VITOR CAMPANHOLO GUIZILINI

ESTUDO E IMPLEMENTAÇÃO DE UMA SOLUÇÃO PARA O PROBLEMA DA LOCALIZAÇÃO E MAPEAMENTO SIMULTÂNEOS EM ROBÔS MÓVEIS

São Paulo
2006

VITOR CAMPANHOLO GUIZILINI

nota final 9.6
(nove e seis)

ESTUDO E IMPLEMENTAÇÃO DE UMA SOLUÇÃO PARA O PROBLEMA DA LOCALIZAÇÃO E MAPEAMENTO SIMULTÂNEOS PARA ROBÔS MÓVEIS

Monografia apresentada à Escola Politécnica da Universidade de São Paulo para a obtenção do título de Engenheiro.

São Paulo
2006

VITOR CAMPANHOLO GUIZILINI

ESTUDO E IMPLEMENTAÇÃO DE UMA SOLUÇÃO PARA O PROBLEMA DA LOCALIZAÇÃO E MAPEAMENTO SIMULTÂNEOS PARA ROBÔS MÓVEIS

Monografia apresentada à Escola Politécnica da Universidade de São Paulo para a obtenção do título de Engenheiro.

Área de Concentração:
Engenharia Mecatrônica e de Sistemas Mecânicos.

Orientador:
Prof. Livre Docente Jun Okamoto Jr.

São Paulo
2006

FICHA CATALOGRÁFICA

Guizilini, Vitor Campanholo
Estudo e implementação de uma solução para o problema da localização e mapeamento simultâneos em robôs móveis / V.C. Guizilini. -- São Paulo, 2006.
p.

Trabalho de Formatura - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos.

1.Robôs (Simulação computacional) 2.Sensor 3.Sistemas autônomos I.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos II.t.

AGRADECIMENTOS

À minha família, por ter feito de mim o que eu sou.

Ao meu orientador, por acreditar em mim e no meu trabalho.

Aos integrantes do LPA, por me apoiarem sempre que eu precisei.

Aos meus amigos, pelos bons momentos e pelo apoio nas horas ruins.

Aos aventureiros de Treitore, pelas muitas horas de diversão e imaginação.

"Our deepest fear is not that we are inadequate.
Our deepest fear is that we are powerful beyond measure."

(Marianne Williamson)

RESUMO

Este projeto tem como objetivo final o estudo e a implementação de uma solução para o problema do SLAM em robôs móveis, permitindo o mapeamento dinâmico de ambientes desconhecidos e a sua localização dentro desse mesmo mapa. Como abordagem para esse problema foi escolhido o *DP-SLAM*, que consiste em um *Filtro de Partículas* gerenciando as diversas posições possíveis (partículas) que o robô pode assumir em um determinado instante. A cada partícula está vinculado um mapa incompleto do ambiente, e através das observações dos sensores são definidos os mapas que melhor representam a realidade. As partículas relativas a esses mapas são então mantidas e propagadas para as próximas iterações, dando origem a novas partículas, enquanto as outras são descartadas, eliminando-se com isso os erros gerados pela navegação antes que eles acumulem a ponto de invalidar o resultado final. Dessa forma, ambas as etapas, localização e mapeamento, são resolvidas simultaneamente, como é o propósito do SLAM.

ABSTRACT

The objective of this work is the study and implementation of a solution to the problem of SLAM in mobile robots, allowing the dynamic mapping of unknown environments and the localization of the robot inside this very map. It is presented here a solution known as *DP-SLAM*, which consists of a particle filter that manages the possible positions for the robot (particles) in a given instant. Each particle has its own map of the environment, and with the information obtained from the sensors it is possible to determine which maps describe more precisely the reality as it is around the robot. These particles are propagated to the next iterations of the algorythm, while the ones less accurate as discarded, allowing the creation of a larger number of useful particles and eliminating the errors before they have a chance to accumulate and invalidate the final results. This way, both problems ares solved simultaneously, as the definition of the most probable particle defines also the position of the robot, as it is the proposition of SLAM.

LISTA DE ILUSTRAÇÕES

SUMÁRIO

1. Introdução

O SLAM (Simultaneous Localization and Mapping) surgiu como uma tentativa de responder à seguinte pergunta: "É possível, para um veículo autônomo iniciado em uma posição desconhecida, criar de maneira incremental um mapa do ambiente ao seu redor enquanto se utiliza desse mesmo mapa para computar dinamicamente sua localização?". Uma resposta afirmativa para essa pergunta permitiria, em essência, o surgimento de veículos verdadeiramente autônomos, capazes de se movimentar em ambientes desconhecidos e sem acesso a informações externas sobre o seu posicionamento atual.

Devido a essa grande importância, desde o seu surgimento no cenário científico, várias abordagens já foram criadas na tentativa de se mostrar que uma solução para o problema do SLAM realmente é possível, além de desejada. Embora possuam o mesmo objetivo e compartilhem dos mesmos preceitos básicos, cada uma de tais abordagens faz uso de métodos e modelos matemáticos diferentes, diferindo até mesmo nas hipóteses iniciais de resolução, procurando maneiras distintas de se resolver o mesmo problema. Diversas ferramentas matemáticas e computacionais já existentes se mostraram úteis na resolução do problema do SLAM, como o caso de Filtros de Partículas, enquanto outras já foram ou ainda estão sendo desenvolvidas com esse único propósito.

Isso faz com que diferentes métodos possuam diferentes pontos fortes e fracos, cobrindo certas possibilidades reais com uma precisão que não é alcançada nos outros, enquanto falham quando afastados das condições para as quais foram idealizados. Esse comprometimento é inevitável, e por isso deve-se escolher a abordagem mais adequada ao tipo de tratamento que será realizado com os resultados obtidos, assim como outros fatores, como regiões mais prováveis de navegação e obstáculos que serão encontrados. Essa é inclusive uma premissa básica no desenvolvimento de vários dos algoritmos já existentes, que embora criados com um intuito genérico são posteriormente adaptados para se sobressaírem em uma determinada condição especificada a priori.

A necessidade de um sistema autônomo de navegação possuir um algoritmo de SLAM surge devido ao acúmulo de erros que são inerentes à maneira como obtém informações do ambiente ao seu redor e também ao modo como ele calcula sua posição atual. Os sensores, responsáveis pela coleta de informações, estão sujeitos a uma grande quantidade de imprecisões, qualquer que seja o tipo utilizado, como é o caso de sensores baseados em sonar, laser ou infravermelho. Tais imperfeições podem surgir devido à diferença do material visado pelo sensor, erros geométricos criados no momento de sua montagem, movimentação durante a tomada de informações, e muitas outras razões.

O cálculo direto de sua posição atual, usualmente feito a partir de dados da odometria, também rapidamente acumula erros devido a problemas como imperfeições no terreno, escorregamento indesejado das rodas e imperfeições nos seus eixos de movimentação. Dessa forma, em pouco tempo esses erros se acumulam e fazem com que o veículo apresente dados incoerentes com o seu estado atual. Erros de localização terminam por invalidar o reconhecimento do ambiente por parte dos sensores, pois o mapa que será construído levará em consideração uma posição incorreta, gerando uma representação pobre do terreno através do qual o veículo deverá navegar, principalmente após um grande período de navegação ininterrupta e sem informações externas.

Sem tais informações, a implementação de qualquer outro algoritmo, seja de planejamento, controle ou navegação, se torna impossível, pois todos dependem diretamente de um mapeamento e de uma localização fieis. É justamente esse o propósito do SLAM, que faz uso de técnicas que possibilitam realizar o mapeamento e a localização de maneira simultânea, levando em consideração o fato de que ambos são interdependentes, e por isso mesmo não podem ser realizados separadamente, com as informações de uma etapa auxiliando na outra reciprocamente. Através de informações recolhidas pelos sensores e da sua posição anterior define-se qual será a nova localização do robô, e com isso um novo mapa é gerado, fazendo uso dessas mesmas informações recolhidas e também da posição atualizada do robô.

Dessa forma, iterativamente, os erros inerentes tanto aos sensores quanto a odometria são sistematicamente eliminados antes que possam se tornar críticos,

impedindo o seu acúmulo e garantindo um resultado coerente com a realidade mesmo após grandes distâncias já terem sido percorridas. A partir desse momento se torna possível aplicar outros algoritmos que visam resolver outros problemas, e por isso mesmo um resultado satisfatório em um tempo hábil se torna ainda mais importante, pois o algoritmo SLAM deverá agir em conjunto com outros que farão uso de seus resultados.

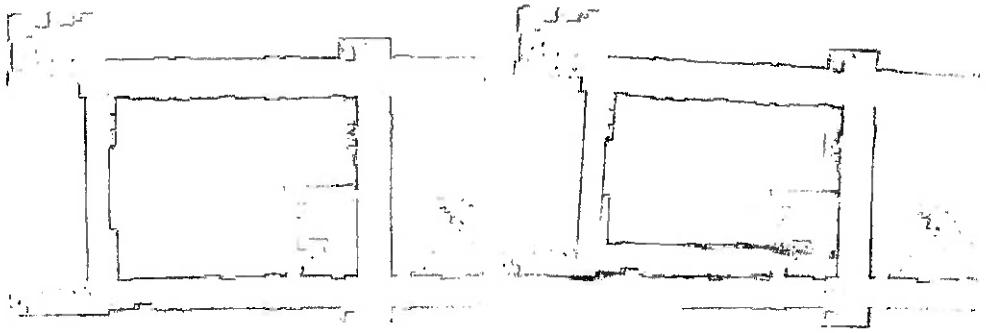
2. Localização e Mapeamento Simultâneos

A solução para o problema da localização e mapeamento simultâneos (SLAM) representa, em muitos aspectos, a maior conquista (Newman, Whyte; 2001) que pode ser alcançada nas pesquisas atuais relacionadas à navegação autônoma de veículos. Isso porque uma resposta afirmativa para o problema do SLAM significaria o surgimento de sistemas capazes de coletar informações do ambiente no qual se encontram de maneira precisa mesmo após longos períodos de tempo, construindo um modelo coerente desse ambiente e sabendo, a cada momento, qual é a sua posição dentro dele. Essa capacidade é de grande importância para diversos outros campos de pesquisa - que dependem dos resultados que o SLAM se propõe a alcançar para que suas aplicações possam ser levadas da teoria à prática - utilizadas em situações reais.

Devido a esse grande valor, desde a sua concepção uma quantidade substancial de pesquisa vem sendo realizada na tentativa de se alcançar resultados que sejam ao mesmo tempo precisos e eficientes. Precisos para que possam ser utilizados da forma correta por aplicações posteriores, e eficientes para que sejam obtidos dentro de um tempo hábil, permitindo aplicações em tempo real. Diversas maneiras de se abordar o problema já foram desenvolvidas e implementadas, obtendo maior ou menor sucesso em determinadas situações, embora ainda não se tenha alcançado uma solução robusta o suficiente para ser utilizada em qualquer tipo de ambiente (Newman, Whyte; 2001).

A proposição básica do SLAM está na realização simultânea das etapas de mapeamento e localização, reconhecendo que ambas são relacionadas entre si, com os resultados de uma influenciando diretamente a qualidade da outra. Abordagens diretas se mostram ineficientes devido aos erros inerentes aos sensores utilizados para a coleta de informações do ambiente e para a localização do robô dentro dele, como é o caso do sistema de odometria. Esses erros não podem ser simplesmente incorporados aos modelos por terem origens diversas e complexas, impossíveis de serem corretamente previstas e, com isso, calculadas previamente (Estrada, Teira, Neirós; 2005).

Erros de localização interferem na construção correta do modelo que o robô tem do ambiente ao seu redor, pois as informações dos sensores são sempre relativas à própria posição do robô, pois estão embarcados nele. E erros na representação do ambiente impedem a localização correta do robô, pois as informações recebidas através de seus sensores não estarão condizentes com o seu modelo do ambiente, impossibilitando o reconhecimento da região onde ele se encontra. Esses pequenos erros gerados a cada iteração rapidamente se acumulam, a ponto de comprometer o resultado final e inviabilizar aplicações posteriores, que façam uso da localização e mapeamento aqui propostos.



(a) (b)

Figura 1 – Importância do SLAM no mapeamento. (Eliazar, Parr; 2003)
(a) Resultados com o SLAM. (b) Resultados sem o SLAM.

A principal abordagem para o problema do SLAM foi introduzida inicialmente por Smith, Self e Cheeseman (1990), que propuseram a utilização do Filtro Extendido de Kalman ("EKF - Extended Kalman Filter") (Welch, Bishop; 2001) para se lidar com as incertezas geradas pelos sensores de um veículo autônomo. Outras possíveis soluções incluíam a evolução da mecânica de precisão, e com isso da precisão dos equipamentos, e também das técnicas de calibração utilizadas. Também nesse trabalho foi proposta a utilização simultânea de diversos sensores para obtenção de informações do ambiente, de forma que seus resultados combinados pudessem fornecer a precisão necessária ao sistema.

O Filtro de Kalman procura determinar a posição do robô a partir da sua posição relativa a um conjunto de landmarks, estruturas naturais ou artificiais que estão distribuídas no ambiente e cuja posição deve ser conhecida. Conforme o robô se

movimenta, a sua posição em relação a esses landmarks se altera, ao mesmo tempo em que novos landmarks entram na região varrida pelos seus sensores e outros desaparecem. Através da maneira como um determinado conjunto de landmarks é observado pelos sensores do robô, pode-se determinar precisamente a sua localização, e ela pode ser corrigida dinamicamente durante a navegação.

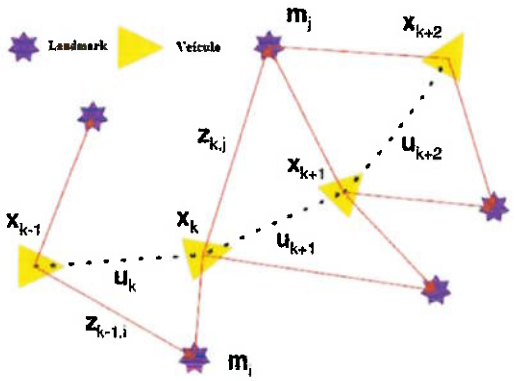


Figura 2 - SLAM realizado com o auxílio de landmarks. (Whyte, 2002)

Trabalhos posteriores tentaram então resolver os principais problemas que essa abordagem apresenta, para que seus resultados permitissem sua utilização em aplicações reais. Entre tais problemas está a dificuldade em se identificar de forma inequívoca cada um dos seus landmarks (Thrun, Fox, Burgard; 1998), e também como definí-los a partir de estruturas naturais do ambiente, eliminando a necessidade da inserção de landmarks artificiais (Choset, Burdick; 2000) (Kuipers, Byun; 1991). Outra limitação do SLAM como apresentado está no rápido crescimento do custo computacional conforme o número de landmarks aumenta (Guivant, Nebot; 2001), obedecendo a um aumento aproximadamente quadrático.

Outras soluções, dentre elas o FastSLAM (Montemerlo, Thrun; 2002), utilizam um *Filtro de Partículas* modificado, que mantém diversos Filtros de Kalman para cada possível posição do robô, levando em consideração a maneira como os landmarks deveriam ser observados caso o robô realmente estivesse naquela posição. Essa abordagem permitiu a diminuição do custo computacional para O(MK), proporcional ao número de partículas M e de landmarks K. Trabalhos posteriores fizeram com que esse valor fosse diminuído para O(Mlog(K)), gerando um aumento de eficiência

computacional ainda maior, permitindo a utilização do FastSLAM em ambientes com grandes quantidades de landmarks. Avanços recentes (Montemerlo, Thrun; 2003) mostram-se capazes até mesmo de realizarem a etapa de localização sem a necessidade de valores de odometria, além de associar corretamente landmarks onde abordagens que fazem uso apenas do EKF inevitavelmente divergirão.

O problema do mapeamento em grande escala também pode ser resolvido de outras maneiras, como proposto em (Estrada, Neira, Tardós; 2005) com a utilização não apenas de um mapa global, mas de vários mapas locais, que são construídos e depois conectados entre si. Isso evita o alto custo computacional de se lidar com todo o mapa já construído, e também diminui a quantidade de landmarks que são utilizadas a cada iteração do algoritmo. Outros autores ainda eliminam a necessidade de criação de mapas ao armazenarem apenas posições relativas entre os diversos landmarks, conseguindo com isso grande controle sobre o tempo de execução do algoritmo e o grau de precisão desejado. Um exemplo dessa abordagem é o GPF (Geometric Projection Filter), descrito em detalhes em (Newman, Whyte; 2006) e que consegue um tempo constante em relação à quantidade de landmarks utilizados.

Já do outro lado estão as abordagens que não fazem uso de landmarks para tentar resolver o problema do SLAM, procurando outras características do ambiente para realizarem o mapeamento e a localização. Dentre essas abordagens a mais utilizada, e a que mais vem evoluindo atualmente é o *DP-SLAM*, proposto inicialmente por Eliazar e Parr em (Eliazar, Parr; 2003) e que também foi implementado pelo autor em questão (Guizilini, 2006) em trabalhos anteriores com algoritmos de SLAM. O *DP-SLAM* elimina a necessidade de landmarks ao manter, ao mesmo tempo, diversos mapas prováveis do ambiente, cada um relacionado a uma possível localização do robô naquele determinado instante. Esses mapas e posições são mantidos através de um *Filtro de Partículas* (Thrun, 2002), com cada partícula sendo uma provável posição e, por conseqüência, também um provável mapa.

Conforme o robô se movimenta, o seu modelo de navegação define quais são suas novas posições prováveis, e novos mapas são gerados a partir daqueles já existentes, possuídos pelas partículas que deram origem às partículas atuais. Apenas as alterações feitas nessa atualização dos sensores é adicionada aos mapas, pois os

mapas antigos com atualizações anteriores são mantidos e podem ser consultados, o que diminui em muito a memória necessária e o tempo de atualização do sistema. Posições que se mostram ineficientes para representar a posição do robô são descartadas juntamente com o seu mapa correspondente, e aquelas mais prováveis são mantidas e propagadas na próxima atualização.

Assim, não existe uma única posição para o robô em um determinado instante, mas sim várias posições prováveis, cada uma com um próprio mapa atribuído. Essa abordagem permite a manutenção de ambigüidades na localização por grandes períodos de tempo sem que o mapa resultante acumule erros, pois quando essa ambigüidade é resolvida os mapas cujas posições foram determinadas como erradas são abandonados sem nenhuma influência no resultado final. Dessa forma o problema do SLAM é resolvido, mas com isso surge um problema computacional, devido ao custo de manutenção de atualização de uma grande quantidade de mapas vinculados às suas partículas correspondentes. Esse problema vem sendo contornado com o desenvolvimento de novas ferramentas de armazenamento de dados, como mostrado em trabalhos posteriores dos mesmos autores (Eliazar, Parr; 2004, 2005).

Figura 3 - Resultado de mapeamento DP-SLAM (distância apxoximada de 220 m). (Eliazar, Parr; 2005)

Seguindo por um caminho diferente, existem as abordagens que procuram uma outra maneira de se modelar o conhecimento que é adquirido através dos sensores dos robôs, armazenando-o de maneiras diferentes àquela mostrada até agora. Métodos apresentados até agora de maneira geral fazem uso dos chamados mapas métricos para armazenar as informações obtidas pelos sensores, que são representações do ambiente através de matrizes, as chamadas grades de ocupação. Conforme o robô se movimenta e coleta informações do ambiente essa células são preenchidas com os valores encontrados, representando a maneira como o sistema supõe que aquela área se encontra. Mapas métricos podem ser tanto binários (com cada célula recebendo o

valor 0 caso esteja livre e 1 caso esteja ocupada ou probabilísticos (recebendo um valor entre 0 e 1, que indica a probabilidade daquela região estar ocupado).

Já os mapas topológicos não se concentram em armazenar todo o conhecimento que o robô adquire, traçando todas as estruturas encontradas, mas sim apenas as regiões que são de alguma forma importantes para a navegação do sistema. Essa abordagem faz uso de grafos como estrutura de armazenamento, com cada nó representando uma região importante e as linhas que os conectam mostrando as diversas maneiras de se movimentar entre essas regiões. Essas linhas permitem a navegação do robô mesmo sem que ele conheça exatamente o caminho que será percorrido, apenas que ambos os locais estão conectados, seja diretamente ou por um conjunto de locais intermediários. Essa é uma forma similar àquela que nós próprios utilizamos para nos locomovermos, memorizando várias instruções simples baseadas em locais de fácil reconhecimento para conseguirmos chegar até lugares mais distantes.

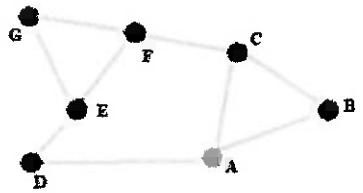


Figura 4 - Exemplo de mapa topológico.

Resultados positivos já foram obtidos com a utilização do SLAM a partir de mapas topológicos (Choset, Nagatani; 2001), que possuem como vantagem um custo computacional menor, já que não necessita da manutenção de toda a informação adquirida de forma explícita. As desvantagens surgem na determinação de quais locais podem ser vistos como importantes para a representação do mapa, e também como reconhecer tais regiões quando o robô passa novamente por elas, após um certo período de tempo. A área de sensores visuais tem feito grandes avanços em direção à resolução de tais problemas (Se, Lowe, Little; 2005) (Folkesson, Jensfelt, Christensen; 2005), fazendo com que o auxílio visual se tornasse uma ferramenta importante em aplicações relacionadas.

Recentemente, têm surgido tentativas de se unir ambas as representações de mapas em um único, criando os chamados mapas híbridos (Kouzoubov, Austin; 2004), que apresentam características de ambos e fazem proveito de suas vantagens. A abordagem topológica facilita a navegação, pois fornece uma trajetória para o robô sem que seja necessária a consulta constante ao mapa, bastando para isso seguir as orientações dos arcos que conectam os dois nós. Já a abordagem métrica permite um maior conhecimento do ambiente ao redor do robô, definindo áreas que ainda não foram exploradas e auxiliando no processo de determinação e conhecimento de regiões importantes para o mapa topológico. Dessa forma, consegue-se aproveitar da síntese que os mapas topológicos oferecem e evita-se a perda de informações que são fornecidas pelos sensores, utilizando-as na criação do mapa métrico para análise posterior.

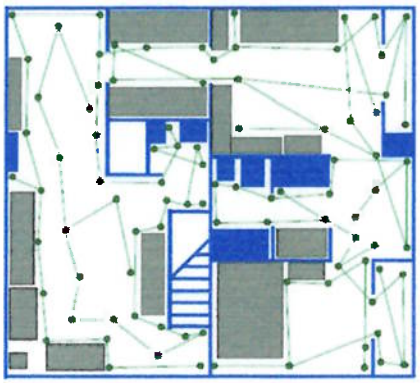


Figura 5 - Mapa híbrido métrico/topológico. (Sujan, Meggiolaro, Bello; 2005)

3. Modelagem

Durante as primeiras etapas da construção do algoritmo de SLAM foi utilizado um ambiente virtual, criado a partir de um software de simulação robótica embarcado em sistema *Linux.* Esse software, o *Gazebo*, é distribuído livremente na Internet e permite a construção de modelos dinâmicos de robôs reais, simulando a maneira como eles se movimentam e interagem com o ambiente ao seu redor. Adicionalmente, é possível modelar também sensores, que adicionados ao modelo virtual permitem que ele colete informações do ambiente virtual, utilizando-as para as mesmas tarefas que um robô real poderia fazer.

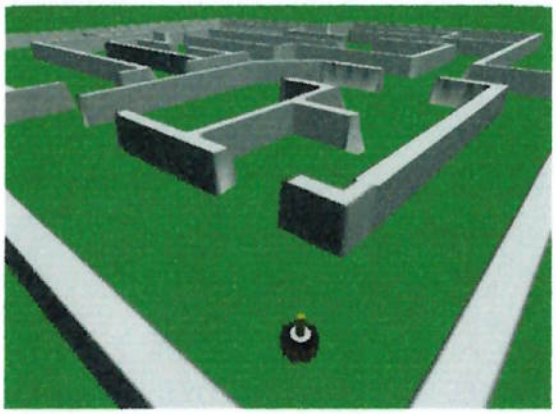


Figura 6 - Ambiente virtual gerado no Gazebo.

A utilização de um modelo virtual permitiu uma maior velocidade na realização de testes em etapas iniciais do desenvolvimento do trabalho, pois a manipulação de ambientes e de robôs se torna mais fácil. A liberdade de construção de cenários para testes é maior, e consegue-se também uma segurança maior enquanto não for alcançada uma certa qualidade nos resultados finais do algoritmo. A interface com o modelo virtual é realizada através do software *Player*, que consegue se comunicar igualmente com robôs reais e virtuais, garantindo pronta transição entre testes reais e virtuais sem a necessidade de alteração do algoritmo.

Assim sendo, foi construído um modelo do robô *MagellanPro*, pertencente ao Laboratório de Percepção Avançada (LPA), do Departamento de Engenharia Mecatrônica e Sistemas Mecânicos da Escola Politécnica da Universidade de São

Por último, foram estudados os erros inerentes a esses sensores, para que o modelo virtual apresente o mesmo desempenho quando colocado diante de objetos. Para isso foram realizados testes e coleta de dados em diversas condições, que permitiu o levantamento das curvas características dos erros, que será suposta como sendo uma distribuição normal. É a partir de tais erros que surge a necessidade de um algoritmo de SLAM, e por isso o modelo virtual também deve apresentá-los, para que possa se beneficiar dos resultados do algoritmo que será implementado. Modelos de tais erros também permitirão um tratamento mais eficiente dos próprios resultados reais, possibilitando um tratamento probabilístico mais eficiente.

3.1. Funções de Navegação

Após a construção do modelo virtual do *MagellanPro* no software *Gazebo*, o próximo passo foi a construção de uma biblioteca de rotinas de navegação, para que o modelo pudesse ser controlado de forma a se movimentar pelo ambiente virtual. Essas rotinas permitem o envio de comandos simples para o robô, fazendo-o movimentar-se até alcançar um objetivo, colocado sob a forma de um par de coordenadas *(X,Y)*. Para alcançar esse objetivo final o robô conta com funções de rotação e translação, que fornecem os dois graus de liberdade necessários.

Visando permitir uma maior liberdade na escolha dos movimentos que serão utilizados pelo robô, ambas as funções foram construídas de maneira independente. Dessa forma, a rotação do robô não interfere em sua translação e vice-versa, embora possam ser utilizadas simultaneamente, permitindo uma navegação eficiente em ambas as dimensões. A simetria radial do *MagellanPro* também foi utilizada para permitir movimento "em marcha ré", quando o robô inverte a sua orientação frontal, visando facilitar o acesso a locais, evitando rotações desnecessárias.

Pode-se fazer o robô alcançar uma coordenada inicialmente rotacionando-o até a orientação desejada e depois transladando-o até o local correto, ou então realizando as duas funções e navegando diretamente até o ponto. A velocidade é uma função da distância entre a posição (ou orientação) do robô em relação àquela que ele deve alcançar para cumprir o seu objetivo. No início do processo é escolhida tanto

Paulo. Sua geometria e propriedades principais foram estudadas visando a maior fidelidade possível entre os resultados obtidos virtualmente e aqueles que seriam obtidos em testes reais. A movimentação do *MagellanPro* é composta por duas rodas conectadas a motores independentes, permitindo sua rotação dentro do seu próprio eixo e translação auxiliada por uma roda traseira.

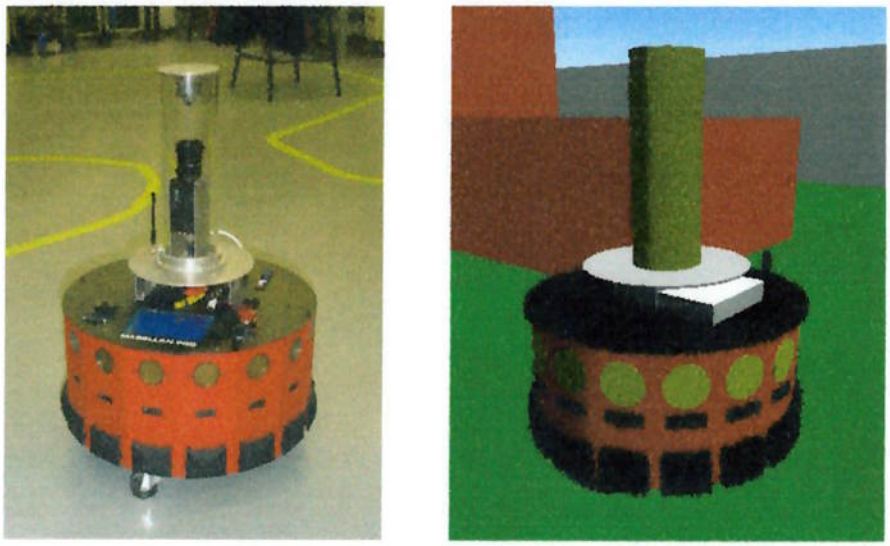


Figura 7 - MagellanPro e seu modelo construído no Gazebo.

Seus sensores também foram modelados, com ênfase nos 16 sensores de sonar que ele possui, distribuídos igualmente em seu perímetro circular, e que serão utilizados nesse trabalho para a coleta de informações do ambiente. O sistema de odometria, já existente no *MagellanPro* e que determina a posição do robô a partir da rotação de suas rodas, também foi implementado no modelo virtual, completando o conjunto de sensores que ele utilizará para o mapeamento e localização simultâneos.

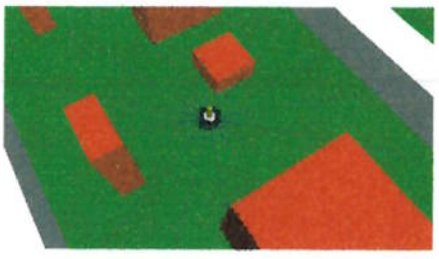


Figura 8 - Modelo de sensores do MagellanPro.

velocidade como aceleração máximas, tanto translacional como rotacional, para evitar deslocamentos bruscos ou velocidades finais muito elevadas.

Conforme o robô se aproxima da posição ou orientação desejadas, sua velocidade diminui até finalmente se anular, mostrando que ele alcançou o seu objetivo final. A sensibilidade da movimentação em relação à diferença entre a posição ou orientação atual e desejada pode ser determinada na própria função, refletindo a maneira como o robô aumenta ou diminui sua velocidade. A função escolhida para realizar esse vínculo entre velocidade e diferença de posição ou orientação é mostrada a seguir (equação 1), com *S* sendo a sensibilidade do sistema.

$$V_t = O_t V_{t,\max}\left(1-\frac{1}{\frac{|\Delta_t|}{V_{t,\max}}+1}\right)^S \qquad V_r = O_r V_{r,\max}\left(1-\frac{1}{\frac{|\Delta_r|}{V_{r,\max}}+1}\right)^S \tag{1}$$

Através dessa função consegue-se a velocidade angular e linear do *MagellanPro* a cada instante, de acordo com a sua distância em relação ao ponto de chegada e à diferença entre a sua orientação e aquela que deveria assumir para estar alinhado com o objetivo. Dessa velocidade é então subtraída a velocidade anterior, de maneira a se encontrar a variação de velocidades (aceleração) à qual o robô será submetido. Caso essa aceleração seja maior do que aquela definida como máximo, a velocidade atual será modificada por um valor igual à aceleração máxima. Caso ela seja menor, a velocidade atual será modificada por essa diferença, fazendo o sistema alcançar a velocidade que foi realmente calculada.

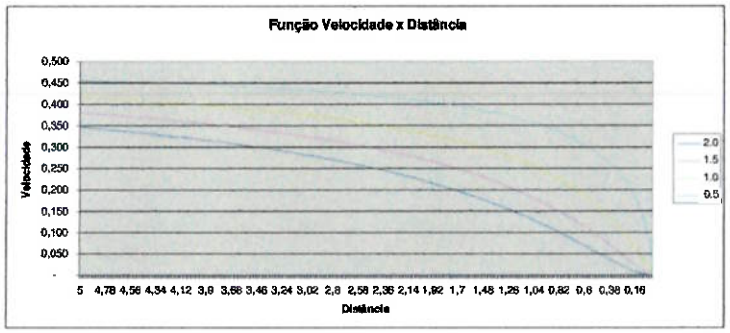


Figura 9 - Função Velocidade para diferentes Sensibilidades

Acima está representado um gráfico com essa mesma função plotada para diferentes valores de Sensibilidade. Pode-se perceber a maneira como a sensibilidade influi na movimentação do robô, pois quando assume valores menores ele tende a permanecer mais tempo em sua velocidade máxima antes de diminuí-la. Esse comportamento pode fazer com que sua velocidade não se anule quando ele alcançar o objetivo final, pois sua inércia fará com que o sistema continue se movimentando. Já valores muito altos para a sensiblidade farão com que o robô se movimente desnecessariamente devagar, o que também não é desejado. Dessa forma, uma calibração deve ser realizada, com testes que permitam a determinação de um valor que permita a otimização da função em cada momento. Para o MagellanPro foram utilizados os seguintes valores:

Tarefa	St	Sr
Rotação	----	1.25
Translação	1.00	0.75
Ambas	1.25	1.00

Tabela 1 - Valores empíricos de Sensbiilidade utilizados.

Outro fator que deve ser levado em consideração é a distância do robô em relação a obstáculos próximos, pois deve-se tentar impedir que ele colida com qualquer objeto que possa estar em seu caminho. Esse comportamento se torna especialmente importante quando são utilizadas simultaneamente tanto a rotina de translação como de rotação, pois nesse caso não há como saber exatamente qual será o caminho percorrido pelo robô. Nesse caso, decidiu-se por alterar as velocidades de rotação e translação de forma a evitar que o robô colida com o obstáculo, de maneira puramente reativa (baseada apenas nos seus sensores imediatos).

A proximidade de obstáculos deve fazer com que a velocidade linear diminua, permitindo que o robô possa rotacionar antes de atingir o obstáculo (supõe-se que o seu objetivo final não esteja localizado em uma área ocupada). Assim sendo, são utilizadas as informações dos seus sensores localizados na sua posição frontal (ou traseira, quando ele estiver se movimentando para trás), e no caso aqui foi utilizado tanto o sensor localizado diretamente na sua frente como aqueles imediatamente

laterais, de maneira a prover mais informações. Inicialmente é realizada uma normalização dos valores de distância (equação 2), e para isso são necessários dois parâmetros:

- *Proximidade (P):* A proximidade máxima de um obstáculo que o robô pode permanecer. Qualquer valor de distância menor do que esse significa que o robô deve parar sua velocidade de translação, continuando apenas com a de rotação.
- *Influência (I):* A partir de qual distância o robô deve começar a se preocupar com a aproximação de um obstáculo.

$$D_i = \frac{\left(\min\left(V_i, I\right) - \mathrm{P}\right)}{I - P} \tag{2}$$

Valores de D menores do que zero são tratados como se fossem zero, o que significa que o sistema não chegará a retroceder para evitar um obstáculo, apenas parar de se movimentar linearmente. Pode-se também perceber que dessa forma os valores sempre estarão entre 0 e 1, mostrando que a velocidade apenas diminuirá devido à presença de obstáculos, nunca aumentará quando o robô estiver distante de qualquer objeto. A próxima etapa então consiste em se fazer a média das distâncias relevantes ao problema (equação 3), no caso as três distâncias frontais ou traseiras, de acordo com a orientação de navegação.

$$F = \left(\frac{\left(16D_i + 8\left(D_{i-1} + D_{i+1}\right)\right)}{32} \right)^{1.5} \tag{3}$$

O valor 1.5 foi calculado empiricamente através de testes, agindo de maneira similar à Sensibilidade utilizada na determinação das velocidades, como já foi mostrado. O valor do sensor localizado diretamente à frente da navegação do robô possui um peso maior, e aqueles imediatamente próximos contribuem cada um com um quarto do valor total, fazendo com que o resultado final continue entre 0 e 1. Esse valor é então multiplicado á velocidade já calculada antes, fazendo-a passar a refletir não somente a distância entre o robô e o objetivo final, mas também a distância entre o robô e os obstáculos próximos.

Já a velocidade angular deve diminuir quando o robô está rotacionando na direção de um obstáculo e aumentar quando ele está rotacionando no sentido de se afastar desse obstáculo. Essa distinção pode ser feita comparando-se o valor da sua normalização com a direção em que sua rotação está acontecendo no momento. Essa normalização é realizada pela comparação dos valores dos sensores de um lado do robô com os sensores do outro lado, para verificar em qual direção existem obstáculos mais próximos. Dividindo-se o robô em seu eixo de translação, são subtraídos os valores de cada sensor com o seu valor simétrico do outro lado.

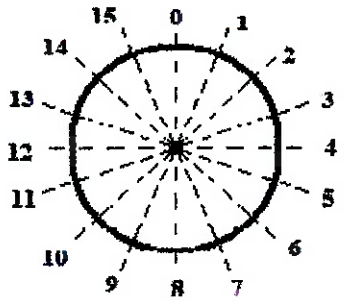


Figura 10 - Diagrama dos sensores de sonar do MagellanPro

$$S_i = \frac{(D_{1+i} + D_{15-i})}{I} \quad i = [0,6] \tag{4}$$

O resultado então são sete valores, com índice entre [0,6], que representam a diferença entre as distâncias *D* de índice *1+i* e *15-i*. É realizada então uma média entre tais distâncias, atribuindo um peso maior àquelas distâncias que estão mais próximas da região do robô que está se movimentando frontalmente (regiões 0 ou 8, quando, respectivamente, o robô está se movimentando para frente ou para trás).

$$\begin{aligned} Frontal \rightarrow F &= \frac{(8S_0 + 4S_1 + 2S_2 + S_3)}{15} \\ Marcha\ Ré \rightarrow F &= \frac{(8S_0 + 4S_1 + 2S_2 + S_3)}{15} \end{aligned} \tag{5}$$

Dessa maneira, valores elevados de proximidade angular indicam uma predominância de obstáculos na região direita do robô, mostrando que ele deve

procurar rotacionar para a esquerda de maneira a evitar esses obstáculos. Já valores negativos indicam o contrário, uma predominância maior de obstáculos na região esquerda, e por isso uma tendência do robô de rotacionar para a direita. Valores pequenos de proximidade angular não indicam a ausência de obstáculos, mas sim uma distribuição semelhante em ambos os lados, e por isso nenhum benefício é conseguido pela rotação para qualquer um dos lados.

Caso o robô realmente esteja rotacionando na direção em que ela é desejada de acordo com a proximidade angular, o sistema amplia essa rotação por um fator de $1 + |P|$, de maneira a fazê-lo completar a rotação mais rapidamente. Caso ele esteja rotacionando na direção contrária, a rotação é retardada por um fator de $1 / (1 + |P|)$, para que o sistema possua tempo de transladar para fora da região de obstáculos. É importante perceber que essa solução não altera o sentido de rotação do robô, da mesma maneira que a translação também não é alterada. Isso acontece porque supõe-se que o sistema possui um caminho livre entre ele e o objetivo final, e por isso jamais tomará uma trajetória que o leve de encontro proposital a um obstáculo.

3.2. Mapeamento

A informação que é recolhida pelos sensores do robô deve ser armazenada de alguma maneira, incorporada ao seu modelo do ambiente para que possa depois se recuperada e utilizada juntamente com as informações colhidas posteriormente pelo sistema. Dessa forma, o conhecimento do robô em relação ao meio que o cerca se acumula com o passar do tempo, permitindo que ele tome decisões mais precisas sobre como realizar as tarefas que lhe foram propostas. Esse conhecimento também deve estar disposto de uma maneira que permita uma recuperação eficiente, permitindo uma utilização imediata de acordo com a sua utilização dentro do sistema.

Sendo a finalidade do SLAM o mapeamento dinâmico de ambientes, a maneira mais natural de se armazenar esse conhecimento adquirido pelos sensores é através da construção de mapas. Existem basicamente duas maneiras de se realizar o mapeamento de um ambiente, através dos chamados mapas topológicos ou métricos. Os primeiros fazem uso de informações qualitativas do ambiente, focalizando

principalmente em suas regiões vistas como mais importantes e mantendo-as conectadas através de regiões sabidamente livres, que permitem a navegação entre essas regiões.

Já os mapas métricos armazenam informação quantitativa, através de uma matriz discretizada do ambiente, com cada célula representando uma região que pode estar ou não ocupada. Para aplicações em navegação terrestre, essa matriz pode ser bidimensional, pois não haverá alteração da cota do robô em relação à sua posição inicial. Essa solução permite também uma localização constante do robô a cada instante, e a informação dos sensores pode ser imediatamente recuperada para auxiliar nessa localização, como é o propósito do SLAM. Por tais razões, esse trabalho fará uso de um mapa métrico para o armazenamento das informações obtidas pelos sensores do *MagellanPro*, incorporando-as ao seu modelo conforme o robô se movimenta. Esse mapa será construído dinamicamente, supondo-se que o sistema não possui nenhum conhecimento *a priori* do ambiente, apenas aquele adquirido durante sua navegação.

Uma maneira de se lidar com as incertezas inerentes aos sensores utilizados na coleta de informação e localização do robô é a utilização de conceitos de probabilidade na determinação das regiões do mapa que estão ocupadas. Abordagens determinísticas, ou binárias, preencheriam um mapa métrico com apenas três valores distintos:

- *Desconhecido (0.5):* Ainda não foi obtida nenhuma informação relativa à célula em questão pelos sensores do robô.
- *Ocupado (1):* Existe na região representada pela célula em questão alguma estrutura que é percebida pelo sensor como um obstáculo à navegação do robô.
- *Livre (0):* A região representada pela célula em questão não apresenta nenhum obstáculo, podendo ser utilizada como parte de uma trajetória de navegação pelo robô.

Na construção inicial do mapa métrico, todas as células são preenchidas com o valor *Desconhecido*, representando a falta de conhecimento do sistema em relação ao ambiente ao seu redor. Conforme ele se movimenta, a informação colhida é

incorporada ao mapa, eliminando essa incerteza e mostrando quais áreas podem ser utilizadas em trajetórias de navegação e quais devem ser evitadas. Caso o ambiente seja estático, informações cumulativas referentes a uma mesma célula serão semelhantes, e no caso de ambientes dinâmicos, essas informações atualizarão o mapa para que ele reflita o estado mais recente de acordo com a capacidade do robô de observar o ambiente.

Mas, os erros inerentes aos sensores fazem com que informações relativas a uma mesma célula tomadas em momentos diferentes não sejam idênticas, mesmo quando não houve nenhuma alteração no ambiente, apenas na posição relativa do robô. Isso porque os sensores utilizados reagem diferentemente em situações diferentes, e por isso podem não concordar no estado de uma célula, criando erros de mapeamento. Um mapa binário não é capaz de lidar com esses erros, incorporando a informação ao mapa de maneira indistinta, tratando cada célula como *ocupada* ou *livre* a despeito do seu estado anterior.

Os mapas probabilísticos procuram lidar com esses erros atribuindo a cada célula não apenas valores binários, mas sim qualquer valor situado entre 0 (que representa uma célula *livre*) e 1 (que representa uma célula *ocupada*). Esse valor indica então a probabilidade de que essa célula realmente possua um obstáculo que impeça a navegação do robô. Com isso, o mapa é construído a partir do acúmulo de valores em cada uma das células conforme os sensores colhem informações relativas a elas, e erros aleatórios têm a sua influência diminuída diante do conjunto de amostras.

No início do processo, todo o mapa recebe o valor 0.5, que representa o estado de desconhecimento em relação ao seu estado verdadeiro, que será encontrado conforme a informação é incorporada. Esse valor pode ser alterado de acordo com previsões *a priori* do ambiente que porventura possam vir a serem feitas, indicando regiões com maior ou menor probabilidade de existência de obstáculos. Após isso, utiliza-se o modelo de erros do sensor, obtido experimentalmente, para verificar qual é a distribuição de probabilidade de posicionamento de um obstáculo encontrado.

Qualquer célula atravessada sem ativar a detecção de obstáculos pelos sensores é tomada como estando livre, e a região ao redor da posição em que foi indicado um obstáculo possui sua probabilidade aumentada. Para essa atualização dos

valores de cada célula foi utilizado o *Teorema de Bayes,* que permite a obtenção da probabilidade *a posteriori* de um determinado evento a partir de sua probabilidade *a priori* e de uma distribuição dentro da qual o evento causador se encaixa. O *Teorema de Bayes* pode ser colocado da seguinte forma:

$$P(H \mid E) = P(E \mid H)\frac{P(H)}{P(E)} \tag{6}$$

Sendo *H* o evento "obstáculo existir na célula em questão" e:

- *H:* O evento "obstáculo estar localizado na célula em questão".
- *E:* O evento "sensor indicar que o obstáculo se encontra na célula em questão".
- P(H|E): A probabilidade *a posteriori*, ou seja, o valor que a célula assumirá após a observação ser incorporada.
- P(E|H): A probabilidade *a priori*, ou seja, o valor inicial apresentado pela célula, obtido em iterações anteriores.
- P(H): A distribuição de probabilidade para o sensor, no caso o modelo de erros construído.
- P(E): Fator de normalização, que faz com que a probabilidade *a posteriori* permaneça dentro do intervalor [0,1].

É importante perceber que não será apenas a célula correspondente à posição indicada pelo sensor para o obstáculo que será atualizada, mas sim todas aquelas ao seu redor, de acordo com o modelo de erro desenvolvido. A região indicada pelo sensor possuirá a probabilidade máxima de ser aquela que realmente está obstruída, e conforme as células atualizadas se afastam dessa posição, esse valor diminui, até um patamar onde não há mais a necessidade de atualização, pois a probabilidade assume valores muito baixos. Ao mesmo tempo, as regiões anteriores à posição indicada pelos sensores devem ter suas probabilidades de estarem ocupadas diminuídas, pois o sensor conseguiu atravessá-las, indicando que estão livres.

Já as regiões que se localizam após a zona de localização do obstáculo não devem ser atualizadas, pois o sistema não adquiriu nenhuma informação relacionada a essa área. Dessa forma, ele se estabiliza no valor de 0.5, que significa

desconhecimento em relação ao novo valor, e que deve fazer com que a probabilidade *a posteriori* seja igual à probabilidade *a priori*.

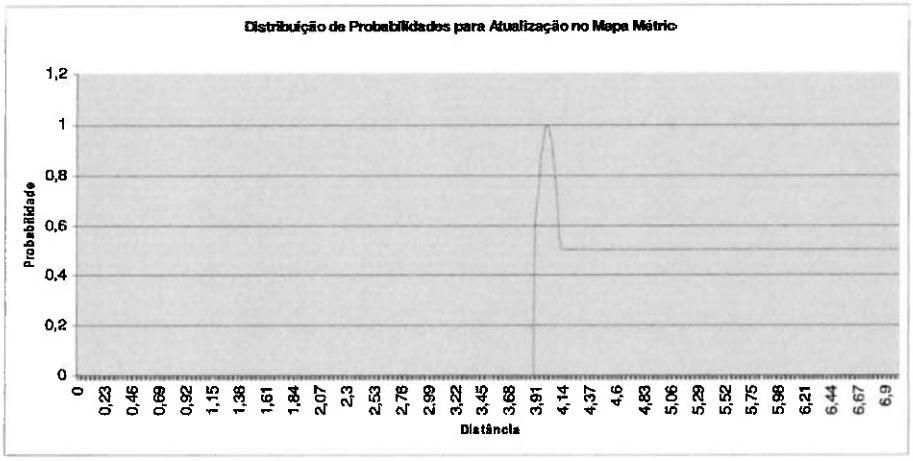


Figura 11 - Função de atualização do mapa probabilístico. ($\mu = 4; \sigma = 0,1$)

Pode-se perceber então que a função de atualização é uma união entre duas funções distintas, sobressaindo-se sempre a maior dentre as duas em cada intervalo. A primeira função é uma gaussiana cujo desvio padrão é calculado de acordo com o modelo de erros construído para o sensor em questão e a média é a distância indicada pela informação recebida do sensor. Já a segunda função é descontínua na posição onde a média da função gaussiana se encontra, assumindo valor constante igual a 0 antes desse ponto e 0.5 depois dele.

A função de distribuição de probabilidades também pode assumir forma bidimensional, quando leva-se em consideração também o erro angular no sensor, em adição ao seu erro radial. Dessa forma, são utilizadas duas gaussianas, cada uma construída a partir do desvio padrão referente ao erro em questão, e ambas com média no mesmo ponto, que é o valor fornecido pelo sensor. Essa possiblidade foi discutida em mais detalhes no capítulo sobre os erros de sonares do *MagellanPro*, e não foi utilizada no desenvolvimento desse projeto, que levou em consideração apenas o erro radial dos sensores de sonar.

Durante as primeiras etapas do desenvolvimento do algoritmo de SLAM aqui proposto, procurou-se utilizar mapas probabilísticos para registrar as informações obtidas pelos sensores sobre o ambiente. No entanto, a utilização de mapas dessa

natureza mostrou-se excessivamente pesada computacionalmente para ser utilizada dentro da aplicação da solução escolhida para se resolver o problema do SLAM. A razão para isso está na quantidade de células do mapa que deverão ser atualizadas a cada iteração do algoritmo, impedindo a utilização de uma quantidade suficiente de mapas para permitir uma solução satisfatória, como será mostrado posteriormente na abordagem escolhida, o *DP-SLAM*.

Uma abordagem probabilística requisitaria, a cada iteração dos sensores, uma atualização simultânea de todas as células varridas pelos sensores, não apenas aquelas que são indicadas como possuidoras de obstáculos, mas também todas aquelas anteriores. Como já foi mostrado anteriormente, as células anteriores teriam sua probabilidade de estarem ocupadas diminuída, e aquelas próximas da região do obstáculos teriam essa probabilidade aumentada.

No caso do *MagellanPro*, que possui 16 sensores de ultra-som com um alcance de 4 metros, em um mapa discretizado em regiões de 5 centímetros, isso significaria uma atualização de 16.4/0,05 células, ou seja, 1280 células. Adicionalmente, deverão ser também recuperados os valores anteriores de cada uma de tais células, e realizados os cálculos relacionados às suas respectivas atualizações. Como a solução escolhida, o *DP-SLAM*, se utiliza de diversos mapas simultâneos para realizar a localização e o mapeamento, essa solução rapidamente se torna computacionalmente proibitiva.

Assim sendo, optou-se por utilizar um mapa binário, que é inicialmente composto por células marcadas como *livres*, e que são atualizadas como *ocupadas* quando um sensor indica a presença de um obstáculo em uma determinada posição. Essa solução pôde ser utilizada sem uma perda significativa de precisão final devido à repetibilidade final calculada para os sensores de sonar utilizados. O desvio padrão do modelo de erros dos sonares do *MagellanPro* alcançou um valor de *0,053*, indicando que a maioria dos erros inerentes ao sensores se localiza dentro do intervalo de discretização de uma única célula.

Assim sendo, a distribuição de erros do sonar não é relevante em uma resolução como aquela procurada para a aplicação proposta nesse trabalho, e pode-se supor que a informação obtida pelo sonar representa corretamente o ambiente. As células

anteriores àquela em que o obstáculo se encontra não precisam ser atualizadas, pois já se encontram com valores de *livre*, e podem continuar dessa forma sem a necessidade da incorporação de nova informação. Adicionalmente, regiões em que o alcance máximo do sonar é alcançado sem a localização de um obstáculo também não precisam ser atualizadas, já que nenhuma célula é marcada como *ocupada.*

Consegue-se, com isso, uma atualização máxima de 16 células a cada iteração, um valor substancialmente menor do que aquele necessário para um mapa probabilístico, de 1280 células. A maior desvantagem desse método escolhido está na sua incapacidade de lidar com ambientes dinâmicos, já que regiões marcadas como *ocupadas* não são alteradas para *livres* caso os sensores deixem de reconhecer obstáculos naquela região. Deve-se, portanto, restringir a utilização da solução aqui proposta para ambientes estáticos, onde os objetos ao redor do robô não se alteram com o passar do tempo.

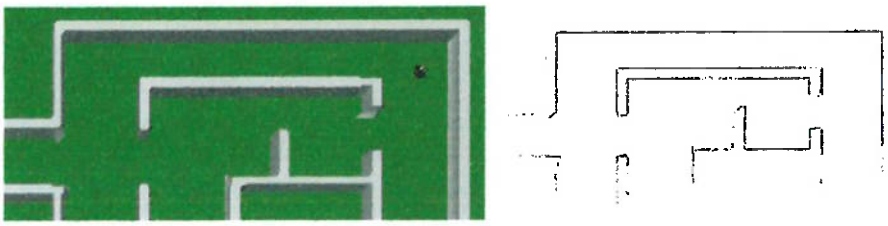


Figura 12 - Ambiente virtual e seu mapa binário correspondente.

3.3. Odometria

3.3.1. Modelo de Odometria

O sistema de odometria implementado no *MagellanPro,* tanto no robô propriamente dito como em seu modelo virtual, faz uso da sua geometria para se definir qual foi o seu deslocamento entre cada tomada de dados, o que acontece a uma freqüência de 10 Hz. Como parâmetros de construção é necessário conhecer o raio R das suas rodas que estão conectadas aos motores independentes, e a separação $2L$ entre elas, passando pelo eixo de rotação do robô. Como variáveis de entrada são

fornecidas as velocidades angulares (ω_r e ω_l) de cada um dos motores, o que permite saber a velocidade angular de cada uma suas rodas.

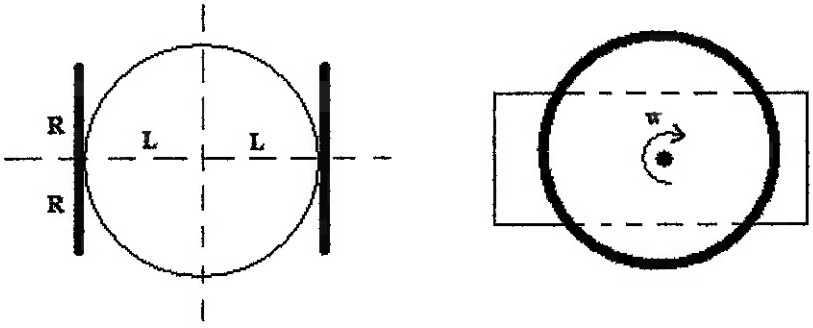


Figura 13 - Esquema de rodas do MagellanPro.

Conhecendo-se o raio das rodas e a velocidade angular de cada uma das rodas, pode-se encontrar (equação 7) a velocidade do ponto em que cada uma das rodas encontra o solo, que representa a velocidade com que cada uma das rodas do robô está se movimentando em relação à superfície de navegação.

$$\begin{aligned} V_r &= R\omega_r \\ V_l &= R\omega_l \end{aligned} \tag{7}$$

No *MagellanPro,* a translação pura é realizada quando ambas as rodas possuem a mesma velocidade angular, e com isso seu deslocamento é o mesmo em todo o seu conjunto. Já a sua rotação pura acontece quando cada uma de suas rodas possuem velocidades angulares com o mesmo módulo, embora estejam rodando no sentido invertido, fazendo com que o robô gire em seu próprio eixo, localizado no ponto médio do segmento que une os dois eixos das rodas. Outras situações farão com que a rotação e a translação ocorram simultaneamente, criando todas as maneiras possíveis para a navegação do *MagellanPro.* Os valores para velocidade de rotação e translação podem ser obtidos da seguinte forma:

$$V_{trans} = \frac{(\omega_r + \omega_l)}{2} \quad V_{rot} = \frac{(\omega_r - \omega_l)}{2L} \tag{8}$$

Pode-se observar que as condições limite de translação e rotação puras descritas acima podem ser alcançadas a partir das fórmulas, fazendo-se com que

apenas uma das velocidades se anule. Para se alcançar então o deslocamento do robô durante o período basta integrar a velocidade dentro do intervalo de tempo entre medições. Como as únicas informações que podem ser obtidas nesse intervalo são os valores de velocidade iniciais e finais, supõe-se uma transição linear entre esses dois valores, fazendo com que a integração se torne do tipo (com F sendo a freqüência de obtenção de dados do sistema):

$$\Delta_{trans} = \frac{\left(V_{trans,i} + V_{trans,f}\right)}{2F} \quad \Delta_{rot} = \frac{\left(V_{rot,i} + V_{rot,f}\right)}{2F} \tag{9}$$

Esses são os valores que devem ser somados aos valores atuais de odometria, atualizando-a para levar em consideração o período de tempo transcorrido desde a última atualização do sistema. Como o seu deslocamento de translação pode acontecer em duas dimensões, nos eixos *x* e *y*, deve-se antes de mais nada calcular as projeções desse deslocamento em cada um desses eixos, para só então somar a contribuição de cada um no deslocamento. O resultado final é, então:

$$\begin{aligned} \alpha_{t+1} &= \alpha_t + \Delta_{rot} \\ X_{t+1} &= X_t + \Delta_{trans} \cos\left(\alpha_{t+1}\right) \\ Y_{t+1} &= Y_t + \Delta_{trans} \sin\left(\alpha_{t+1}\right) \end{aligned} \tag{10}$$

Durante a etapa de atualização de velocidades do sistema, o raciocínio inverso deve ser feito, para que a nova velocidade decorrente dos cálculos realizados pelos algoritmos de navegação possa ser alcançada por ele. Nesse caso, as variáveis de entrada são as velocidades translacionais e rotacionais, e devem ser encontrados os valores de velocidade angular de cada um dos motores. De posse dessas velocidades, a taxa de rotação de cada um dos motores (ω_r e ω_l) pode ser encontrado da seguinte forma:

$$\omega_r = \frac{\left(V_t + V_r L\right)}{R} \quad \omega_l = \frac{\left(V_t - V_r L\right)}{R} \tag{11}$$

3.3.2. Erros de Odometria

O sistema de odometria utilizado pelo *MagellanPro* faz uso da rotação de suas rodas para determinar qual é a sua posição atual, relativa à posição em que ele se encontrava no início da movimentação. Sabendo-se o raio *R* de cada roda e o número de vezes em que cada uma girou, é fácil determinar quantos metros foram percorridos pelo sistema e também qual é a sua orientação.

Mas os resultados oferecidos pela odometria tendem a se tornar imprecisos quando utilizados ininterruptamente por longos períodos de tempo, devido ao acúmulo de erros gerados por processos como deslizamento de rodas e condições irregulares de terreno. Tais erros não existem em um ambiente virtual, e por isso devem ser adicionados para que o modelo também apresente essas imprecisões, tornando-o mais verossímil em relação à realidade.

A posição do *MagellanPro* possui três graus de liberdade, representados em um determinado instante *k* pela sua localização em termos de $X_k = [x_k, y_k, \theta_k]^T$, indicando sua posição e orientação. Sabendo-se qual é a sua posição em um instante imediatamente anterior, $X_{k-1} = [x_{k-1}, y_{k-1}, \theta_{k-1}]^T$, pode-se dividir sua movimentação em duas etapas distintas. A primeira é uma rotação que procura orientar o sistema na direção da posição procurada, e a segunda é uma translação, que o leva em linha reta até o destino, completando a movimentação.

$$\begin{aligned} &\Delta x = x_k - x_{k-1} \qquad \Delta y = y_k - y_{k-1} \qquad \Delta\alpha = \theta_k - \theta_{k-1} \\ &\rho = \sqrt{\Delta x^2 + \Delta y^2} \qquad \theta_k = \arctan(\Delta y / \Delta x) \end{aligned}$$

$$\begin{bmatrix} x_k \\ y_k \\ \theta_k \end{bmatrix} = \begin{bmatrix} x_{k-1} + \rho\cos(\theta_k) \\ y_{k-1} + \rho\sin(\theta_k) \\ \theta_{k-1} + \Delta\alpha \end{bmatrix} \tag{12}$$

Mas esses valores (equação 12) são precisos, não levando em consideração os erros do sistema de odometria, e essa compensação pode ser feita através da adição de um valor aleatório adquirido a partir de uma distribuição gaussiana com média μ_R e

desvio padrão σ_R. Esse desvio padrão é proporcional à própria rotação, aumentando quando o sistema é rotacionado em valores angulares maiores, e com isso chega-se ao seguinte resultado:

$$\theta_{k+1} = \theta_k + \Delta\alpha + N(\mu_R, \sigma_R \Delta\alpha) \tag{13}$$

O próximo passo é então a translação, que é mais complexa por depender parcialmente tanto dos próprios valores de translação como também de valores de rotação, pois erros de orientação adquiridos após o alinhamento inicial também influenciam no resultado da translação. Esse erro de alinhamento pode ser tratado através da divisão da movimentação em K etapas menores, cada uma possuindo seus próprios valores de erros, tomados a partir de distribuições normais com valores μ_{RT} e $\sigma_{RT}\sqrt{K/2}$ para erros de alinhamento e μ_T e $\sigma_T\sqrt{K}$ para a translação propriamente dita. Ambos os desvios são também proporcionais à translação, chegando-se aos valores de:

$$\begin{aligned} x_{k+1} &= x_k + N(\mu_T, \rho\sigma_T)\cos(\theta_{k+1}) \\ y_{k+1} &= y_k + N(\mu_T, \rho\sigma_T)\sin(\theta_{k+1}) \\ \theta_{k+1} &= x_k + N(\mu_{TR}, \rho\sigma_{TR}) \end{aligned} \tag{14}$$

Os diversos valores de μ e σ devem ser obtidos a partir de testes realizados com o próprio robô que será utilizado, de maneira a refletirem a maneira como a sua odometria se comporta nas situações às quais será submetida em situações reais. Erros sistemáticos do sistema serão modelados pelo desvio na média do sistema, enquanto os erros aleatórios são modelados pelo desvio padrão, sendo maiores conforme esse valor aumenta.

3.4. Sonar

3.4.1. Modelo de Sonar

A informação obtida pelos sonares instalados no *MagellanPro* podem ser prontamente utilizadas na obtenção das distâncias encontradas, pois seus valores representam a distância calculada pelo sonar em que um obstáculo se encontra, relativamente à posição em que o sensor está posicionado. Por esse motivo a localização do sistema é fundamental para que um mapeamento seja alcançado, já que toda a informação incorporada ao mapa estará vinculada à posição atual do robô.

Inicialmente são definidos $X_k = [x_k, y_k, \theta_k]^T$ como sendo a posição do robô em um determinado instante *k*, e β a orientação do sensor de sonar em questão, relativamente à orientação definida como 0º pelo algoritmo (a "frente" do robô). A informação fornecida pelo sonar é então um valor de distância, que pode variar entre 0 e 4 metros, sendo esse o alcance máximo em que o sonar pode ser utilizado. Essa distância *d* deve ser então decomposta nas coordenadas *x* e *y*, para que a coordenada correta possa ser atualizada. Essa conversão pode ser feita da seguinte forma (com R_r sendo o raio da circunferência do *MagellanPro*):

$$\begin{aligned} x_{update} &= x_k + (d + R_r)\cos(\theta_k + \beta) \\ y_{update} &= y_k + (d + R_r)\sin(\theta_k + \beta) \end{aligned} \tag{15}$$

Essa coordenada encontrada define um segmento de reta que deve ser alterado no mapa referente à partícula em questão, que tem início na posição ocupada pelo sensor de sonar (quando *d* assume valor nulo, restando na equação apenas R_r) e termina quando *d* assume o valor mostrado pelo sonar. Para cobrir todas as células da matriz referentes ao segmento, o valor de *d* deve variar de acordo com a discretização do mapa, tomando-se cuidado para que cada célula não seja atualizada mais de uma vez, principalmente caso esteja sendo utilizado um mapa probabilístico.

Todas as células anteriores ao valor de *d* igual ao valor fornecido pelo sensor de sonar terão a sua probabilidade de estarem realmente ocupadas diminuída, pois o ultra-

som conseguiu atravessá-las até alcançar a superfície que fê-lo retornar. Caso não esteja sendo utilizado um mapa probabilístico, as células podem ser marcadas como *livres*, caso o mapa inicial tenha sido preenchido com valores de *desconhecido*, ou então simplesmente não marcadas, caso toda a matriz tenha sido preenchida inicialmente já com valores de *livres*.

Já a última célula, aquela preenchida com o valor de *d* fornecido pelo sensor de sonar, pode ser preenchida de duas maneiras, de acordo com o seu valor. Caso o valor retornado seja igual ao alcance máximo do sonar, supõe-se que o ultra-som não bateu em nenhum obstáculo, mas sim se dispersou pelo ambiente sem ser recebido de volta pelo sensor. Nesse caso, aquela célula também estará ocupada, recebendo o mesmo tratamento das células anteriores. Caso o valor seja menor do que o alcance máximo, supõe-se que o sensor encontrou algum obstáculo, e aquela célula tem a sua probabilidade de estar ocupada aumentada, ou então recebe o valor de *ocupada*, representando que naquela região exste um obstáculo.

3.4.2. Erros de Sonar

O *MagellanPro* fará uso de sonares para coletar informações, possuindo em seu perímetro 16 sensores igualmente distribuídos, formando um ângulo de 22,5º entre cada um, permitindo que ele receba informações de todo o ambiente ao seu redor. Cada sensor de sonar retornará, em cada iteração, um único valor de distância, indicando a distância do robô em relação a objetos naquela direção, e permitindo que o mapa seja atualizado com esses dados.

Um sensor de sonar funciona a partir da emissão de ultra-som, que se propaga no ar até bater em alguma estrutura à sua frente, e é então refletido e recebido de volta pelo sensor, que determina então a distância da estrutura através do tempo transcorrido entre emissão e recepção. Embora seja considerado um sensor pontual, a onda de ultra-som viaja em um cone cujo raio cresce conforme se distancia da origem da fonte, criando uma área onde ele pode ser refletido e, com isso, retornar ou não até o sensor para cálculo de distância. Caso esse retorno não aconteça é caracterizado um "falso

negativo", com o sensor retornando a inexistência de obstáculo quando na realidade existe algo naquela direção.

Quando o sinal de ultra-som retorna ao sensor é fornecido então um valor numérico para a distância do robô até o objeto naquela direção. Idealmente, esse número aliado com a orientação do sensor, já conhecida, permitiria a determinação precisa das coordenadas onde o obstáculo se encontra. Mas, na realidade, esse valor está distribuído dentro de uma área, podendo variar tanto no sentido radial, indicando que o objeto se encontra mais próximo ou distante do que está indicado, ou ainda no sentido angular, refletindo a distribuição cônica do sensor, e com isso o objeto estará localizado em uma orientação diferente daquela possuída pelo sensor.

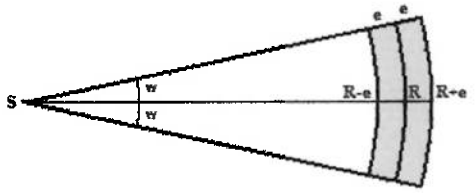


Figura 14 - Área de distribuição dos valores do sonar.

Regiões localizadas antes da posição indicada pelo sonar estão mais propensas a estarem vazias, permitindo que o ultra-som viajasse até encontrar o obstáculo na região indicada. Já regiões localizadas após a posição indicada não recebem nenhum tipo de conhecimento, pois não há como o sensor definir o que se encontra depois do obstáculo encontrado. Assim sendo, pode-se definir através da informação dos sonares três regiões distintas:

- A região anterior à posição do obstáculo, com maior probabilidade de estar vazia.
- A região ao redor da posição do obstáculo, com maior probabilidade de estar ocupada.
- A região posterior à posição do objeto, da qual não há nenhuma informação obtida.

Não há como definir-se exatamente quando uma região termina e quando outra começa, e por isso são definidos valores probabilísticos, que transformam a Figura

2.2.1 em uma imagem tridimensional, cuja altura indica a probabilidade de uma determinada região estar ocupada.

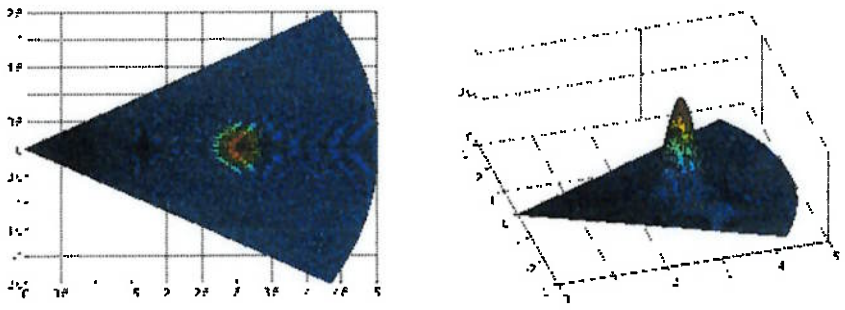


Figura 15 - Exemplo de distribuição gaussiana para sonares.

Essas distribuições podem então ser incorporadas ao mapa do ambiente que está sendo construído, criando os chamados mapas probabilísticos, que armazenam não apenas a existência ou ausência de obstáculos em uma determinada região observada pelo sensor, mas sim a sua probabilidade de existência. O ambiente que será mapeado é discretizado e a cada célula é atribuído um valor que pode variar entre 0 (certeza de ausência de obstáculos) e 1 (certeza de existência de obstáculos). Valores intermediários refletem a probabilidade de existir algo naquela região, mostrando a incerteza inerente aos sensores e permitindo um tratamento dela.

Assim sendo, cada leitura de um sensor de sonar do robô não acarretará a atualização de uma única célula no mapa, mas sim de um conjunto delas, cada uma tendo o seu valor alterado para refletir a informação recebida. Caso o valor de atualização seja baixo, a probabilidade de existir obstáculos naquela área diminui, e caso seja alto o valor aumentará. Assim diminui-se a incorporação de erros no mapa, pois uma grande quantidade de observações realizadas em uma mesma área fará com que o seu valor se aproxime daquele representado pela realidade, com erros aleatórios e instantâneos sendo compensados a longo prazo.

3.5. Modelos Desenvolvidos

A etapa de modelagem dos erros inerentes aos sensores foi realizada com base nos sensores do próprio *MagellanPro*, procurando fazer com que os sensores do modelo virtual se aproximassem o mais possível daqueles apresentados pelos seu robô real. Foram realizados diversos testes com os seus sonares, baseados na colocação de objetos em diferentes distâncias e ângulos, seguidos pela coleta dos valores fornecidos pelos sensores, criando uma lista de valores reais e valores medidos. Esses testes foram realizados com o robô parado e em movimento translacional e rotacional, para refletir as condições nas quais ele será submetido em aplicações reais, assim como a utilização de objetos compostos por diferentes materiais.

Esses valores foram então tratados de forma a montar uma distribuição normal, que representará a distribuição dos erros ao redor do valor fornecidos diretamente pelo sonar, criando uma área de probabilidade de existência do objeto detectado. Os resultados são mostrados na Figura 16, mostrada a seguir.

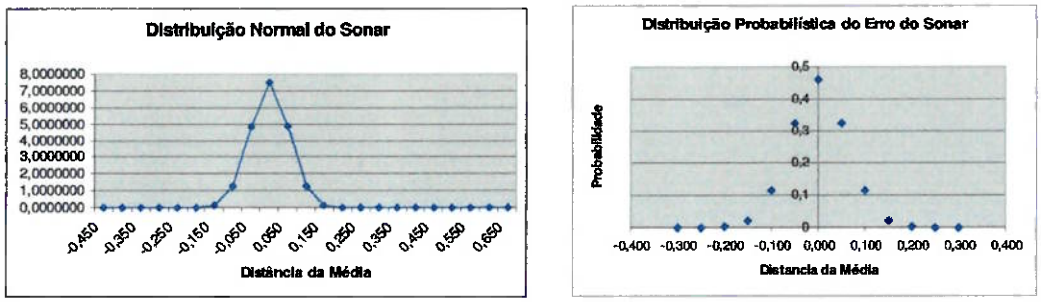


Figura 16 - Distribuição dos erros nos sonares do MagellanPro.

O mapeamento do ambiente pelo *MagellanPro* fará uso de discretização de 50 mm, e pelos resultados alcançados consegue-se chegar até um erro de localização de obstáculos máximo de 200 mm. A maioria desses erros, contudo, está localizada dentro do intervalo de 100 mm, ou uma célula da matriz discretizada o que indica um erro pequeno de acordo com a resolução que se procurará utilizar. Outro fator que pode influenciar o resultado final foi a alta taxa de "falsos positivos" (7,5%), casos em que o sonar retorna a ausência de obstáculos quando eles existem naquela região varrida pelo sonar, por não ter recebido de volta o pulso emitido. Nesses casos não há

uma distribuição de erro propriamente dita, pois não é retornada uma distância, e por isso eles não podem ser considerados juntamente com o modelo que está sendo construído, sendo necessário um tratamento à parte.

Já para os testes de odometria foram utilizados como parâmetros os próprios valores de odometria, devido à dificuldade em se definir uma posição exata para o robô e repetí-la durante vários experimentos independentes. Iniciando-o a uma determinada distância de uma superfície capaz de refletir corretamente o ultra-som fez-se com que ele seguisse em linha reta, e recolhendo-se os valores fornecidos pela odometria e a variação de distâncias pelo sonar foi possível definir os erros inerentes à odometria. Outra experiência consistiu em fazer o robô caminhar paralelamente a uma parede, medindo-se a variação da sua distância em relação a ela, o que permitiu a determinação de erros de rotação durante a navegação. A curva de distribuição alcançada é mostrada a seguir (figura 17), sendo similar àquela adotada em artigos que fazem uso de robôs similares:

Figura 17 - Distribuição dos erros de odometria no MagellanPro.

O modelo de erro obtido nessa etapa será utilizado também na determinação das partículas que serão utilizadas pelo *Filtro de Partículas* na determinação da localização do *MagellanPro* a cada instante, durante a realização do SLAM. A cada iteração serão sorteadas posições dentro desse modelo de erro que poderão representar a localização do robô, e aquelas mais prováveis serão mantidas e utilizadas como base na determinação de novas partículas na próxima iteração. Uma área muito elevada pode levar à necessidade de utilização de muitas partículas, aumentando o custo

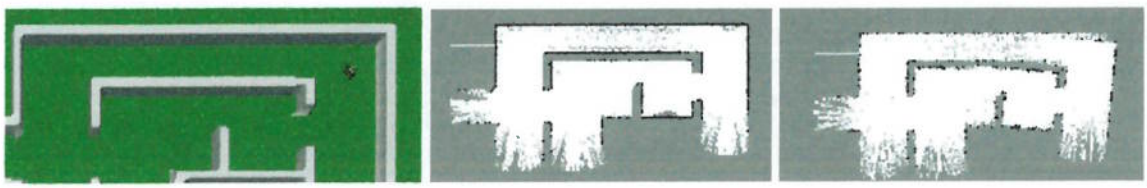
computacional, e uma área diminuta pode não conseguir acompanhar o sistema caso os erros obtidos pela odometria sejam muito elevados.

3.6. Implementação

A etapa de modelagem dos erros dos sensores do *MagellanPro* foi precedida por uma etapa de construção de rotinas básicas de movimentação para o próprio robô, assim como para o seu modelo virtual, pois ambos podem ser acessados da mesma forma através da interface *Player,* e possuem características geométricas similares. Rotinas de rotação e translação foram desenvolvidas, assim como rotinas de acesso aos seus dados de sonar e de odometria, possibilitando um controle inicial do sistema que permitiria a realização dos testes posteriores.

Dessa forma conseguiu-se uma biblioteca composta por comandos simples, como rotacionar o robô de um ângulo fixo, ou fazê-lo caminhar em linha reta por uma determinada distância, parando quando alcançar o seu objetivo de acordo com seus valores de odometria. Esses comandos puderam então ser unidos de forma a dar origem a rotinas mais complexas, como movimentação livre do robô, realizando simultaneamente rotação e translação, e também controle de velocidade baseado na proximidade com obstáculos reconhecidos pelo sonar. Seguiu-se então na implementação de um sistema de mapeamento, que recebesse os valores fornecidos pelos sonares e construísse iterativamente o mapa do ambiente, conforme o robô se movimenta.

Em última análise, conseguiu-se alcançar os resultados esperados por um sistema real quando não submetido aos erros inerentes à utilização de sistemas reais, com imprecisões inerentes. Seguiu-se então à inserção dos modelos de erros construídos, de maneira a fazer com que o algoritmo de SLAM que será desenvolvido se torne necessário também para o modelo virtual. Os resultados são então mostrados nas figuras a seguir (figura 18).



(a) (b) (c)

Figura 18 - Ambiente virtual e resultados de mapeamento
.(a) Ambiente virtual. (b) Mapeamento sem erros. (c) Mapeamento com erros.

Pode-se perceber facilmente na figura 2.4.1 a influência que os erros inerentes aos sensores possuem no resultado final alcançado pelo mapeamento de ambientes por um robô móvel. Quando não levados em consideração no sistema (figura central) o mapeamento ocorre de maneira bastante precisa, conseguindo manter a coerência mesmo após grandes distâncias e períodos de navegação. Já quando esses erros são incorporados logo surgem inconsistência do ambiente, principalmente quando o robô passa diversas vezes pela mesma área, que não serão representadas da mesma forma devido ao erro acumulado entre esses dois momentos. Fica clara a necessidade de um algoritmo de SLAM que consiga manter esses erros pequenos o suficiente para que o resultado final não fique comprometido da maneira como foi mostrada.

4. DP-SLAM

O problema do SLAM, como já foi mostrado anteriormente, é atualmente um tema bastante explorado na comunidade científica, devido à sua importância crescente e aplicações em diversas outras áreas de navegação autônoma de robôs. Diversas abordagens e ferramentas já foram desenvolvidas na tentativa de se abordar esse problema, cada uma fazendo uso de princípios e conceitos diferentes para se chegar aos mesmos resultados. Após um estudo inicial dos diversos algoritmos já existentes, escolheu-se uma delas, o *DP-SLAM* ("Distributed Particle Simultaneous Localization and Mapping") para um estudo mais profundo, e posterior implementação como uma maneira de se resolver o problema do SLAM como proposto nesse trabalho.

O *DP-SLAM* é uma tentativa de se resolver o problema do SLAM sem nenhuma suposição *a priori* do ambiente no qual o robô se movimentará, sendo essa uma das razões para a sua escolha. Nenhuma ferramenta de correção off-line é utilizada, com o algoritmo baseando-se apenas na coleta de informações a partir de odometria e sensores, sem a necessidade de nenhum equipamento externo. Por não fazer uso também de *landmarks* para determinar sua localização, o *DP-SLAM* também não trás consigo os problemas referentes à determinação e reconhecimento dessas estruturas.

O *DP-SLAM* funciona a partir da manutenção de um *Filtro de Partículas*, cujas partículas representam as prováveis posições do robô a cada momento, de acordo com o modelo de erro de sua odometria. Adicionalmente, a cada uma de suas partículas é atribuído um mapa do ambiente, que armazena as informações obtidas pelos sonares até aquele momento em questão, referentes àquela partícula e ao caminho percorrido por ela até chegar naquele ponto. Assim, quando uma posição é escolhida para ser a localização do robô em um determinado instante, define-se automaticamente qual é seu mapa correspondente, bastando para isso recuperar o seu mapa vinculado, e assim ambos os problemas são resolvidos simultaneamente, como é o propósito do SLAM.

A manutenção de diversos mapas permite também a manutenção de ambigüidades por um período maior de tempo, pois quando uma determinada posição é vista como incorreta todas as observações relativas a ela são descartadas sem influenciar o mapa que se torna o correto. Outras abordagens comumente fazem uso

de apenas um mapa, e distribuem todas as suas partículas sobre ele, e com isso erros gerados em casos de ambigüidade são propagados e não podem ser apagados posteriormente, comprometendo o resultado final.

Mas o armazenamento, manutenção e consulta de mapas é uma tarefa que demanda um grande custo computacional, sendo esse o principal problema que o *DP-SLAM* enfrenta para conseguir se tornar uma solução eficiente e robusta para o problema do SLAM. E para contornar essa situação foram desenvolvidas ferramentas que permitam realizar essas tarefas de uma maneira mais eficiente, criando condições para a manutenção simultânea de centenas ou até mesmo milhares de mapas enquanto ainda mantendo a execução em tempo real. Para isso procura-se evitar ao máximo a cópia de informação redundante, com cada iteração do algoritmo armazenando apenas aquela informação que é fornecida pelos sensores enquanto ainda mantém uma maneira de se recuperar aquela incorporada ao sistema em momentos anteriores.

4.1. Árvore Ancestral

A estrutura principal responsável pela manutenção das diversas partículas e mapas necessários para a utilização do *DP-SLAM* como proposto é a *Árvore Ancestral*, ou *Árvore Ancestral*, que faz uso do conceito de ancestralidade entre partículas para evitar a cópia de informação redundante em cada iteração. Uma implementação direta da construção e armazenamento de mapas acarretaria um custo proporcional a $O(MP)$, sendo M o tamanho de cada mapa e P a quantidade de partículas, ou posições prováveis para o robô, que serão utilizadas no algoritmo (esse valor pode variar de iteração para iteração). Considerando que muitas partículas são necessárias para uma implementação eficiente do *DP-SLAM* e que os mapas construídos possuirão um tamanho considerável, cobrindo uma área relativamente grande, essa abordagem rapidamente se tornaria proibitiva.

Mas, é fácil perceber que a construção de diversos mapas completos do ambiente trás consigo uma grande quantidade de informação redundante, que é repetida em todos os mapas e, por isso mesmo, não precisaria ser repetida em todas as

etapas. Para tratar isso é utilizado o conceito de ancestralidade, onde uma partícula criada em uma determinada iteração (geração atual) é vinculada à partícula que lhe deu origem (partícula de uma geração anterior). Sendo A a quantidade de células varridas pelo sensor a cada iteração, cada mapa de uma partícula da geração atual não pode se diferenciar do mapa de seu ancestral direto por mais de A atualizações.

Dessa forma, é necessário armazenar apenas os valores das células que receberam atualizações na última atualização de sensores, e como $A << M$ essa abordagem já acarreta uma diminuição considerável no custo computacional necessário para a manutenção de uma grande quantidade de mapas. Quando se deseja construir o mapa de uma determinada partícula, observa-se as atualizações feitas pela partícula em questão, refletindo a informação obtida nessa iteração. Os valores das células que não foram observadas pela partícula são procurados em seus ancestrais, retornando-se um a um até chegar à partícula inicial, aquela que deu origem a todas as outras e representa a posição inicial do robô. Caso não seja encontrada nenhuma observação, sabe-se que essa posição ainda não foi varrida pelos sensores, sendo por isso ainda desconhecida.

Para que uma *Árvore Ancestral* possa ser utilizada em aplicações que requeiram um longo período de funcionamento é necessário garantir que ela possua um tamanho finito, e um custo computacional constante em relação à quantidade de iterações já realizadas. À primeira vista isso não acontece, pois novas partículas são adicionadas a ela em cada iteração, fazendo com que ela cresça rapidamente e com ela o custo computacional. Mas pode-se provar que é possível manter uma *Árvore Ancestral* com um tamanho finito através das chamadas *Árvores Ancestrais Mínimas*, que possuem as três características mostradas abaixo:

- Possuem exatamente P partículas que não possuem filho nenhum.
- Cada uma de suas partículas que possuem filho possuem no mínimo 2 filhos.
- Apresenta uma profundidade D, que representa o número de partículas entre a raiz e uma partícula que não possua filhos, de no máximo P.

Para manter uma *Árvore Ancestral Mínima* são necessárias duas rotinas de modificação, que são chamadas no início de cada iteração visando todas as partículas

da geração imediatamente anterior. Essas rotinas têm a seguinte finalidade: apagar as partículas que não possuem nenhum filho e unir as partículas que possuem apenas um filho com o seu próprio filho. Com isso o algoritmo de SLAM alcança um tempo de execução praticamente constante e independente da quantidade de iterações que já tenham sido realizadas.

- **Apagando Partículas:** Cada partícula da geração atual possui necessariamente como pai uma partícula da geração anterior, que representa a sua posição imediatamente anterior à movimentação que a levou até aquela posição provável. Caso uma partícula da geração anterior não tenha dado origem a nenhuma partícula na geração atual, ela já não possui nenhuma função na *Árvore Ancestral*, e pode ser removida sem nenhum prejuízo. E caso essa ação faça com que a sua partícula-pai também fique sem nenhum filho, também ela deve ser recursivamente apagada, até que todas as partículas da *Árvore Ancestral* que não pertençam à geração atual possuam ao menos um filho.
- **Unindo Partículas:** Caso uma partícula da geração imediatamente anterior à atual possua apenas um filho, as suas informações referentes ao mapa que será gerado só serão utilizadas pelo seu filho, e por isso ambas podem ser unidas sem prejuízo nenhum para o resultado final. Essa ação é feita copiando-se todas as observações que o pai possui referentes ao mapa para o seu filho, ignorando-se aquelas em cujas coordenadas o filho já possua informação própria. Após isso, o pai da partícula-pai se torna o pai da partícula-filho, e por fim a partícula-filho toma o lugar da partícula-pai na *Árvore Ancestral*, e a partícula-pai pode então ser apagada, por não estar mais ligada a nenhuma partícula. Evita-se dessa forma longas cadeias que possuem apenas um único vínculo entre pais e filhos e que causaria lentidão na recuperação de informações.

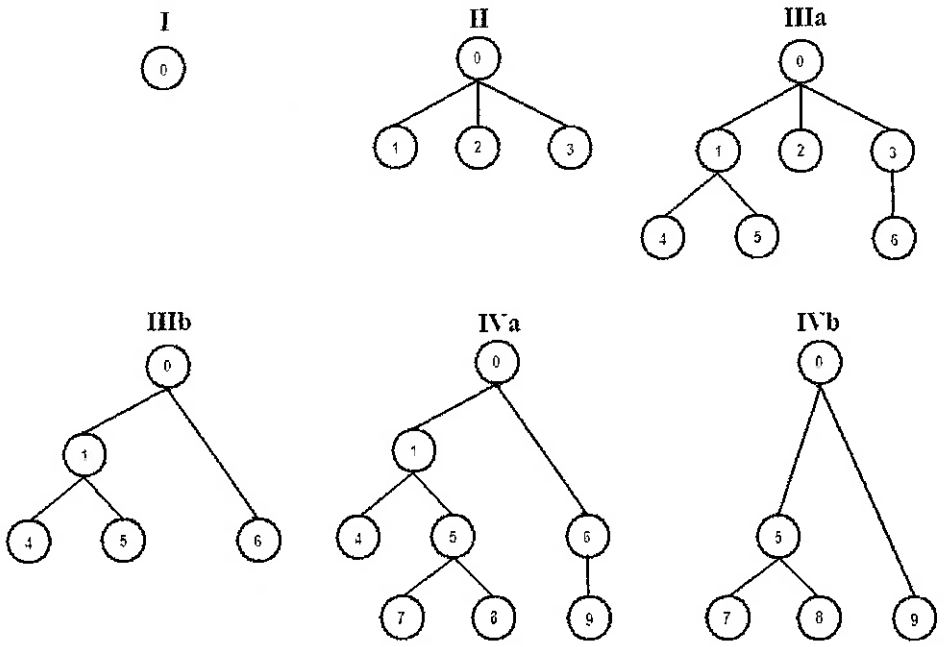


Figura 19 - Exemplo de Árvore Ancestral. (P = 3)

Na figura acima está um exemplo de uma *Árvore Ancestral*, que tem início na partícula 0 e a cada geração são adicionadas três partículas, representando novas posições prováveis do robô em um determinado instante. Partículas que não possuem nenhum filho na geração atual, como é o caso das partículas 2 e 4, são apagadas para a manutenção da *Árvore Ancestral* Mínima, como mostrado. E partículas que tenham um único filho, como 3 e 6, são unidas com seus filhos, respectivamente as partículas 6 e 9, evitando longas cadeias de um único filho. Caso seja necessária a construção do mapa relativo à partícula 7, por exemplo, suas observações são observadas inicialmente, seguidas das observações das partículas 5 e, por fim, da partícula 1. Dessa forma são determinadas todas as atualizações já feitas pelo sistema até chegar na posição relativa à partícula 7, e qualquer outra é tida como desconhecida.

4.2. Mapa de Observações

Embora o armazenamento das várias diferenças entre mapas em princípio seja suficiente para se resolver o problema do mapeamento de diversas posições possíveis

para o robô, ele não é eficiente para se abordar o problema da localização. Durante a etapa de localização é necessária a construção dos mapas relativos a todas as partículas da geração atual, que são então comparados com as observações realizadas pelos sensores, e com isso possa ser determinada a probabilidade de cada mapa ser o correto.

Caso sejam armazenadas apenas as diferenças entre os mapas de uma geração de partículas para a próxima, a construção de mapas requisitaria a passagem por todas as gerações anteriores àquela à qual a partícula cujo mapa será reconstruído pertence. Com isso surge uma relação entre o custo computacional e a quantidade de iterações já realizada, o que deve ser evitado, como já foi dito anteriormente. Também dessa forma a reconstrução de mapas só é possível de maneira completa, sendo necessária a obtenção das informações referentes não apenas às regiões já observadas pelos sensores como também daquelas ainda desconhecidas.

Esses problemas podem ser contornados com o *Mapa de Observações*, que juntamente com a *Árvore Ancestral* permite a manutenção de toda a informação necessária para a implementação do *Filtro de Partículas* que tratará o problema do SLAM da foram como o *DP-SLAM* propõe. Um *Mapa de Observações* é composto por uma matriz de tamanho *M*, com cada célula representando uma área discretizada do mapa, que será preenchida conforme o robô se movimenta pelo ambiente. Cada uma das células é composta por uma estrutura similar a uma *Árvore Ancestral*, com uma partícula inicial que dá origem às outras, cada uma representando a observação daquela célula por uma partícula em específico.

Cada partícula possui um valor único de identificação, e cada uma das observações feitas é armazenada no *Mapa de Observações*, na sua respectiva coordenada e carregando um valor de probabilidade de ocupação e o valor de identificação de sua partícula correspondente. A relação de ancestralidade é mantida no *Mapa de Observações*, mantendo-se as mesmas relações, sendo que a lista de ancestrais construída na *Árvore Ancestral* é utilizada na procura por observações referentes a uma determinada partícula. As mesmas rotinas de manutenção utilizadas na obtenção da *Árvore Ancestral Mínima* devem ser também utilizadas no *Mapa de Observações*, para que também ele mantenha um tamanho finito independente da

quantidade de iterações já realizada. Dessa forma, consegue-se construir mapas apenas parciais, com a escolha de quais coordenadas dentro do *Mapa de Observações* serão recuperadas, garantindo uma maior velocidade na reconstrução de mapas necessária para a etapa de localização.

4.3. Filtro de Partículas

A função principal de um *Filtro de Partículas* é acompanhar uma variável de interesse conforme ela evolui no tempo, seguindo uma função cujas características não são conhecidas, que por algum motivo não pode ser modelada matematicamente. Eventos acontecem para modificar o estado da variável, e observações são realizadas em determinados períodos de tempo que permitem o acompanhamento dessa variável e a realização de correções quando necessário.

Múltiplas cópias (partículas) dessa variável são utilizadas, cada uma associada com um peso que significa a qualidade daquela cópia, ou o quão bem ela representa a realidade. Por ser uma ferramenta recursiva, a cada iteração do algoritmo são criadas novas partículas, e esses pesos são atualizados de maneira a refletir a incorporação de novas informações fornecidas pelos sensores. O conjunto de partículas e de seus respectivos pesos permite a estimativa do verdadeiro estado da variável, existindo diversas maneiras de se chegar a esse resultado.

Um *Filtro de Partículas* possui basicamente três fases, *predição, resampling* e *update,* que se repetem recursivamente de maneira a produzir os resultados desejados. Após uma determinada ação, cada partícula é modificada de acordo com o modelo existente (predição), de maneira a se aproximar do estado que se supõe que ela alcançará após esse período. A essa predição é adicionado um ruído aleatório de forma a simular as imprecisões que ocorrerão e que por algum motivo não puderam ser modeladas, seja pela complexidade ou pela impossibilidade de previsão. Com isso, um conjunto novo de partículas é gerado, no processo conhecido como *resampling* que faz com que as partículas mais prováveis de serem corretas dêem origens a uma quantidade maior de descendentes, e aquelas menos prováveis sejam eliminadas do

processo. Então, o peso de cada partícula é recalculado de acordo com a informação sensorial que é recebida, representando a etapa da atualização.

4.3.1. Predição

A etapa de *predição* é realizada com base no modelo previamente construído de movimentação do robô, fazendo uso de sua posição atual para definir qual será a sua posição após um determinado período de tempo ter se transcorrido. No caso do *DP-SLAM*, é utilizada a posição do robô definida pela partícula-pai, e a partir dela são derivadas uma certa quantidade de partículas, cada uma propagada de acordo com o modelo. A essa posição é incluída uma certa quantidade de ruído, de acordo com o modelo de erro de odometria do sistema, de maneira a simular os erros de localização que possam surgir durante essa movimentação. À posição de cada partícula é adicionado um valor aleatório diferente, de forma a gerar um conjunto de posições distintas.

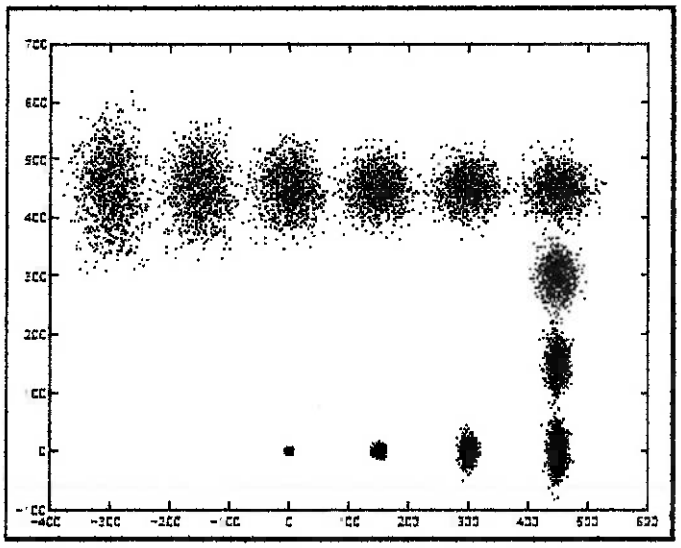


Figura 20 - Distribuição de partículas de acordo com a movimentação do robô.

4.3.2. Resampling

Uma primeira abordagem para o *Filtro de Partículas* seria utilizar sempre as mesmas partículas iteração após iteração, propagando-as de acordo com o modelo e adicionado o ruído de forma usual. Espera-se, com isso, que dentre diversas partículas ao menos uma consiga acompanhar o movimento do robô de maneira satisfatória, mantendo um peso elevado durante todo o processo. Mas o que acontece na prática quando é utilizado sempre o mesmo conjunto de partículas é a divergência entre suas posições representativas, com a maioria das partículas se afastando da posição verdadeira, deixando de serem úteis e aumentando o custo computacional sem melhorar o resultado final.

Para evitar essa divergência de partículas utiliza-se então a etapa de *resampling,* que consiste em descartar as partículas cujo peso se encontra abaixo de um certo limiar e na multiplicação das partículas com um peso elevado. Assim sendo, aquelas partículas que estão seguindo por um caminho que se sabe errado são descartadas, e aquelas que estão seguindo pelo caminho suposto correto são multiplicadas de forma a aumentar a probabilidade de que alguma delas permaneça no caminho correto.

A maneira mais simples de se realizar um *resampling* inicia-se na normalização dos pesos das partículas, realizada pela divisão de seus valores pela soma, de forma que a soma de todos os pesos seja unitária. Após isso, N números são sorteados aleatoriamente dentro do intervalo [0,1], com N sendo a quantidade de partículas que serão descartadas do conjunto para que outras sejam criadas, geralmente o próprio valor P. Após isso, para cada um desses valores segue-se desde a primeira partícula do conjunto atual, somando seus pesos até que a soma final seja maior do que o valor sorteado.

A quantidade de valores que estiverem dentro do intervalo de cada uma das partículas atuais representa a quantidade de cópias dessa partícula que serão propagadas para a próxima iteração. Partículas que possuem um peso maior ocuparão uma faixa maior de valores, aumentando a probabilidade de que os valores aleatórios sorteados estejam dentro de seus limites, e com isso uma maior quantidade de partículas seja escolhida. Outras abordagens foram desenvolvidas no intuito de

acelerar o processo de obtenção de um novo conjunto, permitindo uma maior eficiência também nessa etapa do *Filtro de Partículas*.

4.3.3. Update

É durante a etapa de u*pdate* que as informações do ambiente, coletadas pelos sensores, são recebidas pelo robô, permitindo que ele possa realizar correções relacionadas à etapa de predição, quando foi utilizado o modelo construído. Com base nesses valores são recalculados os pesos referentes a cada uma das partículas, de forma a definir quais possuem maior probabilidade de serem as corretas, e que serão propagadas para a etapa seguinte.

As regiões relevantes dos mapas referentes a cada uma das partículas da geração atual são reconstruídas, indicando qual seria a informação que os sensores deveriam retornar caso o robô se encontrasse naquela posição. Essas regiões são então comparadas com a informação que é de fato recebida, e a partir dessa comparação os pesos das partículas são atualizados, aumentando ou diminuindo conforme a proximidade dos valores do seu mapa correspondente com a informação dos sensores.

Esse novo conjunto é então propagado adiante através de uma nova etapa de predição, caso existam muitas partículas com peso próximo de zero ocorre uma etapa de *resampling*, e então uma nova etapa de *update* acontece, agora fazendo uso da informação coletada na última atualização para definir a nova localização. Todo esse processo deve ser realizado dentro do período de tempo entre atualizações dos sensores, e por isso o ganho de velocidade em qualquer uma das etapas é fundamental, principalmente quando nesse mesmo tempo acontece a construção e manutenção dos mapas, através da *Árvore Ancestral* e do *Mapa de Observações*.

5. Implementação do Algoritmo

O algoritmo de SLAM aqui proposto foi escrito em linguagem C++, aproveitando-se de seus recursos de orientação a objetos, com classes representando suas principais estruturas. Para realizar a interface entre as rotinas programadas e o sistema a ser controlado foi utilizada uma biblioteca referente ao *Player*. Essa biblioteca (*playerc++.h*) permite a coleta de informações referentes aos seus sensores e à sua odometria e o envio de informações referentes à velocidade linear e angular. Essa interface pode ser utilizada tanto no *MagellanPro* quanto no seu modelo virtual, sem a necessidade de nenhuma alteração no código do programa, bastando a utilização de *drivers* específicos para cada plataforma.

A navegação acontece a partir do fornecimento de um conjunto de pontos previamente determinados, que o robô procurará alcançar em uma dada seqüência, utilizando-se para isso das rotinas de navegação construídas. A cada iteração dos sensores é realizado o *DP-SLAM*, procurando compensar os erros acumulados no período de navegação cega do robô, impedindo que eles eventualmente se tornem grandes o suficiente para invalidar o resultado final. A posição mais provável a cada momento é escolhida como sendo a verdadeira e é enviada para o robô de maneira a atualizar o seu sistema de odometria, substituindo a informação calculada internamente e sujeita a erros. A seguir é explicada em maiores detalhes a maneira como cada uma das estruturas responsáveis pelo *DP-SLAM* foi implementada.

Cada uma das estruturas necessárias para o funcionamento do *DP-SLAM* (*DP-Mapping* e *Filtro de Partículas*) foi implementada inicialmente de maneira separada, agindo independentemente do restante do algoritmo. Uma base de dados foi construída, contendo possíveis informações que simulariam aquelas que o sistema receberia durante aplicações reais, assim como os resultados esperados que cada rotina deveria fornecer em resposta. Assim, as estruturas puderam ser testadas separadamente, de forma a garantir seu funcionamento isoladamente antes de serem unidas em um único algoritmo, o *DP-SLAM*.

5.1. DP Mapping

Foi desenvolvida uma classe que representa o *DP-Mapping*, incorporando a estrutura de *Árvore Ancestral* e o *Mapa de Observações,* permitindo a variação de todos os seus parâmetros. Como entrada é fornecida a quantidade de partículas que serão criadas a cada iteração do algoritmo, assim como a quantidade máxima de observações que os sensores podem realizar em cada uma de suas atualizações. A discretização que será utilizada no mapa também deve ser fornecida, assim como o seu tamanho inicial, embora esse valor seja alterado automaticamente pelo sistema caso o robô se movimente além das suas fronteiras determinadas dessa forma. Durante os testes e na obtenção dos resultados finais foi utilizada uma discretização de 0.01 metros.

Para a construção da *Árvore Ancestral* utiliza-se uma estrutura de árvore, com a partícula-raiz sendo a posição inicial do robô, definida como *(0,0,0)* por convenção. A partir desse ponto, a cada iteração são construídas *P* partículas,que armazenam o seu ID, número inteiro que a identifica unicamente dentro de todas as partículas, e também sua posição e orientação no plano de navegação *(X,Y,A),* e o seu peso probabilístico *W*, utilizado pelo *Filtro de Partículas.* Além disso, também são armazenadas, no intuito de facilitar a construção e manutenção da *Árvore Ancestral*, qual a sua posição relativa à ordem dos filhos de sua partícula-pai, e a quantidade total de filhos que ela própria possui.

Cada partícula também possui um ponteiro que a vincula com a sua partícula-pai e outro conjunto de ponteiros que a vincula com os seus filhos. Inicialmente foi decidido definir *P* ponteiros para as partículas-filho, e utilizar apenas aqueles necessários, mas a definição de ponteiros mostrou-se uma tarefa com um custo computacionalmente elevado, e por isso atualmente essa tarefa é feita dinamicamente. Após terem sido determinados todos os filhos de todas as partículas, esse vetor é ordenado de acordo com os respectivos pais das partículas, e com isso consegue-se construir apenas os ponteiros necessários para a criação da nova geração.

Cada uma das partículas também conta com uma lista ligada, que indica as coordenadas nas quais essa partícula possui observações. Foi escolhida uma lista

ligada porque devido às rotinas de manutenção da *Árvore Ancestral,* a quantidade de observações que uma partícula realiza se altera dinamicamente, recebendo novos valores quando há a união entre duas partículas. Ao mesmo tempo, cada uma dessas observações é adicionada na sua coordenada correspondente no *Mapa de Observações*, que é composto de uma matriz com cada célula possuindo uma estrutura similar a uma *Árvore Ancestral.* Ao invés de armazenar posição, orientação e peso, é armazenado o valor respectivo àquela coordenada de acordo com a partícula que a observou, seja um valor probabilístico ou um valor binário.

As coordenadas armazenadas pela partícula são utilizadas somente na ocasião em que essa partícula vai ser excluída, momento esse em que todas essas coordenadas são varridas em busca das observações que possuem o mesmo ID da partícula. Para a reconstrução de mapas, um raciocínio similar é feito, utilizando-se as coordenadas que deseja-se reconstruir, permitindo que apenas uma região do mapa seja recuperada, poupando trabalho computacional desnecessário. Atualizações realizadas na *Árvore Ancestral,* oriundas das rotinas de apagar e unir partículas, devem ser propagadas para o *Mapa de Observações*, de maneira a manter similares as suas estruturas.

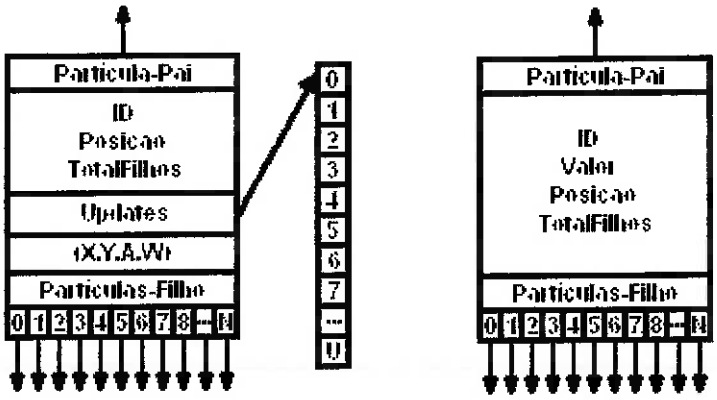


Figura 21 - Estrutura computacional de uma partícula e de uma observação.

A reconstrução de um mapa é realizada a partir da obtenção do caminho ancestral de uma partícula que se encontra na geração atual. A partir do ID da partícula, ela é encontrada no vetor de partículas da geração atual, construído anteriormente à incorporação das partículas na *Árvore Ancestral.* Uma vez localizada,

pode-se seguir o seu ponteiro relativo ao seu pai para encontrar a partícula que deu origem a ela, e que possui suas observações imediatamente anteriores. Esse processo se repete iterativamente enquanto for possível, até uma partícula que não possui pai, que seria a partícula-raiz.

Com isso, gera-se uma lista de ancestrais, que é utilizada ao contrário para a busca do ID no *Mapa de Observações* relativo às coordenadas procuradas, iniciando-se pela partícula-raiz e seguindo até a partícula procurada. Essa seqüência de ancestrais não precisa necessariamente ser seguida pelas observações, pois os sensores de uma partícula podem coletar informações de células que não foram observadas pelas suas partículas anteriores. Mas, ainda assim, a ordenação geral é sempre obedecida, e caso não seja possível encontrar a próxima partícula da seqüência, deve-se seguir para a próxima, e assim até chegar-se à partícula desejada. Caso essa partícula não seja encontrada, utiliza-se a observação da sua partícula ancestral mais próxima, em um processo retroativo que termina com a própria partícula-raiz do *Mapa de Observações* da coordenada em questão.

5.2. Filtro de Partículas

O *Filtro de Partículas* implementado no algoritmo de SLAM aqui desenvolvido teve como objetivo o tratamento probabilístico das diversas posições possíveis que o robô pode assumir em cada momento. A quantidade de partículas que pode ser utilizada pelo *Filtro de Partículas* é limitada, e portanto deve-se procurar manter sempre apenas aquelas que possuem uma maior probabilidade de estarem corretas, refletindo com maior fidelidade o ambiente no qual o robô se movimenta.

Essa semelhança entre o mapa construído e a realidade observada pelos sensores foi calculada a partir da comparação entre ambos os valores, com o Método dos Mínimos Quadrados. Essa comparação é realizada entre a posição dos obstáculos encontrados nos mapas e aqueles mostrados pelos sensores, e repetida para todos os 16 sensores possuídos pelo *MagellanPro.* A partir da posição indicada pelo sensor, é realizada uma média entre todas as posições marcadas como *ocupadas* e que estejam dentro de um certo intervalo dessa posição, no mesmo segmento de reta determinado

pelo caminho do sonar. Essa média é subtraída do valor calculado pelo sonar e então levada ao quadrado, mostrando o erro entre a informação do sonar e aquela incorporada ao mapa.

$$E = \sum_{j=0}^{15} \left(\sum_{k=-5}^{5} C_k - V_o \right)^2 \tag{16}$$

Esse erro é então atribuído como peso da sua partícula referente, indicando a semelhança entre o mapa construído e o ambiente percebido pelos sensores, sendo que quanto menor o valor, maior é essa semelhança. Esse cálculo é repetido para todas as partículas da geração atual, e então os pesos encontrados são normalizados, de forma que sua soma seja igual a 1 e eles passem a representar uma probabilidade. Adicionalmente, optou-se também por inverter a relação entre o valor e a probabilidade, de forma que quanto maior o valor, maior a probabilidade, o que será útil na próxima etapa (equação 17).

$$N = \sum_{i=0}^{P} \left(1 - E_j\right) \quad E_j = \frac{\left(1 - E_j\right)}{N} \tag{17}$$

A próxima etapa é a determinação de quais partículas serão propagadas para a próxima etapa, dando origem a uma quantidade maior de descendentes, e quais serão descartadas. Para isso são sorteados *P* valores aleatórios entre 0 e 1, que são então comparados com o vetor de partículas atuais, com cada uma representando um intervalo de probabilidade. Iniciando-se de um dos extremos do vetor, segue-se somando cumulativamente os pesos das partículas, até que esse valor ultrapasse o valor sorteado. Essa será então a partícula escolhida, que dará origem a uma próxima partícula na geração atual. Pode-se perceber que partículas que possuem uma probabilidade maior de estarem corretas possuem também uma maior chance de serem sorteadas, dando origem com isso a uma partícula na iteração posterior.

Uma vez determinados os pais de cada uma das partículas atuais, elas podem ser adicionadas à *Árvore Ancestral,* posicionadas como descendentes das suas partículas-pai, e recebendo os valores de posição e orientação através da aplicação dos modelos de odometria utilizados e também o ruído gerado pelo modelo de erros.

Partículas que não dão origem a nenhum descendente nessa geração podem ser eliminadas, pois não serão mais utilizadas em gerações posteriores e suas informações já não são mais importantes. Aquelas que têm apenas um descendente podem ser unidas com esse descendente, unindo ambas as observações em uma única partícula com o maior ID.

6. Resultados

Durante o período de desenvolvimento do algoritmo foram realizados testes constantes em ambiente virtual, como uma forma de se validar os resultados parciais que cada uma das ferramentas construídas eram capazes de fornecer. Conseguiu-se dessa forma uma maior segurança na realização dos testes preliminares, poupando a utilização do equipamento real enquanto os resultados não atingiam um patamar seguro de utilização, assim como o isolamento dos erros que possam vir a surgir devido a falhas sistemáticas em partes do algoritmo.

Uma vez que o *DP-SLAM* foi totalmente implementado e seus resultados preliminares se mostraram compatíveis com o que se esperava, iniciou-se os testes finais, ainda em ambiente virtual onde era possível um controle maior das variáveis envolvidas. A discretização utilizada na construção do *Mapa de Observações* foi de 0.01 m, criando regiões de 1 cm^2 que poderiam estar *livres* ou *ocupadas* a cada instante. Procurou-se manter uma resposta em tempo real para o algoritmo, o que proporcionaria uma navegação constante do sistema, sem a necessidade de pausas para o processamento. Foi escolhida uma freqüência de 10 Hz para a amostragem de informações relativas aos sensores do robô, significando que todo o processamento deve ser realizado dentro do período de 0,1 segundos.

Esse tempo necessário pode ser alterado através da modificação da quantidade partículas que é utilizada a cada iteração do algoritmo, com quantidades maiores significando um tempo computacional maior. Durante os testes virtuais conseguiu-se manter uma quantidade de 150 partículas para uma navegação em tempo real, uma quantidade ainda baixa para os padrões necessários em aplicações confiáveis do SLAM. Quantidades maiores de partículas faziam com que o processamento saísse de sincronia com o modelo virtual do robô, tornando-o instável e arruinando o mapeamento e a localização. Os resultados referentes aos testes do *DP-SLAM* em ambientes virtuais são representados na figuras figuras a seguir (figura 22):

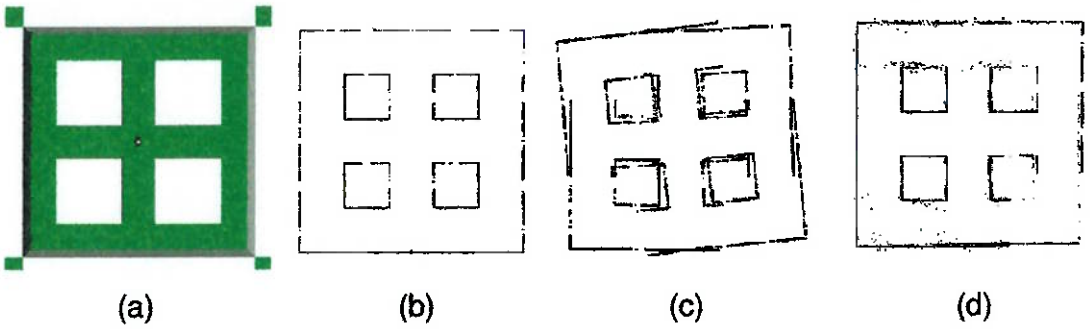


Figura 22 - Resultados finais alcançados em ambiente virtual.
(a) Ambiente virtual. (b) Sem erros. (c) Com erros. (d) Com SLAM

Através da figura acima pode-se perceber a influência que os modelos de erros desenvolvidos para os sensores do robô influenciam no resultado final do mapeamento, e os benefícios que um algoritmo de SLAM pode fornecer ao eliminar esses erros. A primeira imagem representa o ambiente virtual no qual o robô se movimentou, navegando pelos corredores até eventualmente retornar até o ponto inicial. O resultado do mapeamento resultante dessa navegação é mostrado na segunda imagem, que ainda não leva em consideração os erros dos sensores, o que permite a construção de um mapa perfeito do ambiente.

A terceira imagem é o resultado dessa mesma navegação, agora levando em consideração os erros inerentes aos sensores que foram modelados e incorporados ao modelo virtual do robô. Pode-se perceber claramente a diminuição na qualidade final do mapa construído, em especial a dificuldade em se alinhar regiões nas quais o robô passou mais de uma vez. Os erros acumulados durante a navegação fazem com que uma mesma região seja mapeada de maneira diferente a cada vez em que é visitada, impedindo que o sistema reconheça áreas nas quais já passou. Em diversas ocasiões o robô nem ao menos foi capaz de completar o percurso, colidindo com os obstáculos e não conseguindo avançar.

Já a última imagem mostra a mesma navegação, também com os erros inerentes ao sistema, mas com o algoritmo de *DP-SLAM* aqui desenvolvido agindo sobre os dados coletados pelos sensores e atualizando dinamicamente a posição do robô. É possível observar uma sensível melhora no resultado final, com os erros de alinhamento sendo praticamente eliminados, o que comprova a eficiência do algoritmo

em eliminar sistematicamente os erros gerados durante a navegação, impedindo que eles se acumulem a ponto de invalidar o resultado final.

A próxima etapa consistiu em testes em ambientes reais, tendo como plataforma o próprio *MagellanPro*, para o qual o algoritmo de *DP-SLAM* foi desenvolvido. Optou-se pela mesma freqüência de obtenção de informações de seus sensores, que já haviam mostrado serem suficientes para uma coleta eficiente de dados do ambiente. O processamento do algoritmo, embora pudesse ser realizado embarcado dentro do próprio robô, que conta com um computador próprio, foi realizado em um computador externo. As informações relativas aos sonares e à odometria eram enviadas via *wireless* para a central de processamento, que as tratava da maneira correta e enviava comandos para os atuadores do robô, coordenando sua navegação.

A razão para essa escolha está na possiblidade da utilização de processadores mais velozes, permitindo a utilização de uma quantidade maior de partículas do que seria possível caso fosse utilizado apenas o processador do *MagellanPro*. Adicionalmente, os testes em ambientes reais permitiram a utilização de mais partículas por não necessitarem do simulador, que funcionava em paralelo com o algoritmo durante os testes virtuais e tomava parte do tempo de processamento. Conseguiu-se, com isso, alcançar um processamento em tempo real com até 200 partículas, e os resultados são mostrados a seguir:

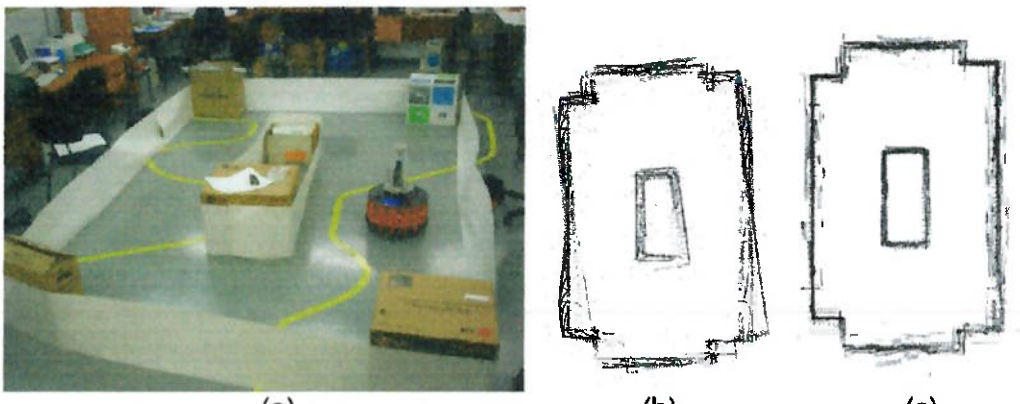


Figura 23 - Resultados finais alcançados em ambientes reais.
(a) Pista de testes. (b) Sem SLAM. (c) Com SLAM.

A primeira imagem mostra a pista de testes construída para a navegação do robô, composta por uma região retangular extena com dimensões 5 x 3 metros, e uma região retangular interna com dimensões 1,5 x 0,6 metros. Através da interface proporcionada pelo *Player* foi possível utilizar o algoritmo construído para o modelo virtual sem nenhuma alteração, inclusive as rotinas de navegação desenvolvidas. A segunda imagem representa o mapa gerado após a navegação completa do robô ao redor da pista de testes, com início e final na região inferior direita.

Pode-se perceber claramente o erro de alinhamento gerado pelo acúmulo dos erros que surgemconforme o robô se movimenta, causando uma deformação dos objetos mapeados. Esse erro se torna evidente durante a etapa de "closed loop", quando o robô retorna até um local já visitado previamente, e o mapa construído durante essa segunda visita não condiz com aquele construído inicialmente. Esse problema é resolvido com a introdução do *DP-SLAM*, que elimina os erros gerados pelos sensores a cada iteração, mantendo-os sempre residuais e impedindo-os de se acumularem a ponto de causarem erros visíveis de mapeamento.

7. Conclusão

Os resultados alcançados durante as etapas de testes às quais o algoritmo desenvolvido nesse trabalho foi submetido, tanto virtuais como reais, comprovam a sua eficiência na resolução do problema do SLAM, validando-o como uma solução para o problema da localização e mapeamento simultâneos. Através do *Filtro de Partículas* implementado, o sistema é capaz de incorporar os erros inerentes aos sensores ao seu próprio modelo de navegação, e com isso eliminá-los antes que se acumulem. Dessa forma, conseguiu-se mapas coerentes com o ambiente pelo qual o robô se movimenta mesmo após um grande período de navegação e grandes distâncias terem sido percorridas.

O armazenamento de mapas necessário é realizado de maneira eficiente através dos conceitos do *DP-Mapping*, que proporcionou um aumento por um fator de 10 na quantidade de mapas que podem ser tratados enquanto ainda mantendo execução em tempo real. Mesmo assim, durante a etapa de testes não foi alcançada uma quantidade suficiente para aplicações confiáveis do SLAM, que se encontra por volta de 1000 partículas. Uma quantidade maior de partículas significa maior probabilidade de surgir uma partícula que representa corretamente a posição do robô em um determinado instante, além de permitir um melhor tratamento de ambigüidades. Processadores mais velozes permitiriam a utilização de mais partículas, assim como um aumento da eficiência do próprio algoritmo referente ao *DP-Mapping*, algo que tem sido alvo de pesquisa constante na comunidade científica.

O tipo de sensor utilizado, do tipo sonar, não se mostrou particularmente eficiente na obtenção de informações do ambiente, por se tratar de um sensor esparso, capaz de fornecer informações apenas pontuais. Dessa forma, não é obtida a cada iteração uma quantidade de informações suficiente para uma localização precisa do robô, com mapas distintos apresentando uma mesma probabilidade de serem corretos. A utilização de sensores mais densos, como é o caso do laser, permitiria um mapeamento mais eficiente do ambiente e também uma localização mais precisa do robô. Outra possibilidade seria a utilização de outro tipo de sensor para complementar a informação que é obtida pelos sonares, como um sistema de visão.

Essas limitações encontradas durante as etapas de desenvolvimento e implementação do algoritmo de *DP-SLAM* serão exploradas em trabalhos posteriores. Procurar-se-á explorar de maneira mais eficiente os pontos positivos proporcionados pela abordagem escolhida, e contornar as suas limitações, através da proposição de novas ferramentas para a realização das diversas etapas necessárias para a localização e mapeamento simultâneos. Espera-se, com isso, alcançar resultados ainda mais robustos e confiáveis, permitindo a utilização dos resultados obtidos em ambientes reais para a realização de outros tipos de tarefas.

8. Referências Bibliográficas

CSORBA, M. *et al.* **A Solution to the Simultaneous Localization and Map Building (SLAM) Problem.** IEEE Transactions on Robotics and Automation, Vol 17, Nº 3, June 2001.

CSORBA, M. **Simultaneous Localization and Map Building.** Robotics Research Group, Department of Engineering Science. University of Oxford. 1997.

ELIAZAR, A.; PARR, R. **DP-SLAM: Fast, Robust Simultaneous Localization and Mapping without Predetermined Landmarks.** Department of Computer Science. Duke University. 2003.

ELIAZAR, A.; PARR, R, **DP-SLAM 2.0.** Department of Computer Science, Duke University. 2004.

ELIAZAR, A.; PARR, R. **Hierarchical Linear/Constant Time SLAM Using Particle Filters for Dense Maps.** Department of Computer Science, Duke University. 2005.

ESTRADA, C.; NEIRA, J.; TARDÓS, J. **Hierarchical SLAM: Real-Time Accurate Mapping of Large Environments.** IEEE Transactions on Robotics, Vol 21, Nº 4, August 2005.

FOLKESSON, J.; JENSFELT, P.; CHRISTENSEN, H. I. **Graphical SLAM using Vision and the Measurement Subspace.** Centre for Autonomous Systems, Royal Institute of Technology. 2005.

FOX, D., BURGARD, W., THRUN, S. **Markov Localization for Mobile Robots in Dynamic Environments.** Journal of Artificial Intelligence Research, vol 11, 1999.

GUIVANT, J.; NEBOT, E. **Optimization of the Simultaneous Localization and Map-Building Algorithm for Real-Time Implementation.** IEEE Transactions on Robotics and Automation. Vol 17, nº 3, June 2001.

KOUZOUBOV, K.; AUSTIN, D. **Hybrid Topological/Metric Approach to SLAM.** Robotic System Labs, Australian National University, Canberra. 2004.

MONTEMERLO, M.; THRUN, S. **FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem.** School of Computer Science, Carnegie Mellon University & Computer Science Department, Stanfor University. 2002.

MONTEMERLO, M.; THRUN, S. **Simultaneous Localization and Mapping with Unknown Data Association using FastSLAM.** IEEE International Conference on Robotics an Automation. Vol 2, September 2003.

NEWMAN, P. WHYTE, H. **A Solution to the Simultaneous Localization and Map Building (SLAM) Problem.** IEEE Transactions on Robotics and Automation. Vol 17, Nº 3, June 2001.

REKLEITIS, I. **A Particle Filter Tutorial for Mobile Robot Localization.** Centre for Intelligent Machines, McGill University, Montreal, Québec.

SMITH, R.; SELF, M.; CHEESEMAN, P. **Estimating Uncertain Spatial Relationships in Robotics.** SRI International, California. 1990.

SUJAN, V.; MEGGIOLARO, M.; BELO, F. **Mobile Robot Simultaneous Localization and Mapping Using Low Cost Visual Sensors.** Cummins Engine Company & Pontifical Catholic University of Rio de Janeiro. 2005.

THRUN, S. **Particle Filters in Robotics.** Computer Science Department. Carnegie Mellon University, Pittsburgh. 2002.

THRUN, S.; FOX, D.; BURGARD, W. **A Probabilistic Approach to Concurrent Mapping and Localization for Mobile Robots**. Machine Learning, 31, 1998.

WELCH, G.; BISHOP, G. **An Introduction to the Kalman Filter.** University of North Carolina, Department of Computer Science. 2001.

WHYTE, H. **Localization, Mapping and the Simultaneous Localization and Mapping (SLAM) Problem.** Australian Centre for Field Robotics. SLAM Summer School, 2002.