

**IMPLEMENTAÇÃO PARALELA DE
ALGORITMOS PARA LOCALIZAÇÃO E
MAPEAMENTO SIMULTÂNEOS COM UMA
ÚNICA CÂMERA.**

ANDRÉ LIMA GASPAR RUAS

IMPLEMENTAÇÃO PARALELA DE
ALGORITMOS PARA LOCALIZAÇÃO E
MAPEAMENTO SIMULTÂNEOS COM UMA
ÚNICA CÂMERA.

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Ciências Exatas da Universidade Federal de Minas Gerais como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

ORIENTADOR: MARIO FERNANDO MONTENEGRO CAMPOS

Belo Horizonte

Maior de 2011

© 2011, André Lima Gaspar Ruas.
Todos os direitos reservados.

R894i Gaspar Ruas, André Lima
Implementação paralela de algoritmos para
localização e mapeamento simultâneos com uma única
câmera. / André Lima Gaspar Ruas. — Belo Horizonte,
2011

xvi, 106 f. : il. ; 29cm

Dissertação (mestrado) — Universidade Federal de
Minas Gerais

Orientador: Mario Fernando Montenegro Campos

1. Computação - Teses. 2. Visão por computador –
Teses. I. Orientador. II. Título.

CDU 519.6*82.10(043)



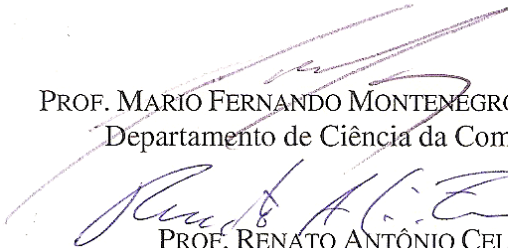
UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO


FOLHA DE APROVAÇÃO

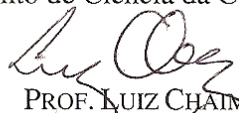
Implementação paralela de algoritmos para localização e mapeamento
simultâneos com uma única câmera


ANDRÉ LIMA GASPAR RUAS

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores:


PROF. MARIO FERNANDO MONTENEGRO CAMPOS - Orientador
Departamento de Ciência da Computação - UFMG


PROF. RENATO ANTÔNIO CELSO FERREIRA
Departamento de Ciência da Computação - UFMG


PROF. LUIZ CHAIMOWICZ
Departamento de Ciência da Computação - UFMG


PROF. BRUNO OTÁVIO SOARES TEIXEIRA
Departamento de Engenharia Eletrônica - UFMG

Belo Horizonte, 12 de julho de 2011.

Resumo

Esse trabalho aborda o problema de localização e mapeamento simultâneos (SLAM) para o caso particular onde se utiliza uma única câmera como sensor (SLAM monocular). O objetivo deste trabalho é o estudo de soluções que sejam capazes de resolver o problema de maneira eficiente, explorando o uso do paralelismo oferecido pelas unidades de processamento gráfico (GPU). Um novo algoritmo paralelo para estimação da pose do robô baseado em filtro de partículas utilizando a fatoração de Rao-Blackwell é proposto e implementado. Assim como no FastSLAM, um mapa é estimado para cada partícula e a distribuição proposta para a nova pose fornece a informação relativa à última observação. Porém, diferentemente dessa técnica, o nosso algoritmo incorpora essa informação usando um segundo filtro de partículas, chamadas subpartículas, para cada partícula. Isso possibilita a adaptação do algoritmo para que possa ser aplicado a um grande número de modelos de locomoção. Foram implementadas também versões paralelas do algoritmo proposto utilizando o SICK laser como sensor e do algoritmo de localização de Monte Carlo. A primeira obteve ganhos em tempo de execução de cerca de 500 vezes quando comparada à versão sequencial enquanto a segunda mostrou ser capaz de localizar o robô em tempo real utilizando quantidades de partículas superiores a 100.000 e ganhos de desempenho superiores a 100 vezes em relação à CPU.

Abstract

This work is on the Simultaneous Localization and Mapping (SLAM) problem for the special case where a single camera is used, called Monocular SLAM. The main goal of this work is the investigation of solutions that would be capable to efficiently tackle the Monocular SLAM problem, exploring parallelism and its implementation on Graphics Processing Units (GPU). A new parallel algorithm for pose estimation based on particle filters and the Rao-Blackwell factorization is proposed and implemented. As in FastSLAM, a new map is estimated for each particle and the proposed distribution for the next pose provides information derived from the last observation. However, unlike FastSLAM, our algorithm incorporates this information using a second sub-particle filter for each particle. This enables the adaptation of the algorithm to a wide range of locomotion models. Parallel versions of the proposed algorithm using a SICK laser and of the Monte Carlo algorithm have also been implemented. In the former Speedups of about 500 times were obtained in comparison with the sequential version, and the later has shown to be able to localize the mobile robot in real time using over 100,000 particles with measured performance over 100 times when compared to the sequential implementation of the method.

Lista de Figuras

1.1	(a) Robô em ambiente subaquático, (b) Sojourner Rover da missão <i>mars pathfinder</i>	3
1.2	Evolução da capacidade de processamento de CPUs e GPUs.	5
3.1	Representação do problema SLAM como uma rede de Bayes.	32
3.2	Diagrama de um robô diferencial em um ambiente plano. Os valores r_x e r_y representam a posição do robô em relação ao referencial do mundo, r_φ a orientação do robô em relação ao eixo x_w , e L_r a distância entre as rodas do robô.	35
4.1	Organização da execução das <i>threads</i> em GPU [CUDA, 2010].	42
4.2	Distribuição da carga em GPUs com números diferentes de multiprocessadores.	43
4.3	Exemplos de padrões de acesso a memória compartilhada para 4 <i>threads</i> e 4 bancos de memória	45
4.4	Operação do tipo <i>fork/join</i> , código sequencial é executado em CPU enquanto código paralelo é executado em GPU [CUDA, 2010].	46
4.5	Exemplo de declaração e invocação de um <i>kernel</i>	46
5.1	Ciclo de execução do algoritmo desenvolvido.	48
5.2	Modelo de observação e intervalo de confiança.	50
5.3	Geração e atualização das subpartículas.	58
5.4	Restrição geométrica entre os pontos observados.	61
5.5	Estrutura do método de otimização.	69
5.6	Algoritmo paralelo.	70
6.1	Mapa <i>simple</i> [Gerkey et al., 2003] utilizado nos testes do MCL paralelo.	76
6.2	Mapa <i>ICEx</i> utilizado nos testes do MCL paralelo.	76

6.3	Distribuição das partículas na localização simples de Monte Carlo implementada em GPU	78
6.4	Tempo por iteração das etapas de predição e atualização das versões paralela e sequencial do MCL.	79
6.5	Tempos por Iteração \times Número de Partículas.	82
6.6	Tempos por Iteração \times Número de subpartículas.	83
6.7	Tempos por Iteração \times Resolução dos Mapas.	83
6.8	Reconstrução do mapa <i>Simple</i> com 200 partículas, 192 subpartículas e ruído de locomoção de 1%	85
6.9	Reconstrução do mapa <i>Simple</i> com 200 partículas, 192 subpartículas e ruído de locomoção de 5%	85
6.10	Robô utilizado para coleta de dados [Ceriani et al., 2009]	86
6.11	Planta baixa do ambiente experimental na Universidade de Milão-Bicocca [Ceriani et al., 2009]	87
6.12	Caminhos estimados com base na hodometria adicionada à ruídos normais com desvios padrão de 5% e 10% da distância percorrida a cada instante	88
6.13	Caminhos estimados pelo filtro usando uma partícula e 384 subpartículas e ruído de 5%	88
6.14	Caminhos estimados pelo filtro usando uma partícula e 384 subpartículas e ruído de 10%	89
6.15	Caminhos estimados pelo filtro usando uma partícula e 192 subpartículas e ruído de 5%	89
6.16	Caminhos estimados pelo filtro usando uma partícula e 192 subpartículas e ruído de 10%	90
6.17	Tempo de cálculo dos custos de correspondência \times número de marcos	91
6.18	Tempo de cálculo dos custos de correspondência \times número de observações	91
6.19	Tempo de cálculo dos custos de correspondência \times número de partículas	92
6.20	Tempo de cálculo dos custos de correspondência \times número de subpartículas	92
6.21	Tempo de atualização dos marcos \times número de observações	93
6.22	Tempo de atualização dos marcos \times número de partículas	93
6.23	Tempo para propagação das subpartículas \times número de partículas	94
6.24	Tempo de atualização das subpartículas \times número de subpartículas	94
6.25	Tempo de atualização das subpartículas \times número de partículas	95
6.26	Tempo de atualização das subpartículas \times número de observações	95

Lista de Tabelas

6.1	Tempos de execução em milissegundos \times número de partículas para o mapa <i>ICEx</i>	77
6.2	Tempos de execução em milissegundos vs. número de partículas para o mapa <i>simple</i>	77
6.3	Tempos de execução para CPU em milissegundos vs. número de partículas para o mapa <i>ICEx</i>	79
6.4	<i>speedup</i> obtido pela versão paralela para o mapa <i>ICEx</i>	80
6.5	Tempos de convergência em milissegundos e erro médio em metros vs. número de partículas para o mapa <i>ICEx</i>	80
6.6	Tempos de convergência em milissegundos e desvio (erro) padrão médio em metros vs. número de partículas para o mapa <i>simple</i>	80
6.7	Tempos de execução da implementação paralela (GPU) e sequencial (CPU) da versão laser e <i>speedup</i> obtido vs número de partículas.	82
6.8	Tempos de execução da implementação paralela (GPU) e sequencial (CPU) da versão laser e <i>speedup</i> obtido vs número de subpartículas.	84
6.9	Tempos de execução da implementação paralela (GPU) e sequencial (CPU) da versão laser e <i>speedup</i> obtido vs resolução dos mapas.	84

Sumário

Resumo	vii
Abstract	ix
Lista de Figuras	xi
Lista de Tabelas	xiii
Lista de Símbolos	2
1 Introdução	3
1.1 Processamento em Unidades Gráficas	5
1.2 Objetivos	6
1.3 Organização do Texto	7
2 Trabalhos Relacionados	9
2.1 SLAM Visual	11
2.2 Detectores de Pontos Salientes	14
2.3 Desenvolvimento em Unidades Gráficas	16
3 Detalhamento do Problema	19
3.1 O problema SLAM	19
3.2 Tratamento probabilístico	20
3.3 Filtro de Bayes	22
3.4 Filtro de Kalman	24
3.4.1 EKF	25
3.4.2 UKF	26
3.5 Filtro de Partículas	29
3.6 Decomposição de Rao-Blackwell	30
3.6.1 Demonstração da fatoração	32

3.7	Modelos	34
3.7.1	Pose e Locomoção	35
3.7.2	Mapa e Observação	36
4	Arquitetura de GPUs	41
4.1	Organização	41
4.2	Modelo de desenvolvimento	44
5	Metodologia	47
5.1	Atualização dos Mapas	49
5.1.1	Inicialização Atrasada	50
5.1.2	Atualização dos marcos inicializados	53
5.1.3	Atualização dos marcos não inicializados	54
5.2	Predição da Pose	57
5.2.1	FastSLAM 2.0	59
5.2.2	Filtro de sub-partículas	60
5.3	Atualização das Partículas	63
5.4	Associação de dados	65
5.4.1	Cálculo dos custos	66
5.4.2	Otimização	68
5.5	Paralelização	70
6	Experimentos	75
6.1	MCL paralelo	76
6.2	SLAM laser paralelo	81
6.3	SLAM monocular paralelo	84
7	Conclusões e Trabalhos futuros	97
	Referências Bibliográficas	99

Lista de Símbolos

s_t	Pose do robô no instante t	4
Θ	Mapa do ambiente	4
a_t	Ação executada pelo robô entre o instante $t - 1$ e t	4
\mathbf{O}_t	Conjunto de observações obtidas pelo robô no instante t	4
θ_n	n -ésimo marco do mapa do ambiente	19
o_i	i -ésima observação obtida pelo robô referente a um único marco ...	19
ν_i	Função que associa a i -ésima observação obtida pelo robô ao marco correspondente	19
τ_i	Função que retorna o instante de obtenção da i -ésima observação .	19
s^t	Conjunto de todas as poses percorridas pelo robô até o instante t ..	20
a^t	Conjunto de todas as ações executadas pelo robô até o instante t ..	20
\mathbf{O}^t	Conjunto de todas as observações obtidas pelo robô até o instante t	20
X_t	Estado no instante t	20
$Bel_{t_1}(s_{t_2}, \Theta)$	Estado de crença a respeito da pose s_{t_2} e do mapa Θ dadas as informações disponíveis no instante t_1	21
$\overline{Bel}_t(s_t, \Theta)$	Estado de crença preditivo no instante t a respeito da pose s_t e do mapa Θ	22
$\mathcal{N}(\mu, \Sigma)$	Distribuição normal multidimensional de média μ e covariância Σ ..	24
$S_t^{(i)}$	Hipótese da i -ésima partícula sobre o caminho percorrido pelo robô até o instante t	48
$Bel_t^{(i)}(\Theta)$	Estado de crença no mapa associado à i -ésima partícula	49
$Bel_t^{(i)}(\theta_j)$	Estado de crença do j -ésimo marco do ambiente associado à i -ésima partícula	49
μ_{θ_i}	Valor esperado para a posição do i -ésimo marco	49
Σ_{θ_i}	Matriz de covariância da crença no i -ésimo marco	49
$s_t^{k,(i)}$	k -ésima sub-partícula gerada a partir da i -ésima partícula para o instante t	60
$w^{k,(i)}$	Peso atribuído à k -ésima sub-partícula gerada a partir da i -ésima partícula	60
$w^{(i)}$	Peso associado à i -ésima partícula	64
$c_{j,l}$	Custo de associar o marco l à observação j	66
d_j	Descritor da j -ésima observação	66

Capítulo 1

Introdução

Para que um robô móvel realize o planejamento de suas ações é necessário que conheça o ambiente, bem sua própria localização. Existem diversas situações como busca e resgate em caso de desastres ou exploração de ambientes extraterrestres ou subaquáticos, como os mostrados na Figura 1.1, nas quais tipicamente não se pode contar com um mapeamento prévio do ambiente. Denomina-se localização e mapeamento simultâneos ou *SLAM* (*simultaneous localization and mapping*) o problema de estimar simultaneamente um mapa de um ambiente desconhecido e a pose de um robô móvel que se movimenta nesse ambiente, a partir das ações do robô e das medidas dos sensores.

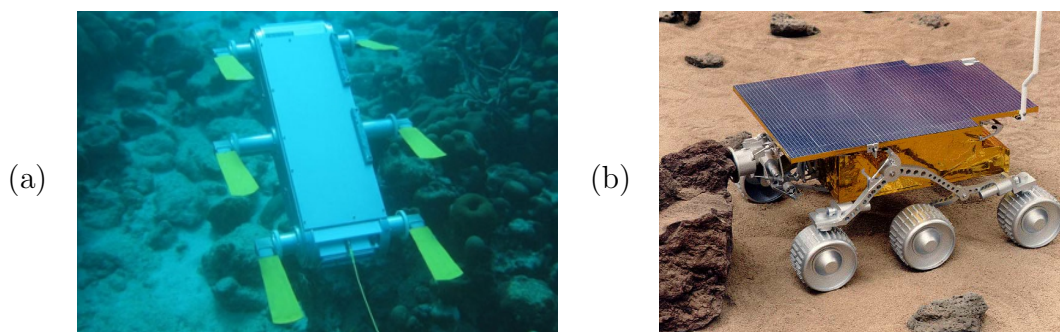


Figura 1.1. (a) Robô em ambiente subaquático, (b) Sojourner Rover da missão *mars pathfinder*.

Se a pose do robô for conhecida *a priori* tem-se um caso especial denominado problema de mapeamento¹. Nessa instância o objetivo é reconstruir um mapa do ambiente baseado nas informações dos sensores e no conhecimento da localização do robô.

¹A menos que seja especificado o contrário o problema de mapeamento com pose conhecida será tratado simplesmente como problema de mapeamento

Definição 1. *Seja um robô equipado com um conjunto de sensores cuja pose s_t no instante t é conhecida, em um ambiente cujo mapa Θ é desconhecido.*

O problema de mapeamento tem objetivo de, a partir dos conjuntos de ações $\{a_1, a_2, \dots, a_t\}$ realizadas, do caminho percorrido $\{s_1, s_2, \dots, s_t\}$ e dos conjuntos de observações $\{\mathbf{O}_1, \mathbf{O}_2, \dots, \mathbf{O}_t\}$ estimar o mapa Θ

Outra instância encontrada na prática consiste em estimar a localização de um robô em um ambiente conhecido. Em diversos ambientes: internos, subaquáticos, e.g, não é possível obter uma leitura de GPS. Medidas de Hodometria acumulam erros rapidamente. Essa instância recebe a denominação de problema de localização, e consiste em recuperar a localização do robô dado um mapa do ambiente e as informações sensoriais.

Definição 2. *Seja um robô equipado com um conjunto de sensores, cuja localização s_t é desconhecida, em um ambiente de mapa Θ conhecido.*

O problema de localização consiste em recuperar a posição s_t , por meio das medidas $\{\mathbf{O}_1, \dots, \mathbf{O}_t\}$ dos sensores do mapa Θ , dos conjuntos de ações $\{a_1, \dots, a_t\}$ executados pelo robô.

Existem inúmeras situações, porém, nas quais não é possível contar com conhecimento do ambiente *a priori* nem com uma estimativa confiável da pose de forma que é necessário que o sistema autônomo seja capaz de recuperar ambos simultaneamente. Esses fatos motivam o estudo do *SLAM*, cuja solução é essencial para a realização de diversas atividades em robótica móvel e, por isso, em anos recentes, esse problema tem sido objeto de diversos estudos pela comunidade científica.

Definição 3. *Seja um robô equipado com um conjunto de sensores, cuja localização s_t é desconhecida, em um ambiente cujo mapa Θ também é desconhecido.*

O problema de localização e mapeamento simultâneos (SLAM) consiste em recuperar simultaneamente a posição s_t e o mapa do ambiente Θ , por meio das observações $\mathbf{O}_1, \dots, \mathbf{O}_t$ dos sensores nos tempos $t = (1, \dots, t)$, e do conjunto de ações a_1, \dots, a_t executados pelo robô.

Apesar dos três problemas serem semelhantes em suas definições, enquanto os problemas de mapeamento e de localização são considerados fáceis de tratar e bem resolvidos na maior parte dos ambientes, o mesmo não ocorre com o *SLAM* devido principalmente à correlação entre a incerteza da pose e do mapa [Smith et al., 1990].

1.1 Processamento em Unidades Gráficas

Para que a pose e o mapa estimados possam ser utilizados na realização de alguma tarefa é necessário que eles possam ser obtidos em tempo real. O desenvolvimento das CPUs convencionais tem sido voltado para processamento sequencial e na redução do número de ciclos de *clock* por instrução. Entretanto, existe limite físico para o quão rápido uma instrução pode ser executada. Devido à dificuldade em reduzir o tempo de resposta das instruções recentemente observa-se uma tendência em aumentar a capacidade de processamento por meio do paralelismo.

O desenvolvimento nas unidades gráficas (GPUs) é focado especificamente em otimizar a renderização de imagens por meio de paralelismo. Essas unidades são projetadas para serem capazes de trabalhar com um grande volume de dados. GPUs atuais contém centenas de núcleos e são capazes de executar concorrentemente milhões de *threads*. O desempenho de pico (*peak-performance*) dessas unidades recentemente tem chegado a uma taxa de execução de pico da ordem de *TeraFLOPs* e ultrapassa em muito o processamento das CPUs convencionais e como pode se ver observado na Figura 1.2, que mostra a evolução da capacidade de processamento dos dois tipos de unidades [CUD, 2010].

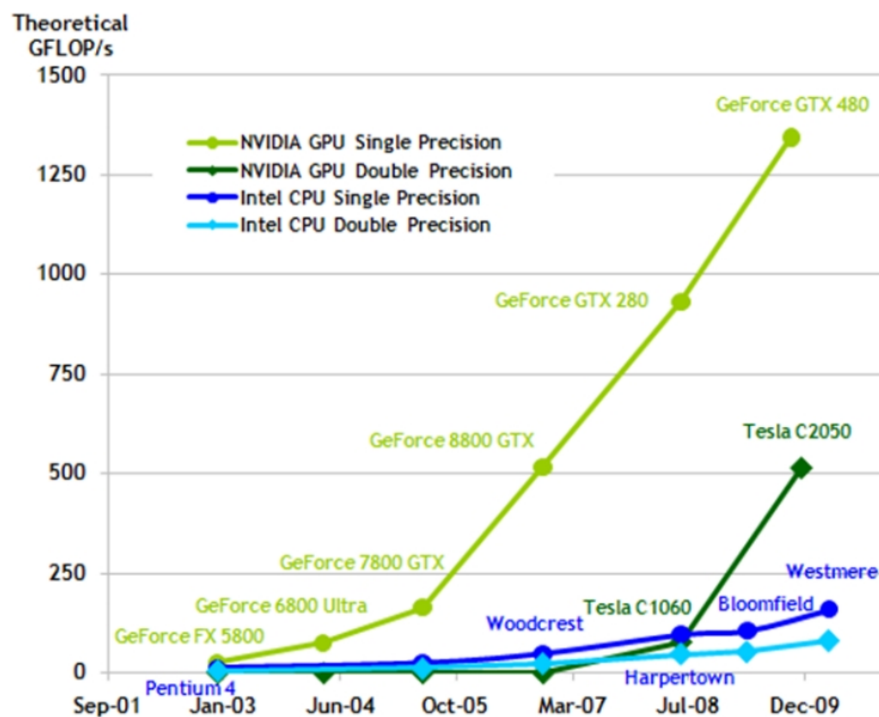


Figura 1.2. Evolução da capacidade de processamento de CPUs e GPUs.

Devido ao poder de processamento paralelo e ao custo acessível, recentemente

tem-se observado um crescimento na utilização dessas unidades para processamentos de propósito geral (*GP – General Purpose*) o que é conhecido, portanto, como *GPGPU Programming*. Originalmente, seu conjunto de instruções era muito limitado, entretanto, demandas crescentes de um espectro mais amplo de processamento pelas aplicações de entretenimento e multimídia, impulsionaram o surgimento de processadores gráficos com conjuntos de instruções cada vez mais flexíveis. GPUs atuais, desde a GeForce 8800 GTX da NVIDIA, são constituídas por uma matriz de processadores paralelos completamente programáveis [Fung and Mann, 2005].

A necessidade de soluções em tempo real, associada ao grande volume de dados envolvidos nos problemas tipicamente encontrados nas áreas de visão computacional e robótica, tem motivando a utilização de paralelismo. Vários algoritmos [Allusse et al., 2008, Fung and Mann, 2005] foram implementados com sucesso em GPUs obtendo ganhos de desempenho expressivos. Além disso, a tendência recente das CPUs se tornarem unidades de processamento paralelo demonstra a necessidade cada vez maior na utilização do paralelismo para obtenção de soluções mais eficientes.

O SLAM monocular pode se beneficiar do uso de paralelismo de diversas formas. GPUs já foram utilizadas para obter ganhos de desempenho na detecção de pontos salientes em imagens [Sinha et al., 2006, Cornelis and Van Gool, 2008], utilizada por diversas soluções do SLAM. Um grande número de soluções apresentadas na literatura fazem uso extensivo de operações matriciais como inversão e produto de matrizes que também podem ser aceleradas com uso de paralelismo. Filtros de partícula, também utilizados em soluções do *SLAM*, também já foram implementados em GPUs com *speedups* superiores a $30\times$ [Ruas et al., 2010].

Esses fatos, em conjunto com a necessidade de soluções eficientes para o problema motivam a utilização de GPUs na abordagem do SLAM.

1.2 Objetivos

O objetivo principal desse trabalho é o estudo de soluções eficientes para o *SLAM*, em especial o *SLAM* monocular, explorando principalmente o uso do paralelismo oferecido pelas unidades de processamento gráfico modernas. Espera-se que a solução possa ser utilizada em robôs reais sem que seja necessário utilizar muitos recursos de *CPU* permitindo que essa seja usada para outras tarefas.

A solução desenvolvida nesse trabalho se baseia em um filtro de partículas Rao-Blackwellizado (*Rao-Blackwellized particle filter*) e considera várias hipóteses de pose solucionando um problema de mapeamento simples para cada partícula de maneira

independente. Essa independência entre as hipóteses é um forte indicativo de que a técnica pode se beneficiar da paralelização.

1.3 Organização do Texto

O capítulo seguinte apresenta uma revisão de alguns trabalhos que se relacionam com problema ou com a solução proposta.

No terceiro capítulo é feito um detalhamento do problema SLAM, onde são apresentados de maneira resumida os fundamentos teóricos necessários para compreensão da solução proposta e são introduzidos o filtro de Bayes e a decomposição de Rao-Blackwell que constituem a base teórica para a solução desenvolvida.

O quarto capítulo introduz de forma resumida o modelo de desenvolvimento e a arquitetura CUDA, que foi utilizada na solução.

O capítulo cinco apresenta a solução desenvolvida, aborda cada etapa ou módulo do algoritmo e demonstra como a paralelização de cada uma das etapas foi realizada.

O sexto capítulo descreve os experimentos realizados para validação da solução, medidas de desempenho, apresenta os resultados obtidos e uma análise dos mesmos. Nesse capítulo também é apresentada uma versão paralela do algoritmo de localização de Monte Carlo [Ruas et al., 2010], que serviu tanto para validar o modelo de locomoção quanto para orientar o desenvolvimento da estratégia de paralelização da solução proposta ao SLAM.

O sétimo e último capítulo apresenta as conclusões obtidas do trabalho e propõe direções para desenvolvimentos futuros.

Capítulo 2

Trabalhos Relacionados

Smith and Cheeseman [1986] foram, possivelmente, os primeiros autores a apresentar o problema (SLAM). Seu trabalho tratou do problema de se estimar de maneira consistente a localização de um robô dado um modelo de mapa associado a incertezas que precisam ser reduzidas à medida que o robô navega o ambiente. Posteriormente Smith et al. [1990] introduziram a técnica de *Stochastic Mapping* (SM) que viria a se tornar a base teórica para diversas soluções propostas para o SLAM. Uma das grandes contribuições desse trabalho foi demonstrar que as incertezas na localização dos marcos no ambiente e na pose do robô são necessariamente correlacionadas e não podem ser tratados como se fossem independentes.

O SM assume que marcos estacionários podem ser consistentemente identificados e reobservados no ambiente ao longo do tempo, de forma que as incertezas na localização, tanto dos marcos quanto do robô, sejam reduzidas à medida que o ambiente for explorado. Essa técnica utiliza distribuições de probabilidades para representar as incertezas de maneira explícita. Para cada instante tem-se um estado de crença sobre a posição do robô e dos marcos representado por uma distribuição de probabilidades sobre todo o espaço de possibilidades. O SM não faz qualquer restrição ao tipo de sensor que deve ser utilizados, e ao longo dos anos um grande número de trabalhos com diferentes sensores como lasers [Ryde and Hu, 2007], sonares [Tardos et al., 2002] e câmeras [Davison et al., 2007, Davison, 2003, Se et al., 2002].

O tratamento probabilístico do SM assume que o processo apresenta a propriedade de Markov e se fundamenta no filtro de Bayes para solução do problema. Porém, na maioria dos casos, esse filtro é impossível de ser implementado de maneira exata, e por isso alguma solução aproximada deve ser utilizada. Um dos filtros mais populares é o filtro de Kalman estendido (EKF) [Holmes et al., 2009]. Esse filtro é uma extensão do o filtro de Kalman para modelos de propagação não lineares. O filtro de

Kalman original introduzido por Kalman [1960] assume que o estado observado possui distribuição normal e os modelos de propagação e de observação são lineares com ruído gaussiano. Nesse caso a distribuição obtida é exata pois transformações lineares preservam a forma normal da distribuição. O filtro estendido trabalha de maneira semelhante diferenciando apenas pelo fato de trabalhar com linearizações locais dos modelos de observação e propagação calculadas em torno das médias por meio do jacobiano.

Apesar de sua popularidade, em diversos casos o EKF gera resultados tendenciosos e superconfiantes, o que já foi reportado por diversos autores [Julier and K., 2004, van der Merwe and Wan, 2001, Lefebvre et al., 2004]. Outra variação muito utilizada é filtro de Kalman *Unscented* (UKF). Para estimar a distribuição de um estado \mathbf{x} n -dimensional, primeiramente são calculados ao redor da moda $2n + 1$ pontos amostrais do espaço de \mathbf{x} denominados pontos sigma (*sigma points*). A cada ponto é aplicada a função de propagação não linear e a partir dos pontos propagados reconstrói-se uma distribuição normal. Para atualizar a distribuição o funcionamento é semelhante. Os pontos sigma são calculados, a eles aplica-se o modelo de observação, e a partir da distribuição gerada computa-se o ganho de Kalman. O UKF tende a gerar resultados mais robustos do que o EKF, porém uma vantagem desse último é que para o caso de estimação de parâmetros o EKF pode ser otimizado para ter complexidade assintótica de $\mathbf{O}(n^2)$ no número de parâmetros, o que não é possível no UKF original.

Em um ambiente estático o SLAM pode ser tratado como um problema de estimação de parâmetros o que justificaria o uso do EKF. van der Merwe and Wan [2001] desenvolveram uma variação do UKF chamada *Square-Root unscented Kalman filter* (SRUKF). Em vez de propagar a matriz de covariância diretamente, o SRUKF propaga a sua raiz quadrada. Dessa forma consegue-se obter a mesma complexidade assintótica do EKF para a estimação de parâmetros.

Um dos problemas em utilizar variações do filtro de Kalman é que esses filtros devem manter a matriz de covariância completa do estado de crença. Em um ambiente com K marcos essa matriz conterá $\mathbf{O}(K^2)$ elementos. A cada observação toda a matriz de covariância deve ser atualizada o que faz com que esses filtros tenham complexidade limitada inferiormente por $\Omega(K^2)$. Percebendo que as incertezas dos marcos somente são correlacionadas por meio da incerteza do caminho, Murphy [1999] concluiu que o estado de crença podia ser fatorado em uma distribuição para o caminho multiplicada por uma distribuição independente para cada marco condicionada ao caminho. Partindo desse resultado Montemerlo et al. [2002] propuseram que o estado de crença fosse estimado sobre todo o caminho percorrido. Os autores, usaram uma técnica conhecida como Rao-Blackwellização [Doucet et al., 2000], a partir da qual desenvolveram um algoritmo baseado em filtro de partículas conhecido como FastSLAM. No FastSLAM o

estado de crença é estimado sobre todo o caminho percorrido a estimação dos marcos é marginalizada, sendo a crença de cada marco é estimada de maneira independente rastreado por um EKF. Os autores propuseram também uma estrutura de árvore para agrupar os marcos, de maneira a possibilitar a estimação da distribuição a *posteriori* com complexidade $\mathbf{O}(M \log K)$ onde M é o número de partículas e K o número de marcos observados. O filtro desenvolvido assume que a cada instante uma única observação é obtida, múltiplas observações podem ser processadas em série, em qualquer ordem, sem prejudicar a distribuição calculada.

Esse filtro porém sofre de um problema conhecido como empobrecimento da amostra caso a distribuição proposta difira muito da verossimilhança da observação. Nesse caso, como poucas partículas recebem um peso relevante, a maior parte é descartada o que reduz a capacidade do filtro em fechar laços (*loop closing*). Os mesmos autores desenvolveram uma nova versão do filtro e denominaram o novo filtro FastSLAM 2.0 [Montemerlo, 2003]. Nessa versão, informações acerca de cada observação são incorporadas à distribuição proposta por meio de EKF da amostragem das partículas. Isso leva a uma distribuição proposta mais próxima da função de verossimilhança da observação o que leva a um descarte menor de partículas e aumenta a capacidade do filtro realizar *loop-closing*. Porém isso faz com que a ordem em que as observações são incorporadas ao filtro passe a influenciar o resultado. Além disso, restringe-se o ruído do modelo de locomoção e a representação dos marcos à modelos normais, apesar do modelo normal se ajustar bem a grande parte das situações.

Filtros de partícula auxiliares [Pitt and Shephard, 1999] permitem a incorporação da última observação na distribuição proposta por meio do uso de um filtro de partículas para predição de cada partícula. Blanco et al. [2010] utilizou esse tipo de filtro para solucionar o problema SLAM utilizando laser, utilizando como modelo de mapa um grid de ocupação. A metodologia desenvolvida pelos autores se assemelha à utilizada nesse trabalho. Para cada partícula utiliza-se múltiplas partículas para gerar um preditiva da próxima pose, cada partícula é atualizada e uma ou mais partículas são sorteadas utilizando *rejection sampling*. Sua abordagem mostrou ser mais geral e robusta do que o FastSLAM, além de encontrar um modelo de mapa denso e contornar o problema da correspondência.

2.1 SLAM Visual

Grande parte das abordagens para o tratamento do SLAM existentes na literatura assumem um ambiente bidimensional devido, principalmente, aos tipos de sensores

utilizados. Sensores de profundidade, como laseres e sonares, são capazes de retornar a distância e orientação dos obstáculos no ambiente. Entretanto, esses sensores geralmente possuem região de varredura planar e, embora existam sensores laser capazes de realizar varreduras em três dimensões (3DLRF), seu custo é muito elevado para a maior parte das aplicações [Ryde and Hu, 2007]. Sensores bidimensionais foram adaptados a bases ou espelhos rotativos por Lingemann et al. [2004], Nuchter et al. [2007] e Ryde and Hu [2007] possibilitando um mapeamento completo do ambiente. Todavia, o tempo para a varredura completa é relativamente alto o que inviabiliza o uso desse tipo de estratégia quando se necessita locomover com maior velocidade.

Câmeras, por outro, lado podem ser usadas para sensoriar o ambiente em três dimensões, além de terem um custo acessível, e alta frequência de amostragem o que motiva sua utilização para tratar o SLAM. Câmeras, porém, estimam diretamente apenas a direção dos marcos no ambiente. Isso faz com que recebam a classificação de sensores *bearing only* (apenas orientação). A posição tridimensional dos marcos pode ser recuperada somente a partir da triangulação com observações obtidas dos mesmos marcos de diferentes pontos de vista [Bailey, 2003]. Uma forma popular de recuperar a informação tridimensional consiste em utilizar duas câmeras previamente calibradas, o que é conhecido como reconstrução estéreo [Forsyth and Ponce, 2002]. Essa estratégia já foi utilizada para solução do SLAM por autores como Se et al. [2002], Davison and Murray [2002], Jung and Lacroix [2003] e Hygounenc et al. [2004] em trabalhos anteriores. No entanto a reconstrução estéreo pode gerar resultados espúrios na presença de vibração e borramento devido ao movimento. Além disso, pode dificultar o uso dos recursos de *zoom* das câmeras.

Kim and Sukkarieh [2003] utilizaram marcos de dimensão conhecida para resolver o problema de estimação de profundidade. Conhecidas as dimensões dos marcos, a estimação da distância entre esses e a câmera torna-se simples. Entretanto, essa abordagem requer estruturação prévia do ambiente o que a torna limitada.

O problema de SLAM, quando tratado apenas com sensores *bearing only*, recebe a denominação de *bearing only* SLAM (BO-SLAM). Pelo fato de câmeras serem o sensor mais popular dessa classe as abordagens denominadas BO-SLAM e SLAM monocular se confundem.

A grande dificuldade nessa abordagem reside no fato de o SM assumir incerteza finita para cada marco. Contudo, as observações de sensores *bearing only* não trazem qualquer informação acerca da profundidade. Sendo assim a incerteza na direção do eixo da profundidade não pode ser considerada finita.

Leonard and Rikoski [2000], apesar de não terem trabalhado diretamente com BO-SLAM, desenvolveram uma técnica, chamada inicialização atrasada, que foi utilizada

por diversos autores para tratar o problema da incerteza na profundidade. Nessa técnica a inclusão dos marcos observados no vetor de estados é adiada até que outra observação do mesmo marco permita a recuperação da profundidade. Costa et al. [2004] propuseram uma abordagem para inicialização atrasada sem necessidade de realização prévia de correspondência. A cada momento todas as correspondências possíveis são verificadas e as mais consistentes são mantidas. No entanto, o grande número de correspondências possíveis reduz a escalabilidade desse método um número grande de marcos.

Outra abordagem para a inicialização atrasada é representar a crença dos marcos por meio de uma mistura de gaussianas gerando uma gaussiana para cada correspondência possível. Nesse sentido um dos trabalhos pioneiros foi o de Davison and Murray [2002], Davison [2003], Davison et al. [2007]. Uma característica importante desses trabalhos é a não necessidade de informação sobre o deslocamento da câmera (por odometria ou sensores inerciais). Além disso, esse trabalho demonstrou ser possível solucionar o BO-SLAM em tempo real utilizando o EKF, apesar de utilizar um número relativamente pequeno de marcos. No entanto, os critérios utilizados para aceitação das correspondências não são descritos.

Jensfelt and Folkesson [2006] apresentaram um método cuja estratégia é permitir que a estimação atrase N quadros, e utilizar esses N quadros para determinar a profundidade dos marcos. Esse atraso também é utilizado para determinar quais pontos constituem bons marcos. Seu trabalho tem como focos principais a gerência de marcos para obter desempenho em tempo real e aumento da qualidade da correspondência dos pontos.

Sola et al. [2005] propuseram uma abordagem para o SLAM monocular que utiliza uma soma geométrica de gaussianas para representar diversas hipóteses de profundidade evitando assim a utilização da inicialização atrasada. Seu método utiliza uma aproximação do *Gaussian Sum Filter* (Filtro da soma de gaussianas) que foi utilizado por Kwok and Dissanayake [2005]. Esse método entretanto não prevê a existência de marcos a grandes profundidades. Lemaire et al. [2005] propuseram um método similar porém utilizando inicialização atrasada.

Eade and T. [2006] basearam-se no algoritmo FastSLAM e desenvolveram uma parametrização na qual os marcos do ambiente são inicialmente representados utilizando o inverso de sua profundidade. Essa parametrização traz as vantagens de conseguir representar marcos a grandes distâncias da câmera e permitir uma inicialização mais simples do que as anteriores. Seu método mantém a representação utilizando o inverso da profundidade até que a distribuição de probabilidades convirja quando então o marco é convertido para representação euclidiana. Outro benefício do método é a

capacidade de utilizar os marcos parcialmente inicializados para estimar a orientação da câmera.

Paolo et al. [2003] trabalharam com a extração da estrutura a partir do movimento. Nesse trabalho a extração da estrutura dos objetos é realizada sem a necessidade da extração de características locais da imagem como pontos salientes. As formas extraídas são aproximadas por planos de maneira que as superfícies devem ser aproximadamente planares. Além disso o movimento da câmera deve ser suficientemente lento para que a geometria seja recuperada com sucesso.

Silveira et al. [2009] também trabalharam o problema do SLAM visual sem extração de características locais. O método proposto pelos autores se baseia nas ideias de Paolo et al. [2003]. Assim como nesse último trabalho, o método proposto por Silveira et al. [2009] também aproxima as superfícies por planos. No entanto esse método é mais robusto a mudanças de iluminação além de permitir movimentos mais rápidos. Por questões de eficiência computacional apenas partes da imagem selecionadas automaticamente pelo algoritmo são utilizadas para reconstrução. Os autores citam o fato do trabalho não realizar *loop closing* como uma desvantagem, já que isso impossibilita a correção da pose e da reconstrução estimada por meio da reobservação de marcos já observados.

2.2 Detectores de Pontos Salientes

Grande parte das soluções propostas para o SLAM visual na literatura assumem que marcos no ambiente que podem ser identificados de maneira consistente nas diversas observações. Diversos algoritmos de extração de pontos salientes em imagens já foram descritos ao longo das últimas décadas [Moravec, 1981] [Harris and Stephens, 1988] [Shi and Tomasi, 1994] [Trajkovic and Hedley, 1998]. Esses primeiros algoritmos fornecem apenas as coordenadas dos pontos salientes na imagem o que dificulta o trabalho de correspondência entre as observações, especialmente quando a posição relativa em que as imagens foram adquiridas é desconhecida, tornando seu uso limitado.

Para o tratamento em tempo real do SLAM visual é mais interessante fazer o uso de descritores. Um descritor é um vetor de tamanho fixo que representa a informação sobre a vizinhança de um ponto. A grande vantagem é que esses podem ser comparados de maneira simples utilizando alguma métrica de distância. Quanto menor a distância maior a probabilidade de que os pontos descritos correspondam à projeção de um mesmo marco. Além disso, a representação por descritores é uma maneira compacta de representar a informação sobre a vizinhança dos pontos salientes. Dessa forma essa

informação pode ser armazenada em um banco de dados para comparações futuras, o que permite realizar correspondências simultâneas entre pontos de diversas imagens [Forsyth and Ponce, 2002].

Descritores não são um conceito recente na literatura [Freeman and Adelson, 1991, Koenderink and van Doorn, 1987], e em anos mais recentemente descritores invariantes a transformações de rotação e escala vêm se tornando disponíveis. O conceito de seleção automática de escala foi introduzido por Lindeberg [1994]. Posteriormente Lowe [1999] desenvolveu o primeiro algoritmo de cálculo de descritores voltado especificamente para seleção automática de escala chamado *Scale Invariant Feature Transform* (SIFT). Primeiramente seleciona-se uma escala que maximiza a quantidade de informação. Posteriormente pontos em regiões de pouco contraste são descartados. Em seguida determinam-se as orientações alinhadas com os picos de histogramas angulares de gradientes de intensidade e, por fim, calculam-se os descritores que são compostos por histogramas angulares de 8 posições em 4×4 regiões vizinhas ao ponto resultando em um descritor de 128 elementos. Esse algoritmo foi posteriormente aprimorado em [Lowe, 2004] introduzindo uma nova etapa entre o primeiro e segundo passo na qual as coordenadas do ponto detectado são ajustadas possibilitando melhores correspondências.

O SIFT serviu como base para diversos trabalhos posteriores de cálculo de descritores. Em especial destacam-se o PCA-SIFT [Ke and Sukthankar, 2004] que utiliza a mesma técnica de cálculo de escala e orientação do SIFT. A diferença é que no cálculo dos descritores os gradientes horizontais e verticais são estimados em uma região (escalada e rotacionada) de 41×41 píxeis gerando um descritor contendo $2(41 - 2)^2$ elementos. Esse descritor posteriormente é submetido ao algoritmo de *Principal Components Analysis* (PCA) para gerar um descritor mais compacto contendo 20 elementos.

Uma análise dos métodos de cálculo de descritores realizada por Mikolajczyk and Schmid [2003] demonstrou a superioridade do SIFT em relação aos métodos disponíveis na época. Essa análise não levou em consideração o PCA-SIFT que ainda não havia sido desenvolvido. Posteriormente os mesmos autores incluíram na comparação o PCA-SIFT em [Mikolajczyk and Schmid, 2005], e mostraram que o método não é mais robusto que o SIFT. No texto os autores definem um algoritmo chamado *Gradient Location and Orientation Histogram* (GLOH). O algoritmo divide a vizinhança em torno do ponto saliente em 3 regiões no sentido radial, e 8 regiões angulares (exceto a região central) onde computam, para cada região, um histograma angular de gradientes de intensidade com 16 posições tendo assim um descritor contendo $(8 \times 2 + 1) \cdot 16 = 272$ elementos que posteriormente é analisado por PCA para chegar a um descritor com 128 elementos. A análise revelou que o GLOH é mais distintivo que o SIFT porém

possui maior custo computacional.

O SIFT apesar das alternativas tornou-se a técnica de escolha na maioria das aplicações devido aos trabalhos de Mikolajczyk and Schmid [2003, 2005] e a um menor custo computacional. Apesar do custo do SIFT ser relativamente menor do que ao de outros algoritmos disponíveis, ainda assim o seu tempo de execução é relativamente alto quando a intenção é realizar processamento em tempo real. Isso motivou Bay et al. [2006] a desenvolverem o algoritmo chamado *Speeded Up Robust Features* (SURF). O foco do trabalho de Bay et al. [2006] foi no aumento do desempenho. O algoritmo inicia calculando uma imagem integral da imagem de entrada. Isso permite o cálculo das somas das intensidades de qualquer subregião em tempo constante. A seleção da escala é realizada aproximando-se o laplaciano da gaussiana (LOG) por médias. Assim como no SIFT, a vizinhança do ponto saliente é dividida em 4×4 sub regiões que são posteriormente divididas em 5×5 regiões sobre as quais são calculadas as respostas a wavelets de Haar, que podem ser rapidamente obtidas da imagem integral. O descritor é dado pela soma das respostas e de seus valores absolutos para cada região resultando num descritor que contém quatro valores por região totalizando 64 elementos em cada descritor. Além disso o SURF calcula o sinal do laplaciano para cada ponto o que permite distinguir rapidamente entre objetos escuros em fundos claros e objetos claros em fundos escuros.

2.3 Desenvolvimento em Unidades Gráficas

O uso de hardware gráfico especializado para otimizar computações de propósito geral surgiu como conceito no fim da década de 70 com o trabalho de England [1978].

Lengyel et al. [1990] utilizaram um hardware gráfico para realizar planejamento de movimentação de robôs. Seu planejador utilizava o hardware para renderizar o espaço de obstáculos em várias camadas em uma série de *bitmaps*, e usaram um algoritmo de programação dinâmica nesses *bitmaps* para realizar o planejamento da trajetória. Seu algoritmo é semelhante ao algoritmo desenvolvido por Donald [1987] sendo a principal diferença que o hardware gráfico foi utilizado para alcançar desempenho em tempo real.

A necessidade de soluções em tempo real para os problemas de visão computacional e robótica associada ao grande volume de dados que necessitam ser processados nessas áreas, tem motivando a utilização de paralelismo. Devido ao poder computacional e ao baixo custo as unidades de processamento gráfico (GPUs) vêm, recentemente, sendo utilizadas para computações paralelas de propósito geral. O uso de GPUs para

computações de propósito geral recebe a designação de GPGPU (*General-purpose computing on graphics processing units*). O crescimento no interesse em GPGPU levou ao desenvolvimento de várias arquiteturas e ambientes de desenvolvimento para esse tipo de sistema. O CUDA [CUD, 2010] da NVIDIA e o padrão OpenCL [Khronos OpenCL Working Group, 2008] são alguns exemplos.

O ambiente CUDA da NVIDIA foi o primeiro ambiente de programação de GPUs baseado em linguagem C e vem sendo o ambiente de desenvolvimento de escolha para diversos trabalhos desde então. Apesar de ser direcionado para hardwares da NVIDIA suas abstrações são representações da classe de processamento paralelo *single instruction multiple data* (SIMD) [Garland et al., 2008]

Em anos recentes, Olson [2009] utilizou o poder computacional das GPUs para realizar um tratamento probabilístico da correlação de escaneamentos horizontais baseados em laser. Seu trabalho assume um ambiente bidimensional e estima a translação e rotação do robô entre dois pontos no plano buscando o ponto que maximiza uma probabilidade *a posteriori*. A busca pelo máximo é realizado avaliando diretamente a função em um grid de pontos em uma região do espaço. Apesar de seu trabalho não apresentar eficiência maior do que métodos clássicos como *Iterative Closest Points* (ICP), *Iterative Closest Lines* ICL ou Hill climbing, as estimativas obtidas em geral são muito mais precisas. Além disso seu método consegue fornecer uma estimativa do erro padrão.

GPUs também foram usadas por Yguel et al. [2008] para preencher de maneira eficiente grids de ocupação. Essa técnica tem demonstrado muito sucesso para realização de SLAM em duas dimensões. Infelizmente ela não escala bem para o caso tridimensional.

Vários algoritmos de álgebra linear foram otimizados com sucesso usando GPUs [Bell and Garland, 2009, Barrachina et al., 2008]. Muitos desses algoritmos são usados com frequência em soluções de problemas de visão computacional e robótico. Alguns exemplos são fatoração QR, LU [Galoppo et al., 2005] e decomposição de Cholesky [Jung and O’Leary, 2006], que são utilizadas nas computações das variações do filtro de Kalman.

Allusse et al. [2008] desenvolveram a biblioteca GpuCV, que contém vários algoritmos utilizados em visão computacional e processamento de imagens desenvolvidos em GPUs para alcançar uma maior capacidade de processamento. Entre os algoritmos apresentados se encontram muitos dos algoritmos disponíveis na biblioteca OpenCV como: operações com matrizes, soma, subtração, multiplicação, média, limiarização, operadores como o de Sobel e Deriche, FFT entre outros. A GpuCV realiza ainda, antes das operações, uma estimativa do tempo de processamento necessário em CPU

ou em GPU, escolhendo dinamicamente a unidade que for mais adequada.

Sinha et al. [2006] utilizaram a GPU para paralelizar o SIFT conseguindo uma taxa de quadros de 10 Hz em imagens com resolução de 640×480 píxeis extraindo em média 800 pontos por imagens. O método SURF também foi paralelizado utilizando GPUs por Cornelis and Van Gool [2008]. Seu método obteve taxas de processamento de quadros de 100 Hz para de 640×480 . Diversas outras versões paralelas desses mesmos detectores utilizando hardwares gráficos foram descritas na literatura, obtendo ganhos significativo de desempenho [Heymann et al., 2007], [Terriberry et al., 2008].

Chariot and Keriven [2008] desenvolveram algoritmos paralelos para realizar correspondências entre pontos salientes alcançando um desempenho até $30\times$ superior ao da busca por vizinhos mais próximos em CPUs convencionais.

O presente trabalho tem como objetivo principal o estudo do uso de paralelismo para solucionar o SLAM monocular de maneira mais eficiente e robusta. Grande parte das soluções propostas para o problema SLAM são baseadas em filtros de partícula ou variações do filtro de Kalman. Tendo em vista que o paralelismo oferecido pelas GPUs já foi utilizado para aumentar a eficiência tanto de filtros de partícula, quanto das operações matriciais envolvidas nas computações do filtro de Kalman, acredita-se que o uso de GPUs pode resultar em soluções mais eficientes para o problema. Além disso, algumas soluções propostas como a de Costa et al. [2004] possuem custo computacional elevado, de forma que, dificilmente conseguiriam ser usadas em uma solução em tempo real em CPUs convencionais. Uma solução considerada ineficiente para execução sequencial poderia se mostrar viável por meio do uso de paralelismo. Dessa forma, o paralelismo pode ser utilizado não só para gerar uma solução mais eficiente, mas também mais robusta.

Capítulo 3

Detalhamento do Problema

Esse capítulo detalhará o problema tratado, além de apresentar uma revisão de conceitos teóricos relacionados ao problema e de alguns trabalhos relacionados. São descritos o filtro de Bayes e a decomposição de Rao-Blackwell, dois conceitos fundamentais para compreensão da solução desenvolvida.

3.1 O problema SLAM

A Definição 3 introduz o *SLAM* de maneira genérica como um problema de estimação conjunta da pose s_t e do mapa Θ dadas as observações \mathbf{O}_t e ações a_t . Para especificar o problema de maneira mais detalhada é preciso apresentar uma definição das variáveis envolvidas. Desse maneira a menos que seja especificado o contrário as seguintes considerações deverão ser observadas no restante desse texto:

- O tempo é discreto.
- O mapa Θ do ambiente é constituído por um conjunto de marcos $\{\theta_1, \theta_2, \dots, \theta_n\}$.
- s_t é a pose do robô no instante t
- entre os instantes $t - 1$ e t o robô executa uma única ação a_t
- Em cada instante t o robô obtém um conjunto $\mathbf{O}_t = \{o_k, o_{k+1}, o_{k+m}\}$ de observações.
- Cada observação o_i se refere a um único marco.
- ν_i é uma função que associa a observação o_i ao marco θ_{ν_i} .
- τ_i é o instante de tempo da observação o_i .

Definem-se ainda, o caminho $s^t = \{s_1, \dots, s_t\}$ como conjunto de todas as poses do robô até o instante t . $a^t = \{a_1, \dots, a_t\}$ o conjunto de todas ações executadas e $O^t = \{O_1, \dots, O_t\}$ o conjunto de todas as observações. Cada um desses elementos será definido com maiores detalhes adiante.

3.2 Tratamento probabilístico

O problema *SLAM*, como formulado, pode ser tratado como um problema de estimação pontual de uma pose s_t e um mapa Θ ótimos segundo algum critério de otimalidade. Esse tipo de abordagem foi utilizada por diversos autores como Tomasi and Kanade [1992] e Hartley [1993].

No entanto, os processos de locomoção e sensoriamento estão sujeitos à fenômenos imprevistos, por exemplo, derrapagens, variações na iluminação, vibração, para citar alguns. Isso leva a incertezas nas medidas e observações que, se não tratadas, podem levar a resultados incorretos [Thrun et al., 2005].

Devido a essas incertezas diversos autores [Montemerlo, 2003, Strasdat et al., 2010] defendem que o SLAM deve ser tratado como um processo estocástico. A ideia nesse tipo de abordagem consiste em estimar um estado de crença na pose s_t e no mapa Θ , representado por meio de uma distribuição de probabilidades subjetiva¹.

Um processo estocástico é uma sequência de variáveis aleatórias X_t indexadas pelo tempo t cuja distribuição de probabilidade varia com t . A variável X_t representa o estado do processo em t . A Definição 4 define o estado estimado no caso do SLAM.

Definição 4. *O estado X_t estimado no SLAM é constituído pela pose s_t no instante t e pelo mapa Θ do ambiente.*

A Definição 5 apresenta o estado de crença no caso do SLAM. Esse estado é definido como a distribuição de probabilidades sobre a pose s_t e o mapa Θ , condicionada às observações O_t e ações a_t e a correspondência ν entre observações o_i e marcos θ_j . Apesar de Θ ser considerado estático em grande parte dos casos, ele deve pertencer ao estado X_t , pois a crença no mapa varia com o passar do tempo.

¹A ideia usar distribuições de probabilidade para representar um grau de crença subjetivo foi proposta por John Maynard Keynes no início da década de 20. A justificativa teórica e filosófica para o conceito de probabilidades subjetivas foram desenvolvidos por Bruno de Finetti (*Fondamenti Logici del Ragionamento Probabilistico*, 1930) e Frank Ramsey (*The Foundations of Mathematics*, 1931)[Gillies, 2000]. Do ponto de vista subjetivista a probabilidade associada a um objeto aleatório X reflete uma crença subjetiva sobre esse objeto. Por exemplo, ao afirmar que a probabilidade de precipitação é de 80% temos o clima como objeto aleatório e uma crença na possibilidade de chuva é de 80% e, existe ainda, 20% de crença de que não irá chover.

Definição 5. *O estado de crença $Bel_{t_1}(s_{t_2}, \Theta)$ é definido como a probabilidade conjunta do estado s_{t_2}, Θ condicionada às observações \mathbf{O}^{t_2} , ações a^{t_2} e a função de correspondência ν .*

$$Bel_{t_1}(s_{t_2}, \Theta) = p(s_{t_2}, \Theta | \mathbf{O}^{t_1}, a^{t_1}, \nu)$$

A associação ν , em geral, não pode ser observada diretamente de modo que ela deveria ser estimada juntamente com a pose e o mapa. Todavia, por enquanto será considerado que ela é conhecida.

A principal vantagem do tratamento probabilístico é possibilitar a estimação, não somente uma solução pontual $\hat{s}_t, \hat{\Theta}$, mas também de uma medida quantitativa de confiança na solução obtida. Além disso esse tipo de abordagem, em grande parte dos casos, não é mais complexo do a estimação pontual [Montemerlo, 2003].

Abordagens probabilísticas geralmente modelam o problema como uma cadeia de Markov.

Uma cadeia de Markov é um processo estocástico que possui a propriedade Markov. Um processo estocástico com a propriedade de Markov possui estados passados e futuros independentes dado o estado atual. Esse conceito é formalizado na definição abaixo.

Definição 6. *Um processo estocástico de tempo discreto $F_t \in \mathcal{F}$ é dito ter a propriedade de Markov se*

$$\forall t | p(F_{t+1} | F_t, F_{t-1}, \dots, F_0) = p(F_{t+1} | F_t).$$

No caso em questão considera-se conhecido um modelo de locomoção dado por:

$$s_t = f(s_{t-1}, a_t) + E,$$

onde E é uma variável aleatória que representa a incerteza na movimentação.

Alternativamente pode-se referir ao modelo de locomoção pela função de densidade de probabilidade.

$$p(s_t | s_{t-1}, a_t).$$

e um modelo de observação:

$$o_i = g(\theta_{\nu_i}, s_{\tau_i}) + R,$$

onde R é uma variável aleatória que representa o ruído de observação.

Alternativamente pode-se referir ao modelo de observação pela função de densidade de probabilidade.

$$p(o_i | \theta_{\nu_i}, s_{\tau_i}, \nu_i).$$

A propriedade de Markov simplifica o problema de estimação, pois basta que seja conhecido o estado de crença $Bel_t(s_t, \Theta)$ para que se possa estimar as crenças futuras.

3.3 Filtro de Bayes

Para que seja estimado o estado de crença $Bel_t(s_t, \Theta)$ é necessário um mecanismo que permita incorporar o conhecimento da ação a_t e as observações \mathbf{O}_t à crença do estado anterior $Bel_{t-1}(X_{t-1})$.

O filtro de Bayes é um método de estimação recursivo que se baseia na propriedade de Markov para atingir esse objetivo. O algoritmo é dividido em duas etapas. A primeira, denominada predição, incorpora à crença $Bel_{t-1}(X_{t-1})$ a ação a_t gerando uma crença preditiva $\overline{Bel}_t(X_t)$ do estado atual no instante t . A denominação preditiva advém do fato dessa distribuição prever o estado (s_t, Θ) sem levar em conta as observações o_t .

Definição 7. A crença preditiva $\overline{Bel}_t(s_t, \Theta)$ é definida como a probabilidade condicional do estado (s_t, Θ) dadas as ações a^t executadas e as \mathbf{O}^{t-1} observações anteriores.

$$\overline{Bel}_t(s_t, \Theta) \equiv p(s_t, \Theta | a^t, \mathbf{O}_{t-1}, \nu)$$

Utilizando o teorema da probabilidade total e a definição de probabilidade condicional pode-se decompor a crença preditiva do seguinte modo:

$$\begin{aligned} \overline{Bel}_t(s_t, \Theta) &= p(s_t, \Theta | a^t, \mathbf{O}_{t-1}, \nu) \\ &= \int_{s_{t-1}} p(s_t, \Theta | s_{t-1}, a_t, \mathbf{O}^{t-1}, a^{t-1}, \nu) \times p(s_{t-1} | a_t, \mathbf{O}^{t-1}, a^{t-1}, \nu) ds_{t-1} \\ &= \int_{s_{t-1}} p(s_t | \Theta, s_{t-1}, a_t, \mathbf{O}^{t-1}, a^{t-1}, \nu) \\ &\quad \times p(\Theta | s_{t-1}, a_t, \mathbf{O}^{t-1}, a^{t-1}, \nu) \times p(s_{t-1} | a_t, \mathbf{O}^{t-1}, a^{t-1}, \nu) ds_{t-1} \\ &= \int_{s_{t-1}} p(s_t | \Theta, s_{t-1}, a_t, \mathbf{O}^{t-1}, a^{t-1}, \nu) \times p(\Theta, s_{t-1} | a_t, \mathbf{O}^{t-1}, a^{t-1}, \nu) ds_{t-1}. \end{aligned}$$

A ação a_t se refere ao deslocamento da pose s_{t-1} para s_t e na ausência das observações \mathbf{O}_t obtidas no instante t não possui influência sobre a probabilidade do estado (s_t, Θ) , podendo ser removida do segundo termo da integral levando a:

$$\begin{aligned}\overline{Bel}_t(s_t, \Theta) &= \int_{s_{t-1}} p(s_t|\Theta, s_{t-1}, a_t, \mathbf{O}^{t-1}, a^{t-1}, \nu) \times p(\Theta, s_{t-1}|\mathbf{O}^{t-1}, a^{t-1}, \nu) ds_{t-1} \\ &= \int_{s_{t-1}} p(s_t|\Theta, s_{t-1}, a_t, \mathbf{O}^{t-1}, a^{t-1}, \nu) \times Bel_{t-1}(\Theta, s_{t-1}) ds_{t-1},\end{aligned}$$

e aplicando a propriedade de Markov o primeiro termo da integral fica reduzido ao modelo de locomoção:

$$\overline{Bel}_t(s_t, \Theta) = \int_{s_{t-1}} p(s_t|\Theta, s_{t-1}, a_t) \times Bel_{t-1}(\Theta, s_{t-1}) ds_{t-1}.$$

A segunda etapa do filtro consiste em atualizar a distribuição preditiva pelo teorema de Bayes para incorporar as observações \mathbf{O}_t levando ao estado de crença.

Pelo teorema de Bayes temos:

$$\begin{aligned}Bel_t(s_t, \Theta) &= p(s_t, \Theta|a^t, \mathbf{O}^t, \nu) \\ &\propto p(s_t, \Theta|a^t, \mathbf{O}^{t-1}, \nu) \times p(\mathbf{O}_t|s_t, \Theta, a^t, \mathbf{O}^{t-1}, \nu) \\ &= \overline{Bel}_t(s_t, \Theta) \times p(\mathbf{O}_t|s_t, \Theta, a^t, \mathbf{O}^{t-1}, \nu)\end{aligned}$$

Utilizando novamente Markov para reduzir o segundo termo ao modelo de observação chega-se a:

$$Bel_t(s_t, \Theta) \propto \overline{Bel}_t(s_t, \Theta) \times p(\mathbf{O}_t|s_t, \Theta, \nu)$$

O Algoritmo 1 descreve o filtro.

Algoritmo 1 Filtro de Bayes.

Filtro de Bayes($Bel_{t-1}(\Theta, s_{t-1}), \mathbf{O}_t, a_t, \nu$)

- 1: $\overline{Bel}_t(s_t, \Theta) = \int_{s_{t-1}} p(s_t|\Theta, s_{t-1}, a_t) \times Bel_{t-1}(\Theta, s_{t-1}) ds_{t-1}$.
 - 2: $\eta \leftarrow \left(\int_{s_t} \int_{\Theta} \overline{Bel}_t(s_t, \Theta) \times p(\mathbf{O}_t|s_t, \Theta, \nu) d\Theta ds_t \right)^{-1}$
 - 3: **for all** s_t, Θ **do**
 - 4: $Bel_t(s_t, \Theta) \leftarrow \eta \times \overline{Bel}_t(s_t, \Theta) \times p(\mathbf{O}_t|s_t, \Theta, \nu)$
 - 5: **end for**
 - 6: **return** $Bel_t(s_t, \Theta)$
-

3.4 Filtro de Kalman

Sob a suposição da propriedade de Markov, o filtro de Bayes é capaz de estimar a crença $Bel_t(s_t, \Theta)$ para qualquer instante t a partir da crença inicial $Bel_1(s_1, \Theta)$. Entretanto, na maior parte dos casos é impossível implementá-lo na forma apresentada. Em geral as funções envolvidas são demasiadamente complexas para serem representadas em suas formas analíticas. Ainda assim o filtro de Bayes é interessante do ponto de vista teórico pois serve de base para diversos algoritmos de estimação de estado existentes na literatura [Thrun et al., 2005].

Um caso particular ocorre quando tanto o modelo de propagação (locomoção) quanto o de observação são lineares com ruído gaussiano. Nesse caso as equações do filtro se tornam mais simples e podem ser representadas em sua forma analítica. O filtro de Kalman [Kalman, 1960] é uma especialização do filtro de Bayes para tal caso. Esse filtro já foi utilizado com bastante sucesso na solução de vários problemas de estimação em diversas áreas distintas.

Suponha um processo x_t de tempo discreto regido por

$$\begin{aligned}x_t &= F x_{t-1} + G c_t + e_t \\y_t &= H x_t + r_t \\e_t &\sim \mathcal{N}(0, Q_t) \\r_t &\sim \mathcal{N}(0, R_t)\end{aligned}$$

onde c_t é controle, y_t uma observação do processo e e_t e r_t são ruídos normais com média 0 e variâncias Q_t e R_t respectivamente.

O estado de crença $Bel_t(x_t)$ mantido pelo filtro de Kalman é sempre uma distribuição normal representada por sua média \hat{x}_t e matriz de covariância P_t :

$$Bel_t(x_t) = \mathcal{N}(\hat{x}_t, P_t).$$

A crença é atualizada recursivamente da seguinte forma:

primeiramente é computada a distribuição preditiva $\mathcal{N}(\bar{x}_t, \bar{P}_t)$.

$$\begin{aligned}\bar{x}_t &= F \hat{x}_{t-1} + G c_t \\ \bar{P}_t &= F P_{t-1} F^T + Q_t.\end{aligned}$$

em seguida calcula-se o ganho de Kalman K_t :

$$K_t = \bar{P}_t H^T (H \bar{P}_t H^T + R_t)^{-1}.$$

Finalmente atualiza-se \hat{x}_t e P_t gerando o novo estado de crença:

$$\begin{aligned}\hat{x}_t &= \bar{x}_t + K_t (y_t - H \bar{x}_t) \\ P_t &= (I - K_t H) \bar{P}_t (I - K_t H)^T + K_t R_t K_t^T.\end{aligned}$$

O ganho de Kalman, K_t , pode ser interpretado com sendo a razão do grau de confiança entre a crença preditiva $\mathcal{N}(\bar{x}_t, \bar{P}_t)$ gerada e a observação obtida.

O filtro de Kalman pode ser derivado diretamente do filtro de Bayes substituindo as equações desse último pelos modelos apresentados. Além disso, pode-se mostrar que para o caso linear gaussiano o estado estimado é ótimo no sentido de que a distância quadrática entre o valor estimado \hat{x}_t e x_t é mínima.

No entanto os modelos de locomoção e de observação do SLAM dificilmente caem no caso linear gaussiano. Apesar disso, várias extensões do filtro de Kalman já foram propostas na literatura para modelos não lineares, dentre elas, o filtro de Kalman estendido (EKF) é a mais popular e é o algoritmo de escolha para grande parte das soluções do SLAM propostas [Montemerlo, 2003].

Outra extensão interessante é o filtro de Kalman *unscented* (UKF) que é utilizado nesse trabalho para atualizar as distribuições dos marcos individuais do ambiente. Pode-se mostrar que essa variação gera melhores aproximações das distribuições reais do que o EKF.

As duas subseções seguintes farão uma breve apresentação das equações desses dois filtros.

3.4.1 EKF

O filtro de Kalman estendido é uma extensão do filtro de Kalman para modelos de propagação e observação não lineares que lineariza esses modelos por meio do jacobiano em torno do valor esperado. Suas equações são semelhantes às do filtro de Kalman, diferindo apenas por essa linearização.

Seja um processo x_t de tempo discreto tal que

$$\begin{aligned}x_t &= f(x_{t-1}, c_t, e_t) \\ y_t &= h(x_t, r_t),\end{aligned}$$

onde c_t é controle, y_t uma observação do processo e e_t e r_t são ruídos normais com média 0 e variâncias Q_t e R_t respectivamente.

$$e_t \sim \mathcal{N}(0, Q_t)e$$

$$r_t \sim \mathcal{N}(0, R_t).$$

Assim como no filtro de Kalman, o estado de crença do EKF é representado por

$$Bel_t(x_t) = \mathcal{N}(\hat{x}_t, P_t).$$

A atualização recursiva se dá do seguinte modo:

Computa-se a preditiva:

$$\begin{aligned}\bar{x}_t &= f(\hat{x}_{t-1}, c_t, 0) \\ F_t &= \nabla_{x_{t-1}} f(x_{t-1}, c_t, e_t)|_{x_{t-1}=\hat{x}_{t-1}, c_t=c, e_t=0} \\ L_t &= \nabla_{e_t} f(x_{t-1}, c_t, e_t)|_{x_{t-1}=\hat{x}_{t-1}, c_t=c, e_t=0} \\ \bar{P}_t &= F_t P_{t-1} F_t^T + L_t Q_t L_t^T,\end{aligned}$$

Calcula-se o ganho de Kalman:

$$\begin{aligned}H_t &= \nabla_{x_t} h(x_t, r_t)|_{x_t=\bar{x}_t, r_t=0} \\ M_t &= \nabla_{r_t} h(x_t, r_t)|_{x_t=\bar{x}_t, r_t=0} \\ K_t &= \bar{P}_t H_t^T (H_t \bar{P}_t H_t^T + M_t R_t M_t^T)^{-1}\end{aligned}$$

e computa-se \hat{x}_t e P_t por meio de

$$\begin{aligned}\hat{x}_t &= \bar{x}_t - K_t [y_t - h(\bar{x}_t, 0)] \\ P_t &= (I - K_t H_t) \bar{P}_t\end{aligned}$$

Ressalta-se que diferente do filtro de Kalman, o EKF não é ótimo para o caso geral. Ele tende a gerar “bons” resultados caso os modelos envolvidos sejam aproximadamente lineares e os intervalos de tempo envolvidos sejam pequenos, no entanto pode gerar resultados superconfiantes caso as não linearidades dos modelos façam com que os valores esperados das funções de densidade de probabilidade envolvidas difiram dos valores computados de \bar{x}_t e $h(\bar{x}_t, 0)$.

3.4.2 UKF

A dificuldade no tratamento de sistemas não lineares é que, em geral, as transformadas não lineares das funções de densidade probabilidade são complexas e de difícil tratamento. O EKF trabalha sob a suposição de que as transformações envolvidas são aproximadamente lineares em torno das médias, porém caso isso não se mostre verdadeiro, pode gerar resultados espúrios.

O filtro de Kalman unscented trabalha sob as seguintes premissas:

- a aplicação de transformações não lineares em pontos é relativamente simples,
- é possível aproximar o estado de crença por um conjunto de pontos individuais chamados pontos sigma.

Da mesma forma que no EKF, o UKF aproxima o estado de crença por uma normal.

Seja um processo x_t de tempo discreto de n dimensões regido por:

$$\begin{aligned}x_t &= f(x_{t-1}, c_t) + e_t \\y_t &= h(x_t) + r_t,\end{aligned}$$

onde c_t é controle, y_t uma observação do processo e e_t e r_t são ruídos normais com média 0 e variâncias Q_t e R_t respectivamente.

$$\begin{aligned}e_t &\sim \mathcal{N}(0, Q_t)e \\r_t &\sim \mathcal{N}(0, R_t),\end{aligned}$$

a atualização recursiva procede da seguinte maneira:

computa-se $2n + 1$ pontos sigma $\sigma_{\hat{x}_{t-1}}^{(0)}, \sigma_{\hat{x}_{t-1}}^{(1)}, \dots, \sigma_{\hat{x}_{t-1}}^{(2n)}$ dados por

$$\sigma_{\hat{x}_{t-1}}^{(i)} = \begin{cases} \hat{x}_{t-1} & \text{se } i = 0, \\ \hat{x}_{t-1} + [\sqrt{(n + \lambda)P_{t-1}}]_i & \text{se } i = 1 \dots n, \\ \hat{x}_{t-1} - [\sqrt{(n + \lambda)P_{t-1}}]_{i-n} & \text{se } i = n + 1 \dots 2n, \end{cases}$$

onde $\sqrt{(n + \lambda)P_{t-1}}$ é a raiz quadrada da matriz $(n + \lambda)P_{t-1}$ tal que $(\sqrt{(n + \lambda)P_{t-1}})(\sqrt{(n + \lambda)P_{t-1}})^T = (n + \lambda)P_{t-1}$ e $[\sqrt{(n + \lambda)P_{t-1}}]_i$ é a i -ésima coluna de tal matriz.

λ é um parâmetro de escala definido por:

$$\lambda = \alpha^2(n + \kappa) - n$$

onde α determina o espalhamento dos pontos sigma em torno da média e κ é um parâmetro de escala secundário, geralmente definido como 0.

A distribuição preditiva é obtida por

$$\begin{aligned}\bar{x}_t &= \sum_{i=0}^{2n} \omega_m^{(i)} \cdot f(\sigma_{\hat{x}_{t-1}}^{(i)}, c_t) \\ \bar{P}_t &= \sum_{i=0}^{2n} \omega_c^{(i)} \cdot [\bar{x}_t - f(\sigma_{\hat{x}_{t-1}}^{(i)}, c_t)][\bar{x}_t - f(\sigma_{\hat{x}_{t-1}}^{(i)}, c_t)]^T + Q_t,\end{aligned}$$

onde

$$\begin{aligned}\omega_m^{(0)} &= \frac{\lambda}{n+\lambda} \\ \omega_c^{(0)} &= \frac{\lambda}{n+\lambda} + (1 - \alpha^2 + \beta) \\ \omega_m^{(i)} &= \omega_c^{(i)} = \frac{1}{2n+\lambda} \quad \text{se } i \neq 0.\end{aligned} \tag{3.1}$$

β é usado para incorporar o conhecimento à priori a respeito da distribuição de x_t e sabe-se que para distribuições normais $\beta = 2$ é ótimo [Mer, 2000].

Para a atualização da preditiva devem ser computados os $2n + 1$ pontos sigma dessa distribuição:

$$\bar{x}_t^{(i)} = \begin{cases} \bar{x}_t & i = 0 \\ \bar{x}_t + [\sqrt{(n+\lambda)\bar{P}_t}]_i & i = 1 \dots n \\ \bar{x}_t - [\sqrt{(n+\lambda)\bar{P}_t}]_i & i = n+1 \dots 2n \end{cases}$$

O ganho de Kalman é obtido do seguinte modo:

$$\begin{aligned}\bar{y}_t &= \sum_{i=0}^{2n} \omega_m^{(i)} \cdot h(\bar{x}_t^{(i)}) \\ \bar{H}_t &= \sum_{i=0}^{2n} \omega_c^{(i)} \cdot [\bar{y}_t - h(\bar{x}_t^{(i)})][\bar{y}_t - h(\bar{x}_t^{(i)})]^T + R_t \\ L_t &= \sum_{i=0}^{2n} \omega_c^{(i)} \cdot [\bar{x}_t - \bar{x}_t^{(i)}][\bar{y}_t - h(\bar{x}_t^{(i)})]^T \\ K_t &= L_t \bar{H}_t^{-1},\end{aligned}$$

e conclui-se a atualização fazendo:

$$\begin{aligned}\hat{x}_t &= \bar{x}_t + K_t(y_t - \bar{y}_t) \\ P_t &= \bar{P}_t - K_t \bar{H}_t K_t^T\end{aligned}$$

Pode-se mostrar que os resultados gerados pelo UKF levam a aproximações mais próximas do estado de crença “real” do que o EKF nos casos em que os modelos envolvidos não podem ser bem aproximados pela expansão de Taylor de primeira ordem [Simon, 2006].

3.5 Filtro de Partículas

As variações do filtro de Kalman utilizam uma representação paramétrica do estado crença aproximando este por uma normal. Filtros de partícula no entanto utilizam um conjunto finito de hipóteses sobre o espaço amostral, ou partículas, para representar a crença de maneira que a densidade das partículas tende a ser proporcional à crença da região amostrada. A aproximação tende a melhorar com o aumento do número de partículas utilizadas e, no limite, quando o número de partículas tende a infinito, o estado de crença pode ser reconstituído de maneira exata [Simon, 2006].

Filtro de partículas foram utilizados com sucesso na solução de diversos problemas de estimação de estado, incluindo localização com mapa conhecido [Thrun et al., 2001], rastreamento [Isard and Blake, 1998], dentre outros.

Na localização de Monte Carlo um conjunto de amostras sobre o espaço de poses do robô representa o estado crença na pose e a cada passo as partículas são propagadas por meio do modelo de propagação, Um peso proporcional à verossimilhança da observação é atribuído e as partícula então são reamostradas com probabilidade proporcional a esse peso.

A atualização do estado de crença no filtro de partículas procede do modo descrito a seguir.

Seja x_t um processo estocástico de tempo discreto tal que são conhecidos um modelo de propagação e de observação apresentado a propriedade de Markov:

$$p_p(x_t|x_{t-1}, c_t)e$$

$$p_o(y_t|x_t).$$

O estado de crença $Bel_t x_t$ é representado por meio de um conjunto de n amostras:

$$Bel_t(x_t) = \{x_t^{(1)}, x_t^{(2)}, \dots, x_t^{(n)}\}$$

A atualização a partir da crença no instante anterior se dá em três passos: Amostragem, Importância e Reamostragem:

No primeiro passo é gerada a crença preditiva amostrando cada partícula por meio do modelo de propagação. O Algoritmo 2 demonstra o funcionamento:

No passo seguinte, Importância, à cada amostra gerada é atribuído um peso proporcional à verossimilhança da observação y_t obtida no instante t . Esse comportamento está demonstrado no Algoritmo 3.

Algoritmo 2 Amostragem.

Amostragem($Bel_{t-1}(x_{t-1}), c_t$)

- 1: $\overline{Bel}_t(x_t) \leftarrow \emptyset$
 - 2: **for** $i = 1 \dots n$ **do**
 - 3: $\bar{x}_t^{(i)} \sim p_p(x_t | x_{t-1}^{(i)}, c_t)$. {Gera uma amostra aleatória $\bar{x}_t^{(i)}$ pelo modelo de propagação}
 - 4: $\overline{Bel}_t(x_t) \leftarrow \overline{Bel}_t(x_t) \cup \{\bar{x}_t^{(i)}\}$
 - 5: **end for**
 - 6: **return** $\overline{Bel}_t(x_t)$.
-

Algoritmo 3 Importância.

Importância($\overline{Bel}_t(x_t), y_t$)

- 1: **for** $i = 1 \dots n$ **do**
 - 2: $\omega_t^{(i)} \leftarrow p_o(y_t | \bar{x}_t^{(i)})$. {Atribui à i -ésima partícula peso proporcional à verossimilhança da observação dada a hipótese da partícula.}
 - 3: **end for**
 - 4: **return** ω_t .
-

Finalmente as partículas são reamostradas com probabilidade proporcional ao peso atribuído na etapa anterior. O Algoritmo 4 demonstra como a reamostragem pode ser realizada.

Apesar das vantagens do filtro de partículas é sabido que o número de partículas necessárias para representar o estado x_t cresce exponencialmente com a sua dimensionalidade [Montemerlo, 2003]. Como o estado estimado no SLAM cresce de dimensão à medida que se observam mais marcos, o custo do filtro convencional cresce exponencialmente com o número de marcos existentes no ambiente o que inviabiliza seu uso. Porém, conforme será visto na seção seguinte, o problema do SLAM é dotado de uma propriedade de independência dos marcos condicionada ao conhecimento do caminho. Tal propriedade permite decompor o estado de crença em uma distribuição de probabilidades para o caminho s^t percorrido e uma distribuição para cada marco θ_i condicionada ao caminho s^t . Essa decomposição, conhecida como decomposição de Rao-Blackwell, possibilita o uso de um filtro de partículas para computar a crença no caminho s^t marginalizando-se o cálculo das crenças em cada marco θ_i por meio de outros filtros.

3.6 Decomposição de Rao-Blackwell

A dificuldade do *SLAM* advém do fato de que as incertezas nas localizações dos marcos são correlacionadas por meio da incerteza da pose s_t . Isso pode ser observado na Figura

Algoritmo 4 Reamostragem.Reamostragem($\overline{Bel}_t(x_t), \omega_t$)

```

1:  $S_t \leftarrow 0$ 
2: for  $i = 1 \dots n$  do
3:    $S_t \leftarrow S_t + \omega_t^{(i)}$ 
4:    $\Omega_t^{(i)} \leftarrow S_t$ . {Computa a soma de prefixos dos pesos}
5: end for
6:  $Bel_t(x_t) \leftarrow \emptyset$ 
7: for  $i = 1 \dots n$  do
8:    $u \sim U(0, S_t)$  {Gera um número aleatório com distribuição uniforme entre 0 a soma dos pesos}
9:    $min \leftarrow 0$ 
10:   $max \leftarrow n$ 
11:  while ( $\Omega_t^{(min)} < \Omega_t^{(max)}$ ) do {Pesquisa binária}
12:     $mid \leftarrow \frac{max-min}{2}$ 
13:    if  $u < \Omega_t^{(j)}$  then
14:       $max \leftarrow mid$ 
15:    else
16:       $min \leftarrow mid$ 
17:    end if
18:  end while
19:   $x_t^{(i)} \leftarrow \bar{x}_t^{(j)}$ 
20:   $Bel_t(x_t) \leftarrow Bel_t(x_t) \cup \{x_t^{(i)}\}$ 
21: end for
22: return  $Bel_t(x_t)$ .

```

3.1, na qual o problema é representado como uma rede de Bayes. Essa correlação força a estimação conjunta da pose s_t e da localização dos marcos θ_i .

Observando a Figura 3.1 percebe-se que as incertezas nas localizações dos marcos θ_i somente estão correlacionadas umas com as outras por meio da incerteza no caminho s^t . Dessa forma, se o caminho s^t percorrido pelo robô fosse conhecido as distribuições de cada marco poderiam ser estimadas de maneira independente.

$$p(s^t, \Theta | a^t, \mathbf{O}^t, \nu) = p(s^t | a^t, \mathbf{O}^t, \nu) \prod_{i=1}^n p(\theta_i | s^t, a^t, \mathbf{O}^t, \nu) \quad (3.2)$$

A fatoração do estado de crença apresentada na Equação 3.2 foi apresentada por Murphy [1999]. Ela permite que o SLAM seja decomposto caso a *posteriori* seja estimada sobre todo o caminho s^t ao invés de somente a pose s_t mais.

Montemerlo et al. [2002] utilizaram essa estratégia para tratar o problema. O algoritmo proposto por esses autores estima o caminho s^t percorrido por meio de um

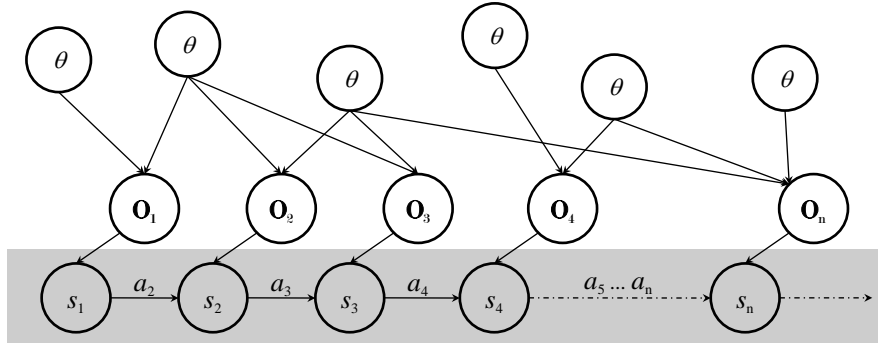


Figura 3.1. Representação do problema SLAM como uma rede de Bayes.

filtro de partículas, sendo que para cada partícula é solucionado um problema de mapeamento com pose conhecida.

A validade dessa fatoração está demonstrada na subseção seguinte.

3.6.1 Demonstração da fatoração

Partindo da definição de probabilidade condicional, a distribuição em questão pode ser fatorada em:

$$p(s^t, \Theta | a^t, \mathbf{O}^t, \nu) \stackrel{\text{Condiciona}}{=} p(s^t | a^t, \mathbf{O}^t, \nu) p(\Theta | s^t, a^t, \mathbf{O}^t, \nu),$$

assim para demonstrar que a fatoração é válida basta, portanto demonstrar que

$$p(\Theta | s^t, a^t, \mathbf{O}^t, \nu) = \prod_{i=1}^n p(\theta_i | s^t, a^t, \mathbf{O}^t, \nu). \quad (3.3)$$

A demonstração será feita por indução sobre t . Para isso serão utilizados dois resultados intermediários. Primeiramente considere que se um marco θ_i não é observado sua crença deve se manter a mesma, o que é explicitado pela Equação 3.4.

$$p(\theta_i | s^t, a^t, \mathbf{O}^t, \nu) = p(\theta_i | s^{t-1}, a^{t-1}, \mathbf{O}^{t-1}, \nu), \nexists \{o_j \in \mathbf{O}_t | \nu_j = i\}. \quad (3.4)$$

Por outro lado considere uma observação $o_j \in \mathbf{O}_t$ tal que $\nu_j = i$. Como a pose s_t é conhecida, o_j deve depender somente de θ_i . Além disso o_j deve ser a única observação que possui influência sobre a crença de θ_i . Desse modo pode-se concluir que:

$$\begin{aligned} p(\theta_i | s^t, a^t, \mathbf{O}^t, \nu) &= p(\theta_i | s^t, a^t, \mathbf{O}^{t-1}, o_j, \nu) \\ &\stackrel{\text{Bayes}}{\propto} p(o_j | \theta_i, s^t, a^t, \mathbf{O}_{t-1}, \nu) p(\theta_i | s^t, \mathbf{O}_{t-1}, a^t, \nu) \\ &\stackrel{\text{Markov}}{=} p(o_j | \theta_i, s_t, \nu) p(\theta_i | s^t, \mathbf{O}_{t-1}, a^t, \nu). \end{aligned} \quad (3.5)$$

Como a pose s_t e a ação a_t não podem trazer informações a respeito de θ_i sem que haja uma observação no instante t , elas podem ser removidas do segundo fator à direita levando a

$$p(\theta_i|s^t, a^t, \mathbf{O}^{t-1}, o_j, \nu) \propto p(o_j|\theta_i, s_t, \nu)p(\theta_i|s^{t-1}, \mathbf{O}_{t-1}, a^{t-1}, \nu). \quad (3.6)$$

Resolvendo a Equação 3.6 equação chega-se à Equação 3.7:

$$p(\theta_i|s^{t-1}, \mathbf{O}_{t-1}, a^{t-1}, \nu) \propto \frac{1}{p(o_j|\theta_i, s_t, \nu)}p(\theta_i|s^t, \mathbf{O}_t, a^t, \nu). \quad (3.7)$$

assumindo-se a Equação 3.8 como hipótese de indução:

$$p(\Theta|s^{t-1}, \mathbf{O}^{t-1}, a^{t-1}, \nu) = \prod_{i=1}^n p(\theta_i|s^{t-1}, \mathbf{O}^{t-1}, a^{t-1}, \nu) \quad (3.8)$$

obtém-se

$$\begin{aligned} p(\Theta|s^t, \mathbf{O}^t, a^t, \nu) &\stackrel{\text{Bayes}}{\propto} p(\mathbf{O}_t|\Theta, s^t, a^t, \mathbf{O}^{t-1}, \nu)p(\Theta|s^t, \mathbf{O}^{t-1}, a^t, \nu) \\ &\stackrel{\text{Markov}}{\propto} p(\mathbf{O}_t|\Theta, s^t, a^t, \mathbf{O}^{t-1}, \nu)p(\Theta|s^{t-1}, \mathbf{O}^{t-1}, a^{t-1}, \nu) \\ &\stackrel{\text{Hipótese}}{\propto} p(\mathbf{O}_t|\Theta, s^t, a^t, \mathbf{O}^{t-1}, \nu) \prod_{i=1}^n p(\theta_i|s^{t-1}, \mathbf{O}^{t-1}, a^{t-1}, \nu). \end{aligned}$$

Como para $t = 0$ nenhuma observação foi incorporada à *posteriori* a hipótese é trivialmente verdadeira.

Seja $\mathbf{O}_t = \{o_k, \dots, o_{k+l}\}$. Como cada observação o_j depende somente de s_{τ_j} e θ_{ν_j} conclui-se que

$$p(\Theta|s^t, \mathbf{O}^t, a^t, \nu) \propto \prod_{j=k}^{k+l} p(o_j|\theta_{\nu_j}, s^t, a^t, \mathbf{O}^{t-1}, \nu) \prod_{i=1}^n p(\theta_i|s^{t-1}, \mathbf{O}^{t-1}, a^{t-1}, \nu). \quad (3.9)$$

Definindo o conjunto Θ_t dos marcos para os quais existe um observação em t pode-se escrever a Equação 3.9 como:

$$p(\Theta|s^t, \mathbf{O}^t, a^t, \nu) \propto \prod_{j=k}^{k+l} p(o_j|\theta_{\nu_j}, s^t, a^t, \mathbf{O}^{t-1}, \nu) p(\theta_{\nu_i}|s^{t-1}, \mathbf{O}^{t-1}, a^{t-1}, \nu) \prod_{\theta_i \notin \Theta_t}^n p(\theta_i|s^{t-1}, \mathbf{O}^{t-1}, a^{t-1}, \nu). \quad (3.10)$$

Substituindo na Equação 3.10 o resultado obtido na Equação 3.7, obtém-se:

$$p(\Theta|s^t, \mathbf{O}^t, a^t, \nu) \propto \prod_{\theta_i \in \Theta_t}^n p(\theta_i|s^t, \mathbf{O}^t, a^t, \nu) \prod_{\theta_i \notin \Theta_t}^n p(\theta_i|s^{t-1}, \mathbf{O}^{t-1}, a^{t-1}, \nu), \quad (3.11)$$

e aplicando a Equação 3.4 ao resultado anterior chega-se a

$$p(\Theta|s^t, \mathbf{O}^t, a^t, \nu) \propto \prod_{\theta_i \in \Theta_t}^n p(\theta_i|s^t, \mathbf{O}^t, a^t, \nu) \prod_{\theta_i \notin \Theta_t}^n p(\theta_i|s^t, \mathbf{O}^t, a^t, \nu) = \prod_{i=1}^n p(\theta_i|s^t, \mathbf{O}^t, a^t, \nu).$$

Finalmente, aplicando o teorema da probabilidade total, pode se concluir que

$$\begin{aligned} p(\Theta|s^t, \mathbf{O}^t, a^t, \nu) &= \eta \prod_{i=1}^n p(\theta_i|s^t, \mathbf{O}^t, a^t, \nu) \\ &= \frac{\prod_{i=1}^n p(\theta_i|s^t, \mathbf{O}^t, a^t, \nu)}{\int_{\theta_1} \dots \int_{\theta_n} \prod_{i=1}^n p(\theta_i|s^t, \mathbf{O}^t, a^t, \nu) d\theta_1 \dots d\theta_n} \\ &= \frac{\prod_{i=1}^n p(\theta_i|s^t, \mathbf{O}^t, a^t, \nu)}{\prod_{i=1}^n \int_{\theta_i} p(\theta_i|s^t, \mathbf{O}^t, a^t, \nu) d\theta_i} \\ &= \prod_{i=1}^n p(\theta_i|s^t, \mathbf{O}^t, a^t, \nu). \end{aligned}$$

■

Note que a prova depende de cada $\theta_j \in \Theta_t$ ser condicionalmente independente dados Θ e a pose s_t .

3.7 Modelos

Para que a especificação do problema seja completa é necessário que sejam especificados um modelo de pose e de locomoção, bem como um modelo de mapa e de observação.

Os modelos de pose e locomoção serão baseado em um robô diferencial se movendo-se em uma superfície plana, e o modelo de observação será o de uma câmera do tipo *pinhole* com projeção perspectiva. A câmera será considerada fixa em relação

ao referencial do robô. Como câmeras são capazes de recuperar a informação tridimensional do ambiente, o mapa do ambiente será modelado como um conjunto de pontos em \mathbb{R}^3 .

3.7.1 Pose e Locomoção

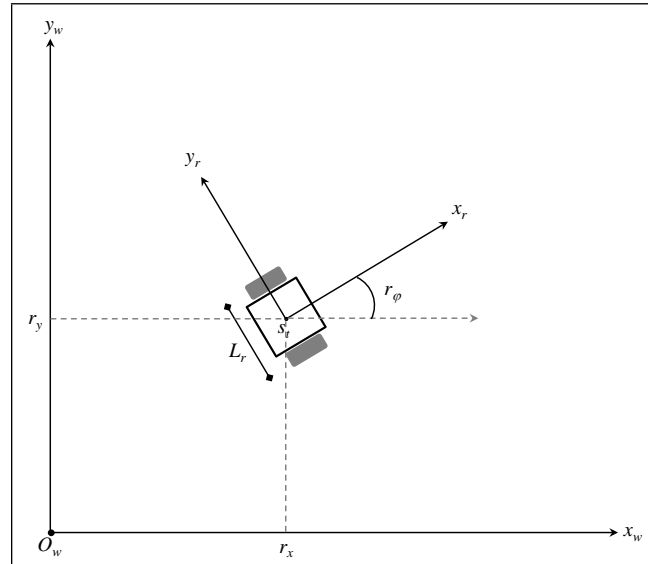


Figura 3.2. Diagrama de um robô diferencial em um ambiente plano. Os valores r_x e r_y representam a posição do robô em relação ao referencial do mundo, r_φ a orientação do robô em relação ao eixo x_w , e L_r a distância entre as rodas do robô.

A Figura 3.2 mostra uma representação de um robô diferencial r em um ambiente plano. Sua pose s_t pode ser especificada por meio de sua posição em relação ao referencial do mundo $(x, y) = (r_x, r_y)$ e sua orientação $\varphi = r_\varphi$. Como o robô se move no plano, a coordenada do robô em relação ao eixo z bem como a orientação em torno dos eixos x e y podem ser desconsiderados. Assim a pose s_t do robô no instante t pode ser representada como

$$s_t = \begin{pmatrix} r_{x_t} \\ r_{y_t} \\ r_{\varphi_t} \end{pmatrix}.$$

A ação a_t em questão consiste em deslocar as rodas direita e esquerda de quantidades ΔD_r e ΔE_r respectivamente, assim representa-se a ação a_t por:

$$a_t = \begin{pmatrix} \Delta D_{r_t} \\ \Delta E_{r_t} \end{pmatrix}.$$

Sejam L_r a distância entre as rodas desse robô e $\frac{dD_{r_t}}{d\tau}$ e $\frac{dE_{r_t}}{d\tau}$ a velocidade instantânea das rodas direita e esquerda respectivamente. Pode se mostrar que para um robô diferencial sem derrapagem a derivada da pose s_t com respeito ao tempo pode ser descrita do seguinte modo [Siegwart and Nourbakhsh, 2004]:

$$\frac{ds_t}{d\tau} = \begin{pmatrix} \frac{\cos r_{\varphi_t}}{2} & \frac{\cos r_{\varphi_t}}{2} \\ \frac{\sin r_{\varphi_t}}{2} & \frac{\sin r_{\varphi_t}}{2} \\ \frac{1}{L_r} & -\frac{1}{L_t} \end{pmatrix} \begin{pmatrix} \frac{dD_{r_t}}{d\tau} \\ \frac{dE_{r_t}}{d\tau} \end{pmatrix}.$$

Não ocorrendo derrapagens o deslocamento Δs_t ocorrido entre os instantes $t - 1$ e t deverá ser igual a

$$\Delta s_t = \int_{t-1}^t \frac{ds_t}{d\tau} d\tau.$$

Considerando as velocidades $\frac{dD_{r_t}}{d\tau}$ e $\frac{dE_{r_t}}{d\tau}$ constantes no intervalo $(t - 1, t)$ pode-se resolver a integral para encontrar

$$\Delta s_t = \begin{pmatrix} \Delta r_{x_t} \\ \Delta r_{y_t} \\ \Delta r_{\varphi_t} \end{pmatrix} = \begin{pmatrix} \frac{\cos(r_{\varphi_t} + \Delta r_{\varphi_t}/2)}{2} & \frac{\cos(r_{\varphi_t} + \Delta r_{\varphi_t}/2)}{2} \\ \frac{\sin(r_{\varphi_t} + \Delta r_{\varphi_t}/2)}{2} & \frac{\sin(r_{\varphi_t} + \Delta r_{\varphi_t}/2)}{2} \\ \frac{1}{L_r} & -\frac{1}{L_t} \end{pmatrix} \begin{pmatrix} \Delta D_{r_t} \\ \Delta E_{r_t} \end{pmatrix}.$$

Conclui-se o modelo de locomoção considerando que esse deslocamento é exato quando são conhecidos os valores reais de ΔD_{r_t} e ΔE_{r_t} e que $\widehat{\Delta D_{r_t}}$ e $\widehat{\Delta E_{r_t}}$ são observações desses valores tais que:

$$\begin{pmatrix} \Delta D_{r_t} \\ \Delta E_{r_t} \end{pmatrix} \sim \mathcal{N} \left(\begin{pmatrix} \widehat{\Delta D_{r_t}} \\ \widehat{\Delta E_{r_t}} \end{pmatrix}, \begin{pmatrix} \alpha \times \widehat{\Delta D_{r_t}} + \beta & 0 \\ 0 & \alpha \times \widehat{\Delta E_{r_t}} + \beta \end{pmatrix} \right),$$

onde α e β são constantes escalares.

3.7.2 Mapa e Observação

O modelo de observação é definido como

$$p(\mathbf{O}_t | \Theta, s_t, \nu).$$

Como já foi mencionado o modelo adotado é baseado em um câmera tipo do *pinhole* e cada marco no ambiente é representado por um ponto em \mathbb{R}^3 .

$$\theta_i = \begin{pmatrix} \theta_{x_i} \\ \theta_{y_i} \\ \theta_{z_i} \end{pmatrix}.$$

Cada observação o_j será um ponto na imagem detectado por meio de alguma técnica de detecção de pontos salientes como *SIFT* ou *SURF*:

$$o_j = \begin{pmatrix} o_{u_j} \\ o_{v_j} \end{pmatrix}.$$

Dado que cada observação o_j é considerada independente dado Θ , s_t e ν pode-se fatorar o modelo em

$$p(\mathbf{O}_t | \Theta, s_t, \nu) = \prod_{o_i \in \mathbf{O}_t} p(o_i | \Theta, s_t, \nu).$$

Além disso, como a observação o_i refere-se somente ao marco θ_{ν_i} esse modelo pode ser reescrito do seguinte modo:

$$p(\mathbf{O}_t | \Theta, s_t, \nu) = \prod_{o_i \in \mathbf{O}_t} p(o_i | \theta_{\nu_i}, s_t, \nu).$$

Para concluir o modelo de observação é preciso definir $p(o_i | \theta_{\nu_i}, s_t, \nu)$. Seja p um ponto do ambiente descrito em relação ao referencial da câmera como:

$$p_c = \begin{pmatrix} x_c \\ y_c \\ z_c \end{pmatrix}$$

e sejam

$$o = \begin{pmatrix} u \\ v \end{pmatrix}$$

as coordenadas de sua projeção no plano de imagem.

Sendo f a distância focal da câmera, pode-se mostrar que p_c e o devem obedecer à seguinte relação:

$$\zeta \begin{pmatrix} u \\ v \\ f \end{pmatrix} = \begin{pmatrix} x_c \\ y_c \\ z_c \end{pmatrix}.$$

Resolvendo ζ chega-se à Equação 3.12

$$\begin{pmatrix} u \\ v \end{pmatrix} = \frac{f}{z_c} \begin{pmatrix} x_c \\ y_c \end{pmatrix}. \quad (3.12)$$

Sendo C_t a pose da câmera no instante t definida em relação ao ambiente por

$$C_t = \begin{pmatrix} C_{x_t} \\ C_{y_t} \\ C_{z_t} \\ C_{\varphi[x]_t} \\ C_{\varphi[y]_t} \\ C_{\varphi[z]_t} \end{pmatrix},$$

pode-se encontrar uma matriz de rotação ${}^w R_{C_t}$ ortonormal e um vetor de translação T_{C_t} que descrevem o referencial da câmera em relação ao ambiente.

Conhecidos ${}^w R_{C_t}$ e T_{C_t} , um ponto p_w cujas coordenadas em relação ao referencial do mundo são dadas por

$$p_w = \begin{pmatrix} x_w \\ y_w \\ z_w \end{pmatrix}$$

pode ser descrito no referencial C_t da câmera é dada como:

$$p_c = {}^w R_{C_t}(p_w - T_{C_t}). \quad (3.13)$$

Concluí-se o modelo de observação assumindo que, dada a posição de um marco θ_i , o valor esperado pela observação pode ser obtido das equações 3.12 e 3.13, e que a incerteza em relação à direção de observação pode ser aproximada por uma normal de média 0 e covariância fixa e independente da profundidade.

A Equação 3.14 mostra o modelo de observação. $p_c^{\theta_{\nu_j}}$ representa a posição do marco θ_{ν_j} , o_j a observação desse marco e r_t o ruído.

$$\begin{aligned} p_c^{\theta_{\nu_j}} &= \begin{pmatrix} \theta_{\nu_j} \\ x_c \\ y_c \\ z_c \end{pmatrix} = {}^w R_{C_t}(\theta_i - T_{C_t}) \\ o_j &= \frac{f}{z_c^{\theta_{\nu_j}}} \begin{pmatrix} \theta_{\nu_j} \\ x_c \\ y_c \end{pmatrix} + r_t \\ r_t &\sim \mathcal{N}(0, R_o). \end{aligned} \tag{3.14}$$

Capítulo 4

Arquitetura de GPUs

Como o próprio nome sugere, GPUs (Graphical Processing Units) surgiram como um hardware especializado para aceleração de aplicações gráficas, principalmente pelo uso das interfaces OpenGL e DirectX. Devido ao paralelismo inerente aos algoritmos de renderização, as GPUs vieram a se tornar unidades de processamento paralelo.

GPUs atuais chegam a uma taxa de pico de execução da ordem de *TeraFLOPs*, ultrapassando em muito o poder de processamento das CPUs convencionais. Consequentemente o interesse na utilização dessas unidades para processamentos de propósito geral vem crescendo com o passar do tempo. Esse crescente interesse levou ao surgimento de arquiteturas e modelos voltados especificamente para o desenvolvimento de aplicações de propósito geral em GPUs dos quais o CUDA [CUD, 2010] da NVIDIA e o padrão OpenCL [Khronos OpenCL Working Group, 2008] são alguns exemplos.

Nesse capítulo será feita uma breve revisão sobre alguns detalhes da arquitetura paralela das GPUs da NVIDIA e sobre o modelo de programação paralelo do CUDA. As seções seguintes apresentarão a organização das GPUs e seu modelo de desenvolvimento.

4.1 Organização

As GPUs na arquitetura CUDA são divididas em multiprocessadores cuja quantidade varia conforme o preço e a tecnologia empregada em cada unidade. Cada multiprocessador, por sua vez, é composto de 8 processadores, que acessam uma memória compartilhada de baixa latência. Cada multiprocessador possui um banco de registradores próprio de propósito geral de 32 bits, cuja quantidade varia entre 1024 nas unidades mais antigas a 4096 nas mais recentes.

O conjunto de *threads* na GPU executa como um *grid*, que por sua vez é dividido em blocos de *threads* como mostrado na Figura 4.1. O *grid* pode conter milhões de blocos, e a criação de *threads* em GPUs é realizada completamente em hardware e é praticamente instantânea os multiprocessadores são responsáveis pela criação, gerencia e escalonamento das *threads*. Cada bloco é alocado a um multiprocessador e não migra até o fim de sua execução. Um bloco pode conter até 512 *threads* nas unidades antigas e 1024 *threads* nas unidades atuais. As *threads* de um mesmo bloco podem se sincronizar por meio de barreira. Ou seja, ao chegar à barreira, uma *thread* somente pode prosseguir em sua execução após todas as *threads* do bloco chegarem à barreira. Unidades recentes possuem conjuntos de instruções atômicas que podem ser usadas para implementar outras primitivas de sincronização, porém isso pode ter um impacto significativo no desempenho sendo desaconselhado.

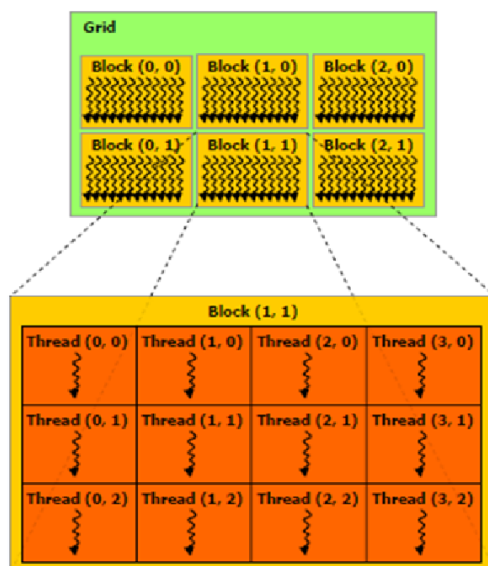


Figura 4.1. Organização da execução das *threads* em GPU [CUD, 2010].

A independência entre *threads* de blocos distintos possibilita que um bloco seja executado em qualquer multiprocessador paralelamente aos outros blocos. Isso permite a escalabilidade do código no número de multiprocessadores da GPU, ou seja um código ao ser executado em uma GPU com um número maior de multiprocessadores tem seus blocos redistribuídos automaticamente aumentando o desempenho (a Figura 4.2 exemplifica essa característica).

As *threads* em um mesmo bloco são divididas em grupo de 32, chamados *warps*. Ao receber um bloco de *threads* para executar o multiprocessador divide esse bloco em *warps* que passa para o escalonador de *warps* para serem escalonados. Os *warps*

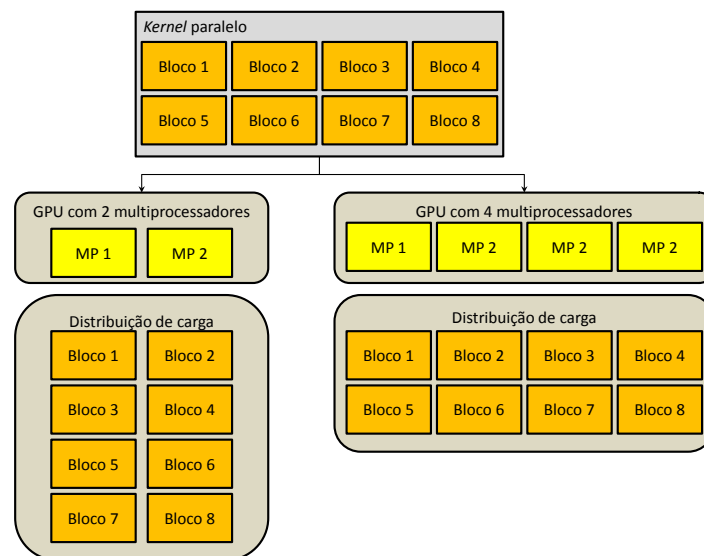


Figura 4.2. Distribuição da carga em GPUs com números diferentes de multiprocessadores.

sempre são compostos por *threads* com índices consecutivos, com o primeiro contendo as threads 0 à 31 do bloco e assim por diante.

As *threads* em um *warp* iniciam sua execução ao mesmo tempo e em um mesmo endereço de programa, mas seus caminhos de execução podem divergir. No entanto as *threads* no *warp* são capazes de executar apenas uma única instrução comum de cada vez, e desta forma, divergência entre os caminhos de execução em um mesmo *warp* provoca redução de desempenho, pois as *threads* divergentes devem esperar até que as outras terminem sua execução para prosseguirem.

O contexto de execução de cada *warp* é sempre armazenado no próprio multiprocessador durante todo o ciclo de vida do *warp*. Dessa forma, a troca de contexto não possui custo algum. Todavia o número de *threads* e blocos residentes no multiprocessador depende da quantidade de registradores e tamanho da memória compartilhada e da quantidade de memória utilizada por cada *thread*. Se a memória compartilhada ou o número de registradores não for suficiente para executar um bloco a execução do *kernel* falhará. Além disso, uma baixa quantidade de *threads* residentes por multiprocessador pode prejudicar a performance, principalmente quando do acesso à memória global. Desse modo, muitas vezes é mais eficiente dividir o código em vários *kernels* pequenos, de forma a aumentar o número de *threads* residentes, em vez de realizar o processamento em um único *kernel*.

Todos os multiprocessadores possuem acesso a uma mesma memória global. O acesso a essa memória possui custo elevado e pode comprometer o desempenho caso essa memória seja muito requisitada. O acesso a blocos consecutivos de memória é

mais rápido do que o acesso a blocos aleatórios pois o hardware é capaz de agrupar esses acessos em um acesso único. GPUs mais recentes da NVIDIA também possuem uma memória cache em 2 níveis. O primeiro nível é composto por uma cache compartilhada pelo multiprocessador, enquanto que o segundo nível é compartilhado por todos os multiprocessadores. Isso reduz significativamente o impacto do acesso à memória global.

Além da memória global cada multiprocessador possui acesso a uma memória compartilhada. Diferentemente das memórias *caches* das CPUs, a carga dos dados na memória compartilhada deve ser realizada “manualmente”. Para obter uma largura de banda mais alta, a memória compartilhada é dividida em módulos de memória de tamanho igual, chamados bancos, que podem ser acessados simultaneamente. Dessa forma quaisquer n requisições de leitura ou escrita que recaiam sobre n bancos de memória distintos podem ser processados simultaneamente. Caso duas ou mais requisições recaiam sobre o mesmo banco ocorre um conflito de banco e assim uma delas deve aguardar o termino da outra. O banco acessado por uma requisição pode ser determinado pelo resto da divisão do endereço de memória acessado pela quantidade de bancos de memória compartilhada. Por ser determinístico o acesso à memória compartilhada pode ser planejado de forma a evitar conflitos de banco.

A Figura 4.3 mostra diferentes padrões de acesso à memória compartilhada. Nos dois primeiros exemplos o acesso é feito a bancos diferente e a uma mesma palavra no mesmo banco, desse modo não ocorre conflito de banco. No terceiro exemplo as *threads* tentam acessar palavras distintas do mesmo banco o que causa um conflito de banco.

4.2 Modelo de desenvolvimento

O ambiente CUDA é baseado em linguagem C. Em seu modelo de desenvolvimento, a aplicação se divide em um programa sequencial hospedeiro que tipicamente executa na CPU e procedimentos paralelos chamados *kernels* que, tipicamente, executam na GPU. Um *kernel* é um procedimento sequencial que quando chamado é executado N vezes em paralelo por N *threads* distintas. Um *kernel* consiste de um *grid* cujo número de *threads* e de blocos deve ser especificado em tempo de invocação. A Figura 4.4 mostra como é a separação e o fluxo de execução do código sequencial e paralelo.

O código hospedeiro suporta todas as funcionalidades existentes em C/C++, no entanto apenas um subconjunto é suportado pelo código de dispositivo (*kernels*). Apesar de permitir a declaração de classes, o código de dispositivo não suporta herança, além disso as primeiras GPUs permitam apenas declarações de funções *inline* e não

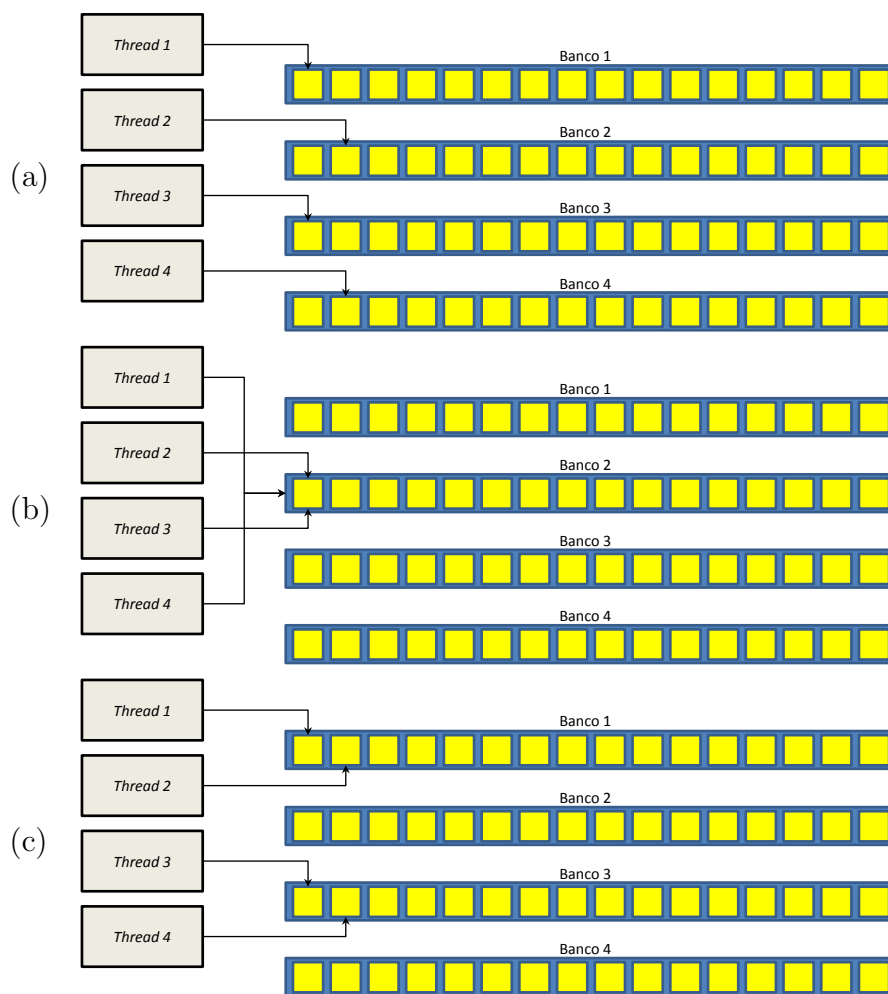


Figura 4.3. Exemplos de padrões de acesso a memória compartilhada para 4 *threads* e 4 bancos de memória: (a) cada *thread* acessa uma palavra em um banco distinto, (b) as *threads* acessam a mesma palavra no segundo banco, (c) conflito de banco entre as *threads* 1, 2 e 3, 4.

permitiam recursão. Para declarar uma função para ser usada em dispositivo usa-se a palavra chave `__device__`.

A Figura 4.5 apresenta um exemplo de declaração e invocação de um *kernel*. O *kernel* é declarado como um procedimento precedido pela palavra chave `__global__`. Na invocação do *kernel* são passados os apontadores para os vetores A, B e C que devem ter sido previamente alocados na memória do dispositivo. A configuração do *kernel* é especificada no momento da invocação usando `<<< ... >>>` sendo que as quantidades *blocks* e *threadsPerBlock* passadas indicam respectivamente o número de blocos e o número de *threads* por bloco.

Cada bloco e cada *thread* do bloco possui um identificador. Esses identificadores

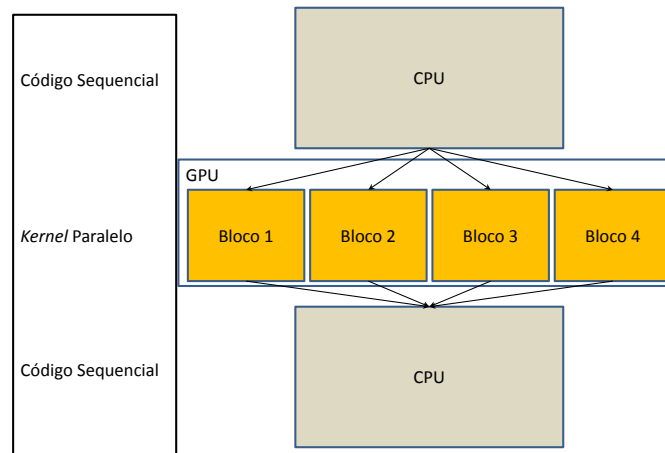


Figura 4.4. Operação do tipo *fork/join*, código sequencial é executado em CPU enquanto código paralelo é executado em GPU [CUD, 2010].

```

__global__ void VecAdd(float *A, float *B, float *C, int N) {
    const int i = blockIdx.x * blockDim.x + threadIdx.x;
    if (i < N)
        C[i] = A[i] + B[i];
}

int main() {
    ...
    const int threadsPerBlock = 256;
    const int blocks = (N - 1) / threadsPerBlock + 1;
    VecAdd<<<blocks, threadsPerBlock>>>(A, B, C, N);
    ...
}

```

Figura 4.5. Exemplo de declaração e invocação de um *kernel*.

podem ser acessados por meio das variáveis *blockIdx.x* e *threadIdx.x*. A variável *blockDim.x* informa o número de *threads* por bloco.

As *threads* no ambiente CUDA podem acessar dados de diversos espaços de memória. Cada *thread* possui uma memória local e as *threads* em um mesmo bloco dividem uma memória compartilhada. Para declarar um vetor ou variável na memória compartilhada utiliza-se a palavra chave **__shared__** variáveis na memória compartilhada são visíveis a todas as *threads* no mesmo bloco.

Fatores como conflitos de banco e divergência de caminhos de execução podem ser ignorados no desenvolvimento do código, porém isso poderia ocasionar perda de desempenho do código. Assim caso se deseje um código mais eficiente esses fatores devem ser levados em consideração.

O próximo capítulo apresentará a solução desenvolvida. O algoritmo proposto nesse trabalho baseia-se em filtros de partículas e, como será exposto na seção 5.5, adotou-se como estratégia de paralelização a utilização de um bloco por partícula, de forma a alcançar um alto grau de paralelismo.

Capítulo 5

Metodologia

A solução desenvolvida nesse trabalho é inspirada principalmente pelo algoritmo FastSLAM 2.0 [Montemerlo, 2003] e como nesse método, a última observação é incorporada à distribuição proposta antes que as novas partículas sejam geradas. Porém em vez de fazê-lo por meio do EKF, o método aqui proposto incorpora essa informação por meio de um segundo filtro de partículas que executa para cada partícula, estratégia semelhante à utilizada por Blanco et al. [2010].

A metodologia desenvolvida pode ser dividida em seis etapas distintas.

1. Extração de pontos salientes.
2. Geração de amostras da pose.
3. Associação de dados.
4. Atualização das amostras.
5. Atualização da pose.
6. Atualização do mapa.

Nesse trabalho optou-se por utilizar o algoritmo SURF [Bay et al., 2006] para obtenção do conjunto de pontos salientes do modelo de observação, devido principalmente à eficiência do método.

O estado de crença $Bel_t(s^t, \Theta)$ é representado por um conjunto de M partículas

$$Bel_t(s^t, \Theta) = \{X_t^{(1)}, X_t^{(2)}, \dots, X_t^{(M)}\}$$

sendo que cada partícula é representada por uma hipótese para a última pose percorrida associada a uma hipótese de mapa¹.

$$X_t^{(i)} = \left(s_t^{(i)} \quad Bel_t^{(i)}(\theta_1), \quad \dots \quad Bel_t^{(i)}(\theta_{|\Theta|}) \right)$$

Em cada passo, para cada partículas $X_{t-1}^{(i)}$, primeiramente são geradas K hipóteses de pose $\{s_t^{(i,1)}, \dots, s_t^{(i,K)}\}$ segundo o modelo de locomoção do robô, das quais escolhe-se uma com base em sua verossimilhança para representar a nova pose. Em seguida calcula-se o peso associado à partícula e atualiza-se a distribuição do mapa associado, então as partículas são reamostradas de acordo com o peso atribuído.

Para realizar a associação de dados foi desenvolvida uma heurística que busca a melhor associação para cada observação, mantendo a restrição de unicidade e descartando algumas observações mais ambíguas.

A Figura 5.1 demonstra as conexões entre os diversos módulos do algoritmo. Todos os módulos foram desenvolvidos de forma executar na GPU. Cada um foi implementado com uma sequência de *kernels* sendo a CPU utilizada apenas para sincronização.

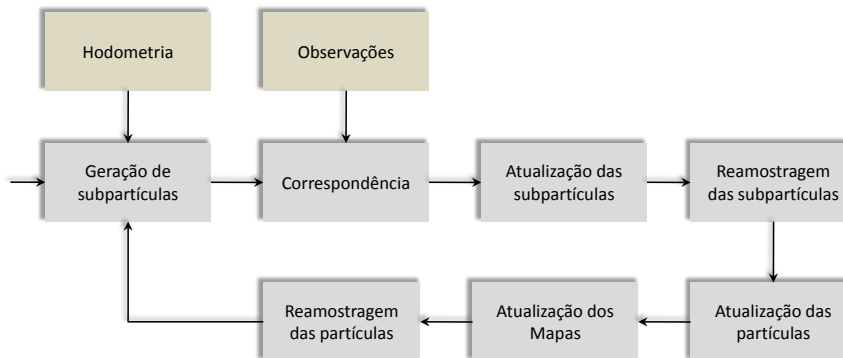


Figura 5.1. Ciclo de execução do algoritmo desenvolvido.

As seções seguintes apresentarão com mais detalhes cada um dos módulos desenvolvidos. Para auxiliar o desenvolvimento das equações serão definidas as seguintes quantidades:

1. $S_t^{(i)}$ é a hipótese de caminho percorrido pela i -ésima partícula até o instante t

$$S_t^{(i)} = \{s_1^{(i)}, s_2^{(i)}, \dots, s_t^{(i)}\};$$

¹Apesar de o estado de crença ser estimado sobre todo o caminho percorrido pelo robô apenas a informação da última pose é necessária de maneira que as poses anteriores podem ser desconsideradas [Montemerlo, 2003]

2. $Bel_t^{(i)}(\Theta)$ é o estado de crença no mapa relacionado à i -ésima partícula

$$\begin{aligned} Bel_t^{(i)}(\Theta) &= p(\Theta | S_t^{(i)}, a^t, \mathbf{O}_t, \nu) \\ &= \prod_{\theta_i \in \Theta} p(\theta_i | S_t^{(i)}, a^t, \mathbf{O}_t, \nu) \\ &= \{Bel_t^{(i)}(\theta_1), \dots, Bel_t^{(i)}(\theta_{|\Theta|})\}, \end{aligned}$$

3. $Bel_t^{(i)}(\theta_j)$ é o estado de crença do j -ésimo marco relacionado à i -ésima partícula

$$Bel_t^{(i)}(\theta_j) = p(\theta_j | S_t^{(i)}, a^t, \mathbf{O}_t, \nu).$$

5.1 Atualização dos Mapas

Essa seção apresentará a metodologia utilizada para estimação da crença no mapa. Para cada partícula uma distribuição distinta condicionada à hipótese de caminho percorrido é estimada para o mapa. A independência da crença de cada marco condicionada ao conhecimento do caminho permite que a estimação do mapa seja decomposta em um problema de estimação separado para cada marco.

O teorema do limite central afirma que a distribuição amostral da média de um conjunto de variáveis aleatórias independentes e identicamente distribuídas se aproxima-se cada vez mais de uma normal quando o tamanho da amostra cresce. Como cada marco do ambiente é considerado estático o teorema do limite central é um indicativo de que a crença nos marcos pode ser aproximada por uma distribuição normal dada uma quantidade suficiente de observações independentes. Dessa forma optou-se por representar o estado de crença de cada marco θ_i possuindo uma quantidade suficiente de observações por uma normal, de forma paramétrica por meio de sua média μ_{θ_i} e pela matriz de covariância Σ_{θ_i} . No entanto, marcos com apenas uma observação possuem incerteza infinita no eixo da profundidade, já que toda a informação acerca da profundidade do marco é perdida quando da projeção no plano de imagem. Sendo assim, a representação por meio de uma distribuição normal não degenerada não é possível. Além disso, se a quantidade de observações for pequena e com pouca paralaxe, a aproximação da distribuição por uma normal trará muita perda de informações.

Devido a esse problema optou-se por utilizar outra representação para o estado de crença dos marcos observados apenas um ou poucas vezes. Posteriormente, quando o estado de crença puder ser aproximado por uma normal sem que haja muita perda de informação, passa-se a utilizar a primeira representação mencionada. Essa técnica descrita por Leonard and Rikoski [2000] é denominada inicialização atrasada.

5.1.1 Inicialização Atrasada

Nesse trabalho optou-se por utilizar duas formas de representação para os marcos não inicializados. Quando da primeira observação o_j do marco θ_{ν_j} , representa-se o estado de crença por um ponto e uma direção contendo uma pequena incerteza, sendo a incerteza no eixo dessa direção infinita. O ponto é o ponto focal $f_t^{(i)}$ da câmera quando da observação e a crença na direção é representada por uma normal $\mathcal{N}(\tilde{\mu}_{\nu_j}^{(i)}, \tilde{\Sigma}_{\nu_j}^{(i)})$ bidimensional para a projeção do marco na imagem. Esse modelo de representação é descrito pela Equação 5.1.

$$Bel_t^{(i)}(\theta_{\nu_i}) = \left(f_{\tau_i}^{(i)} \quad \tilde{\mu}_{\nu_j}^{(i)} \quad \tilde{\Sigma}_{\nu_j}^{(i)} \right) \quad (5.1)$$

A média $\tilde{\mu}_{\nu_j}^{(i)}$ é primeiramente fornecida pela observação o_j e a covariância $\tilde{\Sigma}_{\nu_j}^{(i)}$ é fornecida pelo modelo de observação. A crença no eixo da direção é considerada uniforme pelo princípio da razão insuficiente². A Figura 5.2 demonstra o intervalo de confiança de um marco com apenas uma observação.

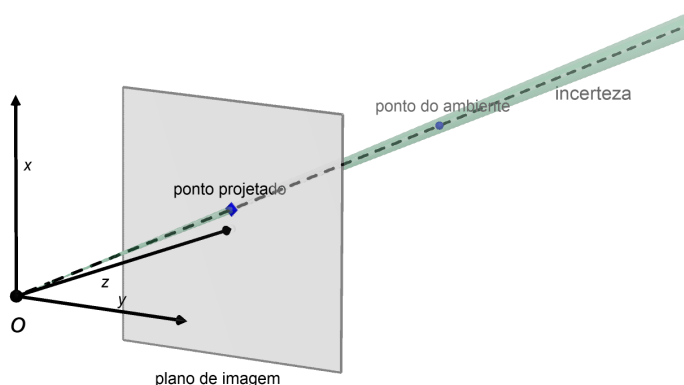


Figura 5.2. Modelo de observação e intervalo de confiança.

Quando o mesmo marco θ_i é observado de dois pontos de vista diferentes com uma paralaxe mensurável, a informação a respeito da profundidade passa a existir de modo que a representação inicial, que não representa a informação de profundidade, passa a ser insuficiente. Porém a crença “real” do marco pode não ser, ainda, bem aproximada por uma normal. Dessa forma, optou-se nesse trabalho por utilizar uma

²O princípio da razão insuficiente afirma que não havendo qualquer informação a respeito de determinada quantidade não há razão suficiente para crer que um valor seja mais provável do que outro, sendo assim a crença a respeito do valor dessa quantidade deve ser representada por uma distribuição uniforme.

mistura ponderada de distribuições normais para representar o estado de crença nesse estágio. A Equação 5.2 demonstra essa representação para o estado de crença.

$$\begin{aligned}
 Bel_t^{(i)}(\theta_j) &= \left\{ \left(\begin{array}{c} w_{t,\theta_j}^{1,(i)} \\ \mu_{t,\theta_j}^{1,(i)} \\ \Sigma_{t,\theta_j}^{1,(i)} \end{array} \right), \left(\begin{array}{c} w_{t,\theta_j}^{2,(i)} \\ \mu_{t,\theta_j}^{2,(i)} \\ \Sigma_{t,\theta_j}^{2,(i)} \end{array} \right), \dots, \left(\begin{array}{c} w_{t,\theta_j}^{K,(i)} \\ \mu_{t,\theta_j}^{K,(i)} \\ \Sigma_{t,\theta_j}^{K,(i)} \end{array} \right) \right\} \\
 &= \sum_{n=1}^K (w_{t,\theta_j}^{n,(i)}) \times \mathcal{N}(\mu_{t,\theta_j}^{n,(i)}, \Sigma_{t,\theta_j}^{n,(i)})
 \end{aligned} \tag{5.2}$$

Dado $Bel_{t-1}^{(i)}(\theta_j)$ representado segundo o Modelo 5.1 e uma observação $o_l \in \mathbf{O}_t$ tal que $\nu_l = j$, a passagem da representação dada pela Equação 5.1 à descrita pela Equação 5.2 é feita da forma descrita a seguir.

Como apenas uma observação o_k do marco θ_j foi obtida até o momento, o estado de crença $Bel_{t-1}^{(i)}(\theta_j)$ pode ser escrito como

$$Bel_{t-1}^{(i)}(\theta_j) = p(\theta_j | o_k, \mathbf{S}_{t-1}^{(i)}, a^{t-1}).$$

Desejamos que a nova crença aproxime

$$p(\theta_j | o_k, o_l, \mathbf{S}_t^{(i)}, a^t) \propto p(o_l | \theta_j, o_k, \mathbf{S}_t^{(i)}, a^t) p(\theta_j | o_k, \mathbf{S}_t^{(i)}, a^t).$$

Pela propriedade de Markov sabe-se que o_l depende apenas do marco θ_j e da pose s_t no tempo t . Assim, o primeiro fator se reduz ao modelo de observação

$$p(\theta_j | o_k, o_l, \mathbf{S}_t^{(i)}, a^t) \propto p(o_l | \theta_j, s_t^{(i)}) p(\theta_j | o_k, \mathbf{S}_t^{(i)}, a^t).$$

Como θ_j depende de s_t somente por meio da observação o_l , s_t pode ser removido do segundo fator levando à

$$p(\theta_j | o_k, o_l, \mathbf{S}_t^{(i)}, a^t) \propto p(o_l | \theta_j, s_t^{(i)}) Bel_{t-1}^{(i)}(\theta_j).$$

A nova crença, portanto, deve aproximar o produto da verossimilhança da observação com o estado de crença anterior.

Isso é realizado da seguinte forma:

1. Geram-se K hipóteses $\{o_l^{(1)}, \dots, o_l^{(K)}\}$ para o_l segundo o modelo de observação $o_l^{(n)} \sim p(o_l | \theta_j, s_t^{(i)})$.

2. Calcula-se a linha $E_l = o_t + v_l \cdot \alpha$ epipolar no plano da imagem obtida em t assumindo as poses $s_{\tau_k}^{(i)}$ e $s_t^{(i)}$ para o robô quando da obtenção das observações.
3. Calcula-se o desvio σ_l a com base na covariância de $p(o_l|\theta_j, s_t^{(i)})$ na direção da linha epipolar obtida no passo anterior.
4. De $Bel_{t-1}^{(i)}(\theta_j)$ compute os cinco pontos sigma $\sigma_k^0 \dots \sigma_k^4$ da distribuição $\mathcal{N}(\tilde{\mu}_{\nu_j}^{(i)}, \tilde{\Sigma}_{\nu_j}^{(i)})$
5. Para cada $o_l^{(n)}$ gerado calcula-se o valor $Max_{\theta_j}^{n,(i)}$ como sendo o valor mais verossímil de θ_j segundo $p(\theta_j|o_k, o_l^{(n)}, S_t^{(i)}, a^t)$ (Intuitivamente esse valor é dado pelo ponto no espaço que mais se aproxima das retas definidas pelas observações obtidas);
6. Calcula-se os pontos mais prováveis para θ_j da mesma maneira que no passo anterior segundo as seguintes distribuições:

$$\begin{aligned}\sigma_{\theta_j}^1 &= Max(\theta_j : p(\theta_j|\sigma_k^1, o_l^{(n)}, S_t^{(i)}, a^t)) \\ \sigma_{\theta_j}^2 &= Max(\theta_j : p(\theta_j|\sigma_k^2, o_l^{(n)}, S_t^{(i)}, a^t)) \\ \sigma_{\theta_j}^3 &= Max(\theta_j : p(\theta_j|\sigma_k^3, o_l^{(n)}, S_t^{(i)}, a^t)) \\ \sigma_{\theta_j}^4 &= Max(\theta_j : p(\theta_j|\sigma_k^4, o_l^{(n)}, S_t^{(i)}, a^t)) \\ \sigma_{\theta_j}^5 &= Max(\theta_j : p(\theta_j|o_k, o_l^{(n)} + v_l \cdot \sigma_l, S_t^{(i)}, a^t)) \\ \sigma_{\theta_j}^6 &= Max(\theta_j : p(\theta_j|o_k, o_l^{(n)} - v_l \cdot \sigma_l, S_t^{(i)}, a^t))\end{aligned}$$

7. para cada amostra n calculam-se $\mu_{\theta_j}^{n,(i)}$ e $\Sigma_{\theta_j}^{n,(i)}$ segundo:

$$\begin{aligned}\mu_{\theta_j}^{n,(i)} &= \frac{1}{2} Max_{\theta_j}^{n,(i)} + \frac{1}{6 \cdot 2} \sum_{i=1}^6 \sigma_{\theta_j}^i \\ \Sigma_{\theta_j}^{n,(i)} &= \frac{1}{2} [\mu_{\theta_j}^{n,(i)} - Max_{\theta_j}^{n,(i)}]^T [\mu_{\theta_j}^{n,(i)} - Max_{\theta_j}^{n,(i)}] + \frac{1}{6 \cdot 2} \sum_{i=1}^6 [\mu_{\theta_j}^{n,(i)} - \sigma_{\theta_j}^i]^T [\mu_{\theta_j}^{n,(i)} - \sigma_{\theta_j}^i]\end{aligned}$$

Colocando de forma mais intuitiva, o que o algoritmo faz é gerar hipóteses sobre a projeção mais recente do marco segundo o ruído do modelo de observação, obter a covariância do marco computando a incerteza nas direções paralela e ortogonal à linha epipolar por meio da transformada sigma da crença, e estimar a incerteza no eixo da profundidade por meio da incerteza do modelo de observação na direção da linha epipolar dada nova observação.

5.1.2 Atualização dos marcos inicializados

O teorema do limite central assegura que, após serem obtidas suficientes observações independentes de um marco θ_j , sua crença se aproxima de uma normal. Sendo assim, passa-se a utilizar esse tipo de distribuição representada por meio de seus parâmetros para representar a crença do marco quando a divergência entre a nova distribuição e antiga representação por meio de uma mistura de gaussianas se torna pequena o suficiente. A divergência entre as representações é estimada por meio da divergência de Kullback-Leibler [Kullback and Leibler, 1951] que é computada para cada marco na representação dada pela Equação 5.2 a cada observação. A partir dessa aproximação passa-se a se referir ao marco θ_j como inicializado.

$$\begin{aligned} Bel_t^{(i)}(\theta_j) &= \left(\mu_{t,\theta_j}^{(i)} \quad \Sigma_{t,\theta_j}^{(i)} \right) \\ &= \mathcal{N}(\mu_{t,\theta_j}^{(i)}, \Sigma_{t,\theta_j}^{(i)}). \end{aligned} \quad (5.3)$$

A atualização das crenças de marcos inicializados é realizada por meio de um UKF, considerando-se somente a etapa de atualização dado que os marcos são estáticos.

O modelo de observação é uma projeção perspectiva com erro normal constante

$$o_j = P(\theta_{\nu_j}, s_t) + r_t.$$

Seja $o_j \in \mathbf{O}_t$ uma observação correspondente ao marco θ_{ν_j} e sejam $\mu_{t-1,\theta_{\nu_j}}^{(i)}$ e $\Sigma_{t-1,\theta_j}^{(i)}$ respectivamente a posição esperada e a matriz covariância do estado de crença no instante $t - 1$ correspondentes a i -ésima partícula. A atualização da crença é realizada do seguinte modo:

Primeiramente computam-se os pontos sigma de θ_{ν_j}

$$\bar{\sigma}_{t-1,\nu_j}^{(i)} = \begin{cases} \mu_{t-1,\theta_{\nu_j}}^{(i)} & i = 0 \\ \mu_{t-1,\theta_{\nu_j}}^{(i)} + [\sqrt{(n + \lambda)\Sigma_{t-1,\theta_j}^{(i)}}]_i & i = 1 \dots 3 \\ \mu_{t-1,\theta_{\nu_j}}^{(i)} - [\sqrt{(n + \lambda)\Sigma_{t-1,\theta_j}^{(i)}}]_i & i = 4 \dots 6 \end{cases}$$

calculam-se, então, (i) o valor esperado \bar{o}_j da observação dada a crença em θ_{ν_j} no instante anterior, (ii) sua matriz de covariância \bar{H}_j e (iv) o ganho de Kalman K_j .

$$\begin{aligned}
\text{(i)} \quad \bar{o}_j &= \sum_{k=0}^6 \omega_m^{(k)} \cdot P(\bar{\sigma}_{t-1}^{(k)}, s_t^{(i)}) \\
\text{(ii)} \quad \bar{H}_j &= \sum_{k=0}^6 \omega_c^{(k)} \cdot [\bar{o}_j - P(\bar{\sigma}_{t-1}^{(k)}, s_t^{(i)})] \times [\bar{o}_j - P(\bar{\sigma}_t^{(k)}, s_t^{(i)})]^T + R_t \\
\text{(iii)} \quad L_j &= \sum_{k=0}^6 \omega_c^{(k)} \cdot [\mu_{t-1, \theta_{\nu_j}}^{(i)} - \bar{\sigma}_t^{(k)}] \times [\bar{o}_j - P(\bar{\sigma}_t^{(k)}, s_t^{(i)})]^T \\
\text{(iv)} \quad K_j &= L_j \bar{H}_j^{-1}
\end{aligned}$$

onde $s_t^{(i)}$ é a pose associada à i -ésima partícula no instante t e os pesos ω são definido conforme a Equação 3.1

Concluí-se a atualização computando $\mu_{t, \theta_{\nu_j}}^{(i)}$ e $\Sigma_{t, \theta_{\nu_j}}^{(i)}$ como mostrado a seguir:

$$\begin{aligned}
\mu_{t, \theta_{\nu_j}}^{(i)} &= \mu_{t-1, \theta_{\nu_j}}^{(i)} + K_j \times (o_j - \bar{o}_j) \\
\Sigma_{t, \theta_{\nu_j}}^{(i)} &= \Sigma_{t-1, \theta_{\nu_j}}^{(i)} - K_j \bar{H}_j K_j^T
\end{aligned}$$

5.1.3 Atualização dos marcos não inicializados

Marcos não inicializados têm sua crença representada por uma mistura de normais:

$$\begin{aligned}
Bel_t^{(i)}(\theta_j) &= p(\theta_j | S_t^{(i)}, \mathbf{O}^t, a^t, \nu) \\
&= \sum_{n=1}^K \left((w_{t, \theta_j}^{n, (i)}) \mathcal{N}(\mu_{t, \theta_j}^{n, (i)}, \Sigma_{t, \theta_j}^{n, (i)}) \right) \\
&= \sum_{n=1}^K \left((w_{t, \theta_j}^{n, (i)}) p_n(\theta_j | S_t^{(i)}, \mathbf{O}^t, a^t, \nu) \right).
\end{aligned}$$

Seja $o_l \in \mathbf{O}_t$ uma observação tal que $\nu_l = j$. Dada a unicidade da observação o_l sabe-se que θ_j é independente de qualquer observação em $\mathbf{O}_t - o_l$ reduzindo-se a crença a

$$Bel_t^{(i)}(\theta_j) = p(\theta_j | o_l, S_t^{(i)}, \mathbf{O}^{t-1}, a^t, \nu)$$

que, conforme já demonstrado no Capítulo 3, pode ser fatorada em:

$$Bel_t^{(i)}(\theta_j) = \eta p(o_l | \theta_j, s_t^{(i)}, \nu) Bel_{t-1}^{(i)}(\theta_j).$$

Como a crença $Bel_t^{(i)}(\theta_j)$ deve ser aproximada por uma normal pode-se expandi-la juntamente com $Bel_{t-1}^{(i)}(\theta_j)$ para obter:

$$\begin{aligned}
& \sum_{n=1}^K \left[(w_{t,\theta_j}^{n,(i)}) \mathcal{N}(\mu_{t,\theta_j}^{n,(i)}, \Sigma_{t,\theta_j}^{n,(i)}) \right] \\
& \approx \eta p(o_l | \theta_j, s_t^{(i)}, \nu) \sum_{n=1}^K \left[(w_{t-1,\theta_j}^{n,(i)}) \mathcal{N}(\mu_{t-1,\theta_j}^{n,(i)}, \Sigma_{t-1,\theta_j}^{n,(i)}) \right] \\
& \approx \eta \sum_{n=1}^K \left[(w_{t-1,\theta_j}^{n,(i)}) p(o_l | \theta_j, s_t^{(i)}, \nu) \mathcal{N}(\mu_{t-1,\theta_j}^{n,(i)}, \Sigma_{t-1,\theta_j}^{n,(i)}) \right].
\end{aligned} \tag{5.4}$$

Cada nova média $\mu_{t,\theta_j}^{n,(i)}$ e matriz de covariância $\Sigma_{t,\theta_j}^{n,(i)}$ pode ser obtida utilizando um UKF para atualizar cada um das médias $\mu_{t-1,\theta_j}^{n,(i)}$ e matrizes de covariância $\Sigma_{t-1,\theta_j}^{n,(i)}$ da crença no instante $t-1$ de maneira semelhante à realizada para os marcos inicializados:

$$\begin{aligned}
\bar{\sigma}_{t,j}^{(k)} | n &= \begin{cases} \mu_{t-1,\theta_j}^{n,(i)} & i = 0 \\ \mu_{t-1,\theta_j}^{n,(i)} + [\sqrt{n\Sigma_{t-1,\theta_j}^{n,(i)}}]_k & k = 1 \dots 3 \\ \mu_{t-1,\theta_j}^{n,(i)} - [\sqrt{n\Sigma_{t-1,\theta_j}^{n,(i)}}]_{k-3} & k = 4 \dots 6 \end{cases} \\
\bar{o}_l | n &= \sum_{k=0}^6 \omega^{(k)} \cdot P(\bar{\sigma}_t^{(k)} | n, s_t^{(i)}) \\
\bar{H}_l | n &= \sum_{k=0}^6 \omega^{(k)} \cdot [\bar{o}_l | n - P(\bar{\sigma}_t^{(k)} | n, s_t^{(i)})]^T \times [\bar{o}_l | n - P(\bar{\sigma}_t^{(k)} | n, s_t^{(i)})] + R_t \\
L_l | n &= \sum_{k=0}^6 \omega^{(k)} \cdot [\mu_{t-1,\theta_j}^{n,(i)} - \bar{\sigma}_{t,j}^{(k)} | n] \times [\bar{o}_l | n - P(\bar{\sigma}_{t,j}^{(k)} | n, s_t^{(i)})]^T \\
K_l | n &= L_l | n (\bar{H}_l | n)^{-1} \\
\mu_{t,\theta_j}^{n,(i)} &= \mu_{t-1,\theta_j}^{n,(i)} + (K_l | n)(o_l - \bar{o}_l | n) \\
\Sigma_{t,\theta_j}^{n,(i)} &= \Sigma_{t-1,\theta_j}^{n,(i)} - (K_l | n)(\bar{H}_l | n)(K_l | n)^T.
\end{aligned}$$

As distribuições geradas $\mathcal{N}(\mu_{t,\theta_j}^{n,(i)}, \Sigma_{t,\theta_j}^{n,(i)})$ são aproximadamente as distribuições anteriores atualizadas pelo teorema de Bayes:

$$\mathcal{N}(\mu_{t,\theta_j}^{n,(i)}, \Sigma_{t,\theta_j}^{n,(i)}) \approx \frac{p(o_l | \theta_j, s_t^{(i)}, \nu) \mathcal{N}(\mu_{t-1,\theta_j}^{n,(i)})}{\int_{\theta_j} p(o_l | \theta_j, s_t^{(i)}, \nu) \mathcal{N}(\mu_{t-1,\theta_j}^{n,(i)}) d\theta_j}.$$

Pode-se mostrar também que a verossimilhança de cada observação dada a crença no mapa pode ser aproximada pela probabilidade da inovação [Simon, 2006], definida na Equação 5.5.

$$\begin{aligned}
& \int_{\theta_j} p(o_l | \theta_j, s_t^{(i)}, \nu) \mathcal{N}(\mu_{t-1, \theta_j}^{n, (i)}) d\theta_j \\
& \approx \frac{1}{2\pi \sqrt{|\overline{H}_l|_n}} e^{-\frac{1}{2} \left[(o_l - \bar{o}_l|_n)^T (\overline{H}_l|_n)^{-1} (o_l - \bar{o}_l|_n) \right]} \\
& \approx \mathcal{N}(o_l : \bar{o}_l|_n, \overline{H}_l|_n).
\end{aligned} \tag{5.5}$$

Substituindo esse resultado na equação 5.4 e resolvendo para $w_{t, \theta_j}^{n, (i)}$ obtém-se:

$$\begin{aligned}
& \sum_{n=1}^K \left((w_{t, \theta_j}^{n, (i)}) \mathcal{N}(\mu_{t, \theta_j}^{n, (i)}, \Sigma_{t, \theta_j}^{n, (i)}) \right) \\
& \approx \sum_{n=1}^K \left[\eta \mathcal{N}(o_l : \bar{o}_l|_n, \overline{H}_l|_n) (w_{t-1, \theta_j}^{n, (i)}) \mathcal{N}(\mu_{t, \theta_j}^{n, (i)}, \Sigma_{t, \theta_j}^{n, (i)}) \right] \\
& \Rightarrow (w_{t, \theta_j}^{n, (i)}) = \eta [(w_{t-1, \theta_j}^{n, (i)}) \mathcal{N}(o_l : \bar{o}_l|_n, \overline{H}_l|_n)] \\
& = \frac{(w_{t-1, \theta_j}^{n, (i)}) \mathcal{N}(o_l : \bar{o}_l|_n, \overline{H}_l|_n)}{\int_{\theta_j} \left[\sum_{n=1}^K (w_{t-1, \theta_j}^{n, (i)}) \mathcal{N}(o_l : \bar{o}_l|_n, \overline{H}_l|_n) \mathcal{N}(\mu_{t, \theta_j}^{n, (i)}, \Sigma_{t, \theta_j}^{n, (i)}) \right] d\theta_j} \\
& = \frac{(w_{t-1, \theta_j}^{n, (i)}) \mathcal{N}(o_l : \bar{o}_l|_n, \overline{H}_l|_n)}{\sum_{n=1}^K \left[(w_{t-1, \theta_j}^{n, (i)}) \mathcal{N}(o_l : \bar{o}_l|_n, \overline{H}_l|_n) \int_{\theta_j} \mathcal{N}(\mu_{t, \theta_j}^{n, (i)}, \Sigma_{t, \theta_j}^{n, (i)}) d\theta_j \right]} \\
& = \frac{(w_{t-1, \theta_j}^{n, (i)}) \mathcal{N}(o_l : \bar{o}_l|_n, \overline{H}_l|_n)}{\sum_{n=1}^K \left[(w_{t-1, \theta_j}^{n, (i)}) \mathcal{N}(o_l : \bar{o}_l|_n, \overline{H}_l|_n) \right]}.
\end{aligned}$$

Assim, para atualizar as crenças para os marcos não inicializados, primeiramente atualiza-se cada uma das distribuições na mistura por meio de UKF e em seguida computam-se os novos pesos $w_{t, \theta_j}^{n, (i)}$ como o produto normalizado do n -ésimo peso $w_{t-1, \theta_j}^{n, (i)}$ no instante anterior pela verossimilhança da observação segundo a n -ésima distribuição normal.

$$w_{t, \theta_j}^{n, (i)} = \frac{w_{t-1, \theta_j}^{n, (i)} \cdot \mathcal{N}(o_l : \bar{o}_l|_n, \overline{H}_l|_n)}{\sum_{n=1}^K \left[w_{t-1, \theta_j}^{n, (i)} \cdot \mathcal{N}(o_l : \bar{o}_l|_n, \overline{H}_l|_n) \right]}. \tag{5.6}$$

5.2 Predição da Pose

Algoritmos de localização por filtro de partículas usualmente geram amostras das novas poses baseando-se somente no modelo de locomoção do robô e nas amostras do instante anterior

$$s_t^{(i)} \sim p(s_t | s_{t-1}^{(i)}, a_t).$$

Como mencionado o algoritmo proposto nesse trabalho gera novas hipóteses a partir de uma distribuição que incorpora explicitamente as observações \mathbf{O}_t obtidas no instante t assim como no algoritmo FastSLAM 2.0. A Equação 5.7 descreve essa distribuição.

$$s_t^{(i)} \sim p(s_t | \mathbf{S}_{t-1}^{(i)}, \mathbf{O}^t, a^t, \nu). \quad (5.7)$$

A distribuição descrita na Equação 5.7 pode ser fatorada utilizando o teorema de Bayes em

$$p(s_t | \mathbf{S}_{t-1}^{(i)}, \mathbf{O}^t, a^t, \nu) \propto p(\mathbf{O}_t | s_t, \mathbf{S}_{t-1}^{(i)}, a^t, \mathbf{O}^{t-1}, \nu) p(s_t | \mathbf{S}_{t-1}^{(i)}, \mathbf{O}^{t-1}, a^t, \nu).$$

O segundo fator do lado direito da equação pode ser reduzido ao modelo de localização pela propriedade de Markov para obter:

$$p(s_t | \mathbf{S}_{t-1}^{(i)}, \mathbf{O}^t, a^t, \nu) \propto p(\mathbf{O}_t | s_t, \mathbf{S}_{t-1}^{(i)}, a^t, \mathbf{O}^{t-1}, \nu) p(s_t | s_{t-1}^{(i)}, a_t).$$

Usando o teorema da probabilidade total para condicionar a observação no primeiro fator à direita ao mapa Θ obtém-se:

$$p(s_t | \mathbf{S}_{t-1}^{(i)}, \mathbf{O}^t, a^t, \nu) \propto \int_{\Theta} p(\mathbf{O}_t | \Theta, s_t, \mathbf{S}_{t-1}^{(i)}, a^t, \mathbf{O}^{t-1}, \nu) p(\Theta | s_t, \mathbf{S}_{t-1}^{(i)}, a^t, \mathbf{O}^{t-1}, \nu) d\Theta \times p(s_t | s_{t-1}^{(i)}, a_t)$$

O primeiro fator no interior da integral pode ser reduzido ao modelo de observação pela propriedade de Markov:

$$p(s_t | \mathbf{S}_{t-1}^{(i)}, \mathbf{O}^t, a^t, \nu) \propto \int_{\Theta} p(\mathbf{O}_t | \Theta, s_t, \nu) p(\Theta | s_t, \mathbf{S}_{t-1}^{(i)}, a^t, \mathbf{O}^{t-1}, \nu) d\Theta \times p(s_t | s_{t-1}^{(i)}, a_t).$$

E sem que haja uma observação no tempo t a pose s_t e ação a_t não podem trazer

qualquer informação sobre o mapa. Dessa forma, eles podem ser removidos do segundo fator da integral para obter a Equação 5.8

$$\begin{aligned}
 p(s_t | S_{t-1}^{(i)}, \mathbf{O}^t, a^t, \nu) &\propto \int_{\Theta} p(\mathbf{O}_t | \Theta, s_t, \nu) p(\Theta | S_{t-1}^{(i)}, a^{t-1}, \mathbf{O}^{t-1}, \nu) d\Theta \times p(s_t | s_{t-1}^{(i)}, a_t) \\
 &\propto \int_{\Theta} p(\mathbf{O}_t | \Theta, s_t, \nu) Bel_{t-1}^{(i)}(\Theta) d\Theta \times p(s_t | s_{t-1}^{(i)}, a_t)
 \end{aligned}
 \tag{5.8}$$

A nova distribuição proposta, descrita pela Equação 5.8, é igual ao produto da distribuição gerada segundo o modelo de locomoção do robô pela verossimilhança das últimas observações \mathbf{O}_t obtidas, dado o estado de crença do mapa no instante de tempo anterior.

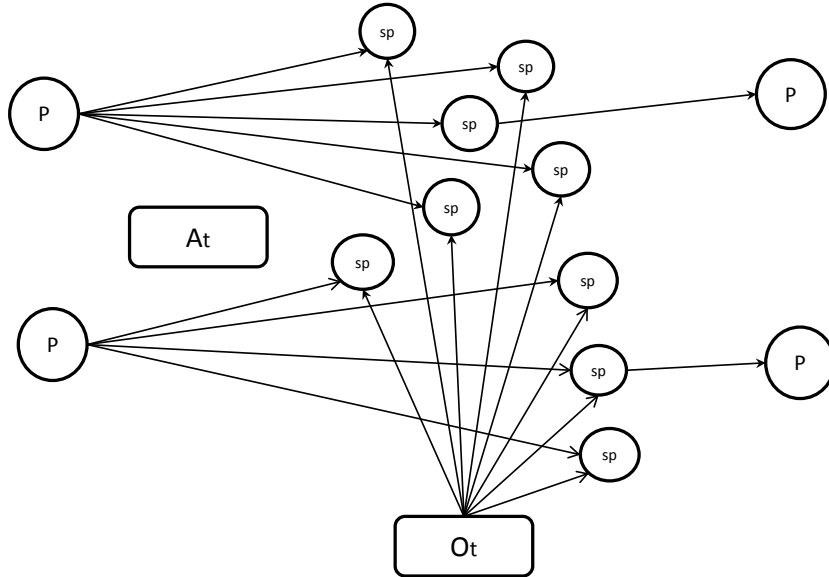


Figura 5.3. Geração e atualização das subpartículas.

A distribuição descrita na Equação 5.8 é idêntica à utilizada no FastSLAM 2.0 [Montemerlo, 2003]. Entretanto, para calcular as novas amostras os autores supuseram que os ruídos envolvidos fossem aproximadamente normais e que o modelo de observação fosse aproximadamente linear. Além disso, o algoritmo proposto considera que em cada instante apenas uma observação (o_t) é obtida e a incorporação de mais de uma observação pode ser processada sequencialmente.

O algoritmo de predição desenvolvido nesse trabalho realiza a incorporação da nova observação utilizando um filtro de partículas auxiliar [Pitt and Shephard, 1999] que é executado por cada partícula. Optou-se por denominar as partículas auxiliares

utilizadas no filtro de sub-partículas para diferenciá-las das hipóteses mais gerais. Por ser baseado em filtro de partículas ele pode se adaptar a qualquer tipo de modelo ou distribuição de probabilidades paramétrica ou não.

A duas subseções seguintes mostram como a etapa de predição é realizada pelo FastSLAM 2.0, e pelo filtro auxiliar respectivamente.

5.2.1 FastSLAM 2.0

O FastSLAM 2.0 assume que os modelos de observação e locomoção podem ser descritos, de forma aproximada, da forma descrita na Equação 5.9

$$\begin{aligned}
 o_t &= g(s_t, \theta_{\nu_t}) + \varepsilon_t \\
 s_t &= h(s_{t-1}, a_t) + \delta_t \\
 \varepsilon_t &\sim \mathcal{N}(0, R_t) \\
 \delta_t &\sim \mathcal{N}(0, P_t).
 \end{aligned} \tag{5.9}$$

O FastSLAM 2.0 assume que a cada instante t obtém-se uma única observação o_t , para gerar novas amostras, cada pose $s_{t-1}^{(i)}$ é propagada pelo modelo de locomoção e atualizada por meio de um EKF, da maneira que será descrita a seguir.

$$\begin{aligned}
 \bar{s}_t &= h(s_{t-1}^{(i)}, a_t) \\
 \bar{\Sigma}_{s_t} &= P_t \\
 \bar{\theta}_{\nu_t} &= E_{Bel_{t-1}^{(i)}(\theta_{\nu_t})}[\theta_{\nu_t}] \\
 \bar{\Sigma}_{\theta_{\nu_t}} &= E_{Bel_{t-1}^{(i)}(\theta_{\nu_t})}[(\theta_{\nu_t} - \bar{\theta}_{\nu_t})^t \times (\theta_{\nu_t} - \bar{\theta}_{\nu_t})] \\
 \bar{o}_t &= g(\bar{s}_t, \bar{\theta}_{\nu_t}) \\
 G_{(\theta_{\nu_t})} &= \nabla_{(\theta_{\nu_t})} g(s_t, \theta_{\nu_t})|_{s_t=\bar{s}_t, \theta_{\nu_t}=E[Bel_{t-1}^{(i)}(\theta_{\nu_t})]} \\
 G_{(s_t)} &= \nabla_{(s_t)} g(s_t, \theta_{\nu_t})|_{s_t=\bar{s}_t, \theta_{\nu_t}=\bar{\theta}_{\nu_t}}.
 \end{aligned}$$

$g(s_t, \theta_{\nu_t})$ pode ser aproximado pela do seguinte modo:

$$g(s_t, \theta_{\nu_t}) \approx o_t + G_{(\theta_{\nu_t})} \times (\theta_{\nu_t} - \bar{\theta}_{\nu_t}) + G_{(s_t)} \times (s_t - \bar{s}_t)$$

onde o_t é distribuído segundo

$$o_t \sim \mathcal{N}(o_t + G_{(s_t)} \times (s_t - \bar{s}_t), Z_t),$$

e Z_t é dado pela equação abaixo:

$$Z_t = G_{(\theta_{\nu_t})} \times \overline{\Sigma_{\theta_{\nu_t}}} G_{(\theta_{\nu_t})}^T + R_t.$$

Substituindo esse resultado na Equação 5.8 e resolvendo a convolução obtém-se a Equação 5.10

$$p(s_t | S_{t-1}^{(i)}, o^t, a^t, \nu) \propto e^{-\frac{1}{2}([\tilde{o}_t^T \times Z_t^{-1} \times \tilde{o}_t] + [\tilde{s}_t \times P_t^{-1} \times \tilde{s}_t])} \quad (5.10)$$

na qual \tilde{o}_t e \tilde{s}_t são definidos como:

$$\begin{aligned} \tilde{o}_t &= o_t - \bar{o}_t + G_{(s_t)} \times (s_t - \bar{s}_t) \\ \tilde{s}_t &= s_t - \bar{s}_t. \end{aligned}$$

A distribuição descrita na Equação 5.10 é calculada para cada observação o_t e dessa distribuição, a nova hipótese de pose é amostrada.

5.2.2 Filtro de sub-partículas

O algoritmo proposto incorpora a informação das observações \mathbf{O}_t obtidas no instante t utilizando um segundo filtro de partículas para cada partícula. Deseja-se que a hipótese de pose $s^{(i)}_t$ obedeça a distribuição descrita pela Equação 5.7. Como mostrado na Equação 5.8 essa distribuição é igual ao produto da distribuição gerada segundo o modelo de locomoção pela verossimilhança das observações \mathbf{O}_t , dado o estado de crença do mapa no instante de tempo anterior $Bel_{t-1}^{(i)}(\Theta)$

$$p(s_t | S_{t-1}^{(i)}, \mathbf{O}^t, a^t, \nu) \propto p(s_t | s_{t-1}^{(i)}, a_t) \times \int_{\Theta} p(\mathbf{O}_t | \Theta, s_t, \nu) Bel_{t-1}^{(i)}(\Theta) d\Theta.$$

A geração de nova pose é realizada em três passos: (i) Amostragem, (ii) Importância e (iii) Reamostragem, conforme será descrito a seguir.

(i) Amostragem :

Primeiramente geram-se K hipóteses a partir do modelo de locomoção, como mostrado na equação abaixo

$$\overline{Bel}^{(i)}(s_t^{(i)}) = \{s_t^{k,(i)} \sim p(s_t | s_{t-1}^{(i)}, a_t), k = 1 \dots K\}.$$

(ii) Importância :

Em seguida, a cada amostra $s_t^{k,(i)}$ deve ser atribuído um peso $w^{k,(i)}$ para compensar a diferença entre a distribuição gerada e a desejada. Assim esse peso deve ser

igual à:

$$w^{k,(i)} = \int_{\Theta} p(\mathbf{O}_t | \Theta, s_t^{k,(i)}, \nu) \text{Bel}_{t-1}^{(i)}(\Theta) d\Theta. \quad (5.11)$$

Pelo exposto na Seção 3.6, esse peso deve ser igual a

$$w^{k,(i)} = \prod_{o_j \in \mathbf{O}_t} \int_{\theta_{\nu_j}} p(o_j | \theta_{\nu_j}, s_t^{k,(i)}, \nu) \text{Bel}_{t-1}^{(i)}(\theta_{\nu_j}) d\theta_{\nu_j}, \quad (5.12)$$

ou seja o peso deve ser o produto da verossimilhança da observação o_k de cada marco θ_{ν_k} . Na Seção 5.1 mostrou-se como as verossimilhanças dos marcos inicializados podem ser aproximadas pela verossimilhança da inovação descrita pela Equação 5.5. Os pesos aqui são computados do mesmo modo.

Os marcos não inicializados representados conforme a Equação 5.2 possuem crença representada como uma mistura de gaussianas. A contribuição das observações desses marcos é obtida calculando-se um somatório da inovação de cada normal ponderada pelo peso dessa normal na mistura.

A verossimilhança dos marcos na representação inicial, dada pela Equação 5.1, é obtida a partir da restrição epipolar, mostrada na Figura 5.4. Para obtê-la computa-se a distância entre a linha epipolar formada pela projeção do vetor \vec{V}_1 no plano de imagem atual e o ponto observado, em seguida computa-se a verossimilhança dessa distância dado o ruído da câmera. O cálculo dessa distância

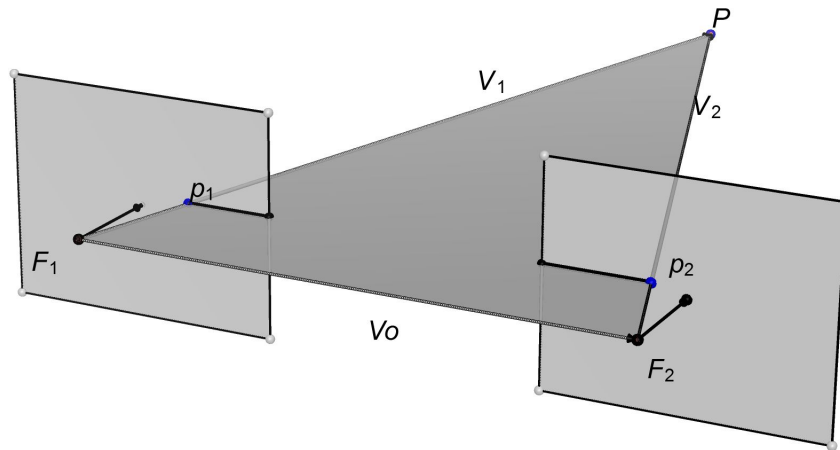


Figura 5.4. Restrição geométrica entre os pontos observados.

é realizado da seguinte maneira:

Primeiramente note que os vetores \vec{V}_1 , \vec{V}_2 e \vec{V}_o devem obedecer à seguinte

relação:

$$\vec{V}_1 = \vec{V}_2 + \vec{V}_o.$$

Dados os pontos p_1 e p_2 pode-se obter os vetores \vec{V}_1 e \vec{V}_2 a menos de uma constante cada e, dadas as hipóteses de pose $s_t^{k,(i)}$ e s_{τ_j} , pode-se calcular diretamente o vetor \vec{V}_o . Denominando os vetores $\overline{F_1 p_1} = \vec{v}_1$ e $\overline{F_2 p_2} = \vec{v}_2$ pode-se reescrever \vec{V}_1 e \vec{V}_2 como:

$$\vec{V}_1 = \lambda_1 \cdot \vec{v}_1 e$$

$$\vec{V}_2 = \lambda_2 \cdot \vec{v}_2.$$

Substituindo esses valores na equação original obtém-se:

$$\begin{aligned} \lambda_1 \cdot \vec{v}_1 &= \lambda_2 \cdot \vec{v}_2 + \vec{V}_o \\ \Rightarrow \vec{v}_1 &= \frac{\lambda_2}{\lambda_1} \cdot \vec{v}_2 + \frac{1}{\lambda_1} \vec{V}_o \end{aligned}$$

Tomando os produto internos $\langle \vec{v}_1 \cdot \vec{v}_2 \rangle$ e $\langle \vec{v}_1 \cdot \vec{V}_o \rangle$ chega-se ao sistema 5.13:

$$\begin{cases} \langle \vec{v}_1 \cdot \vec{v}_2 \rangle = \frac{\lambda_2}{\lambda_1} \cdot |\vec{v}_2|^2 + \frac{1}{\lambda_1} \langle \vec{V}_o \cdot \vec{v}_2 \rangle \\ \langle \vec{v}_1 \cdot \vec{V}_o \rangle = \frac{\lambda_2}{\lambda_1} \cdot \langle \vec{V}_o \cdot \vec{v}_2 \rangle + \frac{1}{\lambda_1} |\vec{V}_o|^2 \end{cases} \quad (5.13)$$

Resolvendo o sistema 5.13 chega-se à equação 5.14

$$\begin{cases} \frac{\lambda_2}{\lambda_1} = \frac{|\vec{V}_o|^2 \cdot \langle \vec{v}_1 \cdot \vec{v}_2 \rangle - \langle \vec{v}_1 \cdot \vec{V}_o \rangle \cdot \langle \vec{V}_o \cdot \vec{v}_2 \rangle}{|\vec{V}_o|^2 \cdot \langle \vec{v}_1 \cdot \vec{V}_o \rangle - \langle \vec{v}_1 \cdot \vec{v}_2 \rangle \cdot \langle \vec{V}_o \cdot \vec{v}_2 \rangle} \\ \frac{1}{\lambda_1} = \frac{|\vec{V}_o|^2 \cdot \langle \vec{v}_1 \cdot \vec{V}_o \rangle - \langle \vec{v}_1 \cdot \vec{v}_2 \rangle \cdot \langle \vec{V}_o \cdot \vec{v}_2 \rangle}{|\vec{V}_o|^2 \cdot \langle \vec{v}_1 \cdot \vec{V}_o \rangle - \langle \vec{v}_1 \cdot \vec{v}_2 \rangle \cdot \langle \vec{V}_o \cdot \vec{v}_2 \rangle} \end{cases} \quad (5.14)$$

A distância D_{epi} entre o ponto e a linha epipolar pode ser calculada tomando a norma do vetor $\vec{v}_1 - \frac{\lambda_2}{\lambda_1} \cdot \vec{v}_2 + \frac{1}{\lambda_1} \cdot \vec{V}_o$ como descrito abaixo:

$$D_{epi} = \left| \vec{v}_1 - \frac{\lambda_2}{\lambda_1} \cdot \vec{v}_2 + \frac{1}{\lambda_1} \cdot \vec{V}_o \right|.$$

Assim a contribuição da observação o_j referente um marco θ_{ν_j} que se encontra na representação dada pela Equação 5.1 é proporcional à

$$\frac{1}{\sqrt{2\pi\sigma_{epi}^2}} e^{-\frac{D_{epi}}{2\sigma_{epi}}}$$

onde, σ_{epi} é igual ao erro do modelo de observação adicionado ao erro da representação projetado na direção ortogonal à linha epipolar.

(iii) Reamostragem :

Após o peso das amostras ser calculado, uma amostra deve ser escolhida para representar a nova pose da partícula, com probabilidade proporcional ao peso atribuído. O Algoritmo 5 mostra como isso é feito para cada partícula $Bel^{(i)}(s_t, \Theta)$

Algoritmo 5 Reamostra-Subpartícula.

Reamostra-Subpartícula($\overline{Bel}^{(i)}(s_t), \{w^{1,(i)}, \dots, w^{K,(i)}\}$)

```

1:  $Sum^{(i)} \leftarrow 0$ 
2: for  $k = 1 \dots K$  do
3:    $Sum^{(i)} \leftarrow Sum^{(i)} + w^{k,(i)}$ 
4:    $W^{k,(i)} \leftarrow Sum^{(i)}$ . {Computa a soma de prefixos dos pesos}
5: end for
6:  $u \sim U(0, Sum^{(i)})$  {Gera um número aleatório com distribuição uniforme entre 0 a soma dos pesos}
7:  $min \leftarrow 0$ 
8:  $max \leftarrow K$ 
9: while ( $W_t^{min,(i)} < W_t^{max,(i)}$ ) do {Pesquisa binária}
10:   $mid \leftarrow \frac{max-min}{2}$ 
11:  if  $u < W^{mid,(i)}$  then
12:     $max \leftarrow mid$ 
13:  else
14:     $min \leftarrow mid$ 
15:  end if
16: end while
17: return  $s_t^{k,(i)}$ .

```

5.3 Atualização das Partículas

Objetiva-se que as hipóteses geradas para o caminho $S_t^{(i)}$ obedecem à distribuição “real” do estado de crença

$$Bel_t(S_t) = p(S_t | O^t, a^t, \nu).$$

Assumindo que as partículas obtidas no instante $t - 1$ sigam a distribuição $p(S_{t-1} | O^{t-1}, a^{t-1}, \nu)$, segue que as hipóteses $s_t^{(i)}$ propostas para a nova pose, geradas pelo método descrito na Seção 5.2, devem ser distribuídas segundo

$$s_t^{(i)} \sim p(s_t | S_{t-1}^{(i)}, \mathbf{O}^t, a^t, \nu) p(S_{t-1}^{(i)} | \mathbf{O}^{t-1}, a^{t-1}, \nu).$$

O peso atribuído a cada partícula deve ser igual à razão entre a distribuição proposta e a distribuição objetivada. Dessa forma, cada peso $w^{(i)}$ deve ser dado por:

$$w^{(i)} = \frac{p(S_t^{(i)} | O^t, a^t, \nu)}{p(s_t^{(i)} | S_{t-1}^{(i)}, \mathbf{O}^t, a^t, \nu) p(S_{t-1}^{(i)} | \mathbf{O}^{t-1}, a^{t-1}, \nu)}. \quad (5.15)$$

Expandindo o numerador do lado direito da Equação 5.15 pela definição de probabilidade condicional e aplicando o teorema de Bayes obtém-se:

$$\begin{aligned} w^{(i)} &= \frac{p(s_t^{(i)} | S_{t-1}^{(i)}, \mathbf{O}^t, a^t, \nu) p(S_{t-1}^{(i)} | \mathbf{O}^t, a^t, \nu)}{p(s_t^{(i)} | S_{t-1}^{(i)}, \mathbf{O}^t, a^t, \nu) p(S_{t-1}^{(i)} | \mathbf{O}^{t-1}, a^{t-1}, \nu)} \\ &\propto \frac{p(\mathbf{O}_t | S_{t-1}^{(i)}, \mathbf{O}^{t-1}, a^t, \nu) p(S_{t-1}^{(i)} | O^{t-1}, a^{t-1}, \nu)}{p(S_{t-1}^{(i)} | O^{t-1}, a^{t-1}, \nu)}. \end{aligned}$$

Na ausência da observação \mathbf{O}_t , a_t pode ser removido do fator à direita do numerador levando a:

$$\begin{aligned} w^{(i)} &\propto \frac{p(\mathbf{O}_t | S_{t-1}^{(i)}, \mathbf{O}^{t-1}, a^t, \nu) p(S_{t-1}^{(i)} | O^{t-1}, a^{t-1}, \nu)}{p(S_{t-1}^{(i)} | O^{t-1}, a^{t-1}, \nu)} \\ &\propto p(\mathbf{O}_t | S_{t-1}^{(i)}, \mathbf{O}^{t-1}, a^t, \nu). \end{aligned}$$

Utilizando o teorema da probabilidade total para condicionar a observação à s_t e Θ , obtém-se a Equação 5.16

$$\begin{aligned} w^{(i)} &\propto \int_{s_t} \int_{\Theta} p(\mathbf{O}_t | s_t, \Theta, S_{t-1}^{(i)}, \mathbf{O}^{t-1}, a^t, \nu) p(\Theta | s_t, S_{t-1}^{(i)}, \mathbf{O}^{t-1}, a^t, \nu) d\Theta p(s_t | S_{t-1}^{(i)}, a^t) ds_t \\ &\propto \int_{s_t} \int_{\Theta} p(\mathbf{O}_t | s_t, \Theta, \nu) p(\Theta | S_{t-1}^{(i)}, \mathbf{O}^{t-1}, a^{t-1}, \nu) d\Theta p(s_t | S_{t-1}^{(i)}, a^t) ds_t \end{aligned} \quad (5.16)$$

A integral em termos de s_t da Equação 5.16 pode ser aproximada pela média da integral mais interna avaliada para as K sub-partículas geradas no passo de predição. Isso nos leva à Equação 5.17

$$\begin{aligned} w^{(i)} &\propto \frac{1}{K} \sum_{k=1}^K \int_{\Theta} p(\mathbf{O}_t | s_t^{k,(i)}, \Theta, \nu) p(\Theta | S_{t-1}^{(i)}, \mathbf{O}^{t-1}, a^{t-1}, \nu) d\Theta. \\ &= \frac{1}{K} \sum_{k=1}^K \int_{\Theta} p(\mathbf{O}_t | s_t^{k,(i)}, \Theta, \nu) Bel_{t-1}^{(i)}(\Theta) d\Theta. \end{aligned} \quad (5.17)$$

Pode-se notar que a integral em termos de Θ é idêntica à definição dos pesos $w^{k,(i)}$ das subpartículas da etapa de predição dada pela Equação 5.11. Substituindo esse valor na Equação 5.17 obtém-se a Equação 5.18

$$w^{(i)} \propto \frac{1}{K} \sum_{k=1}^K w^{k,(i)}. \quad (5.18)$$

Ou seja, o peso $w^{(i)}$ de cada partícula deve ser proporcional à média dos pesos $w^{k,(i)}$ calculados para as sub-partículas K na etapa de predição.

5.4 Associação de dados

Até então foi assumido que a correspondência ν entre observações o_j e marcos θ_{ν_j} é dada. Porém, em geral essa associação não pode ser observada diretamente. Dessa forma, é necessário que essa associação seja estimada a cada instante t .

Diversas soluções propostas abordam essa questão por meio da escolha da hipótese de associação mais verossímil a cada instante. Entretanto, as ambiguidades inerentes ao problema fazem com que a possibilidade de uma associação incorreta seja significativa. Dessa forma, a estratégia adotada nesse trabalho consiste em gerar uma hipótese de associação para cada partícula. Essa mesma estratégia foi proposta por Montemerlo [2003] para solucionar o problema. A cada momento a correspondência $\nu^{(i)}$ mais verossímil é escolhida para cada partícula. O método proposto por Montemerlo [2003] no entanto, por questões principalmente de eficiência, ignora a restrição de unicidade o que aumenta significativamente a possibilidade de associações incorretas.

Gamallo et al. [2009] trabalharam com problema de SLAM visual por meio de câmeras omnidirecionais. Os autores utilizaram em sua solução o algoritmo FastSLAM 2.0, no entanto sua solução para a etapa de associação garante a unicidade da correspondência. A solução proposta, em cada passo, computa a verossimilhança de cada associação possível entre as observações e os marcos. Em seguida a solução ótima é computada por meio do algoritmo húngaro [Kuhn, 1955]. Esse algoritmo possui complexidade assintótica de $O(n^3)$ onde n é o tamanho da matriz de custos. Desse modo o desempenho da solução pode ficar prejudicado caso o tamanho do conjunto \mathbf{O}_t de observações seja grande.

Nesse trabalho desenvolveu-se uma heurística para solucionar o problema da associação. A heurística desenvolvida utiliza uma estratégia dual. Primeiramente são descartados aqueles marcos cujas crenças se projetam em regiões distantes da área observável do plano de imagem. Computa-se então para cada observação $o_j \in \mathbf{O}_t$ um

custo $c_{j,l}$ para a associação entre o_j e cada marco θ_l plausível. Após calculados os custos uma solução inicial, não necessariamente viável, é gerada tomando-se o menor custo $c_{j,l}$ para cada observação o_j .

A cada passo calculam-se custos Δ_j de se substituir a correspondência atual de cada observação o_j pela próxima melhor associação. Em seguida procura-se por pares (o_{j_1}, o_{j_2}) de observações conflitantes, ou seja que estejam associadas ao mesmo marco. Então, para cada par conflitante substitui-se a correspondência da observação que trará o menor aumento na função de custos.

Esses passos são repetidos para cada observação até que se obtenha uma solução viável, ou até que o número de trocas para determinada observação atinja um determinado limite. Nesse último caso a observação em questão é descartada e prossegue-se a execução com as observações remanescentes.

A solução proposta pode ser dividida em duas etapas principais: (i) Cálculo dos custos e (ii) Otimização. Essas etapas explicadas com maiores detalhes nas duas subseções seguintes.

5.4.1 Cálculo dos custos

A etapa de correspondência busca encontrar para cada partícula uma associação entre as observações $o_j \in \mathbf{O}_t$ e os marcos $\theta_l \in Bel^{(i)t-1}$ que constam nos mapas de cada partícula. Para isso o primeiro passo deve ser calcular um conjunto de custos $C^{(i)} = \{c_{j,l}\}$ de correspondência entre uma observação o_j um marco θ_l . Esses custos devem levar em consideração a verossimilhança de cada observação dada a crença na pose $s_t^{(i)}$ prevista para o robô e na posição do marco.

Diversos detectores de pontos salientes retornam, além da posição (x, y) do ponto no plano de imagem, um descritor d_j da vizinhança do ponto na imagem. Um descritor d_j é um vetor de tamanho fixo que resume a informação a respeito das variações das intensidades na vizinhança do ponto detectado. Desse modo, o custo da correspondência deve levar em consideração tanto a informação a respeito das restrições geométricas associadas, quanto a distância entre os descritores da observação e das observações anteriores do marco.

Para cada partícula, a correspondência deve ser escolhida a maximizar a verossimilhança da observação. Dessa forma, optou-se por utilizar uma soma da contribuição do par observação o_j , marco θ_l para o peso da partícula, caso esses sejam considerados correspondentes:

$$G_{j,l} \propto \frac{1}{K} \sum_{k=1}^K \int_{\theta_l} p(o_j | s_t^{k,(i)}, \theta_l, \nu_{j,l}) p(\theta_l | S_{t-1}^{(i)}, \mathbf{O}^{t-1}, a^{t-1}, \nu_{j,l}) d\theta_l \quad (5.19)$$

Para comparação dos descritores deve-se buscar um descritor d_l de uma das observações anteriores do marco. Como diferenças na orientação do robô podem ocasionar discrepâncias nos descritores, é desejável que essa observação tenha sido obtida de uma orientação semelhante à orientação atual do robô.

Para que essa observação possa ser encontrada rapidamente utiliza-se um vetor para cada marco θ_l , que pode conter de 1 a n descritores distintos, e onde cada elemento do vetor referente a um intervalo angular de $\frac{2\pi}{n}$ com respeito à orientação do robô no momento da observação. Dessa forma, dados uma pose e um marco, pode-se determinar imediatamente uma observação anterior obtida de uma pose semelhante. Após determinado um descritor d_l adequando para o marco, esse descritor é comparado com o descritor d_j tomando a distância euclidiana.

O custo associado à correspondência é obtido subtraindo-se da distância entre os descritores o logaritmo do custo descrito pela Equação 5.19. O custo $c_{j,l}$ resultante é descrito pela Equação 5.20.

$$c_{j,l} = \frac{|d_j - d_l|}{2\sigma_d} - \log G_{j,l}. \quad (5.20)$$

O vetor de descritores de cada marco θ_l , mencionado anteriormente, é preenchido conforme as correspondências são estimadas, do seguinte modo:

1. Cada índice do vetor contém a orientação do robô no momento da observação além de uma referência para o descritor relacionado.
2. Quando da primeira observação de um marco θ_l , atribuí-se à todas as posições $idx = 1 \dots n$ desse vetor a orientação estimada para robô, e uma referência para o descritor da observação correspondente.
3. Após determinada uma correspondência entre uma observação o_j e um marco θ_l , compara-se a orientação do robô com as orientações associadas à cada índice idx do vetor.
4. Se essa orientação for mais próxima do centro do intervalo de referência ($\frac{2\pi \cdot idx}{n}$) do que a orientação corrente, atualizam-se a referência para o descritor e a orientação do robô no índice.

Essa forma de preencher o vetor de descritores garante que cada posição do vetor contenha um descritor, e que a observação o_j que gerou esse descritor seja o mais próxima possível do intervalo angular de referência para cada índice.

Após calculados os custos de correspondências deve-se buscar uma correspondência de custo mínimo que obedeça à restrição de unicidade, a forma como essa correspondência é buscada é descrita na subseção seguinte.

5.4.2 Otimização

Para buscar a correspondência entre marcos e observações considera-se que duas observações o_{j_1} e o_{j_2} obtidas no mesmo instante t devem corresponder a marcos θ_{l_1} e θ_{l_2} distintos. Após calculados os custos $c_{j,l}$, deve-se buscar a correspondência que minimize o somatório desses custos e que obedeça essa restrição. Isso é feito solucionando o seguinte problema de otimização:

$$\min \sum_{j|o_j \in \mathbf{O}_t} \sum_{l|\theta_l \in \Theta} c_{j,l} \cdot \nu_{j,l}$$

Sujeito à:

$$\begin{aligned} \nu_{j,l} &\in (0, 1) && \forall j, l | o_j \in \mathbf{O}_t, \theta_l \in \Theta \\ \sum_{j|o_j \in \mathbf{O}_t} \nu_{j,l} &= 1 && \forall l | \theta_l \in \Theta \\ \sum_{l|\theta_l \in \Theta} \nu_{j,l} &\leq 1 && \forall j | o_j \in \mathbf{O}_t \end{aligned}$$

Esse modelo considera que cada marco θ_l pode ter no máximo uma observação o_j associada e que cada observação o_l deve estar associada a um único marco θ_l . No entanto uma observação o_j pode se referir a um marco θ_l que ainda não foi observado, e portanto não se encontra no mapa. Isso é solucionado de maneira simples, adicionando-se um possível marco não observado θ_l no modelo para cada observação $o_j \in \mathbf{O}_t$.

O algoritmo desenvolvido funciona de maneira semelhante ao algoritmo proposto por Gale and Shapley [1962] para o problema dos casamentos estáveis. Para cada observação é mantida uma lista $[\theta_{l_1}, \dots, \theta_{l_m}]$ ordenada pelos custos, contendo m marcos de menor custo. A Figura 5.5 mostra uma representação dessa estrutura.

O algoritmo funciona do seguinte modo:

- Inicialmente para cada observação o_j é proposta uma correspondência $\nu_{j,l}$ com o marco de menor custo.

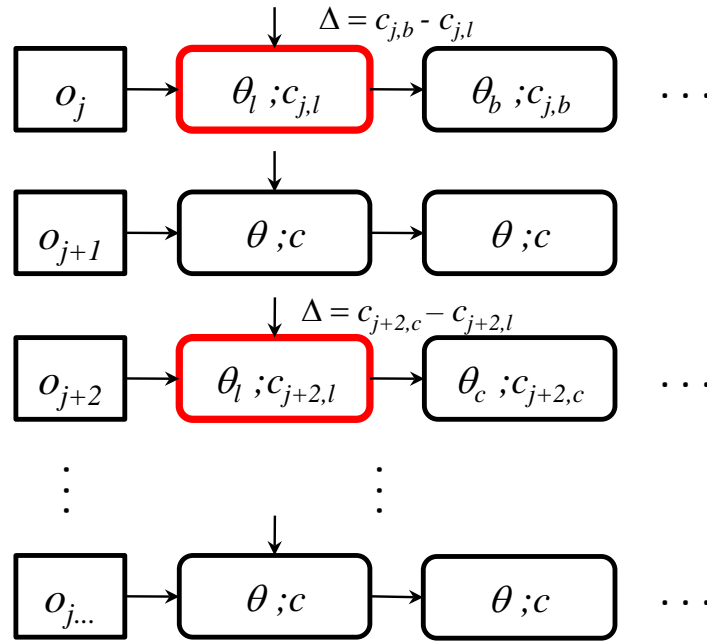


Figura 5.5. Estrutura do método de otimização.

- Caso duas observações o_{j_1} e o_{j_2} sejam associadas a um mesmo marco θ_l tem-se um conflito e a solução é inviável.
- Nesse caso calculam-se os custos Δ_{j_1} e Δ_{j_2} de substituir o marco θ_l pelo próximo melhor marco para as duas observações o_{j_1} e o_{j_2} . Seja θ_{l_1} o próximo marco na lista de o_{j_1} e θ_{l_2} o próximo marco na lista de o_{j_2} . Δ_{j_1} e Δ_{j_2} podem ser obtidos facilmente segundo a equação abaixo:

$$\Delta_{j_1} = c_{j_1, l_1} - c_{j_1, l}$$

$$\Delta_{j_2} = c_{j_2, l_2} - c_{j_2, l}$$

- Escolhe-se então a observação conflitante o_{j_1} ou o_{j_2} cujo valor de Δ_j seja mínimo e propõe-se o próximo melhor marco θ_{l_1} ou θ_{l_2} para tal observação.
- Os passos (2) a (4) são repetidos até que seja obtida uma solução viável, ou até que alguma observação esgote sua lista de marcos. Nesse último caso a observação é descartada e um custo fixo para o descarte é acrescido da solução final.

O custo assintótico do algoritmo descrito é de $O(|\mathbf{O}_t| \cdot m \log m)$ para o pior caso. Tomando um m fixo o custo portanto é linear no número de observações. O custo computacional para cálculo dos custos de correspondência é da ordem de $O(|\mathbf{O}_t| \times |\text{Bel}_{t-1}^{(i)}(\Theta)|)$ para cada partícula, dominando o custo da etapa de correspondência.

5.5 Paralelização

Adotou-se como estratégia de paralelização a utilização blocos distintos para realizar os processamentos referentes a cada partícula. Em todos os módulos cada bloco é responsável por realizar computações referentes a uma única partícula. A independência entre as partículas faz com que pouca sincronização seja necessária o que torna essa estratégia bastante eficiente.

Com exceção do cálculo dos custos de correspondência, em todas as etapas o processamento é realizado por uma sequência de *kernels* cada um com um bloco por partícula. Na etapa de cálculo dos custos um bloco é utilizado para cada par (partícula, observação). Dessa forma, caso o filtro esteja operando com N partículas e em um instante t o conjunto \mathbf{O}_t contiver M observações o número de blocos será $N \times M$. A Figura 5.6 ilustra a estrutura da solução desenvolvida, com cada faixa representa uma etapa, ou módulo, da solução. Cada círculo representa uma partícula, e cada retângulo laranja representa um bloco.

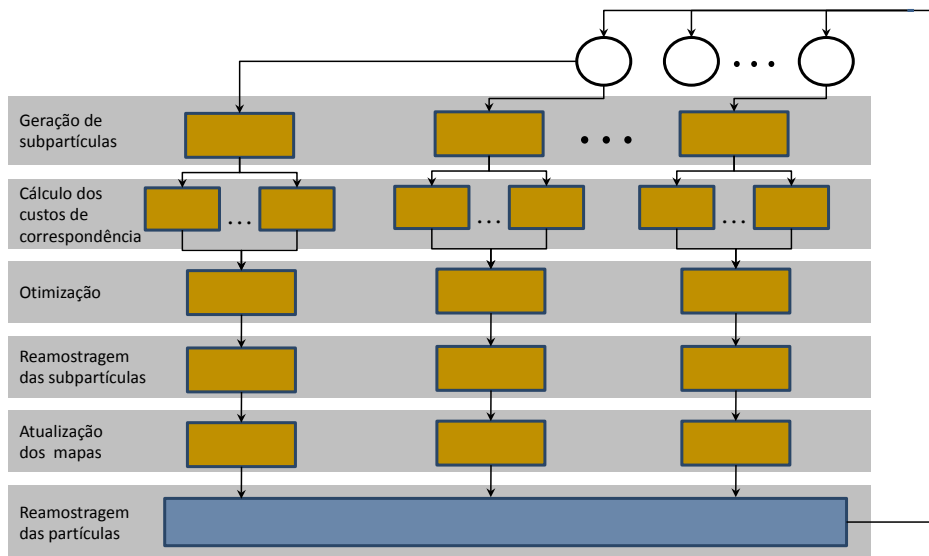


Figura 5.6. Algoritmo paralelo.

O Algoritmo 6 descreve o comportamento da etapa de geração das subpartículas. Cada bloco é responsável por gerar subpartículas referentes a uma partícula, sendo cada subpartícula gerada por uma *thread* distinta do respectivo bloco.

Na etapa de cálculo dos custos um bloco é criado para cada par (partícula, observação) e cada bloco possui uma *thread* por subpartícula. Inicialmente são calculadas as distâncias quadráticas entre os descritores dos marcos e das observações. Cada *thread*, então, realiza um loop sob os marcos do mapa da respectiva partícula computando

Algoritmo 6 Geração de Subpartículas.Geração de Subpartículas($Bel_t(s_{t-1}, \Theta), a_t, S[,]$)

-
- 1: **for each** $s_{t-1}^{(i)} \in Bel_t(s_{t-1}, \Theta)$ **parallel do** {uma partícula por bloco}
 - 2: **for each** $s_t^{k,(i)}$ **parallel do** {uma subpartícula por *thread* do bloco}
 - 3: $s_t^{k,(i)} \sim p(s_t | s_{t-1}^{(i)}, a_t)$. {Gera a k -ésima subpartícula com base no modelo de locomoção.}
 - 4: $S[i, k] \leftarrow s_t^{k,(i)}$
 - 5: **end for**
 - 6: **end for**
-

a verossimilhança da respectiva observação dada a hipótese de pose da subpartícula. Em seguida os custos da correspondência são calculados e então são inseridos na linha de uma matriz de custos por meio de *insertion sort* paralelo, uma matriz de custos é utilizada para cada partícula sendo que essas matrizes possuem uma linha para cada observação. O Algoritmo 7 descreve essa sequência de execução.

Algoritmo 7 Cálculo de custos de correspondência.Custos de Correspondência($Bel_t(s_{t-1}, \Theta), \mathbf{O}_t, S[,], C[[,]]$)

-
- 1: **for each** $Bel_{t-1}^{(i)}(\Theta) \in Bel_t(s_{t-1}, \Theta)$, **each** $o_j \in \mathbf{O}_t$ **parallel do** {um bloco para cada par (partícula, observação)}
 - 2: **for** $l \leftarrow 1 \dots |Bel_{t-1}^{(i)}(\Theta)|$ **do** {Para cada marco θ_l da partícula}
 - 3: calcula distância $d_{j,l}$ entre os descritores da observação e do marco.
 - 4: $c_{j,l} \leftarrow d_{j,l}$
 - 5: **for each** $s_t^{k,(i)} \in S[i, .]$ **parallel do** {uma subpartícula por *thread* do bloco}
 - 6: $G_{j,l} \leftarrow Soma_atomica(G_{j,l}, p(o_j | \theta_l, s_t^{k,(i)}))$. {Gera a k -ésima subpartícula com base no modelo de locomoção.}
 - 7: **end for**
 - 8: $c_{j,l} \leftarrow c_{j,l} - \ln \frac{G_{j,l}}{|S[i, .]|}$
 - 9: Insere $c_{j,l}$ em paralelo na matriz de custos $C[i]$
 - 10: **end for**
 - 11: **end for**
-

O Algoritmo 8 mostra como é feita a escolha das correspondências. Esse passo utiliza uma *thread* para cada observação. Inicialmente é proposto o marco de menor custo como correspondente a cada observação. Enquanto não for obtida uma correspondência viável o algoritmo repete os seguintes passos:

1. Primeiramente os valores Δ_j (Subseção 5.4.2) são calculados para cada observação.
2. Cada *thread* verifica se sua proposta de correspondência está em conflito com alguma outra proposta cujo valor Δ_j seja maior. Essa verificação foi implementada

utilizando um *array* que, a cada passo, é ordenado pelo marco e pelo valor de Δ utilizando o algoritmo PCM-sort [Herruzo et al., 2007]. Cada *thread* verifica se o correspondente proposto é igual ao correspondente da posição anterior no *array*, detectando assim os conflitos.

3. Caso a verificação descrita no item anterior se mostre verdadeira, cada *thread* propõe o próximo marco na lista como correspondente para sua observação.

Como ao final de cada passo todas as observações possuem algum correspondente uma solução e sempre encontrada, Além disso como os marcos nunca são reconsiderados o algoritmo sempre termina. Usando um raciocínio análogo ao de Gale and Shapley [1962] pode-se mostrar que cada observação pode entrar em conflito no máximo $|\mathbf{O}_t| - 1$ vezes. Além disso, o número de iterações do algoritmo tende a ser pequeno o que o torna bastante eficiente na maior parte dos casos. Por motivos de simplificação a verificação do limite de correspondências propostas por observação foi omitida.

Algoritmo 8 Otimização.

Otimização($C[\cdot, \cdot], M[\cdot, \cdot]$)

```

1: for each  $C_i(\Theta) \in C$  do {um bloco para cada matriz de custos das partículas}
2:    $M_i \leftarrow M[i]$  { $M_i$  irá receber as correspondências da  $i$ -ésima partícula}
3:   while  $M_i$  não for uma correspondência viável do
4:     for  $j = 1 \dots |\mathbf{O}_t|$  parallel do {uma thread por observação}
5:        $M_i[j]$  recebe o marco referente ao índice  $idx_j$  da matriz de custos { $idx_i$ 
representa o índice corrente da matriz de custos  $C_i$  para a observação  $o_j$ }
6:        $\Delta_j \leftarrow C_i[j, idx_j + 1] - C_i[j, idx_j]$  {Calcula valor de  $\Delta$ }
7:       if  $M_i[j]$  possui conflito com algum  $M_i[k]$  and  $\Delta_j < \Delta_k$  then
8:          $idx_j \leftarrow idx_j + 1$  {Faz a troca que for aumentar menos a soma dos custos.}
9:       end if
10:      barreira()
11:    end for
12:  end while
13: end for

```

A etapa de atualização das subpartículas é mostrada no Algoritmo 9. Nessa etapa cada *thread* é responsável por calcular o peso de uma única subpartícula. Para isso, cada *thread* realiza um laço sobre as correspondências estimadas computado a verossimilhança das observações. Após concluído o calculo dos pesos seu somatório deve ser calculado, e para cada partícula deve ser sorteada uma subpartícula como hipótese para a nova pose com probabilidade proporcional ao peso (Algoritmo 5).

Algoritmo 9 Atualização das subpartículas.

 Atualização das subpartículas($S[\cdot, \cdot], M[\cdot, \cdot], Bel_t(s_{t-1}, \Theta), \mathbf{O}_t$)

- 1: **for each** $Bel_{t-1}^{(i)}(\Theta) \in Bel_t(s_{t-1}, \Theta)$ **do** {um bloco para cada partícula}
 - 2: **for each** $s_t^{k,(i)} \in S[i, \cdot]$ **parallel do** {uma subpartícula por *thread* do bloco}
 - 3: **for** $j \leftarrow 1 \dots |\mathbf{O}_t|$ **do**
 - 4: obtém marco θ_l do vetor de correspondências $M[i]$
 - 5: $w_t^{k,(i)} \leftarrow w_t^{k,(i)} \times p(o_j | \theta_l, s_t^{k,(i)})$
 - 6: **end for**
 - 7: **end for**
 - 8: **end for**
-

O Algoritmo 10 mostra etapa de atualização dos mapas. Essa utiliza uma *thread* para cada observação em \mathbf{O}_t , sendo que cada *thread* é responsável por atualizar conforme mostrado na Seção 5.1., o marco correspondente a essa observação

Algoritmo 10 Atualização dos Marcos.

 Atualização dos Marcos($M[\cdot, \cdot], Bel_t(s_{t-1}, \Theta), \mathbf{O}_t$)

- 1: **for each** $Bel_{t-1}^{(i)}(\Theta) \in Bel_t(s_{t-1}, \Theta)$ **do** {um bloco para cada partícula}
 - 2: **for each** $o_j \in \mathbf{O}_t$ **parallel do** {uma observação por *thread*}
 - 3: Atualiza marco θ_l correspondente
 - 4: **end for**
 - 5: **end for**
-

Finalmente na etapa de reamostragem calcula-se a soma de prefixos dos pesos das partículas e reamostram-se as partículas da mesma forma apresentada no Algoritmo 4, para cada partícula a ser amostrada toma-se uma das partículas (i) com probabilidade proporcional ao peso atribuído e copia-se a pose $s^{(i)t}$ e o mapa $Bel_t^{(i)}(\Theta)$ para a nova partícula.

Sejam M a quantidade partículas, N a quantidade de subpartículas, K a quantidade de marcos no ambiente e L a quantidade de observações.

Pode-se perceber que o tempo de execução de execução é dominado pelo Algoritmo 7, que possui complexidade assintótica da ordem de $O(M \times N \times K \times L)$. As complexidades dos demais algoritmos apresentados são: $O(M \times N)$ para o Algoritmo 6, $O(M \times L^3 \times \log L)$ para o pior caso do Algoritmo 8, $O(M \times N \times L)$ para o Algoritmo 9 e $O(M \times L)$ para o Algoritmo 10.

Executando em uma máquina com infinitos processadores, os tempos de execução dos algoritmos 7, 6, 8, 9 e 10 seriam proporcionais à $O(K)$, $O(1)$, $O(L^3)$, $O(L)$ e $O(1)$ respectivamente. Como a quantidade de marcos K , em geral, tende a dominar as demais, o tempo de execução ainda seria dominado pelo Algoritmo 7. No entanto, seria linear no número de marcos.

Capítulo 6

Experimentos

Os algoritmos paralelos apresentados foram desenvolvidos em ambiente CUDA disponibilizado pela NVIDIA [CUDA, 2010]. Uma versão sequencial do algoritmo para o SLAM também foi desenvolvida em C++. Nos testes do SLAM foram utilizadas bases de dados disponibilizadas pelo projeto Rawseeds [Ceriani et al., 2009]. Nos experimentos, onde indicado, foram adicionados ruídos normais $\mathcal{N}(0, \sigma_e)$ à leitura de hodometria de cada roda do robô.

Como parte do desenvolvimento também foram implementadas uma versão paralela e uma sequencial do algoritmo de localização de Monte Carlo (MCL) e do algoritmo de SLAM proposto nesse trabalho utilizando um SICK laser como sensor¹. O modelo de locomoção utilizado foi o mesmo modelo descrito na Subseção 3.7.1, utilizado para o problema SLAM, o que permitiu sua validação. A implementação foi testada em ambiente de simulação utilizando dois mapas: o mapa *simple*, disponibilizado como exemplo na plataforma Player/Stage [Gerkey et al., 2003], mostrado na Figura 6.1, e a planta baixa do terceiro andar do prédio do Instituto de Ciências Exatas da UFMG (*ICEx*) mostrado na Figura 6.2. A movimentação do robô foi controlada por um algoritmo de passeio aleatório.

Os experimentos foram realizados em máquina com CPU AMD Athlon ×2 7750 com 2 Gbytes de memória DDR2 800 MHz e uma placa de vídeo NVIDIA GeForce 570 com 1,28 Gbytes de memória GDDR5 executando o sistema operacional Linux Ubuntu 10.04.1 LTS.

¹Daqui para frente nos referiremos à essa versão apenas por SLAM laser

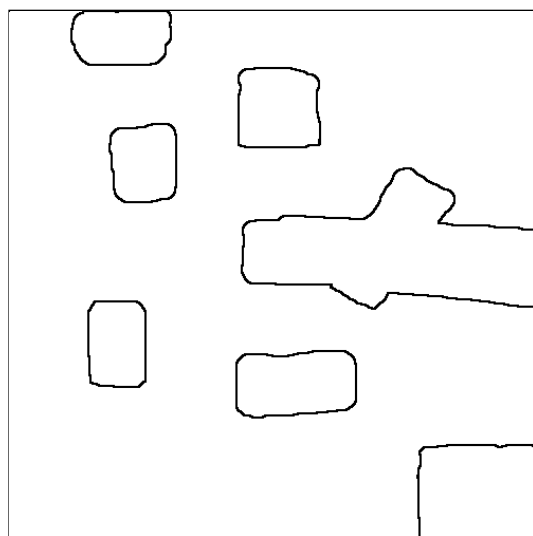


Figura 6.1. Mapa *simple* [Gerkey et al., 2003] utilizado nos testes do MCL paralelo.

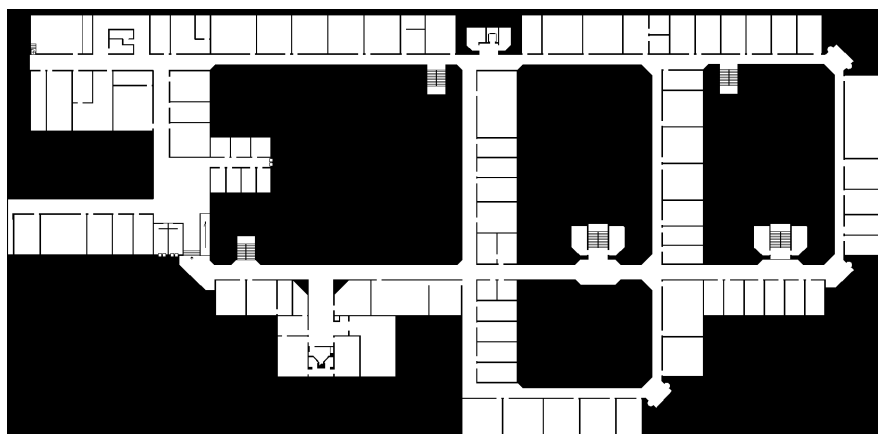


Figura 6.2. Mapa *ICEx* utilizado nos testes do MCL paralelo.

6.1 MCL paralelo

Para o MCL paralelo foram realizados experimentos usando 5.000, 10.000, 20.000, 30.000 e 50.000 partículas para cada um dos mapas com 1000 iterações por teste. Cada experimento foi repetido cinco vezes com o robô partindo de uma mesma posição inicial escolhida de forma aleatória para comparação de desempenho, e foi adotada uma distribuição inicial uniforme para a pose. Foram adquiridos dados de tempo de execução das etapas de propagação e atualização, desvio padrão da posição (erro de posição) e o tempo de convergência da distribuição. O tempo de convergência foi calculado como o tempo necessário para o erro padrão se estabilizar. O erro foi calculado como o desvio padrão da distância absoluta entre a posição “real” do robô e a posição estimada, e

foi considerado estabilizado quando nas próximas 10 iterações não sofrer variação total maior do que o desvio padrão do modelo de locomoção para uma iteração.

A Figura 6.3 mostra a distribuição das partículas na localização de Monte Carlo em três momentos distintos. Primeiramente como não se possui nenhum conhecimento da pose as partículas são distribuídas de maneira uniforme como pode ser visto na Figura 6.3a.

A Figura 6.3b mostra a distribuição após 55 iterações do filtro, a ambiguidade do ambiente faz com que o estado de crença se concentre em duas hipóteses principais. A Figura 6.3c mostra a crença após 108 iterações as novas observações obtidas permitem a desambiguação da pose do robô.

partículas	predição (ms)	atualização (ms)
5.000	$0,4 \pm 0,1$	$3,1 \pm 0,2$
10.000	$0,6 \pm 0,2$	$5,8 \pm 0,3$
20.000	$0,8 \pm 0,2$	$10,7 \pm 0,5$
30.000	$0,9 \pm 0,2$	$16,5 \pm 0,8$
50.000	$1,3 \pm 0,2$	$27,1 \pm 1,2$

Tabela 6.1. Tempos de execução em milissegundos \times número de partículas para o mapa *ICEx*.

partículas	predição (ms)	atualização (ms)
5.000	$0,4 \pm 0,1$	$4,9 \pm 0,3$
10.000	$0,6 \pm 0,2$	$9,9 \pm 0,5$
20.000	$0,7 \pm 0,2$	$17,8 \pm 0,7$
30.000	$0,9 \pm 0,2$	$25,1 \pm 1,1$
50.000	$1,4 \pm 0,2$	$40,2 \pm 1,6$

Tabela 6.2. Tempos de execução em milissegundos vs. número de partículas para o mapa *simple*.

As tabelas 6.1 e 6.2 apresentam as médias e desvios padrão dos tempos de execução em milissegundos para os mapas *ICEx* e *simple* respectivamente. Pode-se notar que o tempo de execução da etapa de predição em pouco varia, nos testes realizados conseguiu-se obter uma taxa de 10 Hz para a etapa de predição para 4 milhões de partículas. O tempo de execução é dominado pelo passo de atualização, um desempenho maior nessa etapa poderia ter sido alcançado utilizando pré-processamento nos mapas. Um fato interessante é que o tempo de execução da etapa de atualização foi consideravelmente menor no mapa *ICEx* do que mapa *simple*, apesar de o primeiro mapa ser mais complexo. Isso se deve à quantidade menor de espaços vazios no mapa *ICEx*,

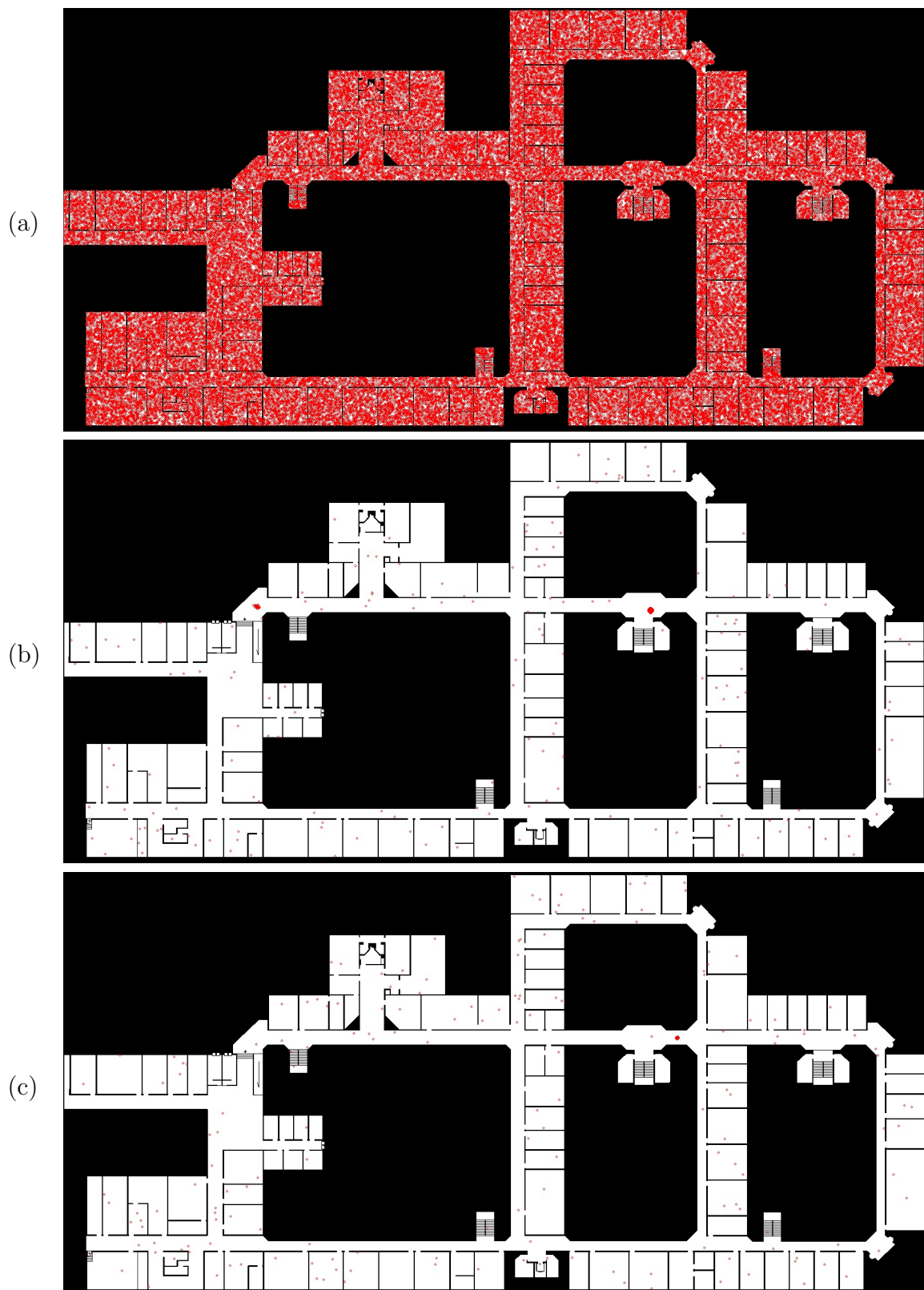


Figura 6.3. Distribuição das partículas na localização simples de Monte Carlo implementada em GPU: (a) distribuição inicial uniforme das partículas, (b) após 55 iterações ainda existe ambiguidade entre duas regiões, (c) após 108 iterações a distribuição converge para uma única localização.

o que leva a uma avaliação mais rápida da distância esperada, e consequentemente, do modelo de observação.

partículas	predição (ms)	atualização (ms)
5.000	$1,3 \pm 0,2$	312 ± 51
10.000	$2,5 \pm 0,7$	618 ± 86
20.000	$5,2 \pm 0,6$	1156 ± 194
30.000	$7,3 \pm 0,7$	1759 ± 315
50.000	$12,8 \pm 0,6$	2930 ± 492

Tabela 6.3. Tempos de execução para CPU em milissegundos vs. número de partículas para o mapa *ICEx*.

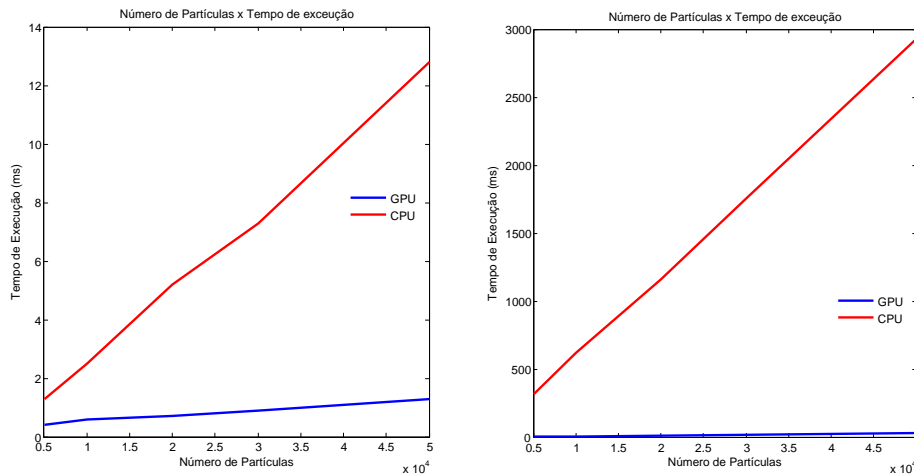


Figura 6.4. Tempo por iteração das etapas de predição e atualização das versões paralela e sequencial do MCL.

A Tabela 6.3 apresenta as médias e desvios dos tempos de execução em milissegundos para a versão sequencial executada para o mapa *ICEx* e A Figura 6.4 mostra a comparação dos tempos de execução das etapas de predição e atualização do MCL na versão paralela e sequencial. Note que o tempo de processamento em GPU, principalmente para a etapa de atualização, é muito inferior ao tempo de processamento sequencial.

A Tabela 6.4 apresenta os ganhos obtidos através da execução em GPU, para todas as quantidades de partículas utilizadas os ganhos obtidos na etapa de atualização foram superiores à $100\times$.

Observando as tabelas 6.5 e 6.6 vemos que o tempo de convergência (estabilização) e o desvio (erro) padrão foram muito maiores para o mapa *ICEx* do que para

partículas	predição	atualização
5.000	3×	104×
10.000	4×	106×
20.000	7×	107×
30.000	8×	107×
50.000	9×	108×

Tabela 6.4. *speedup* obtido pela versão paralela para o mapa *ICEx*.

partículas	convergência (ms)	erro (m)
5.000	3.811,4	0,66 ± 0,16
10.000	3.404,2	0,45 ± 0,24
20.000	3.411,6	0,34 ± 0,23
30.000	3.417,2	0,24 ± 0,20
50.000	3.428,0	0,17 ± 0,17

Tabela 6.5. Tempos de convergência em milissegundos e erro médio em metros vs. número de partículas para o mapa *ICEx*.

partículas	convergência (ms)	erro (m)
5.000	918,8	0,10 ± 0,29
10.000	210,4	0,04 ± 0,18
20.000	218,3	0,03 ± 0,17
30.000	225,7	0,06 ± 0,17
50.000	240,8	0,04 ± 0,17

Tabela 6.6. Tempos de convergência em milissegundos e desvio (erro) padrão médio em metros vs. número de partículas para o mapa *simple*.

o mapa *simple* devido a maior quantidade de regiões ambíguas² e dimensão do primeiro mapa. Os desvios padrão diminuem de forma consistente com o aumento do número de partículas. Como o tempo por iteração é sempre inferior ao período de amostragem do sensor (100 ms) nenhuma informação é descartada. Assim não se percebe o crescimento do erro característico do algoritmo tradicional de localização de Monte Carlo, que ocorre quando o tempo necessário para atualização é maior do que o tempo de leitura do sensor [Thrun et al., 2001].

Em todos os casos, o número de iterações até a convergência (estabilização) manteve aproximadamente o mesmo comportamento. No caso do mapa *simple*, que possui poucas ambiguidades, a convergência para 5.000 partículas levou aproximadamente 9

²Nesse contexto uma região do mapa é considerada ambígua se existem outras regiões distintas onde é esperado que os sensores do robô retornam aproximadamente os mesmos valores, estando localizado em qualquer uma delas.

iterações, mas em todos os demais casos a convergência ocorre em cerca de 3 iterações. O aumento do tempo para quantidades maiores de partículas se deve ao aumento do tempo de processamento de cada iteração.

No caso do mapa *ICEx*, o robô leva cerca de 3.300 ms para atingir um local de desambiguação³ e por isso, o tempo de convergência é maior do que no mapa *simple*. A partir desse ponto os tempos de convergência seguem um comportamento semelhante aos do mapa *simple*.

6.2 SLAM laser paralelo

Nesse trabalho também foi desenvolvida uma versão do algoritmo de SLAM proposto utilizando um SICK Laser como sensor. O modelo de mapa utilizado para o laser foi o de grade de ocupação. Uma grade diferente é computada para cada partícula e o peso associado à cada subpartícula é proporcional à verossimilhança da observação dado o mapa mais recente. O peso da cada partícula é proporcional à soma dos pesos das subpartículas.

Foram realizados experimentos em simulação na plataforma *Player/Stage* variando o ruído do modelo de locomoção. Além disso, foram feitas medidas de tempo de execução para a versão paralela e para uma versão sequencial do algoritmo variando o número de partículas, o número de subpartículas e a resolução do mapa gerado. Nesses experimentos os outros parâmetros foram mantidos fixos em 192 subpartículas e 8,53 píxeis por metro (*ppm*) ao variar o número de partículas, 100 partículas e 8,53 *ppm* ao variar o número de subpartículas e 100 partículas e 192 subpartículas ao variar a resolução do mapa. Todas as medidas de tempo de execução foram tomadas utilizando o mapa *simple* mostrado na Figura 6.1.

A Tabela 6.7 apresenta uma comparação entre o tempo de execução da versão sequencial e paralela do algoritmo. Nota-se que os ganhos de desempenho obtidos foram ainda superiores aos obtidos na localização de Monte Carlo. A principal razão para esse aumento é que, apesar de o algoritmo realizar um número semelhante de acessos à memória principal, o número de operações de ponto flutuante feitas para atualizar as subpartículas é muito superior às realizadas no algoritmo de localização.

A Figura 6.5 demonstra graficamente o resultado apresentado na Tabela 6.7. Pode-se perceber que o tempo de execução da versão paralela do algoritmo é desprezível quando comparado com a versão sequencial. Nota-se também que o tempo é proporcional ao número de partículas.

³Aqui, entende-se local de desambiguação como sendo a primeira posição do mapa na qual o robô

Partículas	GPU (ms)	CPU (ms)	Speedup
5	15,1	$1,04 \times 10^3$	69×
10	15,7	$1,65 \times 10^3$	105×
20	16,4	$2,86 \times 10^3$	174×
50	21,1	$6,47 \times 10^3$	307×
100	32,4	$1,26 \times 10^4$	389×
150	42,0	$1,87 \times 10^4$	445×
200	55,0	$2,48 \times 10^4$	451×
300	79,1	$3,66 \times 10^4$	463×
400	103,3	$4,94 \times 10^4$	478×

Tabela 6.7. Tempos de execução da implementação paralela (GPU) e sequencial (CPU) da versão laser e *speedup* obtido vs número de partículas.

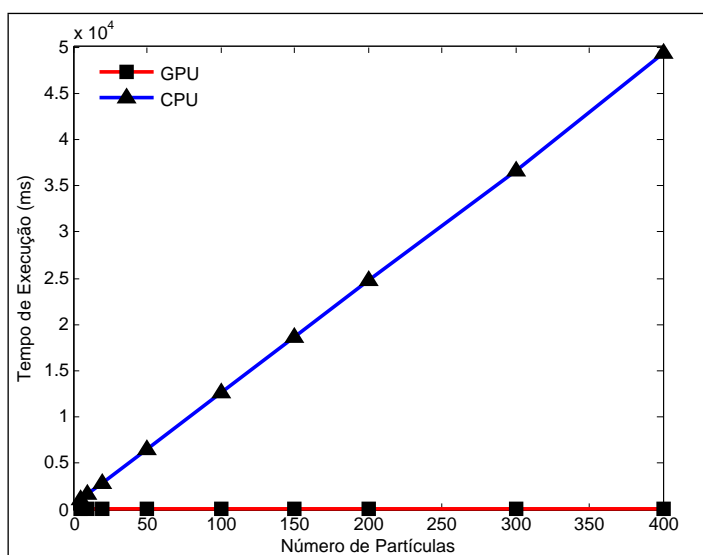


Figura 6.5. Tempos por Iteração × Número de Partículas.

A Tabela 6.8 e a Figura 6.6 apresentam a comparação dos tempos de execução da versão sequencial e paralela do algoritmo ao variar-se o número de subpartículas. Os resultados são semelhantes aos apresentados pela Tabela 6.8 e pela Figura 6.6. Nota-se que o tempo de execução também varia linearmente com o número de subpartículas.

A Tabela 6.9 e a Figura 6.7 mostram uma comparação dos tempos de execução da versão sequencial e paralela do algoritmo ao variar-se a resolução do mapa. Nesse caso o tempo de execução é proporcional ao quadrado da resolução.

As figuras 6.8 e 6.9 apresentam exemplos de uma reconstrução do mapa mostrado na Figura 6.1, utilizando 200 partículas e 192 subpartículas, com ruídos de 1% e 5% na hometria. Pode-se notar que o mapa apresentado na Figura 6.8 é um pouco mais consegue eliminar as possibilidades de se encontrar em uma região distinta.

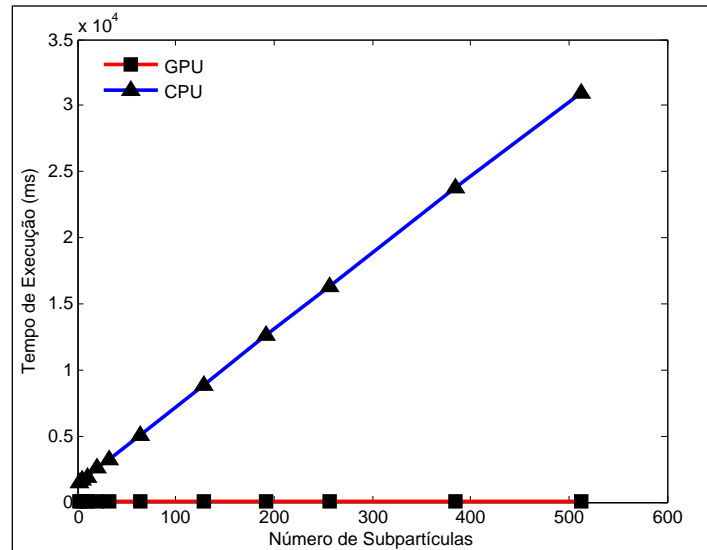


Figura 6.6. Tempos por Iteração \times Número de subpartículas.

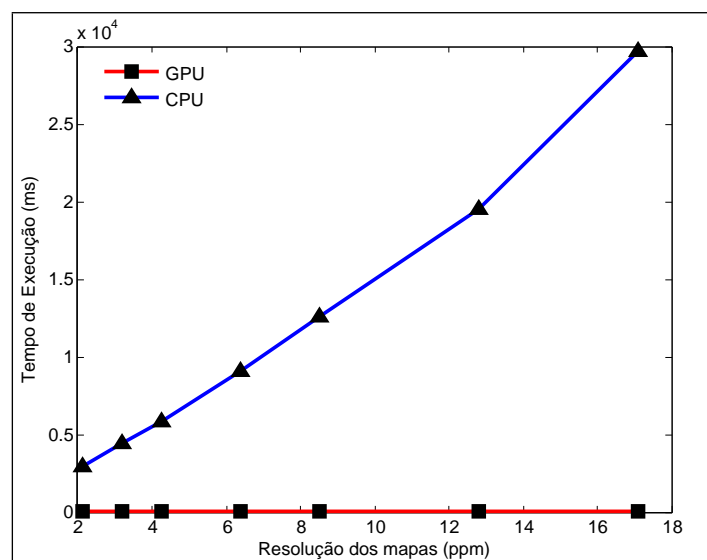


Figura 6.7. Tempos por Iteração \times Resolução dos Mapas.

Subpartículas	GPU (ms)	CPU (ms)	Speedup
2	19,0	$1,49 \times 10^3$	78×
5	19,2	$1,65 \times 10^3$	85×
10	18,9	$1,93 \times 10^3$	101×
20	19,0	$2,56 \times 10^3$	134×
32	19,1	$3,25 \times 10^3$	169×
64	19,9	$5,11 \times 10^3$	257×
128	24,1	$8,82 \times 10^3$	366×
192	32,5	$1,26 \times 10^4$	388×
256	36,1	$1,63 \times 10^4$	451×
384	51,6	$2,38 \times 10^4$	461×
512	61,8	$3,10 \times 10^4$	502×

Tabela 6.8. Tempos de execução da implementação paralela (GPU) e sequencial (CPU) da versão laser e *speedup* obtido vs número de subpartículas.

Resolução dos mapas	GPU (ms)	CPU (ms)	Speedup
2,13 ppm	16,1	$2,90 \times 10^3$	180×
3,20 ppm	18,2	$4,43 \times 10^3$	243×
4,27 ppm	20,2	$5,85 \times 10^3$	290×
6,40 ppm	25,4	$9,11 \times 10^3$	359×
8,53 ppm	32,2	$1,26 \times 10^4$	391×
12,8 ppm	49,7	$1,95 \times 10^4$	393×
17,1 ppm	70,4	$2,97 \times 10^4$	422×

Tabela 6.9. Tempos de execução da implementação paralela (GPU) e sequencial (CPU) da versão laser e *speedup* obtido vs resolução dos mapas.

preciso.

6.3 SLAM monocular paralelo

Nos testes de validação do SLAM foram utilizadas bases de dados fornecidas pelo projeto RawSeeds[Ceriani et al., 2009]. Essas bases são compostas por medidas de sensores obtidas por meio de um robô percorrendo em um ambiente real. A Figura 6.11 mostra uma foto do robô utilizado para coleta dos dados utilizados nos experimentos. O conjunto de sensores do robô é composto por três câmeras frontais, uma câmera omnidirecional, dois SICK laser, sonares, uma unidade inercial e um GPS. Nos experimentos realizados foram utilizados apenas os dados obtidos pela câmera frontal do robô, além das medidas de odometria.

Os experimentos foram realizados na Universidade de Milão-Bicocca. A Figura

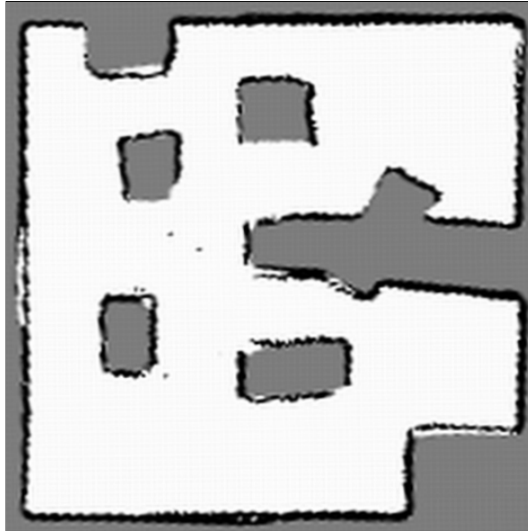


Figura 6.8. Reconstrução do mapa *Simple* com 200 partículas, 192 subpartículas e ruído de locomoção de 1%

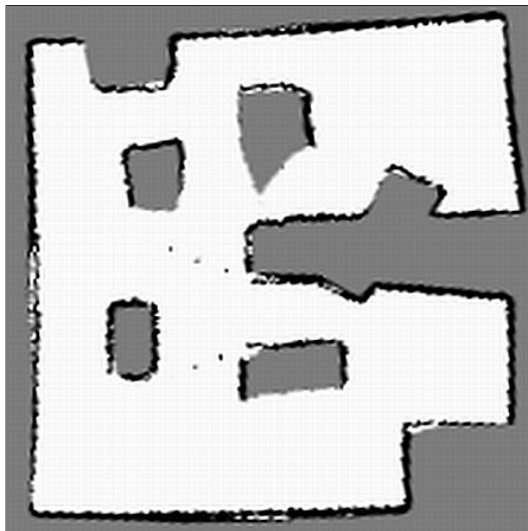


Figura 6.9. Reconstrução do mapa *Simple* com 200 partículas, 192 subpartículas e ruído de locomoção de 5%

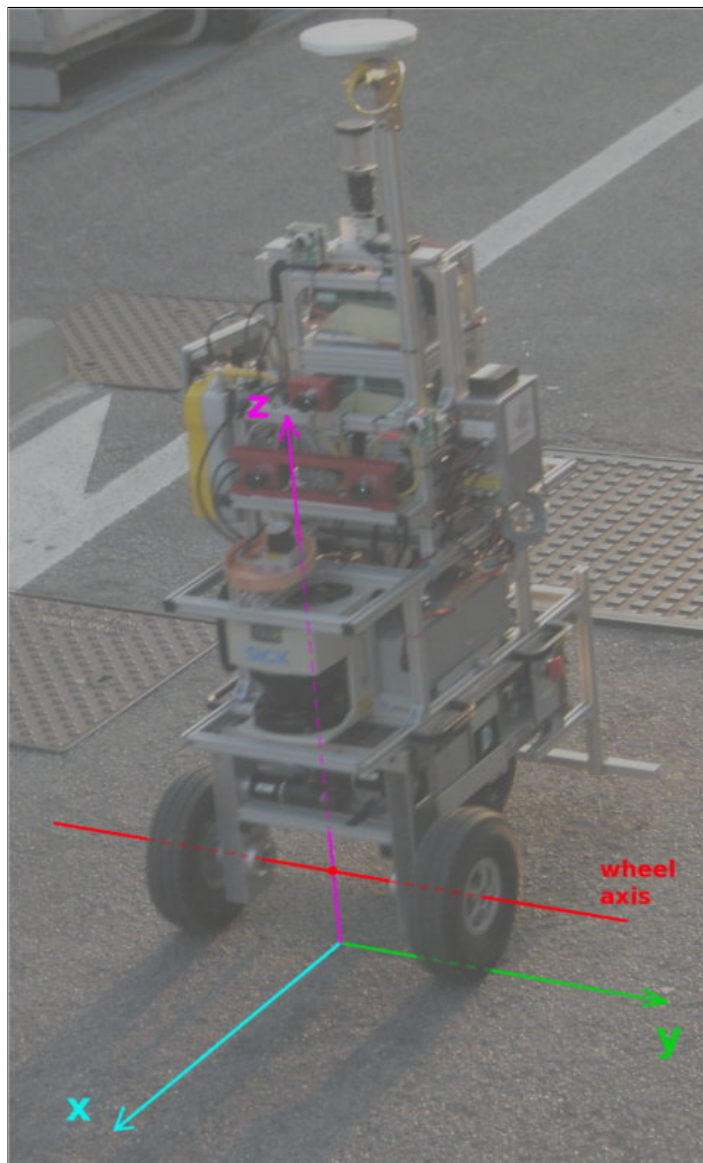


Figura 6.10. Robô utilizado para coleta de dados [Ceriani et al., 2009]

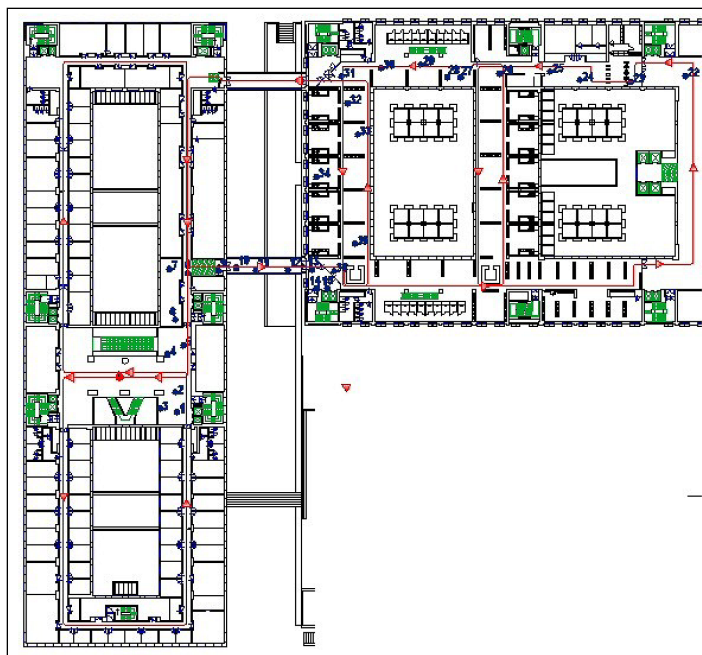


Figura 6.11. Planta baixa do ambiente experimental na Universidade de Milão-Bicocca [Ceriani et al., 2009]

mostra uma planta baixa do ambiente percorrido pelo robô. Também consta da base dados um ground truth com parte do caminho percorrido pelo robô.

A Figura 6.12 mostra estimativas do caminho do robô baseadas somente em hometria. Às medidas de hometria foram adicionados ruídos normais com média 0 e desvios padrão de 5% e 10% da distância percorrida pela robô a cada instante de tempo. As figuras 6.13 e 6.14 mostram caminhos estimados pelo filtro utilizando 384 subpartículas para as medidas de hometria geradas com ruídos de 5% e 10% respectivamente. Cada um dos caminhos mostrados nas figuras foram estimados em uma execução do filtro utilizando uma partícula. Pode-se notar que o caminho estimado é mais próximo do caminho real percorrido pelo robô (em verde) em ambos casos. Isso demonstra que o modelo de incorporação das observações na distribuição proposta pelas subpartículas de fato tende a gerar hipóteses de pose menos ruidosas. Além disso, indica que partículas mais próximas ao caminho real do robô tendem a receber maior peso.

Nota-se, também, que apesar de os caminho gerados com ruído de 5% se aproximarem mais do ground truth do que com 10% essa diferença não é tão discrepante quanto os caminho mostrados na Figura 6.12.

As figuras 6.15 e 6.16 mostram caminhos estimados pelo filtro utilizando 192 subpartículas para os mesmos ruídos. Percebe-se que a quantidade menor de subpartículas tende a gerar resultados com maior variância como era de se esperar.

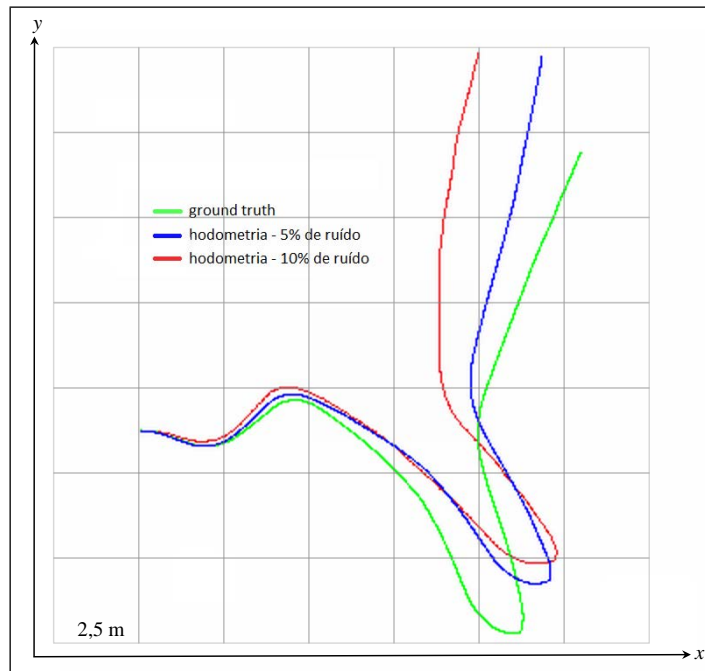


Figura 6.12. Caminhos estimados com base na hometria adicionada à ruídos normais com desvios padrão de 5% e 10% da distância percorrida a cada instante

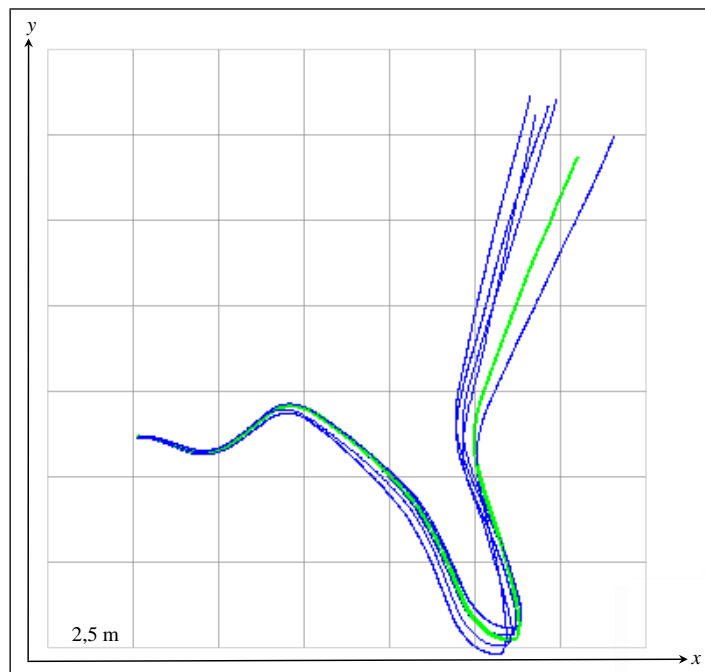


Figura 6.13. Caminhos estimados pelo filtro usando uma partícula e 384 subpartículas e ruído de 5%

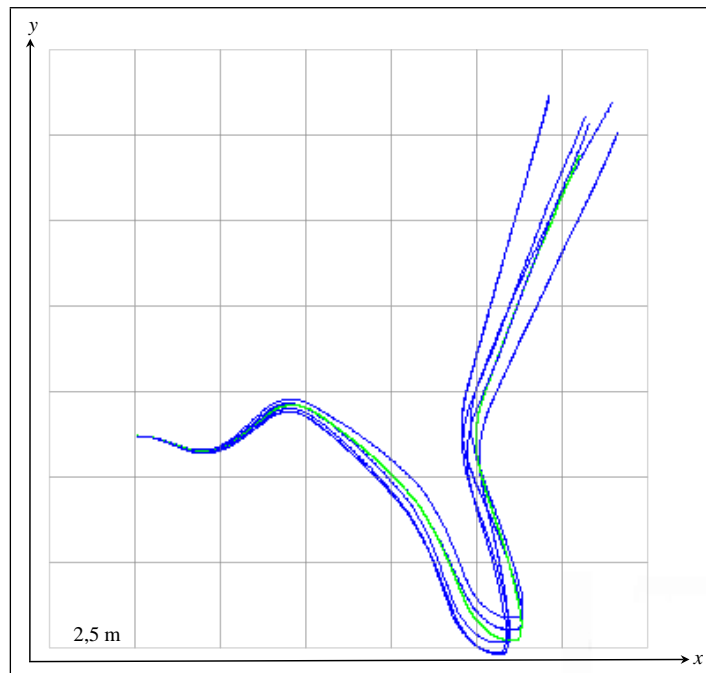


Figura 6.14. Caminhos estimados pelo filtro usando uma partícula e 384 sub-partículas e ruído de 10%

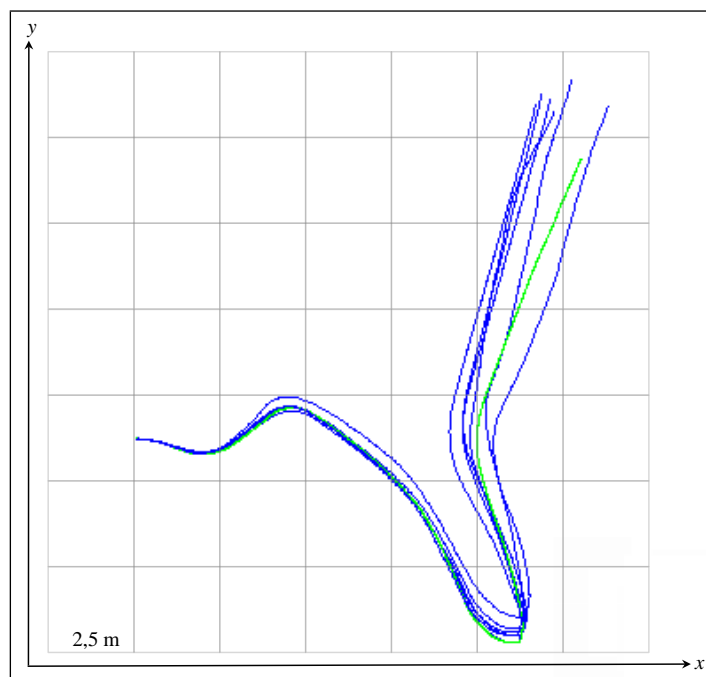


Figura 6.15. Caminhos estimados pelo filtro usando uma partícula e 192 sub-partículas e ruído de 5%

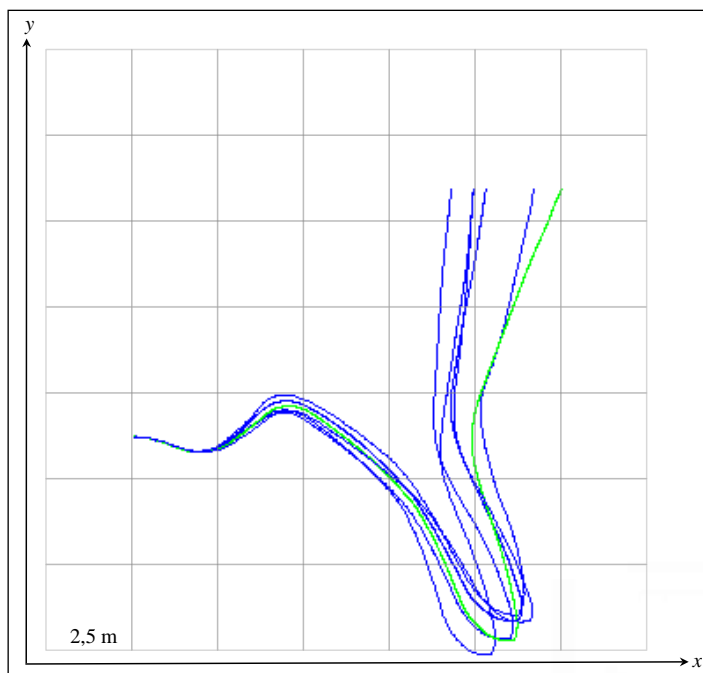


Figura 6.16. Caminhos estimados pelo filtro usando uma partícula e 192 subpartículas e ruído de 10%

As figuras 6.17 6.18 6.19 e 6.20 mostram a variação dos tempos gastos para cálculo dos custos de correspondência dos marcos, para cada uma das formas de representação desenvolvidas (Equações 5.1, 5.2 e 5.3), em relação à: número de marcos, número de partículas, número de subpartículas e observações. Nota-se na Figura 6.20 que o tempo de cálculo dos custos de correspondência para os marcos nas representações referentes às equações 5.1 e 5.3 decresce com o aumento da quantidade de subpartículas até atingir 32 subpartículas. Isso porque quantidades de subpartículas inferiores a 32 causam divergências no $warp \leftarrow [W]$ o que leva a um aumento no tempo de processamento. O mesmo não ocorre com a representação referente à Equação 5.2 pois os cálculos referentes à etapa de cálculo dos custos para essa representação foram paralelizados em relação à cada normal da mistura, e não em relação às subpartículas.

Observando as figuras 6.17 6.18 e 6.19, pode-se perceber que o tempo de execução varia linearmente com o número de marcos, observações e partículas.

As Figuras 6.21 e 6.22 mostram os tempos de execução da etapa de atualização dos marcos nas representações referentes às equações 5.2 e 5.3 em relação ao número de observações e número de partículas. O tempo de execução dessa etapa independe da quantidade de subpartículas ou de marcos utilizada. Nota-se que o tempo de execução dessa etapa é cerca de duas ordens de grandeza menores do que as correspondências. Assim como na etapa de correspondência, percebe-se uma relação linear entre as quan-

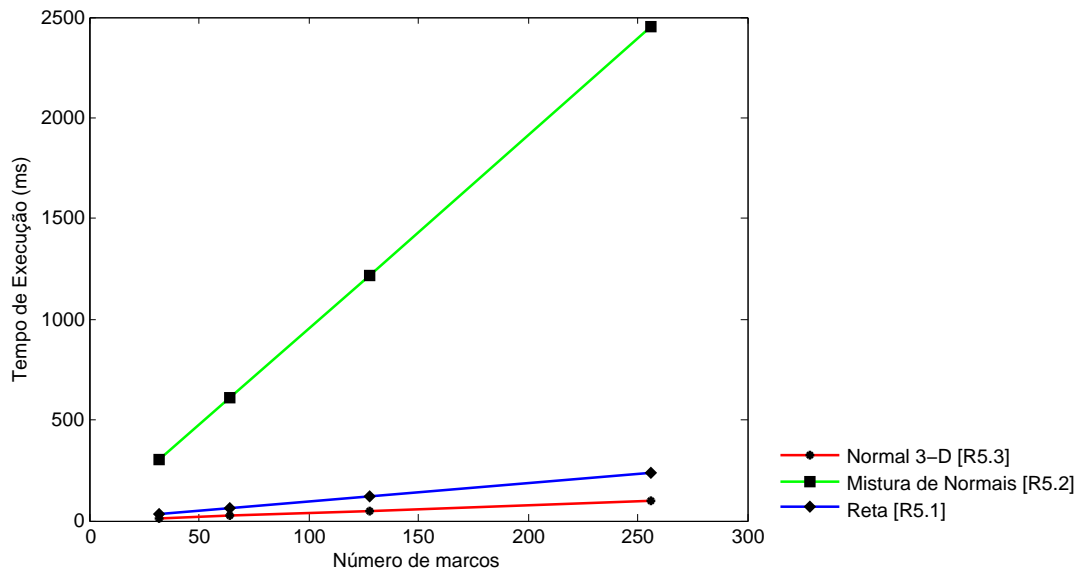


Figura 6.17. Tempo de cálculo dos custos de correspondência \times número de marcos

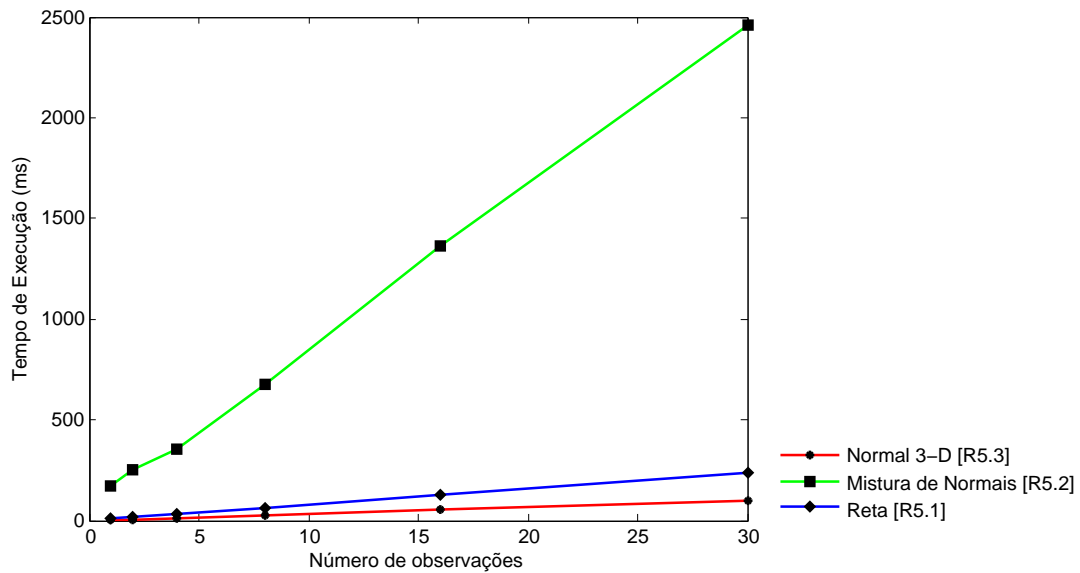


Figura 6.18. Tempo de cálculo dos custos de correspondência \times número de observações

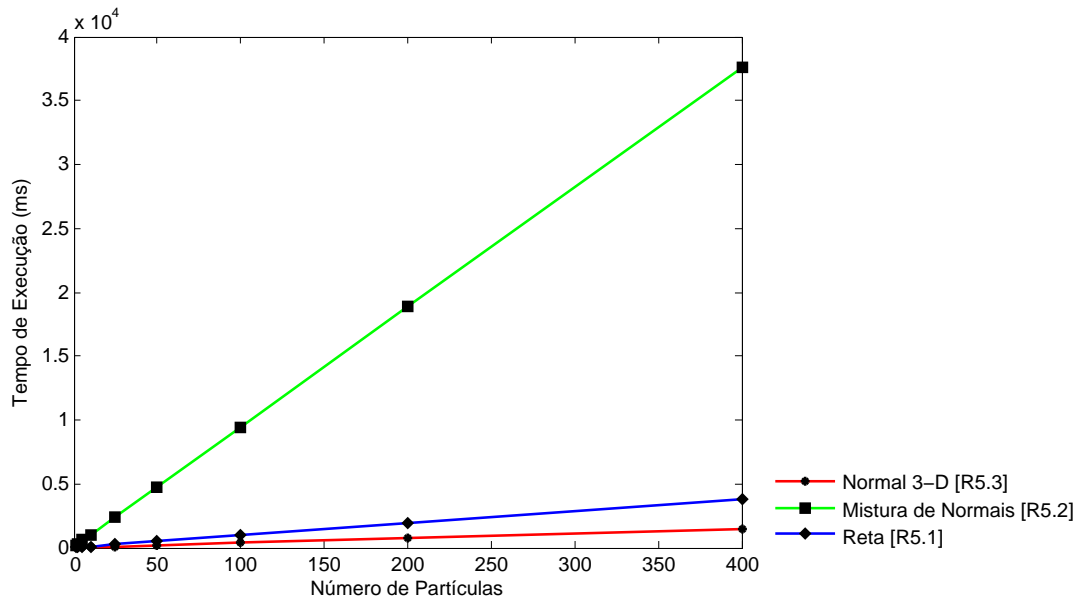


Figura 6.19. Tempo de cálculo dos custos de correspondência \times número de partículas

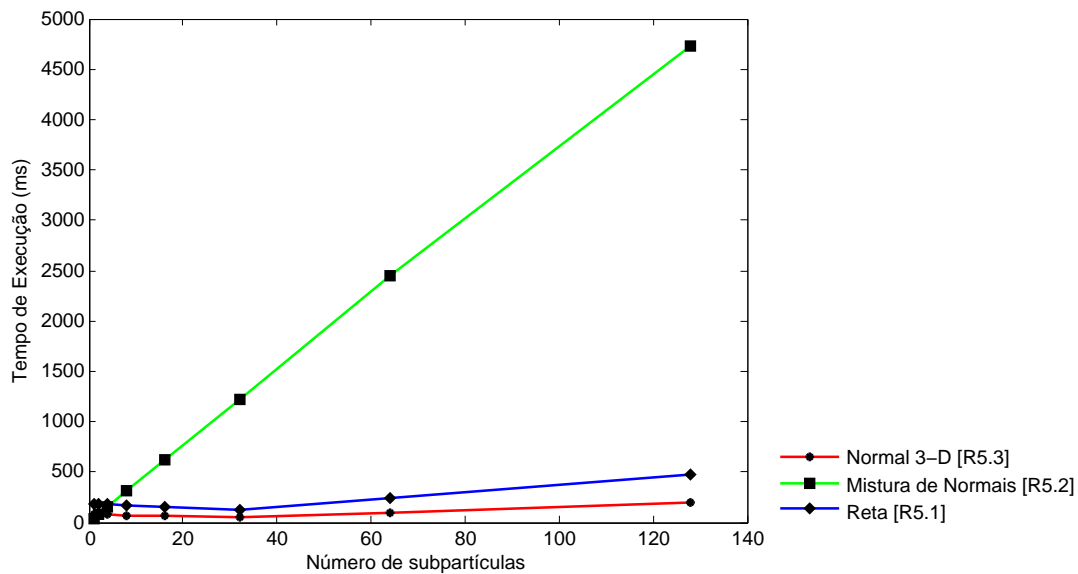


Figura 6.20. Tempo de cálculo dos custos de correspondência \times número de subpartículas

tidades.

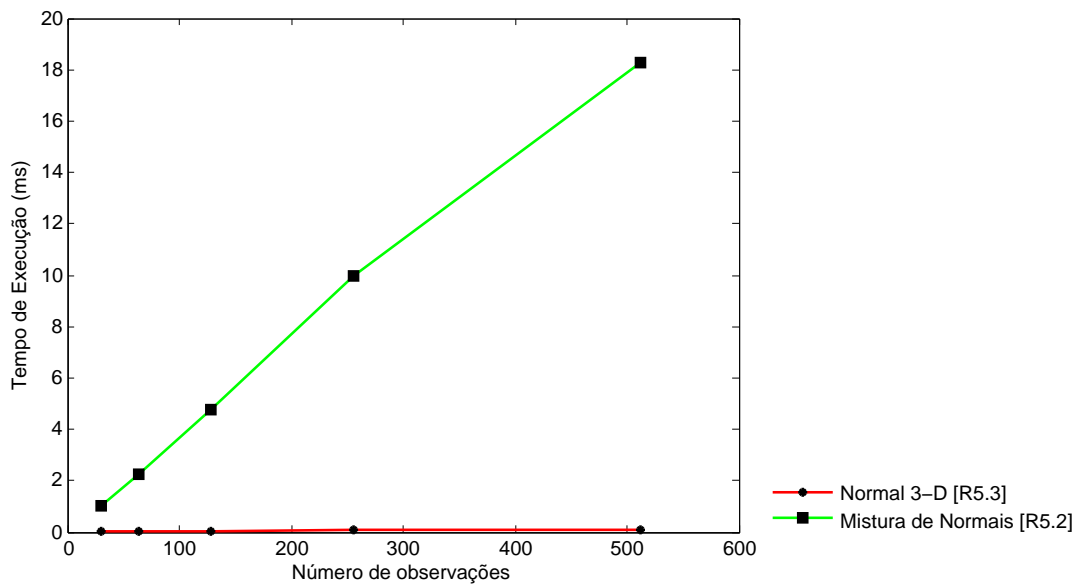


Figura 6.21. Tempo de atualização dos marcos \times número de observações

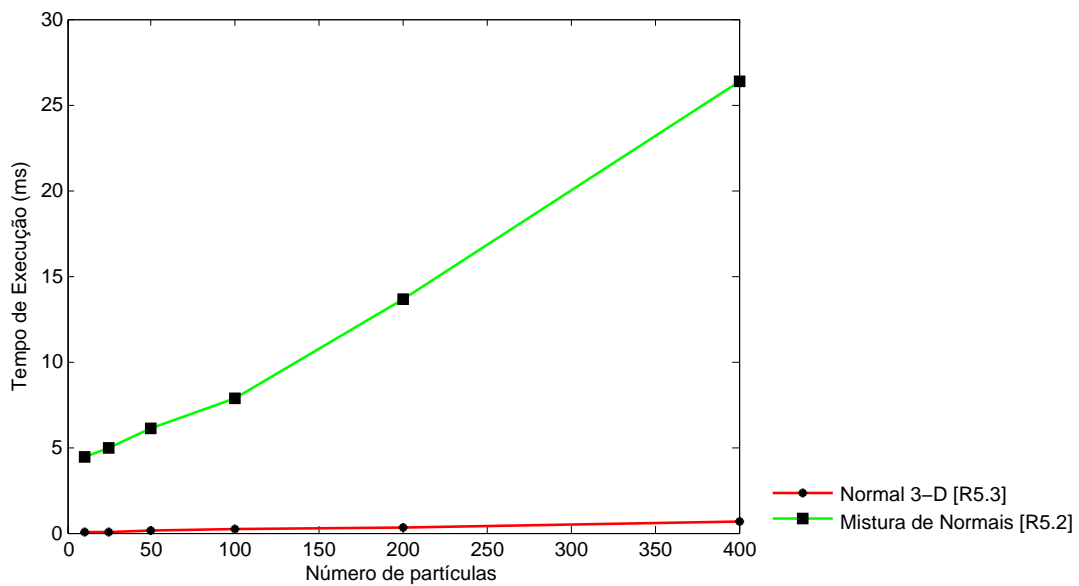


Figura 6.22. Tempo de atualização dos marcos \times número de partículas

A Figura 6.23 mostra a evolução do tempo de propagação das subpartículas com o número de partículas. Nota-se que o tempo gasto com essa etapa é muito inferior ao das demais etapas do filtro.

As figuras 6.24 , 6.25, 6.26 mostram as relações entre os tempos necessários para atualização das subpartículas e número de subpartículas, partículas e observações. Do

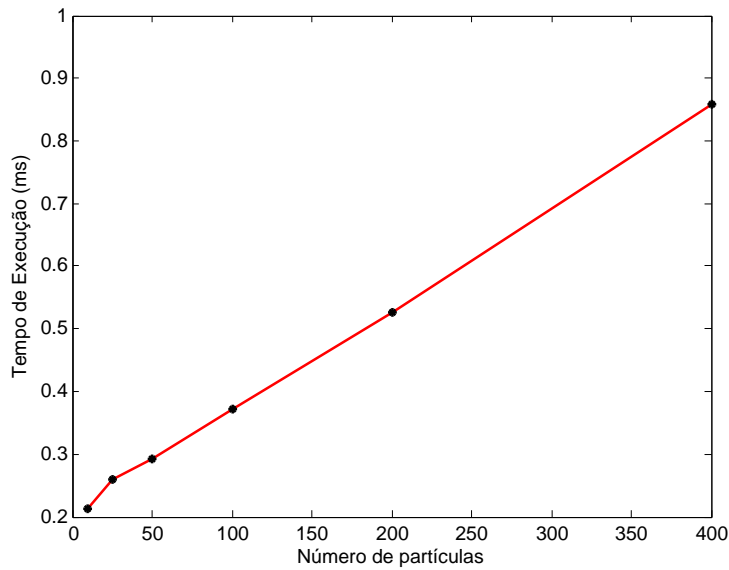


Figura 6.23. Tempo para propagação das subpartículas \times número de partículas

mesmo modo que na etapa de correspondência, observa-se que o tempo de execução é decrescente até 32 subpartículas, quando passa a crescer linearmente. O perfil do tempo de execução dessa etapa é semelhante ao tempo da etapa de atualização dos marcos.

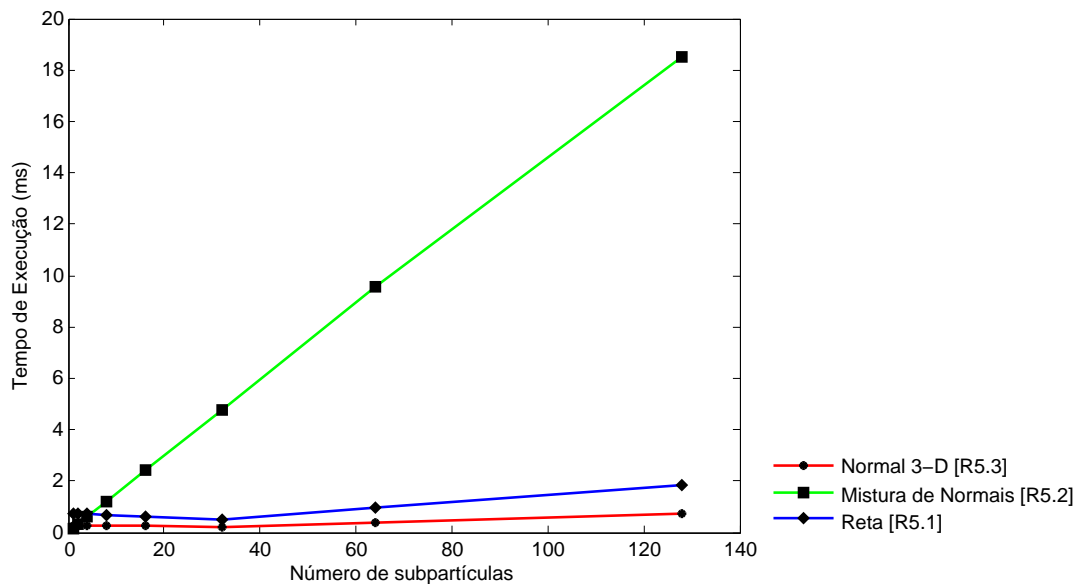


Figura 6.24. Tempo de atualização das subpartículas \times número de subpartículas

Analisando os tempos envolvidos em cada etapa do processo, percebe-se que o tempo de execução do filtro é dominado pela etapa de correspondência. Isso ocorre

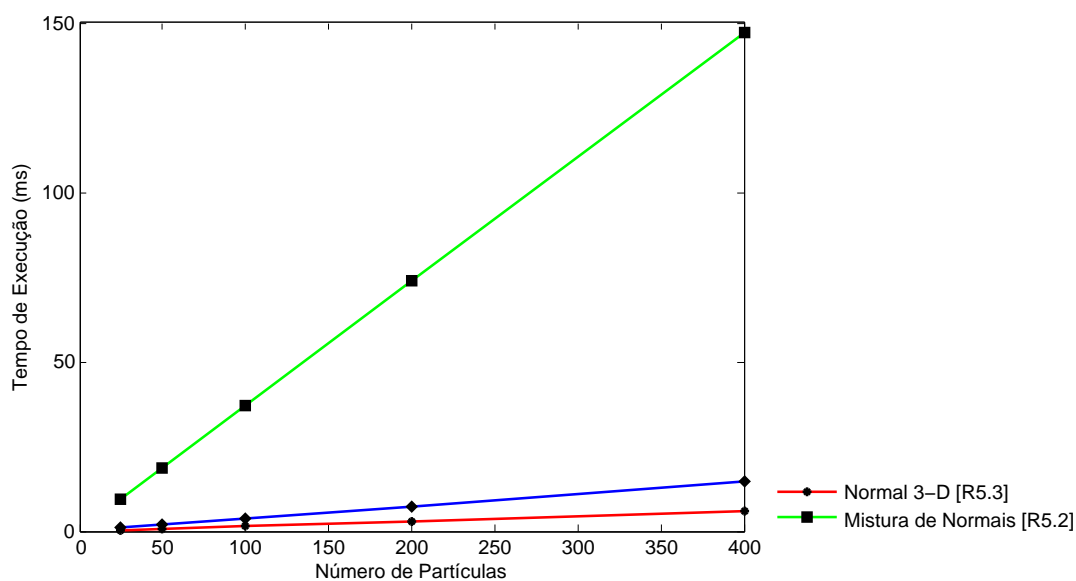


Figura 6.25. Tempo de atualização das subpartículas \times número de partículas

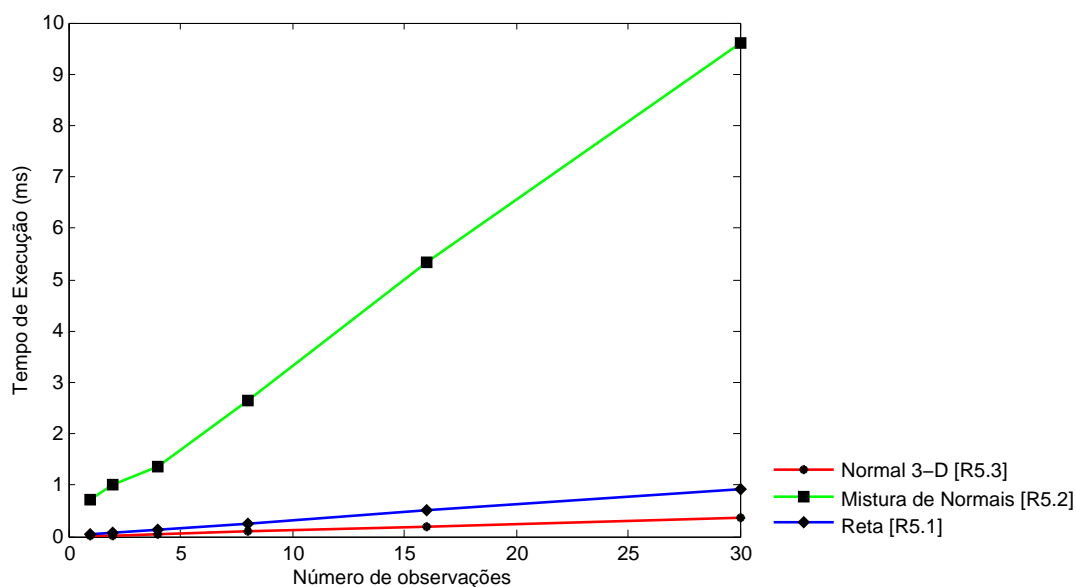


Figura 6.26. Tempo de atualização das subpartículas \times número de observações

porque nessa etapa, para cada subpartícula realiza-se uma comparação entre todas as observações e todos os marcos, além da comparação dos descritores. Há algumas possibilidades para redução desse custo. Uma delas seria a realização dessa comparação para um subconjunto das subpartículas apenas ou o descarte de marcos inviáveis de serem observados. Outra possibilidade seria realizar primeiro a comparação dos descritores cujo custo é mais baixo, e descartar os marcos cuja diferença dos descritores seja maior do que um limiar previamente estabelecido.

Capítulo 7

Conclusões e Trabalhos futuros

Esse trabalho realizou um estudo sobre o problema SLAM para o caso particular onde se utiliza uma única câmera como sensor (SLAM monocular), e sobre como o uso do paralelismo, principalmente de GPUs pode trazer ganhos de desempenho para que se consiga uma execução em tempo real.

Nesse trabalho, foi proposto um novo algoritmo para solução do problema SLAM. O algoritmo desenvolvido foi inspirado no algoritmo FastSLAM [Montemerlo et al., 2002, Montemerlo, 2003] e se baseia na decomposição de Rao-Blackwell e filtro de partículas. Assim como no FastSLAM, um mapa é estimado para cada partícula e a distribuição proposta para a nova pose fornece a informação relativa à última observação. Porém, diferentemente dessa técnica, o nosso algoritmo incorpora essa informação usando um segundo filtro de partículas para cada partícula, e pode ser adaptado para se aplicar à qualquer modelo tanto de locomoção e de observação.

Devido à dificuldade em reduzir o tempo de resposta das instruções de CPUs convencionais, recentemente observa-se uma tendência em aumentar a capacidade de processamento por meio do paralelismo. Uma versão paralela do algoritmo proposto foi desenvolvida e implementada para execução em GPUs da NVIDIA. Os resultados experimentais obtidos indicam que o algoritmo é capaz de estimar corretamente o caminho percorrido pelo robô. A solução desenvolvida é capaz de se adaptar a GPUs com quantidades maiores de núcleos, além de poder ser adaptada para outras unidades paralelas. Em uma máquina com infinitos processadores a etapa de cálculo dos custos possui complexidade de tempo proporcional ao número de marcos no ambiente, e o tempo de execução de todas as outras etapas cresce com o número de observações a cada instante. Com exceção do módulo de estimação das correspondências, todos os módulos do algoritmo desenvolvido se mostraram capazes de executar em tempo real.

Uma versão paralela do algoritmo de localização de Monte Carlo [Ruas et al.,

2010] foi implementada, a qual mostrou ser capaz de localizar o robô em tempo real utilizando quantidades maiores do que de 100.000 partículas. Ganhos de desempenho superiores à $100\times$ foram obtidos em relação ao tempo de execução da versão sequencial. Por meio dos testes executados com a versão paralela do algoritmo, também foi possível validar o modelo de locomoção apresentado no Capítulo 3.

Também foi desenvolvida uma versão paralela e uma sequencial do algoritmo utilizando laser como sensor. A versão paralela obteve ganhos de desempenho de cerca de $500\times$ em relação à versão sequencial e se mostrou capaz de mapear o ambiente em tempo real.

Como a etapa de cálculo dos custos de correspondência é o único impeditivo para uma solução de tempo real, pretende-se futuramente otimizar essa etapa e realizar testes com outras métricas de custos mais simples de serem calculadas. Pretende-se também, adaptar a solução proposta para outros tipos de sensores, como lasers ou sonares, e possivelmente um conjunto de sensores heterogêneos.

Tendo em vista que o tempo necessário para calcular os custos de correspondência com as misturas de gaussianas é muito superior ao para os demais modelos, outra possibilidade de direção futura é o teste com outras parametrizações para os marcos.

Além disso, planeja-se também a realização de experimentos com robôs reais para avaliar o funcionamento do método proposto em ambientes reais e em situações diversas.

Referências Bibliográficas

(2000). *The unscented Kalman filter for nonlinear estimation*.

(2010). *CUDA Programming Guide 3.1*. NVIDIA.

Allusse, Y., Horain, P., Agarwal, A., and Saipriyadarshan, C. (2008). GpuCV: an opensource GPU-accelerated framework for image processing and computer vision. In *Proceeding of the 16th ACM international conference on Multimedia*, pages 1089–1092, New York, NY, USA. ACM Press.

Bailey, T. (2003). Constrained initialisation for bearing-only SLAM. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, volume 2, pages 1966–1971.

Barrachina, S., Castillo, M., Igual, F. D., Mayo, R., and Quintana-Ortí, E. S. (2008). Solving dense linear systems on graphics processors. In *Proceedings of the 14th international Euro-Par conference on Parallel Processing (Euro-Par)*, pages 739–748, Berlin, Heidelberg. Springer-Verlag.

Bay, H., Tuytelaars, T., and Gool, L. V. (2006). SURF: Speeded up robust features. In *Proceedings of the ninth European Conference on Computer Vision*.

Bell, N. and Garland, M. (2009). Implementing sparse matrix-vector multiplication on throughput-oriented processors. In *Proceedings of the ACM/IEEE conference on Supercomputing*, New York, NY, USA. ACM Press.

Blanco, J.-L., González, J., and Fernández-Madrigal, J.-A. (2010). Optimal filtering for non-parametric observation models: Applications to localization and slam. *The International Journal of Robotics Research.*, 29:1726–1742.

Ceriani, S., Fontana, G., Giusti, A., Marzorati, D., Matteucci, M., Migliore, D., Rizzi, D., Sorrenti, D. G., and Taddei, P. (2009). Rawseeds ground truth collection systems for indoor self-localization and mapping. *Autonomous Robots*, 27:353–371.

- Chariot, A. and Keriven, R. (2008). GPU-boostered online image matching. In *Proceedings of the 19th International Conference on Pattern Recognition*, pages 1–4.
- Cornelis, N. and Van Gool, L. (2008). Fast scale invariant feature detection and matching on programmable graphics hardware. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 1–8.
- Costa, A., Kantor, G., and Choset, H. (2004). Bearing-only landmark initialization with unknown data association. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 1764–1769.
- Davison, A. and Murray, D. (2002). Simultaneous localization and map-building using active vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 24(7):865–880.
- Davison, A. J. (2003). Real-time simultaneous localisation and mapping with a single camera. In *Proceedings of the Ninth IEEE International Conference on Computer Vision (ICCV)*, pages 1403–1410.
- Davison, A. J., Reid, I. D., Molton, N. D., and Stasse, O. (2007). MonoSLAM: Real-time single camera SLAM. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 29(6):1052–1067.
- Donald, B. R. (1987). A search algorithm for motion planning with six degrees of freedom. *Artif. Intell.*, 31(3):295–353.
- Doucet, A., Freitas, N. d., Murphy, K. P., and Russell, S. J. (2000). Rao-blackwellised particle filtering for dynamic bayesian networks. In *UAI '00: Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence*, pages 176–183, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.
- Eade, E. and T., D. (2006). Scalable monocular SLAM. In *Proceedings of IEEE Computer Vision and Pattern Recognition*.
- England, J. N. (1978). A system for interactive modeling of physical curved surface objects. In *Proceedings of the fifth annual conference on Computer graphics and interactive techniques*, pages 336–340, New York, NY, USA. ACM Press.
- Forsyth, D. A. and Ponce, J. (2002). *Computer Vision: A Modern Approach*. Prentice Hall. ISBN 0-130-85198-1.

- Freeman, W. and Adelson, E. (1991). The design and use of steerable filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 13(9):891–906.
- Fung, J. and Mann, S. (2005). OpenVIDIA: parallel GPU computer vision. In *Proceedings of the 13th annual ACM international conference on Multimedia*, pages 849–852, New York, NY, USA. ACM Press.
- Gale, D. and Shapley, L. S. (1962). College Admissions and the Stability of Marriage. *The American Mathematical Monthly*, 69(1):9–15.
- Galoppo, N., Govindaraju, N. K., Henson, M., and Manocha, D. (2005). LU-GPU: Efficient algorithms for solving dense linear systems on graphics hardware. In *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, page 3, Washington, DC, USA. IEEE Computer Society.
- Gamallo, C., Mucientes, M., and Regueiro, C. V. (2009). Visual fastslam through omnivision. pages 128–135, University of Ulster, Londonderry, United Kingdom.
- Garland, M., Le Grand, S., Nickolls, J., Anderson, J., Hardwick, J., Morton, S., Phillips, E., Zhang, Y., and Volkov, V. (2008). Parallel computing experiences with CUDA. *Micro, IEEE*, 28(4):13–27.
- Gerkey, B., Vaughan, R., and Howard, A. (2003). The player/stage project: Tools for multi-robot and distributed sensor systems. In *11th International Conference on Advanced Robotics (ICAR 2003)*, Coimbra, Portugal.
- Gillies, D. (2000). *Philosophical Theories of Probability*. Routledge, UK.
- Harris, C. and Stephens, M. (1988). A combined corner and edge detector. In *Proceedings of the Fourth Alvey Vision Conference*, pages 147–151.
- Hartley, R. I. (1993). Euclidean reconstruction from uncalibrated views. In *Applications of Invariance in Computer Vision*, pages 237–256. Springer-Verlag.
- Herruzo, E., Ruiz, G., Benavides, J. I., and Plata, O. (2007). A new parallel sorting algorithm based on odd-even mergesort. In *Proceedings of the 15th Euromicro International Conference on Parallel, Distributed and Network-Based Processing, PDP '07*, pages 18–22, Washington, DC, USA. IEEE Computer Society.
- Heymann, S., Fröhlich, B., Medien, F., Müller, K., and Wiegand, T. (2007). SIFT implementation and optimization for general-purpose GPU. In *Proceedings of the 15th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*.

- Holmes, S., Klein, G., and Murray, D. (2009). An $o(n^2)$ square root unscented kalman filter for visual simultaneous localization and mapping. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 31(7):1251–1263.
- Hygounenc, E., Jung, I.-K., Souères, P., and Lacroix, S. (2004). The autonomous blimp project of LAAS-CNRS: Achievements in flight control and terrain mapping. *International Journal of Robotics Research*, 23:473–511.
- Isard, M. and Blake, A. (1998). Condensation - conditional density propagation for visual tracking. *International Journal of Computer Vision*, 29:5–28.
- Jensfelt, P. and Folkesson, J. and Kragic, D. C. H. (2006). Exploiting distinguishable image features in robotics mapping and localization. In *Proceedings of European Robotics Symposium*, Palermo, Italy.
- Julier, S. J. and K., U. J. (2004). Unscented filtering and nonlinear estimation. *Proceedings of the IEEE*, 92(3):401–422.
- Jung, I.-K. and Lacroix, S. (2003). Simultaneous localization and mapping with stereo-vision. In *Proceedings of the Eleventh International Symposium of Robotics Research (ISRR)*.
- Jung, J. H. and O’Leary, D. P. (2006). Cholesky Decomposition and Linear Programming on a GPU. Master’s thesis, University of Maryland.
- Kalman, R. E. (1960). A new approach to linear filtering and prediction problems. *Transactions of the ASME Journal of Basic Engineering*, (82 (Series D)):35–45.
- Ke, Y. and Sukthankar, R. (2004). PCA-SIFT: A more distinctive representation for local image descriptors. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2, pages 506–513.
- Khronos OpenCL Working Group (2008). *The OpenCL Specification, version 1.0.29*.
- Kim, J.-H. and Sukkarieh, S. (2003). Airborne simultaneous localisation and map building. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, volume 1, pages 406–411.
- Koenderink, J. and van Doorn, A. (1987). Representation of local geometry in the visual system. *Biological Cybernetics*, 55(6):367–375.
- Kuhn, H. W. (1955). The Hungarian method for the assignment problem. *Naval Research Logistic Quarterly*, 2:83–97.

- Kullback, S. and Leibler, R. A. (1951). On information and sufficiency. *Annals of Mathematical Statistics*, 22:79–86.
- Kwok, N. M. and Dissanayake, G. (2005). Bearing-only SLAM using a SPRT based gaussian sum filter. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.
- Lefebvre, T., Bruyninckx, H., and de Schutter, J. (2004). Kalman filters for non-linear systems: A comparison of performance. *International Journal of Control*, 77:639–653.
- Lemaire, T., Lacroix, S., and Sola, J. (2005). A practical bearing-only SLAM algorithm. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*.
- Lengyel, J., Reichert, M., Donald, B. R., and Greenberg, D. P. (1990). Real-time robot motion planning using rasterizing computer graphics hardware. In *Proceedings of the 17th Annual Conference on Computer Graphics and Interactive Techniques*, pages 327–335.
- Leonard, J. J. and Rikoski, R. J. (2000). Incorporation of delayed decision making into stochastic mapping. In *Proceedings of the Seventh International Symposium on Experimental Robotics (ISER)*, pages 533–542.
- Lindeberg, T. (1994). Scale-space theory: A basic tool for analysing structures at different scales. *Journal of Applied Statistics*, pages 224–270.
- Lingemann, K., Surmann, H., Nuchter, A., and Hertzberg, J. (2004). Indoor and outdoor localization for fast mobile robots. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 3, pages 2185–2190.
- Lowe, D. G. (1999). Object recognition from local scale-invariant features. In *Proceedings of the seventh IEEE International Conference on Computer Vision*, pages 1150–1157, Corfu, Greece.
- Lowe, D. G. (2004). Distinctive image features from scale-invariant key points. *International Journal of Computer Vision*, 2:91–110.
- Mikolajczyk, K. and Schmid, C. (2003). A performance evaluation of local descriptors. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2, pages 257–263.

- Mikolajczyk, K. and Schmid, C. (2005). A performance evaluation of local descriptors. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 27(10):1615–1630.
- Montemerlo, M. (2003). *FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem with Unknown Data Association*. PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA.
- Montemerlo, M., Thrun, S., Koller, D., and Wegbreit, B. (2002). FastSLAM: A factored solution to the simultaneous localization and mapping problem. In *Proceedings of the AAAI National Conference on Artificial Intelligence*, Edmonton, Canada. AAAI.
- Moravec, H. (1981). Rover visual obstacle avoidance. In *Proceedings of the Seventh International Joint Conference on Artificial Intelligence(IJCAI)*, volume 2, pages 785–790, Vancouver.
- Murphy, K. (1999). Bayesian map learning in dynamic environments. In *In Neural Info. Proc. Systems (NIPS)*, pages 1015–1021. MIT Press.
- Nuchter, A., Lingemann, K., Hertzberg, J., and Surmann, H. (2007). 6D SLAM — 3D mapping outdoor environments. *Journal of Field Robotics*, 24(8-9):699–722.
- Olson, E. (2009). Real-time correlative scan matching. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 4387–4393, Kobe, Japan.
- Paolo, H. J., Favaro, P., and Soatto, S. (2003). A semi-direct approach to structure from motion. *The Visual Computer*, 19:377–394.
- Pitt, M. K. and Shephard, N. (1999). Filtering via Simulation: Auxiliary Particle Filters. *Journal of the American Statistical Association*, 94(446):590–599.
- Ruas, A., Campos, M. F., and Chaimowicz, L. (2010). Implementação paralela da localização de monte carlo para robôs móveis em GPUs comerciais. In *XVIII Congresso Brasileiro de Automática (CBA) 2010*.
- Ryde, J. and Hu, H. (2007). Mobile robot 3D perception and mapping without odometry using multi-resolution occupancy lists. In *Proceedings of the IEEE International Conference on Mechatronics and Automation(ICMA)*, pages 331–336.
- Se, S., Lowe, D., and Little, J. (2002). Mobile robot localization and mapping with uncertainty using scale-invariant visual landmarks. *International Journal of Robotics Research*, 21:735–758.

- Shi, J. and Tomasi, C. (1994). Good features to track. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 593–600.
- Siegwart, R. and Nourbakhsh, I. R. (2004). *Introduction to Autonomous Mobile Robots*. Bradford Book.
- Silveira, G., Malis, E., and Rives, P. (2009). Registro direto de imagens para SLAM visual. In *IX Simpósio Brasileiro de Automação Inteligente (SBAI)*.
- Simon, D. (2006). *Optimal State Estimation: Kalman, H Infinity, and Nonlinear Approaches*. Wiley & Sons, 1. Auflage edition.
- Sinha, S. N., michael Frahm, J., Pollefeys, M., and Genc, Y. (2006). GPU-based video feature tracking and matching. Technical report, In Workshop on Edge Computing Using New Commodity Architectures.
- Smith, R., Self, M., and Cheeseman, P. (1990). Estimating uncertain spatial relationships in robotics. In *Autonomous Robot Vehicles*, pages 167–193. Springer-Verlag, New York, Inc., New York, NY, USA.
- Smith, R. C. and Cheeseman, P. (1986). On the representation and estimation of spatial uncertainty. *International Journal of Robotics Research*, 5(4):56–68.
- Sola, J., Monin, A., Devy, M., and Lemaire, T. (2005). Undelayed initialization in bearing only SLAM. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*.
- Strasdat, H., Montiel, J. M. M., and Davison, A. J. (2010). Real-time monocular slam: Why filter? In *ICRA*, pages 2657–2664.
- Tardos, J. D., Neira, J., Newman, P. M., and Leonard, J. J. (2002). Robust mapping and localization in indoor environments using sonar data. *The International Journal of Robotics Research*., 21:311–330.
- Terriberry, T., French, L., and Helmsen, J. (2008). GPU accelerating speeded-up robust features. In *Symposium on 3D Data Processing, Visualization and Transmission*.
- Thrun, S., Burgard, W., and Fox, D. (2005). *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press.
- Thrun, S., Fox, D., Burgard, W., and Dellaert, F. (2001). Robust Monte Carlo localization for mobile robots. *Artificial Intelligence*, 128(1-2):99–141.

- Tomasi, C. and Kanade, T. (1992). Shape and motion from image streams under orthography: a factorization method. *International Journal of Computer Vision*, 9(2):137–154–154.
- Trajkovic, M. and Hedley, M. (1998). Fast corner detection. *Image and Vision Computing*, 16(2):75–87.
- van der Merwe, R. and Wan, E. (2001). The square-root unscented kalman filter for state and parameter estimation. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 6, pages 3461–3464.
- Yguel, M., Aycard, O., and Laugier, C. (2008). Efficient GPU-based construction of occupancy grids using several laser range-finders. *International Journal of Vehicle Autonomous Systems*, 6:48 – 83.