IMPLEMENTAÇÃO DO DP-SLAM EM TEMPO REAL PARA ROBÔS MÓVEIS USANDO SENSORES ESPARSOS

VITOR CAMPANHOLO GUIZILINI, JUN OKAMOTO JR., FABIANO ROGÉRIO CORREA, VALDIR GRASSI JR.

Escola Politécnica da Universidade de São Paulo
Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos
Av. Prof. Mello Moraes, 2231
05508-030 São Paulo, SP, Brasil
E-mails: vitor.guizilini@poli.usp.br, jokamoto@usp.br, fabiano.correa@poli.usp.br, vgrassi@usp.br

Abstract - The concept of Simultaneous Localization and Mapping (SLAM) has been extensively used for autonomous navigation, as a way to compensate for the errors that sensors used to gather information from the environment inherently have. The approaches used in SLAM probabilistically incorporate such errors and are able to remove them before they accumulate and invalidate the final results. This paper describes one solution to the problem of SLAM, known as DP-SLAM, which is based not only on multiple positions of the robot in one given moment, but also on multiple maps of the environment. Each map is incomplete and built-in dynamically through the use of efficient structures of data management. The comparison between the possible maps and the information given by the sensors allows the algorithm to elect the most probable map of the environment at each moment along with its instantaneous position. This solution has been implemented using a sparse sensor to obtain information of the environment and tested in both virtual and real situations. The results show that it is possible to conduct a real-time navigation while keeping substantial improvements in both localization and mapping.

Keywords: *SLAM, Localization and Mapping, Mobile Robot, DP-SLAM, Autonomous Navigation.*

Resumo - O conceito de localização e mapeamento simultâneos (SLAM) é extensivamente utilizado na navegação autônoma, como uma maneira de compensar os erros que os sensores utilizados para obter informações do ambiente possuem e que não podem ser totalmente eliminados. As abordagens utilizadas no SLAM incorporam probabilisticamente tais erros e são capazes de removê-los antes que acumulem e invalidem os resultados finais. Esse artigo descreve uma solução para o problema do SLAM, conhecida como DP-SLAM, que é baseada não apenas em múltiplas posições do robô em um dado instante, mas também em múltiplos mapas do ambiente. Cada mapa é incompleto e construído dinamicamente através do uso de estruturas eficientes de manutenção de dados. A comparação entre cada um dos possíveis mapas com a informação obtida pelos sensores permite que o algoritmo escolha o mapa mais provável a cada momento e com isso a sua posição instantânea. Essa solução foi implementada e testada em ambientes virtuais e reais, fazendo uso de sensores esparsos para a coleta de informação. Os resultados mostram que é possível nessas condições conseguir executar a navegação em tempo real com uma melhora substancial nos resultados tanto de localização como de mapeamento.

Palavras-Chave: SLAM, DP-SLAM, Robô Móvel, Navegação Autônoma.

1 – Introdução

A localização e o mapeamento são dois componentes fundamentais para o funcionamento de qualquer sistema autônomo de navegação, permitindo que o veículo seja capaz de obter informações do ambiente ao seu redor e saber onde se encontra dentro desse mesmo ambiente e com isso poder cumprir a sua missão. A solução para cada um desses problemas é relativamente simples como já foi mostrado em vários estudos que abordam a solução para o problema da localização dado o mapa (Fox et al., 1999) e vice-versa (Elfes, 1989). Contudo, a grande maioria das aplicações relacionadas à navegação autônoma implica no desconhecimento de ambos, tanto a localização precisa do robô quanto um mapa acurado do ambiente através do qual ele irá se movimentar.

Um robô autônomo deve ser capaz de navegar através de um ambiente inicialmente desconhecido, mapeando seu conteúdo baseado somente nas informações coletadas pelos seus sensores e ao mesmo tempo se localizar dentro desse mesmo mapa, sem o auxílio de nenhum equipamento externo. Para fazer isso, o sistema precisa lidar com ambos os problemas de localização e mapeamento, resolvendo-os durante a navegação. É fácil perceber que ambos os problemas estão de alguma forma conectados, pois sem uma localização precisa o mapa resultante ficará comprometido, o que dificultará ainda mais uma localização posterior. Assim, pequenos erros gerados a cada etapa rapidamente se acumulam e afastam os resultados cada vez mais da realidade (Fig. 1).

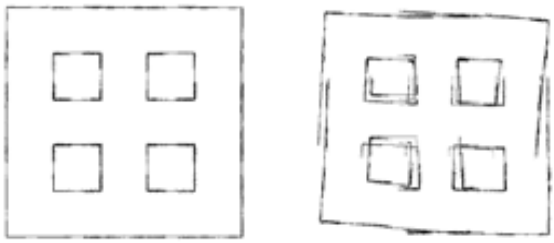


Fig 1. Mapeamento sem SLAM.
(a) Ambiente percorrido. (b) Resultado do mapeamento.

Esse é o problema do SLAM, cujo objetivo é lidar com ambos os problemas ao mesmo tempo, usando os resultados de um para melhorar a precisão do outro. Como não é possível remover completamente os erros inerentes aos sensores usados para coletar informação e nem incorporá-los aos modelos construídos para predição, pois eles são imprevisíveis, soluções para o problema do SLAM usualmente recorrem a abordagens probabilísticas,

tratando os erros na tentativa de sistematicamente removê-los antes que eles possam acumular.

As primeiras abordagens para se resolver o problema do SLAM surgiram no final da década de 80 (Smith, Self e Cheeseman, 1988), com a utilização de Filtros de Kalman e *landmarks* como uma tentativa de se representar as incertezas geradas durante a navegação. Ao se assumir uma distribuição gaussiana de erros, uma grande simplificação dos cálculos se tornava possível (Welch e Bishop, 2001). Desde então, essa tem sido a principal abordagem para o problema do SLAM, que vem se desenvolvendo bastante na última década, como é o caso do FastSLAM (Montemerlo e Thrun, 2002).

No entanto, não é sempre que os erros do sistema podem ser simplificados como distribuições gaussianas, permitindo a utilização de Filtros de Kalman. Nesses casos, a abordagem probabilística predominante é a de Filtros de Partículas, introduzido na comunidade científica por Gordon, Salmond e Smith, em 1993. Um Filtro de Partículas funciona através de amostragem, mantendo um conjunto de partículas que procuram simular a densidade de probabilidade de localização do sistema. Essa abordagem já foi testada com grande sucesso em ambientes internos, mas devido ao seu alto custo computacional encontra dificuldades na aplicação em tempo real.

A solução conhecida como DP-SLAM, introduzida inicialmente por Eliazar e Parr em 2003, lida justamente com essa limitação, procurando evitar o alto custo computacional e com isso conseguir desempenho adequado em tempo real. Através da utilização de estruturas de manutenção de dados eficientes, o DP-SLAM consegue manter partículas que representam não apenas prováveis posições do Robô, mas também prováveis mapas do ambiente. Ao evitar a utilização de *landmarks*, o DP-SLAM evita também as rotinas de detecção e reconhecimento necessárias para evitar problemas como o da ambigüidade (Montemerlo e Thrun, 2003). A amostragem de partículas é realizada com base apenas nos dados obtidos diretamente dos sensores, sem a necessidade de tratamento prévio.

Até esse momento, implementações do DP-SLAM tem se utilizado de sensores de coleta de dados densos, como laser scanner (Eliazar e Parr, 2003-4-5), para obter informações do ambiente. Aqui é apresentada uma implementação do DP-SLAM que se baseia apenas em sensores esparsos, do tipo anel de sonares, procurando com isso conseguir resultados satisfatórios durante uma navegação em tempo real. As próximas sessões desse artigo detalham os principais conceitos e ferramentas do DP-SLAM, juntamente com detalhes para a implementação dessa solução sob as condições indicadas acima. Para concluir, ao final são mostrados e discutidos os resultados de uma implementação de DP-SLAM tanto em ambiente virtual como real.

2 – Filtros de Partículas em SLAM

Os Filtros de Partículas têm como principal função acompanhar uma variável conforme ela evolui no tempo, seguindo uma função que por algum motivo não pode ser precisamente modelada. Eventos acontecem para modificar o estado da variável, e observações são realizadas em determinados períodos de tempo que permitem a realização de correções relativas ao estado da variável. Não é necessária nenhuma restrição em relação à natureza da função de probabilidade que será acompanhada, permitindo aplicações muito mais genéricas, como é o caso da "localização global" (Borenstein, Everett e Feng, 1996) e "seqüestro do robô" (Engelson e McDermott, 1992).

Um Filtro de Partículas funciona recursivamente, tirando conclusões sobre o estado da variável desejada conforme observações relativas ao ambiente são obtidas a partir dos sensores (Thrun, 2002). Durante a inicialização, um conjunto $S^0 = \{s_1^0, \dots, s_N^0\}$ de N amostras é criado, baseado na função de probabilidade $p(x^0)$ relativa à posição real x^0 do robô. O problema se torna então encontrar $p(x^t|Z^t, U^t)$, que é a probabilidade do robô estar em um determinado instante t em uma posição x^t dadas as informações coletadas dos sensores $Z^t = \{z^0, \dots, z^t\}$ e os controles $U^t = \{u^0, \dots, u^t\}$ armazenados desde o início da navegação.

Em um instante t > 0, um vetor $u^t = \{u_1^t, \dots, u_A^t\}$ de controles e um vetor $z^t = \{z_1^t, \dots, z_B^t\}$ de observações são recebidos e incorporados ao sistema. Uma nova partícula s_n^t é gerada para cada partícula $s_n^{t-1} \in S^{t-1}$, retirada de $p(x^t|u^t, s_n^{t-1})$, criando um novo conjunto $\bar{S}^t$. Após isso, N partículas são amostradas $\bar{S}^t$, de forma que $s_n^t \in \bar{S}^t$ são amostrados com uma probabilidade proporcional a $p(z^t|s_n^t)$, criando o conjunto S^t. Tomando $N \to \infty$ esse método recursivo leva ao conjunto S^t que converge uniformemente para a densidade de probabilidade desejada $p(x^t|Z^t, U^t)$.

Na navegação autônoma, geralmente a odometria é utilizada para fornecer uma estimativa da posição atual do robô baseando-se na rotação das rodas. Erros relacionados a esse movimento são modelados como uma translação linear, representando os erros sistemáticos e uma distribuição gaussiana com $\mu = 0$, representando o ruído branco. Esses valores são determinados empiricamente através de testes e podem ser constantes ou variar de acordo com diversos parâmetros como a magnitude do movimento.

O problema de localização, como já foi mostrado anteriormente, é simples de ser resolvido caso o mapa do ambiente já tenha sido fornecido. Isso porque dada uma posição é possível determinar qual seria a informação que seus sensores deveriam retornar, e com isso compará-los com a informação que é de fato recebida. Como foi notado por Murphy (1999), a informação relativa a cada sensor pode ser tratada independentemente, e assim a diferença $\delta_m^t = |d_m^t - d'^t_m|$ entre cada medida d_m^t e o valor estimado d'^t_m pode ser adicionado como uma forma de "pesar" das partículas, verificando sua semelhança com a realidade.

3 – *Distributed Particle* SLAM (DP-SLAM)

A maioria das abordagens para o problema do SLAM que se utiliza de Filtros de Partículas para acompanhar o estado do sistema mantém apenas um único mapa do ambiente, e distribuem as partículas sobre esse mapa representando a posição do robô. Mas essa abordagem não é ótima, pois gera erros que acumulam com o passar do tempo, já que o estado oculto do sistema

não é apenas a posição do sistema, mas também o próprio mapa. Ambigüidades geradas localmente são incorporadas ao mapa global e não poderão ser descartadas conforme o sistema coleta informação suficiente para resolvê-las.

Um dos aspectos principais do DP-SLAM é que cada partícula é composta não apenas por uma posição, mas também um mapa, criado através da acumulação de observações coletadas pelos sensores do robô. Assim sendo, as comparações feitas entre os dados coletados pelos sensores são realizadas não em relação a um único mapa supostamente correto, mas sim contra *P* mapas incompletos e possivelmente errados. Quando uma partícula é eleita a mais provável de ser a correta, tanto o problema da localização como do mapeamento são resolvidos simultaneamente, como é o objetivo do SLAM.

O problema então se torna o custo computacional que essa solução em princípio acarreta. Pois cada partícula agora representa além de uma posição também um mapa, que não é uma estrutura simples. Outras abordagens, como o FastSLAM (Montemerlo e Thrun, 2002) recorrem aqui a um conjunto de *landmarks* determinadas e unicamente reconhecidas para representar um mapa de forma paramétrica, aumentando o desempenho final do algoritmo. O DP-SLAM mostra que isso não é necessário e que através da utilização de estruturas eficientes de manutenção de dados é possível conseguir manipular partículas suficientes para se resolver o problema do SLAM apenas com a informação coletada diretamente dos sensores. Essas estruturas são a maior contribuição do DP-SLAM para a comunidade científica, e são detalhadas nas sessões a seguir, juntamente com detalhes de implementação.

3.1 – Árvore Ancestral

No DP-SLAM, quando uma particula é amostrada não apenas a sua posição tem que ser propagada para as próximas partículas, mas também o mapa associado a ela. Uma maneira direta de se fazer isso seria copiar todo o mapa sempre que uma partícula é amostrada, o que teria um custo computacional de *O(MP)*, com *M* sendo o tamanho do mapa e *P* o número de partículas existentes. Esse custo rapidamente se torna proibitivo conforme o tamanho do ambiente e a quantidade de partículas aumentam, sem mencionar o hardware necessário para se armazenar toda essa informação.

Dessa forma, é introduzido o conceito de *Ancestralidade* entre partículas, como uma tentativa de se formular uma solução para esse problema. Isso se deve ao fato de que a grande maioria dessa informação é redundante e com isso não seria necessário propagá-la. Quando uma partícula da iteração *n* gera uma partícula na iteração *n+1*, chamamos a partícula da geração *n* o pai e as partículas geradas por ela na geração *n+1* seus filhos (e também irmãos entre si). Supondo que o sensor do robô coleta informações referentes a *A* células a cada atualização, é fácil perceber que o mapa de um filho não pode ser diferente do mapa de seu pai em mais de *A* células simultaneamente.

Como *A* é geralmente muito menor do que *M* em aplicações relacionadas ao SLAM, uma melhor solução seria armazenar apenas as diferenças entre mapas, e dessa forma o problema de mapeamento se resolveria. Por outro lado, surgiria um problema de localização, que pede a reconstrução do mapa. Para isso ser feito, seria necessário coletar informações de todos os ancestrais da partícula, cuja quantidade cresce linearmente com as iterações do algoritmo. Assim, o custo computacional aumentaria conforme o robô navegasse pelo ambiente, eventualmente invalidando essa solução.

O desafio se torna então criar uma estrutura capaz de proporcionar uma cópia eficiente de mapas através das gerações ainda mantendo uma complexidade independente do número de iterações do algoritmo. O DP-SLAM consegue isso através da Árvore Ancestral, uma estrutura do tipo árvore onde cada nó representa uma partícula, com o nó raiz sendo a posição inicial do robô (a primeira geração) e suas folhas, as partículas da geração atual. Juntamente com o Mapa de Observações, a Árvore Ancestral garante uma solução robusta para o problema de rápido mapeamento e localização.

Durante a etapa de amostragem do Filtro de Partículas, *P* novas partículas são geradas pela propagação do modelo dinâmico do robô. Elas serão então os novos nós da Árvore Ancestral, e serão incluídos nela através dos vínculos de ancestralidade, onde um vetor de ponteiros para cada um deles é mantido, permitindo rápido acesso. Cada nó possui um ponteiro que o liga ao seu pai, e um vetor de ponteiros que o liga aos seus filhos, permitindo a navegação através da árvore, alcançando qualquer partícula através da sua ancestralidade.

A cada nó também é atribuído um ID único e as coordenadas que representam a sua posição dentro do ambiente, como $s_n^t = \{x, y, \theta\}$ (posição e orientação) para um robô que navega em um ambiente bidimensional. Por último, o robô possui também uma lista das células que foram atualizadas pelos seus sensores, que são utilizadas para acessar o mapa de Observações que contém o valor das observações. Essa lista será utilizada caso o nó tenha de ser apagado, permitindo que suas observações também sejam acessadas e apagadas.

O tamanho limitado da Árvore Ancestral é conseguido pela remoção de nós desnecessários ao final de cada iteração do algoritmo. Dessa forma consegue-se o que é chamado de Árvore Ancestral Mínima, cujo tamanho em seu estado permanente é finito e independente da quantidade de iterações (Eliazar e Parr, 2003). Uma Árvore Ancestral nesse estado possui três características facilmente demonstráveis e que garantem o seu tamanho finito:

- Possui exatamente *P* folhas.
- Tem um fator de divisão mínimo de 2.
- Profundidade máxima igual a *P*.

Esse estado pode ser alcançado pela remoção, ou fusão, de nós desnecessários, deletando a informação que não é mais importante ou reorganizando a informação de forma a melhorar a eficiência. Existem dois tipos de nós desnecessários:

- Aquele que não tem nenhum filho na geração mais recente. Esse nó pode ser removido, e caso isso faça com que o seu pai também fique sem filhos, o processo se repete iterativamente.

- Aquele que apenas possui um filho na geração mais recente. A informação desse nó apenas é útil para o seu único filho, e por isso suas observações podem ser unidas em um único nó. Esse nó possuirá então o ID do filho e o seu pai será o pai do nó que foi deletado (pulando com isso uma geração).

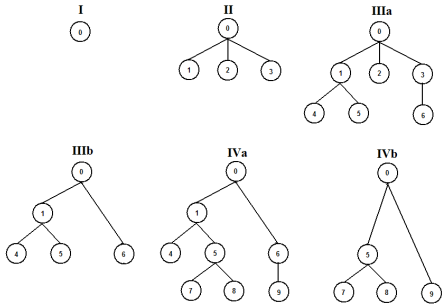


Fig 2. Exemplo de Árvore Ancestral (P = 3).

Uma maneira de implementar eficientemente essas duas rotinas é manter um vetor de ponteiros que levam não apenas às partículas da geração atual, mas também às partículas da geração imediatamente anterior. Assim, basta percorrer esse vetor para encontrar os nós que cumprem os requisitos necessários para serem removidos. Contudo, sempre que um nó é removido, seu pai também deverá ser verificado, pois ele pode ter se tornado desnecessário também.

3.2 – Mapa de Observações

O *Mapa de Observações* complementa a *Árvore Ancestral* por ser o local onde as informações obtidas através dos sensores do robô são armazenadas, criando múltiplos mapas, cada um relacionado a uma posição. É possível reconstruir o mapa referente a qualquer partícula que esteja na Árvore Ancestral, e principalmente, é possível reconstruir mapas parcialmente, recuperando apenas informação relevante. Dessa forma a reconstrução se torna independente da variável *M*, que tende assumir valores elevados no SLAM.

Composto por uma única grade de ocupação, cada célula do Mapa de Observações é responsável por guardar a informação referente a uma pequena área discretizada do mapa. Essa informação é armazenada através de uma árvore balanceada, similar àquela utilizada pela Árvore Ancestral, mas ao invés de partículas os nós armazenam as observações realizadas pelas partículas.

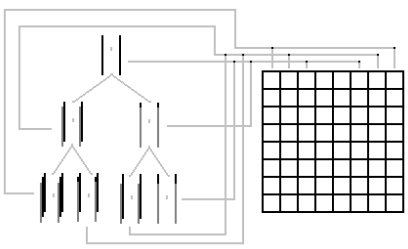


Fig 3. Esquema de um Mapa de Observações.

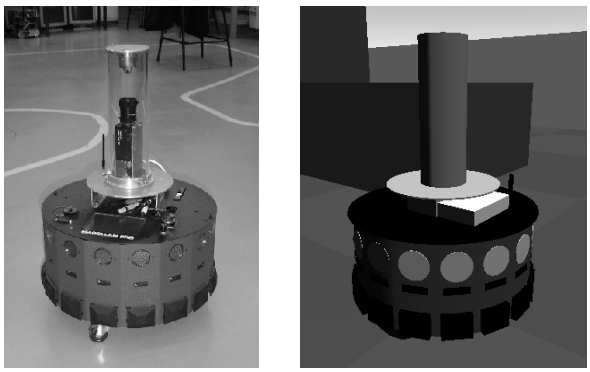
No início, o Mapa de Observações é criado como uma matriz de árvores vazias, com a raiz representando o desconhecimento em relação àquela região. Quando uma observação é feita em uma célula específica, a sua árvore é acessada e essa informação é armazenada com o ID da partícula que a gerou, para acesso posterior. O conceito de Ancestralidade também é utilizado, com cada observação mantendo um vínculo com seu pai e filhos para permitir a navegação através dela.

Para reconstruir um mapa, inicialmente é gerada uma lista dos ancestrais referentes àquela partícula, seguindo até a raiz. Essa lista é utilizada para acessar cada uma das células do Mapa de Observação que serão reconstruídas, procurando seqüencialmente pelos seus ancestrais, começando pelos mais antigos e seguindo até a partícula em si. Caso a partícula não seja encontrada, isso significa que a partícula não coletou nenhuma informação relevante a essa célula, então é utilizada a observação mais recente realizada por um de seus ancestrais.

É importante notar que as mesmas rotinas que foram criadas para manter a Árvore Ancestral em seu estado mínimo também devem ser utilizadas no Mapa de Observações, para manter o seu tamanho finito a despeito da quantidade de iterações realizadas e também para garantir a equivalência de ancestrais. É por isso que cada nó da Árvore Ancestral armazena uma lista das coordenadas que atualizou, para o caso em que ele seja deletado ou unido a outro nó apenas as células relevantes precisem ser acessadas.

4 – Implementação e Resultados

Para a validação do DP-SLAM com sensores esparsos, como proposto, o algoritmo aqui descrito foi implementado e testado tanto em ambientes virtuais como em uma plataforma robótica real. Para isso foi utilizado um *MagellanPro*, robô não holonômico com acionamento diferencial, equipado comvisao omidirecional, 16 sensores de sonar e de infravermelho e bumpers. Os testes virtuais foram realizados em ambiente Linux com o auxílio do software *Gazebo*, que permite a simulação dinâmica tridimensional de robôs. Os principais parâmetros geométricos e inerciais do robô foram transferidos para o seu modelo, assim como os seus sensores e sistema de odometria. A Figura 4 mostra o robô *MagellanPro* e o seu modelo virtual.



(a) (b)

Fig 4. Plataformas de teste.
(a) MagellanPro. (b) Modelo virtual.

A interface de comunicação entre o algoritmo e os sensores e atuadores do robô foi feita através do *Player*, software que permite acesso tanto a plataformas reais quanto virtuais sem a necessidade de qualquer alteração do código. A coleta de dados baseou-se apenas no anel de sonares (16 sensores igualmente espaçados em um único plano), obtendo-se com isso uma quantidade de informação muito inferior àquela alcançada por um sensor laser, com medidas de distância a cada 0.5º numa faixa de 180º. A diminuição na quantidade de informação incorporada a cada atualização do sistema contribuiu também para diminuir o custo computacional final do algoritmo.

Os algoritmos desenvolvidos aqui foram implementados em linguagem C++ e testados em um Pentium IV de 3.20 GHz. Inicialmente rotinas simples de navegação foram criadas, permitindo ao robô se deslocar através de uma lista de pontos pré-determinados, utilizando a odometria para se localizar e os sonares para mapear o ambiente. Primeiramente, o sistema foi testado sem erros para a verificação de que os modelos criados comportavam-se conforme o esperado, (Fig. 5b).

A seguir, os modelos de erro foram determinados empiricamente através de testes de campo com o *MagellanPro* (Thrun, 2001) e inseridos dentro do modelo virtual. Os resultados obtidos no simulador começaram então a refletir a necessidade de um algoritmo de SLAM para compensar os erros acumulados conforme o robô navega. As Figuras 5c e 5d mostram respectivamente o mapeamento e a localização considerando os erros de odometria e sonares. Nesses experimentos foi utilizada uma freqüência de atualização na odometria e nos sonares de 10 Hz, e o processamento de informação foi realizado em tempo real, simultaneamente à sua obtenção, com os resultados sendo impressos ao final da navegação.

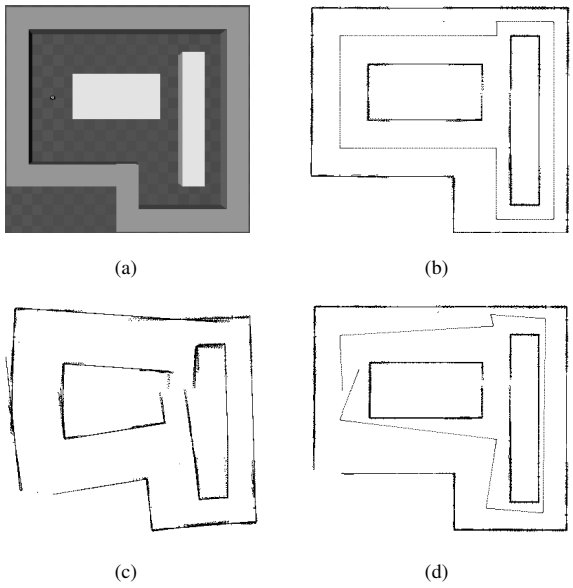


Fig 5. Resultados da simulação virtual (65 m de navegação)
(a) Ambiente virtual. (b) Localização e Mapeamento sem erros.
(c) Mapeamento com erros. (d) Localização com erros.

O algoritmo de DP-SLAM foi então inserido no sistema de navegação, juntamente com os erros necessários para se verificar o comportamento do algoritmo DP-SLAM. O principal objetivo desses experimentos foi a obtenção de resultados em tempo real. Para conseguir isso mantendo a mesma freqüência de atualização (10 Hz), a quantidade de partículas utilizada teve que ser diminuída bastante, até aproximadamente 100 partículas simultaneamente. Os resultados obtidos são mostrados na Figura 6.

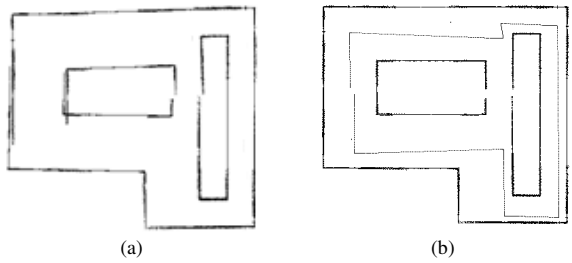


Fig 6. Resultados do DP-SLAM em ambiente virtual.
(a) Mapeamento. (b) Localização.

Como pode ser visto nessa figura, ambos os resultados foram substancialmente melhorados com a utilização do DP-SLAM. A pequena quantidade de partículas não afetou os resultados como era esperado, com resultados satisfatórios sendo obtidos em testes com até 75 partículas. Ao invés disso, foi afetada a capacidade do algoritmo de lidar com situações ambíguas, por não ser capaz de amostrar uma quantidade suficiente de prováveis posições. Assim sendo, a qualidade dos resultados de localização variou bastante dependendo da escolha de partículas relevantes ou não realizada pelo algoritmo de acordo com o seu modelo. Um aumento na quantidade de partículas em princípio resolveria esse problema, proporcionando uma melhor representação e tratamento dos erros do sistema.

Foram então realizados experimentos usando o *MagellanPro* real. O robô foi programado para navegar ao redor de um obstáculo no centro, em linhas retas, e retornar até o ponto inicial, em uma trajetória de aproximadamente 10m (Fig. 7). Foram utilizados limitadores de velocidade (0.5m/s) e aceleração (0.05 m/s^2) para permitir uma coleta consistente dos dados e evitar movimentos bruscos. As Figuras 8a e 8b mostram os resultados obtidos sem a utilização do DP-SLAM tanto para mapeamento como para localização, respectivamente. Já as Figuras 8c e 8d mostram os resultados com o DP-SLAM.

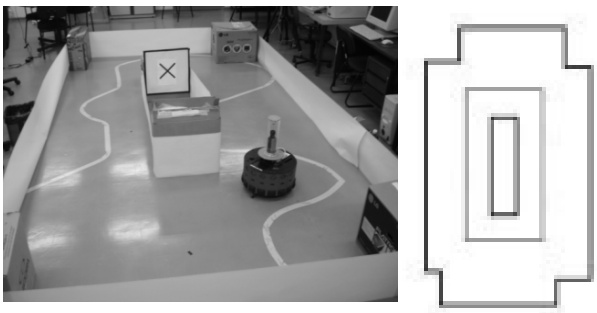


Fig 7. Pista de testes para o DP-SLAM.

Em testes reais foi possível alcançar navegação em tempo real com até 200 partículas, mantendo constantes os outros parâmetros. Isso se deve ao fato do simulador representar um custo computacional significativo que o robô real não possui. Novamente, pode-se perceber uma melhora significativa nos resultados, com o problema de "closed-loop" evidente anteriormente sendo resolvido. É mostrado com isso que que é possível implementar o DP-SLAM com sensores esparsos mantendo a localização do robô dentro de limites aceitáveis ao longo de sua trajetória e simultaneamente mapeando o ambiente. A degradação da qualidade do mapeamento se dá principalmente devido às características pontuais do sonar, que dificulta a determinação dos contornos dos obstáculos.

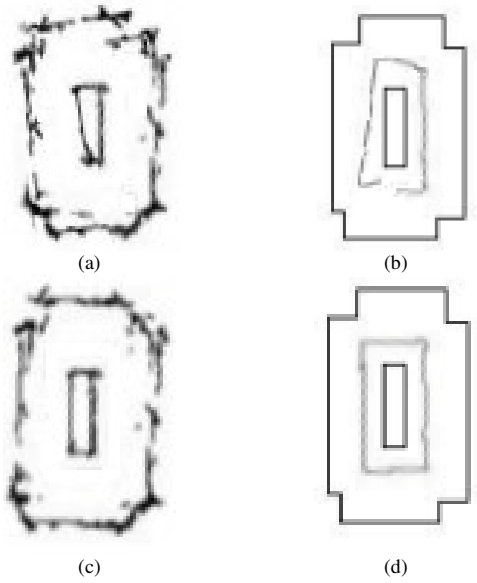


Fig 8. Resultados de DP-SLAM em ambientes reais.
(a) Mapeamento sem SLAM. (b) Localização sem SLAM.
(c) Mapeamento com SLAM. (d) Localização com SLAM.

5 – Conclusão

Esse artigo discutiu uma solução para o problema da localização e mapeamento simultâneos, que é de grande importância para o campo da navegação autônoma. A solução apresentada aqui, o DP-SLAM, é capaz de manter múltiplos mapas do ambiente, cada um correspondendo a uma possível posição do robô, e escolher qual deles é mais provável de ser o correto. Sensores esparsos de informação foram utilizados para realizar as observações do ambiente, uma abordagem diferente daquela que geralmente é utilizada em soluções para o problema do SLAM (Eliazar e Parr, 2003).

Para validar a solução, o algoritmo apresentado aqui foi implementado e testado em ambientes virtuais e reais. Os resultados obtidos mostram que o DP-SLAM promove uma melhoria significativa na localização e mapeamento mesmo com a utilização de sensores esparsos como fonte de informação. Foi possível alcançar navegação em tempo real, embora a quantidade de partículas necessárias para isso tenha sido demasiadamente pequena (em torno de 100), dificultando a representação estatística dos erros do sistema. Um aumento na eficiência do algoritmo ou na capacidade de processamento permitiria a utilização de mais partículas, e com isso uma melhoria ainda maior nos resultados alcançados.

Agradecimentos

Gostaríamos de agradecer aos integrantes do LPA pelo apoio e companheirismo durante o desenvolvimento desse trabalho, e também à FAPESP pelo auxílio financeiro (2006/00968-1) e infra-estrutura.

Referências Bibliográficas

Borenstein, J., Everett, B., Feng, L. **Navigating Mobile Robots: Systems and Techniques.** A. K. Peters, 1996.

Elfes, A. **Occupancy grids: A probabilistic framework for robot perception and navigation,** Ph. D. Dept. Elect. Eng. Carnegie Mellon University, 1989.

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.

Engelson, S., McDermott, D. **Error correction in mobile robot map learning.** In ICRA. 1992.

Fox, D., Burgard, W., Dellaert, F., Thrun, S. **Monte Carlo localization: Efficient position estimation for mobile robots**. In *AAAI-99*, 1999.

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.

Murphy, K. **Bayesian map learning in dynamic environments.** Advances in Neural Information Processing Systems 11. MIT Press, 1999.

Smith, R., Self, M., Cheeseman, P. **Estimating uncertain spatial relationships in robotics**, in Uncertainty in Artificial Intelligence, New York: Elsevier Science, 1988, vol. 2, pp. 435–461

Thrun, S. **Learning occupancy grid maps with forward sensor models.** School of Computer Science, Carnegie Mellon University, Pittsburgh. 2001.

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

Welch, G., Bishop, G. **An introduction to the Kalman Filter.** University of North Caroline, Department of Computer Science. 2001.