robot Using Fuzzy Logic and Reinforcement Learning.** IEEE Transactions on Systems, Man and Cybernetics, vol. 25, no. 3, pp 464-77. 1995.

[Ber95] K. Berns, R. Dillman, S. Piekenbrock. **Neural Networks for the Controle of a Six-Legged Walking Machine**. Robotics and Autonomous Systems 14. 1995.

[Bon97] R. P. Bonasso., et al. **Experiences with an Architecture for Intelligent Reactive Agents.** Journal of Experimental and Theoretical AI, 9(2). 1997.

[Bor89] Borenstein, J. and Koren, Y. **Real-time Obstacle Avoidance for Fast Mobile Robots.** IEEE Transac-tions on Systems, Man, and Cybernetics, Vol. 19, No. 5, pp. 1179-1187. 1989.

[Bor91] Borenstein J. and Koren Y. **The Vector Field Histogram – Fast Obstacle Avoidance for Mobile Robots.** IEEE Journal of Robotics and Automation, 7(3): 278-288. 1991.

[Bor94] Borenstein, J. and Feng, L., **UMBmark: A Method for Measuring, Comparing, and Correcting Dead-reckoning Errors in Mobile Robots**. *Technical Report, The University of Michigan UM-MEA*M-94-22. 1994.

[Bor96] Borenstein, J.; Everett, H. R. e Feng, L. **Where am I? Sensors and Methods for Mobile Robot Positioning.** Technical Report. University of Michigan. 1996.

[Bor96b] J. Borenstein, B. Everett, and L. Feng. **Navigating Mobile Robots: Systems and Techniques.** A. K. Peters, Ltd., Wellesley, MA, 1996.

[Bra87] Bratley, P.; Fox, B. L. and Schrage, L. E. **A Guide to Simulation.** Springer-Verlag, New York, 2nd Edition. 1987.

[Bri79] M. Briot; J.C. Talou and G. Bauzil. **Le Systeme de Perception du Robot HILARE**. 2eme Congress AFCET/IRIA, Toulouse, France. 1979.

[Bro86] R. A. Brooks. **A robust Layered Control System for a Mobile Robot.** IEEE Journal of Robotics and Automation RA-2, p. 14-23, 1986.

[Bro91] R. A. Brooks. **Intelligence without reason.** In Proc. Intl. Joint Conf. Artificial Intelligence, pages 569-595, 1991.

[Bro91b] Brooks, R. A. **Artificial Life in Real Robots.** Proc. of European Conference on Artificial Life. 1991.

[Byr92] Byrne, R.H., Klarer, P.R., and Pletta, J.B. **Techniques for Autonomous Navigation.** Sandia Report SAND92-0457, Sandia National Laboratories, Albuquerque, NM. 1992.

[Cor90] T. Cormen, C. Leiserson, R. Rivest. **Introduction to Algorithms**. MIT Electrical Engineering and Computer Science Series. MIT Press. 1990.

[Cox91] Cox I. J. **Blanche – An Experiment in Guidance and Navigation of an Autonomous Robot Vehicle.** IEEE Transactions on Robotics and Automation 7, pp. 193-204. 1991.

[Cro95] Crowley J. L. **Mathematical Foundations of Navigation and Perception for an Autonomous Mobile Robot.** Reasoning With Uncertainty in Robotics. Amsterdam, Netherlands, pp 9-51. 1995.

[Cro98] Crowley, J. L.; Wallner F. and Schiele B. **Position Estimation Using Principal Components of Range Data.** Proc. of the IEEE International Conference on Robotics and Automation (Leuven, Belgium), pp.3121-3128. 1998.

[Cyb02] Cyberbotics, **Professional Mobile Robot Simulation.** Ultima Atualização: Mai 2002. "http://www.cyberbotics.com".

[Die98] F. Dieter. **Markov Localization: A Probabilistic Framework for Mobile Robot Localization and Navigation.** Institute of Computer Science III, University of Bonn, Germany. Doctoral Thesis. 1998.

[Dou98] Doucet, A. **On Sequential Simulation-based Methods for Bayesian Filtering.** Tech. Report, Dept. of Engineering, Univ. of Cambridge. 1998.

[Dud00] G. Dudek and M. Jenkin. **Computational Principles of Mobile Robotics.** Cambridge University Press, Cambridge, UK. 2000.

[Dud96] Dudek G. and Zhang C. **Visin-based Robot Localization Without Explicit Object Models.** Proc of the IEEE International Conference on Robotics and automation. 1996.

[Elf87] Elfes A. **Sonar-Based Real-World Mapping and Navigation.** IEEE Journal of Robotics and Automation, RA-3(3):249-265, 1987.

[Elf89] Elfes A. **Occupancy Grids: A Probabilistic Framework for Robot Perception and Navigation.** PhD Thesis, Carnegie Mellon University. 1989.

[End00] Endo Y. *et al* **Missionlab:User Manual for Missionlab 4.0.** Technical Report, College of Computing, Georgia Institute of Technology. 2000.

[Fol94] J.D. Foley. **Introduction to Computer Graphics.** Reading : Addison-Wesley, xxviii, 557p. 1994.

[Fox98] D. Fox. **Markov Localization: A Probabilistic Framework for Mobile Robot Localization and Navigation.** Institute of Computer Science III, University of Bonn, Germany. Doctoral Thesis. 1998.

[Fox99] Fox, D.; Burgard, W.; Thrun, S. **Markov Localization for Mobile Robots in Dynamics Environments.** Journal of Artificial Intelligence Research, Volume 11, p 391-427. 1999.

[Fox99b] D. Fox, W. Burgard, F. Dellaert, and S. Thrun. **Monte Carlo localization: Efficient position estimation for mobile robots.** In Proc. of the National Conference on Artificial Intelligence (AAAI). 1999.

[Fra93] Fraichard T. and Laugier C. **Dynamic Trajetory Planning, Path-Velocity Decomposition and Adjacent Paths.** Proceedings of IJCAI-93, Chambery, France, 28 Aug - 3 Sep, pp 1592-7. 1993.

[Gat92] E. Gat. **Integrating Planning and Reacting in a Heterogeneous Asynchronous Architecture for Controlling Real-World Mobile Robots.** AAAI-92 Proceedings, AAAI Press, 1992.

[Gla95] Glasius, R.; Komoda, A.; Gielen, S. **Neural Network Dynamics for Path Planning and Obstacle Avoidance.** Neural Networks 8, 1, pp. 125-133. 1995.

[Gut96] Gutmann, J. S. and Schlegel, C. **Amos: Comparison of Scan Matching Approaches for Self-Localization in Indoor Environments.** In Proceedings of the 1st Euromicro Workshop on Advanced Mobile Robots, pp. 61-67. 1996.

[Hal90] J. Hallam. **Intelligent Sensing and Control: Notes on Control Theory (part 2).** Lecture Notes, Department of AI, University of Edinburgh. 1990.

[Hay85] B. Hayes-Roth. **A Blackboard Architecture for Control.** Artificial Intelligence 26, 1985.

[Hei00] F. J. Heinen. **Robótica Autônoma: Integração entre Planificação e Comportamento Reativo.** Série Produção Discente, Editora Unisinos. São Leopoldo, RS. 2000.

[Hei01] Heinen, Osorio, Fortes. **Controle inteligente de veículos autônomos: automatização do processo de estacionamento de carros**. Anais do X SEMINCO-FURB, Blumenau, SC. 2001.

[Isa98] Isard, M. and Blake, A. **Condensation-Conditional Density Propagation for Visual Tracking.** International Journal of Computer Vision 29(1). 1998.

[Kal60] Kalman, R. E. **A new approach to linear filtering and prediction problems.** Trans. of the ASME, Journal of basic engineering, 82:35-45.1960.

[Kam97] Kamon, I. and Rivlin, E. **Sensory-Based Motion Planning with Global Proofs**. IEEE Transactions on Robotics and Automation, 13 (6), pp. 814-821. 1997.

[Kha90] Khatib, O. **Real-Time Obstacle Avoidance for Manipulators and Mobile Robots.** IEEE International Conference on Robotics and Automation, St. Louis, Missouri, pp. 500-505. March 25-28, 1990.

[Kir81] Kirkpatrick, S. and Stoll, E. P. **A Very Fast Shift-register Sequence Random Number Generator.** Journal of Computational Physics, 40:517-526. 1981.

[Kit96] Kitagawa, G. **Monte Carlo Filter and Smoother for Nongaussian Nonlinear State Space Models.** Journal of Computacional and Graphical Statistics 5(1). 1996.

[Knu81] Knuth, D. E. **Seminumerical Algorithms**. Volume 2 of The Art of Computer Programming, chapter 3. Addison-Wesley, Reading, MA, 2nd Edition. 1981.

[Kor98] D. Kortenkamp, R.P. Bonasso, and R. Murphy, editors. **AI-based Mobile Robots: Case studies of successful robot systems.** MIT Press, Cambridge, MA, 1998.

[Kro86] Krogh, B. H. and Thorpe, C. E. **Integrated Path Planning and Dynamic Steering Control for Autonomous Vehicles.** Proceedings of the 1986 IEEE International Conference on Robotics and Automation, San Francisco, California, pp. 1664-1669. April 7-10, 1986.

[Kui88] Kuipers, B. J. **A Robust, Qualitative Method for Robot Spatial Learning.** Proc. of the Seventh National Conference on Artificial Intelligence, pp. 774-779, Saint Paul, Minnesota. 1988.

[Kui91] Kuiper B. and Byun Y. **A Robot Exploration and Mapping Strategy Based on a Semantic Hierarchy of Spatial Representations.** Robotics and Autonomous Systems 8, pp. 47-63. 1991.

[Lai89] J. E. Laird, E. S. Yager, C. M. Tuck, M. Hucka. **Learning in Tele-autonomous Systems using SOAR.** NASA Conf. On Space Robotics Proceedings, 1989.

[Leh51] Lehmer, D. H. **Mathematical Methods in Large-Scale Computing Units.** In 2nd Symposium on Large-Scale Digital Calculating Machinery, pp. 141-146, Cambridge, MA. Harvard University Press. 1951.

[Len90] Lengyel J. et al. **Real--time robot motion planning using rasterizing computer graphics hardware.** In Proc. of SIGGRAPH, pages 327--335, Dallas, Texas, 1990.

[Leo90] J. Leonard, H. Durrant-Whyte, and I.J. Cox. **Dynamic map building for an autonomous mobile robot.** In Proc. IEEE Int. Conf. on Robotics and Automation, 1990.

[Leo91] Leonard, J. J. and Durrant-Whyte, H. F. **Mobile robot localization by tracking geometric beacons.** IEEE Transactions on Robotics and Automation, 7(3):376-382. 1991.

[Lew73] Lewis, T. G. and Payne, W. H. **Generalized Feedback Shift Register Pseudorandom Number Algorithm.** Journal of the Association for Computing Machinery, 20(3):456-468. 1973.

[Lu97] Lu, F. and Milios, E. **Robot Pose Estimation in Unknown Environments by Matching 2D Range Scans.** Journal of Intelligent and Robotic Systems, 18:249-275. 1997.

[Luc99] G.W. Lucas. **Rossum's Playhouse (RP1) Homepage.** Ultima Atualização: Jun 1999. “http://rossum.sourceforge.net/sim.html”.

[May90] Maybeck, P. S. **The Kalman Filter: An Introduction to Concepts.** Autonomous Robot Vehicles. Springer-Verlag, Berlin. 1990.

[Med98] Adelardo A.D. Medeiros. **A Survey of Control Architectures for Autonomous Mobile Robots.** JBCS - Journal of the Brazilian Computer Society, special issue on Robotics, vol. 4, n. 3 (ISSN 0104-650004/98). 1998.

[Mic97] Olivier Michel, **Khepera Simulator Homepage.** Ultima Atualização: Dez 1997. "http://diwww.epfl.ch/lami/team/michel/khep-sim/".

[Mil85] Miller, D. **A Spatial Representation system for mobile robots.** Proc. of the IEEE International Conference on Robotics and Automation, pp. 122-127, St. Louis, Missouri. 1985.

[Mir02] Gonzalo Rodriguez Mir, **MobotSim Homepage.** Ultima Atualização: Fev 2002. "http://www.mobotsim.com".

[Mit89] Mitchell et al. **Theo: A framework for self-improving systems.** Architectures for Intelligence. Erl-baum.1989.

[Mit90] T.M. Mitchell, **Becoming increasingly reactive (mobile robots).** AAAI-90 Proceedings, MIT Press. 1990.

[Moo01] Moon, I.; Miura, J.and Shirai, Y. **Automatic Extraction of Visual Landmarks for a Mobile Robot Under Uncertainty of Vision and Motion.** Proc. of IEEE Int. Conf. on Robotics and Automation, pp. 1188-1193, Korea. 2001.

[Mor85] Moravec, H. P. and Elfes, A. **High Resolution Maps from Wide Angle Sonar.** IEEE Conference on Robotics and Automation, Washington D.C., pp. 116-121. 1985.

[Mor88] Moravec, H. P. **Sensor fusion in certainty grids for mobile robots.** AI Magazine 61-74. 1988.

[Mor90] H. Moravec. **The Stanford CART and the CMU Rover**. Autonomous Robot Vehicles, I.J. Cox and G.T. Wilfong eds. Springer-Verlag. 1990.

[New90] A. Newell. **Unifield Theories of Cognition.** Harvard University Press, 1990.

[Nie92] Niederreiter, H. **Random Number Generation and Quasi-Monte Carlo Methods.** Tech. Report, SIAM, Philadelphia. 1992.

[Nil69] N. J. Nilsson. **A Mobile Automaton: An Application of Artificial Intelligence Techniques.** Proc. of IJCAI, reprinted in Autonomous Mobile Robots: Control, Planning and Architecture, 2(1) :233-239. 1969.

[Nil80] Nilsson N. J. **Principles of Artificial Intelligence**. Tioga Plublishing Company. 1980.

[Oor97] Oore S.; Hinton G. and Dudek G. **A Mobile Robot That Learns its Place.** Neural Computation 3, no. 9, pp. 683-699. 1997.

[Oro93] J. O´Rourke. **Computacional Geometry in C.** Cambridge University Press. 1993.

[Pan90] G. Pang, H. C. Shen. **Intelligent Control of an Autonomous Mobile Robot in a Hardardous Material Spill Accident - a Blackboard Structure Approach.** Robotics and Autonomous Systems 6, 1990.

[Par88] Park, S. K. And Miller, K. W. **Random Number Generators: Good Ones are Hard to Find.** Communications of the ACM, 31(10):1191-1200. 1988.

[Per79] Lozano-Perez, T., and Wesley, M. A. **An algorithm for planning collision-free paths among polyhedral objects.** Communications ACM 22, 10. 1979.

[Rei85] Reif J. and Sharir M. **Motion Planning in the Presence of Moving Obstacles.** Proceedings of the 26th Annual IEEE Symposium on Foundations of Computer Science, Portland, OR, pp 144-54. 1985.

[Ros93] P. Rosembloom, J. Laird, A.Newell. **The SOAR Papers: Research on Integrated Intelligence.** MIT Press, 1993.

[Ros95] J. K. Rosenblatt, C. Thorpe. **Combining Multiple Goals in a Bahavior-based Architecture.** International Conference on Intelligent Robots and Systems (IROS) Proceedings, 1995.

[Row79] Rowat, P. F. **Representing the spatial experience and solving spatial problems in a simulated robot environment.** Ph.D. thesis, University of British Columbia. 1979.

[Rub81] Rubinstein, R. Y. **Simulation and the Monte Carlo Method**. Wiley Series in Probability and Mathematical Statistics. John Wiley & Sones, New York. 1981.

[Rub88] Rubin, D. **Using the SIR Algorithm to Simulate Posterior Distributions.** Bayesian Statistics 3. Oxford University Press. 1988.

[Rus95] S. Russel, P. Norvig, **Artificial Intelligence a modern approach**, Prentice Hall 1995.

[Sam90] Samson C, and Ait-Abderrahim K. **Mobile robot control, part 1: Feed-back control of a nonholonomic wheeled cart in cartesian space.** Technical Report 1288, INRIA. 1990.

[Sha96] M. Shaw, D. Garlan. **Software Architecture, Perspectives on an Emerging Discipline**, Prentice Hall 1996.

[Shu93] Shutherland, K. T. and Thompson, W. B. **Inexact Navigation.** Proc. of IEEE Int. Conf. on Robotics and Automation, pp. 1-7, 1993.

[Sim95] Simmons R. and Koenig S. **Probabilistic Robot Navigation in Partially Observable Environments.** Proceedings of the International Joint Conference on Artificial Intelligence, pp. 1080-1087. 1995.

[Smi87] Smith R. and Cheeseman P. **On the Estimation and Representantion of Spatial Uncertainty.** International Journal of Robotics Research, vol. 5, no. 4, pp 56-68. 1987.

[Smi90] T. Smithers. **Intelligent Sensing and Control: Control and Control Strategies.** Lecture Notes, Department of AI, University of Edinburgh. 1990.

[Ste94] Stentz A. **Optimal and Efficient Path Planning for Partially-Known Environments.** Proceedings of the IEEE International Conference on Robotics and Automation. 1994.

[Sug88] Sugihara K. **Some Location Problems for Robot Navigation Using a Single Camera.** Computer Vision, Graphics, and Image Processing 42, pp. 112-129. 1988.

[Suz90] Suzuki H. and Arimoto A. **A Recursive Method of Trajectory Planning for a Point-Like Mobile Robot in Transient Environment Utilizing Paint Procedure.** Journal of the Robotics Society of Japan, val. 8, pp 9-19. 1990.

[Tau65] Tausworthe, R. C. **Random Numbers Generated by Linear Recurrence Modulo Two.** Mathematics of Computation, 19:201-209. 1965.

[Thr96] Thrun, S. and Bücken, A. **Learning maps for indoor mobile robot navigation.** Tech. Report CMU-CS-96-121, Carnegie Mellon University. 1996.

[Til89] Tilove, R. B. **Local Obstacle Avoidance for Mobile Robots Based on the Method of Artificial Potentials.** General Motors Research Laboratories, Research Publication GMR-6650. September 1989.

[Too71] Toothill, J. P. R.; Robinson, W. D. and Adams, A. G. **The Runs Up-and-Down Performance of Tausworthe Pseudo-Random Number Generators.** ACM Journal, 18(3):381-399. 1971

[Tor92] Torrance, M. C. **The Case for a Realistic Mobile Robot Simulator.** Working Notes of the AAAI Fall Symposium on Applications of Artificial Intelligence to Real-World Autonomous Mobile Robots, Cambridge, MA. 1992.

[Tri02] Trinity College. **Fire-Fighting Home Robot Contest Website.** Ultima Atualização: Abr 2002. “http://www.trincoll.edu/events/robot/”.

[Wal50] W. G. Walter. **An Imitation of Life**. Scientific American, 182(5) :42-45. 1950.

[Wei95] Weiss, G. and Puttkamer, E. **A Map Based on Laserscans Without Geometric Interpretation.** Intelligent Autonomous systems (IAS-4), IOS Press, pp. 403-407. 1995.

[Woo94] Wooldrige, M. e Jennings, N. **Intelligent Agents: Theory and Pratice.** Knowledge Engineering Review, 1994.

[Woo98] Mason Woo. **OpenGL Programming Guide: The Official Guide to Learning OpenGL.** Addison-Wesley, 650p. 1998.